EE 746-Neuromorphic Computing, Homework 2

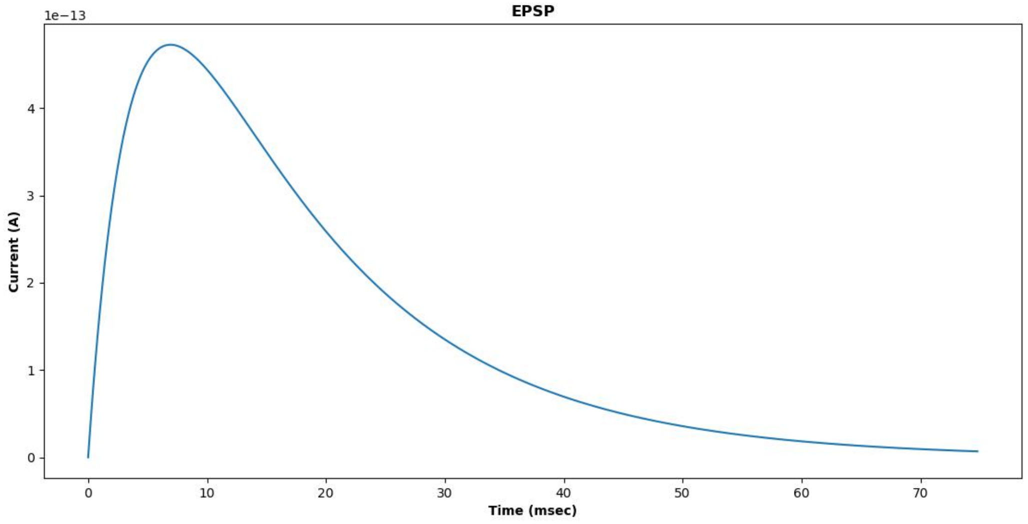
Shashwat Shukla
150260025

The codes for each of the parts has been named and numbered accordingly and stored in a single folder titled “Code”. Within this folder the code for say question 1 is contained in the file “q1.py”. There is a folder titled “Alternate” within the “Code” folder, and it contains code to simulate an alternative implementation of questions 3, 4 and 5 and are included due to potential ambiguity in the wording of these questions.
All the code has been written in Python 2.7. To see the results of any of the codes, simply open a terminal and type “python filename.py”, replacing filename with the name of the file to be run.

Question 1

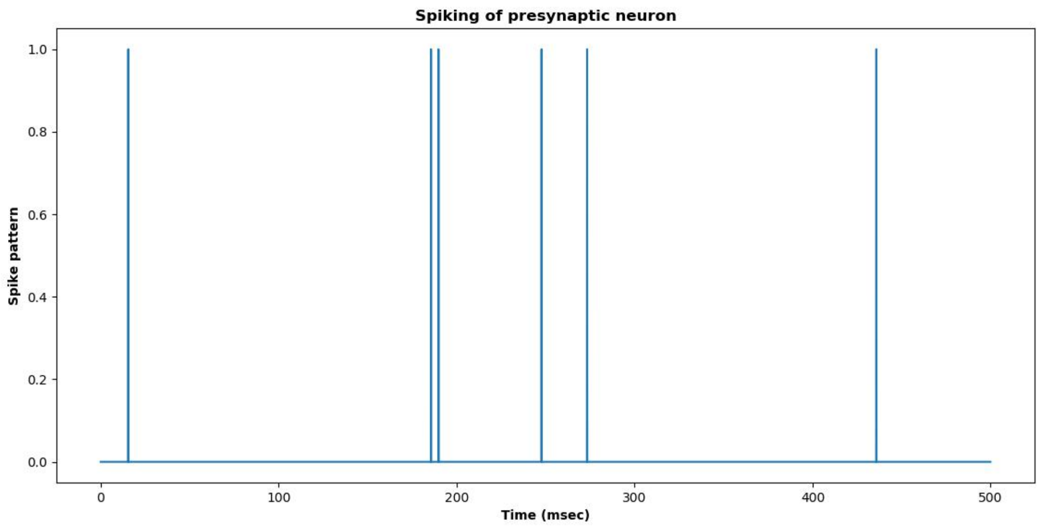
a) A Poisson process is characterized by a waiting time that follows a Poisson distribution. Such a distribution for wait time can be obtained by dividing the process into small time bins and letting the probability of spiking across each bins be independent and identically distributed, with the probability of spiking in a time-bin being a Bernoulli random variable with p = λ * dt
Hence we can simulate a Poisson process for M time-steps by generating an M length random vector, with each element of this vector being sampled uniformly from 0 to 1. We then compare each element of this vector with p, and if the value is less than p then we set the value of this element to 1, indicating that the neuron has spiked in this time bin. Otherwise, the value of this element is set to 0, indicating that the neuron did not spike in this time bin.
I have implemented vectorised Numpy code in Python to do this efficiently for N neurons and M time-steps.

