```
//*************************************************
//*************************************************
// IPSC Dynamic Clamp
// A collection of scripts written by Jeremy Atherton to model the steady state GP->STN synapse
// initial intention was for the output of conductance waveforms for use with dynamic clamp
// extended to allow modelling of various scenarios and output useful graphs for figures
//*************************************************
//*************************************************




//*****************************************************************************
// Modular scripts (usually only called by main scripts)
//*****************************************************************************


//*************************************************
// singleIPSCmaker()
// Feb 2012, Jeremy Atherton
// Makes a post synaptic potential (could be used for IPSC or EPSC)
// Output is 1 s long with sampling frequency, rise and decay taus as specified
// Script is based on code from Wavemaker.ipf
//
// Update log:
// Feb 2012 -- initial version
//*************************************************
Function/WAVE singleIPSCmaker(sampleFrequency, tauRise, tauDecay)
// Required Parameters
Variable sampleFrequency // in kHz
Variable tauRise, tauDecay // in ms

// initialise some variables
Variable peakTime, peakAmp, scaleFactor

// Work out correct scaling factor
// - PSP waveforms never reach the set peak because they are the sum of two exponentials
// - To address this problem we differentiate the waveform (f(t)') and solve for t when
```

```
// f(t)' is zero (ie. the maximum reached during the PSP; equation 1). We then plug this t
// value into our waveform function (f(t)) to find the amplitude of the peak (equation 2).
// We can then calculate a scaling factor to scale this amplitude to the size that we want.
peakTime = tauRise * ln((tauDecay + tauRise) / tauRise)
peakAmp = (1 - exp(-(peakTime / tauRise))) * exp(-(peakTime / tauDecay))
scaleFactor = 1/peakAmp
print "nscaleFactor = " + num2str(scaleFactor) // <--- scale factor readout if desired for debugging

// Calculate the number of samples and sampling interval in the sweep
variable samples = 1000 * sampleFrequency
variable sampleInterval = 1000/samples

// Make wave for output
String ConductanceS = UniqueName("Conductance", 1, 0 )
Make/N=(samples) $ConductanceS
Wave Conductance = $ConductanceS
SetScale/P x 0,sampleInterval/1000,"s", Conductance

// Build the IPSC
Conductance = scaleFactor * (1 - exp(-p * sampleInterval/ tauRise)) * exp(-p * sampleInterval/ tauDecay)

return Conductance // Return a wave to the calling script
End

//***********************************************
// makeTrain()
// Feb 2012, Jeremy Atherton
// Makes a train of digital spikes (1 sample long) for use in makeIPSCTrain()
// Output is 10 s long with frequency of ~33 Hz and CV of 0.65
//
// Update log:
// Feb 2012 -- initial version
//***********************************************
Function/WAVE makeTrain()
// Make blank wave for output
Make/O/N=200000 outTrain
SetScale/P x 0,5e-05,"s", outTrain
outTrain = 0
```

```
variable startInt = enoise(0.5)+0.5 // start at a random point in the first interspike interval
variable isi, freq = 33, cv = 0.65
variable sd = cv/freq // calculate sd from cv and ISI (1/freq) -> cv = sd/isi = sd * freq

variable firstisi=(gnoise(sd) + (1/freq)) * startInt // calculate time of first spike
variable timeOfSpike = firstisi
do
outTrain[x2pnt(outTrain,timeOfSpike)] = 1 // add spike
isi=gnoise(sd) + (1/freq) // calculate next ISI
timeOfSpike += isi // move to next spike
while(timeOfSpike<10)

return outTrain // Return a wave to the calling script
End

//*************************************************
// makeIPSCTrain()
// Feb 2012, Jeremy Atherton
// Makes a train of IPSCs (or EPSCs) using singleIPSCmaker() and makeTrain()
//
//
// Update log:
// Feb 2012 -- initial version
//*************************************************
Function/WAVE makeIPSCTrain(pattern)
Variable pattern // The script builds patterned and unpatterned versions of the train

// Make the action potential train using makeTrain()
String inTrainS = UniqueName("inTrain", 1, 0 )
Wave TempInTrain = makeTrain()
duplicate TempInTrain $inTrainS
Wave inTrain = $inTrainS
KillWaves TempInTrain

// Make the IPSC template using singleIPSCmaker(sampleFrequency, tauRise, tauDecay)
Wave oneIPSC = singleIPSCmaker(20, 0.4, 7.7) // rise and decay taus measured from cells 2011_04_27-3 to 2011_05_04-3 (n=9)
```

```
// Calculate the conductance of a single IPSC
Variable IPSCcond
do // Numbers from steady state conductance (excluding failures) from cells 090305Ainp2 to 2011_06_21-3
IPSCcond = gnoise(0.269478702764278)+0.118958098946936
while(IPSCcond > 0.681219444293613) // Largest conductance not to be larger than the experimentally measured maximum
IPSCcond = 10^IPSCcond // Log-Normal distribution so draw from normally distributed population (gnoise()) an then raise 10 to the power of the result
IPSCcond = IPSCcond < 0 ? 0 : IPSCcond // Redundant with Log-Normal: check for negative values
oneIPSC*=IPSCcond // scale the template IPSC appropriately

