Java/JNI/C/Fortran based HSVD/HLSVD custom plugins for the *j*MRUI software system: Development, installation and usage

R. de Beer and D. van Ormondt
Applied Physics, TU Delft, NL
E-mail: r.debeer@tudelft.nl

2015-04-08 14:08

Index Terms

Java/JNI/C/Fortran Makefile project, Eclipse ADT bundle, *j*MRUI software system, HSVD- and HLSVD-based custom plugin

I. Introduction

We have developed two Java/JNI/C/Fortran based [1] custom plugins for the *j*MRUI software system, aiming at applying the HSVD [2] and the HLSVD [3] algorithm. They are to be used for performing SVD-based quantification of *j*MRUI signal files with the *mrui* file extension. The plugins were developed for the Windows and Linux operating system, using the Eclipse Java IDE of the Eclipse ADT bundle [4].

The work was done in the context of providing additional *research* tools for the *j*MRUI software system. The goal of the plugins is to give insight into the SVD-based quantifications, particularly concerning the choice of the value of the hyper-parameter *Number of Components* (ncom in the plugin GUI). [1]. To that end the plugins write many standard outputs of intermediate results to the corresponding plugin log files. Also, we have put effort into making the maximum value of *Number of Components* as large as possible (depending on the size of the Hankel data matrix, concerned, of course).

This manuscript concerns a description of developing, installing and running the HSVD and HLSVD custom plugin on the Windows/Linux platform.

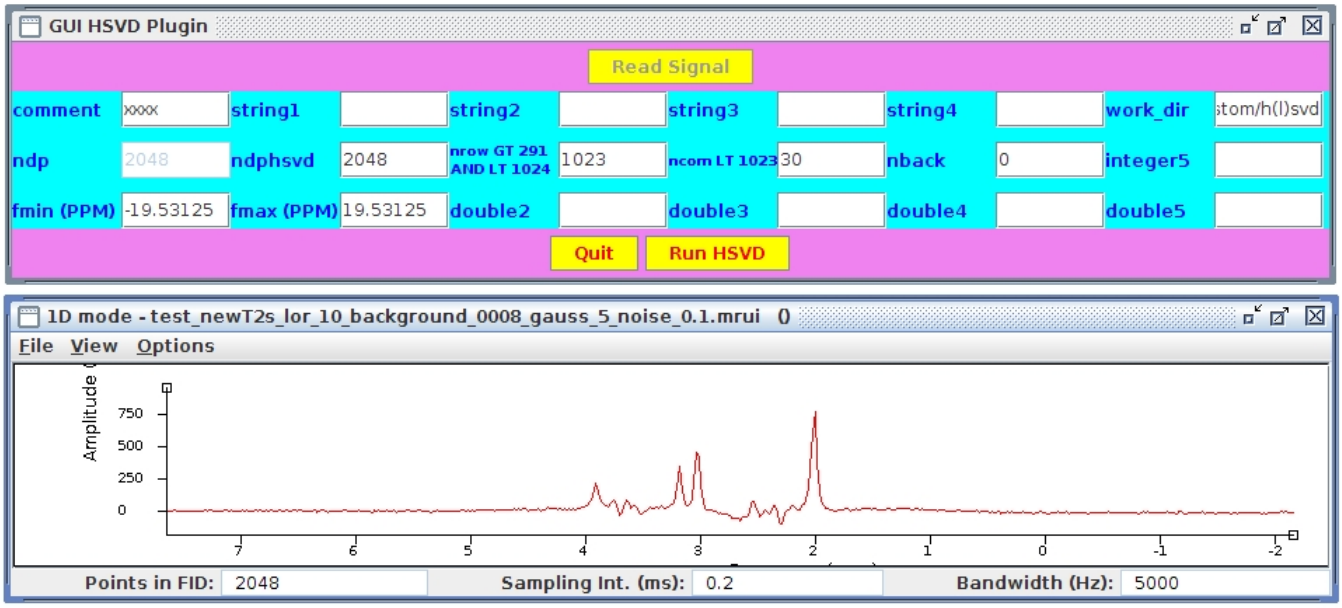


Figure 1: GUI of the HSVD plugin. Also shown is the FFT of the *j*MRUI *mrui* signal file, that was chosen to be quantified by the HSVD algorithm. Note the input limits for *Number of Rows* and *Number of Components*. These limits are shown, after changing the GUI text field AND indicating that the text entry is complete by pressing the computer *Enter*.

[1] A component in H(L)SVD is a mono-exponentially damped sinusoid with a complex-valued amplitude.

II. Plugin Development

A. Integrated Development Environment (IDE)

The plugins were developed by applying the same kind of Java/JNI/C/Fortran Makefile approach, as we have used recently for the *j*MRUI MonteCarlo custom plugin, the latter being created for performing Monte Carlo studies of simulated *in vivo* MRS signals [5]. This means, that we have worked with two Java/JNI/C/Fortran Makefile projects within the Eclipse Java IDE of the Eclipse ADT bundle [4]. As a result, the GUI structure of the HSVD and HLSVD plugin is equal and also similar to the GUI structure of the MonteCarlo plugin (see, as example, Figure 1 for the HSVD GUI).

B. Details of applied computer codes

1) SVD of the Hankel data matrix: In the plugins, the HSVD or HLSVD algorithm is carried out by Fortran code. The main difference between the two codes is the SVD handling of Hankel data matrix. For HSVD [2] this is realized by using the Lapack subroutine `zgesvd.f` [6] and for HLSVD by using our (home-written) code `lanczos.f` [3].

After having carried out the SVD of the Hankel data matrix, the calculation of the component parameters (amplitude, frequency, damping constant and phase) is the same for both plugins. This part of the calculation again is based on using Lapack subroutines.

2) Monitoring the plugin-progress with automatic Gnuplot updates: In the HSVD and HLSVD algorithm the computational time for a large part is determined by the SVD of the Hankel data matrix. Since in our Fortran computer code the 2D Hankel matrix is allocated with size $(\mathrm{nrow}, \mathrm{ncol})$, where $\mathrm{nrow} < \frac{\mathrm{ndp}}{2}$ and $\mathrm{ncol} = \mathrm{ndp} - \mathrm{nrow} + 1$ [2] , it means that the SVD can become a computational burden when $\mathrm{nrow} = \frac{\mathrm{ndp}}{2} - 1$ and $\mathrm{ndp} \geq 2048$. In that case it is convenient to perform real-time monitoring in order to track the progress of the Fortran code. To that end we have created a progress bar based upon calling the Gnuplot graphing utility [7] from our Fortran code. The essential part of the code is shown in Figure 2 .