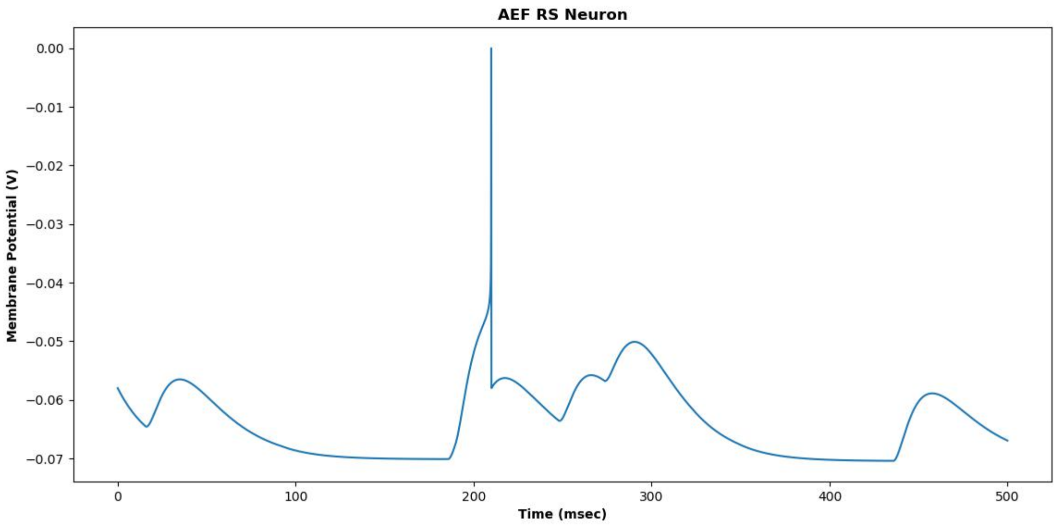
b) The AEF neuron is simulated using code that I implemented in homework 1. To efficiently compute the postsynaptic current, I computed the EPSP current waveform due to a single spike and have truncated it at (5 * tau) seconds. The truncation is justified because both the exponential functions that define the EPSP will die out in 5 time constants (of the slower expontial tau, with the tau_s component dying out even faster). The validity of this truncation can also be visually inferred from the plot of the truncated EPSP waveform: (the waveform clearly dies out before 5 * tau)



This waveform was computed once and stored in an array.
The postsynaptic current due to multiple spikes is then simply a convolution between this EPSP waveform and the spiketrain, multiplied by the synaptic. This greatly speeds up computation and makes for cleaner code.

Output from one run:

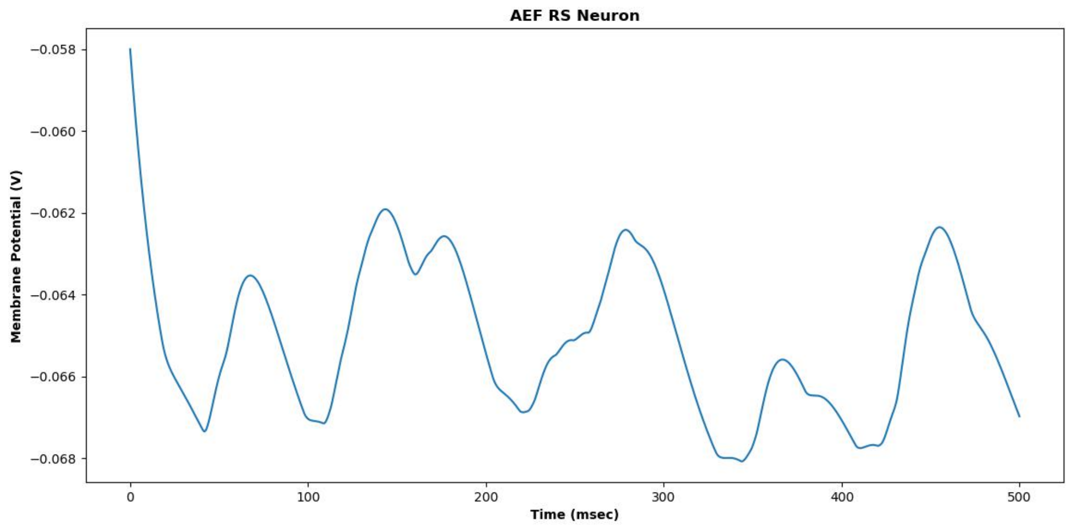




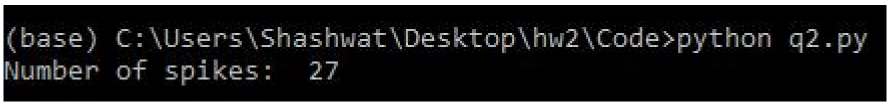
It can clearly be seen that the neuron spikes when two spikes are really close (the two spikes just before 200ms), but not when they come a little far apart (the two spiked between 200ms and 300ms), due to the fact that the EPSPs add up most effectively for closely spaced spikes.

