Computational Neuroscience assignment 1

303034908

March 1, 2019

1 Hodgkin Huxley Model [10]

Neuronal action potentials are driven by the interaction between the ionic gradient and the electrical gradient of the Na+ and K+ ions across the cell membrane. Each ion has its own resting potential when the two gradients balance. Each ion channel has both voltage and time dependencies.

For large depolarisations (10-15mV), Na+ ions flow in quickly causing more depolarisation and more Na+ channel opening. This causes the rising part of the action potential. After a few milliseconds the Na+ channels close and the K+ channels open, causing the hyperpolarisation, undershoot and return to resting membrane potential.

This pattern can be described by four main differential equations. The first one, dV/dt is shown below:

$$c_m \frac{dV}{dt} = -g_L(E_L - V) + g_{Na} m^3 h(E_{Na} - V) + g_K n^4 (E_K - V) + I_e/A$$

where c_m is the specific membrane capacitance, g_K, g_{Na} and g_L are the specific conductances for each ion, where L is the leaky current, E_L, E_{Na} and E_L are the reversal potential for each ion, and I_e/A is the injected current to initialise the action potential. The other variables m, n, and h are termed the gating variables and themselves have differential equations to describe their changing values over voltage and time. These equations can be found in chapter 5 of TN by Dayan Abbott (esp. 5.6, 5.17-5.19, 5.22, 5.24 and 5.25).

Figure 1 shows the voltage V, and the gating variables m, n and h as a function of time with the following initial values: V = -65 mV, m = 0.0529, h = 0.5961, n = 0.3177; and following parameters: I_e/A = 200 nA/mm^2, cm = 10 nF mm^{-2}, E_Na = 50 mV, E_k = -77 mV, E_l = -54 mV, g_Na = 1200 mS mm^{-2}, g_k = 360 mS mm^{-2}, g_l = 3 mS mm^{-2}.

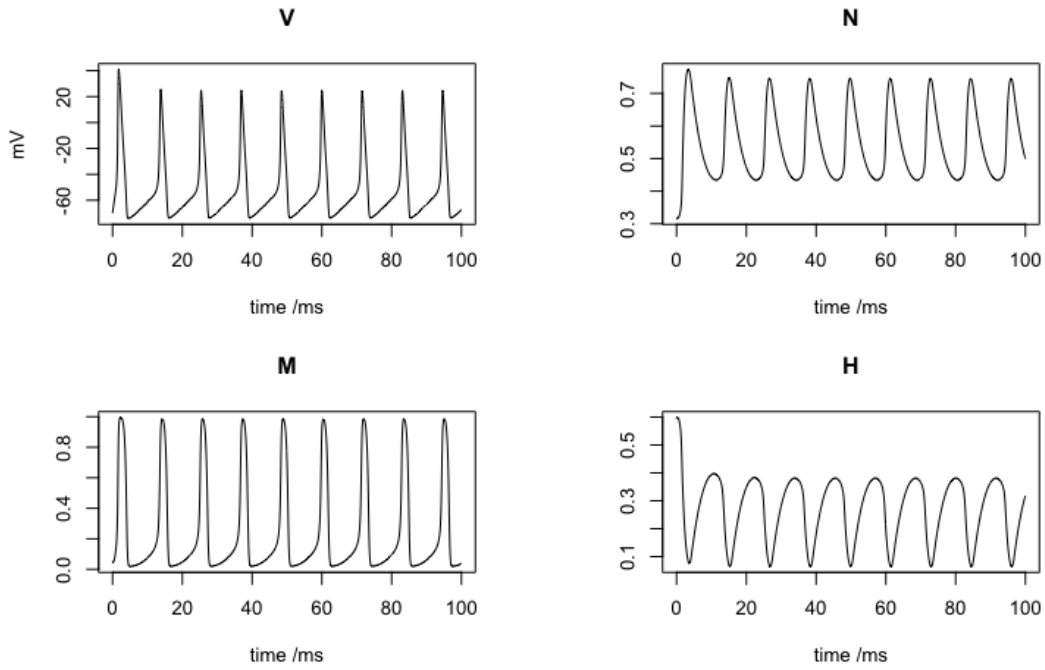

Figure 1: Hodgkin Huxley model of neuron with parameters specified above and $I_e/A = 200nA/mm^2$

By varying the external current from 0 to 500 nA/mm^2 we can see in figure 2 how the firing rate jumps discountinuously from zero to 60 Hz when the current passes through the minimum value required to produced sustained firing at around 75 nA/mm^2.

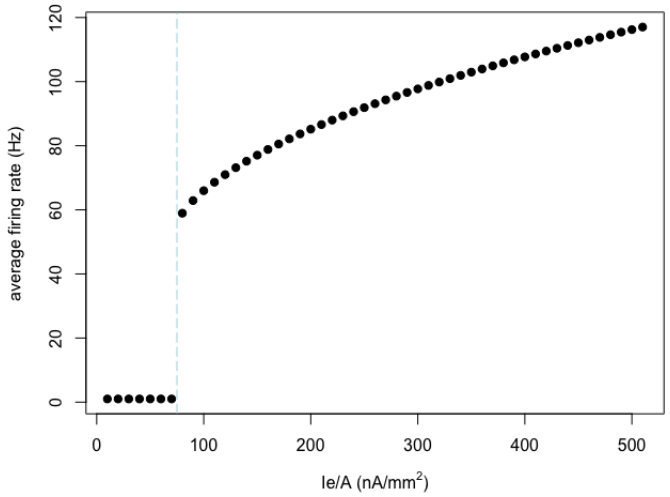

Figure 2: Firing rate as function of external current

Finally, figure 3 shows the trace for firing rate when I_e/A is held at -50nA/mm^2 for 5ms followed by a shift to 0 nA/mm^2. By simply priming the neuron to being very sensitive to new input currents, the small shift in Ie/A back to 0 was able to cause the membrane potential to rise to threshold and cause an action potential at 10ms from the start.

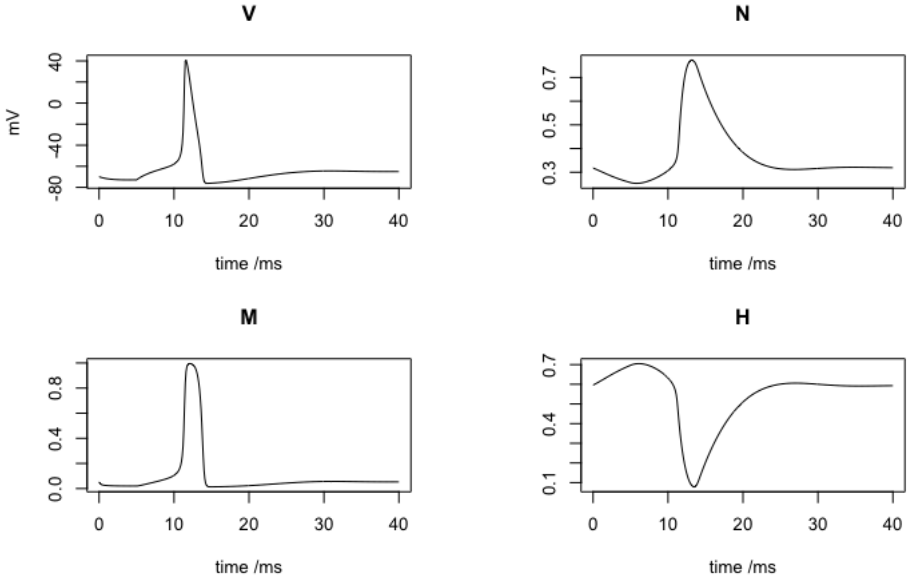

Figure 3: Stimulation from 0 to 50 at 5ms

2 Coupled integrate and fire neurons [15 marks]

A model of two coupled integrate-and-fire neurons was built. Both neurons obeyed the following equations:

$$\tau_m \frac{dV}{dt} = E_L - V - r_m \bar{g}_s P_s (V - E_s) + R_m I_e$$

with $E_L = -70mV$, $V_{th} = -54mV$, $V_{reset} = -80mV$, $\tau_m = 20ms$, $r_m \bar{g}_s = 0.15$ and $R_m I_e = 18mV$.

