Manual for Elchemea Analytical

Søren Koch

January 22, 2016

Contents

Chapter 1

Introduction to Elchemea Analytical

Elchemea Analytical is a generalized visualization / fitting software package for visualizing impedance data. The Elchemea Analytical system is based on Perl and Apache and all graphics/fitting is done using Gnuplot®.

The main features of Elchemea Analytical are listed below:

- Simulation of impedance spectra using a wide variety of discrete impedance elements (R,C,L,Q,W,H,RQ,RC,O,Od,G,Gd,T,Td) as well as parallel and/or series connections of those.
- Fitting of impedance models to measured impedance data in a wide frequency range (from 10^{-100} to 10^{100} Hz).
- Easy determination of start parameters for a wide range of the predefined impedance elements (RC,RQ,O,Od,G,Gd,T,Td) using the 'find-values' build in algorithms.
- Easy integration to Elchemea© and Risoe Fuel cells and solid stage chemistry division Fuel Cell Control system© impedance acquisition and test system control software packages.
- Uses only open source software (OSS).

The first part of this documentation is an overview of the user interface (section 3) mainly intended for new users of the system. The second part (chapter 6) is mainly intended for more advanced users and system administrators as it contains information regarding configuration. It is assumed that any administrators has a fairly advanced knowledge of Unix system administration and Perl programming.

Chapter 10 contains the documentation for the different Perl module supplied by the DTU Energy at Technical University of Denmark Impedance visualisation and analysis control software software.

Chapter 2

License

Chapter 3

User interface

The DTU Energy at Technical University of Denmark Impedance visualisation and analysis control software system is based on the Apache web server software (Open Source Software, OSS). In figure 3.1 the main page is shown as an example of the web pages.

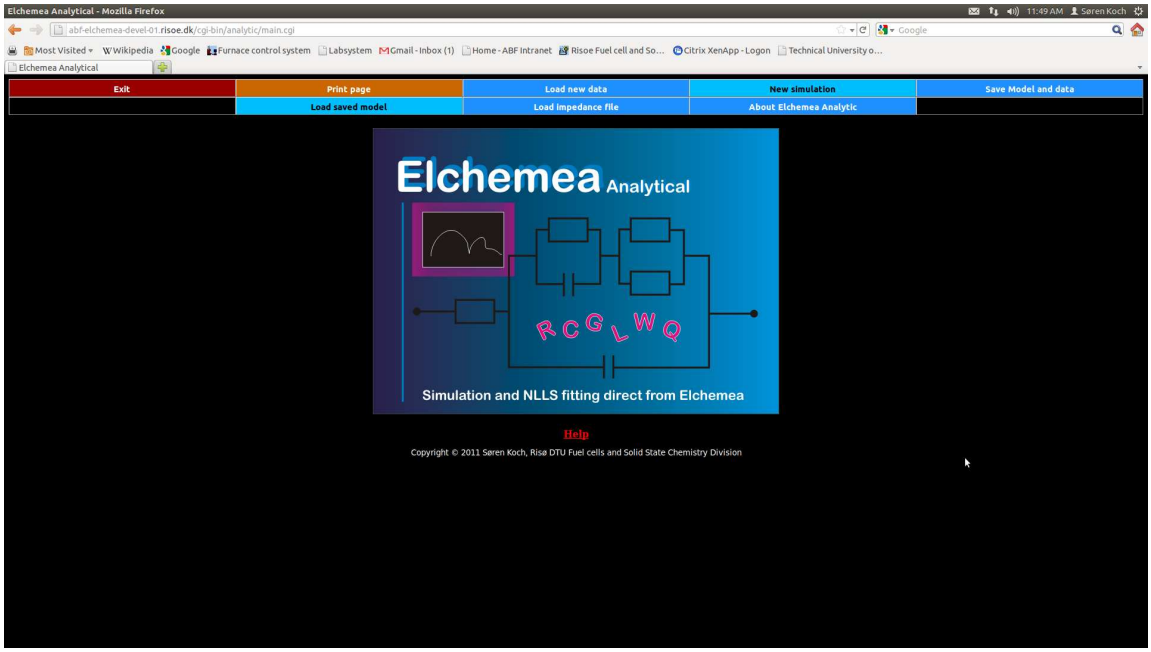

Figure 3.1: The main page.

3.1 Simulation

In order to simulate spectra, press the 'New simulation' button, and then build your impedance model element by element by using the 'Add element' button. In general, all elements will be placed in series, except if they are part of an explicit parallel or serial connection (by using the 'Par' or 'Ser' elements at the bottom of the element list). For a description of the different possible impedance elements, refer chapter 9.

It is possible to view impedance models and data in the impedance plane (which is the

default), the admittance plane, the complex modulus plane and the complex capacitance plane. To change between plotting planes, pres the 'Advanced plot options' button and select the desired plotting plane.

Figure 3.2 shows an example of a simple series connection of three impedance elements plotted in the impedance plane. A Gerisher element (G) a RC element (parallel connection of a resistor and a capacitor) and an inductor (L) viewed i the impedance plane.

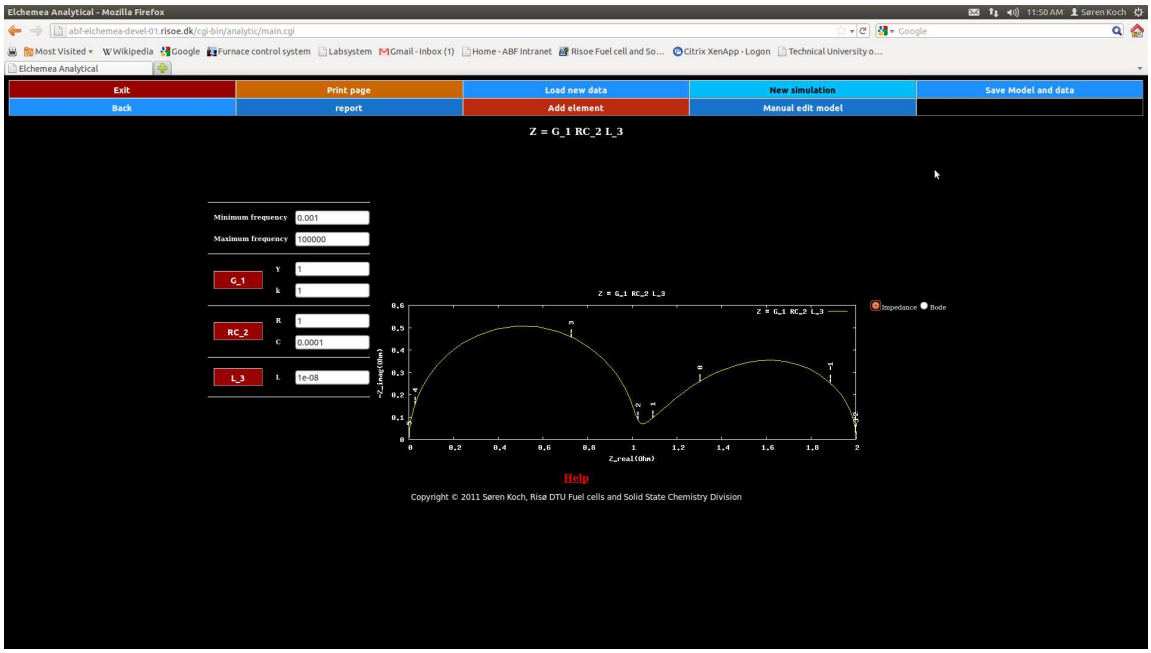

Figure 3.2: An example impedance simulation using simple series connection of three impedance elements.

It is also possible to view impedance models and data in the admittance plane ($Y = Z^{-1}$), complex modulus plane ($M = j\omega Z$) and in the complex capacitance plane ($C^* = \frac{Y}{j\omega}$), this can be changed in in the advanced setup. Default is to view in the impedance plane.

If custom series and / or parallel connection elements are used, the model description on the left of the page looks slightly different than the one shown in figure 3.2. Figure 3.3 shows an example of how this may look. The way to read the complex models presented in this way is as follows:

The first two red buttons (labeled 'L_1' and 'R_2') are simple impedance elements connected in series (as normal) and in series with the third element represented by the leftmost blue button (labeled 'Parallel ([L_5 R_6] C_8'). This button (labeled 'Parallel ([L_5 R_6] C_8') indicates a parallel connection of all elements between the lines above and below this button. In this case it is a parallel connection of two elements; an capacitor ('C_8') and a series connection (labeled 'Serial [L_5 R_6]'). The series connection (represented by the second blue button) contains all elements to the right of it which is between the white lines above and below the blue button (in this case 'L_5' and 'R_6').

Thus the complete model is Z = L_1 + R_2 + Par(Ser(L_5 + R_6), C_8).

In this way it is possible to build arbitrary complex impedance models, either for simulation purposes or for fitting to actual measured spectra.

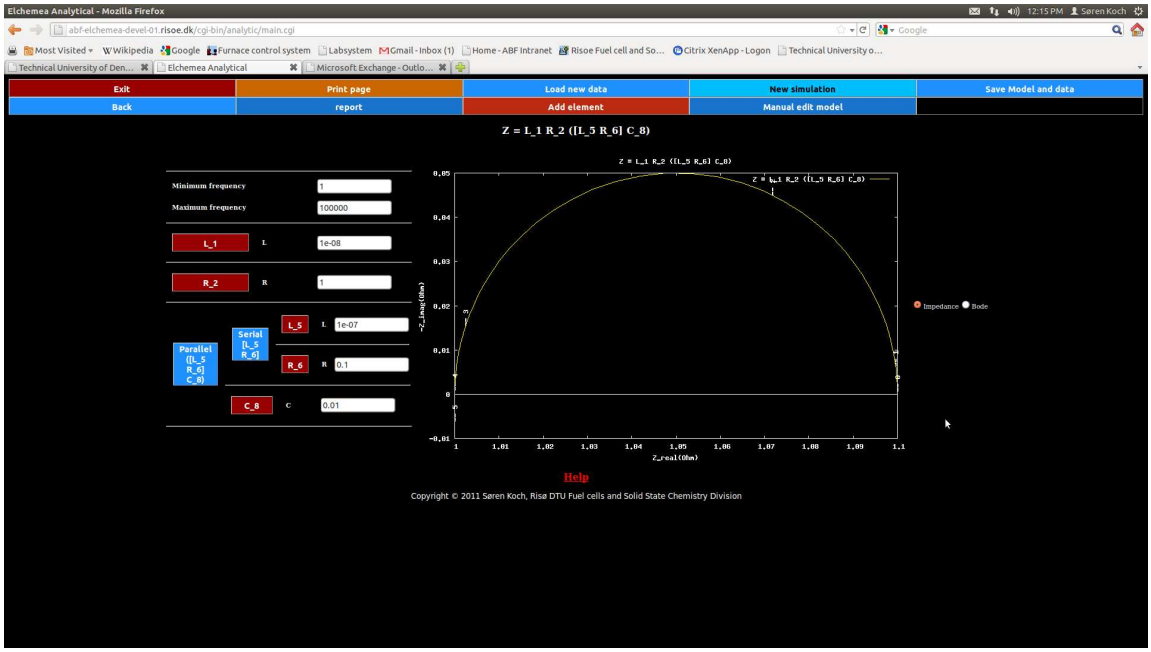

Figure 3.3: An example impedance simulation using parallel and serial connections of discrete elements.