//make two versions of each train; with and without the pattern
String inTrainPS = UniqueName("inTrainP", 1, 0 )
duplicate inTrain $inTrainPS
Wave inTrainP = $inTrainPS
addPattern(inTrainP,pattern) // fudge: use 0 for no pattern (will still get two versions of each train)

//prepare waves for IPSCs
String ipscTrainS = UniqueName("ipscTrain", 1, 0 )
duplicate inTrain $ipscTrainS
Wave ipscTrain = $ipscTrainS
ipscTrain=0
String ipscTrainPS = UniqueName("ipscTrainP", 1, 0 )
duplicate inTrainP $ipscTrainPS
Wave ipscTrainP = $ipscTrainPS
ipscTrainP=0

// Calculate release probability
Variable pr, lpr
do // Numbers from steady state transmission from cells 090115Binp1 and 090305Ainp2 to 2011_06_21-3
lpr = gnoise(0.516174043299264)+0.544918334094934
while(lpr > 1.542) // Highest release probability not to be larger than the experimentally measured maximum
pr = 10^lpr // Log-Normal distribution so draw from normally distributed population (gnoise()) an then raise 10 to the power of the result
pr=pr<0 ? 0 : pr // Redundant with Log-Normal: check for negative values
pr/=100 // Experimentally measured values are for %; we want a scaling factor
//pr = 1 // use for no depression
//print pr // uncomment to check output

// //For normally distributed responses (if required)
```

```
// pr = gnoise(8.35689297169535)+6.96712003448276
// pr=p<0 ? 0 : pr
// pr/=100

// Add IPSCs to the IPSC trains at times defined by the spike trains
// Where the patterned and unpatterned spike trains are identical output should be identical
Variable e
variable i,j
for(i=0;i<numpnts(inTrain);i+=1)
if(inTrain[i]==1 || inTrainP[i]==1) // spike in either unpatterned or patterned train
e = enoise(0.5)+0.5 // random # between 0 and 1
if(e <= pr) //is the random # less than the release probability? If so, transmission is succesful; if not, it's a failure
for(j=0;j<numpnts(oneIPSC);j+=1) // loop to add the template IPSC to the IPSC trains at the appropriate place
if(i+j>=numpnts(ipscTrain)) // stop if gets to the end of the IPSC train wave
break
else
if(intrain[i]==1) // add to unpatterned IPSC train (if needed)
ipscTrain[i+j] += oneIPSC[j]
endif
if(intrainP[i]==1)// add to patterned IPSC train (if needed)
ipscTrainP[i+j] += oneIPSC[j]
endif
endif
endfor
endif
endif
endfor

killWaves oneIPSC // IPSC template not needed any more
return ipscTrain // Return a wave to the calling script
End

//*************************************************
// addPattern()
// Feb 2012, Jeremy Atherton
// Produces patterned output for makeIPSCTrain()
// --NB. played with this a lot for the various simulations run:
// ----case 0 will give no pattern
```

```
// ----cases 1 & 2 have varied as needed
// ----case 3 produces a pause at 5 s into the train
//
// Update log:
// Feb 2012 -- initial version
//*******************************************
Function addPattern(aWave, pattern)
Wave aWave
Variable pattern

Variable timeOfSpike

switch(pattern) // numeric switch
case 0: // execute if case matches expression
break // no pattern
case 1:
aWave[x2pnt(aWave, 4.995 ),x2pnt(aWave, 5.051 )] = 0
timeOfSpike = 5
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.005
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.01
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.025
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.0317
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.0384
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.0451
aWave[x2pnt(aWave,timeOfSpike)] = 1
break // exit from switch
case 2: // execute if case matches expression
aWave[x2pnt(aWave, 2.995 ),x2pnt(aWave, 3.1 )] = 0
// timeOfSpike = 3
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 3.01
// aWave[x2pnt(aWave,timeOfSpike)] = 1
```

```
// timeOfSpike = 3.02
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 3.03
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 3.04
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 3.05
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 3.06
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 3.07
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 3.08
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 3.09
// aWave[x2pnt(aWave,timeOfSpike)] = 1

aWave[x2pnt(aWave, 3.995 ),x2pnt(aWave, 4.1 )] = 0
// timeOfSpike = 4
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 4.01
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 4.02
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 4.03
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 4.04
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 4.05
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 4.06
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 4.07
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 4.08
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 4.09
// aWave[x2pnt(aWave,timeOfSpike)] = 1
```

```
aWave[x2pnt(aWave, 4.995 ),x2pnt(aWave, 5.1 )] = 0
// timeOfSpike = 5
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 5.01
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 5.02
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 5.03