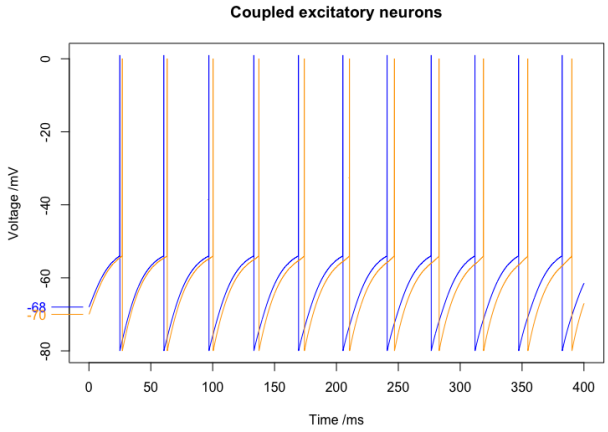

Figure 4: Coupled excitatory neurons starting from similar voltage values but gradually shifting to an out-of-synch pattern of firing

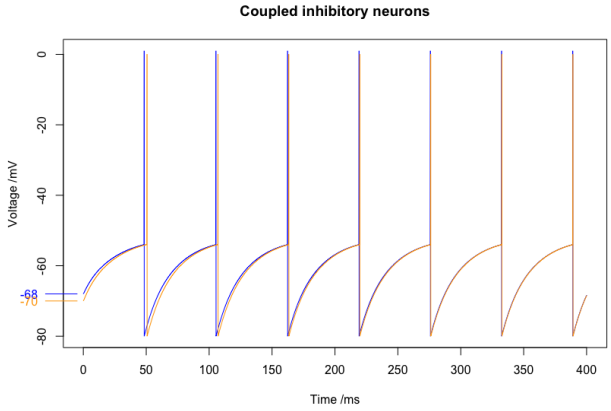

Figure 5: Coupled inhibitory neurons starting from similar voltage values and gradually shifting to an in-synch pattern of firing

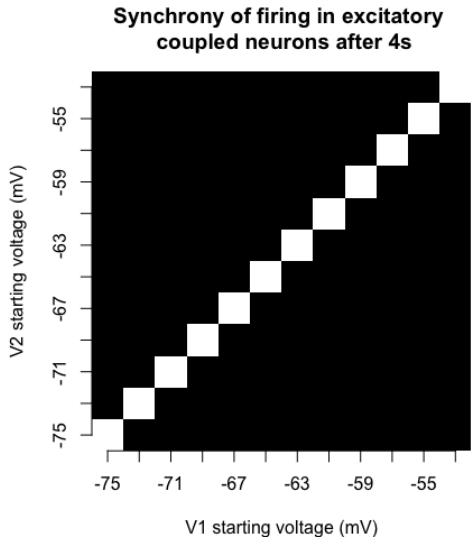

Figure 6: Coupled excitatory neurons starting from similar voltage values but gradually shifting to an out-of-synch pattern of firing

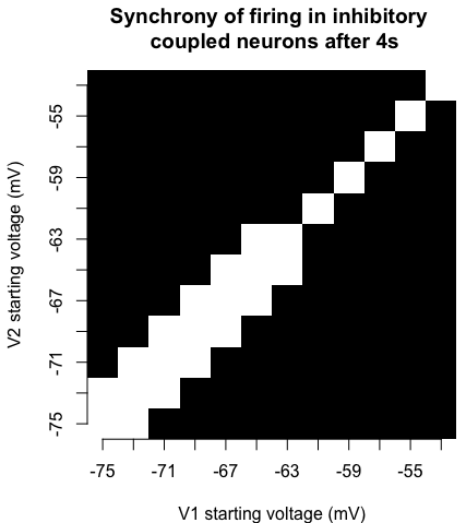

Figure 7: Coupled inhibitory neurons starting from similar voltage values and gradually shifting to an in-synch pattern of firing

To incorporate multiple presynaptic spikes, P_s was described by differential equations below:

$$\tau_s \frac{dPs}{dt} = exp(1)P_{max}z - P_s$$

$$\tau_s \frac{dz}{dt} = -z$$

with $\tau_s = 10ms$ and $P_{max} = 0.5$ and the additional rule that z is set to 1 whenever a presynaptic spike arrives.

Figure 4 and 5 show how the firing pattern is dependent on the type (excitatory or inhibitory) or the reciprocal synaptic connections: for the same parameters and same initial voltages, the excitatory coupled neurons gradually become asynchronous, whilst the inhibitory neurons gradually become synchronous.

To investigate the different combination of parametres, such as altering E_s between excitatory and inhibitory, the initial membrane voltage for each neuron and the strengths and time constants of the reciprocal synapses, a number of simulations were run.

Figures 6 and 7 show the synchronicity of firing for different starting voltages. Synchronicity, in white, is defined by taking the last spike in a 4 second period for each neuron and checking if they are within 0.5ms of each other.

Figures 9 and 8 show the firing rates for different values of τ_m and τ_s, the time constants for each neuron for excitatory and inhibitory coupling. As shown, the combination of low τ_m and τ_s give the highest firing rates.

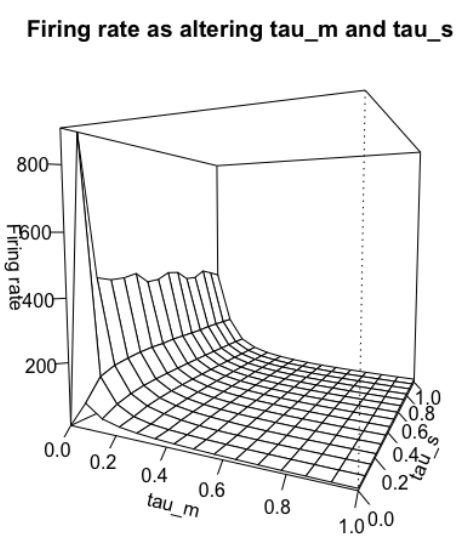

Figure 8: Firing rate of excitatory neurons with varying τ_m and τ_s values

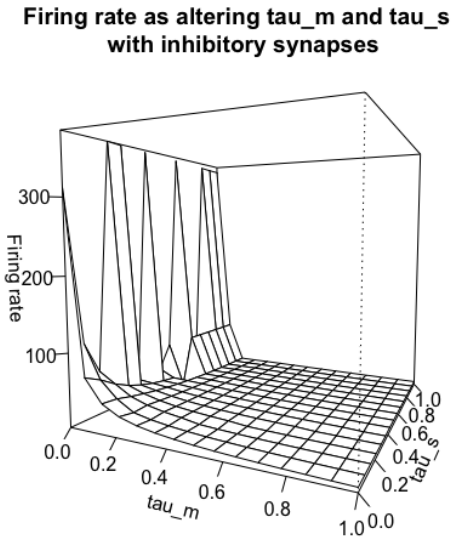

Figure 9: Firing rate of inhibitory neurons with varying τ_m and τ_s values

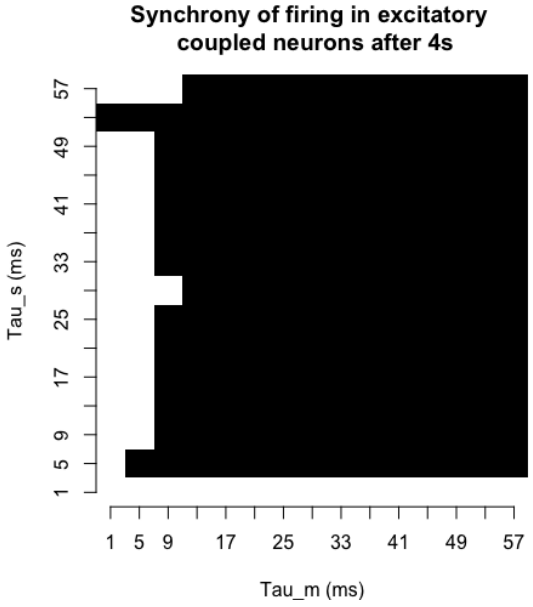

Figure 10: Synchrony of firing in excitatory neurons while varying τ_m and τ_s

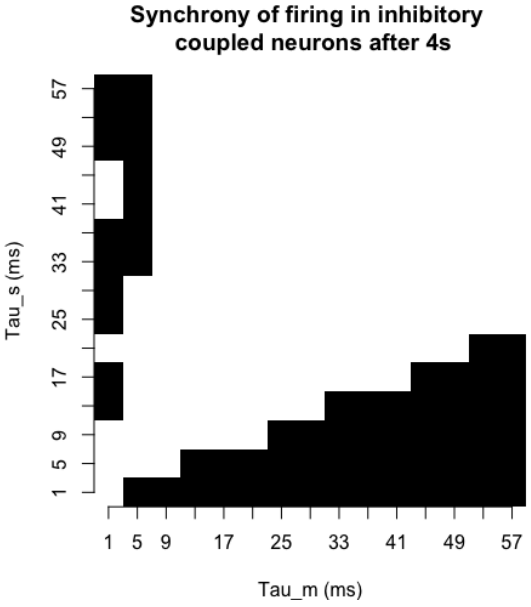

Figure 11: Synchrony of firing in inhibitory neurons while varying τ_m and τ_s

Figures 11 and 10 also show the synchronicity of firing for varying values of τ_m and τ_s for excitatory and inhibitory synapses. In the excitatory system, the low τ_m leads to synchronicity of firing even when usually this is not the case in excitatory

Finally, the effect of synaptic strength, $r_m g_s$ on firing rate and synchronicity was investigated. The results are shown in figures 12 and 13. From these figures we can see how in the excitatory system, increasing synaptic strength simply increases the firing rate in the system. Instead, in the inhibitory system, the firing rate does not change continuously. Moreover, during $0.5 < r_m g_s 1$, the firing rate of V1 is 0 whilst V2 continues at 19 Hz, showing a much more complex dynamic than in the excitatory system.