If a model like the one in figure 3.3 is to be created, start a new simulation and follow the steps below:

1. Press 'Add element' and select 'L'.
2. Press 'Add element (in the pop-up window)'.
3. Press 'Add element' and select 'R'.
4. Press 'Add element (in the pop-up window)'.
5. Press 'Add element' and select 'Par' (a new window will open).
6. Press red 'Add element' button and select 'Ser' (a new window will open).
7. Press red 'Add element' button and select 'L' (in the pop-up window).
8. Change value to 1e-8 and press blue 'Add element' button
9. Press 'Add element' and select 'R' (in the pop-up window).
10. Press 'Add element' (in the pop-up window).
11. Press 'Close serial' (serial container window will close).
12. Press 'Add element' and select 'C'.
13. Change value to 0.01 and press 'Add element'.
14. Press 'Close parallel' (parallel container window will close).

15. The resulting model should now be identical to the one in figure 3.3.

From the above it can be seen that pressing the 'close' buttons effectively acts as an end parenthesis for the container element in question.

If an element is to be deleted, simple press the red button designating the element in question. If deleting elements results in an empty container (parallel or serial), then the container itself is also deleted.

It is also possible to manually edit the impedance model. To do this press the 'Manual edit model' button, this brings up a small editor where the model is displayed. The model consists of a number of lines, each designating a single element (an exception to this is the container elements, which takes up 2 lines). The container elements start with a line with a single bracket. Angled brackets '[' for series connections and normal brackets '(' for parallel connections. This is similar to the nomenclature used in the old DOS program 'equivcrt' by Dr. Bernard A. Boukamp. The corresponding line with the end bracket denotes the end of the container element. *Note that containers may contain containers!* Also note that if you are manually editing models, keep track of the brackets, as misaligned brackets may invalidate the model and / or result in an unwanted model!

3.2 Data analysis

In order to analyze measured impedance spectra, use one of the load options on the main page (figure 3.1). Figure 3.4 shows how an freshly loaded data file may look. If no prior model has been defined and the data is viewed in the impedance plane, the Elchemea Analytical generates a simple impedance model to start on if the 'Add serial R and L' option is set to 'Yes' on loading the data file. This consists of an inductor in series with a resistor and the values are determined from the data points in the highest frequency decade. The real value of the data point corresponding to the highest frequency determines the resistance and the inductance is determined by doing a Kramers Kröning transformation on the data points in the highest frequency decade (Algorithm and program supplied by Christopher Graves). Notice however that in case the Kramers Kröning method fails the imaginary value of the first data point (highest frequency) is used instead. This is to ensure that a value is always obtained, and this is known to happen for some combinations of Python and SE-Linux.

By using the radio button and dropdown menus on the right it is possible to switch between Nyquist view and Bode view, set the fitting weight method, background color and if the plot is to show sub-arcs or not (Notice that the last option only has effect if plotting in the impedance plane).

If some data points are to be excluded from the analysis, click on the data graph and then maneuver the pointer to the data point which is to be deleted and press the 'Delete' button. Repeat this procedure until the data set has been sanitized from any erroneous data.

Caution: Do not delete data points merely because they do not fit the chosen model!

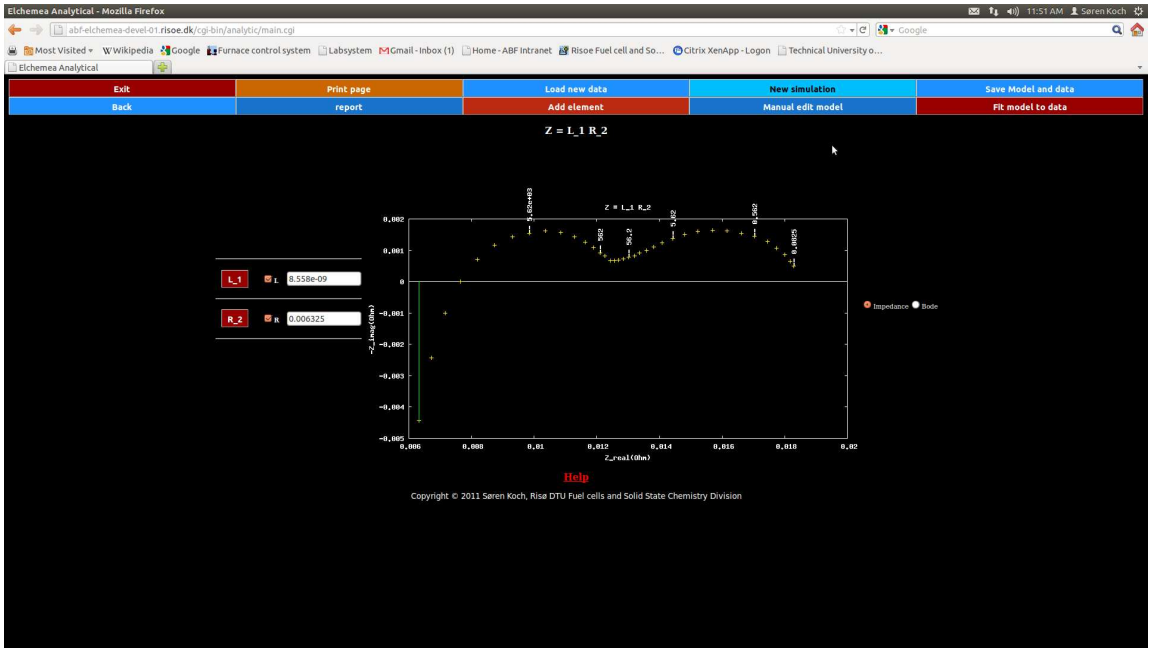

Figure 3.4: Example of an freshly loaded impedance spectrum.

Only delete data points if they are known to be problematic (a single point lying far away from the rest is a good candidate, but not a sure sign)!

A special feature of the Elchemea Analytical is the ability to determine acceptable start values for parameters for some of the predefined complex impedance elements (RC, RQ, O, G and T). To use this, select the appropriate element to add, and then press the 'Find values' button, which will bring up an additional window like the one shown in figure 3.5.

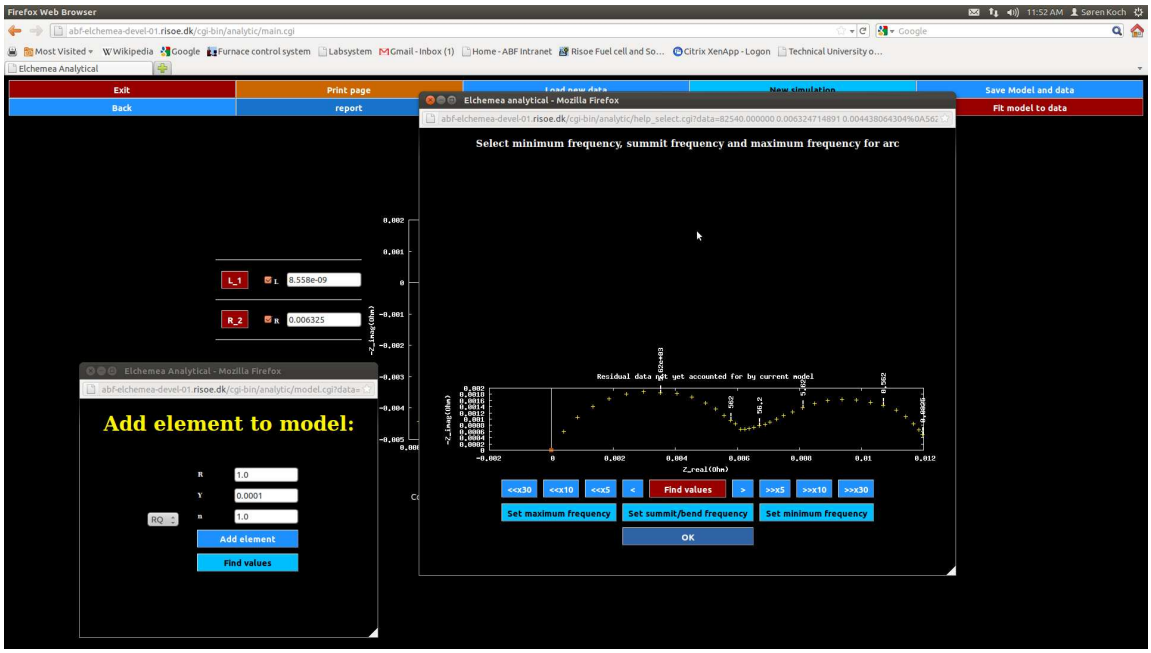

Figure 3.5: Adding an impedance element to a model using the 'find-values' system.

Use the arrow buttons to move the frequency marker to the desired data points and select

the right data points using the three blue buttons below ('Set maximum frequency', 'Set summit/bend frequency' and 'Set minimum frequency'). Once these three frequencies has been selected (which can be seen on the display as changes in the color and symbol of the data points), press the red 'Find values' button. This will cause the Elchemea Analytical system to try and find sensible start parameters for the selected data range for the element in question, and in case a fit converges the resulting impedance model will be shown as seen in figure 3.6. If you are satisfied, simply press the 'OK' button (which causes the find values window to close) and then the 'Add element' button, which will add the chosen element with the determined start parameters to the current model. It is possible to zoom using the X-range fields at the bottom (for instance if a small arc sits next to a large one, the small one may be extremely hard to determine). Simply change the minimum or maximum impedance to be displayed and the plot/select area will be updated accordingly.

