Electrochemical Impedance Spectroscopy (EIS) Analysis

Steven A. Policastro[[1]](#footnote-1) and Rachel M. Anderson

Center for Corrosion Science and Engineering, U.S. Naval Research Laboratory, 4555 Overlook Avenue SW Washington, DC 20375

This document serves as a summary and README for the classes and functions that were developed to analyze EIS data. This project was developed to facilitate the analysis of electrochemical impedance spectroscopy (EIS) data obtained from potentiostatic EIS experiments.

**Note**: The code has not been tested to determine if it works with galvanostatic EIS data.

The main function depends on files contained in the following sub-directories:

* "Circuit Element Models"
* "Equivalent Circuit Models"
* "Fitting Algorithms"

Data files are expected to be found in the "Data" directory. Fit results are stored in the "Fit" directory.

All functions and classes are written in MATLAB®.

Record of Revisions:

* Created - June 2024
* Last revision: 12-July-2024

# Main Function

This function serves as the entry point for the project to analyze and fit EIS data. It can be customized depending on the physical system the EIS data was collected from, but it needs to perform the following actions:

* Clear the command window, all figures, and all variables from the workspce.
* Add the paths to the sub-directories containing the functions for the equivalent circuit impedance calculations and other supporting classes.
* Create a cell array of data filenames
* Iterate through the filenames to extract the impedance data.
* Instantiate an instance of the eisFitController class for each datafile, pass the data to the constructor as well as the equivalent circuit type, and initial guess of the fit parameter values

Definitions for the variables in the Main function are provided below:

* datafilenames = cell arrray containinng filenames of the raw Gamry data files
* selectedEquivalentCircuit = character vector specifying the equivalent circuit to be used to fit the data. Available circuits are listed in the following sub-section
* vectorOfInitialParameterEstimates = array of values for the initial parameter estimates
* legendString = character array of the descriptions of the data to be plotted on the output plots

## Available Equivalent Circuits:

Use Table 1 to specify the equivalent circuit impedance function to use in the fitting routine. In the provided code example, it can be seen that ‘ModifiedRandles’ was the circuit that was selected.

Table 1. Available equivalent circuit impedance functions for fitting to EIS data. The 2nd column lists the value for the selectedEquivalentCircuit class instance that is created for each datafile. The 3rd column lists the number of initial parameter values that must be passed to the fitting function.

|  |  |  |
| --- | --- | --- |
| **Circuit Name** | **Name to pass to the EISFitController Instance** | **Number of fit parameters needed in  vector** |
| Undamaged Coating | UndamagedCoating | 3 |
| Modified Undamaged Coating | ModifiedUndamagedCoating | 4 |
| Randles | Randles | 5 |
| Modified Randles | ModifiedRandles | 6 |
| Nested Randles Coating Defect | NestedRandlesCoatingDefect | 5 |
| Rapid Electrochemical Assessment of Paint (REAP) | REAP | 6 |
| Modified REAP | ModifiedREAP | 7 |

Examples of the output text with the fit parameter values and a Bode plot of the EIS data and fit are shown in Figure 1 and Figure 2, respectively.

function eisAnalysisMain  
 clc;  
 clear all;  
 addpath( ...  
 'Circuit Element Models', ...  
 'Equivalent Circuit Models', ...  
 'Fitting Algorithms', ...  
 'Data', ...  
 'Fits' ...  
 )  
 format short  
 base\_data\_dir = 'Data';  
 base\_fits\_dir = 'Fits';  
 ext1 = '.dta';  
 ext2 = '.DTA';  
 datafilenames = {'EIS\_PureNi\_Anodic\_24hrOCP\_\_495\_Trial3'};  
 legendString = 'AM - 504h NSW';  
  
