Assignment 4

Meta

Author: Parmandeep Chaddha
Date: February 2, 2022

Objective

Program the integrate and fire model of the neuron using Julia.

Implementation

1. `integrate_and_fire.jl`

```
module IntegrateAndFire

using Plots

export IntegrateAndFireContainer, initializeIntegrateAndFire, fire,
plotIandF

struct IntegrateAndFireContainer
    deltaT::Float64
    capacitor::Float64
    resistor::Float64
    threshold::Float64
    spikeValue::Float64
    injectionCurrent::Float64
    injectionTimes::Tuple{Float64,Float64}
    times::Vector{Float64}
    voltages::Vector{Float64}
    currents::Vector{Float64}
end


"""
initializeIntegrateAndFire()

Initializes the integrate and fire structure.
    deltaT::Float64 - The time step
    startTime::Float64 - Start of the IandF calculations.
    injectionStartTime::Float64 - Start of the current injection.
    injectionEndTime::Float64 - End of the current injection.
    capacitor::Float64 - Value of the membrane capacitance.
    resistor::Float64 - Resistance of the membrane.
    threshold::Float64 - The minimum value that must be reached before
"fire".
    spikeValue::Float64 - The value to which the neuron fires before
depolarizing.
```

```
    injectionCurrent::Float64 - The current, in A, to be injected.
    initialVoltage::Float64 - The potential, in V, across the membrane at `startTime`.
"""
function initializeIntegrateAndFire(
    deltaT::Float64,
    startTime::Float64,
    injectionStartTime::Float64,
    injectionEndTime::Float64,
    capacitor::Float64,
    resistor::Float64,
    threshold::Float64,
    spikeValue::Float64,
    injectionCurrent::Float64,
    initialVoltage::Float64,
)
    injectionTimes = injectionStartTime, injectionEndTime
    voltages = [initialVoltage]
    times = [startTime]
    currents = (injectionStartTime > startTime) ? [0] : [injectionStartTime]

    iAndF = IntegrateAndFireContainer(
        deltaT,
        capacitor,
        resistor,
        threshold,
        spikeValue,
        injectionCurrent,
        injectionTimes,
        times,
        voltages,
        currents
    )

    return iAndF
end


"""
fire

Fire the nueron specified in the `IntegrateAndFireContainer` for a specified `runTime`.
"""
function fire(iAndF::IntegrateAndFireContainer, runTime::Float64)
    if runTime < (iAndF.times[end] + iAndF.deltaT)
        ErrorException("The run time must be larger than the startTime + deltaT.")
    end

    currentTime = iAndF.times[end]
    totalTime = runTime + iAndF.deltaT
    while (currentTime <= totalTime)
```

```
        currentTime += iAndF.deltaT
        _appendToList(currentTime, iAndF.times)
        _updateCurrent(iAndF)
        _updateVoltage(iAndF)
    end

end


"""
plotIandF
Plot either the voltage, 'voltage', or current, 'current', against time.
"""
function  plotIandF(iAndF::IntegrateAndFireContainer, which::String = "voltage")
    if which == "voltage"
        y = iAndF.voltages
        yLabel = "Voltage (V)"
    elseif which == "current"
        y = iAndF.currents
        yLabel = "Current (A)"
    else
        ErrorException("`which` shoud either be `voltage` or `current` not $(which)")
    end

    plot(iAndF.times, y)
    plot!(
        title="$(yLabel) versus Time (s) for Neuron Modelled Using `IntegrateAndFire`",
        ylabel=yLabel,
        xlabel="Time (s)",
        titlefontsize=8,
    )
    savefig("./assignment4/$(yLabel)_integrate_and_fire.png")
end


"""
_appendToList
Append a `value` to a `list`. Wrapper around the push function.
"""
function _appendToList(value, list)
    push!(list, value)
end


"""
_updateCurrent
Update the current.
"""
function _updateCurrent(iAndF::IntegrateAndFireContainer)
    # Determine whether there is current being injected or not.
    injectionCurrent = 0
    if (iAndF.times[end] > iAndF.injectionTimes[1]) &&
(iAndF.times[end] < iAndF.injectionTimes[2])
        injectionCurrent = iAndF.injectionCurrent
    end
    _appendToList(injectionCurrent, iAndF.currents)
end


"""
_dVdt
Calculate the derivative of voltage with respect to time.
"""
function _dVdt(res::Float64, cap::Float64, current::Float64,
voltage::Float64)
    return ( 1 / (res*cap) ) * (res*current - voltage)
end


"""
_updateVoltage
Calculate and update the voltage for the current time point.
"""
function _updateVoltage(iAndF::IntegrateAndFireContainer)
    lastVoltage = iAndF.voltages[end]
    tolerance = 1.e-1

    # If the last voltage is near or above the `spikeValue`, reset
voltage to 0.
    if abs(lastVoltage - iAndF.spikeValue) < tolerance
        voltage = 0
    elseif (lastVoltage > iAndF.threshold)
        voltage = iAndF.spikeValue
    else
        voltage = lastVoltage + _dVdt(iAndF.resistor, iAndF.capacitor,
iAndF.currents[end], lastVoltage)*iAndF.deltaT
    end
    _appendToList(voltage, iAndF.voltages)
end

end # module
```