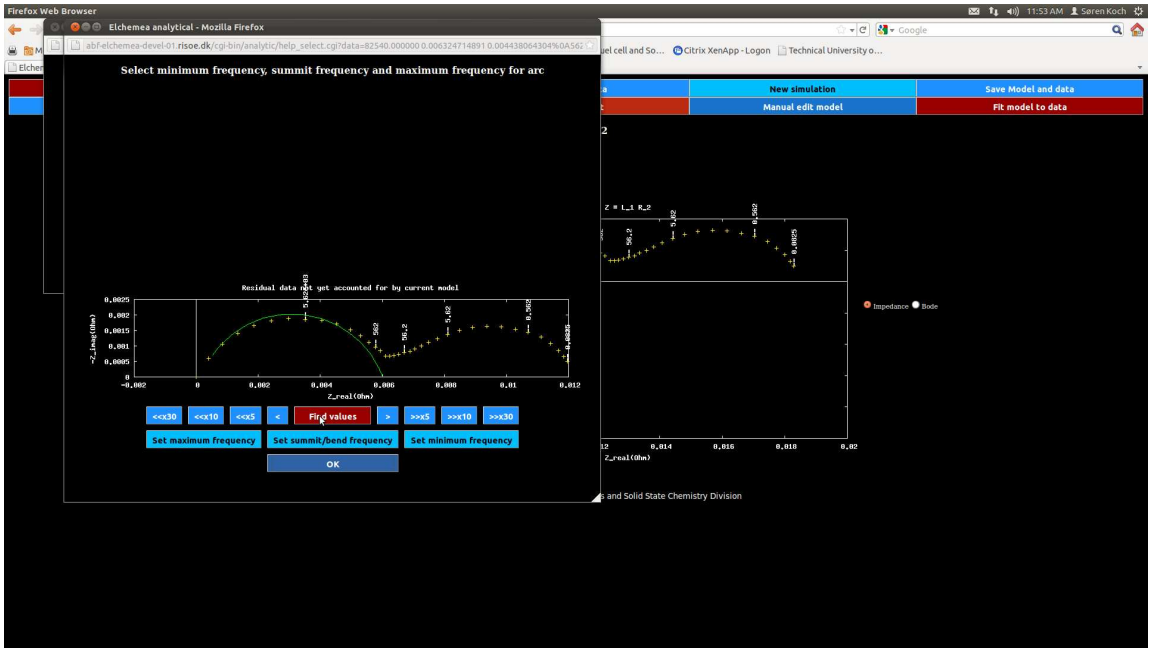

Figure 3.6: Finding the values for an RQ element using 'find-values' on a measured impedance spectrum.

Once a suitable model has been build, fit the model to the measured data by pressing the 'Fit model to data' button. This will cause the Elchemea Analytical to try and fit the model to the data, and in case the fit converges, the resultant parameter values will be displayed and the system will ask if the values should be copied to the model. If the resulting values are sensible, press 'Yes' and the resulting fit parameters will be displayed in the model section as well as appended to the fit result table. This table contains a line for each accepted fit and includes all model parameters as well as the mean and maximum error for the fit in question. Thus if more than one file is fitted (or a single data set is fitted to more than one model), it is possible to get a list of all the fitted data afterwords for later comparison / further data analysis. To access the fit result table, press the 'Show fit table' below the graph. This brings up a new window with a tab delimited table which can be copied / saved to a local file as appropriate. Figure 3.7 shows an example of a fit result.

Figure 3.7: Example result of a fit of a model to an actually measured impedance spectrum.

It is possible to exclude some parameters from the fitting (in effect locking them to the current value) by unclicking the check box associated with the parameter in question. This is useful if external information indicates that a specific impedance parameter must have a specific value. After the fit has been run, press the 'Report' button to get a pdf report of the fit in question. This report will include parameter values, statistics regarding accuracy of the fit, parameter correlation as well as Nyquist, Bode and error plots of the model and data.

It is known that some versions of AdobeReader® in some cases has problems displaying the pdf documents created by Elchemea Analytical, however the fast open source pdf viewer Ghosview® (http://pages.cs.wisc.edu/~ghost/) is available for free for a wide range of operating systems.

3.3 Saving and reloading models

Once a model has been created it is possible to save the model (and any associated data). To do so, press the 'Save model and data' button, which will open a save dialog asking where to place the save data.

In order to load a saved data set/model, press the 'Load saved model' button on the main page and select the data file to load (The default extension will be .ea).

It is also possible to save just the model, if this is to be desired, simple open the 'manual edit model' page and copy the model to the clipboard and then save it in a local text file. To restore a model thus saved, open a new simulation or data file and press the 'manual edit model' button and then paste the saved model instead of the present model. Finish

by pressing 'OK'.

3.4 Hints on fitting

Fitting impedance data is as much an art as it is science. The most important part of data modeling / fitting is to select the most appropriate model, and in this respect Occam's Razor is an extremely important tool! Do not select a more complex model than necessary based on the available information as it is always possible to get a good fit if enough impedance elements are added to the model.

Due to the mathematics in some of the impedance elements distributed with Elchemea Analytical, sometimes the fitting does not terminate or crash (indicated by either an error message or simply that no response is received from the fitting attempt). Especially the Gerisher, Finite length warburg and bounded warburg elements are prone to this as they include one or more of the hyperbolic trigonometric functions, and in some cases the fitting algorithm will select values which results in a singularity (division by zero or infinite) and the fitting algorithm fails. When this happens, try to slightly change the start parameters and see if it is possible to avoid the singularity.

3.5 Batchfitting

It is possible to do batch process fitting without using the GUI. To do this use the command line program *batchfit.pl* (usually located in */usr/local/bin/analytic/*). If the program is called without any arguments a description of how to use it is output. In the batchtest directory found in the Elchemea Analytic distribution directory there is an example of how to use it. Simply run the *run_batchfit_test.bash* program to test with the files in that directory.

Chapter 4

Plotting multiple files

It is also possible to plot multiple impedance file in the same plot by using the multiplot module of Elchemea Analytical. To access this module, navigate to the front page and press the 'Plot multiple files' button, which brings the multiplot page up as shown in figure 4.1.

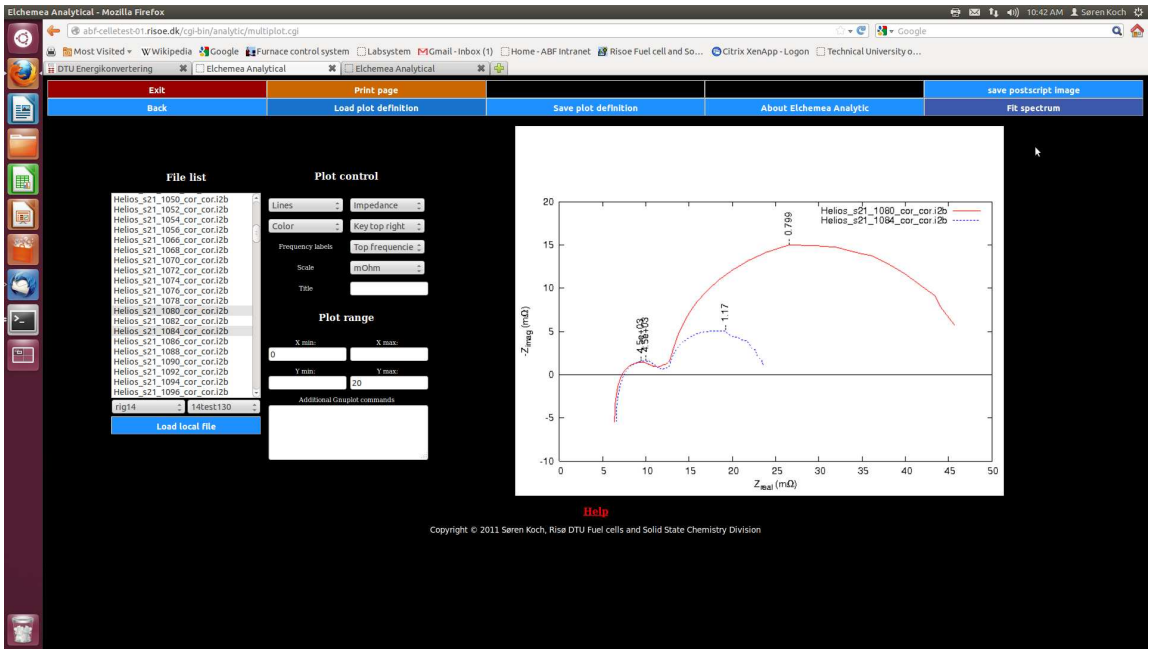

Figure 4.1: Multiplot page showing an example of 2 impedance spectra plotted on the same graph.

The left part is the file control section. It contains a multi select box showing both the available and selected files. To plot files load the impedance files from your local system by pressing the 'load local file' or select user name, session number and files (in case the Elchemea© or RFCcontroll© software is installed on the server along with Elchemea Analytic). Note that the user/rig name and test/session select boxes are only available as indicated above. If files are loaded from a local resource, it is automatically selected for plot. To deselect/deselect a file press 'Ctrl' and click on the file to select/deselect. If the order of the legend keys are to be changed, move the file names up and down in the

file list as appropriate as Elchemea Analytics always uses the files/legends in the order they are shown in the file list. To move a file, select it and use the 'u' and 'd' keys to move the file up and down in the list (Note that only one file can be moved at a time).

If the legend for a particular file is to be changed, simply double click on the file to specify a new legend for that file.

The center part of the multiplot page is the plot control area, where placement of legend, frequency labels, plot ranges etc. can be controlled. If 'User defined' is selected for frequency labels, an additional field becomes visible where the frequencies to show on the plot can be typed in. For each frequency in this list (separated by commas) the first data point in each file below the frequency is indicated along with a label showing the frequency. The text field at the bottom is for additional Gnuplot commands if additional labels, arrows or similar is to be included in the graph (refer the Gnuplot manual for information on gnuplot commands).

The resulting graph shown on the right part of the page can simply be copied or saved (right click on the graph) and if an postscript file is preferred (for use with Latex documents for instance) press the 'Save postscript image' button.

