Vienna University of Technology

Master programme: Technical Physics

Diploma Thesis

Optimization of a low power spectrometer for Total Reflection X-ray Fluorescence analysis

A master thesis submitted in partial fulfilment of the requirements for the degree of Master of Science

in the research group Radiation Physics

at the

Institute of Atomic & Subatomic Physics

Supervisor: Ao Univ.Prof.Dipl.Ing.Dr. Christina Streli

Author: Kraihamer Mathias Matr. Nr.: 1126978

Address: Forellenweg 7 A-5301 Eugendorf

Eugendorf, September 5, 2017
Location, Date

Author

Abstract

Total Reflection X-ray Fluorescence analysis (TXRF) is a very accurate method for chemical analysis of trace elements in the range of ng/g concentrations and pg for absolute masses. The rising demand for small, mobile and also easy to handle devices for such chemical analyses led to the construction of a compact spectrometer by the Atominstitut x-ray laboratory. This spectrometer operates with a 50 W low power x-ray source and a peltier cooled silicon drift detector that does not depend on liquid nitrogen to get cooled. Special beam preparation optics has been attached to the tube to ensure a monochromatic excitation of the sample. The spectrometer can be operated with mains voltage and a mobile computer.

The first part of this thesis includes the description of the spectrometer structure with all its main parts as well as the detailed setting up procedure. Some improvements has been incorporated into the former version of the spectrometer before manufacturing all the components. The spectrometer has been installed on the optical table in the X-ray laboratory to ensure stable operation. One of the improvements is the rough z-and angle adjustment system. This system aids to pre adjust the height and the angle of the tube with its optics to ensure a proper beam alignment. Another improvement concerns the reflex tracing system of the beam. The old version of the spectrometer uses a fluorescence screen, a vacuum feed through and a expensive camera to display the reflex. This system has been replaced by a camera chip covered with aluminium foil at the end of the beam line that displays the reflex. The last task comprised the determination of the detection limits for this new measurement set-up.

The second part of this thesis enfolds the description of the self written software that is capable of performing the deconvolution of TXRF spectra as well as quantify them. The fitting results of the self written software are compared with results from the well established deconvolution software AXIL. The quantification of a certified reference material from the the National Institute of Standards & Technology (NIST) form the conclusion of this thesis.

Kurzfassung

Totalreflexions-Röntgenfluoreszenzanalyse (TXRF) ist eine sehr genaue Methode zur chemischen Analyse von Spurenelementen im Bereich von ng/g Konzentrationen sowie pg für absolute Massen. Die steigende Nachfrage nach kleinen, mobilen und auch leicht handhabbaren Geräten führte zur Entwicklung eines kompakten Spektrometers durch das Atominstitut-Röntgenlabor. Dieses Spektrometer arbeitet mit einer 50 W Niederleistungsröntgenröhre und einem Peltier-gekühlten Silizium-Drift-Detektor, der nicht mit flüssigem Stickstoff gekühlt werden muss. Ein spezielles an der Röntgenröhre angebrachtes Monochromatorsystem gewährleistet eine monochromatische Anregung der Probe. Das Spektrometer kann mit Netzspannung und einem mobilen Rechner betrieben werden.

Der erste Teil dieser Arbeit beschreibt den Aufbau des Spektrometers und erläutert die Verbesserungen im Vergleich zum Vorgänger. Das Spektrometer wurde auf dem optischen Tisch im Röntgenlabor installiert, um einen stabilen Betrieb zu gewährleisten. Das neue grobe z-und Winkeleinstellsystem hilft dabei Höhe und Winkel der Röntgenröhre einzustellen, um eine korrekte Strahlausrichtung zu gewährleisten. Eine weitere Verbesserung betrifft die Anzeige des Reflexes. Die alte Version des Spektrometers verwendet einen Fluoreszenzschirm, eine Vakuumdurchführung und eine teure Kamera, um den Reflex anzuzeigen. Die aktuelle Version arbeitet mit einem Kamerachip am Ende des Strahls, welcher mit einer Aluminiumfolie bedeckt ist und daher nur für hochenergetische Strahlung (Röntgenstrahlung) sensitiv ist. Anschließend wurden die Nachweisgrenzen für diesen neuen Messaufbau ermittelt.

Der zweite Teil dieser Arbeit umfasst die Beschreibung der selbstgeschriebenen Software welche in der Lage ist, TXRF-Spektren zu fitten und auf Basis dessen eine Quantifizierung durchzuführen. Die gefitteten Ergebnisse des selbstgeschriebenen Programms werden mit den Ergebnissen des bewährten Evaluierungsprogramms AXIL verglichen. Die Quantifizierung von NIST1643 im Vergleich zu AXIL und der vom National Institute of Standards & Technology zertifizierte Werte bilden den Abschluss dieser Arbeit.

Acknowledgements

I would first like to thank my thesis advisors Ao.Univ.Prof.Dr.Christina Streli and Ao.Univ.Prof.i.R.Dipl.-Ing.Dr.techn. Peter Wobrauschek of the Institute of Atomic & Subatomic Physics at Vienna University of Technology. The door to Prof. Streli's and Prof. Wobrauschek's office was always open whenever I ran into a trouble spot or had a question about my research or writing. They consistently allowed this thesis to be my own work, but steered me in the right the direction whenever he thought I needed it.

I would also like to acknowledge Dipl.-Ing.Josef Prost of the Institute of Atomic & Subatomic Physics at Vienna University of Technology as the second reader of this thesis, and I am gratefully indebted for his very valuable comments on this thesis as well as the entire working group of the X-ray laboratory for the pleasant working environment and the opportunity for stimulating and instructive conversations.

I would also like to acknowledge the workshop with its head Ing.Herbert Hartmann for the fast and conscientious manufacturing of the components as well as Walter Klikovich for the precise realisation of the mechanical drawings.

Finally, I must express my very profound gratitude to my parents and to my girlfriend for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Author

Mathias Kraihamer

Contents

List of Figures

List of Tables

1 Introduction and Motivation

Total reflection X-ray fluorescence analysis (TXRF) is an analytical method with which trace elements can be detected qualitatively and quantitatively without destruction down to a lower limit of a few picograms. TXRF is used today in many different areas like environmental analysis, analysis of archaeological objects and forensics. The rising demand for small, mobile and also easy to handle devices for such chemical analyses led to the development of a compact spectrometer by the Atominstitut x-ray laboratory.

The desire for a prototype on site to integrate and verify technical improvements led to this master thesis. Earlier versions of this spectrometer has been sold to different laboratories in different countries with great feedback but there is always room for improvement. Attention has been paid to the prealignment of the x-ray tube and the reflex detection system. Another improvement has been made by using a detector with a bigger detection volume.

The motivation to develop a new software for the fitting and quantification procedure arose from the partly very complex dealing with the DOS based program AXIL. To run AXIL on a PC it is necessary to install a DOS emulator like "DOSBox". This emulator solution works but is sometimes very unstable which leads to program crashes. Two other important facts which lead to the new software is the time-consuming quantification procedure on the one hand and the complicated batch fitting on the other hand. Both of this tasks can be done very easily in the new software.

The first part of this thesis (section 2) is a short introduction into physical principles which form the foundation of the spectrometer. Section 3 describes the used hardware and the improvements that has been done during this thesis. The circuit of the whole measurement set-up can be found in section 4. The different tasks for the beam adjustment procedure are described in section 5. Section 6 contains the description of the software as well as specific theoretical derivations that belong to the software, for example detection limits and fit models. The measurement results which has been done during the thesis are collected in section 7.

2 Physical principles

2.1 The Atomic spectrum

In 1859 Gustav Kirchhoff (1824-1887) and Robert Bunsen (1811-1899) found out that atoms of a specific species, emit and absorb light only in certain wavelengths. The unity of these specific wavelengths is called absorption- or emission spectrum of the atom. Conversely, this specific spectrum makes it possible to conclude from existing absorption or emission wavelengths on the corresponding element (spectral analysis).[3]

In 1885, Johann Jakob (1825-1898) Balmer found that the emission spectrum of the the hydrogen atom consists of a series of lines whose wavelengths λ_k obey a simple law. Balmer was able to describe the inverse wavelengths (wave numbers) $\bar{\nu_k} = \frac{1}{\lambda_k}$ by the formula:

$$\bar{\nu_k} = R_y \cdot (\frac{1}{n_1^2} - \frac{1}{n_2^2}) \tag{1}$$

wherein only the integer values $n_1 = 2$ (Balmer series) and $n_2 = 3, 4, 5, ...$ appear. $R_y = 109678 cm^{-1}$ is the so called Rydberg constant which is expressed in $[cm^{-1}]$ in spectrometry, since the wave numbers are measured in $[cm^{-1}]$.[3]

Later on Theodore Lyman (1874-1954) and Friedrich Paschen (1865-1947) found further series $n_1 = 1$ (Lyman series) and $n_1 = 3$ (Paschen series) which obey the same law. A new atomic model introduced by Niels Bohr offers a resonable describtion of this phenomena.[3]

2.2 Bohr's atomic model

In the early 20th century, scattering experiments by Lord Ernest Rutherford (1871-1937) reveal, that the positive charge of an atom is not homogeneous distributed as it was predicted by Joseph John Thomson (1856-1940) in the "Plum pudding" model in 1903. In 1911 Rutherford introduced a new atomic model, in which the positive charged core is very dense and small (about 3000 smaller than the atomic radius) and centred in the middle of the atom, whereas the electrons are distributed around the core to shield the positive charge to explain the neutrality of atoms. He suggests that the electrons orbit the positive core like planets a star. However the model had one very disastrous problem, the law of classical mechanics predict, that an orbiting electron emits electromagnetic radiation (maxwell equations). This loss of energy would lead to a collapse of the electron into the core.[3]

Niels Bohr proposed in 1913 the now called "Bohr model of the Atom". He circumvents the problem of the collapsing electron with his three postulates:

1.) Instead of the infinite possible trajectories around the core, merely special trajectories are possible. Electrons on this trajectories produce no electromagnetic radiation. This are the stationary states of the atom.

2.) The electron can change the state from one stationary state to another which is called a quantum jump. During a quantum jump of different states with different energy, electromagnetic radiation is getting absorbed or emitted. The frequency of the radiation is only depending on the energy difference and not the moving frequency. The relation between frequency and Energy is given by the relation that Max Plank discovered:

$$\Delta E = h \cdot \nu \tag{2}$$

3.) As the orbit radius increases, the transition to classical conditions takes place (correspondence principle).

Niels Bohr theoretically derived the experimental observation of Palmer (equation 1) with the aid of his newly introduced model.[3]

2.3 The electromagnetic spectrum

The electromagnetic spectrum is the entirety of all electromagnetic waves with various wavelength. The spectrum is divided into different energy areas. This areas are historical originated and combine electromagnetic waves with similar properties like the visible light and radio waves (figure 1).

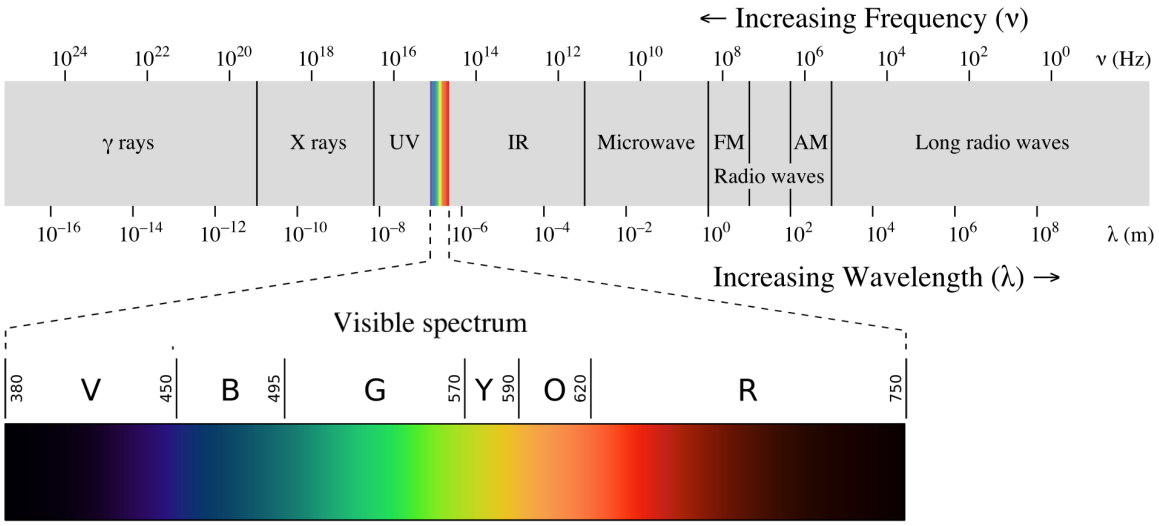


Figure 1: The electromagnetic spectrum [23].

Since the electromagnetic wave propagates in vacuum with the speed of light, the relation between frequency and wavelength is:

$$\lambda \cdot \nu = c \tag{3}$$

With an other point of view, light also behaves like particles called photons. The energy of these photons can be calculated with:

$$E = h \cdot \nu = \frac{h \cdot c}{\lambda} \tag{4}$$

2.4 X-radiation

During his experiments on a discharge tube, Wilhelm Roentgen (1845-1923) discovered 1895 a invisible radiation that permeates matter which is opaque for visible light. Since he did not know much about their nature, he called them X-rays. In 1901 Wilhelm Roentgen receives the first Nobel price for the discovery of x-radiation.[3]
X-radiation is roughly divided into soft- and hard x-rays and located between UV-radiation on the lower energetic side and gamma radiation on the higher energetic side (see fig 2).

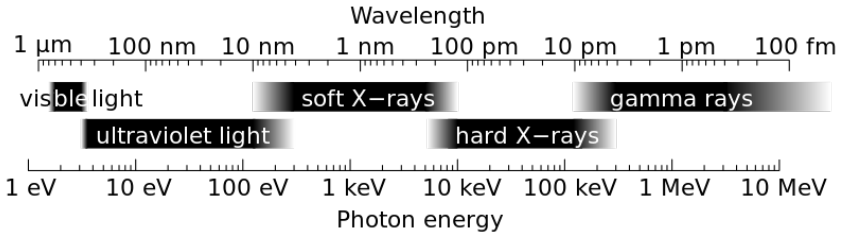


Figure 2: The range of x-radiation [26].

The relation between energy and frequency of the x-radiation can be expressed with equation 4. After inserting the values for h and c, one obtains a compact conversion formula:

$$E[keV] = \frac{12.39}{\lambda[\mathring{A}]} \tag{5}$$

2.4.1 Generation of X-radiation

X-ray radiation can be generated by two different processes:

- the deceleration of energetic electrons (keV-MeV) caused by atomic nuclei in matter produces *Bremsstrahlung* with a continuous spectral intensity distribution.
- *characteristic x-radiation* is generated by electron transitions from higher energy states E_i of heavy atoms (eg. copper, tungsten) into vacant places (holes) in deeper electron shells with energies E_k.

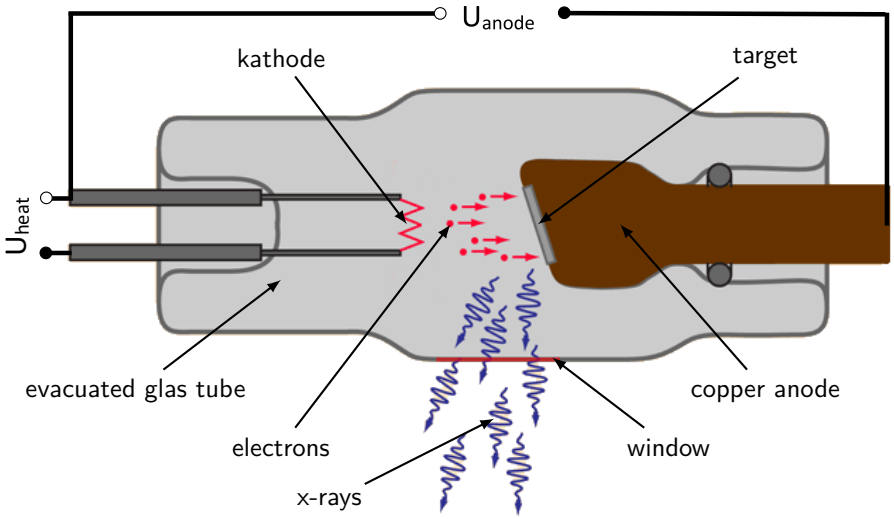


Figure 3: Inner structure of a x-ray tube [8].

Both effects are exploited in the x-ray tube (figure 3) which consists basically of a incandescent filament (cathode) and an anode. Electrons are released at the incandescent filament which is supplied by the filament voltage U_{heat}. The potential difference of U_{anode} accelerates the free electrons from the cathode to the anode where they get strongly decelerated at the target. This process is performed in a vacuum to ensure that the electrons reach the anode without loosing all of their gained energy due to scattering on air molecules. The generated x-radiation leaves the evacuated x-ray tube through a beryllium window which has a very low absorption coefficient for x-rays. Only about 1% of the electrical input energy is converted to x-radiation, the remaining 99% of the energy is heating the anode. The heat must be dissipated in order to avoid overheating. High power x-ray tubes are usually cooled with water, whereas low power x-ray tubes can be cooled with a cooling fan.[20]

A typical x-ray tube spectrum with characteristic lines and the continuous Bremsspectrum can be seen in figure 4.

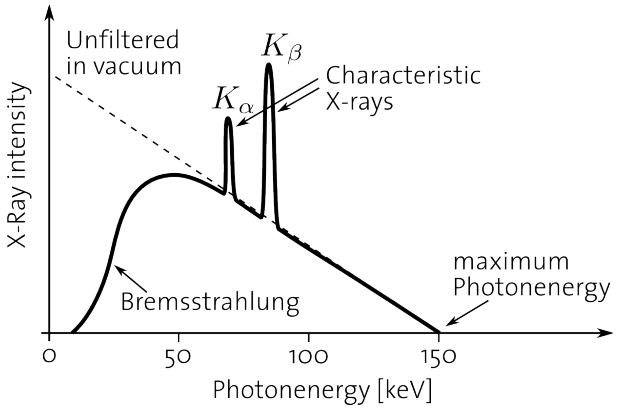


Figure 4: Spectrum of a typical x-ray tube [1].

Further sources of x-radiation are synchrotrons and free electron lasers. However, these are large facilities which are very important in chemical analysis but have no practical application for mobile spectrometers and therefore are not treated in this thesis.

2.4.2 Bremsstrahlung

The electrons impinging at the anode are either deflected in the coulomb field, where they emit a part of their energy $e \cdot U_{anode}$ in form of Bremsstrahlung, or they collide with the electrons of the shells and thus gradually release their energy as heat energy. This results in a continuous spectrum since only a part of the kinetic energy of the electron is converted into radiation. Some collisions are strong enough to remove the electron from a shell which leads to characteristic x-radiation (see chapter 2.4.3).
There is a wavelength limit at the high energy side of the spectrum which corresponds to the conversion of the whole kinetic energy of one electron to one photon. This minimum wavelength can be determined by the Duane-Hunt law:

$$\lambda_{min}[\mathring{A}] = \frac{h \cdot c}{e \cdot U_{anode}} = \frac{12.39}{e \cdot U_{anode}[keV]} \tag{6}$$

This limiting wavelength therefore depends only on the passed acceleration voltage and is independent of the anode material (see figure 5 right). The shape of the spectrum depends on the velocity distribution of the electrons and the used target. The left graph in figure 5 shows a simulated spectrum with different anode voltages and thus different minimum wavelengths.

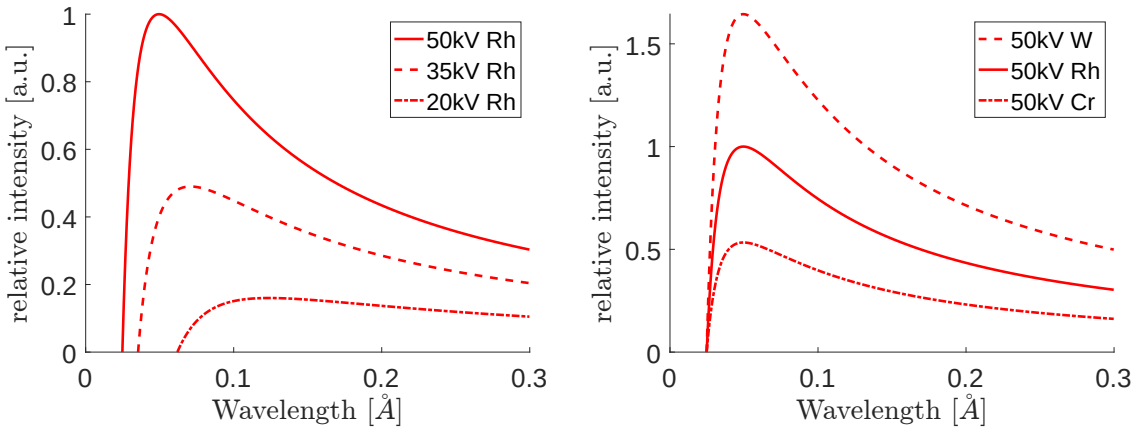


Figure 5: Simulated spectra of Bremsstrahlung under different conditions with the aid of Kramers' law [14].

2.4.3 Characteristic x-radiation

If the kinetic energy of the electrons is high enough, they can also remove electrons from the shells of the anode atoms due to impact ionisation. The holes in the shells are filled by other electrons, producing *characteristic x-radiation*. The energy of the emitted photon can be determined by equation 7 where E_i is the the higher and E_f the lower energy level.

$$E_i - E_f = \Delta E = h \cdot \nu = E_{ph} \tag{7}$$

Each transition leads to a hole in the initial shell, which is filled by an even higher electron. The result is a cascade of photons with different characteristic energies corresponding to the target element.
However, there are forbidden transitions between two certain states. Selection rules are used to determine whether a transition between two states is possible by emission or absorption of electromagnetic radiation. This selection rules can be theoretically derived in quantum mechanics. Figure 6 shows the selection rules and the allowed transitions for the K,L,M and N shell which are the most important in x-ray spectroscopy.

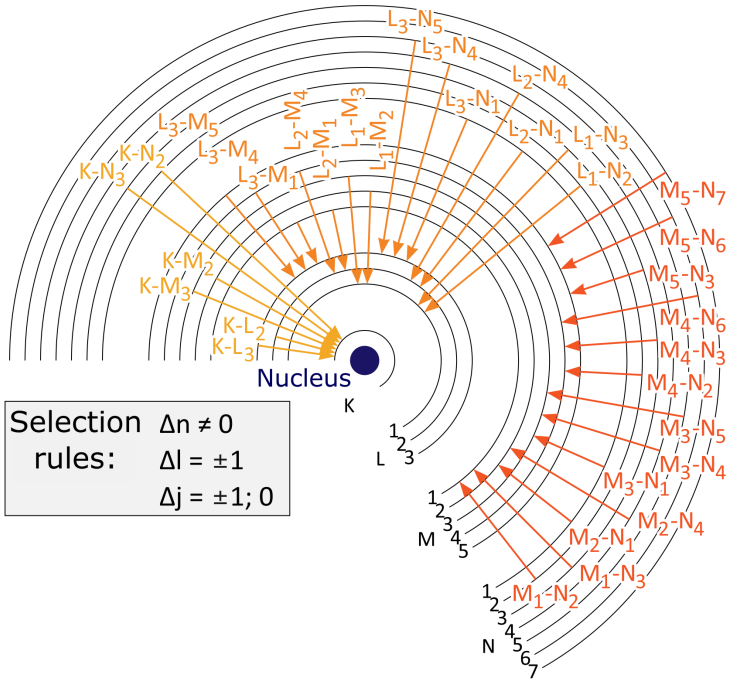


Figure 6: Allowed transitions from a higher to a lower energy level [5].

The transitions can be named in two different ways which are compared in table 1:

Siegsbahn notation

The Siegsbahn notation consists of two main characters. The first one is a capital latin letter K,L or M which describes the final shell of the transition. The second character is a greek letter $\alpha, \beta, \gamma, ..$ which numbers the different intensities consecutively starting with the highest. Additional numbers as well as apostrophes are needed to name the sub line, since newer technology allows higher precision and thus a further differentiation.

IUPAC notation

The IUPAC notation is very reasoned, it consists of two characters separated by a minus. The first character with its index name the final shell of the transition. The second character with its index name the initial shell. The IUPAC notation of the International Union of Pure and Applied Chemistry should replace the traditional Siegbahn notation, but it is still relatively rare.

Final shell	Initial shell	IUPAC	Siegsbahn
K	L_3	$K-L_3$	$K\alpha_1$
K	L_2	$K-L_2$	$K\alpha_2$
K	M_3	$K-M_3$	$K\beta_2$
K	M_2	$K-M_2$	$K\beta_3$
K	N_3	$K-N_3$	$K\beta_2^{'}$
K	N_2	$K-N_2$	$K\beta_2^{''}$
L_3	M_5	L_3-M_5	$L\alpha_1$
L_3	M_4	L_3-M_4	$L\alpha_2$
L_3	M_1	L_3-M_1	Ll
L_2	M_4	L_2-M_4	$L\beta_1$
L_2	M_1	L_2-M_1	$L\eta$
L_1	M_3	L_1-M_3	$L\beta_4$
L_1	M_2	L_1-M_2	$L\beta_5$
L_3	N_5	L_3-N_5	$L\beta_3$
L_3	N_4	L_3-N_4	$L\beta_{15}$
L_3	N_1	L_3-N_1	$L\beta_7$
L_2	N_4	L_2-N_4	$L\gamma_1$
L_2	N_1	L_2-N_1	$L\gamma_5$
L_1	N_3	L_1-N_3	$L\gamma_3$
L_1	N_2	L_1-N_2	$L\gamma_2$
M_5	N_7	M_5-N_7	$M\alpha_1$
M_5	N_6	M_5-N_6	$M\alpha_2$
M_4	N_6	M_4-N_6	$M\beta$
M_3	N_5	M_3-N_5	$M\gamma$

Table 1: Comparison between the Siegsbahn and the IUPAC notation [6].

Moseley law

The Moseley law describes the dependence of the different characteristic lines on the atomic number Z:

$$\frac{1}{\lambda} = k \cdot (Z-\sigma)^2 = k \cdot (Z_{eff})^2 \tag{8}$$

The root of the inverse wavelength $\sqrt{\frac{1}{\lambda}}$ is proportional to the effective atomic number as it is depicted in figure 7. σ is the screening constant which describes the shielding of the nuclear charge by electrons, which are located between the nucleus and the electron in the initial shell. σ as well as k are only depending on the initial and the final shell.

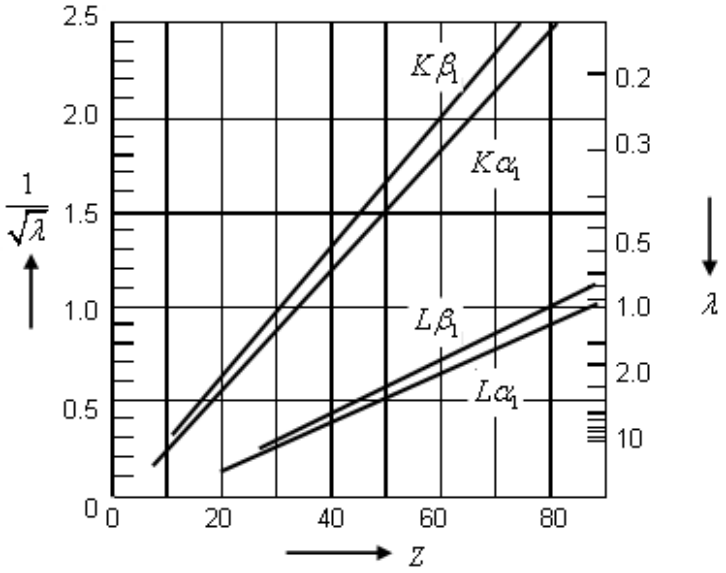


Figure 7: Mosleys law [17].

2.5 X-ray fluorescence analysis

X-ray fluorescence analysis is a non-destructive analytical method which makes it possible to examine samples qualitatively and quantitatively with respect to the chemical composition. The sample with the unknown composition is excited with X-ray radiation, which therefore emits characteristic radiation. This emitted radiation is then measured either directly with a energy dispersive detector (EDXRF) or via an analyzer-crystal and a goniometer (WDXRF).

2.5.1 Wavelength dispersive X-ray fluorescence analysis (WDXRF)

WDXRF is based on the Bragg condition which is illustrated in figure 8. X-rays usually penetrates an irradiated crystal in which the radiation is getting absorbed and partly transmitted if crystal is thin enough.

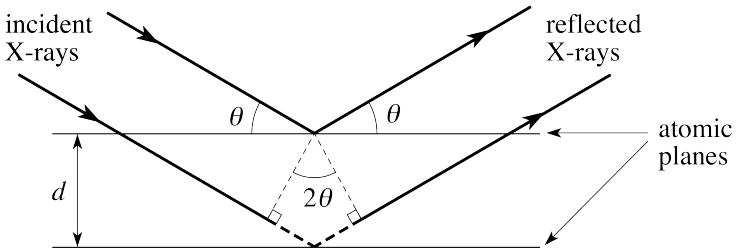


Figure 8: Illustration of the Bragg condition.

However, if the excited radiation is in the range of the spacing between the atoms, interference phenomena will occur. Constructive interference takes place when the wavelength, the atomic distance, and the angle of incidence meet Bragg's law:

$$n \cdot \lambda = 2 \cdot d \cdot sin(\Theta) \tag{9}$$

where Θ is the angle of incidence and d is the atomic distance. This equation states, that every incidence wavelength can get reflected only at one specific angle if only the first order is considered $n = 1$.
A wavelength-dispersive spectrum is then obtained by scanning a specific angle range with a detector. Every angular position of the spectrum corresponds to one particular wavelength. The resolution is dependent on the analyser crystal as well as on the optical design, particularly collimation, spacing and positional reproducibility. The effective resolution may vary from 20 eV in an inexpensive benchtop to 5 eV or less in a laboratory instrument.[20]

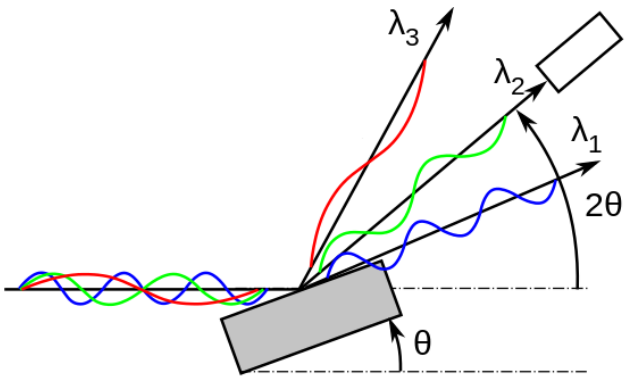


Figure 9: Shematic of a WDXRF system [27].

2.5.2 Energy dispersive X-ray fluorescence analysis (EDXRF)

In EDXRF an energy dispersive detector in combination with a multi-channel analyzer is used to simultaneously collect the fluorescence radiation emitted from the sample. The resolution is dependent on the resolution of the detector, it can vary from 150 eV or less for a liquid nitrogen cooled Si(Li) detector, 150-220 eV for various solid state detectors and 600 eV or more for gas filled proportional counters.[20]

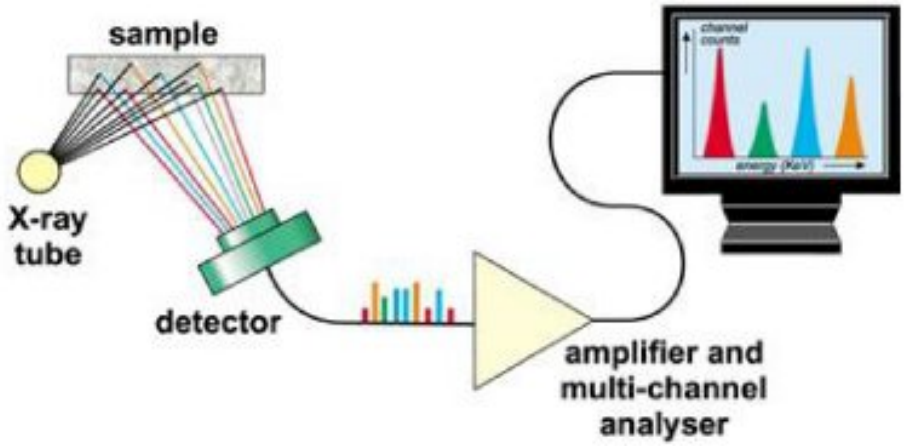


Figure 10: Shematic of a EDXRF system [15].

Table 2 shows a comparison of advantages and disadvantages for the two different measurement set-ups.

WDXRF	**EDXRF**
⊕ higher count rates (good statistics)	⊖ limited max. count rate
⊕ precise quantification	
⊕ excellent energy resolution	⊖ worse energy resolution
⊕ low spectral background	⊖ high spectral background
⊖ sequential multi element measurement	⊕ simultaneously multi element measurement
⊖ low solid angle	⊕ high solid angle
⊖ inflexible (defined sample types)	⊕ flexible set-up
⊖ more expensive	⊕ less expensive
⊖ sample preparation required	⊕ no or only small sample preparation necessary

Table 2: Advantages and disadvantages of WDXRF compared to EDXRF [20].

2.5.3 Monochromatic x-rays

Monochromatic radiation can be achieved with different types of monochromators. The monochromator used in this thesis is a mirror which reflects a specific energy range of the incident x radiation. The ratio $\frac{\Delta E}{E}$, is called the relative spectral resolution of the monochromator where ΔE denotes the energy width and E the energy of photons to be reflected. There are two different monochromator mirrors which can be used in x ray physics, a crystal monochromator mirror and a multilayer monochromator mirror. The foundation of crystal monochromators is bragg's law (see figure 8) while a multilayer monochromator consists of a substrate on which high and low Z layers are applied. The alternating layers of high-Z and low-Z materials create a periodic structure of different electron densities similar to atomic planes in crystals. The individual layers have a thickness of the order of nano meters.

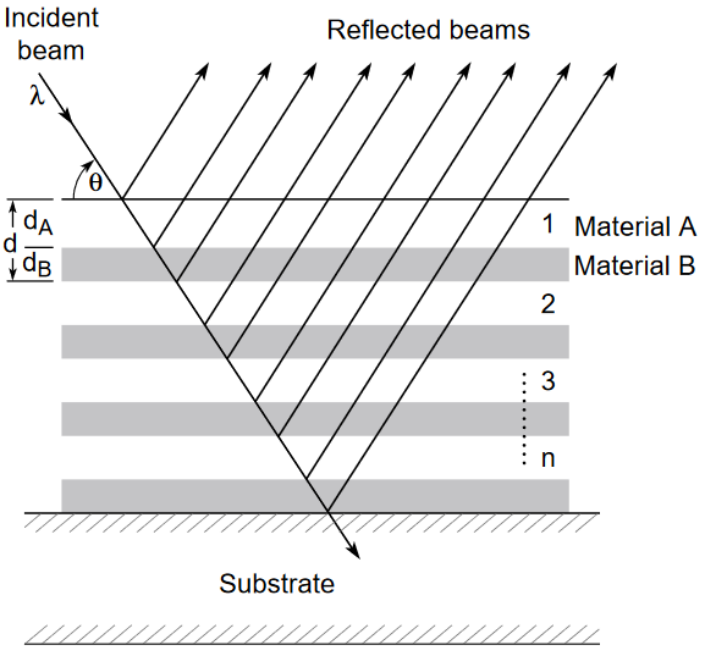


Figure 11: Schematic of a multilayer reflector of n bilayer pairs. The parameters λ ,Θ, and d are chosen to satisfy the familiar Bragg equation, but the relative thicknesses of the high- and low-Z materials are also critical in optimizing reflectivity. The total reflectivity is the vector sum of the complex reflectionm coefficients at each interface, with the different path lengths taken into account. [21][4-2].

The main difference of crystal and multilayer monochromator mirrors is their relative spectral energy resolution. The crystal monochromator has a very small relative spectral energy resolution compared to the multilayer crystal. Figure 12 shows a comparison of the reflectivity of a multilayer and a crystal monochromator mirror.

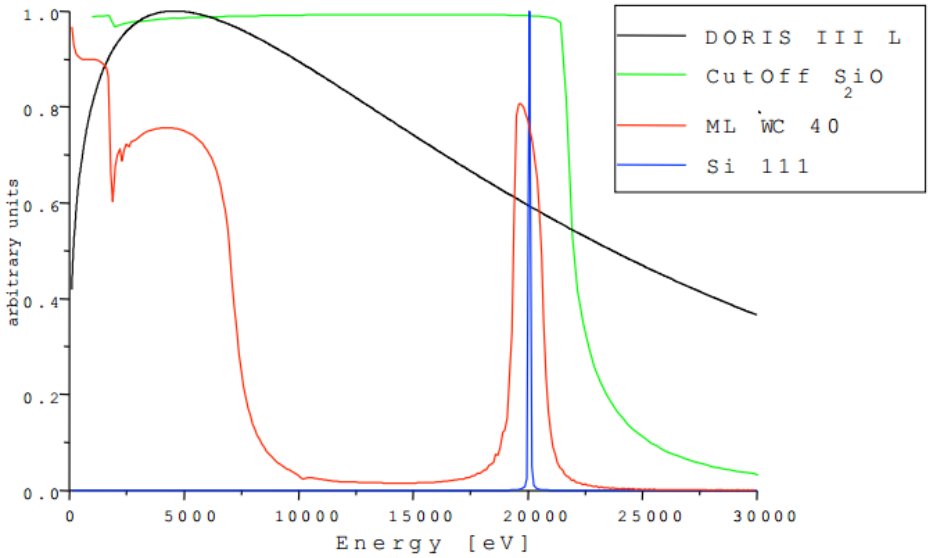


Figure 12: Comparison of reflectivity of multilayer vs. crystal monochromator [7].

The in general broader bandwidth of multilayer mirrors compared to single crystal mirrors makes them more suitable for the spectrometer discussed in this thesis due to the higher amount of reflected photons (area below the blue and red intensity curve).[19]

2.5.4 Detection of x-rays

For WDXRF it is sufficient to use a Geiger-Mueller counter since the energy distinction is made by the analyser crystal.

A EDXRF set-up requires a detector which can distinguish the energy of the photons. Such detectors are based on the inner photoelectric effect and are therefore made of semiconductor materials like silicon and germanium. Semiconductor detectors are working similar to a reverse biased diode. The size of the depletion region depends on the bias voltage, the higher the voltage, the broader is the depletion region. Almost the entire voltage drops inside the depletion region resulting in a high field strength inside this region. Impinging x-rays excite electrons to the conduction band and therefore produce electron-hole pairs. The electrons and the holes are drifting in opposite directions due to the electric field. The electrons are drifting to the positive terminal whereas the holes are drifting to the negative terminal. This electron and hole drift generates a current which is proportional to the energy of the photon since higher energy photons can excite more electron-hole pairs. Since the depletion region, which defines the detection volume, is very narrow for a ordinary reverse biased diode, the efficiency is very low. To circumvent the problem with the narrow depletion region one can add an extra lowly doped silicon layer (i-layer) between a highly doped p and n-side. The applied bias voltage is now dropping over the hole intrinsic layer and increases therefore the depletion region to the width of the i-layer. Such a set-up is called pin-diode and is used for many applications concerning photon detection.

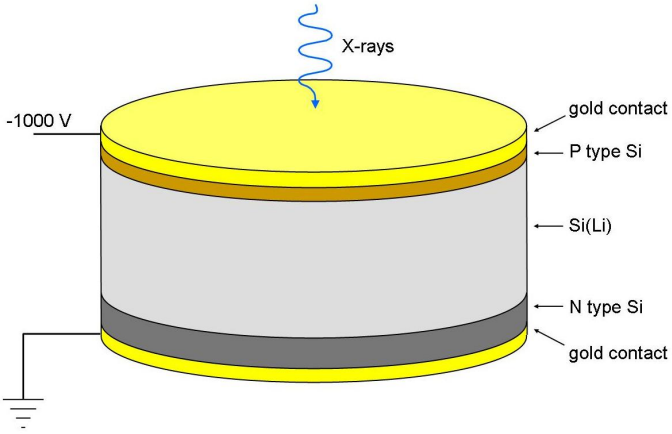


Figure 13: Structure of a Si(Li) detector crystal [25].

A very common EDXRF detector is the so called Si(Li) detector, its typical structure is depicted in figure 13. The Si(Li)-detector is basically a pin-diode with its highly doped p and n material and the intrinsic layer in between which is doped with Li. Si generally has impurities in the ppb range, typically by B. To neutralize the trivalent B, the crystal is doped with the monovalent Li to ensure a high resistance. Gold contacts on both sides of the pin-diode build the junction between conductor and semiconductor.
In order to minimize thermal leakage current and prevent the diffusion of the Li atoms, the detector crystal must be cooled. Such detectors are usually cooled by liquid nitrogen. The detector crystal is embedded in a vacuum housing to prevent dust and humidity from penetration. The entry window of the housing is made of Be, since low-energy radiation is getting well transmitted due to the low order number.

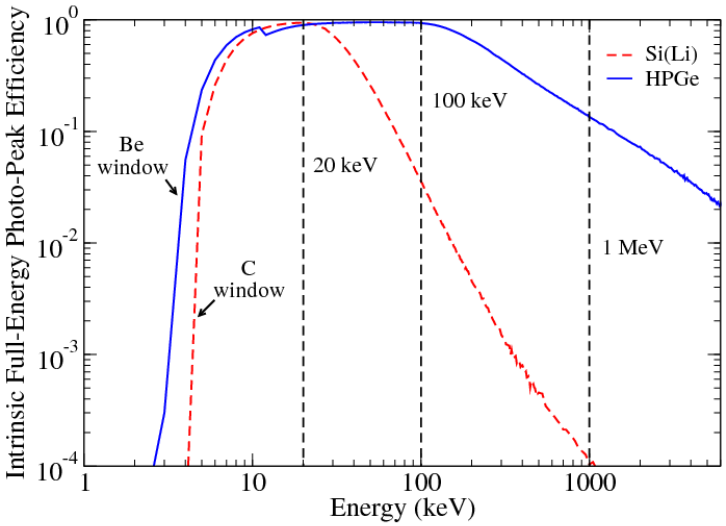


Figure 14: A comparison of the simulated intrinsic efficiencies for a TITAN Si(Li) detector and a HPGe detector [10].

Two very important detector parameter are detector efficiency and the full width half maximum (FWHM) of the detected lines. The detector efficiency indicates the ratio of the detected to the incident photons with respect to their energy. It is dependent on the entry window and the absorption ability of the detector material. The material of the entry window absorbs low energy photons and therefore decreases the efficiency. The absorbtion coefficient and the dimensions of the detector crystal specifies the efficiency

behaviour in the high energy area. Figure 14 shows the efficiency of a Si(Li) and a HPGe (High purity germanium) detector with C and Be window, respectively. The FWHM indicates the width of the peak at the half intensity. Characteristic lines are very sharp (few eV) but the response peak gets broadened due to the intrinsic properties of semiconductors and signal processing. The resolution of a detector is usually indicated by the FWHM of the Mn-Kα Line (5.9 keV), which occur in the Fe-55 decay.

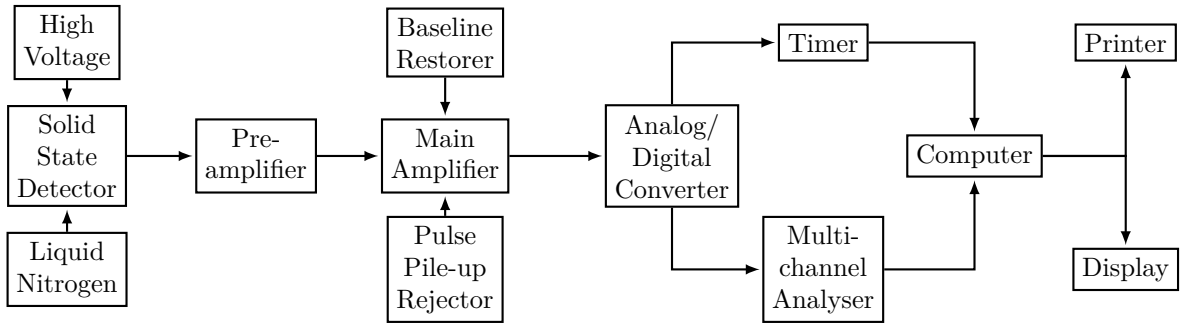


Figure 15: Signal processing scheme [16].

The flow chart in figure 15 illustrates the different steps of signal processing. The pulse caused by the photo-current of the detector is passed to a FET which converts it to a voltage pulse in the milivolt range (pre amplifier). The amplitude of this pulse is proportional to the energy of the incident photon. The **pre amplifier** is usually located close to the detector crystal and also cooled with liquid nitrogen to ensure low leakage currents. The **main amplifier** amplifies the millivolt pulse into the voltage range and shapes it to a constant duration for further treatment. A high shaping time improves the signal-to-noise ratio but it also increases the probability that two pulses overlap (pulse-pileup effect). The **Pulse pile-up rejector** recognizes overlapping pulses and rejects them. This procedure results in a smaller count rate which can be compensated with a higher measurement time. The **Analog/Digital-converter** converts the analogue, amplified and shaped pulse to a digital signal which is then passed to the **Multichannel analyser**. The Multichannel analyser sorts the pulses into channels with respect to the corresponding photon energy and submits the data to a computer. The **spectrum** can be obtained by plotting the number of detected photons as a function of the channel number.[16]

A newer but very commonly used detector is called silicon-drift-detector (SSD). The SSD is based on the same principle as the Si(Li) where photons generate electron-hole pairs within the depletion region which then drift to the positive and negative terminal, respectively. The difference to the Si(Li) detector is the design of the depletion region. Figure 16 shows the structure of such a SSD. The SSD is based on a 0.3-0.5mm thick n doped silicon wafer [16]. The side of the detector crystal which is faced to the incident x-rays is homogeneously p+ doped. The doping structure of the opposite side is made up of annular p+ doped rings. The SSD is biased with a negative voltage on both sides to generate the depletion region in the n doped volume. The voltage of the p+ ring structure is stepwise decreasing towards the center. The transversal field generated by the applied voltage causes electrons to drift to a small collection electrode in the center. The holes are drifting to the p+ doped electrodes.

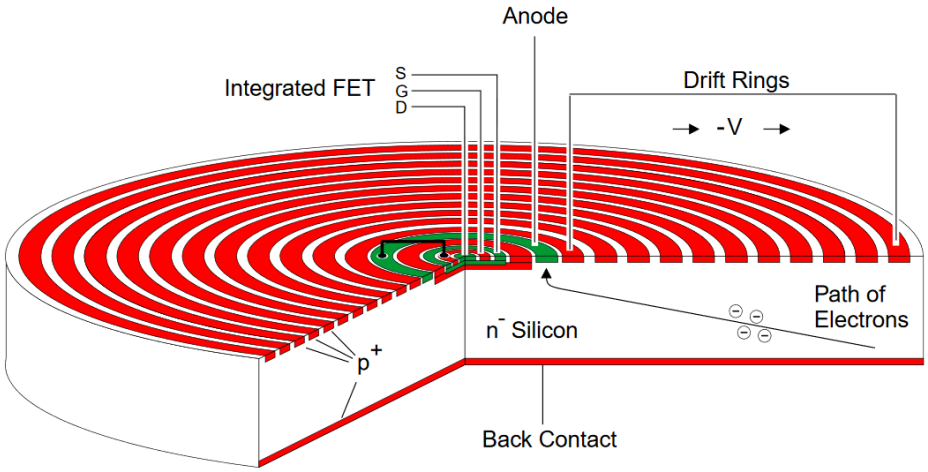


Figure 16: Shematic of a silicon drift detector [4].

The radiation-sensitive volume is smaller than in conventional semiconductor detectors, resulting in a lower efficiency with energies above 20 keV. The noise of the output signal is significantly smaller due to the small detector volume and the integrated FET-pre-amplifier. Because of the lower noise, it is sufficient to cool the detector to about -20° C which is done by small Peltier elements.
Due to the very small central collecting anode, the capacitance is substantially lower compared to conventional detectors, which results in significantly higher maximum counting rates (100,000 to 1 million cps) and thus a shorter measuring times due to a lower dead time.[16]

2.5.5 Quantitative analysis

The qualitative analysis of a spectrum is relatively simple. At least two known lines are needed to calibrate the energy axis. The known lines should be far apart to improve the accuracy. The elements in the sample can then be obtained by comparing the energy value of the individual unknown peaks with tabulated values. Note that different lines may have similar energies so smaller lines can be hidden by bigger lines.
However, the quantitative analysis is more difficult. Figure 17 depicts the geometry for the theoretical derivation of equation 10.

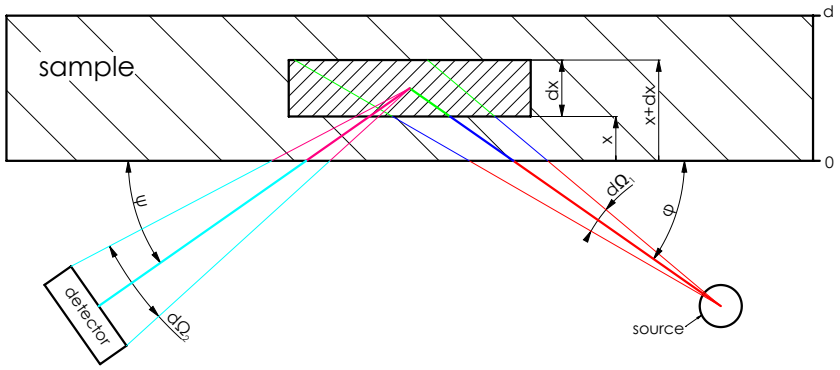


Figure 17: Geometry for the derivation of the primary fluorescence intensity.

The red part between source and sample represents the intensity of the x-ray source with its solid angle. The intensity loss through the sample to the active region is represented by the blue part according to Beer-Lambert's law. The green part between x and $x + dx$ is getting excited and emits fluorescence radiation. The intensity loss through the sample from the active region to the surface of the sample is represented by the violet part. The last to factors in the equation represent the absorption between sample and air and the detector efficiency for the characteristic energy photons.

With these considerations one obtains the formula for the energy dependent intensity of a specific line $I(E_{i,j,k})$:

$$I(E_{i,j,k}) = \int_{E_{abs}}^{E_{max}} dE \int_{0}^{d} dx \int_{\Omega_1} \frac{1}{4\pi r_1^2} \cdot d\Omega_1 \int_{\Omega_2} \frac{1}{4\pi r_2^2} \cdot d\Omega_2 \cdot I_0(E) \cdot exp\left(-\left[\left(\frac{\mu}{\rho}\right)_E \cdot \frac{x}{sin(\phi)} \cdot \rho_s\right]\right) \cdot$$
$$c_i \cdot \frac{\rho_s}{sin(\phi)} \cdot \left(\frac{\tau_k}{\rho}\right)_{i,E} \cdot \omega_{i,j} \cdot p_{i,j,k} \cdot V_i(E) \cdot exp\left(-\left[\left(\frac{\mu}{\rho}\right)_{E_{i,j,k}} \cdot \frac{x}{sin(\psi)} \cdot \rho_s\right]\right) \cdot f(E_{i,j,k}) \cdot \epsilon(E_{i,j,k}) \tag{10}$$

with:

i ..... element i
j ..... final shell
k ..... initial shell
r_1 ..... distance between sample and source
r_2 ..... distance between sample and detector
$I(E_{i,j,k})$ ..... intensity of a specific line
E_{max} ..... maximum excitation energy
E_{abs} ..... energy of the absorption edge of element i and shell k
d ..... thickness of the sample perpendicular to sample surface
Ω_1 ..... solid angle of the source
Ω_2 ..... solid angle of the detector
$I_0(E)$ ..... intensity of the source in dependence of the energy
$\left(\frac{\mu}{\rho}\right)_E$ ..... mass attenuation coefficient in dependence of the energy
ϕ ..... angle of the incident radiation
ρ_s ..... density sample
c_i ..... concentration of the element
$\left(\frac{\tau_k}{\rho}\right)_{i,E}$ ..... photoelectric mass absorption coefficient of element i and shell k
$\omega_{i,j}$ ..... fluorescence yield of the shell j
$p_{i,j,k}$ ..... emission probability for the initial shell k
$V_i(E)$ ..... factor that takes secondary excitation into account
$\left(\frac{\mu}{\rho}\right)_{E_{i,j,k}}$ ..... mass attenuation coefficient of the Energy $E_{i,j,k}$
ψ ..... take-off angle of the fluorescence radiation
$f(E_{i,j,k})$ ..... absorption of the fluorescence radiation between sample and detector
$\epsilon(E_{i,j,k})$ ..... sensitivity of the detector

The characteristic line intensity $I(E_{i,j,k})$ is dependent on many different factors, in particular of the element i and the shells j and k. Terms depending only on physical properties of the atoms are called Fundamental Parameters.

After the integration with respect to x, equation 10 becomes:

$$I(E_{i,j,k}) = \int_{E_{abs}}^{E_{max}} dE \int_{\Omega_1} \frac{d\Omega_1}{4\pi r_1^2} \int_{\Omega_2} \frac{d\Omega_2}{4\pi r_2^2} \cdot I_0(E) \cdot A(E) \cdot c_i \cdot \frac{\rho_s \cdot d}{sin(\phi)} \cdot \sigma_{i,j,k} \cdot V_i(E) \cdot f(E_{i,j,k}) \cdot \epsilon(E_{i,j,k}) \tag{11}$$

with the total absorption factor:

$$A(E) = \frac{1 - exp\left(-\left[\left(\frac{\left(\frac{\mu}{\rho}\right)_E}{sin(\phi)} + \frac{\left(\frac{\mu}{\rho}\right)_{E_{i,j,k}}}{sin(\psi)}\right) \cdot \rho_s \cdot d\right]\right)}{\left[\frac{\left(\frac{\mu}{\rho}\right)_E}{sin(\phi)} + \frac{\left(\frac{\mu}{\rho}\right)_{E_{i,j,k}}}{sin(\psi)}\right] \cdot \rho_s \cdot d} \tag{12}$$

and the fluorescence cross section:

$$\sigma_{i,j,k}(E) = \left(\frac{\tau_k}{\rho}\right)_{i,E} \cdot \omega_{i,j} \cdot p_{i,j,k} \tag{13}$$

2.5.6 Monochromatic excitation and thin film approximation

The equation can be further simplified, if the sample is very thin and excited with monochromatic radiation, as it is the case with TXRF. The integration with respect to the energy is not necessary since there is only one excitation energy. Absorption and secondary excitation do not occur since the total absorption factor becomes 1 because $\lim_{x \to 0} \frac{1-exp(-x)}{x} = 1$.
If monochromatic radiation is used for the excitation and the thin-film approximation is applied, equation 11 becomes:

$$I(E_{i,j,k}) = \int_{\Omega_1} \frac{d\Omega_1}{4\pi r_1^2} \int_{\Omega_2} \frac{d\Omega_2}{4\pi r_2^2} \cdot I_0(E) \cdot c_i \cdot \frac{\rho_s \cdot d}{sin(\phi)} \cdot \sigma_{i,j,k} \cdot f(E_{i,j,k}) \cdot \epsilon(E_{i,j,k}) \tag{14}$$

The intensity $I(E_{i,j,k})$ is proportional to the element concentration c_i which is valid only in this special case. The relation between intensity and concentration is called absolute sensitivity S_i:

$$S_{i,j,k} = \frac{I(E_{i,j,k})}{c_i} \tag{15}$$

The relative sensitivities are obtained by dividing the absolute sensitivities of the respective elements by the absolute sensitivity of a reference element. The measuring conditions must always be the same since the sensitivity depends on factors like voltage, geometry and the applied optics.
The relative sensitivity S_i^{rel} with respect to a internal standard s is defined as:

$$S_i^{rel} = \frac{S_{i,j,k}}{S_s} \tag{16}$$

with:
$S_{i,j,k}$ ..... absolute sensitivity of the characteristic line i,j,k
S_s ..... absolute sensitivity of the internal standard

The unknown concentration for the element of interest can than be calculated by:

$$c_i = \frac{I(E_{i,j,k})}{I_s} \cdot \frac{1}{S_i^{rel}} \cdot c_s \tag{17}$$

with:
c_s ..... is the concentration of the internal standard
$I(E_{i,j,k})$ ..... intensity of the specific line
I_s ..... intensity of the internal standard
S_i^{rel} ..... relative sensitivity

$I(E_{i,j,k})$ and I_s can be obtained by fitting the acquired spectrum, c_s is known and the relative sensitivity S_i^{rel} can be determined with standard samples. The fitting procedure including background estimation and axis calibration as well as the quantification for unknown samples can be seen in section 6.

