CHALMERS

Evaluation of Adaptive Methods for Developing Algorithms for Measurement of X-ray Radiation Properties

Master of Science Thesis in the Master Degree Programme Complex Adaptive Systems

THOMAS LOVÉN

Department of Signals and Systems
Division of Complex Adaptive Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Performed at Unfors RaySafe
Gothenburg, Sweden, 2013

Evaluation of Adaptive Methods for Developing Algorithms for Measurement of X-ray Radiation Properties

Thomas Lovén

Evaluation of Adaptive Methods for Developing Algorithms for Measurement of X-ray Radiation Properties
Master of Science Thesis in the Master Degree Programme Complex Adaptive Systems
THOMAS LOVÉN

Performed at
Unfors RaySafe AB
Uggledalsvägen 29
SE-427 40 Billdal, Sweden
Telephone: +46 (0)31 719 97 00

Examiner
Prof. Bo Håkansson
Department of Signals and Systems
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
Telephone: +46 (0)31 772 18 07

Supervisors
Mattias Gustavsson, Unfors RaySafe AB
Fredrik Oskarson, Unfors RaySafe AB

Abstract

The risks of X-ray exposure to the human body are well documented and its link to cancer proven. However, X-ray imaging remains an important part of modern medical science and the risks are in some circumstances considered acceptable. The exposure should always be kept as low as reasonably achievable, which requires detailed knowledge of the workings and performance of the X-ray equipment.

Unfors RaySafe develops and markets a range of instruments used for calibrating and testing X-ray generators and related equipment. One such instrument is the X2 R/F sensor, which measures radiation using four photo diodes. From the currents generated in the four diodes, information about several parameters of the machine and set-up can be extracted. Today, the parameters are calculated with algorithms developed through a combination of expert knowledge, brute-forcing and guesswork. In this study, an attempt is made to develop new measurement algorithms for one parameter using adaptive programming methods.

The parameter chosen for the focus of this study was generator acceleration voltage. Attempts were made to determine acceleration voltage using artificial neural networks and using a linear genetic programming algorithm. Both methods require a large number of matched input-output sets for training. Rather than making actual measurements, a numerical model was constructed to simulate the X-ray generator, radiation filtering and the X-ray detection device.

The constructed model is shown to be sufficiently accurate for use within this study, and can be improved for use in actual development through more careful calibration. Both methods attempted for developing measurement algorithms prove to work, and both generate algorithms with an output accuracy comparable to the currently used methods.

In conclusion, while the developed algorithms are not suitable for use in a finished product, they give some insight as to how the current algorithms or even detector design can be simplified. The developed numerical model can be generalized and used in future development projects.

For Anneli

Contents

Chapter 1

Introduction

1.1 Background

1.1.1 Medical X-ray imagery

The study of X-rays by Wilhelm Röntgen in the late nineteenth century opened a new path in medical sciences. Suddenly, the insides of a patients body could be seen without the risks associated with an operation.

The penetrating properties of X-rays are highly dependent of the energy of the radiations photons. Photons with higher energy penetrate deeper and through denser materials than photons with a lower energy. In medical imaging, this has to be taken into consideration during generation of the X-ray radiation. If the photon energy is too high it may pass right through bone and flesh alike, and give a highly over-exposed picture. If the photon energy is too low, it may be absorbed completely in the patients body. In order to make a high quality image, the photon energy must therefore be selected carefully with consideration the patient as well as the type of X-ray imagery. For example, in mammography the photon energy should be just high enough for the radiation to pass through healthy tissue, but low enough that it is absorbed by cancerous cells, which are only slightly denser [1].

While the energy of the most energetic photons in the X-ray radiation is important, a large number of low energy photons may also ruin the image. Those photons also add to the radioactive skin dose to the patient. Such low energy photons can be filtered away by passing the radiation through thin sheets of metal[2].

1.1.2 X-ray protection

Very soon after their discovery, it became apparent that X-rays come with their own set of risks. Today the connection between exposure to X-ray radiation and cancer is well studied and proven. The United Nations Scientific Committee on the Effects of Atomic Radiation have compiled several studies and estimate a total 4.3-7.2 % risk of contracting cancer per Sievert dose[3].

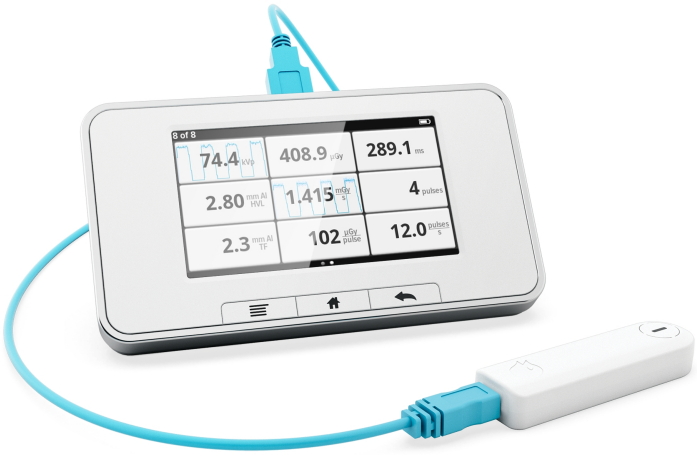

Figure 1.1: The X2 platform with Radioscopy/Fluoroscopy sensor.

The benefits of X-rays, both in medicine and other areas, are generally considered great enough to risk limited amounts of exposure. Several countries have governmental limits to the X-ray dose an individual may be subject to during a year or a lifetime. A common rule of thumb is the "ALARA-principle" – As Low As Reasonably Achievable.

What seems like the easiest way to reduce X-ray exposure is to simply think twice before making a new X-ray image. Is it really necessary? If it is, the next step is to get it right the first time, so that you won't have to remake the exposure later. This involves a lot of expert knowledge, but also being certain that your machine is well calibrated and outputs what you expect.

Unfors RaySafe develops and markets a range of measurement instruments for X-ray radiation. Typical use cases include calibration of X-ray machines in hospital environments. The `X2 platform` (seen in fig. 1.1), used in this study, is one such instrument which is designed to be very easy to use.

The operator or technician places the instruments sensor in the X-ray beam path, sets up the machine and makes an X-ray exposure. The instrument measures the radiation, calculates which machine parameters it corresponds to and displays it on the base station. It also displays other interesting data, such as the half value layer and combined radioactive dose.

The basic `R/F sensor` (for Radioscopy/Fluoroscopy) of the X2 platform gets its measurements from four photo diodes. In other words, the sensor transforms the space of machine-parameters, measurement setup and environmental conditions etc. to a four dimensional space of measurements. The X2 base unit uses those four values to calculate the wanted machine parameter.

1.2 Aim of study

The main purpose of this study is to evaluate whether – and if so, how well – adaptive optimization methods can be used to develop algorithms for X-ray property measurements with the X2 platform. Several methods were attempted and their performance evaluated. The optimization methods and their evaluation is described in chapter 3.

Most adaptive optimization methods require a large amount of data for training and evaluation. Rather than making a large number of actual measurements, a model should be developed which can simulate the X-ray setup and measurement device. The model is further discussed in chapter 2.

1.3 Scope

While the X2 platform and its R/F sensor can be used to measure a number of properties of the radiation, this study attempts only to solve the problem of measuring the energy of the most energetic photons.

The model and, by extension, the adaptive algorithms will consider only one detector – the one supplied by Unfors RaySafe for this study - and not consider differences between detectors. Likewise only the X-ray machines and tubes available for measurement will be considered. The model of the measurement setup will not include any simulated noise.

Chapter 2

X-ray system modelling

2.1 Theoretic background

2.1.1 X-ray generation

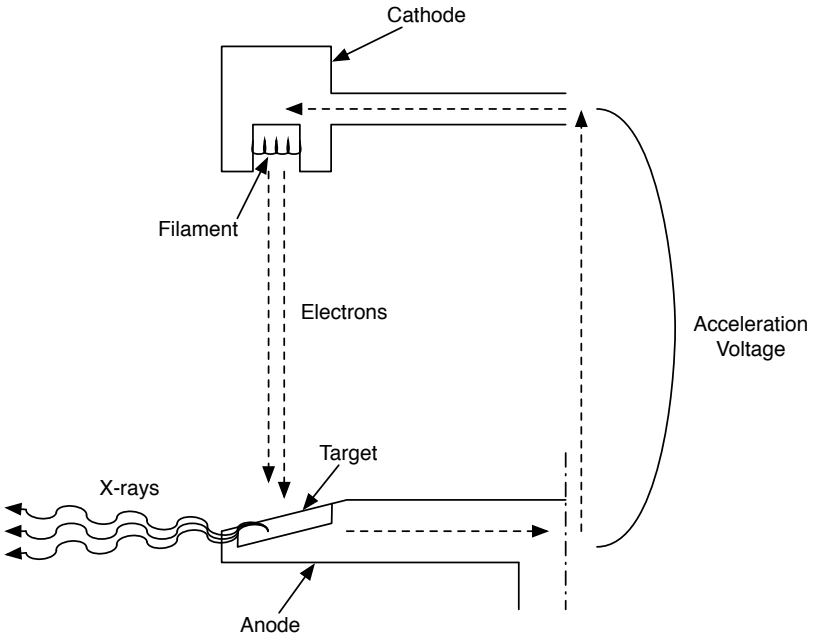

Figure 2.1: A schematic diagram of an X-ray tube. Electrons are accelerated from a filament towards the target where they interract with the target atoms to create X-ray radiation.

Figure 2.1 shows a schematic diagram of an X-ray tube such as might be used in medical applications. Inside the tube, a focused beam of electrons is accelerated towards a metal `target` by an `acceleration voltage` (`AV`).

X-rays are generated when the energized electrons interact with the atoms of the target[4]. Most electrons will be deflected one or many times by the electric field of the nuclei, losing some of their energy in the form of photons in the process. The radiation generated by this process is – for historical reasons – known as `Bremsstrahlung` and has a continuous energy spectrum.

Some electrons may interact by knocking an orbital electron from the lower energy levels of a target atom. As an electron from a higher energy level the vacancy a photon will be emitted carrying the excess energy. This energy and thus the frequency of the photon is characteristic to the target material and the radiation generated by this process is therefore called `characteristic radiation`.

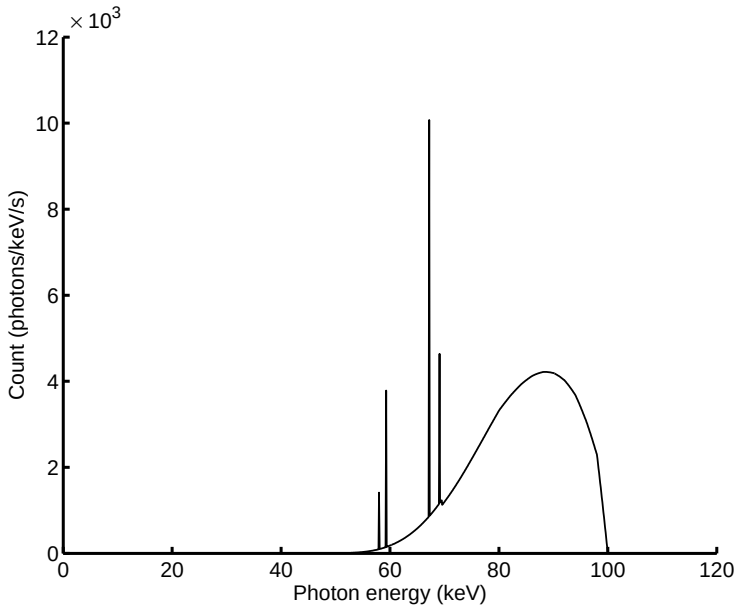

Figure 2.2: A typical X-ray spectrum generated using SpekCalc. The sharp, peaked characteristic radiation and the smooth Bremsstrahlung are combined into a combined spectrum.

These two effects combine to form the output radiation. The intensity of the radiation is directly proportional to the `electron current` (`EC`) through the X-ray tube. A typical X-ray spectrum can be seen in fig. 2.2. X-ray production only accounts for a few percent of the energy in the electrons, though. Most of the energy is converted to heat.

2.1.2 Interaction with matter

There are several ways in which photons can interact with matter. Which ones are more likely depends on the photon energy[5]. In the energy range normally used for medical X-ray imagery, the dominating interactions are the photoelectric effect and Compton scattering.

In photoelectric interaction a photon expends all of its energy to free an electron which is bound to a nucleus. Some of the energy is used to overcome the binding energy of the electron to the nucleus and the rest is transformed into kinetic energy of the electron.

In Compton interaction only some of the energy of the incident photon is transfered into kinetic energy. The rest is emitted in the form of another photon. The emitted photon will have a lower energy than the incident one and can interact again.

The interactions may remove an electron from the lower energy levels of the atom. Akin to the generation of characteristic radiation, this vacancy will soon be filled by an electron from a higher energy level and the difference in energy emitted in the form of a photon. Radiation emitted in this way is referred to as `secondary radiation`.

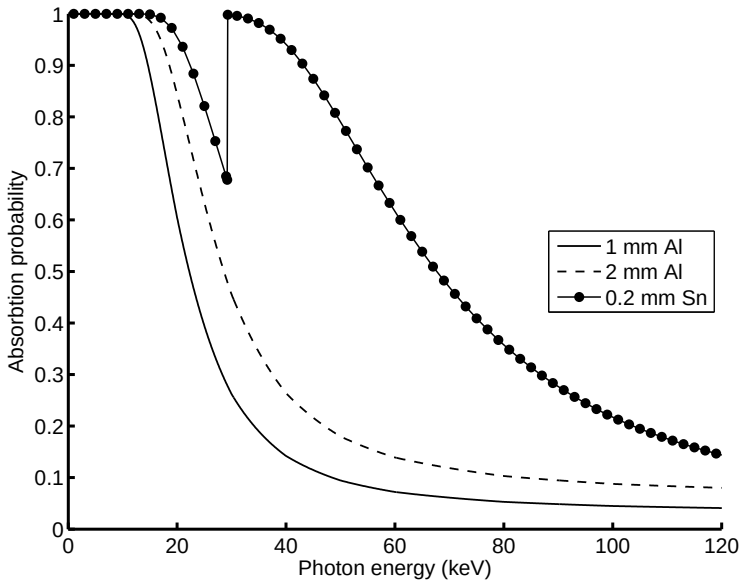

Figure 2.3: Fraction of incident photons absorbed by 1 mm of aluminum, 2 mm of aluminum and 0.2 mm of tin respectively. Note the K-edge effect in the tin absorption.

The probability of each type of interaction depends on the energy of the incident photon, with higher probability at lower energies. The combined probability of interactions form an absorption or attenuation probability. In fig. 2.3 the absorption probability of a photon when passing through 1 mm of aluminum, 2 mm of aluminum and 0.2 mm of tin is shown as a function of photon energy as calculated by the method described in section 2.2.2.

The sudden increase in absorption probability for tin seen in the figure is called a K-edge and is an effect of its electron structure. The photon energy of the discontinuity corresponds exactly to the energy required to excite an electron in the K-shell of tin[6].

Since most of the electron-atom interactions which generate the X-ray radiation take place inside the anode plate as opposed to on its surface, the radiation exiting the X-ray

tube has already passed through and been partially attenuated by some of the anode material.

The constant bombardment of high energy electrons may over time wear off some of the target material which may then deposit onto the exit window of the X-ray tube, causing further attenuation of the beam.

2.1.3 Beam quality and the N-series

The AV of the X-ray tube sets an upper limit to the energy of the photons in the X-ray radiation, and by placing a `filter` – for example a sheet of metal – in the beam path the number of photons with low energy can be limited. In this way it is possible to generate radiation with a narrow and well defined energy spectrum.

Since measuring the exact energy composition of is difficult, X-ray radiation is generally classified by way of AV and filtration. Those properties combined form the `beam quality`.

Note that the beam quality is not a measure of quality in the way of usability or purity of the radiation or anything similar. Instead it is simply a collective description of the conditions in which the radiation was generated.