It is possible to save the data and plot definitions for the current work by pressing the 'save plot definition' button. The resulting file can then be reloaded at a later time for further work. It is important to note, that as Elchemea Analytical is a multi user system, it includes an automatic clean up facility which removes files (uploaded data files and generated image files) after one hour of inactivity. Thus if the user expects to take a break from the Elchemea Analytical system, remember to use the 'save plot definition' beforehand so as to not loose any work.

If the scaling of the data is not as intended, the scaling of the data can be changed by changing the 'z(x)' function definition (default is z(x) = x). To do so, simply write the new definition in the 'additional gnuplot commands' field. For instance if the sample has a 6 cm^2 area and the graph should report the impedance in area specific resistance, write the following in the text area:

```
z(x) = x * 6
set ylabel '-Z_{imag} ({/Symbol W} cm^2)'
set xlabel 'Z_{real} ({/Symbol W} cm^2)'
```

The first line changes the scaling and the two following lines change the axis descriptions to include cm^2 (leave them out if only the scaling are to be changed). Notice however that you likely can not simply use copy and paste as the ' in the above text will be represented as a utf-8 character (type manually instead)!

The above lines can also be used in the fitting part of Elchemea Analytical, however as this part uses the png output device (in order tom be able ot plot on black background), some of the text may look different from what is expected (specifically '{/Symbol W}' is interpreted entirely as text).

Chapter 5

File formats

DTU Energy at Technical University of Denmark Impedance visualisation and analysis control software supports a number of different file formats including gamry(R) and ZPlot(R) / ZView(tm) files.

The native file format is the i2b file format however. The file format is described below with an example file:

```
Idun:/home/EFA/rig14/1255/4/s4_1007.i2b
Desc: 14test78_PS128052
:2009:01:17:23:17:30   1232230650  , Meas.:



37
82451 0.006685137356 0.00141940337
56173 0.0070404900979 0.0002431376363
38270 0.007484563409 -0.000686494133
26073 0.008028417169 -0.001394965951
17764 0.008634536255 -0.001947261525
12102 0.009370725592 -0.002396327231
8245 0.010187221398 -0.002722736196
5617 0.011033464434 -0.002911896725
3827 0.011876708228 -0.003031150569
2607 0.012699248889 -0.002996166924
1776 0.013426438402 -0.002909587586
1210 0.014135420185 -0.00274866887
824.5 0.014798469288 -0.002688357568
561.7 0.015415708783 -0.002545973023
382.7 0.016064610363 -0.002514285078
260.7 0.016722169513 -0.002355489951
177.6 0.017329395388 -0.002180886816
121 0.017877782005 -0.001969379374
82.45 0.018305518738 -0.001746990494
56.17 0.018698563455 -0.00151536534
38.27 0.01889247348 -0.001359174472
```

```
26.07 0.019155759436 -0.00132627701
17.76 0.019422716189 -0.001266016758
12.1 0.019661051995 -0.001295590415
8.245 0.019888830093 -0.001464146169
5.617 0.020308127403 -0.001725404958
3.827 0.020971000603 -0.001804639011
2.607 0.021602216143 -0.001638322228
1.776 0.02195975717 -0.00134448105
1.21 0.022350984057 -0.001060789257
0.8245 0.022530834712 -0.000697843738
0.5617 0.022641762282 -0.0004145079483
0.3827 0.0226543601828 -0.0003907922012
0.2607 0.0227450009778 -0.0003855313026
0.1776 0.0228211856604 -0.0001882806252
0.121 0.0229167510741 -0.0001616599266
0.0825 0.0229392097983 -0.0002138122947
```

The first 6 lines can contain text information (including meta information). Usually the third line contains a time stamp, but DTU Energy at Technical University of Denmark Impedance visualisation and analysis control software does not use this.

The 7'th line contains an integer describing the number of data points, and the following lines (nr 8 and onwards) is the actual data. Each data point is in the format:

Frequency Real_part Imaginary_part

Chapter 6

Installation and system maintenance

This chapter describes how to install or upgrade a Elchemea Analytical system.

6.1 Requirements

The Elchemea Analytical requires the following software to be installed on the target system prior to installation:

- A Linux type operating system (Only tested with CentOS® 5 and 6, but will likely work on other Linux type systems as well).
- Gnuplot® version 4.0 or later.
- The Apache® web server version 2.2.3-43 or later (it is possible that earlier versions of Apache will also work, however this is not tested).
- The Perl® interpreter version 5.8.8 or later.
- The Perl Time::Hires module (installation of this varies between distributions, thus needs to be installed manually, refer your distribution manual as to how).
- ImageMagick version 6.2.8 05/07/12 Q16 or later.
- GPL Ghostscript version 8.70 (2009-07-31) or later.
- Gnu 'make'. Other 'make' packages than the one from Gnu may also work, but has not been tested.
- A functioning connection to the Internet. The reason for this is that Elchemea Analytical downloads and installs several Perl modules from CPAN.org during the installation.

6.2 Installation

In order to install the Elchemea Analytical system, unpack the tar-ball in a suitable location, cd into the resulting ElchemeaAnalytical directory and run *make*.

Inspect the output of the make program and resolve any errors.

Once all errors have been resolved, run *make test* followed by *make install*.

In order to ensure that all servers start upon system reboot, add the following line to */etc/rc.local*:

/usr/local/bin/analytic/start_servers &

Finally, start up a web browser and point to this address *hostname.domain/cgi-bin/analytic/main.cgi* (substitute host name and domain with the appropriate values for your system) to check if the system is properly configured, the resulting page should look like figure 3.1;

6.2.1 Command line only installation

It is possible to install Elchemea Analytical as a command line only tool (for instance on workstations without a web server). In order to do this follow the above steps except instead of running *make install*, one should run *make install_cmdonly* and omit *start_servers* part as this is only needed for the server installation.

The command line only mode is useful for workstation use if multiple impedance files are to be fitted using the same model. In this case it may be too tedious to manually upload each file to a server and fit and much simpler to use the 'batchfit.pl' program supplied with Elchemea Analytical in both server and command line only mode.

6.2.2 SE-Linux

If SE-Linux is installed on the serer and running in enforcing mode, the default configuration of SE-Linux will prevent the Apache web server from accessing the LATEX server listening on port 4050 as well as the execution of various scripts in /usr/local/bin/analytic.

To allow Apache to connect to port 4050 as well as execute the scripts in /usr/local/bin/analytic/, (on CentOS) execute the script *set_SELinux_rules.bash* found in the Elchame Analytical installation directory.

6.3 maintenance

Generally the Elchemea Analytical system requires little maintenance and the Elchemea Analytical system includes a facility for automatic software updates, to enable this, simply add the following line to root's crontab file:

*0 8 * * 1 /usr/local/bin/analytic/analytic_updateer.pl ≫ /root/update_log.txt &*

This will update the system once every Monday. The automatic update system then fetches any new version which may have been deployed within the last week and installs this if it passes the software test (*make test*).

Chapter 7

Server structure

The programs mentioned in *italics* below all reside in the */usr/local/bin/analytic* and are written in Perl.

7.1 LATEX-server

The Latex server (LATEX-server) is responsible for compiling Latex reports of the fit results of individual fits. It honors the following commands:

- debug: Turns debug on and off.
- exit: Shuts down the server cleanly.
- unlink: Unlink the specified file name. Note this is a potential security risk, so the Latex server should not be accessible from external sources!
- compile: compiles the document with the specified file name. The resulting pdf file is placed in *'/home/http/html/analytic/png/'*.

Chapter 8

System command interface (command line)

Although the Elchemea Analytical is designed to be used primarily through the web interface some programs can be accessed from the command line. Below is a list of the most used command line tools for the Elchemea Analytical system:

- */usr/local/bin/analytic/z_to_i2b*: This program converts Z-plot files to the 'i2b' file format.
- */usr/local/bin/analytic/gamry_to_i2b*: This program converts a Gamry® file to the 'i2b' format.
- */usr/lcoal/bin/analytic/batchfit.pl*: This program can be used to fit multiple spectra to the same model. Call the program without any arguments to get a description of how to use it as well as a list of possible options.

Chapter 9

Impedance elements

Elchemea Analytical supports a number of discrete impedance elements. The following sections describe each element. In all the following sections, ω denotes the angular frequency ($\omega = 2\pi f$), j the entity $\sqrt{-1}$ and Z the complex impedance.

9.1 Ohmic resistor (R)

The most simple impedance elements is the ohmic resistor. The impedance of this element is does not have an imaginary component, and is simply the resistance R:

$$Z = R \tag{9.1}$$

It is not possible to use the 'find values' function on this element.

9.2 Inductor (L)

The inductive element L has no real component in the impedance and the impedance of the C element can be described as:

$$Z = j\omega L \tag{9.2}$$

It is not possible to use the 'find values' function on this element.

9.3 Capacitor (C)

The capacitive element C does also not have a real part and is:

$$Z = \frac{1}{j\omega C} \tag{9.3}$$

It is not possible to use the 'find values' function on this element.

9.4 Constant phase element (Q)

The constant phase element Q has both a real and an imaginary part and can be expressed as:

$$Z = \frac{1}{Y(j\omega)^n} \tag{9.4}$$

It is not possible to use the 'find values' function on this element.

9.5 Warburg impedance (W)

The classical warburg diffusion element is infinite and the impedance of this element is:

$$Z = \frac{\sigma}{\sqrt{j\omega}} \tag{9.5}$$

where σ is the warburg coefficient.

It is not possible to use the 'find values' function on this element.

9.6 Havriliak-Negami relaxation (H)

Havriliak-Negami relaxation is an empirical modification of the Debye relaxation model, accounting for the asymmetry and broadness of the dielectric dispersion curve.

The impedance of the element is:

$$Z = \frac{(1 + (j\omega\tau)^\alpha)^\beta}{j\omega\Delta C} \tag{9.6}$$

where τ is the characteristic time constant, ΔC is the difference in capacitance, α is the assumetry exponent and β is the broadness exponent.

For $\beta = 1$ the Havriliak-Negami equation reduces to the ColeCole equation, for $\alpha = 1$ to the Cole-Davidson equation.

It is not possible to use the 'find values' function on this element.