2.5.7 Total Reflection X-ray Fluorescence analysis (TXRF)

The total reflection x-ray fluorescence (TXRF) is a analytical method based on the effect of total reflection, as the name implies. A typical TXRF set-up can be seen in figure 18. The samples for TXRF are generally liquid solutions that are applied to special sample holder made of quartz (SiO_2) or silicon wafers.

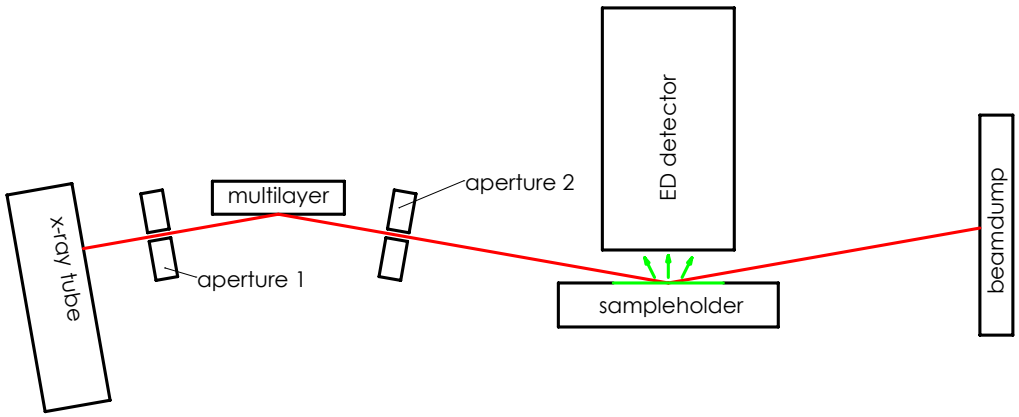


Figure 18: Geometry of a TXRF-system.

A x-ray tube generates x-rays with an energy corresponding to the characteristic lines of the anode and a continuous spectrum (bremsstrahlung). The x-rays are collimated by aperture 1. A multilayer crystal after aperture 1 is used to select the desired wavelength. Aperture 2 is used to further narrow the selected wavelength (see chapter 5). The collimated and monochromatic x-ray beam is getting total reflected at the sample holder. The excitation of the sample and consequently the count rate is therefore twice as high because the photons penetrate the sample twice. Since the angle of total reflection is very small, the detector can be placed very close to the sample which increases the solid angle and therefore further increases the count rate for this set-up.

Total Reflection of X-rays

The refractive index of x-rays is given by [20]:

$$n = 1 - \delta + i\beta \tag{18}$$

where δ and β are associated with dispersion and absorption respectively. The dispersion constant $\delta \approx 10^{-6}$ is depending on the wavelength of the incident radiation and various material constants. The absorption constant $\beta \approx 10^{-8}$ is depending on the mass attenuation coefficient of the material and thus also on the energy of the incident radiation.[20]
Similar to light optics one can determine the angle of total reflection with the aid of snells law:

$$\Theta_c \approx \sqrt{2\delta} \propto \frac{1}{E}\sqrt{\rho} \tag{19}$$

Refraction no longer occurs below the angle of total reflection Θ_c, the surface acts as an ideal mirror and the penetration depth is a few nm (figure 19). The critical angle is depended on the energy of the incident beam and the material.

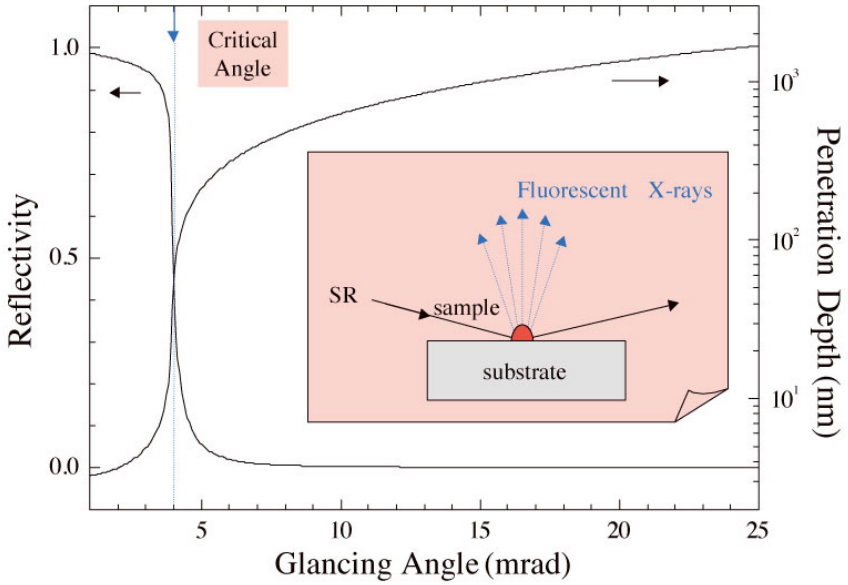


Figure 19: Reflectivity and penetration depth for Si substrate, 8keV x-rays [11].

The intensity of the fluorescence radiation depends not only on the angle of incidence, but also on the shape of the sample. Figure 20 shows the fluorecence intensity depending on whether it is a thin surface layer, a residue on the surface or a bulk material.

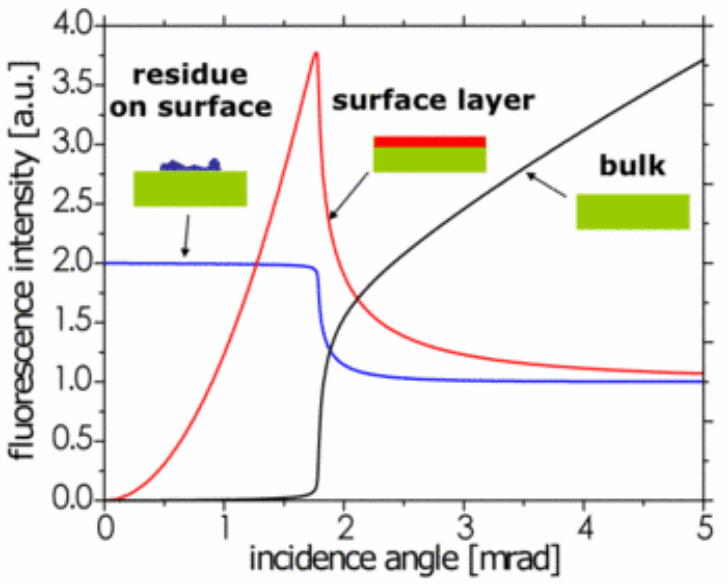


Figure 20: Intensity dependent on the angle of incidence and the sample shape [20].

Sample preparation

Figure 21 shows the different steps of the sample preparation process. The steps of the sample preparation in the TXRF are shown in figure 2.26. For the quantification of unknown samples, it is important to add an internal standard to the sample solution before the analysis. A clean sample holder ensures that the measurement is not falsified.

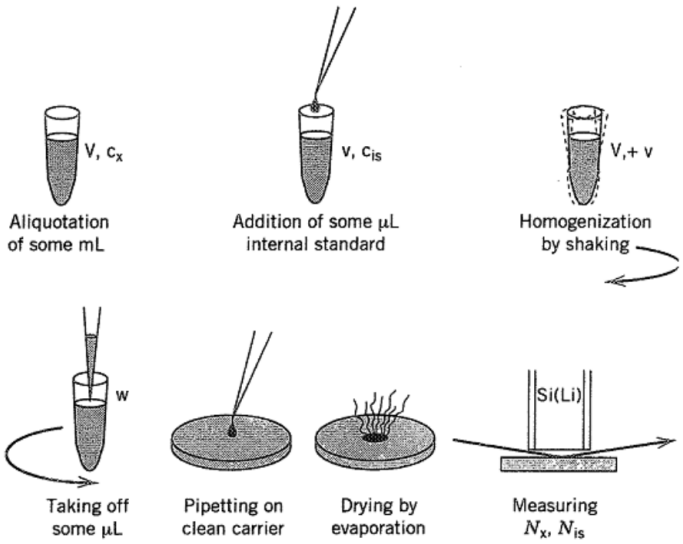


Figure 21: Fluid sample preparation procedure [13].

The procedure of a previous work has been applied for the production of clean sample carriers (blanks):

The contaminated sample carriers are placed on an absorbent paper towel. Subsequently a few drops of the alkaline solvent Decon 90 are added to each sample carrier. The surface can be cleaned with a clean wipe if there are still visible impurities left on the surface of the sample carrier.

After a few minutes the reflectors are washed with tri-distilled water and acetone. If the sample carriers were not very heavily contaminated (e.g. blanks, which are located in the laboratory air for a longer time), it is sufficient to clean the sample carriers with acetone and measure them to check the purity.

The next step is to put the samples in a Teflon holder and heat them in dilute nitric acid (HNO_3) to about $60°C$ for about 15 minutes. Then, the samples are again rinsed with tri-distilled water and acetone.

The cleaned reflectors must be measured with a spectrometer before usage to ensure cleanliness. The reflectors must be cleaned once more if contaminants can be detected in the spectra.

Since the surface of quartz reflectors is hydrophilic, the drops are very easily dissipated. One can make the surface hydrophobic by applying 10 μl of a silicone oil (Serva) to the reflector.[16]

3 Hardware set-up

3.1 Introduction

The foundation of the following hardware set-up is the current total reflection x-ray fluorescence spectrometer "WOBISTRAX" [28] which is capable of analysing high Z and low Z elements with two different easy exchangeable x-ray tubes. The two low power tubes are air cooled with a Cr- and Rh-anode, respectively. Cr-$K\alpha$ radiation is used to excite the K-shells of the elements C to Ti while the Rh-$K\alpha$ radiation excites the K-shells from K to Mo and the L-shells up to U. To avoid absorption and scattering effects of low Z photons in air as well as suppress the appearance of Ar-K peaks it is necessary to evacuate the spectrometer chamber before measuring. To simplify measurements a sample rotator is installed in the chamber which provides space for 12 sample holders.

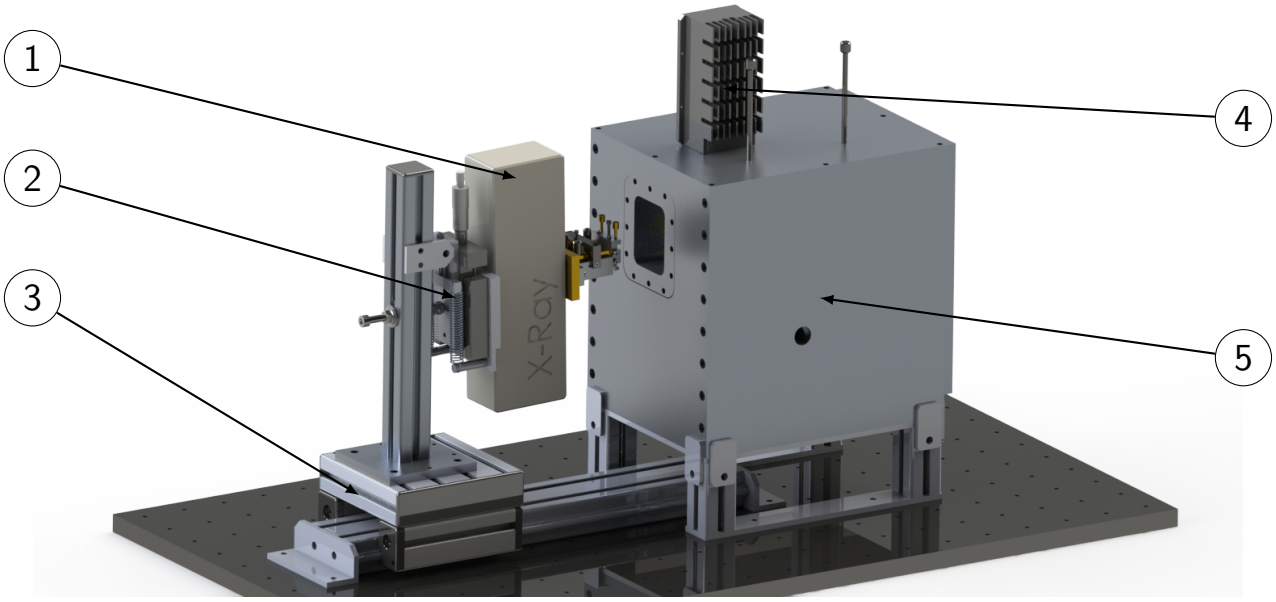


Figure 22: A rendered picture of the spectrometer set-up.

The aim of this thesis was to revise and build such a set-up in the x-ray lab located in the basement of the Institute of Atomic & Subatomic Physics. Contrary to the description above only the Rh-tube is installed because the Cr-tube is not purchased yet. An additional Cr-tube can be installed later on with minimal changes in the construction. Figure 22 shows a rendered picture of the revised spectrometer set-up. The spectrometer consists of the following parts which are described below:

- low power Rh-tube with attached optics (1)
- mechanism for a rough tube adjustment of tilt angle and height (2)
- carriage to move the tube in x-direction (3)
- silicon drift detector with its control box (4)
- vacuum chamber with all its inner workings (5)
- control box for the tube (CSU)
- motor controller box to drive the two servomotors

The whole measurement set-up is placed on a optic table to avoid disturbances through vibrations. The casing is mounted fix on the table whereas the x-ray tube must be slide-able for adjustment purposes (see section 5) and is therefore mounted on a carriage that can be translated in x-direction.

3.2 The x-ray tube with its optics

The core element of this set-up is a x-ray tube of the series iMOXS-MFR developed by IfG - Institute for Scientific Instruments GmbH. The iMOXS-MFR is a brilliant low-power microfocus x-ray source for x-ray fluorescence spectrometers. The tube is operated at the maximum power of 35W with a operation voltage of 50kV and a current of 700mA. High voltage and tube filament current are provided by the control and power supply unit (CSU, see section 3.3). More Information about the iMOXS-MFR is provided in the appendix. The X-ray tube is embedded in a aluminium housing with a fan for cooling and a safety shutter. The pre-installed filter wheel with four positions has been removed.
A sophisticated system of two apertures and a adjustable multilayer (goniometer) has been attached to the beam outlet of the tube to prepare the beam for measurement purposes. Figure 23 and 24 show the attached beam optics.

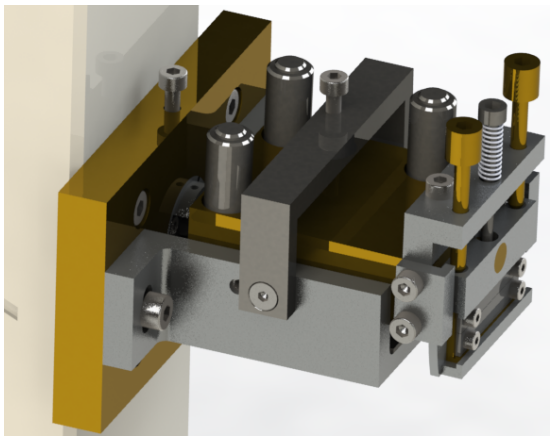


Figure 23: The goniometer and aperture 2 with adjustment screws.

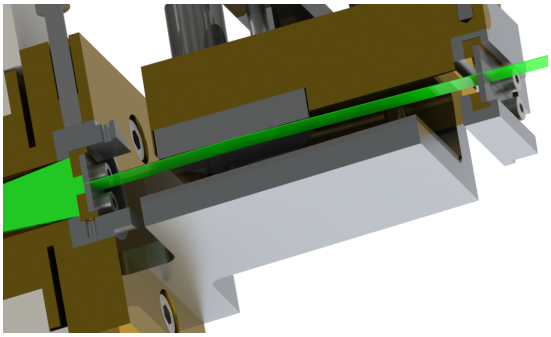


Figure 24: Cross-section of the attached beam optics with beam (green).

Figure 23 shows the adjustable second aperture and the block with three attached micrometer callipers to adjust the angle of the multilayer. The screw in the yoke on the top of the multilayer block is used to fix the current position of the multilayer block. The position and the angle of the second aperture can be adjusted with the two golden screws on the top of it.
Figure 24 shows a cross-section of the goniometer system. The beam (green) impinges on the first aperture where it is getting collimated. This aperture has to be adjusted to obtain the highest intensity. Thereafter the beam is getting reflected at the multilayer. The second aperture is used to select the correct wavelength (reflected Rh-$K\alpha$) since the beam is not exactly parallel and therefore a small wavelength range fulfil the Bragg-condition. The procedure for optimal adjustment is mentioned in section 5 below.

Technical parameters of the multilayer

The multilayer has been produced by the company AXO DRESDEN GmbH with the following parameters:

multilayer material	Pd/B4C
layer thickness	$d = 3,2nm \pm 0.2nm$
ratio of the single layers	$\Gamma \approx 0,33$
substrate	fused silica
diameter	$1''$
thickness	$0.25''$

Table 3: Parameters of the multilayer.

3.3 The x-ray supply

The CSU (fig. 25) is a compact electronic power supply unit for X-ray sources developed by IfG - Institute for Scientific Instruments GmbH. It includes high voltage supply, regulated filament power supply, interlock input, USB interface and connection for tube protection housing including shutter control. It supports a wide range voltage input 90-260VAC and a frequency of 50 or 60Hz at a power of 150W. The device can be used manual in the stand-alone mode or via remote control by software.[9] More Information about the CSU is provided in the appendix.

Figure 25: The CSU unit for the x-ray source [9].

3.4 Rough Z- and angle-adjustment

The beam must reach the sample almost horizontally, so the x-ray tube including goniometer 1 must be inclined backwards twice the angle of the Bragg-reflex. This angle is about 1.5° for the Rh-tube and 4° for Cr-tube depending on the used multilayer. The adjustment component (fig. 26) offers a continuously adjustable inclination angle and a step less height adjustment of the beam outlet. The metal lug on both sides of the aluminium profile can be used for rough height adjustment since the Z-carriage has a range of only 25mm. The angle can be adjusted with the screw. Turning the screw clockwise will increase the inclination angle, counter-clockwise will decrease it. The two pull-springs are necessary to support the weak spring in the Z-carriage because it is designed for horizontal usage. This part of the measurment set-up is not mounted yet but it will be needed when the additional Cr-tube is used in this setup.

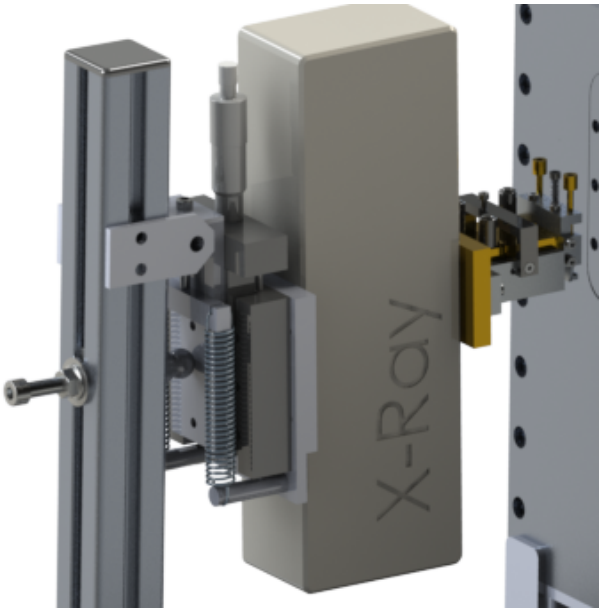


Figure 26: The Z- and angle-adjustment system.

3.5 Carriage in x-direction

The static part of the translation system in x-direction consists of a 40x120 ITEM-profile with a shaft on the left and right side. This profile is mounted on the optic table as mentioned before. The moveable part consists of an 160x40 ITEM-profile with two roller guides.

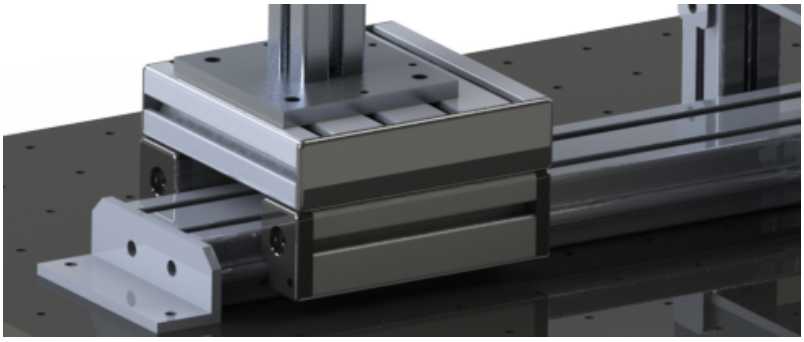


Figure 27: Carriage in x-direction.

3.6 The detector

The centrepiece of this set-up is a detector of the series AXAS-M developed by KETEK GmbH. The detection area of the detector amounts to $100mm^2$. The energy resolution of the detector depends on the shaping time and is in the range of 126 to 136eV at 5.9keV. The Be detector window is $25\mu m$ thick.

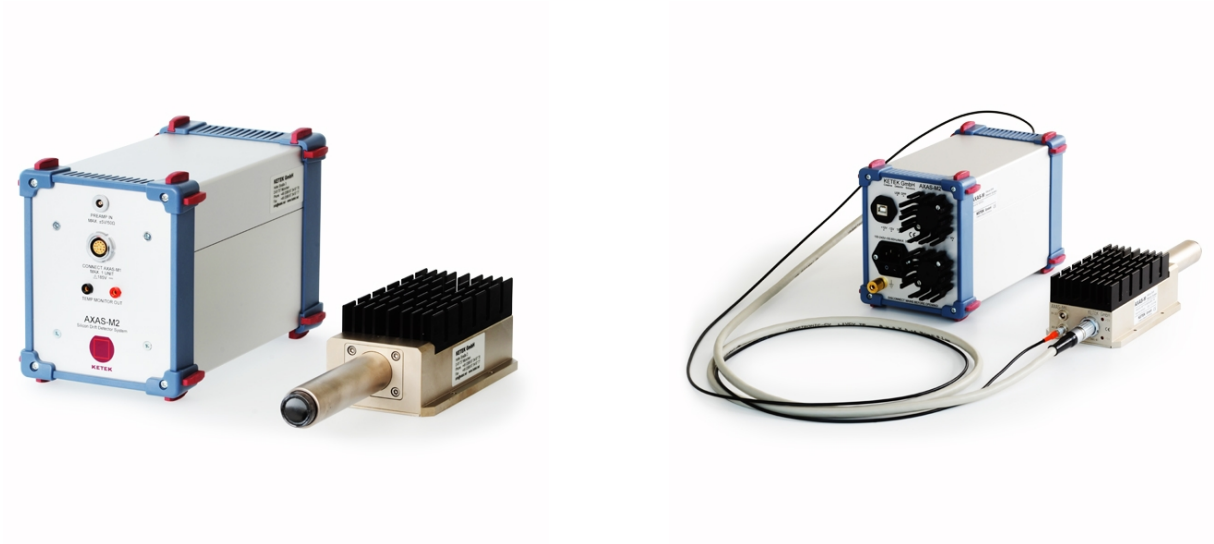


Figure 28: The Ketek AXAS-M $100mm^2$ detector with control box front and rear view [12].

More Information about the detector can be found on the internet: `https://www.ketek.net/sdd/complete-systems/axas-m/`

Problem with noise peak at about 1.7eV

A very annoying proberty of this detector set-up is the arbitrary appearance of a noise peak at 1.74eV and a count rate of 762 Count/sec. It was not possible to find the reason for this noise peak. It is superimposed with the silicon peak of the reflector and does therefore not affect measurements.

3.7 The vacuum chamber with its inner working

The sealed vacuum chamber is needed to maintain the vacuum environment for low Z measurements. It contains the sample rotator, the sample lifter, the goniometer 2 for angle adjustment, the camera and the detector. The angle of the goniometer 2 can be adjusted with the three shafts at the top of the chamber. The power supply and the signal cables are guided inwards with several vacuum feed-through components. The lid is provided with a safety switch which stops the X-ray beam as soon as it is opened.

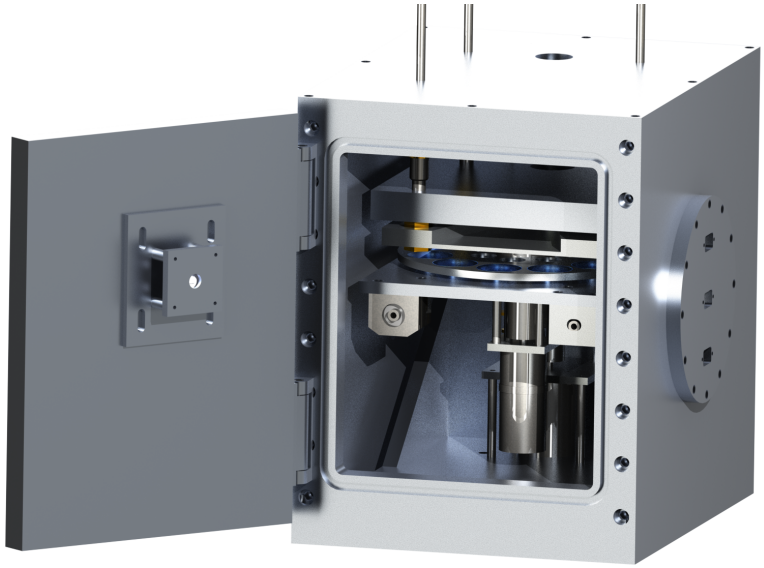


Figure 29: The measurement chamber with its inner workings.

3.7.1 The vacuum chamber

The casing is made of 20 mm thick aluminium to avoid deformation because of the vacuum inside. UHU plus endfest 2-K-Epoxidharzkleber has been used to ensure a airtight connection between the aluminium plates. It is a Solvent free 2-component epoxy resin adhesive for heavy-duty requirements. The components that must be removable because of maintenance or sample exchange has been sealed up with O-rings. The beam entrance is covered with a $8\mu m$ Kapton film to minimize the absorption of the entrance window.

3.7.2 The sample rotator

Figure 30: Sample rotator with 12 sample holder made of ultrapure Suprasil®.

The sample rotator is capable of carrying 12 reflectors. It is driven by a stepper motor controlled by the LabView program via the controller box. The sample rotator is guided by two linear guiding the ensure the correct position. The sample rotator can be moved out of the vacuum chamber to exchange sample holders. Every place is marked with a number between 1 and 12. The same numbers can be found in the LabView software.

3.7.3 The sample lifter

The sample lifter is driven by a stepper motor via a threaded spindle to transform the rotation into a linear motion. The linear bearing ensures a well guided way through the sample rotator. An adjustable end switch is used to indicate the bottom position. The top position must be set in the LabView measurement program. A spring in the top part of the lifter provides a tight fit of the sample holder on the goniometer plane. Figure 31 shows the lifter in top (a) and bottom position (b).

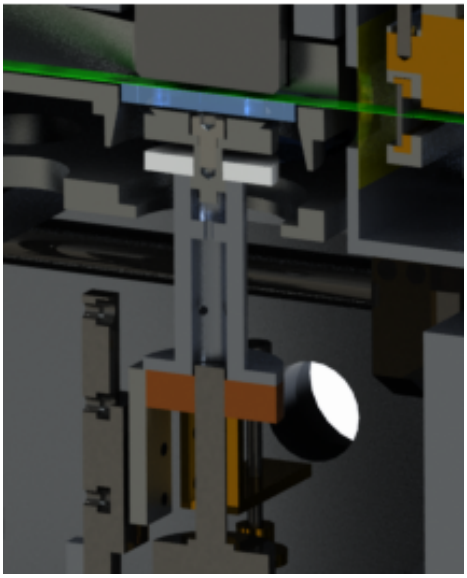
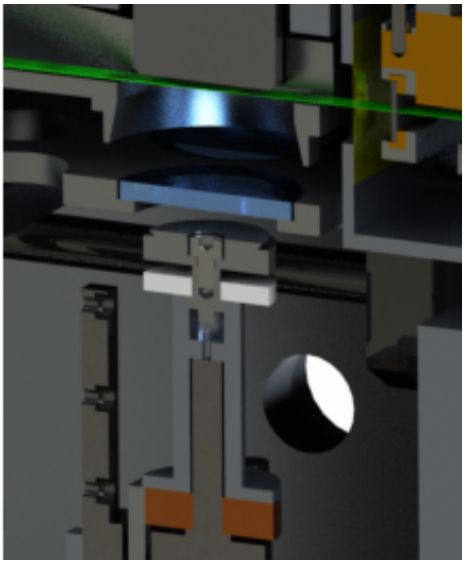
(a)  (b) 

Figure 31: The sample lifter in top (a) and bottom (b) position.

3.7.4 The camera

A CCD camera chip covered with a conventional aluminium foil is used to track the behaviour of the beam. It is supplied with a voltage of 12V DC. The common video output signal can be decoded by a screen. The camera chip is mounted on a plate with slotted holes to ensure the adjustability.

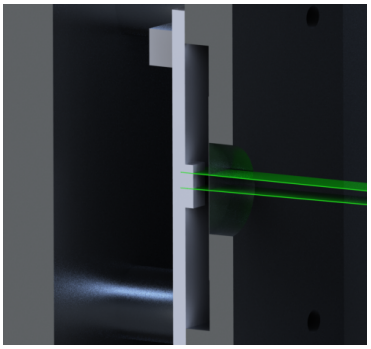


Figure 32: A rendered picture of the two incident beams.

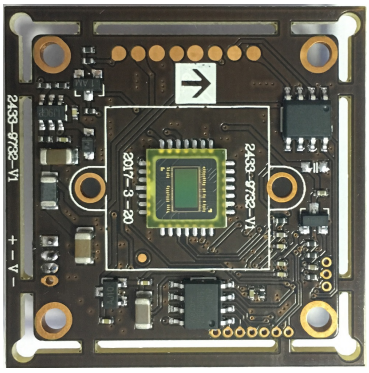


Figure 33: The camera chip which is implemented in the measurement set-up.

Figure 32 shows a crosssection of the camera with the two incident beams (ordinary beam and reflex), the aluminium foil blocks the visible light. Figure 33 shows a picture of the used camera chip. If the sample lifter is at its bottom position or if the reflector is not well enough aligned, the screen will look like figure 34a. It shows the ordinary beam without total reflection. If the the reflector is at the top position and well aligned, total reflection can be achieved and the screen will look like figure 34b.

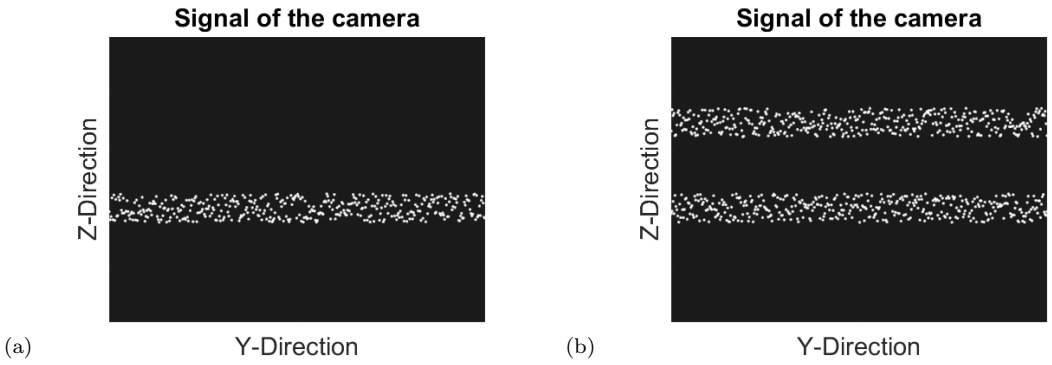


Figure 34: Screen with (a) and without (b) total reflection.

3.8 The motor controller

The motor controller consists of two transformers with 12 and 24 V and a control unit for the two stepper motors to drive the sample lifter and the sample rotator. The motor controller is connected via 2 RS232 cables, one for the lifter motor, the other one for the rotator motor. The casing of the motor controller has been purchased at RS components and is made of steel. Two 10cm fans has been mounted inside the casing to avoid overheating of the electronics.

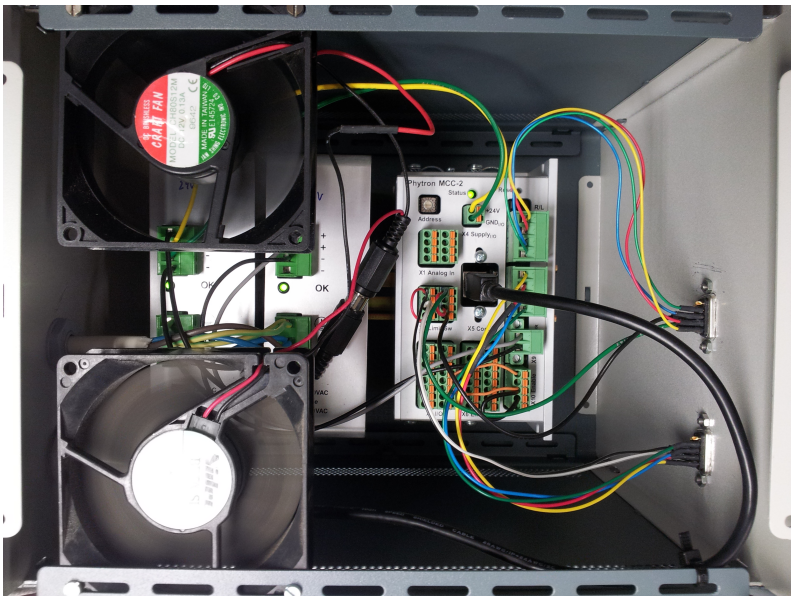


Figure 35: Components of the motor controller.

4 Circuit of the measurement set-up

Figure 36 shows the circuit of the measurement set-up. The dashed black rectangle represents the vacuum chamber. The serial connectors at the bottom of the dashed rectangle depict the vacuum feed through for supply and control signals.

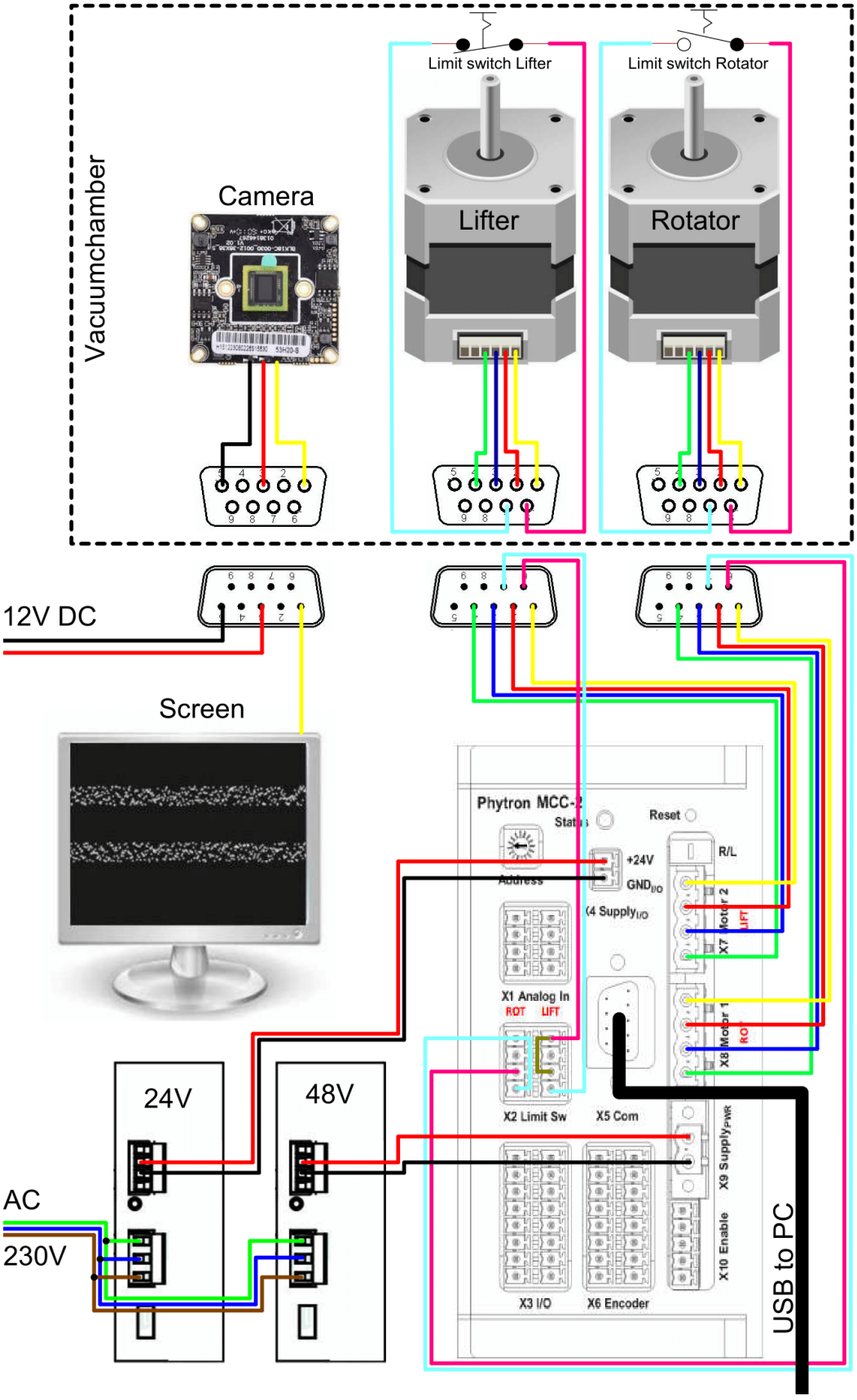


Figure 36: Main circuit of the set-up.

5 Adjustment of the x-ray beam

The beam adjustment is divided into two steps, separation of the Bragg reflex and the alignment of the separated beam to obtain total reflection at the sample holder. The separation of the planned excitation energy can be achieved with goniometer 1 and two apertures which are mounted on the x-ray tube (see section 5.2). Goniometer 2 serves the purpose to align the separated beam in the goniometer.
Figure 37 shows the end result of an adjusted beam line (green) from the X-ray source to the sample holder.

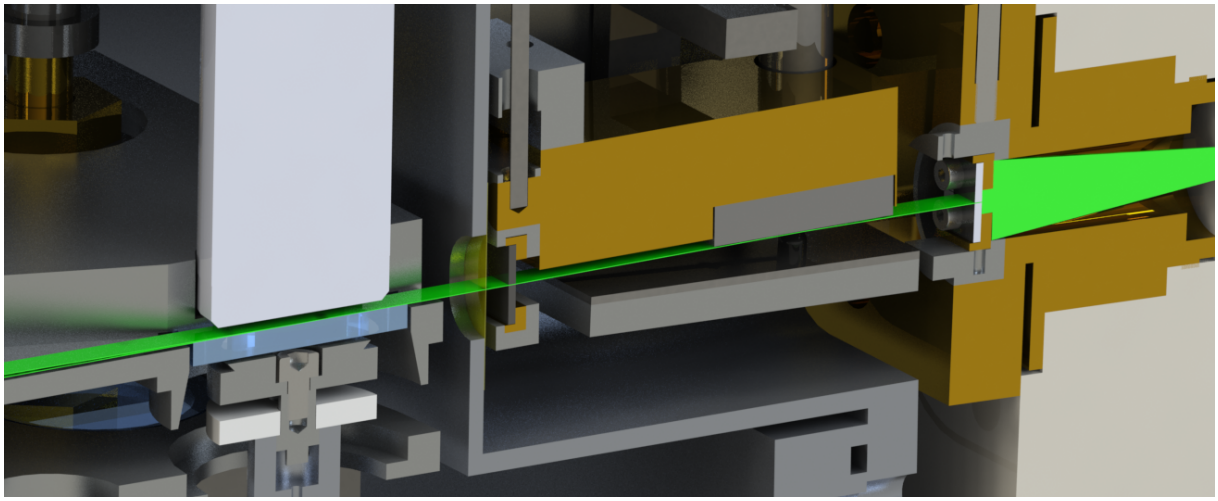


Figure 37: Beam guideance of the spectrometer

5.1 Visualisation of the beam

Different methods has been used to make the beam or the intensity of the beam visible. A fluorescence film and a magnifier are very useful instruments to make the beam visible but it is only suitable in dark places and for relatively high intensities. Another possibility is a CCD chip covered with aluminium foil that can be crossed by x-ray photons. The behaviour of the beam can than be watched on an screen. A SI(Li) detector has been used to check the behaviour of the intensity and the energy during adjustment.

5.2 Separation of the Bragg reflex with goniometer 1

The main parts of the beam separation system are aperture 1, aperture 2 and the goniometer 1. The two apertures are made of Tantalum, the goniometer consists of a carrier block with a attached multilayer and three micrometer screws for adjustment. Aperture 1 serves as collimator to generate a very thin beam. The goniometer 1 serve the purpose to adjust the multilayer crystal parallel to the beam but is also needed to scan the incident angle of the beam to find the desired energy. Figure 38 shows a schematic diagram of the beam separation system. The beam is not perfectly collimated after the aperture 1 and therefore impinges on the multilayer with different angles. As a result, a small energy range slightly above and below the desired energy is reflected because every angle of incidence corresponds to a wavelength which satisfies the Bragg equation. The aperture 2 can be adjusted in height and angle. The height adjustment is used to choose the right wavelength and the angle adjustment is needed to align the aperture 2 parallel to the Bragg reflex.

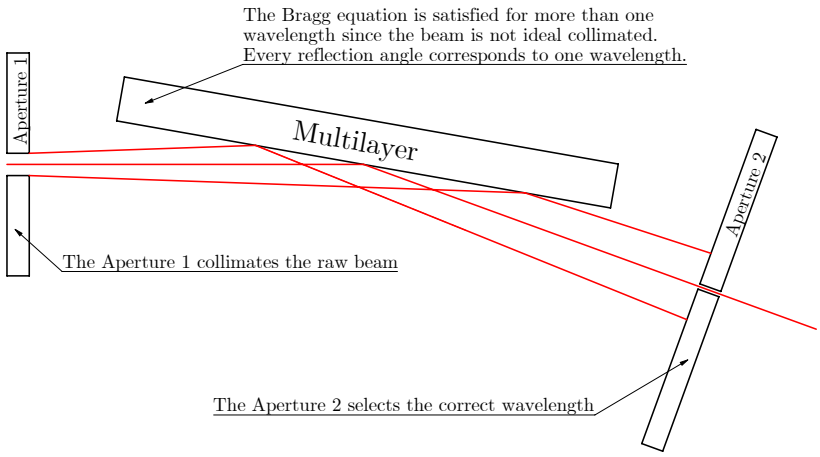


Figure 38: Schematic diagram of the beam separation system

In order to adjust aperture 1, the whole goniometer 1 must be removed by opening the two screws (1) at the aluminium carrier. The screw (2) must be loosened to open the clamp for the aperture carrier. The thread in the carrier of aperture 1 can be used for the angle adjustment of the aperture. The correct adjustment is achieved, if the out coming beam has the highest possible intensity (aperture parallel to the raw beam from the anode). The screw (2) must be tightened again to keep the aperture 1 in place.

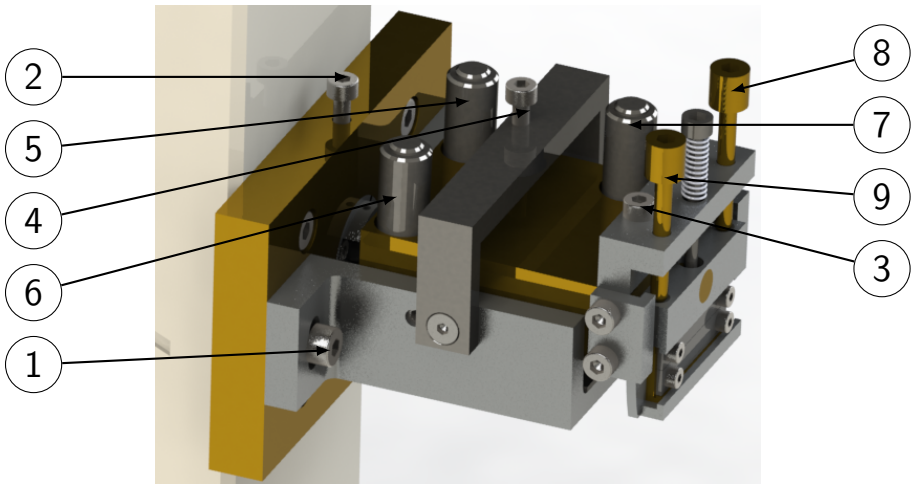


Figure 39: Presentation of the different screws.

After the adjustment of aperture 1, the whole goniometer can be mounted again on the tube. The aperture 2 must be removed from the goniometer with the 2 screws (3). The micrometer screws should be in the middle of their range to assure the highest possible adjustment range. The slotted holes are provided to adjust the aluminium carrier in a way that the beam can barely cross the goniometer. The two micrometer screws (5) and (6) closer to the tube are used to adjust the parallelism of the multilayer to the beam while the third micrometer screw (7) is used to scan the angle of incidence.
The multilayer must be dipped into the beam with the third micrometer to check the parallelism of the beam. The beam will disappear as a whole if it is already parallel. The beam is not parallel if one side of the beam disappears first. The edge of the multilayer carrier where the beam disappeared first must be raised in order to approach parallelism, e.g. the right side of the multilayer carrier must be raised if the right side of the beam disappears first. If the beam is parallel, the two screws (5) and (6) should be

used now together to raise or reduce the height of the multilayer in order to keep the parallelism. The third micrometer screw (7) can now be used to scan the incident angle in order to find the Bragg reflex of the desired energy.
The adjustment is correct if one can obtain three maxima while scanning the angle. The first maximum at the smallest angle is the total reflected beam. The second maxima is the Bragg reflex of the Rh-$K\alpha$ line. The last maxima is the Bragg reflex of the Rh-$K\beta$ line.
The aperture 2 must be mounted again on the goniometer 1. The screws (8) and (9) can be used to adjust the height of the aperture in order to select the correct energy as well as to align the aperture 2 parallel to the beam. The aperture 2 also reduces the scattering radiation which in turn reduces the background noise. A last check with the SI(Li) detector assures, that the correct energy is selected.

5.3 Adjustment of the Reflex with goniometer 2

The beam must enter the vacuum chamber almost horizontal since the angle for total reflection on the sample holder is very small. The set screw in the bottom plate provides a rough adjustment for the angle of the beam. The height of the beam can be adjusted with the micrometer stage attached at the back of the x-ray tube. To check if the beam is horizontally on can use the carriage in x direction. The beam is almost horizontal, if the beam on the camera is not moving up or down when moving the tube in +/- x direction. After the horizontal adjustment a sample holder must be set to the measurement position either manual or with the LabView measurement software.
The plane for the second goniometer should be horizontal and the beam should be barely able to cross the vacuum chamber which means the the beam is slightly above the surface of the sample holder. The sample holder must be dipped into the beam with the third micrometer screw to check if it is parallel to the beam. The beam will disappear as a whole if it is already parallel. The beam is not parallel if one side of the beam disappears first. The edge of the goniometer 2 where the beam disappeared first must be lowered in order to approach parallelism, e.g. the right side of the goniometer carrier must be lowered if the right side of the beam disappears first. The two micrometer screws are not needed any more if the sample holder is parallel to the beam. The third micrometer screw can now be used to scan the angle of incidence. It should be possible to find a range where the reflex can be seen. If no reflex occurs, the angle may be too large for total reflection. This problem can be solved by reducing the height of the whole beam. The end result should look like Fig 34b if the CCD camera is mounted in the right way.

6 Software

6.1 Introduction

The data evaluation software has been developed in "Matlab R2016a" due to the nice GUI package as well as the predefined fit function "fit()". The software is divided in two separate parts, the spectrum fitting tool and the QTXRF quantification tool. Moreover the multifit tool makes it possible to evaluate a whole list of spectra with the same predefined properties. The entire code of this software can be found in the Attachment.
The first step is to load a spectrum with the file-format "SPE". The background reduction is step number two. The Energy axis will be calibrated with the aid of some well known lines in step number 3. The fit-model as well as some properties must be set before starting the fit procedure. The results are shown in the results tab but can also be saved as ASR-file. The ASR-file is the foundation for element quantification.

6.2 Main Window

The Interface shown in figure 40 is the start and main window of the fitting software. Right after the start of the Software there are only three options to choose. The button "Select spectrum" loads a chosen spectrum file with file extension *.SPE into the program for further treatment and shows a preview of the chosen spectrum in the bottom right corner. The button "Fit multiple spectra" opens a new window that allows the evaluation of a list of spectra with the predefined parameter files.

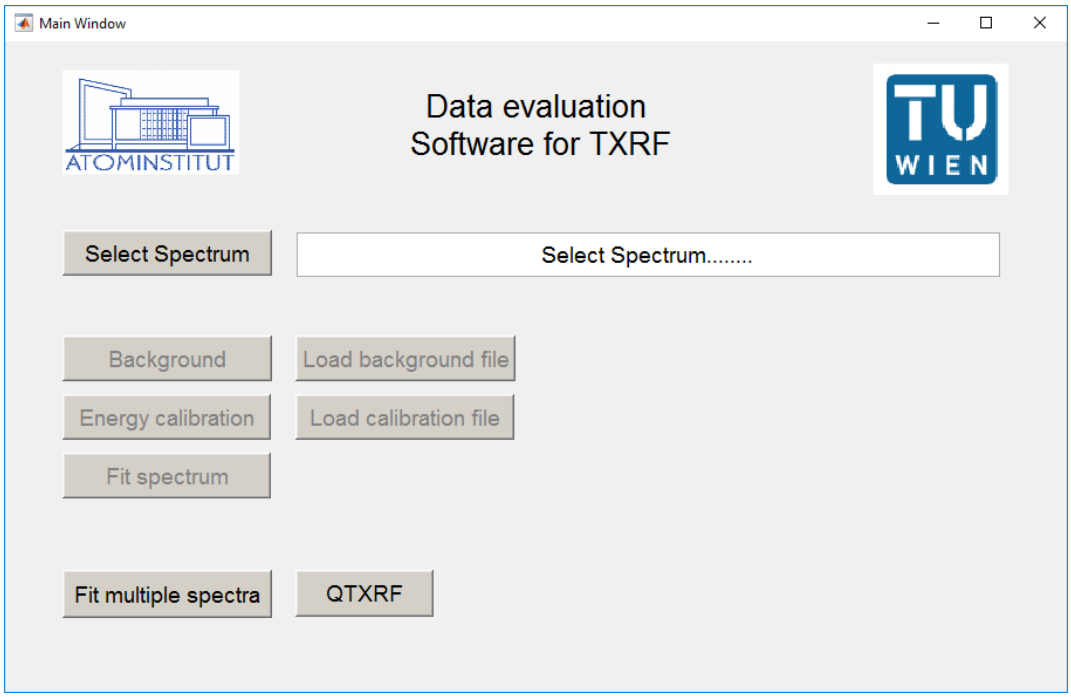


Figure 40: Main Window of the MKFIT-Software

6.3 Background Window

After the selection of the spectrum and the appearance of the preview, the button "Background" and "Load background file" is active. The button "Background" opens the background window which is an interface to estimate the background of the loaded spectrum with different methods described below, furthermore the region of interest (ROI) can be set in this window that is shown in figure 41. The scale of the ordinate can be changed in panel "Scale".

