Delft University of Technology

Research Minor

TA-MI-077

Mincomp - a program to calculate a likely mineralogical bulk composition from XRD and XRF results

Author:
Jaap Regelink

7th of November 2014

Mincomp - a program to calculate a likely mineralogical bulk composition from XRD and XRF results

Research Minor

TA-MI-077

Faculty:
Faculty of Civil Engineering and Geosciences

Department:
Department of Geoscience and Engineering

Author:
Jaap Regelink

Studentnumber:
4064526

Date:
7th of November 2014

Supervisor:
Dr. K.H.A.A. Wolf

Commission:
Drs. M.M. van Tooren
Dr. K.H.A.A. Wolf

Date Presentation:
25th of November 2014

Abstract

A lot of X-ray diffraction and X-ray fluorescence tests are performed in the department of Geoscience and Engineering to calculate a rocks likely mineralogical bulk composition. The old program used for this task was considered not user friendly enough, therefore an updating process of the old Mincomp program has been performed.
During this updating process attention has been paid to justify the presence of the minerals in the Mineral Inventory of the program, and to the programming sequence by comparing Mincomp to other available programs, in order to write a new program.
The results were compared to other available programs when writing of the program was finished, to identify differences between the programs and identify the shortcomings of Mincomp, as well as a justification of the used method.
In general, the results of Mincomp are comparable with the results of other available programs, however the results on some samples differed in the calculated amount of Kaolinite and Illite, this is mainly caused by the allocation order of the program. It is recommended to investigate this difference a bit more and possibly revise the algorithm that is used at this moment.

Contents

Chapter 1

Introduction

This report deals with the updating project of the Mincomp computer program designed by K.H.A.A. Wolf. In order to do so, a literature study was performed and updated data were used to renew the program.
The need for a consistent mineral quantification, based on X-ray diffraction and X-ray fluorescence data of rock sampleswas the main reason for starting this project. The older Mincomp program performs this job really well, but was considered not user-friendly enough and not up-to-date enough to be used throughout the department. Therefore an updating process has been carried out.

Mincomp is designed for calculating a normative mineral composition of sedimentary rocks, and focuses on the minerals mainly found in sedimentary rocks, therefore the inventory of minerals included in Mincomp will be discussed. The reason to include a specific mineral, but also the chemical composition of a specific mineral will be discussed, the ideal chemical composition is used mostly. Apart from the main minerals found in sedimentary rocks, some less general minerals are also included such as glauconite and anorthite. More rare minerals like Manganese-bearing minerals are not included in this program.
Apart from the inventory of minerals, an inventory of Mincomp comparable programs is also included. The differences in mineral composition, inventory of minerals and calculation process will be discussed. Programs in this inventory differ in the calculation method, making use of Linear Algebra or using an Algorithm. Differences in the mineral inventory included in the different programs were also observed, as some of the compared programs were designed to calculate the likely mineralogical bulk composition of a specific type of rock in mind.
Finally Mincomps results will be compared to the results of the Mincomp comparable programs, to create a benchmark and validate the results of Mincomp. Extra attention has been paid to the differences in calculated amounts of minerals, as differences were sometimes quite significant. For each available dataset, Mincomp has been run to calculate a mineralogical bulk composition in two ways, to validate its results.

Chapter 2

Inventory of minerals

2.1 Mineral overview

Mincomp was developed for analyzing sedimentary rocks, therefore the list of minerals is limited to common sedimentary minerals. A couple of trace minerals are included, such as rutile, but most of the trace elements are not included in this program. This is because the focus of the program is to give a likely mineralogical bulk composition; the allocation of trace elementsc which make up 1 % of the samplec was not considered a primary aspect of the program.
There are some minerals that have a variable chemistry, for example chlorite and montmorillonite. In these cases the empirical formula has been used in order to calculate the amount of these minerals.

Principal minerals in the Earth's Crust	
Mineral	**Presence in %, based on the actual mineral composition**
Quartz	12
Potash feldspars	12
Plagioclase feldspars	39
Micas	5
Amphiboles	5
Pyroxenes	11
Olivines	3.6
Clay minerals and Chlorite	4.6
Calcite and Aragonite	1.5
Dolomite	0.5
Magnetite and Titanomagnetite	1.5
Other minerals like Garnet, Kyanite, etc.	4.9
Coal and hydro-carbons	accessory
Total	**100**

Table 2.1: Abundance of minerals in the Earth's Crust. (Ronov and Yaroshevsky, 1967)

From the data that Ronov and Yaroshevsky (1967) present in table 2.1 the most important sedimentary minerals were selected, this list was extended by the information Wolf (2006) presented. Individual minerals were researched based on the works of Deer et al. (1966), Anthony et al. (1995) and Barthalmy (2013). The included minerals are presented in table 2.1, and are discussed in detail in this chapter.

Mineral	Chemical formula	ρ $[^{g}/_{cm^3}]$	M $[^{g}/_{mol}]$	V $[^{cm3}/_{mol}]$
Pyrite	FeS_2	5.01	119.99	23.95
Hematite	Fe_2O_3	5.3	159.7	30.13
Rutile	TiO_2	4.25	79.87	18.79
Gibbsite	$Al(OH)_3$	2.34	78.004	33.34
Goethite	$FeO(OH)$	3.8	88.858	23.38
Halite	$NaCl$	2.17	58.44	26.93
Calcite	$CaCO_3$	2.71	100.09	36.93
Dolomite	$CaMg(CO_3)_2$	2.84	184.41	64.93
Magnesite	$MgCO_3$	3	84.32	28.11
Siderite	$FeCO_3$	3.96	115.86	29.26
Anhydrite	$CaSO_4$	2.97	136.95	46.11
Apatite	$Ca_5(PO_4)_3(OH)$	3.19	506.318	158.72
Chlorite	$FeMg_4Al(Si_3Al)O_{10}(OH)_8$	2.65	587.384	221.65
Glauconite	$K_{0.6}Na_{0.05}Fe_{1.5}Mg_{0.4}Al_{0.3}Si_{3.8}O_{10}(OH)_2$	2.67	426.93	159.90
Muscovite	$K_2Al_4(Si_6Al_2)O_{20}(OH)_4$	2.82	796.652	282.50
Kaolinite	$Al_2Si_2O_5(OH)_4$	2.6	258.172	99.30
Illite	$KAl_2(Si_3Al)O_{10}(OH)_2$	2.75	398.326	144.85
Montmorillonite	$Ca_{0.17}Na_{0.31}Mg_{0.33}Al_{1.67}Si_4O_{10}(OH)_{2,61}$	2.35	383.77	163.30
Quartz	SiO_2	2.62	60.09	22.94
Albite	$NaAlSi_3O_8$	2.62	262.24	100.09
Anorthite	$CaAl_2Si_2O_8$	2.73	279.02	102.21
Orthoclase	$KAlSi_3O_8$	2.56	278.35	108.73
Water	H_2O	0.998	18.016	18.05
Organic Matter	$CH_{0.732}O_{0.046}S_{0.004}N_{0.013}$	-	13.794	-

Table 2.2: Mineral list, formulas from Deer et al. (1966), density from Barthalmy (2013)

Mineral weight is calculated with element weights from Tro (2010). Mineral densitys are the averaged values from Barthalmy (2013).

Pyrite FeS_2 If S is present in the XRF data, this trace element is allocated to Pyrite. Sulphur usually is present in very small amounts, so it is allocated previous to the bulk in allocation stage 1.

Hematite Fe_2O_3 Hematite is a key component in iron ores, and is accountable for the common red coloration of rocks (Deer et al., 1966). Hematite is calculated using excess iron, or if present in XRD analysis.

Rutile TiO_2 As Ti usually only occurs as a trace element, it is allocated in the first stage to Rutile. Other programs use Anatase, but Rutile is more common in sediments. (Wolf, 2006)

Gibbsite $Al(OH)_3$ The aluminahydroxide Gibbsite is one of the three main components of bauxites and laterites (Deer et al., 1966). It can be used for excess Al, but was not available in the first version of MINCOMP.

Goethite $FeO(OH)$ The iron-hydroxide Goethite is also incorporated in this program, Goethite commonly occurs as a weathering product from other iron-bearing minerals, but also accumulates as a precipitate from marine waters. In some iron ores it is the main component. (Deer et al., 1966)

Halite $NaCl$ The salt Halite is also common in sedimentary rocks. It can occur by evaporation of seawater, which leads to the deposition of Halite (Deer et al., 1966), in this program, all chlorine is allocated to Halite.

Calcite $\mathbf{CaCO_3}$ Calcite is one of the most common minerals on earth, as the main mineral in most limestones. It occurs as a primary precipitate and in the form of fossil shells. (Deer et al., 1966) Calcite can be calculated with Ca, when there is excess Ca after alumina-silicate allocation, or if presence is proven from XRD data or thin-sections.

Dolomite $\mathbf{CaMg(CO_3)_2}$ Dolomite is another common mineral in limestones, it can form as a primary mineral but is more common as secondary mineral when Calcite or Aragonite reacts with Magnesium (Deer et al., 1966). Dolomite can be calculated when excess Ca or Mg is present, or if presence is proven from XRD data or thin-sections.

Magnesite $\mathbf{MgCO_3}$ If excess amounts of Mg are present after allocation stage 2, it can be allocated to Magnesite. Otherwise, Magnesite is only allocated if presence is proven from XRD data or thin-sections.

Siderite $\mathbf{FeCO_3}$ With Siderite as well as the other carbonates, it is usually only allocated if it is present from XRD data or thin-sections. Another case is an excess amount of iron after allocation stage 2, and a high enough total weight loss to compensate for the CO_2.

Anhydrite $\mathbf{CaSO_4}$ Anhydrite is used instead of Gypsum, because the attached water is allocated to the total weight loss. Anhydrite is calculated if SO_3 is measured by the XRF-test.

Apatite $\mathbf{Ca_5(PO_4)_3(OH)}$ Apatite is not uncommon in sedimentary rocks, it occurs as detrital sedimentary mineral. (Deer et al., 1966; Wolf, 2006) Fluorine occurs in many common rock-forming minerals which occur in both igneous and sedimentary rocks, such as apatite, silicates such as muscovite, and a range of amphiboles and mica minerals. Substitution of the OH^- ion is commonplace. (Salminen et al., 2005) Fluorine is by far the most abundant halogen in sedimentary rock types. Clastic sediments can contain up to percentage level amounts of fluorine. (Salminen et al., 2005)
However, Calcium-Apatite or Hydroxyapatite is used instead of the more common fluor-apatite, this is because fluorine is usually only measured in very low amounts in the XRF analysis, and the current version of Mincomp doesn't support a variable chemistry.

Chlorite $\mathbf{FeMg_4Al(Si_3Al)O_{10}(OH)_8}$ Chlorite is a common mineral in argillaceous sediments, in which it can occur as authigenic or detrital mineral. Because of the size of the crystals it is usually very difficult to characterize the minerals.
For Chlorite, the ideal formula of Clinochlore is used.

Glauconite $\mathbf{K_{0.6}Na_{0.05}Fe_{1.5}Mg_{0.4}Al_{0.3}Si_{3.8}O_{10}(OH)_2}$ Glauconite is a sheet-silicate which occurs almost exclusively in marine sediments, particularly greensands (Deer et al., 1966; Anthony et al., 1995), and is therefore considered in this program. Because of the variation in chemical formula, the empirical formula presented at the website Webmineral.com is used.(Barthalmy, 2013)

Muscovite $\mathbf{K_2Al_4(Si_6Al_2)O_{20}(OH)_4}$ Muscovite is a very common mineral in igneous rocks, but less common in sedimentary rocks as initially believed. It is often mixed with chlorite and montmorillonite. (Deer et al., 1966)

Kaolinite $\mathbf{Al_2Si_2O_5(OH)_4}$ Probably the most common clay mineral is Kaolinite, it is formed principally by the hydrothermal alteration or weathering of feldspars and other silicates. Kaolinite isn't subject to much variation.

Illite $\mathbf{KAl_2S(Si_3Al)O_{16}(OH)_2}$ Illite is a very common clay mineral if many shales and mudstones, but can also occur in limestones. It can be deposited after weathering of silicates, but can also be formed during diagenesis. (Deer et al., 1966) Here Illite is calculated using K.

Montmorillonite $\mathbf{Ca_{0.17}Na_{0.31}Mg_{0.33}Al_{1.67}Si_4O_{10}(OH)_{2.61}}$ Montmorillonite, a member of the smectite-group, is a very common clay mineral, and widely found in soils and shales which have resulted from weathering of basic rocks. Montmorillonite will only form if there is enough Magnesium available. (Deer et al., 1966) Since the mineral can have great variability, the empirical formula for Montmorillonite from (Deer et al., 1966) is used.

Quartz SiO_2 One of the most common minerals in the world, a high amount of quartz is often present in sedimentary rocks. Mincomp calculates the amount of quartz after the allocation of alumina-silicates, all excess Si is allocated to Quartz.

Albite $NaAlSi_3O_8$ Albite and Anorthite are end-members of the plagioclase group, both are incorporated in the mineral list. Albite is a common authigenic mineral and sedimentary mineral (Deer et al., 1966) and is therefore included.
The amount of Albite can be calculated with Na.

Anorthite $CaAl_2Si_2O_8$ The other end-member of the plagioclase group, Anorthite, is also present in the program. Since Anorthite is the first mineral that is formed when the magma cools down, it is also the most vulnerable to weathering, and therefore less likely to occur in a sedimentary rock. If the presence of Anorthite is proven by the XRD test, it can be calculated with Ca.

Orthoclase $KAlSi_3O_8$ The K-feldspar Orthoclase is usually only calculated if its presence is proven by the XRD test. It is a very common mineral in igneous rocks, but it can also be present in argillaceous sediments as a weathered mineral. The weathering products of Orthoclase are used for the formation of different clays.

Chapter 3

Inventory of programs

The goal of normative analysis is to determine the mineralogy of rocks from their bulk chemical composition. A norm is a calculated inventory of mineral abundances in a rock, and is accurate when these approach or equal the actual mineral amounts, collectively referred to as the mode. (Caritat et al., 1994) There have been developed a number of computer programs to calculate these norms for sedimentary rocks over the last decades, for example: Sednorm (Cohen and Ward, 1991) , Moduscalc (Laube et al., 1996) , LPNorm (Caritat et al., 1994) , A2M (Posch and Kurz, 2007) and Minlith (Rosen et al., 2004). All programs obviously have in common that they calculate a mineral norm, but there are some differences between the programs.

The biggest difference between the available programs lies in the calculation methods, Sednorm and Minlith rely on an algorithm of allocating different element-oxides to different minerals, in a pre-defined routine. The others rely on Linear Algebra to solve a system of x equations with x unknown variable, this of course results in different outcomes.
Linear programming calculation methods often try to find a 'best-fit' approximation to the sample (Laube et al., 1996) while algorithm-based programs rely more on the experience of the user. It must be stressed that different solution techniques generally give different results. (Rosen et al., 2004)
Both methods have specific advantages and disadvantages, while linear algebra is a more sophisticated calculation method and provides room for statistical routines to, for example, estimate the degree of reliability, (Laube et al., 1996) it usually provides no room for experience-based operator input.
Algorithm based programs have the downside that the allocation process is rigid and predefined, and therefore allow less variation in chemical composition and the list of minerals that is used for calculation (Caritat et al., 1994). However, these programs do provide room for experience-based operator input (Cohen and Ward, 1991), the operator therefore can influence the calculation method based on extra knowledge of the sample.

Following this introduction a discussion about several available programs will follow in the next sections.

3.1 Sednorm

Developed in 1991 by Cohen and Ward, Sednorm was one of the first programs developed to calculate a normative mineral composition. It uses a predefined allocation routine to allocate element-oxides to certain minerals. It was also one of the first computer programs that gave some space for user-input. Some of the operator choices that could be made were the distribution of K into Muscovite/Illite or into K-feldspar, but also the Ca:Na ratio in Smectite could be set.
The developers have chosen a rather small selection of minerals that are incorporated in the calculation sequence, this is because they claim these minerals make up the bulk content of most sediments. (Cohen and Ward, 1991)

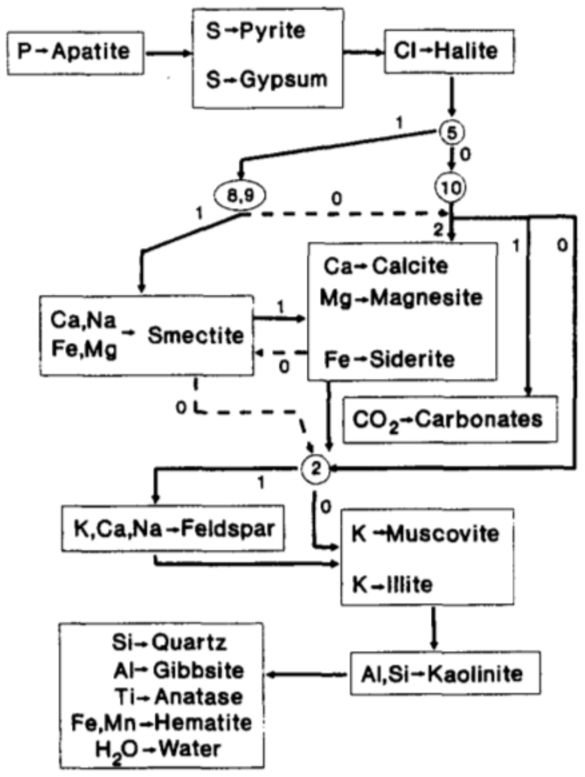


Figure 3.1: Flowchart for Sednorm. (Cohen and Ward, 1991)

Sednorm options available in calculation sequence	
Option	Function
1	Distribute K into Muscovite/Illite
2	Include or exclude feldspar
3	Set distribution of K Muscovite/Illite : Feldspar in ratio
4	Incorporate sulphur as sulfide(Pyrite) or as sulphate(Gypsum)
5	Include or exclude smectite
6	Set ratio of Ca:Na in Smectite in ratio
7	Set ratio of Mg:Fe in Smectite in ratio
8	Distribute Mg initially into Dolomite or Smectite
9	Distribute Fe initially into Smectite or Siderite
10	Availability of CO_2 data
11	Fix H_2O at the initial concentration
12	Do or do not review options selected after calculation

Table 3.1: Sednorm options available in calculation sequence. (Cohen and Ward, 1991)

3.2 LPNorm

Developed in 1993 by (Caritat et al., 1994), LPNorm uses linear algebra to calculate the normative mineral composition. The program was developed, bearing in mind the drawbacks of a fixed algorithm method. The developers therefore tried to overcome these drawbacks. They mention the fixed and rigid allocation, the inability to take chemical variability into account and the restricted list of minerals available for calculation.
The program creates a system of equations and calculates a 'best-fit' solution to the problem. Since this sometimes can result in unsatisfactory results (in terms of high slack wt%), the objective function also

can be maximized. In this case, the program tries to find a solution with as less slack wt% possible. (Caritat et al., 1994)
Here, slack refers to the percentage of unallocated weight.