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 5.04
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 5.05
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 5.06
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 5.07
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 5.08
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 5.09
// aWave[x2pnt(aWave,timeOfSpike)] = 1

aWave[x2pnt(aWave, 5.995 ),x2pnt(aWave, 6.1 )] = 0
// timeOfSpike = 6
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 6.01
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 6.02
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 6.03
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 6.04
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 6.05
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 6.06
// aWave[x2pnt(aWave,timeOfSpike)] = 1
```

```
// timeOfSpike = 6.07
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 6.08
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 6.09
// aWave[x2pnt(aWave,timeOfSpike)] = 1

aWave[x2pnt(aWave, 6.995 ),x2pnt(aWave, 7.1 )] = 0
// timeOfSpike = 7
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 7.01
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 7.02
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 7.03
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 7.04
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 7.05
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 7.06
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 7.07
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 7.08
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 7.09
// aWave[x2pnt(aWave,timeOfSpike)] = 1

aWave[x2pnt(aWave, 7.995 ),x2pnt(aWave, 8.1 )] = 0
// timeOfSpike = 8
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 8.01
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 8.02
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 8.03
// aWave[x2pnt(aWave,timeOfSpike)] = 1
```

```
// timeOfSpike = 8.04
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 8.05
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 8.06
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 8.07
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 8.08
// aWave[x2pnt(aWave,timeOfSpike)] = 1
// timeOfSpike = 8.09
// aWave[x2pnt(aWave,timeOfSpike)] = 1
break
case 3: // execute if case matches expression
aWave[x2pnt(aWave, 5 ),x2pnt(aWave, 5.1 )] = 0
break
case 4: //100Hz 100ms
aWave[x2pnt(aWave, 4.995 ),x2pnt(aWave, 5.1 )] = 0
timeOfSpike = 5
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.01
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.02
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.03
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.04
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.05
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.06
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.07
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.08
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.09
aWave[x2pnt(aWave,timeOfSpike)] = 1
```

```
break
case 5: //33Hz 100ms
aWave[x2pnt(aWave, 4.995 ),x2pnt(aWave, 5.1 )] = 0
timeOfSpike = 5
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.0303
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.0606
aWave[x2pnt(aWave,timeOfSpike)] = 1
timeOfSpike = 5.0909
aWave[x2pnt(aWave,timeOfSpike)] = 1
break
case 6: // pause 100ms
aWave[x2pnt(aWave, 5 ),x2pnt(aWave, 5.1 )] = 0
break
endswitch
End
```

```
//*************************************************************************************************
// Main scripts
//*************************************************************************************************


//***********************************************
// addIPSCTrain()
// Feb 2012, Jeremy Atherton
// This is a main script
// Runs the IPSC train generating scripts 60 times to produce 60 unpatterned and 60 patterned fibres
// Results are sum of 1 patterned-59 unpatterned; 10 patterned-50 unpatterned; and 60 patterned
//
// Update log:
// Feb 2012 -- initial version
//***********************************************
Function addIPSCTrain(pattern)
Variable pattern
Variable i,j,offset
```

```
Make/N=200000 outWave01, outWave10, outWave60
SetScale/P x 0,5e-05,"s", outWave01, outWave10, outWave60

String addWaveS
for(i=0;i<60;i+=1)
Wave addWave = makeIPSCTrain(pattern)
addWaveS = NameOfWave(addWave)
addWaveS = addWaveS[0,8] + "P" + addWaveS[9]
Wave addWaveP = $addWaveS
if(i<1)
outWave01 += addWaveP
outWave10 += addWaveP
outWave60 += addWaveP
elseif(i<10)
outWave01 += addWave
outWave10 += addWaveP
outWave60 += addWaveP
elseif(i<60)
outWave01 += addWave
outWave10 += addWave
outWave60 += addWaveP
endif
killwaves addWave, addWaveP
endfor
Display/K=0 root:outWave60,root:outWave10,root:outWave01
ModifyGraph rgb(outWave60)=(0,0,65535),rgb(outWave10)=(0,65535,0)
End

//***********************************************
// addIPSCTrainMeas()
// Feb 2012, Jeremy Atherton
// This is a main script
// Modified from addIPSCTrain()
// Results are for 60 fibres with 10, 20 , 30 , 40 , 50, or 60 patterned fibres