9.7 Finite length warburg impedance (O) and depressed / flattend finite length warburg impedance (Od)

The finite length warburg impedance (also sometimes called finite diffusion impedance with transmissive boundry conditions) behaves as the Warburg element at high frequency,

but at low frequency behaves more like an RC circuit (refer section 9.11). The impedance of this element is normally defined as:

$$Z = \frac{\tanh\left((Bj\omega)^n\right)}{(Yj\omega)^n} \tag{9.7}$$

However in Elchemea Analytical the following transformation is used:

$$Z = \frac{\tanh\left((Bj\omega)^n\right)}{(Yj\omega)^n} \tag{9.8}$$

$$Z = \frac{B^n \tanh\left((Bj\omega)^n\right)}{Y^n (Bj\omega)^n} \tag{9.9}$$

$$Z = \left(\frac{B}{Y}\right)^n \frac{\tanh\left((Bj\omega)^n\right)}{(Bj\omega)^n} \tag{9.10}$$

$$Z = Z_0 \frac{\tanh\left((Bj\omega)^n\right)}{(Bj\omega)^n} \tag{9.11}$$

$$\tag{9.12}$$

where

$$Z_o = \left(\frac{B}{Y}\right)^n \tag{9.13}$$

For the normal finite length warburg element (O) n is fixed at $\frac{1}{2}$ whereas it is below this for the depressed version (Od).

The 'find values' function can be used on O and Od elements.

9.8 Finite capacity warburg impedance (T) and depressed / flattned finite capacity warburg impedance (Td)

The finite capacity warburg diffusion element (also sometimes called finite diffusion impedance with blocking boundry conditions) behaves like the classical Warburg at high frequency (as the O element does) but as the frequency gets lower the imaginary part of the impedance goes asymptotically towards infinite. The impedance or the T element is:

$$Z = \frac{\coth\left((Bj\omega)^n\right)}{(Yj\omega)^n} \tag{9.14}$$

where coth is the hyperbolic co-tangent function. However in Elchemea Analytical the following transformation is used:

$$Z = \frac{\coth\left((Bj\omega)^n\right)}{(Yj\omega)^n} \tag{9.15}$$

$$Z = \frac{B^n \coth\left((Bj\omega)^n\right)}{Y^n (Bj\omega)^n} \tag{9.16}$$

$$Z = \left(\frac{B}{Y}\right)^n \frac{\coth\left((Bj\omega)^n\right)}{(Bj\omega)^n} \tag{9.17}$$

$$Z = Z_0 \frac{\coth\left((Bj\omega)^n\right)}{(Bj\omega)^n} \tag{9.18}$$

$$\tag{9.19}$$

where

$$Z_o = \left(\frac{B}{Y}\right)^n \tag{9.20}$$

For the normal finite capacity warburg element (T) n is fixed at $\frac{1}{2}$ whereas it is below this for the depressed version (Od). Also note that for the depressed version the asymptote is not vertical when viewed in the complex plane.

The 'find values' function can be used on T and Td elements.

9.9 Gerisher impedance (G) and depressed / flattend gerisher impedance (Gd)

The Gerisher and depressed Gerisher element has the same overall features as the O element in that they behave like the Warburg element at high frequencies and like an RC circuit at low frequencies, however the mathematical desorption is different than that for the O element as described below:

$$Z = \frac{1}{Y(k + j\omega)^n} \tag{9.21}$$

For the normal Gerisher (G) element n is fixed at $\frac{1}{2}$ whereas it is below this for the depressed gerisher (Gd).

The 'find values' function can be used on G and Gd elements.

9.10 De Levie impedance (dL)

The de Levie impedance element is the impedance of a porous electrode in a solution with active redox species [1]. The impedance of this element is defined as:

$$Z = \sqrt{R_i R_r} \cdot \frac{\coth\left(d\sqrt{\frac{R_i}{R_r}}\sqrt{1 + Y\left(j\omega\right)^n}\right)}{\sqrt{1 + Y\left(j\omega\right)^n}} \tag{9.22}$$

where coth is the hyperbolic co-tangent function, d is the electrode thckness, R_i is the resistance unit length of the ionic conductor (electrolyte), R_r is the Faradaic reaction resistanc unit length and Y and n are the pseudo capacitance (in FS^{n-1}/cm) and phase

[1] Chapter 9.1, Electrochemical Impedance Spectroscopy and its Applications by A. Lasia, Springer New York (2014)

angle (in 1/2 pi radians) of the constant phase element describing the double layer capacitance.

It is not possible to use the 'find values' function on this element.

9.11 Parallel R-C circuit (RC)

The RC circuit is a parallel connection of a normal resistor and a capacitor. The complex impedance of this circuit is

$$Z = \frac{1}{R^{-1} + Cj\omega} \tag{9.23}$$

The reason for using the RC element as a separate element and not using the Par element to 'build' one is that by utilizing the analytical description of the RC circuit, it is possible to use the 'find values' function to obtain good start guesses for the parameters. which would not be possible for a custom designed 'Par' elements with a R and a C element (refer section 9.15).

9.12 Parallel R-Q circuit (RQ)

Similar to the RC element the RQ element is a parallel connection of a resistor and a capacitive element, but in this case the constant phase element. The impedance of the RQ elements is:

$$Z = \frac{1}{R^{-1} + Y(j\omega)^n} \tag{9.24}$$

Similarly to the RC circuit it is possible to use the 'find values' function on the RQ element in order to obtain good start guesses for the parameters as opposed to a custom Par element with a R and a Q element.

9.13 Parallel R-L circuit (RL)

The RL circuit is a parallel connection of a normal resistor and an inductor. The complex impedance of this circuit is

$$Z = \frac{1}{R^{-1} + \frac{1}{Lj\omega}} \tag{9.25}$$

The reason for using the RL element as a separate element and not using the Par element to 'build' one is that by utilizing the analytical description of the RL circuit, it is possible to use the 'find values' function to obtain good start guesses for the parameters. which would not be possible for a custom designed 'Par' elements with a R and a L element (refer section 9.15).

9.14 Serial connection of elements (Ser)

This element is merely a container element which consists of a number of serially connected impedance elements and the combined impedance is:

$$Z = Z_1 + Z_2 + Z_3 + \ldots + Z_n \tag{9.26}$$

Due to the fact that the elements within the container may be any impedance element (**including other container elements!**) it is impossible to use the find values function on a 'Ser' element.

9.15 parallel connection of elements (Par)

As with the Ser element, this element is a container but instead of serially connected impedance elements it is a number of parallel connected impedance elements, thus the impedance of the element is:

$$Z = \frac{1}{\frac{1}{Z_1} + \frac{1}{Z_2} + \frac{1}{Z_3} + \ldots + \frac{1}{Z_n}} \tag{9.27}$$

Due to the fact that the elements within the container may be any impedance element (**including other container elements!**) it is impossible to use the find values function on a 'Par' element.

Chapter 10

Module specifications

This chapter contains the module specification for the Perl modules supplied as part of the ElchemeaAnalytic software suite. It includes function descriptions including number and type of any function arguments. Some of the modules are object oriented (with only a publicly accessible constructor) and in other cases the modules are function orientated.

In the case of function orientated modules, any functions exported by the module are described, both for what it does, as well as number and types of arguments.

In the case of the object oriented modules, any inheritance is also described (usually in the beginning of the module description). For the object instances, usually only the member functions intended to be public is described (as Perl does not have a true private function declaration). Note that some of the object orientated modules define more than one class type, but as all the class types in this case behave similarly (polymorphic), only the main class is described as the subsequent class definitions implements the main class type behavior.

Each module is described in it's own section.

10.1 Debug

Use: my $id = Debug→new();

This class is intended to be a base class for other classes to derive from so that easy debug functionality can be included.

Utility class for debugging. It contains the following member functions:

$id→debug()	Sets or gets the debug level: level 0 is no debug, level 5 is complete debug including stack backtrace. This class only uses level 0 (no debug), level 1-4 (debug iformation displayed) and 5 , debug info displayed with complete stack backtrace. The levels 1-4 lets other modules define debug levels inbetween the ones used here.
$id→writedebug($,[$])	Writes the string to standard error if debuglevel is 1 or higher. If overide is specified (second argument which is optional), debug level 5 is assumed for this debug.
$id→die($)	Appends stack backtrace to argument string and calls CORE::die
$id→print_setup()	Prints out the complete current setup including all member functions and data fields (uses Class::Inspector).

10.2 SemaforeFile

Inherits from Debug (refer section 10.1).

This package makes file inout/output on multiprocess systems more easy by encapsulating file locking. To define a new semaforefile use the new method:

my $id = SemaforeFile→new($filename,$lockfile);

my $id = SemaforeFile→new($filename);

If the lockfile is not specified, the default (/var/lock/SemaforeFile/SemaforeFile.lock or /tmp/SemaforeFile.lock) is used instead. This form should generally notbe used however, as in some cases /var/lock/SemaforeFile/SemaforeFile.lock can not be used and files in /tmp/ will from time to time be deleted...

The package includes the following simple public methods on semafore files:

$id→readonly()	Returns true if the file is readonly for the current user
$id→exist()	Returns false if the file does not exists;
$id→chmod($)	Sets the file permissions according to CORE::chmod
$id→filename()	Returns the filename of the semafore file

$id→readlines()	Returns the content of the file as an array with one line in each element Note thet it removes any trailing newline from the read lines!
$id→writeline(@)	Writes the arguments to the file (NB: Overwrites file and add a newline to each argument if they do not already have it).
$id→append(@)	Appends the arguments to the file (Also adds newlines if nescesarry).

It is not nescesarry to check for file esistence in readlines as an empty array is returned if the file does note exist Note that the readlines function should only be used on small files as it globs the entire content to memory! For large files, use the more advanced member functions (see below). Also note that trailing newlines are removed from the individual lines. If this are not desired, use the readline() method described below.

The module also includes the following methods for advanced use: Note none of these functions check if the file exist before trying to open! The unsafe versions of open and close does not lock or unlock (assumes the user does this explicitly!)