3.3 Moduscalc

Developed (Laube et al., 1996), Moduscalc calculates the normative mineral composition with linear algebra. Therefore a system of equations in the form $Ax = b$ is generated, with the vector A containing the minerals, and the vector x containing the individual weight portions. Because this system is usually overdetermined, the number of element-oxides is greater than the number of minerals, it can not be solved exactly. To overcome this problem, Moduscalc tries to calculate a 'best-fit' solution to the problem.
Apart from calculating a normative mineral composition, Moduscalc also calculates the likelihood of the solution, as well as the quality of calculation. (Laube et al., 1996)

3.4 Minlith

Minlith is a newer computer program, developed by (Rosen et al., 2004). Minlith uses an experience-based algorithm to calculate the normative mineral composition. It is aimed at mature sediments, but can be used, with care, for younger sediments. The algorithm is built based on a reference database of 600 samples, instead of user-experience. Also in this program the operator-input is limited. In order to comply to the statistical data from the reference database, different mineral assemblages are pre-defined and the computer program calculates which assemblage matches the sample the most. (Rosen et al., 2004)

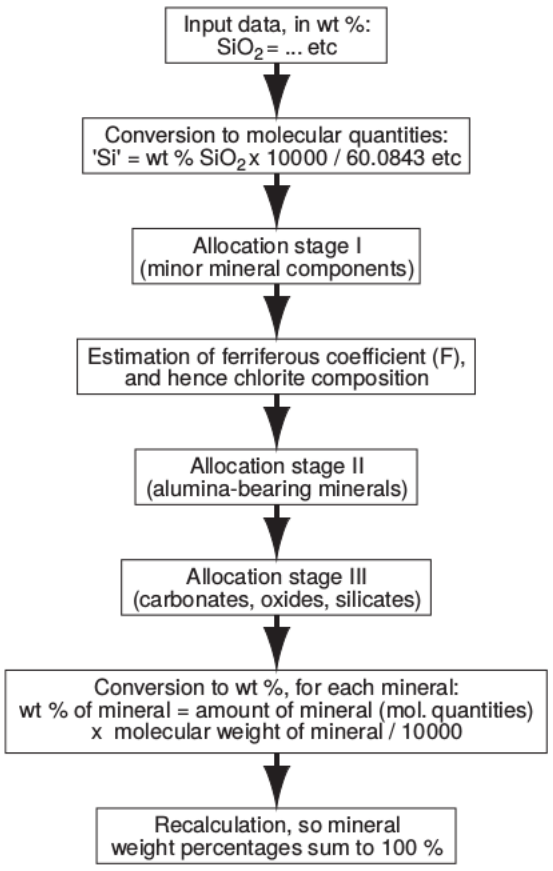

Input data, in wt %: SiO_2 = ... etc
Conversion to molecular quantities: 'Si' = wt % SiO_2 x 10000 / 60.0843 etc
Allocation stage I (minor mineral components)
Estimation of ferriferous coefficient (F), and hence chlorite composition
Allocation stage II (alumina-bearing minerals)
Allocation stage III (carbonates, oxides, silicates)
Conversion to wt %, for each mineral: wt % of mineral = amount of mineral (mol. quantities) x molecular weight of mineral / 10000
Recalculation, so mineral weight percentages sum to 100 %

Figure 3.2: Flowchart for Minlith. (Rosen et al., 2004)

3.5 A2M

A2M is the newest program discussed in this report developed (Posch and Kurz, 2007). This program also uses linear algebra to calculate the normative mineral composition. A2M differs from the other programs, apart from the most likely norm, it calculates all possible outcomes from the system of equations. These are then represented as a convex polyhedron in the solution space. This polyhedron contains all possible solutions to the set of linear equations.
However, *all* minerals is not completely true, A2M also uses a pre-defined list of minerals, but it does calculate all the different options. (Posch and Kurz, 2007)
Less knowledge about the sample generally results in very big deviations in the possible outcomes, the result is not precise.

3.6 Comparison of the programs

Most of the papers discussing the different programs show correlation graphs, in which the correlation between the calculated normative mineral composition and the actual mineral composition is shown.
Most authors refer to other programs and make comparisons between them, but unfortunately only the authors of LPNorm showed a real data comparison between LPNorm and Sednorm. These results were close to eachother, the main difference was the weight percentages of Quartz and Kaolinite. But results were generally alike. In chapter 5, Mincomp will be compared to the other programs.

Program	Method	Minerals	Intuitive	Variation in chemical composition
Sednorm	Algorithm	18	yes	no
LPNorm	Linear Algebra	10	No	semi
Moduscalc	Linear Algebra	12	No	No
Minlith	Algorithm	25	No	semi
A2M	Linear Algebra	∞	No	yes
Mincomp	Algorithm	22	semi	no

Table 3.2: Comparison of available programs.

Chapter 4

Algorithm

The algorithm can be separated into different steps, according to the flow diagram of Mincomp. Since Mincomp was written from scracth, the procedure is explained in this chapter.

4.1 Program workflow

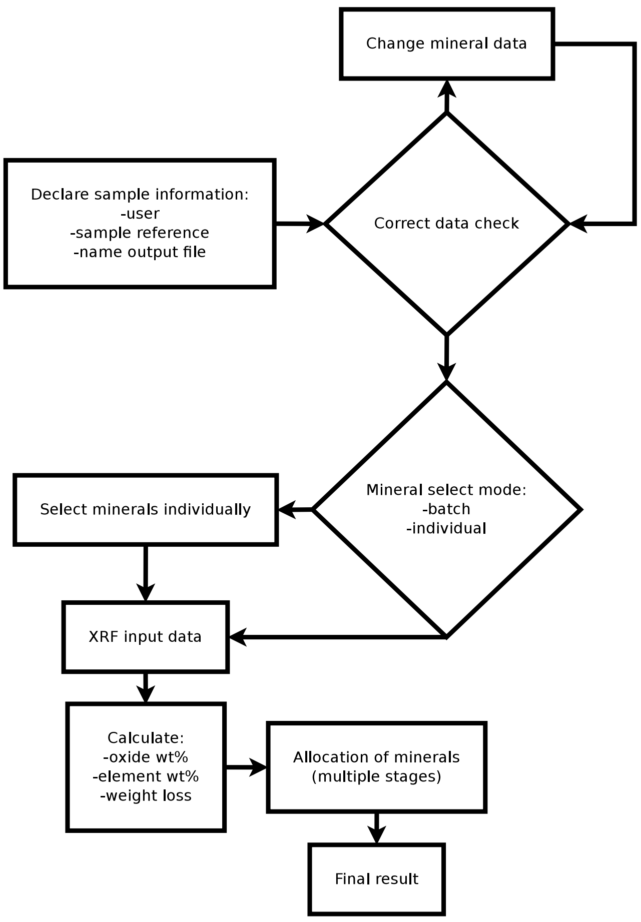


Figure 4.1: Flowdiagram Mincomp.

4.1.1 Step 1 - Provide metadata and check data

As the user starts the program, either from the .exe in Windows or in a Linux shell, the program asks to provide metadata about the user and the sample, to make it easier to look up the results after the result has been calculated.

Mincomp then displays the inventory of minerals, with the used densitys and molar weights. The user can change these values if he likes, but it is not necessary for the program to function.

4.1.2 Step 2 - XRD input data

The data gained from XRD and XRF tests are not readily usable for the program. For example, they could be delivered in a .docx or .pdf file and these files can't be read by the program.
The XRD data has to be inserted first, the program will prompt a choice for included minerals in the calculation process.
The user can choose between *Batch* and *Individual* mode. When choosing *Batch* the program will include all minerals in the calculation process and tries to calculate an amount for each mineral. When choosing *Individual* mode the user can individually select minerals which will be included in the calculation process.
The Batch mode is useful when knowledge about the sample is scarce, the results from Mincomp can give a first insight in a likely mineralogical bulk composition. The Individual mode is useful when there is better knowledge about the rock's mineral content, as only the minerals present in the rock can be selected and included in the calculation process, yielding a more accurate result.

4.1.3 Step 3 - XRF input data

To insert the XRF results into Mincomp is the next step. The program will ask for the amount of the following element oxides, in order of increasing element weight: F, Na_2O, MgO, Al_2O_3, SiO_2, P_2O_5, P, SO_3, S, Cl, K_2O, CaO, TiO_2 and Fe_2O_3. The data from an XRF analysis is usually in the form of weight percentages element-oxides, for example: 33,2 wt% Al_2O_3. The XRF input data is converted to molar quantities since Mincomp doesn't calculate with molar percentages, but with molar quantities. A sample weight has to be entered in order to convert to molar quantities, when this field is left blank, Mincomp then uses a default weight of 1000.0mg.

4.1.4 Step 4 - first allocation stage

The allocation of minerals is split into 3 stages, with in the first stage the allocation of trace minerals. These are common sedimentary minerals which usually contain the amount of the elements allocated in stage 1. The whole amount of these elements is allocated to these trace minerals. The following minerals are allocated:

Mineral	Control oxide
Pyrite	S
Rutile	Ti
Halite	Cl
Anhydrite	SO_3
Apatite	P

These minerals are allocated to the corresponding trace elements, and therefore allocated first.

4.1.5 Step 4 - second allocation stage

During the second allocation stage the alumina-silicates are allocated, this process is less straightforward than the first allocation stage, and consists of more minerals. If different minerals with the same controlling oxide are proven to be present from the XRD results, then arbitrary choices in allocation have to be made. For example: Muscovite and Illite are both calculated based on the available amount of K_2O, since it is not possible to quantify the individual amounts of these minerals based on XRF data, an arbitrary distribution of K_2O has to be made. The user can specify this distribution and is not bounded by the options provided by the program. The following minerals are allocated:

Mineral	Control oxide
Chlorite	Mg,Fe
Glauconite	K,Na,Mg
Muscovite	K
Illite	K
Montmorillonite	Ca,Na,Mg
Albite	Na
Anorthite	Ca
Orthoclase	K

4.1.6 Step 5 - third allocation stage

With the trace elements and the alumina-silicates allocated, the remainder is usually made up from quartz and carbonates. There are still a few options available in the third allocation stage. Excess Al_2O_3 can be allocated to Gibbsite or Kaolinite, excess Fe_2O_3 can be allocated to Hematite or Siderite, excess CaO can be allocated to Calcite or Dolomite and excess MgO can be allocated to Magnesite or Dolomite. The availability of these options depend on the selected minerals for calculation and the availability of a specific element in this allocation stage. When different minerals with the same element have to be calculated, an arbitrary division has to be made; this is explained in section 4.1.5. The third allocation stage consists of the following minerals:

Mineral	Control oxide
Hematite	Fe
Gibbsite	Al
Goethite	Fe
Calcite	Ca
Dolomite	Ca,Mg
Magnesite	Mg
Siderite	Fe
Kaolinite	Al
Quartz	Si

4.1.7 Step 6 - Conversion of data and final result

The contents are calculated with molar quantities, since the input data were provided in weight percentages, the output data is converted back to weight percentages.
Since the minerals won't exactly add up to 100 % the data are normalized to 100 %. Bear in mind that the outcome is a *likely* mineralogical bulk composition and that the calculation process is based upon some assumptions. Therefore the result is not an exact match to the rocks mineral content. The final result is presented in both weight percentages and volume percentages of the sample, also a graph is made to quickly review the result of the program.

Chapter 5

Results

Mincomp has been tested for usability and reliability, but comparison with similar programs is the most important part to verify results. Rock sample data presented by other authors are used to calculate a result with Mincomp and this result is compared to the outcome of other programs.

The comparison with each program is divided into two parts. The first part is a comparison with Mincomp with all minerals included in the calculation sequence. This way, Mincomp tries to calculate an amount for each mineral specified in it's calculation list. The result is not necessarily accurate but could provide some first insights in a possible mineralogical bulk composition.
The second part is a comparison with Mincomp with exactly the same minerals as were calculated in the other program, this has been carried out to minimize differences in the calculation process and therefore minimize the differences between the results. The aim is to calculate an accurate result which doesn't differ that much from the results of other programs. Differences are acceptable, but have to be explainable.

The results of Mincomp are compared with the results of other programs by the use of the datasets presented by the authors of Sednorm (Cohen and Ward, 1991) , as they were one of the first developers of a normative calculation program, most other programs also refer to this dataset, therefore this was the easiest way to compare results.
Caritat et al. (1994) also used this data-set for comparison with their program, LPNorm, they used the Bersham Mudstone (Nicholls, 1962) for comparison.
Rosen et al. (2004) didn't use the Sednorm dataset for reference, but the authors presented another dataset for testing, this set is included in this report, since Mincomp has been comparised with Minlith as well.
Posch and Kurz (2007) didn't present any test result for A2M unfortunately, so a comparison was not possible. This was also the case with the program Modan of Paktunc (1998), they didn't present test results, so comparison with Modan is also not possible.

5.1 Comparison with Sednorm

Cohen and Ward (1991) used the following datasets:

	Carbonate-altered lithic siltstone (Ward et al., 1990)	Bersham Mudstone (Nicholls, 1962)	Average sedimentary rock (Garrels and Mackenzie, 1971)
Element oxide	wt%	wt%	wt%
Na_2O	0.8	0.6	0.9
MgO	1.8	0.3	2.6
Al_2O_3	15.5	20.6	14.6
SiO_2	52.5	62.6	59.7
P_2O_5	0.2	0.2	0.0
SO_3	0.04	0.02	0.0
Cl	0.0	0.0	0.0
K_2O	1.2	3.3	3.2
CaO	8.9	0.3	4.8
TiO_2	0.8	0.9	0.0
Fe_2O_3	4.1	1.1	4.8
MnO	0.1	0.02	0.0
H_2O	1.5	4.8	3.4
CO_2	11.2	0.9	4.7

Table 5.1: Datasets presented by Cohen and Ward (1991), for the program Sednorm.

5.1.1 Comparison with all minerals selected

Carbonate-altered lithic siltstone

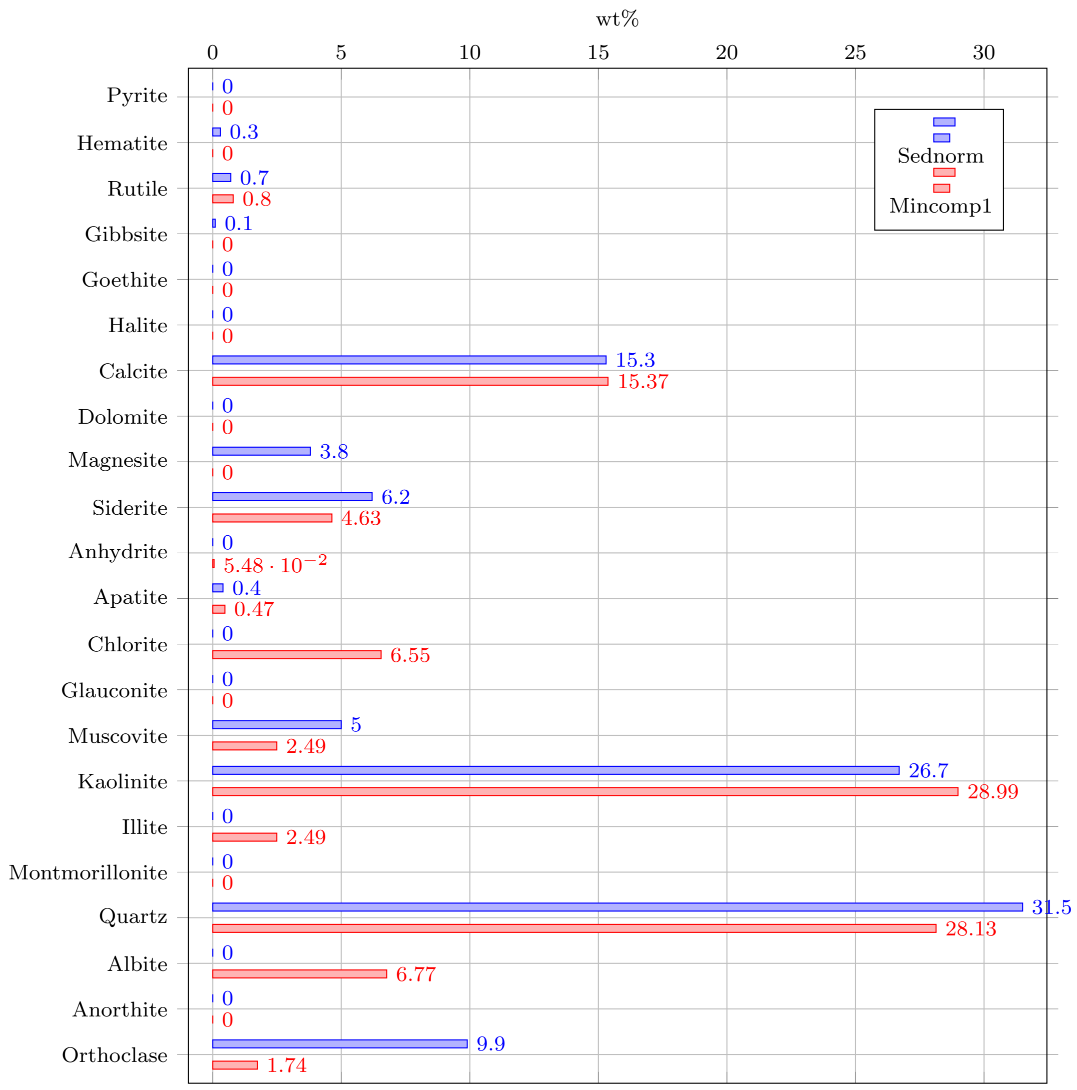


Figure 5.1: Results for Carbonate-altered lithic siltstone dataset (Ward et al., 1990).