// Used to produce graph of synchronisation vs. integral of conductance
//
// Update log:
// Feb 2012 -- initial version
```

```
//*************************************************
Function addIPSCTrainMeas(pattern)
Variable pattern
Variable i,j,offset
Make/N=200000 outWave00,outWave10, outWave20, outWave30,outWave40, outWave50, outWave60
SetScale/P x 0,5e-05,"s", outWave00,outWave10, outWave20, outWave30,outWave40, outWave50, outWave60

String addWaveS
for(i=0;i<60;i+=1)
Wave addWave = makeIPSCTrain(pattern)
addWaveS = NameOfWave(addWave)
addWaveS = addWaveS[0,8] + "P" + addWaveS[9]
Wave addWaveP = $addWaveS
if(i<10)
outWave00 += addWave
outWave10 += addWaveP
outWave20 += addWaveP
outWave30 += addWaveP
outWave40 += addWaveP
outWave50 += addWaveP
outWave60 += addWaveP
elseif(i<20)
outWave00 += addWave
outWave10 += addWave
outWave20 += addWaveP
outWave30 += addWaveP
outWave40 += addWaveP
outWave50 += addWaveP
outWave60 += addWaveP
elseif(i<30)
outWave00 += addWave
outWave10 += addWave
outWave20 += addWave
outWave30 += addWaveP
outWave40 += addWaveP
outWave50 += addWaveP
outWave60 += addWaveP
elseif(i<40)
```

```
outWave00 += addWave
outWave10 += addWave
outWave20 += addWave
outWave30 += addWave
outWave40 += addWaveP
outWave50 += addWaveP
outWave60 += addWaveP
elseif(i<50)
outWave00 += addWave
outWave10 += addWave
outWave20 += addWave
outWave30 += addWave
outWave40 += addWave
outWave50 += addWaveP
outWave60 += addWaveP
elseif(i<60)
outWave00 += addWave
outWave10 += addWave
outWave20 += addWave
outWave30 += addWave
outWave40 += addWave
outWave50 += addWave
outWave60 += addWaveP
endif
killwaves addWave, addWaveP
endfor
End

//*************************************************
// makeIPSCTrain2() and addIPSCTrain2()
// Feb 2012, Jeremy Atherton
// These are main scripts
// Modification of the generic scripts for if already have the trains
// --must have generated patterned and unpatterned trains beforehand
// Results are sum of 1 patterned-59 unpatterned; 10 patterned-50 unpatterned; and 60 patterned
//
// Update log:
// Feb 2012 -- initial version
```

```
//*************************************************
Function addIPSCTrain2()
Variable i,j,offset
Make/N=200000 outWave01, outWave10, outWave60
SetScale/P x 0,5e-05,"s", outWave01, outWave10, outWave60

String addWaveS
for(i=0;i<60;i+=1)
Wave addWave = makeIPSCTrain2(i)
addWaveS = NameOfWave(addWave)
addWaveS = addWaveS[0,8] + "P" + addWaveS[9]
Wave addWaveP = $addWaveS
if(i<1)
outWave01 += addWaveP
outWave10 += addWaveP
outWave60 += addWaveP
elseif(i<10)
outWave01 += addWave
outWave10 += addWaveP
outWave60 += addWaveP
elseif(i<60)
outWave01 += addWave
outWave10 += addWave
outWave60 += addWaveP
endif
killwaves addWave, addWaveP
endfor
Display/K=0 root:outWave60,root:outWave10,root:outWave01
ModifyGraph rgb(outWave60)=(0,0,65535),rgb(outWave10)=(0,65535,0)
End

// Goes with addIPSCTrain2() -- notes above
Function/WAVE makeIPSCTrain2(run)
Variable run

String inTrainS = "inTrain" + num2str(run)
Wave inTrain = $inTrainS
String inTrainPS = "inTrainP" + num2str(run)
```

```
Wave inTrainP = $inTrainPS

Wave oneIPSC = singleIPSCmaker(20, 0.4, 7.7) // rise and decay taus measured from cells 2011_04_27-3 to 2011_05_04-3 (n=9)

Variable IPSCcond
do
IPSCcond = gnoise(0.269478702764278)+0.118958098946936
while(IPSCcond > 0.681219444293613)
IPSCcond = 10^IPSCcond
IPSCcond = IPSCcond < 0 ? 0 : IPSCcond
oneIPSC*=IPSCcond

//prepare waves for IPSCs
String ipscTrainS = UniqueName("ipscTrain", 1, 0 )
duplicate inTrain $ipscTrainS
Wave ipscTrain = $ipscTrainS
ipscTrain=0
String ipscTrainPS = UniqueName("ipscTrainP", 1, 0 )
duplicate inTrainP $ipscTrainPS
Wave ipscTrainP = $ipscTrainPS
ipscTrainP=0

Variable pr, lpr
do
lpr = gnoise(0.516174043299264)+0.544918334094934
while(lpr > 1.542)
pr = 10^lpr
pr=pr<0 ? 0 : pr
pr/=100
//pr = 1 // use for no depression

Variable e
variable i,j
for(i=0;i<numpnts(inTrain);i+=1)
if(inTrain[i]==1 || inTrainP[i]==1)
e = enoise(0.5)+0.5
if(e <= pr)
for(j=0;j<numpnts(oneIPSC);j+=1)
```

```
if(i+j>=numpnts(ipscTrain))
break
else
if(intrain[i]==1)
ipscTrain[i+j] += oneIPSC[j]
endif
if(intrainP[i]==1)
ipscTrainP[i+j] += oneIPSC[j]
endif
endif
endfor
endif
endif
endfor

killWaves oneIPSC
return ipscTrain
End

//*****************************************************************************************
// Output scripts for making pretty graphs
//*****************************************************************************************


//***********************************************
// makeSpikeRasters()
// Feb 2012, Jeremy Atherton
// Produce raster plots from spike trains generated by makeTrain()
// Use makeWL() to construct the wave list wl for input
//
// Update log:
// Feb 2012 -- initial version
//***********************************************
Function makeSpikeRasters(wl)
String wl
String wn
```

```
String outXS, outYS
display // blank graph for output (will add rasters using AppendToGraph)

Variable i
do
wn = StringFromList(i, wl) // cycle through input wave list
if(WaveExists($wn)==0) // stop when run out of waves from list
break
endif

PickPeaks($wn,0.666667,baseline=0.666667,filter=0) // pick the peaks in the spike train