The International Organization for Standardization have defined a set of beam qualities known as the N-series which can be used to calibrate X-ray machines and measurement devices[7]. The AV and filtrations of the N-series can be seen in table 2.1.

Name	Acceleration Voltage $[kV]$	Al $[mm]$	Cu $[mm]$	Sn $[mm]$	Pb $[mm]$
N-10	10	0.1	0	0	0
N-15	15	0.5	0	0	0
N-20	20	1.0	0	0	0
N-25	25	2.0	0	0	0
N-30	30	4.0	0	0	0
N-40	40	4.0	0.21	0	0
N-60	60	4.0	0.6	0	0
N-80	80	4.0	2.0	0	0
N-100	100	4.0	5.0	0	0
N-120	120	4.0	5.0	1.0	0
N-150	150	4.0	0	2.5	0
N-200	200	4.0	2.0	3.0	1.0
N-250	250	4.0	0	2.0	3.0
N-300	300	4.0	0	3.0	5.0

Table 2.1: Description of the beam qualities in the N-series as defined in ISO 4037-1.

2.1.4 X-ray detection

Both of the described interaction mechanisms result in free electrons inside the target material. This forms the basis in most modern X-ray detection and measurement devices[8].

A common X-ray detector is the Ionization Chamber in which the X-rays pass through a chamber filled with a gas such as air or a noble gas. The radiation ionizes some gas atoms and by applying a voltage over the chamber the free electrons form a current which can be measured.

Ionization chambers are bulky and require a high voltage to accelerate the free ions, so the R/F sensor of the X2 platform relies instead on common silicon photo diodes.

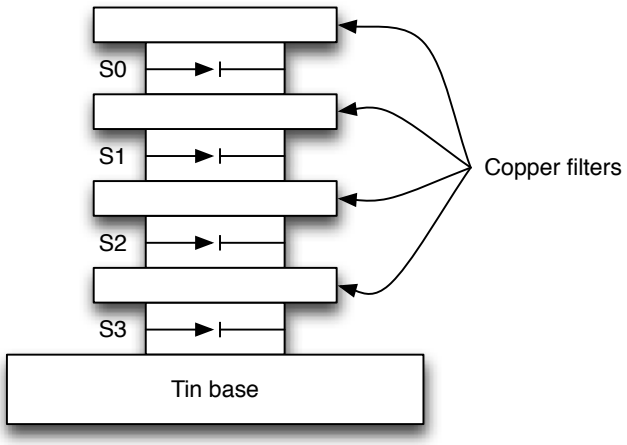

Figure 2.4: A schematic diagram of the Radioscopy/Fluoroscopy sensor of the X2 platform. The four diodes are separated by copper sheets and are mounted on a tungsten base. The radiation source should be above the stack.

The R/F sensor contains four diodes called S_0 through S_3 which are put in layers with thin sheets of copper (see fig. 2.4. In this way each sensor receives the same radiation but with different levels of filtration. The entire stack is placed on top of a tungsten base plate and each diode is connected to an integrated circuit which measures the generated current and sends the data to the base unit by means of a USB cable.

2.2 Model construction

The aim of this part of the project was to construct a computerized model of the X-ray measurement setup including generator, filtration and measurement device.

The model input should be an X-ray beam quality in the form of generator and measurement setup parameters, namely:

- X-ray tube acceleration voltage
- X-ray tube electron current
- X-ray tube anode material and angle
- Additional filtration
- Distance from anode focus point to measurement device.

The model output should be four values equal to the currents generated by the four measurement diodes in an X2 R/F sensor subject to X-ray radiation of the same beam quality.

2.2.1 X-ray generation

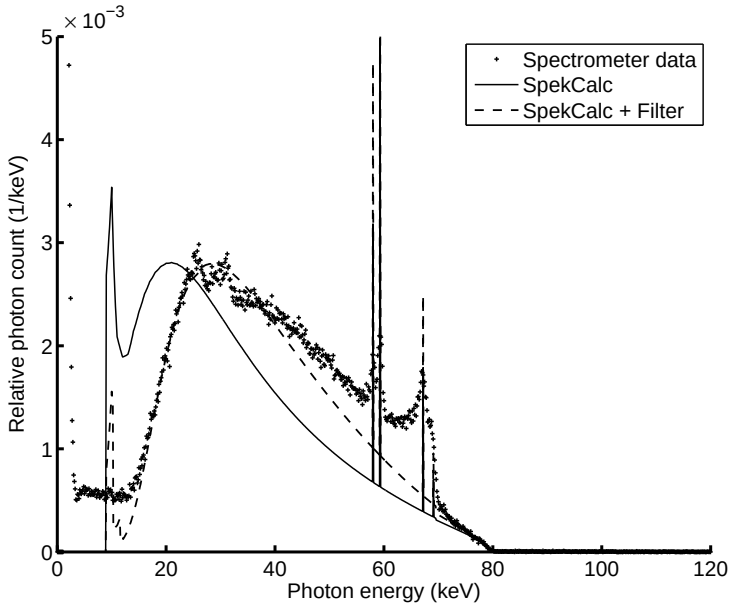

Figure 2.5: Normalized spectrometer measurements from a Philips Velara X-ray machine set at 80 kV AV and with no additional filtering. The figure also shows the SpekCalc output for the same settings as well as with 7 µm of additional filtering.

A computer program called SpekCalc was used to generate the basic X-ray spectra. SpekCalc is developed at McGill University of Montreal, Canada and is based on a deterministic model by Poludniowski and Evans at The Institute of Cancer Research of London, U.K. It can be used to generate X-ray spectra from tungsten anodes at AV up

to 130 kV. It also simulates filtration by some common elements and calculates a number of useful information about the radiation, such as energy content and administered radioactive dose[9–11].

Most of the features of SpekCalc were left unused for this project, as it was primarily used to generate simple unfiltered spectra which were saved and converted for use in MATLAB. The spectra were generated with a 12 ° anode angle and with AV from 10 to 300 kV in 1 kV increments. The spectra were set up as histograms with bin widths of 0.1 keV.

A few spectra from SpekCalc were compared to measurements made with an Amptek X123 X-ray spectrometer. Figure 2.5 shows a measurement with the spectrometer (dotted line) at a Philips Velara machine fitted with an SRM 0608 ROT-GS 505 tube. The measurement is made with with 80 kV AV and no filtering is compared to a SpekCalc spectra (solid line) with the same settings. The agreement is rather bad, with the most common photon energy offset and the overall shape of the spectrum being different.

The X-ray machines used in this study were fitted with well used X-ray tubes. Under the assumption that a layer of tungsten would have deposited onto the exit window (as described in section 2.1.2), the SpekCalc spectrum was adjusted with a filtration equivalent of 7 µm of tungsten as described in section 2.2.2. It was also adjusted for the sensitivity of the spectrometer as per the manufacturers specification[12]. The resulting spectrum (dashed line) agreed much better with the spectrometer measurement. The remaining difference is limited to energies below 69.525 keV, which is the energy of the K-edge of tungsten. This leads to the assumption that the some of the disagreement is due to secondary radiation from the tungsten aperture of the spectrometer itself or simply a numerical effect due to the finite energetic resolution of the spectrometer.

2.2.2 X-ray filtration

Filtration was modelled using measurements of the photon absorption probability of several common filter elements from the X-Ray Attenuation and Absorption for Materials of Dosimetric Interest database (XAAMDI)[13].

XAAMDI is one of three databases provided by the American National Institute of Standards and Technology. The data for the relevant elements and energy ranges from all three databases were compared, and the differences were found negligible. In the end XAAMDI was chosen for its usability and high resolution within the range 10-300 keV.

Data is provided in the form of a table of the photon mass attenuation coefficient μ/ρ ($[^{\mathrm{cm}^2}/_{\mathrm{g}}]$) as a function of photon energy. From this the absorption probability can be calculated as

$$A = e^{\mu/\rho \,\cdot\, \rho \,\cdot\, L}$$

where ρ is the filter density and L is its thickness.

This, in turn, is used to calculate the number of photons remaining after filtration

$$C_{out} = C_{in}/A = C_{in}e^{-\mu/\rho \,\cdot\, \rho \,\cdot\, L}$$

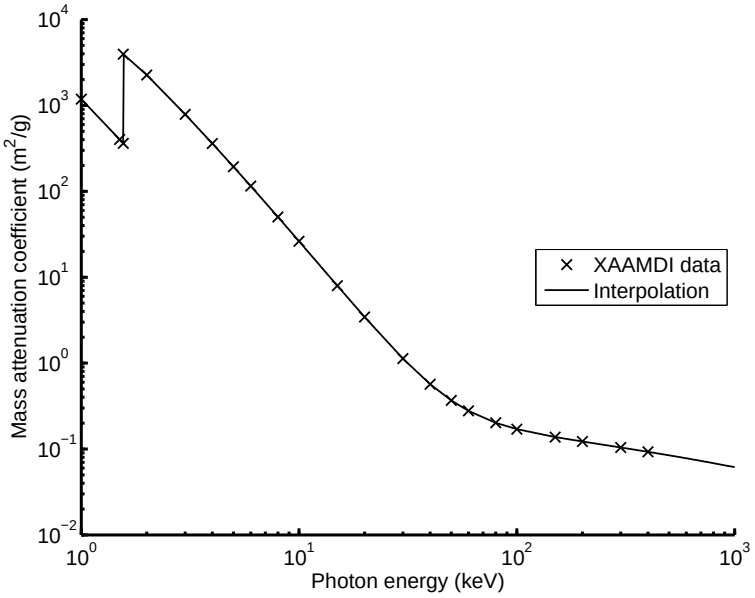

Figure 2.6: The mass attenuation coefficient of aluminum according to the XAAMDI database. The solid line shows the interpolation used throughout the project.

Since measurements are only available for a few tabulated energies, a logarithmic interpolation was used in between, i.e. two-point linear interpolation of $\log(\mu/\rho)$ as a function of $\log(E)$. This interpolation is illustrated in fig. 2.6.

Figure 2.7 shows the spectrometer measurements of unfiltered 80 kV radiation and radiation of the N-80 beam quality. The solid line in the figure shows data from SpekCalc after a simulated filtering to N-80 with data from XAAMDI as well as the extra 7 µm of tungsten described in section 2.2.1. The effect of the presumed secondary emissions from the spectrometer aperture can still be seen. Besides that, the difference between measurements and simulation is minimal.

2.2.3 X-ray detection

Definitions

The following section requires a bit of nomenclature to be established.

The following definitions are made with N being the number of energy bins in the spectrum, E_i being the mean energy and C_i the count of photons in bin i.

The **total number of photons** is calculated as the number of photons in the spectrum. This is used for normalizing spectra (as in fig. 2.7).

$$C_{tot} = \sum_{i=1}^{N} C_i \qquad [1/\mathrm{mAs}]$$

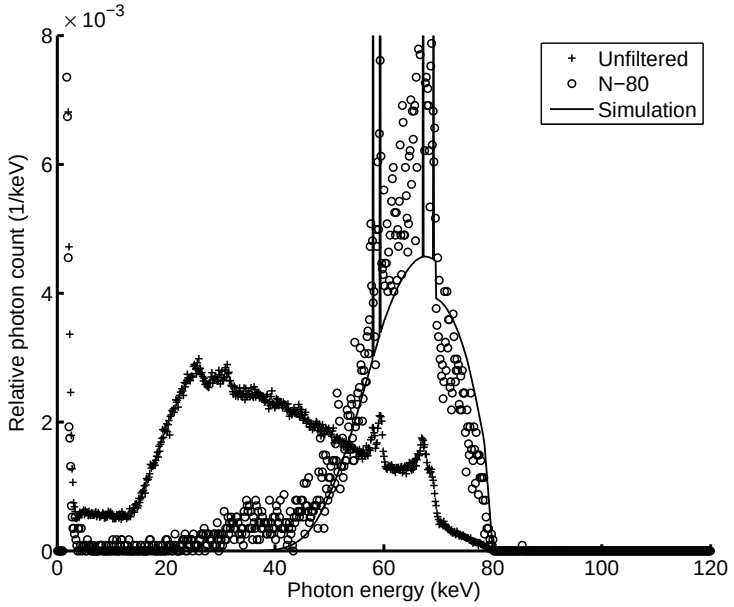

Figure 2.7: Normalized spectrometer measurements of unfiltered 80 kV X-ray radiation and radiation of the N-80 beam quality. The solid line shows simulations from SpekCalc after filtering with data from XAAMDI. The normalization constant is about 60 times greater for the unfiltered radiation than the filtered which makes the noise in the latter signal more prevalent.

The **total energy content** is a sum of the kinetic energy carried by the photons in one second of radiation.

$$E_{tot} = \sum_{i=1}^{N} E_i C_i \qquad [\mathrm{keV/mAs}]$$

The **mean energy** is an ordinary arithmetic mean of the energies of all photons in the spectrum.

$$\bar{E} = E_{tot}/C_{tot} \qquad [\mathrm{keV}]$$

The **peak count** is the number of photons with the most common energy *excluding characteristic radiation.*

$$\begin{aligned} C_{peak} &= C_{i_{peak}} && [1/\mathrm{mAs}] \\ i_{peak} &= i : C_i \geq C_j, && \forall j \end{aligned}$$

Finally, the **bulk radiation** is the radiation with energies limited above and below the mean energy by bin with a photon count exactly half the peak count.

Figure 2.8 shows an X-ray spectrum for the N-100 beam quality with 1 mA EC and illustrates the described properties.

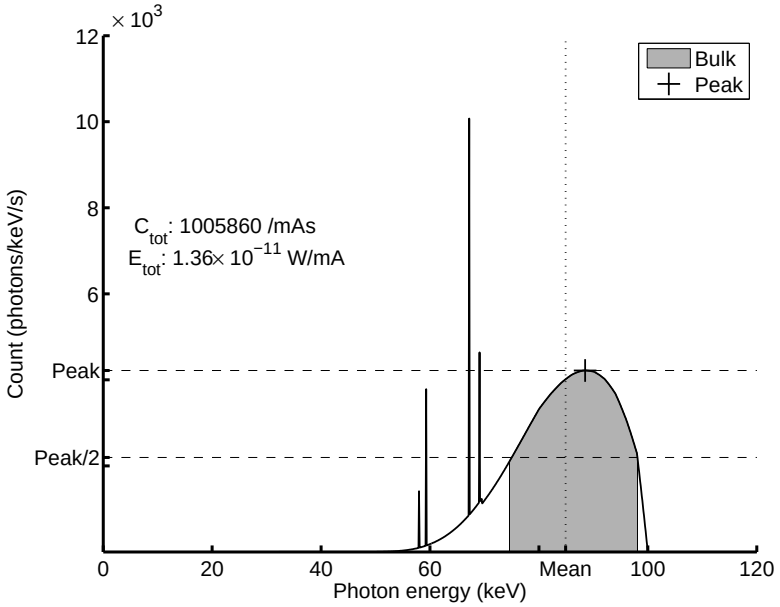

Figure 2.8: Illustration of spectral properties for the N-80 beam quality.

Diode sensitivity

The aim of this part of the model is to translate X-ray spectra to a detector output in the form of an electric current. The simplest way would be to assume that every photon reaching the detector gives rise to a current which in some way depends on the photons energy and that this dependence is continuous.

An attempt was made to find the diodes sensitivity by measuring the current output from one sensor subject to a number of carefully chosen beam qualities. The beam qualities were chosen to maximize spectral coverage in the common X-ray energy interval 30-300 kV. Assuming that the major contribution to the detector current comes from the bulk radiation, care was taken to minimize the overlap of bulks between beam qualities. Finally, the filtration was adjusted to match what was available in the test lab at Unfors Raysafe. The chosen beam qualities are listed in table 2.2 and their normalized spectra are shown in fig. 2.9.