When we take a look at the result of Mincomp in comparison with the result of Sednorm, we see that in general the results are much alike and that the differences mainly occur because every mineral is included in Mincomp's calculation process for this run. There is not much difference between the result for the minerals that are mainly present in the sample, the differences between Kaolinite and Quartz are only 3%.

Mincomp doesn't calculate an amount for Magnesite, contrary to Sednorm, this is because magnesium is allocated to Chlorite in an earlier stage. Sednorm uses Magnesite to allocate excess Magnesium.
Sednorm shows a higher percentage of Siderite, and does include Hematite as well, contrary to Mincomp. Iron is partly allocated to Chlorite in the second allocation stage, excess iron is allocated to Siderite only.
Since it is not possible to quantify the exact amounts of Muscovite and Illite when both are present in the calculation process, an arbitrary division had to be made. In this case an even distribution has been chosen. The sum of the weight percentages of Muscovite and Illite is equal to the amount of Muscovite calculated by Sednorm.
Contrary to Sednorm, Mincomp has calculated an amount for Albite, since it was included and enough weight was available to allocate an amount to Albite.
Mincomp calculated a lower amount of Orthoclase than Sednorm, in Mincomp Orthoclase is calculated after allocation of potassium to Illite and Muscovite, opposite to Sednorm which calculates Orthoclase before Illite and Muscovite.

Bersham Mudstone

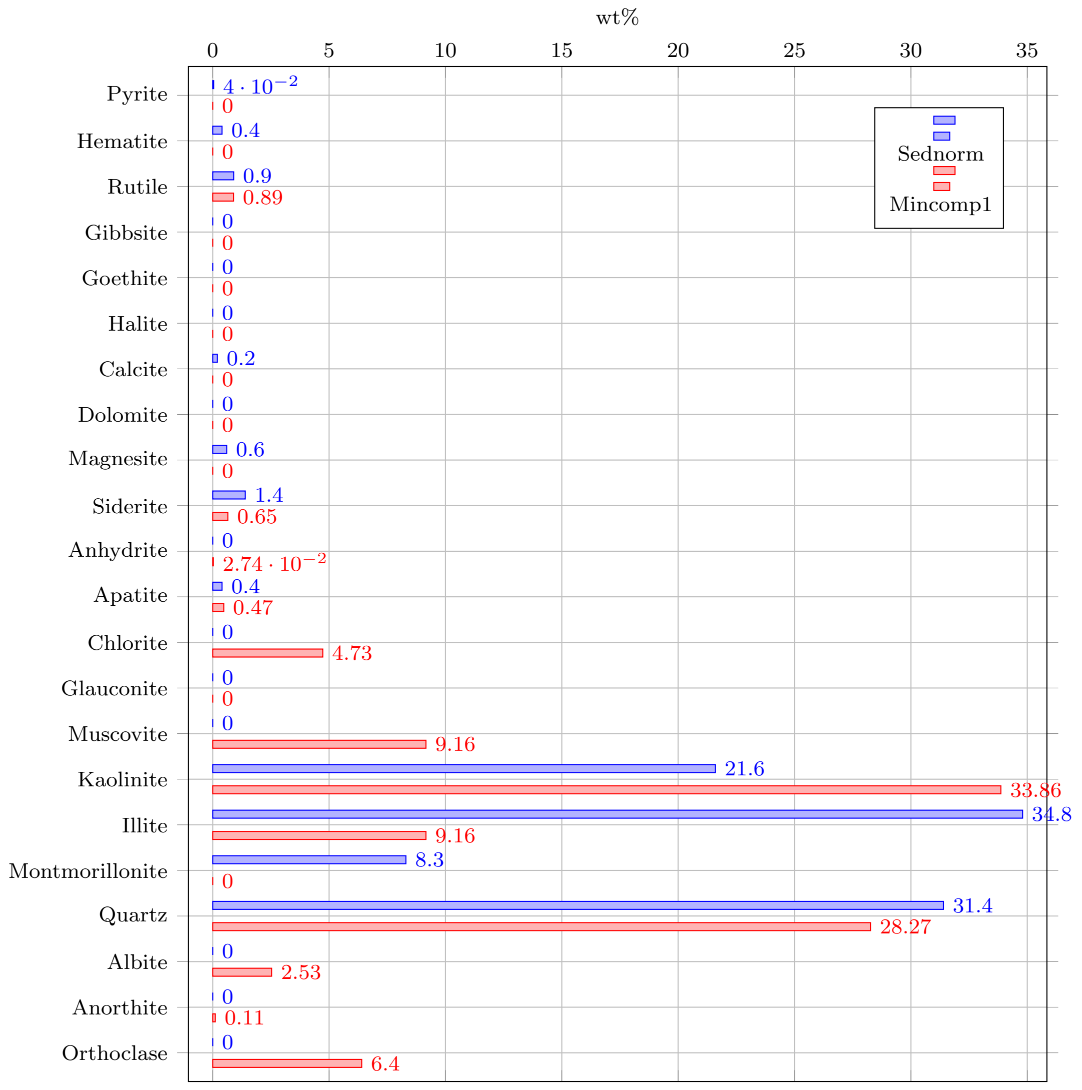


Figure 5.2: Results for Bersham Mudstone dataset (Nicholls, 1962).

The results for the Bersham Mudstone dataset (Nicholls, 1962) show roughly the same differences as for the Carbonate-altered lithic siltstone dataset (Ward et al., 1990).
We see a difference in the amount of Chlorite, Mincomp calculates an amount because Chlorite is included in the calculation process, while Sednorm doesn't.
For the Bersham Mudstone the authors didn't include Muscovite in the calculation process but decided to allocate all K_2O to Illite. In Mincomp this amount is distributed to three different minerals; Muscovite, Illite and Orthoclase, hence the difference.
Mincomp does calculate a higher amount for Kaolinite, this is because of its position in the calculation process.
The difference in wt% Montmorillonite can be explained by the difference in the used chemical composition, Sednorm uses $NaMgAl_3[Si_8O_{20}](OH)_4$ while Mincomp uses $Ca_{0.17}Na_{0.31}Mg_{0.33}Al_{1.67}Si_4O_{10}(OH)_{2,61}$. Since there is almost no CaO present in the sample, and the little amount of CaO is used to allocate Anhydrite, there is none left to calculate an amount for Montmorillonite.

5.1.2 Comparison with exactly the same minerals

Carbonate-altered lithic siltstone

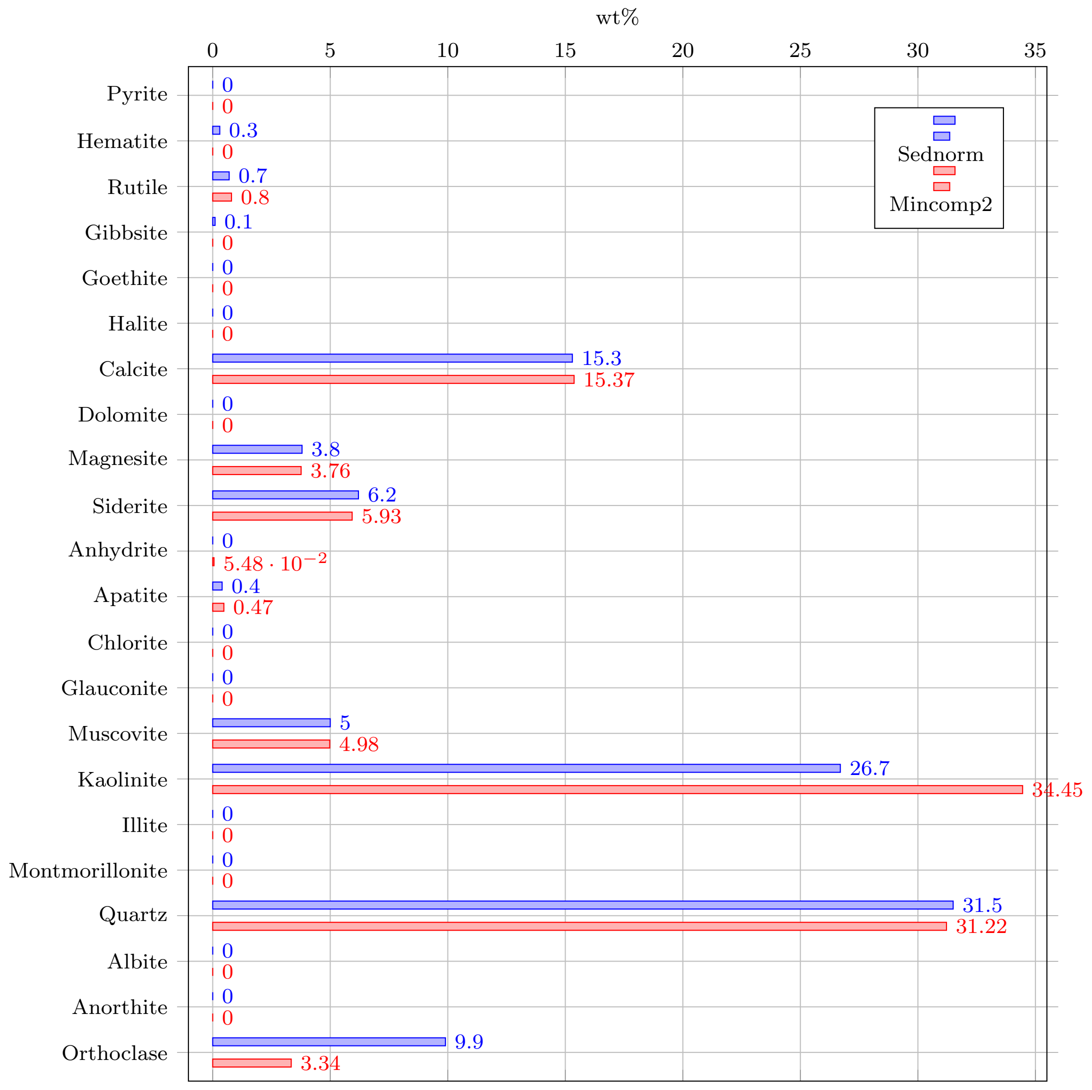


Figure 5.3: Results for Carbonate-altered lithic siltstone dataset (Ward et al., 1990).

The results for the Carbonate-altered lithic siltstone dataset are much alike, the only big difference is in Kaolinite and Orthoclase.
Mincomp calculates a higher amount of Kaolinite, this is because of the allocation order as earlier explained, the amount of Orthoclase is lower because of the lower availability of Al_2O_3 after allocation of Kaolinite.

Bersham Mudstone

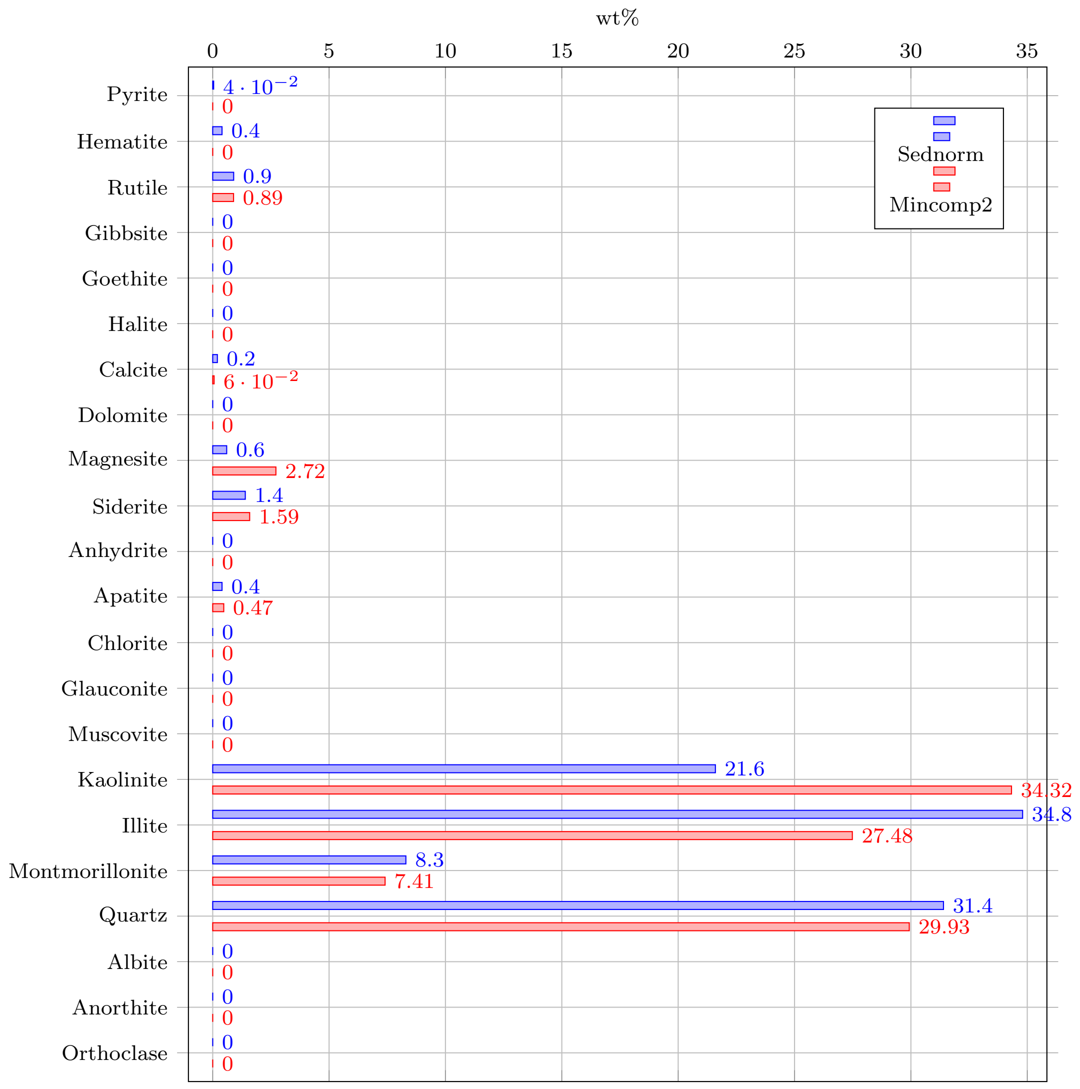


Figure 5.4: Results for Bersham Mudstone dataset (Nicholls, 1962).

For most of the minerals there are only minor differences in the results, the differences for Kaolinite, Illite, Montmorillonite and Quartz are a greater.
Sednorm calculates a lower amount of Kaolinite than Mincomp, and a higher amount of Illite, while the chemical composition of Illite is different from the used chemical composition in Mincomp, the allocation order is the main reason for the difference in the amount of these minerals.

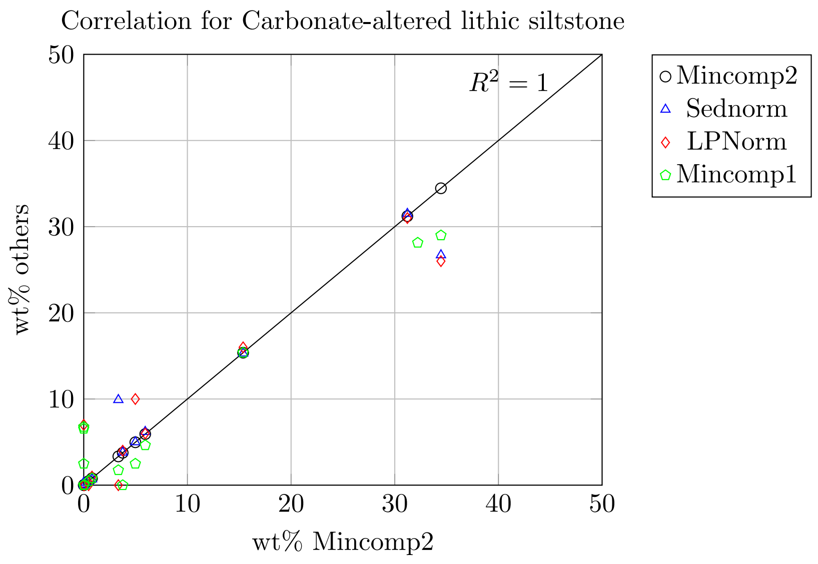
5.1.3 Combined graphs and tables

Mineral	Sednorm wt%	Mincomp-1 wt%	Mincomp-2 wt%
Pyrite	0.0 %	0.0 %	0.0 %
Hematite	0.3 %	0.0 %	0.0 %
Rutile	0.7 %	0.7987 %	0.7987 %
Gibbsite	0.1 %	0.0 %	0.0 %
Goethite	0.0 %	0.0 %	0.0 %
Halite	0.0 %	0.0 %	0.0 %
Calcite	15.3 %	15.3738 %	15.3738 %
Dolomite	0.0 %	0.0 %	0.0 %
Magnesite	3.8 %	0.0 %	3.7607 %
Siderite	6.2 %	4.6344 %	5.9320 %
Anhydrite	0.0 %	0.05478 %	0.05478 %
Apatite	0.4 %	0.4739 %	0.4739 %
Chlorite	0.0 %	6.5493 %	0.0 %
Glauconite	0.0 %	0.0 %	0.0 %
Muscovite	5.0 %	2.4895 %	4.9791 %
Kaolinite	26.7 %	28.9876 %	34.4543 %
Illite	0.0 %	2.4895 %	0.0 %
Montmorillonite	0.0 %	0.0 %	0.0 %
Quartz	31.5 %	28.134 %	31.216 %
Albite	0.0 %	6.7657 %	0.0 %
Anorthite	0.0 %	0.0 %	0.0 %
Orthoclase	9.9 %	1.7395 %	3.3398 %

Table 5.2: Program results for Carbonate-altered lithic siltstone data (Ward et al., 1990). Mincomp-1 refers to the first run with all minerals included. Mincomp-2 refers to the second run with individually selected minerals.

