Online Resource 6 - Summary of the code used to simulate quantitative vasopressin secretion.

The code has to be launched with the Scilab software (version 4.1.2).

```
// ****************************************
// AVP level simulation based on magnocellular
// neurons simulation
// Louis Nadeau - Louis.Nadeau.3@ulaval.ca
// ****************************************
```

//Command to simulate AVP vs Osmo relationship

```
//Osmo=[-20,-15,-10,-5,0,5,10,15,20,30]; for i=1:length(Osmo);
AVPQ_temp=AVP_Simulation(200000,4,3,51,Osmo(i),4,10);AVP(i)=mean(AVPQ_temp($-$/4:$));end;
```

//Function that simulate the AVP secretion caused by a sequence of spikes. The sequence should be a set of time where a spike occurs and the time scale should be in millisecond.

```
function AVPQ=AVP_Simulation(Time,multiplicator,Initial_AVP,ECF_Volume,Osmo, Number_Of_Cores,Seed)
lines(0);//Added so that its not necessary to press y or n after each page of output from scilab
```

//Prepare the parameters for the different population,
//The order and number of each parameter are important, there should be an equal number of parameters for each type of firing behavior and the order : phasic,irregular and continuous must be respected so that the population ratio modulate the correct AVP secretion

```
Current=[-1,-1.2,-1.4,-1,-1.2,-1.4,-1,-1.2,-1.4]; //Injected current
Ratio_D=[0.75,0.75,0.75,4,4,4,0.25,0.25,0.25];      //Ratio of D increase relative to the default D values for each population
Ratio_PSP_Freq=[1,1,1,1,1,1,1,1,1];                 //Ratio of EPSP frequency increase relative to the default frequency for each population
Ratio_PSP_Amp=[1,1,1,1,1,1,1,1,1];                  //Ratio of EPSP amplitude increase relative to the default amplitude for each population
Random_Seed=[Seed,Seed,Seed,Seed,Seed,Seed,Seed,Seed,Seed];      //Seed for the random generator during the neuron simulation of each population
```