outXS = UniqueName("outX", 1, 0)
outYS = UniqueName("outY", 1, 0)
duplicate peakDataX $outXS
duplicate peakDataY $outYS
Wave outY = $outYS
outY = i+1
AppendToGraph outY vs $outXS // add raster to graph
ModifyGraph mode($outYS)=3,marker($outYS)=10,rgb($outYS)=(0,0,0) // change display to markers for raster plot
i+=1
while(1)
End

//*************************************************
// makeWL()
// Feb 2012, Jeremy Atherton
// Used to make a wave list for input to makeSpikeRasters()
// Calling syntax could be makeSpikeRasters(makeWL())
// Could be modified for generic use
//
// Update log:
// Feb 2012 -- initial version
//*************************************************
Function/S makeWL()
String wl =""
String wn =""
```

```
Variable i
for(i=0;i<60;i+=1) // adjust this loop and logic to get the required combination of patterned and unpatterned trains in the raster plot
if(i<1)
sprintf wn, "inTrainP%d" i
wl = wl + wn + ";"
else
sprintf wn, "inTrain%d" i
wl = wl + wn + ";"
endif
endfor

return wl
End



//*************************************************
//*************************************************
//*************************************************
//*************************************************
// Used for final generation of trains for the Dynamic Clamp
Function addIPSCTrain3()
Variable i,j,offset
Make/N=200000 outWave100hz100ms, outWave33hz100ms, outWave0hz100ms
SetScale/P x 0,5e-05,"s", outWave100hz100ms, outWave33hz100ms, outWave0hz100ms

String addWave100hz100msS,addWave33hz100msS,addWave0hz100msS
for(i=0;i<60;i+=1)
Wave addWave100hz100ms=makeIPSCTrain3()
addWave100hz100msS = NameOfWave(addWave100hz100ms)
addWave33hz100msS = addWave100hz100msS[0,8] + "P33hz100ms" + addWave100hz100msS[20]
Wave addWave33hz100ms = $addWave33hz100msS
addWave0hz100msS = addWave100hz100msS[0,8] + "P0hz100ms" + addWave100hz100msS[20]
Wave addWave0hz100ms = $addWave0hz100msS
```

```
outWave100hz100ms += addWave100hz100ms
outWave33hz100ms += addWave33hz100ms
outWave0hz100ms += addWave0hz100ms

//killwaves addWave100hz100ms,addWave33hz100ms,addWave0hz100ms
endfor
Display outWave100hz100ms
Display outWave33hz100ms
Display outWave0hz100ms
End

Function/WAVE makeIPSCTrain3()
// Make the action potential train using makeTrain()
String inTrainS = UniqueName("inTrain", 1, 0 )
Wave TempInTrain = makeTrain()
duplicate TempInTrain $inTrainS
Wave inTrain = $inTrainS
KillWaves TempInTrain

// Make the IPSC template using singleIPSCmaker(sampleFrequency, tauRise, tauDecay)
Wave oneIPSC = singleIPSCmaker(20, 0.4, 7.7) // rise and decay taus measured from cells 2011_04_27-3 to 2011_05_04-3 (n=9)

// Calculate the conductance of a single IPSC
Variable IPSCcond
do // Numbers from steady state conductance (excluding failures) from cells 090305Ainp2 to 2011_06_21-3
IPSCcond = gnoise(0.269478702764278)+0.118958098946936
while(IPSCcond > 0.681219444293613) // Largest conductance not to be larger than the experimentally measured maximum
IPSCcond = 10^IPSCcond // Log-Normal distribution so draw from normally distributed population (gnoise()) an then raise 10 to the power of the result
IPSCcond = IPSCcond < 0 ? 0 : IPSCcond // Redundant with Log-Normal: check for negative values
oneIPSC*=IPSCcond // scale the template IPSC appropriately

//make versions of the train for each pattern (100hz100ms; 33hz100ms; 0hz100ms)
//100hz100ms
String inTrainP100hz100msS = UniqueName("inTrainP100hz100ms", 1, 0 )
duplicate inTrain $inTrainP100hz100msS
Wave inTrainP100hz100ms = $inTrainP100hz100msS
addPattern(inTrainP100hz100ms,4)
```

```
//33hz100ms
String inTrainP33hz100msS = UniqueName("inTrainP33hz100ms", 1, 0 )
duplicate inTrain $inTrainP33hz100msS
Wave inTrainP33hz100ms = $inTrainP33hz100msS
addPattern(inTrainP33hz100ms,5)

//0hz100ms
String inTrainP0hz100msS = UniqueName("inTrainP0hz100ms", 1, 0 )
duplicate inTrain $inTrainP0hz100msS
Wave inTrainP0hz100ms = $inTrainP0hz100msS
addPattern(inTrainP0hz100ms,6)

//prepare waves for IPSCs
String ipscTrainP100hz100msS = UniqueName("ipscTrainP100hz100ms", 1, 0 )
duplicate inTrainP100hz100ms $ipscTrainP100hz100msS
Wave ipscTrainP100hz100ms = $ipscTrainP100hz100msS
ipscTrainP100hz100ms=0
String ipscTrainP33hz100msS = UniqueName("ipscTrainP33hz100ms", 1, 0 )
duplicate inTrainP33hz100ms $ipscTrainP33hz100msS
Wave ipscTrainP33hz100ms = $ipscTrainP33hz100msS
ipscTrainP33hz100ms=0
String ipscTrainP0hz100msS = UniqueName("ipscTrainP0hz100ms", 1, 0 )
duplicate inTrainP0hz100ms $ipscTrainP0hz100msS
Wave ipscTrainP0hz100ms = $ipscTrainP0hz100msS
ipscTrainP0hz100ms=0

// Calculate release probability
Variable pr, lpr
do // Numbers from steady state transmission from cells 090115Binp1 and 090305Ainp2 to 2011_06_21-3
lpr = gnoise(0.516174043299264)+0.544918334094934
while(lpr > 1.544) // Highest release probability not to be larger than the experimentally measured maximum
pr = 10^lpr // Log-Normal distribution so draw from normally distributed population (gnoise()) an then raise 10 to the power of the result
pr=pr<0 ? 0 : pr // Redundant with Log-Normal: check for negative values
pr/=100 // Experimentally measured values are for %; we want a scaling factor
