Homework Assignment 2

**Part 1 – Binary search algorithm in Neuron**

A binary search algorithm starts at the middle element of an ordered array and checks whether the middle element is greater or less than the search element. If the elements match, usually the index or position is returned. Otherwise, this process is repeated on the upper or lower half of the ordered array based on whether or not the array is ordered in ascending or descending value.

For pseudo code of a binary search algorithm for an ascending array (i.e. array=[1 2 3]), the following recursive method can be used to identify the index of the search value given that the search value exists within the array.

binarySearch(array, value)

define midpoint

if midpoint==value

return index of midpoint

if midpoint > value

return binarySearch(array[0:midpoint],value)

if midpoint < value

return binarySearch(array[midpoint:end],value)

Unfortunately, it was difficult to implement a function in hoc that allowed for an array as an input so the following iterative method was used to determine the index of a value in an array.

// ---- ITERATIVE BINARY SEARCH ALGORITHM ----

// (implemented w/ zero-indexed programming)

vector = [0:0.1:10]

key = 4.3

minimumIndex = 0

maximumIndex = vector.size()-1

while (maximumIndex >= minimumIndex){

middleIndex = (maximumIndex - minimumIndex) / 2 + minimumIndex

middleValue = vector[middleIndex]

if (middleValue == key){ return middleIndex }

else if (middleValue < key) { minimumIndex = middleIndex + 1 }

else if (middleValue > key) { maximumIndex = middleIndex - 1 }

}

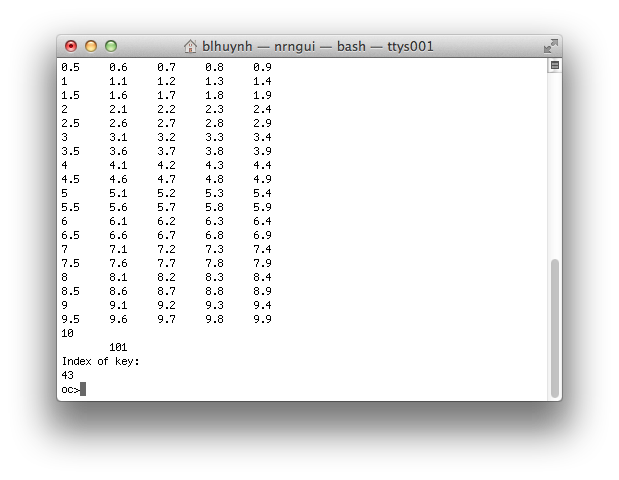


Figure : Binary Search Algorithm implemented in NEURON

**Part 2 – Model axon in NEURON**

Given – Fiber diameter

Diameter of node:

Length of myelin:

The internodal or axoplasmic resistance, , is calculated using the equation

where is the axoplasmic resistivity, is the internodal length, is the node diameter, and is the myelin diameter. Since consecutive nodes of Ranvier are separated by internodal spaces, the internodal resistance, , can represent these spaces if the myelin is assumed to be a perfect insulator and the internode is modeled as a tube of axoplasm. Therefore, the product of the axoplasmic resistivity and internodal length can be divided by the cross-sectional area of the nerve fiber to calculate the internodal resistance.

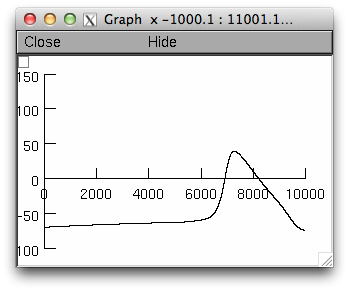


Figure : Vm(t) at 45th node

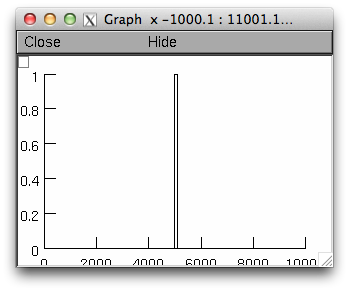


Figure : Istim(t)

**Part 3 – Intracellular threshold with PW = 0.1ms**

The binary search algorithm was paired with the modeled axon to determine the minimum threshold to stimulate the axon. Using a resolution of 1 pA, the minimum threshold was calculated to be . The upper bound of 1 nA was determined to be superthreshold and the final value of pA was also determined to be superthreshold.

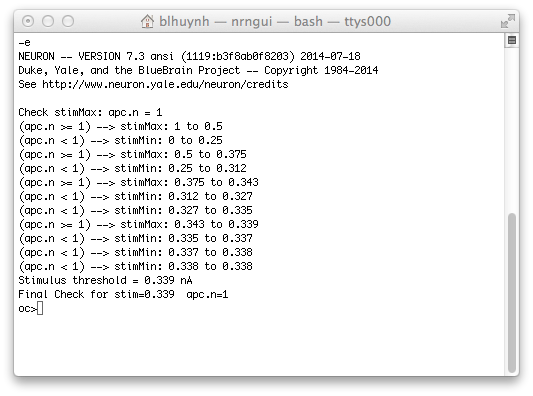


Figure : Threshold Finder via NEURON Binary Search Algorithm

**Part 4 – Extracellular Threshold with PW = 0.1 ms**

A binary search algorithm was applied to the same modeled axon but with extracellular stimulation with extracellular resistivity . A cathodic stimulus was also used with a pulse width of 0.1 ms from an electrode 1 mm away in the perpendicular direction from the middle node of Ranvier. A screenshot of NEURON finding the threshold to be pA is shown in Figure 5.

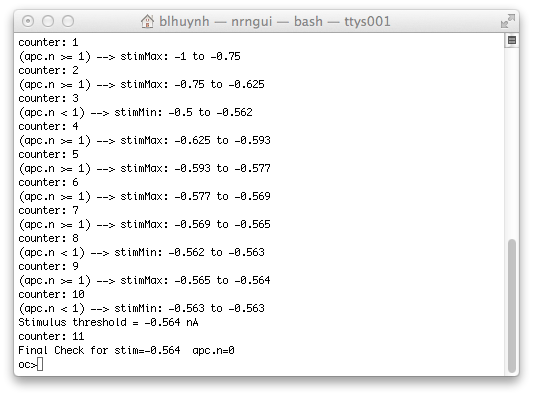
****

Figure : Extracellular Threshold with PW = 0.1 ms

**Code & Output**

**Part 1 – Binary Search Algorithm in NEURON** (binarySearchAlgorithm.hoc)

load\_file("nrngui.hoc")

func round(){

if ($1>0){

return int($1+0.5)

} else {

return int($1-0.5)

}

}

objref v1

proc createVector(){

startVal = $1

stopVal = $2

stepVal = $3

numel = (stopVal-startVal)/stepVal

v1 = new Vector(numel+1)

v1.x[0] = startVal

for i=1,numel{

v1.x[i] = v1.x[i-1]+stepVal

}

}

// Create list of numbers 0:0.1:10 to search from

createVector(0,10,0.1)

findValue = 4.3

v1.printf()

// ---- ITERATIVE binary search ----

proc iterative(){

key = $1

imin = 0

imax = v1.size()-1

while (imax >= imin){

imid = round((imax-imin)/2 + imin)

midValue = v1.x[imid]

// print imid

// print midValue

// print key

if (midValue == key){

// print imid

break

} else if (midValue < key) {

imin = imid + 1

} else if (midValue > key) {

imax = imid - 1

}

// print "\n"

}

if (midValue != key) {

print "Key not found."

} else {

print "Index of key: "

print imid

}

}

iterative(findValue)