Fortran based Gnuplot code

```
! create gnuplot command file
  OPEN(10,ACCESS='SEQUENTIAL',FILE='gp_bar.txt')
  write(10,*) 'set size 1,1'
  write(10,*) 'set terminal wxt size 50,600 position 800,0'
  write(10,*) 'set title "Progress\nafter\nstart\nFortran"'
  write(10,*) 'unset key'
  write(10,*) 'unset xtics'
  write(10,*) 'unset ytics'
  write(10,*) 'set border linewidth 5.0'
  write(10,*) 'set xrange [0.999:1.001]'
  write(10,*) 'set yrange [0.0:100.0]'
  write(10,*) 'plot "hsvd_bar_data.txt" using 1:2 with &
    lines linecolor rgb "blue" linewidth 5.0'
  write(10,*) 'pause 0.2'
  write(10,*) 'reread'
  CLOSE(10,STATUS='KEEP')

! plot curve with gnuplot
  ret=SYSTEM('START /B C:\"Program Files"\gnuplot\bin\&
    gnuplot gp_bar.txt')
```

Java based GUI code

```
  private void quitActionPerformed (java.awt.event.
    ActionEvent evt) {
    try {
      Process p = Runtime.getRuntime().exec("TASKKILL
        /F /IM gnuplot.exe");
    }
    catch(Exception e) {}
    dispose();
  }
```

For Windows

Fortran based Gnuplot code

```
! create gnuplot command file
  OPEN(10,ACCESS='SEQUENTIAL',FILE='gp_bar.txt')
  write(10,*) 'set size 1,1'
  write(10,*) 'set terminal X11 size 60 position 800'
  write(10,*) 'set title "Progress\nafter\nstart\nFortran"'
  write(10,*) 'unset key'
  write(10,*) 'unset xtics'
  write(10,*) 'unset ytics'
  write(10,*) 'set border linewidth 1.5'
  write(10,*) 'set xrange [0.999:1.001]'
  write(10,*) 'set yrange [0.0:100.0]'
  write(10,*) 'plot "hsvd_bar_data.txt" using 1:2 with &
    lines linecolor rgb "blue" linewidth 20.0'
  write(10,*) 'pause 0.2'
  write(10,*) 'reread'
  CLOSE(10,STATUS='KEEP')

! plot curve with gnuplot
  ret=SYSTEM('gnuplot gp_bar.txt &')
```

Java based GUI code

```
  private void quitActionPerformed (java.awt.event.
      ActionEvent evt) {
    try {
      Process p = Runtime.getRuntime().exec("killall
        gnuplot_x11");
    }
    catch(Exception e) {}
    dispose();
  }
```

For Linux

Figure 2: Fortran based Gnuplot code (upper part) for *monitoring* the plugin-progress (HSVD example) and Java based GUI code (lower part) for *killing* the corresponding operating system process. Note the `pause` and `reread` command in the Gnuplot code, creating an *infinite loop*. Note also, that the Gnuplot process is running in the *background*.

When looking at Figure 2 (HSVD example), we like to make the following remarks:

- The Gnuplot code for plotting the progress bar is written from Fortran to the disk file gp_bar.txt.
- The data-points for the Gnuplot plotting curve are supposed to be present in the disk file hsvd_bar_data.txt (also written from Fortran, at various moments during the progress of the HSVD algorithm; the latter is not shown in Figure 2).

[2] nrow is the number of rows and ncol the number of columns of the 2D Hankel data matrix and ndp is the number of complex-valued data-points of the MRS signal, concerned.

- After each `pause` of 0.2 s, the data-points of hsvd_bar_data.txt are plotted again (due to the `reread` command), thus realizing an infinite Gnuplot loop.
- To prevent *blocking* of the Fortran code, the Gnuplot process is set to run in the background.
- This background Gnuplot process is killed via Java code, when clicking the Quit GUI button (see again Figure 1).

3) Monitoring HSVD vs HLSVD: In the case of HSVD *all* singular values of the Hankel data matrix are calculated, this in contrast to the case of HLSVD, where *only the signal-related* singular values are to be calculated (invoking the Lanczos algorithm; this is where the reduction in the SVD computational time comes from [2] [3]).

The Lapack subroutine `zgesvd.f` [6] offers, as far as we know, no possibility to track the calculation of the singular values. That is to say, in the Fortran code we can only track the begin and end of the call to this library subroutine. In the case of HLSVD, however, we are using our own home-written `lanczos.f` code, which has given us the possibility to track the number of singular values, determined.

The Lanczos algorithm is suited for this tracking purpose, since the singular values are determined in an *iterative* way. In fact, in the Fortran code the progress percentage is determined, each time when reaching one of some specified values of the number of Lanczos iterations, from calculating $100.0 \times \mathrm{nsvd}^{\mathrm{found}}/\mathrm{nsvd}^{\mathrm{asked}}$, where $\mathrm{nsvd}^{\mathrm{asked}}$ is the number of singular values, originally asked for (via setting ncom in the plugin GUI) and $\mathrm{nsvd}^{\mathrm{found}}$ is the number of singular values, actually found by Lanczos (for that specified number of iterations).

III. Plugin Installation

A. Steps to be taken

Installing the plugins (HSVD example [3]) amounts to performing the following steps:

1) Copy the HSVDPlugin_windows.jar and HSVDPlugin_linux.jar JAR files into the *j*MRUI *plugins* directory. Depending on the local operating system, rename one of the two JARs to HSVDPlugin.jar.
2) Copy the libhsvd.dll, callfortran_hsvd.dll, liblapack.dll and libblas.dll files (our own shared-library dll's for Windows), the the libgcc_s_dw2-1.dll, libgfortran-3.dll and libquadmath-0.dll files (extra MinGW dll's for Windows) and the libhsvd.so file (our own shared-library for Linux) into the *j*MRUI *lib* directory.
3) Copy ALL HSVD plugin related *input files* (see below) into a desired *j*MRUI *working directory* (chosen via the *j*MRUI Setup window). Set via the Setup window also the other directories to this same directory.