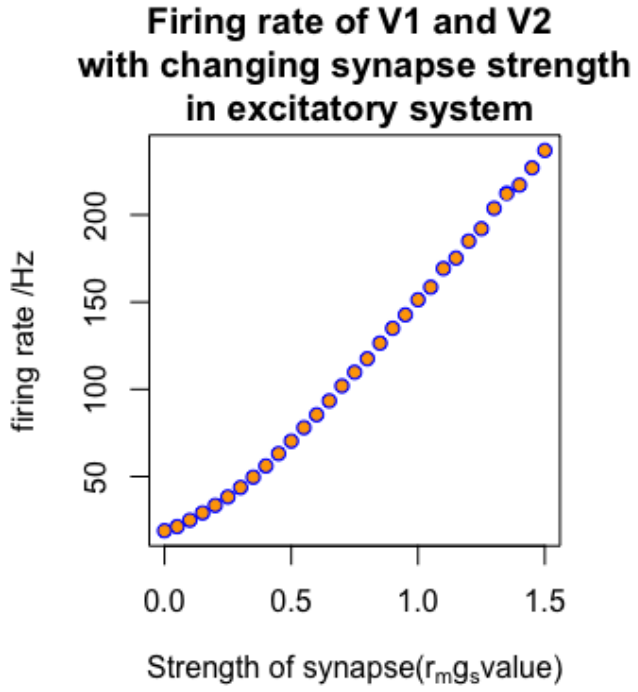

Figure 12: Firing rate for V1 (blue) and V2 (orange) as synaptic strength is increased

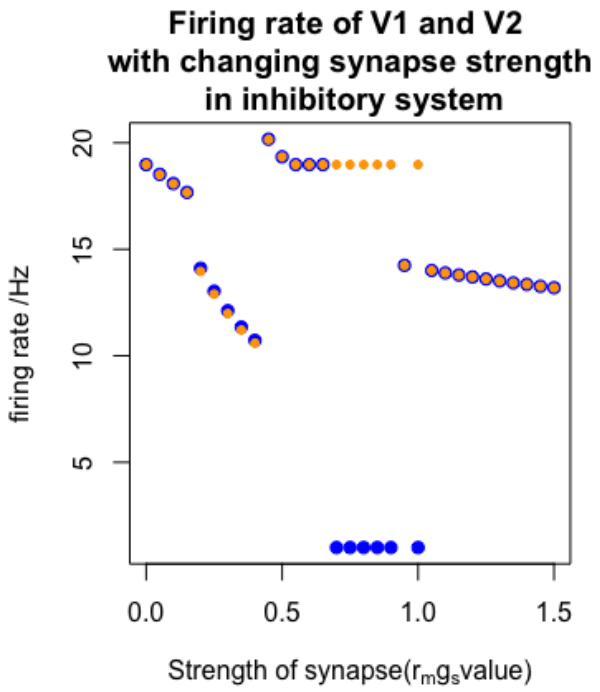

Figure 13: Firing rate for V1 (blue) and V2 (orange) as synaptic strength is increased

3 Networks [5 marks]

A simple model of oscillations arising from interaction of excitatory and inhibitory populations of neurons was built. The firing rate of the excitatory neurons, v_E, and of the inhibitory neurons v_I were characterised by the following equations:

$$\tau_e \frac{dv_E}{dt} = -v_E + [M_{EE}v_E + M_{EI}v_I - \gamma_E]_+$$

and

$$\tau_e \frac{dv_I}{dt} = -v_I + [M_{II}v_I + M_{IE}v_E - \gamma_I]_+$$

The following parameters fixed ($M_{II} = -1$, $M_{EE} = 1.25$, $M_{IE} = 1$, $M_{EI} = -1$, $\gamma_E = -10Hz$, $\gamma_I = 10Hz$).

This model exhibits both static (constant v_E and v_I) and oscillatory activity depending on the values of its parameters. Values of v_E and v_I for which the $\frac{dv_E}{dt}$ and $\frac{dv_I}{dt}$ equal zero can be depicted in a phase-plane as nullclines: locations where each voltage remains constant.

At the point of intersection of these nullclines both dynamic variables are constant, and the system is static. It is, however, critical whether the fixed point of intersection is stable or unstable. If the point is stable, values of v_E and v_I close to the fixed point will be drawn towards it over time. If the intersection point is unstable, nearby configurations are pushed away from it and are only transiently constant.

To assess the stability of this system, three steps need to be taken:

1. Find steady states (intersection of nullclines)
2. Linearise: find Jacobian matrix at steady state
3. Compute eigenvalues: - fixed point is stable if and only if the real part of the eigenvalue is less then 0 ($Re(\lambda) < 0$) - otherwise the fixed point is unstable.

For 2-d systems, stability can also be confirmed if the determinant of the Jacobian is less than 0 and the trace of the Jacobian is greater than 0.

In this system, the eigenvalues are given by:

$$\lambda = 0.5\left(\frac{M_{EE}-1}{\tau_E} + \frac{M_{II}-1}{\tau_I} \pm \sqrt{\left(\frac{M_{EE}-1}{\tau_E} - \frac{M_{II}-1}{\tau_I}\right)^2 + \frac{4M_{EI}M_{IE}}{\tau_E\tau_I}}\right)$$

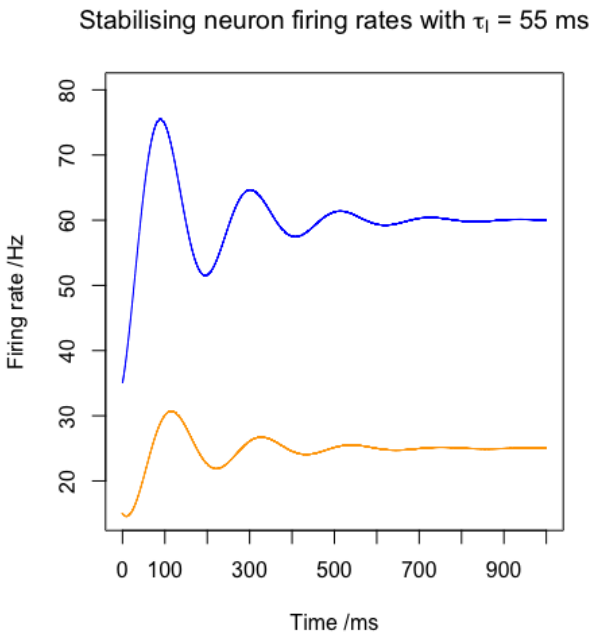

Figure 14: Firing rate of v_E in blue and v_I in orange over time for a stable system.

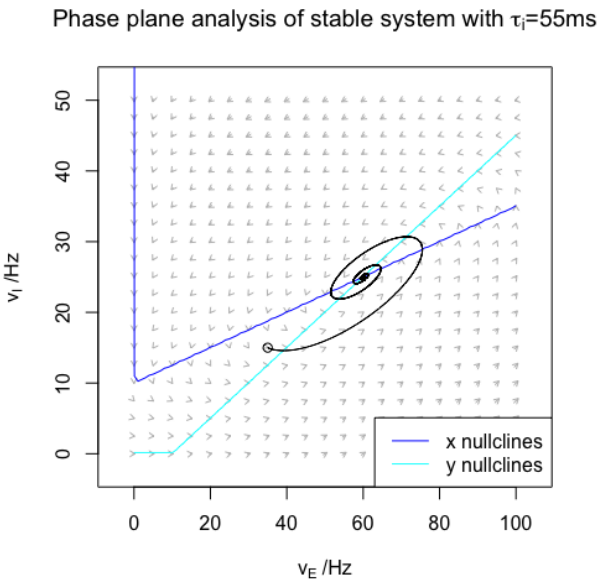

Figure 15: Phase plane analysis of network. Starting at point $v_E = 35$, $v_I = 15$, the attraction pulls the firing rates to the fixed points at $v_E = 60$, $v_I = 25$.

If the factor inside the square root is negative, the square root is imaginary and the eigenvalues form a complex conjugate pair. In this case, we only need to check the real part of the eigenvalues to determine its stability. By changing values of τ_I from 0 to 200, this real value shifts from negative to positive at $\tau_I = 80$.

This indicates that the fixed point is stable if $\tau_I <= 79$ but unstable if greater.

Arrows show direction of gradients over different v_E and v_I values. X nullcline is for v_E, Y nullcline for v_I.

The firing rate and phase plane analysis for stable and unstable networks are shown in figures 14, 15, 16, and 17.

It is worthwhile noting that the further away τ_I is from the threshold of 80ms, the stronger the attraction or repulsion towards the fixed point. Figure 18, shows an example where τ_I is very close to threshold (at 79ms), but the starting point for the trajectory is still far from the nullcline intersection. In this scenario the trajectory takes much longer to reach the fixed point (only 10 seconds of simulated dynamics are shown hence the gap in the middle).

4 Hopfield Network [20]

A Hopfield network with binary units of 400 neurons was built and the following questions were investigated:

1. Storage capacity: How many patterns can it store? How does the sparseness (fraction of units set to +1 rather than -1) of the pattern affect this result?

2. Robustness: how is pattern recall affected by the random loss of weights?

When one of the trained patterns is inputted into a Hopfield network and the output is different, then we know that the network is beginning to be non-functional as even the learned patterns are no longer stable energy minima. Therefore, a function was built to test the network 100 times with a pattern that it had learned, and checking if the output was exactly the same as the input. Counting the successful results out of 100 is the main way I have quantified the success of the network.

The storage capacity was first investigated for a single value of sparseness and increasing number of patterns learned, as shown in figure 19, and then for multiple values of sparseness as shown in figures 20 and 21. As we can see from these plots, an initial breaking of the network is seen as the number of patterns approaches 0.18 times the size of the network. Secondly, low sparseness and high sparseness both decreased the ability of the network to correctly output its learned patterns, with the optimal sparseness being at 50%.

