**Contents of the ‘iv’ struct**

|  |  |
| --- | --- |
| current | List of currents used |
| timertime | ? |
| V\_x | Xth sweep |
| realcurrent | ? |
| bridged | Is it bridge balanced (T/F) |
| holding | ? |
| sweepnum | Number of sweeps used |
| time | Time values at the moment of sampling (e.g. 7000 values) |
| segment | Protocol phase lengths (e.g. 200,800,400) |

**Passive variables (mpassive.m)**

**v0(s)**

mean(Voltage\_values(v0\_avg\_start : v0\_ang\_end))  
Creates a mean for the base voltage between 0.5\*pulse\_start and 0.95\*pulse\_start

**vrs**

mean(Voltage\_values(pulse\_start))  
*The (mean) voltage value at the start of the stimulus*

**vhyp**

vhypstart = find(time\_values < time\_values(vhypend)-lasthypsectocount,1,'last');

vhyp=mean(Voltage\_values(vhypstart:vhypend));  
*Creates a mean voltage for the last ‘*lasthypsectocount’ length interval of the stimulus. (With the 200-800-400 protocol, and 100 ms long*‘*lasthypsectocount’, it’s *the last 12.5 % of it)*

**dvrs**

=v0-vrs;

*The voltage dip immediately after the start of the pulse owing to the electrode capacitance.*

**Dvin**

=vrs-vhyp;

*The voltage difference between the pulse start voltage and the steady state voltage during the stimulus.*

**rin**

=-dvin/(iv.current(sweep))\*1000000;

*The input resistance in MOhm*

**rs**

=-dvrs/(iv.current(sweep))\*1000000;

*The series resistance in MOhm*

**rsag**

dvsag = vrs-data.vsag\_old(sweep)

-dvsag/(iv.current(sweep))\*1000000;

**vrebound**

[data.vrebound(sweep),temp] = max(Volt\_val\_copy(placeof-round(.01\*sampling\_freq) : placeof+round(.01\*sampling\_freq)))

*Searches for the max voltage in the radius of the previously assumed maximum(?) (placeof)*

**trebound**

The time of vrebound (not the index of!)

**dvrebound**

= vrebound-vhyp-dvrs;

*The rebound voltage reduced by the voltage dip and the stimulus steady state voltage*

**vsag**

= min(Voltage\_values(pulse\_start:find(time\_values < 0.4+iv.segment(1)/1000),1,'last'));

*The value of the voltage sag. It finds the min value in the 400ms interval following the start of the stimulus.*

**taunew**

Membrane time constant. Calculated from the pulse start to the sag min.

**samplingrate**

sampling\_freq;

**taustart**

pulse\_start; (time)

**v0**

=mean(data.v0s);  
The mean of the baselines for all sweeps.

**Noiselevel & filterednoiselevel**

Noiselevel = mean(abs(voltage\_values-mean(voltage\_values)))\*1000

filterednoiselevel = mean(abs(voltage\_values-mean(filtered\_voltage\_values)))\*1000

The mean deviance from the mean of the voltages. Butterworth filtering is used.

data.rin\_new = sagstats.rin; Input resistance

data.tau\_new = sagstats.tau; Time constant

data.capacity\_new = sagstats.capacity; Capacity

data.vsag\_new = sagstats.vsag; Sag minimum voltage

data.relsag\_new = sagstats.relsagamp; Relative sag amplitude

data.sagdelay\_new = sagstats.sagdelay; Time of the sag delay

For more details, see the code of sag\_extractor.m

**Active variables (mHH.m)**

**apnum**

apmaskfirst = apmaskfirst(voltage\_values+dvrs(sweep) > -0.010);

[apmask, apnum] = bwlabelhomemade(apmaskfirst);

The number of APs is counted using the relative voltage. If the rel. voltage value surpasses -10mV and the previous rel. voltage val. didn’t surpass it, the AP counter is incremented by 1. Calculated for each sweep.

**Sweep.thresh(time)**

threshold=voltage\_values(thresh\_index)  
data.(['sweep',num2str(sweep)]).tresh(ind) = threshold-abs(threshcorrect\*(voltage\_values(thresh\_index)-voltage\_values(thresh\_index-1)));

The threshold voltage where the AP is thought to commence. This corresponds to the place where the derivative reaches a predetermined value (e.g.10 mV/s). The search for this starts from the AP apex. (Able to estimate the threshold between two voltage values. Corrected using the value below with the formula above. Calculated for each AP. Value and time.