B. Input disk files

The input disk files, that should be present in the plugin working directory, are:

1) At least one *j*MRUI yourname.mrui *mrui* signal file, that should contain the values for the number of data-points (ndp), the time-domain sampling step and the Larmor frequency in its file header.
2) A gui_info.txt text file with (a sort of default) GUI input values.
3) A fftcheck.fil text file, containing possible ndp values (at present only up to 1024), allowed by the mixed-radix FFT used in the Lanczos algorithm.

IV. Plugin Usage

A. Steps to be taken

Running the plugins (HSVD example; similar for HLSVD) amounts to:

1) Launch the plugin GUI via the *j*MRUI Desktop Custom menu.
2) Click the Read Signal GUI button and load the *mrui* signal file, to be quantified by HSVD and minimize its Graphical Window.
3) The GUI input text fields now have contents (obtained by reading the gui_info.txt text file), which represent the text-field values, used in the *previous* plugin session.
4) If the user wants to use these text-field inputs, the Run HSVD GUI button can be clicked. After that the HSVD algorithm is carried out, the results are written to a number of output disk files (see below) and the current values of the text-field inputs are written to gui_info.txt.

[3] For HLSVD, change in the enumerate items hsvd into hlsvd and HSVD into HLSVD.

5) Before clicking the Run HSVD GUI button, the user can decide to change one or more input fields, of course. Note, that then the computer *Enter* should be pressed to indicate, that the text entries, concerned, are complete (see again the caption of Figure 1).

6) The text entry nback indicates the number of data-points that is *reconstructed backwards* (using the parameters of the HSVD components, found; usually set at 0, so nothing happens).

7) The text entries fmin and fmax indicate a minimum and maximum frequency (in PPM), that can be used to perform HSVD-based *filtering*. More concretely, if the signal sampling step is 0.2 ms, its Larmor frequency 128.0 MHz (3T measurement), fmin = 1.5 PPM and fmax = 4.5 PPM, then the original signal is filtered (using the parameters of the HSVD components, found) from -19.531 to 1.5 PPM and from 4.5 to +19.531 PPM [4] . That is to say, the original signal is supposed to be *unchanged* from fmin to fmax.

8) Close the plugin by clicking the Quit GUI button.

B. Output disk files

The following output disk files (HSVD example; similar for HLSVD) are written to the plugin working directory:

1) A callfortran_hsvd_xxxx.log text file, containing the standard-output messages from the Fortran part of the plugin.

2) A gp_bar.txt, hsvd_bar_data.txt and hsvd_sinvals_data.txt text file for monitoring the HSVD Fortran progress and plotting the HSVD determined singular values (by launching the Gnuplot program from the Fortran code).

3) A number of $*$.*mrui* files, called hsvd_<name>_xxxx.mrui, where <name> is recon, residu, signal and signal_filtered and sinvals, respectively and xxxx a comment (GUI entry; see Figure 1). These files are to be viewed via the *j*MRUI system.

4) A sinvals.ps file for viewing a plot of the singular values (using a suited viewer of the operating system, concerned).

5) A track.sv text file, showing the number of singular values, found after multiples-of-ten Lanczos iterations (HLSVD only).

```
.....
ndp =  2048
step (in ms) =   0.20
larmor (in MHz) = 128.00
.....
ndphsvd =  2048
nrow =  1023
ncom =    30
nback =     0
fmin (PPM) =    0.17500000000000000D+01
fmax (PPM) =    0.42000000000000000D+01
.....
 Begin sinvals from zgesvd
   1    0.4021E+03   2    0.2584E+03   3    0.1740E+03   4    0.1011E+03
   5    0.7836E+02   6    0.6494E+02   7    0.5937E+02   8    0.4283E+02
   9    0.3588E+02  10    0.2604E+02  11    0.2349E+02  12    0.2065E+02
  13    0.1904E+02  14    0.1823E+02  15    0.1543E+02  16    0.1342E+02
  17    0.1247E+02  18    0.1225E+02  19    0.1128E+02  20    0.1062E+02
  21    0.1050E+02  22    0.1008E+02  23    0.1004E+02  24    0.9905E+01
  25    0.9876E+01  26    0.9839E+01  27    0.9762E+01  28    0.9697E+01
  .....
 End sinvals from zgesvd

per_svd (via equation) = 96.000 %
per_ev (via equation) = 98.000 %
.....
t_zgesvd - t_start = 18.753 seconds
t_zgeev  - t_start = 18.765 seconds
t_zgelss - t_start = 18.773 seconds
t_finish - t_start = 18.781 seconds
per_svd = 99.851 %
per_ev  = 99.915 %
per_lss = 99.957 %
            ampl            freq             damp            phas
            a.u.            PPM              Hz              degr.
    1   0.013107236574 -18.304828825209  -7.287346475236   66.678133111564
    2   0.012581926403  -9.258746069188 -81.139753376164   -8.662208358294
    3   0.015192366205  -8.807986791830 -31.600597665361  -49.749149648437
    4   5.101255946228   1.665762879489-440.485307964390  -18.511306741197
    5   3.336722986105   2.005830943632 -20.672222708768    9.559904043880
   .....
```

From HSVD log

```
.....
test for check =  0.000
  start iteration no.   1
  start iteration no.   2
  start iteration no.   3
  start iteration no.   4
  start iteration no.   5
 .....
  start iteration no. 750
.....


Begin sinvals from lanczos
   1    0.4021E+03   2    0.2584E+03   3    0.1740E+03   4    0.1011E+03
   5    0.7836E+02   6    0.6494E+02   7    0.5937E+02   8    0.4283E+02
   9    0.3588E+02  10    0.2604E+02  11    0.2349E+02  12    0.2065E+02
  13    0.1904E+02  14    0.1823E+02  15    0.1543E+02  16    0.1342E+02
  17    0.1247E+02  18    0.1225E+02  19    0.1128E+02  20    0.1062E+02
  21    0.1050E+02  22    0.1008E+02  23    0.1004E+02  24    0.9905E+01
  25    0.9876E+01  26    0.9839E+01  27    0.9762E+01  28    0.9697E+01
 .....
End sinvals from lanczos
.....









           ampl            freq             damp            phas
           a.u.            PPM              Hz              degr.
   1   0.013107234017 -18.304828813786  -7.287284798349   66.678219039122
   2   0.012582385547  -9.258745041919 -81.140605529245   -8.662350196589
   3   0.015192332293  -8.807986580506 -31.600910650277  -49.749222395950
   4   5.101260284436   1.665762971539-440.485580013273  -18.511297247770
   5   3.336722945887   2.005830943293 -20.672222526334    9.559905388472
  .....
```

From HLSVD log

Figure 3: Log file (manually edited) as produced by the HSVD plugin. For comparison, the corresponding log file from the HLSVD plugin is also shown.