 for fn = datafilenames  
 ffn1 = fullfile(base\_data\_dir,strcat(char(fn),ext1));  
 ffn2 = fullfile(base\_data\_dir,strcat(char(fn),ext2));  
 selectedEquivalentCircuit = 'ModifiedRandles';  
 vectorOfInitialParameterEstimates = [1.0e1,3.0e4,1.0e-4,0.8,7.0e2,1.5];  
 good = 0;  
 if isfile(ffn1)  
 [~,eisdata] = AnalyzeGamryEISData(ffn1);  
 good = 1;  
 elseif isfile (ffn2)  
 [~,eisdata] = AnalyzeGamryEISData(ffn2);  
 good = 1;  
 end  
 if good == 1  
 eisFitController(eisdata,selectedEquivalentCircuit,vectorOfInitialParameterEstimates,base\_fits\_dir,legendString);  
 else  
 fprintf("File %s not found.",fn,"/n");  
 end  
 end  
end

|  |
| --- |
| ============================== Results from the fit algorithm ============================== Rs = 1.288399e+01 Rp = 6.821130e+06 Y0 = 1.173451e-05 α = 9.577496e-01 σ = 1.993376e+06 B = 1.760177e-01 ============================== Analysis of EIS data completed. |

Figure 1. Example output from the fitting routine.

|  |
| --- |
|  |

Figure 2. Bode plot of the impedance modulus vs potential oscillation frequency and the phase of the response vs the potential oscillation frequency. Open symbols indicate data, the line indicates the fit.

# Principal Supporting Functions and Classes

The principal supporting function that is used by the model is the eisFitController function. This function initiates and calls the important functions for performing the equivalent circuit fit of the EIS data and then displays the results.

# eisFitController

This function calls the specified equivalent circuit model that will fit the data and passes the initial parameter set estimate to the fitting function. It also establishes the constratint limts on the fit parameters for the simplex algorithm and calls the fitting function. Lastly, it stores the results of the fit in output files. Fit parameter values are output to the command window.

## Record of Code Revisions:

* Created - June 2024
* Last revision: 12-July-2024

## Function definition

Function inputs

* T = instance of eisdata class
* fType = name of the equivalent circuit model to be used to analyze the data
* beta0 = array of values for the initial guess for the fit
* fitDirectory = directory where fit results are stored
* legendForPlotString = string of descriptions for the data on the plot

Function outputs

* Files with fit values and list of fit parameters to the command window

function eisFitController(T,fType,beta0,fitDirectory,legendForPlotString)

## Branch execution on equivalent circuit type

This extended switch statement determines the equivalent circuit fit function, the parameters that will be altered to obtain the fit, the constraint limits on the parameters, and initializes the output string

switch char(fType)  
  
 case 'UndamagedCoating'  
 fun = @matrixImpedance\_UndamagedCoatingPlain;  
 activeParams = [1,1,1]';  
 constraintLimits = [ ...  
 1.0e-1, 1.0e3; ...  
 1.0e1, 1.0e12; ...  
 1.0e-12, 1.0e-1 ...  
 ];  
  
 s0 = "Results from the fit algorithm"+ newline;  
 s00 = "=============================="+ newline;  
 s1 = "Rs = %d" + newline;  
 s2 = "Rp = %d" + newline;  
 s3 = "C = %d" + newline;  
 s7 = s00 + s0 + s00 + s1 + s2 + s3 + s00;  
  
 case 'ModifiedUndamagedCoating'  
 fun = @matrixImpedance\_ModifiedUndamagedCoatingPlain;  
 activeParams = [1,1,1,1]';  
 constraintLimits = [ ...  
 1.0e-1, 1.0e3; ...  
 1.0e-1, 1.0e12; ...  
 1.0e-12, 1.0e-1; ...  
 0.4 1.0];  
 s0 = "Results from the fit algorithm"+ newline;  
 s00 = "=============================="+ newline;  
 s1 = "Rs = %d" + newline;  
 s2 = "Rp = %d" + newline;  
 s3 = "Y0 = %d" + newline;  
 s4 = "α = %d" + newline;  
 s7 = s00 + s0 + s00 + s1 + s2 + s3 + s4 + s00;  
  