**Sweep.threshcorrect**

threshcorrect = 1-abs((tresholdvalue-dy(thresh\_index-1))/(dy(thresh\_index)-dy(thresh\_index-1)));

Threshold correction multiplier between 0 and 1. This is due to the voltage drop at the beginning (when no bridge balance is applied). Calculated for each AP.

**Sweep.apend(time)**

data.(['sweep',num2str(sweep)]).apend(ind)=apendd+abs(apendcorrect\*(voltage\_values(apend)-voltage\_values(apend-1)));

data.(['sweep',num2str(sweep)]).apendtime(ind)=apendtime-abs(apendcorrect\*(sampleinterval));

Searching for the end of the AP, where the derivative reaches -0.5 \* the predetermined value. The process is the same as with thresh. Calculated for each AP.

**Sweep.apendcorrect**

apendcorrect = abs((dy(apend)+tresholdvalue/2)/(dy(apend)-dy(apend-1)));

Apend correction multiplier between 0 and 1. This is due to the voltage drop at the beginning (when no bridge balance is applied). Calculated for each AP.

**Sweep.aphwstart**

aphw\_helper = (apmax-threshold)/2+threshold-voltage\_values(temp-1);  
aphwstart(ind) = aphw\_helper /(voltage\_values(temp)-voltage\_values(temp-1))\*sampleinterval+time\_values(temp-1);  
First searches for the start of the halfwidth among the discretized values, then estimates the real value based on the method above. Time value.

**aphwv:** Aphw voltage value

**aphwstop:** Aphw end time, using the same method as aphwstart. Time value.

Calculated for each AP.

**Sweep.apmax(time)**

Maximum during each AP (voltage value and time). Calculated for each AP.

**Sweep.oldthresh(time)/oldapend(time)**

=threshold, apend

Bare, uncorrected threshold/apend. Calculated for each AP.

**Sweep.apamplitude**

data.(['sweep',num2str(sweep)]).apamplitude(ind)=(apmax-threshold)\*1000;

AP amplitude value in mV. Calculated for each AP.

**Sweep.halfwidth**

AP halfwidth. Time from aphwstart to aphwstop in msec. Calculated for each AP.

**Sweep.dvmax** and **dvmin**

dvmaxlength=find(dy0(thresh\_index-3:apend+3)==max(dy0(thresh\_index-3:apend+3)),1,'first');

data.(['sweep',num2str(sweep)]).dvmax(ind)=dy0(thresh\_index+dvmaxlength-4);

Maximum of the derivative (dV/dt) during the AP. Calculated for each AP. *No idea why the 4 is subtracted.*

**Sweep.dvmaxv** and **Sweep.dvmaxt** data.(['sweep',num2str(sweep)]).dvmaxv(ind)=yav(thresh\_index+dvmaxlength-4);

The voltage/time value where the derivative is the maximum. Uses the neighbour-average (moving average) values. Calculated for each AP.

**Sweep.postApexMaxDer** data.(['sweep',num2str(sweep)]).compfail(ind)=max(dy0((find(time\_values==apmaxtime)+1):apend));

The maximum derivative value between the AP apex and the end of the AP. Calculated for each AP. *If positive, it signals errors with the AP max calculation.*

**Sweep.interspike**

interspike(ind-1)=(data.(['sweep',num2str(sweep)]).apmaxtime(ind)-data.(['sweep',num2str(sweep)]).apmaxtime(ind-1))\*1000;

Time interval between spike voltage maximums/AP apices. Calculated for each AP. The first element of the array is the mean of the interspike times, hence the array length matches the number of APs.

**meanspike.sweep.time**A simple time array of a given interval, with the sampling frequency. Describes the size of the meanspike interval.

**meanspike.sweep.t**(x)

=([((round(stepsinoneway/2)+data.(['sweep',num2str(sweep)]).treshcorrect(ind))\*sampleinterval):sampleinterval:stepsinoneway\*sampleinterval+data.(['sweep',num2str(sweep)]).treshcorrect(ind)\*sampleinterval])\*1000;

A ‘neighbouring’/’radial’ time array calculated from a predefined step size and the threshold correction constant/multiplier.