Number	Tube Voltage $[kV]$	Al $[mm]$	Cu $[mm]$	Sn $[mm]$	$\bar{E}$ $[keV]$	E_{tot} $[W/mA]$	Bulk width $[keV]$
1	20	1.0	0	0	17.1	$1.4 \cdot 10^{-9}$	4.3
2	30	4.0	0	0	25.2	$4.7 \cdot 10^{-9}$	7.5
3	40	4.0	0.21	0	33.7	$6.1 \cdot 10^{-9}$	9.8
4	50	4.0	0.6	0	42.5	$6.3 \cdot 10^{-9}$	11.5
5	60	4.0	2.0	0	53.3	$1.2 \cdot 10^{-9}$	9.9
6	70	4.0	2.0	0	59.9	$7.1 \cdot 10^{-9}$	15.2
7	90	4.0	5.0	0	77.7	$4.4 \cdot 10^{-9}$	17.1
8	100	4.0	5.0	0	84.2	$1.3 \cdot 10^{-8}$	23.2
9	120	4.0	5.0	1.0	101.5	$1.4 \cdot 10^{-8}$	28.0
10	150	4.0	0	2.5	125.4	$2.7 \cdot 10^{-8}$	38.0

Table 2.2: Beam qualities used in measurements for finding the diode sensitivity.

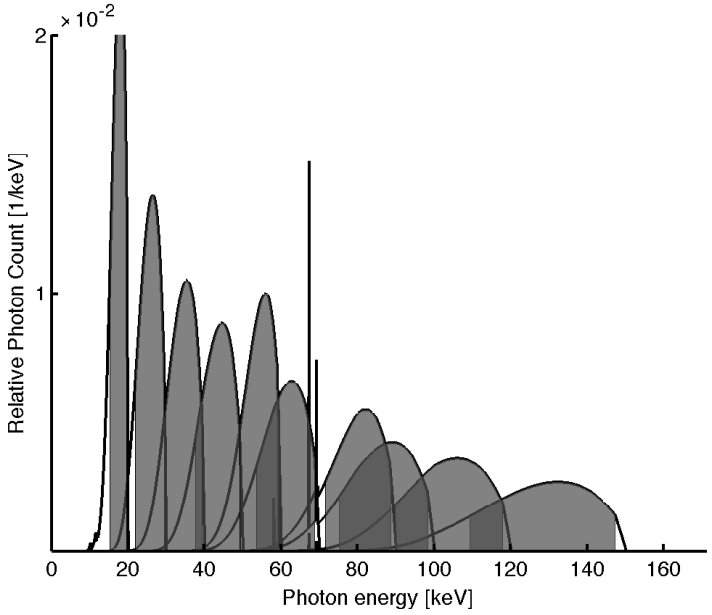

Figure 2.9: Normalized spectra of the beam qualities used in measurements for finding the diode sensitivity.

Measurements of these beam qualities were made at two different X-ray machines. Most measurements were made on the Philips Velara previously described. This machine has a minimum AV of 40 kV, so measurements at lower AV were made on an X-ray machine for mammography. Care was taken to have similar measurement conditions during all measurements. However, variations in atmospheric pressure and ambient or X-ray tube temperature were not measured or accounted for.

The detector used was an Unfors RaySafe X2 R/F sensor and measurements were recorded using X2Term - a program designed by Unfors Raysafe for internal development use. X2Term records the exact currents as measured by the four diodes. At least three measurements were made of each beam quality and the measured currents were averaged.

A spectrum was simulated for each beam quality using SpekCalc and XAAMDI filter data with an additional 7 µm of tungsten added to approximate the deposit on the exit window. The simulated spectra were also adjusted for the distance between the X-ray tube and sensor and for the EC used during measurements.

SpekCalc calculates spectra at a distance of 1000 mm from the X-ray tube focus, and the intensity decreases geometrically such that

$$C_{actual} = C_{SpekCalc} \left(\frac{1000}{d} \right)^2$$

where d is the distance between focus and measurement in mm. The EC affects the X-ray intensity linearly, such that a 100 mA EC causes 100 times more photons to be emitted than a 1 mA EC.

Finally, the mean and total photon energy was calculated (presented in table 2.2) and used to find the diode sensitivity for photons of a certain energy. The results are presented in fig. 2.10.

Verification

Since the sensitivity was calculated using measured values from only the topmost diode, S_0, the measured values from S_1 - S_3 could be used to validate the accuracy of the simulation. For each measured beam quality new spectra were simulated with added filtration corresponding to the copper filters between each diode in the sensor stack. Those spectra were then used to find the expected current from each of the three bottom diodes through

$$I_S = \sum_{i=1}^{N} S(E_i) \cdot C_i$$

where $S(E)$ is the diode sensitivity to photons of energy E. The gaps in the diode sensitivity were filled by a logarithmic interpolation in the same way as was done with the filtration in section 2.2.2.

The results are shown in fig. 2.11. The difference between measured and simulated values for each diode are also listed in table 2.3.

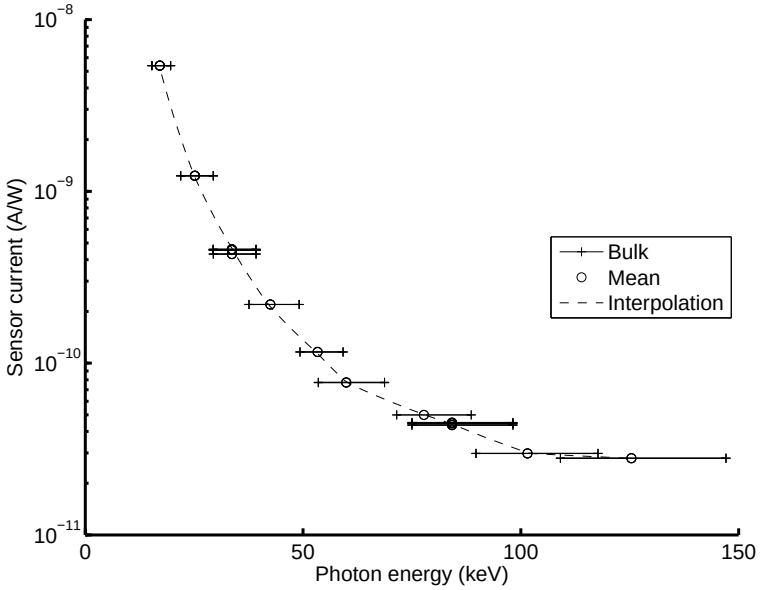

Figure 2.10: Sensitivity of the topmost sensor diode in the Unfors X2 stack as a function of incident photon energy.

From the figures one can see that the general trend is followed closely. The table displays rather large percentual errors – much larger than what was hoped for.

Beam quality see table 2.2	S_0 $[A/W]$	[%]	S_1 $[A/W]$	[%]	S_2 $[A/W]$	[%]	S_3 $[A/W]$	[%]
1	$2 \cdot 10^{-3}$	7	$2 \cdot 10^{-7}$	$\star$	$7 \cdot 10^{-16}$	$\star$	$2 \cdot 10^{-8}$	$\star$
2	$1 \cdot 10^{-3}$	17	$2 \cdot 10^{-5}$	32	$6 \cdot 10^{-8}$	$\star$	$1 \cdot 10^{-8}$	$\star$
3	$3 \cdot 10^{-4}$	12	$5 \cdot 10^{-5}$	16	$2 \cdot 10^{-6}$	29	$8 \cdot 10^{-8}$	$\star$
4	$4 \cdot 10^{-5}$	3	$2 \cdot 10^{-6}$	0	$2 \cdot 10^{-6}$	4	$4 \cdot 10^{-7}$	14
5	$1 \cdot 10^{-5}$	2	$1 \cdot 10^{-5}$	3	$8 \cdot 10^{-6}$	6	$6 \cdot 10^{-7}$	3
6	$4 \cdot 10^{-5}$	9	$2 \cdot 10^{-5}$	7	$6 \cdot 10^{-6}$	5	$1 \cdot 10^{-6}$	4
7	$4 \cdot 10^{-6}$	1	$5 \cdot 10^{-6}$	2	$6 \cdot 10^{-6}$	4	$7 \cdot 10^{-6}$	8
8	$5 \cdot 10^{-6}$	2	$1 \cdot 10^{-5}$	6	$1 \cdot 10^{-5}$	8	$1 \cdot 10^{-5}$	12
9	$2 \cdot 10^{-5}$	8	$3 \cdot 10^{-6}$	2	$2 \cdot 10^{-6}$	1	$6 \cdot 10^{-6}$	6
10	$7 \cdot 10^{-7}$	0	$7 \cdot 10^{-6}$	4	$4 \cdot 10^{-7}$	0	$6 \cdot 10^{-6}$	6

$\star$: Current was too low for reliable measurements to be made.

Table 2.3: Average absolute and percentual error between measurements and simulation of diode currents for the beam qualities listed in Table 2.2.

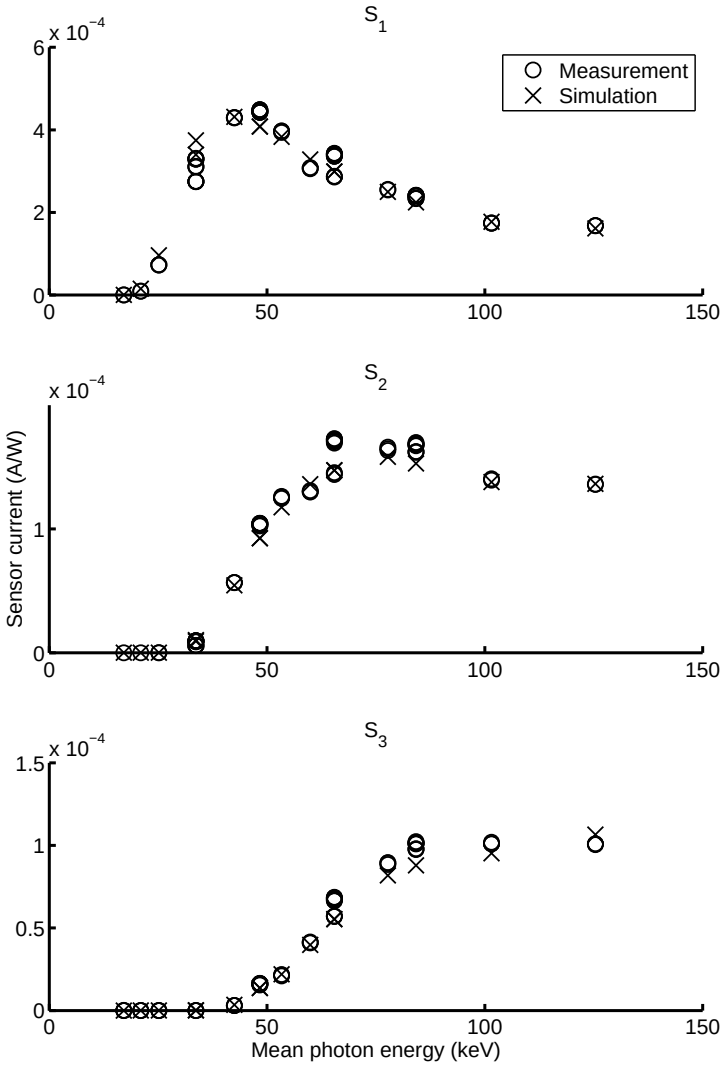

Figure 2.11: Measured and simulated currents for the three bottommost diodes in the sensor stack.

Chapter 3

Development of algorithms

3.1 Adaptive programming methods

In the previous chapter, a model of the X-ray measurement setup, including X-ray generator, filtration and detector was constructed. This model (M) is used in this chapter to generate `data points` in the form of input-output pairs $(Y, \boldsymbol{x})$ where $\boldsymbol{x} = M(Y)$ is the model output for input Y.

The adaptive programming methods will then be used to create algorithms which make a `prediction`

$$\hat{Y} = f(\boldsymbol{x})$$

of the model input.

Although the model input consists of several parameters, Y will refer only to the acceleration voltage unless otherwise stated. The aim is to get an accurate prediction of AV regardless of the other parameters.

This chapter is written with reference to [14, 15] for information on artificial neural networks and to [16] for information on linear genetic programming.

3.1.1 Artificial Neural Networks

Artificial neural networks (`ANN`) is a modelling method inspired by the human brain. Figure 3.1 shows a typical `feed-forward` ANN. The network is built in layers of one or more neuron. In the network of the figure there are two neurons in the `input layer` (the leftmost column) and one in the `output layer` (the rightmost column). Between those are a number of neurons in two `hidden layers`. Feed-forward means that each neuron takes input from every neuron in the layer before it and sends its output to every neuron in the next layer. There is no connection between neurons within the same layer.

All input to the network is passed to the neurons in the input layer and the output from the network is read from the neurons in the output layer. All other neurons are inaccessible from outside the network.

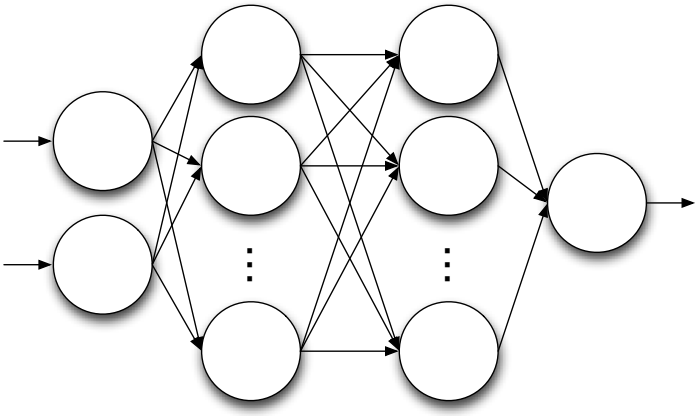

Figure 3.1: A typical feed-forward artificial neural network with two input neurons, two layers of hidden neurons and one output neurons.

The original ANN neuron was proposed by McCulloch and Pitts in 1943. The neuron assigns a weight to each input and outputs a binary signal if their sum exceeds a certain threshold level.

In other words, the output of neuron i is given by

$$v_i = g\left(\sum_{j=1}^{m} w_{ij} v_j - \mu_i\right)$$

where v_j and w_{ij} are the value and weight of input j respectively and μ_i is the neuron's threshold level.

g is called the activation function and is given in its simplest form by

$$g(b_i) = \begin{cases} -1, & b_i \leq 0 \\ 1, & b_i > 0 \end{cases}$$

The activation function governs the network behavior during training. This activation function makes the network predictable but it is prone to make the training get stuck at false solutions. It is also discontinuous, which adds further difficulties to training. A common variation to the activation function:

$$g(b_i) = \tanh(\beta b_i)$$

where β is the noise parameter is continuous, differentiable and able to avoid some false solutions. It also allows for non-binary output.

The threshold level μ_i can be expressed as an extra input $v_0 = -1$ with weight $w_{i0} = \mu_i$. By adding this input to every neuron in the network, the equations become more consistent, and this convention will be used from here on.

The ANN can thus be defined by its neuron configuration, the activation function and the matrix of weights. Training the network then becomes a matter of adjusting the weight matrices $\boldsymbol{W}$ so as to minimize the error

$$|Y_i - \hat{Y}_i| = |Y_i - Net(\boldsymbol{W}, x_i)|$$

where (Y_i, x_i) is a input-output pair of the X-ray measurement model.

Backpropagation

Backwards propagation of errors, or `backpropagation` for short is an ANN training method based on the gradient descent method.

In the following algorithm description, the superscript m indicates a neuron or value belonging to layer $m = 1, \ldots, M$ of the network.

First, go through each layer $m = 1, \ldots, M$ of the network and calculate the output of each neuron normally.

$$v_i^m = g(b_i^m) = g(\sum_j w_{ij}^m v_j^{m-1})$$

Next, calculate the errors or the neurons in the output layer according to

$$\delta_i^M = g'(b_i^M)(y_i - v_i^M)$$

where y_i is the expected output of neuron i.