Population=9000;

```
//The fraction of the population value are from brown1998
//Ratio1=0.263*Population; //Phasic
//Ratio2=0.132*Population; //Irregular
//Ratio3=0.342*Population; //Continuous
//Ratio4=0.263*Population; //Silent - This fraction should be redistributed to the other three types and should arise naturally from inactive neurons of the other types in control
Ratio1=(0.263+0.263/3)*Population //Phasic
Ratio2=(0.132+0.263/3)*Population; //Irregular
Ratio3=(0.342+0.263/3)*Population; //Continuous

Path_To_Temp_Dir="/home/cybernot/tmp/";
Path_To_Scilab="scilab";
//First part of all the bash commands
```

```
Query_command_core="mysql -ucybernot -pmycybernot -N --batch -r -e ""use MagnoSecTest; select Record is
not null from Spike where ";
Sim_command_core=Path_To_Scilab+" -nw -f
/home/cybernot/Documents/Doctorat/Scilab/Neurohypophysis/MagnoNeuron_auto.sce -args ";
Extract_command_core="mysql -ucybernot -pmycybernot -N --batch -r -e ""use MagnoSecTest; select Record
into dumpfile ";
Insert_command_core= "mysql -ucybernot -pmycybernot -N --batch -r -e ""use MagnoSecTest; insert Spike
(Osmo,Time,Current,Ratio_D,Ratio_PSP_Freq,Ratio_PSP_Amp,Seed,Record) values (";
Erase_command_core="mysql -ucybernot -pmycybernot -N --batch -r -e ""use MagnoSecTest; delete from
Spike where ";

//Query the database for the four possible population types
disp("Querying the database");
for i=1:length(Current);
 Mysql_where(i)="Osmo="+string(Osmo)+" and Time>="+string(Time)+" and Current="+string(Current(i))+"
and Ratio_D="+string(Ratio_D(i))+" and Ratio_PSP_Freq="+string(Ratio_PSP_Freq(i))+" and
Ratio_PSP_Amp="+string(Ratio_PSP_Amp(i))+" and Seed="+string(Random_Seed(i))+""""";
 Temporary_Answer=evstr(unix_g(Query_command_core+Mysql_where(i)));
 if ~isempty(Temporary_Answer) then Answer(i)=Temporary_Answer; else Answer(i)=0; end;
end;

if sum(Answer) < length(Current) then //Case where at least one simulation is needed;
 disp("At least one simulation is needed");
 disp("Preparing to launch parralel simulations");
 //Determination of the simulation to run. At least one simulation will be started but no more than four (the
number of cores on this computer)
 for i=1:length(Current);
  if ~(Answer(i)==1) then //Simulate the missing spike train
   Sim_command(i)=Sim_command_core+string(Osmo)+" "+string(Time)+" "+string(Current(i))+"
"+string(Ratio_D(i))+" "+string(Ratio_PSP_Freq(i))+" "+string(Ratio_PSP_Amp(i))+"
"+string(Random_Seed(i))+" "+Path_To_Temp_Dir+"MagnoSec"+string(i)+".txt & ";
  else //Extract the spike train that are already in the database
   Sim_command(i)="";
   unix("rm "+Path_To_Temp_Dir+"MagnoSec"+string(i)+".txt");//Have to be removed manually since INTO
DUMPFILE does not overwrite for security reasons, make sure to have the permission
   unix(Extract_command_core+" '"+Path_To_Temp_Dir+"MagnoSec"+string(i)+".txt'' from Spike where
"+Mysql_where(i));
   //When using a multi-core cpu, in order to maximise waiting time a random simulation should be launch
here to fill the database, this is not implemented yet
  end;
 end;
 disp("Launching parralel simulations");

 Command="";
 Command_Length=0;
 for i=1:length(Current)
   if ~(Sim_command(i)=="") then Command_Length=Command_Length+1;
Command=Command+Sim_command(i); end;
   if ((Command_Length==Number_Of_Cores) | (i==length(Current))) then unix(Command+" wait");
Command=""; Command_Length=0; end;
  end;

 disp("Loading simulations results in the database");
 for i=1:length(Current)
  if ~(Answer(i)==1) then
```

```
    //Erase shorter simulation if there is one
    Erase_Shorter="Osmo="+string(Osmo)+" and Time<"+string(Time)+" and Current="+string(Current(i))+"
and Ratio_D="+string(Ratio_D(i))+" and Ratio_PSP_Freq="+string(Ratio_PSP_Freq(i))+" and
Ratio_PSP_Amp="+string(Ratio_PSP_Amp(i))+" and Seed="+string(Random_Seed(i))+"""";
    unix(Erase_command_core+Erase_Shorter);
    //Insert the new result

unix(Insert_command_core+string(Osmo)+","+string(Time)+","+string(Current(i))+","+string(Ratio_D(i))+","
+string(Ratio_PSP_Freq(i))+","+string(Ratio_PSP_Amp(i))+","+string(Random_Seed(i))+",LOAD_FILE(''"+Path
_To_Temp_Dir+"MagnoSec"+string(i)+".txt''))""");
   end;
 end;

else //Case where there is no simulation needed
 disp("Extracting Spike train from MySQL");
 for i=1:length(Current)
  unix("rm "+Path_To_Temp_Dir+"MagnoSec"+string(i)+".txt");//Have to be removed manually since INTO
DUMPFILE does not overwrite for security reasons, make sure to have the permission
  unix(Extract_command_core+" ''"+Path_To_Temp_Dir+"MagnoSec"+string(i)+".txt'' from Spike where
"+Mysql_where(i));
 end;
end;

disp("Reading Simulation Results");
Unsorted_Release=[ [],[]];
Number_of_Spike=0;
Phasic_AVP=0;
Irreg_AVP=0;
Continuous_AVP=0;
Phasic_Hz=0;
Irreg_Hz=0;
Continuous_Hz=0;
for i=1:length(Current)
 load(Path_To_Temp_Dir+"MagnoSec"+string(i)+".txt",'SpikeRecord');
 Spike_temp=SpikeRecord;
 //When spike vector are extracted from the database, sometimes, their length is longer than the simulation
time and they have to be shortened
 if Spike_temp($) > Time then Spike_temp=Shorten(Spike_temp,Time);end;
 if ~isempty(Spike_temp) then Show_Spike_Plot(Time,Spike_temp,"SpikeTrain "+string(i),i);else disp("Empty
plot for "+string(i)); end;
 //Repeat the simulated spike trains, the spike trains length is increase by a factor of 2^multiplicator
 for j=1:multiplicator
  if ~isempty(Spike_temp) then Spike_temp=cat(1,Spike_temp,Spike_temp+2^(j-1)*Time);end;
 end;
 //Remove spike randomly depending on the ISI, representing the axonal conduction failure, it is important to
do it between the doubling and the release computation
 Spike_temp=Conduct_Failure(Spike_temp);
 disp("Starting the AVP release computation "+string(i));
 Release_temp=Compute_Release(Spike_temp);
 select int((i-1)/3)
 case 0
  Release_temp=Ratio1*Release_temp/3;
  Number_of_Spike=Number_of_Spike+length(Spike_temp)*Ratio1/3;
  Phasic_AVP=Phasic_AVP+sum(Release_temp);
  Phasic_Hz=Phasic_Hz+length(Spike_temp)*Ratio1/3;
```

```
  case 1
   Release_temp=Ratio2*Release_temp/3;
   Number_of_Spike=Number_of_Spike+length(Spike_temp)*Ratio2/3;
   Irreg_AVP=Irreg_AVP+sum(Release_temp);
   Irreg_Hz=Irreg_Hz+length(Spike_temp)*Ratio2/3;
  case 2
   Release_temp=Ratio3*Release_temp/3;
   Number_of_Spike=Number_of_Spike+length(Spike_temp)*Ratio3/3;
   Continuous_AVP=Continuous_AVP+sum(Release_temp);
   Continuous_Hz=Continuous_Hz+length(Spike_temp)*Ratio3/3;
  else
   disp("Never reached");
   break;
  end;
  Unsorted_Release=cat(1,Unsorted_Release,[Spike_temp,Release_temp]);
end;
disp("mofr: "+string(Number_of_Spike/(9000*(Time*2^(multiplicator-1)/1000))));
Total_Release=Phasic_AVP+Irreg_AVP+Continuous_AVP;
disp('Total AVP Release: '+string(Total_Release));

disp('Phasic: '+string(Phasic_AVP/Total_Release)+' '+string((Phasic_Hz/(9000*(Time*2^(multiplicator-
1)/1000)))));
disp('Irreg: '+string(Irreg_AVP/Total_Release)+' '+string((Irreg_Hz/(9000*(Time*2^(multiplicator-
1)/1000)))));
disp('Continous: '+string(Continuous_AVP/Total_Release)+'
'+string((Continuous_Hz/(9000*(Time*2^(multiplicator-1)/1000)))));

Sorted_Release=Unsorted_Release;
disp("Sorting the spikes");
[Matrix,Permutation]=sort(Unsorted_Release,"r"); //Only useful to obtain the permutation matrix (decreasing
order)
disp("Creating the secretion vector");
for i=1:size(Permutation,1) Sorted_Release($-i+1,:)=Unsorted_Release(Permutation(i),:); end;//Recreate the
Release vector, in increasing order, with the correct row elements

disp("Generating the AVP concentration dynamics");
//Compute the plasma AVP dynamics using the release vectors and the quantity of neurons in each population
AVPQ=Compute_AVPQ(Sorted_Release,1.5,Initial_AVP*ECF_Volume)/ECF_Volume;
time_index=Sorted_Release(:,1);
if ~(isempty(time_index) & isempty(AVPQ)) then xset('window',13);clf();plot(time_index,AVPQ); end;
endfunction;
```