**meanspike.sweep.v**(x) meanspike.(['sweep',num2str(sweep)]).(['v',num2str(ind)])=iv.(['v',num2str(sweep)])(-round(stepsinoneway/2)+(find(time\_values == data.(['sweep',num2str(sweep)]).oldtreshtime(ind))+pulse\_start):pulse\_end);

I don’t quite understand it

**vhump**

sag(2,:)=moving(voltage\_values(1:length(find(time\_values<0.4+iv.segment(1)/1000))),round(movingaverageforsag/sampleinterval),'mean');

[data.vhump(sweep),temp]=max(sag(2,:));

If the sweep current is positive and there are no APs recorded, then the script calculates the maximum voltage from start to 400ms after the step began. The maximum calculation uses moving average (for noise reduction), with a predefined constant at the beginning of the script.

**thump**

Time of vhump.

**ramp**

rampstart=find(time\_values<.1+iv.segment(1)/1000 ,1,'last');

data.ramp(sweep,1:2) = polyfit(sag(1,rampstart:end),sag(2,rampstart:end),1);

Calculates the input resistance with straight line fitting (1st order polynomial). If the sweep current is positive and there are no APs recorded it is calculated from step\_start+100ms to step\_start+400ms. If there are APs and a short set of other conditions hold, the calculation interval is different. *See code.*

**reobasesweep**

if any(data.apnum>0) && iv.current(end)>=0

reobasesweep=find(data.apnum>0,1);

The first sweep with action potentials (APs). Must be a positive current sweep, intuitively.

**steadysweep**

if any(data.apnum>8)

steadysweep=find(data.apnum>8,1);

else [~,steadysweep]=max(data.apnum);

The first sweep to contain at least 9 APs. If there’s no such sweep, then the sweep with the most APs.

**sweep.ahpv(time)**

Searches for the minimum between the APend and the next AP’s threshold. Uses moving average. Finds the minimum by incrementing the index while a set of conditions hold.

Only if sweep==reobasesweep || sweep==steadysweep || and(sweep==iv.sweepnum, data.apnum(end)>0)

**sweep.adpv(time)**

Searches for the maximum between the APend and the next AP’s threshold. Uses moving average. Finds the maximum by incrementing the index while a set of conditions hold.

Only if sweep==reobasesweep || sweep==steadysweep || and(sweep==iv.sweepnum, data.apnum(end)>0)

**sweep.maxtime**

The time value of the following

Index of: min(abs(time\_values - The next threshtime,1,'first'))-stepsize\*2, where stepsize is the index equivalent of a 2 ms step.

At the last AP:  
max\_length=length(time\_values)-stepsize\*10

Must be at least 5.

Only if sweep==reobasesweep || sweep==steadysweep || and(sweep==iv.sweepnum, data.apnum(end)>0)

***\*slowevents\* Not working adequately at the moment, in the need of further development***

***(From offsetvoltage.m*** *now incorporated into mHH.m****)***

**HH.sweep.apmax\_corrected**

data.HH.(['sweep',num2str(i)]).apmax\_corrected=data.HH.(['sweep',num2str(i)]).apmax + data.pass.dvrs(i);

Apmax corrected with the voltage dip value.

**HH.sweep.thresh\_corrected**

data.HH.(['sweep',num2str(i)]).apmax\_corrected=data.HH.(['sweep',num2str(i)]).thresh + data.pass.dvrs(i);

Threshold corrected with the voltage dip value.

**HH.sweep.dvmaxv\_corrected** data.HH.(['sweep',num2str(i)]).dvmaxv\_corrected=data.HH.(['sweep',num2str(i)]).dvmaxv + data.pass.dvrs(i);

**HH.sweep.dvminv\_corrected** data.HH.(['sweep',num2str(i)]).dvminv\_corrected=data.HH.(['sweep',num2str(i)]).dvminv + data.pass.dvrs(i);

**HH.sweep.rhump**

dvhump=data.pass.vrs(i)-data.HH.vhump(i);

data.HH.rhump(i)=-dvhump/(iv.current(i))\*1000000;

Only calculated if the current is positive and there are no APs.  
The relative hump voltage is divided by the current. In MOhm.

**Datasum/Calculateelfiz (calculateelfiz\_new.m)**

**RS**

Arithmetic mean of HH.rs from sweeps where the injection current is smaller than -40 pA or larger than +40pA.