[4] The FFT bandwidth is $\frac{1000.0}{(\text{step} \times \text{Larmor})}$ PPM.

V. Results and Discussion

A. *HSVD of a simulated 3T* in vivo *MRS signal*

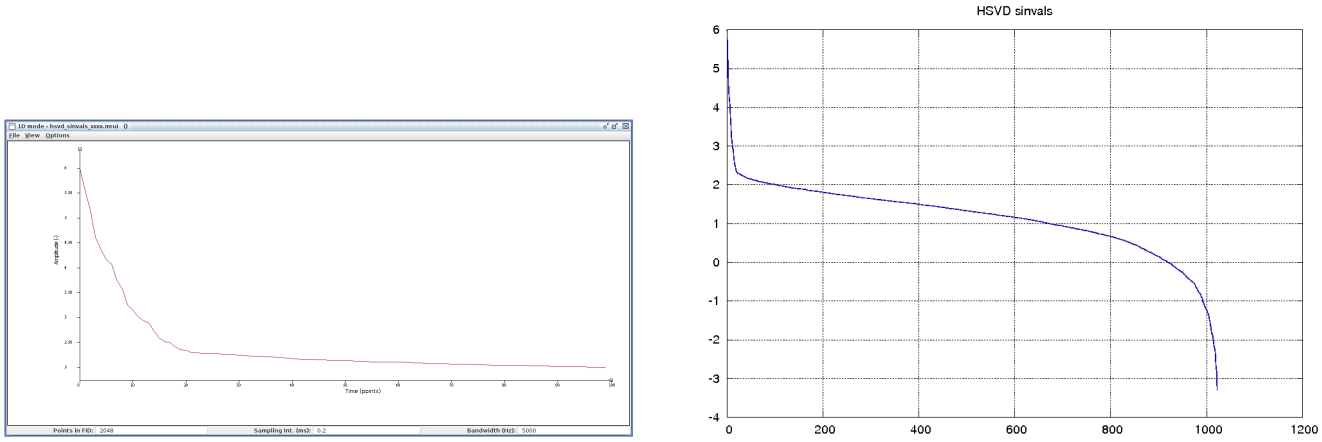
In this subsection we present the HSVD plugin results for a simulated 3T *in vivo* MRS signal. This signal was calculated with our in-house *in vivo* PRESS simulation program [8] [9], based on the GAMMA C++ library [10]. In the calculations of the metabolite database signals, used for constructing the simulated signal, the effects of the transverse relaxation time (T_2) as well as details of the PRESS *in vivo* MRS measurement protocol were taken into account [11].

In Figure 3 some standard outputs from the plugin Fortran code are presented, as obtained by editing the HSVD log file, concerned. For comparison, also the edited log file from the corresponding HLSVD plugin session is shown (see again subsection IV-B for the names of the log files).

When looking at Figure 3, we like to make the following remarks:

1) The Hankel data matrix had size (1023,1026), being the largest size possible for ndphsvd = 2048.
2) For this size of the Hankel data matrix, the computational time of the HSVD Fortran code is completely dominated by the SVD of Lapack subroutine `zgesvd.f` (as indicated by per_svd = 99.851 % in the log file).
3) The corresponding (same GUI input) HLSVD plugin session required 750 Lanczos iterations to find the number of singular values, asked (ncom = 30).
4) The values of the singular values and the component parameters are the same (to a certain precision) for HSVD and HLSVD (as should be expected).

Figure 4 displays two plots of the logarithm of the (1023) singular values, as produced via *j*MRUI's plotting facility and via the Fortran launched Gnuplot. Since the graphics system of *j*MRUI enables interactive zooming, this can be used to establish (estimate) the number of signal-related singular values. In the present case, Figure 4 (a) suggests a value slightly larger than 20 (for the SNR of the signal, concerned; sd_noise = 0.1). If this is true, our GUI input choice of ncom = 30 has been somewhat too large (for the iterative Lanczos algorithm to gain sufficient computational time [3]).



(a) (b)

Figure 4: Plot of the logarithm of the singular values (of the Hankel data matrix), as determined by the Lapack subroutine `zgesvd.f`. (a) The first 100 singular values, displayed with a *j*MRUI session. (b) All 1023 singular values, displayed with Fortran launched Gnuplot. Note, that the *j*MRUI produced plot can be zoomed *interactively*, which makes it easier to estimate the number of signal-related singular values.

A way to judge the results of the HSVD quantitation is to compare the spectrum of its components-based reconstruction with that of the signal. This is done in Figure 5 . It can be seen, that with 30 HSVD-found components the residue between the signal and the reconstruction is surprisingly noisy. Nevertheless, this noisy residue needs NOT to mean, that individual HSVD components, that can be attributed to belong to one of the metabolites, used to create the simulated signal, have the correct metabolite parameter values. This is, because many different Hankel-matrix based singular-value-decompositions are possible (for instance, when changing the size of the Hankel matrix), that all may yield similar noisy residues.