//Compute the average frequency of firing during each time bin function

```
[Freq,BinarySpike,ISIH,ISH_bin]=Compute_ISI_Freq(Reduced_t,Spike,Number_Of_Bins_Freq,Number_Of_Bins_I
SH)
SpikePerBin=zeros(Number_Of_Bins_Freq,1); //Number of spike in each bin
FreqPerBin=zeros(Number_Of_Bins_Freq,1); //Average freqency of each bin
BinarySpike(1:length(Reduced_t))=%nan; //Used to put a small triangle (at +60mv) on the voltage trace for
each detected spike
ISH_bin=zeros(Number_Of_Bins_ISH,1); //Used to store the upper limit of the interspike interval allowed for
each ISI bin
ISIH=zeros(Number_Of_Bins_ISH,1); //Used to store the number of ISI in each ISH_bin
```

```
for i=1:Number_Of_Bins_ISH ISH_bin(i)=i*500.0/Number_Of_Bins_ISH;end; //Sets the size limit of each bin
using a maximal ISI size of 500 ms

if(length(Spike>1)) then
 ISI=(Spike(2:$)-Spike(1:$-1))'; //ISI Vector
 //First Spike is not included in the loop because we need two spikes to compute the ISI

SpikePerBin(1+int(Spike(1)/Time*Number_Of_Bins_Freq))=SpikePerBin(1+int(Spike(1)/Time*Number_Of_Bi
ns_Freq))+1;
 BinarySpike(1+int(Spike(1)/Time*length(Reduced_t)))=60;
 for i=2:length(Spike);

SpikePerBin(1+int(Spike(i)/Time*Number_Of_Bins_Freq))=SpikePerBin(1+int(Spike(i)/Time*Number_Of_Bin
s_Freq))+1; //Increases the spike count in the correct bin

FreqPerBin(1+int(Spike(i)/Time*Number_Of_Bins_Freq))=FreqPerBin(1+int(Spike(i)/Time*Number_Of_Bins
_Freq))+1000/ISI(i-1); //Increase the sum of the intantaneous frequency (in hz) in the correct bin
            BinarySpike(1+int(Spike(i)/Time*length(Reduced_t)))=60; //Sets a +60mv flag at the spike position
            if ISI(i-1)<500 then ISIH(int(ISI(i-1)/500*Number_Of_Bins_ISH)+1)=ISIH(int(ISI(i-
1)/500*Number_Of_Bins_ISH)+1)+1; end;
 end;
end;
if ~(sum(ISIH)==0) then ISIH=ISIH/sum(ISIH); end; //Average each bin by the total number of ISI
```