Question 2

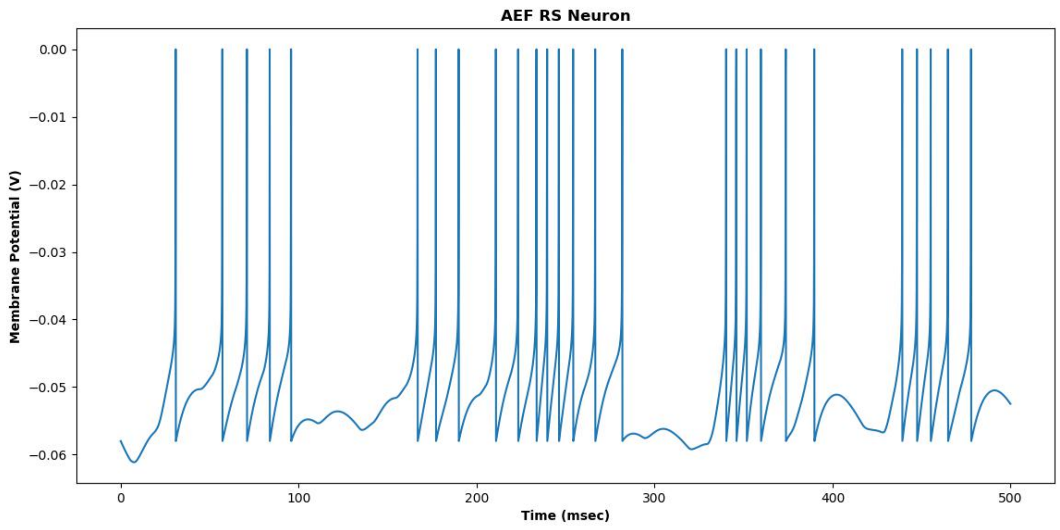
a) For $w_0 = 50$ and $\sigma_w = 5$, no spikes are observed:



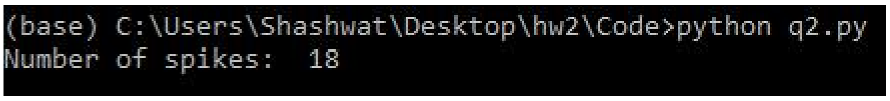
b) For $w_0 = 250$ and $\sigma_w = 25$, spikes are observed:



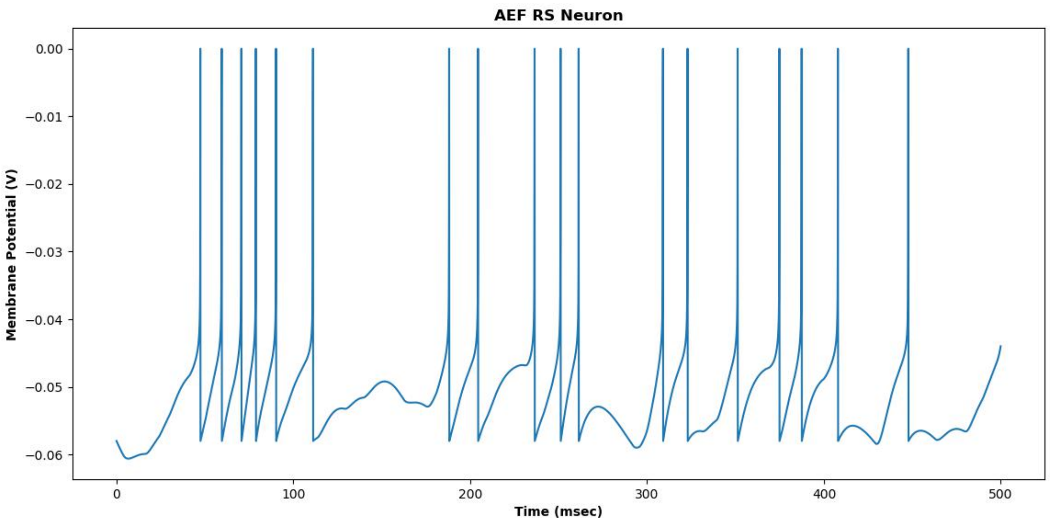
```
(base) C:\Users\Shashwat\Desktop\hw2\Code>python q2.py
Number of spikes:  27
```



From another run:



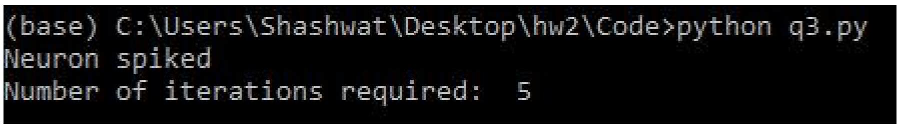
```
(base) C:\Users\Shashwat\Desktop\hw2\Code>python q2.py
Number of spikes:  18
```