B. *Computational times of HLSVD vs HSVD*

As already mentioned in section II-B3 , the reduction in the computational time of HLSVD originates from the fact, that HLSVD is supposed to determine only components, belonging to the signal-related singular values, whereas HSVD determines

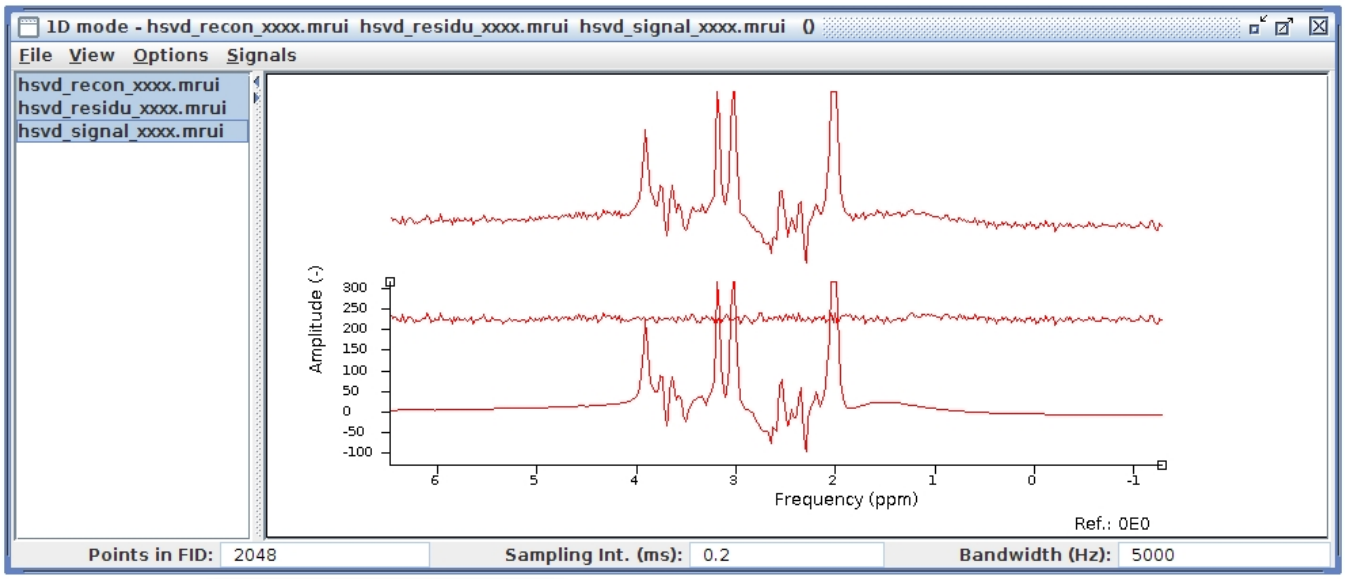


Figure 5: Plot of the spectrum of the HSVD reconstruction (lower), the signal (upper) and the difference between the two (middle). The number of HSVD components ncom = 30.

the components, belonging to all singular values of the Hankel matrix. In other words, the computational time of HLSVD (strongly) increases as a function of the number of components, asked. This important aspect is viewed in Figure 6 (for the same MRS signal, as used in the previous subsection). It can be seen from it, that for this example the HLSVD computational time increases to as much as 70 % of that of HSVD for 45 components.

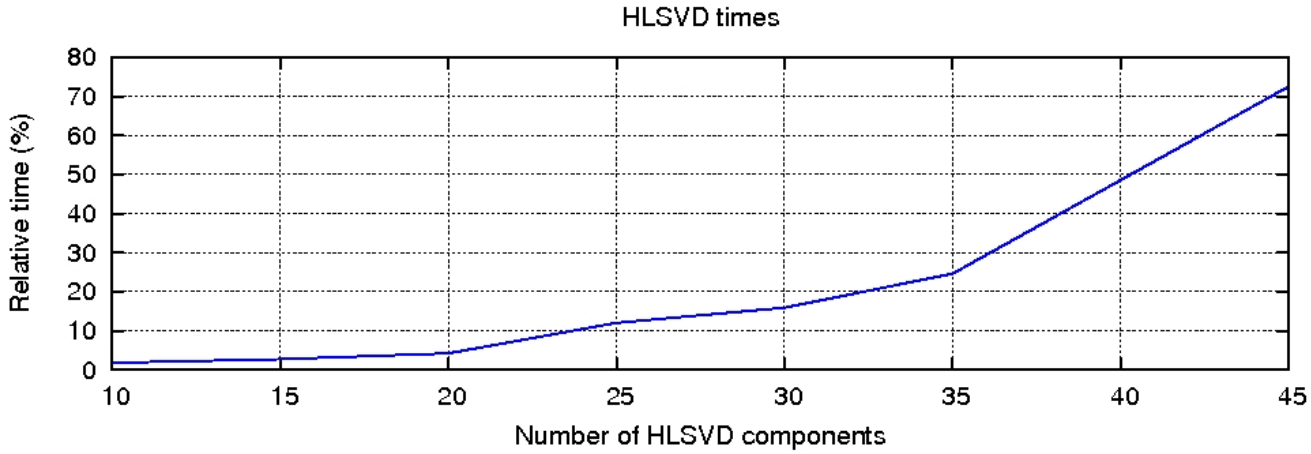


Figure 6: Computational times of HLSVD (in %, relative to the computational times of HSVD) as a function of the number of HLSVD components. The MRS signal, used, was the same as displayed in Figure 5 .

Table I shows the numerical values of the HLSVD computational times, as used for producing Figure 6 , as well as the corresponding number of Lanczos iterations, required for finding the asked number of HLSVD components. The table indicates, that there is a direct relation between the HLSVD computational times and the number of Lanczos iterations.

	Number of HLSVD components						
	10	15	20	25	30	35	45
Computational times (%)	2	3	4	12	16	25	73
Number of Lanczos iterations	100	110	270	620	750	900	1440

Table I: Computational times of HLSVD (in %, relative to the computational times of HSVD) and number of Lanczos iterations as a function of the number of HLSVD components.

C. H(L)SVD and the low-rank problem

In Linear Algebra the rank of a matrix is a measure of its non-degenerateness [12] . Being based on Hankel data matrices of the MRS signal data-points, the H(L)SVD algorithm also have to deal with this rank concept. When testing the two plugins, we have found that low-rank problems may arise, if one wants to determine (many) more H(L)SVD components, than being related to the *signal* singular values. It appears, that this low-rank problem especially can become dominant in cases of signals with a *very low* noise level (if noise is clearly present, the Hankel data matrices usually have *full rank*).