Calculate the errors in the preceding layers, $m = M - 1, \ldots, 1$

$$\delta_i^{m-1} = g'(b_i^{m-1}) \sum_j w_{ij}^m \delta_j^m$$

And finally, use the errors to update the weights of the network.

$$w_{ij}^m := w_{ij}^m + \eta \Delta w_{ij}^m = w_{ij}^m + \eta \delta_i^m v_j^{m-1}$$

where η is the step size or `learning rate`.

In order to prevent the training to get stuck on a local minimum, a fraction of the previous Δw is added to the new in each iteration, i.e the assignment

$$\Delta w_{ij}^m(t) := \Delta w_{ij}^m(t) + \alpha \Delta w_{ij}^m(t-1)$$

is performed before updating the weights. α is called the `momentum parameter`.

3.1.2 Linear Genetic Programming

Basic genetic algorithm

A very basic genetic algorithm is what's affectionately called the 30-monkeys-in-a-bus algorithm.

Imagine a bus filled with monkeys. Two of the monkeys are randomly chosen to drive the bus together for ten minutes. Perhaps one takes the steering wheel and one the pedals. When the ten minutes are up, two new monkeys are chosen to drive. Once all monkeys have driven the bus in some pairing, the pair which crashed the bus into a tree the least number of times are allowed to mate with each other, and have two children. Those children get to replace the two monkeys who crashed the buss the most, who are thrown off. Pushing the theory of evolution to its extremes, after enough generations the buss should be filled with monkeys who are all very good bus drivers [17].

While overly simplified, the 30-monkeys-in-a-bus algorithm explains the idea behind the genetic algorithm used in this study.

In this basic genetic algorithm there is a `population` of `individuals` who each have a number of `chromosomes`. Those chromosomes can be for example a numerical value encoded in binary form.

The fitness of each individual is calculated by decoding the chromosomes and applying the decoded values to the problem to be solved. If the problem is to find the maximum of a function, the fitness value might be the function value at the point given by the chromosomes.

Once all individuals of the population has been thus evaluated and assigned a fitness value, the selection process begins. A number of individuals are randomly selected from the population and pitted against each other. The winner can be determined either randomly, with each contestant's chance of winning adjusted according to their fitness value, or via a tournament. In tournament selection the contestants are sorted and the best individual is picked with some probability. If it's not picked, the second best is picked with the same probability and so on. This probability is called the `tournament selection parameter`.

When two individuals have been chosen in this way, they form two new individuals for the next generation. In most cases the offspring will be exact duplicates of the parents but randomly, with a certain probability, crossover is performed. If crossover happens, the offspring get a combination of the chromosomes of the parents.

The crossover strategy used in this study is the following: For parents A and B with chromosomes $a_1, a_2, \cdots, a_m$ and $b_1, b_2, \cdots, b_n$ respectively, randomly choose two points $\alpha, \beta : 1 \leq \alpha < \beta \leq m$ and two points $\gamma, \delta : 1 \leq \gamma < \delta \leq n$. Then split the chromosome chains at those points and exchange the middle parts, so that the first offspring gets the chromosomes $a_1, \cdots, a_\alpha, b_{\gamma+1}, \cdots, b_\delta, a_{\beta+1}, \cdots, a_m$ and the second offspring the chromosomes $b_1, \cdots, b_\gamma, a_{\alpha+1}, \cdots, a_\beta, b_{\delta+1}, \cdots, b_n$.

Finally, the offspring are subject to random mutation. I.e. with some probability a random bit in one chromosome is flipped or similar depending on the chromosome structure.

The selection, crossing and mutation continues until a new population has been formed. The old population is then deleted and the new one takes its place for the next generation of the algorithm.

The point of the tournament selection parameter and the random mutations is to keep

the diversity of the population high. A high diversity decreases the risk of getting stuck in a local optimum. However, in order not to accidentally throw away a perfect – or at least a very good – chromosome configuration, a few exact copies of the best individual from the previous generation is inserted into every new generation.

Linear genetic programming

What differentiates linear genetic programming (`LGP`) from the basic genetic algorithm is the chromosomes.

In LGP the individuals are computer programs and each chromosome is a computing instruction. The chromosomes used in this study each have four parts: a target, an operation and two operands.

Execution of the instructions means taking the two operands, performing the operation on them and storing the result in the target. The target must be one of the program's storage `registers`. The operands may be a register, an input parameter of a predefined constant. The operation is one of several chosen mathematical operations. No branching or comparing operations were used in this study.

Evaluation of the individual's fitness is performed by executing each instruction sequentially, and then reading one or more values from the program's registers.

Mutation of the instructions are performed by replacing the target, the operation or one operand with a new one picked at random in such a way that the instruction is still valid.

Besides those modifications, the linear genetic programming algorithm is the same as the basic genetic algorithm.

3.2 Evaluation of methods

For measuring fitness, an `evaluation set` of 100 data points was built with input values chosen randomly and uniformly distributed in the following ranges:

Acceleration voltage (AV):	40–150 kV
Electron current (EC):	100–500 mA
Measurement distance:	100–500 mm
Aluminium filter:	3–5 mm
Copper filter:	0–1 mm

Those ranges are chosen to match the current specification of the X2 platform.

The predictions from the generated algorithms was used to calculate the percentual root mean square error:

$$E_{\mathrm{RMS}} = 100 \cdot \sqrt{\frac{1}{100}\sum_{i=1}^{100}\left(\frac{|\hat{Y}_i - Y_i|}{Y_i}\right)^2}$$

Configuration	Input neurons	Input configuration
1	4	$[S_0\, S_1\, S_2\, S_3]$
2	3	$\left[\frac{S_1}{S_0}\ \frac{S_2}{S_0}\ \frac{S_3}{S_0}\right]$
3	6	$\left[\frac{S_1}{S_0}\ \frac{S_2}{S_0}\ \frac{S_3}{S_0}\ \frac{S_2}{S_1}\ \frac{S_3}{S_1}\ \frac{S_3}{S_2}\right]$
4	6	$\left[\frac{S_1}{S_0}\ \frac{S_2}{S_0}\ \frac{S_3}{S_0}\ \left(\frac{S_1}{S_0}\right)^2\ \left(\frac{S_2}{S_0}\right)^2\ \left(\frac{S_3}{S_0}\right)^2\right]$
5	9	$\left[\frac{S_1}{S_0}\ \frac{S_2}{S_0}\ \frac{S_3}{S_0}\ \frac{S_1}{S_0}.\frac{S_2}{S_0}\ \frac{S_1}{S_0}.\frac{S_3}{S_0}\ \frac{S_2}{S_0}.\frac{S_3}{S_0}\ \left(\frac{S_1}{S_0}\right)^2\ \left(\frac{S_2}{S_0}\right)^2\ \left(\frac{S_3}{S_0}\right)^2\right]$
6	6	$\left[\frac{S_1}{S_0}\ \frac{S_2}{S_0}\ \frac{S_3}{S_0}\ \sqrt{\frac{S_1}{S_0}}\ \sqrt{\frac{S_2}{S_0}}\ \sqrt{\frac{S_3}{S_0}}\right]$

Table 3.1: Input configurations used for the neural networks.

where $\hat{Y}_i$ is the prediction and Y_i is the model input for data point i.

The evaluation set is held separate from the `training set`, which is the data points used by the adaptive methods to form the algorithms. Keeping the two sets separate helps to avoid `overfitting`, i.e. adapting the algorithm very closely to the training data at the cost of accuracy when it comes to previously unencountered data points.

Finally, the finished measurement functions should be evaluated with respect to:

- Accuracy
- Robustness
- Speed of execution
- Ease of calibration

3.2.1 Neural Networks

First attempt

A feed-forward artificial neural network with 200 output neurons was created. The network was setup with three hidden layers, each with 100 neurons. The activation function of each neuron was $g(b) = \tanh(0.1b)$ and initial weights were assigned randomly in the range $[-1, 1]$.

The output of the network was interpreted by finding the index of the output neuron with the highest output value. I.e. if the fifth node had the highest output value, AV was assumed to be 5 kV.

Several configurations for the network input were attempted with different functions of sensor currents. Those are listed in table 3.1.

In the model, the sensor currents vary with AV, EC, filtration and the distance between the X-ray source and the sensor. Therefore it is difficult to predict any single one of those parameters from a single current or linear combination of currents. The first configuration listed in table 3.1 was therefore expected to yield predictions with low

Configuration	E_{RMS}	
	Mean	Standard deviation
1	26	8
2	14	2
3	11	1
4	15	2
5	13	1

Table 3.2: Percentual root mean square errors of the first network attempted.

accuracy.

Varying AV and filtration also changes the relationship between currents. For example, the quotient $\frac{S_1}{S_0}$ increases with AV since more photons have higher energy and penetrate deeper into the sensor. This motivates the second and third attempted configuration. The last configurations use some arbitrarily chosen orders and functions of fractions.

Each configuration was trained for 20000 iterations with a training data set of 100 randomly generated measurements. The training set was generated with the same parameters as the evaluation set. The learning rate was set to $\eta = 0.0001$ and the momentum parameter $\alpha = 0.9$.

Five training runs were made for each configuration and the mean E_{RMS} was found and is presented in table 3.2. Figure 3.2 shows the network's predictions of the validation data set at the end of one training run with input configuration 1.

This first attempt shows that ANNs can be used to predict AV from sensor currents. Adding more neurons to the layers did not increase the network's performance. Interestingly, having a larger training set actually hindered training of the network, which might be a sign of overfitting. However, adding a fourth hidden layer with 100 neurons increased the accuracy greatly - almost halving the E_{RMS}. Unfortunately, the integer output of this network brings an inherent error of up to 2.5 %. This inherent error and the signs of overfitting motivated constructing a new network rather than attempting to further optimize this one.

Second attempt

A new feed-forward ANN was constructed, this time with a single output neuron. Otherwise, it was exactly like the network in the first attempt.

The prediction from this network was formed by multiplying the output by 200. The network was trained in the same way as the previous one, with the same learning rate and momentum parameter.

Like before, five training runs were made for each configuration and the mean E_{RMS} are shown in table 3.3. Figure 3.3 shows the prediction of the evaluation set from the end of one training run with input configuration 3.

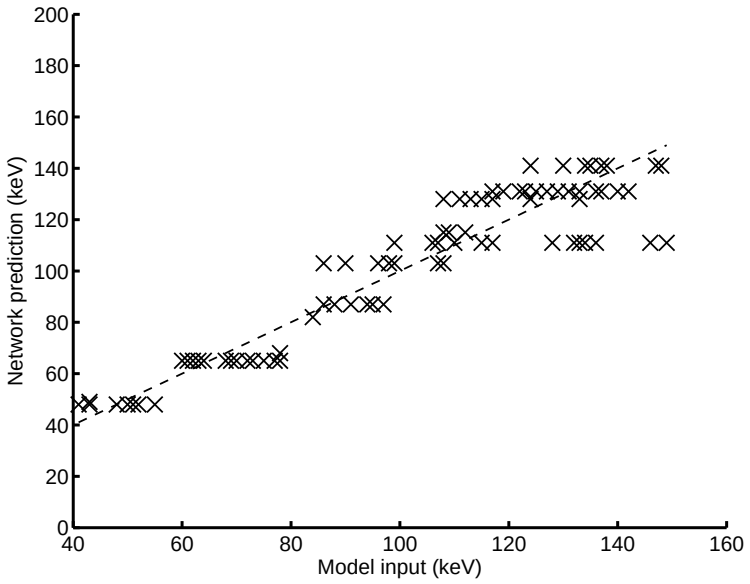

Figure 3.2: Network prediction $\hat{Y}$ compared to model input Y of the evaluation set in the first ANN attempt with input configuration 1. The E_{RMS} is 10.4 %. Note the discrete steps of the prediction. Those are probably an effect of the indexed output evaluation method.

This network showed better performance than the first attempt and, unlike the previous one, was to still visibly improving by the end of the 20000 training iterations.

Like the first network, adding a fourth hidden layer with 100 neurons approximately doubled the accuracy. Unlike the first network, increasing the size of the training set also increased accuracy. With a training set size of 1000 data points, E_{RMS} of less than 2 % were reached in several runs.

It is reasonable to believe that training the network for a longer time would also increase performance, but no attempts of this were made.

3.2.2 Linear Genetic Programming

An implementation of the LGP algorithm described above was set up in MATLAB. Each individual had eight registers r_0-r_3, and the operands were the registers, the four inputs S_0-S_3 from the model and three constants, 1, -1 and 3. Before execution, the registers were loaded with the input values, $r_0 = S_0, \cdots, r_3 = S_3$.

Available operations were a subset of:

Configuration	E_{RMS}	
	Mean	Standard deviation
1	19	0
2	12	4
3	6	1
4	6	1
5	6	0

Table 3.3: Percentual root mean square errors of the second network attempted. Performance is almost twice as high as in the first attempt for some input configurations.

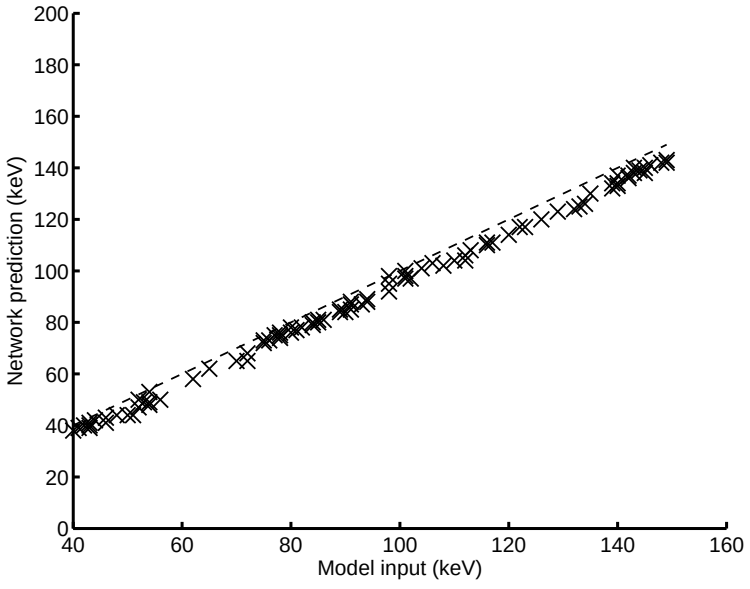

Figure 3.3: Network prediction $\hat{Y}$ compared to model input Y of the evaluation set in the secod ANN attempt with input configuration 3. The E_{RMS} is 2.1 %.

Addition	$r_i = o_{i,1} + o_{i,2}$
Subtraction	$r_i = o_{i,1} - o_{i,2}$
Multiplication	$r_i = o_{i,1} \cdot o_{i,2}$
Division	$r_i = o_{i,1}/o_{i,2}$
Exponentiation	$r_i = o_{i,1}^{o_{i,2}}$
Square root	$r_i = \sqrt{o_{i,1}}$

Fitness was evaluated by running the program for each of the data points in the training set. A least squares fit was then made of a number of the individual's registers to 100 random points from the training set. This fit was used to make predictions from the registers for the rest of the training set and the fitness was calculated from those.

In other words, for a subset L of the training set input-output pairs, register values were obtained by running the program $\boldsymbol{r}_i = f(\boldsymbol{x}_i)$ and a vector $\boldsymbol{\beta}$ was found such that

$$\sum_{i \in L} (\boldsymbol{r}_i \boldsymbol{\beta} - Y_i)^2$$

was minimized. The predictions were then calculated through

$$\hat{Y}_i = \boldsymbol{r}_i \boldsymbol{\beta} = f(\boldsymbol{x}_i)\boldsymbol{\beta}$$

The population size was set to 100 individuals, the crossover probability to 0.2 and the mutation probability to 0.04. Selection was made using a tournament selection of size 5 with tournament selection parameter 0.75. Five copies of the best individual were always inserted into the next generation.

Several different sets of available operations and different number of output registers were attempted and five complete runs of 1000 generations of the genetic algorithm was made for each configuration.

The configuration and the resulting mean E_{RMS} for the evaluation set are presented in table 3.4.