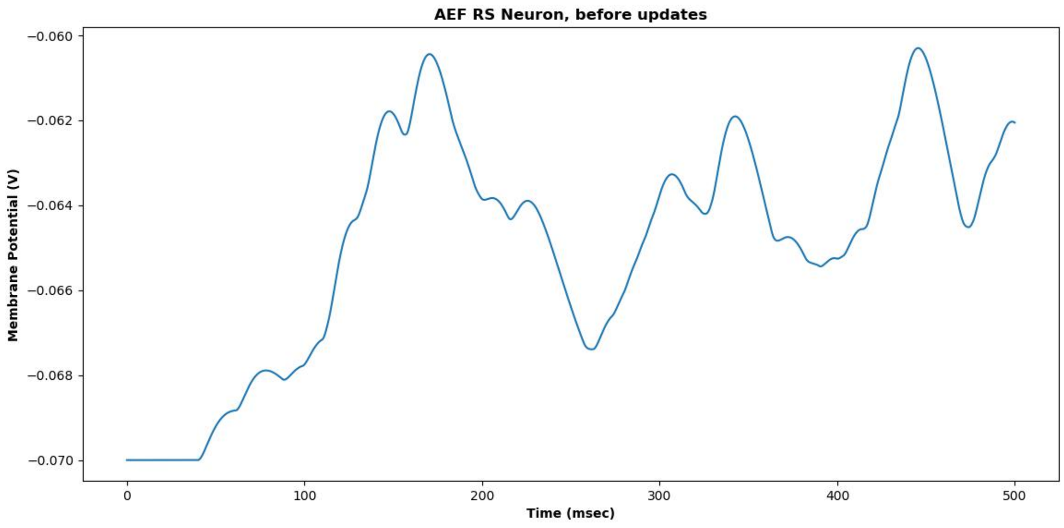
Question 3

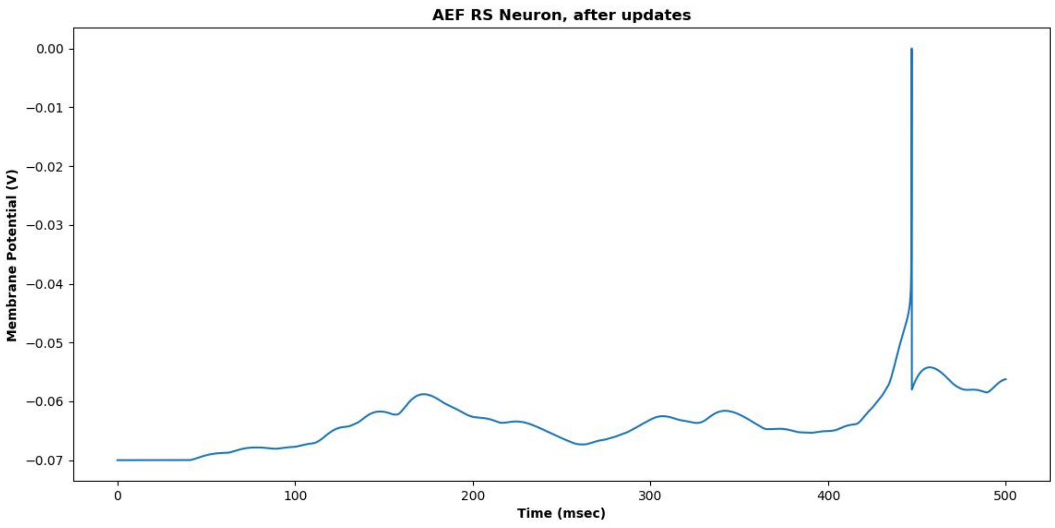
In this question, I have updated the weight for ***each*** synapse using the formula for Δw_k. The question is motivated by stating that the weight for only the presynaptic neuron that fired closest to the postsynaptic spike is updated. However, later in the question it is stated that the formula for updating should be used with each synapse, and hence there is an ambiguity in how the question has been stated.
I choose to go with the interpretation that each syanptic weight is updated, as in this scenario each weight can be seen as being updated in proportion to its contribution to the postsynaptic spike. I have however, for completeness sake also implemented code for the second interpretation and this can be found in the "Alternate" folder in the file "nq3.py".

Output for a run:

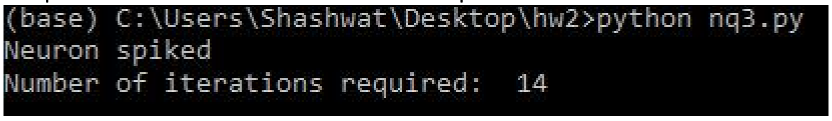


```
(base) C:\Users\Shashwat\Desktop\hw2\Code>python q3.py
Neuron spiked
Number of iterations required:  5
```