Running the Script

In order to run the script, navigate to the `juliaPsych420` folder and run `julia --project=.` to activate the terminal. Then run the file titled `assignment4_script` using `include ("assignment4/assignment4_script.jl")`.

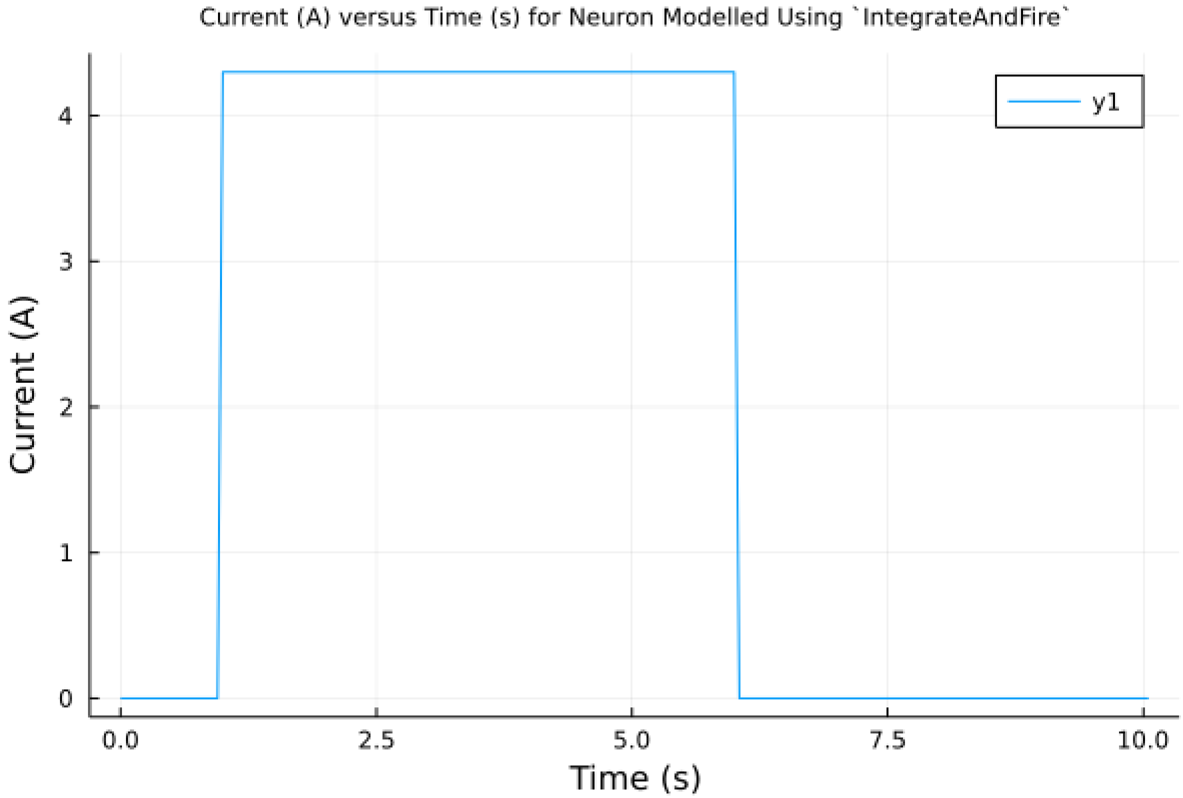
1. That file looks like:

```
include("integrate_and_fire.jl")

using .IntegrateAndFire

function run()
    deltaT = 0.05 # every 100 ms
    startTime = 0. # seconds
    injectionStartTime = 1.
    injectionEndTime = 6.
    capacitor = 1.
    resistor = 2.
    threshold = 3.
    spikeValue = 8.
    injectionCurrent = 4.3
    initialVoltage = 0.


    iAndF = initializeIntegrateAndFire(deltaT, startTime,
injectionStartTime,
        injectionEndTime, capacitor, resistor, threshold, spikeValue,
        injectionCurrent, initialVoltage
    )

    runTime = 10.0

    fire(iAndF, runTime)

    plotIandF(iAndF, "voltage")
    plotIandF(iAndF, "current")

end
```