//pr = 1 // use for no depression
print pr
```

```
// Add IPSCs to the IPSC trains at times defined by the spike trains
// Where the patterned and unpatterned spike trains are identical output should be identical
Variable e
variable i,j
for(i=0;i<numpnts(inTrain);i+=1)
if(inTrainP100hz100ms[i]==1 || inTrainP33hz100ms[i]==1 || inTrainP0hz100ms[i]==1) // spike in any of the trains
e = enoise(0.5)+0.5 // random # between 0 and 1
if(e <= pr) //is the random # less than the release probability? If so, transmission is succesful; if not, it's a failure
for(j=0;j<numpnts(oneIPSC);j+=1) // loop to add the template IPSC to the IPSC trains at the appropriate place
if(i+j>=numpnts(ipscTrainP100hz100ms)) // stop if gets to the end of the IPSC train wave
break
else
if(inTrainP100hz100ms[i]==1) // add to unpatterned IPSC train (if needed)
ipscTrainP100hz100ms[i+j] += oneIPSC[j]
endif
if(intrainP33hz100ms[i]==1)// add to patterned IPSC train (if needed)
ipscTrainP33hz100ms[i+j] += oneIPSC[j]
endif
if(intrainP0hz100ms[i]==1)// add to patterned IPSC train (if needed)
ipscTrainP0hz100ms[i+j] += oneIPSC[j]
endif
endif
endfor
endif
endif
endfor

killWaves oneIPSC // IPSC template not needed any more
return ipscTrainP100hz100ms // Return a wave to the calling script
End

Function addIPSCTrain4()
Variable i,j,offset
Make/N=200000 outWave100hz100ms, outWave33hz100ms, outWave0hz100ms
SetScale/P x 0,5e-05,"s", outWave100hz100ms, outWave33hz100ms, outWave0hz100ms

String addWave100hz100msS,addWave33hz100msS,addWave0hz100msS
```

```
for(i=0;i<60;i+=1)
Wave addWave100hz100ms=makeIPSCTrain4(i)
addWave100hz100msS = NameOfWave(addWave100hz100ms)
addWave33hz100msS = addWave100hz100msS[0,8] + "P33hz100ms" + addWave100hz100msS[20]
Wave addWave33hz100ms = $addWave33hz100msS
addWave0hz100msS = addWave100hz100msS[0,8] + "P0hz100ms" + addWave100hz100msS[20]
Wave addWave0hz100ms = $addWave0hz100msS

outWave100hz100ms += addWave100hz100ms
outWave33hz100ms += addWave33hz100ms
outWave0hz100ms += addWave0hz100ms

killwaves addWave100hz100ms,addWave33hz100ms,addWave0hz100ms
endfor
Display outWave100hz100ms
Display outWave33hz100ms
Display outWave0hz100ms
End
Function/WAVE makeIPSCTrain4(run)
Variable run

String inTrainS = "inTrain" + num2str(run)
Wave inTrain = $inTrainS
inTrainS = "inTrainP100hz100ms" + num2str(run)
Wave inTrainP100hz100ms = $inTrainS
inTrainS = "inTrainP33hz100ms" + num2str(run)
Wave inTrainP33hz100ms = $inTrainS
inTrainS = "inTrainP0hz100ms" + num2str(run)
Wave inTrainP0hz100ms = $inTrainS


// Make the IPSC template using singleIPSCmaker(sampleFrequency, tauRise, tauDecay)
Wave oneIPSC = singleIPSCmaker(20, 0.4, 7.7) // rise and decay taus measured from cells 2011_04_27-3 to 2011_05_04-3 (n=9)

// Calculate the conductance of a single IPSC
Variable IPSCcond
do // Numbers from steady state conductance (excluding failures) from cells 090305Ainp2 to 2011_06_21-3
IPSCcond = gnoise(0.269478702764278)+0.118958098946936
```

```
while(IPSCcond > 0.681219444293613) // Largest conductance not to be larger than the experimentally measured maximum
IPSCcond = 10^IPSCcond // Log-Normal distribution so draw from normally distributed population (gnoise()) an then raise 10 to the power of the result
IPSCcond = IPSCcond < 0 ? 0 : IPSCcond // Redundant with Log-Normal: check for negative values
oneIPSC*=IPSCcond // scale the template IPSC appropriately


//prepare waves for IPSCs
String ipscTrainP100hz100msS = UniqueName("ipscTrainP100hz100ms", 1, 0 )
duplicate inTrainP100hz100ms $ipscTrainP100hz100msS
Wave ipscTrainP100hz100ms = $ipscTrainP100hz100msS
ipscTrainP100hz100ms=0
String ipscTrainP33hz100msS = UniqueName("ipscTrainP33hz100ms", 1, 0 )
duplicate inTrainP33hz100ms $ipscTrainP33hz100msS
Wave ipscTrainP33hz100ms = $ipscTrainP33hz100msS
ipscTrainP33hz100ms=0
String ipscTrainP0hz100msS = UniqueName("ipscTrainP0hz100ms", 1, 0 )
duplicate inTrainP0hz100ms $ipscTrainP0hz100msS
Wave ipscTrainP0hz100ms = $ipscTrainP0hz100msS
ipscTrainP0hz100ms=0

// Calculate release probability
Variable pr, lpr
do // Numbers from steady state transmission from cells 090115Binp1 and 090305Ainp2 to 2011_06_21-3
lpr = gnoise(0.516174043299264)+0.544918334094934
while(lpr > 1.544) // Highest release probability not to be larger than the experimentally measured maximum
pr = 10^lpr // Log-Normal distribution so draw from normally distributed population (gnoise()) an then raise 10 to the power of the result
pr=pr<0 ? 0 : pr // Redundant with Log-Normal: check for negative values
pr/=100 // Experimentally measured values are for %; we want a scaling factor
pr = 1 // use for no depression


// Add IPSCs to the IPSC trains at times defined by the spike trains
// Where the patterned and unpatterned spike trains are identical output should be identical
Variable e
variable i,j
for(i=0;i<numpnts(inTrain);i+=1)
if(inTrainP100hz100ms[i]==1 || inTrainP33hz100ms[i]==1 || inTrainP0hz100ms[i]==1) // spike in any of the trains
```

```
e = enoise(0.5)+0.5 // random # between 0 and 1
if(e <= pr) //is the random # less than the release probability? If so, transmission is succesful; if not, it's a failure