The robustness was initially investigated for a fixed number of patters learned (0.1* size of network (I)) with increasing losses in weights, as shown in figure 22, and then for increasing number of patterns learned, as shown in figures 23 and 24. As shown in these plots, the robustness of the network starts to decrease as around 37% of the weights are lost. Secondly, both increasing the loss of weights increases the damage and

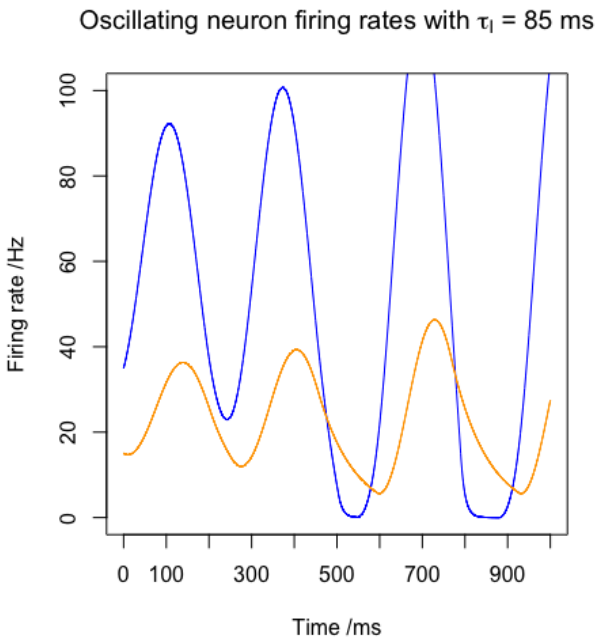

Figure 16: Firing rate of v_E in blue and v_I in orange over time for an unstable system.

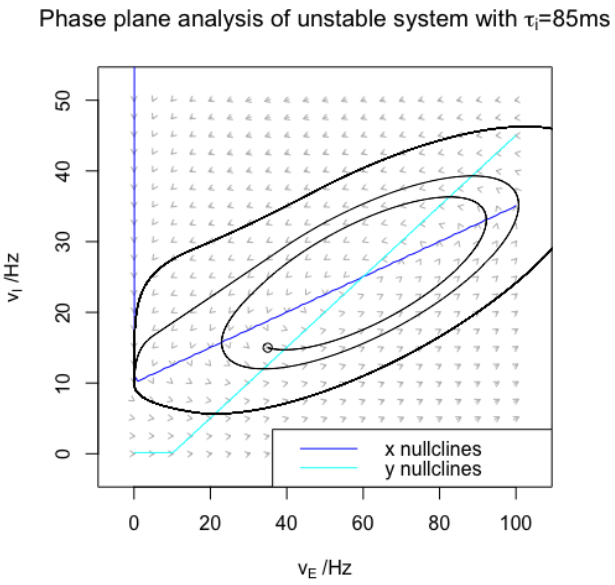

Figure 17: Phase plane analysis of network. Starting at point $v_E = 35$, $v_I = 15$, the repulsion pulls the firing rates away from the fixed points.

difficulty in learning new patterns. For example, with a 70% loss of weights the network was already loosing accuracy with 11 patterns.

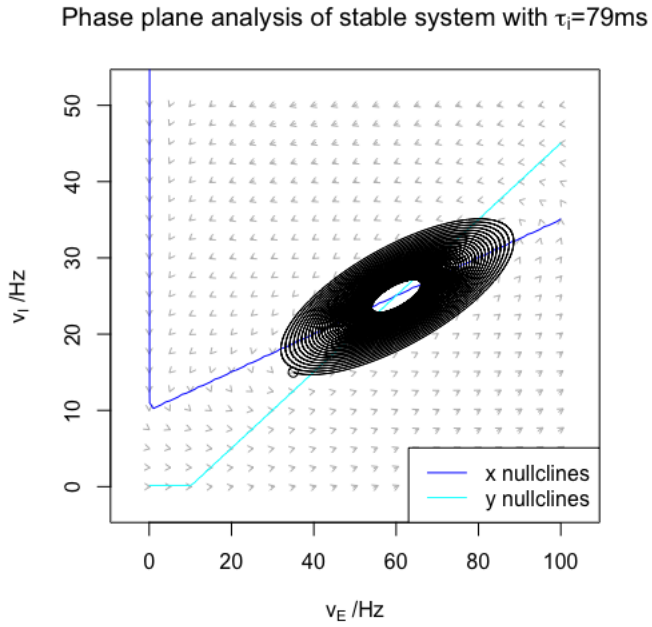

Figure 18: Phase plane analysis where τ_I is close to threshold and starting point is too far for successful attraction.

5 Appendix

```
library(deSolve)
install.packages("pracma")
library(pracma)
require(plot3D)
library(phaseR)


HH.time <- function(t,state, parameters){
  with(as.list(c(state, parameters)), {

    alpha.n <- function(V) (0.01*(V+55))/(1-exp(-0.1*(V+55)))
    beta.n <- function(V) 0.125*exp(-0.0125*(V+65))

    alpha.m <- function(V) (0.1*(V+40)) / (1-exp(-0.1*(V+40)))
    beta.m <- function(V) 4*exp(-0.0556*(V+65))

    alpha.h <- function(V) 0.07*exp(-0.05*(V+65))
    beta.h <- function(V) 1/(1+exp(-0.1*(V+35)))

    tau.n <- function(V) 1/(alpha.n(V) + beta.n(V))
    tau.m <- function(V) 1/(alpha.m(V) + beta.m(V))
    tau.h <- function(V) 1/(alpha.h(V) + beta.h(V))

    n.inf <- function(V) alpha.n(V)/(alpha.n(V) + beta.n(V))
    m.inf <- function(V) alpha.m(V)/(alpha.m(V) + beta.m(V))
    h.inf <- function(V) alpha.h(V)/(alpha.h(V) + beta.h(V))


    im <-  (g.na*(M^3)*H*(V-e.na)) + (g.k*(N^4)*(V-e.k)) + (g.l*(V-e.l))

    ie.a <- ifelse(t>=5, 0, -50 )

    dV <- (ie.a - im )/cm
    dN <- (n.inf(V) - N)/tau.n(V)
    dM <- (m.inf(V) - M)/tau.m(V)
    dH <- (h.inf(V) - H)/tau.h(V)



    return(list(c(dV, dN, dM, dH)))
```

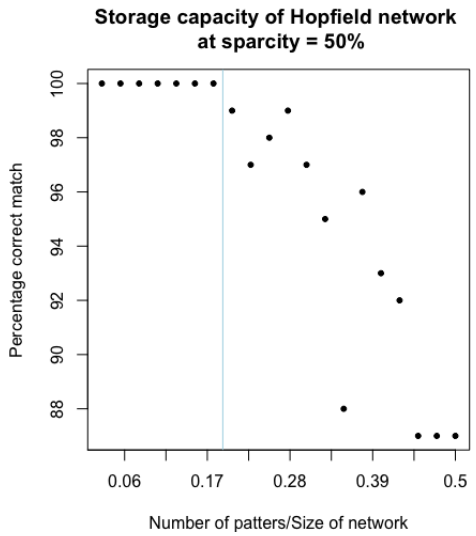

Figure 19: Network capacity with ideal sparseness of 50%

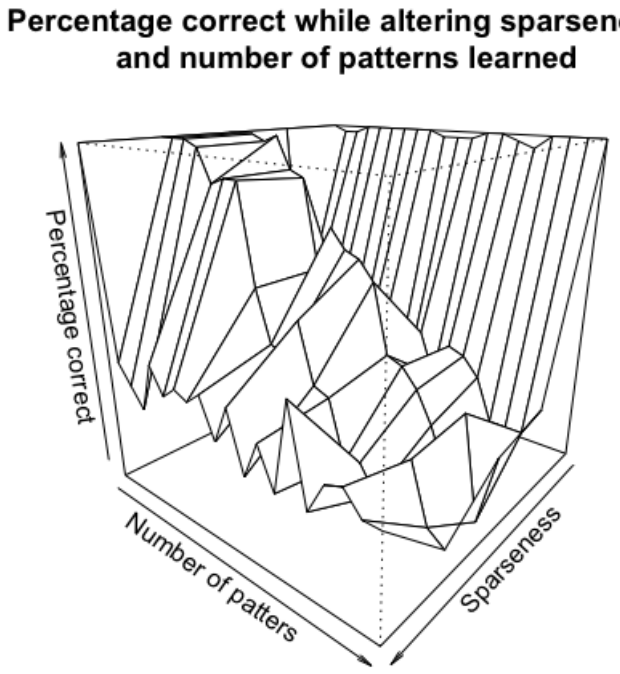

Figure 20: Testing capacity of Hopfield network with varying sparseness

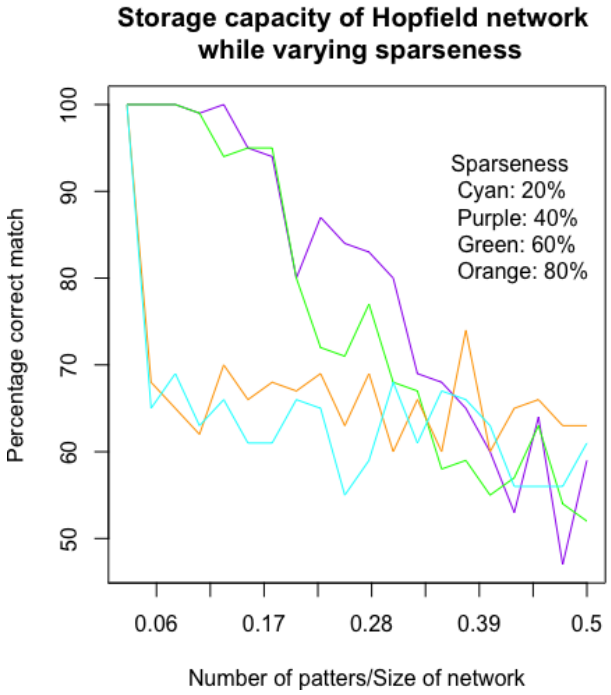

Figure 21: Testing capacity of Hopfield network with varying sparseness

```
  })
}

parameters <- c(cm =10, e.na = 50, e.k = -77, e.l = -54,
                g.na = 1200, g.k = 360, g.l =3, ie.a = 0 )
state <- c(V = -70, N=0.3177, M = 0.0529, H = 0.5961 )
times <- seq(0, 40, by = 0.1)
out <- ode(y = state, times = times, func = HH.time, parms = parameters)
plot(out, xlab = "time /ms", ylab=c("mV", "", "", ""))



find_peaks <- function (x, m = 3){
  # x: vector of sequence
  # m: number of points on either side of peak that must be smaller than peak
  # return: index of peaks in sequence
  x <- ifelse(x>-54.4, x, -80)
  shape <- diff(sign(diff(x, na.pad = FALSE)))
```