//Calculates the average frequency per bin by dividing the sum by the number of spike-1 in each bin

```
Freq=zeros(Number_Of_Bins_Freq,1);
for i=1:1:Number_Of_Bins_Freq
          if(SpikePerBin(i) < 2) then
            Freq(i)=0;
          else
            Freq(i)=FreqPerBin(i)/(SpikePerBin(i)-1);
          end;
end;
endfunction
```

//Shows the voltage trace, the intantaneous frequency and the interspike interval histogram

```
function Show_Spike_Plot(Time,SpikeRecord,Title,WinNum)
x=10000.0; //Resolution of the time axis
Reduced_t=(Time/x:Time/x:Time); //Defines the reduced time vector
Length_Of_Bin_Freq=1000; //Size (in ms) of the average frequency calculation
if Length_Of_Bin_Freq>Reduced_t($)/2 then Length_Of_Bin_Freq=Reduced_t($)/2; end;
if modulo(Time,Length_Of_Bin_Freq)==0 then
   Number_Of_Bins_Freq=int(Time/Length_Of_Bin_Freq);
else
   Number_Of_Bins_Freq=int(Time/Length_Of_Bin_Freq)+1;
end;
Number_Of_Bins_ISH=50; //Number of bin for the interspike histogram
[Freq,BinarySpike,ISH,ISH_bin]=Compute_ISI_Freq(Reduced_t,SpikeRecord,Number_Of_Bins_Freq,Number_Of_
Bins_ISH);

Title="Position of spike: "+Title+" neurons";
```

```
xset('window',WinNum);clf();
xtitle(Title,"time (ms)","Spike");
subplot(3,1,1)
plot2d(Reduced_t,BinarySpike,style=-4,rect=[0,-60,Reduced_t($),60]);xgrid()

subplot(3,1,2)
xtitle("mofr: "+string(1000*length(SpikeRecord)/Time)+" V: "+string(variance(Freq))+" V/mofr:
"+string(variance(Freq)*Time/(1000*length(SpikeRecord))),"time (ms)","Hz");
plot2d([0,Time/Number_Of_Bins_Freq/2:Time/Number_Of_Bins_Freq:Time],[0;Freq],rect=[0,0,Reduced_t($),
max(Freq)]);xgrid()

subplot(3,1,3)
xtitle("Interspike Interval Histogram","ms","relative population");
plot2d(ISH_bin,ISH,rect=[0,0,max(ISH_bin),max(ISH)]);xgrid()
endfunction

function modulation=Time_Fatigue(Time_In_Burst)
modulation=1;
endfunction

function modulation=Freq_Fatigue(Delta)
modulation=1;
endfunction
```