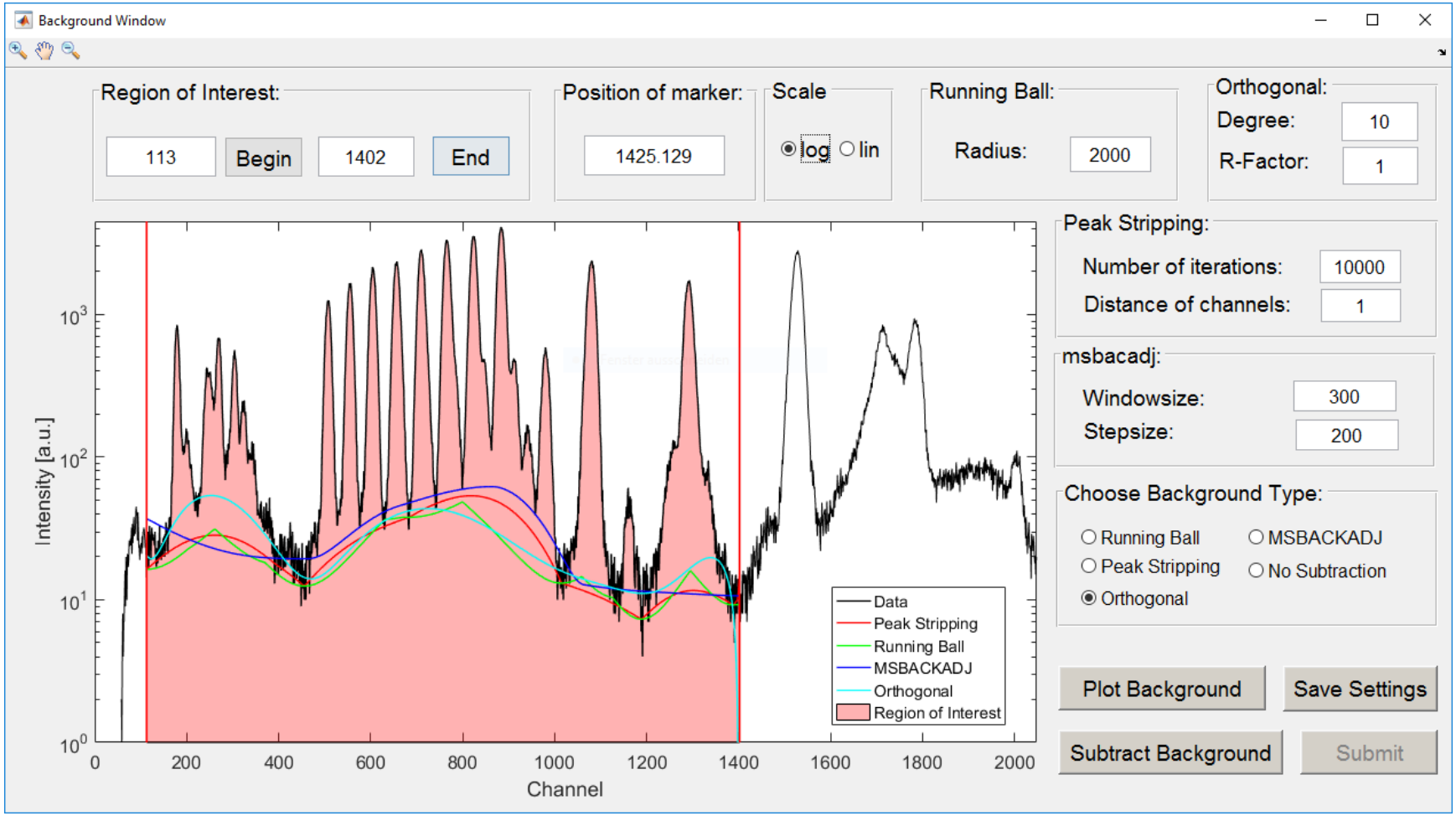


Figure 41: Background Window of the MKFIT-Software

6.3.1 Region of interest (ROI)

The region of interest or ROI is the part of the spectrum that is later on used to fit the selected lines. So all the data outside the ROI does not contribute or affect the fit results. The region of interest can be set in the first panel on the upper left. The panel "Position of marker" shows the position of the last click inside the axes. The Button "Begin" sets the left limit while the button "End" sets the right limit of the ROI to the value indicated in the panel "Position of marker".

6.3.2 Peak Stripping

This method is based on the removal of rapidly varying structures in a spectrum by comparing the channel content y_i with the channel content of its neighbours.

$$m_i = \frac{y_{i-1} + y_{i+1}}{2} \tag{20}$$

If y_i is bigger than m_i, the content of channel i y_i will be replaced by m_i. A generalisation of this method is to vary also the distance of the next neighbour.

$$m_i = \frac{y_{i-w} + y_{i+w}}{2} \tag{21}$$

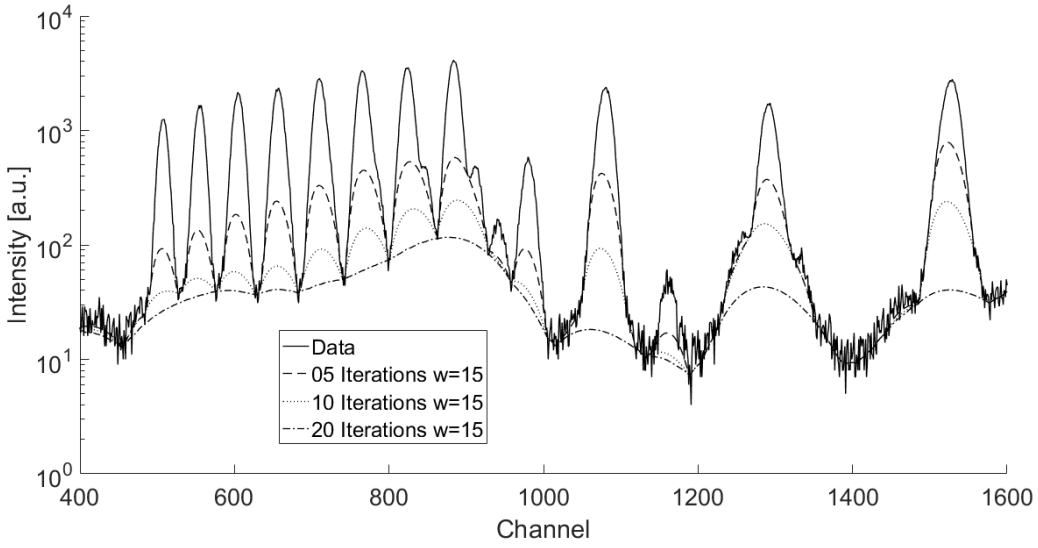


Figure 42: Peak stripping algorithm

The peak stripping algorithm tends to connect local minima, hence it is very sensitive to fluctuations in the spectrum due to counting statistics. Because of this misfeature, smoothing prior to the peak stripping algorithm is mandatory. This Software uses a Savitzky-Golay filter with a polynomial order of 2 and a frame length of 11. The Iterations and the width can be freely chosen.
Good results can be achieved, when the width w of the generalised algorithm is set to the average FWHM of the peaks [22, pp. 275-276].

6.3.3 Rolling Ball

The idea of this algorithm is from a publication concerning image Biomedical Image Processing [18]. Imagine a 2D surface with the channel content being the height. A circle rolling over the back side of the surface creates the background. The radius of the ball can be freely chosen. A radius of 100 means a ball with a radius of 100 channels.

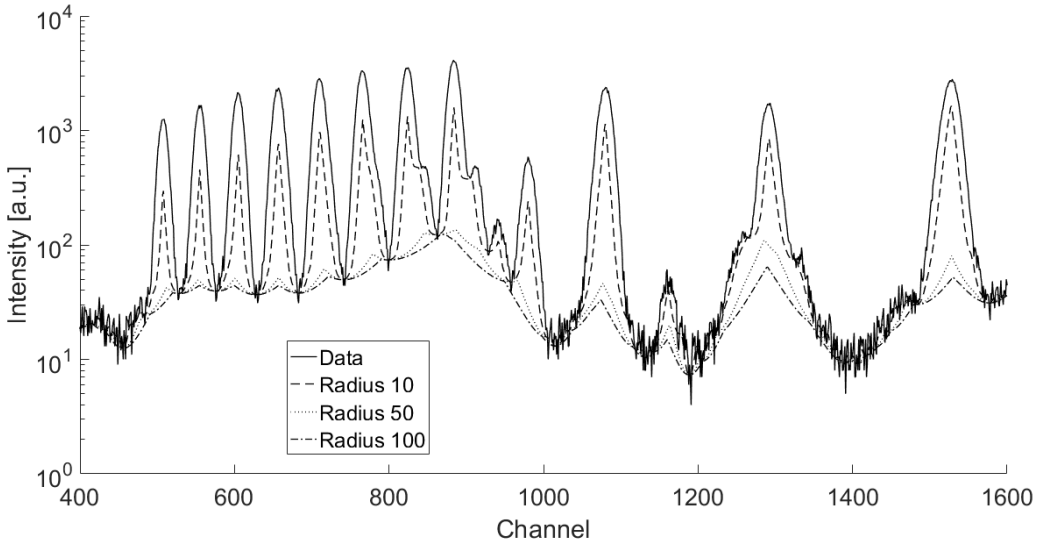


Figure 43: Running ball algorithm with Radius 10,50 and 100.

6.3.4 Orthogonal Polynomial

The background is fitted using orthogonal polynomials where the weights of the least square fit are iteratively changed to exclude the data channels from the fit. Compared to polynomials, orthogonal polynomials of a much higher degree can be fitted to the experimental data without running into problems with ill-conditioned normal equations and oscillating terms. Steenstrup proposed the following method where it is possible to exclude data channels and weight the background channels for the fit automatically. First, the Background is estimated with a orthogonal polynomial of degree m and weights $w_i = \frac{1}{y_i}$ where y(i) is the background estimation. To adjust the weights after every fit, following criteria are used:

$$w_i = \frac{1}{y_i^2} \quad if \quad y_i \leq y(i) + r * \sqrt{y(i)} \tag{22}$$

$$w_i = 0 \quad if \quad y_i > y(i) + r * \sqrt{y(i)} \tag{23}$$

Where r is a factor that specifies if a data point belongs to the background or not. A r-factor of 2 means that a data point that is higher than 2 times the standard error σ plus estimated background will be excluded. If y_i is normally distributed $y_i > y(i) + 2 * \sqrt{y(i)}$ holds for 97.7% of the channels containing only continuum [22, pp. 276-278].

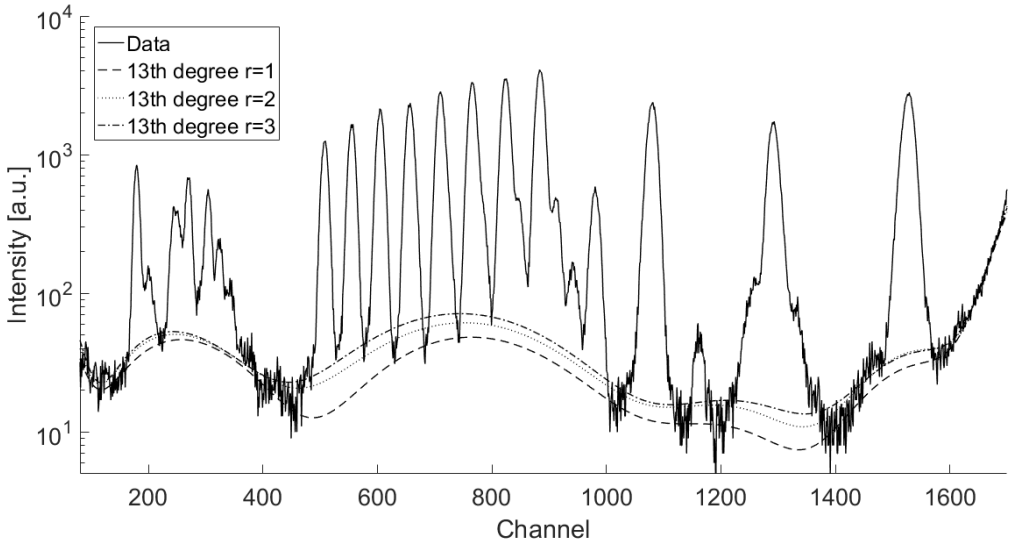


Figure 44: Orthogonal polynomials algorithm with different r values

Figure 44 shows a orthogonal polynomial with 13th degree and three different r factors. The higher the r factor, the more drifts the polynomial up into the data peaks. Figure 45 shows three different polynomial with r=2 but different degrees. The tendency of the polynomial to fit into narrower peaks rises with the degree. If the changes per iteration fall below a pre-set value, the current polynomial will be set to the estimated background.

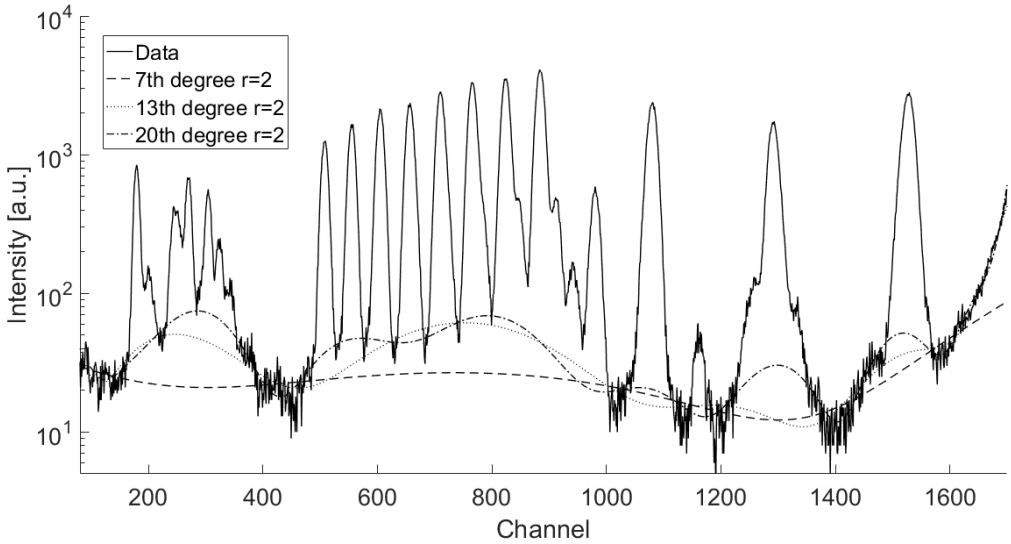


Figure 45: Orthogonal polynomials of different degrees

6.3.5 Matlab baseline correction

This algorithm is implemented in Matlab and can be accessed via the function:
msbackadj(channel,data,'WindowSize',window,'StepSize',step). Multiple shifted windows of adjustable width and shift are used to determine points which are then connected via a spline. "Windowsize" and "Stepsize" correspond to the width and the shift, respectively.

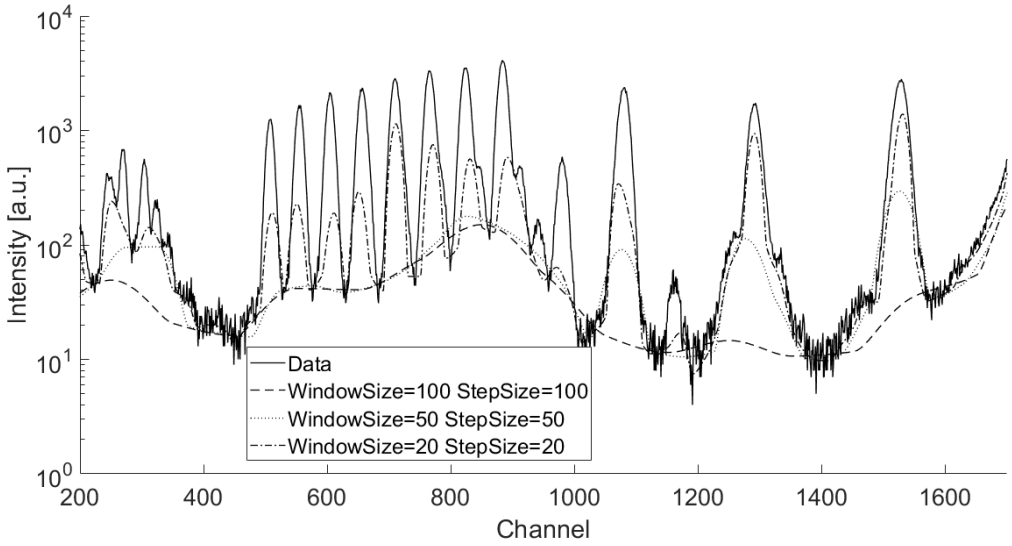


Figure 46: msbackadj algorithm with different parameters

Figure 46 shows the algorithm with equal 'StepSize' and 'WindowSize' of 100, 50 and 20 respectively. The 'WindowSize' parameter should be bigger than the FWHM of the smallest peak. The background approximation reduces the single peaks if the 'WindowSize' is too small.

6.3.6 Estimation and processing of the background

After setting the ROI and all the parameters for the selected background method the button "Plot background" calculates all background methods and shows the results in the graph. The chosen method can then be selected in the radio buttons panel. By pressing the button "Subtract background" the program subtracts the selected background from the original spectrum. All the properties set in this interface can be saved with the button "Save settings". This file can then be loaded via the button "Load background file". Such a file is also needed for the multifit tool where it predefines the background procedure for the selected spectra.

6.4 Energy axis calibration

After the background estimation, the button "Energy calibration" and "Load calibration file" are active. The button "Energy calibration" opens the Energy calibration window (fig. 47) which is an interface to calibrate the x-axis of the loaded spectrum. The scale of the ordinate can be changed in panel "Scale" to "lin" or "log". Two specified data pairs (channel number and appropriate energy value) are mandatory for the calibration of the energy axis but additional data pairs within a big range of the spectrum will generate a more precise calibration. A specific part of the spectrum can be magnified by pressing the button "Left Border" or "Right Border" after setting the marker.

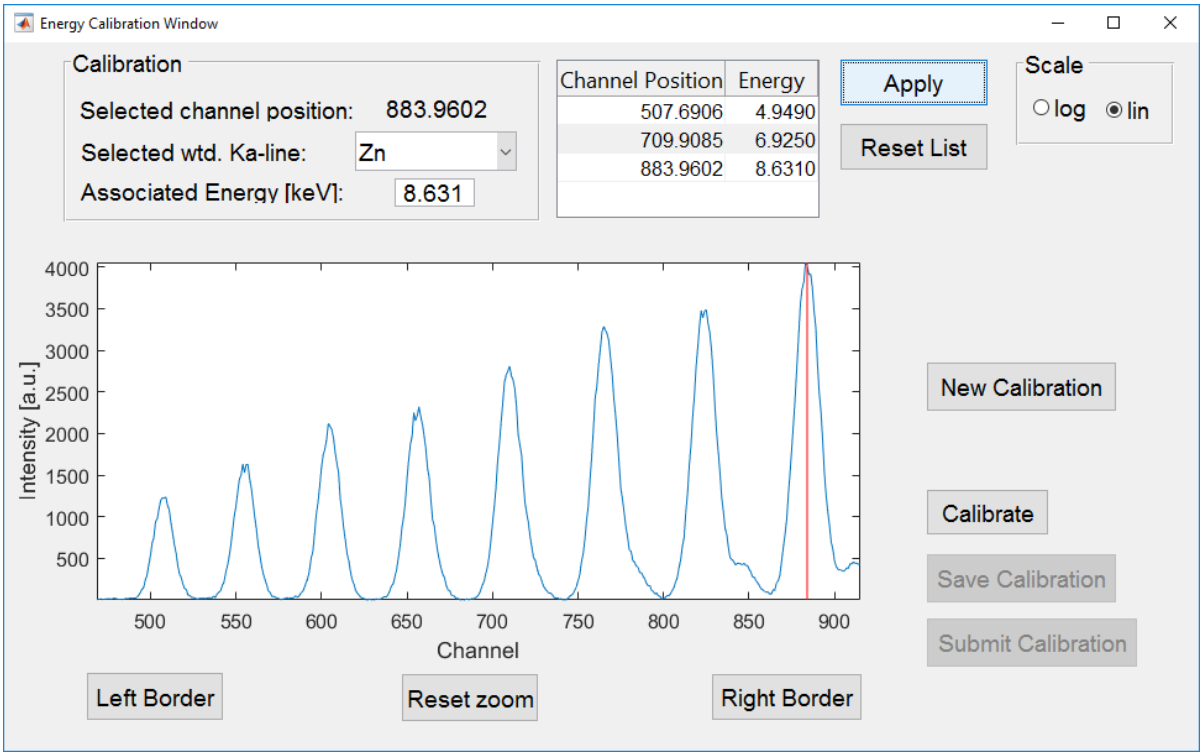


Figure 47: Energy calibration Window of the MKFIT-Software

A new data pair can be added by clicking on the peak of a well known line, choose the weighted $K\alpha$ line in the list or insert the energy manually and press the button "Apply". The button "Calibrate" is getting active after adding at least two data pairs. Pressing the button execute a unweighed line fit to the data pairs with the following equation:

$$E(i) = ZERO + GAIN \cdot i \tag{24}$$

with the channel number i and the two fit parameter ZERO and GAIN.
If the fit is appropriate, the button "Save Calibration" saves the calibration to a file. This calibration file can be accessed with the button "Load calibration file" instead of the interface "Energy calibration". Such a file is also needed for the multifit tool where it predefines the calibration parameter for the selected spectra. The button "Submit Calibration" closes the Interface and internally saves the calibration for further treatment.

6.5 Fitting Tool

After the background estimation and energy axis calibration, the button "Fit spectrum" opens the Fitting tool interface which consists of six different windows that can be accessed by clicking on the tabs bar at the top of the interface.

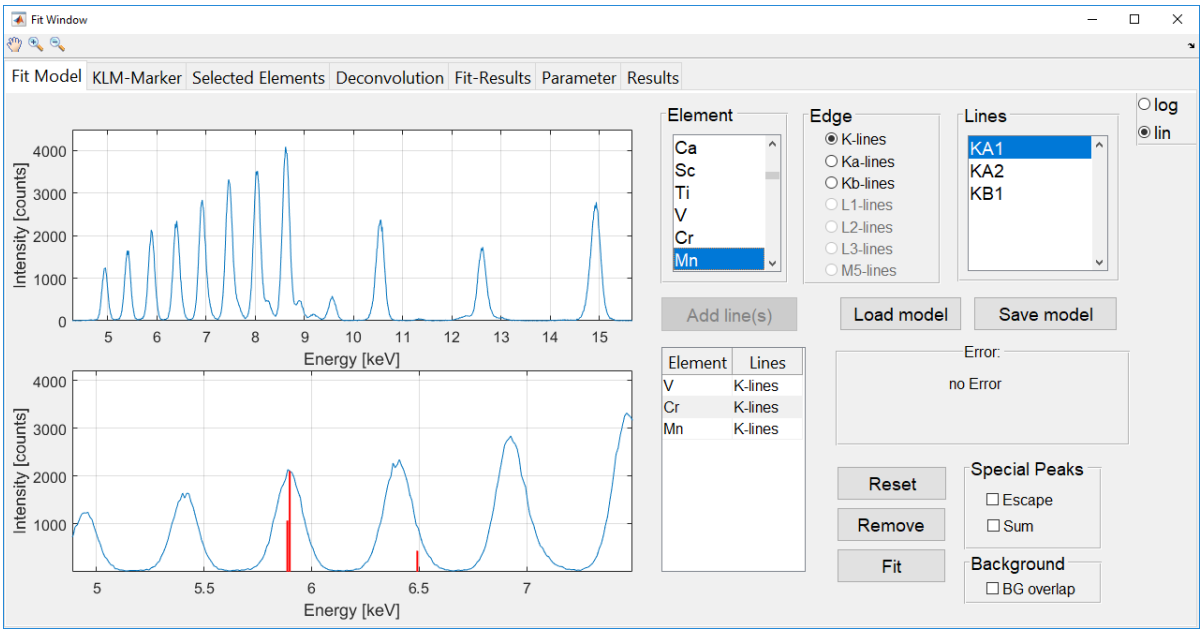


Figure 48: "Fit model" tab of the Fitting tool Window of the MKFIT-Software

6.5.1 Fit Model

The upper chart on the left shows the whole loaded spectrum background reduced and axis calibrated. With the aid of on the -button it is possible to zoom into the spectrum. The -button resets the zoom and displays the whole spectrum again. The lin/log scale of the y-axis can be changed in the upper right corner. The lower chart indicates the current chosen elements lines with its relative sensitivities. The Element can be chosen in the "Element" panel next to the first chart. If the list-box is selected, one can use the arrow keys ↓ or ↑ to change the selection of the elements. The active radio buttons in the "Line families" panel indicate which lines are available for the selected element. The "Lines" panel lists the appropriate lines to the selected family. The error panel display an error if the chosen line family is not in the range of the spectrum.
The button "Include line(s)" below the "Element" panel adds the selected line family to the list. This list is the foundation for the fit model. An added family can be removed by pressing the button "Remove" after selecting the unwanted family in the list. The button "Reset" resets the whole list, with the button "Save model" one can save the list of lines in a file which can be accessed by pressing the button "Load model". Additional special peaks like escape and sum peaks (description below) can be included to the fit model by ticking the respective boxes.
There are two different options for the peak background estimation. The default method scans the whole spectrum and register how much peaks are overlapping for each channel where the peak width is assumed to be the FWTM (full width tenth maximum). This peak width is also used to calculate the reduced χ^2_P value in equation 29. After the scan, each channel of the background is divided by number of overlapping peaks at this channel. The background for each peak is then calculated by summing up the modified background channels in the range of the peaks FWTM. By ticking the box "BG overlap" the background is not modified before summing up the background channels in the range of the peaks FWTM. The button "Fit" generates a fit model which is then fitted to the spectrum.
The fit-model is a sum of Gaussian functions since the response function of the detector is in first approximation Gaussian shaped. In reality the response function is a convolution of a Gaussian profile (detector response) with a Lorentz profile (natural linewidth) which is called Voigt profile. The natural linewidth is very small compared to the linewidth of the detector which makes the approximation valid. The more complex response function called Voigt profile is used for high energy lines like K lines of U and Pb. A comparison between Voigt, Gaussian and Lorentz profile can be seen in figure 49.

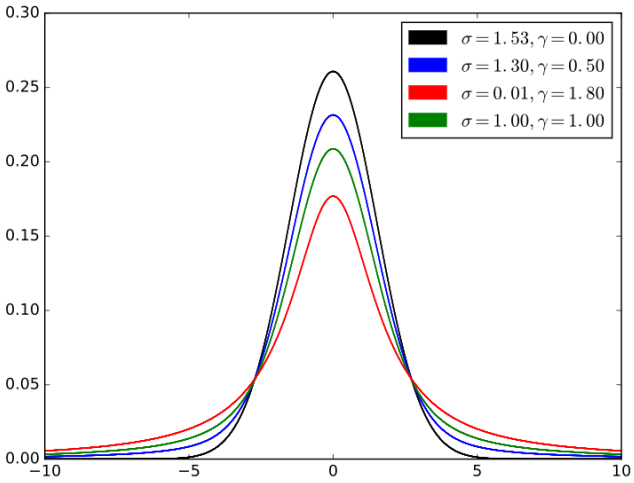


Figure 49: Plot of the centered Voigt profile for four cases. Each case has a full width at half-maximum of very nearly 3.6. The black and red profiles are the limiting cases of the Gaussian ($\gamma = 0$) and the Lorentzian ($\sigma = 0$) profiles respectively [24].

The response function for one line is given by:

$$G(i, E_j) = \frac{GAIN}{\sigma \cdot \sqrt{2\pi}} \cdot exp\left(-\frac{[E_j - E(i)]}{2\sigma^2}\right) \tag{25}$$

with:
i ..... Channel number
E_j ..... Energy of the line j
$E(i)$ ..... Energy which corresponds to channel i
$GAIN$ ..... Fitparameter GAIN
σ ..... Peak width

The peak width σ can be determined with:

$$\sigma = \sqrt{\left(\frac{NOISE}{2.3548}\right)^2 + 3.85 \cdot FANO \cdot E_j} \tag{26}$$

with:
$NOISE$ ..... Noise parameter of the detector (FWHM of the noise peak)
$FANO$ ..... Fano factor of the detector (description below)
3.85 ..... energy in eV required to produce an electron-hole pair in silicon

The Fano factor takes into account, that the individual events of the energy-loss process in semiconductors are not strictly independent which leads to a deviation of the Poisson behavior and thus to a additional factor in the peak width equation.

The known intensity relation between lines of the same family can be used to derive a response function for a whole family by summing up the separate lines weighted with their relative intensities:

$$
\begin{aligned}
y(i)_P &= A \sum_{j=0}^{N_p} R_j \cdot G(i, E_j) \\
&= A \sum_{j=0}^{N_p} R_j \cdot \frac{GAIN}{\sqrt{(\frac{NOISE}{2.3548})^2 + 3.85 \cdot FANO \cdot E_j} \cdot \sqrt{2\pi}} \cdot exp\left(- \frac{[E_j - E(i)]}{2 \cdot \left[(\frac{NOISE}{2.3548})^2 + 3.85 \cdot FANO \cdot E_j \right]} \right)
\end{aligned}
\tag{27}
$$

with:
A ..... Fitparameter A that corresponds to the peak area of the line family
j ..... Index of line which runs from 0 to N_p
R_j ..... Relative intensity of the line j

The sum of all line families in the created list leads to the overall response function. The fit-parameter are GAIN, ZERO, NOISE, FANO and the peak area parameter $A_1, A_2, A_3, ...$. Optimizing the energy and resolution calibration parameters instead of the position and width of each peak drastically reduces the dimensionality of the problem. [22, pp. 295-297]

Escape Peaks

Escape peaks are spurious spectral lines that occur in X-ray and gamma spectroscopy and can simulate the presence of non-existing elements in the measured sample. These peaks emerge through "escape" effects in the detector. An incident photon which is passing through the detector volume can produce a photo electron from an inner shell of a crystal atom (Si). The ionized atom can emit fluorescence X-ray photon which is usually reabsorbed in the detector. However, there is a chance that the photon which is usually a $K\alpha$ photon escape from the detector crystal and thus carries off the definite energy of the $Si - K\alpha$. This makes additional peaks visible in the spectrum with a correspondingly smaller energy value.

Sum Peaks

Sum peaks are also spurious spectral lines that occur in X-ray and gamma spectroscopy and can simulate the presence of non-existing elements in the measured sample. Sum peaks arise when two photons hit the detector almost simultaneously so that the processing electronics erroneously register the sum of the two photon energies rather than each one individually. A smaller beam intensity can prevent the problem of sum peaks with the disadvantage of a poorer statistic.

6.5.2 KLM-marker

The tab "KLM-marker" shows all lines in the library for a specific element (fig. 50). It is a mandatory tool to distinguish elements when lines are overlapping. In the upper right corner, one can change the y-axis to linear or logarithmic scale. The colored lines represent the relative sensitivities of the respective line family.

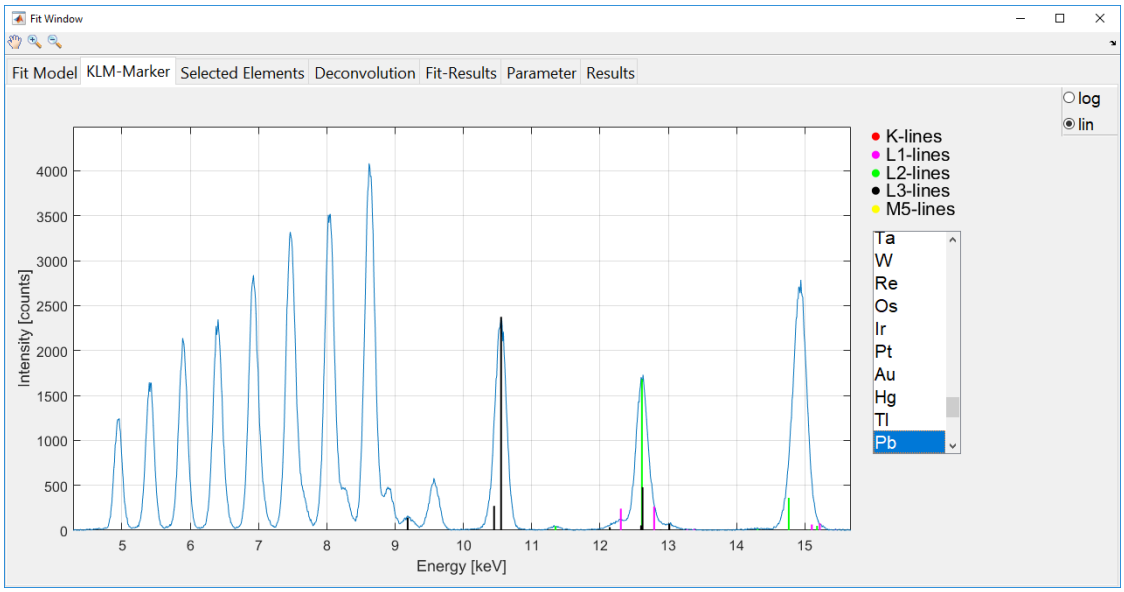


Figure 50: "KLM-marker" tab of the MKFIT-Software

6.5.3 Selected elements

The tab "Selected elements" indicates the lines that has been added to the fit model (fig. 51). In the upper right corner, one can change the y-axis to linear or logarithmic scale. The red lines represent the relative sensitivities of the added lines.

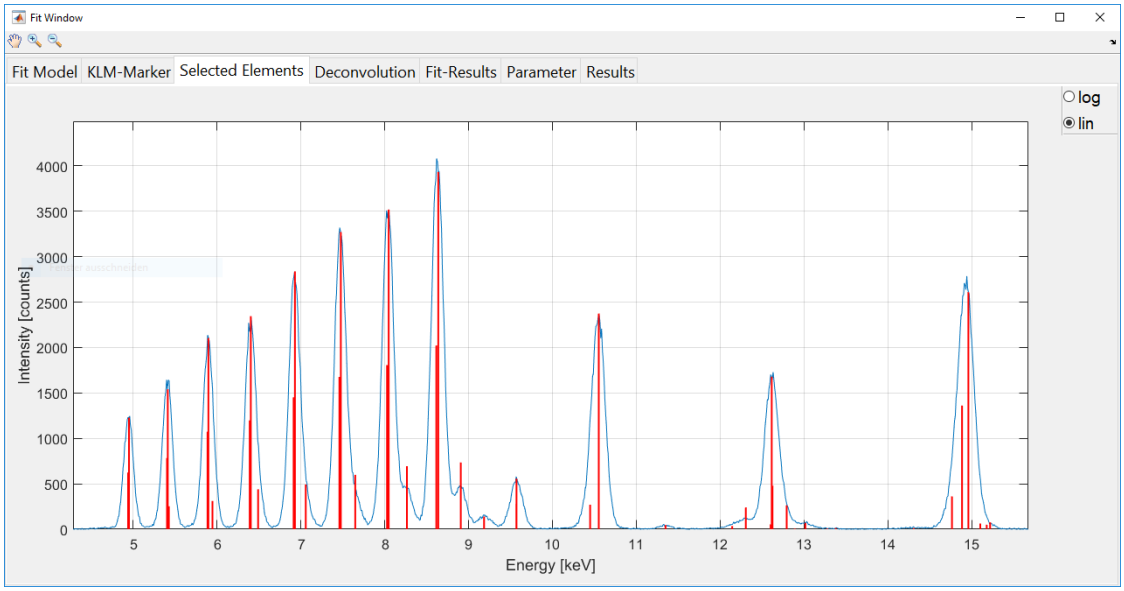


Figure 51: "Selected elements" tab of the MKFIT-Software

6.5.4 Deconvolution

The tab "Deconvolution" shows the measures spectrum and all the fitted line families plotted separately. The sum of all this plots gives the red graph in figure 53. In the upper right corner, one can change the y-axis to linear or logarithmic scale.

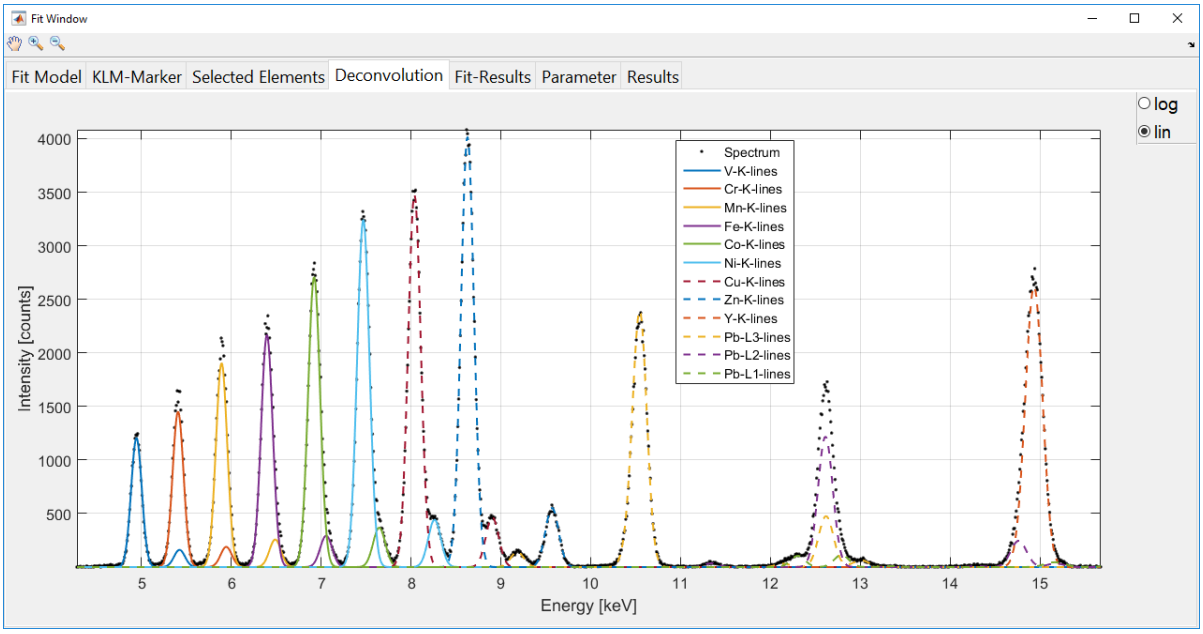


Figure 52: "Deconvolution" tab of the MKFIT-Software

6.5.5 Fit result

The program switches automatically to the tab "Fit result" by pressing the "Fit" button which can be seen in figure 53. The upper graph shows the measured data without background (blue graph) and the fitted model (red graph). As well as in all tabs the -button can be used to zoom into the spectrum. The -button resets the zoom and displays the whole spectrum again. The lower graph shows the residual of the fit:

$$Res(i) = \frac{y_i - y(i)}{\sigma_i} = \frac{y_i - y(i)}{\sqrt{\sigma_D^2 + \sigma_B^2}} = \frac{y_i - y(i)}{\sqrt{y_i + y_B}} \tag{28}$$

The two green lines show a typical statistical boundary of $\pm 3 \cdot \sigma$. This boundary means that if the residual is inside this boundaries, no peaks has been overlooked with a probability of 99.73%.

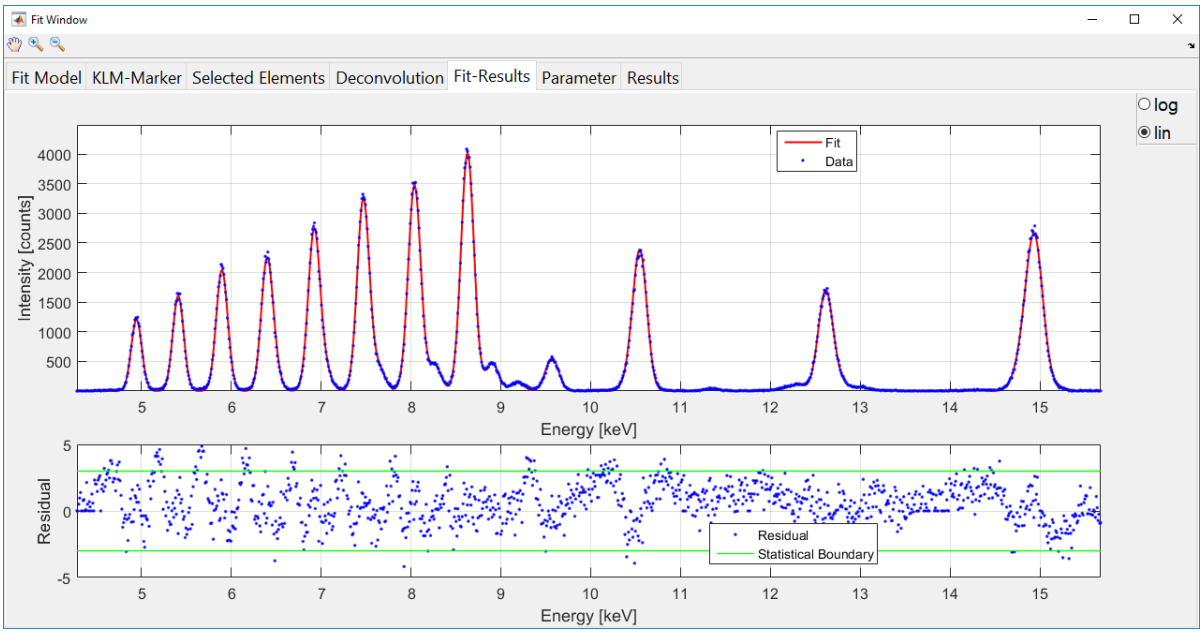


Figure 53: "Fit Results" tab of the MKFIT-Software

6.5.6 Parameter

The tab "Parameter" shows all adjustable parameter for the fit procedure. Robust fitting algorithms can be choosen in the list box to the right of the parameter table. "LAR" changes the weights after each fit with respect to the absolute residuals. "Bisquare" changes the weights after each fit with respect to the squared residuals. This method is less affected by outliners. "off" doesn't change the fit weights and is therefore the fastest option. The data of the line energy and relative intensity library comes the "Handbook of X-Ray Spectrometry" [22]. The NIST homepage is the data source of the used mass-attenuation file: `https://physics.nist.gov/PhysRefData/XrayMassCoef/ElemTab/z14.html`.

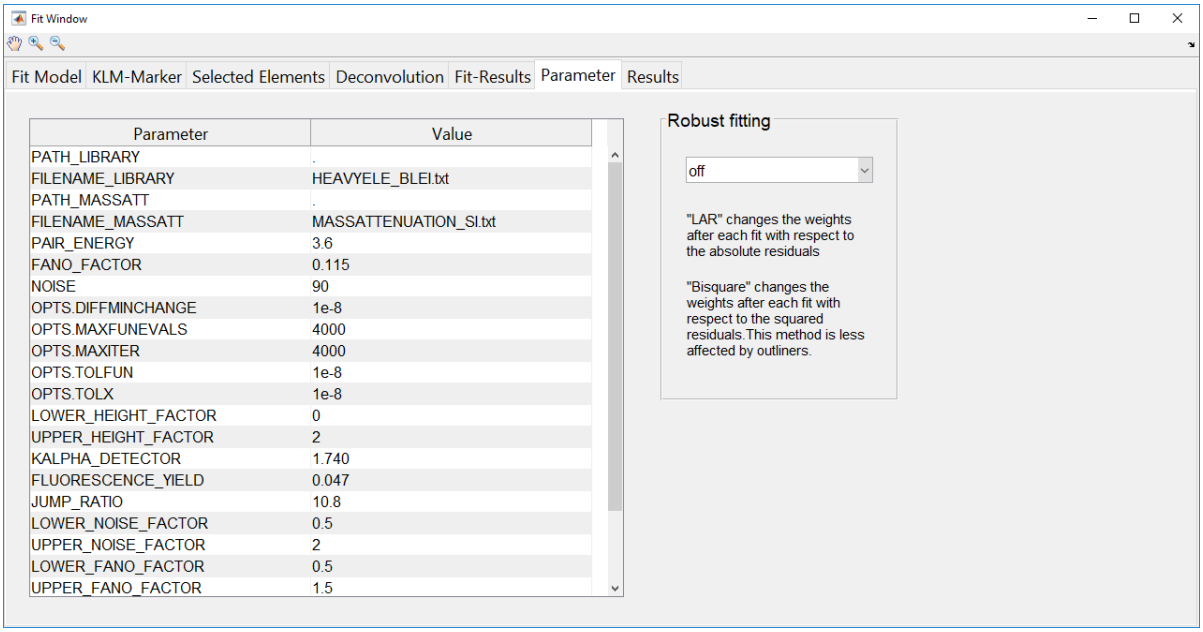


Figure 54: "Parameter" tab of the MKFIT-Software

The following list describes the adjustable parameter. Factor of lower/upper parameter means that the

factor times the current value of the parameter is the lower/upper limit for the fit algorithm.

PATH_LIBRARY ..... Path to the element library file
FILENAME_LIBRARY ..... Filename of the element library file
PATH_MASSATT ..... Path to the massattenuation file
FILENAME_MASSATT ..... Filename of the massattenuation file
PAIR_ENERGY ..... Pair energy in eV
FANO_FACTOR ..... Value of the fano factor
NOISE ..... Noise of the detector in eV
OPTS.DIFFMINCHANGE ..... Minimum change of the evaluated function
OPTS.MAXFUNEVALS ..... Maximum number od function evaluations
OPTS.MAXITER ..... Maximum number of iterations
OPTS.TOLFUN ..... Termination tolerance on the function value
OPTS.TOLX ..... Termination tolerance on the x value
LOWER_HEIGHT_FACTOR .. Factor of the lower height limit
UPPER_HEIGHT_FACTOR ... Factor of the upper height limit
KALPHA_DETECTOR ..... $K\alpha$ value of the detector (escape peak)
FLUORESCENCE_YIELD ..... Fluorescence yield of the detector (escape peak)
JUMP_RATIO ..... Minimum change of the evaluated function
LOWER_NOISE_FACTOR ..... Factor of the lower noise limit
UPPER_NOISE_FACTOR ..... Factor of the upper noise limit
LOWER_FANO_FACTOR ..... Factor of the lower fano factor limit
UPPER_FANO_FACTOR ..... Factor of the upper fano factor limit
LOWER_ZERO_FACTOR ..... Factor of the lower zero limit
UPPER_ZERO_FACTOR ..... Factor of the upper zero limit
LOWER_GAIN_FACTOR ..... Factor of the lower gain limit
UPPER_GAIN_FACTOR ..... Factor of the upper gain limit

The following list shows all adjustable parameters with their default values:

```
PATH_LIBRARY          .
FILENAME_LIBRARY      HEAVYELE_BLEI.txt
PATH_MASSATT          .
FILENAME_MASSATT      MASSATTENUATION_SI.txt
PAIR_ENERGY           3.6
FANO_FACTOR           0.115
NOISE                 90
OPTS.DIFFMINCHANGE    1e-8
OPTS.MAXFUNEVALS      4000
OPTS.MAXITER          4000
OPTS.TOLFUN           1e-8
OPTS.TOLX             1e-8
LOWER_HEIGHT_FACTOR   0
UPPER_HEIGHT_FACTOR   1.5
KALPHA_DETECTOR       1.740
FLUORESCENCE_YIELD    0.047
JUMP_RATIO            10.8
LOWER_NOISE_FACTOR    0.5
UPPER_NOISE_FACTOR    2
LOWER_FANO_FACTOR     1
UPPER_FANO_FACTOR     1
LOWER_ZERO_FACTOR     0.9
UPPER_ZERO_FACTOR     1.1
LOWER_GAIN_FACTOR     0.9
UPPER_GAIN_FACTOR     1.1
```

6.5.7 Results

The tab "Results" shown in figure 55 is made up of four different panels. The first and biggest panel on the left shows the peak area and the standard deviation of each line family (green) in the fit model. Furthermore is the peak area, the standard deviation, the background and the Chi-square listed for all separate lines in the family below the green entry. The peakarea is determined by the fitparameter and the standard deviation is calculated with the aid of the covarianz matrix since the diagonal elements of the kovariance matrix are the variances of the fitparameters.

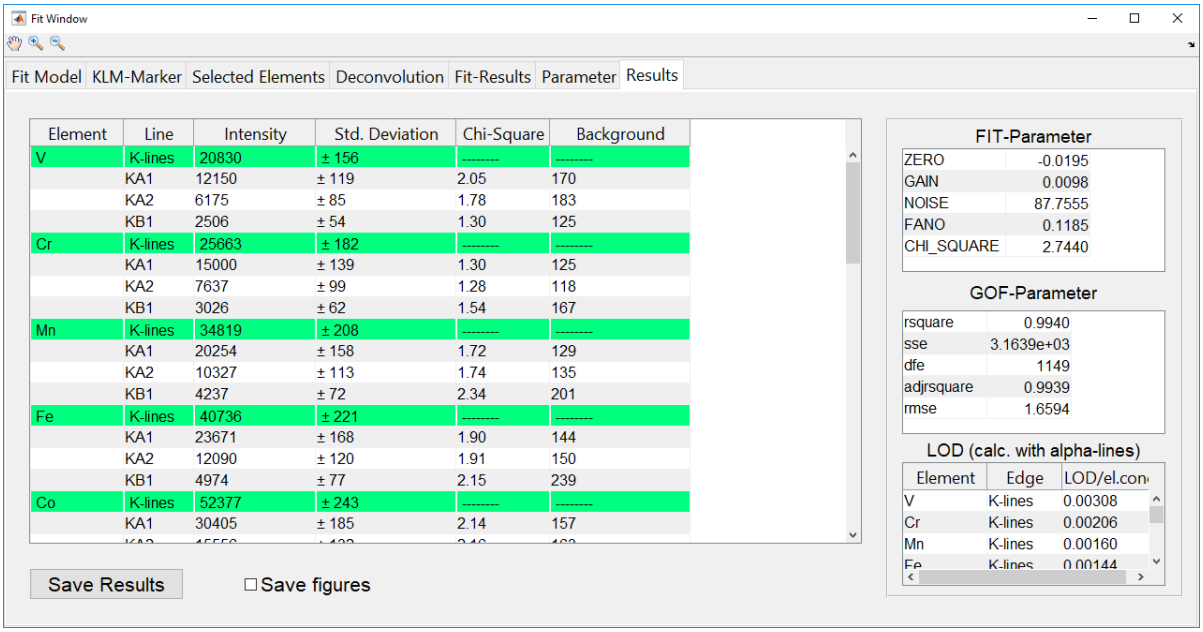


Figure 55: "Results" tab of the MKFIT-Software

The background for each peak depends on the selected option in the Fit-Model.

Chi-square

The Chi-square value is calculated with the following equation and contains information about fit quality as well as overlapping lines.

$$\chi_P^2 = \frac{1}{n_1 - n_2} \sum_{n_2}^{m} \frac{1}{\sigma_i^2} \cdot [y_i - y(i)]^2 \tag{29}$$

with:
n_1 ..... channel number at the end (FWTM) of the peak
n_2 ..... channel number of the beginning (FWTM) of the peak
m ..... $n_2 - n_1$
$sigma_i$ ..... standard deviation of the channel
y_i ..... value of the fit model at channel i
$y(i)$ ..... channel content of channel i

High values of χ_P^2 indicate that the peak is fitted poorly and the resulting peak area should be used with caution. [22, p. 311]
Pannel two shows the fit parameter and the chisquare for the whole fit. Panel three shows the standard Matlab "goodness of fit" (GOF) parameter. Panel four shows the detection limits of the elements.

Detection Limits

The lower limit of detection (LLD) is one of the most traditional limit of detection in XRF spectroscopy [2]. Since the net peak intensity I_N is equal to the difference between the measured peak intensity I_P and the background peak intensity I_B, the standard error for the net peak intensity I_N is equal to equation 30.

$$\sigma_N = \sqrt{\sigma_P^2 + \sigma_B^2} \tag{30}$$

Based on a confidence interval of 95%, the net peak intensity I_N must exceed 2 times the standard error.

$$I_N \geq 2\sigma_N = 2\sqrt{\sigma_P^2 + \sigma_B^2} \tag{31}$$

For very small net peak intensities the relations 32 are satisfied.

$$I_B \gg I_N,\ I_P \sim I_B,\ \sigma_P \sim \sigma_B \tag{32}$$

Because of $\sigma_P \sim \sigma_B$ the inequality 31 becomes

$$I_N \geq 2\sqrt{2\sigma_B^2} = 2\sqrt{2}\sigma_B \sim 3\sigma_B \tag{33}$$

The LLD is the concentration corresponding to I_N where S [cps/conc.] is the slope of the calibration line and m is the concentration.

$$LLD = \frac{3\sigma_B}{N_N} * m = \frac{3\sqrt{N_B}}{N_N} * m = \frac{3\sqrt{I_B * t}}{I_N * t} * m = \frac{3}{S} * \sqrt{\frac{I_B}{t}} \tag{34}$$

6.6 Multi Fitting Tool

The multi fitting tool makes it possible to fit a lot of spectra with just one click. First one has to choose a list of spectra with the button "Select path" in the "Data" tab (fig 56). After that background-file, calibration-file and lines-file must be chosen. Special peaks can be selected in the "Special Peak" panel. The process of the multi spectra fitting can be seen in the "current file" panel which shows the number of how many spectra has been fitted already. The "Fit results" tab is the same as in the single fit procedure. The name of the ASR-file will be the same as the SPE-file.

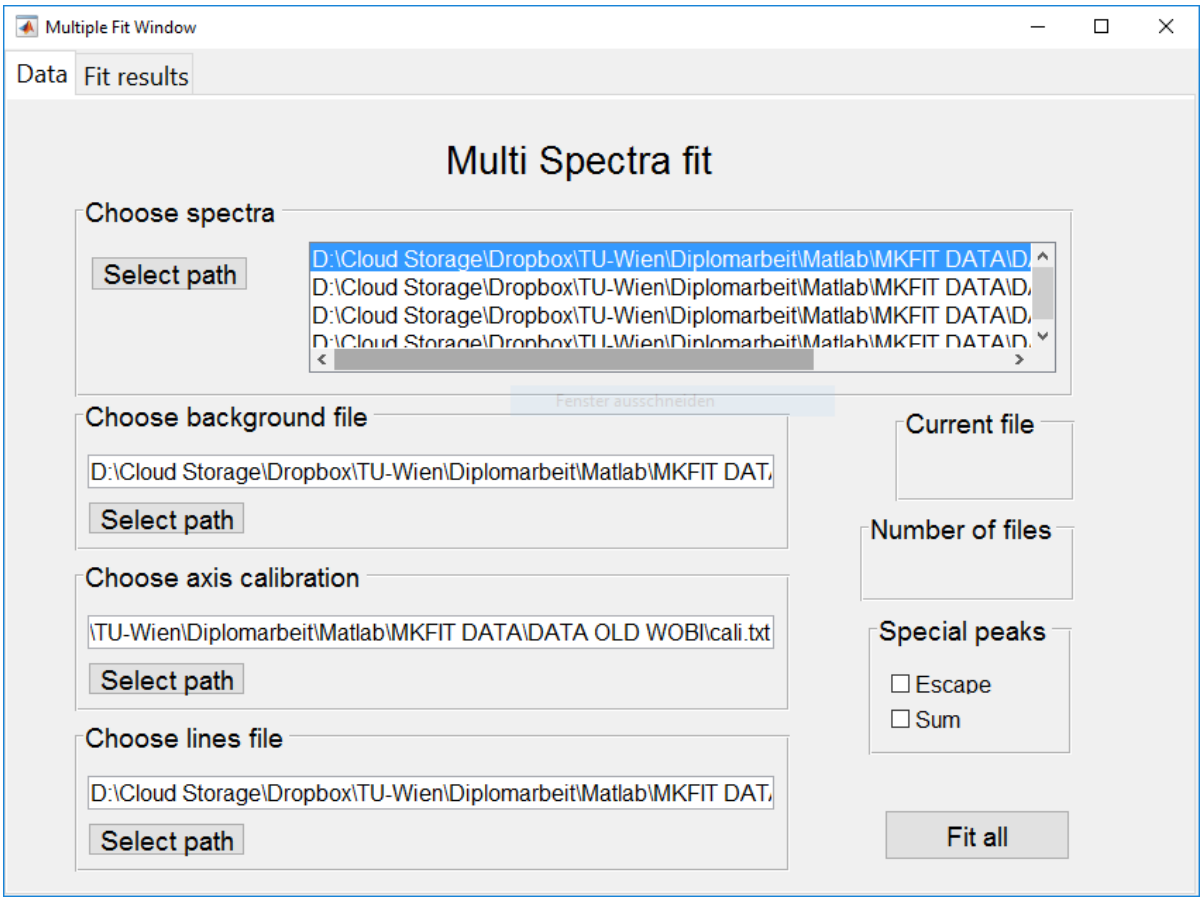


Figure 56: "Data" tab of the Fit multiple spectra Interface

6.7 Quantitative TXRF

With the QTXRF tool which can be accessed via the button "QTXRF" one can quantify spectra with a relative sensitivity file that has been created before.