Mineral	Sednorm wt%	Mincomp-1 wt%	Mincomp-2 wt%
Pyrite	0.04 %	0.0 %	0.0 %
Hematite	0.4 %	0.0 %	0.0 %
Rutile	0.9 %	0.8945 %	0.8945 %
Gibbsite	0.0 %	0.0 %	0.0 %
Goethite	0.0 %	0.0 %	0.0 %
Halite	0.0 %	0.0 %	0.0 %
Calcite	0.2 %	0.0 %	0.06 %
Dolomite	0.0 %	0.0 %	0.0 %
Magnesite	0.6 %	0.0 %	2.7151 %
Siderite	1.4 %	0.6488 %	1.5873 %
Anhydrite	0.0 %	0.0274 %	0.0 %
Apatite	0.4 %	0.4739 %	0.4739 %
Chlorite	0.0 %	4.7284 %	0.0 %
Glauconite	0.0 %	0.0 %	0.0 %
Muscovite	0.0 %	9.1615 %	0.0 %
Kaolinite	21.6 %	33.8644 %	34.3226 %
Illite	34.8 %	9.1614 %	27.484 %
Montmorillonite	8.3 %	0.0 %	7.4068 %
Quartz	31.4 %	28.266 %	29.93 %
Albite	0.0 %	2.5306 %	0.0 %
Anorthite	0.0 %	0.1116 %	0.0 %
Orthoclase	0.0 %	6.4014 %	0.0 %

Table 5.3: Mincomp and Sednorm results compared on Bersham Mudstone dataset. Mincomp-1 refers to the first run with all minerals included. Mincomp-2 refers to the second run with individually selected minerals.



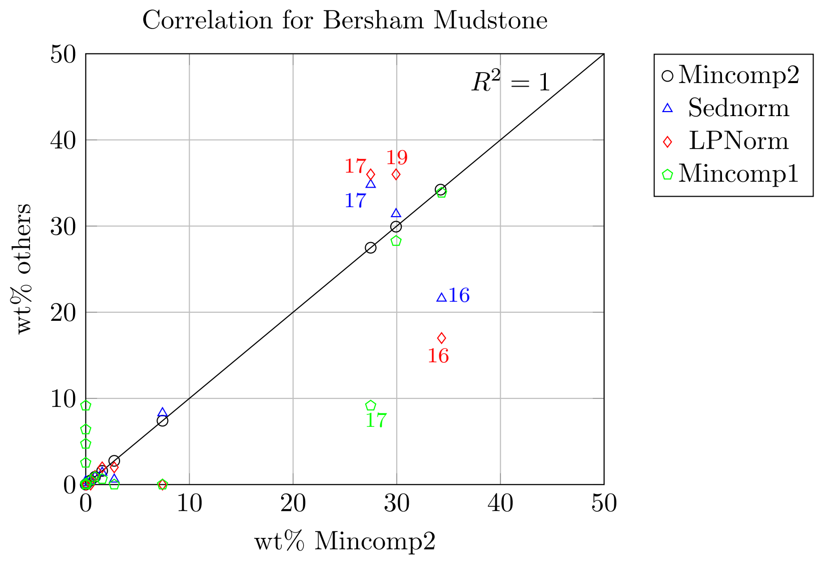


Figure 5.5: Results for Bersham Mudstone data. **16** - Kaolinite, **17** - Illite, **19** - Quartz.

5.2 Comparison with Minlith

Rosen et al. (2004) used the following dataset:

Element oxide	Weight percentage
Na_2O	0.43 %
MgO	3.04 %
Al_2O_3	17.9 %
SiO_2	66.57 %
P_2O_5	0.01 %
P	0.0 %
SO_3	0.0 %
S	0.026 %
Cl	0.0 %
K_2O	4.13 %
CaO	0 %
TiO_2	0.73 %
Fe_2O_3	0.85 %
FeO	3.01 %
$(FeO)_t$	3.78 %
MnO	0.01 %
C	0.11 %

Table 5.4: Chemical Analysis from Mumme et al. (1996) for the program Minlith

The authors of Minlith use a ferriferous coefficient to determine the amount of iron in the mineral Chlorite, therefore they calculate FeO_t with : $FeO_t = 0.9 \cdot FeO + Fe_2O_3$
In order to compare, the total Fe is calculated and divided by 2, to estimate the amount of Fe_2O_3 since Mincomp doesn't include FeO.

Based on these values, they calculated the mineral quantities of Carbon, Rutile, Pyrite, Albite, Chlorite, Illite, Orthoclase, Serpentine, Pyrolusite and Quartz. In the following table, the result calculated with Mincomp is also presented. Note, that some of the calculated minerals in Minlith are not present in Mincomp and vice-versa, therefore comparison is unfortunately not that accurate.
The authors of Minlith used Serpentine $Mg_3[Si_2O_5](OH)_4$ and Pyrolusite MnO_2 , while Mg is used for allocating serpentine, and Mn is used for Pyrolusite.

Manganese is not included in Mincomp, Mincomp is focused on accurate calculation of the bulk of the material, not on trace amounts of rarer elements, therefore Pyrolusite is not included. Serpentine also isn't included in Mincomp, serpentine is usually present in magmatic rocks for example in Dunite (Deer et al., 1966), but is not that common in sedimentary rocks. Magnesium is used for calculating Dolomite or Montmorillonite. Since there is only one test result available for Minlith a comprehensive comparison is not possible, also the lack of several minerals (Muscovite, Glauconite, Hematite and Anhydrite) in Minlith does not support the comparison, instead two less common minerals are included.
In general the results are quite similar but a more thorough comparison would have been favourable.

5.2.1 Comparison with all minerals selected

S3 sample

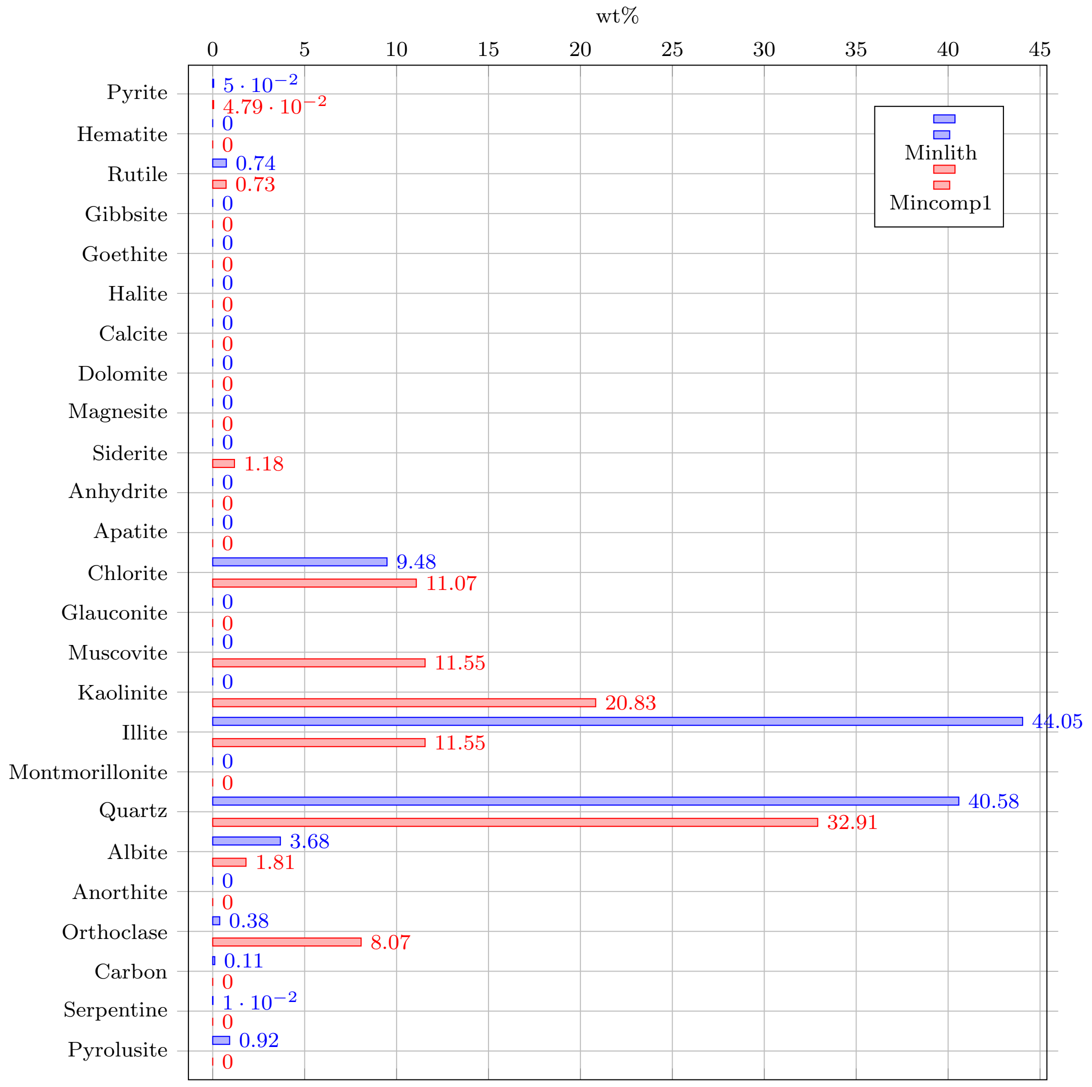


Figure 5.6: Results for S3 dataset (Mumme et al., 1996).

The big difference between results is the distribution of potassium between different minerals. In the first Mincomp run every mineral was included, therefore the available amount of potassium has been distributed between Illite, Muscovite and Orthoclase. Minlith doesn't divide the available mass between different minerals like Mincomp does. The amount of Illite is therefore much lower in Mincomp than in Minlith. Another reason is a difference in the molecular formula of Illite, the authors of Minlith incorporate iron and magnesium as well, resulting in a much greater molar mass of the mineral.
Mincomp calculates an amount for Kaolinite, while Minlith didn't incorporate Kaolinite in their calculation.

5.2.2 Comparison with exactly the same minerals

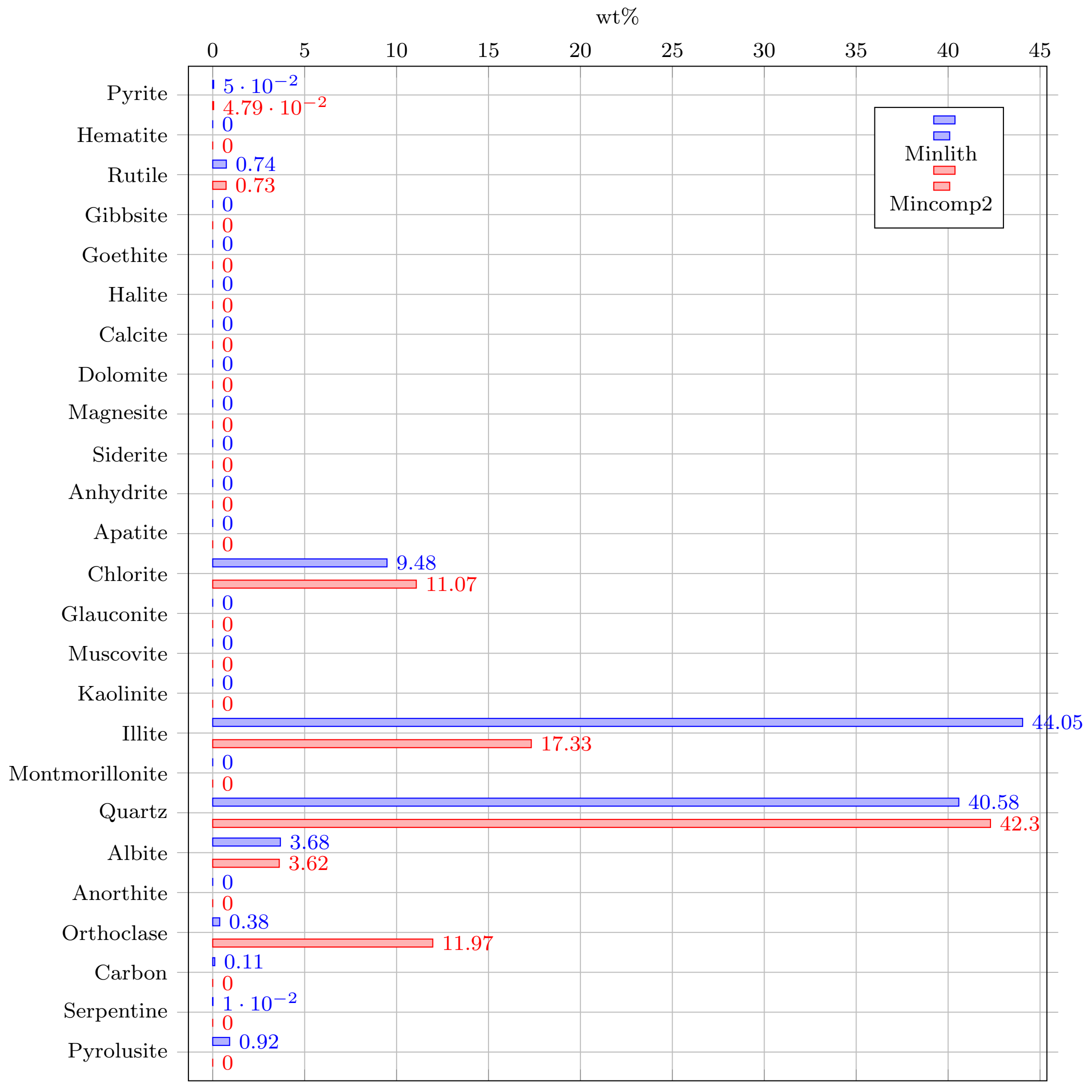


Figure 5.7: Results for S3 dataset (Mumme et al., 1996).

When we look at the second run with Mincomp, the differences become smaller. In general the results are very similar, minor differences in most minerals, and only great differences in Illite and Orthoclase. As K_2O is divided between Orthoclase and Illite, the amount of Illite becomes much larger than the amount of Illite calculated by Minlith, Minlith also incorporates Mg and Fe in the chemical composition of Illite. (Rosen et al., 2004)

5.2.3 Combined graphs and tables

Mineral	Minlith wt%	Mincomp-1 wt%	Mincomp-2 wt%
Pyrite	0.05 %	0.0479 %	0.0479 %
Hematite	-	0.0 %	0.0 %
Rutile	0.74 %	0.7268 %	0.7268 %
Gibbsite	0.0 %	0.0 %	0.0 %
Goethite	0.0 %	0.0 %	0.0 %
Halite	0.0 %	0.0 %	0.0 %
Calcite	0.0 %	0.0 %	0.0 %
Dolomite	0.0 %	0.0 %	0.0 %
Magnesite	0.0 %	0.0 %	0.0 %
Siderite	0.0 %	1.1818 %	0.0 %
Anhydrite	-	0.0 %	0.0 %
Apatite	0.0 %	0.0 %	0.0 %
Chlorite	9.48 %	11.0722 %	11.0722 %
Glauconite	-	0.0 %	0.0 %
Muscovite	-	11.5515 %	0.0 %
Kaolinite	0.0 %	20.8293 %	0.0 %
Illite	44.05 %	11.551 %	17.327 %
Montmorillonite	0.0 %	0.0 %	0.0 %
Quartz	40.58 %	32.905 %	42.303 %
Albite	3.68 %	1.8094 %	3.6189 %
Anorthite	0.0 %	0.0 %	0.0 %
Orthoclase	0.38 %	8.0713 %	11.9678 %
Carbon	0.11 %	-	-
Serpentine	0.01 %	-	-
Pyrolusite	0.92 %	-	-

Table 5.5: Minlith norms compared to Mincomp on Mumme et al (1996) S3 sample. Mincomp-1 refers to the first run, with all minerals included. Mincomp-2 referst to the second run, with individually selected minerals.

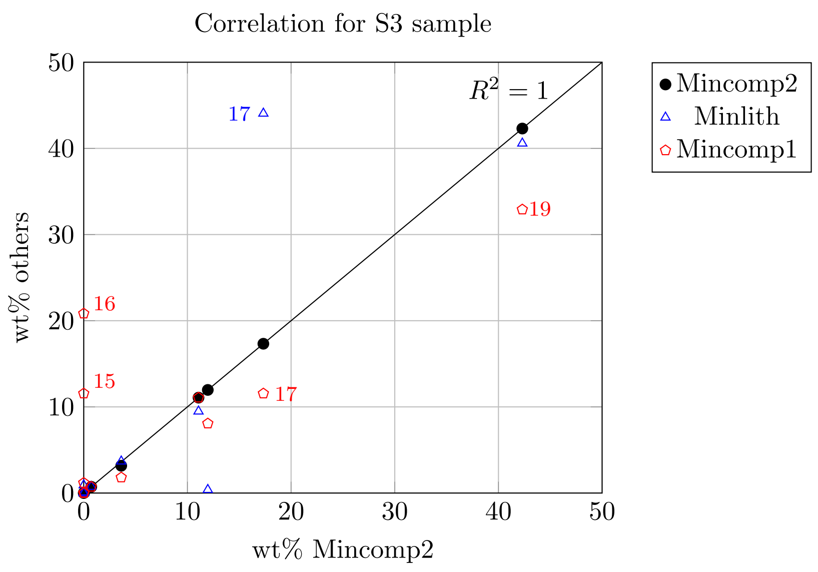


Figure 5.8: Results for S3 Sample. **15** - Muscovite, **16** - Kaolinite, **17** - Illite, **19** - Quartz.