$id→lock_ex()	Locks file for exclusive use (Read, Write or Anppend)
$id→lock_sh()	Locks file for shared access (Read only)
$id→lock_ex_nb()	Locks file for exclusive use non blocking (Check return status!)
$id→lock_sh_nb()	Locks file for shared access non blocking (Check return status!)
$id→unlock()	Unlocks file
$id→open_read()	Opens the file for reading (locks file shared if not already locked)
$id→open_readback()	Opens the file for reading backwards (locks file shared if not already locked)
$id→open_write()	Opens the file for writing (locks file exclusive if not already locked exclusive)
$id→open_append()	Opens the file for appending (locks file exclusive if not already locked exclusive)
$id→close()	Closes the file and unlocks it
$id→open_read_unsafe()	
$id→open_readback_unsafe()	
$id→open_write_unsafe()	
$id→open_append_unsafe()	
$id→close_unsafe()	
$id→mtime()	Returns the time of modification of the file as reported by File::stat→mtime, returns 0 if the file does not exist.
$id→readline()	Reads and returns the next line from the file, assumes an open file Raises an excpeption (die) if not.
$id→fh()	Returns the underlying file handle for direct IO (Use with care!)

Additionally the $id→debug($) member function (inherited from Debug.pm) can turn debug information on and off $id→debug($level) turns debug on and $id→debug(0) turns debug off ($level is the debug level, 1 - 5) This may be usefull if deadlock is encountered (so that the individual file locking operations can be monitored! If $id→debug() is called without arguments it returns the status (i-e if debug in on 1 is returned else 0.

10.3 SocketClient

This module defines a number of communication functions used for accessing tcp:IP sockets on local and/or remote systems. The functions defined are listed below:

socket_client_raw($$@)	Base function used by all subsequent functions, handles the raw tcp:IP cummunication. Arguemnts: server, port, [additional args to server]. The server can either be a ip-address or a hostname. Any additional arguments gets serialised with tab characters and 2 newlines are appended to the resulting string before transmission.
socket_client($$@)	Same as above, but catches any communication errors in an eval guard.
socket_client_nocr($$@)	Same as above, but do not append any newlines to the transmitted string.
socket_client_raw_nocr($$@)	same as socket_client_raw() but do not append newlines.
serial_client($@)	communicates with a local serial server (which handles hardware communication on the serial port. Arguments: tty, args_to_server. The server is assumed to be the local server (either localhost or the public IP address of the server) and the port number is the tty number added to 202020 (Note wraparound!).
GPIB_client()	Communicates with the GPIB-server. Arguments are passed to the GPIB-server serialised with tab characters using socket_client_nocr(). The server is assumed to be the local server (either localhost or the public IP address of the server) and the port number is 12345.
serial_client_raw($@)	Same as serial_client() but without eval guard.
GPIB_client_raw()	Same as GPIB_client() but without eval guard.

10.4 Impedance::Header

This module specifies global file locations and other global variables used for the Elchemea-Analytical software package.

The module also specifies the colors of the resulting gnuplot figures (this is specified in the $gpheader variable).

10.5 Impedance::IMPCGI

This module contains a number of utility functions for outputting properly formatted html code for user interface generation. Thus it mainly extends the CGI.pm module by Lincoln D. Stein. The module exports these functions in two groups.

The :html group exports these functions:

print_header($[%])	Prints the header information. Arguments: title. Any additional optional arguments (in the form of a hash) will be parsed along to the header() function supplied by CGI.pm. The functionautomatically appends a call to a javascript function logging users out after some time of no actions.
print_end()	Prints the help button and ends the html output with the proper tag.
print_hidden()	Prints a number of hidden fields used to maintain state.
logout()	Printys a logout button.
action($)	Prints a hidden field with an action parameter with the specified value which can be used for program control flow.
EFA_start_html()	A wrapper for CGI::start_html. Any arguments (in the form of a hash) are passed to CGI::start_html. Automatically appends a reference to the javascript source file on the server.
js_back()	Prints the javascript for gping backwards (uses the browser.back() fjavascrpt call).
get_CGI_value($)	Retrieves the value of the specified CGI parameter (supplied by the web browser.
get_CGI_value_clean($)	Same as get_CGI_value, but does pattern match on the retrieved value and only returns the part that matches. The pattern match is [\w\s\.\,]*. This has the benefit of untainting the returned parameter value (For taint checks in perl and web access, refer Lincoln D. Steins book *Official Guide to Programming with CGI.pm*)

The :cgi group of functions include the following:

get_CGI_value($)	See above.
get_CGI_value_clean($)	See above.
action($)	See above
menu_button(@)	Prints a menu button. Arguments: name, value, style. The name will be the CGI parameter name, the value vill be the text on the button and the style is a style class name to use for displaying.
create_menu_field	Prints the html tags to create a menu field.

top_nav_bar_start()	Prints the html tags to start the top navigation bar (table specifications etc.)
top_no_button()	Prints a no action button (goes nowhere) in the top navigation bar.
top_nav_bar_button()	prints a top navigation button. Arguments: File, name, value, style, [optional additional name, value and force triplets]. The file is the cgi-script to be called upon button press, the name,value and style arguments are passed to menu_button() and the additional optional arguments are used to initialise and print hidden html fields in the form of name-value pairs and a force argument (1 for force value, 0 for allow reuse of value).
tab_newrow()	Prints a new row in the top navigation bar.
top_js_return()	Prints a top navigation return button (uses the javascript printed by js_back(), see above)
end_top_bar()	Prints the end of the top navigation bar.

An additional function which is often used (but which is not exported by default) is the format_model($) function which parses a given impedance model and removes any empty serial or parallel connections as well as empty lines. The format_model function also removes any "\r" characters which may have been added by Microsoft based browsers. The function returns the resulting model.

10.6 Impedance::Base

Inherits from Debug (refer section 10.1).

This module defines all member functions which an ElchemeaAnalytical Impedance element must honor. Most of the functions are merely stups intended to be overloaded by the individual class definitions.

The module also includes class defininitions on the basic discrete elements (R, C, L and Q, Q beeing the constant phase element).

The module is intended to be used in conjunction with gnuplot but can be used as is (but no fitting will be possible).

To obtain a element instance call one of the constructors as show below:

$id = Impedance::Base→new($);

$id = Impedance::R→new($);

$id = Impedance::C→new($);

$id = Impedance::L→new($);

$id = Impedance::Q→new($);

All the constructors accepts a single argument which must be the impedance element number. It is advisable to make sure that element id's are unique in order to be able to

distinguish.

All Impedance element instances has the following member functions:

$id→type()	Returns the type of impedance element.
$id→name()	Returns the name of the element (usually the type and the id)
$id→nr()	Returns the element id.
$id→function_name()	Returns the Gnuplot function name of the element.
$id→value($[$])	Sets or gets the value of the specified element parameter (If 2 arguments are specified, the second is the value to be set).
$id→description($)	Returns the description for the specified tag. valid tags are 'name' and the tags returned by the tags() member function. The 'name' tag returns a text string describing the whole element and the other strings describe the specified parameter. If called with no arguments, it returns the string for the 'name' tag.
$id→tags()	Return a list of valid element parameter names.
$id→functions()	Returns a string containing all the functions and variable decleratins for gnuplot.
$id→F($)	Returns the impedance value of the element at the specified frequency. The returned impedance is of type Math::Complex.
$id→fit()	Returns a list of possible fit variables (all starting with value 1) for use with gnuplot.
$id→save()	Returns a string containing functions to save the final fit variables (base value multiplied with fit variable). This is also for gnuplot use.
$id→print_line()	Returns a string defining the impedance element. The format is described below.
$id→helperfunctions()	Returns a list of paramter names for which special help functions exist for calculating usefull estimates for start values for fitting. Note, that some impedance elements does not have any way of determining good start values in which case the helperfunctions() member function simply returns an empty list.

$id→helpfunction(@)	This function has multiple uses. If called without any arguments, it returns the list from the helperfunctions() member function. If called with one argument it returns a string describing how many additional arguments must be parsed to it (which may wary depending on impedance element) and in which order. If called with more than one argument, the first Argument is the parameter name to calculate start value for, and the additional arguments (which must be a string of the form: "$frequency $real_value $imaginary_value") are used for the calculation.
$id→f_imax()	Returns the frequency for which the imaginary value of the impedance is at it's maximum (negative) value. If the element type makes such a calculation invalid, the function returns undefined. If the frequency is either infinite or the DC case, the reported frequencies will be 1e100 Hz and 1e-100 Hz respectively. For frequency independent elements the function returns 0.

The print_line member function returns a string which can be used to reload an impedance element. The format is:

Type: parameter1=value1, parameter2=value2, ...

An exception to this is the container elements (serial and parallel) which is defined and implemented in the Impedance::Complex class.

10.7 Impedance::RQ

Inherits from Impedance::Base (refer section 10.6).

This module implements Impedance::Base for R-C and R-L parallel connections as well as the R-Q parallel connection. (The Q beeing a constant phase element).

To obtain an instance, call one of the constructors:

$id = Impedance::RC→new($id);

$id = Impedance::RL→new($id);

$id = Impedance::RQ→new($id);

The module defines no additional member functions.

10.8 Impedance::W

Inherits from Impedance::Base (refer section 10.6).

This module implements Impedance::Base for diffusion type elements (W, O, Od, G, Gd and T). W is the Warburg element, O is the Finite length Warburg element, Od is a depressed / flattened finite length warburg element, G is the Gerisher element, Gd is a depressed / flattened Gerisher element, T is the 'Bounded Warburg' element and dL is the 'de Levie' element.

To obtain an instance call one of the constructors:

$id = Impedance::W→new($nr);

$id = Impedance::O→new($nr);

$id = Impedance::Od→new($nr);

$id = Impedance::G→new($nr);

$id = Impedance::Gd→new($nr);

$id = Impedance::T→new($nr);

$id = Impedance::dL→new($nr);

The module defines no additional member functions.

10.9 Impedance::H

Inherits from Impedance::Base (refer section 10.6).

This module implements Impedance::Base for diffusion type elements (Specifically the HavriliakNegami element).

To obtain an instance call one of the constructors:

$id = Impedance::H→new($nr);

The module defines no additional member functions.

10.10 Impedance::Complex

Inherits from Impedance::Base (refer section 10.6).

This module defines series and parallel connections of impedance elements. The resulting element can be treated as any other Impedance::Base derived element, thus it is possible using the serial and parralel elements to build arbitrary complex circuit layouts!