6.7.1 Create new relative sensitivity file

First of all a relative sensitivity file has to be created by pressing the button "Create new sensitivity file". A dialog box opens and asks for the number of file-sets. A spectrum with the same elements but different concentrations for element sensitivity is one file-set.

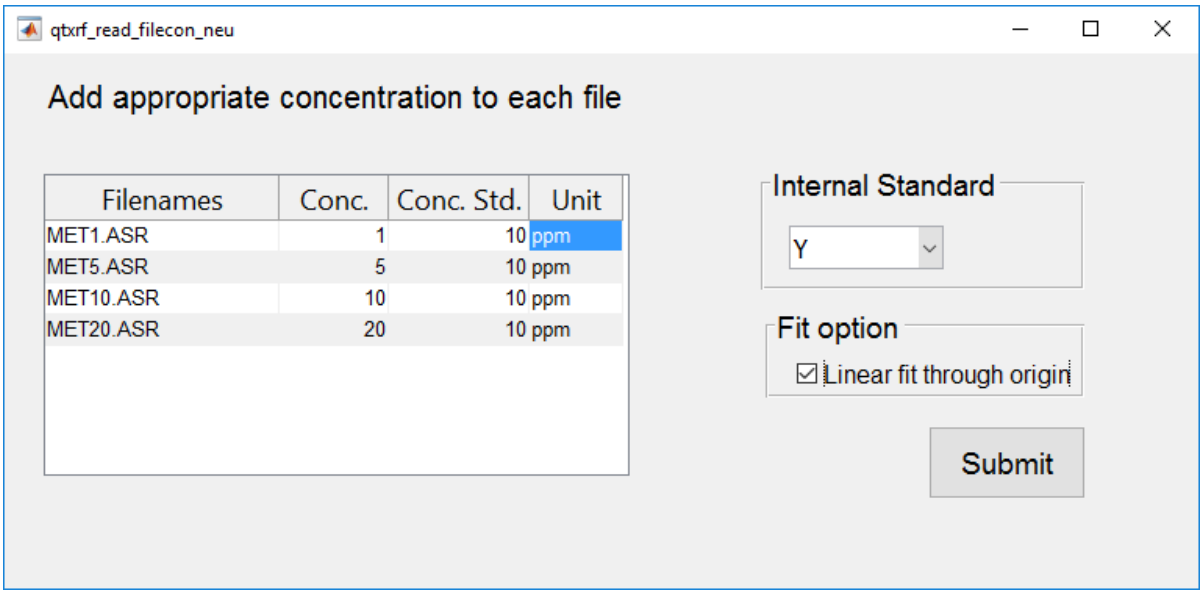


Figure 57: File concentration window

After submitting the number of file-sets the file concentration window appears which can be seen in figure 57. The concentration of the multi standard as well as the concentration of the internal standard must written into the table. After the internal standard and the fit option "Linear fit through origin" has been chosen, the button submit opens another window which can be seen in figure 58.

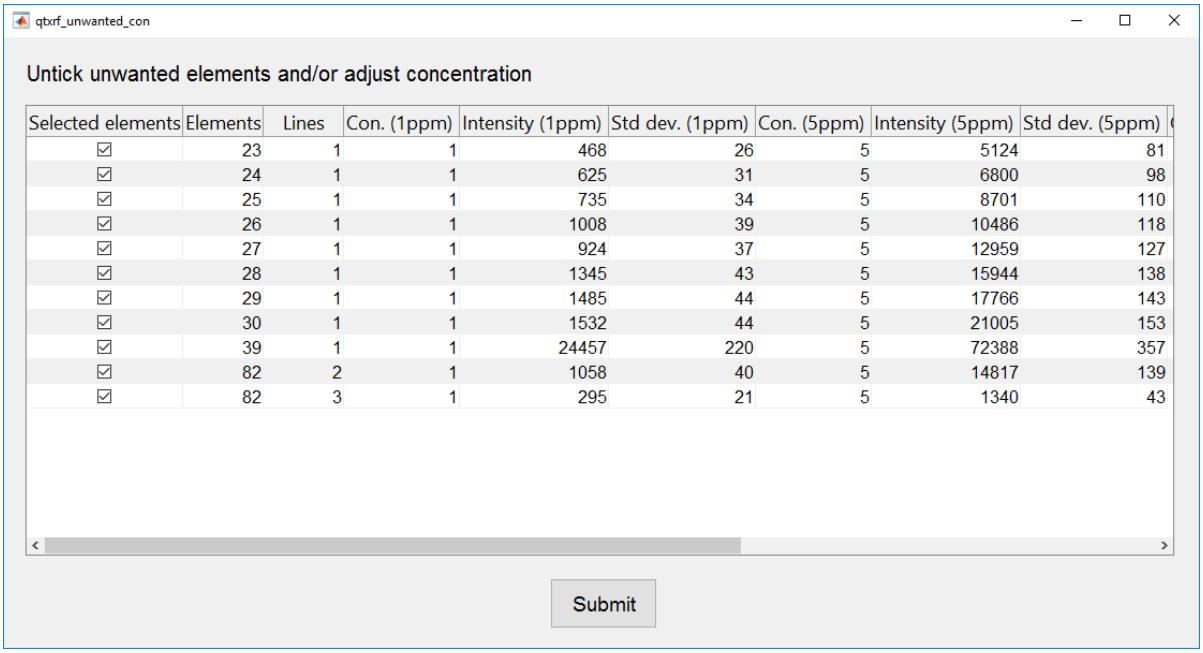


Figure 58: Unwanted elements window

In this window unwanted elements can be removed from the calibration by ticking the box in the first column. One can also change the concentration of each file. The button "Submit" starts the element sensitivity fit for all ticked elements. Figure 59 shows the fit result of one element with concentrations of 1ppm,5ppm,10ppm and 20ppm.

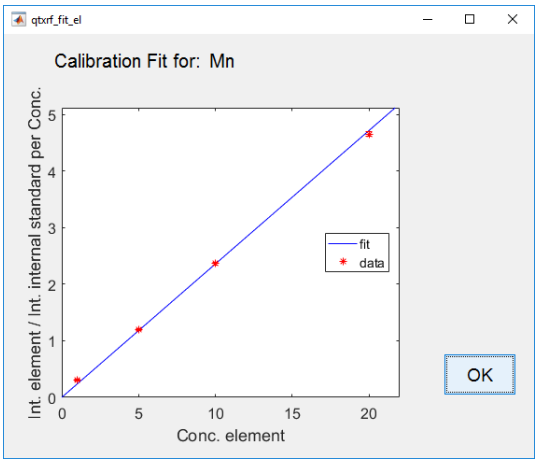


Figure 59: Fit result of one element with different concentrations

The "Fit relative sensitivity" window (fig 60 appears automatically when all the elements has been fitted. The chart to the right shows all the fitted element sensitivities while the chart to the left shows the sensitivities of the family that has been selected in the list box above. The button Fit/Cache fits the element sensitivities for the chosen family and polynomial degree and saves it internally. The button "Save" saves the relative sensitivity file to a to a specific location. This file and the ASR file are the foundation of the following quantification.

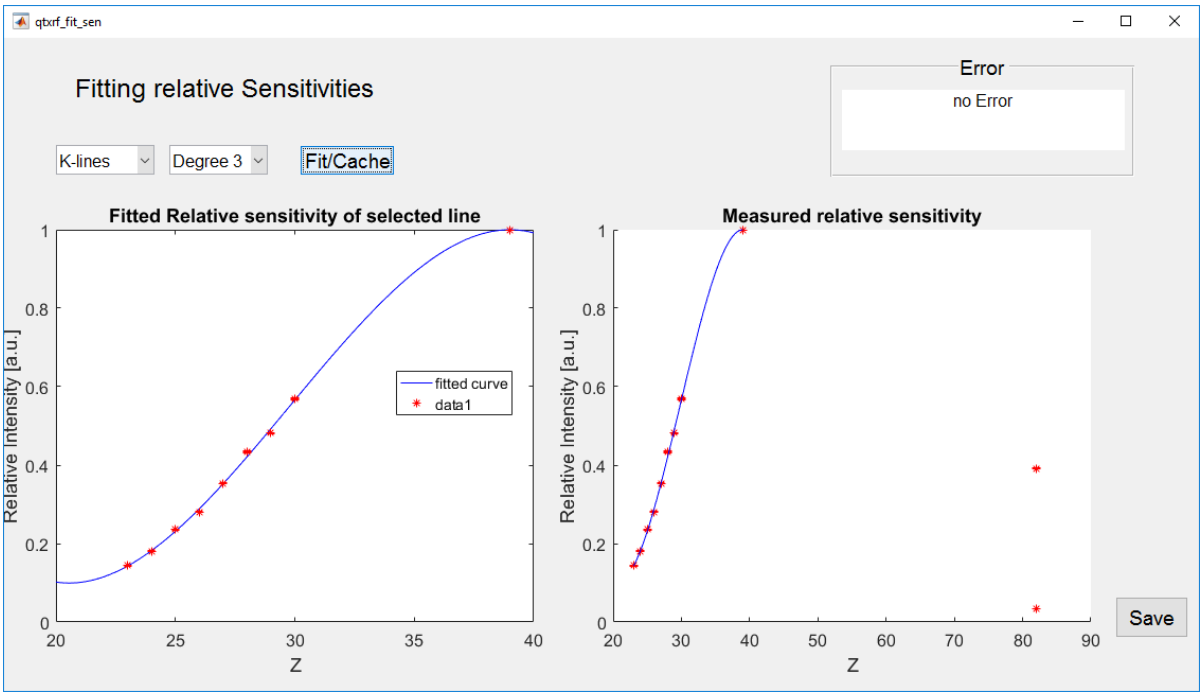


Figure 60: "Fit relative sensitivity" window

6.7.2 QTXRF Main Window

Figure 61 shows the Main Window of QTXRF. After setting the relative sensitivity file and the ASR file as well as the concentration of the internal standard on can quantify the chosen ASR file by pressing "Calculate". The button "Save" saves the result file to a specific location.

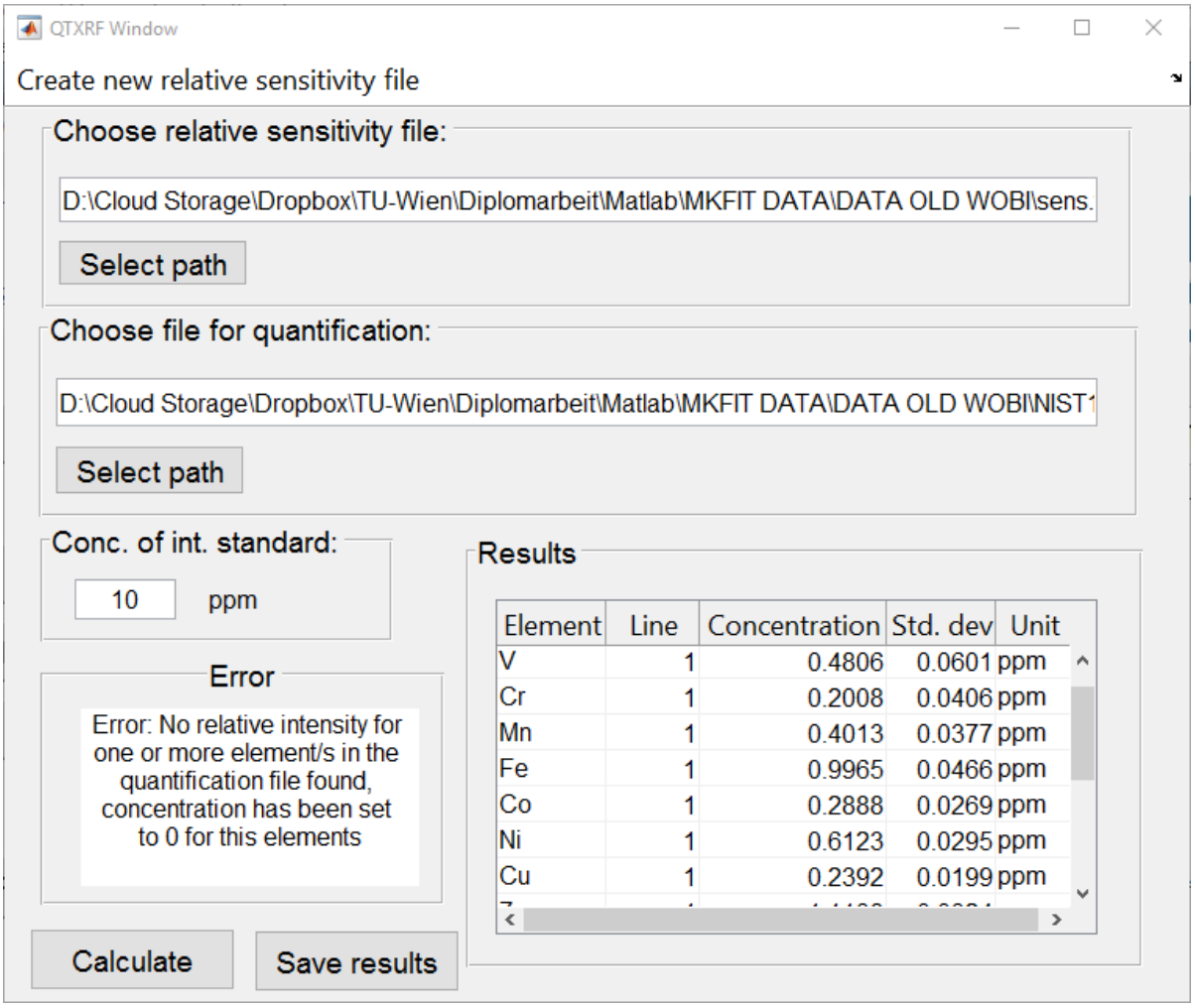


Figure 61: Main Window of the QTXRF tool

7 Measurements and Results

7.1 Detection Limits measured with Sr samples

3 different sample reflectors has been prepared with $1\mu l$ of a $10ppm$ Sr solution, so the mass on each sample holder ammounts to $10ng$ Sr. The Sr samples with the number 1,2 and 3 has been measured three times each with a live time of 300 seconds. The formula to calculate the detection limits can be found in chapter 6.5.7. Table 4 shows the limit of detection (LOD) of each sample with standard deviation (STD) in piko gram.

Sample no.	LOD [pg]	STD [pg]
1	33.3	± 1.8
2	39.6	± 3.9
3	31.8	± 2.8

Table 4: Detection Limits for 10ng Sr calculated with equation 34

Combining the three measurements results in a detection limit for this measurement setup of $35 \pm 3.2pg$.

7.2 Evaluation of the fitting capability of MKFit

The following chapter is a comparison of the well established evaluation program AXIL with the self programmed matlab GUI MK-Fit. Multi-element standard spectra and a spectra of NIST1640 has been chosen to compare the peak fitting capability. The following multi-element standards spectra has been used to compare and quantify:

Name	Elements	concentration [ppm]
MET20	V, Cr, Mn, Fe, Co, Ni, Cu, Zn, Pb, Y(Std)	20
MET10	V, Cr, Mn, Fe, Co, Ni, Cu, Zn, Pb, Y(Std)	10
MET05	V, Cr, Mn, Fe, Co, Ni, Cu, Zn, Pb, Y(Std)	5
MET01	V, Cr, Mn, Fe, Co, Ni, Cu, Zn, Pb, Y(Std)	1
ALK20	Rb, Sr, Ba, Y(Std)	20
ALK10	Rb, Sr, Ba, Y(Std)	10
ALK05	Rb, Sr, Ba, Y(Std)	5
ALK01	Rb, Sr, Ba, Y(Std)	1

Table 5: Multi-element spectra

Y has been chosen as the internal standard with a concentration of 10ppm in each sample. The first plot in the following subsections shows the result of the fit where the red line depicts the fit-function while the blue line depicts the background reduced spectrum. The second plot shows the residual plot of the fit-function. The green lines at ±3 is the statistical boundary. The bar plot afterwards compares the result of AXIL and MK-Fit with green and red bars and the table at the end contains the comparison including the relative deviation from MK-Fit to AXIL.

7.2.1 Multi element standard MET20

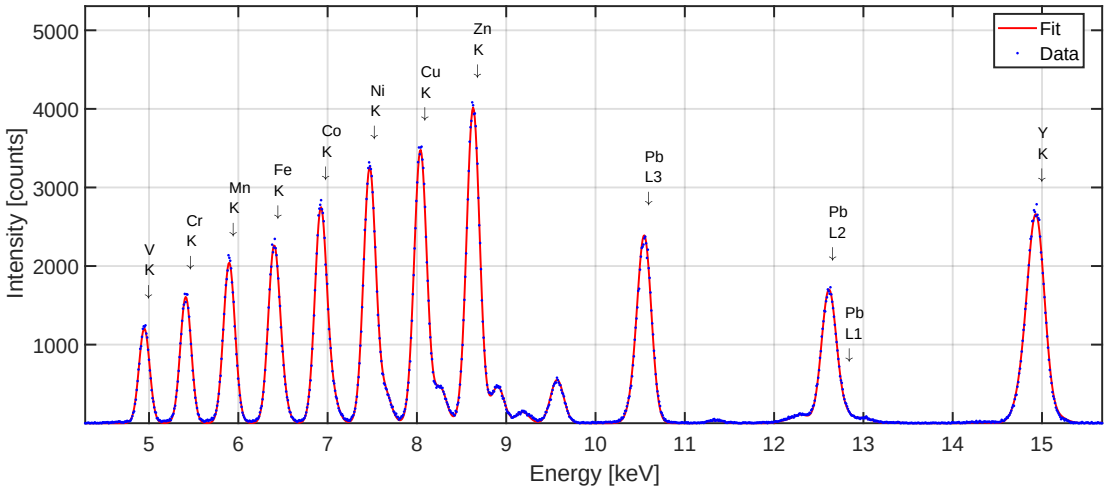


Figure 62: Fitresults MET20

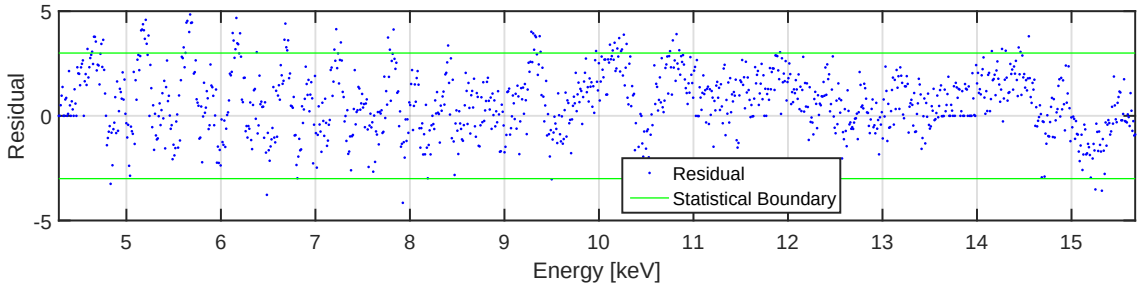


Figure 63: Residuals MET20

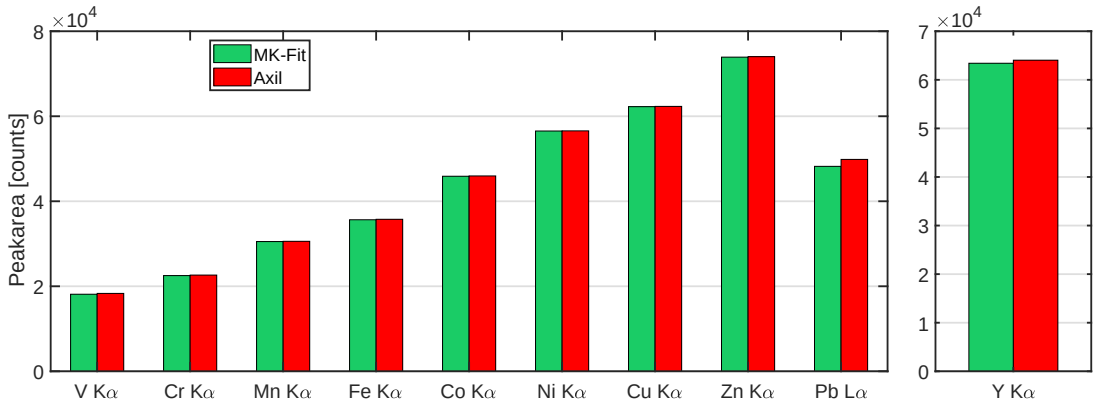


Figure 64: Comparison MET20

Element	Line	Peakarea			Std. Deviation	
		AXIL	MKFit	re. %	AXIL	MKFit
V	$K\alpha$	18130	18273	0.79	137	146
Cr	$K\alpha$	22514	22601	0.39	160	171
Mn	$K\alpha$	30521	30717	0.64	183	194
Fe	$K\alpha$	35653	35489	-0.46	195	207
Co	$K\alpha$	45888	46128	0.52	215	227
Ni	$K\alpha$	56530	56635	0.19	234	247
Cu	$K\alpha$	62257	62215	-0.07	242	256
Zn	$K\alpha$	73910	73757	-0.21	259	275
Y	$K\alpha$	63412	63947	0.84	268	293
Pb	$L\alpha$	48216	49784	3.25	158	247

Table 6: Comparison of the fitting results of file: MET20.SPE

7.2.2 Multi element standard MET10

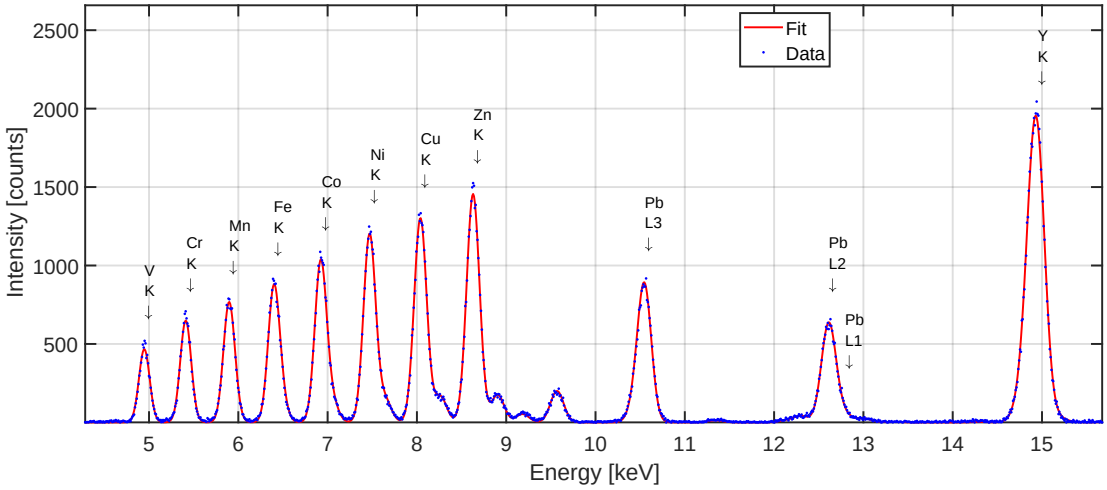


Figure 65: Fitresults MET10

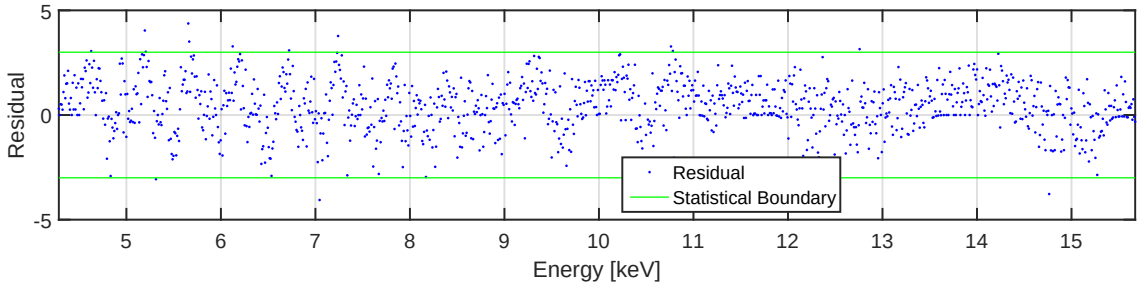


Figure 66: Residuals MET10

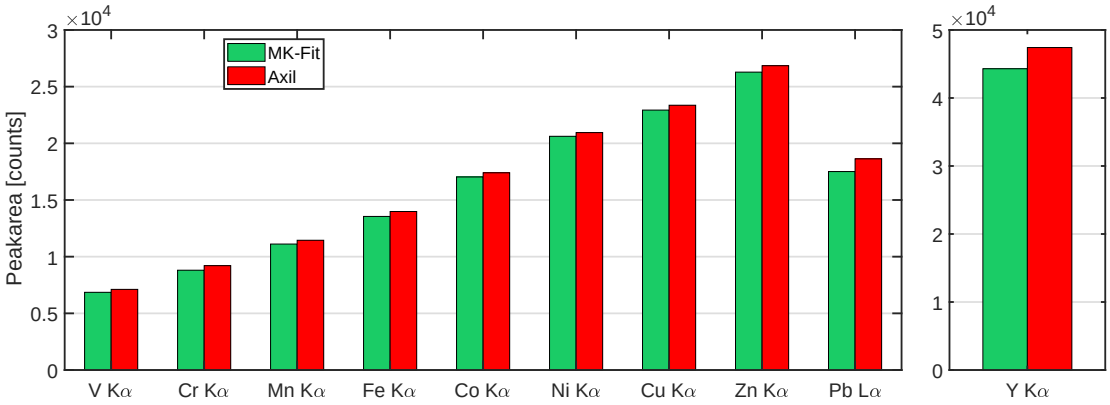


Figure 67: Comparison MET10

Element	Line	Peakarea			Std. Deviation	
		AXIL	MKFit	re. %	AXIL	MKFit
V	$K\alpha$	6856	6998	2.07	75	92
Cr	$K\alpha$	8806	9272	5.29	90	109
Mn	$K\alpha$	11114	11378	2.38	101	120
Fe	$K\alpha$	13550	14170	4.58	108	130
Co	$K\alpha$	17032	17440	2.40	118	141
Ni	$K\alpha$	20620	20876	1.24	128	151
Cu	$K\alpha$	22933	23426	2.15	133	158
Zn	$K\alpha$	26282	26894	2.33	142	167
Y	$K\alpha$	44300	47690	7.65	278	249
Pb	$L\alpha$	17509	18561	6.01	88	153

Table 7: Comparison of the fitting results of file: MET10.SPE

7.2.3 Multi element standard MET5

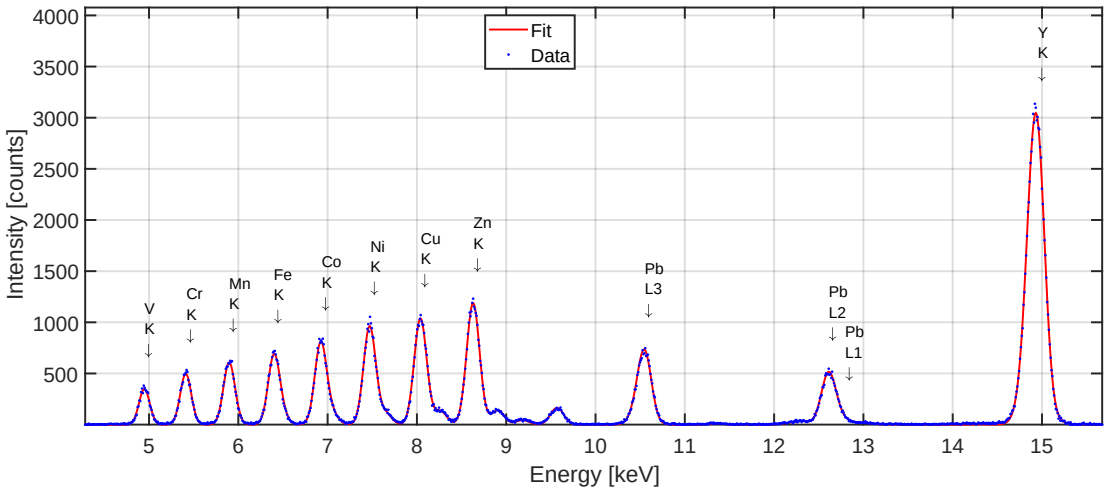


Figure 68: Fitresults MET5

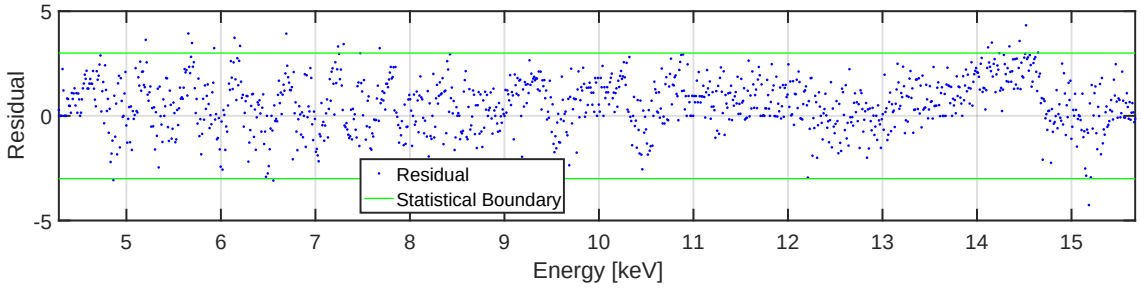


Figure 69: Residuals MET5

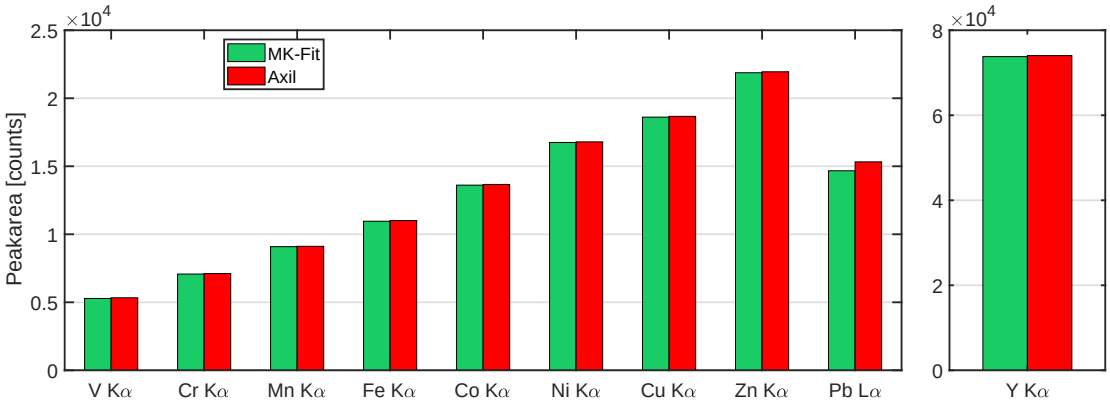


Figure 70: Comparison MET5

Element	Line	Peakarea			Std. Deviation	
		AXIL	MKFit	re. %	AXIL	MKFit
V	$K\alpha$	5281	5348	1.27	75	80
Cr	$K\alpha$	7076	7115	0.55	90	96
Mn	$K\alpha$	9088	9089	0.01	101	107
Fe	$K\alpha$	10960	11198	2.17	108	116
Co	$K\alpha$	13613	13593	-0.15	118	125
Ni	$K\alpha$	16750	16710	-0.24	128	136
Cu	$K\alpha$	18614	18678	0.34	133	141
Zn	$K\alpha$	21878	21934	0.26	142	151
Y	$K\alpha$	73790	73829	0.05	278	304
Pb	$L\alpha$	14670	15348	4.62	88	139

Table 8: Comparison of the fitting results of file: MET5.SPE

7.2.4 Multi element standard MET1

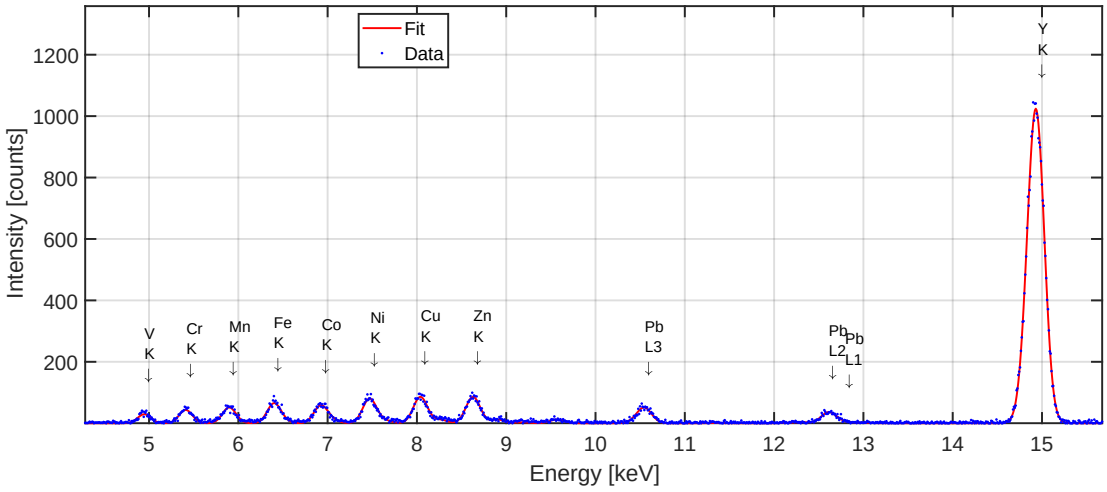


Figure 71: Fitresults MET1

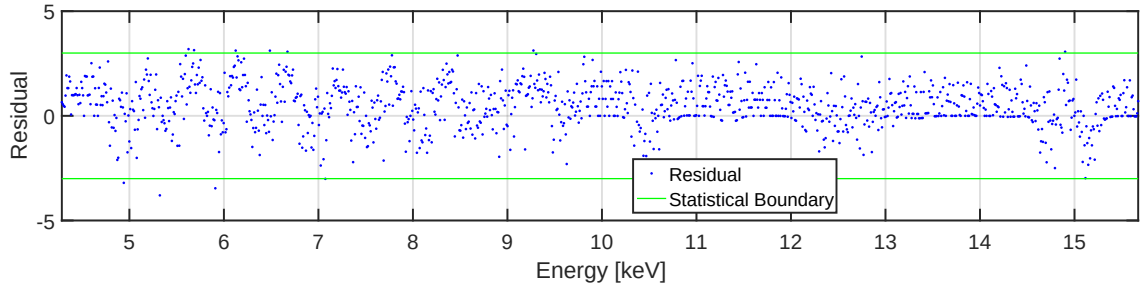


Figure 72: Residuals MET1

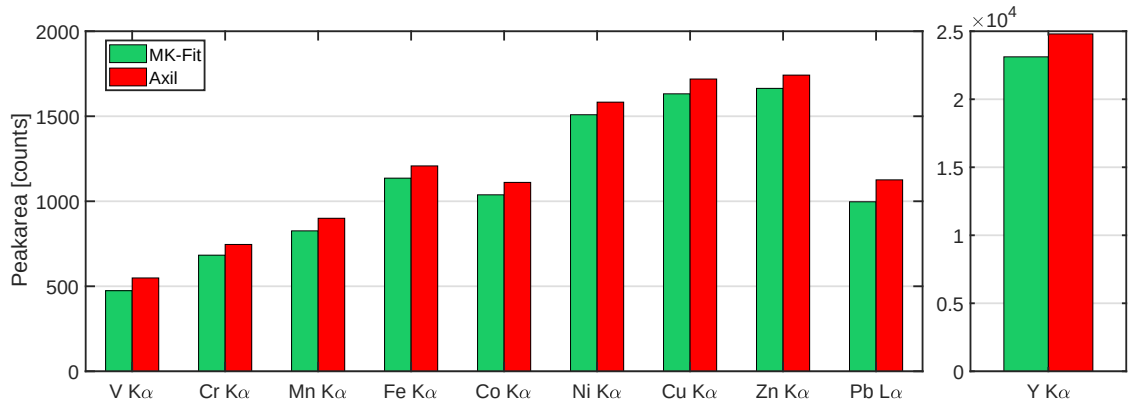


Figure 73: Comparison MET1

Element	Line	Peakarea			Std. Deviation	
		AXIL	MKFit	re. %	AXIL	MKFit
V	$K\alpha$	474	592	24.89	75	27
Cr	$K\alpha$	683	737	7.91	90	32
Mn	$K\alpha$	826	866	4.84	101	35
Fe	$K\alpha$	1136	1185	4.31	108	39
Co	$K\alpha$	1038	1146	10.40	118	38
Ni	$K\alpha$	1509	1558	3.25	128	43
Cu	$K\alpha$	1632	1736	6.37	133	45
Zn	$K\alpha$	1664	1707	2.58	142	45
Y	$K\alpha$	23119	24711	6.89	278	176
Pb	$L\alpha$	997	1163	16.65	88	41

Table 9: Comparison of the fitting results of file: MET1.SPE

7.2.5 Multi element standard ALK20

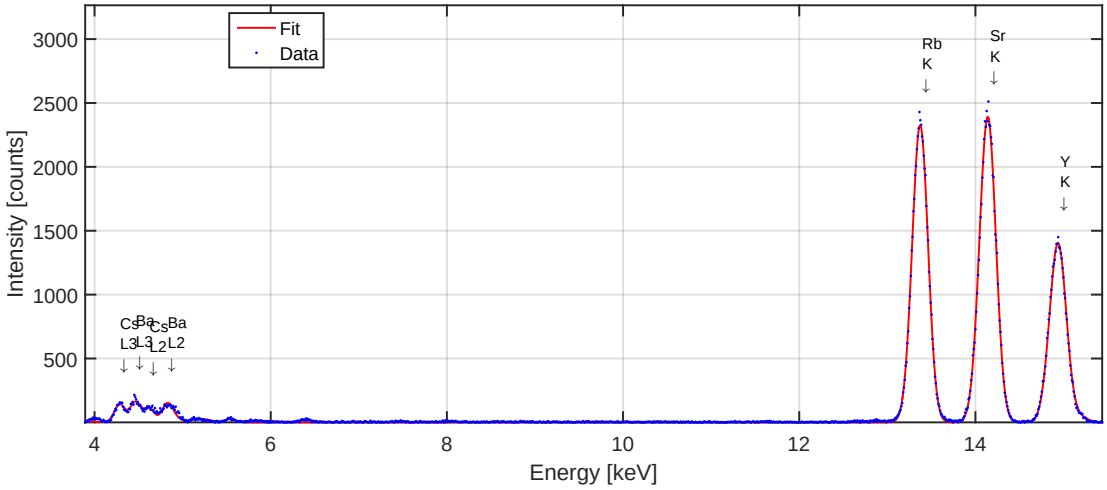


Figure 74: Fitresults ALK20

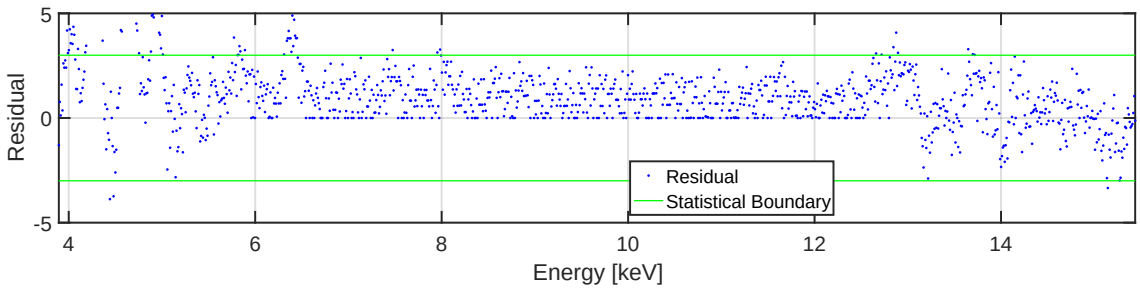


Figure 75: Residuals ALK20

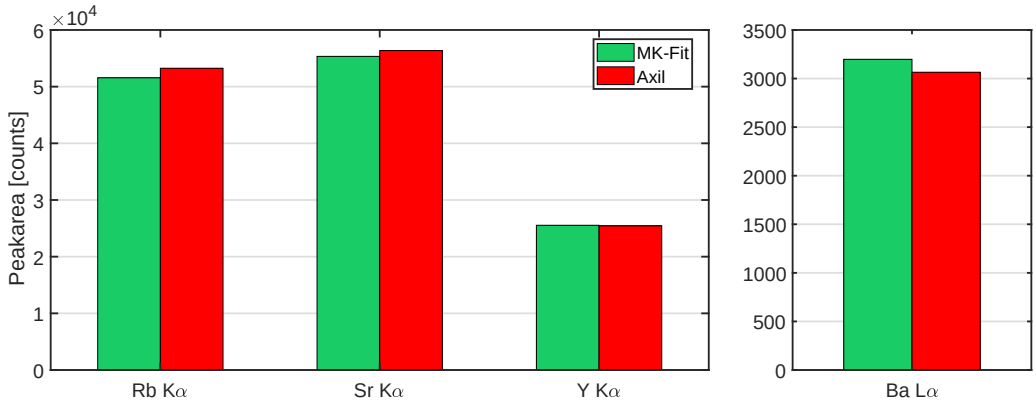


Figure 76: Comparison ALK20

Element	Line	Peakarea			Std. Deviation	
		AXIL	MKFit	re. %	AXIL	MKFit
Rb	$K\alpha$	51563	53247	3.27	230	251
Sr	$K\alpha$	55342	56358	1.84	238	261
Y	$K\alpha$	25539	25438	-0.40	191	210
Ba	$L\alpha$	3198	3065	-4.16	0	65

Table 10: Comparison of the fitting results of file: ALK20.SPE

7.2.6 Multi element standard ALK10

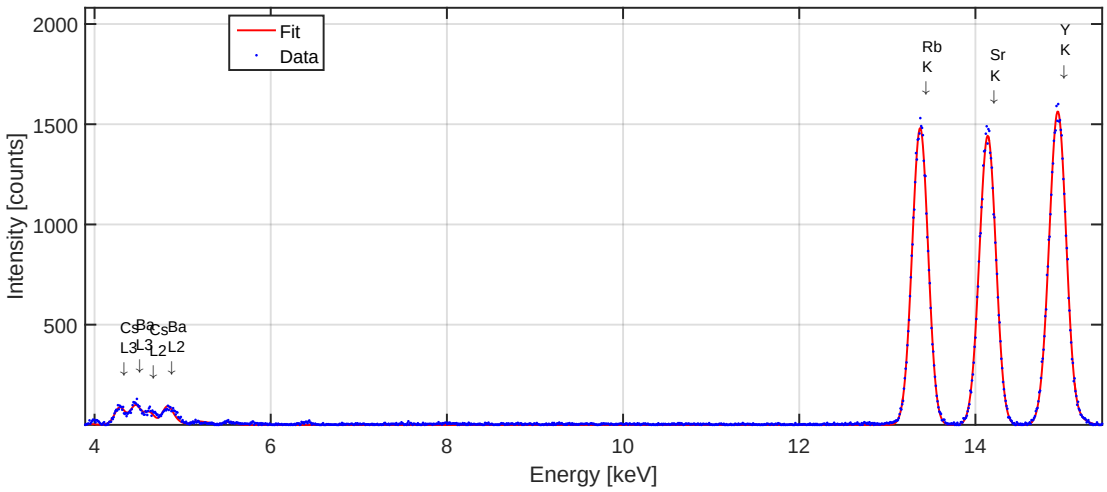


Figure 77: Fitresults ALK10

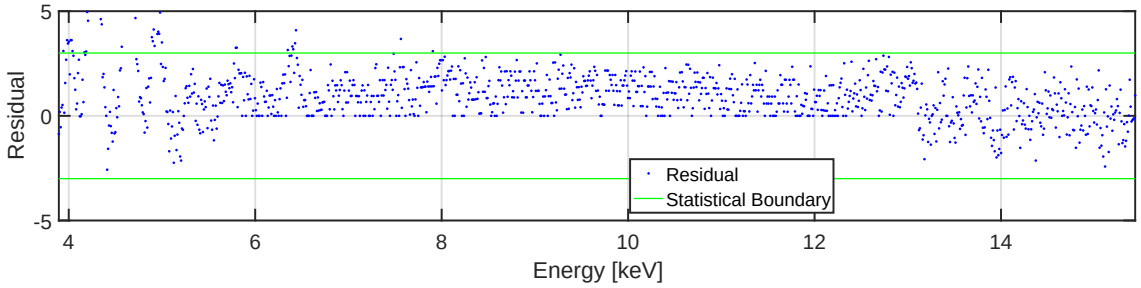


Figure 78: Residuals ALK10

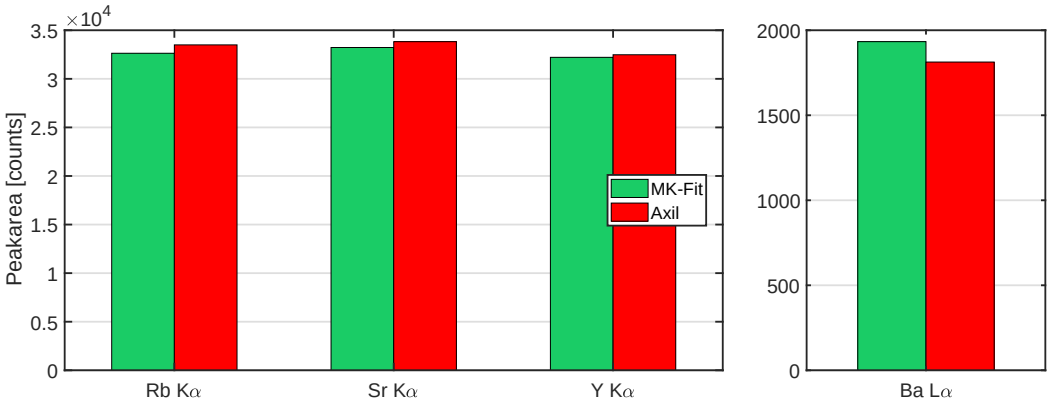


Figure 79: Comparison ALK10

Element	Line	Peakarea			Std. Deviation	
		AXIL	MKFit	re. %	AXIL	MKFit
Rb	$K\alpha$	32632	33495	2.64	230	201
Sr	$K\alpha$	33225	33825	1.81	238	203
Y	$K\alpha$	32213	32480	0.83	191	219
Ba	$L\alpha$	1933	1813	-6.21	0	51

Table 11: Comparison of the fitting results of file: ALK10.SPE

7.2.7 Multi element standard ALK5

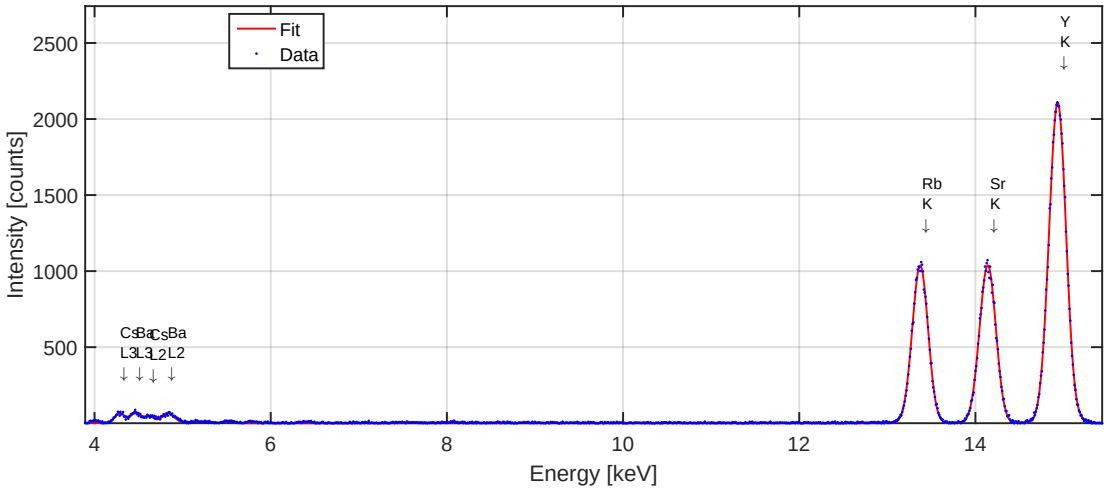


Figure 80: Fitresults ALK5

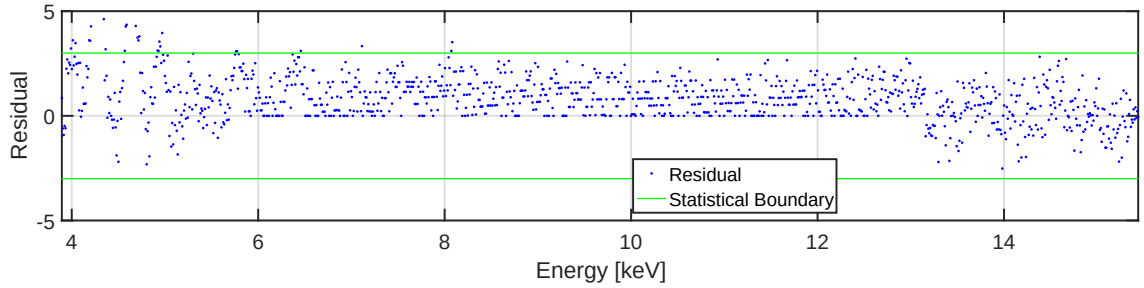


Figure 81: Residuals ALK5

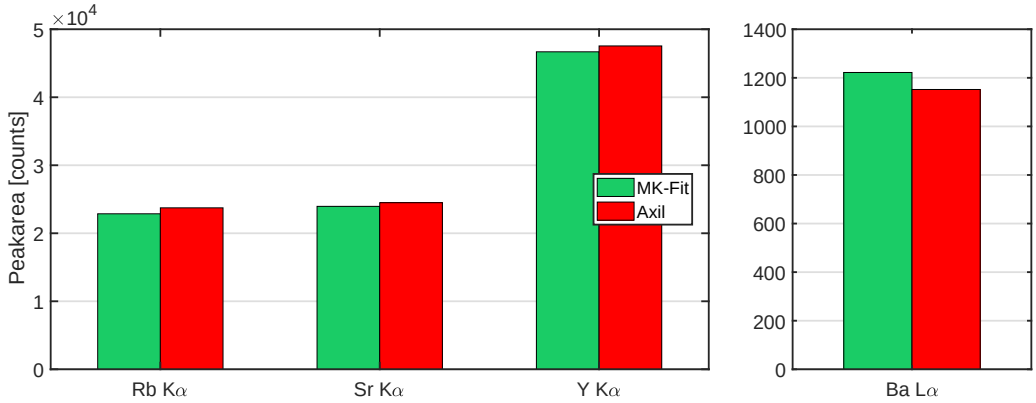


Figure 82: Comparison ALK5

Element	Line	Peakarea			Std. Deviation	
		AXIL	MKFit	re. %	AXIL	MKFit
Rb	$K\alpha$	22866	23742	3.83	230	169
Sr	$K\alpha$	23958	24497	2.25	238	173
Y	$K\alpha$	46684	47547	1.85	191	252
Ba	$L\alpha$	1222	1152	-5.73	0	42

Table 12: Comparison of the fitting results of file: ALK5.SPE

7.2.8 Multi element standard ALK1

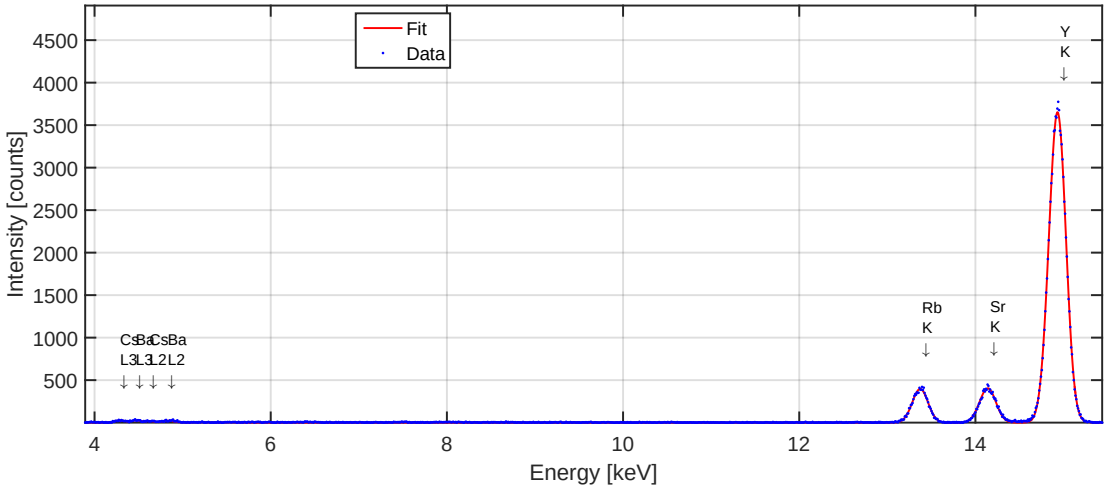


Figure 83: Fitresults ALK1

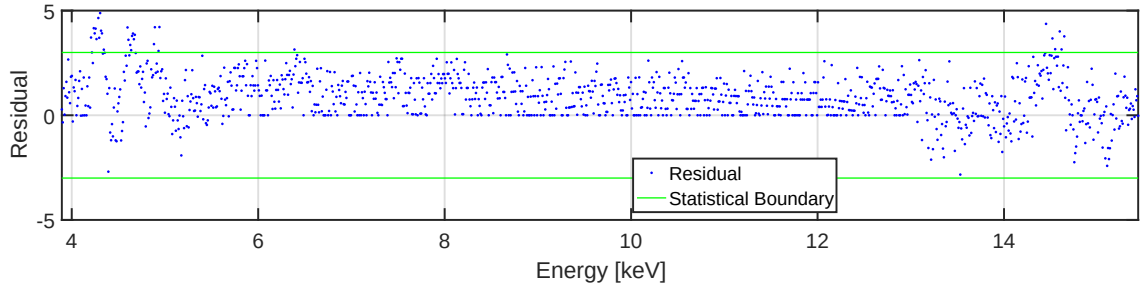


Figure 84: Residuals ALK1

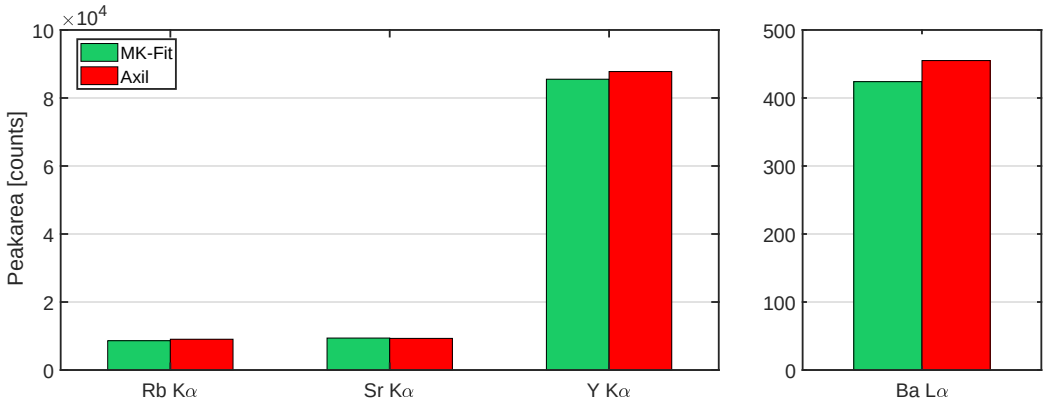


Figure 85: Comparison ALK1

Element	Line	Peakarea			Std. Deviation	
		AXIL	MKFit	re. %	AXIL	MKFit
Rb	$K\alpha$	8617	9053	5.06	230	106
Sr	$K\alpha$	9400	9302	-1.04	238	110
Y	$K\alpha$	85522	87770	2.63	191	328
Ba	$L\alpha$	424	455	7.31	0	28