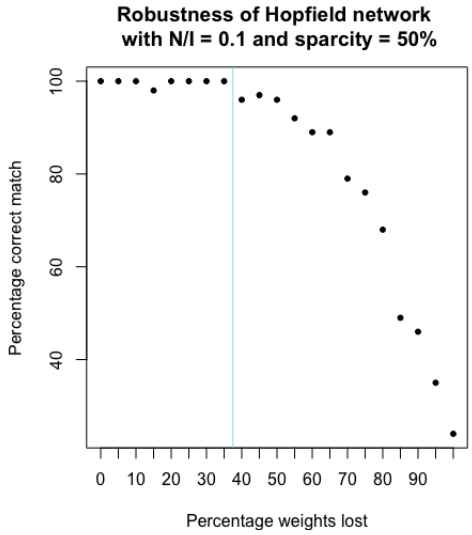

Figure 22: Caption

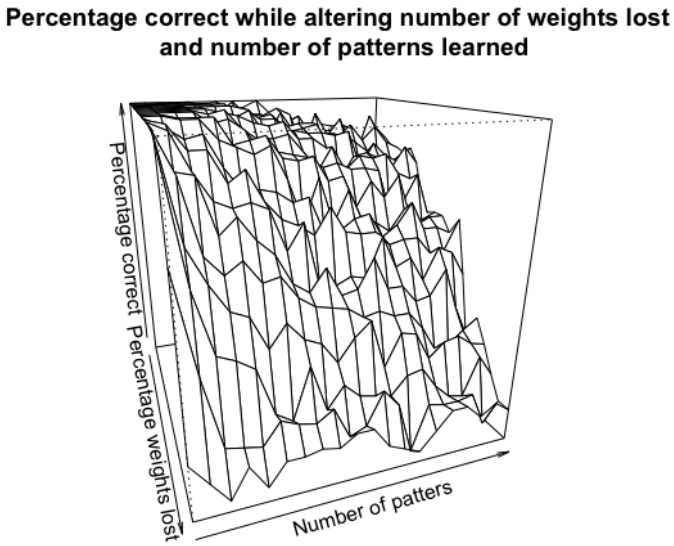

Figure 23: Testing robustness of Hopfield network with weight loss

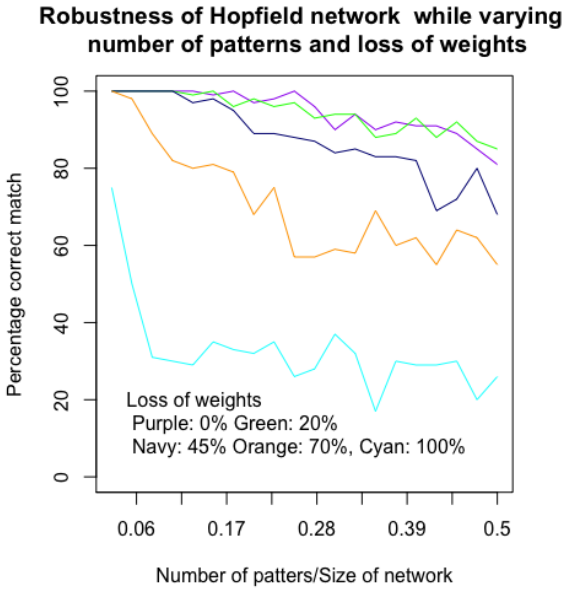

Figure 24: Testing capacity of Hopfield network with weight loss and number of patterns learned

```
  pks <- sapply(which(shape < 0), FUN = function(i){
    z <- i - m + 1
    z <- ifelse(z > 0, z, 1)
    w <- i + m + 1
    w <- ifelse(w < length(x), w, length(x))
    if(all(x[c(z : i, (i + 2) : w)] <= x[i + 1])) return(i + 1) else return(numeric(0))
  })
  pks <- unlist(pks)
  pks
}



ISI <- function(spike.train, res){
  # spike.train: vector of spike times
  # res: resolution of times (i.e. how many points per millisecond)
  # return:
  #     - inter spike intervals (ms) for each spike
  #     - average inter spike interval
```

```
  #     - average firing rate
  isi <- integer(length(peaks)-1)
  if (length(spike.train) >2){
    isi <- diff(spike.train)/res
    ave.ISI <- mean(isi)
    ave.FR <- 1000/ave.ISI #1000ms in a second
  }else {
    isi <- NA
    ave.ISI <- NA
    ave.FR <- 1
  }

  return(list(isi, ave.ISI, ave.FR ))
}




fr <- c()
for (i in seq(0, 500,10)){
HH.time <- function(t,state, parameters){
  # t: sequence from 0 to T with a certain resolution
  # state: initial values for V, M, H, and N
  # parameters: set values for differential equations i.e. cm, e.na etc.
  # return: values of V, M, H, and N over time t as matrix
  with(as.list(c(state, parameters)), {


    alpha.n <- function(V) (0.01*(V+55))/(1-exp(-0.1*(V+55)))
    beta.n <- function(V) 0.125*exp(-0.0125*(V+65))

    alpha.m <- function(V) (0.1*(V+40)) / (1-exp(-0.1*(V+40)))
    beta.m <- function(V) 4*exp(-0.0556*(V+65))

    alpha.h <- function(V) 0.07*exp(-0.05*(V+65))
    beta.h <- function(V) 1/(1+exp(-0.1*(V+35)))

    tau.n <- function(V) 1/(alpha.n(V) + beta.n(V))
    tau.m <- function(V) 1/(alpha.m(V) + beta.m(V))
    tau.h <- function(V) 1/(alpha.h(V) + beta.h(V))

    n.inf <- function(V) alpha.n(V)/(alpha.n(V) + beta.n(V))
    m.inf <- function(V) alpha.m(V)/(alpha.m(V) + beta.m(V))
    h.inf <- function(V) alpha.h(V)/(alpha.h(V) + beta.h(V))


    im <-  (g.na*(M^3)*H*(V-e.na)) + (g.k*(N^4)*(V-e.k)) + (g.l*(V-e.l))

    dV <- (ie.a - im )/cm
    dN <- (n.inf(V) - N)/tau.n(V)
    dM <- (m.inf(V) - M)/tau.m(V)
    dH <- (h.inf(V) - H)/tau.h(V)

    return(list(c(dV, dN, dM, dH)))
  })
}

parameters <- c(cm =10, e.na = 50, e.k = -77, e.l = -54,
                g.na = 1200, g.k = 360, g.l =3, ie.a = i )
state <- c(V = -65, N=0.3177, M = 0.0529, H = 0.5961 )
times <- seq(0, 1000, by = 0.1)
out <- ode(y = state, times = times, func = HH.time, parms = parameters)

spike.train <- find_peaks(out[,2])
isi.results <- ISI(spike.train, 10) # to be set as 1/('by" value of "times" variable in seq function
    )
ave.fr <- isi.results[[3]]
fr <- c(fr, ave.fr)
par(mfrow=c(1,1))
}


par(mfrow=c(1,1))
par(mar=c(4,4,4,4))
plot(fr,pch=19, xlab=expression(paste("Ie/A (nA/mm"^"2", ")", sep="")),
     ylab="average firing rate (Hz)", xaxt = "n")
```

```r
axis(side=1, at=c(0, 10, 20, 30, 40, 50), labels=c("0", "100", "200", "300", "400", "500") )
abline(v=7.5, col="lightblue", lty="longdash")
```

```r
## coupled integrate fire


integrate.fire <- function(time, h, parameters, v.one.init=-68, v.two.init=-70 ){
  with(as.list(parameters),{

  V1 <- integer(length(seq(0, time, h)))
  V2 <- integer(length(seq(0, time, h)))
  Ps1 <- integer(length(seq(0, time, h)))
  Ps2 <- integer(length(seq(0, time, h)))
  Z1 <- integer(length(seq(0, time, h)))
  Z2 <- integer(length(seq(0, time, h)))

  set.seed(4123)
  V1[1] <- v.one.init #runif(min=v.reset,max=v.th, 1)
  V2[1] <- v.two.init #runif(min=v.reset,max=v.th, 1)
  Ps1[1] <- 0.1
  Ps2[1] <- 0.1
  Z1[1] <- 1
  Z2[1] <- 1

  fr1 <- c()
  fr2 <-c()

  for (t in 1:(length(seq(0, time, h))-1)){
    dPs <- function(Ps, Z) ((exp(1)*pmax*Z - Ps) /tau.s)
    dZ <- function(Z) -Z/tau.s
    dV <- function(V, Ps) (e.l - V - rm.gs*Ps*(V - e.s) + bigrm.ie)/tau.m

    V1[t+1] <- V1[t] + h*dV(V1[t], Ps1[t])
    V2[t+1] <- V2[t] + h*dV(V2[t], Ps2[t])

    Ps1[t+1] <- Ps1[t] + h*dPs(Ps1[t], Z1[t])
    Ps2[t+1] <- Ps2[t] + h*dPs(Ps2[t], Z2[t])

    Z1[t+1] <- Z1[t] + h*dZ(Z1[t])
    Z2[t+1] <- Z2[t] + h*dZ(Z2[t])

    if (V1[t+1] > v.th){
      Z2[t+1] <- 1
      V1[t+1] <- v.reset
      fr2 <- c(fr2, t+1)

    }

    if (V2[t+1] > v.th){
      Z1[t+1] <- 1
      V2[t+1] <- v.reset
      fr1 <- c(fr1, t+1)
    }
  }
  return(list(V1, V2, Ps1, Ps2, Z1, Z2, fr1, fr2))
  })
}



parameters <- c(e.s = 0, e.l = -70,
                #e.s : excitatory or inhibitory effect on neuron of a successful input AP
                #e.l : resting membrane potential of neuron
                v.th = -54, v.reset=-80,
                #v.th : threshold voltage to initiate AP
                #v.reset : voltage reset when AP is fired
                tau.m = 20,tau.s = 10,
                #tau.m :
                #tau.s :
                rm.gs = 0.15, bigrm.ie = 18,
```

```
                #rm.gs :
                #bigrm.ie :
                pmax =0.5
                #pmax :
)


#example excitatory vs inhibitory

time <- 400
coupled.1 <- integrate.fire(time, 0.1, parameters )

coupled.1[[1]][find_peaks(coupled.1[[1]], m=3)] <- 1
coupled.1[[2]][find_peaks(coupled.1[[2]], m=3)] <- 0

#par(mfrow=c(2,1), mar=c(1,4,2,2))
par(mfrow=c(1,1), mar=c(4,4,4,2))
plot(coupled.1[[1]], xlab="Time /ms", ylab="Voltage /mV", type="l", xaxt ="n", col="blue")
axis(side=1, at=seq(0, time*10, 500), labels=seq(0, time, 50))
segments(x0= -300, x1=-50, y0=coupled.1[[1]][1], y1=coupled.1[[1]][1],col="blue")
text(x=-500, y=coupled.1[[1]][1], srt=0, adj = 0, labels = as.character(round(coupled.1[[1]][1])),
    xpd = TRUE, col="blue")
title(main="Coupled inhibitory neurons")
#plot(coupled.1[[2]], xlab="Time /ms", ylab="Voltage /mV", type="l", xaxt ="n", col="orange")
lines(coupled.1[[2]], col="orange")
segments(x0= -300, x1=-50, y0=coupled.1[[2]][1], y1=coupled.1[[2]][1],col="orange")
text(x=-500, y=coupled.1[[2]][1], srt=0, adj = 0, labels = as.character(round(coupled.1[[2]][1])),
    xpd = TRUE, col="orange")

seq(-75, -53, 2)


#investigate initial voltage

investigate.initial.voltage <- function(time, h, Es){
  vs <- seq(-75, -53, 2)
  firing.rate.V1 <- matrix(0, nrow= length(vs), ncol=length(vs) )
  firing.rate.V2 <- matrix(0, nrow=length(vs), ncol=length(vs) )
  synched <- matrix(0, nrow=length(vs), ncol=length(vs) )
  for (v1 in 1:length(vs)){
    for (v2 in 1:length(vs)){
      parameters <- c(e.s = Es, e.l = -70,