Accuracy-wise, LGP yields some very good results. A few runs with configuration 7 yielded root mean square errors of less than one percent for the evaluation set. The generated algorithms are also easy to calibrate, which is a matter of finding $\boldsymbol{\beta}$ from measurements using the least squares method. The accuracy of the calibration depends roughly on the number of measurements used.

Figure 3.5 shows one equation proposed by the LGP algorithm in configuration 7. This contains a large number of exponential operations, which might be too slow to perform on the embedded processor of the X2 base unit. A noteworthy fact is that this equation contains no reference to S_0. In fact, several of the equations from the LGP algorithm were of three or even two variables.

A much simpler function is proposed in configuration 4.

$$\hat{Y} = \beta_1 \frac{S_1^2 S_3^3}{S_2^5} + \beta_2 \frac{S_1^4 S_3^6}{S_2^{10}} + \beta_3 \frac{S_1^2 S_3^2}{S_2^4} + \beta_4 \frac{S_1 S_3}{S_2}$$

#	Operators	Outputs	E_{RMS}	
			Mean	Standard deviation
1	$r_j + r_k, r_j - r_k, r_j * r_k, r_j/r_k, r_j^{r_k}, \sqrt{r_j}$	4	4	2
2	$r_j + r_k, r_j - r_k, r_j * r_k, r_j/r_k, r_j^{r_k}, \sqrt{r_j}$	2	7	1
3	$r_j + r_k, r_j - r_k, r_j * r_k, r_j/r_k, r_j^{r_k}, \sqrt{r_j}$	1	13	4
4	$r_j * r_k, r_j/r_k$	4	2	0.2
5	$r_j * r_k, r_j/r_k$	2	7	2
6	$r_j * r_k, r_j/r_k$	1	14	1
7	$r_j * r_k, r_j/r_k, r_j^{r_k}$	4	1	0.3
8	$r_j * r_k, r_j/r_k, r_j^{r_k}$	2	6	0.9
9	$r_j * r_k, r_j/r_k, r_j^{r_k}$	1	9	3

Table 3.4: Input configurations and percentual root mean square error of the different attempted configurations for the LGP algorithm.

This function gives an E_{RMS} of 2.5 % for the validation set but contains very high orders of the input variables. This means the gradients are large, and small variations such as noise have large effects on the prediction. Note that this function also does not depend on S_0.

A more successful run of configuration 4 proposed the following function:

$$\hat{Y} = \beta_1 \frac{S_3^3}{S_2^3} + \beta_2 \frac{S_3}{S_2} + \beta_3 \frac{S_3^2}{S_1 S_2} + \beta_4$$

This function, which gives an E_{RMS} of 1.4 % for the validation set, has small gradients, high accuracy, is easy to calibrate and executes quickly assuming multiplication and division is fast. Its predictions for the validation set are presented in fig. 3.4.

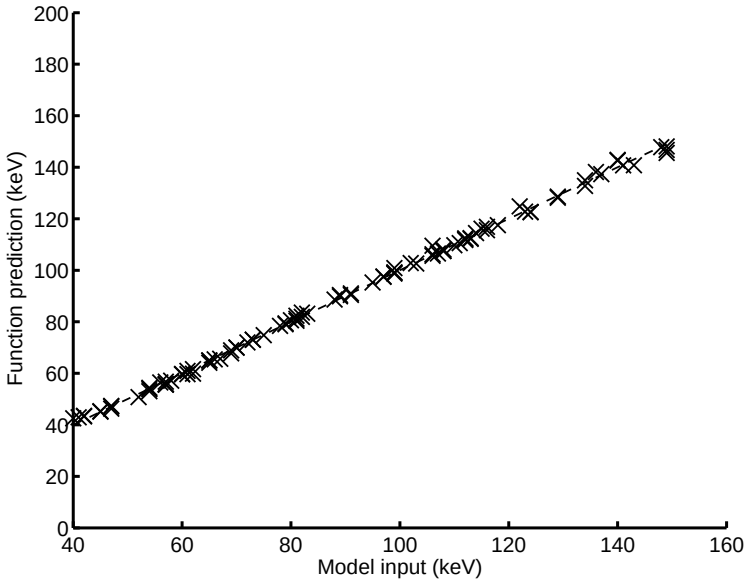

Figure 3.4: Predictions of the validation set by a function proposed by the LGP algorithm in configuration 4. E_{RMS} is 1.4 %.

$$
\begin{aligned}
\hat{Y} = \\
&\beta_1 27^{\frac{S_3}{S_2}} \left(\frac{S_3}{S_2}\right)^{1+\frac{3S_3}{S_2}} \left(\left(\frac{S_1 S_3}{S_2^2}\right)^{27^{\frac{S_3}{S_2}} \left(\frac{S_3}{S_2}\right)^{\frac{3S_3}{S_2}}} \right)^{27^{\frac{S_3}{S_2}} \left(\frac{S_3}{S_2}\right)^{\frac{3S_3}{S_2}}} \\
&+\beta_2 \left(\left(\frac{S_1 S_3}{S_2^2}\right)^{27^{\frac{S_3}{S_2}} \left(\frac{S_3}{S_2}\right)^{\frac{3S_3}{S_2}}} \right)^{27^{\frac{S_3}{S_2}} \left(\frac{S_3}{S_2}\right)^{\frac{3S_3}{S_2}}} \\
&+\beta_3 \left(\left(\frac{S_1 S_3}{S_2^2}\right)^{27^{\frac{S_3}{S_2}} \left(\frac{S_3}{S_2}\right)^{\frac{3S_3}{S_2}}} \right)^{\left(\left(\frac{S_3}{S_2}\right)^{-\frac{S_3}{S_2}} \right)^{\frac{S_3}{S_2}}} \\
&+\beta_4 27^{\frac{S_3}{S_2}} \left(\frac{S_3}{S_2}\right)^{\frac{3S_3}{S_2}}
\end{aligned}
$$

Figure 3.5: A function for measuring AV from the diode currents of an X2 R/F sensor as proposed by the LGP algorithm with configuration 7. This equation gives an E_{RMS} of 1.05 % for the evaluation set for some $\boldsymbol{\beta}$. Interestingly, this function has no dependence on S_0.

Chapter 4

Discussion and conclusion

Summary

In this study a model for an entire X-ray measurement setup has been proposed, developed and evaluated. The finished model follows the criteria on input and output and delivers the expected output with less than 10 % error in most of the input range, which is a bit larger than what was hoped for. The errors are believed to mainly be caused by errors in the measurements of the diode sensitivity.

Further, two adaptive methods have been attempted to solve the problem of finding the X-ray tube acceleration voltage from current measurements in this simulation of the X2 R/F sensor. Both the artificial neural networks trained through backpropagation as well as the linear genetic programming algorithm proved to be able to solve the problem.

Results

The similarities between the behavior of simulations and measurements imply that the model is well suited for research or development use, if not for production in its current state.

The first attempted ANN, which used an integer valued classification output, had low accuracy and showed tendencies of overfitting to the training data.

The accuracy of the second attempted ANN was rather good, but the network has tens of thousands of degrees of freedom. This makes it hard to calibrate for variations in the sensor diodes and thus unfeasible for a finished product. However, ANNs are very robust if trained correctly, and it is not unreasonable to believe that this robustness might overcome the need for calibration.

The LGP algorithm produced some functions with very high accuracy – in some cases better than the current specification of the X2[18]. Calibrating the functions was shown to be simple to a certain extent. Speed might be a problem, since the functions generated by LGP were often long and complex. The generated functions also had large gradients, making them sensitive to noise.

Possible improvements and future research

The X-ray measurement system model could easily be imporoved by more careful measurement of diode sensitivity which could increase its accuracy. Its simmilarity in behavior to measurements means it is already usable for research, and could be used for replacing actual measurement in the early stages of the development process of new sensor designs.

By adjusting the calculation of fitness value, the LGP algorithm could be made to favor simple functions which are possible to execute quickly on an embedded device. Likewise it could be made to favor functions with small gradients or conditional branching could be added to build entirely different kinds of functions. The last function described in this thesis proves the ability of the LGP to present functions which perform well according to all listed criteria.

I believe more can be learned about the modelled measurement system by studying the behavior of the LGP algorithm. One interesting behavior is the LGP solutions' tendency to ignore S_0. I believe this is caused by the magnitude of S_0 drowning the details contained in the other signals. An other interesting behavior is the very common occurance of $\frac{S_3}{S_2}$ which reminds of a different method for measuring acceleration voltage. In this method, an ionization chamber is used to measure the radioactive dose with two different filtrations[19]. The acceleration voltage is then calculated as a function of the quotient between the two measurements.

Bibliography

[1] Harjit Singh and Janet A. Neutze. *Radiology Fundamentals, Introduction to Imaging & Technology*. Fourth edition. Springer Science+Business Media, 2012.

[2] Perry Sprawls. *Physical Principles of Medical Imaging*. Second edition. Medical Physics Publishing Corporation, 1995.

[3] United Nations Scientific Committee on the Effects of Atomic Radiation. *Effect of Inonizing Radiation, Volume I*. Tech. rep. UNSCEAR, 2006.

[4] Kenneth S. Krane. *Introductory Nuclear Physics*. John Wiley & Sons, Inc., 1988. ISBN: 078-0-471-80553-3.

[5] Carl Nordling and Jonny Österman. *Physics Handbook for Science and Engineering*. Studentlitteratur, Lund, 2006.

[6] William R. Hendee, Geoffrey S. Ibbott, and Eric G. Hendee. *Radiation Therapy Physics*. Third Edition. John Wiley & Sons, Inc., 2005.

[7] International Organization for Standardization. *ISO 4037-1: X-ray and gamma reference radiation for calibrating dosemeters and doserate meters and for determining their response as a function of photon energy*. International Organization for Standardization, 1996.

[8] Glenn F. Knoll. *Radiation Detection and Measurement*. Third Edition. John Wiley & Sons, Inc., 2000. ISBN: 0-471-07338-5.

[9] G Poludniowski et al. "SpekCalc : a program to calculate photon spectra from tungsten anode x-ray tubes". In: *Physics in Medicine and Biology* 54 (2009), N433–N438.

[10] Gavin Poludniowski and Philip Evans. "Calculation of x-ray spectra emerging from an x-ray tube. Part I. Electron penetration characteristics in x-ray targets". In: *Medical Physics* 34 (2007), pp. 2164–2174.

[11] Gavin Poludniowski. "Calculation of x-ray spectra emerging from an x-ray tube. Part II. X-ray production and filtration in x-ray targets". In: *Medical Physics* 34 (2007), pp. 2175–2186.

[12] Amptek. *Complete X-Ray Spectrometer X-123*. URL: `http://www.amptek.com/x123.html`.

[13] National Institute for Standards and Technology. *Note on the X-Ray Attenuation Databases*. URL: `http://physics.nist.gov/PhysRefData/XrayNoteB.html`.

[14] Simon Haykin. *Neural Networks and Learning Machines*. Third Edition. Pearson Education Inc., 2009. ISBN: 978-0-13-129376-2.

[15] Bernhard Mehlig. *FFR135 Lecture Notes*. distributed in Artificial Neural Networks at Chalmers University of Technology, Gothenburg. 2011.

[16] Mattias Wahde. *Biologically Inspired Optimization Methods - An introduction*. WIT Press, 2008. ISBN: 978-1-84564-148-1.

[17] Peter Nordin and Johanna Wilde. *Humanoider: Självlärande robotar och artificiell intelligens*. Liber Förlag, 2003. ISBN: 9789147051915.

[18] *RaySafe X2 Specification*. Ray Safe. 2013.

[19] Béla Kári et al. *Physical foundations of non-invasive X-ray tube voltage measurement*. URL: `http://physics.nist.gov/PhysRefData/XrayNoteB.html`.

Much of the information on workings of X-ray machines, radiation and measurement devices has been provided through personal communication with – among others – Mattias Gustavsson, Fredrik Oskarson and Anders Fransson at Unfors RaySafe.