**burstspikes**

If there are APs && the interspike data field isn’t empty and contains at least 2 ISI fields && the first AP comes in the 150ms interval following the start of the current injection:

*If the 1st ISI is shorter than 20ms:*

* *While: a next AP exists AND (the next ISI is shorter than twice the current ISI OR the next ISI is shorter than 10ms) -> increment num of burstspikes*
* *If there are still APs left, declares a restISImin: the minimum of the consecutive ISIs (otherwise restISImin is 100 ms)*
* *If there are more than 8 burstspikes OR the time elapsed between the first burstspike and the last is longer than 80ms OR restISImin is shorter than the longest ISI between the burst spikes -> set burstspikes to 0*

**noiselevel**data.pass.noiselevel;

**filterednoiselevel**data.pass.filterednoiselevel;

**sampleinterval**1/data.pass.samplingrate\*1000000;

**sagsold**data.pass.rsag./data.pass.rin(1:length(data.pass.rsag));

**datasum.sagsold**

data.pass.rsag./data.pass.rin\_old(1:length(data.pass.rsag))

**datasum.RS**

RS/RScount;

**sags**data.pass.rsag/data.pass.rinnew;

**lastsag**datasum.sags(end);

**lastrelsag\_new**data.pass.relsag\_new(end);

**lastsagdelay\_new**

data.pass.sagdelay\_new(end);

**lastvsag\_new**

data.pass.vsag\_new(end);

**sagmedian**

median(datasum.sags);

for i=2:4

if length(datasum.sags)>=i && std(datasum.sags(1:i))<.1

datasum.sagmedian=median(datasum.sags(1:i));

end

end

**rebound**

datasum.rebounds=data.pass.dvrebound./data.pass.dvin(1:size(data.pass.dvrebound,2));

datasum.rebound=datasum.rebounds(1);

**HUMP,humps**

if isfield(data.HH,'rhump')

datasum.humps=data.HH.rhump./data.pass.rin(1:length(data.HH.rhump));

datasum.HUMP=(datasum.humps(end));

**Rins**

data.pass.rin(1:5)  
Input resistance  
  
**Rin**

mean(datasum.Rins(1:3))

**Rinstd**std(datasum.Rins(1:3))  
If the stdev is large, (>Rin/10), Let Rin = Rins(1), rinstd = 0;

**vresting**  
v0 in mV

**v0**

mean of v0s

**v0std**

std of v0s

**apnum, maxapnum, maxapsweep, maxapcurrent**apnum, max place of apnum

**reobasesweep** (if none: NaN)

**fIslope**

Maximum APnum divided by (Max AP current minus last current without AP)

**apreduce**1 minus (last AP num divided by AP maxnum)

**taunews**First 5 (or less) taunew

**taunew1, taunewfail1**First element of the respective arrays.

**THE FOLLOWING ARE ONLY GIVEN NUMBER VALUES WHEN THERE ARE APS  
(Else, they are assigned the value NaN)**

**apnumfromreobase**

Nonzero elements of APnum array (from reobase to end)

**steadycurrent, steadyAPnum**Steadycurrent & the APnum of the steadysweep

**goodsteadyAPnum**

steadyAPnum - burstspikes(datasum.steadysweep) - 1

**AP EVENTS, from reobase sweep**

**threshold**thresh\_corrected(1) in mV

**apampl**AP amplitude (1)

**aprisetime**AP max time (1) – thereshold time (1)

**aphw**AP halfwidth (1)

**apwidth**apendtime (1) - thereshold time (1) in msec

**apstartenddiff**thresh (1) – apend (1) in mV

**dvmax, dvmin**dvmax(1)

**dvmaxv, dvminv**dvmaxv\_corrected(1) in mV

**ahpampl**apend(1) – ahpv(1) in mV. Only if the next threshold is not yet reached. If adptime(1) and ahptime(1) are too close OR adpwidth>20 (too large), this is set zero.

**ahpwidth**

ahptime(1) – apendtime(1) in msec. Only if the next threshold is not yet reached. If adptime(1) and ahptime(1) are too close OR adpwidth>20 (too large), this is set zero.

**adpampl**adpv(1) - apend(1) in mV. Only if the next threshold is not yet reached.

**adpwidth**

adptime(1) – apendtime(1) in msec. Only if the next threshold is not yet reached.