                      #e.s : excitatory or inhibitory effect on neuron of a successful input AP
                      #e.l : resting membrane potential of neuron
                      v.th = -54, v.reset=-80,
                      #v.th : threshold voltage to initiate AP
                      #v.reset : voltage reset when AP is fired
                      tau.m = 20,tau.s = 10,
                      #tau.m :
                      #tau.s :
                      rm.gs = 0.15, bigrm.ie = 18,
                      #rm.gs :
                      #bigrm.ie :
                      pmax =0.5
                      #pmax :
      )
      coupled <- integrate.fire(time, h, parameters, v.one.init=vs[v1], v.two.init = vs[v2])
      print(c(vs[v1], vs[v2]))

      V1.peaks <- find_peaks(coupled[[1]])
      V2.peaks <- find_peaks(coupled[[2]])

      V1.isi <- ISI(V1.peaks, 1/h)
      V2.isi <- ISI(V2.peaks, 1/h)

      firing.rate.V1[v1,v2] <- V1.isi[[3]]
      firing.rate.V2[v1,v2] <- V2.isi[[3]]

      if (abs(V1.peaks[length(V1.peaks)] - V2.peaks[length(V2.peaks)]) <5){
        synched[v1,v2] <- TRUE
      } else{
        synched[v1,v2] <- FALSE
      }
    }
  }
  return(list(synched, firing.rate.V1,firing.rate.V2))
```

```
}



init.volt.excit <- investigate.initial.voltage(4000, 0.1, -80)


vs <- seq(-75, -53, 2)
par(mar=c(4,4,4,2))
image(init.volt.excit[[1]], axes=FALSE, col = grey(seq(0, 1, length = 2)))
axis(1, at=seq(0,1,length.out = 12), label=as.character(vs) )
axis(2, at=seq(0,1,length.out = 12), label=as.character(vs) )
title(xlab="V1 starting voltage (mV)", ylab="V2 starting voltage (mV)",
      main="Synchrony of firing in inhibitory \n coupled neurons after 4s")




#investigate tau of synapses


investigate.tau <- function(time, h, Es){
  vs <- seq(1,60,4)
  firing.rate.V1 <- matrix(0, nrow=length(vs), ncol=length(vs))
  firing.rate.V2 <- matrix(0, nrow=length(vs), ncol=length(vs))
  synched <- matrix(0, nrow=length(vs), ncol=length(vs))

  for (i in 1:length(vs)){
    for (j in 1:length(vs)){
      parameters <- c(e.s = Es, e.l = -70,
                      #e.s : excitatory or inhibitory effect on neuron of a successful input AP
                      #e.l : resting membrane potential of neuron
                      v.th = -54, v.reset=-80,
                      #v.th : threshold voltage to initiate AP
                      #v.reset : voltage reset when AP is fired
                      tau.m = vs[i],tau.s = vs[j],
                      #tau.m :
                      #tau.s :
                      rm.gs = 0.15, bigrm.ie = 18,
                      #rm.gs :
                      #bigrm.ie :
                      pmax =0.5
                      #pmax :
      )
      coupled <- integrate.fire(time, h, parameters)

      print(c(vs[i], vs[j]))

      V1.peaks <- find_peaks(coupled[[1]])
      V2.peaks <- find_peaks(coupled[[2]])

      V1.isi <- ISI(V1.peaks, 1/h)
      V2.isi <- ISI(V2.peaks, 1/h)

      firing.rate.V1[i,j] <- V1.isi[[3]]
      firing.rate.V2[i,j] <- V2.isi[[3]]

      if (abs(V1.peaks[length(V1.peaks)] - V2.peaks[length(V2.peaks)]) <5){
        synched[i,j] <- TRUE
      } else{
        synched[i,j] <- FALSE
      }
    }
  }
  return(list(firing.rate.V1, firing.rate.V2, synched))
}


tau.investigation.inhib <- investigate.tau(4000, 0.1, -80)
tau.investigation.excit <- investigate.tau(4000, 0.1, 0)


#edit tau.investigation for choice of excit or inhib
vs <- seq(1,60,4)
par(mar=c(4,4,4,2))
image(tau.investigation[[3]], axes = FALSE, col = grey(seq(0, 1, length = 2)))
```

```r
axis(1, at=seq(0,1,length.out = 15), label=as.character(vs) )
axis(2, at=seq(0,1,length.out = 15), label=as.character(vs) )
title(xlab="Tau_m (ms)", ylab="Tau_s (ms)",
      main="Synchrony of firing in inhibitory \n coupled neurons after 4s")

persp(z=tau.investigation[[2]], theta=25, phi=5, scale=T,
      ticktype = "detailed",
      xlab="tau_m",
      ylab="tau_s",
      zlab="Firing rate",
      main="Firing rate as altering tau_m and tau_s \n with inhibitory synapses")




#investigate strength of synapses

investigate.strength <- function(time, h, Es){
  vs <- seq(0, 1.5, 0.05)
  firing.rate.V1 <- integer(length(vs))
  firing.rate.V2 <- integer(length(vs))
  synched <- integer(length(vs))
  for (s in 1:length(vs)){
    parameters <- c(e.s = Es, e.l = -70,
                    #e.s : excitatory or inhibitory effect on neuron of a successful input AP
                    #e.l : resting membrane potential of neuron
                    v.th = -54, v.reset=-80,
                    #v.th : threshold voltage to initiate AP
                    #v.reset : voltage reset when AP is fired
                    tau.m = 20,tau.s = 10,
                    #tau.m :
                    #tau.s :
                    rm.gs = vs[s], bigrm.ie = 18,
                    #rm.gs :
                    #bigrm.ie :
                    pmax =0.5
                    #pmax :
    )
    coupled <- integrate.fire(time, h, parameters, v.one.init=-65, v.two.init = -67)
    print(vs[s])

    V1.peaks <- find_peaks(coupled[[1]])
    V2.peaks <- find_peaks(coupled[[2]])

    V1.isi <- ISI(V1.peaks, 1/h)
    V2.isi <- ISI(V2.peaks, 1/h)

    firing.rate.V1[s] <- V1.isi[[3]]
    firing.rate.V2[s] <- V2.isi[[3]]

    if (abs(V1.peaks[length(V1.peaks)] - V2.peaks[length(V2.peaks)]) < 2){
      synched[s] <- TRUE
    } else{
      synched[s] <- FALSE
    }

  }
  return(list(synched, firing.rate.V1,firing.rate.V2))
}


strength.investigation.excit <- investigate.strength(4000, 0.1, 0)
strength.investigation.inhib <- investigate.strength(4000, 0.1, -80)


strengths <-  seq(0, 1.5, 0.05)
plot(strengths, strength.investigation.excit[[3]],
     xlab=expression(paste("Strength of synapse", "(",r[m]*g[s], "value)")),
     ylab="firing rate /Hz",
     main="Firing rate of V1 and V2 \n with changing synapse strength \n in excitatory system",
     pch=19, col="blue")
points(x= strengths,y=strength.investigation.excit[[2]], col="orange", pch=20)
```

```r

## Networks

Mee <- 1.25
Mie <- 1
Mii <- -1
Mei <- -1
gamma.e <- -10
gamma.i <- 10
t.e <- 10

lambda.neg <- integer(200)
lambda.pos <- integer(200)
for (t.i in 1:200){
  lambda.neg[t.i] <- 0.5*((Mee-1)/t.e + (Mii-1)/t.i) - sqrt( as.complex(( (Mee-1)/t.e - (Mii-1)/t.i)^2 + (4*Mei*Mie)/(t.e*t.i) ) )
  lambda.pos[t.i] <- 0.5*((Mee-1)/t.e + (Mii-1)/t.i) + sqrt( as.complex(( (Mee-1)/t.e - (Mii-1)/t.i)^2 + (4*Mei*Mie)/(t.e*t.i) ) )
}
lambdas <- list(lambda.neg,lambda.pos)



simulation.network.ode <- function(t,state, parameters){
  with(as.list(c(state, parameters)), {
    dVe <- (-Ve + ifelse((Mee*Ve + Mei*Vi - gamma.e)>0, max(Mee*Ve + Mei*Vi - gamma.e), 0))/tau.e