Table 13: Comparison of the fitting results of file: ALK1.SPE

7.2.9 Sample NIST1643

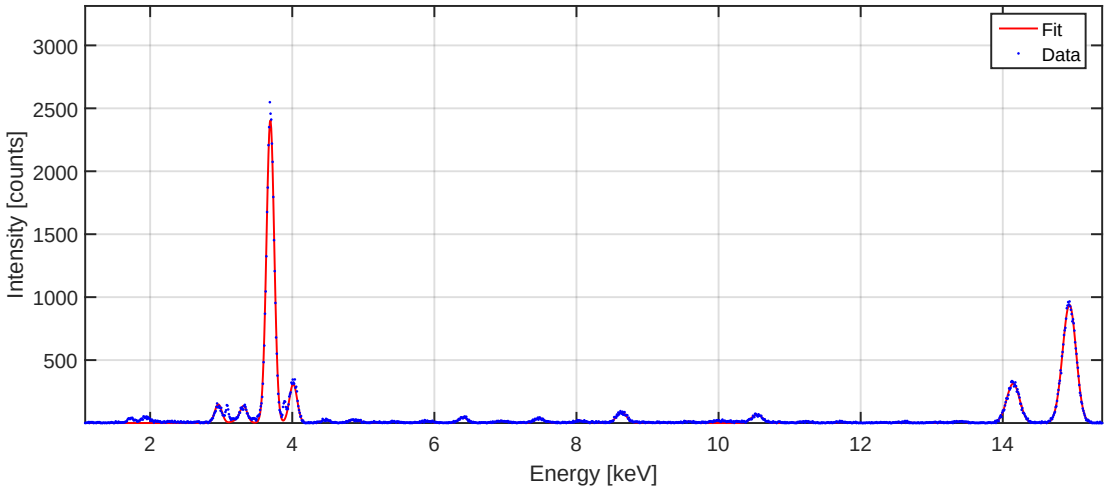


Figure 86: Fitresults NIST1643

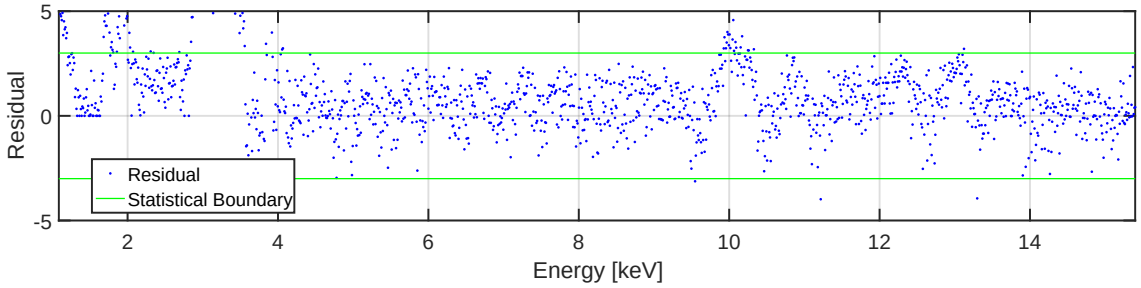


Figure 87: Residuals NIST1643

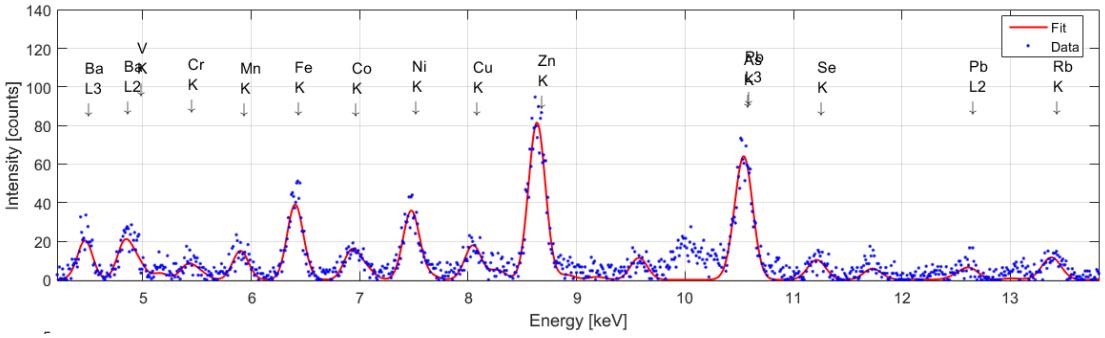


Figure 88: Zoomed fitresults NIST1643

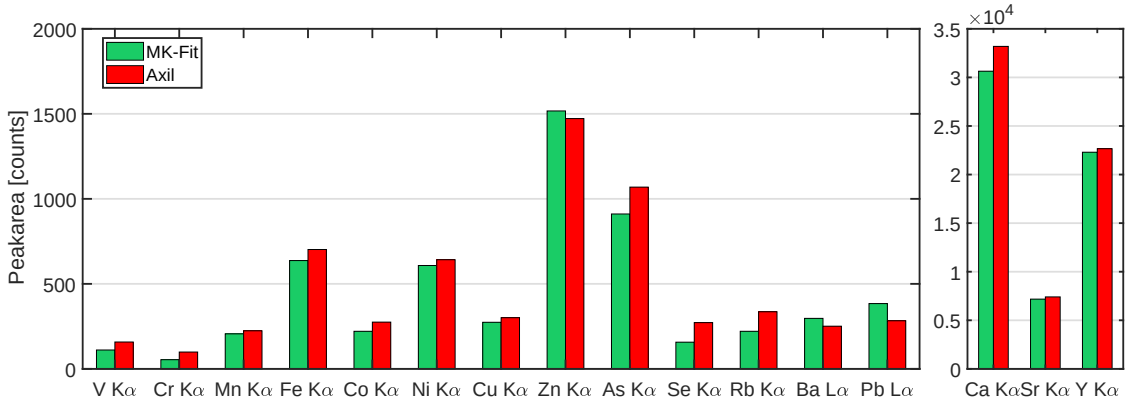


Figure 89: Comparison NIST1643

Element	Line	Peakarea			Std. Deviation	
		AXIL	MKFit	re. %	AXIL	MKFit
Ca	$K\alpha$	30638	33192	8.34	0	183
V	$K\alpha$	111	158	42.34	0	21
Cr	$K\alpha$	54	99	83.33	0	17
Mn	$K\alpha$	206	224	8.74	0	20
Fe	$K\alpha$	637	702	10.20	0	31
Co	$K\alpha$	221	275	24.43	0	22
Ni	$K\alpha$	608	642	5.59	0	29
Cu	$K\alpha$	274	301	9.85	0	23
Zn	$K\alpha$	1517	1472	-2.97	0	43
As	$K\alpha$	911	1069	17.34	0	111
Se	$K\alpha$	157	272	73.25	0	21
Rb	$K\alpha$	221	336	52.04	0	24
Sr	$K\alpha$	7179	7408	3.19	0	96
Y	$K\alpha$	22299	22661	1.62	0	167
Ba	$L\alpha$	297	251	-15.49	0	23
Pb	$L\alpha$	384	283	-26.30	0	123

Table 14: Comparison of the fitting results of file: NIST1643.SPE

7.3 Quantification of the Sample NIST1643

The evaluated multi-element spectra has then been used to create a element sensitivity file for quantification. Table 15 contains the results of the quantification. The certified value has been determined by the National Institute of Standards & Technology. The certificate of the NIST1643 sample is attached to the appendix.

Element	Concentration			Std. Deviation		
	Certified	AXIL	MKFit	Certified	AXIL	MKFit
V	0.3786	0.344	0.4806	±0.0059	±0.050	±0.0601
Cr	0.2040	0.131	0.2008	±0.0024	±0.022	±0.0406
Mn	0.3897	0.393	0.4013	±0.0045	±0.032	±0.0377
Fe	0.9810	0.968	0.9965	±0.0140	±0.054	±0.0466
Co	0.2706	0.272	0.2888	±0.0032	±0.018	±0.0269
Ni	0.6241	0.620	0.6123	±0.0069	±0.026	±0.0295
Cu	0.2276	0.238	0.2392	±0.0031	±0.018	±0.0199
Zn	0.7850	1.161	1.1138	±0.0220	±0.045	±0.0324
Rb	0.1414	0.133	0.1195	±0.0018	±0.01	±0.0107
Sr	3.2310	3.105	3.1959	±0.0360	±0.074	±0.0451

Table 15: Comparison of the NIST1643 sample

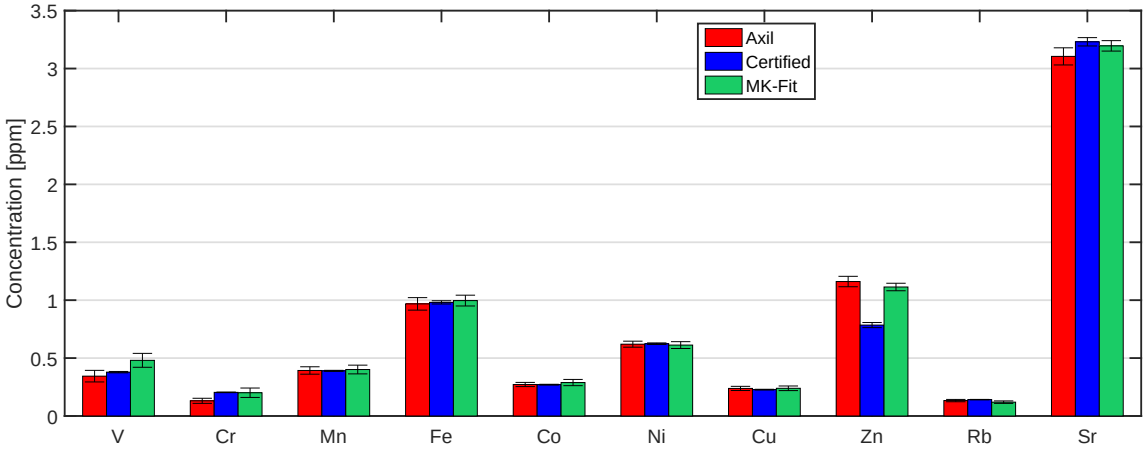


Figure 90: Bar plot including the error bars of the NIST1643 sample

As one can see in the bar plot above, the quantification of MK-Fit fits quite well the the results of AXIL and the certified value. However, there is a big deviation of the Zn concentration in the results from AXIL and MK-Fit. Since both evaluation program results have almost the same error relative to the certified value it is very likely that the the error is independent from the spectrum evaluation. The origin of the error is believed to be a contamination of the solution. Another possible error source of zinc is the detector itself since even spectra of blanc samples contain zinc.

8 Concluding remarks and perspectives

8.1 Concluding Remarks

The main goal of this thesis was it to built a total reflection x-ray fluorescence spectrometer in the x-ray lab located in the basement of the Institute of Atomic & Subatomic Physics and develop a program with GUI for spectra fitting and sample quantification. The spectrometer should be able to analyse high Z and low Z elements with two different, easily interchangeable X-ray tubes. The CAD model of the spectrometer which has been developed for a lab in India in 2015 formed the foundation of the new spectrometer. A few changes/improvements were carried out before the production of the individual parts.

The main improvements affect the rough z-and angle adjustment system, the reflex tracing system and the use of a new detector. The rough z-and angle adjustment system aids to pre adjust the height and the angle of the tube with its optics to ensure a proper beam alignment. The new reflex tracing system operates with a regular camera chip that has been covered with an aluminium foil. This foil blocks the low energy photons like visible light which makes the chip only sensitive for high energetic photons like x-rays. The use of a detector with a bigger detection area compared to the former version of the spectrometer further improved the detection limits of the device.

The spectrometer has been built up on an optical table to protect the device from vibrations and ensure a stable operation. The very accurate carriage system in x-direction ensures a proper position of the x-ray tube even after a few movements back and forward for maintenance purposes.

After completion of the spectrometer, three measurements were performed to determine the detection limits. The combination of the three measurements results in a detection limit of $35 \pm 3.2pg$ for a Sr sample with 10ng.

The self written program with a GUI is able to fit TXRF spectra and perform sample quantification. Multi-element standard spectra and a spectra of NIST1640 has been chosen to compare the peak fitting capability of the self written program MKFit and the well established evaluation program AXIL. Figure 90 shows the quantification results of the NIST1643 sample with AXIL and MKFit compared to value certified by the National Institute of Standards & Technology. The quantification results of the two programs AXIL and MKFit are both very close to the certified value which means that both programs can be used equally to fit and quantify TXRF spectra.

8.2 Perspectives

As well as in all physical instruments, there is also potential for improvement in this device. The second low power Cr x-ray tube is necessary for Low Z measurements in vacuum but it is not purchased yet. The additional tube can be mounted in the future with only a few small changes.
Another difficulty is the sample lifter that presses the sample holder against the plane which is supported by three micrometer screws. A very fine tuning is necessary to ensure a tight fit of the sample holder while not changing the angle of the plane.
A program is never complete and bugless, so there are a lot of features left that can be added in the future. The background estimation methods can be extended to ensure a better estimation for non TXRF spectra. Voigt peak profiles for high energy peaks could improve the fit accuracy of high Z elements like uranium. The option sum peaks is still not implemented properly.

References

[1] University of Basel. *Spectrum of an x-ray tube*. [Online; accessed May 3, 2017]. 2017. URL: https://miac.unibas.ch/PMI/01-BasicsOfXray-media/figs/Characteristic_Spectrum.png.

[2] Lloyd A. Currie. "Limits for qualitative detection and quantitative determination. Application to radiochemistry." In: *Analytical Chemistry* 40.3 (1968), pp. 586–593. DOI: 10.1021/ac60259a007. eprint: http://dx.doi.org/10.1021/ac60259a007. URL: http://dx.doi.org/10.1021/ac60259a007.

[3] W. Demtröder. *Experimentalphysik 3: Atome, Moleküle und Festkörper*. Experimentalphysik / Wolfgang Demtröder. Springer, 2010. ISBN: 978-3-642-03910-2.

[4] Thermo Fischer. *Crosssection of a silicon drift detector*. [Online; accessed May 6, 2017]. 2017. URL: https://www.thermofisher.com/blog/wp-content/uploads/2015/01/driftdesign1.jpg.

[5] Georg-August-Universität Göttingen. *Abb. 6624 Erlaubte Übergänge von einem höheren auf ein freies, niedrigeres Energieniveau*. [Online; accessed April 29, 2017; modified]. 2017. URL: https://lp.uni-goettingen.de/get/text/6634.

[6] Georg-August-Universität Göttingen. *Vergleich von IUPAC- und Siegsbahnnotation*. [Online; accessed April 29, 2017]. 2017. URL: https://lp.uni-goettingen.de/get/text/6634.

[7] G.Pepponi. "Synchrotron Radiation induced Total Reflection X-Ray Fluorescence Analysis applied to Material Science." PhD thesis. Vienna University of Technology, 2003.

[8] HyperPhysics. *X-ray tube*. [Online; accessed April 28, 2017; modified]. 2017. URL: http://hyperphysics.phy-astr.gsu.edu/hbase/quantum/imgqua/xtube.gif.

[9] IFG. *CSU - Control and Supply Unit*. [Online; accessed May 28, 2017]. 2017. URL: http://www.ifg-adlershof.de.

[10] Inspire. *A comparison of the simulated intrinsic efficiencies for a TITAN Si(Li) detector and an HPGe detector*. [Online; accessed May 19, 2017]. 2017. URL: https://inspirehep.net/record/1298424/files/EffCompare.png.

[11] SAKURAI Kenji. *Principle of total-reflection X-ray fluorescence (TXRF) technique (reflectivity and penetration depth for Si substrate, 8keV x-rays)*. [Online; accessed May 19, 2017]. 2017. URL: https://user.spring8.or.jp/sp8info/?p=1746.

[12] Ketek. *AXAS-M*. [Online; accessed May 28, 2017]. 2017. URL: https://www.ketek.net/sdd/complete-systems/axas-m.

[13] Reinhold Klockenkämper. *Total Reflection X-Ray Fluorescence Analysis*. Experimentalphysik / Wolfgang Demtröder. John Wiley & Sons, 1997. ISBN: 0-471-30524-3.

[14] H. A. Kramers. "XCIII. On the theory of X-ray absorption and of the continuous X-ray spectrum." In: *Philosophical Magazine* 46.275 (1923), pp. 836–871. DOI: 10.1080/14786442308565244. eprint: http://dx.doi.org/10.1080/14786442308565244. URL: http://dx.doi.org/10.1080/14786442308565244.

[15] American Pharmaceutical. *EDXRF sheme*. [Online; accessed May 5, 2017]. 2017. URL: http://media.americanpharmaceuticalreview.com/m/28/article/124874-1.jpg.

[16] Univ.Ass. Dipl.-Ing. Josef Prost. "Technische Verbesserungen an einem Spektrometer für Niederleistungs-Totalreflexions-Röntgenfluoreszenzanalyse und Anwendungen bei Umweltproben." Master Thesis. TU-Vienna, Atomistitut, 2012.

[17] Norwegian University of Science and Technology. *Moseley's law*. [Online; accessed May 3, 2017]. 2017. URL: http://folk.ntnu.no/floban/KJ3055/XRay/Moseley/Moseleylaw_files/image010.gif.

[18] S.R. Sternberg. "Biomedical Image Processing." In: *Computer* 16 (1983), pp. 22–34. ISSN: 0018-9162. DOI: doi.ieeecomputersociety.org/10.1109/MC.1983.1654163.

[19] Ao Univ.Prof.Dipl.Ing.Dr. Christina Streli. "134.048 Physikalische Analytik." In: *XRP1-Script* (2015).

[20] Ao Univ.Prof.Dipl.Ing.Dr. Christina Streli. "141.211 X-Ray Analytical Methods." In: *Script* (2016).

[21] A.C. Thompson et al. *X-ray Data Booklet*. Lawrence Berkeley Laboratory, 2009.

[22] R.E. Van Grieken and A.A. Markowicz. *Handbook of X-ray Spectrometry: Methods and Techniques*. Drugs and the Pharmaceutical Sciences. Marcel Dekker, Incorporated, 1993. ISBN: 0-8247-8483-9.

[23] Wikipedia. *EM-Spectrum*. [Online; accessed April 28, 2017]. 2017. URL: https://commons.wikimedia.org/wiki/File_talk:EM_spectrum.svg.

[24] Wikipedia. *Plot of the centered Voigt profile for four cases*. [Online; accessed August 28, 2017]. 2017. URL: https://en.wikipedia.org/wiki/Voigt_profile#/media/File:VoigtPDF.svg.

[25] Wikipedia. *Structure of a Si(Li) detector crystal*. [Online; accessed May 28, 2017]. 2017. URL: https://de.wikipedia.org/wiki/Si(Li)-Detektor#/media/File:DmedxrfSiLiDetector.jpg.

[26] Wikipedia. *X-ray range*. [Online; accessed April 28, 2017]. 2017. URL: https://commons.wikimedia.org/wiki/File:X-ray_range.svg.

[27] Wikiwand. *WDXRF sheme*. [Online; accessed May 5, 2017; modified]. 2017. URL: https://upload.wikimedia.org/wikipedia/commons/thumb/a/a8/Analyse_wdx.svg/440px-Analyse_wdx.svg.png.

[28] P. Wobrauschek et al. "A novel vacuum spectrometer for total reflection x-ray fluorescence analysis with two exchangeable low power x-ray sources for the analysis of low, medium, and high Z elements in sequence." In: *Review of Scientific Instruments* (2015).

A Data sheets

A.1 NIST 1643e data sheet

National Institute of Standards & Technology

Certificate of Analysis

Standard Reference Material® 1643e

Trace Elements in Water

This Standard Reference Material (SRM) is intended primarily for use in evaluating methods used in the determination of trace elements in fresh water. SRM 1643e consists of approximately 250 mL of acidified water in a polyethylene bottle, which is sealed in an aluminized plastic bag to maintain stability. SRM 1643e simulates the elemental composition of fresh water. Nitric acid is present at a concentration of approximately 0.8 mol/L to stabilize the trace elements.

Certified Values: The certified values for 29 elements in SRM 1643e are listed in Table 1. All values are reported both as mass fractions (µg/kg) and as mass concentrations (µg/L). A NIST certified value is a value for which NIST has the highest confidence in its accuracy in that all known or suspected sources of bias have been investigated or accounted for by NIST [1]. The certified values are the average of the gravimetrically prepared value and a value determined by either inductively coupled plasma mass spectrometry (ICP-MS) or inductively coupled plasma optical emission spectrometry (ICP-OES). The expanded uncertainty for each certified value is calculated as

$$U = ku_c$$

where k is the coverage factor for a 95 % confidence interval and u_c is the combined standard uncertainty calculated according to the ISO and NIST Guides [2]. The value of u_c is intended to represent, at the level of one standard deviation, the combined effect of uncertainty components associated with the gravimetric preparation, the ICP-MS or ICP-OES determination, method bias [3], and stability.

Information Value: A rhenium values is listed in Table 2 for information purposes only. An information value is considered to be a value that will be of interest to the SRM user, but insufficient information is available to assess the uncertainty associated with the value [1]. The information value is based on results from one NIST method.

Expiration of Certification: This certification of SRM 1643e is valid, within the measurement uncertainties specified, until **31 March 2014**, provided the SRM is handled in accordance with instructions given in this certificate. This certification is nullified if the SRM is damaged, contaminated, or modified.

Maintenance of SRM Certification: NIST will monitor this SRM over the period of its certification. If substantive technical changes occur that affect the certification before the expiration of this certification, NIST will notify the purchaser. Registration (see attached sheet) will facilitate notification.

Coordination of the NIST technical measurements was under the direction of T.A. Butler and G.C. Turk of the NIST Analytical Chemistry Division. The ICP-MS analyses were performed by T.A. Butler, L.L. Yu, and G.C. Turk. The ICP-OES analyses were performed by T.A. Butler and G.C. Turk.

Statistical analysis of the experimental data was performed by S.D. Leigh and D.D. Leber of the NIST Statistical Engineering Division.

The support aspects involved in the issuance of this SRM were coordinated through the NIST Measurement Services Division.

Stephen A. Wise, Chief
Analytical Chemistry Division

Gaithersburg, MD 20899
Certificate Issue Date: 26 March 2009
See Certificate Revision History on Last Page

Robert L. Watters, Jr., Chief
Measurement Services Division

Table 1. Certified Values, Expanded Uncertainties, and Coverage Factors for Trace Elements in SRM 1643e

Element	Mass Fraction (µg/kg)			Mass Concentration (µg/L)			*k*
Aluminum	138.33	±	8.4	141.8	±	8.6	3.2
Antimony	56.88	±	0.60	58.30	±	0.61	2.0
Arsenic	58.98	±	0.70	60.45	±	0.72	2.0
Barium	531.0	±	5.6	544.2	±	5.8	2.0
Beryllium	13.64	±	0.16	13.98	±	0.17	2.0
Bismuth	13.75	±	0.15	14.09	±	0.15	2.0
Boron	154.0	±	3.8	157.9	±	3.9	2.4
Cadmium	6.408	±	0.071	6.568	±	0.073	2.0
Calcium	31 500	±	1 100	32 300	±	1 100	2.8
Chromium	19.90	±	0.23	20.40	±	0.24	2.0
Cobalt	26.40	±	0.32	27.06	±	0.32	2.0
Copper	22.20	±	0.31	22.76	±	0.31	2.1
Iron	95.7	±	1.4	98.1	±	1.4	2.0
Lead	19.15	±	0.20	19.63	±	0.21	2.0
Lithium	17.0	±	1.7	17.4	±	1.7	3.2
Magnesium	7 841	±	96	8 037	±	98	2.0
Manganese	38.02	±	0.44	38.97	±	0.45	2.0
Molybdenum	118.5	±	1.3	121.4	±	1.3	2.0
Nickel	60.89	±	0.67	62.41	±	0.69	2.0
Potassium	1 984	±	29	2 034	±	29	2.1
Rubidium	13.80	±	0.17	14.14	±	0.18	2.0
Selenium	11.68	±	0.13	11.97	±	0.14	2.0
Silver	1.036	±	0.073	1.062	±	0.075	3.2
Sodium	20 230	±	250	20 740	±	260	2.0
Strontium	315.2	±	3.5	323.1	±	3.6	2.0
Tellurium	1.07	±	0.11	1.09	±	0.11	3.2
Thallium	7.263	±	0.094	7.445	±	0.096	2.0
Vanadium	36.93	±	0.57	37.86	±	0.59	2.1
Zinc	76.5	±	2.1	78.5	±	2.2	2.6

Table 2. Information Value for Trace Elements in SRM 1643e

Element	Mass Fraction (µg/kg)	Mass Concentration (µg/L)
Rhenium	110	113

Preparation of Material: SRM 1643e was prepared at NIST using only high purity reagents. The containers were acid cleaned before use. In the preparation, a polyethylene cylindrical tank was filled with deionized water and sufficient nitric acid to make the solution approximately 0.8 mol/L. Known masses of the matrix elements (sodium, potassium, calcium, and magnesium) were added to the tank solution as solutions prepared from the same primary materials used to prepare the SRM 3100 Series of Single Element Solutions. Known masses of the other elements were then added to the tank solution using weighed aliquots of the SRM 3100 Series. The final total mass of the tank solution was determined, allowing calculation of the gravimetrically prepared mass fraction for each element. Mass concentrations were calculated using the measured density of 1.025 g/mL. After mixing thoroughly, the solution was transferred to clean 250 mL polyethylene bottles.

INSTRUCTIONS FOR USE

Precautions: The SRM should be shaken before use because of possible water condensation. To prevent possible contamination of the SRM, **DO NOT** insert pipettes into the bottle. Samples should be decanted at a room temperature of 17 ºC to 27 ºC. After use, the bottle should be recapped tightly and returned to the aluminized plastic bag, which should be folded and sealed with sealing tape. This safeguard will protect the SRM from possible environmental contamination and long-term evaporation.

The accuracy of trace element determinations, especially at the µg/L level, is limited by contamination. Apparatus should be scrupulously cleaned and only high purity reagents employed. Sampling and manipulations, such as evaporations, should be done in a clean environment, such as a Class-100 clean hood.

REFERENCES

[1] May, W.; Parris, R.; Beck, C.; Fassett, J.; Greenberg, R.; Guenther, F.; Kramer, G., Wise; S., Gills, T.; Colbert, J.; Gettings, R.; MacDonald, B.; *Definition of Terms and Modes Used at NIST for Value-Assessment of Reference Materials for Chemical Measurements*; NIST Special Publication 260-136, U.S. Government Printing Office: Washington, DC (2000).

[2] ISO; *Guide to the Expression of Uncertainty in Measurement*, ISBN 92-67-10188-9, lst ed. International Organization for Standardization, Geneva, Switzerland, (1993); see also Taylor, B.N.; Kuyatt, C.E.; *Guidelines for Evaluating and Expressing the Uncertainty of NIST Measurement Results*; NIST Technical Note 1297, U.S. Government Printing Office, Washington, DC (1994); (available at http://physics.nist.gov/Pubs/).

[3] Levenson, M.S.; Banks, D.L.; Eberhart, K.R.; Gill, L.M.; Guthrie, W.F.; Liu, H.K.; Vangel, M.G.; Yen, J.H.; Zhang, N.F.; *An Approach to Combining Results From Multiple Methods Motivated by the ISO GUM,* J. Res. Natl. Inst. Stand. Technol., Vol. 105, p. 521 (2000).

Certificate Revision History: 26 March 2009 (Addition of a Rhenium information value and editorial revisions); 16 March 2004 (Original certificate date).

Users of this SRM should ensure that the certificate in their possession is current. This can be accomplished by contacting the SRM Program at: telephone (301) 975-2200; fax (301) 926-4751; e-mail srminfo@nist.gov; or via the Internet at http://www.nist.gov/srm.

A.2 iMOXS_MFR data sheet

iMOXS® MFR

IFG Institute for Scientific Instruments
Member of the Helmut Fischer Group

Compact X-ray source for XRF and XRD
A Modular System

iMOXS® is a modular and high brightness X-ray source developed for a multitude of applications. The system is based on a micro-focus, side-window, air-cooled, and cost effective X-ray tube. In combination with different types of X-ray optics, a highly intensive parallel or convergent beam can be generated.

The radiation source consists of the following components

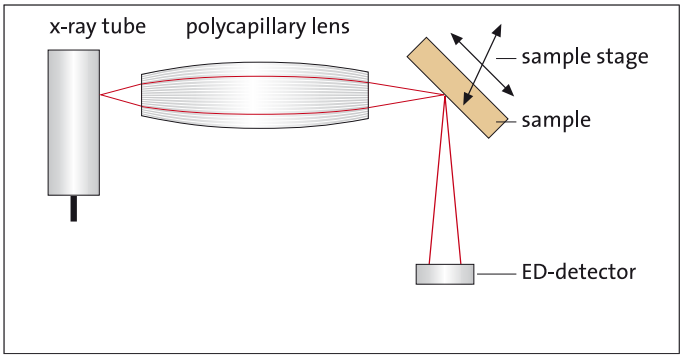
- Micro-focus, air cooled, low-power X-ray tube, available with various target materials
- Tube housing for radiation and magnetic protection with shutter, filter wheel and adjustment devices for X-ray optics.
- Tubes are pre-aligned adjusted in order to allow easy and rapid exchange with minimal effect upon the spot position. This enables the use of tubes with different anode materials without major alignment procedures
- X-ray optics on request according to the special requirements of an application. In general there are available cylindrical or shaped mono-capillaries, poly-capillary lenses or half lenses etc.
- Control and supply unit adapted to the tube requirements with HV supply, filament heating and safety lock circuitry for radiation safety. Parameter setting manually or via PC
- Mounting platform for installation of XRF, XRD or other devices according to user demand

The source is designed according to the German Röntgenverordnung and has the approval of the German Federal Office for Radiation Protection (BfS).

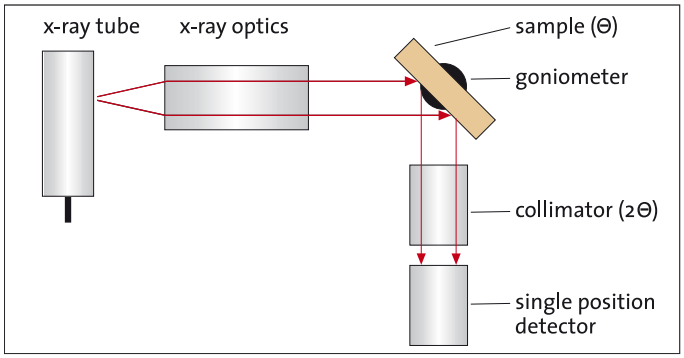
Highest brightness due to tube spots down to 50 µm

Small and compact design for small footprint and portable applications

Combination with different X-ray optics for a wide variety of applications



Scheme of an ED-micro-XRF spectrometer



Scheme of an XRD setup with X-ray optics

Examples of Application

XRF and micro XRF

The source can be used for the excitation of X-ray fluorescence. In this case heavy anode materials like Rh, Mo or W are used. Usually X-ray fluorescence is a typical method for analysis of large sample areas of homogeneous specimens. By use of focussing X-ray optics, for example capillary optics, convergent polychromatic beams of high intensity can be generated. Small sample areas down to 10 µm can be excited which enables position sensitive micro-XRF with good spatial resolution. Since the X-ray beam is fixed, the examination of linear or area distributions can be achieved by moving the sample mounted on a stage. An overview of the arrangement of an ED-XRF experiment with spatial resolution is shown in the top left figure. In comparison to electron excitation, which offers high spatial resolution, the benefit of micro-XRF is a higher sensitivity for trace elements. This is achieved by a greatly improved peak-to-background ratio and a higher excitation efficiency for heavy elements. The described source is already in use for several applications in the field of X-ray fluorescence, for example micro-XRF in a SEM (iMOXS® SEM), in-line process control applications, coating characterisation etc.

XRD and micro XRD

Examinations of non-homogeneous samples, with regard to chemical composition as well as to phase structure and distribution, are of increasing interest. Micro-XRD requires an X-ray source with high brightness. In that case, the radiation should be monochromatic to suppress the β-lines for a better sensitivity for diffraction lines. This can be accomplished by a combination of the source with X-ray optics, realising either beam shaping only (for example capillary optics where β-lines can be suppressed with filters) or beam shaping and monochromatising (for example by a muliple layer BRAGG-mirror or HOPG optics). By means of beam shaping, a convergent beam can be generated with spot sizes in the range of a few tens of µm with high brightness, or a parallel beam with low divergence. The compact unit combining the X-ray source and optics offers simple ways to set up a wide variety of diffraction experiments such as single crystal or powder diffraction, stress analysis, texture analysis etc. In this case, typically X-ray tubes with Cr, Cu or Mo anodes are used. XRD can be used in different configurations. A general arrangement is shown in the top right schematic. The relatively small and light weighted source also allows the construction of portable instruments.

Technical Data	
Tube	Anode Material: Cr, Cu, Co, Mo, Rh, Ag, W, and more Max. tube parameters: 50 kV, 30 W Typical spot size: 50 x 50 µm² Exit window: 100 µm Be Filter wheel with up to 3 primary filters on request
X-ray optics (optional)	Combination with different X-ray optics (capillary optics, multi-layer optics, HOPG) possible
Control and Supply Unit	50 kV, 800 µA, stability: approximately 0.02 %
Radiation safety	Shutter and safety lock circuitry implemented in compliance with the German X-ray ordinance
Dimensions (H x W x D), weight	Source: 279 x 57 x 115 mm, approx. 3.5 kg Control unit: 150 x 360 x 300 mm, approx. 8 kg

IFG Institute for Scientific Instruments
Member of the Helmut Fischer Group
Rudower Chaussee 29/31 · 12489 Berlin
Germany
Phone: +49-30-6392-6500
Fax: +49-30-6392-6501
info@ifg-adlershof.de · www.ifg-adlershof.de

B Code for the GUI's

B.1 main.m

```
function varargout = main(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;

gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @main_OpeningFcn, ...
                   'gui_OutputFcn',  @main_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before main is made visible.
function main_OpeningFcn(hObject, eventdata, handles, varargin)

% Set name of the figure
set(handles.main, 'Name', 'Main Window');

% Set buttons inactive
set(handles.background,'Enable','off')
set(handles.auto_background,'Enable','off')
set(handles.fit_lines,'Enable','off')
set(handles.axis_calibration,'Enable','off')
set(handles.auto_calibration,'Enable','off')

% Load TU-Wien logo
axes(handles.atominstitut);
imshow('TU_Logo.gif');

% Load ATI logo
axes(handles.tuwien)
imshow('Atominstitut.png');

% Set default font size for axes
set(0,'DefaultAxesFontSize',12)

% Initialize
handles.energy=0;
handles.data=0;
handles.count=0;
```

```
52  handles.header='';
53  handles.path='';
54  handles.filename='';
55  handles.channelroi=0;
56  handles.dataroi=0;
57  handles.backgroundroi=0;
58  handles.zero=0;
59  handles.gain=0;
60
61  % Create axes for preview
62  handles.axes3=axes('Position',[0.57,0.15,0.38,0.45],'visible','off');
63
64  % Choose default command line output for main
65  handles.output = hObject;
66
67  % Update handles structure
68  guidata(hObject, handles);
69
70  % --- Outputs from this function are returned to the command line.
71  function varargout = main_OutputFcn(hObject, eventdata, handles)
72
73  % Get default command line output from handles structure
74  varargout{1} = hObject;
75
76  % --- Executes on button press in select_spectrum.
77  function select_spectrum_Callback(hObject, eventdata, handles)
78
79  % Set buttons inactive
80  set(handles.background,'Enable','off')
81  set(handles.auto_background,'Enable','off')
82  set(handles.fit_lines,'Enable','off')
83  set(handles.axis_calibration,'Enable','off')
84  set(handles.auto_calibration,'Enable','off')
85  set(handles.txt_path,'string','')
86
87  try
88      % Reset energy and data arrays
89      handles.energy=[];
90      handles.data=[];
91
92      % Select path for ASR file
93      [file_ASR, path_ASR] = uigetfile( ...
94          { '*.SPE','SPE-files (*.SPE)'; ...
95          '*.*',  'All Files (*.*)'}, ...
96          'Please select spectrum:',handles.path);
97
98      % Save path in handle structure
99      handles.path=path_ASR;
100     handles.filename=file_ASR;
101
102     % Shows the file name and path in the edit text field
103     set(handles.txt_path,'string',fullfile(path_ASR,file_ASR))
104
105     % Read spe-file
106     [handles.channel,handles.data,handles.header]...
```

```
107             =read_spefile(file_ASR,path_ASR);
108
109         % Clear axes before new preview
110         cla(handles.axes3)
111
112         % Set axis limits
113         xmin=0;
114         xmax=max(handles.channel);
115
116         % Plot preview
117         axes(handles.axes3);
118         plot(handles.channel,handles.data);
119         xlim(handles.axes3,[xmin xmax]);
120         xlabel('Channel');
121         ylabel('Intensity [counts]');
122
123         % Activate buttons for background reduction
124         set(handles.background,'Enable','on')
125         set(handles.auto_background,'Enable','on')
126         set(handles.axes3,'visible','on')
127     catch
128         errordlg('Error during reading file!');
129         set(handles.txt_path,'string','')
130         cla(handles.axes3)
131         set(handles.axes3,'visible','off')
132     end
133
134     % Update handles structure
135     guidata(hObject, handles);
136
137     % --- Executes on button press in background.
138     function background_Callback(hObject, eventdata, handles)
139
140     try
141         % Reset data arrays
142         handles.channelroi=[];
143         handles.dataroi=[];
144         handles.backgroundroi=[];
145
146         % Start background_fit
147         [handles.channelroi,handles.dataroi,handles.backgroundroi]...
148             =background_fit(handles.channel,handles.data);
149
150         % Set buttons active/inactive
151         set(handles.axis_calibration,'Enable','on');
152         set(handles.auto_calibration,'Enable','on');
153         set(handles.background,'Enable','off')
154         set(handles.auto_background,'Enable','off')
155     catch
156         errordlg('Problem occured during background fit');
157     end
158
159     % Update handles structure
160     guidata(hObject, handles);
161
```

```
162
163 % --- Executes on button press in auto_background.
164 function auto_background_Callback(hObject, eventdata, handles)
165
166 try
167     % Select path for backgroundfile
168     [file_back,path_back]= uigetfile('*.txt','Select background file',...
169         handles.path);
170
171     % Reset data arrays
172     handles.channelroi=[];
173     handles.dataroi=[];
174     handles.backgroundroi=[];
175
176     [handles.dataroi,handles.channelroi,handles.backgroundroi]...
177         = auto_background(handles.data,path_back,file_back);
178
179     % Set buttons active/inactive
180     set(handles.axis_calibration,'Enable','on')
181     set(handles.auto_calibration,'Enable','on')
182     set(handles.background,'Enable','off')
183     set(handles.auto_background,'Enable','off')
184 catch
185     errordlg('Problem occured during automatic background fit');
186 end
187
188 % Update handles structure
189 guidata(hObject, handles);
190
191 % --- Executes on button press in axis_calibration.
192 function axis_calibration_Callback(hObject, eventdata, handles)
193
194 %try
195     % Start axis calibration
196     [output]=fit_calibration(handles.channelroi,handles.dataroi);
197
198     handles.energy = output.energy;
199     handles.zero = output.zero;
200     handles.gain = output.gain;
201     % Set buttons active/inactive
202     set(handles.fit_lines,'Enable','on');
203     set(handles.axis_calibration,'Enable','off')
204     set(handles.auto_calibration,'Enable','off')
205 %catch
206 %    errordlg('Problem occured during axis calibration');
207 %end
208
209 % Update handles structure
210 guidata(hObject, handles);
211
212
213 % --- Executes on button press in auto_calibration.
214 function auto_calibration_Callback(hObject, eventdata, handles)
215
216 try
```

```
217      [file_cali,path_cali]=uigetfile('*.txt',...
218          'Select file for axis calibration',handles.path);
219
220      %Axis calibration
221      [handles.energy,handles.zero,handles.gain]...
222          = read_axiscalibration(handles.channelroi,file_cali,path_cali);
223
224      % Set buttons active/inactive
225      set(handles.fit_lines,'Enable','on')
226      set(handles.axis_calibration,'Enable','off')
227      set(handles.auto_calibration,'Enable','off')
228  catch
229      errordlg('Problem occured during automatic axis calibration');
230  end
231
232  % Update handles structure
233  guidata(hObject, handles);
234
235
236  % --- Executes on button press in fit_lines.
237  function fit_lines_Callback(hObject, eventdata, handles)
238
239
240  fit_input.energy = handles.energy;
241  fit_input.dataroi = handles.dataroi;
242  fit_input.header = handles.header;
243  fit_input.backgroundroi = handles.backgroundroi;
244  fit_input.zero = handles.zero;
245  fit_input.gain = handles.gain;
246  fit_input.channelroi = handles.channelroi;
247  fit_input.filename = handles.filename;
248  fit_input.path = handles.path;
249
250  % Start fit_program GUI
251  fit_program_main(fit_input);
252
253
254  % --- Executes on button press in multi_fit.
255  function multi_fit_Callback(hObject, eventdata, handles)
256
257  %try
258      % Starts multi fit GUI
259      multi_fit_main();
260  %catch
261  %    errordlg('Problem occured during multi fit');
262  %end
263
264
265  % --- Executes on button press in qtxrf.
266  function qtxrf_Callback(hObject, eventdata, handles)
267
268  try
269      % Starts QTXRF
270      qtxrf();
271  catch
```

```
272     errordlg('Problem occured during QTXRF');
273 end
274
275 % --- Executes during object creation, after setting all properties.
276 function txt_path_CreateFcn(hObject, eventdata, handles)
277
278 % Hint: edit controls usually have a white background on Windows.
279 %       See ISPC and COMPUTER.
280 if ispc && isequal(get(hObject,'BackgroundColor'),...
281         get(0,'defaultUicontrolBackgroundColor'))
282     set(hObject,'BackgroundColor','white');
283 end
284
285
286 % --- Executes when user attempts to close main.
287 function main_CloseRequestFcn(hObject, eventdata, handles)
288
289 % Hint: delete(hObject) closes the figure
290 delete(hObject);
```

B.2 background_fit.m

```
1  function varargout = background_fit(varargin)
2
3  % Begin initialization code - DO NOT EDIT
4  gui_Singleton = 1;
5  gui_State = struct('gui_Name',       mfilename, ...
6                     'gui_Singleton',  gui_Singleton, ...
7                     'gui_OpeningFcn', @background_fit_OpeningFcn, ...
8                     'gui_OutputFcn',  @background_fit_OutputFcn, ...
9                     'gui_LayoutFcn',  [] , ...
10                    'gui_Callback',   []);
11 if nargin && ischar(varargin{1})
12     gui_State.gui_Callback = str2func(varargin{1});
13 end
14
15 if nargout
16     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
17 else
18     gui_mainfcn(gui_State, varargin{:});
19 end
20 % End initialization code - DO NOT EDIT
21
22 function [] = draw_line_buttondown(varargin)
23
24 % WindowButtonDownFunction
25
26 % Get structure
27 S = varargin{3};
28
29 % Get limits
30 x_lim = get(S.ax,'xlim');
31 y_lim = get(S.ax,'ylim');
32
33 % Current point
34 current_p = get(S.ax,'currentpoint');
```

```
35 current_x = current_p(1,1);
36 current_y = current_p(1,2);
37
38 % Check if current point in Axes
39 in_axis = current_x >= 0 && current_y >= 0 &&...
40     current_x <= x_lim(2) && current_y <= y_lim(2);
41
42 if in_axis
43
44     % Set point to textbox
45     set(S.tx(1),'String',num2str(current_x));
46     % Plot data
47     plot(S.channel,S.data,'k.');
48     xmin=0;
49     xmax=max(S.channel);
50
51     % Plot area
52     area(S.channel,S.data,'FaceAlpha',0.3,'FaceColor','r');
53     xlim(S.ax,[xmin xmax]);
54     ylim(S.ax,y_lim);
55     xlabel('Channel');
56     ylabel('Intensity [counts]');
57
58     % Check lin or log
59     if get(S.log,'Value')
60         set(S.ax,'YScale','log');
61     else
62         set(S.ax,'YScale','lin');
63     end
64
65     % Plot line as marker
66     line([current_x current_x],y_lim,'Color','k');
67
68 end
69
70 % --- Executes just before background_fit is made visible.
71 function background_fit_OpeningFcn(hObject, eventdata, handles, varargin)
72
73 % Name of the figure
74 set(handles.background_fit, 'Name', 'Background Window');
75
76 % Set default font size for axes
77 set(0,'DefaultAxesFontSize',12)
78
79 % Set button subtract and submit inactive
80 set(handles.subtract,'Enable','off')
81 set(handles.submit,'Enable','off')
82
83 % Set the uiradio buttons to group
84 set(handles.radio_runningball,'Parent',handles.buttongroup);
85 set(handles.radio_peakstripping,'Parent',handles.buttongroup);
86 set(handles.radio_msbackadj,'Parent',handles.buttongroup);
87 set(handles.radio_noback,'Parent',handles.buttongroup);
88 set(handles.radio_orthogonal,'Parent',handles.buttongroup);
89
```

```
90 % Default value
91 set(handles.radio_peakstripping,'Value',1);
92
93 % Get input
94 handles.channel=varargin{1};
95 handles.data=varargin{2};
96
97 % Set range txt fields to min an max from channel
98 set(handles.txt_from,'String',min(handles.channel));
99 set(handles.txt_to,'String',max(handles.channel));
100
101
102 % Set axis limits
103 xmin=0;
104 xmax=max(handles.channel);
105 ymin=0;
106 ymax=max(handles.data)*1.1;
107
108 % Plot spectrum
109 axes(handles.axes1);
110 plot(handles.channel,handles.data,'k.');
111 area(handles.channel,handles.data,'FaceAlpha',0.3,'FaceColor','r');
112 xlim(handles.axes1,[xmin xmax]);
113 ylim(handles.axes1,[ymin ymax]);
114 xlabel('Channel');
115 ylabel('Intensity [counts]');
116
117
118 % Initialize output
119 handles.datacorr=[];
120 handles.ymaxbackplot=max(handles.data)*1.1;
121
122 % Choose default command line output for background_fit
123 handles.output = hObject;
124
125
126 % Create S for function input
127 input.channel=handles.channel;
128 input.data=handles.data;
129 input.ax = gca;
130 input.tx(1) = handles.marker_pos;
131 input.log = handles.radio_log;
132
133 % Set the 'windowbuttondownfcn'
134 set(gcf,'windowbuttondownfcn',{@draw_line_buttondown,input})
135
136 handles.input_bdf=input;
137
138 % Update handles structure
139 guidata(hObject, handles);
140
141
142 % --- Outputs from this function are returned to the command line.
143 function varargout = background_fit_OutputFcn(hObject, eventdata, handles)
144
```

```
145 % UIWAIT makes background_fit wait for user response (see UIRESUME)
146 uiwait(handles.background_fit);
147
148 % Retrieve handles
149 handles = guidata(hObject);
150
151 % Get default command line output from handles structure
152 varargout{1} = handles.newchannel;
153 varargout{2} = handles.datacorr;
154 varargout{3} = handles.background;
155
156 % Delete figure after output
157 delete(handles.background_fit);
158
159
160 % --- Executes on button press in plot.
161 function plot_Callback(hObject, eventdata, handles)
162
163 set(gcf,'windowbuttondownfcn',{@draw_line_buttondown,handles.input_bdf})
164
165 % Read range for estimation of background
166 fromchannel=str2double(get(handles.txt_from,'String'));
167 tochannel=str2double(get(handles.txt_to,'String'));
168
169 % Can't be smaller than 0 and higher than max
170 if fromchannel<1 || fromchannel>max(handles.channel)-10
171     fromchannel=1;
172 end
173
174 % Can't be higher than mac(channel)
175 if tochannel<10 || tochannel>max(handles.channel)
176     tochannel=max(handles.channel);
177 end
178
179 % Can't be higher than tochannel adn difference must be higher than 15
180 if fromchannel>=tochannel || tochannel-fromchannel<15
181         fromchannel=1;
182         tochannel=max(handles.channel);
183 end
184
185
186 % Overwrite range
187 set(handles.txt_from,'String',num2str(fromchannel));
188 set(handles.txt_to,'String',num2str(tochannel));
189
190 % Generate new channel vector
191 handles.newchannel=(fromchannel:tochannel);
192
193 % Old size
194 sizeold=size(handles.channel,2);
195
196 % Data for background estimation
197 datacal=handles.data(fromchannel:tochannel);
198
199 % New channel variable
```

```
200 sizenew=size(datacal,2);
201 channelcal=(1:sizenew);
202
203 % Leading and trailing zeros
204 zerofrom(1:fromchannel-1)=0;
205 zeroto(1:sizeold-sizenew-fromchannel+1)=0;
206
207
208 % Get data from the different background methods
209 window=str2double(get(handles.text_window,'String'));
210 step=str2double(get(handles.text_step,'String'));
211 radius=str2double(get(handles.text_runningball,'String'));
212 iterations=str2double(get(handles.text_peakstripping,'String'));
213 distance=str2double(get(handles.distance,'String'));
214 degree=str2double(get(handles.text_degree,'String'));
215 rfactor=str2double(get(handles.rfactor,'String'));
216
217
218 % Golay filter for background method 1-3
219 databcal = sgolayfilt(datacal,2,11);
220 databcal(databcal<0)=0;
221
222 % Do peakstripping
223 backpeak = background_peak_stripping(databcal,iterations,distance);
224
225 % Do running ball
226 backrun = background_running_ball(channelcal,databcal,radius);
227
228 % Do msbackadj
229 YB = msbackadj(channelcal(:),databcal(:),'WindowSize',window,...
230     'StepSize',step);
231 backms=databcal-YB';
232 backms(backms<0) = 0;
233
234 % Do peakstripping
235 backortho = background_ortho(channelcal,datacal,degree,rfactor);
236
237 % Datacal with leading and trailing zeros for area color
238 dataarea=horzcat(zerofrom,datacal,zeroto);
239
240 % Plot spectrum
241 axes(handles.axes1);
242 plot(handles.channel,handles.data,'k');
243 hold on
244
245 % Color the range of background estimation
246 areahandle=area(handles.channel,dataarea,'FaceAlpha',0.3,'FaceColor','r');
247
248 % Plot spectrum and different backgrounds
249 plothandle=plot(handles.newchannel,datacal,'k',handles.newchannel,...
250     backpeak,'r',handles.newchannel,backrun,'g',handles.newchannel,...
251     backms,'b',handles.newchannel,backortho,'c','LineWidth',1);
252
253 % Plot lines for ROI border
254 line([fromchannel fromchannel],[1 max(handles.data)*2],'Color','r',...
```

```
255         'Linewidth',1.2);
256     line([tochannel tochannel],[1 max(handles.data)*2],'Color','r',...
257         'Linewidth',1.2);
258
259     hold off
260
261     % Set axis limits (ymax limit has been set to the double value of the
262     %highest background to compare different background methods
263     xmin=0;
264     xmax=max(handles.channel);
265     ymin=0;
266     maxp=max(backpeak);
267     maxr=max(backrun);
268     maxm=max(backms);
269     maxo=max(backortho);
270     ymax=max([maxp maxr maxm maxo])*2;
271
272     % Safe maximum for log-lin change
273     handles.ymaxbackplot=ymax;
274
275
276     xlim(handles.axes1,[xmin xmax]);
277     ylim(handles.axes1,[ymin ymax]);
278     xlabel('Channel');
279     ylabel('Intensity [counts]');
280     plothandle(end+1)=areahandle;
281     legend(plothandle,{'Data' 'Peak Stripping' 'Running Ball' 'MSBACKADJ'...
282         'Orthogonal' 'Region of Interest' 'Data'},'Location','best');
283
284     % Check the scale
285     if get(handles.radio_log,'Value')
286         set(handles.axes1,'YScale','log');
287         xmin=0;
288         xmax=max(handles.channel);
289         ymin=1;
290         ymax=max(handles.data)*1.1;
291         xlim(handles.axes1,[xmin xmax]);
292         ylim(handles.axes1,[ymin ymax]);
293     end
294
295     % Add to handles structure
296     handles.backrun=backrun;
297     handles.backpeak=backpeak;
298     handles.backms=backms;
299     handles.backortho=backortho;
300     handles.datacal=datacal;
301     handles.from=fromchannel;
302     handles.to=tochannel;
303
304     % Set subtract button active
305     set(handles.subtract,'Enable','on')
306
307     % Update handles structure
308     guidata(hObject, handles);
309
```

```
310
311 % --- Executes on button press in subtract.
312 function subtract_Callback(hObject, eventdata, handles)
313
314 set(gcf,'windowbuttondownfcn',{})
315
316 % Get values of the radio buttons
317 x1=get(handles.radio_runningball,'Value');
318 x2=get(handles.radio_peakstripping,'Value');
319 x3=get(handles.radio_msbackadj,'Value');
320 x4=get(handles.radio_noback,'Value');
321 x5=get(handles.radio_orthogonal,'Value');
322
323 back=[];
324
325 % Get the background method that is selected
326 if x1==1
327     back=handles.backrun;
328 elseif x2==1
329     back=handles.backpeak;
330 elseif x3==1
331     back=handles.backms;
332 elseif x4==1
333     back=0;
334 elseif x5==1
335     back=handles.backortho;
336 end
337
338
339 handles.background=back;
340
341 % Subtract the choosen background from the spectrum
342 handles.datacorr=handles.datacal-back;
343 handles.datacorr(handles.datacorr<0)=0;
344
345 % Plot spectrum with and without background
346 axes(handles.axes1);
347 plot(handles.newchannel,handles.datacal,'k.',handles.newchannel,...
348     handles.datacorr,'b.');
349
350 % Axis limits for this button
351 xmin=0;
352 xmax=max(handles.channel);
353 ymin=0;
354 ymax=max(handles.data)*1.1;
355 xlim(handles.axes1,[xmin xmax]);
356 ylim(handles.axes1,[ymin ymax]);
357 xlabel('Channel');
358 ylabel('Intensity [counts]');
359 legend('Data','Data without Background')
360
361 if get(handles.radio_log,'Value')
362     set(handles.axes1,'YScale','log');
363     xmin=0;
364     xmax=max(handles.channel);
```

```
365         ymin=0.5;
366         ymax=max(handles.data)*1.1;
367         xlim(handles.axes1,[xmin xmax]);
368         ylim(handles.axes1,[ymin ymax]);
369     end
370
371     legend('Location','best');
372
373     set(handles.submit,'Enable','on')
374
375     % Set subtract button active
376     set(handles.subtract,'Enable','off')
377
378     % Update handles structure
379     guidata(hObject, handles);
380
381     % --- Executes on button press in submit.
382     function submit_Callback(hObject, eventdata, handles)
383
384     %uiresume
385     uiresume(handles.background_fit);
386
387
388     % --- Executes when user attempts to close background_fit.
389     function background_fit_CloseRequestFcn(hObject, eventdata, handles)
390
391     uiresume(handles.background_fit);
392
393     % Hint: delete(hObject) closes the figure
394     delete(hObject);
395
396
397     % --- Executes when selected object is changed in radiogroup.
398     function radiogroup_SelectionChangedFcn(hObject, eventdata, handles)
399
400     % Log/lin scale
401     if get(handles.radio_log,'Value')
402         set(handles.axes1,'YScale','log');
403         xmin=0;
404         xmax=max(handles.channel);
405         ymin=1;
406         if length(get(gca,'children'))==2
407             ymax=max(handles.data)*1.1;
408         else
409             ymax=handles.ymaxbackplot;
410         end
411         ymax=max(handles.data)*1.1;
412         xlim(handles.axes1,[xmin xmax]);
413         ylim(handles.axes1,[ymin ymax]);
414     end
415
416     if get(handles.radio_lin,'Value')
417         set(handles.axes1,'YScale','lin');
418         xmin=0;
419         xmax=max(handles.channel);
```

```
420         ymin=0;
421         if length(get(gca,'children'))==2
422             ymax=max(handles.data)*1.1;
423         else
424             ymax=handles.ymaxbackplot;
425         end
426         xlim(handles.axes1,[xmin xmax]);
427         ylim(handles.axes1,[ymin ymax]);
428     end
429
430
431     % Update handles structure
432     guidata(hObject, handles);
433
434
435     % --- Executes on button press in save.
436     function save_Callback(hObject, eventdata, handles)
437
438     % Get values
439     x1=get(handles.radio_runningball,'Value');
440     x2=get(handles.radio_peakstripping,'Value');
441     x3=get(handles.radio_msbackadj,'Value');
442     x4=get(handles.radio_noback,'Value');
443     x5=get(handles.radio_orthogonal,'Value');
444
445     str01=get(handles.txt_from,'String');
446     str02=get(handles.txt_to,'String');
447     str1=get(handles.text_runningball,'String');
448     str21=get(handles.text_peakstripping,'String');
449     str22=get(handles.distance,'String');
450     str31=get(handles.text_window,'String');
451     str32=get(handles.text_step,'String');
452     str41=get(handles.text_degree,'String');
453     str42=get(handles.rfactor,'String');
454
455     % Ask for file path and name input for result file
456     [filename_back,path_back]=uiputfile( ...
457     {  '*.txt','txt-files (*.txt)'; ...
458        '*.*',  'All Files (*.*)'}, ...
459        'Please choose filename for background file:');
460
461     % Open result file
462     fileID = fopen(fullfile(path_back,filename_back),'w');
463
464
465     % Get the background method that is selected
466     if x1==1
467         fprintf(fileID,'1 %s %s %s\n',str1,str01,str02);
468     elseif x2==1
469         fprintf(fileID,'2 %s %s %s %s\n',str21,str22,str01,str02);
470     elseif x3==1
471         fprintf(fileID,'3 %s %s %s %s\n',str31,str32,str01,str02);
472     elseif x4==1
473         fprintf(fileID,'4 %s %s \n',str01,str02);
474     elseif x5==1
```

```
475         fprintf(fileID,'5 %s %s %s %s\n',str41,str42,str01,str02);
476     end
477
478     fclose(fileID);
479
480     % --- Executes on button press in button_begin.
481     function button_begin_Callback(hObject, eventdata, handles)
482
483     % Copy the current position to begin
484     str_begin = get(handles.marker_pos,'String');
485     begin_ch = ceil(str2double(str_begin));
486     set(handles.txt_from,'String',num2str(begin_ch));
487
488     % --- Executes on button press in button_end.
489     function button_end_Callback(hObject, eventdata, handles)
490
491     % Copy the current position to end
492     str_end = get(handles.marker_pos,'String');
493     end_ch = floor(str2double(str_end));
494     set(handles.txt_to,'String',num2str(end_ch));
```