*+ Slow components if enabled*

**firstapmaxtime**The time of the first AP relative to the start of the step.

**postApexMaxDer**mean(postApexMaxDer), if >0, set to 1.

**STEADY STATE APs** (for each AP in steadysweep)

**steadyaphws**

halfwidth

**steadyapampls**

apamplitude

**steadythresholds**

thresh\_corrected in mV

**steadyapwidths**

apendtime – threshtime in msec

**steadyapstartenddiffs**

thresh – apend in mV

**steadyahpampls**apend(1) – ahpv(1) in mV. Only if the next threshold is not yet reached. If adptime(1) and ahptime(1) are too close OR adpwidth>20 (too large), this is set zero.

**steadyahpwidths**

ahptime(1) – apendtime(1) in msec. Only if the next threshold is not yet reached. If adptime(1) and ahptime(1) are too close OR adpwidth>20 (too large), this is set zero.

**steadyadpampls**adpv(1) - apend(1) in mV. Only if the next threshold is not yet reached.

**steadyadpwidths**

adptime(1) – apendtime(1) in msec. Only if the next threshold is not yet reached.

**steadyintervals**If the apnum of the steadysweep is larger than the burst spikes there +1, then this is the mean of the interspike intervals from #burstspike-th element +2 to the end of the array.

**burstinterval**

If there are burst spikes at the steady sweep, this is the mean of the interspike interval times from 2nd element to the #burstspike-th element.

**These steady features are averaged and stored in a variable with same name but without plural (e.g. steadyapwidth = mean(steadyapwidths) )**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
if datasum.burstspikes(datasum.steadysweep)+2<datasum.apnum(datasum.steadysweep)

datasum.steadyaphw = mean(deleteoutliers(datasum.steadyaphws(datasum.burstspikes(datasum.steadysweep)+1:end-1)));

datasum.steadyapampl = mean(deleteoutliers(datasum.steadyapampls(datasum.burstspikes(datasum.steadysweep)+1:end-1)));

datasum.steadythreshold = mean(deleteoutliers(datasum.steadythresholds(datasum.burstspikes(datasum.steadysweep)+1:end-1)));

datasum.steadythresholddiff = mean(deleteoutliers(datasum.steadythresholds(datasum.burstspikes(datasum.steadysweep)+1:end-1)-data.pass.v0s(datasum.steadysweep)\*1000));

datasum.steadyapwidth = mean(deleteoutliers(datasum.steadyapwidths(datasum.burstspikes(datasum.steadysweep)+1:end-1)));

datasum.steadyapstartenddiff = mean(deleteoutliers(datasum.steadyapstartenddiffs(datasum.burstspikes(datasum.steadysweep)+1:end-1)));

datasum.steadyahpampl = mean(deleteoutliers(datasum.steadyahpampls(datasum.burstspikes(datasum.steadysweep)+1:end-1)));

datasum.steadyahpwidth = mean(deleteoutliers(datasum.steadyahpwidths(datasum.burstspikes(datasum.steadysweep)+1:end-1)));

datasum.steadyadpampl = mean(deleteoutliers(datasum.steadyadpampls(datasum.burstspikes(datasum.steadysweep)+1:end-1)));

datasum.steadyadpwidth = mean(deleteoutliers(datasum.steadyadpwidths(datasum.burstspikes(datasum.steadysweep)+1:end-1)));

datasum.steadyahpslowampl = mean(deleteoutliers(datasum.steadyahpslowampls(datasum.burstspikes(datasum.steadysweep)+1:end-1)));

datasum.steadyahpslowwidth = mean(deleteoutliers(datasum.steadyahpslowwidths(datasum.burstspikes(datasum.steadysweep)+1:end-1)));

elseif datasum.burstspikes(datasum.steadysweep)==datasum.apnum(datasum.steadysweep)

datasum.steadyaphw = datasum.steadyaphws(end);

datasum.steadyapampl = datasum.steadyapampls(end);

datasum.steadythreshold = datasum.steadythresholds(end);

datasum.steadythresholddiff = datasum.steadythresholds(end)-data.pass.v0s(datasum.steadysweep)\*1000;

datasum.steadyapwidth = datasum.steadyapwidths(end);

datasum.steadyapstartenddiff = datasum.steadyapstartenddiffs(end);

datasum.steadyahpampl = datasum.steadyahpampls(end);

datasum.steadyahpwidth = datasum.steadyahpwidths(end);

datasum.steadyadpampl = datasum.steadyadpampls(end);

datasum.steadyadpwidth = datasum.steadyadpwidths(end);

datasum.steadyahpslowampl = datasum.steadyahpslowampls(end);

