CorrectCoxVramp – a python function to correct for space-clamp errors in voltage-ramp experiments

Ricardo Murphy (ricardo_murphy@fastmail.fm)
University of Oslo, Domus Medica, Division of Physiology,
Pb. 1103, Blindern, 0317 Oslo, Norway

Introduction

The voltage-ramp technique is used to estimate the dependence of a stationary membrane current (*I*) on an applied voltage (*V*). As the name implies, *V* is linearly ramped while *I* is recorded. It is assumed that the ramp is sufficiently slow so that at each point *I* is effectively stationary, i.e. if the ramp were stopped at some point *I* would not change (an assumption that can, of course, be tested experimentally). Hence the *V*-ramp provides no information about kinetics. Nevertheless, it is still useful for estimating the *V*-dependence of a pharmacologically isolated specific conductance $g(V) = i(V)/(V–E)$, where *i* is the current density (e.g. mA cm^{-2}) and *E* is the reversal potential. However, a serious problem when applying the *V*-ramp technique to neurons is that *V* is clamped only at the recording site. This will often be the soma, or perhaps an axonal bleb or varicosity, although proximal dendrites can also be patched. Away from the recording site *V* will not be clamped at the desired values ("space-clamp error"), but the membrane current at those points will contribute to the current measured at the recording site. Hence the interpretation of the recorded values of *I* and *V* is not straightforward; rather one must in some way correct for this space-clamp error. Various approaches have been tried (reviewed briefly by Murphy *et al*., 2021). Here we take as our starting point the method described by Cox (2008), which provides an initial estimate of $i(V)$. This is then improved upon iteratively. Full details of the method are given in Murphy *et al.* (2021). Here I describe the use of a Python function `CorrectCoxVramp`, which operates in conjunction with the simulator NEURON (Carnevale and Hines, 2009) to correct for space-clamp errors in *V*-ramp data and so return an estimate of $i(V)$.

Using the function CorrectCoxVramp

Create a folder (which we will call the "input folder") and put your NEURON files into it, together with a text file containing the *V*-ramp data. An example input folder (`CorrectCoxVrampExample`) is provided. The example ramp-data file is `NaP_Mean_IV.txt`; your data must conform to the data in that file (although the unit of current can be nA rather than pA; see below). Copy the file `CorrectCoxVramp.py` to your input folder. Next create a parameter file and save it to the input folder; you can use the provided `parameters.txt` as a template. The meaning of the parameters is given in the Table 1. Put

your own Python script(s) into the input folder and import `CorrectCoxVramp` by putting a line like the following near the start of your script:

```
import CorrectCoxVramp as cox
```

Calling the function from your script is simple; `result` is a dictionary:

```
result = cox.CorrectCoxVramp('parameters.txt')
```

An example script is provided (`CorrectCoxVrampExample.py`), together with an example NEURON model (the .hoc and .mod files in `CorrectCoxVrampExample`; remember to compile the latter). You can use any parameter file name you like, but parameter names and the colons in the parameter file must not be changed. As some parameters are needed by the NEURON model, these can be loaded from the parameter file by your hoc script (see `mossy_fiber_run.hoc`). You may never need to use `result`, but in case you do its entries are given in Table 2.

Table 1. Definition of parameters.

Parameter	**Example**	**Description**
`neuron_file`	`model.hoc`	NEURON model file.
`maxit`	`9`	Maximum number of iterations.
`DV(mV)`	`0.1`	See ΔV and Eq. (17) in Murphy *et al.* (2021).
`Iunit`	`nA`	Unit of current in `tVIfile` (pA or nA; the latter will be converted to the former).
`Optimise_Cm`	`yes`	yes = optimise C_m and $i(V)$; otherwise C_m fixed.
`Interpolation`	`simple`	Type of interpolation (see Technical Notes).
`maxIerror_tol(pA)`	`0.1`	Maximum allowed error on I.
`Icap_error_tol(pA)`	`0.1`	Maximum allowed error on the capacitative current (I_C) if `Optimise_Cm` = yes.
`V_LJP_corrected`	`no`	Is V corrected for the liquid junction potential?
`tVIfile`	`Vramp.txt`	V-ramp data (t, V, I), e.g. `NaP_Mean_IV.txt`.
`coxdir`	`tempCox`	Temporary folder for intermediate data.