B.3 fit_calibration.m

```
1   function varargout = fit_calibration(varargin)
2
3   % Begin initialization code - DO NOT EDIT
4   gui_Singleton = 1;
5   gui_State = struct('gui_Name',       mfilename, ...
6                      'gui_Singleton',  gui_Singleton, ...
7                      'gui_OpeningFcn', @fit_calibration_OpeningFcn, ...
8                      'gui_OutputFcn',  @fit_calibration_OutputFcn, ...
9                      'gui_LayoutFcn',  [] , ...
10                     'gui_Callback',   []);
11  if nargin && ischar(varargin{1})
12      gui_State.gui_Callback = str2func(varargin{1});
13  end
14
15  if nargout
16      [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
17  else
18      gui_mainfcn(gui_State, varargin{:});
19  end
20  % End initialization code - DO NOT EDIT
21
22
23  % --- Executes just before fit_calibration is made visible.
24  function fit_calibration_OpeningFcn(hObject, eventdata, handles, varargin)
25
26  set(handles.fit_calibration, 'Name', 'Energy Calibration Window');
27
28  % Get input
29  handles.channel=varargin{1};
30  handles.data=varargin{2};
31
32  % Set button calibrate and submit inactive
33  set(handles.calibrate,'Enable','off')
```

```
34 set(handles.submit,'Enable','off')
35 set(handles.save,'Enable','off')
36 set(handles.newcali_button,'Enable','off')
37
38 % Set minimum for plot
39 handles.xmin=0;
40 handles.ymin=1;
41
42 % Set maximum for plot
43 handles.xmax=max(handles.channel);
44 handles.ymax=max(handles.data);
45
46
47 % Set vector with weighted average of the Kalpha lines
48 handles.weighted_averagekalpha=[0.052;0.110;0.185;0.282;0.392;0.523;...
49     0.677;0.851;1.041;1.254;1.487;1.740;2.015;2.307;2.622;2.957;3.312;...
50     3.690;4.088;4.508;4.949;5.411;5.895;6.400;6.925;7.472;8.041;8.631;...
51     9.243;9.876;10.532;11.210;11.907;12.630;13.375;14.142;14.933;...
52     15.746;16.584;17.443;18.327;19.235;20.167;21.123;22.104;23.109;...
53     24.139;25.193;26.274;27.380;28.512;29.669;30.854;32.065;33.302;...
54     34.569;35.864;37.185;38.535;39.914;41.323;42.761;44.229;45.728;...
55     47.257;48.818;50.410;52.035;53.693;55.382;57.106;58.864;60.655;...
56     62.482;64.346;66.246;68.185;70.160;72.176;74.228;76.321;78.460;...
57     80.636;82.855;85.124;87.436;89.790;92.190;94.643;97.143];
58
59 % Set elements
60 handles.weighted_averagekalphastr={'Li';'Be';'B';'C';'N';'O';'F';'Ne';...
61     'Na';'Mg';'Al';'Si';'P';'S';'Cl';'Ar';'K';'Ca';'Sc';'Ti';'V';'Cr';...
62     'Mn';'Fe';'Co';'Ni';'Cu';'Zn';'Ga';'Ge';'As';'Se';'Br';'Kr';'Rb';...
63     'Sr';'Y';'Zr';'Nb';'Mo';'Tc';'Ru';'Rh';'Pd';'Ag';'Cd';'In';'Sn';...
64     'Sb';'Te';'I';'Xe';'Cs';'Ba';'La';'Ce';'Pr';'Nd';'Pm';'Sm';'Eu';...
65     'Gd';'Tb';'Dy';'Ho';'Er';'Tm';'Yb';'Lu';'Hf';'Ta';'W';'Re';'OS';...
66     'Ir';'Pt';'Au';'Hg';'Tl';'Pb';'Bi';'Po';'At';'Rn';'Fr';'Ra';'Ac';...
67     'Th';'Pa';'U'};
68
69 % Fill popup with elements
70 set(handles.kalpha_lines,'String',handles.weighted_averagekalphastr);
71
72 % Plot calibrated spectrum
73 axes(handles.axes1);
74 plot(handles.channel,handles.data);
75 xlim(handles.axes1,[handles.xmin handles.xmax]);
76 ylim(handles.axes1,[handles.ymin handles.ymax]);
77 xlabel('Channel');
78 ylabel('Intensity [counts]');
79
80 handles.q=1;
81
82 % Create input for button down function
83 input.channel=handles.channel;
84 input.data=handles.data;
85 input.ax = gca;
86 input.tx(1) = handles.position;
87 input.lin = handles.button_lin;
88 input.log = handles.button_log;
```

```
89
90 %Set the button detector.
91 set(gcf,'windowbuttondownfcn',{@draw_line_buttondown,input})
92
93 % Update handles structure
94 guidata(hObject, handles);
95
96 % UIWAIT makes fit_calibration wait for user response (see UIRESUME)
97 uiwait(handles.fit_calibration);
98
99 function [] = draw_line_buttondown(varargin)
100
101 % WindowButtonDownFunction
102
103 % Get structure
104 S = varargin{3};
105
106 % Get limits
107 x_lim = get(S.ax,'xlim');
108 y_lim = get(S.ax,'ylim');
109
110 % Current point
111 current_p = get(S.ax,'currentpoint');
112 current_x = current_p(1,1);
113 current_y = current_p(1,2);
114
115 % Check if current point in Axes
116 in_axis = current_x >= 0 && current_y >= 0 &&...
117     current_x <= x_lim(2) && current_y <= y_lim(2);
118
119 if in_axis
120
121     % Set point to textbox
122     set(S.tx(1),'String',num2str(current_x));
123     % Plot data
124     plot(S.channel,S.data);
125     xlim(S.ax,x_lim);
126     ylim(S.ax,y_lim);
127     xlabel('Channel');
128     ylabel('Intensity [a.u.]');
129
130     if get(S.log,'Value')
131         set(S.ax,'YScale','log');
132     else
133         set(S.ax,'YScale','lin');
134     end
135
136     %Plot line as marker
137     line([current_x current_x],y_lim,'Color','r');
138
139 end
140
141
142 % --- Outputs from this function are returned to the command line.
143 function varargout = fit_calibration_OutputFcn(hObject, eventdata, handles)
```

```
144
145 % Generate output structure
146 output.energy = handles.energy;
147 output.zero = handles.zero;
148 output.gain = handles.gain;
149
150 % Get default command line output from handles structure
151 varargout{1} = output;
152
153 %delete figure after output
154 delete(handles.fit_calibration);
155
156
157 % --- Executes on button press in apply.
158 function apply_Callback(hObject, eventdata, handles)
159
160 % Save energy and channel in handles.cali
161 handles.cali(handles.q,1)=str2double(get(handles.position,'String'));
162 handles.cali(handles.q,2)=str2double(get(handles.cali_energy,'String'));
163
164 % Show saved points in uitable
165 set(handles.cali_points,'Data',handles.cali);
166
167 % With more than one calibration point set button calibrate active
168 if handles.q>=2
169     set(handles.calibrate,'Enable','on')
170 end
171
172 % Global counting variable
173 handles.q=handles.q+1;
174
175 % Update handles structure
176 guidata(hObject, handles);
177
178
179 % --- Executes on button press in reset.
180 function reset_Callback(hObject, eventdata, handles)
181
182 % Reset handles.cali
183 handles.cali(:,:)='';
184
185 % Reset global counting variable
186 handles.q=1;
187
188 % Show reseted handles.cali in uitable
189 set(handles.cali_points,'Data',handles.cali);
190
191 % Update handles structure
192 guidata(hObject, handles);
193
194
195
196 % --- Executes on button press in calibrate.
197 function calibrate_Callback(hObject, eventdata, handles)
198
```

```
199 x=handles.cali(:,1);
200 y=handles.cali(:,2);
201
202 % Fit datapoints
203 yfit=fit(x,y,'poly1');
204 gain = yfit.p1;
205 zero = yfit.p2;
206
207 % Create handles vector to plot fit with fitfunction
208 x2=max(handles.channel);
209 x1=(0:0.1:x2);
210
211 % Plot fitfunction and data points
212 plot(x1,x1*gain+zero,'b-',x,y,'r*');
213
214 title('Energy axes calibration');
215
216 % Set axis limits
217 xlim(handles.axes1,[0 max(handles.channel)]);
218 ylim(handles.axes1,[0 max(handles.channel)*gain+zero]);
219 xlabel('Channel');
220 ylabel('Energy [keV]');
221
222 % Set result global
223 handles.gain=gain;
224 handles.zero=zero;
225 handles.energy=handles.channel*gain+zero;
226
227 % Set button submit active
228 set(handles.submit,'Enable','on')
229 set(handles.save,'Enable','on')
230 set(handles.newcali_button,'Enable','on')
231
232 % Set button submit active
233 set(handles.left_button,'Enable','off')
234 set(handles.right_button,'Enable','off')
235 set(handles.resetzoom,'Enable','off')
236 set(handles.apply,'Enable','off')
237 set(handles.reset,'Enable','off')
238
239 set(gcf,'windowbuttondownfcn',[])
240
241 guidata(hObject, handles);
242
243 % --- Executes on button press in submit.
244 function submit_Callback(hObject, eventdata, handles)
245
246 % Submit
247 uiresume(handles.fit_calibration);
248
249
250 % --- Executes on button press in save.
251 function save_Callback(hObject, eventdata, handles)
252
253 % Ask for file path and name input for relativities file
```

```
254 [filename,path]=uiputfile( ...
255 {  '*.txt','txt-files (*.txt)'; ...
256    '*.*',  'All Files (*.*)'}, ...
257    'Please choose filename for axis calibration:');
258
259 % Open relativities textfile
260 fileID = fopen(fullfile(path,filename),'w');
261
262 fprintf(fileID,'-----------------------------------\n');
263 fprintf(fileID,'Axis calibration coefficients:\n');
264 fprintf(fileID,'-----------------------------------\n');
265 fprintf(fileID,'%f\n',handles.gain);
266 fprintf(fileID,'%f\n',handles.zero);
267
268 fclose(fileID);
269
270
271 % --- Executes on selection change in kalpha_lines.
272 function kalpha_lines_Callback(hObject, eventdata, handles)
273
274 % Get selected line
275 selectedline=get(handles.kalpha_lines,'Value');
276
277 % Set energy value
278 set(handles.cali_energy,'String',...
279     num2str(handles.weighted_averagekalpha(selectedline)));
280
281
282 % --- Executes when selected object is changed in panel_linlog.
283 function panel_linlog_SelectionChangedFcn(hObject, eventdata, handles)
284
285 if get(handles.button_log,'Value')
286     set(gca,'YScale','log');
287 else
288     set(gca,'YScale','lin');
289 end
290
291
292 % --- Executes on button press in left_button.
293 function left_button_Callback(hObject, eventdata, handles)
294
295 % Set new axis limit for zoom function
296 handles.xmin = str2double(get(handles.position,'String'));
297 xlim(handles.axes1,[handles.xmin handles.xmax]);
298
299 guidata(hObject, handles);
300
301 % --- Executes on button press in right_button.
302 function right_button_Callback(hObject, eventdata, handles)
303
304 % Set new axis limit for zoom function
305 handles.xmax = str2double(get(handles.position,'String'));
306 xlim(handles.axes1,[handles.xmin handles.xmax]);
307
308 % Update handles structure
```

```
309 guidata(hObject, handles);
310
311 % --- Executes on button press in resetzoom.
312 function resetzoom_Callback(hObject, eventdata, handles)
313
314 %set minimum for plot
315 handles.xmin=0;
316 handles.ymin=1;
317 %set maximum for plot
318 handles.xmax=max(handles.channel);
319 handles.ymax=max(handles.data);
320
321 xlim(handles.axes1,[handles.xmin handles.xmax]);
322 ylim(handles.axes1,[handles.ymin handles.ymax]);
323
324 % Update handles structure
325 guidata(hObject, handles);
326
327
328 % --- Executes on button press in newcali_button.
329 function newcali_button_Callback(hObject, eventdata, handles)
330
331 % Set minimum for plot
332 handles.xmin=0;
333 handles.ymin=1;
334
335 % Set maximum for plot
336 handles.xmax=max(handles.channel);
337 handles.ymax=max(handles.data);
338
339 % Plot calibrated spectrum
340 axes(handles.axes1);
341 plot(handles.channel,handles.data);
342 xlim(handles.axes1,[handles.xmin handles.xmax]);
343 ylim(handles.axes1,[handles.ymin handles.ymax]);
344 xlabel('Channel');
345 ylabel('Intensity [counts]');
346
347 % Reset handles.cali
348 handles.cali(:,:)='';
349
350 % Reset global counting variable
351 handles.q=1;
352
353 % Show reseted handles.cali in uitable
354 set(handles.cali_points,'Data',handles.cali);
355
356 % Set button submit active
357 set(handles.submit,'Enable','off')
358 set(handles.save,'Enable','off')
359
360 % Set button submit active
361 set(handles.left_button,'Enable','on')
362 set(handles.right_button,'Enable','on')
363 set(handles.resetzoom,'Enable','on')
```

```
364 set(handles.apply,'Enable','on')
365 set(handles.reset,'Enable','on')
366
367 % Create input for button down function
368 input.channel=handles.channel;
369 input.data=handles.data;
370 input.ax = gca;
371 input.tx(1) = handles.position;
372 input.lin = handles.button_lin;
373 input.log = handles.button_log;
374
375 % Set the button detector.
376 set(gcf,'windowbuttondownfcn',{@draw_line_buttondown,input})
377
378 % Update handles structure
379 guidata(hObject, handles);
```

B.4 fit_program_main.m

```
1  function varargout = fit_program_main(varargin)
2
3  % Begin initialization code - DO NOT EDIT
4  gui_Singleton = 1;
5  gui_State = struct('gui_Name',       mfilename, ...
6                     'gui_Singleton',  gui_Singleton, ...
7                     'gui_OpeningFcn', @fit_program_main_OpeningFcn, ...
8                     'gui_OutputFcn',  @fit_program_main_OutputFcn, ...
9                     'gui_LayoutFcn',  [] , ...
10                    'gui_Callback',   []);
11 if nargin && ischar(varargin{1})
12     gui_State.gui_Callback = str2func(varargin{1});
13 end
14
15 if nargout
16     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
17 else
18     gui_mainfcn(gui_State, varargin{:});
19 end
20 % End initialization code - DO NOT EDIT
21
22
23 % --- Executes just before fit_program_main is made visible.
24 function fit_program_main_OpeningFcn(hObject, eventdata,handles,varargin)
25
26 % Set name of the figure
27 set(handles.figure1, 'Name', 'Fit Window');
28
29 % Path for parameter file
30 path_para=fileparts(which('fit_program_main.m'));%pwd;%'./';
31 file_para='FIT_PARAMETER.txt';
32
33 % msgbox(fullfile(path_para,file_para));
34
35 % Read Parameter file
36 parameter = read_parameterfile(path_para,file_para);
37
```

```
38 % Get input
39 fit_input = varargin{1};
40
41 % Add fit_input to handles structure
42 handles.energy = fit_input.energy;
43 handles.data = fit_input.dataroi;
44 handles.header = fit_input.header;
45 handles.background = fit_input.backgroundroi;
46 handles.zero = fit_input.zero;
47 handles.gain = fit_input.gain;
48 handles.channel = fit_input.channelroi;
49 handles.filename = fit_input.filename;
50 handles.path = fit_input.path;
51
52 % Add Edges to handles structure
53 handles.edges={'K-lines','L1-lines','L2-lines','L3-lines',...
54     'Ka-lines','Kb-lines','M5-lines'};
55
56 % Add Elements to handles structure
57 handles.elements={'He';'Li';'Be';'B';'C';'N';'O';'F';'Ne';...
58     'Na';'Mg';'Al';'Si';'P';'S';'Cl';'Ar';'K';'Ca';'Sc';'Ti';...
59     'V';'Cr';'Mn';'Fe';'Co';'Ni';'Cu';'Zn';'Ga';'Ge';'As';'Se';...
60     'Br';'Kr';'Rb';'Sr';'Y';'Zr';'Nb';'Mo';'Tc';'Ru';'Rh';'Pd';...
61     'Ag';'Cd';'In';'Sn';'Sb';'Te';'I';'Xe';'Cs';'Ba';'La';'Ce';...
62     'Pr';'Nd';'Pm';'Sm';'Eu';'Gd';'Tb';'Dy';'Ho';'Er';'Tm';'Yb';...
63     'Lu';'Hf';'Ta';'W';'Re';'Os';'Ir';'Pt';'Au';'Hg';'Tl';'Pb';'Bi';...
64     'Po';'At';'Rn';'Fr';'Ra';'Ac';'Th';'Pa';'U'};
65
66
67 % Set axis limits
68 xmin=min(handles.energy);
69 xmax=max(handles.energy);
70 ymin=0;
71 ymax=1.1*max(handles.data);
72
73 % Create tabgroup
74 handles.tgroup = uitabgroup('Parent', handles.figure1,...
75     'SelectionChangedFcn',@tab_changed);
76
77 % Add name of tab 2 to handles structure for callback function
78 handles.name_tab2 = '<html><FONT SIZE="5">Selected Elements</FONT></html>';
79
80 % Create tabs
81 tab1 = uitab('Parent', handles.tgroup, 'Title',...
82     '<html><FONT SIZE="5">Fit Model</FONT></html>');
83
84 tab7 = uitab('Parent', handles.tgroup, 'Title',...
85     '<html><FONT SIZE="5">KLM-Marker</FONT></html>');
86
87 tab2 = uitab('Parent', handles.tgroup, 'Title', handles.name_tab2);
88
89 tab3 = uitab('Parent', handles.tgroup, 'Title',...
90     '<html><FONT SIZE="5">Deconvolution</FONT></html>');
91
92 tab4 = uitab('Parent', handles.tgroup, 'Title',...
```

```
93      '<html><FONT SIZE="5">Fit-Results</FONT></html>');
94
95  tab5 = uitab('Parent', handles.tgroup, 'Title',...
96      '<html><FONT SIZE="5">Parameter</FONT></html>');
97
98  tab6 = uitab('Parent', handles.tgroup, 'Title',...
99      '<html><FONT SIZE="5">Results</FONT></html>');
100
101
102
103 % Add handle of tab4 to handles structure
104 handles.tab4=tab4;
105
106 % Add components to tabs
107 % Tab 1 = Data
108 set(handles.axes1,'Parent', tab1);
109 set(handles.axes2,'Parent', tab1);
110 set(handles.fitbutton,'Parent', tab1);
111 set(handles.lines2fit,'Parent', tab1);
112 set(handles.remove,'Parent', tab1);
113 set(handles.reset,'Parent', tab1);
114 set(handles.include_line,'Parent', tab1);
115 set(handles.panel_error,'Parent', tab1);
116 set(handles.panel_element,'Parent', tab1);
117 set(handles.panel_lines,'Parent', tab1);
118 set(handles.create_linesfile,'Parent', tab1);
119 set(handles.load_linesfile,'Parent', tab1);
120 set(handles.panel_escape,'Parent', tab1);
121 addLinLog(tab1, [handles.axes1, handles.axes2],'tab1_linlog');
122
123 % Create panel and checkbox for background
124 panel_backcheck = uipanel('Parent', tab1, 'Units', 'normal', 'Position',...
125     [0.805 0.04 0.116 0.1],'Title','Background','FontSize',14);
126
127 handles.backcheck = uicontrol('Parent', panel_backcheck,'Style',...
128     'checkbox','String','BG overlap','Units', 'normal','Position',...
129     [0.152 0 1 1],'FontSize',12);
130
131 % Create uibuttongroup for line selection
132 handles.bg_edges = uibuttongroup(tab1,'Title','Line families',...
133     'SelectionChangedFcn',{@selection_changed},...
134     'FontSize',14,'Units', 'normal','Position', [0.67 0.64 0.116 0.34]);
135
136 % Create radiobuttons for line selection
137 handles.radio_K = uicontrol('Parent', handles.bg_edges,...
138     'Style','radiobutton','String','K-lines','Units',...
139     'normal','Position', [0.152 0.84 1 0.15],'FontSize',12);
140
141 handles.radio_Ka = uicontrol('Parent', handles.bg_edges,...
142     'Style','radiobutton','String','Ka-lines','Units',...
143     'normal','Position', [0.152 0.70 1 0.15],'FontSize',12);
144
145 handles.radio_Kb = uicontrol('Parent', handles.bg_edges,...
146     'Style','radiobutton','String','Kb-lines','Units',...
147     'normal','Position', [0.152 0.56 1 0.15],'FontSize',12);
```

```
148
149 handles.radio_L1 = uicontrol('Parent', handles.bg_edges,...
150     'Style','radiobutton','String','L1-lines','Units',...
151     'normal','Position', [0.152 0.42 1 0.15],'FontSize',12);
152
153 handles.radio_L2 = uicontrol('Parent', handles.bg_edges,...
154     'Style','radiobutton','String','L2-lines','Units',...
155     'normal','Position', [0.152 0.28 1 0.15],'FontSize',12);
156
157 handles.radio_L3 = uicontrol('Parent', handles.bg_edges,...
158     'Style','radiobutton','String','L3-lines','Units',...
159     'normal','Position', [0.152 0.14 1 0.15],'FontSize',12);
160
161 handles.radio_M5 = uicontrol('Parent', handles.bg_edges,...
162     'Style','radiobutton','String','M5-lines','Units',...
163     'normal','Position', [0.152 0 1 0.15],'FontSize',12);
164
165 % Disable edges for He
166 set(handles.radio_Kb,'Enable','off');
167 set(handles.radio_L1,'Enable','off');
168 set(handles.radio_L2,'Enable','off');
169 set(handles.radio_L3,'Enable','off');
170 set(handles.radio_M5,'Enable','off');
171
172 % Fill axes1
173 axes(handles.axes1);
174 plot(handles.energy,handles.data);
175 xlim(handles.axes1,[xmin xmax]);
176 ylim(handles.axes1,[ymin ymax]);
177 xlabel('Energy [keV]');
178 ylabel('Intensity [counts]');
179 grid on
180
181 % Caption for axes2
182 axes(handles.axes2);
183 xlim(handles.axes2,[xmin xmax]);
184 ylim(handles.axes2,[ymin ymax]);
185 xlabel('Energy [keV]');
186 ylabel('Intensity [counts]');
187 grid on
188
189 % Fill list with elements
190 set(handles.list_elements,'String',handles.elements);
191
192 % Tab 7 = KLM-Marker
193
194 handles.axes8 = axes('Parent',tab7,'Units','normalized',...
195     'Position',[0.06 0.11 0.7 0.81]);
196 addLinLog(tab7, handles.axes8,'tab7_linlog');
197
198 plot(handles.energy,handles.data);
199 xlabel('Energy [keV]');
200 ylabel('Intensity [counts]');
201
202 handles.list_KLM=uicontrol('Parent',tab7,'Style','Listbox',...
```

```
203       'Units','normalized','Position',[0.78  0.26  0.08  0.45],...
204       'FontSize',16,'Callback',@changeKLM);
205
206   handles.errorKLM=uicontrol('Parent',tab7,'Style','Text','String',...
207       'no Error','Units','normalized','Position',[0.78  0.11  0.12  0.10],...
208       'FontSize',16);
209
210   set(handles.list_KLM,'String',handles.elements);
211
212   a=1.025;
213   b=a+0.02;
214   c=0.8;
215
216   text(a, c+4*0.045, 'T', 'Color', 'r','Units','normalized','Fontsize',16);
217   text(b, c+4*0.045, 'K-lines','Units','normalized','Fontsize',16);
218
219   text(a, c+3*0.045, 'T', 'Color', 'm','Units','normalized','Fontsize',16);
220   text(b, c+3*0.045, 'L1-lines','Units','normalized','Fontsize',16);
221
222   text(a, c+2*0.045, 'T', 'Color', 'g','Units','normalized','Fontsize',16);
223   text(b, c+2*0.045, 'L2-lines','Units','normalized','Fontsize',16);
224
225   text(a, c+0.045, 'T', 'Color', 'k','Units','normalized','Fontsize',16);
226   text(b, c+0.045, 'L3-lines','Units','normalized','Fontsize',16);
227
228   text(a, c, 'T', 'Color', 'y','Units','normalized','Fontsize',16);
229   text(b, c, 'M5-lines','Units','normalized','Fontsize',16);
230
231   %Fill axes 8
232   axes(handles.axes8);
233   xlim(handles.axes8,[xmin xmax]);
234   ylim(handles.axes8,[ymin ymax]);
235   xlabel('Energy [keV]');
236   ylabel('Intensity [counts]');
237   grid on
238
239   % Tab 2 = Selected Elements
240   handles.axes4 = axes('Parent',tab2,'Units','normalized',...
241       'Position',[0.06  0.11  0.86  0.82]);
242   addLinLog(tab2, handles.axes4,'tab2_linlog');
243
244   % Tab 3 = Deconvolution
245   handles.axes5 = axes('Parent',tab3,'Units','normalized',...
246       'Position',[0.06  0.11  0.86  0.82]);
247   addLinLog(tab3, handles.axes5,'tab3_linlog');
248
249   % Fill axes5
250   axes(handles.axes5);
251   plot(handles.energy,handles.data);
252   xlim(handles.axes5,[xmin xmax]);
253   ylim(handles.axes5,[ymin ymax]);
254   xlabel('Energy [keV]');
255   ylabel('Intensity [counts]');
256   grid on
257
```

```
% Tab 4 = Fit-Results
handles.axes6 = axes('Parent',tab4,'Units','normalized',...
    'Position',[0.06 0.44 0.86 0.50]);

handles.axes7 = axes('Parent',tab4,'Units','normalized',...
    'Position',[0.06 0.09 0.86 0.25]);

addLinLog(tab4, handles.axes6,'tab4_linlog');

% Tab 5 = Parameter
handles.table_para = uitable('Parent', tab5,'RowName',{},...
    'Data',parameter,'FontSize',12,...
    'ColumnWidth',{300 300},'ColumnName',...
    {'<html><FONT SIZE="5">Parameter</FONT></html>';...
    '<html><FONT SIZE="5">Value</FONT></html>'},...
    'ColumnEditable',[false true],...
    'Units', 'normal', 'Position', [0.02 0.05 0.5 0.90]);

panel_robust = uipanel('Parent', tab5,'Position', [0.55 0.42 0.2 0.55],...
    'Title','Robust fitting','FontSize',14);

handles.robust_type = uicontrol('Parent', panel_robust,'Style','popup',...
    'String', {'off','LAR','Bisquare'},'FontSize',12,...
    'Units', 'normal','Position', [0.1 0.8 0.8 0.1]);

uicontrol('Parent', panel_robust,'Style','text',...
    'Units', 'normal','Position',[0.1 0.3 0.8 0.4],...
    'HorizontalAlignment','left','FontSize',11,...
    'String',['"LAR" changes the weights after each fit ',...
    'with respect to the absolute residuals']);

uicontrol('Parent', panel_robust,'Style','text',...
    'Units', 'normal','Position',[0.1 0.05 0.8 0.4],...
    'HorizontalAlignment','left','FontSize',11,...
    'String',['"Bisquare" changes the weights after each fit ',...
    'with respect to the squared residuals.',...
    'This method is less affected by outliners.']);

% Tab 6 = Results
handles.save = uicontrol('Parent',tab6,'String','Save Results','Units',...
    'normalized','Position',[0.02 0.044 0.13 0.06],'FontSize',16,...
    'Style','pushbutton','Callback',@save_Callback);


handles.check_fig = uicontrol('Parent',tab6,'String','Save figures',...
    'Units','normalized','Position',[0.2 0.044 0.13 0.06],...
    'FontSize',16,'Style', 'checkbox');

panel_gof = uipanel('Parent', tab6, 'Units', 'normal',...
    'Position', [0.74 0.05 0.25 0.9]);

handles.table_results = uitable('Parent', tab6,'RowName',{},...
    'ColumnWidth',{100 75 130 150 100 150},'ColumnName',...
    {'<html><FONT SIZE="5">Element</FONT></html>';...
    '<html><FONT SIZE="5">Line</FONT></html>';...
```

```
    '<html><FONT SIZE="5">Intensity</FONT></html>';...
    '<html><FONT SIZE="5">Std. Deviation</FONT></html>';...
    '<html><FONT SIZE="5">Chi-Square</FONT></html>';...
    '<html><FONT SIZE="5">Background</FONT></html>'},...
    'Units', 'normal', 'Position', [0.02 0.15 0.7 0.8]);

handles.table_gof = uitable('Parent', panel_gof,'RowName',{},...
    'ColumnWidth',{90 90},'ColumnName',[],'Units', 'normal',...
    'Position', [0.05 0.34 0.9 0.26]);

handles.table_fitpara = uitable('Parent', panel_gof,'RowName',{},...
    'ColumnWidth',{110 90},'ColumnName',[],'Units','normal',...
    'Position', [0.05 0.68 0.9 0.26]);

handles.table_LOD = uitable('Parent', panel_gof,'RowName',{},...
    'ColumnWidth',{90 80 100},'ColumnName',...
    {'<html><FONT SIZE="5">Element</FONT></html>';...
    '<html><FONT SIZE="5">Edge</FONT></html>';...
    '<html><FONT SIZE="5">LOD/el.conc.</FONT></html>'},...
    'Units','normal','Position', [0.05 0.02 0.9 0.26]);

uicontrol('Parent', panel_gof,'Style','text','String',...
    'LOD (calc. with alpha-lines)','Units', 'normal',...
    'Position', [0.05 0.28 0.9 0.05],'FontSize',14);

uicontrol('Parent', panel_gof,'Style','text','String',...
    'GOF-Parameter','Units', 'normal', 'Position',...
    [0.05 0.28+0.33 0.9 0.05],'FontSize',14);

uicontrol('Parent', panel_gof,'Style','text','String',...
    'FIT-Parameter','Units', 'normal', 'Position',...
    [0.05 0.28+0.66 0.9 0.05],'FontSize',14);

% Set button fit,save and reset inactive
set(handles.save,'Enable','off');
set(handles.fitbutton,'Enable','off');
set(handles.reset,'Enable','off');
set(handles.remove,'Enable','off');


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Element Library path
path_lib=parameter{1,2};
file_lib=parameter{2,2};

% Read library output is a cell
%cell{element,1}->Klines
%cell{element,2}->L1lines
%cell{element,3}->L2lines
%cell{element,4}->L3lines
%cell{element,x}{1,1}-->String of line
%cell{element,x}{1,2}-->energy of line
%cell{element,x}{1,3}-->relativ of line

```

```
368 [handles.all_lines]=read_library(path_lib,file_lib);
369
370 % Initialisation
371 handles.energyK=0;
372 handles.energyL1=0;
373 handles.energyL2=0;
374 handles.energyL3=0;
375 handles.energyM5=0;
376 handles.relK=0;
377 handles.relL1=0;
378 handles.relL2=0;
379 handles.relL3=0;
380 handles.relM5=0;
381
382 % Initialise Lines2fit table
383 data2fit={};
384 set(handles.lines2fit,'Data',data2fit);
385
386 % Select more than one line
387 set(handles.list_lines,'Max',2);
388
389 % Choose default command line output for fit_program_main
390 handles.output = hObject;
391
392 % Update handles structure
393 guidata(hObject, handles);
394
395 % --- Outputs from this function are returned to the command line.
396 function varargout = fit_program_main_OutputFcn(hObject,eventdata,handles)
397
398 % Get default command line output from handles structure
399 varargout{1} = handles.output;
400
401 function selection_changed(hObject,event)
402
403 % Retrieve handles structure
404 handles = guidata(hObject);
405
406 % Plot selected edges
407 list_elements_Callback(handles.list_elements,event,handles)
408
409 function changeKLM(hObject,event)
410
411
412 % Retrieve handles structure
413 handles = guidata(hObject);
414
415
416 %Reset error
417 set(handles.errorKLM,'String','');
418
419 % Get selected element
420 [~,element_num] = get_element_line(handles.list_KLM);
421
422 % Get all edges from selected element
```

```
423 data_edges = get_element_edge_data(element_num,handles.all_lines);
424
425 axes(handles.axes8);
426 plot(handles.energy,handles.data);
427 xlabel('Energy [keV]');
428 ylabel('Intensity [counts]');
429 grid on
430
431 hold on
432
433 plot_KLMmarker(handles.data,handles.energy,handles.errorKLM,data_edges)
434
435
436 hold off
437 % Retrieve the status of the radiobutton group (lin/log)
438 linlog_group = findobj('Tag','tab7_linlog');
439 radio = get(linlog_group,'SelectedObject');
440 string = get(radio,'String');
441
442 % Set the Scale
443 if strcmp(string,'lin')
444     set(handles.axes8,'YScale','lin');
445 else
446     set(handles.axes8,'YScale','log');
447 end
448 % Set axis limits
449 xmin=min(handles.energy);
450 xmax=max(handles.energy);
451 ymin=0;
452 ymax=1.1*max(handles.data);
453
454 xlim(handles.axes8,[xmin xmax]);
455 ylim(handles.axes8,[ymin ymax]);
456
457 a=1.025;
458 b=a+0.02;
459 c=0.8;
460
461 text(a, c+4*0.045, 'T', 'Color', 'r','Units','normalized','Fontsize',16);
462 text(b, c+4*0.045, 'K-lines','Units','normalized','Fontsize',16);
463
464 text(a, c+3*0.045, 'T', 'Color', 'm','Units','normalized','Fontsize',16);
465 text(b, c+3*0.045, 'L1-lines','Units','normalized','Fontsize',16);
466
467 text(a, c+2*0.045, 'T', 'Color', 'g','Units','normalized','Fontsize',16);
468 text(b, c+2*0.045, 'L2-lines','Units','normalized','Fontsize',16);
469
470 text(a, c+0.045, 'T', 'Color', 'k','Units','normalized','Fontsize',16);
471 text(b, c+0.045, 'L3-lines','Units','normalized','Fontsize',16);
472
473 text(a, c, 'T', 'Color', 'y','Units','normalized','Fontsize',16);
474 text(b, c, 'M5-lines','Units','normalized','Fontsize',16);
475
476 function tab_changed(hObject,~)
477
```

```
478 % Retrieve handles structure
479 handles = guidata(hObject);
480
481 % Retrieve selected tab
482 selected_tab = get(handles.tgroup,'SelectedTab');
483
484 % If selected tab is tab 2 plot selected lines
485 if strcmp(selected_tab.Title,handles.name_tab2)
486
487     % Reset linemarker on tab2
488     cla(handles.axes4,'reset')
489
490     % Set axis limits
491     xmin=min(handles.energy);
492     xmax=max(handles.energy);
493     ymin=0;
494     ymax=1.1*max(handles.data);
495
496     axes(handles.axes4);
497     plot(handles.energy,handles.data);
498     xlim(handles.axes4,[xmin xmax]);
499     ylim(handles.axes4,[ymin ymax]);
500     xlabel('Energy [keV]');
501     ylabel('Intensity [counts]');
502
503     % Retrieve the status of the radiobutton group (lin/log)
504     linlog_group = findobj('Tag','tab2_linlog');
505     radio = get(linlog_group,'SelectedObject');
506     string = get(radio,'String');
507
508     % Set the Scale
509     if strcmp(string,'lin')
510         set(handles.axes4,'YScale','lin');
511     else
512         set(handles.axes4,'YScale','log');
513     end
514
515     % Retrieve lines2fit
516     data2fit = get(handles.lines2fit,'Data');
517
518     % Number of edges to fit
519     size_edges = size(data2fit,1);
520
521     for i=1:size_edges
522         % Find element of edge i
523         element_num=find(strcmp(handles.elements,data2fit{i,1})==1)+1;
524
525         % Retrieve edge data for specific element
526         data_edges = get_element_edge_data(element_num,handles.all_lines);
527
528         % Retrieve line energies for selected edges
529         [~,energy,rel,~] = data_for_selected_edge(data_edges,...
530             data2fit{i,2},handles.edges);
531         % Plot lines for edge i
532         plotall_linemarker(energy,rel,handles.data,handles.energy,...
```

```
533                 handles.axes4,handles.error,'r')
534     end
535
536 end
537
538 % --- Executes on selection change in list_elements.
539 function list_elements_Callback(hObject, eventdata, handles)
540
541 % Set include button inactive
542 set(handles.include_line,'Enable','off');
543
544 % Reset error
545 set(handles.error,'String',{'no Error'});
546
547 % Set all radiobuttons inactive
548 set(handles.radio_K,'Enable','off')
549 set(handles.radio_Ka,'Enable','off')
550 set(handles.radio_Kb,'Enable','off')
551 set(handles.radio_L1,'Enable','off')
552 set(handles.radio_L2,'Enable','off')
553 set(handles.radio_L3,'Enable','off')
554 set(handles.radio_M5,'Enable','off')
555
556
557 % Reset default value from list_lines
558 set(handles.list_lines,'Value',1);
559
560 % Get selected element
561 [~,element_num] = get_element_line(handles.list_elements);
562
563 % Get all edges from selected element
564 data_edges = get_element_edge_data(element_num,handles.all_lines);
565
566 % Initialise variable
567 index=[0,0,0,0,0,0,0];
568
569 % Ask if one of the edges is empty
570 if ~isempty(data_edges.currentKenergy)
571     index(1)=1;
572 end
573
574 if ~isempty(data_edges.currentL1energy)
575     index(2)=1;
576 end
577
578 if ~isempty(data_edges.currentL2energy)
579     index(3)=1;
580 end
581
582 if ~isempty(data_edges.currentL3energy)
583     index(4)=1;
584 end
585
586 if ~isempty(data_edges.currentKaenergy)
587     index(5)=1;
```

```
588 end
589
590 if ~isempty(data_edges.currentKbenergy)
591     index(6)=1;
592 end
593
594 if ~isempty(data_edges.currentM5energy)
595     index(7)=1;
596 end
597
598 % If no edge has been found error
599 if nnz(index)==0
600     set(handles.list_lines, 'String', {'no lines'});
601     set(handles.error,'String',...
602         'Error: No lines for this element in library!!!');
603
604 % If edge has been found
605 else
606     % Find index of available lines
607     indexlines=find(index==1);
608
609     % Set available Edges active
610     if any(indexlines == 1)
611         set(handles.radio_K,'Enable','on')
612     end
613
614     if any(indexlines == 5)
615         set(handles.radio_Ka,'Enable','on')
616     end
617
618     if any(indexlines == 6)
619         set(handles.radio_Kb,'Enable','on')
620     end
621
622     if any(indexlines == 2)
623         set(handles.radio_L1,'Enable','on')
624     end
625
626     if any(indexlines == 3)
627         set(handles.radio_L2,'Enable','on')
628     end
629
630     if any(indexlines == 4)
631         set(handles.radio_L3,'Enable','on')
632     end
633
634     if any(indexlines == 7)
635         set(handles.radio_M5,'Enable','on')
636     end
637
638     % Retrieve selected radio box
639     selected = get(handles.bg_edges,'SelectedObject');
640
641     % Retrieve status of selected box
642     enable=get(selected,'Enable');
```

```
643
644     % Set radiobutton to the first available edge
645     if strcmp(enable,'off')
646         switch indexlines(1)
647             case 1
648                 set(handles.bg_edges,'SelectedObject',handles.radio_K)
649             case 2
650                 set(handles.bg_edges,'SelectedObject',handles.radio_L1)
651             case 3
652                 set(handles.bg_edges,'SelectedObject',handles.radio_L2)
653             case 4
654                 set(handles.bg_edges,'SelectedObject',handles.radio_L3)
655             case 5
656                 set(handles.bg_edges,'SelectedObject',handles.radio_Ka)
657             case 6
658                 set(handles.bg_edges,'SelectedObject',handles.radio_Kb)
659             case 7
660                 set(handles.bg_edges,'SelectedObject',handles.radio_M5)
661         end
662     end
663
664     % Retrieve selected radio box
665     selected = get(handles.bg_edges,'SelectedObject');
666
667     % Retrieve selected string
668     handles.selected_str=get(selected,'String');
669
670     % Plot the edge that corresponds to the selected radiobutton
671     switch handles.selected_str
672         case 'K-lines'
673             set(handles.list_lines,'String',data_edges.currentKstring);
674             plot_linemarker(data_edges.currentKenergy,...
675                 data_edges.currentKrel,handles.data,handles.energy,...
676                 handles.axes2,handles.error,'r')
677         case 'L1-lines'
678             set(handles.list_lines,'String',data_edges.currentL1string);
679             plot_linemarker(data_edges.currentL1energy,...
680                 data_edges.currentL1rel,handles.data,handles.energy,...
681                 handles.axes2,handles.error,'k')
682         case 'L2-lines'
683             set(handles.list_lines,'String',data_edges.currentL2string);
684             plot_linemarker(data_edges.currentL2energy,...
685                 data_edges.currentL2rel,handles.data,handles.energy,...
686                 handles.axes2,handles.error,'c')
687         case 'L3-lines'
688             set(handles.list_lines,'String',data_edges.currentL3string);
689             plot_linemarker(data_edges.currentL3energy,...
690                 data_edges.currentL3rel,handles.data,handles.energy,...
691                 handles.axes2,handles.error,'m')
692         case 'Ka-lines'
693             set(handles.list_lines,'String',data_edges.currentKastring);
694             plot_linemarker(data_edges.currentKaenergy,...
695                 data_edges.currentKarel,handles.data,handles.energy,...
696                 handles.axes2,handles.error,'r')
697         case 'Kb-lines'
```

```
698             set(handles.list_lines,'String',data_edges.currentKbstring);
699             plot_linemarker(data_edges.currentKbenergy,...
700                 data_edges.currentKbrel,handles.data,handles.energy,...
701                 handles.axes2,handles.error,'r')
702         case 'M5-lines'
703             set(handles.list_lines,'String',data_edges.currentM5string);
704             plot_linemarker(data_edges.currentM5energy,...
705                 data_edges.currentM5rel,handles.data,handles.energy,...
706                 handles.axes2,handles.error,'k')
707     end
708
709 end
710
711 % Get table with lines2fit
712 table=get(handles.lines2fit,'Data');
713
714 % Get selected element
715 [element_string] = get_element_line(handles.list_elements);
716
717
718 % Retrieve error
719 error=get(handles.error,'String');
720
721 set(handles.include_line,'Enable','on');
722
723 % If no error
724 if ~strcmp(error,'no Error')
725     set(handles.include_line,'Enable','off');
726 end
727
728 % If same element and same line does not exist in lines2fit
729 for i=1:size(table,1);
730     if strcmp(table(i,1),element_string) && strcmp(table(i,2),...
731             handles.selected_str)
732         set(handles.include_line,'Enable','off');
733         set(handles.error,'String','Edge already included');
734     end
735 end
736
737 % Add data_edges to handles structure
738 handles.data_edges=data_edges;
739
740 % Update handles structure
741 guidata(hObject,handles)
742
743
744
745 % --- Executes on button press in include_line.
746 function include_line_Callback(hObject, eventdata, handles)
747
748 % Get selected element
749 [element_string,~] = get_element_line(handles.list_elements);
750
751 % Retrieve data from data2fit table
752 data2fit=get(handles.lines2fit,'Data');
```

```
753
754 % Add data to table
755 if isempty(data2fit)
756     data2fit{1,1}=char(element_string);
757     data2fit{1,2}=char(handles.selected_str);
758 else
759     data2fit{end+1,1}=char(element_string);
760     data2fit{end,2}=char(handles.selected_str);
761 end
762
763 % Change buttons active/inactive
764 set(handles.include_line,'Enable','off');
765 set(handles.reset,'Enable','on');
766 set(handles.remove,'Enable','on');
767
768 % Show the selected elements and lines in uitable
769 set(handles.lines2fit,'Data',data2fit);
770
771 % Enable Fit button
772 set(handles.fitbutton,'Enable','on')
773
774 % Set element list as active object to prevent an additional buttonclick in
775 % the listbox after pressing the fitbutton.
776 uicontrol(handles.list_elements);
777
778 % Update handles structure
779 guidata(hObject,handles)
780
781
782 % --- Executes on button press in fitbutton.
783 function fitbutton_Callback(hObject, eventdata, handles)
784
785 % Retrieve data from lines2fit table
786 data_2fit=get(handles.lines2fit,'Data');
787
788 % Number of edges to fit
789 size_edges=size(data_2fit,1);
790
791 % Reset
792 handles.elnum=[];
793 handles.linenum=[];
794 handles.energyasr=[];
795 handles.elnum(1:size_edges)=0;
796 handles.linenum(1:size_edges)=0;
797 handles.energyasr(1:size_edges)=0;
798 cellenergy=cell(size_edges);
799 cellrel=cell(size_edges);
800 cellstr=cell(size_edges);
801
802 % Generate data for fit model
803 for i=1:size_edges
804
805     edge_str=data_2fit{i,2};
806     element_str=data_2fit{i,1};
807     element_num=find(strcmp(handles.elements,element_str)==1)+1;
```

```
808
809     % Retrieve edge data for element_str
810     data_edges = get_element_edge_data(element_num,handles.all_lines);
811
812     % Choose the right energy for selected line to generate formula
813     [strlines,energy,rel,num] = data_for_selected_edge(data_edges,...
814         edge_str,handles.edges);
815
816     handles.elnum(i)=element_num;
817     handles.linenum(i)=num;
818     handles.energyasr(i)=energy(1);
819     cellenergy{i}=energy;
820     cellrel{i}=rel;
821     cellstr{i}=strlines;
822 end
823
824
825 % Escape or not
826 escape=get(handles.tick_escape,'Value');
827
828 % Get fit_parameter
829 parameter=get(handles.table_para,'Data');
830
831 % Retrieve the robust option
832 robust_val = get(handles.robust_type,'Value');
833 robust_list = get(handles.robust_type,'String');
834 robust = robust_list{robust_val};
835
836 % Reset axes
837 cla(handles.axes5);
838 cla(handles.axes6);
839 cla(handles.axes7);
840
841 % Change selected tab to fit_results
842 handles.tgroup.SelectedTab=handles.tab4;
843
844 % Call fit function
845 input.robust = robust;
846 input.energyin = handles.energy;
847 input.datain = handles.data;
848 input.cellenergy = cellenergy;
849 input.cellstr = cellstr;
850 input.linenum = handles.linenum;
851 input.cellrel = cellrel;
852 input.axes_result = handles.axes6;
853 input.axes_residual = handles.axes7;
854 input.parameter = parameter;
855 input.escape = escape;
856 input.sum = get(handles.ckSumPeaks,'Value');
857 input.background = handles.background;
858 input.zero = handles.zero;
859 input.gain = handles.gain;
860 input.channelroi = handles.channel;
861 input.backcheck = get(handles.backcheck,'Value');
862 input.lines2fit = handles.lines2fit;
```

```
863
864 % Call fit function
865 [output] = fit_results(input);
866
867 % Arrange output
868 handles.aparameter = output.aparameter; % Fitparameter (for Area)
869 fitresult = output.fitresult; % Handle to fitresult
870 gof = output.gof; % Handle to godness of fit
871 handles.plotelextra = output.plotelextra; % Separat functions for each edge
872 handles.allparameter = output.fitparameter; % All fitparameter
873
874 % Generate data for gof table
875 table_gof(:,1)={'rsquare','sse','dfe','adjrsquare','rmse'};
876 table_gof(:,2)=num2cell([gof.rsquare,gof.sse,gof.dfe,...
877     gof.adjrsquare,gof.rmse]);
878
879 % Set goodness of fit
880 set(handles.table_gof,'Data',table_gof,'FontSize',12);
881 %handles.table_gof.Position(3) = handles.table_gof.Extent(3);
882
883 % Generate data for fitparameter table
884 numapara=size(handles.aparameter,2);
885 zero=handles.allparameter(numapara+1);
886 gain=handles.allparameter(numapara+2);
887 noise=handles.allparameter(numapara+3);
888 fano=handles.allparameter(numapara+4);
889
890 table_fitparameter(:,1)={'ZERO','GAIN','NOISE','FANO','CHI_SQUARE'};
891 table_fitparameter(:,2)=num2cell([zero,gain,noise,fano,...
892     output.red_chi_square]);
893
894 % Set parameter of fit
895 set(handles.table_fitpara,'Data',table_fitparameter,'FontSize',12);
896
897 % Get names of coefficients
898 names=coeffnames(fitresult);
899
900 [fullarea,fullstddev,area_extra,std_extra,Ydata,area_asr,std_asr]...
901     = seperate_peakarea(handles.aparameter,handles.allparameter,...
902     handles.plotelextra,handles.channel,names,output.std_dev,...
903     input.cellrel,output.index_asr);
904
905 % Set axis limits
906 xmin=min(handles.energy);
907 xmax=max(handles.energy);
908
909 % Plot function for edges separately
910 ys = get(handles.axes5,'YScale');
911 if strcmp(ys, 'lin')
912     ymin=0;
913 else
914     ymin = 0.1;
915 end
916 ymax=max(handles.data);
917
```

```
axes(handles.axes5);
xlabel('Energy [keV]');
ylabel('Intensity [counts]');
xlim(handles.axes5,[xmin xmax]);
ylim(handles.axes5,[ymin ymax]);

hold on

% Plot spectrum
plot(handles.energy,handles.data,'k.');

% Plot function for each edge
for i=1:size(handles.aparameter,2)
    if i<7
        plot(handles.energy,Ydata(i,:),'-','LineWidth',1.5);
    elseif i<14
        plot(handles.energy,Ydata(i,:),'--','LineWidth',1.5);
    else
        plot(handles.energy,Ydata(i,:),'-.','LineWidth',1.5);
    end
end

% Generate data for legend
data = get(handles.lines2fit, 'data');
strelements=data(:,1);
strlines=data(:,2);

strall=strcat(strelements,'-',strlines)';

legendlabel=horzcat('Spectrum',strall);

legend(legendlabel,'Location', 'Best' );


% Function for background color
colergen = @(color,text) ['<html><table border=0 width=400 bgcolor=',...
    color,'><TR><TD>',text,'</TD></TR> </table></html>'];

% Generate data for limit of detection
size_edges=size(fullarea,2);

index_LOD=0;
data_LOD{1,1}='';

for i=1:size_edges
    if ~(output.back_peak_asr(i)==0)
        index_LOD=index_LOD+1;
        LOD=(3*sqrt(output.back_peak_asr(i)))/area_asr(i);
        data_LOD{index_LOD,1}=char(strelements(i));
        data_LOD{index_LOD,2}=char(strlines(i));
        data_LOD{index_LOD,3}=sprintf('%.5f',LOD);
    end
end

set(handles.table_LOD,'Data',data_LOD,'FontSize',12);
```

```
973
974
975 % Generate data for results
976 colorel='#00ff7f'; % grün
977 index=1;
978
979 % Loop for generating data to fill the results table
980 for i=1:size_edges
981     data{index,1}=colergen(colorel,char(strelements(i)));
982     data{index,2}=colergen(colorel,char(strlines(i)));
983     data{index,3}=colergen(colorel,sprintf('%.0f',fullarea(i)));
984     data{index,4}=colergen(colorel,sprintf('± %.0f',fullstddev(i)));
985     data{index,5}=colergen(colorel,sprintf('--------'));
986     data{index,6}=colergen(colorel,sprintf('--------'));
987
988     size_lines=size(area_extra{i},1);
989
990     for j=1:size_lines
991         index=index+1;
992
993         data{index,1}=sprintf('');
994         data{index,2}=char(cellstr{i}(j,1));
995         data{index,3}=sprintf('%.0f',area_extra{i}(j,1));
996         data{index,4}=sprintf('± %.0f',std_extra{i}(j,1));
997         data{index,5}=sprintf('%.2f',output.chi_peak_cell{i}(1,j));
998         if isnan(output.chi_peak_cell{i}(1,j))
999             data{index,5}=colergen('#FF0000','Out of ROI');
1000        end
1001        data{index,6}=sprintf('%.0f',output.back_peak_cell{i}(1,j));
1002        if isnan(output.back_peak_cell{i}(1,j))
1003            data{index,6}=colergen('#FF0000','Out of ROI');
1004        end
1005    end
1006    index=index+1;
1007 end
1008
1009 % Fill table
1010 set(handles.table_results,'Data',data,'FontSize',12);
1011
1012 %
1013 %%%%%%%%%%%%%Begin Save FITRESULT AXES%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1014 %%Name of the current file
1015 % [~,name,~]=fileparts(handles.filename);
1016 %
1017 %%path of fit results graphic
1018 % path_fig=['D:\Cloud Storage\Dropbox\TU-Wien\Diplomarbeit\LateX\Master
      Thesis\pictures\figures\' name '_fit'];
1019 %
1020 %
1021 %%set output pdf format to fitresults figure size
1022 % set(output.fig_fit,'Units','Inches');
1023 % pos = get(output.fig_fit,'Position');
1024 % set(output.fig_fit,'PaperPositionMode','Auto','PaperUnits','Inches','
      PaperSize',[pos(3), pos(4)])
1025 %
```

```
1026 %%print figure as pdf vector graphic
1027 % print(output.fig_fit,path_fig,'-dpdf','-r0');
1028 %
1029 %%close figure
1030 % close(output.fig_fit);
1031 %
1032 %%path of residual graphic
1033 % path_res=['D:\Cloud Storage\Dropbox\TU-Wien\Diplomarbeit\LateX\Master
       Thesis\pictures\figures\' name '_res'];
1034 %
1035 %%set output pdf format to residual figure size
1036 % set(output.fig_res,'Units','Inches');
1037 % pos = get(output.fig_res,'Position');
1038 % set(output.fig_res,'PaperPositionMode','Auto','PaperUnits','Inches','
       PaperSize',[pos(3), pos(4)])
1039 %
1040 %%print figure as pdf vector graphic
1041 % print(output.fig_res,path_res,'-dpdf','-r0');
1042 %
1043 %%close figure
1044 % close(output.fig_res);
1045 %%%%%%%%%%%%%%%END Save FITRESULT AXES%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1046
1047 % Add results to handle structure
1048 handles.area_asr=area_asr;
1049 handles.std_asr=std_asr;
1050 handles.chi_peak_asr = output.chi_peak_asr;
1051 handles.fig_fit = output.fig_fit;
1052 handles.fig_res = output.fig_res;
1053
1054
1055 % Set save button enable after fit
1056 set(handles.save,'Enable','on')
1057
1058 % Update handles structure
1059 guidata(hObject,handles)
1060
1061 % --- Executes on button press in reset.
1062 function reset_Callback(hObject, eventdata, handles)
1063
1064 set(handles.fitbutton,'Enable','off');
1065
1066 % Reset data in lines2fit table
1067 set(handles.lines2fit,'Data',{});
1068
1069 % Update handles structure
1070 guidata(hObject,handles)
1071
1072
1073 % --- Executes on button press in save.
1074 function save_Callback(hObject, eventdata)
1075
1076 handles = guidata(hObject);
1077
1078 % Ask for file path and name input for result file
```

```
[filename_res,path_res]=uiputfile( ...
{   '*.ASR','ASR-files (*.ASR)'; ...
    '*.*',  'All Files (*.*)'}, ...
    'Please choose filename for results file:',handles.path);

% Write data to file
write2asrfile(filename_res,path_res,handles.linenum,handles.elnum,...
     handles.header,handles.energyasr,handles.area_asr,...
     handles.std_asr,handles.chi_peak_asr)

fig_check = get(handles.check_fig,'Value');

if fig_check == 1

     [~,name,~]=fileparts(filename_res);

     %path of fit results graphic
     path_fig=[path_res name '_fit'];

     %set output pdf format to fitresults figure size
     set(handles.fig_fit,'Units','Inches');
     pos = get(handles.fig_fit,'Position');
     set(handles.fig_fit,'PaperPositionMode','Auto','PaperUnits',...
          'Inches','PaperSize',[pos(3), pos(4)])

     %print figure as pdf vector graphic
     print(handles.fig_fit,path_fig,'-dpdf','-r0');

     %path of residual graphic
     path_res=[path_res name '_res'];

     %set output pdf format to residual figure size
     set(handles.fig_res,'Units','Inches');
     pos = get(handles.fig_res,'Position');
     set(handles.fig_res,'PaperPositionMode','Auto','PaperUnits',...
          'Inches','PaperSize',[pos(3), pos(4)])

     %print figure as pdf vector graphic
     print(handles.fig_res,path_res,'-dpdf','-r0');

end

% --- Executes on button press in create_linesfile.
function create_linesfile_Callback(hObject, eventdata, handles)

% Ask for file path and name input for result file
[filename_lines,path_lines]=uiputfile( ...
{   '*.txt','txt-files (*.txt)'; ...
    '*.*',  'All Files (*.*)'}, ...
    'Please choose filename for lines file:',handles.path);

% Open result file
fileID_lines = fopen(fullfile(path_lines,filename_lines),'w');

% Retrieve lines to fit
```

```
1134 element_line=get(handles.lines2fit,'Data');
1135 num=size(element_line,1);
1136
1137 % Write into result file
1138
1139 fprintf(fileID_lines,'Lines_begin\n');
1140 fprintf(fileID_lines,'-----------------------------------\n');
1141
1142 for i=1:num
1143     fprintf(fileID_lines,'%s %s \n',element_line{i,:});
1144 end
1145
1146 fprintf(fileID_lines,'-----------------------------------\n');
1147 fprintf(fileID_lines,'Lines_end\n');
1148
1149
1150 % --- Executes on button press in load_linesfile.
1151 function load_linesfile_Callback(hObject, eventdata, handles)
1152
1153 % Select path for linesfile
1154 [file,path]= uigetfile('*.txt','Select lines file',handles.path);
1155
1156 % Read lines file
1157 [elementsfile,linesfile,numberfile]=read_linesfile(path,file);
1158 data2fit=cell(numberfile,2);
1159
1160 for i=1:numberfile
1161     data2fit{i,1}=char(elementsfile(i));
1162     data2fit{i,2}=char(linesfile(i));
1163 end
1164
1165 % Show the selected elements and lines in uitable
1166 set(handles.lines2fit,'Data',data2fit);
1167
1168 set(handles.reset,'Enable','on');
1169 set(handles.remove,'Enable','on');
1170 set(handles.fitbutton,'Enable','on')
1171
1172 % Update handles structure
1173 guidata(hObject,handles)
1174
1175
1176 % --- Executes on button press in remove.
1177 function remove_Callback(hObject, eventdata, handles)
1178
1179 % Retrieve selected cell
1180 index = handles.selectedcell;
1181
1182 data_lines2fit = get(handles.lines2fit,'Data');
1183
1184 if size(data_lines2fit,1)<2
1185     set(handles.fitbutton,'Enable','off');
1186 end
1187
1188 % Remove selected line
```

```
1189 if ~isempty(index)
1190     line2remove=index(1);
1191     data_lines2fit(line2remove,:)=[];
1192     set(handles.lines2fit,'Data',data_lines2fit)
1193 end
1194
1195 % Update handles structure
1196 guidata(hObject,handles)
1197
1198
1199 % --- Executes when selected cell(s) is changed in lines2fit.
1200 function lines2fit_CellSelectionCallback(hObject, eventdata, handles)
1201
1202 % Save selected cell
1203 handles.selectedcell = eventdata.Indices;
1204
1205 % Update handles structure
1206 guidata(hObject,handles)
```