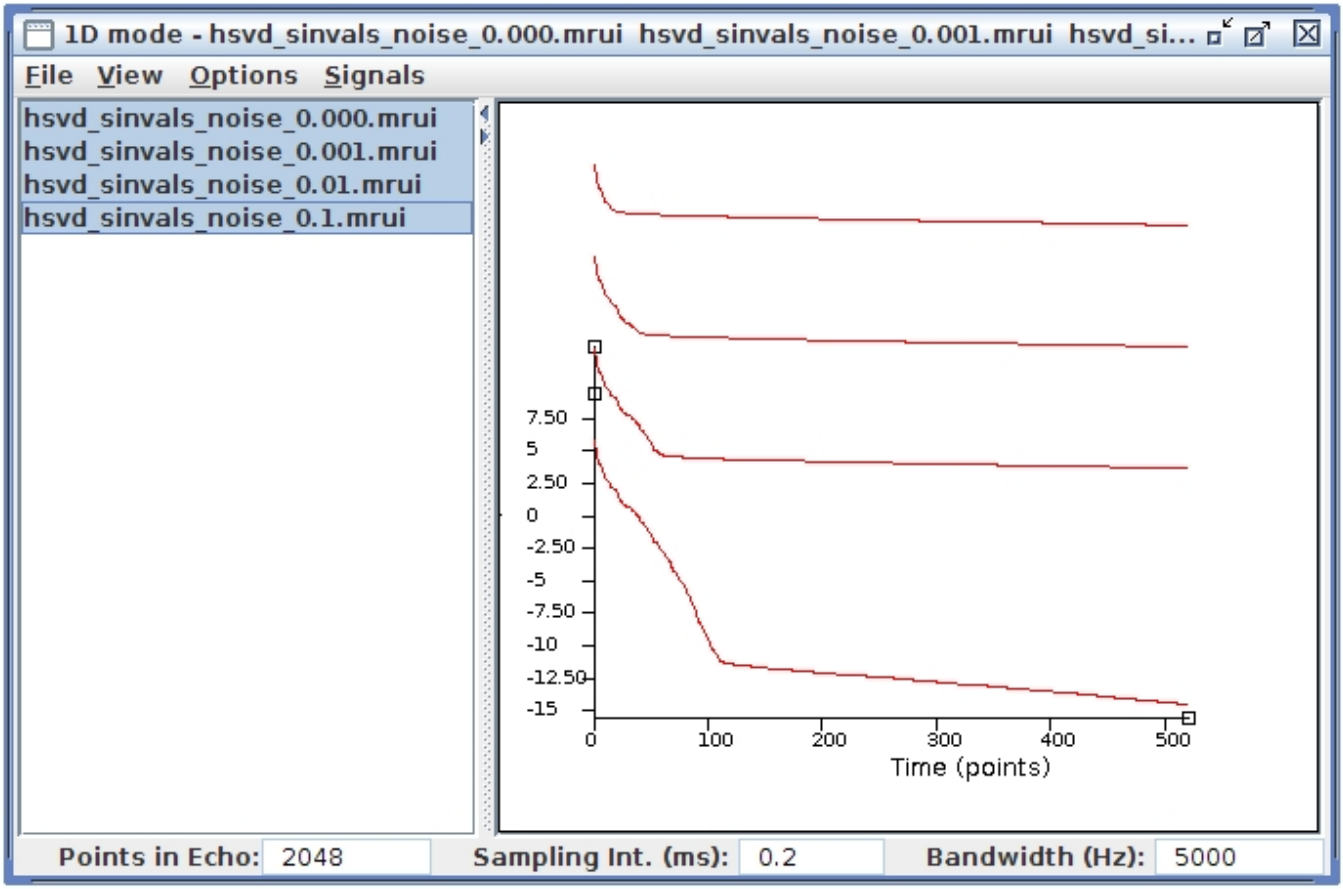


Figure 7: The (logarithm of the) first 500 singular values of the case study of subsection V-A as a function of the standard deviation of the added noise (increases from bottom to top).

To illustrate the issue of *signal-related* singular values, we have plotted in Figure 7 the first 500 singular values of the Hankel data matrix (again based on the HSVD case study of subsection V-A) as a function of the standard deviation of the added noise (sd_noise = 0, 0.001, 0.01 and 0.1 from bottom to top). It shows, that with increasing size of the noise the signal-related singular values can be less clearly separated from the noise-related singular values. As a consequence, the number of signal-related HSVD components, that can be determined, decreases.

We were able to demonstrate a low-rank problem in this case study by choosing ncom = 120 in the HSVD of the noiseless signal (sd_noise = 0). It appeared, that the rank of the Vandermonde matrix (derived from the Hankel data matrix), being used in the Lapack subroutine `zgelss.f` to calculate the parameters of the HSVD components, was only as small as 1, whereas it should be equal to ncom to arrive at a proper HSVD solution.

In the case of HLSVD the low-rank problem may already manifest itself in an earlier stage of the calculation, namely in the Lanczos algorithm. If one chooses a too large value of ncom, this may result into requiring many Lanczos iterations (see again Table I).

D. Backwards reconstruction

If the input field nback is set in the H(L)SVD plugin GUI (see again Figure 1) to a value unequal to zero, the backwards reconstruction functionality is being applied. In this subsection we view an example of this plugin possibility (see Figure 8).

In order to arrive at this result, we have taken the following steps:

1) The *noiseless version* of the signal of subsection V-A was truncated with the *j*MRUI system by removing the first 10 data-points (ndp decreased from 2048 to 2038).
2) This truncated signal was quantified by the HSVD plugin, using a (1018,1021) Hankel data matrix and ncom = 30.
3) The signal was reconstructed, using the parameters of the 30 HSVD components and with additionally extrapolating the reconstructed signal backwards over 10 data-points. This yields again ndp = 2048.
4) In the *j*MRUI-produced Figure 8 the timed domains of the original signal and the reconstructed signal are plotted one over the other, thus giving the opportunity (after zooming) to compare the time domains at the beginning.