    dVi <- (-Vi + ifelse((Mii*Vi + Mie*Ve - gamma.i)>0, max(Mii*Vi + Mie*Ve - gamma.i), 0))/tau.i
    return(list(c(dVe, dVi)))
  })
}

parameters <- c(Mee = 1.25, Mei = -1, gamma.e = -10, tau.e = 10,
                Mii = -1, Mie = 1, gamma.i = 10, tau.i = 85)
state <- c(Ve = 35, Vi = 15)
times <- seq(0, 1000, by = 0.1)
out <- ode(y = state, times = times, func = simulation.network.ode, parms = parameters)

par(mar=c(4,4,4,4), mfrow=c(1,1))
plot(out[,c(1,2)], xlab="Time /ms", xaxt='n' ,ylab="Firing rate /Hz",
     main=expression(paste("Stabilising neuron firing rates with ",
                           tau[I], " = 55 ms")),
     type="l", ylim=c(15,80), col="blue" )
axis(1, at=seq(0,1000,100), labels=seq(0,1000,100))
lines(out[,1],out[,3], col="orange")

par(mar=c(4,4,4,4), mfrow=c(1,1))
plot(out[,c(1,2)], xlab="Time /ms", xaxt='n' ,ylab="Firing rate /Hz",
     main=expression(paste("Oscillating neuron firing rates with ",
                           tau[I], " = 85 ms")),
     type="l", ylim=c(0,100), col="blue" )
axis(1, at=seq(0,1000,100), labels=seq(0,1000,100))
lines(out[,1],out[,3], col="orange")



simulation.network.phaseR <- function(t,y, params){
  Ve <- y[1]
  Vi <- y[2]
  Mee = params[1]
  Mei = params[2]
  gamma.e = params[3]
  tau.e = params[4]
  Mii = params[5]
  Mie = params[6]
  gamma.i = params[7]
  tau.i = params[8]
  dVe <- (-Ve + ifelse((Mee*Ve + Mei*Vi - gamma.e)>0, max(Mee*Ve + Mei*Vi - gamma.e), 0))/tau.e
  dVi <- (-Vi + ifelse((Mii*Vi + Mie*Ve - gamma.i)>0, max(Mii*Vi + Mie*Ve - gamma.i), 0))/tau.i
  dy <- numeric(2)
  dy[1] <- dVe
  dy[2] <-dVi
  list(dy)
}

params <- c(1.25,-1,-10, 10, -1, 1, 10, 79)
```

```
532
533 par(pty="m")
534 sim.network.flowField <- flowField(simulation.network.phaseR,
535                                    xlim = c(0,100),
536                                    ylim = c(0, 50),
537                                    parameters = params,
538                                    arrow.type = "proportional",
539                                    add = FALSE,
540                                    xlab=expression(paste(v[E], " /Hz")),
541                                    ylab=expression(paste(v[I], " /Hz")),
542                                    main=expression(paste("Phase plane analysis of stable system with "
    , tau[i], "=79ms")))
543
544 nullclines(simulation.network.phaseR,
545            xlim = c(0,100),
546            ylim = c(0, 100),
547            parameters = params,
548            add=TRUE,
549            legend=c(expression(paste(v[E])), expression(paste(v[I])) ) )
550
551
552
553 trajectory(simulation.network.phaseR, y0 = c(59.75,24.75),
554            tlim=c(0,10000),  tstep = 0.1,
555            parameters = params)
556
557
558
559 kill<- 300
560
561
562 # Hopfield
563
564 Hopfield <- function(patterns, new.pattern, kill=0){
565   weights <- t(patterns)%*%patterns
566   diag(weights) <- NA
567   k <- 0
568   if (kill > 0){
569     weights[sample(dim(weights)[1], kill), sample(dim(weights)[2], kill)] <- NA
570     weights <- weights + t(weights)
571   }
572   repeat {
573     i <- sample(dim(patterns)[2], 1, replace=T)
574     prev.nodes <- new.pattern
575     new.pattern[i] <- ifelse( sum(weights[,i]*new.pattern, na.rm = T) >= 0, 1, -1)
576     new.nodes <- new.pattern
577
578     if (all.equal(new.nodes, prev.nodes) == T){
579       k <- k+1
580     }
581     if (k >= 2){
582       #print("2 iterations with no changes")
583       break
584     }
585
586   #m <- matrix(new.pattern, sqrt(length(new.pattern)), sqrt(length(new.pattern)))
587   #par(mar=c(0, 0, 0, 0))
588   #image(m, axes = FALSE, col = grey(seq(0, 1, length = 2)))
589   }
590   return(list(new.pattern, weights))
591 }
592
593
594 mutate.generate <- function(pattern, mistakes = 3, N){
595   patterns <- matrix(pattern, nrow=N, ncol=length(pattern))
596   for (n in 1:N){
597     for (i in sample(1:length(pattern), mistakes)){
598       patterns[n, i] <- pattern[i]*-1
599     }
600   }
601   return(patterns)
602 }
603
604
605
606 # par(mar=c(0, 0, 0, 0), pty="s")
```

```
# m <- matrix(patterns[1,], sqrt(length(patterns[1,])), sqrt(length(patterns[1,])))
# image(m, axes = FALSE, col = grey(seq(0, 1, length = 2)))
#
# new.pattern <- mutate.generate(patterns[2,], mistakes = I^2/50, N=1)
# new.pattern.image <- matrix(new.pattern, sqrt(length(new.pattern)), sqrt(length(new.pattern)))
# image(new.pattern.image, axes = FALSE, col = grey(seq(0, 1, length = 2)))

check.hopf <- function(patterns, hopf.pattern, strict = FALSE){
  differences <- integer(dim(patterns)[1])
  overlap <- integer(dim(patterns)[1])
  if (strict == TRUE ){
    max.diff <- 0
  } else {
    x <- as.integer(sum(patterns[1,]==1)/(sqrt(dim(patterns)[2])*4))
    max.diff <- ifelse(x==0, 1, x)
  }
  for (pattern in 1:(dim(patterns)[1])){
    differences[pattern] <- sum(patterns[pattern,] != hopf.pattern)
    overlap[pattern] <- as.numeric( ( patterns[pattern,]%*%as.vector(hopf.pattern) ) /dim(patterns)
    [2])
  }
  norm.diff <- differences/(dim(patterns)[2])
  best.match <- which.min(differences)
  learned <- ifelse(min(differences) <= max.diff, T, F)

  return(list(max(overlap),learned, best.match ))
}




I <- 20
num.patterns <- 10
mistakes <- I^2/2
new.mistakes <- as.integer(mistakes/30)

patterns <- mutate.generate(rep(-1, I^2), mistakes= mistakes , N= num.patterns )

new <- sample(1:num.patterns, 1)
new.pattern <- mutate.generate(patterns[new,], mistakes = ifelse(new.mistakes==0, 1, new.mistakes) ,
     N=1)
#new.pattern <- patterns[new, ]
image(matrix(patterns[new,], I, I), axes = FALSE, col = grey(seq(0, 1, length = 2)))
image(matrix(new.pattern, I, I), axes = FALSE, col = grey(seq(0, 1, length = 2)))


hopf.temp <- Hopfield(patterns, new.pattern)
hopf.pattern <- hopf.temp[[1]]
image(matrix(hopf.pattern, I, I), axes = FALSE, col = grey(seq(0, 1, length = 2)))

test.check <- check.hopf(patterns, hopf.pattern, strict = F)

test.check

length(patterns[new,])



storage.capacity <- function(I, K, S, single.pattern.set=F, kill){
  # I: sqrt of number of nodes in network
  # S: sparseness of patterns
  # K: number of repetitions
  # p: sequence of number of patters
  if (single.pattern.set==T){
    p <- c(I^2*0.1)
  } else{
    p <- seq(1, I^2/2, by=I/2)
  }
  learned <- matrix(NA, nrow=length(p), ncol=K)
  overlap <- matrix(NA, nrow=length(p), ncol=K)
  match <- matrix(NA, nrow=length(p), ncol=K)
  for (num.patterns in 1:length(p) ){
    patterns <- mutate.generate(rep(-1, I^2), mistakes=S, N=p[num.patterns] )
    new <- sample(1:p[num.patterns], 1)
    new.mistakes <- as.integer(S/ (3*I))