 case 'Randles'  
 fun = @matrixImpedance\_RandlesPlain;  
 activeParams = [1,1,1,1,1]';  
 constraintLimits = [ ...  
 1.0e-1, 1.0e3; ...  
 1.0e1, 1.0e12; ...  
 1.0e-12, 1.0e-1; ...  
 1.0e-1,1.0e12; ...  
 1.0e-2, 1.0e2 ...  
 ];  
 s0 = "Results from the fit algorithm"+ newline;  
 s00 = "=============================="+ newline;  
 s1 = "Rs = %d" + newline;  
 s2 = "Rp = %d" + newline;  
 s3 = "C = %d" + newline;  
 s5 = "σ = %d" + newline;  
 s6 = "B = %d" + newline;  
 s7 = s00 + s0 + s00 + s1 + s2 + s3 + s5 + s6 + s00;  
  
 case 'ModifiedRandles'  
 fun = @matrixImpedance\_ModifiedRandlesPlain;  
 activeParams = [1,1,1,1,1,1]';  
 constraintLimits = [ ...  
 1.0e-1, 1.0e3; ...  
 1.0e-3, 1.0e12; ...  
 1.0e-12, 1.0e-1; ...  
 0.4, 1.0; ...  
 1.0e-1,1.0e12; ...  
 1.0e-2, 1.0e2 ...  
 ];  
 s0 = "Results from the fit algorithm"+ newline;  
 s00 = "=============================="+ newline;  
 s1 = "Rs = %d" + newline;  
 s2 = "Rp = %d" + newline;  
 s3 = "Y0 = %d" + newline;  
 s4 = "α = %d" + newline;  
 s5 = "σ = %d" + newline;  
 s6 = "B = %d" + newline;  
 s7 = s00 + s0 + s00 + s1 + s2 + s3 + s4 + s5 + s6 + s00;  
  
 case 'NestedRandlesCoatingDefect'  
 fun = @matrixImpedance\_NestedRandlesCoatingDefectPlain;  
 activeParams = [1,1,1,1,1]';  
 constraintLimits = [ ...  
 1.0e-1, 1.0e3; ...  
 1.0e1, 1.0e12; ...  
 1.0e-12, 1.0e-1; ...  
 1.0e1, 1.0e12; ...  
 1.0e-12, 1.0e-1; ...  
 ];  
 s0 = "Results from the fit algorithm"+ newline;  
 s00 = "=============================="+ newline;  
 s1 = "Rs = %d" + newline;  
 s2 = "Rc = %d" + newline;  
 s3 = "Cc = %d" + newline;  
 s4 = "Rp = %d" + newline;  
 s5 = "Cdl = %d" + newline;  
 s7 = s00 + s0 + s00 + s1 + s2 + s3 + s4 + s5 + s00;  
  
 case 'REAP'  
 fun = @matrixImpedance\_REAPPlain;  
 activeParams = [1,1,1,1,1,1]';  
 constraintLimits = [ ...  
 1.0e-1, 1.0e3; ...  
 1.0e-1, 1.0e12; ...  
 1.0e-12, 1.0e-1; ...  
 1.0e-1,1.0e12; ...  
 1.0e-12, 1.0e-1; ...  
 0.4, 1.0; ...  
 ];  
 s0 = "Results from the fit algorithm"+ newline;  
 s00 = "=============================="+ newline;  
 s1 = "Rs = %d" + newline;  
 s2 = "Rpo = %d" + newline;  
 s3 = "Cc = %d" + newline;  
 s4 = "Rp = %d" + newline;  
 s5 = "Y0 = %d" + newline;  
 s6 = "α = %d" + newline;  
 s7 = s00 + s0 + s00 + s1 + s2 + s3 + s4 + s5 + s6 + s00;  
  
 case 'ModifiedREAP'  
 fun = @matrixImpedance\_ModifiedREAPPlain;  
 activeParams = [1,1,1,1,1,1,1]';  
 constraintLimits = [ ...  
 1.0e-1, 1.0e3; ...  
 1.0e-1, 1.0e12; ...  
 1.0e-12, 1.0e-1; ...  
 0.4, 1.0; ...  
 1.0e-1,1.0e12; ...  
 1.0e-12, 1.0e-1; ...  
 0.4, 1.0; ...  
 ];  
  