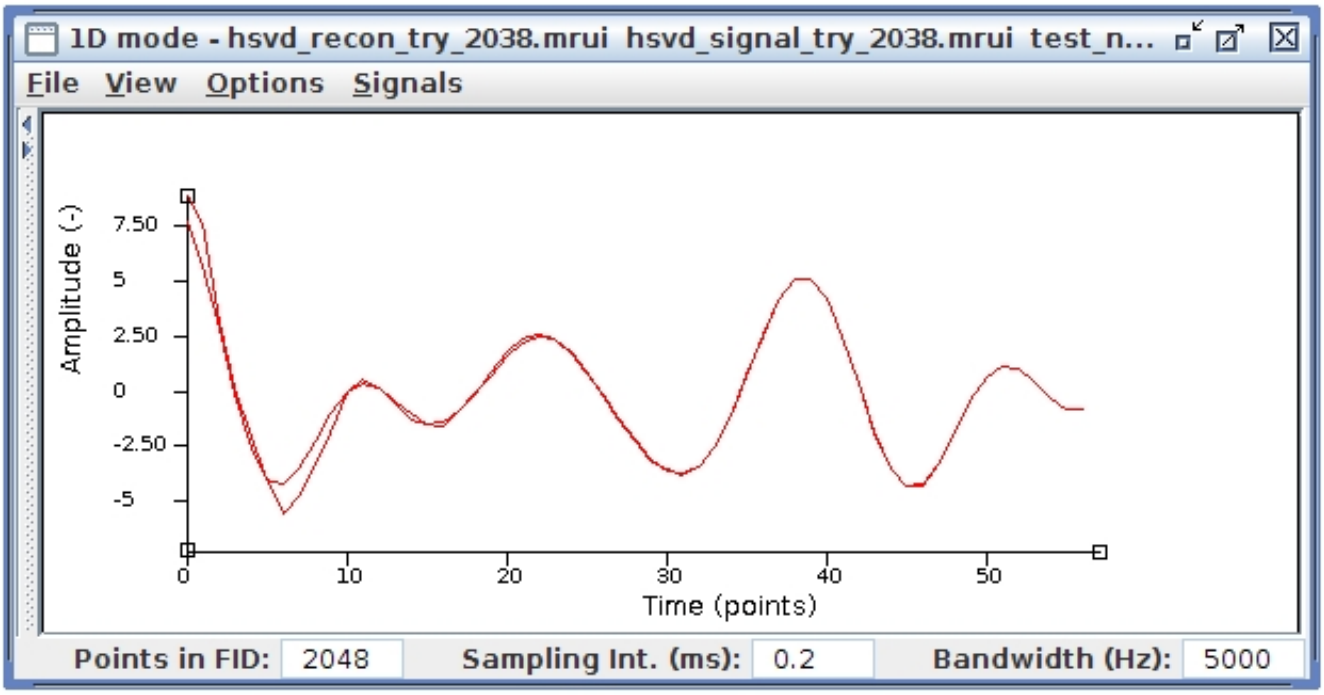


Figure 8: Backwards reconstruction over a time-domain region of 10 points, using the parameters of an HSVD quantification of the *noiseless version* of the signal of subsection V-A . Note the deviation between the true signal and the HSVD-based reconstructed signal at the beginning of the plot (starting at about data-point number 23; see text).

The backwards-reconstruction functionality of the H(L)SVD plugin is in fact *extrapolation*. We like to remark, that numerical methods of extrapolation can lead to erroneous estimations [13]. It has to be handled with care. For this reason we have used the noiseless version of the test signal. With noise, we think it is dangerous to extrapolate, especially over a *large* time range as 10 data-points (a *few* data-points may be acceptable for a good SNR).

Although we have truncated (and then reconstructed) the first 10 data-points, it is clear from Figure 8 , that the true and reconstructed signal already start deviating at around data-point 23. We think, that this is partly due to the fact, that we have used a value of ncom = 30. When looking at the bottom plot of Figure 7 , it can be seen that for the noiseless signal a higher value of the number components should be more appropriate (but be careful with the low-rank problem).

E. Using the H(L)SVD plugin as a filter

In Figure 9 we demonstrate the filter functionality of the H(L)SVD plugin. By choosing in the plugin GUI the input values fmin = 2.14 PPM and fmax = 3.8 PPM, the signal if filtered in the regions from -19.531 to 2.14 PPM and from 3.8 to +19.531 PPM. It should be emphasized, that the region 2.14 to 3.8 PPM is supposed to be unchanged (but be careful for the effects of the H(L)SVD quantification of broad overlapping background signals).