```

```
    #new.pattern <- mutate.generate(patterns[new,] ,
    #                                mistakes = ifelse(new.mistakes==0, 1, new.mistakes),
    #                                N=1)
    new.pattern <- patterns[new, ]
    for (k in 1:K){
      #print(c( k, p[num.patterns], S))
      hopf.temp <- Hopfield(patterns, new.pattern, kill)
      hopf.pattern <- as.vector(hopf.temp[[1]])
      weights <- hopf.temp[[2]]

      l.temp <- check.hopf(patterns, hopf.pattern, strict=T )
      overlap[num.patterns, k] <- l.temp[[1]]
      learned[num.patterns, k] <- l.temp[[2]]

      if ( isTRUE(all.equal(l.temp[[3]], new)) == T) {
        match[num.patterns, k] <- T
      } else{
        match[num.patterns,k ] <- F
      }
    }
  }
  return(list(overlap, learned, match, weights))
}

test.storage.cap <- storage.capacity(20, 100, 200)


par(pty="s")
plot(apply(test.storage.cap[[2]], 1, sum), xaxt="n",
     xlab="Number of patters/Size of network",
     ylab="Percentage correct match",
     main="Storage capacity of Hopfield network \n at sparcity = 50%", pch=20)
axis(1, at=seq(0,20,length.out=10), label=round(
  seq(1, 20^2/2, length.out=10)/400, 2) )
abline(v=7.5, col="lightblue")

storage.capacity.sparseness <- function(I){
  s <- seq(1, I^2, by=4*I)
  p <- seq(1, I^2/2, by=I/2)
  overlap <- matrix(NA, nrow=length(s), ncol= length(p) )
  learned <- matrix(NA, nrow=length(s), ncol=length(p) )
  match <- matrix(NA, nrow=length(s), ncol= length(p) )
  for (ones in 1:length(s) ){
    storage.res <- storage.capacity(I, K=100, S=s[ones])
    overlap[ones,] <- apply(storage.res[[1]], 1, sum)
    learned[ones, ] <- apply(storage.res[[2]], 1, sum)
    match[ones,] <- apply(storage.res[[3]], 1, sum)

  }
  return(list(overlap, learned, match ))
}

sparce.pattern.capacity <- storage.capacity.sparseness(20)


sparce.pattern.capacity[[2]]
persp(z=sparce.pattern.capacity[[2]], theta=130, phi=20, scale=T,
      ticktype = "simple",
      xlab="Sparseness",
      ylab="Number of patters",
      zlab="Percentage correct",
      main="Percentage correct while altering sparseness \n and number of patterns learned")

plot(sparce.pattern.capacity[[2]][3,], type='l',
     xaxt="n", col="purple",
     xlab="Number of patters/Size of network",
     ylab="Percentage correct match",
     main="Storage capacity of Hopfield network \n while varying sparseness", pch=20)
axis(1, at=seq(0,20,length.out=10), label=round(
  seq(1, 20^2/2, length.out=10)/400, 2) )
lines(sparce.pattern.capacity[[2]][4,], col="green")
lines(sparce.pattern.capacity[[2]][5,], col="orange")
lines(sparce.pattern.capacity[[2]][2,], col="cyan")

legend(x=13, y=99, legend="Sparseness \n Cyan: 20% \n Purple: 40% \n Green: 60% \n Orange: 80%", bty
     = "n")
```

```r


robustness <- function(I){
  r <- seq(0, I^2, I)
  p <- seq(1, I^2/2, by=I/2)
  overlap <- matrix(NA, length(r), length(p))
  learned <- matrix(NA, length(r), length(p))
  match <- matrix(NA, length(r), length(p))
  #weights <- list()

  for (k in 1:length(r) ){
    print(r[k])
    storage.res <- storage.capacity(I, K=100, S=I^2/2, single.pattern.set = F, kill=r[k])
    overlap[k,] <- apply(storage.res[[1]], 1, sum)
    learned[k,] <- apply(storage.res[[2]], 1, sum)
    match[k,] <- apply(storage.res[[3]], 1, sum)
    #weights[[k]] <- storage.res[[4]]
    }
  return(list(overlap,learned, match,weights ))
}

storage.res <- storage.capacity(I, K=100, S=I^2/2, single.pattern.set = T, kill=390)
apply(storage.res[[2]], 1, sum)

test.single.robust <- robustness(20)

test.single.robust[[2]]

par(pty="s")
plot(test.single.robust[[2]], xaxt="n",
     xlab="Percentage weights lost",
     ylab="Percentage correct match",
     main="Robustness of Hopfield network \n with N/I = 0.1 and sparcity = 50%", pch=20)
axis(1, at=1:21, label=seq(0, I^2, I)/400*100 )
abline(v=8.5, col="lightblue")


persp(z=test.single.robust[[2]], theta=70, phi=20, scale=T,
      ticktype = "simple",
      xlab="Percentage weights lost",
      ylab="Number of patters",
      zlab="Percentage correct",
      main="Percentage correct while altering number of weights lost \n and number of patterns
    learned")

plot(test.single.robust[[2]][1,], type='l', ylim=c(0,100),
     xaxt="n", col="purple",
     xlab="Number of patters/Size of network",
     ylab="Percentage correct match",
     main="Robustness of Hopfield network  while varying \n number of patterns and loss of weights",
     pch=20)
axis(1, at=seq(0,20,length.out=10), label=round(
  seq(1, 20^2/2, length.out=10)/400, 2) )
lines(test.single.robust[[2]][5,], col="green")
lines(test.single.robust[[2]][10,], col="navy")
lines(test.single.robust[[2]][15,], col="orange")
lines(test.single.robust[[2]][20,], col="cyan")

legend(x=0, y=30, legend="Loss of weights \n Purple: 0% Green: 20% \n Navy: 45% Orange: 70%, Cyan:
    100%", bty = "n")

test.single.robust[[2]][15,]




### new method

improved.Hopfield <- function(patterns, new.pattern, L){
  weights <- t(patterns)%*%patterns
  diag(weights) <- 0

  t <- patterns
  t[t==-1] <- 0
  for (l in 1:L){
```

```
    diag(weights) <- 0
    activations <- patterns %*% weights
    outputs <- sigmoid(activations)
    errors <- t - outputs
    gradients <- t(patterns) %*% errors
    gradients <- gradients + t(gradients)

    weights <- weights + 0.01*(gradients - 0.1*weights)
  }
}









######################


# nullclines <- function(){
#   Ve.xnull <- integer(101)
#   Vi.xnull <- integer(101)
#   Ve.ynull <- integer(101)
#   Vi.ynull <- integer(101)
#   Mii <- -1
#   Mee <- 1.25
#   Mei <- -1
#   Mie <- 1
#   gamma.i <- 10
#   gamma.e <- -10
#
#   #x/Ve nullcline
#   for (e in 0:100){
#     Vi.xnull[e+1] <- e*((1-Mee)/Mei) + gamma.e/Mei
#   }
#   for (i in 0:100){
#     Ve.xnull[i+1] <- i*(Mei/(1-Mee)) - gamma.e/(1-Mee)
#   }
#
#   #ynullcline
#   for (e in 0:100){
#     Vi.ynull[e+1] <- (e*Mie)/(1-Mii) - gamma.i/(1-Mii)
#   }
#   for (i in 0:100){
#     Ve.ynull[i+1] <- i*((1-Mii)/Mie) + gamma.i/Mie
#   }
#
#   return(list(Vi.xnull,Ve.xnull, Vi.ynull, Ve.ynull))
# }
#
# p <- nullclines()
#
# plot(x = 0:100, y = p[[1]], type="l", col='red', xlim=c(0,100), ylim=c(0, 100))
# lines(x=p[[2]], y=0:100, col='red')
# lines(x=0:100, y=p[[3]], col='blue')
# lines(x=p[[4]], y=0:100, col='blue')
```