5.3 Comparison with LPNorm

5.3.1 Comparison with all minerals selected

Carbonate-altered lithic siltstone

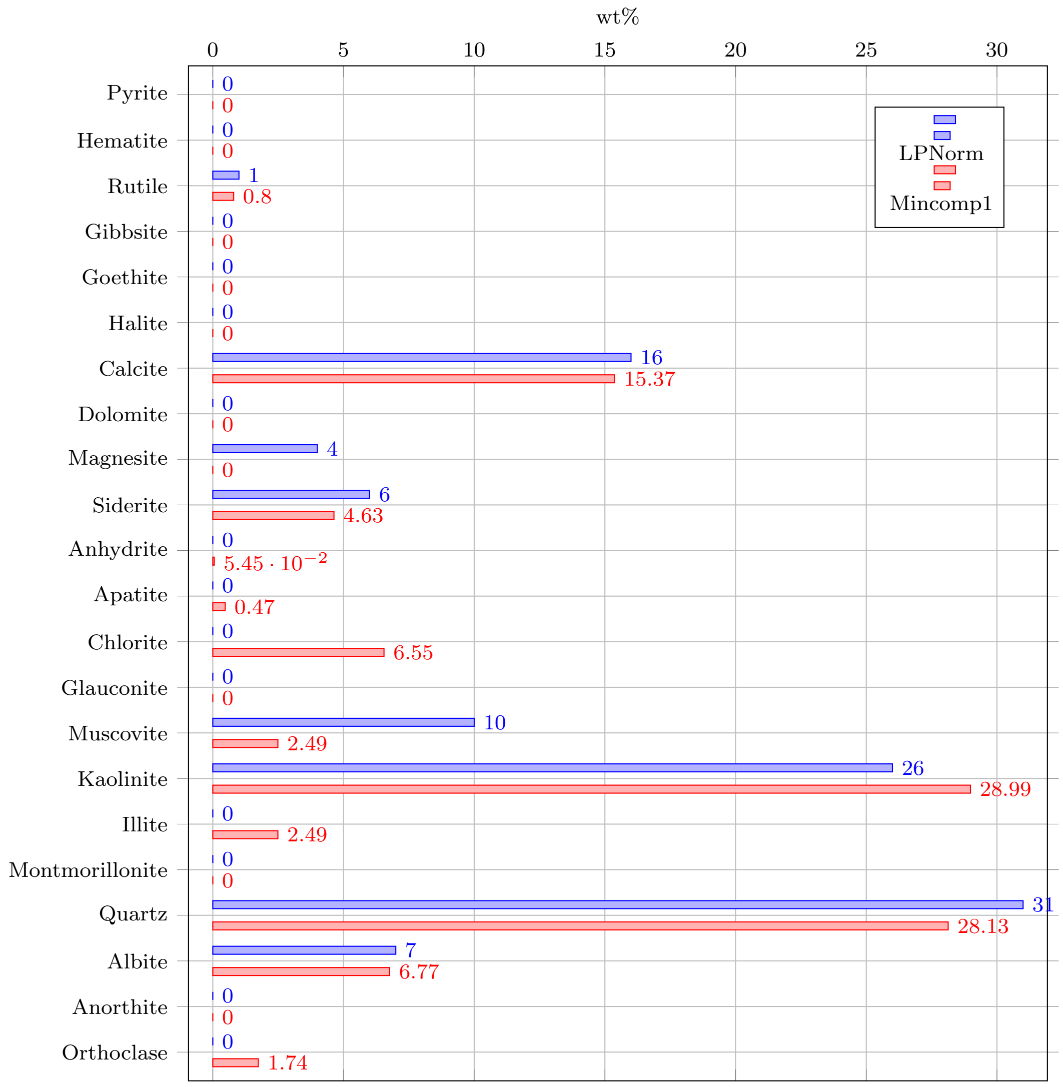


Figure 5.9: Results for Carbonate-altered lithic siltstone dataset (Ward et al., 1990).

The results are in general alike, with only big differences in the amounts of Muscovite, Orthoclase, Chlorite and Magnesite.
The difference in the amounts of Magnesite and Chlorite is explained by the distribution of MgO to Chlorite in allocation stage 2, therefore no MgO is available for allocation of Magnesite in allocation stage 3.
The difference in Muscovite is because of the distribution of K_2O to Muscovite, Illite and Orthoclase, Minlith allocates all K_2O to Muscovite for this sample.

Bersham Mudstone

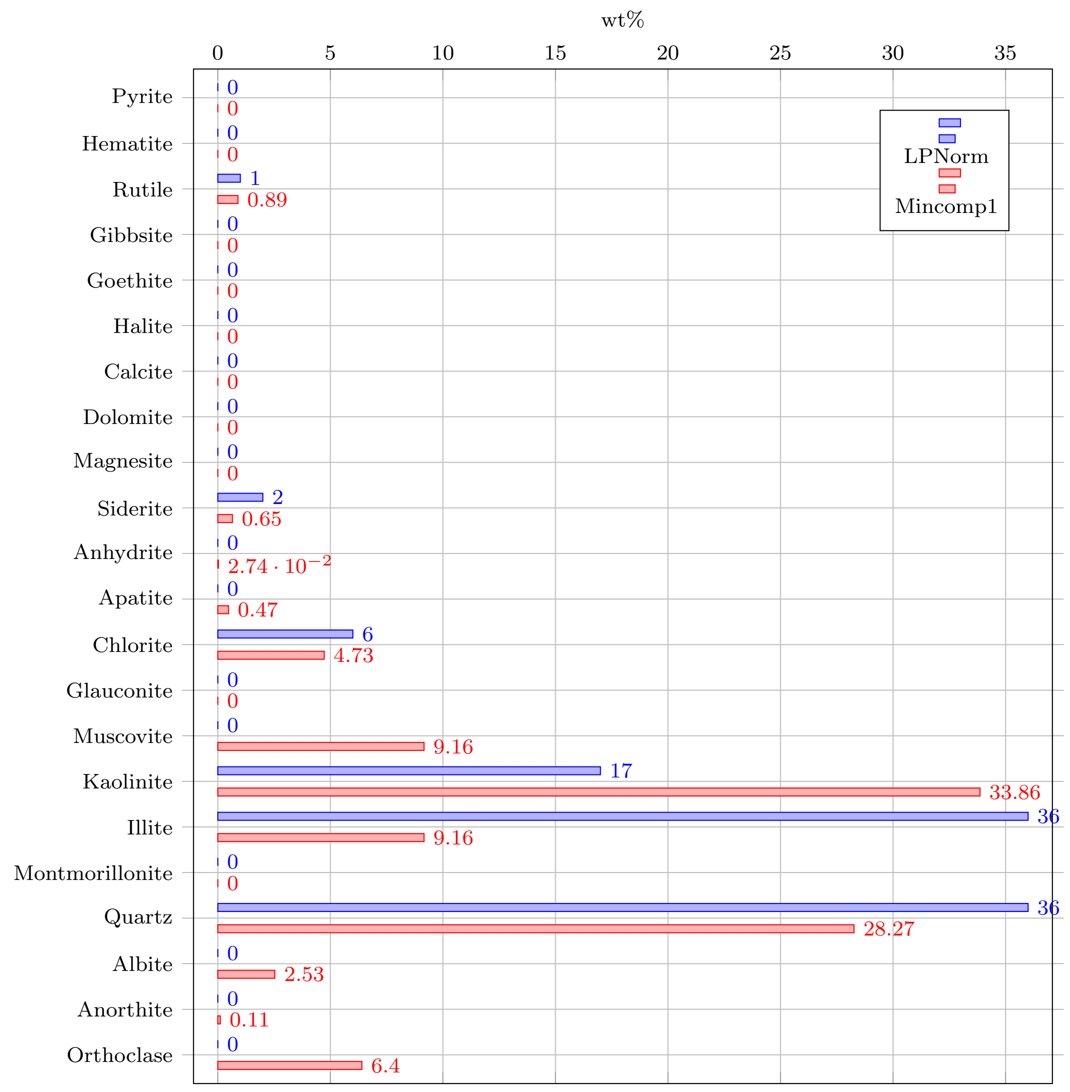


Figure 5.10: Results for Bersham Mudstone dataset (Nicholls, 1962).

The difference in Muscovite/Illite/Orthoclase is explained earlier, and is because of the distribution of K_2O to these three minerals, therefore Mincomp calculates an amount for Orthoclase and Muscovite while Minlith only included Illite.
The amount of Kaolinite is a lot higher, because of the allocation of Kaolinite prior to Illite.
The amount of Quartz is lower in the results of Mincomp, this is because a great deal of SiO_2 is allocated to alumina-silicates.

5.3.2 Comparison with exactly the same minerals

Carbonate-altered lithic siltstone

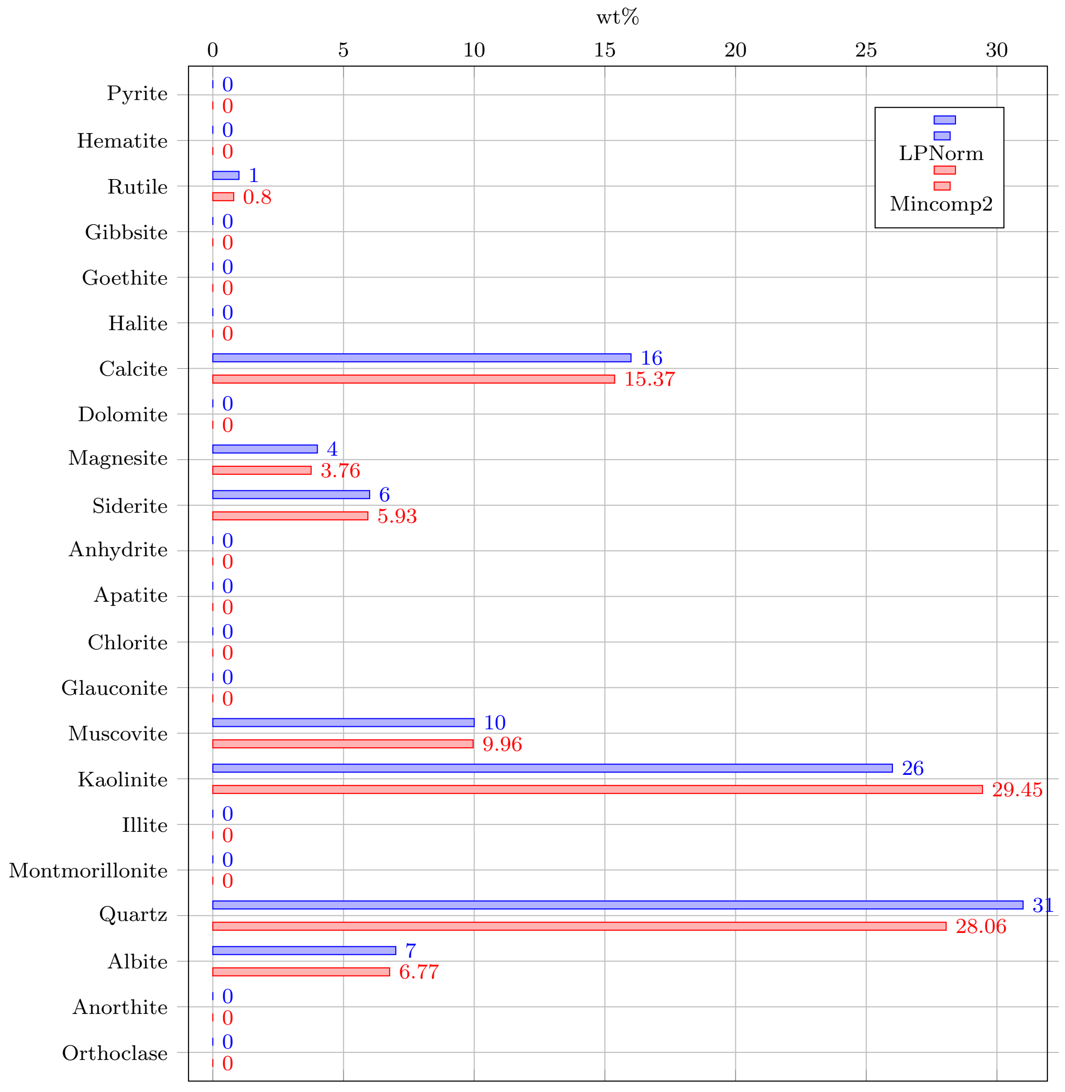


Figure 5.11: Results for Carbonate-altered lithic siltstone dataset (Ward et al., 1990).

The result of both programs are much alike, there are only minor differences in the results for Kaolinite and Quartz. The surplus of Kaolinite is compensated by a lower amount of Quartz.

Bersham Mudstone

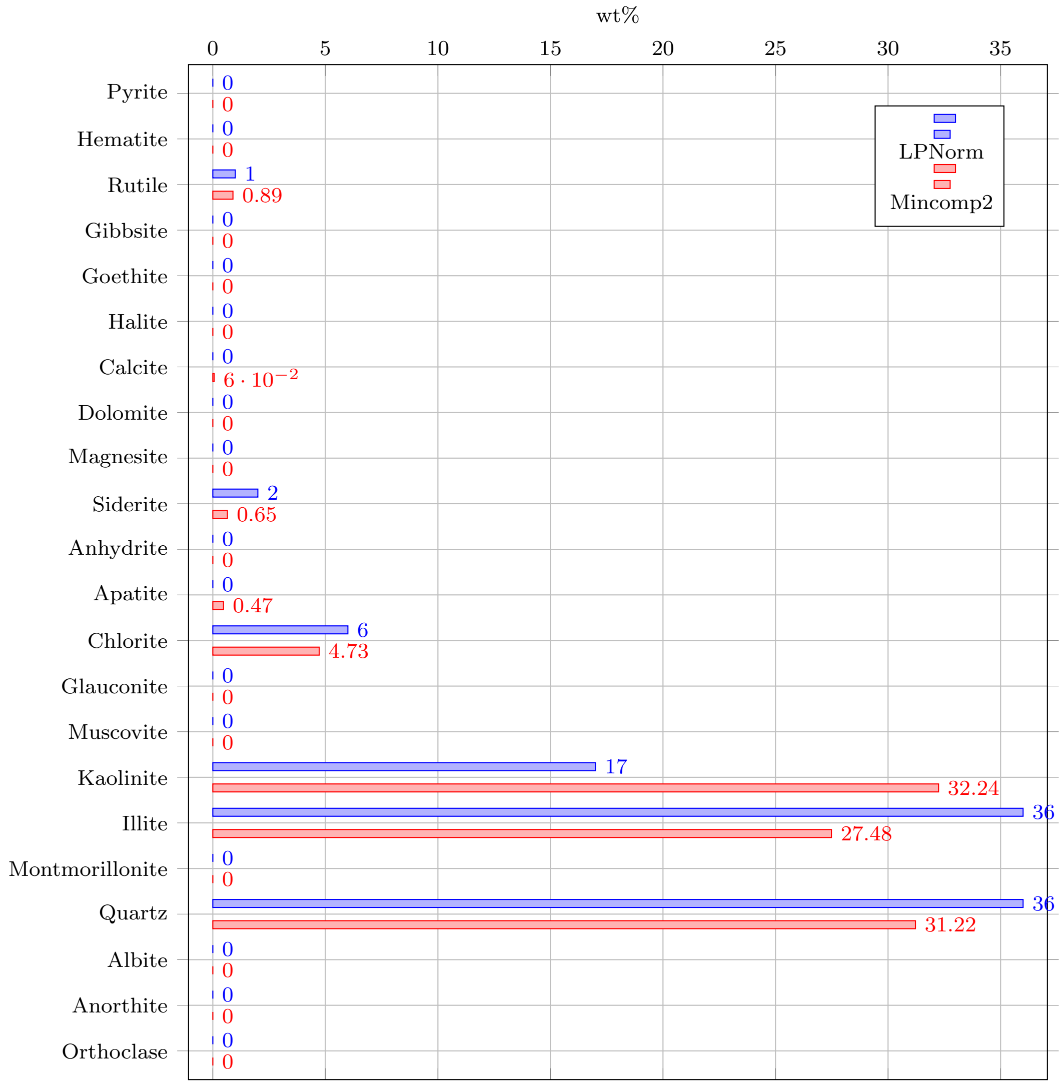


Figure 5.12: Results for Bersham Mudstone dataset (Nicholls, 1962).

The obvious differences are observed for Kaolinite and Illite, Mincomp calculating a 15% higher amount for Kaolinite and a 9% lower amount for Illite. This is again due to the allocation used in the program, a higher amount is allocated to Kaolinite this way.
The lower amount of Quartz is due to the fact of the high amount of SiO_2 allocated to both Illite and Kaolinite.