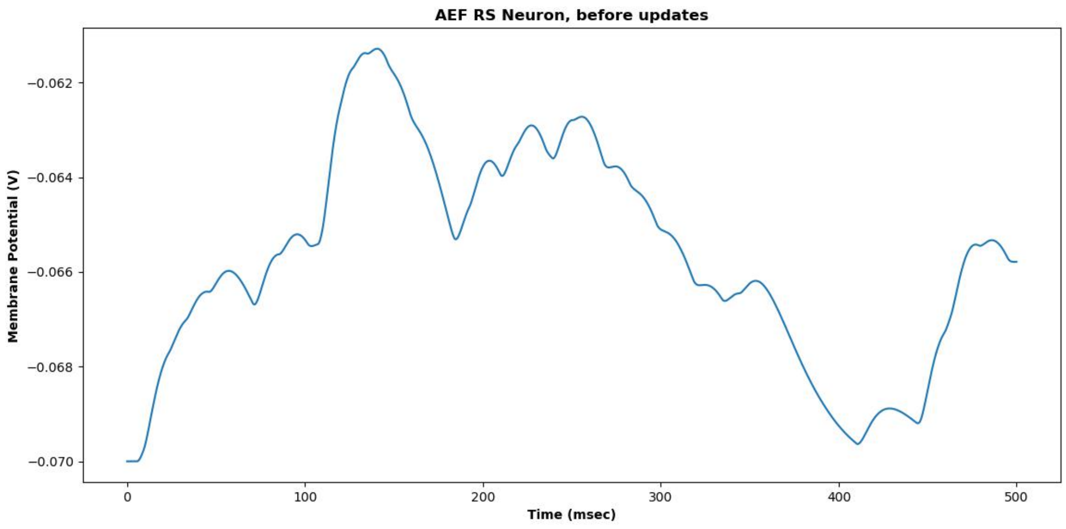


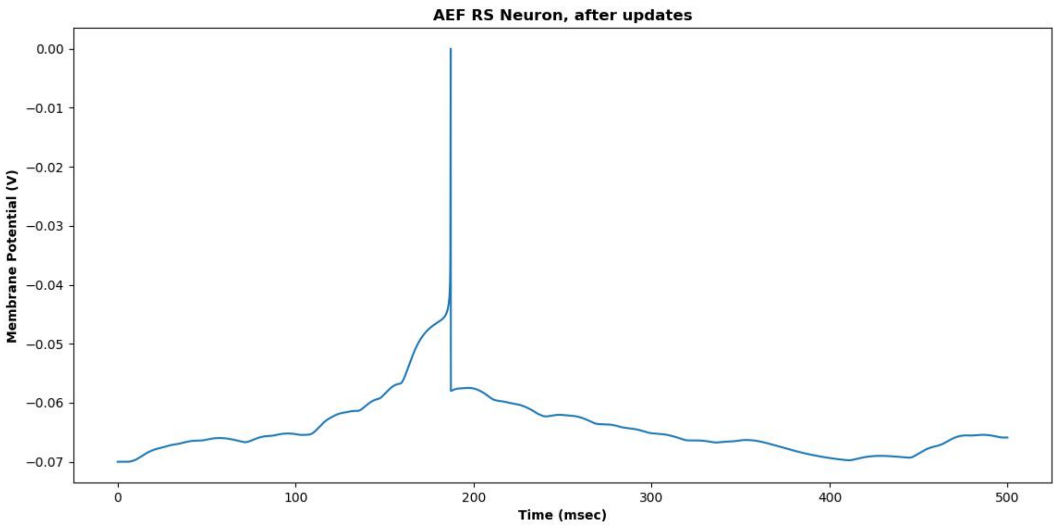


Output for a run with the alternative interpretation:



```
(base) C:\Users\Shashwat\Desktop\hw2>python nq3.py
Neuron spiked
Number of iterations required:  14
```



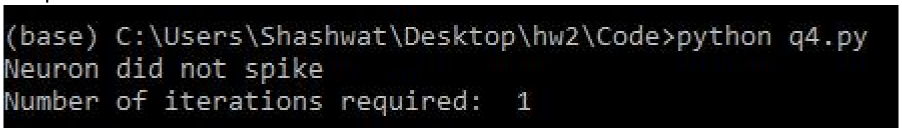


We hence see that the alternative rule also yields the desired outcome, but requires a larger number of updates.

