ELLIS 2.2 Manual

Table of Contents

Ellis 2.2

Ellis is a curve-fitting package exclusively for fitting complex functions of one independent variable. It has been used extensively in the optimization of electrochemical impedance models including simple randles cells, warburg diffusion elements, as well as more complicated models such as transmission lines and the point defect model of passivity.

Quickstart

1. Install Ellis

For automatic use, Install the three files (DA, Library and optionally, Batch Tools) into the Igor Procedures Folder. For Selective use, Install in your user procedures folder and include the files in your procedure. Ellis relies on gencurvefit and igor pro version 6.2 or later (see resources).

2. Operate Ellis

Ellis2 is operated via the Ellis2 menu and by editing tables. The menu is designed such that the operator works from the top to the bottom, beginning with option 0:

Click Ellis2 -> Paste your data here

Two tables will appear. Ignore the algsetup table for now. Paste your impedance data into the first 3

columns of the dataTable. (w,Z1,Z2). If your data is in Hz, use the Frequency Conversion submenu to convert into angular frequency (w).

3. Run your setup function of choice from the menu

Ellis2->Run your setup function -> Randles for instance.

IF no figures are open, do Ellis2->Display tables and Graphs

To do the fit, choose Ellis2 -> Fit

That's it!

Programming Functions for Ellis2

You can, of course, program your impedance function in the procedure window of igor pro (windows->procedure window->procedure window). However, you may find it more useful to save these functions as individual .ipf files, so that you can load them at will into any igor experiment, which can be helpful when analyzing a large quantity of data.

Each function for Ellis2 has three parts; The Menu, the setup function and the impedance function.

1. The Menu

Adds your setup function to the ellis2 menu for easy access. In the example below, the menu entry has two arguments, some text in quotations, which will be what the user sees in the menu, and the function to be run. In this case, "setupMyFunction()", which calls the setup function.

```
Menu "Ellis2" //don't edit this
      Submenu "1. Run your setup function" //don't edit this
            "1.C Setup MyFunction", setupMyFunction() //menu entry
      End
End
```

2. The Setup Function

prepares Ellis to handle your funciton correctly. It defines the names, limits and initial values of the coefficients, whether or not the coefficients should vary, and tells igor the name of the impedance function. Finally, it must call preflight1(), which processes all of this new information (and with a little help from the data table), automatically sets up the rest of the things ellis needs to do the fit. The name of the setup function must match the name declared in the menu item.

```
Function setupMyFunction() //This is an impedance function for a randles cell.  As
such, there are 3 coefficients, RS, Cdl and RP.
      Make/T/O coeffnames={"RS","Cdl","RP"}     // make a list of coeff. names
      Make/O/D coefs = {100,0.000001,3e3} // make the initial coefficients
      Make/O/D limits ={{0,0,0},{1e3,0.1,1e8}} // make the lower and upper limits.
      //Make sure that coefs fall within these limits, or you will have problems.
      Make/O/I/N=(numpnts(coefs)) heldlist=0         // Makes the held list
      //decides whether or not each coefficient should be held.
      //0 = not held.  1 = held. (during fit).  Using this line, EVERYTHING will
      //vary unless dictated by the user.
      //Make/O/I/N=(numpnts(coefs)) heldlist={1,0,0} //use this line instead of
      //the above line if you need to specify coefficients that should ALWAYS be
      //held.  In this example, I have set RS to be held by default.
      string/g currentmodel="Zrandles" //tell ellis the impedance function's name.
      make/t/o tempwavenames={"complexer"} //Declare any temporary waves you need.
      //ALL functions since Ellis 2.2 REQUIRE "complexer".  You may add extra
      //entries to this list and the appropriate temporary complex waves will be
      //made during preflight.
      preflight1() //Don't edit this.  This calls the preflight that processes
      //everything above and makes ellis ready to do a fit.  All setup functions
      //are required to call preflight1.
End
```

Waves required to be made by the setup function (assuming a coefficient vector length N):

coeffnames: Text wave of length N containing the name of each coefficient in the desired order.

coefs: double wave of length N containing a set of default initial values.

limits: 2-dimensional double wave of dimensions 2xN in the following format: {{lower limits},{upper limits}}

heldlist: Integer wave of length N containing only the numbers 1 and 0. Coefficients to be held are marked by "1", while coefficients to vary are marked by "0".

currentmodel: string that contains the name of the impedance function.

tempwavenames: Text wave containing a list of temporary complex waves to be generated by preflight.

The setup function can perform other duties as dictated by the needs of the operator, but the above is the bare minimum requirement.

3. The Impedance function

Impedance functions are in a special format because igor cannot fit complex functions and data natively.

The name of the impedance function must match the name declared in the setup function. There are three arguments for this function, because it is an "all-at-once" function.

The first argument (here, w) is the solution vector provided to the function by the genetic algorithm. It is identical in properties to the coefs wave declared in the setup function.

The second argument (here, complexcat) is the return wave. During the function, the results will be written to this wave. The name of this wave should not be altered.