B.5 multi_fit_main.m

```
1  function varargout = multi_fit_main(varargin)
2
3  % Begin initialization code - DO NOT EDIT
4  gui_Singleton = 1;
5  gui_State = struct('gui_Name',       mfilename, ...
6                     'gui_Singleton',  gui_Singleton, ...
7                     'gui_OpeningFcn', @multi_fit_main_OpeningFcn, ...
8                     'gui_OutputFcn',  @multi_fit_main_OutputFcn, ...
9                     'gui_LayoutFcn',  [] , ...
10                    'gui_Callback',   []);
11 if nargin && ischar(varargin{1})
12     gui_State.gui_Callback = str2func(varargin{1});
13 end
14
15 if nargout
16     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
17 else
18     gui_mainfcn(gui_State, varargin{:});
19 end
20 % End initialization code - DO NOT EDIT
21
22
23 % --- Executes just before multi_fit_main is made visible.
24 function multi_fit_main_OpeningFcn(hObject, eventdata, handles, varargin)
25
26 % Name of figure
27 set(handles.figure1, 'Name', 'Multiple Fit Window');
28
29 % Path for parameter file
30 path=fileparts(which('multi_fit_main.m'));%pwd;%'\';
31 file='FIT_PARAMETER.txt';
32
33 % Read Parameter file
34 handles.parameter = read_parameterfile(path,file);
35
```

```
handles.tgroup = uitabgroup('Parent', handles.figure1);

tab1 = uitab('Parent', handles.tgroup, 'Title',...
    '<html><FONT SIZE="5">Data</FONT></html>');

tab2 = uitab('Parent', handles.tgroup, 'Title',...
    '<html><FONT SIZE="5">Fit results</FONT></html>');

set(handles.axes2,'Parent', tab2);
set(handles.axes3,'Parent', tab2);
addLinLog(tab2, handles.axes2, 'linlog');

set(handles.headline,'Parent', tab1);
set(handles.axis,'Parent', tab1);
set(handles.background,'Parent', tab1);
set(handles.spectra,'Parent', tab1);
set(handles.lines,'Parent', tab1);
set(handles.fit_all,'Parent', tab1);
set(handles.panelescape,'Parent', tab1);

set(handles.panelnumber,'Parent', tab1);
set(handles.panelcurrent,'Parent', tab1);


% Choose default command line output for multi_fit_main
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes multi_fit_main wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = multi_fit_main_OutputFcn(hObject, eventdata, handles)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in select_background.
function select_background_Callback(hObject, eventdata, handles)

%get handle of selected relative sensitivities file
get(handles.txt_background);

%Open path for relative sensitivities file
[handles.filename_back, handles.path_back,~] = uigetfile( ...
{  '*.txt','txt-files (*.txt)'; ...
   '*.*',  'All Files (*.*)'}, ...
   'Please select background file:');

%shows the file name and path in the edit text field
set(handles.txt_background,'string',...
```

```
91      fullfile(handles.path_back,handles.filename_back))
92
93
94  % Update handles structure
95  guidata(hObject, handles);
96
97
98  % --- Executes on button press in select_spe.
99  function select_spe_Callback(hObject, eventdata, handles)
100
101 %get handle of selected relative sensitivities file
102 get(handles.txt_spe);
103
104 %Open path for relative sensitivities file
105 [handles.filename_spe, handles.path_spe,~] = uigetfile( ...
106    {'*.SPE','SPE-files (*.SPE)'; ...
107    '*.*', 'All Files (*.*)'}, ...
108    'Please select spectrum:', ...
109     'MultiSelect', 'on');
110
111 %shows the file name and path in the edit text field
112 set(handles.txt_spe,'string',...
113     fullfile(handles.path_spe,handles.filename_spe))
114
115
116 % Update handles structure
117 guidata(hObject, handles);
118
119 % --- Executes on button press in select_lines.
120 function select_lines_Callback(hObject, eventdata, handles)
121
122 %get handle of selected relative sensitivities file
123 get(handles.txt_lines);
124
125 %Open path for relative sensitivities file
126 [handles.filename_lines, handles.path_lines,~] = uigetfile( ...
127 {  '*.txt','txt-files (*.txt)'; ...
128    '*.*', 'All Files (*.*)'}, ...
129    'Please select lines file:');
130
131 %shows the file name and path in the edit text field
132 set(handles.txt_lines,'string',...
133     fullfile(handles.path_lines,handles.filename_lines))
134
135 % Update handles structure
136 guidata(hObject, handles);
137
138 % --- Executes on button press in select_cali.
139 function select_cali_Callback(hObject, eventdata, handles)
140
141 %get handle of selected relative sensitivities file
142 get(handles.txt_cali);
143
144 %Open path for relative sensitivities file
145 [handles.filename_cali, handles.path_cali,~] = uigetfile( ...
```

```
146 {   '*.txt','txt-files (*.txt)'; ...
147     '*.*',  'All Files (*.*)'}, ...
148     'Please select calibration file:');
149
150 %shows the file name and path in the edit text field
151 set(handles.txt_cali,'string',...
152      fullfile(handles.path_cali,handles.filename_cali))
153
154
155 % Update handles structure
156 guidata(hObject, handles);
157
158 % --- Executes on button press in fit_all.
159 function fit_all_Callback(hObject, eventdata, handles)
160
161 path_file_spe=fullfile(handles.path_spe,handles.filename_spe);
162 numfile=size(path_file_spe,2);
163
164 %if the output of filename is no cell
165      if ~iscell(handles.filename_spe)
166          handles.filename_spe={handles.filename_spe};
167          numfile=1;
168      end
169
170 set(handles.number,'String',num2str(numfile));
171
172
173 for i=1:numfile
174
175      set(handles.iteration,'String',num2str(i));
176
177      %read spe file
178      [channel,data,header]=read_spefile(handles.filename_spe{i},...
179          handles.path_spe);
180
181      [dataroi,channelroi,backgroundroi] = auto_background(data,...
182          handles.path_back,handles.filename_back);
183
184      [energyroi,zero,gain] = read_axiscalibration(channelroi,...
185          handles.filename_cali,handles.path_cali);
186
187
188
189      path=pwd;%'\';
190      file='HEAVYELE_BLEI.txt';
191
192      %Read library output is a cell
193      %cell{element,1}->Klines
194      %cell{element,2}->L1lines
195      %cell{element,3}->L2lines
196      %cell{element,4}->L3lines
197      %cell{element,x}{1,1}-->String of line
198      %cell{element,x}{1,2}-->energy of line
199      %cell{element,x}{1,3}-->relativ of line
200
```

```
201     [all_lines]=read_library(path,file);
202
203
204     [elementsfile,linesfile,numberfile]=...
205         read_linesfile(handles.path_lines,handles.filename_lines);
206
207     elements={'H';'He';'Li';'Be';'B';'C';'N';'O';'F';'Ne';'Na';'Mg';...
208         'Al';'Si';'P';'S';'Cl';'Ar';'K';'Ca';'Sc';'Ti';'V';'Cr';'Mn';...
209         'Fe';'Co';'Ni';'Cu';'Zn';'Ga';'Ge';'As';'Se';'Br';'Kr';'Rb';...
210         'Sr';'Y';'Zr';'Nb';'Mo';'Tc';'Ru';'Rh';'Pd';'Ag';'Cd';'In';...
211         'Sn';'Sb';'Te';'I';'Xe';'Cs';'Ba';'La';'Ce';'Pr';'Nd';'Pm';...
212         'Sm';'Eu';'Gd';'Tb';'Dy';'Ho';'Er';'Tm';'Yb';'Lu';'Hf';'Ta';...
213         'W';'Re';'OS';'Ir';'Pt';'Au';'Hg';'Tl';'Pb';'Bi';'Po';'At';...
214         'Rn';'Fr';'Ra';'Ac';'Th';'Pa';'U'};
215
216     edges={'K-lines','L1-lines','L2-lines','L3-lines','K-alpha',...
217         'K-beta','M5-lines'};
218
219
220     for k=1:numberfile
221
222         pos_element=strcmp(elements,elementsfile(k));
223         element_num=find(pos_element==1);
224         selected_str = linesfile(k);
225
226         data_edge = get_element_edge_data(element_num,all_lines);
227
228         [strlines,energy,rel,num] = data_for_selected_edge(data_edge,...
229             selected_str,edges);
230
231         elnum(k)=element_num;
232         linenum(k)=num;
233         energyasr(k)=energy(1);
234         cellstr{k}=strlines;
235         cellenergy{k}=energy;
236         cellrel{k}=rel;
237
238
239     end
240     input.robust = 'off';
241     input.energyin = energyroi;
242     input.datain = dataroi;
243     input.cellenergy = cellenergy;
244     input.cellrel = cellrel;
245     input.cellstr = cellstr;
246     input.axes_result = handles.axes2;
247     input.axes_residual = handles.axes3;
248     input.parameter = handles.parameter;
249     input.escape = get(handles.escape,'Value');
250     input.sum = get(handles.ckSumPeaks,'Value');
251     input.background = backgroundroi;
252     input.zero = zero;
253     input.gain = gain;
254     input.channelroi = channelroi;
255     input.linenum = linenum;
```

```
256       input.backcheck = 0;
257
258       [output] = fit_results(input);
259
260
261
262       %get names of coefficients
263       names=coeffnames(output.fitresult);
264
265
266       [~,~,~,~,~,area_asr,std_asr] = seperate_peakarea(output.aparameter,...
267       output.fitparameter,output.plotelextra,channelroi,names,...
268       output.std_dev,input.cellrel,output.index_asr);
269
270
271       filenamewr1 = strsplit(handles.filename_spe{i},'.');
272       filenamewr = char(strcat(filenamewr1(1),'.ASR'));
273
274       write2asrfile(filenamewr,handles.path_spe,linenum,elnum,header,...
275       energyasr,area_asr,std_asr,output.chi_peak_asr)
276
277   end
278
279
280
281   % Update handles structure
282   guidata(hObject, handles);
```

B.6 qtxrf.m

```
1   function varargout = qtxrf(varargin)
2
3   % Begin initialization code - DO NOT EDIT
4   gui_Singleton = 1;
5   gui_State = struct('gui_Name',       mfilename, ...
6                      'gui_Singleton',  gui_Singleton, ...
7                      'gui_OpeningFcn', @qtxrf_OpeningFcn, ...
8                      'gui_OutputFcn',  @qtxrf_OutputFcn, ...
9                      'gui_LayoutFcn',  [] , ...
10                     'gui_Callback',   []);
11  if nargin && ischar(varargin{1})
12      gui_State.gui_Callback = str2func(varargin{1});
13  end
14
15  if nargout
16      [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
17  else
18      gui_mainfcn(gui_State, varargin{:});
19  end
20  % End initialization code - DO NOT EDIT
21
22
23  % --- Executes just before qtxrf is made visible.
24  function qtxrf_OpeningFcn(hObject, eventdata, handles, varargin)
25
26  set(handles.qtxrf, 'Name', 'QTXRF Window');
```

```
27
28 % Choose default command line output for qtxrf
29 handles.output = hObject;
30
31 % Update handles structure
32 guidata(hObject, handles);
33
34
35
36 % --- Outputs from this function are returned to the command line.
37 function varargout = qtxrf_OutputFcn(hObject, eventdata, handles)
38
39 % Get default command line output from handles structure
40 varargout{1} = handles.output;
41
42 % --- Executes on button press in select_asr.
43 function calculate_Callback(hObject, eventdata, handles)
44
45 %Reset error message
46 set(handles.error_qtxrf,'String','no Error');
47
48 %read concentration of the internal standard in the quantification field
49 concentration=str2double(get(handles.file_con,'String'));
50
51 %read calibration file
52 [internal,coeffK,coeffL,coeffM,error] =...
53     read_calibrationfile(handles.filename2,handles.path2);
54
55 set(handles.error_qtxrf,'String',error);
56
57 %reads the quantification file
58 [calArray1,~] = read_asrfile(handles.filename1, handles.path1);
59
60 %linenumbers of the elements
61 linenumber=calArray1{1}{2};
62
63 %element numbers in the quantification file
64 vector=calArray1{1}{1};
65
66 %intensity of the elements in the quantification file
67 intensity=calArray1{1}{4};
68
69 %std_deviation of the intensity of the elements in the quantification file
70 std_deviation=calArray1{1}{5};
71
72 %number of the elements in the quantification file
73 number=size(intensity);
74
75 %values for the relative sensitivities in the quantification file
76 for i=1:number
77
78     switch linenumber(i)
79         %values for the relative sensitivities of the k lines
80         case 1
81             if ~isempty(coeffK)
```

```
82                  dataforquantification(i)=polyval(coeffK, vector(i));
83              else
84                  dataforquantification(i)=inf;
85                  set(handles.error_qtxrf,'String',...
86                      ['Error: No relative intensity for one '...
87                      'or more element/s in the quantification file found, '
                            ...
88                      'concentration has been set to 0 for this elements']);
89              end
90          %values for the relative sensitivities of the L lines
91          case 2
92              if ~isempty(coeffL)
93                  dataforquantification(i)=polyval(coeffL, vector(i));
94              else
95                  dataforquantification(i)=inf;
96                  set(handles.error_qtxrf,'String',...
97                      ['Error: No relative intensity for one '...
98                      'or more element/s in the quantification file found, '
                            ...
99                      'concentration has been set to 0 for this elements']);
100             end
101         %values for the relative sensitivities of the M lines
102         case 3
103             if ~isempty(coeffM)
104                 dataforquantification(i)=polyval(coeffM, vector(i));
105             else
106                 dataforquantification(i)=inf;
107                 set(handles.error_qtxrf,'String',...
108                     ['Error: No relative intensity for one '...
109                     'or more element/s in the quantification file found, '
                            ...
110                     'concentration has been set to 0 for this elements']);
111             end
112     end
113
114 end
115
116 %row index of the internal std
117 for i=1:number
118     if internal==vector(i);
119        std=i;
120     end
121 end
122
123 %calculation of the quantification
124 for i=1:number
125 conc(i)=(intensity(i)/intensity(std))*concentration...
126     *(1./dataforquantification(i));
127 concstddev(i)=(std_deviation(i)/intensity(i))*conc(i);
128 end
129
130
131 %elements of the quantification file
132 concmatr(:,1)=calArray1{1}{1};
133
```

```
134 %concentration of the quantification file
135 concmatr(:,2)=conc;
136
137 %std dev concentration of the quantification file
138 concmatr(:,3)=concstddev;
139
140
141 %Vector with elements to U
142 elements={'H';'He';'Li';'Be';'B';'C';'N';'O';'F';'Ne';'Na';'Mg';...
143     'Al';'Si';'P';'S';'Cl';'Ar';'K';'Ca';'Sc';'Ti';'V';'Cr';'Mn';...
144     'Fe';'Co';'Ni';'Cu';'Zn';'Ga';'Ge';'As';'Se';'Br';'Kr';'Rb';'Sr';...
145     'Y';'Zr';'Nb';'Mo';'Tc';'Ru';'Rh';'Pd';'Ag';'Cd';'In';'Sn';'Sb';...
146     'Te';'I';'Xe';'Cs';'Ba';'La';'Ce';'Pr';'Nd';'Pm';'Sm';'Eu';'Gd';...
147     'Tb';'Dy';'Ho';'Er';'Tm';'Yb';'Lu';'Hf';'Ta';'W';'Re';'OS';'Ir';...
148     'Pt';'Au';'Hg';'Tl';'Pb';'Bi';'Po';'Rn';'Fr';'Ra';'Ac';'Th';'Pa';'U'};
149
150 %generation of the output
151 for i=1:number
152
153 data{i,1}=char(elements(concmatr(i,1)));
154 data{i,2}=linenumber(i);
155 data{i,3}=concmatr(i,2);
156 data{i,4}=concmatr(i,3);
157 data{i,5}=char('ppm');
158 end
159
160 %Print data into table
161 set(handles.results,'Data',data);
162
163 %safe handles
164 guidata(hObject, handles);
165
166
167 % --- Executes when user attempts to close qtxrf.
168 function qtxrf_CloseRequestFcn(hObject, eventdata, handles)
169
170 % Hint: delete(hObject) closes the figure
171 delete(hObject);
172
173
174 % --- Executes on button press in select_relsen.
175 function select_relsen_Callback(hObject, eventdata, handles)
176
177 %get handle of selected relative sensitivities file
178 get(handles.selected_relsen);
179
180 %Open path for relative sensitivities file
181 [handles.filename2, handles.path2, handles.filterindex2] = uigetfile( ...
182 {  '*.txt','txt-files (*.txt)'; ...
183    '*.*',  'All Files (*.*)'}, ...
184    'Please select relative sensitivity file:');
185
186 %shows the file name and path in the edit text field
187 set(handles.selected_relsen,'string',...
188     fullfile(handles.path2,handles.filename2))
```

```
189
190 %safe handles
191 guidata(hObject, handles);
192
193 % --- Executes on button press in select_asr.
194 function select_asr_Callback(hObject, eventdata, handles)
195
196 %Select path for ASR file
197 [handles.filename1, handles.path1, handles.filterindex1] = uigetfile( ...
198 {  '*.ASR','ASR-files (*.ASR)'; ...
199    '*.*',  'All Files (*.*)'}, ...
200    'Please select file for Quantification:');
201
202 %shows the file name and path in the edit text field
203 set(handles.selected_asr,'string',...
204     fullfile(handles.path1,handles.filename1))
205
206 %safe handles
207 guidata(hObject, handles);
208
209 function create_file_Callback(hObject, eventdata, handles)
210
211 %number of filesets
212 numfilesets = qtxrf_numfilesets();
213
214
215 for i=1:numfilesets
216     %select path for spectrum
217     [file, path] = uigetfile( ...
218     {  '*.ASR','ASR-files (*.ASR)'; ...
219         '*.*',  'All Files (*.*)'}, ...
220         'Please select files for Sensitivity curve:', ...
221         'MultiSelect', 'on');
222
223     %read calibration files
224     [calArray,filename] = read_asrfile(file, path);
225
226     %number of files
227     index=size(filename,2);
228
229     %read fileconcentrations and internal standard from manual input
230     [calconcentration,internal,intconcentration,origin]=...
231         qtxrf_read_filecon_neu(filename,index);
232
233     %reads the selection of the unwanted lines
234     [matrixcal,indexorder]=...
235         qtxrf_unwanted_con(calArray,index,calconcentration);
236
237
238
239     %sets the concentration of the internal std of the files in
240     %the right order (fewer concentration of the files first)
241     intconcentration=intconcentration(indexorder);
242
243     %fit calibration lines
```

```
244     [sensitivityvalues]=qtxrf_fit_el(matrixcal,internal,...
245         intconcentration,origin);
246 
247     if numfilesets > 1
248         values{i}=sensitivityvalues;
249     end
250 
251 end
252 
253 %Concatenate sensitivity values
254 if numfilesets > 1
255     for i=1:(numfilesets-1)
256         sensitivityvalues=vertcat(values{i},values{i+1});
257     end
258 end
259 
260 
261 %starts the sensitivity fit window
262 [fit_data,line_index]=qtxrf_fit_sen(sensitivityvalues);
263 
264 %if the output of filename is no cell
265 if ~iscell(fit_data)
266    fit_data={fit_data};
267 end
268 
269 
270 %ask for file path and name input for relativities file
271 [filename3,path3]=uiputfile( ...
272 {  '*.txt','txt-files (*.txt)'; ...
273    '*.*',  'All Files (*.*)'}, ...
274    'Please choose filename for sensitivity file:');
275 
276 %open relativities textfile
277 fileID1 = fopen(fullfile(path3,filename3),'w');
278 
279 
280 %write to file
281 fprintf(fileID1,'Relative Sensitivities\n');
282 fprintf(fileID1,'----------------------------\n');
283 fprintf(fileID1,'%s\n', datestr(now));
284 fprintf(fileID1,'----------------------------\n');
285 
286 
287 fprintf(fileID1,'Internal\n');
288 fprintf(fileID1,'%f\n' ,internal);
289 fprintf(fileID1,'Internal\n');
290 fprintf(fileID1,'----------------------------\n');
291 
292 %write coefficients of each linetype
293 index_type = find(line_index==1);
294 
295 for i=1:size(index_type,2)
296     fprintf(fileID1,['Line' num2str(index_type(i)) '\n']);
297     fprintf(fileID1,'%0.12f\n' ,fit_data{index_type(i)});
298     fprintf(fileID1,['Line' num2str(index_type(i)) '\n']);
```

```
299 end
300
301 fprintf(fileID1,'------------------------------\n');
302 fprintf(fileID1,'Used calibration files:\n');
303
304 %write filepath's of used calibration files
305 for i=1:index;
306 fprintf(fileID1,'%s\n',fullfile(path,filename{i}));
307 end
308
309 fprintf(fileID1,'------------------------------\n');
310
311 fclose(fileID1);
312
313 guidata(hObject, handles);
314
315
316 % --- Executes on button press in save_results.
317 function save_results_Callback(hObject, eventdata, handles)
318
319 %ask for file path and name input for result file
320 [filename4,path4]=uiputfile( ...
321 {  '*.txt','txt-files (*.txt)'; ...
322    '*.*',  'All Files (*.*)'}, ...
323    'Please choose filename for results file:');
324
325 %open result file
326 fileID4 = fopen(fullfile(path4,filename4),'w');
327
328 %write to file
329 fprintf(fileID4,'Results: \r\n');
330 fprintf(fileID4,'------------------------------\r\n');
331 fprintf(fileID4,'%s\r\n', datestr(now));
332 fprintf(fileID4,'------------------------------\r\n');
333
334 %get data from table in qtxrf window
335 results=get(handles.results,'Data');
336
337 %ask for line size
338 sizeresults=size(results);
339
340 %write concentrations into results file
341 for i=1:sizeresults(1,1)
342         fprintf(fileID4,'%3s%4i%10.3f    ś %.3f%s \r\n',results{i,:});
343 end
344
345 %write used file path's for calculation into result file
346 fprintf(fileID4,'------------------------------\r\n');
347 fprintf(fileID4,'File with relative sensitivities:\r\n');
348 fprintf(fileID4,'%s\r\n',fullfile(handles.path2,handles.filename2));
349
350 fprintf(fileID4,'------------------------------\r\n');
351 fprintf(fileID4,'File for quantification:\r\n');
352 fprintf(fileID4,'%s\r\n',fullfile(handles.path1,handles.filename1));
353 fprintf(fileID4,'------------------------------\r\n');
```

```
354
355 %close file
356 fclose(fileID4);
```

B.7 qtxrf_fit_el.m

```
1  function varargout = qtxrf_fit_el(varargin)
2
3  % Begin initialization code - DO NOT EDIT
4  gui_Singleton = 1;
5  gui_State = struct('gui_Name',       mfilename, ...
6                     'gui_Singleton',  gui_Singleton, ...
7                     'gui_OpeningFcn', @qtxrf_fit_el_OpeningFcn, ...
8                     'gui_OutputFcn',  @qtxrf_fit_el_OutputFcn, ...
9                     'gui_LayoutFcn',  [] , ...
10                    'gui_Callback',   []);
11 if nargin && ischar(varargin{1})
12     gui_State.gui_Callback = str2func(varargin{1});
13 end
14
15 if nargout
16     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
17 else
18     gui_mainfcn(gui_State, varargin{:});
19 end
20 % End initialization code - DO NOT EDIT
21
22
23 % --- Executes just before qtxrf_fit_el is made visible.
24 function qtxrf_fit_el_OpeningFcn(hObject, eventdata, handles, varargin)
25
26 %matrixcal is the matrix with the intensities for each line and
27 %concentration
28 matrixcal=varargin{1,1};
29
30 %internal standard input
31 internal=varargin{1,2};
32
33
34 %internal concentration input
35 internalconcentration=varargin{1,3};
36
37 number_concentration=size(internalconcentration,1);
38
39 %through origin or not
40 origin=varargin{1,4};
41
42 %size of the matrix
43 sizematrix=size(matrixcal);
44
45 %define axes
46 axes(handles.axes1);
47
48 %element vector for figure name
49 elements={'H';'He';'Li';'Be';'B';'C';'N';'O';'F';'Ne';'Na';'Mg';...
50     'Al';'Si';'P';'S';'Cl';'Ar';'K';'Ca';'Sc';'Ti';'V';'Cr';'Mn';...
```

```
51      'Fe';'Co';'Ni';'Cu';'Zn';'Ga';'Ge';'As';'Se';'Br';'Kr';'Rb';'Sr';...
52      'Y';'Zr';'Nb';'Mo';'Tc';'Ru';'Rh';'Pd';'Ag';'Cd';'In';'Sn';'Sb';...
53      'Te';'I';'Xe';'Cs';'Ba';'La';'Ce';'Pr';'Nd';'Pm';'Sm';'Eu';'Gd';...
54      'Tb';'Dy';'Ho';'Er';'Tm';'Yb';'Lu';'Hf';'Ta';'W';'Re';'OS';'Ir';...
55      'Pt';'Au';'Hg';'Tl';'Pb';'Bi';'Po';'At';'Rn';'Fr';'Ra';'Ac';'Th';...
56      'Pa';'U'};
57
58  %find array index for internal standard
59  for m=1:sizematrix(1)
60      if (matrixcal(m,1)==internal)
61          std=m;
62      end
63  end
64  disp(matrixcal)
65  %generates a global variable slope where the first column is the element,
66  %the second column is the linetype and the third column is the slope
67  for m=1:sizematrix(1)
68
69      handles.slope(m,1)=matrixcal(m,1); %take elements
70      handles.slope(m,2)=matrixcal(m,2); %take linetype
71
72      if m==std
73          handles.slope(m,3)=1; %for internal std slope is 1
74          handles.slope(m,4)=0; %Error 0
75      else
76
77      for p=1:number_concentration
78              y(p)=matrixcal(m,1+3*p)/(matrixcal(std,1+3*p)/...
79                  internalconcentration(p)); %generates the y value
80
81
82              x(p)=matrixcal(m,3*p); %generates the x value
83
84              std_dev(p)=sqrt((matrixcal(std,2+3*p)/...
85                  matrixcal(std,1+3*p))^2+(matrixcal(m,2+3*p)/...
86                  matrixcal(m,1+3*p))^2)*y(p);
87      end
88
89      %set axes limit
90      ymin=0;
91      ymax=max(y)*1.1;
92      xmin=0;
93      xmax=max(x)*1.1;
94
95      %through origin or not
96      %fit through origin
97
98      Weights=1./(std_dev).^2;
99
100     xvect=[0 xmax];
101
102
103     if origin
104         ymat(:,1)=y';
105         conmat(:,1)=x';
```

```
106         ft1 = fittype('a*x');
107         [yfit1,~,output]=fit(conmat,ymat,ft1,'Weights',Weights);
108         para=coeffvalues(yfit1);
109         plot(xvect,para(1)*xvect,'b-',conmat, ymat,'r*');
110         %set caption for axes
111         legend({'fit','data'});
112         xlabel('Conc. element');
113         ylabel('Int. element / Int. internal standard per Conc.');
114         legend('Location','best')
115         hold on
116         errorbar(conmat, ymat,std_dev,'r*');
117         hold off
118         ylim(handles.axes1,[ymin ymax])
119         xlim(handles.axes1,[xmin xmax])
120         J=output.Jacobian;
121         cov=inv(J'*J);
122         std_dev_para = sqrt(diag(cov));
123     else
124         ymat(:,1)=y';
125         conmat(:,1)=x';
126         ft = fittype('a*x+b');
127         [yfit,~,output]=fit(conmat,ymat,ft,'Weights',Weights);
128         para=coeffvalues(yfit);
129         plot(xvect,xvect*para(1)+para(2),'b-');
130         %set caption for axes
131         legend({'fit','data'});
132         xlabel('Conc. element / Conc. internal standard');
133         ylabel('Int. element / Int. internal standard');
134         legend('Location','best')
135         hold on
136         errorbar(conmat, ymat,std_dev,'r*');
137         hold off
138         ylim(handles.axes1,[ymin ymax])
139         xlim(handles.axes1,[xmin xmax])
140         J=output.Jacobian;
141         cov=inv(J'*J);
142         std_dev_para = sqrt(diag(cov));
143     end
144
145
146     handles.slope(m,3)=para(1);
147     handles.slope(m,4)=std_dev_para(1);
148     set(handles.element_name,'string',elements(matrixcal(m,1)));
149
150     %wait until user click submit
151     uiwait(handles.fit_calibration);
152
153     end
154 end
155
156
157 % Update handles structure
158 guidata(hObject, handles);
159
160
```

```
161
162 % --- Outputs from this function are returned to the command line.
163 function varargout = qtxrf_fit_el_OutputFcn(hObject, eventdata, handles)
164
165 varargout{1} = handles.slope;
166
167 close(handles.fit_calibration);
168
169 % --- Executes on button press in confirm_fit.
170 function confirm_fit_Callback(hObject, eventdata, handles)
171 % hObject    handle to confirm_fit (see GCBO)
172 % eventdata  reserved - to be defined in a future version of MATLAB
173 % handles    structure with handles and user data (see GUIDATA)
174
175 uiresume(handles.fit_calibration)
```

B.8 qtxrf_fit_sen.m

```
1  function varargout = qtxrf_fit_sen(varargin)
2
3  % Begin initialization code - DO NOT EDIT
4  gui_Singleton = 1;
5  gui_State = struct('gui_Name',       mfilename, ...
6                     'gui_Singleton',  gui_Singleton, ...
7                     'gui_OpeningFcn', @qtxrf_fit_sen_OpeningFcn, ...
8                     'gui_OutputFcn',  @qtxrf_fit_sen_OutputFcn, ...
9                     'gui_LayoutFcn',  [] , ...
10                    'gui_Callback',   []);
11 if nargin && ischar(varargin{1})
12     gui_State.gui_Callback = str2func(varargin{1});
13 end
14
15 if nargout
16     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
17 else
18     gui_mainfcn(gui_State, varargin{:});
19 end
20 % End initialization code - DO NOT EDIT
21
22
23 % --- Executes just before qtxrf_fit_sen is made visible.
24 function qtxrf_fit_sen_OpeningFcn(hObject, eventdata, handles, varargin)
25
26 %generate handles for the slopes of the fit
27 handles.xy=varargin{1};
28
29 %set default value for the dropdown menues
30 set(handles.fitgrade,'String',{'Degree 1','Degree 2','Degree 3',...
31     'Degree 4','Degree 5','Degree 6','Degree 7'});
32 set(handles.line,'String',{'K-lines','L-lines','M-lines'});
33
34 %plot the measured slopes into the right graph
35 axes(handles.axes1);
36 errorbar(handles.xy(:,1),handles.xy(:,3),handles.xy(:,4),'r*');
37 title('Measured relative sensitivity');
38 xlabel('Z');
```

```
39 ylabel('Relative Intensity [a.u.]');
40 legend('Location','best')
41
42 %plot the default linetype slopes into the left graph
43 axes(handles.axes2);
44 k= find(handles.xy(:,2)==1);
45 errorbar(handles.xy(k,1),handles.xy(k,3),handles.xy(k,4),'r*');
46 title('Measured relative sensitivity for selected line');
47 xlabel('Z');
48 ylabel('Relative Intensity [a.u.]');
49 legend('Location','best')
50
51 %set gobal variable for right plot. It is set two 1 when coefficients
52 %exists for line 1,2 or 3.
53 handles.coeff_in(1:3)=0;
54
55
56 % Choose default command line output for qtxrf_fit_sen
57 handles.output = hObject;
58
59 % Update handles structure
60 guidata(hObject, handles);
61
62 % waits until user press save button to submit the fits
63 uiwait(handles.qtxrf_fit_sen);
64
65
66 % --- Outputs from this function are returned to the command line.
67 function varargout = qtxrf_fit_sen_OutputFcn(hObject, eventdata, handles)
68
69 % Set output
70 varargout{1} = handles.coeff;
71 varargout{2} = handles.coeff_in;
72
73 %varargout{1} =handles.sensfit
74 close(handles.qtxrf_fit_sen)
75
76
77 % --- Executes on button press in save.
78 function save_Callback(hObject, eventdata, handles)
79
80 uiresume(handles.qtxrf_fit_sen);
81
82 % Update handles structure
83 guidata(hObject, handles);
84
85 % --- Executes on button press in fit.
86 function fit_Callback(hObject, eventdata, handles)
87
88 %reset error field after each fit
89 set(handles.errorfield,'String','no Error');
90
91 %get selected fitoption
92 selectedfit=get(handles.fitgrade,'Value');
93
```

```
94 %get selected linetype
95 selectedline=get(handles.line,'Value');
96
97 xy=handles.xy;
98
99 axes(handles.axes2);
100
101 %this switch askes wich line and plot has been selected
102 switch selectedline
103     case 1                          %line k has been selected
104
105         k= find(xy(:,2)==1);        %find rows with linetype 1
106         sizek=size(k);              %how much linetypes 1
107
108             if sizek(1,1)>2         %no fit for less than two rows
109                 switch selectedfit
110                     case 1
111                         sensfit=fit(xy(k,1),xy(k,3),'poly1');
112                     case 2
113                         sensfit=fit(xy(k,1),xy(k,3),'poly2');
114                     case 3
115                         if sizek(1,1)>3
116                             sensfit=fit(xy(k,1),xy(k,3),'poly3');
117                         else
118                             set(handles.errorfield,'String',...
119                                 'Error: This fit option needs more values
                                        !!!');
120                         end
121                     case 4
122                         if sizek(1,1)>4
123                             sensfit=fit(xy(k,1),xy(k,3),'poly4');
124                         else
125                             set(handles.errorfield,'String',...
126                                 'Error: This fit option needs more values
                                        !!!');
127                         end
128                     case 5
129                         if sizek(1,1)>5
130                             sensfit=fit(xy(k,1),xy(k,3),'poly5');
131                         else
132                             set(handles.errorfield,'String',...
133                                 'Error: This fit option needs more values
                                        !!!');
134                         end
135                     case 6
136                         if sizek(1,1)>6
137                             sensfit=fit(xy(k,1),xy(k,3),'poly6');
138                         else
139                             set(handles.errorfield,'String',...
140                                 'Error: This fit option needs more values
                                        !!!');
141                         end
142                     case 7
143                         if sizek(1,1)>7
144                             sensfit=fit(xy(k,1),xy(k,3),'poly7');
```

```
145                         else
146                             set(handles.errorfield,'String',...
147                                 'Error: This fit option needs more values
                                        !!!');
148                         end
149                 end
150
151                 handles.coeff{1}=coeffvalues(sensfit);
152                 handles.coeff_in(1)=1;
153
154             else
155                 set(handles.errorfield,'String',...
156                     'Error: 3 values are minimum to fit K-lines!!!');
157             end
158
159     case 2          %line L has been selected
160
161         k= find(xy(:,2)==2);
162         sizek=size(k);
163
164             if sizek(1,1)>2
165                 switch selectedfit
166                     case 1
167                         sensfit=fit(xy(k,1),xy(k,3),'poly1');
168                     case 2
169                         sensfit=fit(xy(k,1),xy(k,3),'poly2');
170                     case 3
171                         if sizek(1,1)>3
172                             sensfit=fit(xy(k,1),xy(k,3),'poly3');
173                         else
174                             set(handles.errorfield,'String',...
175                                 'Error: This fit option needs more values
                                        !!!');
176                         end
177                     case 4
178                         if sizek(1,1)>4
179                             sensfit=fit(xy(k,1),xy(k,3),'poly4');
180                         else
181                             set(handles.errorfield,'String',...
182                                 'Error: This fit option needs more values
                                        !!!');
183                         end
184                     case 5
185                         if sizek(1,1)>5
186                             sensfit=fit(xy(k,1),xy(k,3),'poly5');
187                         else
188                             set(handles.errorfield,'String',...
189                                 'Error: This fit option needs more values
                                        !!!');
190                         end
191                     case 6
192                         if sizek(1,1)>6
193                             sensfit=fit(xy(k,1),xy(k,3),'poly6');
194                         else
195                             set(handles.errorfield,'String',...
```

```
196                                 'Error: This fit option needs more values
                                        !!!');
197                             end
198                         case 7
199                             if sizek(1,1)>7
200                                 sensfit=fit(xy(k,1),xy(k,3),'poly7');
201                             else
202                                 set(handles.errorfield,'String',...
203                                     'Error: This fit option needs more values
                                        !!!');
204                             end
205                     end
206
207
208                     %ylim(handles.axes1,[ymin ymax])
209                     %xlim(handles.axes1,[ymin ymax])
210                     %title('Relative sensitivity for L-lines');
211                     handles.coeff{2}=coeffvalues(sensfit);
212                     handles.coeff_in(2)=1;
213
214                 else
215                     set(handles.errorfield,'String',...
216                         'Error: 3 values are minimum to fit L-lines!!!');
217                 end
218
219         case 3      %line M has been selected
220
221             k= find(xy(:,2)==3);
222             sizek=size(k);
223
224                 if sizek(1,1)>2
225                     switch selectedfit
226                         case 1
227                             sensfit=fit(xy(k,1),xy(k,3),'poly1');
228                         case 2
229                             sensfit=fit(xy(k,1),xy(k,3),'poly2');
230                         case 3
231                             if sizek(1,1)>3
232                                 sensfit=fit(xy(k,1),xy(k,3),'poly3');
233                             else
234                                 set(handles.errorfield,'String',...
235                                     'Error: This fit option needs more values
                                        !!!');
236                             end
237                         case 4
238                             if sizek(1,1)>4
239                                 sensfit=fit(xy(k,1),xy(k,3),'poly4');
240                             else
241                                 set(handles.errorfield,'String',...
242                                     'Error: This fit option needs more values
                                        !!!');
243                             end
244                         case 5
245                             if sizek(1,1)>5
246                                 sensfit=fit(xy(k,1),xy(k,3),'poly5');
```

```
247                             else
248                                 set(handles.errorfield,'String',...
249                                     'Error: This fit option needs more values
                                        !!!');
250                             end
251                         case 6
252                             if sizek(1,1)>6
253                                 sensfit=fit(xy(k,1),xy(k,3),'poly6');
254                             else
255                                 set(handles.errorfield,'String',...
256                                     'Error: This fit option needs more values
                                        !!!');
257                             end
258                         case 7
259                             if sizek(1,1)>7
260                                 sensfit=fit(xy(k,1),xy(k,3),'poly7');
261                             else
262                                 set(handles.errorfield,'String',...
263                                     'Error: This fit option needs more values
                                        !!!');
264                             end
265                     end
266
267                     handles.coeff{3}=coeffvalues(sensfit);
268                     handles.coeff_in(3)=1;
269
270                 else
271                     set(handles.errorfield,'String',...
272                         'Error: 3 values are minimum to fit M-lines!!!');
273                 end
274
275 end
276
277
278 %plot result of the fit
279 plot(sensfit,'b-');
280 hold on
281 errorbar(xy(k,1),xy(k,3),xy(k,4),'r*');
282 hold off
283 xlabel('Z');
284 title('Fitted Relative sensitivity of selected line');
285 ylabel('Relative Intensity [a.u.]');
286 legend('Location','best')
287
288
289 axes(handles.axes1);
290 cla reset %reset axes1
291 hold on   %keep the plots till reset
292
293 errorbar(xy(:,1),xy(:,3),xy(:,4),'r*'); %plot all lines
294
295 %plot fit for k lines if handles.coeff==1
296 if handles.coeff_in(1)
297     k1=find(xy(:,2)==1);
298     x1=linspace(min(xy(k1,1)),max(xy(k1,1)),100);
```

```
299     y1=polyval(handles.coeff{1}, x1);
300     plot(x1,y1,'b-');
301 end
302
303 %plot fit for l lines if handles.coeff==2
304 if handles.coeff_in(2)
305     k2=find(xy(:,2)==2);
306     x2=linspace(min(xy(k2,1)),max(xy(k2,1)),100);
307     y2=polyval(handles.coeff{2}, x2);
308     plot(x2,y2,'b-');
309 end
310
311 %plot fit for m lines if handles.coeff==3
312 if handles.coeff_in(3)
313     k3=find(xy(:,2)==3);
314     x3=linspace(min(xy(k3,1)),max(xy(k3,1)),100);
315     y3=polyval(handles.coeff{3}, x3);
316     plot(x3,y3,'b-');
317 end
318
319
320 title('Measured relative sensitivity');
321 xlabel('Z');
322 ylabel('Relative Intensity [a.u.]');
323 %legend('Location','best')
324
325 % Update handles structure
326 guidata(hObject, handles);
327
328
329
330 % --- Executes on selection change in line.
331 function line_Callback(hObject, eventdata, handles)
332
333 %plot the selected linetype slopes in the left graph
334 xy=handles.xy;
335 selectedline=get(handles.line,'Value');
336 k= find(xy(:,2)==selectedline);
337 axes(handles.axes2);
338 errorbar(xy(k,1),xy(k,3),xy(k,4),'r*');
339 title('Measured Relative sensitivity of selected line');
340 xlabel('Z');
341 ylabel('Relative Intensity [a.u.]');
342 legend('Location','best')
```

B.9 qtxrf_numfilesets.m

```
1 function varargout = qtxrf_numfilesets(varargin)
2
3 % Begin initialization code - DO NOT EDIT
4 gui_Singleton = 1;
5 gui_State = struct('gui_Name',       mfilename, ...
6                    'gui_Singleton',  gui_Singleton, ...
7                    'gui_OpeningFcn', @qtxrf_numfilesets_OpeningFcn, ...
8                    'gui_OutputFcn',  @qtxrf_numfilesets_OutputFcn, ...
9                    'gui_LayoutFcn',  [] , ...
```

```
                   'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before qtxrf_numfilesets is made visible.
function qtxrf_numfilesets_OpeningFcn(hObject, eventdata,handles,varargin)

% Choose default command line output for qtxrf_numfilesets
handles.output = hObject;



uiwait(handles.numfilesets);

% Update handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = qtxrf_numfilesets_OutputFcn(hObject,eventdata,handles)

guidata(hObject);

filesets=str2double(get(handles.txt_num,'String'));

% Get default command line output from handles structure
varargout{1} = filesets;

delete(handles.numfilesets);


% --- Executes on button press in submit.
function submit_Callback(hObject, eventdata, handles)
% hObject    handle to submit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

uiresume(handles.numfilesets);
```

B.10 qtxrf_read_filecon_neu.m

```
function varargout = qtxrf_read_filecon_neu(varargin)

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @qtxrf_read_filecon_neu_OpeningFcn,
```

```
                   ...
                   'gui_OutputFcn',  @qtxrf_read_filecon_neu_OutputFcn, ...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before qtxrf_read_filecon_neu is made visible.
function qtxrf_read_filecon_neu_OpeningFcn(hObject,eventdata,handles,
    varargin)

%generate elements vector for pop up internal standard
elements={'H';'He';'Li';'Be';'B';'C';'N';'O';'F';'Ne';'Na';'Mg';'Al';...
    'Si';'P';'S';'Cl';'Ar';'K';'Ca';'Sc';'Ti';'V';'Cr';'Mn';'Fe';'Co';...
    'Ni';'Cu';'Zn';'Ga';'Ge';'As';'Se';'Br';'Kr';'Rb';'Sr';'Y';'Zr';...
    'Nb';'Mo';'Tc';'Ru';'Rh';'Pd';'Ag';'Cd';'In';'Sn';'Sb';'Te';'I';...
    'Xe';'Cs';'Ba';'La';'Ce';'Pr';'Nd';'Pm';'Sm';'Eu';'Gd';'Tb';'Dy';...
    'Ho';'Er';'Tm';'Yb';'Lu';'Hf';'Ta';'W';'Re';'OS';'Ir';'Pt';'Au';...
    'Hg';'Tl';'Pb';'Bi';'Po';'At';'Rn';'Fr';'Ra';'Ac';'Th';'Pa';'U'};

%set pop up with elements
set(handles.internal_std,'string',elements);

%set pop up with default value Ga
set(handles.internal_std,'value',31);

%set handle to std value Ga
handles.internal=31;

%input filenames
handles.filename=varargin{1,1};

%number_concentration is the number of files
number_concentration=varargin{1,2};


%generate a table with filename, concentration of the file, concentration
%of the internal std and ppm
for i=1:number_concentration

files{i,1}=char(handles.filename(i));
files{i,2}=double(0);
files{i,3}=double(1);
files{i,4}='ppm';
end

```

```
61
62 %fill table with data
63 set(handles.files_cons, 'Data', files);
64
65 %only column 2 and 3 are editable
66 set(handles.files_cons,'ColumnEditable',logical([0 1 1 0]))
67
68
69 % Update handles structure
70 guidata(hObject, handles);
71
72 %wait thill user filled the concentrations and internal std
73 uiwait(handles.read_fileconcentration)
74
75
76 % --- Outputs from this function are returned to the command line.
77 function varargout=qtxrf_read_filecon_neu_OutputFcn(hObject,eventdata,
       handles)
78
79 guidata(handles.read_fileconcentration)
80
81 %get data from changed uitable
82 tablematrix=get(handles.files_cons,'Data');
83
84 %output
85 varargout{1} = cell2mat(tablematrix(:,2));
86 varargout{2} = get(handles.internal_std,'Value');
87 varargout{3} = cell2mat(tablematrix(:,3));
88 varargout{4} = get(handles.origin,'Value');
89
90 %after output delete handles
91 delete(handles.read_fileconcentration);
92
93 % --- Executes when user attempts to close read_fileconcentration.
94 function read_fileconcentration_CloseRequestFcn(hObject, eventdata, handles
       )
95
96 % Hint: delete(hObject) closes the figure
97 delete(hObject);
98
99
100 % --- Executes on button press in submit.
101 function submit_Callback(hObject, eventdata, handles)
102
103 uiresume(handles.read_fileconcentration)
```