for(j=0;j<numpnts(oneIPSC);j+=1) // loop to add the template IPSC to the IPSC trains at the appropriate place
if(i+j>=numpnts(ipscTrainP100hz100ms)) // stop if gets to the end of the IPSC train wave
break
else
if(inTrainP100hz100ms[i]==1) // add to unpatterned IPSC train (if needed)
ipscTrainP100hz100ms[i+j] += oneIPSC[j]
endif
if(intrainP33hz100ms[i]==1)// add to patterned IPSC train (if needed)
ipscTrainP33hz100ms[i+j] += oneIPSC[j]
endif
if(intrainP0hz100ms[i]==1)// add to patterned IPSC train (if needed)
ipscTrainP0hz100ms[i+j] += oneIPSC[j]
endif
endif
endfor
endif
endif
endfor

killWaves oneIPSC // IPSC template not needed any more
return ipscTrainP100hz100ms // Return a wave to the calling script
End

Function addIPSCTrain5()
Variable i
for(i=0;i<60;i+=1)
Wave addWave100hz100ms=makeIPSCTrain5(i)
endfor
End
Function/WAVE makeIPSCTrain5(run)
Variable run

String inTrainS = "inTrain" + num2str(run)
Wave inTrain = $inTrainS

// Make the IPSC template using singleIPSCmaker(sampleFrequency, tauRise, tauDecay)
```

```
Wave oneIPSC = singleIPSCmaker(20, 0.4, 7.7) // rise and decay taus measured from cells 2011_04_27-3 to 2011_05_04-3 (n=9)

// Calculate the conductance of a single IPSC
Variable IPSCcond
do // Numbers from steady state conductance (excluding failures) from cells 090305Ainp2 to 2011_06_21-3
IPSCcond = gnoise(0.269478702764278)+0.118958098946936
while(IPSCcond > 0.681219444293613) // Largest conductance not to be larger than the experimentally measured maximum
IPSCcond = 10^IPSCcond // Log-Normal distribution so draw from normally distributed population (gnoise()) an then raise 10 to the power of the result
IPSCcond = IPSCcond < 0 ? 0 : IPSCcond // Redundant with Log-Normal: check for negative values
oneIPSC*=IPSCcond // scale the template IPSC appropriately


//prepare waves for IPSCs
String ipscTrainS = UniqueName("ipscTrain", 1, 0 )
duplicate inTrain $ipscTrainS
Wave ipscTrain = $ipscTrainS
ipscTrain = 0

// Calculate release probability
Variable pr, lpr
do // Numbers from steady state transmission from cells 090115Binp1 and 090305Ainp2 to 2011_06_21-3
lpr = gnoise(0.516174043299264)+0.544918334094934
while(lpr > 1.544) // Highest release probability not to be larger than the experimentally measured maximum
pr = 10^lpr // Log-Normal distribution so draw from normally distributed population (gnoise()) an then raise 10 to the power of the result
pr=pr<0 ? 0 : pr // Redundant with Log-Normal: check for negative values
pr/=100 // Experimentally measured values are for %; we want a scaling factor
//pr = 1 // use for no depression


// Add IPSCs to the IPSC trains at times defined by the spike trains
// Where the patterned and unpatterned spike trains are identical output should be identical
Variable e
variable i,j
for(i=0;i<numpnts(inTrain);i+=1)
if(inTrain[i]==1) // spike in any of the trains
e = enoise(0.5)+0.5 // random # between 0 and 1
if(e <= pr) //is the random # less than the release probability? If so, transmission is succesful; if not, it's a failure
```

```
for(j=0;j<numpnts(oneIPSC);j+=1) // loop to add the template IPSC to the IPSC trains at the appropriate place
if(i+j>=numpnts(ipscTrain)) // stop if gets to the end of the IPSC train wave
break
else
if(inTrain[i]==1) // add to unpatterned IPSC train (if needed)
ipscTrain[i+j] += oneIPSC[j]
endif
endif
endfor
endif
endif
endfor

killWaves oneIPSC // IPSC template not needed any more
return ipscTrain // Return a wave to the calling script
End


//*********************************************************************
//****Second iteration of experiments*****
//*********************************************************************

//To make unique waveforms for each sweep and each cellÑeach protocol to be only used once
Function makeDCsweeps(runs)
Variable runs

Variable i
for(i=0;i<runs;i+=1)
String savDF= GetDataFolder(1) // Save current DF for restore.
String run = "run" + num2str(i)
NewDataFolder/O/S $(run)
addIPSCTrain6(0)
SetDataFolder savDF
endfor
End
Function makeDCsweepsDetonator(runs)
Variable runs
```

```
Variable i
for(i=0;i<runs;i+=1)
String savDF= GetDataFolder(1) // Save current DF for restore.
String run = "run" + num2str(i)
NewDataFolder/O/S $(run)
addIPSCTrain6(0)
Wave ipscTrainDetonator = root:ipscTrainDetonator
Wave outWave
outWave+=ipscTrainDetonator
SetDataFolder savDF
endfor
End


Function addIPSCTrain6(pattern)
Variable pattern

Variable i,j,offset
Make/N=200000 outWave
SetScale/P x 0,5e-05,"s", outWave
Make/N=0 probWave, ampWave //to store fibre steady state probabilities and amplitudes of transmission (generated in makeIPSCTrain6(pattern))

for(i=0;i<60;i+=1)
Wave addWave=makeIPSCTrain6(pattern)
outWave += addWave
endfor
//Display outWave
End

Function/WAVE makeIPSCTrain6(pattern)
Variable pattern
// Make the action potential train using makeTrain()
String inTrainS = UniqueName("inTrain", 1, 0 )
Wave TempInTrain = makeTrain()
duplicate TempInTrain $inTrainS
Wave inTrain = $inTrainS
KillWaves TempInTrain
```

```
// Make the IPSC template using singleIPSCmaker(sampleFrequency, tauRise, tauDecay)