 s0 = "Results from the fit algorithm"+ newline;  
 s00 = "=============================="+ newline;  
 s1 = "Rs = %d" + newline;  
 s2 = "Rpo = %d" + newline;  
 s3 = "Y0c = %d" + newline;  
 s4 = "αC = %d" + newline;  
 s5 = "Rp = %d" + newline;  
 s6 = "Y0dl = %d" + newline;  
 s8 = "αdl = %d" + newline;  
 s7 = s00 + s0 + s00 + s1 + s2 + s3 + s4 + s5 + s6 + s8 + s00;  
  
 case 'PolynomialTestPlain'  
 fun = @PolynomialTestPlain;  
 activeParams = [1,1]';  
 constraintLimits = [ ...  
 1.0e-9, 1.0e3; ...  
 1.0e-9, 1.0e12; ...  
 ];  
  
 end

## Instantiation of simplexFit class

An instance of the simplexFit class is created with the fit function and constraint limits passed to the constructor. Some local variables are then defined and then a check is made to determine if the parallelization toolbox is installed. If it is available, a parallel for loop is used. If not, a sequential for loop is used. In both cases, the simplex fit algorithm is called multiple times because the initial simplex is generated using the initial guess and some randomly generated points near the initial guess. The fit parameters that return with the lowest mean-square error are output and used to estimate the fit impedance that is plotted.

fitClass = simplexFit(fun,constraintLimits);  
 fun2 = @fitClass.fitFn;  
 check = ver('parallel');  
  
 if ~isempty(check)  
 numIters = 10\*numel(activeParams);  
 bFits = zeros(numIters,length(beta0));  
 mseVals = zeros(numIters,1);  
 parfor iter = 1:numIters  
 % ========================  
 % Fitting routine called  
 [mdlEIS1,mseVals(iter,1),~] = fun2([T.Freq,T.Zreal,T.Zimag],beta0,activeParams);  
 % ========================  
 bFits(iter,:) = mdlEIS1.coefficients(:);  
 end  
 else  
 numIters = numel(activeParams) + 1;  
 bFits = zeros(numIters,length(beta0));  
 mseVals = zeros(numIters,1);  
 % ========================  
 for iter = 1:numIters  
 % ========================  
 % Fitting routine called  
 [mdlEIS1,mseVals(iter,1),~] = fun2([T.Freq,T.Zreal,T.Zimag],beta0,activeParams);  
 % ========================  
 bFits(iter,:) = mdlEIS1.coefficients(:);  
 end  
 end  
  
 [~,iMSE] = min(mseVals);  
 beta1 = bFits(iMSE,:);  
 fprintf(s7,beta1);  
 [zMod1,~,~,zPhase1,~,~] = fun(beta1,T.Freq);  
 [zMod0,~,~,zPhase0,~,~] = fun(beta0,T.Freq);  
 outputString = "Analysis of EIS data completed.";  
 disp(outputString)  
 outputString = newline;  
 disp(outputString)  
 plotBodeEIS([T.Freq,T.Freq],[T.Zmod,zMod1],[T.Zphz,zPhase1],legendForPlotString)  
 % ========================  
 % Write the fit values and data to an output file  
 % ========================  
 Aexp = [T.Freq,T.Zmod,T.Zphz, zMod0, zPhase0, zMod1, zPhase1];  
 oN = strcat(legendForPlotString,'.csv');  
 outputName = fullfile(fitDirectory, oN);  
 writematrix(Aexp, outputName)  
end  
%------------- END OF CODE --------------

[*Published with MATLAB® R2024a*](https://www.mathworks.com/products/matlab)

1. Contact info: [steven.a.policastro.civ@us.navy.mil](mailto:steven.a.policastro.civ@us.navy.mil) [↑](#footnote-ref-1)