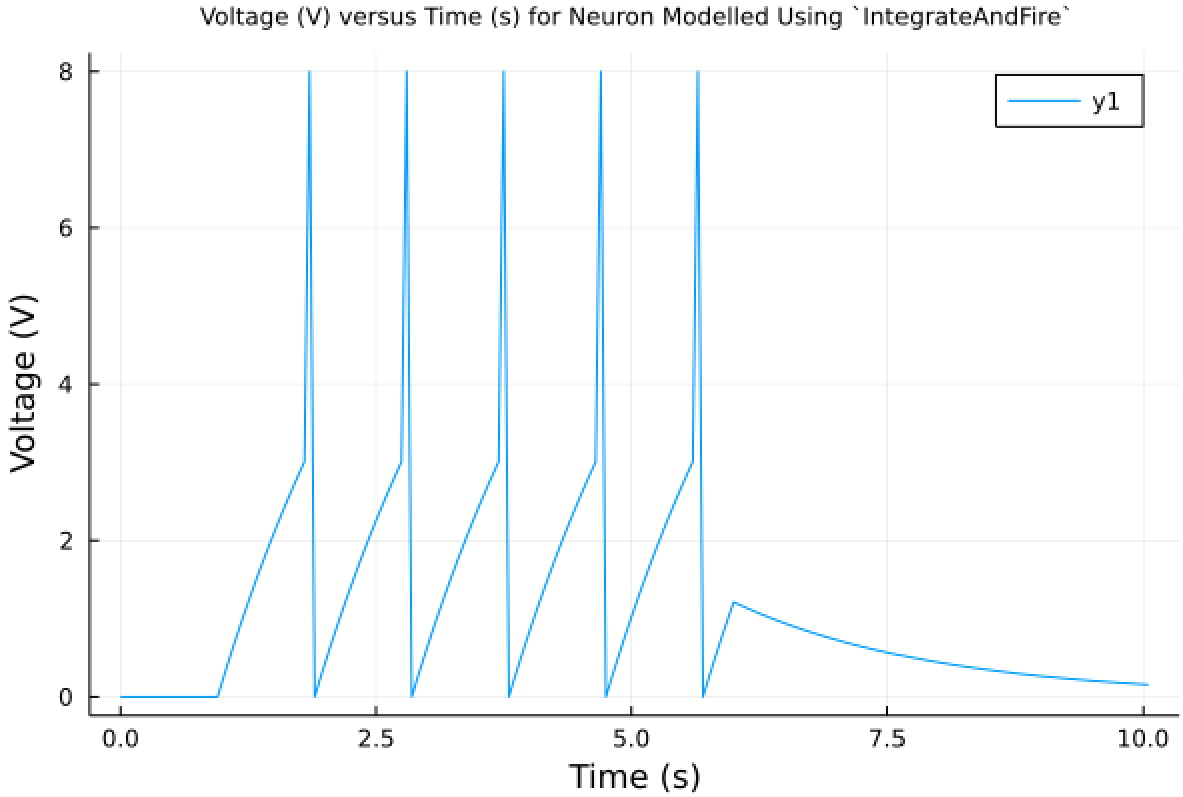
Results

The output current and voltage plots match the plots given in the lectures.

1. Current versus Time



2. Voltage versus Time


Voltage (V) versus Time (s) for Neuron Modelled Using `IntegrateAndFire`
y1
Voltage (V)
Time (s)