In effect the module defines two container types which from the outside behaves as a single impedance element. The Impecance::Complex container elements together with the Impedance::Base derived elements implements the Composite pattern with the container elements (Impedance::Complex derived) acting as GoF Composite classes and the Impedance::Base derived classes acting as Gof Leaf classes

To obtain an element instance, call one of the constructors:

$id = Impedance::Ser→new($id);

$id = Impedance::Par→new($id);

The module defines four new member functions:

$id→elements()	Returns a list of Impedance::Base (or derived) elements.
$id→elements_all()	Returns the complete element list for all elements in the container (recursively).
$id→element($)	Returns the element instance with the specified id (usually obtained from the elements member function).
$id→add_element($$)	Adds an element or the list of elements contained within the element instance. The arguments are type and id of the element to add. It returns the element instance created. The element is created using the Impedance::Device::new_device($$) factory function.

The print_line() function inherited from Impedance::Base is overloaded with slightly different behaviour. Instead of a string containing just a single line, the return value of print_line() contains multiple lines. The first line contains a start identifier (either '(' or '[' depending on type) and the last line contains the corresponding end identifier (')' or ']'). All intermediate lines are obtained by calling print_line() on the individual element(s) in the container. Note that this may include additional complex elements resulting in nested parantheses (Which is completely valid behaviour)!

Additionally the tags() function is also overloaded so that it returns a list of the elements in the container instead of valid parameter names (as the complex element is a container it does not by itself have any parameters of which to set or get the value).

It is also not possible to define any help functions on complex elements, thus any calls to helperfunctions() merely results in the empty list.

Notice that f_imax() can only report summit frequencies between 1e-100 and 1E100 Hz Thus if the true sumit frequency of the complex element used is outside this range it will be reported wrongly! Additionally, the f_imax() function first tries to determine a summit frequency in the R,-X quadrant (corresponding to where an arc from a parallewl RC circuit will be found). If no summit frequency is found it reports the summit frequency in the R,X plane (Where an parallel R-L circuit will be found).

10.11 Impedance::Device

This module defines two functions, the first is list_device_types() which returns a list of valid impedance element type names.

The second function is a factory function: new_device($$). It accepts two arguments, the first is an element type name (one of the types returned by list_device_types()) and the second argument is the element id (integer). The new_device($$) function returns the impedance instance created.

10.12 Impedance::Model

Inherits from Debug (refer section 10.1).

This module defines how to handle impedance models. It utilises the impedance elements defined by Impedance::Base and the derived classes.

To obtain an instance, call one of the constructors:

$id = Impedance::Model→new();

$id = Impedance::Model→new($data);

$id = Impedance::Model→new($model_ref);

$id = Impedance::Model→new($model_ref,$minf,$maxf);

The first constructor merely initialises a new empty model. The second constructor initalises an empty model but adds the impedance data in in the supplied data string to the model instance. The third constructor copies the model from the supplied model reference and copies the data from that instance as well. The fourth constructor copies as the third, but only copies those data which lies within the frequency range specified by the last two arguments! This can be used for partial fitting where only a specific frequency range is needed.

The individual impedance elements are strored in internal data structure which makes sure that only unique impedance element id's are used.

The Impedance::Model class implements the Composite patern (together with the Impedance::Base and Impedance::Complex derived classes) with Impedance::Model acting as a Gof Component class as well as a Gof Composite class.

All Impedance::Model instance has the following public member functions:

$id→parse($)	This function accepts a string defining an impedance model. The string must be lines of the format defined by the Impedance::Base function print_line(). The function parses this string and initialises the correct impedance elements based on this. The function returns the id number of the last impedance element added.
$id→print_model()	Returns a string which can be parsed by $id→parse($)
$id→device_types()	Wrapper for Impedance::Device::list_device_types().
$id→elements()	Returns a list of impedance element names (nr).
$id→element($)	Returns the impedance element specified. The function also gets elements from inside containers!
$id→elements_all()	Returns the complete element list of the model. Recursively calls into any container elements. Note that unlike the elements() function, the actual device instances are returned as opposed to the element names.
$id→model_text()	Returns a string representing the impedance model

$id→delete_element($)	Deletes the specified impedance element.
$id→add_element($[$])	Adds an impedance element of the specified type. If an additional arguemnt is specified, the id of the new element will be set to this number. If no second argument is specified, hte next id is simply chosen. The function returns a reference to the added element.
$id→fit()	Returns a list of possible fit paramerters for current model.
$id→data([$])	Gets or sets the impedance data in the internal data field.
$id→delete_point($)	Deletes the point specified from the internal data array. Returns the new data in the same way as data(). Note that index 0 is the point at the highest frequency.
$id→print_fit([$])	Returns a string containing all the functions nescesarry for gnuplot to initialise and fit the given model to the impedance data either in the specified filename or (in absence of an argument) in the internal data field. All data points are given even weight.
$id→print_fit_weight([$])	Same as print_fit(), however the actual fitting is done with uneven weight, so that data points with small values (absolute length of impedance vector) gets higher weight than points with larger values (weight inversely proportional to length).
$id→set_show_arcs($)	Sets wether the plot functions should include the individual arcs in the plots or not. An argument of 0 disabels the arcs, 1 includes them.
$id→print_plot([$])	Returns a string containing the function declerations and plot definitions for gnuplot to plot the specified data (if any) if no datafile name is specified, it uses the data in the internal data field.
$id→print_plot_range($$[$])	Similar to print_plot, except it must have the min and max range specified as the first 2 arguments. The effect is that any labels are only printed if they have x-values within the specified range.
$id→print_bode([$])	Same as above, but for bode plots.
$id→print_plot_eps([$])	Same as print_plot, but for eps file output.
$id→print_bode_eps([$])	Same as above, but for bode plots.
$id→print_plot_error([$])	Returns a string containing the function declerations and plot definitions for gnuplot to plot the difference between the model and the specified data (if any) if no datafile name is specified, it uses the data in the internal data field.

$id→print_imp_sim($$)	Returns the function declaratioins nescesarry for gnuplot to plot an impedance plot of the current model. Arguments are the frequency range to plot (min and max).
$id→print_bode_sim($$)	Same as above, but for bode plot.
$id→F($)	Returns the impedance of the current model for the specified frequency. The returned impedance is of type Math::Complex.
$id→subset($$)	Returns a subset of the data in the internal data field based on the specified minimum and maximum frequency.
$id→subtract_model()	Returns the residual of the data in the data field once the impedance of the current model has been subtracted (subtraction done in the indivudual data points!).
$id→get_error()	Returns the mean and maximum error for the data and model chosen The error is calculated as the absolute difference of the data from the mode both the real and for the imaginary part. The error is then normalised with the modulus of the data value. This is calculated for all frequencies in the data set and the mean and maximum values are returned (as percentages).
$id→minf()	Returns the minimum frequency in the data set.
$id→maxf()	Returns the maximum frequency in the data set.
$id→minr()	Returns the minimum real part of the impedance in the data set.
$id→maxr()	Returns the maximum real part of the impedance in the data set.
$id→mini()	Returns the minimum imaginary part of the impedance in the data set.
$id→maxi()	Returns the maximum imaginary part of the impedance in the data set.
$id→scale_factor([$])	Sets the scale factor function to the specified value. If no arguments, the scale factor function is set to the default 1, that is no scaling. Note that only simple proportionality scaling is possible.
$id→get_scale()	Returns the current scale factor.
$id→set_limit($$)	Sets the plot limit of one of the fololowing tags: 'xmin', xmax', 'ymin' and 'ymax' to the specified value. Arguments: tag, value.
$id→get_limit($)	Returns the limit of the specified tag (see set_limit()).
$id→set_xlabel($)	Explicitly sets the text string to be displayed in the xlabel.
$id→set_ylabel($)	Explicitly sets the text string to be displayed in the ylabel.

$id→set_ylabel_bode($)	Explicitly sets the text string to be displayed in the ylabel in bode plots.
$id→set_bode_mode($)	Sets wether bode plot should only plot imaginary value or both real and imaginary (default). If called with a true argument mode is set to only imaginary values.
$id→set_admittance_mode($)	Sets wether or not the plots should be of the impedance, admittance, complex modulus or compelx capacitance. Default is impedance. Valid values can be obtained from admittance_mode_values().
$id→admittance_mode_values()	Returns a list of valid values for set_admittance_mode($).
$id→adv_opt([@])	Sets or gets a list of additional gnuplot commands (for instance user defined axis labels etc.). Arguments are a list of commands. If called without arguments, the current list of additional commands is returned

Additionally the following private member functions are also defined. Although Perl does not permit true private functions, do not use these functions from outside the class instances!

$id→print_plot_common()	This function returns the function declarations for all elementsin the impedance model.
$id→print_plot_main()	This function does the actual work of print_plot() and print_bode()
$id→print_plot_main_range()	This function does the actual work of print_plot_range().
$id→print_plot_eps_common()	Similar as above, bur for eps output.
$id→dev_sort_fmax()	Returns a hash of device names and summit frequencies Only returns devices for which summit frequencies can be correctly calculated.
$id→print_simulation()	This function does the actual work of print_imp_sim() and print_bode_sim().
$id→save()	This function saves the content of the data field to a temporary file (in /temp) It returns the filename of the temporary file.
$id→file()	Returns the name of the data file.

Chapter 11

Web service interface

It is possible to use Elchemea Analytical as an web service. To do so call the following *ajax_model.cgi* web script with the parameters specified below. The script is located in *host.domain/cgi-bin/analytic/ajax_model.cgi* (substitute host.domain with the correct host name and domain name of your Elchemea Analytical installation).

1. 'ajax', value: '1'
2. 'action', value: 'fit' or 'fitlist'. If fitlist is selected, the web service returns a list of possible free fit parameters for the specified model
3. 'model', value: the impedance model as specified in chapter 3.
4. 'fitmode', value: 'Even' or 'Inverse'. Determines if data points have even weight (which is default) or weight is inversely proportional to absolute value of impedance (length of impedance vector). Optional parameter, only used for 'fit' action.
5. 'data', value: Impedance data in the form of multiple lines, each line in the form: frequency real_value imaginary_value
   Note that the first line must be the highest frequency. Only used for 'fit' action
6. 'fitlist', value: comma separated list of free fitting parameters. The elements specified must be from the list returned by the fitlist action (refer item 2), only used for 'fit' action.