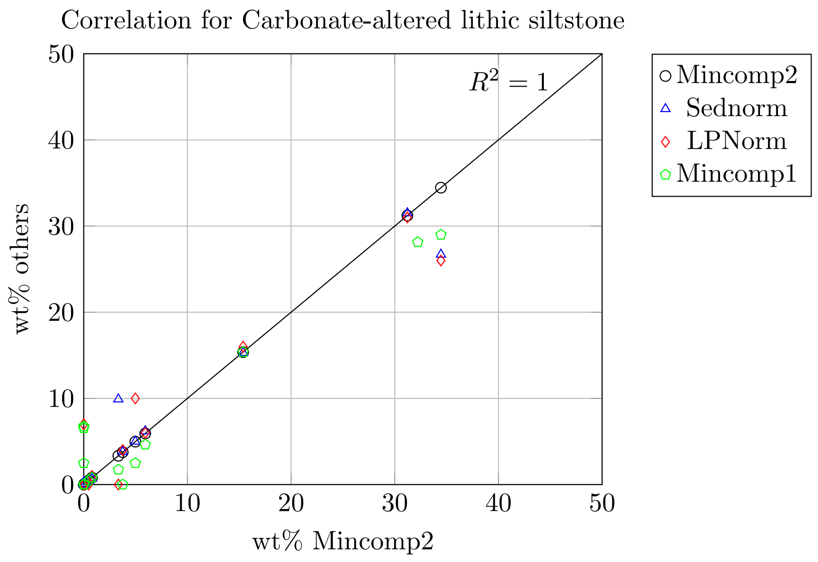
5.3.3 Combined graphs and tables

Mineral	LPNorm wt%	Mincomp-1 wt%	Mincomp-2 wt%
Pyrite	0 %	0.0 %	0.0 %
Hematite	0 %	0.0 %	0.0 %
Rutile	1 %	0.7987 %	0.7987 %
Gibbsite	0 %	0.0 %	0.0 %
Goethite	0 %	0.0 %	0.0 %
Halite	0 %	0.0 %	0.0 %
Calcite	16 %	15.3738 %	15.8842 %
Dolomite	0 %	0.0 %	0.0 %
Magnesite	4 %	0.0 %	3.761 %
Siderite	6 %	4.6344 %	5.932 %
Anhydrite	0 %	0.05478 %	0.0 %
Apatite	0 %	0.4739 %	0.0 %
Chlorite	0 %	6.5493 %	0.0 %
Glauconite	0 %	0.0 %	0.0 %
Muscovite	10 %	2.4895 %	9.958 %
Kaolinite	26 %	28.9876 %	29.446 %
Illite	0 %	2.4895 %	0.0 %
Montmorillonite	0 %	0.0 %	0.0 %
Quartz	31 %	28.134 %	28.056 %
Albite	7 %	6.7657 %	6.7654 %
Anorthite	0 %	0.0 %	0.0 %
Orthoclase	0 %	1.7395 %	0.0 %

Table 5.6: LPNorm norms compared to Mincomp, Carbonate-altered lithic siltstone. (Ward et al., 1990)

Mineral	LPNorm wt%	Mincomp-1 wt%	Mincomp-2 wt%
Pyrite	0 %	0.0 %	0.0 %
Hematite	0 %	0.0 %	0.0 %
Rutile	1 %	0.8945 %	0.8945 %
Gibbsite	0 %	0.0 %	0.0 %
Goethite	0 %	0.0 %	0.0 %
Halite	0 %	0.0 %	0.0 %
Calcite	0 %	0.0 %	0.06 %
Dolomite	0 %	0.0 %	0.0 %
Magnesite	0 %	0.0 %	0.0 %
Siderite	2 %	0.6488 %	0.6488 %
Anhydrite	0 %	0.0274 %	0.0 %
Apatite	0 %	0.4739 %	0.4739 %
Chlorite	6 %	4.7284 %	4.7284 %
Glauconite	0 %	0.0 %	0.0 %
Muscovite	0 %	9.1615 %	0.0 %
Kaolinite	17 %	33.8644 %	32.244 %
Illite	36 %	9.1614 %	27.484 %
Montmorillonite	0 %	0.0 %	0.0 %
Quartz	36 %	28.266 %	31.217 %
Albite	0 %	2.5306 %	0.0 %
Anorthite	0 %	0.1116 %	0.0 %
Orthoclase	0 %	6.4014 %	0.0 %

Table 5.7: Program results for Bersham Mudstone dataset (Nicholls, 1962).



5.4 Moduscalc

The authors of Moduscalc Laube et al. (1996) included two datasets with their article, it are two samples of cuttings of a hydrothermally altered siliciclastic sedimentary rock. They focus on the chemical end-member of the alteration process. For example, May (1994) showed that the alteration process was dominated by the mineral reaction: $chlorite \rightarrow kaolinite + dolomite + ankerite + siderite$.
However, many minerals are not included, which makes a comparison difficult and inaccurate. The authors included two extra minerals; Rhodochrosite and Al-Celadonite. Al-Celadonite makes up for relatively hihg percentages of the sample, which doesn't benefit the comparison.
The XRF-results of Laube et al. (1996) state the presence of FeO, while Mincomp uses Fe_2O_3, the number of moles of FeO was divided by 2 to estimate the amount of Fe_2O_3.

Element oxide	Sample LM41	Sample LM50
	wt%	wt%
F	0	0
Na_2O	0.11	0.16
MgO	1.48	1.18
Al_2O_3	18.43	17.01
SiO_2	59.41	61.28
P_2O_5	0.14	0.14
P	0	0
SO_3	0	0
S	0	0
Cl	0	0
K_2O	4.84	4.15
CaO	0.28	0.24
TiO_2	0.93	0.9
Fe_2O_3	0	0
FeO	3.072	5.956

Table 5.8: Chemical analyses from Laube et al. (1996) for Moduscalc

5.4.1 Comparison with all minerals selected

LM41 sample

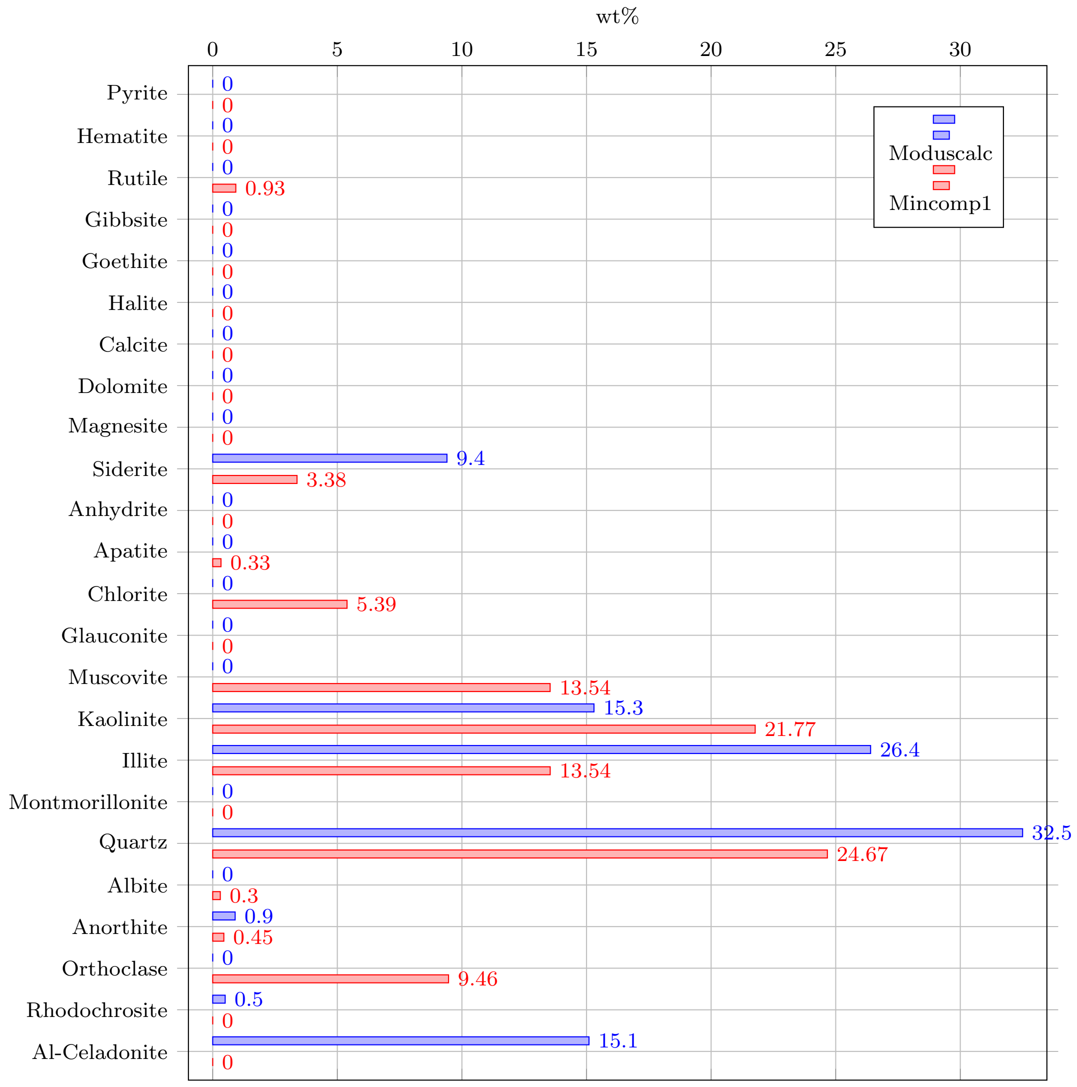


Figure 5.13: Results for LM41 dataset (Laube et al., 1996).

The difference in Siderite is explained by the allocation of Fe_2O_3 to Chlorite as well. Since all minerals were included and a sufficient amount of MgO was available, Chlorite was allocated.
Again, greater differences in the amounts of Muscovite/Illite and Kaolinite. This is of the distribution of K_2O to Muscovite, Illite and Orthoclase instead of only Illite in Minlith.
The higher amount of Kaolinite, due to the allocation order, makes up for the lower amount of Quartz. Mincomp doesn't include Rhodochrosite and Al-Celadonite.

LM50 sample

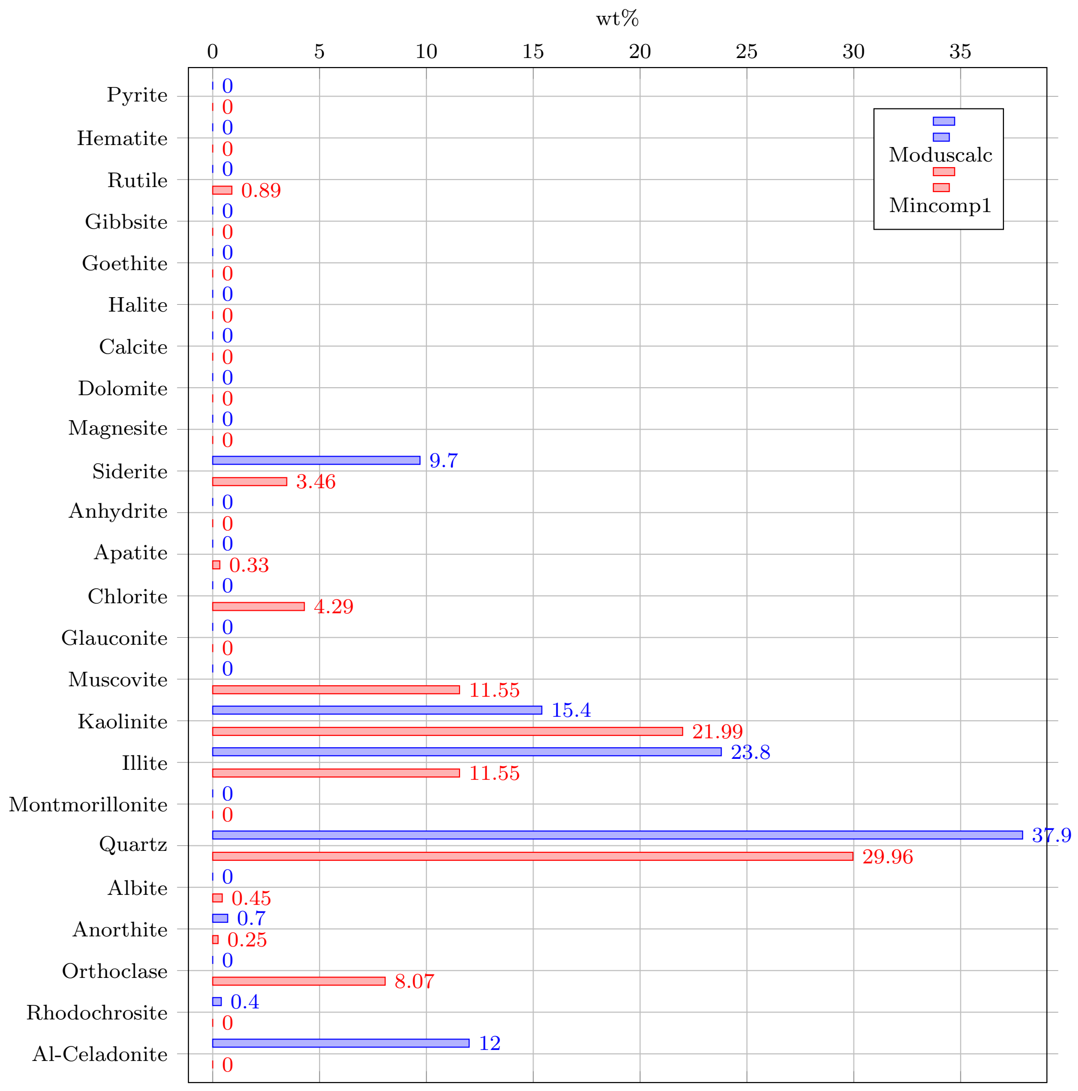
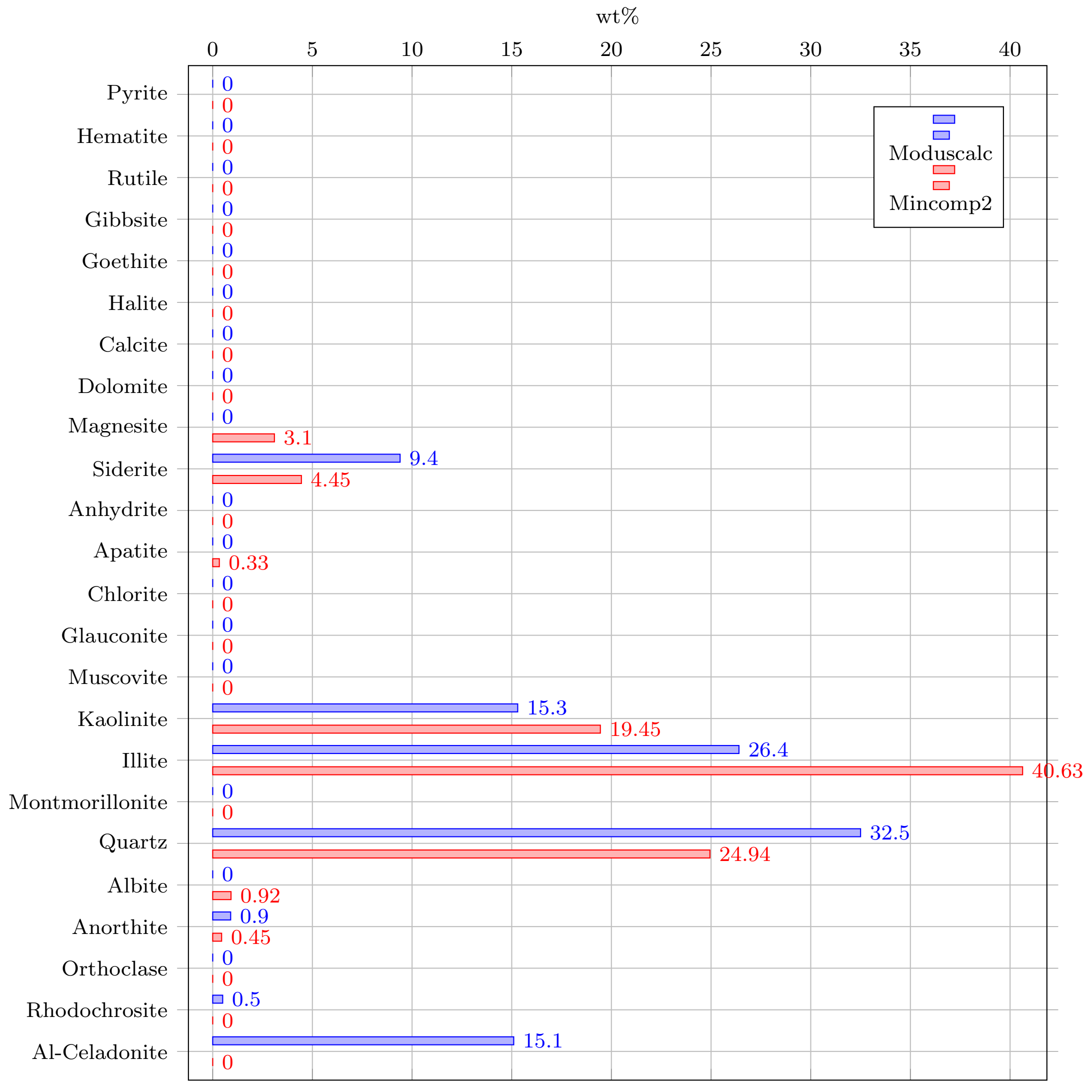


Figure 5.14: Results for LM50 dataset (Laube et al., 1996).

We see differences in Siderite, this is due to the fact that a percentage of the available iron is allocated to Chlorite, which isn't present in Moduscalc.
The difference in Illite is due to the fact that when several potassium-bearing minerals are present in the calculation list of Mincomp, the total amount of potassium is evenly distributed to the different minerals.
The amount of Kaolinite is slightly higher, due to the allocation order of Mincomp.
There is a small difference in the amount of Quartz, this is because more SiO_2 is used to calculate Kaolinite, Illite, Muscovite and Orthoclase.

5.4.2 Comparison with exactly the same minerals

LM41 sample



Mineral	Moduscalc	Mincomp2
Pyrite	0	0
Hematite	0	0
Rutile	0	0
Gibbsite	0	0
Goethite	0	0
Halite	0	0
Calcite	0	0
Dolomite	0	0
Magnesite	0	3.1
Siderite	9.4	4.45
Anhydrite	0	0
Apatite	0	0.33
Chlorite	0	0
Glauconite	0	0
Muscovite	0	0
Kaolinite	15.3	19.45
Illite	26.4	40.63
Montmorillonite	0	0
Quartz	32.5	24.94
Albite	0	0.92
Anorthite	0.9	0.45
Orthoclase	0	0
Rhodochrosite	0.5	0
Al-Celadonite	15.1	0

Figure 5.15: Results for LM41 dataset (Laube et al., 1996).

The amount of Kaolinite is a bit greater, due to allocation order.
The amount of Illite is a lot greater than from Moduscalc, since all available K_2O is allocated to Illite it makes up a huge amount of the sample. In Moduscalc a great amount of K_2O is allocated to Al-Celadonite, which isn't present in Mincomp.
The amount of Quartz is lower, due to the high amount of Illite.