`outputdir`	`output`	Output folder (absolute or relative); **will be deleted if it already exists**.
`Area(um^2)`	`100`	Membrane area of isopotential compartment (e.g. soma, bleb, dendritic section); nseg = 1.
`diam(um)`	`0.4 0.4`	List of neurite diameters (minimum 2).
`Ra(Ohm.cm)`	`100`	Axial resistivity.
`Raccess(MOhm)`	`10`	Patch access resistance.
`Rseal(GOhm)`	`Inf`	Patch seal resistance.
`Cm(uF/cm^2)`	`1.0`	Membrane capacitance (initial or fixed value).
`LJP(mV)`	`2.0`	Liquid junction potential[1].
`gL_fraction`	`0.01`	Use this fraction of $i(V)$ to estimate the leak conductance (g_L) and reversal potential (E_L)[2].
`VRm(mV)`	`-85 -65`	Estimate the specific membrane resistance (R_m) over this V range[2].
`thold(ms)`	`200`	Pre-ramp holding interval.
`tstart(ms)`	`400`	V-ramp start time.
`poly`	`2 3 -80`	Specifies the polynomial(s) used to smooth I[3].
`do_plots`	`yes`	yes = do plots for each iteration.

[1]Used to calculate the leak current through the seal resistance, $(V + \mathrm{LJP})/R_{seal}$.

[2]Not necessary for the algorithm to work; leave blank if you don't want these results.

[3]Smoothing polynomial(s); `cox.test_smooth('parameters.txt')` plots the fit.

`3`	Fit a cubic polynomial to $I(V)$ and $I^*(V)$ (see Murphy *et al*. (2021) for the latter).
`4 4.0`	Find the 'best-fit' polynomial of degree ≤ 4 using forward selection with an F-to-enter value (float) of 4.0 (Kutner *et al.*, 2004).
`2 3 -80`	Splice together polynomials of degrees 2 and 3 at $V \approx -80$ mV (initial guess).

	Don’t fit a polynomial; use linear interpolation.

Table 2. Definition of entries in the `result` dictionary.

Key	**Description**
`t (ms)`	Sampling times for the *V*-ramp data.
`V (mV)`	Observed *V* values after correcting for R_{access} (and LJP if necessary).
`I (pA)`	Observed *I* values after correcting for R_{seal} and capacitative current.
`predI (pA)`	Predicted *I* values after correcting for R_{seal} and I_C. Pairs with `V`.
`cd (mA/cm^2)`	Estimated current density. Pairs with `V`.
`Cm (uF/cm^2)`	Estimated membrane specific capacitance.
`Icap_error (pA)`	Error on the predicted capacitative current (I_C).
`maxIerror (pA)`	Maximum error on the predicted injected current.
`done`	1 = done, 0 = not done.
`Run time (s)`	What it says.
`Best iteration`	Ditto.

Results are also saved to the output folder; file names are given in Table 3. The parameter and *V*-ramp data files are also copied to the output folder. If you wish the temporary folder `coxdir` to be copied to the output folder, add an argument `saveCox = 'yes'` when calling `CorrectCoxVramp`. During the optimisation process, a number of temporary files are also created in the input folder. These files, which are described in Table 4, allow your NEURON program to interact with `CorrectCoxVramp`.

To run a series of optimisations using different parameter values (e.g. different R_a values or different fixed C_m values) put the call to `CorrectCoxVramp` in a loop, creating a new parameter file (containing a different output folder name) for each turn of the loop. E.g.

```
Ra = [90, 100, 110]; j = 0
file = open('parameters.txt','r')
paras = file.readlines(); file.close()
while j < len(paras):
    if 'Ra(Ohm.cm)' in paras[j]: jRa = j
    if 'outputdir' in paras[j]: jout = j
    j = j + 1
i = 0
while i < len(Ra):
    outputdir = 'output_'+str(Ra[i])
    paras[jRa] = 'Ra(Ohm.cm): '+str(Ra[i])+'\n'
    paras[jout] = 'outputdir: '+outputdir+'\n'
    parafile = 'parameters_'+str(Ra[i])+'.txt'
    file = open(parafile,'w')
    file.writelines(paras); file.close()
    result = CorrectCoxVramp(parafile,saveCox = 'yes')
    i = i + 1
```