The third argument (here, freqw), is the independent variable's wave, generally frequency, although you are not explicitly limited to frequency. Because this wave is internal to the function, you can call it whatever you wish, so long as you are consistent.

<igor>

```
//Impedance Function
//Define your impedance function.  You must define a complex impedance function,
not "parts 1 and 2"

Threadsafe Function MyFunction(w,complexcat,freqw): fitfunc
      wave complexcat, w, freqw//declare wave names used by default.  Don't edit
      wave/c/d complexer // don't edit this.  Note that you must have a properly
      //scaled complexer wave in your data directory.  If you have made a proper
      //setup function, this will be handled automatically.
      variable/g midpointt //pull in midpoint value.  This is what divides the
      //real and imaginary parts.  Do not edit this.
      variable/g/c ki//pull in complex constant  Do not edit this.  You can also
      //pull in kii, which is equal to (1-j)
      //variable/g/c/kii //Optional.  See above.
      variable Rs, Rp, Cdl //declare variable names; you can edit this.  For
      //sanity's sake, these should match coeffnames from your setup function.
      Rs=w[0] //assign your variable names to the solution wave.  Edit this.
      Cdl=w[1]
            Rp=w[2]

      //Calculate your function here.  Make sure to keep track of temporary waves
      //and variables.  Each line in this section should (in general) begin with a
      //temporary wave name as defied in tempwavenames in your setup function.,
      //however you can do more complicated things here as well.
      //This example is simple enough to be computed on one line, so we do not
      //need to use this section.

      complexer[0,midpointt-1] = Rs+1/(1/Rp+1/(ki/-(Cdl*freqw)))
      //Compute the final value of the complex function here.
      complexcat[0,midpointt-1]=real(complexer[p]) // don't edit this line.
      //Writes real part to complexcat first half
      complexcat[midpointt, ]=imag(complexer[p-midpointt]) //don't edit this line.
      //Writes imaginary part to complexcat second part
End
```

Advanced Programming

The package that Ellis relies on, gencurvefit, is unable to receive constraints. This is generally not a problem, because every parameter must be constrained within the limits imposed by the user. However, constraints may be imposed by careful design of the objective function. For instance, in a physical model in which the total number of sites must add up to 1, but there are multiple types of active sites, it would be ideal to specify that c1,c2,c3 are between 0 and 1 and that c1 + c2 <1 and

c3=1-c1-c2, yielding two varying coefficients that are directly interpretable. This is unfortunately not possible due to the use of gencurvefit to provide the fitting engine. Therefore, the user needs to be clever in order to ensure that the system remains physical. This is accomplished by using pseudocoefficients:

The following pseudo-coefficient parameters are inputs for the model

```
0 < c1x < 1
0 < c2x < 1
0 < c3x < 1
```

and internally, the real coefficients are calculated and used:

```
c1=c1x/(c1x+c2x+c3x)
c2=c2x/(c1x+c2x+c3x)
c3=c3x/(c1x+c2x+c3x)
```

Another example of advanced programming is the use of logic in the objective function. For instance, it is possible to calculate a negative film thickness when working with the point defect model. In the past, this was not important, because it meant that the barrier layer was non-physical and therefore the thickness was equal to zero. The computer, however, does not understand this and with new developments (the bi-layer model and the incorporation of potential drops across the film, for instance), it is important to have a physical dimension for the film even if it is zero because it is used in subsequent calculations.

```
Lss=(1/b3)*log(c)-(a3*(w[16]-w[22]*w[17]))*(1/b3)-(c3*w[19])*(1/b3) //Lss is
calculated
if (Lss<0)  // if Lss is negative, set the thickness to zero.
      Lss=0
else
      Lss=Lss
endif
```

General Notes

ki is the complex value sqrt(-1), provided by ellis. It is faster computationally to use ki than to calculate this during evaluation. In the same way, kii is 1-ki, which is useful for the warburg impedance and is a much faster implementation.

Note the use of the temporary complex wave "complexer", which was specified in the setup function above.

Also, notice how the solution vector was assigned to various human-readable variables. This is optional but strongly encouraged. While this method uses slightly more memory, it does not impact execution performance (in some cases it is actually faster), and allows you to much more easily maintain your code.

Algorithm Control

The algsetup table contains optional controls for controling the genetic algorithm. For more complete explinations, see gencurvefit's help file.

Method: 0 or 1 (Default 0): Method of cost function. 0: Chi squared. 1: Absolute error

Quiet: 0 or 1 (Default 1): Set 0 to have more information dumped into the history window during the fit. May be useful for debugging.

Population: integer>0. (Default 20) the population size is determined by multiypliyng the population value by the number of varying coefficients. For example, a 3 parameter model with population 20 would have a population size of 60. A larger population will require more time per generation, but it will also have a larger diversity than a smaller population.

Crossover: number between 0 and 1. (Default 0.8) Determines the fraction of solutions allowed to exchange information to produce daughter solutions.

Mutator: number between 0 and 1. (Default 0.8) Controls the degree of mutation. 1 is the highest (everything mutates all the time) and 0 means that solutions will never mutate.