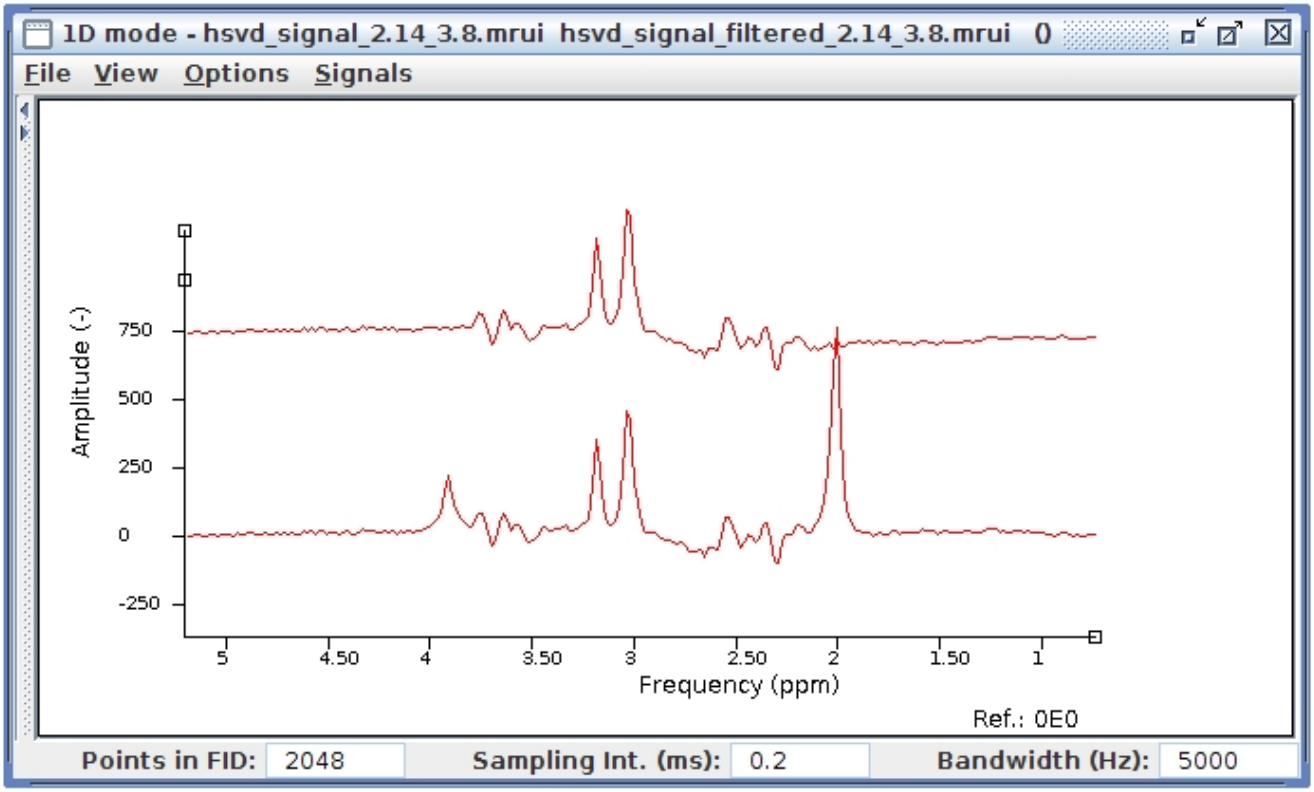


Figure 9: Plot of the filtered signal (top) and the original signal (bottom; signal of Figure 5). The GUI filter input values were set at fmin = 2.14 PPM and fmax = 3.8 PPM.

VI. Conclusions

Summarizing, we like to make the following remarks/conclusions:

1) We have developed HSVD/HLSVD-based plugins for the (TRANSACT supported) *j*MRUI software system, that should be used as research tools in the field of the *in vivo* MRS.
2) The plugins were developed for the Windows and Linux operating system, using the Eclipse Java IDE.
3) We have put effort into making the maximum value of the number of H(L)SVD components as large as possible, given the size of the Hankel data matrix.
4) The progress of the H(L)SVD algorithm is monitored by a Fortran-code produced system call to the Gnuplot graphing utility.
5) Choosing an appropriate value for the number of H(L)SVD components can be facilitated by interactive plots of the singular values. In case of HLSVD this choice for the number of components also strongly affects the computational time of the algorithm.
6) In case of H(L)SVD of MRS signals with a very low level of the noise (especially simulated noiseless signals), one must be aware of the low-rank problem.
7) The plugins offer the functionality of extrapolating a reconstructed H(L)SVD signal backwards (at the beginning of the signal). This must be handled with care. Also, the functionality of H(L)SVD-based filtering of the MRS signals is provided.

Acknowledgement

This work was done in the context of FP7 - PEOPLE Marie Curie Initial Training Network Project PITN-GA-2012-316679-TRANSACT [14].

REFERENCES

[1] WikipediA, the free encyclopedia, "Java Native Interface," http://en.wikipedia.org/wiki/Java_Native_Interface, 2014, JNI is a programming framework that allows Java code to call native applications. 1

[2] H. Barkhuijsen, R. de Beer and D. van Ormondt, "*Improved Algorithm for Noniterative Time-Domain Model Fitting to Exponentially Damped Magnetic Resonance Signals*," *J. Magn. Res.*, vol. 73, pp. 553–557, 1987, Introduction of HSVD to magnetic resonance time domain signals. 1, 2, 3

[3] R. de Beer, D. van Ormondt and W.W.F. Pijnappel, "Quantification of 1-D and 2-D magnetic resonance time domain signals," *Pure and Appl. Chem.*, vol. 64, pp. 815–823, 1992, Introduction of (the Lanczos based) HLSVD to magnetic resonance time domain signals. 1, 2, 3, 5

[4] developer.android.com, "Get the Android SDK," http://developer.android.com/sdk/index.html, 2014, . 1, 2

[5] R. de Beer and D.van Ormondt, "Monte Carlo Modeling for *in vivo* MRS: Generating and quantifying simulations via the Windows, Linux and Android platform," 2015, Report on behalf of the EU-funded TRANSACT project. 2

[6] LAPACK: Linear Algebra PACKage, "zgesvd.f," http://www.netlib.org/lapack/explore-html/d6/d42/zgesvd_8f_source.html, 2015, ZGESVD computes the singular value decomposition (SVD) for GE matrices. 2, 3

[7] Gnuplot, "gnuplot homepage," http://www.gnuplot.info/, 2015, Gnuplot is a portable command-line driven graphing utility for Linux, OS/2, MS Windows, OSX, VMS, and many other platforms. 2

[8] R. de Beer, D. van Ormondt, J.W. van der Veen and D. Graveron-Demilly, "Creating the GammaPress custom plug-in for the *j*MRUI platform. Simulating metabolite basis sets for *in vivo* Magnetic Resonance Spectroscopy," in *Proceedings ICT.OPEN 2011*. Veldhoven, The Netherlands: NWO/STW, 14-15 November 2011, pp. 16 – 20. 5

[9] J.W. van der Veen, D. van Ormondt and R. de Beer, "Simulating Metabolite Basis Sets for *in vivo* MRS Quantification. Incorporating details of the PRESS Pulse Sequence by means of the GAMMA C++ library," in *Proceedings ICT.OPEN 2012*. WTC Rotterdam, The Netherlands: NWO/STW, 22-23 Qctober 2012, pp. 1–6. 5

[10] S.A. Smith, T.O. Levante, B.H. Meier, R.R. Ernst, "Computer Simulations in Magnetic Resonance. An Object-Oriented Programming Approach," *J. Magn. Reson. Series A*, vol. 106, pp. 75 – 105, 1994. 5

[11] J.W. van der Veen, J. Shen, D. van Ormondt and R. de Beer, "Quantifying In vivo 1H MRS in the human brain at 3T with simulated metabolite basis sets. Including details of the in vivo PRESS measurement protocol and MRS-scanner setup and performing MRS line shape analysis based on self-deconvolution," 2013, Report on behalf of the the MRS Core Facility, NIMH, NIH, USA. 5

[12] WikipediA, the free encyclopedia, "Rank (linear algebra)," http://en.wikipedia.org/wiki/Rank_(linear_algebra), 2015, In linear algebra, the rank of a matrix A is the size of the largest collection of linearly independent columns of A (the column rank) or the size of the largest collection of linearly independent rows of A (the row rank). 7

[13] ——, "Extrapolation," http://en.wikipedia.org/wiki/Extrapolation, 2015, In mathematics, extrapolation is the process of estimating, beyond the original observation range. 8

[14] TRANSACT European Union project, "Welcome to Transact!" http://www.transact-itn.eu/, 2013. 9