Technical notes

As discussed in Murphy *et al*. (2021), the method involves the solution of a nonlinear ODE in the current density, i (their Eq. 13). Specification of the ODE requires estimation of a fictitious current (I^*), which is determined by linear interpolation (or extrapolation if this s not possible). The interpolation method used by Murphy *et al*. (2021) is obtained by setting `Interpolation = simple` in the parameter file. Alternative choices are `standard` and `modified`, which are taken from Gerald and Wheatly (1984). In principle these should be more robust than `simple`. `modified` is faster than `standard` and so is the recommended method. The initial value $i = 0$ is taken at $V = V_{rest}$. In principle there can be multiple V_{rest} values but they must be stable equilibrium points (i.e. $dI/dV > 0$ for $I = 0$). If necessary such points are manufactured by linear extrapolation from the beginning and/or end of the $I(V)$ data. If an unstable equilibrium point is present ($dI/dV < 0$ for $I = 0$) it is excluded from the analysis. Eq. (13) is then solved for points to the left and right of the unstable point, followed by concatenation of the two solutions to obtain the final result for $i(V)$. Eq. (13) is solved for $i(V)$ using function `scipy.integrate.solve_ivp` from the *SciPy* Python package. Depending on the setting of `poly`, $I(V)$ and $I^*(V)$ are smoothed using the *SciPy* function `optimize.least_squares`, or the *NUMPY* function `polyfit`, or linearly interpolated using the *SciPy* function `interpolate.interp1d`. Values of V_{rest} are estimated as the roots of the fitted functions, obtained with the function `numpy.roots` for the polynomials or by linear interpolation. All analyses are performed on the smoothed or interpolated data. To check the fit of the polynomial(s) to the raw ramp current, call `cox.test_smooth('parameters.txt')` and if necessary adjust `poly` in the parameters file before calling `CorrectCoxVramp`.

Table 3. Output files names. # is the best iteration number.

File name	Description
`cd_#.txt`	Estimated $i(V)$ in two columns (V, i).
`CmRmL_#.txt`	Estimates of C_m, R_m, g_L and E_L.
`IterationTimeErrors_#.txt`	Run time and errors (I and I_C).
`printout.txt`	Run time, errors and C_m for each iteration.
`Ierror_#.txt`	Prediction errors for I at each V.
`IcapIholdVholdVrest.txt`	Estimate of I_C, I_{hold}, V_{hold} and V_{rest}.
`PredIcapIholdVholdVrest_#.txt`	Predicted values of I_C, I_{hold}, V_{hold} and V_{rest}.
`Raw_tVI.txt`	Raw observed t, V, I (excluding pre-ramp data).
`tVI.txt`	Corrected observed t, V, I (excluding pre-ramp).
`Raw_tVpredI_#.txt`	Raw predicted t, V, I (excluding pre-ramp).
`tVpredI_#.txt`	Corrected predicted t, V, I (excluding pre-ramp).
`Pred_`*`fn`*`_#.txt`	Raw predicted t, V, I (including pre-ramp), where *fn* is the name of the V-ramp data file (`tVIfile` in the parameter file, excluding the ".txt").

Table 4. Temporary files in the input folder.

File name	Description
`tVpredI.txt`	Input for `CorrectCoxVramp`. Save your NEURON-model predictions (t, V, I), including pre-clamp holding data, to this file. The format must conform to that of `tVpredI.txt` in `CorrectCoxVrampExample`.

`flag.txt`	Input for `CorrectCoxVramp`. Create this file with the following hoc code (or equivalent) when your NEURON program has finished running.
	`objref file`
	`proc finish() {`
	`  file = new File()`
	`  file.wopen("flag.txt")`
	`  file.printf("Done")`
	`  file.close()`
	`  quit()`
	`}`
	Anything other than `"Done"` will cause `CorrectCoxVramp` to terminate.

`cd.txt`	Input for your NEURON model. The current best estimate of the current density as a function of V, $i(V)$, in two columns (V, i). Pass these data as a FUNCTION_TABLE to your .mod file (see `cd.txt` and the .hoc and .mod files in folder `CorrectCoxVrampExample`). Note that $i(V)$ includes the leak current density, $g_L(V - E_L)$.

`CmRmL.txt`	Input for your NEURON model. The current best estimate of C_m (μF cm^{-2}). This will not change from the value set in the parameter file if `Optimise_Cm = no`. Import it with something like:
	`objref file`
	`strdef s`
	`file = new File()`
	`file.ropen("CmRmL.txt")`
	`file.scanstr(s)`
	`Cm = file.scanvar()`
	`file.close()`
	Additional lines in the file contain estimates of R_m, g_L and E_L.

Acknowledgement

Funding.

References

Carnevale, N. T. and Hines M. L. (2009). *The NEURON Book*, Cambridge University Press.

Cox S. J. (2008). Direct correction of non-space-clamped currents via Cole's theorem. *Journal of Neuroscience Methods* **169**(2): 366-373.

Gerald C.F. and Wheatly P.O. (1984). *Applied Numerical Analysis*. Addison-Wesley.

Kutner M.H., Nachtsheim C.J., Newman E.L. and Li W. (2004). *Applied Linear Statistical Models*. McGraw-Hill/Irwin

Murphy R., Alle H., Geiger J. and Storm J. (2021). Estimation of persistent sodium-current density in poorly space-clamped mossy fiber axons.