Appendix A

Source code listing

X-ray measurement system model

MeasurementViewer.m

```
function MeasurementViewer()
% A tool for viewing X2 measurements

  global measurementFile
  measurementFile = 'Measurements.mat';

  OpenControlWindow();


end

function controlWindow = OpenControlWindow()
  global measurementFile
  load(measurementFile);

  controlWindow = figure('Tag','ControlWindow');

  set(controlWindow, 'DeleteFcn', @CleanQuit);
  set(controlWindow, 'Position', [520, 80, 240, 60]);
  set(controlWindow, 'MenuBar', 'none');
  set(controlWindow, 'Name', 'Measurement Viewer', 'NumberTitle', 'off');

  % Close program
  controls.quitBtn = uicontrol('Style', 'Pushbutton', ...
    'String', 'Quit', ...
    'Callback', @CleanQuit, ...
    'Position', [20, 20, 60, 20]);
  % View old measurement
  controls.viewBtn = uicontrol('Style', 'Pushbutton', ...
    'String', 'View', ...
    'Callback', @ViewOld, ...
```

```
    'Position', [90, 20, 60, 20]);
  % Import new measurement
  controls.addBtn = uicontrol('Style', 'Pushbutton', ...
    'String', 'Add', ...
    'Callback', @LoadFile, ...
    'Position', [160, 20, 60, 20]);

  controls.plotWindows = [];
  controls.data = meas_data;
  set(controlWindow, 'UserData', controls);

end

function LoadFile(handler, ~)
  % Import data from a measurement file

  p = handler;
  p = get(p, 'Parent');
  controls = get(p, 'UserData');
  [fileName, pathName, filterIndex] = uigetfile('*.csv.2','Select file');
  if(fileName ~= 0)
    newData = importdata(strcat([pathName, fileName]), ',',130);
    d.filename = fileName;
    d.sensor = sscanf(newData.textdata{2,1}, '%*s %*s %d');
    d.kvp = sscanf(newData.textdata{6,1}, '%*s %d');
    d.ma = sscanf(newData.textdata{7,1}, '%*s %d');
    d.time = sscanf(newData.textdata{8,1}, '%*s %d');
    d.dist = sscanf(newData.textdata{9,1}, '%*s %d');
    d.filter = sscanf(newData.textdata{10,1}, '%*s %s');
    d.kvmeas = sscanf(newData.textdata{105,1}, '%*s %f',1);
    d.data = newData.data;

    d.s0 = 0;
    d.s1 = 0;
    d.s2 = 0;
    d.s3 = 0;

    d.index = 0;

    newPlot = ViewFile(d,p);

    controls.plotWindows = [controls.plotWindows newPlot];
    set(p, 'UserData', controls);

  end
end

function ViewOld(handler, ~)
  % View measurement

  p = handler;
```

```
  p = get(p, 'Parent');
  controls = get(p, 'UserData');

  % Select measurement to view from list
  lstr = num2str([1:length(controls.data)]');
  lstr = '';
  for i=1:length(controls.data)
    lstr{i} = strcat([num2str(i) ':' num2str(controls.data(i).kvp) 'kVp ', controls.dat
  end
  [Selection, ok] = listdlg('ListString',lstr);

  if (ok == 0)
    return;
  end

  d = controls.data(Selection);
  d.index = Selection;

  newPlot = ViewFile(d,p);

  controls.plotWindows = [controls.plotWindows newPlot];
  set(p, 'UserData', controls);
end

function newPlot = ViewFile(d,p)
  % Open up a window for viewing a measurement

  newPlot = figure('Tag', 'DataWindow','Units','Normalized');
  c.controls = p;

  c.data = d.data;
  c.time = d.time;
  c.d = d;
  c = AutoData(c);
  subplot(1,2,1);

  h = plot(d.data(:,5:8));
  c.fig = get(h(1),'Parent');

  set(c.fig, 'Position', [0.1 0.1 0.4 0.8])
  h = uipanel('Position', [0.55 0.1 0.4 0.8]);

  c.AutoBtn = uicontrol(h, 'Style', 'Pushbutton', ...
    'String', 'Auto', ...
    'Callback', @DoReAuto, ...
    'Position', [10, 20, 60, 20]);
  c.SaveBtn = uicontrol(h, 'Style', 'Pushbutton', ...
    'String', 'Save', ...
    'Callback', @SaveData, ...
    'Position', [80, 20, 60, 20]);
  c.DeleteBtn = uicontrol(h, 'Style', 'Pushbutton', ...
```

```
    'String', 'Delete', ...
    'Callback', @DeleteData, ...
    'Position', [150, 20, 60, 20]);

  c.Title = uicontrol(h, 'Style', 'Edit', ...
    'String', strcat(['kV:', num2str(d.kvp),...
    ' mA:', num2str(d.ma), ' Filter:' num2str(d.filter)]), ...
    'Position', [10, 60, 200, 30]);
  if(d.filename ~= 0)
    set(c.Title, 'String', d.filename);
  end

  uicontrol(h, 'Style', 'Text', ...
    'String', 'kV:', ...
    'Position', [0, 110, 60, 10]);
  c.kV = uicontrol(h, 'Style', 'Edit', ...
    'String', num2str(d.kvp), ...
    'Callback', @EditField, ...
    'Position', [40, 100, 60, 30]);
  uicontrol(h, 'Style', 'Text', ...
    'String', num2str(d.kvmeas), ...
    'Position', [110, 110, 60, 10]);

  uicontrol(h, 'Style', 'Text', ...
    'String', 'mA:', ...
    'Position', [0, 150, 60, 10]);
  c.mA = uicontrol(h, 'Style', 'Edit', ...
    'String', num2str(d.ma), ...
    'Callback', @EditField, ...
    'Position', [40, 140, 60, 30]);
  c.mAs1 = uicontrol(h, 'Style', 'Text', ...
    'String', ['*' num2str(d.time) '='], ...
    'Position', [110, 150, 40, 10]);
  c.mAs2 = uicontrol(h, 'Style', 'Edit', ...
    'String', num2str(d.time*d.ma/1000), ...
    'Callback', @EditField, ...
    'Position', [150, 140, 60, 30]);

  uicontrol(h, 'Style', 'Text', ...
    'String', 'filter:', ...
    'Position', [0, 190, 60, 10]);
  c.filter = uicontrol(h, 'Style', 'Edit', ...
    'String', d.filter, ...
    'Callback', @EditField, ...
    'Position', [40, 180, 60, 30]);
  uicontrol(h, 'Style', 'Text', ...
    'String', 'Dist:', ...
    'Position', [110, 190, 60, 10]);
  c.dist = uicontrol(h, 'Style', 'Edit', ...
    'String', num2str(d.dist), ...
    'Callback', @EditField, ...
```

```
    'Position', [150, 180, 60, 30]);

c.s0en = uicontrol(h, 'Style', 'RadioButton', ...
    'String', 'S0', ...
    'Value', 1, ...
    'Callback', @DoRedraw, ...
    'Position', [10, 280, 100, 30]);
c.s0v = uicontrol(h, 'Style', 'Text', ...
    'String', num2str(d.s0),...
    'Position', [50, 283, 50, 20]);
c.s1en = uicontrol(h, 'Style', 'RadioButton', ...
    'String', 'S1', ...
    'Value', 1, ...
    'Callback', @DoRedraw, ...
    'Position', [10, 260, 100, 30]);
c.s1v = uicontrol(h, 'Style', 'Text', ...
    'String', num2str(d.s1),...
    'Position', [50, 263, 50, 20]);
c.s2en = uicontrol(h, 'Style', 'RadioButton', ...
    'String', 'S2', ...
    'Value', 1, ...
    'Callback', @DoRedraw, ...
    'Position', [10, 240, 100, 30]);
c.s2v = uicontrol(h, 'Style', 'Text', ...
    'String', num2str(d.s2),...
    'Position', [50, 243, 50, 20]);
c.s3en = uicontrol(h, 'Style', 'RadioButton', ...
    'String', 'S3', ...
    'Value', 1, ...
    'Callback', @DoRedraw, ...
    'Position', [10, 220, 100, 30]);
c.s3v = uicontrol(h, 'Style', 'Text', ...
    'String', num2str(d.s3),...
    'Position', [50, 223, 50, 20]);

c.saveState = uicontrol(h, 'Style', 'Text', ...
    'String', 'SAVED', ...
    'Position', [50, 310, 100, 20]);

if(c.d.index == 0)
  set(c.saveState, 'String', 'Not␣in␣data')
end

set(c.fig, 'Tag', 'figure');
set(c.fig, 'UserData', newplot);
c.selectedLine = 0;
c = DrawPlot(c);

c.savedPos = [0  0];

set(newPlot, 'UserData', c);
```

```
  set(h, 'UserData',newPlot);

  set(newPlot, 'WindowButtonMotionFcn', @MoveMouse);
  set(newPlot, 'WindowButtonUpFcn', {@SelectLine, 0});

  RedrawPlot(c);

end

function SaveData(handle, ~)
  % Save changes to measurement
  global measurementFile
  c = get(handle, 'Parent');
  d = get(c, 'UserData');
  c = get(d, 'UserData');

  n.filename = c.d.filename;
  n.sensor = c.d.sensor;
  n.kvp = c.d.kvp;
  n.ma = c.d.ma;
  n.time = c.d.time;
  n.dist = c.d.dist;
  n.filter = c.d.filter;
  n.kvmeas = c.d.kvmeas;
  n.data = c.d.data;
  n.s0 = c.mean0;
  n.s1 = c.mean1;
  n.s2 = c.mean2;
  n.s3 = c.mean3;

  ctrl = c.controls;
  controls = get(ctrl, 'UserData');
  meas_data = controls.data;

  if(c.d.index ~= 0)
    meas_data(c.d.index) = n;
  else
    meas_data = [meas_data n];
  end
  set(c.saveState, 'String', 'Saved');
  controls.data = meas_data;
  set(ctrl, 'UserData', controls);
  save(measurementFile,'meas_data');

end

function DeleteData(handle, ~)
  % Delete measurement from file

  global measurementFile
  c = get(handle, 'Parent');
```

```
  d = get(c, 'UserData');
  c = get(d, 'UserData');

  choice = questdlg('Are you sure you wish to delete the measurement?',...
    'Measurement deletion', ...
    'Yes','No','No');

  if(strcmp(choice, 'No') == 1)
    return;
  end

  ctrl = c.controls;
  controls = get(ctrl, 'UserData');
  meas_data = controls.data;

  if(c.d.index ~= 0)
    meas_data(c.d.index) = [];
    set(c.saveState, 'String', 'Deleted');
  else
  end
  controls.data = meas_data;
  set(ctrl, 'UserData', controls);
  save(measurementFile,'meas_data');
end

function DoReAuto(handle, ~)
  % Remake autoalign of trig and meanlines
  c = get(handle, 'Parent');
  d = get(c, 'UserData');
  c = get(d, 'UserData');
  c = AutoData(c);
  set(d, 'UserData', c);
  RedrawPlot(c);
  set(c.saveState, 'String', 'Changed');
end

function c = AutoData(c)
  % Automatically align trig and meanlines

  top = sort(c.data(:,5));
  top = top(end-10);
  c.trigl = find(c.data(:,5)> top*0.1,1);
  c.trigr = find(c.data(:,5)> top*0.1,1,'last');
  c.time = ((c.trigr-c.trigl)/1600)/10;

  c.mean0 = mean(c.data(c.trigl:c.trigr,5));
  c.mean1 = mean(c.data(c.trigl:c.trigr,6));
  c.mean2 = mean(c.data(c.trigl:c.trigr,7));
  c.mean3 = mean(c.data(c.trigl:c.trigr,8));
```

```
end

function DoRedraw(handle, ~)
  % Redraw everything
  c = get(handle, 'Parent');
  c = get(c, 'UserData');
  c = get(c, 'UserData');
  RedrawPlot(c);
end

function c = DrawPlot(c)
  % Draw currents

  data = c.data;
  fig = c.fig;

  top = max(max(c.data(:,5:8)));
  btm = min(min(c.data(:,5:8)));

  subplot(fig)

   %HitTest off = won't interfere with click and drag
  plots.ps0 = plot(fig, data(:,5),'HitTest','off');
  hold all
  plots.ps1 = plot(fig, data(:,6),'HitTest','off');
  plots.ps2 = plot(fig, data(:,7),'HitTest','off');
  plots.ps3 = plot(fig, data(:,8),'HitTest','off');
  hold off

  hold on
  % Draw meanlines
  plots.m0p = plot(fig, [c.trigl, c.trigr],[c.mean0, c.mean0],...
    'k--', 'LineWidth',2);
  plots.m1p = plot(fig, [c.trigl, c.trigr],[c.mean1, c.mean1],...
    'k--', 'LineWidth',2);
  plots.m2p = plot(fig, [c.trigl, c.trigr],[c.mean2, c.mean2],...
    'k--', 'LineWidth',2);
  plots.m3p = plot(fig, [c.trigl, c.trigr],[c.mean3, c.mean3],...
    'k--', 'LineWidth',2);

  % Draw triglines
  plots.tl = plot(fig, [c.trigl c.trigl],[btm,top],'k--','LineWidth',2);
  plots.tr = plot(fig, [c.trigr c.trigr],[btm,top],'k--','LineWidth',2);
  hold off

  % Make lines clickable
  set(plots.tl, 'ButtonDownFcn', {@SelectLine, 1})
  set(plots.tr, 'ButtonDownFcn', {@SelectLine, 2})
  set(plots.m0p, 'ButtonDownFcn', {@SelectLine, 3})
  set(plots.m1p, 'ButtonDownFcn', {@SelectLine, 4})
  set(plots.m2p, 'ButtonDownFcn', {@SelectLine, 5})
```

```
  set(plots.m3p, 'ButtonDownFcn', {@SelectLine, 6})

  c.plots = plots;
end

function RedrawPlot(c)
  % Redraw everyting

  % S0
  if(get(c.s0en,'Value') == 0)
    set(c.plots.ps0, 'Visible', 'off');
    set(c.plots.m0p, 'Visible', 'off');
  else
    set(c.plots.ps0, 'Visible', 'on');
    set(c.plots.m0p, 'Visible', 'on');
  end

  % S1
  if(get(c.s1en,'Value') == 0)
    set(c.plots.ps1, 'Visible', 'off');
    set(c.plots.m1p, 'Visible', 'off');
  else
    set(c.plots.ps1, 'Visible', 'on');
    set(c.plots.m1p, 'Visible', 'on');
  end

  % S2
  if(get(c.s2en,'Value') == 0)
    set(c.plots.ps2, 'Visible', 'off');
    set(c.plots.m2p, 'Visible', 'off');
  else
    set(c.plots.ps2, 'Visible', 'on');
    set(c.plots.m2p, 'Visible', 'on');
  end

  % S3
  if(get(c.s3en,'Value') == 0)
    set(c.plots.ps3, 'Visible', 'off');
    set(c.plots.m3p, 'Visible', 'off');
  else
    set(c.plots.ps3, 'Visible', 'on');
    set(c.plots.m3p, 'Visible', 'on');
  end

  set(c.fig, 'Xlim', [0, length(c.data(:,1))]);

  % Draw triglines and meanlines
  lines = [c.plots.tl, c.plots.tr, c.plots.m0p, c.plots.m1p, c.plots.m2p, c.plots.m3p];
  set(c.plots.m0p, 'Color','black');
  set(c.plots.m1p, 'Color','black');
  set(c.plots.m2p, 'Color','black');
```

```
    set(c.plots.m3p, 'Color','black');
    set(c.plots.tl, 'Color','black');
    set(c.plots.tr, 'Color','black');

    if(c.selectedLine ~= 0)
      set(lines(c.selectedLine), 'Color', 'magenta');
    end

    set(c.plots.tl, 'XData', [c.trigl c.trigl]);
    set(c.plots.tr, 'XData', [c.trigr c.trigr]);
    set(c.plots.m0p, 'YData', [c.mean0 c.mean0]);
    set(c.plots.m1p, 'YData', [c.mean1 c.mean1]);
    set(c.plots.m2p, 'YData', [c.mean2 c.mean2]);
    set(c.plots.m3p, 'YData', [c.mean3 c.mean3]);

    % Update display values
    set(c.mAs1, 'String', ['*' num2str(c.time) '=']);
    set(c.mAs2, 'String', num2str(c.time*c.d.ma));
    set(c.s0v, 'String', num2str(c.mean0));
    set(c.s1v, 'String', num2str(c.mean1));
    set(c.s2v, 'String', num2str(c.mean2));
    set(c.s3v, 'String', num2str(c.mean3));

    drawnow();
end

function SelectLine(handle, ~, line)
    % A trig- or meanline was clicked, select it and prepare for moving

    if(strcmp(get(handle, 'Tag'), 'DataWindow') == 1)
      g = handle;
    else
      g = get(handle, 'Parent');
      g = get(g, 'Parent');
    end
    c = get(g, 'UserData');

    c.selectedLine = line;
    mpos = get(gcf, 'CurrentPoint');
    c.savedPos = mpos(1,:);

    set(g, 'UserData', c);

    RedrawPlot(c);

end

function MoveMouse(handle, ~)
    % Move triglines when moving mouse (if a line is clicked)

    c = get(handle, 'UserData');
```

```
if(c.selectedLine == 0)
  return;
end

mpos = get(gcf, 'CurrentPoint');

lines = [c.plots.tl, c.plots.tr, c.plots.m0p, c.plots.m1p, c.plots.m2p, c.plots.m3p];

h2 = get(lines(c.selectedLine),'Parent');

if(c.selectedLine < 3)
  % Trig lines
  move = c.savedPos(1) - mpos(1,1);
  lim = get(h2, 'XLim');
  move = move *(lim(2));
  d = get(lines(c.selectedLine),'XData');
  d = d - move;
  set(lines(c.selectedLine),'XData',d);
else
  % Mean lines
  move = c.savedPos(2) - mpos(1,2);
  lim = get(h2, 'YLim');
  move = move *(lim(2)-lim(1));
  d = get(lines(c.selectedLine),'YData');
  d = d - move;
  set(lines(c.selectedLine),'YData',d);
end

c.savedPos = mpos(1,:);

% Update measurement
d = get(lines(1),'XData');
c.trigl = d(1);
d = get(lines(2),'XData');
c.trigr = d(1);
d = get(lines(3),'YData');
c.mean0 = d(1);
d = get(lines(4),'YData');
c.mean1 = d(1);
d = get(lines(5),'YData');
c.mean2 = d(1);
d = get(lines(6),'YData');
c.mean3 = d(1);

c.time = (c.trigr-c.trigl)/16000;

% Mark measurement as unsaved
set(c.saveState, 'String', 'Changed');

set(handle, 'UserData', c);
RedrawPlot(c);
```

```
end

function EditField(handle, ~)
  % A field was edited.

  g = get(handle, 'Parent');
  g = get(g, 'UserData');
  c = get(g, 'UserData');

  % Update measurement
  switch(handle)
    case c.mA
      c.d.ma = str2num(get(handle, 'String'));
    case c.mAs2
      mas = str2num(get(handle, 'String'));
      c.d.ma = mas/c.time;
      set(c.mA, 'String', num2str(c.d.ma));
    case c.kV
      c.d.kvp = str2num(get(handle, 'String'));
    case c.filter
      c.d.filter = get(handle, 'String');
    case c.dist
      c.d.dist = str2num(get(handle, 'String'));
  end

  % Mark measurement and unsaved
  set(c.saveState, 'String', 'Changed');

  set(g, 'UserData',c);
  RedrawPlot(c);
end

function CleanQuit(handler, ~)
  % Close all open windows and quit
    p = handler;
    if(strcmp(get(p,'Tag'),'ControlWindow') ~= true)
        p = get(p, 'Parent');
    end

    controls = get(p, 'UserData');
    for i = 1:length(controls.plotWindows)
      if(ishandle(controls.plotWindows(i)))
          delete(controls.plotWindows(i));
      end
    end

    delete(p)
end
```