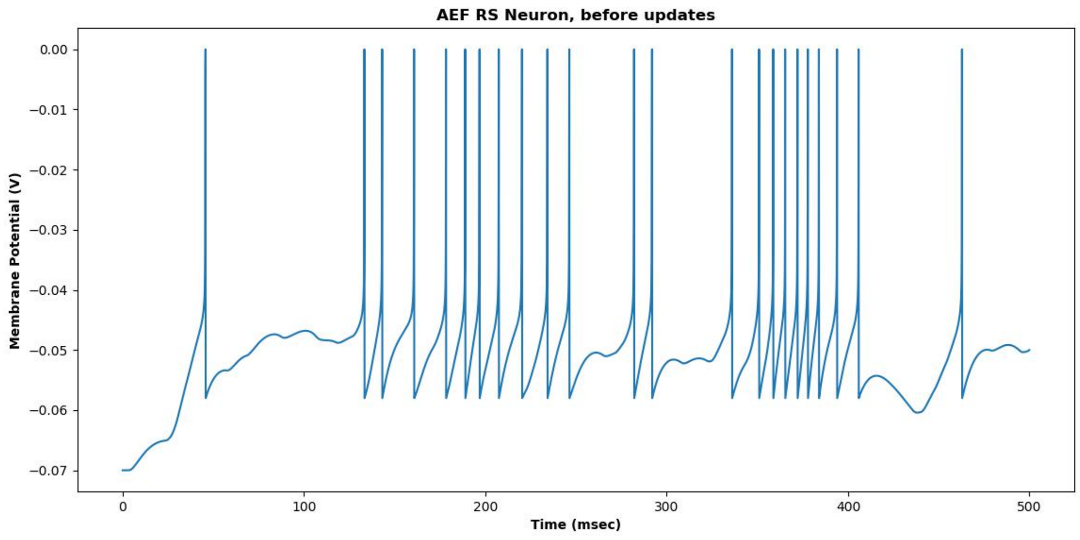
Question 4

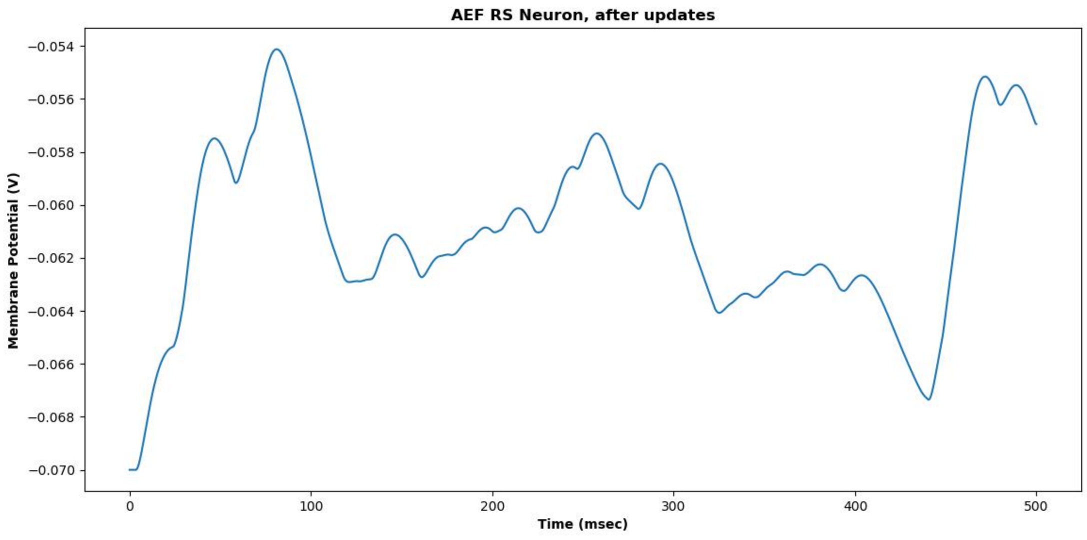
In this question (as in question 3 as well), I have updated the weight for ***each*** synapse using the formula for Δw_k . Each spike is treated independently and every synaptic weight is updated for each spike, and hence a given synaptic weight might be updated multiple times in one iteration. The alternative interpretation is that the weight for only the presynaptic neuron that fired closest to the postsynaptic spike is updated.
I choose to go with the interpretation that each syanptic weight is updated, as in this scenario each weight can be seen as being updated in proportion to its contribution to the postsynaptic spike. Perhaps even more importantly, the alternative algorithm in many cases fails to achieve the desired result (as I proceed to explain below). I have however, for completeness sake also implemented code for the second interpretation and this can be found in the "Alternate" folder in the file "nq4.py".

Output for a run:

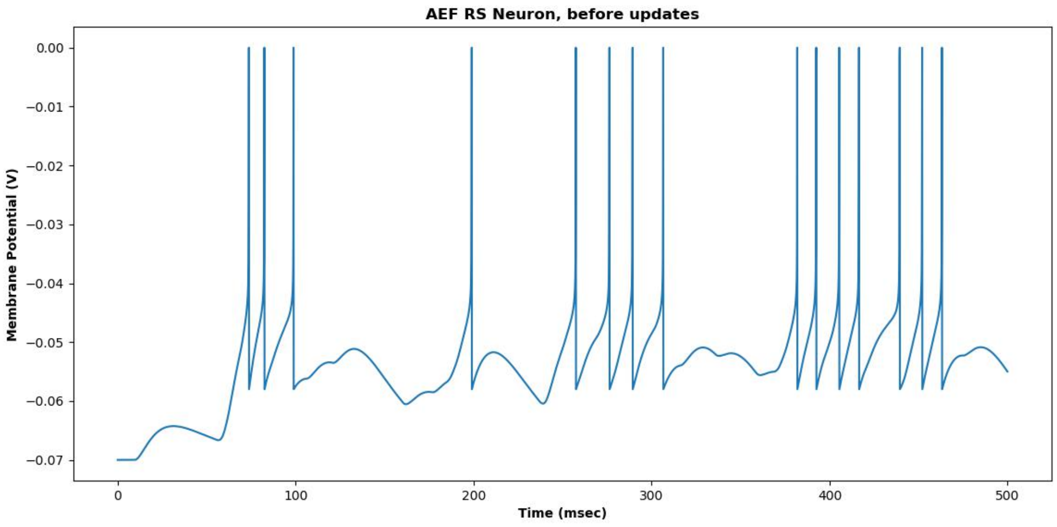


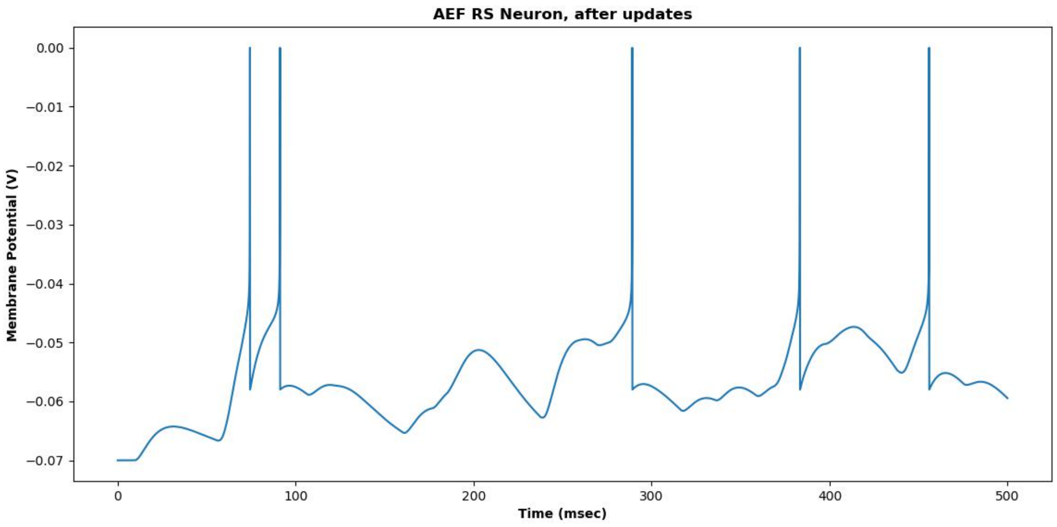
```
(base) C:\Users\Shashwat\Desktop\hw2\Code>python q4.py
Neuron did not spike
Number of iterations required:  1
```





Output for the alternative interpretation:



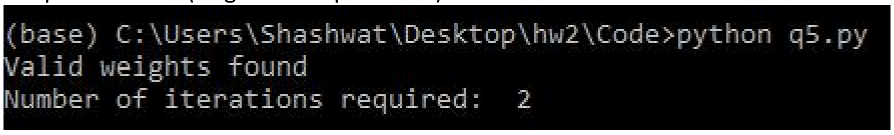


Note the this alternative algorithm failed in this case to suppress all the spikes. The explantion is as follows:
Let three presynaptic neurons have spiked very close to each other, such that the current from any two neurons itself suffices to make the postsynaptic neuron spike for the current set of synaptic weights. In this alternative algorithm, only the weight for one of these three presynaptic neurons is reduced (for the one that spikes last among these three). As the synaptic weights for the other two presynaptic neurons remains unchanged (due to this spike), this spike is not suppressed. Hence this alternate algorithm fails in such cases.

Question 5

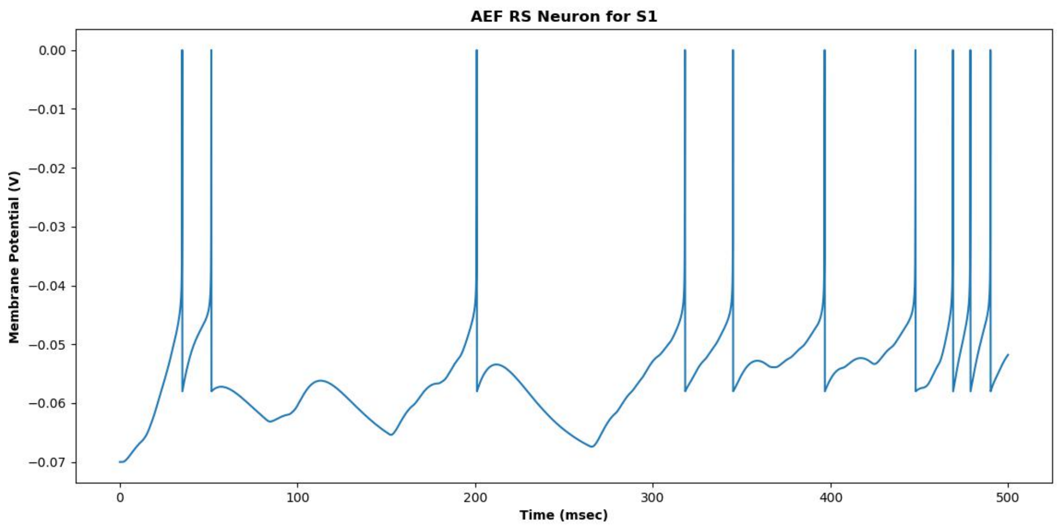
This question combines the algorithms implemented in questions 3 and 4. An alternate version for the algorithm in this question, that uses the alternate algorithms for questions 3 and 4 can be found in the "Alternate" folder in the file "nq5.py". However, the output from this alternate algorithm is not documented here as the alternate algorithm for question 4 fails for the values used in this question: $w_0 = 200$ and $\sigma_w = 20$.