B.11 qtxrf_unwanted_con.m

```
1 function varargout = qtxrf_unwanted_con(varargin)
2
3 % Begin initialization code - DO NOT EDIT
4 gui_Singleton = 1;
5 gui_State = struct('gui_Name',       mfilename, ...
6                    'gui_Singleton',  gui_Singleton, ...
7                    'gui_OpeningFcn', @qtxrf_unwanted_con_OpeningFcn, ...
8                    'gui_OutputFcn',  @qtxrf_unwanted_con_OutputFcn, ...
```

```
                   'gui_LayoutFcn',  []  , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before qtxrf_unwanted_con is made visible.
function qtxrf_unwanted_con_OpeningFcn(hObject,eventdata,handles,varargin)

%array with filenames, concentrations
calArray=varargin{1,1};

%number of files
number_concentration=varargin{1,2};

%vector with the concentration
concentration=varargin{1,3};

%order the concentration (less concentration first)
[concentrationorder,indexorder] = sort(concentration);

%element vector
elements={'H';'He';'Li';'Be';'B';'C';'N';'O';'F';'Ne';'Na';'Mg';...
    'Al';'Si';'P';'S';'Cl';'Ar';'K';'Ca';'Sc';'Ti';'V';'Cr';'Mn';...
    'Fe';'Co';'Ni';'Cu';'Zn';'Ga';'Ge';'As';'Se';'Br';'Kr';'Rb';'Sr';...
    'Y';'Zr';'Nb';'Mo';'Tc';'Ru';'Rh';'Pd';'Ag';'Cd';'In';'Sn';'Sb';...
    'Te';'I';'Xe';'Cs';'Ba';'La';'Ce';'Pr';'Nd';'Pm';'Sm';'Eu';'Gd';...
    'Tb';'Dy';'Ho';'Er';'Tm';'Yb';'Lu';'Hf';'Ta';'W';'Re';'OS';'Ir';...
    'Pt';'Au';'Hg';'Tl';'Pb';'Bi';'Po';'At';'Rn';'Fr';'Ra';'Ac';'Th';...
    'Pa';'U'};

%line vector
line={'K';'L';'M';'N';'O';'P'};

%generate table: first column YES/NO, second column element number, third
%column linetype, from fourth column upwards intensities in the order of
%the concentration

%generate column names
cnames{1}='<html><FONT SIZE="5">Selected elements</FONT></html>';
cnames{2}='<html><FONT SIZE="5">Elements</FONT></html>';
cnames{3}='<html><FONT SIZE="5">Lines</FONT></html>';

%fill matrixcal with elements ans linetypes
matrixcal(:,1)=calArray{1,1}{1,1};
matrixcal(:,2)=calArray{1,1}{1,2};

```

```
number_of_lines=size(matrixcal(:,1));
edit=[true false false];

for k=1:number_concentration
    edit=horzcat(edit,[true true]);
    concentrationvector(1:number_of_lines)=concentrationorder(k);
    cnames{end+1}=['<html><FONT SIZE="5">Con. (' ...
        num2str(concentrationorder(k)) 'ppm) </FONT></html>'];
    cnames{end+1}=['<html><FONT SIZE="5">Intensity (' ...
        num2str(concentrationorder(k)) 'ppm) </FONT></html>'];
    cnames{end+1}=['<html><FONT SIZE="5">Std dev. (' ...
        num2str(concentrationorder(k)) 'ppm) </FONT></html>'];
    %fill column for each concentration
    matrixcal(:,end+1)=concentrationvector;
    matrixcal(:,end+1)=calArray{1,indexorder(k)}{1,4};
    matrixcal(:,end+1)=calArray{1,indexorder(k)}{1,5};
end

numbermatrix=size(matrixcal);

%change the table into cell inclucing the true/false option in first column
for i=1:numbermatrix(1,1)
    array{i,1}=true;
    for l=1:numbermatrix(1,2)
        array{i,l+1}=matrixcal(i,l);
    end
end

%fill table with intensities
set(handles.select_unwanted, 'Data', array,'ColumnName',cnames,...
    'ColumnEditable', edit);

%generate global variable index to send the order of the concentration to
%the output
handles.index=indexorder;


% Update handles structure
guidata(hObject, handles);

%wait until user unticks the unwanted elements
uiwait(handles.qtxrf_unwanted)


% --- Outputs from this function are returned to the command line.
function varargout = qtxrf_unwanted_con_OutputFcn(hObject,eventdata,handles
    )

%get all the elements
allelements=get(handles.select_unwanted,'Data');

%number of all lements
number=size(allelements);

%write a new table with selected elements
```

```
q=1;
for i=1:number(1,1)
    if cell2mat(allelements(i,1))==1
    selectedelements(q,:)=allelements(i,2:number(1,2));
    q=q+1;
    end
end


%output for selected lines
varargout{1} = cell2mat(selectedelements);

%order index for concentrations
varargout{2} = handles.index;

%after output delete handles
delete(handles.qtxrf_unwanted);


% --- Executes on button press in submit.
function submit_Callback(hObject, eventdata, handles)

uiresume(handles.qtxrf_unwanted)
```

C Subroutines

C.1 addLinLog.m

```
function addLinLog(tab, axes, tag)
    bg = uibuttongroup(tab,'Visible','off',...
                       'Position',[0.95 0.9 1 1],...
                       'SelectionChangedFcn',@linlogSelection,...
                        'UserData', axes, 'Tag',tag);

    % Create three radio buttons in the button group.
    r1 = uicontrol(bg,'Style',...
                       'radiobutton',...
                       'String','lin',...
                       'Position',[0 0 50 30],...
                       'Fontsize',14,...
                       'HandleVisibility','off');

    r2 = uicontrol(bg,'Style','radiobutton',...
                       'String','log',...
                       'Position',[0 30 50 30],...
                       'Fontsize',14,...
                       'HandleVisibility','off');
     bg.Visible = 'on';
```

C.2 auto_background.m

```
function [datacorr,newchannel,back] = auto_background(data,path,file)

% Initialisation
back=[];

```

```
% Read backgroundfile
[backg_settings] = read_backgroundfile(path,file);

% Parameters
size_data=size(backg_settings,1);
fromchannel=backg_settings(size_data-1);
tochannel=backg_settings(size_data);

% Generate new channel vector
newchannel=(fromchannel:tochannel);

% Data for background estimation
datacal=data(fromchannel:tochannel);


% New channel variable
sizenew=size(datacal,2);
channelcal=(1:sizenew);

%Golay filter for background method 1-3
databcal = sgolayfilt(datacal,2,11);
databcal(databcal<0)=0;

% Selected method
method=backg_settings(1,1);

switch  method
    case 1
    % Do running ball
    radius=backg_settings(2,1);
    back = background_running_ball(channelcal,databcal,radius);

    case 2
    % Do peakstripping
    iterations=backg_settings(2,1);
    distance=backg_settings(3,1);
    back = background_peak_stripping(databcal,iterations,distance);

    case 3
    % Do msbackadj
    window=backg_settings(2,1);
    step=backg_settings(3,1);
    YB = msbackadj(channelcal(:),databcal(:),'WindowSize',window,'StepSize'
        ,step);
    back=databcal-YB';
    back(back<0) = 0;

    case 4
    % Do nothing
    back(1:size(channelcal,2)) = 0;

    case 5
    % Do orthogonal poly
    degree=backg_settings(2,1);
    rfactor=backg_settings(3,1);
```

```
60      back = background_ortho(channelcal,datacal,degree,rfactor);
61  end
62
63
64
65  % Subtract background and remove negative intensity
66  datacorr=datacal-back;
67  datacorr(datacorr<0)=0;
```

C.3 background_ortho.m

```
1   function [zfinal]=background_ortho(x,y,degree,r)
2
3   % No zero elements allowed
4   y=y+1;
5
6   % Initialisation
7   cold(1:degree)=0;
8   mu=0;
9   m=0;
10  N=size(y,2);
11  zfinal(1:N)=0;
12
13  w=1./y;
14
15  poly(1:N)=1;
16
17  % Start polynomial
18  polynom(1,:)=poly;
19
20  % Creating polynomials
21  while 1
22  mu=mu+1;
23  gamma(1)=sum(w.*(polynom(1,:).^2));
24  alpha(1)=sum(w.*x.*(polynom(1,:).^2)/gamma(1));
25  beta(1)=0;
26  c(1)=sum((w.*y.*polynom(1,:))/gamma(1));
27  polynom(2,:)=(x-alpha(1)).*polynom(1,:);
28  sigmac(1)=sqrt(1/gamma(1));
29
30      for j=1:degree-1
31              gamma(j+1)=sum(w.*(polynom(j+1,:).^2));
32              alpha(j+1)=sum(w.*x.*(polynom(j+1,:).^2)/gamma(j+1));
33              beta(j+1)=sum(w.*x.*polynom(j+1,:).*polynom(j,:)/gamma(j));
34              c(j+1)=sum(w.*y.*polynom(j+1,:)/gamma(j+1));
35              polynom(j+2,:)=(x-alpha(j+1)).*polynom(j+1,:)- beta(j+1).*
                    polynom(j,:);
36              sigmac(j+1)=sqrt(1/gamma(j+1));
37      end
38
39      zfinal(:,:)=0;
40      for j=1:degree
41          z=c(j)*polynom(j,:);
42          zfinal=zfinal+z;
43      end
44      E=sum(w.*(y-zfinal).^2);
```

```
45      f=N-degree-m;
46      s=sqrt(E/f);
47      m=0;
48      sigmacend=s.*sigmac;
49      if all(cold-c<sigmacend)
50          break
51      else
52          for i=1:N
53              if y(i)>=(zfinal(i)+r*sqrt(zfinal(i)))
54                  %w(i)=1./((y(i)-zfinal(i)).^2);
55                  w(i)=0;
56              else
57                  w(i)=1./((zfinal(i)).^2);
58                  %w(i)=1./(zfinal(i));
59                  m=m+1;
60              end
61          end
62      end
63      cold=c(:)';
64
65      if mu==1000
66          break
67      end
68  end
```

C.4 background_peak_stripping.m

```
1   function [baseline] =background_peak_stripping(y,number,distance)
2
3   n=size(y,2);
4
5   % Calculate interations
6   iterations=n-1-distance;
7
8   y=sqrt(y);
9   %y=log(y+1);
10
11  if distance==1
12      for j=1:number
13          for i=distance+1:iterations
14              m=(y(i-distance)+y(i+distance))*0.5;
15              if(m<y(i))
16                  y(i)=m;
17              end
18          end
19      end
20
21  else
22
23      for j=1:number-8
24          for i=distance+1:iterations
25              m=(y(i-distance)+y(i+distance))*0.5;
26              if(m<y(i))
27                  y(i)=m;
28              end
29          end
```

```
30      end
31
32      for k=1:8
33          distance=round(distance/sqrt(2));
34          iterations=n-1-distance;
35          for i=distance+1:iterations
36              m=(y(i-distance)+y(i+distance))*0.5;
37              if(m<y(i))
38                  y(i)=m;
39              end
40          end
41      end
42  end
43
44  baseline=y.^2;
45  %baseline=exp(y)-1;
```

C.5 background_running_ball.m

```
1   function [baseline] =background_running_ball(x,y,radius)
2
3   n=size(y,2);
4   fac=max(y)/max(x);
5   y=y/fac;
6   dx=max(x)/n;
7   radius=floor(radius/dx);
8
9   ybegin(1:radius)=y(1);
10  yend(1:radius)=y(n);
11  ywork=horzcat(ybegin,y,yend)+radius;
12
13  z=2*radius;
14
15  yball=sqrt(radius^2-(-radius:radius).^2);
16
17  for i=1:n
18      baseline(i)=min(ywork(i:i+z)-yball);
19  end
20
21  baseline=baseline*fac;
```

C.6 data_for_selected_edge.m

```
1   function [strlines,energy,rel,num] = data_for_selected_edge(input,...
2       selected_str,edges)
3
4   % Find position of edge
5   pos_edge=strcmp(edges,selected_str);
6
7   % +1 because He is the first element in the list
8   index=find(pos_edge==1);
9
10  % Choose the right energy for selected line to generate formula
11  switch index
12      case 1
13          num=1;
```

```
14          strlines=input.currentKstring;
15          energy=input.currentKenergy;
16          rel=input.currentKrel;
17      case 2
18          num=0;
19          strlines=input.currentL1string;
20          energy=input.currentL1energy;
21          rel=input.currentL1rel;
22      case 3
23          num=0;
24          strlines=input.currentL2string;
25          energy=input.currentL2energy;
26          rel=input.currentL2rel;
27      case 4
28          num=2;
29          strlines=input.currentL3string;
30          energy=input.currentL3energy;
31          rel=input.currentL3rel;
32      case 5
33          num=1;
34          strlines=input.currentKastring;
35          energy=input.currentKaenergy;
36          rel=input.currentKarel;
37      case 6
38          num=0;
39          strlines=input.currentKbstring;
40          energy=input.currentKbenergy;
41          rel=input.currentKbrel;
42      case 7
43          num=3;
44          strlines=input.currentM5string;
45          energy=input.currentM5energy;
46          rel=input.currentM5rel;
47  end
```

C.7 fit_results.m

```
1  function [output] = fit_results(input)
2
3  % Arrange input
4  tic
5  energyin = input.energyin;
6  datain = input.datain;
7  cellenergy = input.cellenergy;
8  cellrel = input.cellrel;
9  cellstr = input.cellstr;
10 axes_result = input.axes_result;
11 axes_residual = input.axes_residual;
12 parameter = input.parameter;
13 %escape = input.escape;
14 %input.sum = 1;
15 background = input.background;
16 zero = input.zero;
17 gain = input.gain;
18 channelroi = input.channelroi;
19 line_asr = input.linenum;
```

```
20 robust = input.robust;
21 lines2fit = input.lines2fit;
22
23
24
25 % Retrieve paramters from table
26 pair_energy=str2double(parameter{5,2});
27 fano_factor=str2double(parameter{6,2});
28 noise=str2double(parameter{7,2});
29
30 kalpha_detector=str2double(parameter{15,2});
31 fluores_yield=str2double(parameter{16,2});
32 jump_ratio=str2double(parameter{17,2});
33
34 path=parameter{3,2};
35 file=parameter{4,2};
36
37 % Read mass attenuation file for escape peaks
38 [energy_mass,massat] = read_massatfile (path,file);
39
40 % Interpolate mass attenuation coefficient
41 massat_kalpha = interp1(energy_mass,massat,kalpha_detector);
42
43
44 % Initiate
45 stringfin='';
46 size_edges=size(cellenergy,2);
47 output.plotelextra=cell(1,size_edges);
48 startarea = zeros(1,size_edges);
49 parameters_a = cell(1,size_edges);
50
51
52 % Create fit model, for each edge and sum up to one equation
53 for i=1:size_edges
54
55     % Energy vector of current edge
56     energy=cellenergy{i};
57
58     % Relative intensity vector of current edge
59     rel=cellrel{i};
60
61     % Find position of the first energy value
62     a=energyin-energy(1);
63     [~,index]=min(abs(a));
64
65     % Calculate startvalue for peakarea
66     fwhm=sqrt(noise^2+2.35^2*fano_factor*pair_energy*energy*1000)/1000;
67     sigma=fwhm/(2*sqrt(2*log(2)));
68     startarea(i)=datain(index)*sqrt(pi)*sigma(1)/(gain*rel(1));
69
70     % Parameter a goes from a001,a002,a003,...,a009,a010,...
71     if i<=9
72         stringa1=sprintf('+a%i%i%i',0,0,i);
73     else
74         stringa1=sprintf('+a%i%i',0,i);
```

```
75      end
76
77      %c1=noise
78      %c2=fano_factor
79      %b1=zero
80      %b2=gain
81
82      % Create fit formula
83      stringa2=cell(1,size(energy,1));
84
85      for l=1:size(energy,1)
86          % Generating string for sigma (input for formular string)
87          sigmastring=...
88              sprintf(...
89              'sqrt(c1^2+2.35^2*c2*%f*%f*1000)/1000/(2*sqrt(2*log(2)))',...
90              pair_energy,energy(l));
91
92          % Generating string for line
93          stringa2{l}=...
94              strcat(sprintf(...
95              '+((b2/(%s*sqrt(2*pi)))*%f*exp(-0.5*(((b1+b2*x)-%f)/(',...
96              sigmastring,rel(l),energy(l)),sigmastring,'))^2))');
97
98          %%BEGIN ESCAPE
99          if input.escape==1
100             massat_impinging = interp1(energy_mass,massat,energy(l));
101             f_escape=0.5*fluores_yield*(1-1/jump_ratio)* ...
102                 (1-(massat_kalpha./massat_impinging).*...
103                 log(1+massat_impinging./massat_kalpha));
104             eta=f_escape/(1-f_escape);
105
106             if startarea(i)*eta>1
107                 % Generating string for escape sigma (input for formular
                        string)
108                 sigmaescstring=...
109                     sprintf(...
110                     'sqrt(c1^2+2.35^2*c2*%f*%f*1000)/1000/(2*sqrt(2*log(2))
                            )',...
111                     pair_energy,energy(l)-kalpha_detector);
112                 % Generating string for escape line
113                 stringa2{l}=...
114                     strcat(...
115                     stringa2{l},sprintf(...
116                     '+((b2/(%s*sqrt(2*pi)))*%f*%f*exp(-0.5*(((b1+b2*x)-%f)
                            /(',...
117                     sigmaescstring,eta,rel(l),energy(l)-kalpha_detector)
                            ,...
118                     sigmaescstring,'))^2))');
119             end
120         end
121         %%END ESCAPE
122     end
123
124     % Concatenate strings
125     stringa3=strcat(stringa1,'*(',stringa2{:},')');
```

```
126
127     % Save strings separate for separate plot in deconvolution tab
128     output.plotelextra{i}=stringa3;
129
130     % Save string to stringfin
131     stringfin=strcat(stringfin,stringa3);
132
133 end
134
135 % Prepare Data for fit
136 [xData, yData] = prepareCurveData(channelroi,datain);
137 ft = fittype(stringfin,'independent', 'x');
138
139 % Set fit options
140 opts = fitoptions( 'Method', 'NonlinearLeastSquares' );
141 opts.Display = 'notify';
142
143 opts.Robust=robust;
144 %'off' (default) | 'LAR' | 'Bisquare'
145 opts.Algorithm='Trust-Region';
146 %'Levenberg-Marquardt' (default) | 'Trust-Region'
147 opts.DiffMinChange=str2double(parameter{8,2});
148 opts.MaxFunEvals=str2double(parameter{9,2});
149 opts.MaxIter=str2double(parameter{10,2});
150 opts.TolFun=str2double(parameter{11,2});
151 opts.TolX=str2double(parameter{12,2});
152
153
154 % Set the start value and bounds for peakarea
155 for i=1:(size_edges)
156     opts.Lower(i) = startarea(i)*str2double(parameter{13,2});
157     opts.StartPoint(i)= startarea(i);
158     opts.Upper(i) = startarea(i)*str2double(parameter{14,2});
159 end
160
161 % Set the start value and bounds for zero
162 opts.Lower(end+1)=zero-abs(zero*(1-str2double(parameter{22,2})));
163 opts.StartPoint(end+1)=zero;
164 opts.Upper(end+1)=zero+abs(zero*(str2double(parameter{23,2})-1));
165
166 % Set the start value and bounds for gain
167 opts.Lower(end+1)=gain*str2double(parameter{24,2});
168 opts.StartPoint(end+1)=gain;
169 opts.Upper(end+1)=gain*str2double(parameter{25,2});
170
171 % Set the start value and bounds for noise
172 opts.Lower(end+1)=noise*str2double(parameter{18,2});
173 opts.StartPoint(end+1)=noise;
174 opts.Upper(end+1)=noise*str2double(parameter{19,2});
175
176 % Set the start value and bounds for fano factor
177 opts.Lower(end+1)=fano_factor*str2double(parameter{20,2});
178 opts.StartPoint(end+1)=fano_factor;
179 opts.Upper(end+1)=fano_factor*str2double(parameter{21,2});
180
```

```
% Start weights for the fit
opts.Weights=1./(datain+background+1);

toc

tic
% Fit procedure
[output.fitresult, output.gof, o] = fit( xData, yData, ft, opts );
toc
tic




% Get fitparameter
output.fitparameter=coeffvalues(output.fitresult);

% Save the a parameter in handles.aparameter
output.aparameter=output.fitparameter(1:size_edges);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%SUMPEAKS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% if(input.sum==1)
%
% %stringfin
% stringsum='+c1*(';
%
% % for i=1:size_edges
% %
% %      % Energy vector of current edge
% %      energy=cellenergy{i};
%
% for j=1:size_edges
%      for k=1:size_edges
%
%      stringsum = strcat(stringsum,sprintf('(%f*%f)' ,...
%          output.aparameter(j),output.aparameter(j)));
%      end
% end
%
% stringsum=strcat(stringsum,')');
%
%
% % if input.sum == 1
% %
% %      opts.Lower(end+1)= 0;
% %      opts.StartPoint(end+1)=0.01;
% %      opts.Upper(end+1)= 1;
% %
% % end
% stringsum
%
% end

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%SUMPEAKS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Evaluate fitted function
output.fitted_spec=feval(output.fitresult,channelroi);


% Calculate the std deviation of parameters
J=o.Jacobian;
cov=inv(J'*J);
output.std_dev = sqrt(diag(cov));

% Residual
residual=(datain-output.fitted_spec')./sqrt(datain+background);

% Chi-square
chi_square=sum((1./(datain+background+1)).*...
    (datain-output.fitted_spec').^2);

% Reduced chi-square
output.red_chi_square=chi_square/(size(datain,2) -...
    size(output.aparameter,2));




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%% Plot fit with data and create figure for saving%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ys = get(axes_result, 'YScale');

axes(axes_result);
plot(energyin,output.fitted_spec,'r-',...
    energyin, yData,'b.','linewidth',1.5);
set(axes_result, 'YScale', ys);

leg_fit=legend('Fit','Data',  'Location', 'Best' );


data = get(lines2fit, 'data');
number2fit = size(data,1);

% Set axis limits
xmin=min(energyin);
xmax=max(energyin);

% Check scale
if strcmp(ys, 'lin')
    ymin=0;
else
    ymin = 0.1;
end

% Set limits
```

```
ymax=1.3*max(datain);
xlim(axes_result,[xmin xmax]);
ylim(axes_result,[ymin ymax]);

ydiff = ymax-ymin;

% for i=1:number2fit
%     data(i,2) = strrep(data(i,2),'-lines','');
%     str=[data(i,1) data(i,2) '\downarrow'];
%     pointx = cellenergy{i,1}(1,1);
%     channelx = pointx/gain-zero;
%     pointy = feval(output.fitresult,channelx);
%     text(pointx,pointy+ydiff*0.15,str,'FontSize',12);
%
% end




% Label axes
xlabel ('Energy [keV]')
ylabel ('Intensity [counts]')
grid on






% Generate figure for saving the graph as pdf in fit_program
output.fig_fit=figure('visible','off','units','normalized',...
    'position',[.1 .1 .9 .7]);
copyobj([leg_fit,axes_result],output.fig_fit);
set(gca,'Units','normalized')
set(gca,'OuterPosition',[0 0 1 1])
set(gca,'position',[0.08 0.15 0.85 0.8])
set(gca,'LineWidth',1.5,'FontSize',16)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%% Plot residuals with data and create figure for saving%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

axes(axes_residual);
plot(energyin,residual,'b.','linewidth',1);

% Label axes
xlabel ('Energy [keV]')
ylabel ('Residual')
line([0 max(energyin)],[3 3],'Color','g','linewidth',1);
line([0 max(energyin)],[-3 -3],'Color','g','linewidth',1);
leg_res=legend('Residual', 'Statistical Boundary',...
    'Location', 'Best');
grid on
```

```
346
347 % Set axis limits
348 xmin=min(energyin);
349 xmax=max(energyin);
350 ymin=-5;
351 ymax=+5;
352
353 xlim(axes_residual,[xmin xmax]);
354 ylim(axes_residual,[ymin ymax]);
355
356 % Generate figure for saving the graph as pdf in fit_program
357 output.fig_res=figure('visible','off','units','normalized',...
358     'position',[.1 .1 .9 .4]);
359 copyobj([leg_res,axes_residual],output.fig_res);
360 set(gca,'Units','normalized')
361 set(gca,'OuterPosition',[0 0 1 1])
362 set(gca,'position',[0.06 0.25 0.9 0.70])
363 set(gca,'LineWidth',1.5,'FontSize',16)
364
365 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
366
367 % Calculate chi square and background for every peak
368 [output_chi_back]=peak_chisquare_back(cellenergy,output.fitparameter,...
369     channelroi,pair_energy,datain,output.fitted_spec,background,...
370     input.backcheck,cellstr,line_asr);
371
372 % Add chi square and background to output
373 output.chi_peak_cell=output_chi_back.chi_peak_cell;
374 output.back_peak_cell=output_chi_back.back_peak_cell;
375 output.chi_peak_asr=output_chi_back.chi_peak_asr;
376 output.back_peak_asr=output_chi_back.back_peak_asr;
377 output.index_asr=output_chi_back.index_asr;
378
379 toc
```

C.8 get_element_edge_data.m

```
1  function output = get_element_edge_data(element_num,all_lines)
2
3      % Get line energies for each edge
4      output.currentKenergy=all_lines{element_num,1}{1,2};
5      output.currentL1energy=all_lines{element_num,2}{1,2};
6      output.currentL2energy=all_lines{element_num,3}{1,2};
7      output.currentL3energy=all_lines{element_num,4}{1,2};
8      output.currentM5energy=all_lines{element_num,5}{1,2};
9
10     % Get line relativities for each edge
11     output.currentKrel=all_lines{element_num,1}{1,3};
12     output.currentL1rel=all_lines{element_num,2}{1,3};
13     output.currentL2rel=all_lines{element_num,3}{1,3};
14     output.currentL3rel=all_lines{element_num,4}{1,3};
15     output.currentM5rel=all_lines{element_num,5}{1,3};
16
17     % Get line names for each edge
18     output.currentKstring=all_lines{element_num,1}{1,1};
19     output.currentL1string=all_lines{element_num,2}{1,1};
```

```
20      output.currentL2string=all_lines{element_num,3}{1,1};
21      output.currentL3string=all_lines{element_num,4}{1,1};
22      output.currentM5string=all_lines{element_num,5}{1,1};
23
24      % Get only Kalpha and Kbeta lines
25      numberoflines=size(output.currentKenergy,1);
26
27      if numberoflines == 1
28
29          output.currentKastring=all_lines{element_num,1}{1,1}(1);
30          output.currentKaenergy=all_lines{element_num,1}{1,2}(1);
31          output.currentKarel=all_lines{element_num,1}{1,3}(1);
32
33          output.currentKbstring=[];
34          output.currentKbenergy=[];
35          output.currentKbrel=[];
36
37      elseif numberoflines == 2
38
39          output.currentKastring=all_lines{element_num,1}{1,1}(1);
40          output.currentKaenergy=all_lines{element_num,1}{1,2}(1);
41          output.currentKarel=all_lines{element_num,1}{1,3}(1);
42
43          output.currentKbstring=all_lines{element_num,1}{1,1}(2);
44          output.currentKbenergy=all_lines{element_num,1}{1,2}(2);
45          output.currentKbrel=all_lines{element_num,1}{1,3}(2);
46
47      elseif numberoflines == 3
48
49          output.currentKastring=all_lines{element_num,1}{1,1}(1:2);
50          output.currentKaenergy=all_lines{element_num,1}{1,2}(1:2);
51          output.currentKarel=all_lines{element_num,1}{1,3}(1:2);
52
53          output.currentKbstring=all_lines{element_num,1}{1,1}(3);
54          output.currentKbenergy=all_lines{element_num,1}{1,2}(3);
55          output.currentKbrel=all_lines{element_num,1}{1,3}(3);
56
57      elseif numberoflines == 4
58
59          output.currentKastring=all_lines{element_num,1}{1,1}(1:2);
60          output.currentKaenergy=all_lines{element_num,1}{1,2}(1:2);
61          output.currentKarel=all_lines{element_num,1}{1,3}(1:2);
62
63          output.currentKbstring=all_lines{element_num,1}{1,1}(3:4);
64          output.currentKbenergy=all_lines{element_num,1}{1,2}(3:4);
65          output.currentKbrel=all_lines{element_num,1}{1,3}(3:4);
66
67      elseif numberoflines == 5
68          output.currentKastring=all_lines{element_num,1}{1,1}(1:2);
69          output.currentKaenergy=all_lines{element_num,1}{1,2}(1:2);
70          output.currentKarel=all_lines{element_num,1}{1,3}(1:2);
71
72          output.currentKbstring=all_lines{element_num,1}{1,1}(3:5);
73          output.currentKbenergy=all_lines{element_num,1}{1,2}(3:5);
74          output.currentKbrel=all_lines{element_num,1}{1,3}(3:5);
```

```
75
76      else
77          output.currentKastring=[];
78          output.currentKaenergy=[];
79          output.currentKarel=[];
80
81          output.currentKbstring=[];
82          output.currentKbenergy=[];
83          output.currentKbrel=[];
84      end
```

C.9 get_element_line.m

```
1  function [element_string,element_num] = get_element_line(h_elements)
2
3  % Get selected element
4  el_index = get(h_elements,'Value');
5  % +1 because H is not in the list
6  element_num = el_index+1;
7  % Get selected element string
8  list_element=get(h_elements,'String');
9  % Save selected element string in array
10 element_string=list_element(el_index);
```

C.10 linlogSelection.m

```
1  function linlogSelection(source, callbackdata)
2
3  % Axes handles for lin/log change
4  axesData = get(source,'UserData');
5
6  for i=1:length(axesData)
7      axes = axesData(i);
8      if strcmp(callbackdata.NewValue.String,'lin')
9          set(axes,'YScale','lin');
10         yl = ylim(axes);
11         yl(1) = 0;
12         ylim(axes,yl);
13     else
14         set(axes,'YScale','log');
15         yl = ylim(axes);
16         yl(1) = 0.1;
17         ylim(axes,yl);
18     end
19 end
```

C.11 peak_chisquare_back.m

```
1  function [output] = peak_chisquare_back(cellenergy,fitparameter,...
2      channelroi,pair_energy,datain,fitted_spec,background,backcheck,...
3      cellstr,lineasr)
4
5  % Size of input
6  size_edges=size(cellenergy,2);
7
8  % Create cell that indicates the lines that show up in ASR
```

```
output.index_asr=cell(1,size_edges);

%Find lines that show up in ASR file
for i=1:size_edges
    %If KA line is in list
    if lineasr(i)==1
        poska=strfind(cellstr{i},'KA');
        indexka = find(not(cellfun('isempty', poska)));
        output.index_asr{i} = indexka;
    end
    %If LA line is in list
    if lineasr(i)==2
        posL3M5=strfind(cellstr{i},'L3M5');
        indexL3M5 = find(not(cellfun('isempty', posL3M5)));
        posL3M4=strfind(cellstr{i},'L3M4');
        indexL3M4 = find(not(cellfun('isempty', posL3M4)));
        indexL3=[indexL3M5;indexL3M4];
        output.index_asr{i} = indexL3,
    end
    %If MA line is in list
    if lineasr(i)==3
        posM5=strfind(cellstr{i},'M5');
        indexM5 = find(not(cellfun('isempty', posM5)));
        output.index_asr{i} = indexM5;
    end

end


% Aparameter=fitparameter(1:size_edges);
zero=fitparameter(size_edges+1);
gain=fitparameter(size_edges+2);
noise=fitparameter(size_edges+3);
fano=fitparameter(size_edges+4);

% Error for each channel
std_sqr=datain+1;

% Residuum for each channel
res=datain-fitted_spec';

% Initialisation
output.chi_peak_cell=cell(1,size_edges);
output.back_peak_cell=cell(1,size_edges);
output.chi_peak_asr(1:size_edges)=0;
output.back_peak_asr(1:size_edges)=0;

% Find Number of peak overlaps (FWIM) for each background channel
if backcheck==1
    channelcounter_lines(1:size(channelroi,2))=1; % background overlap
else
    % Distribution of the background to overlapping peaks
    channelcounter_lines(1:size(channelroi,2))=0;
    for i=1:size_edges

```

```
64          % Determine upper and lower limit for each peak (FWTM)
65          energy = cellenergy{i};
66          peakchannel=(energy-zero)./gain;
67          fwhm=sqrt(noise^2+2.35^2*fano*pair_energy.*energy.*1000)./1000;
68          fwtm=fwhm.*sqrt(log(10)/log(2));
69          channelfwtm=fwtm./gain;
70          upper_limit=ceil(peakchannel+channelfwtm./2);
71          lower_limit=ceil(peakchannel-channelfwtm./2);
72          size_lines=size(cellenergy{i},1);
73
74          % If peak is not in the range of the spectrum
75          for j=1:size_lines
76              if lower_limit(j)>max(channelroi)
77                  continue
78              end
79
80              if upper_limit(j)<min(channelroi)
81                  continue
82              end
83
84              for k=lower_limit(j):upper_limit(j)
85                  index=find(channelroi==k);
86                  channelcounter_lines(index)=channelcounter_lines(index)+1;
87              end
88          end
89      end
90  end
91
92
93  % Determine the background per channel (with/without overlap)
94  for i=1:size_edges
95      energy = cellenergy{i};
96      % Calculate fwtm in units of channels
97      peakchannel=(energy-zero)./gain;
98      fwhm=sqrt(noise^2+2.35^2*fano*pair_energy.*energy.*1000)./1000;
99      fwtm=fwhm.*sqrt(log(10)/log(2));
100     channelfwtm=fwtm./gain;
101     % Upper and lower limit (FWTM)
102     upper_limit=ceil(peakchannel+channelfwtm./2);
103     lower_limit=ceil(peakchannel-channelfwtm./2);
104
105     size_lines=size(cellenergy{i},1);
106     chi_peak_vec=[];
107     chi_peak_vec(1:size_lines)=0;
108     back_peak_vec=[];
109     back_peak_vec(1:size_lines)=0;
110
111     % Chisquare for ASR file
112     if ~isempty(output.index_asr{i})
113
114         upper_limit_asr = max(upper_limit(output.index_asr{i}));
115         lower_limit_asr = min(lower_limit(output.index_asr{i}));
116
117         if lower_limit_asr>max(channelroi) ||...
118                 upper_limit_asr>max(channelroi) ||...
```

```
119                 upper_limit_asr<min(channelroi) ||...
120                 lower_limit_asr<min(channelroi)
121
122             output.chi_peak_asr(i)=NaN;
123             continue
124         end
125
126         chi_peak_sqr_asr = 0;
127
128         for k=lower_limit_asr:upper_limit_asr
129             index=find(channelroi==k);
130             chi_peak_sqr_asr=...
131                 chi_peak_sqr_asr+((1/std_sqr(index))*(res(index))^2);
132         end
133
134     output.chi_peak_asr(i)=...
135         chi_peak_sqr_asr/(upper_limit_asr-lower_limit_asr);
136     end
137
138     %Chi-square and Background for each line
139     for j=1:size_lines
140         chi_peak_sqr=0;
141         back_peak=0;
142
143         % Skip lines that are not in the ROI
144         if lower_limit(j)>max(channelroi) ||...
145                 upper_limit(j)>max(channelroi) ||...
146                 upper_limit(j)<min(channelroi) ||...
147                 lower_limit(j)<min(channelroi)
148
149             back_peak_vec(j)=NaN;
150             chi_peak_vec(j)=NaN;
151             continue
152         end
153
154         % Sum up background and calculate chi square
155         for k=lower_limit(j):upper_limit(j)
156             index=find(channelroi==k);
157             chi_peak_sqr=...
158                 chi_peak_sqr+((1/std_sqr(index))*(res(index))^2);
159             back_peak=...
160                 back_peak+background(index)/channelcounter_lines(index);
161         end
162
163         back_peak_vec(j)=back_peak;
164         chi_peak_vec(j)=chi_peak_sqr/(upper_limit(j)-lower_limit(j));
165
166     end
167
168     output.back_peak_asr(i)=sum(back_peak_vec(output.index_asr{i}));
169     output.chi_peak_cell{i}=chi_peak_vec;
170     output.back_peak_cell{i}=back_peak_vec;
171 end
```

C.12 plot_KLMmarker.m

```
function plot_KLMmarker(data,energy,error,data_edges)

energystruct={data_edges.currentKenergy,data_edges.currentL1energy,...
    data_edges.currentL2energy,data_edges.currentL3energy,...
    data_edges.currentM5energy};
relstruct={data_edges.currentKrel,data_edges.currentL1rel,...
    data_edges.currentL2rel,data_edges.currentL3rel,...
    data_edges.currentM5rel};
color={'r','m','g','k','y'};
m=0;
n=0;
for i=1:5
    if ~isempty(energystruct{i})
        num=size(energystruct{i},1);
        X(1,:)=energystruct{i};
        X(2,:)=X(1,:);
        Y(1,1:num)=0.001;
        a=energy-X(1,1);
        [~,index]=min(abs(a));
        Y(2,1)=data(index)/relstruct{i}(1);
        Y(2,:)=Y(2,1)*relstruct{i}(:);

        hold on
        for j=1:num
            % Check if lines exist that are in range of the spectrum
            if X(1,j)<max(energy) && X(1,j)>min(energy)
                line(X(:,j),Y(:,j),'Color',color{i},'Linewidth',1.5);
                n=n+1;
            end

        end
        clear X Y
    else
        m=m+1;
    end

end

if m==5
    disp('bla')
    set(error,'String','no edges in Library');
end

if m~=5 && n==0
    set(error,'String','no edges in ROI');
end
```

C.13 plot_linemarker.m

```
function plot_linemarker(currentenergy,currentrel,data,energy,axes2,error,
    color)

if ~isempty(currentenergy)
    num=size(currentenergy,1);
    X(1,:)=currentenergy;
    X(2,:)=X(1,:);
```

```
7       Y(1,1:num)=0.000001;
8       a=energy-X(1,1);
9       [~,index]=min(abs(a));
10      Y(2,1)=data(index)/currentrel(1);
11      Y(2,:)=Y(2,1)*currentrel(:);
12
13
14      % Limits for the second plot to see all the lines
15      xmin=min(min(X))-1;
16      if(xmin<0)
17          xmin=0;
18      end
19      xmax=max(max(X))+1;
20
21      if(xmax>max(energy))
22          xmax=max(energy);
23      end
24      ymin=0;
25      ymax=max(max(Y)*2);
26
27      if(ymax<max(data)*0.02)
28          ymax=max(data)*0.02;
29      end
30
31      % Check if lines exist that are in range of the spectrum
32      if min(min(X))<max(energy) && max(max(X))>min(energy)
33
34          % Plot in axes 2 the spectrum including the lines
35          axes(axes2);
36          yscale = get(axes2,'YScale');
37          plot(energy,data);
38
39          if strcmp(yscale,'lin');
40              ylim(axes2,[ymin ymax]);
41          else
42              ylim(axes2,[0.1 ymax]);
43          end
44          xlim(axes2,[xmin xmax]);
45          set(axes2,'YScale',yscale);
46          line(X,Y,'Color',color,'Linewidth',1.5);
47          xlabel('Energy [keV]');
48          ylabel('Intensity [counts]');
49          grid on
50      else
51          set(error,'String','Error: No line(s) is/are in the range of the
                spectrum for this edge!!!');
52      end
53
54
55  else
56
57      set(error,'String','Error: Unknown');
58
59  end
```

C.14 plotall_linemarker.m

```
function plotall_linemarker(currentenergy,currentrel,data,energy,h_axes,
    error,color)

% Plot lines in tab 2
num=size(currentenergy,1);
X(1,:)=currentenergy;
X(2,:)=X(1,:);
Y(1,1:num)=1;
a=energy-X(1,1);
[~,index]=min(abs(a));
Y(2,1)=data(index)/currentrel(1);
Y(2,:)=Y(2,1)*currentrel(:);

if min(min(X))<max(energy) && max(max(X))>min(energy)

    disp(X)
    % Plot in axes 2 the spectrum including the lines
    axes(h_axes);
    hold on
    line(X,Y,'Color',color,'Linewidth',1.5);
    xlabel('Energy [keV]');
    ylabel('Intensity [counts]');
    grid on
else
    set(error,'String','Error: Some lines in File out of range!!!');
end
```

C.15 read_asrfile.m

```
function [calArray,file] = read_asrfile(file, path)


if (~isequal(file,0))

    % If the output of filename is no cell
    if ~iscell(file)
        file={file};
    end

    % Number of files
    numfile = numel (file);

    for index=1:numfile
        % Combines path and file
        filename=fullfile(path,file{index});
        fileID = fopen(filename,'r');
        stop=1;
        k=1;
        while stop
            C = textscan(fileID,'%s %s %[^\n]',1);
            x=strcmp(C{1,1},'$PEAKS:');
            if x==1 %Stop when $DATA line found
                stop=0;
            end
```

```
26          k=k+1;
27          end
28          % Read line after $DATA-line
29          y = textscan(fileID,'%f %[^\n]',1);
30          % Number of datapoints
31          lines=y{1,2};
32          % Read peaks
33          formatSpec = '%f%f%f%f%f%f%f%[^\n\r]';
34          calArray{index} = textscan(fileID, formatSpec, 'Delimiter', ' ', '
                MultipleDelimsAsOne', true,'ReturnOnError', false);
35          calArray{index}(7)=[];
36
37      end
38          fclose(fileID);
39  end
40  clearvars filename path fileID C column1 column2 dataline indexdataline
        number lines datatext Raw ans;
```

C.16 read_axiscalibration.m

```
1  function [energy,zero,gain] = read_axiscalibration(channel,file,path)
2
3
4  filename=[path file];
5  fileID = fopen(filename,'r');
6
7  C = textscan(fileID,'%f %[^\n]',2,'HeaderLines',3);
8
9  coeff=C{1,1};
10
11 energy = coeff(2)+channel*coeff(1);
12 zero = coeff(2);
13 gain = coeff(1);
```

C.17 read_backgroundfile.m

```
1  function [data] = read_backgroundfile(path,file)
2
3  filename=[path file];
4  fileID = fopen(filename,'r');
5
6  C = textscan(fileID,'%s');
7
8  data=str2double(C{1,1});
```

C.18 read_calibrationfile.m

```
1  function [internal,coeffK,coeffL,coeffM,error] = read_calibrationfile(
       filename,path)
2
3  internal=0;
4
5  error='no error';
6
7  % Opens the relative sensitivity file
8  fileID = fopen(fullfile(path,filename),'r');
```

```
9
10 % Read relative sensitivity file
11 koeff = textscan(fileID,'%s%[^\n\r]','Delimiter','','MultipleDelimsAsOne',
       true, 'ReturnOnError', false);
12
13 % Find Internal begin and end an take the value between as internal
14 stringint=strcmp(koeff{1,1},'Internal');
15 indexint=find(stringint==1);
16
17 % If there is no such Keyword rturn error
18 if ~isempty(indexint)
19     internal=str2double(koeff{1,1}(indexint(1)+1,1));
20 else
21     error='Error:No internal standard in the relative intensities file
           found!!!!';
22 end
23
24 % Find Line1 begin end an take the values between as coefficient
25 stringK=strcmp(koeff{1,1},'Line1');
26 indexK=find(stringK==1);
27 if ~isempty(indexK)
28     coeffK(:)=str2double(koeff{1,1}(indexK(1)+1:indexK(2)-1,1));
29 else
30     coeffK=[];
31 end
32
33 % Find Line2 begin end an take the values between as coefficient
34 stringL=strcmp(koeff{1,1},'Line2');
35 indexL=find(stringL==1);
36 if ~isempty(indexL)
37     coeffL(:)=str2double(koeff{1,1}(indexL(1)+1:indexL(2)-1,1));
38 else
39     coeffL=[];
40 end
41
42 % Find Line3 begin end an take the values between as coefficient
43 stringM=strcmp(koeff{1,1},'Line3');
44 indexM=find(stringM==1);
45 if ~isempty(indexM)
46     coeffM(:)=str2double(koeff{1,1}(indexM(1)+1:indexM(2)-1,1));
47 else
48     coeffM=[];
49 end
```

C.19 read_library.m

```
1 function [all_lines]=read_library(path,file)
2
3 if path=='.'
4     path=fileparts(which('read_library.m'));%pwd;
5 end
6 % file='HEAVYELE_BLEI.txt'
7
8 filename = fullfile(path,file);
9
10 fileID = fopen(filename,'r');
```

```
C = textscan(fileID,'%s %f %f');
fclose(fileID);

str_elements=C{1};
energy_elements=C{2};
rel_elements=C{3};
elements={'H';'He';'Li';'Be';'B';'C';'N';'O';'F';'Ne';'Na';'Mg';'Al';'Si';'
    P';'S';'Cl';'Ar';'K';'Ca';'Sc';'Ti';'V';'Cr';'Mn';'Fe';'Co';'Ni';'Cu';'
    Zn';'Ga';'Ge';'As';'Se';'Br';'Kr';'Rb';'Sr';'Y';'Zr';'Nb';'Mo';'Tc';'Ru'
    ;'Rh';'Pd';'Ag';'Cd';'In';'Sn';'Sb';'Te';'I';'Xe';'Cs';'Ba';'La';'Ce';'
    Pr';'Nd';'Pm';'Sm';'Eu';'Gd';'Tb';'Dy';'Ho';'Er';'Tm';'Yb';'Lu';'Hf';'Ta
    ';'W';'Re';'Os';'Ir';'Pt';'Au';'Hg';'Tl';'Pb';'Bi';'Po';'At';'Rn';'Fr';'
    Ra';'Ac';'Th';'Pa';'U'};

numel=size(elements,1);


% For each element
for l=1:numel
    % Find current element
    posel=strcmp(str_elements, elements(l));
    % Find index of current element
    indexel=find(posel,1);
    % Find number of lines for current element
    num_lines=energy_elements(indexel);

    lines=str_elements(indexel+1:indexel+1+num_lines);
    energy=energy_elements(indexel+1:indexel+1+num_lines);
    relativ=rel_elements(indexel+1:indexel+1+num_lines);

    posK=regexp(lines, '^K', 'match');
    posL1=regexp(lines, '^L1', 'match');
    posL2=regexp(lines, '^L2', 'match');
    posL3=regexp(lines, '^L3', 'match');
    posM5=regexp(lines, '^M5', 'match');


    indexK=find(~cellfun(@isempty,posK));
    indexL1=find(~cellfun(@isempty,posL1));
    indexL2=find(~cellfun(@isempty,posL2));
    indexL3=find(~cellfun(@isempty,posL3));
    indexM5=find(~cellfun(@isempty,posM5));

    all_lines{l,1}={lines(indexK),energy(indexK),relativ(indexK)};
    all_lines{l,2}={lines(indexL1),energy(indexL1),relativ(indexL1)};
    all_lines{l,3}={lines(indexL2),energy(indexL2),relativ(indexL2)};
    all_lines{l,4}={lines(indexL3),energy(indexL3),relativ(indexL3)};
    all_lines{l,5}={lines(indexM5),energy(indexM5),relativ(indexM5)};

end

%Read library output is a cell
%cell{1,1}->Klines
%cell{1,2}->L1lines
%cell{1,3}->L2lines
```

```
%cell{1,4}->L3lines
%cell{1,5}->Mlines
%cell{1,x}{element,1}-->String of line
%cell{1,x}{element,2}-->energy of line
%cell{1,x}{element,3}-->relativ of line
```

C.20 read_linesfile.m

```
function [elements,lines,number]=read_linesfile(path,file)

filename=[path file];
fileID = fopen(filename,'r');

C = textscan(fileID,'%s%[^\n\r]');

pos_begin=strcmp(C{1,1},'Lines_begin');
pos_end=strcmp(C{1,1},'Lines_end');


index_begin=find(pos_begin==1);
index_end=find(pos_end==1);

number=index_end-index_begin-3;

frewind(fileID);

data_energy = textscan(fileID, '%s %s %[^\n]' ,number,'HeaderLines',2);

elements=data_energy{1,1};

lines=data_energy{1,2};
```

C.21 read_massatfile.m

```
function [energy,massat] = read_massatfile (path,file)

if path=='.'
    path=fileparts(which('read_massatfile.m'));%pwd;
end

filename=fullfile(path,file);
fileID=fopen(filename,'r');

C=textscan(fileID,'%f %f %[^\n\r]','Delimiter',' ', 'MultipleDelimsAsOne',
    true, 'ReturnOnError', false);

energy=C{1,1};
massat=C{1,2};
```

C.22 read_parameterfile.m

```
function parameter = read_parameterfile(path,file)

filename=fullfile(path,file); %[path file];
fileID = fopen(filename,'r');

```

```
C = textscan(fileID,'%s %s %[^\n\r]','Delimiter','\t', 'MultipleDelimsAsOne
    ', true,  'ReturnOnError', false);

parameter(:,1)=C{1,1};
parameter(:,2)=C{1,2};
```

C.23 read_spefile.m

```
function [channel,data,header] = read_spefile(file,path)

% Initialisation
channel=[];
data=[];
header=[];

m=0;
if (~isequal(file,0))
    % Combines path and file
    filename=[path file];
    fileID = fopen(filename,'r');
    stop=1;
    k=1;
    while stop
        if m==100
            break
        end
        m=m+1;
        C = textscan(fileID,'%s %s %[^\n]',1);
        x=strcmp(C{1,1},'$DATA:');
        if x==1 % Stop when $DATA line found
            stop=0;
        end
    k=k+1;
    end
    % Read line after $DATA-line
    y = textscan(fileID,'%f %f %[^\n]',1);
    % Number of datapoints
    number=y{1,2}+1;
    % Number of lines to read (10 datapoints per line)
    lines=ceil(number/10);
    % Read data
    datatext = textscan(fileID,'%f%f%f%f%f%f%f%f%f%f %[^\n]',lines,...
        'Delimiter',' ','MultipleDelimsAsOne',true,...
        'EmptyValue',0.0,'ReturnOnError',false);
    % Change cell to matrix
    Raw = [datatext{1:end-1}];
    % Reshape matrix to vector
    data=reshape(Raw',1,lines*10);
    % Cut off the zeros that are to much at the end
    data=data(1:number);
    % Generate channel vector
    channel=(1:size(data,2));
    % Set cursor to the top
    frewind(fileID);

    % Get header
```

```
49      C1 = textscan(fileID,'%s',k-2,'Delimiter','\n');
50      header=[C1{1,1}];
51
52      fclose(fileID);
53  end
54  clearvars filename file path fileID C column1 column2 dataline
        indexdataline number lines datatext Raw ans;
```

C.24 seperate_peakarea.m

```
1   function [aparameter,std_dev,area_extra,std_extra,Ydata,...
2       area_asr,std_asr] = seperate_peakarea(aparameter,allparameter,...
3       plotel_extra,channelroi,coeffnames,std_dev,cellrel,index)
4
5
6
7   % The variable index contains 1 for lines that show up in ASR file.
8   % Initialisation of parameters
9   for i=1:size(allparameter,2)
10      eval(sprintf('%s=%f',coeffnames{i,1},allparameter(i)));
11  end
12
13  area_extra=cell(1,size(aparameter,2));
14  std_extra=cell(1,size(aparameter,2));
15  area_asr(1:size(aparameter,2))=0;
16  std_asr(1:size(aparameter,2))=0;
17
18
19  Ydata(size(aparameter,2),size(channelroi,2))=0;
20  y(1:size(channelroi,2))=0;
21
22  % Save function for every edge
23  for j=1:size(aparameter,2)
24      for i = 1:size(channelroi,2)
25          x=channelroi(i);
26          y(i)=eval(plotel_extra{j});
27      end
28      % Data to plot lines separatly
29      Ydata(j,:)=y;
30  end
31
32  % Area for all lines
33  for j=1:size(aparameter,2)
34      absolute=sum(cellrel{j});
35      % Area for each line
36      area_extra{j}=aparameter(j)*cellrel{j}*(1./absolute);
37      % Area for line in ASR
38      area_asr(j)=sum(area_extra{j}(index{j}));
39  end
40
41  % Std for all lines
42  for j=1:size(aparameter,2)
43      % STD for each line
44      std_extra{j}=std_dev(j)*sqrt(cellrel{j}.*(1./absolute));
45      % STD for line in ASR
46      std_asr(j)=sqrt(sum(std_extra{j}(index{j}).^2));
```

```
47 end
```

C.25 write2asrfile.m

```
1  function write2asrfile(file,path,lines,elements,header,energyasr,area_asr,
       std_asr,chi_asr)
2
3  % Open result file
4  fileID = fopen(fullfile(path,file),'w');
5
6  % Number of k lines
7  k=sum(lines==1);
8  % Number of l lines
9  l=sum(lines==2);
10 % Number of m lines
11 m=sum(lines==3);
12
13 % Header line number
14 numheader=size(header,1);
15
16 % Write to file
17
18 % Write header into file
19 for i=1:numheader
20     fprintf(fileID,'%s\r\n',header{i});
21 end
22
23 fprintf(fileID,'$PEAKS:    \r\n');
24
25 fprintf(fileID,'%7i\r\n',k+l+m);
26
27 % Find position of lines for asr
28 posK=lines==1;
29 posL=lines==2;
30 posM=lines==3;
31
32 % Find index of lines for asr
33 indexK=find(posK==1);
34 indexL=find(posL==1);
35 indexM=find(posM==1);
36
37 % Element for lines of asr
38 elnumK=elements(indexK);
39 elnumL=elements(indexL);
40 elnumM=elements(indexM);
41
42 % Energy
43 energyasrK=energyasr(indexK);
44 energyasrL=energyasr(indexL);
45 energyasrM=energyasr(indexM);
46
47 % Chi-square
48 chi_asr_k = chi_asr(indexK);
49 chi_asr_l = chi_asr(indexL);
50 chi_asr_m = chi_asr(indexM);
51
```

```
52 % Area
53 area_asr_k = area_asr(indexK);
54 area_asr_l = area_asr(indexL);
55 area_asr_m = area_asr(indexM);
56
57 % Std area
58 std_asr_k = std_asr(indexK);
59 std_asr_l = std_asr(indexL);
60 std_asr_m = std_asr(indexM);
61
62 % Sort elements
63 [~,indexKsort]=sort(elnumK);
64 [~,indexLsort]=sort(elnumL);
65 [~,indexMsort]=sort(elnumM);
66
67 % Write different lines into file
68 for i=1:size(indexKsort,2)
69     fprintf(fileID,'%4i%4i%11.3f%11.0f.%11.0f.%11.2f\r\n',...
70         elnumK(indexKsort(i)),1,energyasrK(indexKsort(i)),area_asr_k(
            indexKsort(i)),...
71         std_asr_k(indexKsort(i)),chi_asr_k(indexKsort(i)));
72 end
73
74 for i=1:size(indexLsort,2)
75     fprintf(fileID,'%4i%4i%11.3f%11.0f.%11.0f.%11.2f\r\n',...
76         elnumL(indexLsort(i)),2,energyasrL(indexLsort(i)),area_asr_l(
            indexLsort(i)),...
77         std_asr_l(indexLsort(i)),chi_asr_l(indexLsort(i)));
78 end
79
80 for i=1:size(indexMsort,2)
81     fprintf(fileID,'%4i%4i%11.3f%11.0f.%11.0f.%11.2f\r\n',...
82         elnumM(indexMsort(i)),3,energyasrM(indexMsort(i)),area_asr_m(
            indexMsort(i)),...
83         std_asr_m(indexMsort(i)),chi_asr_m(indexMsort(i)));
84 end
85
86 fclose(fileID);
```