FindSensitivity.m

```
function FindSensitivity()
  clear all
  clc
  close all
  set(0,'DefaultAxesFontSize',14)
  set(0,'DefaultAxesFontName','Helvetica')


  measurementFile = 'Measurements.mat';
  sensitivityFile = 'Diode.mat';

  % Choose plot type
  % 1: X axis in keV
  % 0: X axis in nm
  plotscale = 0;

  load(measurementFile);
  sensData = zeros(size(meas_data,2),4);

  for i = 1:size(meas_data,2) % For each measurement
    dd = meas_data(i);
    display(['Measurement ', num2str(i) ':']);
    display(['  Set     : ' num2str(dd.kvp), 'kV']);
    display(['  Meaured: ' num2str(dd.kvmeas), 'kV']);
    display(['  Filter : ', dd.filter]);



    % Find spektra and properties for measurement
    [sp, spd] = SimpleFilterSpectra(round(dd.kvp), dd.ma,...
      dd.filter, dd.dist, 0.007);
    meanCurrent = dd.s0/spd.photonEnergy;

    % This gives one data point...
    sensData(i,:) = [spd.mean spd.peakMin spd.peakMax meanCurrent];

    display(['  Mean energy: ', num2str(spd.mean), 'keV']);
    display(['  Current    : ', num2str(meanCurrent), 'A/W']);
  end


  if(plotscale == 0)
    lambda = @(x) 6.626e-34*2.998e8./(x*1.602e-19.*1e3).*1e9;
  else
    lambda = @(x) x;
  end

  f = figure('color', 'white');
```

```
% Plot bulk indicators
for i = 1:size(sensData,1)
  % To get a linear plot, change this 'loglog' to 'plot'
  h = loglog(lambda([sensData(i,2) sensData(i,3)]), ...
    [sensData(i,4) sensData(i,4)],'k-+');
  hold on
  h2 = plot(lambda(sensData(i,1)),sensData(i,4),'ko');

  % Right click a circle to see which measurement it corresponds to
  hcmenu = uicontextmenu;
  dd = meas_data(i);
  item1 = uimenu(hcmenu, 'label', ...
    [num2str(i) ':' num2str(dd.kvp), dd.filter]);
  set(h, 'uicontextmenu', hcmenu);
  set(h2, 'uicontextmenu', hcmenu);

  % This line makes the legend show the right things
  plot([0 0],[0 0],'k--','HitTest','off')
end

hold off

% Find unique mean energies (rounded to integer)
kvs = unique(round(sensData(:,1)))
ms = kvs;
ls = kvs;
rs = kvs;
% Average measurements for each point
for i = 1:length(kvs)
  inds = (round(sensData(:,1)) == kvs(i));
  ms(i) = mean(sensData(inds,4)); % Sensitivity
  ls(i) = mean(sensData(inds,2)); % Lower limit
  rs(i) = mean(sensData(inds,3)); % Upper limit
end

global diodeSens
diodeSens = [kvs, ms, ls, rs]

% Plot interpolation of diode sensitivity
hold on
x = linspace(1, 300, 200);
plot(lambda(x), interpolateSensitivity(x),'k--','HitTest','off');
hold off

if(plotscale == 0)
  xlabel('Photon wavelength (nm)');
else
  xlabel('Photon energy (keV)');
end
ylabel('Sensor current (A/W)');
set(gca,'fontSize',14,'lineWidth',2,'box','off', 'Layer', 'top')
```

```
  % Save the sensitivity
  save(sensitivityFile,'diodeSens')

end

function y = interpolateSensitivity(x)
  global diodeSens
  X = log(diodeSens(:,1));
  Y = log(diodeSens(:,2));
  x = log(x);

  y = interp1(X, Y, x, 'linear');
  y = exp(y);
end
```

SimpleFilterSpectra.m

```
function [spectrum data] = ...
  SimpleFilterSpectra(kV, mA, filter, mmair, tungsten)
  % Input
  %   kV: the kilovoltage setting
  %   mA: the amperage setting
  %   filter: the name of a filter
  %   mmair: distance between X-ray tube and sensor
  %   tungsten: Extra tungsten filtration
  % Output
  %   spektra: X-ray spectrum
  %   data: X-ray spectrum properties

  % Get basic spectrum
  unfiltered = GetXRaySpectrum(kV,mA);

  % Filter spectrum
  f = [mmair 0 0 0 0 0 0];
  if(nargin == 5)
    f(7) = tungsten;
  end
  spectrum = FilterXRaySpectrum(unfiltered, filter, f);

  % Recalculate for distance
  distanceFactor = (1000/mmair)^2;
  spectrum(:,2) = spectrum(:,2).*distanceFactor;

  % Calculate spectral properties
  data = GetSpectraParameters(spectrum);

end
```

GetXRaySpectrum.m

```
function spectra = GetXRaySpectrum(kV, mA)
% kV should be a multiple of 10 between 10 and 300
% Returns an unfiltered X-ray spectrum with 0.1 keV bins from 1 to 300 keV

spectraFile = 'XraySpectra.mat';

persistent spectras
if(isempty(spectras))
  load(spectraFile);
end

  kVmax = 300;
spectra = spectras(:,[1 kV-8]);
spectra(:,2) = spectra(:,2).*mA;

end
```

FilterXRaySpectrum.m

```
function specout = FilterXRaySpectrum(specin, filtername, filters)

% specin(:,1) = energies [keV
% specin(:,2) = # photons
% filtername = Name of a predefined filter
% filters = Extra filtration
% filters(1) thickness of air [mm]
% filters(2) thickness of al [mm]
% filters(3) thickness of cu [mm]
% filters(4) thickness of sn [mm]
% filters(5) thickness of pb [mm]
% filters(6) thickness of si [mm]
% finters(7) thickness of w [mm]

  numFilters = 5;

  namedFilter = zeros(1,numFilters);

  if(nargin == 2)
    if(ischar(filtername))
      filters = zeros(1,numFilters);
      namedFilter = FilterFromName(filtername);
    else
      filters = filtername;
    end
  end
  if(nargin == 3)
    namedFilter = FilterFromName(filtername);
  end

  if(length(filters) > length(namedFilter))
    namedFilter = [namedFilter zeros(1, length(filters)-length(namedFilter)) ];
  end

    filters = filters + namedFilter;
persistent air_xaamdi
if(isempty(air_xaamdi))
  air_xaamdi = importdata('FilterData/air_xaamdi.txt');
  air_xaamdi(6,1) = air_xaamdi(6,1) + 1e-10; % Remove K-edges
end

persistent cu_xaamdi
if(isempty(cu_xaamdi))
  cu_xaamdi = importdata('FilterData/cu_xaamdi.txt');
  cu_xaamdi(4,1) = cu_xaamdi(4,1) + 1e-10;
  cu_xaamdi(13,1) = cu_xaamdi(13,1) + 1e-10;
end

persistent al_xaamdi
```

```
if(isempty(al_xaamdi))
  al_xaamdi = importdata('FilterData/al_xaamdi.txt');
  al_xaamdi(4,1) = al_xaamdi(4,1) + 1e-10;
end

persistent sn_xaamdi
if(isempty(sn_xaamdi))
  sn_xaamdi = importdata('FilterData/sn_xaamdi.txt');
  sn_xaamdi(6,1) = sn_xaamdi(6,1) + 1e-10;
  sn_xaamdi(9,1) = sn_xaamdi(9,1) + 1e-10;
  sn_xaamdi(12,1) = sn_xaamdi(12,1) + 1e-10;
  sn_xaamdi(20,1) = sn_xaamdi(20,1) + 1e-10;
end

persistent pb_xaamdi
if(isempty(pb_xaamdi))
  pb_xaamdi = importdata('FilterData/pb_xaamdi.txt');
  pb_xaamdi(5,1) = pb_xaamdi(5,1) + 1e-10;
  pb_xaamdi(8,1) = pb_xaamdi(8,1) + 1e-10;
  pb_xaamdi(11,1) = pb_xaamdi(11,1) + 1e-10;
  pb_xaamdi(14,1) = pb_xaamdi(14,1) + 1e-10;
  pb_xaamdi(17,1) = pb_xaamdi(17,1) + 1e-10;
  pb_xaamdi(24,1) = pb_xaamdi(24,1) + 1e-10;
  pb_xaamdi(27,1) = pb_xaamdi(27,1) + 1e-10;
  pb_xaamdi(30,1) = pb_xaamdi(30,1) + 1e-10;
  pb_xaamdi(38,1) = pb_xaamdi(38,1) + 1e-10;
end

persistent si_xaamdi
if(isempty(si_xaamdi))
  si_xaamdi = importdata('FilterData/si_xaamdi.txt');
  si_xaamdi(4,1) = si_xaamdi(4,1)+1e-10;
end

persistent w_xaamdi
if(isempty(w_xaamdi))
  w_xaamdi = importdata('FilterData/w_xaamdi.txt');
  w_xaamdi(4,1) = w_xaamdi(4,1)+1e-10;
  w_xaamdi(7,1) = w_xaamdi(7,1)+1e-10;
  w_xaamdi(10,1) = w_xaamdi(10,1)+1e-10;
  w_xaamdi(13,1) = w_xaamdi(13,1)+1e-10;
  w_xaamdi(16,1) = w_xaamdi(16,1)+1e-10;
  w_xaamdi(24,1) = w_xaamdi(24,1)+1e-10;
  w_xaamdi(27,1) = w_xaamdi(27,1)+1e-10;
  w_xaamdi(30,1) = w_xaamdi(30,1)+1e-10;
  w_xaamdi(38,1) = w_xaamdi(38,1)+1e-10;
end


  % Density data for each filter
  air_density = 1.225e-6;
```

```
al_density = 2.6941;
cu_density = 8.9400;
sn_density = 7.3000;
pb_density = 11.330;
si_density = 2.3200;
w_density = 19.300;

energies = specin(:,1);
specout = specin;

if(filters(1) > 0) % Air
    atten = calculateAttenuation(air_xaamdi, air_density,...
      energies, filters(1));
    specout(:,2) = specout(:,2).*atten;
end

if(filters(2) > 0) % Al
  atten = calculateAttenuation(al_xaamdi, al_density,...
    energies, filters(2));
  specout(:,2) = specout(:,2).*atten;
end

if(filters(3) > 0) % Cu
    atten = calculateAttenuation(cu_xaamdi, cu_density,...
      energies, filters(3));
    specout(:,2) = specout(:,2).*atten;
end

if(filters(4) > 0) % Sn
    atten = calculateAttenuation(sn_xaamdi,sn_density,...
      energies, filters(4));
    specout(:,2) = specout(:,2).*atten;
end

if(filters(5) > 0) % Pb
    atten = calculateAttenuation(pb_xaamdi, pb_density,...
      energies, filters(5));
    specout(:,2) = specout(:,2).*atten;
end

if(length(filters) >= 6)
if(filters(6) > 0) % Si
    atten = calculateAttenuation(si_xaamdi, si_density,...
      energies, filters(6));
    specout(:,2) = specout(:,2).*atten;
end
end
if(length(filters) >= 7)
if(filters(7) > 0) % W
    atten = calculateAttenuation(w_xaamdi, w_density,...
      energies, filters(7));
```

```
        specout(:,2) = specout(:,2).*atten;
    end
    end

end


function atten = calculateAttenuation(att, density, energies, thickness)
    X = log(att(:,1).*1e3);
    Y = log(att(:,2));
    x = log(energies);
    y = interp1(X, Y, x, 'linear');

    my = exp(y);

    my = my.*density;
    my = my.*0.1; % Attenuation per mm

    atten = exp(-my.*thickness);
end


function filter = FilterFromName(str)

    filter = [];
    if(strcmp(str,'') == 1)
        filter = [0 0 0 0];
        return
    end
    load('FilterData/filterdef.mat');

    str = lower(str);
    str = strrep(str, '-','');

    for i = 1:size(filters,2)
        if(strcmp(str, filters(i).name) == 1)
            filter = filters(i).filter;
            break;
        end
    end
end
```

GetSpectraParameters.m

```
function pars = GetSpectraParameters(spectra)

  energy = spectra(:,1);
  photonCount = round(spectra(:,2));

  % Mean energy
  pars.mean = sum(energy.*photonCount)/sum(photonCount);

  % Find the peak energy
  peaks = ([0; diff(photonCount)]);
  peakCount = photonCount.*(peaks < 1);
  [~, iPeak] = max(peakCount);
  pars.peakMid = energy(iPeak);
  pars.ipeakMid = iPeak;

  % Width of spectral peak is measured as where the histogram values
  % are higher than widthFactor*peakMax.
  widthFactor = 0.5;
  edgeCount = photonCount(iPeak).*widthFactor;

for i = 1:length(photonCount)-1
  % Find start of peak
  if((photonCount(i) <= edgeCount) && (photonCount(i+1) > edgeCount))
    pars.peakMin = energy(i);
    pars.ipeakMin = i;
  end
  % Find end of peak
  if((photonCount(i) >= edgeCount) && (photonCount(i+1) < edgeCount))
    pars.peakMax = energy(i);
    pars.ipeakMax = i;
  end
  % Find kVp
  if((photonCount(i) > 0) && (photonCount(i+1) <= 0))
    kVp = energy(i+1);
    %break;
  end
end

% Width of the peak
pars.peakWidth = pars.peakMax - pars.peakMin;

pars.kVp = kVp;

% Number of photons and total photon energy
pars.photons = sum(photonCount);
pars.photonEnergy = sum(photonCount.*energy);

end
```

SimulateCurrent.m

```
function I = SimulateCurrent(kV, mA, dist, namedFilter, filter)
  % Input:
  %   kV: the kilovoltage setting
  %   mA: the amperage setting
  %   namedFilter: the name of a filter
  %   filter: Extra filtration

  sensitivityFile = 'Diode.mat';

  f1Thickness = XXX CLASSIFIED;
  f2Thickness = XXX CLASSIFIED;
  f3Thickness = XXX CLASSIFIED;

  global diodeSens

  if(isempty(diodeSens))
    load(sensitivityFile);
  end

  distanceFactor = (1000/dist)^2;

  % Get basic spectrum
  unfiltered = GetXRaySpectrum(kV,mA);
  unfiltered(:,2) = unfiltered(:,2).*distanceFactor;

  % Common filtration
  if(nargin == 4)
    if(ischar(namedFilter))
      filter = [0 0 0 0 0];
    else
      filter = [];
    end
  end

  if length(filter) < 7
    filter = [filter zeros(1,7-length(filter))];
  end
  % 0.007 mm extra tungsten filtering
  filter = [dist 0 0 0 0 0 0.007] + filter;

  % Filter spektras for each sensor
  spektra1 = FilterXRaySpectrum(unfiltered, namedFilter, filter);
  spektra2 = FilterXRaySpectrum(spektra1, [0 0 f1Thickness 0 0 0 0]);
  spektra3 = FilterXRaySpectrum(spektra2, [0 0 f2Thickness 0 0 0 0]);
  spektra4 = FilterXRaySpectrum(spektra3, [0 0 f3Thickness 0 0 0 0]);

  persistent sensitivity
  if(isempty(sensitivity))
    % Load sensitivity and interpolate for each bin
```

```
    sensitivity = exp(interp1(log(diodeSens(:,1)), ...
      log(diodeSens(:,2)), log(spektra1(:,1)),'linear'));
    sensitivity(isnan(sensitivity)) = 0;

    ilow = find(spektra1(:,1) < 16);
    ihigh = find(spektra1(:,1) > 125);
    sensitivity(ihigh) = exp( ...
      log(diodeSens(end-1,2)) + ...
      (log(spektra1(ihigh,1))-log(diodeSens(end-1,1))).* ...
      (log(diodeSens(end,2))-log(diodeSens(end-1,2)))./ ...
      (log(diodeSens(end,1))-log(diodeSens(end-1,1))) ...
      );
    sensitivity(ilow) = exp( ...
      log(diodeSens(2,2)) + ...
      (log(spektra1(ilow,1))-log(diodeSens(2,1))).* ...
      (log(diodeSens(1,2))-log(diodeSens(2,2)))./ ...
      (log(diodeSens(1,1))-log(diodeSens(2,1))) ...
      );
  end

  % Calculate currents
  I(1) = sum(spektra1(:,2).*sensitivity.*spektra1(:,1));
  I(2) = sum(spektra2(:,2).*sensitivity.*spektra2(:,1));
  I(3) = sum(spektra3(:,2).*sensitivity.*spektra3(:,1));
  I(4) = sum(spektra4(:,2).*sensitivity.*spektra4(:,1));

end
```