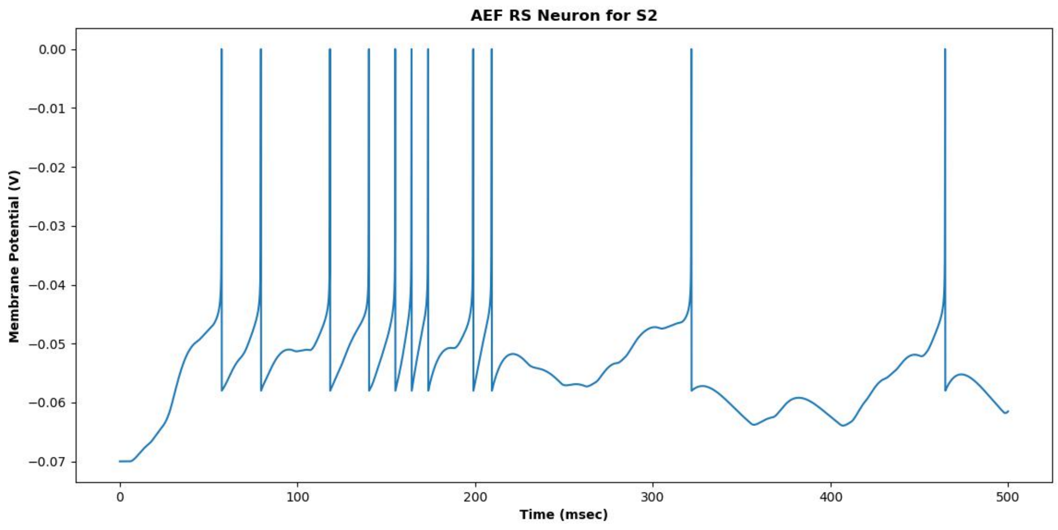
(a), (b) , (c) :
Output for a run (original interpretation):



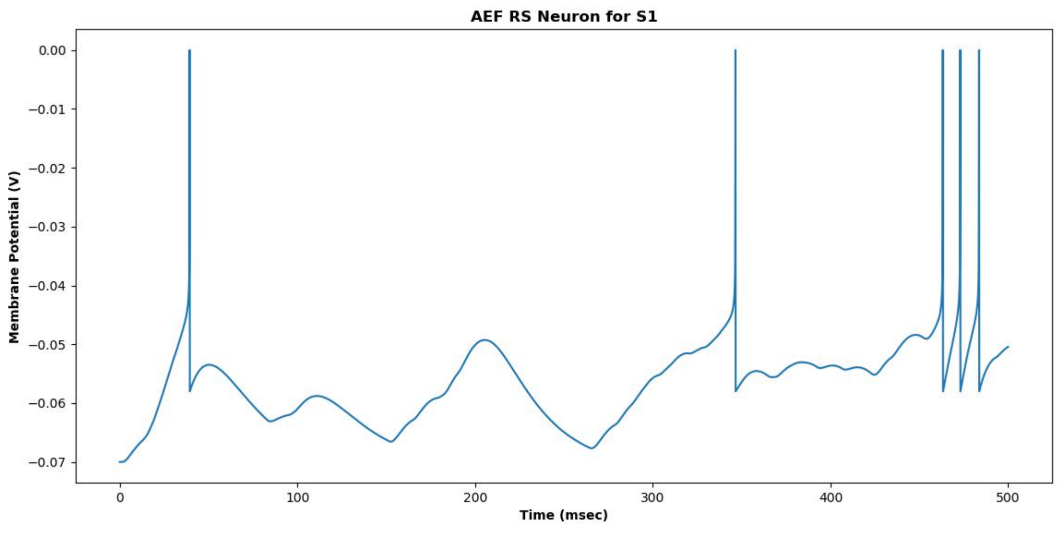
```
(base) C:\Users\Shashwat\Desktop\hw2\Code>python q5.py
Valid weights found
Number of iterations required:  2
```

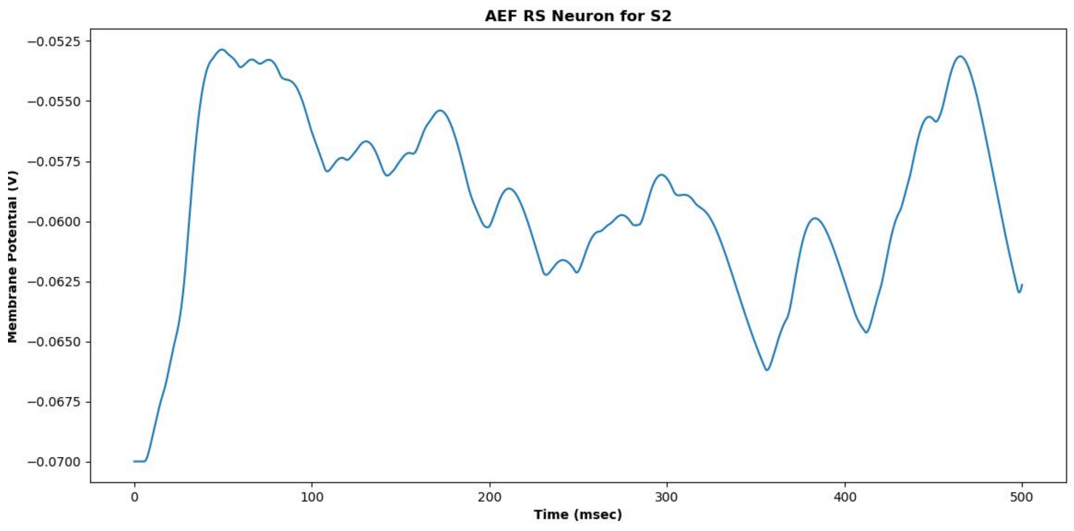
Before weight updates:





After updates:





d) This part can be solved by simply reversing the roles of S1 and S2, and thus the desired set of weights can be obtained by rerunning the code for updates with S1 and S2 swapped with each other (and some random initialisation of the synaptic weights).