Wave oneIPSC = singleIPSCmaker(20, 0.4, 7.7) // rise and decay taus measured from cells 2011_04_27-3 to 2011_05_04-3 (n=9)

// Calculate the conductance of a single IPSC
Variable IPSCcond
do // Numbers from steady state conductance (excluding failures) from cells 090305Ainp2 to 2011_06_21-3
IPSCcond = gnoise(0.269478702764278)+0.118958098946936
while(IPSCcond > 0.681219444293613) // Largest conductance not to be larger than the experimentally measured maximum
IPSCcond = 10^IPSCcond // Log-Normal distribution so draw from normally distributed population (gnoise()) an then raise 10 to the power of the result
IPSCcond = IPSCcond < 0 ? 0 : IPSCcond // Redundant with Log-Normal: check for negative values
oneIPSC*=IPSCcond // scale the template IPSC appropriately
Wave ampWave
addPnt(ampWave,IPSCcond)

//add the pattern
addPattern(inTrain,pattern)

//prepare wave for IPSCs
String ipscTrainS = UniqueName("ipscTrain", 1, 0 )
duplicate inTrain $ipscTrainS
Wave ipscTrain = $ipscTrainS
ipscTrain=0

// Calculate release probability
Wave probWave //to store fibre steady state probabilities of transmission (wave created in addIPSCTrain6(pattern))
Variable pr, lpr
do // Numbers from steady state transmission from cells 090115Binp1 and 090305Ainp2 to 2011_06_21-3
lpr = gnoise(0.516174043299264)+0.544918334094934
while(lpr > 1.544) // Highest release probability not to be larger than the experimentally measured maximum
pr = 10^lpr // Log-Normal distribution so draw from normally distributed population (gnoise()) an then raise 10 to the power of the result
pr=pr<0 ? 0 : pr // Redundant with Log-Normal: check for negative values
pr/=100 // Experimentally measured values are for %; we want a scaling factor
//pr = 1 // use for no depression
//print pr
addPnt(probWave,pr)
```

```
// Add IPSCs to the IPSC train at times defined by the spike trains
Variable e
variable i,j
for(i=0;i<numpnts(inTrain);i+=1)
if(inTrain[i]==1 ) // spike in the train
e = enoise(0.5)+0.5 // random # between 0 and 1
if(e <= pr) //is the random # less than the release probability? If so, transmission is succesful; if not, it's a failure
for(j=0;j<numpnts(oneIPSC);j+=1) // loop to add the template IPSC to the IPSC trains at the appropriate place
if(i+j>=numpnts(ipscTrain)) // stop if gets to the end of the IPSC train wave
break
else
ipscTrain[i+j] += oneIPSC[j]
endif
endfor
endif
endif
endfor
killWaves oneIPSC // IPSC template not needed any more
return ipscTrain // Return a wave to the calling script
End

Function batchExportATF(runs)
Variable runs

Variable i
Variable aFile
String filenameS
for(i=0;i<runs;i+=1)
String savDF= GetDataFolder(1) // Save current DF for restore.
String run = "run" + num2str(i)

Wave aWave = root:$(run):outWave
Duplicate aWave xWave
xWave=pnt2x(aWave,p) // the scaling on aWave needs to be set in seconds
fileNameS = "Macintosh HD:Users:Jeremy:Documents:00WORK:AMUB2012:Dynamic Clamp Waveforms:ATFHighPLowP:DCLowPRun" +
num2str(i+5) + ".atf"
```

```
Open aFile as filenameS // open file for writing
fprintf aFile, "ATF\t1.0\r0\t2\r\"Time (s)\"\t\"Voltage (V)\"\t\r" // write header
wfprintf aFile, "" xWave, aWave
Close aFile
KillWaves xWave
endfor
End

Function highProbRun(runs)
Variable runs

Variable i
String savDF= GetDataFolder(1) // Save current DF for restore.
String newDF
for(i=0;i<runs;i+=1)
newDF = "root:run" + num2str(i)
SetDataFolder newDF
addIPSCTrain6a()
SetDataFolder savDF
endfor
End
Function addIPSCTrain6a()
Variable i,j,offset
Make/N=200000 outWaveHighP
SetScale/P x 0,5e-05,"s", outWaveHighP

String addWaveS
for(i=0;i<60;i+=1)
Wave addWave = makeIPSCTrain6a(i)
outWaveHighP += addWave
endfor
End
Function/WAVE makeIPSCTrain6a(run)
Variable run

String inTrainS = "inTrain" + num2str(run)
Wave inTrain = $inTrainS
```

```
Wave oneIPSC = singleIPSCmaker(20, 0.4, 7.7) // rise and decay taus measured from cells 2011_04_27-3 to 2011_05_04-3 (n=9)

Variable IPSCcond
Wave ampWave
IPSCcond=ampWave[run]
oneIPSC*=IPSCcond

//prepare waves for IPSCs
String ipscTrainS = UniqueName("ipscTrainHighP", 1, 0 )
duplicate inTrain $ipscTrainS
Wave ipscTrain = $ipscTrainS
ipscTrain=0

Variable pr, lpr
// do
// lpr = gnoise(0.516174043299264)+0.544918334094934
// while(lpr > 1.542)
// pr = 10^lpr
// pr=pr<0 ? 0 : pr
// pr/=100
pr = 1 // use for no depression

Variable e
variable i,j
for(i=0;i<numpnts(inTrain);i+=1)
if(inTrain[i]==1)
e = enoise(0.5)+0.5
if(e <= pr)
for(j=0;j<numpnts(oneIPSC);j+=1)
if(i+j>=numpnts(ipscTrain))
break
else
if(intrain[i]==1)
ipscTrain[i+j] += oneIPSC[j]
endif
endif
endfor
endif
```

```
endif
endfor

killWaves oneIPSC
return ipscTrain
End

Function DCMeasure(wl)
String wl

String wn
Variable index, before, during, beforeArea, duringArea
print "Before"
do
wn = StringFromList(index, wl,",")
if(WaveExists($wn )==0)
break
endif
before += mean($wn, 4900, 5000 )
beforeArea += area($wn, 4900, 5000 )

index+=1
while(1)
print before/5
print beforeArea/5
index=0
print "During"
do
wn = StringFromList(index, wl,",")
if(WaveExists($wn )==0)
break
endif

during += mean($wn, 5000, 5100 )
duringArea += area($wn, 5000, 5100 )

index+=1
```

```
while(1)
print during/5
print duringArea/5
End
```