LM50 sample

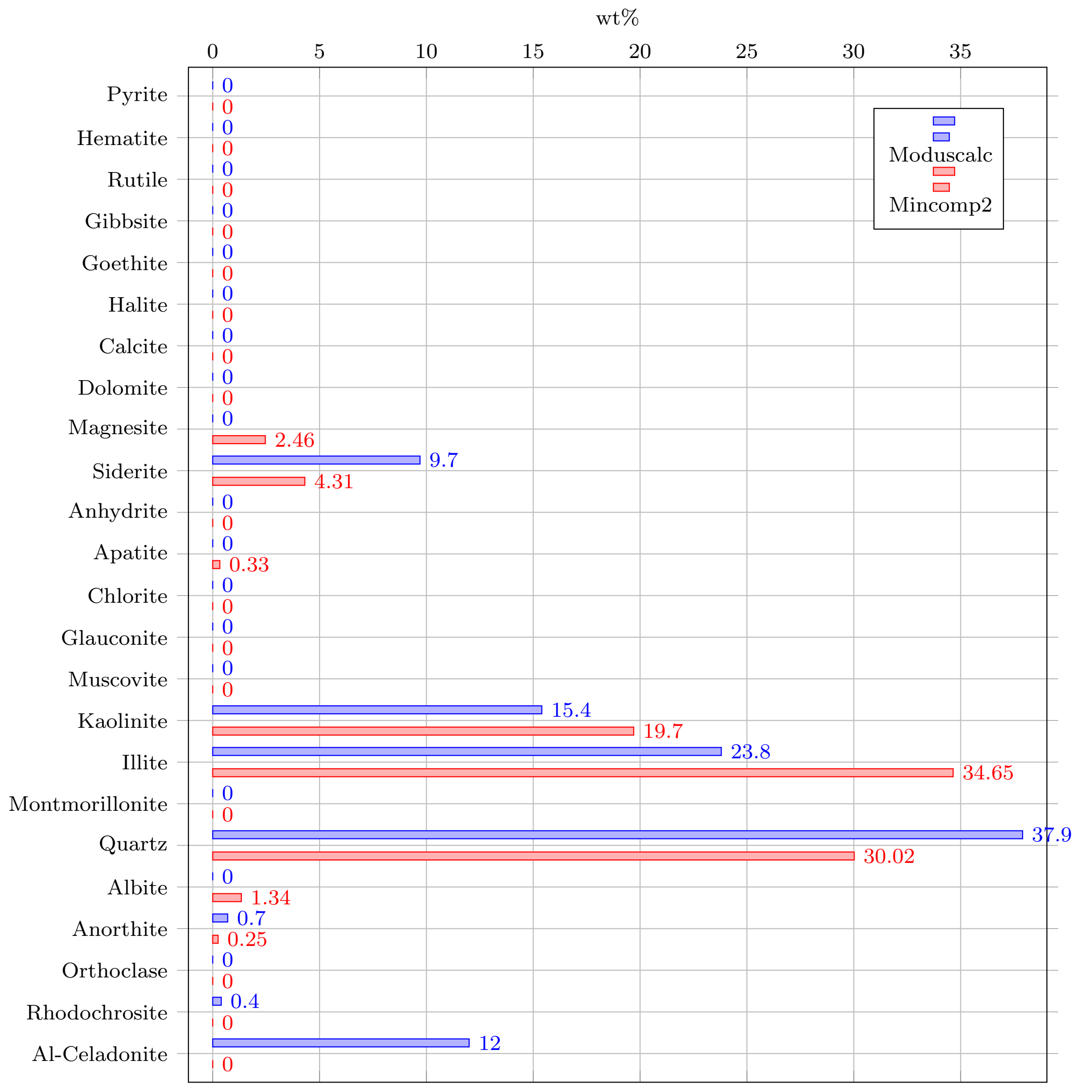


Figure 5.16: Results for LM50 dataset (Laube et al., 1996).

The second run shows high amounts of Kaolinite and Illite, the relatively high amount of Kaolinite is because of teh allocation order, the high amount of Illite is because all potassium is allocated to Illite in the second run. Moduscalc distributes potassium into Illite and Al-Celadonite, hence the difference. The lower amount of Quartz is due to the high amounts of Kaolinite and Illite.

5.4.3 Combined graphs and tables

Mineral	Moduscalc wt%	Mincomp-1 wt%	Mincomp-2 wt%
Pyrite	-	0.0 %	0.0 %
Hematite	-	0.0 %	0.0 %
Rutile	-	0.9264 %	0.0 %
Gibbsite	-	0.0 %	0.0 %
Goethite	-	0.0 %	0.0 %
Halite	-	0.0 %	0.0 %
Calcite	-	0.0 %	0.0 %
Dolomite	-	0.0 %	0.0 %
Magnesite	0 %	0.0 %	3.095 %
Siderite	9.4%	3.383 %	4.449 %
Anhydrite	-	0.0 %	0.0 %
Apatite	-	0.332 %	0.332 %
Chlorite	-	5.39 %	0.0 %
Glauconite	-	0.0 %	0.0 %
Muscovite	-	13.54 %	0.0 %
Kaolinite	15.3 %	21.769 %	19.4468 %
Illite	26.4 %	13.543 %	40.629 %
Montmorillonite	-	0.0 %	0.0 %
Quartz	32.5 %	24.666 %	24.943 %
Albite	0 %	0.3041 %	0.9178 %
Anorthite	0.9 %	0.446 %	0.446 %
Orthoclase	-	9.463 %	0.0 %
Rhodochrosite	0.5%	-	-
Al-Celadonite	15.1 %	-	-

Table 5.9: Test results for Moduscalc LM41 dataset, Mincomp-1 refers to run with all minerals included, Mincomp-2 refers to run with individually selected minerals.

Mineral	Moduscalc wt%	Mincomp-1 wt%	Mincomp-2 wt%
Pyrite	-	0.0 %	0.0 %
Hematite	-	0.0 %	0.0 %
Rutile	-	0.8945 %	0.0 %
Gibbsite	-	0.0 %	0.0 %
Goethite	-	0.0 %	0.0 %
Halite	-	0.0 %	0.0 %
Calcite	-	0.0 %	0.0 %
Dolomite	-	0.0 %	0.0 %
Magnesite	0.0 %	0.0 %	2.462 %
Siderite	9.7 %	3.464 %	4.31 %
Anhydrite	-	0.0 %	0.0 %
Apatite	-	0.332 %	0.332 %
Chlorite	-	4.288 %	0.0 %
Glauconite	-	0.0 %	0.0 %
Muscovite	-	11.55 %	0.0 %
Kaolinite	15.4 %	21.99 %	19.70 %
Illite	23.8 %	11.55 %	34.65 %
Montmorillonite	-	0.0 %	0.0 %
Quartz	37.9 %	29.96 %	30.02 %
Albite	0.0 %	0.446 %	1.34 %
Anorthite	0.7 %	0.25 %	0.251 %
Orthoclase	-	8.07 %	0.0 %
Rhodochrosite	0.4 %	-	-
Al-Celadonite	12.0 %	-	-

Table 5.10: LM50 sample from Moduscalc, Mincomp-1 refers to the first run with all minerals included, Mincomp-2 refers to the second run with individually selected minerals.

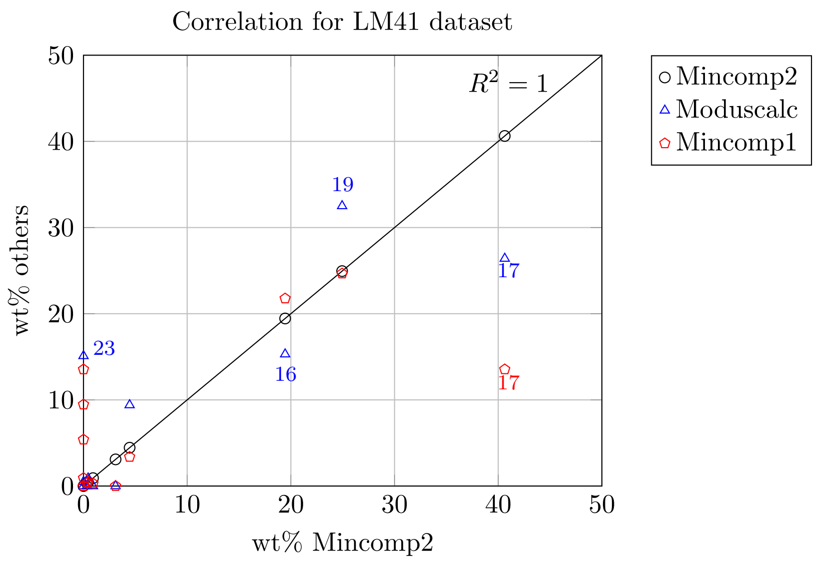


Figure 5.17: Results for LM41 dataset. **16** - Kaolinite, **17** - Illite, **19** - Quartz, **23** - Orthoclase.

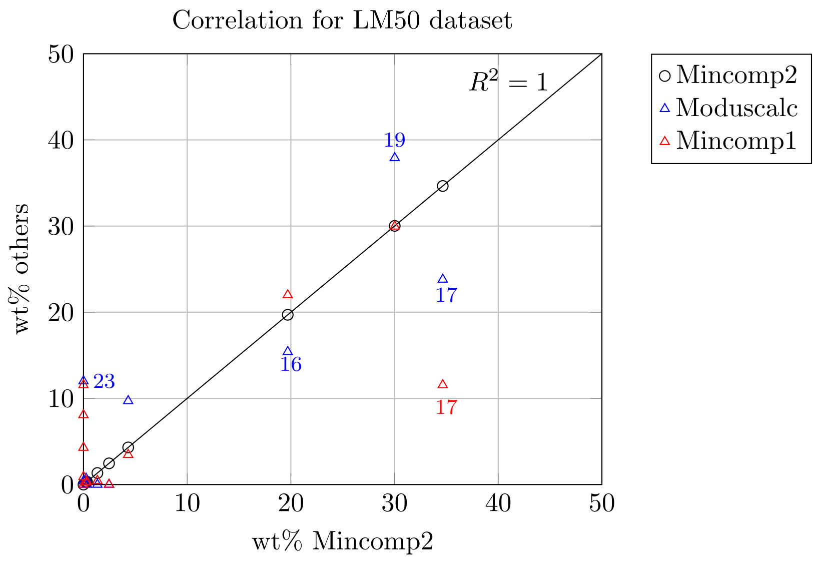


Figure 5.18: Results for LM50 dataset. **16** - Kaolinite, **17** - Illite, **19** - Quartz, **23** - Orthoclase.

Chapter 6

Discussion

The computer program Mincomp was developed to calculate a likely mineralogical bulk composition of sedimentary rocks in a quick and easy way. The algorithm relies on X-ray diffraction and X-ray fluorescence data of the sample and follows a set of rules to calculate the synthetic mineral content. Mincomp incorporates the most common sedimentary minerals and is therefore useful for calculating the synthetic mineral content of all sorts of sedimentary rocks. It makes use of a rigid allocation sequence, which delivers constant results. The program can be used in two different modes, batch mode and individual mode. While making use of the batch mode, the program evaluates every mineral incorporated in the program and tries to allocate an amount to this mineral. Batch mode is useful if almost no information is available for a specific sample. While making use of the individual mode, the user can specify which minerals are likely to be present in the sample, for more accurate results. It can't be stressed enough, that the accuracy of Mincomps calculation improves when more information is available about the sample.
Test results of comparison between different programs available show acceptable results for many different sedimentary rocks, the results are in general alike. Minor differences between quantities allocated to specific minerals occur, but are explainable and are mostly due to the algorithm sequence, differences in chemical formula, and the absence of specific minerals in other program.
Mincomp is written in Python 2.7 and relies on NumPy 1.8.1, Mincomp is executable on most platforms, as an independent executable on Windows and in a terminal on Linux; deployment on a Virtual Machine is also possible.
Mincomp is a quick and simple method to obtain quantitative mineralogical information about rock samples when XRD and XRF results are available, and provides a first insight in likely mineralogical bulk compositions.

Chapter 7

Recommendations

7.1 Programming-related recommendations

Mincomp is written in Python 2.7 and relies on NumPy 1.8.1. Python is a very versatile programming language and the Numerical Python module is excellent for numerical data analysis. However, the programming style of Mincomp can be further improved by making use of Python Pandas. Python Pandas is an extra module for data analysis and provides flexible and fast dataframes. While making use of Python Pandas the necessity to transform between different data types becomes obsolete. Python Pandas provides much better dataframes compared to the NumPy arrays used in this version of Mincomp.

While it could be improved in terms of efficiency, one has to say that with the current hardware the computational times are already small. Improvement of data management would not necessary benefit the user in terms of notable reduced computing time. Already the computing time is in the order of miliseconds, but the program will be better structured and won't have to perform irrelevant data transformations anymore.

7.2 Algorithm-related recommendations

At this point, Mincomp usually calculates a higher amount of Kaolinite than most other programs, and calculates a smaller amount of Quartz, this is due to the allocation sequence. While Kaolinite is calculated in the second stage, a lot of Sodium is allocated to Kaolinite. While allocating these clays a lot of SiO_2 is used, this is subtracted from the total, so in the end when Quartz is allocated, less SiO_2 is available to calculate Quartz.

It is recommended to test how a different order of allocating minerals would affect the final result of Mincomp. If a better allocation order can be created it is recommended to implement it in a newer version of Mincomp.

One can choose to incorporate more minerals in the program, or more element oxides from the XRF results. The program can benefit from it in terms of a more precise allocation of 'trace' minerals, but it wouldn't affect the bulk composition of the sample.

Acknowledgements

I hereby would like to thank my supervisor Dr. Karl-Heinz Wolf, who supported me throughout my minor project. His support and guidance were of great value to me in order to finish project. Without his encouragement and patience this project would not have been finished.

I also would like to thank Drs. Maaike van Tooren, who provided answers to my numerous mineral-related questions, especially to questions which weren't answers in the books.

Bibliography

John W. Anthony, Richard A. Bideaux, Kenneth W. Bladh, and Monte C. Nichols. *Handbook of Mineralogy.* Mineral Data Publishing, Tucson, Arizona, 1995.

Barthalmy. Mineralogy database, November 2013. URL `http://www.webmineral.com`.

Patrice De Caritat, John Bloch, and Ian Hutcheon. Lpnorm: A linear programming normative analysis code. *Computers and Geosciences*, 20(3):313–347, 1994.

David Cohen and Colin R. Ward. Sednorm - a program to calculate a normative mineralogy for sedimentary rocks based on chemical analyses. *Computers and Geosciences*, 17(9):1235–1253, 1991.

W.A. Deer, R.A. Howie, and J. Zussman. *An introduction to the rock-forming minerals.* Longmans, Green and Co. Ltd, 1966.

R.M. Garrels and F.T. Mackenzie. *Evolution of sedimentary rocks.* W.W. Norton, New York, 1971.

N. Laube, S. Hergarten, and H.J. Neugebauer. Moduscalc - a computer program to calculate a mode from a geochemical rock analysis. *Computers and Geosciences*, 1996.

W.G. Mumme, G. Tsambourakis, I.C. Madsen, and R.J. Hill. Improved petrological modal analysis from x-ray powder diffraction data by use of the rietveld method. *Journal of Sedimentary Research*, 1996.

G.D. Nicholls. A scheme for recalculating the chemical analysis of argillaceous rocks for comparative purposes. *American Mineralogist*, 47(1-2):34–46, 1962.

A. Dogan Paktunc. Modan: an interactive computer program for estimating mineral quantities based on bulk composition. *Computers and Geosciences*, 24(5):425–431, 1998.

Maximilian Posch and Daniel Kurz. A2m - a program to compute all possible mineral modes from geochemical analyses. *Computers and Geosciences*, 33:563–572, 2007.

A. B. Ronov and A. A. Yaroshevsky. Chemical composition of the earth's crust. *Geophysical Monograph Series*, 13:37–57, 1967.

Oleg M. Rosen, Ali A. Abbyasov, and John C. Tipper. Minlith - an experience-based algorithm for estimating the likely mineralogical compositions of sedimentary rocks from bulk chemical analyses. *Computers and Geosciences*, 30:647–661, 2004.

R. Salminen, M.J. Batista, M. Bidovec, A. Demetriades, B. De Vivo, W. De Vos, M. Duris, A. Gilucis, V. Greogorauskiene, J. Halamic, P. Heitzmann, A. Lima, G. Jordan andG. Klaver, P. Klein, J. Lis, J. Locutura, K. Marsina, A. Mazreku, P.J. O'Connor, R.T. Ottesen S.Å. Olsson, V. Petersell, J.A. Plant, S. Reeder, I. Salpeteur, H. Sandström, U. Siewers, A. Steenfelt, and T. Tarvainen. *Geochemical Atlas of Europe. Part 1: Background Information, Methodology and Maps.* Geological Survey of Finland, Espoo, 2005.

Nivaldo J. Tro. *Principles of Chemistry, a molecular approach.* Pearson Education International, 2010.

C.R. Ward, D.R. Cohen, A. Crouch, D. Panich, S. Schaller, and P.K. Dutta. Assessment of gas ignitability risk by frictional effects from coal mine rocks; end-of-grand report. Technical report, National Energy Research Development and Demonstration Program, Commonwealth Department of Primary Industries and Energy, Canberra, 1990.

K.H.A.A. Wolf. *The interaction between underground coal fires and their roof rocks.* PhD thesis, TU Delft, 2006.