//Compute the freq. facilitation based on fig2 of Bicknell1988, where 3 Hz was set to modulation = 1 and the function was estimated by two slopes between 0 and 13 and between 13 and 52

```
function modulation=Freq_Facilitation(Delta)
modulation=1;
Instant_Freq=1000/Delta; //Compute the instantaneous frequency, the 1000 converts the Current and Last
unit (ms) in hertz

//The following computation is a piecewise linear interpolation of the modulation
if Instant_Freq < 6.5 then modulation=Instant_Freq*0.339;
else if Instant_Freq < 13 then modulation=2.2+(Instant_Freq-6.5)*0.6;
    else if Instant_Freq < 26 then modulation=6.1+(Instant_Freq-13)*-0.0231;
            else modulation=5.8+(Instant_Freq-26)*-0.0538; //This is an interpolation until Instant_Freq
reaches 52 and for larger value this is an extrapolation
            end;
    end;
  end;
endfunction
```

//Compute the conductance failure effect based on figure 2 of Hobbach 1988?

```
//function Short_Spike=Conduct_Failure(Spike)
//cursor=1; //current length of Short_Spike
//Short_Spike(cursor)=Spike(1);

//for i=2:length(Spike);
//  Delta=Spike(i)-Short_Spike(cursor); //Delta is computed based on the last spike received at the terminal
(Short_Spike) and not on the last spike that was transmited by the soma (Spike(i-1))
//  Instant_Freq=1000/Delta;
//  if rand()<(1.053-((1)./(1+exp((Instant_Freq-22.5)/-2.5)))) then
//  Short_Spike(cursor+1)=Spike(i)
```

```
//  cursor=cursor+1;
//  end;
//end;
//endfunction
```

//Compute the conductance failure effect based on figure 2 of Bielefeldt1993

```
function Short_Spike=Conduct_Failure(Spike)
cursor=1; //current length of Short_Spike
Short_Spike(cursor)=Spike(1);
for i=2:length(Spike);
  Delta=Spike(i)-Short_Spike(cursor); //Delta is computed based on the last spike received at the terminal
(Short_Spike) and not on the last spike that was transmited by the soma (Spike(i-1))
  Instant_Freq=1000/Delta;
  if rand()>((3.2*Instant_Freq-32)/100) then
  Short_Spike(cursor+1)=Spike(i)
  cursor=cursor+1;
  end;
end;
endfunction

function AVP_Release=Compute_Release(Spike)
AVP_Release=zeros(length(Spike),1);
Basal_Secretion=1.3E-5;
if(length(Spike)>1) then
  Last_Spike=Spike(1);                  //Position of the last spike
  AVP_Release(1)=Basal_Secretion;       //The first spike can not have fatigue ou facilitation
  Time_In_Burst=0;                      //Length of the current burst
  Burst_Threshold=1500;                 //Minimal number of time in ms between two consecutive spikes to
consider them in different burst

  for i=2:length(Spike);
    Current_Spike=Spike(i);
    Delta=Current_Spike-Last_Spike;
    if Delta < Burst_Threshold then Time_In_Burst=Time_In_Burst+Delta; end;

AVP_Release(i)=Basal_Secretion*Time_Fatigue(Time_In_Burst)*Freq_Fatigue(Delta)*Freq_Facilitation(Delta);
    Last_Spike=Current_Spike;
  end;
else
  if length(Spike)==1 then AVP_Release=Basal_Secretion; end;
  if length(Spike)==0 then AVP_Release=[]; end;
end;
endfunction
```

//Shorten a spike vector so that the last spike occurence does not exceed the time limit

```
function Short=Shorten(Record,Time_Limit)
index=find(Record>Time_Limit,1)-1;
//Normally a test should be performed here to prevent an error if index is empty, however, the test
performed in the calling function makes sure that it does not happend
Short=Record(1:index);
//Instead of 1 in the last line, a random starting point could be drawn so that the Shortened vector is different
each time which would be more realistic and would potentially increase statistic
endfunction;
```

//Dynamics of the amount of AVP contained in a rat (plasma, ECF, ICF, Organs)

```
function AVPQ=Compute_AVPQ(Release,Half_Life,Initial_Quantity)
tau=Half_Life/log(0.5); //Calculation of the time constant -the name of the variable is misleading
AVPQ=Initial_Quantity*exp(Release(1,1)/(60000*tau))+Release(1,2);//decay of AVP between the beginning
of the simulation and the first spike
for i=2:size(Release,1)
AVPQ(i)=AVPQ(i-1)*exp((Release(i,1)-Release(i-1,1))/(60000*tau))+Release(i,2);
end;
endfunction;
```