datasum.steadyahpslowwidth = datasum.steadyahpslowwidths(end);

else

datasum.steadyaphw = datasum.steadyaphws(datasum.burstspikes(datasum.steadysweep)+1);

datasum.steadyapampl = datasum.steadyapampls(datasum.burstspikes(datasum.steadysweep)+1);

datasum.steadythreshold = datasum.steadythresholds(datasum.burstspikes(datasum.steadysweep)+1);

datasum.steadythresholddiff= datasum.steadythresholds(datasum.burstspikes(datasum.steadysweep)+1)-data.pass.v0s(datasum.steadysweep)\*1000;

datasum.steadyapwidth = datasum.steadyapwidths(datasum.burstspikes(datasum.steadysweep)+1);

datasum.steadyapstartenddiff = datasum.steadyapstartenddiffs(datasum.burstspikes(datasum.steadysweep)+1);

datasum.steadyahpampl = datasum.steadyahpampls(datasum.burstspikes(datasum.steadysweep)+1);

datasum.steadyahpwidth = datasum.steadyahpwidths(datasum.burstspikes(datasum.steadysweep)+1);

datasum.steadyadpampl = datasum.steadyadpampls(datasum.burstspikes(datasum.steadysweep)+1);

datasum.steadyadpwidth = datasum.steadyadpwidths(datasum.burstspikes(datasum.steadysweep)+1);

datasum.steadyahpslowampl = datasum.steadyahpslowampls(datasum.burstspikes(datasum.steadysweep)+1);

datasum.steadyahpslowwidth = datasum.steadyahpslowwidths(datasum.burstspikes(datasum.steadysweep)+1);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**ahpwidthnew**

ahpwidth+apwidth

**ahpamplnew**

apstartenddiff + ahpampl

**accomodation**

If apnum(steadysweep) - burstspikes(steadysweep) > 3, then:  
The last interspike interval (ISI) divided by innterspike(burstspikes(steadysweep)+2).

The ratio of the ISI interval following the first non-burst spike and the last ISI.

**ISIchange**

polyfit(steadysweep.apmaxtime(burstspikes(steadysweep)+2:end), steadysweep.interspike(burstspikes(steadysweep)+2:end),1);

The first coefficient.

**ISIchangeevsapnum**

polyfit(1:length(steadysweep.interspike(burstspikes(steadysweep)+2:end)), steadysweep.interspike(burstspikes(steadysweep)+2:end),1)

The first coefficient.

**bursting**

burstspikes(steadysweep)

**aphwchange** polyfit(steadysweep.apmaxtime(burstspikes(steadysweep)+1:end), steadysweep.halfwidth(burstspikes(steadysweep)+1:end),1);

The first coefficient.

**aphwchangevsapnum**

polyfit(steadysweep.halfwidth(burstspikes(steadysweep)+1:end)), steadysweep.halfwidth(burstspikes(steadysweep)+1:end),1);

The first coefficient.

**apamplchange**

polyfit(steadysweep.apmaxtime(burstspikes(steadysweep)+1:end), steadysweep.apamplitude(burstspikes(steadysweep)+1:end),1);

The first coefficient.

**apamplchangevsapnum**

polyfit(1:length(steadysweep.apamplitude(burstspikes(steadysweep)+1:end)), steadysweep.apamplitude(burstspikes(steadysweep)+1:end),1);

The first coefficient.

**excitability**

polyfix([0:20:(steadysweep- reobasesweep+1)\*20],

[0, apnumfromreobase(1:( steadysweep – reobasesweep + 1))],0,0,1);

(Polyfit through a fixed point, which is (0,0) here)

The first coefficient.

if isfield(data.HH, 'ramp') && datasum.reobasesweep>1 && size(data.HH.ramp,1)>datasum.reobasesweep-2

datasum.ramp=data.HH.ramp(datasum.reobasesweep-1,1);

end

if isfield(data.HH, 'ramp') && size(data.HH.ramp,1)>datasum.reobasesweep-1 && data.HH.(['sweep',num2str(datasum.reobasesweep)]).apmaxtime(1)>.3

datasum.rheobaseramp=data.HH.ramp(datasum.reobasesweep,1);

end

if datasum.reobasesweep>1

datasum.rampnew=data.HH.rampnew(datasum.reobasesweep-1,1);

else

datasum.rampnew=0;

end

datasum.rheobaserampnew=data.HH.rampnew(datasum.reobasesweep,1);