Seed: (Default 0). Set to 0, a random seed is used for every fit. When set to any other number, the same seed will be used, so identical results should be obtained when repeating the fit with the same seed. It can be used to troubleshoot algorithms.

Iterations: integer>0. (Default 100). Controls the number of generations of evolution for the algorithm to traverse. A large number of iterations will take more time to complete, while a short number of iterations may not converge.

Strategy: integer from 0 to 9 (default 0): Determines selection of differential evolution strategy. It is genereally safe to leave this at the default, 0.

/TEMP: Default 0. Set this to any other value and a monte carlo method will be used instead of an absolute method during selection. Slows convergence, covers wider area, requires more computational time.

Tolerance: (Default 1e-07) The fit will stop once the standard deviation of the error is less than this value. Lower values will force exhaustive search while higher values will terminate sooner.

Trials: integer>=0. (default 0). Used with the menu command "run error stack", fit will repeat for the set number of trials, appending the error trajectory to the error vs. time plot.

Trial number: (Default 0). Nothing more than a counter to tell you what trial you are on during an error stack procedure.

Data Structure

The data currently being worked on by ellis is located in root:ellis:onDeck. Ellis2 only operates within this folder!

Saving and restoring:
To make a complete copy of this data, open the Data Browser and option-drag (alt-drag on a pc) the onDeck folder to another location. You can then rename the new folder. If you would like to restore one of your saved folders, option-drag the contents of that folder into onDeck. You will need to close all of the open tables and graphs before doing this.

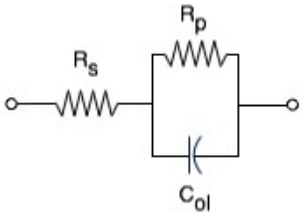
Macros

Ellis2 adds various graph and table macros to igor. If you are missing a specific window, you can open it again, by clicking windows -> graph macros (or table macros). Some of the available windows are not used by default and can be found here.

Impedance Library

A library of impedance functions are included with Ellis to help you get started.

Randles Cell

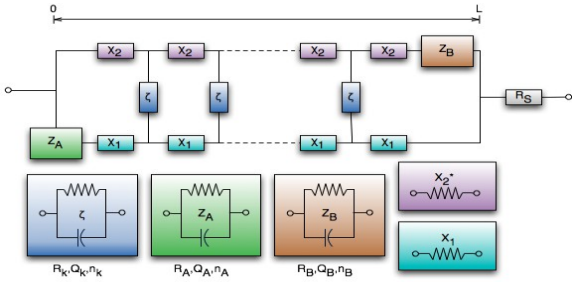


QPE

Identical to the Randles Cell, but Cdl is replaced with a constant phase element.

Transmission Line

An infinite series of impedance elements, this model is often used for porsity. In an unbalanced transmisison line, X2 is assumed to be 0.



$$Z = R_S + \frac{L\lambda\chi_1\chi_2(\chi_1+\chi_2)S_\lambda + \chi_1[\lambda\chi_1 S_\lambda + L\chi_2 C_\lambda]Z_A + \chi_2(\lambda\chi_2 S_\lambda + L\chi_1 C_\lambda)Z_B + \left[2\chi_1\chi_2 + \left(\chi_1^2+\chi_2^2\right)C_\lambda + \frac{L}{\lambda}\chi_1\chi_2 S_\lambda\right]\frac{Z_A Z_B}{\chi_1+\chi_2}}{(\chi_1+\chi_2)\left[\lambda(\chi_1+\chi_2)S_\lambda + (Z_A+Z_B)C_\lambda + \frac{Z_A Z_B S_\lambda}{\lambda(\chi_1+\chi_2)}\right]}$$

Troubleshooting

Make sure not to include frequency 0 or infinity in your data. This happens sometimes by accident and results in a failure by ellis.

If you are getting strange problems or failures, try turning on the debug mode of igor: Procedure -> Enable debugger to troubleshoot your function.

Changes since version 1

Significant performance increases (5x-20x speed)
Real-time bode plots (no longer need to refresh static plots for phase angle and modulus)
Real-time fit analytics (variation of parameters, progress vs. time)

Finer Control over the genetic algorithm

All-at-once functions now mandatory
Complex functions now mandatory - single function per system
All values are stored as 64-bit floating point complex or reals.

Future Features

enable masking
enable weighting
enable batching

Changelog

2.2 release 1

First public release of ellis

2.2 release 2

Bug fix: Phase angle is properly displayed. This was a display bug and in no way affects the curve fit.

Change: Order of data table is now wdata, z1data, z2data. This change was made it is the default order for most incoming data files.

New Feature: User update Function option. *Advanced feature* most users will not need to worry about this.

New Feature: Calculate and display confidence bands via a monte carlo method. (normal

distribution about coefs +/- W_sigma)

New Function added to the library: N-degree voigt model

Resources

Quickstart Video https://vimeo.com/43998169
Gencurvefit http://www.igorexchange.com/project/gencurvefit
Easy Multithreading http://www.igorexchange.com/project/multithreader
Igor Pro http://www.wavemetrics.com/