Artificial Neural Network

```
function ANN
global diodeSens
load diode.mat
  clc
  close all

  norm = 200;
  steps = 20000;
  eta = 0.0001;
  epsilon = 0.9;

  net = newNetwork([9 100 100 100],1);

  % Generate evaluation data set
  for i=1:100
    evSet(:,i) = getTestData()';
  end
  kV = evSet(1,:);

  % Generate training data set
  for i=1:1000
    trainingSet(:,i) = getTestData()';
  end

  prediction = norm*runNetwork(net, evSet(2:end,:));
  error = 100*sqrt(mean(abs((kV-prediction)./kV).^2));

  % Training graph
  figure('Color', 'White');
  errPlot = plot([0 1],[error error], 'k', 'LineWidth', 1.2);
  set(gca,'fontSize',14,'lineWidth',2,'box','off');
  xlabel('Iteration');
  ylabel('E_{RMS}');

  % Performance graph
  f2 = figure('Color', 'White');
  predictionPlot = plot(kV, norm*runNetwork(net, evSet(2:end,:)), 'kx',...
    'LineWidth', 1.2, 'MarkerSize', 15);
  hold on
  plot(kV,kV,'k--', 'LineWidth', 1.2);
  hold off
  set(gca,'fontSize',14,'lineWidth',2,'box','off')
  xlabel('Model input (keV)');
  ylabel('Network prediction (keV)');
  xlim([40,160]);
  ylim([0,200]);

  best = 100;
  for(i = 1:steps)
```

```
    net = trainNetwork(net, trainingSet(2:end,:),...
      trainingSet(1,:)./norm, eta, epsilon);

    if(mod(i,10) == 0) % Redraw plots every 10 iterations to save time
      prediction = norm*runNetwork(net, evSet(2:end,:));
      error = 100*sqrt(mean(abs((kV-prediction)./kV).^2));
      X = [get(errPlot,'XData') i];
      Y = [get(errPlot,'YData') error];
      set(errPlot, 'XData',X,'YData',Y);

      set(predictionPlot, 'YData', prediction);
      drawnow
      if(error < best)
        best = error;
      end

      clc
      error
      best
    end
  end

end

function y = g(b)
  beta = 0.1;
  y = tanh(beta*b);
end
function y = gprim(b)
  beta = 0.1;
  y = sech(beta*b).^2;
end

function net = trainNetwork(net, input, solution, eta, epsilon)
  % Trains a network using simple back-propagation
  persistent prevDelta

  [output V] = runNetwork(net, input);
  nLayers = length(net);

  delta{nLayers} = (solution-output);
  delta{nLayers} = delta{nLayers}.* ...
    gprim(net{nLayers}*[V{nLayers}; -1*ones(1,size(V{nLayers},2))]);

  for(i = nLayers:-1:2)
    delta{i-1} = (net{i}'*delta{i});
    delta{i-1}(end,:) = [];
    delta{i-1} = delta{i-1}.* ...
      gprim(net{i-1}*[V{i-1}; -1*ones(1,size(V{i-1},2))]);
  end
  if(isempty(prevDelta))
```

```
    for i=1:nLayers
      prevDelta{i} = zeros(size(delta{i}));
    end
  end
  for(i = 1:nLayers)
    d = delta{i};
    delta{i} = delta{i} + epsilon*prevDelta{i};
    prevDelta{i} = d;
    dW{i} = delta{i}*[V{i}; -1*ones(1,size(V{i},2))]';
    net{i} = net{i} + dW{i}.*eta;
  end

end

function [result, V] = runNetwork(net, input)
  % Run network

  nLayers = length(net);
  prevV = input;
  for(i = 1:nLayers)
    V{i} = prevV;
    newV = g(net{i}*[prevV; -1*ones(1,size(prevV,2))]);
    prevV = newV;
  end
  V{nLayers+1} = newV;
  result = newV;
end

function net = newNetwork(nNodes, startLevel)
  % Generate a neural network with random weights

  nLayers = length(nNodes);
  nNodes = [nNodes 1];
  for(i = 1:nLayers)
    net{i} = (rand(nNodes(i+1), nNodes(i)+1)-0.5).*2.*startLevel;
  end
end

function data = getTestData()
  % Generate data points

  kv = 40 + round(110*rand(1,1)); % 40-150
  ma = 100 + 400*rand(1,1); % 100-500

  filter = [0     3*rand(1,1) 2*rand(1,1) 0    0 ];
  %          air   al          cu          sn   pb

  dist = 100 + 400*rand(1,1); % 100-500

  I = SimulateCurrent(kv,ma,dist,'',filter);
```

```
    A1 = I(2)/I(1);
    A2 = I(3)/I(1);
    A3 = I(4)/I(1);
    A4 = I(3)/I(2);
    A5 = I(4)/I(2);
    A6 = I(4)/I(3);

    %data = [kv, I];
    %data = [kv, A1, A2, A3];
    %data = [kv, A1, A2, A3, A4, A5, A6];
    %data = [kv, A1, A2, A3, A1^2, A2^2, A3^2];
    %data = [kv, A1, A2, A3, A1*A2, A1*A3, A2*A3, A1^2, A2^2, A3^2];
    data = [kv, A1, A2, A3, sqrt(A1), sqrt(A2), sqrt(A3)];

end
```

Linear Genetic Programming

```
function LGPTest

  clc
  close all hidden
  warning('off','MATLAB:rankDeficientMatrix');

  global mutationProb
  global outputRegisters
  global tournamentSize
  global tournamentSelectionParameter

  populationSize = 100;
  tournamentSize = 5;
  tournamentSelectionParameter = 0.75;
  crossProbability = 0.20;
  mutationProb = 0.04;
  numGenerations = 1000;
  bestCopies = 5;
  outputRegisters = 4;

  % Generate evaluation data set
  for i=1:100
    evSet(i,:) = getTestData();
  end
  kV = evSet(1,:);

  % Generate training data set
  for i=1:100
    trainingSet(i,:) = getTestData();
  end

  % Prepare figures
  f1 = figure('Color', 'White');
  set(f1,'Color', 'White');
  plot(evSet(:,1), evSet(:,1), 'k—', 'LineWidth', 1.2);
  hold on
  predictionPlot = plot(evSet(:,1), evSet(:,1), 'kx', ...
    'LineWidth', 1.2, 'MarkerSize', 15);
  hold off
  set(gca,'fontSize',14,'lineWidth',2,'box','off')
  xlabel('Model␣input␣(keV)');
  ylabel('Best␣prediction␣(keV)');
  ylim([0, 200]);

  f2 = figure('Color', 'White');
  fitPlot = plot(0,0, 'k—','LineWidth', 1.2);
  hold on
  fitPlot2 = plot(0,0,'k', 'LineWidth', 1.2);
  hold off
```

```
legend('Mean','Best');
set(gca,'fontSize',14,'lineWidth',2,'box','off')
xlabel('Generation');
ylabel('Root mean square error [%]');

figure(f1)
h = waitbar(0);
oldfit = 0;
fitness = zeros(populationSize,1);

% Initialize population
population = InitializePopulation(populationSize);

out = DecodeIndividual(population{1}, evSet(:,2:end))
out = [out, ones(size(out,1),1)];
fac = (out\evSet(:,1))';
res = out*fac';
set(predictionPlot, 'YData', res);

pause

% Main loop
for i = 1:numGenerations

  % Update progress bar
  waitbar(i/numGenerations,h,num2str(i));

  bestFitness = inf;
  bestInd = 0;

  % Evaluate fitness of each individual
  for j = 1:populationSize
    ind = population{j};
    fit = TestFitness(ind, trainingSet);
    fitness(j) = fit;
    % Keep track of very best individual
    if(fit < bestFitness)
      bestInd = j;
      bestFitness = fit;
    end
  end

  % Check best individual with the evaluation data set
  correct = evSet(:,1);
  out = DecodeIndividual(population{bestInd}, evSet(:,2:end));
  out = [out ones(size(out,1),1)];
  fac = (out\evSet(:,1))';
  res = out*fac';
  bestfitness = sqrt(nanmean(abs((correct-res)./correct).^2))*100;

  % Update output
```

```
    if(bestfitness ~= oldfit)
      clc
      oldfit = bestfitness;
      ReadOut(population{bestInd});
      display(['Best: ', num2str(bestfitness)]);
    end

    % Uppdate plots
    set(predictionPlot, 'YData', res)

    X = get(fitPlot,'XData');
    Y = get(fitPlot, 'YData');
    Y2 = get(fitPlot2, 'YData');
    fitness2 = fitness;
    fitness2(isinf(fitness2)) = nan;
    set(fitPlot, 'XData', [X i], 'YData', [Y nanmean(fitness2)])
    set(fitPlot2, 'XData', [X i], 'YData', [Y2 bestfitness])
    drawnow


    % Create new population
    newPopulation = population;

    for(j = 1:2:populationSize)
      % Select two winners
      iWinner1 = TournamentSelect(fitness);
      iWinner2 = TournamentSelect(fitness);
      c1 = population{iWinner1};
      c2 = population{iWinner2};

      % Crossover (possibly)
      r = rand;
      if(r < crossProbability)
        newPair = Cross(c1, c2);
      else
        newPair = {c1, c2};
      end
      newPopulation{j} = Mutate(newPair{1});
      newPopulation{j+1} = Mutate(newPair{2});
    end

    % Make some copies of the best individual
    for j = 1:bestCopies
      newPopulation{i} = population{bestInd};
    end

    for(j=1:populationSize)
      population{j} = newPopulation{j};
    end

  % main loop
```

```
  end
end

function fitness = TestFitness(ind, testData)
  % Calculate fitness for an individual in the population

  correct = testData(:,1);
  out = DecodeIndividual(ind, testData(:,2:end));

  % Calculate factors from 25 random points
  i = randperm(size(testData,1));
  i = i(1:25);
  fac = (out(i,:)\testData(i,1));

  res = out*fac;
  correct = testData(:,1);

  fitness = 100*sqrt(nanmean((abs(correct-res)./ ...
    max(abs(correct),abs(res))).^2));
  if(isnan(sum(res)))
    fitness = inf;
  end

end

function output = DecodeIndividual(chromosome, input)

% Instructions
% 1 = +
% 2 = -
% 3 = *
% 4 = /
% 5 = ^
% 6 = log

% Targets
% 1 = register a
% 2 = register b
% 3 = register c
% 4 = register d

% Sources
% 5 = input 1
% 6 = input 2
% 7 = input 3
% 8 = input 4
% 9 = const 1
% 10 = const 2
% 11 = const 3
% 12 = const 4
```

```
  global outputRegisters
  chromosomeLength = size(chromosome,1);

  registers = [input input ones(size(input,1),1)*[1, 3, -1]];

  % Step through each gene
  for i = 1:chromosomeLength
    instruction = chromosome(i,1);
    target = chromosome(i,2);
    input1 = registers(:,chromosome(i,3));
    input2 = registers(:,chromosome(i,4));

    registers(:,target) = ...
      ((instruction == 1).*(input1+input2)) + ...
      ((instruction == 2).*(input1-input2)) + ...
      ((instruction == 3).*(input1.*input2)) + ...
      ((instruction == 4).*(input1./input2)) + ...
      ((instruction == 5).*(input1.^input2)) + ...
      ((instruction == 6).*(sqrt(input1)));
  end

  % Return some registers
  output = registers(:,1:outputRegisters);
  output(isnan(output)) = 0;

end

function population = InitializePopulation(populationSize)
  % Initialize randomized population

  population = cell(populationSize, 1);
  for i = 1:populationSize
    population{i} = NewChromosome(5+randi(20,1));
  end
end

function newGene = NewChromosome(numGenes)
  % Create one new chromosome

  numInstructions = 2;
  numOutputs = 8;
  numInputs = 11;
  newGene = [];
  for i = 1:numGenes
    instruction = randi(numInstructions, 1) + 2;
    output = randi(numOutputs,1);
    input1 = randi(numInputs, 1);
    input2 = randi(numInputs, 1);
    newGene = [newGene; [instruction output input1 input2]];
  end
```

```
end

function gene = Mutate(gene)
  % Mutate a gene
  global mutationProb

  nGene = size(gene,1);
  for i = 1:nGene
    r = rand;
    if(r < mutationProb)
      gene(i,:) = NewChromosome(1);
    end
  end

end

function genes = Cross(chr1, chr2)
  % Cross two chromosomes

  nGene1 = size(chr1,1);
  nGene2 = size(chr2,1);

  cp1 = sort(randi(nGene1,1,2));
  cp2 = sort(randi(nGene2,1,2));

  genes{1} = [chr1(1:cp1(1),:); ...
              chr2(cp2(1):cp2(2),:); ...
              chr1(cp1(2):end,:)];
  genes{2} = [chr2(1:cp2(1),:); ...
              chr1(cp1(1):cp1(2),:); ...
              chr2(cp2(2):end,:)];

end

function ind = TournamentSelect(fitness)
  % Tournament selection
  global tournamentSize
  global tournamentSelectionParameter

  inds = randi(length(fitness),tournamentSize, 1);

  for i = 1:tournamentSize -1
    if(fitness(inds(i)) > fitness(inds(i+1)))
      t = inds(i);
      inds(i) = inds(i+1);
      inds(i+1) = t;
    end
    r = rand;
    if(r < tournamentSelectionParameter)
      ind = inds(i);
      return
```

```
    end
  end
  ind = inds(end);
end

function ReadOut(chromosome)
  % Print chromosome in human readable format

  disp('Best individual:');
  strs = {'a','b','c','d','e','f','g','h','1','3','-1'};
  instr = {'+','-','*','/','^'};
  res = {'a','b','c','d','e','f','g','h'};
  lGene = size(chromosome,1);
  for i = lGene:-1:1
    for j = 1:8
      if(chromosome(i,1) == 6)
        res{j} = strrep(res{j},strs{chromosome(i,2)}, ...
          sprintf('Sqrt[%s]',strs{chromosome(i,3)}));
      else
        res{j} = strrep(res{j},strs{chromosome(i,2)}, ...
          sprintf('(%s%s%s)',strs{chromosome(i,3)}, ...
            instr{chromosome(i,1)},strs{chromosome(i,4)}));
      end
    end
  end
  strs = {'a','b','c','d','e','f','g','h','ln','--'};
  subs = {'s0','s1','s2','s3','s0','s1','s2','s3','Log','+'};
  for j = 1:8
  stri = res{j};
  for i = 1:length(strs)
    stri = strrep(stri, strs{i}, subs{i});
  end
  disp(stri)

  vars = '';
  if(~isempty(findstr(stri,'s0')))
    vars = strcat(vars, 's0 ');
  end
  if(~isempty(findstr(stri,'s1')))
    vars = strcat(vars, 's1 ');
  end
  if(~isempty(findstr(stri,'s2')))
    vars = strcat(vars, 's2 ');
  end
  if(~isempty(findstr(stri,'s3')))
    vars = strcat(vars, 's3 ');
  end
  disp(['Included variables: ' vars])
  end
end
```

```
function data = getTestData()
  % Generate data points

  kv = 40 + round(110*rand(1,1)); % 40-150
  ma = 100 + 400*rand(1,1); % 100-500

  filter = [0     3*rand(1,1) 2*rand(1,1) 0    0 ];
  %          air   al          cu          sn   pb

  dist = 100 + 400*rand(1,1); % 100-500

  I = SimulateCurrent(kv,ma,dist,'',filter);

  data = [kv, I];
end
```