In the case 'fit' action is selected and the fit converges, the resulting response would start with the string 'OK' on a single line followed by the fit result. This would include arc summit frequencies as well as pseudo capacitance's (in the case of RQ elements) as well as the full set of final parameter values. The output would also include references to a number of png files temporarily located on the web server (of approx 10 minutes or so). These images can be downloaded separately and shows the fit result and error plot for the data and model in question.

The file *webservice.html* found by pointing your browser to
host.domain/analytic/webservice.html contains two web forms with the necessary form

elements to do a impedance fitting using the web interface as well as testing the 'fitlist' action. By submitting the 'fit' test form with the default parameters, the output should look like this (Your browser likely 'eats' the newlines, so use view source):

```
OK
L_1:1.509e-08
R_2:0.01823
R_3:0.009409
Y_3:0.0387
n_3:0.7735
INFO: Maximum frequency for arc for element RQ_3: 4442
INFO: Pseudo capacitance for arc for element RQ_3: 0.003808
R_4:0.00219
Y_4:3.358
n_4:0.8441
INFO: Maximum frequency for arc for element RQ_4: 53.64
INFO: Pseudo capacitance for arc for element RQ_4: 1.355
R_5:0.008561
Y_5:29.01
n_5:0.9397
INFO: Maximum frequency for arc for element RQ_5: 0.7008
INFO: Pseudo capacitance for arc for element RQ_5: 26.53
FITDATA


*******************************************************************************
Wed Dec  7 12:26:16 2011


FIT:    data read from '/tmp/fitset_26083.dat' u 2:1:3:(1)
        #datapoints = 74
function used for fitting: h(x,y)
fitted parameters initialized with current variable values



Iteration 0
WSSR        : 1.32424e-05       delta(WSSR)/WSSR   : 0
delta(WSSR) : 0                 limit for stopping : 1e-05
lambda  : 0.00565434

initial set of free parameter values

fl_1            = 1
fr_2            = 1
fy_3            = 1
n_3             = 0.8003
fr_3            = 1
fy_4            = 1
n_4             = 0.7235
fr_4            = 1
fr_5            = 1
fy_5            = 1
n_5             = 0.9762

After 9 iterations the fit converged.
final sum of squares of residuals : 2.32063e-07
rel. change during last iteration : -8.43531e-07

degrees of freedom (ndf) : 63
rms of residuals      (stdfit) = sqrt(WSSR/ndf)      : 6.06922e-05
variance of residuals (reduced chisquare) = WSSR/ndf : 3.68354e-09

Final set of parameters            Asymptotic Standard Error
=======================            ==========================

fl_1            = 1.10797          +/- 0.008277     (0.7471%)
fr_2            = 0.97708          +/- 0.003211     (0.3286%)
```

```
fy_3            = 1.16728          +/- 0.1147       (9.828%)
n_3             = 0.773495         +/- 0.01046      (1.352%)
fr_3            = 1.01687          +/- 0.01714      (1.686%)
fy_4            = 0.519381         +/- 0.09599      (18.48%)
n_4             = 0.844051         +/- 0.04384      (5.194%)
fr_4            = 0.670101         +/- 0.05151      (7.687%)
fr_5            = 1.08185          +/- 0.01273      (1.177%)
fy_5            = 0.928554         +/- 0.01251      (1.347%)
n_5             = 0.9397           +/- 0.007901     (0.8408%)


correlation matrix of the fit parameters:

               fl_1   fr_2   fy_3   n_3    fr_3   fy_4   n_4    fr_4   fr_5   fy_5   n_5
fl_1            1.000
fr_2           -0.623  1.000
fy_3            0.485 -0.773  1.000
n_3            -0.536  0.837 -0.992  1.000
fr_3            0.510 -0.822  0.923 -0.930  1.000
fy_4           -0.104  0.185 -0.319  0.305 -0.483  1.000
n_4             0.225 -0.379  0.586 -0.566  0.742 -0.924  1.000
fr_4           -0.317  0.520 -0.757  0.735 -0.870  0.767 -0.928  1.000
fr_5            0.114 -0.196  0.314 -0.303  0.429 -0.755  0.720 -0.740  1.000
fy_5           -0.102  0.177 -0.282  0.272 -0.380  0.633 -0.612  0.584 -0.441  1.000
n_5            -0.075  0.128 -0.207  0.199 -0.288  0.555 -0.515  0.566 -0.876  0.140  1.000
END_TEXT
FILE:model_26083.png
REPORT:report_model_26083.png
ERROR:report_error_26083.png
```

Note that some of the numbers may be slightly different as a randum number generator is invovlved. The last three lines indicates filenames, and in order to access the individual files, point your browser to *host.domain/analytic/png/* and select the file(s) specified.

Chapter 12

Troubleshooting

12.1 Proxy server preventing automatic updating

In order for the automatic software updater (analytic_updater.pl) to work through a web proxy, create the file '/home/analytic/proxy.conf'. The file should contain a single line with the proxy server name as well as the proxy port as shown below:

http://proxy.foo.bar:1234

In this case the proxy port is port 1234.

12.2 Server error is reportet when starting Elchemea Analytical

- Inspect /var/log/httpd/error_log
- If SE-linux running in enforcing mode, try and disable it by using 'setenforce 0'.
- If htis resolved the error, inscpect the file /var/log/audit/audit.log and find the files/directories with conflicting SElinux labels. The likely culprits may be '/var/lock' and '/var/SemaforeFile/', refer SE-linux documentation as to how to do see and change labels.
- Reenable SE-linux by running 'setenforce 1' and check.

12.3 Model section of view is mangled

It is known that some versions of Microsoft Internet Explorer® display some of the html elements wrongly, thus leading to mangled model views. In this case either upgrade your browser, or switch to Firefox or Google Chrome (ElchemeaAnalytical has not been tested with Apples Safari browser).

12.4 After fitting, pressing the report button only says 'no report ready'

- Check that the LATEX server is running. To do this type the following in a terminal:

  ps -ef | grep LATEX

  The output should look something like this:

```
sofc  10320 10297  0 10:17 pts/3  00:00:00 grep LATEX
sofc  19042     1  0 Aug02 ?      00:00:00 /usr/bin/perl /usr/local/bin/analytic/LATEX-server
```

  If the last line is not observed, start it by calling
  /usr/local/bin/analytic/start_servers as root.

- If the server is running, check that only one version of the LATEX.pm module is installed. Older versions fo Risø Fuel Cells and Solid State Chemistry division fuel cell control system installed modules in a different location than Elchemea Analytic, and depending on search path, the Elchemea Analytic installer may not have discovered the older version and installed the new wersion independently. If this happens, the old version may be used by the LATEX-server, and unfortunately the old module misses some functions needed by the server.

  To fix this, locate and delete the old module.

- If the problem persists (or only one version of LATEX.pm exists, then if SE-Linux is running in enforcing mode, then it may prevent the Apache webserver from accessing the LATEX-server on localhost port 4050. The easiest way to fix this is to run SE-Linux in permissive mode (nonenforcing). However, be aware that SE-Linux is part of the Linux intrusion detection system / security system, and thus running SE-Linux in nonenforcing mode may potentially expose the system to external threats it would otherwise be protected from. To run SE-Linux in non-enforcing mode in oreder to check if this is the cause of the problem, consult your Linux distribution manual (on CentOS version 6.x SE-Linux can be turned off by typing 'setenforce 0' in a root terminal).

  The specific problem with SELinux is that Apache needs to be able to connect to port 4050 (the LATEX-server) and be able to read the corresponding PDF file (a separate issue). To allow apache to connect to port 4050 as well as execute the scripts in /usr/local/bin/analytic/ (on CentOS) execute the script *set_SELinux_rules.bash* found in the ElchameAnalytics installation directory.

12.5 Fitting does not finish (page displays 'working...' and stops)

Check that the 'utf8.def' file is included in your LaTeX installation, if not, locate the following line in LATEX.pm and uncoment it (place a # in front)

```
$res .= '   \usepackage[utf8]{inputenc}'.''\n'' if ($utf8);
```

Note that this has to be done in the LATEX.pm file used by the implementation (Not the one in the insttall directory!).

12.6 Fitting takes too short time and no response is recieved

Likely your model contains one or more of the elements which contains a trigonometric function in the mathematical description (this includes Gerisher, Finite length Warburn and Bounded Warburg) and the fitting routine hit a value which resulted in infinite impedance (refer section 3.4).

12.7 Graphs not shown correctly and/or pages does not finish loading

Check that the default lock file (called SemaforeFile.lock) for the SemafoeFile.pm module has the right permissions. It is located in /tmp and should have permissions 666 (Yes, I know the number of evil...) Durring normal operation, it will be created with this permission, but sometimes the system mauy clean up the temp directory, and in this case sometimes it may be created with the wrong permissions. To resolve this, simply remove the file or manually set the right permissions (both operations may be nescesarry to do as root).

12.8 My screen is not wide enough to show all information

This can happen if you are using an older screen/projector which only allows a maximum horisontal resolution of 1024 pixels. To correct this, edit the file Header.pm in the Impedance direstory (likely placed somewhere under /lib) and change the size variable from "set size 1.3,1.1" to "set size 0.9,0.7" as well as the pssize varialbe from "800x600" to "600x400" (Remember to change both variables!). Notice however that this is a site wide variable, so all users of Elchemea Analytic on this server will be affected.

12.9 Multiplot graphs are sideways

Make sure that gnuplot, ImageMagick and Ghostscript are up to date. On Centos/RHEL this can be achieved by executing the following commands as root:

```
yum update gnuplot
yum update ImageMagick
yum update ghostscript
```

Other distributions handle this in a different way, refer the distribution manual as to hwo to update software packages.

12.10 Multiplotting suddenly fails with an error message including the string 'all points y value undefined'

Due to the multiuser nature of the Elchemea Analytical system, uploaded files can not be saved indefinite on the server, and if a file has been left unused for some time (usually an hour) it will be deleted. This usually does not happen when working normally, but if the user leaves the Elchemea analytical session for extended time and then returns, this may happen (see section 4).

12.11 Some of the last tics on the graps is missing (graph goes to 100 but tics only shown to 70 for instance).

This is caused by certain versions of Gnuplot. Version 4.2.6 is known to do this. To correct this, edit the file Header.pm in the Impedance direstory (likely placed somewhere under /lib) and change the size variable from "set size 1.3,1.1" to "set size 1,1". Notice however that this is a site wide variable, so all users of Elchemea Analytic on this server will be affected.