Appendix A - Mincomp User Manual

Introduction

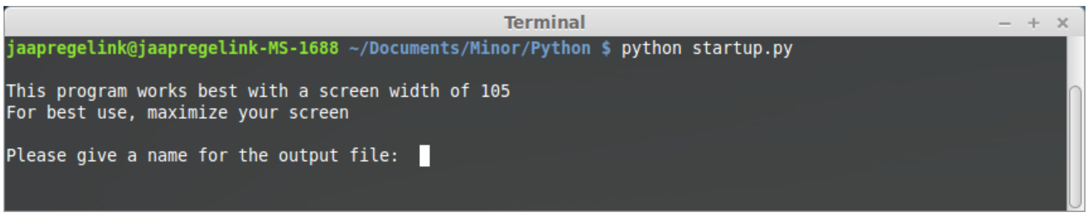
This manual describes how to use the program *Mincomp* to calculate the synthetic mineral composition based on X-ray diffraction (*XRD*) and X-ray fluorescence (*XRF*) test results. The output is a '.dat' file in which all the information is stored, and a graph giving an overview of the sample content.
The program is executed in the command-line for ease of use, and simplicity.

Installing Mincomp - *Windows*

The program doesn't need to be installed on a computer, but can be executed from a commandline, or by doubleclicking *'startup.exe'*. *'startup.exe'* is located in the *'dist'* folder. The program runs on a Virtual Machine on Linux as well (Apple Mac is not tested).
The program is written in Python 2.7, while using NumPy 1.8.1, however the user doesn't need to have anything installed on his computer, all dependencies are included in the package.
Note that *'startup.exe'* needs to be in the same folder as all the other files, for ease of use a shortcut to *'startup.exe'* can be created.

Installing Mincomp - *Linux*

Running Mincomp on Linux is even easier, the file can be loaded in the terminal right away. The file is located in the *'dist'* folder, and it needs to be there in order for file dependencies. By entering the command *"python startup.py"* the program will be loaded into the terminal. See figure 7.1. Python 2.7 needs to be installed in order to run the program.

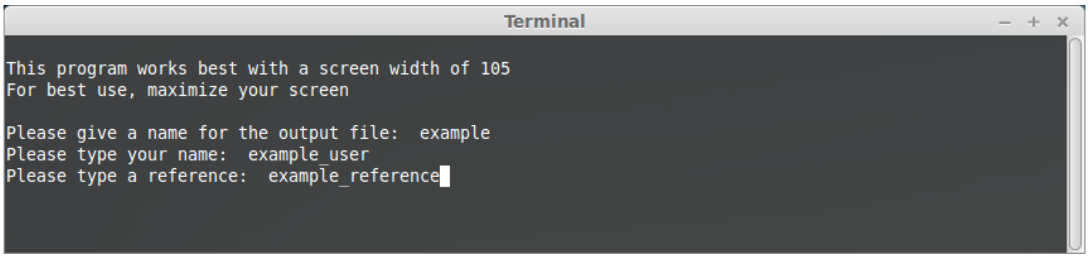


```
jaapregelink@jaapregelink-MS-1688 ~/Documents/Minor/Python $ python startup.py

This program works best with a screen width of 105
For best use, maximize your screen

Please give a name for the output file:
```

Figure 7.1: Starting Mincomp in a terminal window on Linux.

Starting Mincomp

When starting the application, a command windows shows up. It is important that the width of the command windows is at least 105, otherwise problems will arise with text formatting. The program would still work, but it would like less nice. Therefore, a width of at least 105 is recommended. The first part of the program is to fill in your information. How you want the output file to be named, your own name, and a project reference. You can see how this screen looks in figure 7.2.

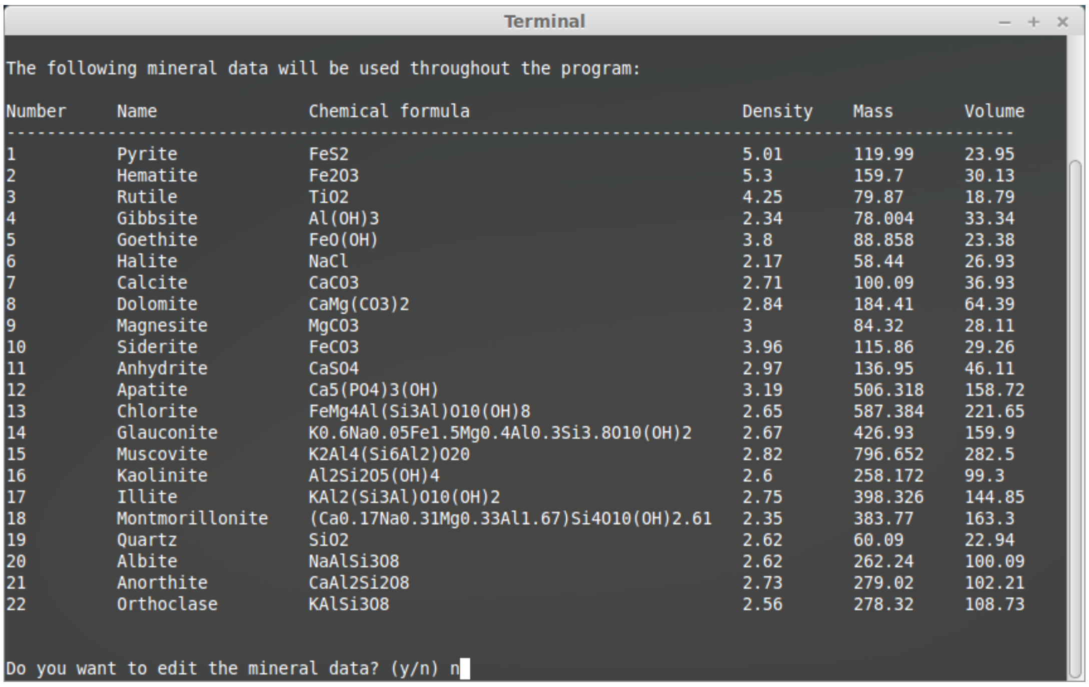


```
This program works best with a screen width of 105
For best use, maximize your screen

Please give a name for the output file:  example
Please type your name:  example_user
Please type a reference:  example_reference
```

Figure 7.2: The startup screen from Mincomp, with filled-in information.

The mineral table and reviewing information

The next step in the process is to review the mineral data that is used in the program. The values are all from *An introduction to the Rock Forming Minerals* by Deer, Howie and Zussman. In the table the used mineral weights and densitys are displayed, you can change the mineral weights and densities if you would like to, but it is necessary. See figure 7.3.



```
The following mineral data will be used throughout the program:

Number     Name              Chemical formula                              Density    Mass       Volume
---------------------------------------------------------------------------------------------------------
1          Pyrite            FeS2                                          5.01       119.99     23.95
2          Hematite          Fe2O3                                         5.3        159.7      30.13
3          Rutile            TiO2                                          4.25       79.87      18.79
4          Gibbsite          Al(OH)3                                       2.34       78.004     33.34
5          Goethite          FeO(OH)                                       3.8        88.858     23.38
6          Halite            NaCl                                          2.17       58.44      26.93
7          Calcite           CaCO3                                         2.71       100.09     36.93
8          Dolomite          CaMg(CO3)2                                    2.84       184.41     64.39
9          Magnesite         MgCO3                                         3          84.32      28.11
10         Siderite          FeCO3                                         3.96       115.86     29.26
11         Anhydrite         CaSO4                                         2.97       136.95     46.11
12         Apatite           Ca5(PO4)3(OH)                                 3.19       506.318    158.72
13         Chlorite          FeMg4Al(Si3Al)O10(OH)8                        2.65       587.384    221.65
14         Glauconite        K0.6Na0.05Fe1.5Mg0.4Al0.3Si3.8O10(OH)2        2.67       426.93     159.9
15         Muscovite         K2Al4(Si6Al2)O20                              2.82       796.652    282.5
16         Kaolinite         Al2Si2O5(OH)4                                 2.6        258.172    99.3
17         Illite            KAl2(Si3Al)O10(OH)2                           2.75       398.326    144.85
18         Montmorillonite   (Ca0.17Na0.31Mg0.33Al1.67)Si4O10(OH)2.61      2.35       383.77     163.3
19         Quartz            SiO2                                          2.62       60.09      22.94
20         Albite            NaAlSi3O8                                     2.62       262.24     100.09
21         Anorthite         CaAl2Si2O8                                    2.73       279.02     102.21
22         Orthoclase        KAlSi3O8                                      2.56       278.32     108.73


Do you want to edit the mineral data? (y/n) n
```

Figure 7.3: The mineral table, with the used data.

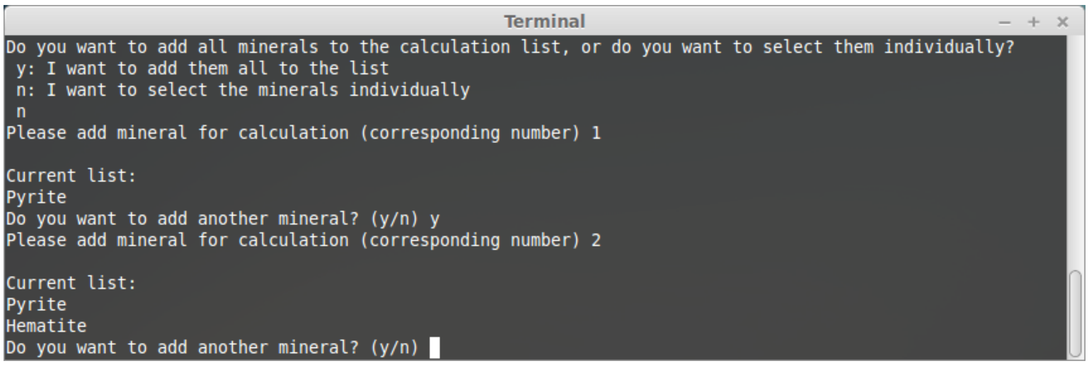
Adding minerals to the calculation list

The result from the X-ray diffraction test will be a list of minerals present in the sample.
These minerals have to be added to Mincomp's *'calculation list'*, this can be done in two ways:

- Batch - add all minerals available in Mincomp to the list
- Individual - select and add minerals individually to the list

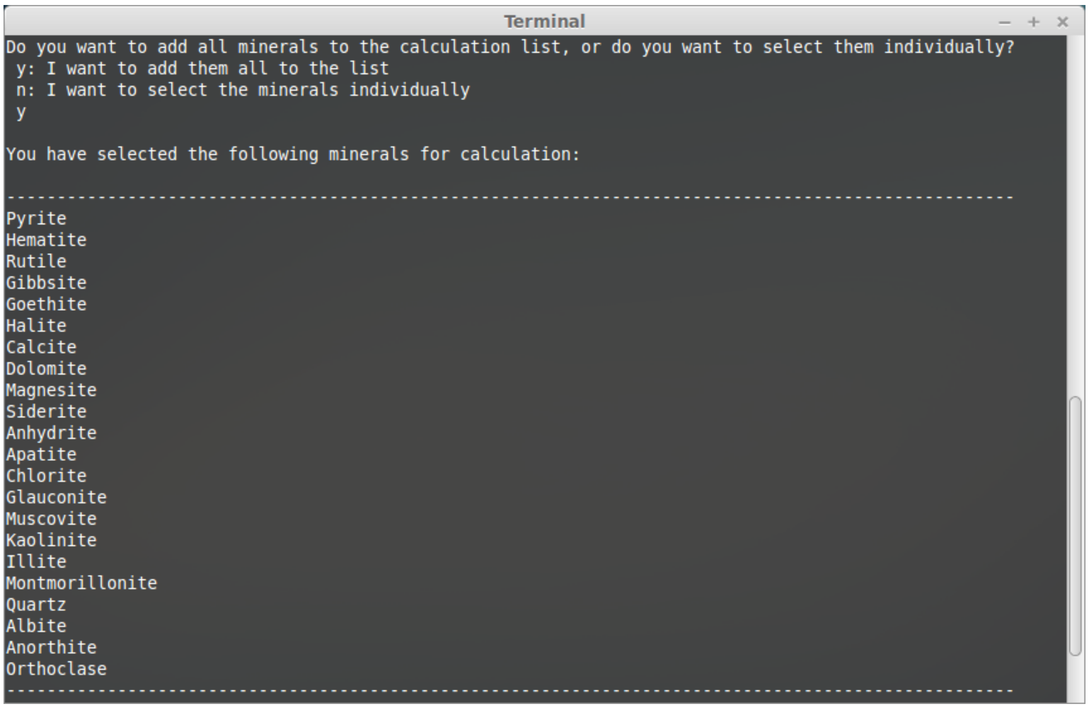
While using the *Batch* option, all minerals specified in Mincomp will be added to the list, and the program will try to calculate it's amount present in the rock sample. The minerals don't have to be necessarily present in the sample. See figure 7.4.
While using the *Individual* option, you have to select minerals specified in Mincomp individually and add them to the list. The program will calculate it's amount present in the rock sample. While using the individual option, only the minerals you expect to be present in the rock sample will be processed in the

program sequence, and an amount will be calculated. It gives a more precise end result in comparison to the *Batch* option. See figure 7.5.



```
Do you want to add all minerals to the calculation list, or do you want to select them individually?
 y: I want to add them all to the list
 n: I want to select the minerals individually
 n
Please add mineral for calculation (corresponding number) 1

Current list:
Pyrite
Do you want to add another mineral? (y/n) y
Please add mineral for calculation (corresponding number) 2

Current list:
Pyrite
Hematite
Do you want to add another mineral? (y/n) 
```

Figure 7.4: Individually selecting minerals, and adding them to the calculation list.



```
Do you want to add all minerals to the calculation list, or do you want to select them individually?
 y: I want to add them all to the list
 n: I want to select the minerals individually
 y

You have selected the following minerals for calculation:

----------------------------------------------------------------------------------------------------
Pyrite
Hematite
Rutile
Gibbsite
Goethite
Halite
Calcite
Dolomite
Magnesite
Siderite
Anhydrite
Apatite
Chlorite
Glauconite
Muscovite
Kaolinite
Illite
Montmorillonite
Quartz
Albite
Anorthite
Orthoclase
----------------------------------------------------------------------------------------------------
```

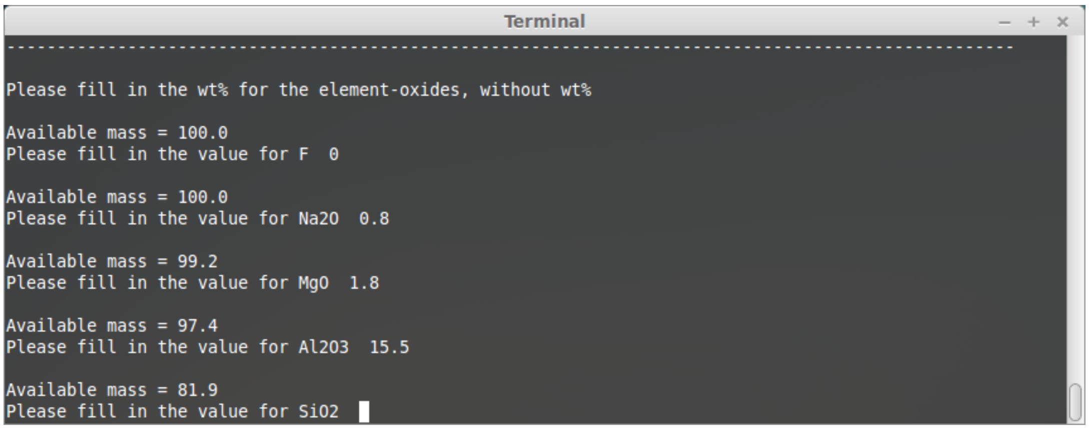
Figure 7.5: Adding all the minerals to the calculation list with Batch mode.

Inserting X-ray Fluorescence data

The X-ray Fluorescence (XRF) data must be inserted manually, by typing in the weight percentage of a specific element oxide. The process is straight-forward, and the program checks if the values do not exceed 100 %. If the sample amount is known, it can be inserted to calculate the molar amounts as well. If the sample amount is kept empty, a default value of 1000.0 mg is used. The sample amount is not really needed for calculation, but it is needed for the program sequence. When all the information is inserted, the program will calculate the molar amounts of element oxides and the molar amounts of specific elements. It will summarize the data and display it as a table, see figure 7.7.

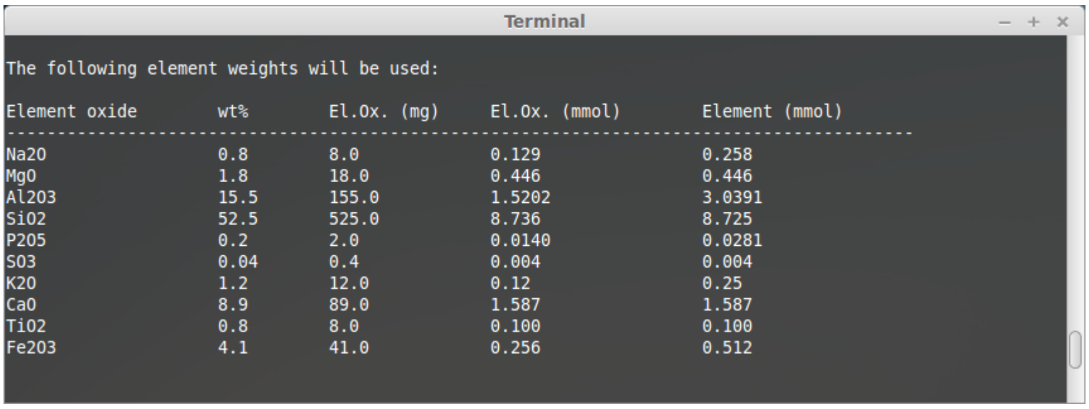
Allocation stages

The greatest part of the allocation stages is straight-forward and is executed without input from the user. In the first allocation stage the amounts of trace minerals are allocated, during the second stage



```
Terminal
-----------------------------------------------------------------------------------------------

Please fill in the wt% for the element-oxides, without wt%

Available mass = 100.0
Please fill in the value for F  0

Available mass = 100.0
Please fill in the value for Na2O  0.8

Available mass = 99.2
Please fill in the value for MgO  1.8

Available mass = 97.4
Please fill in the value for Al2O3  15.5

Available mass = 81.9
Please fill in the value for SiO2 
```

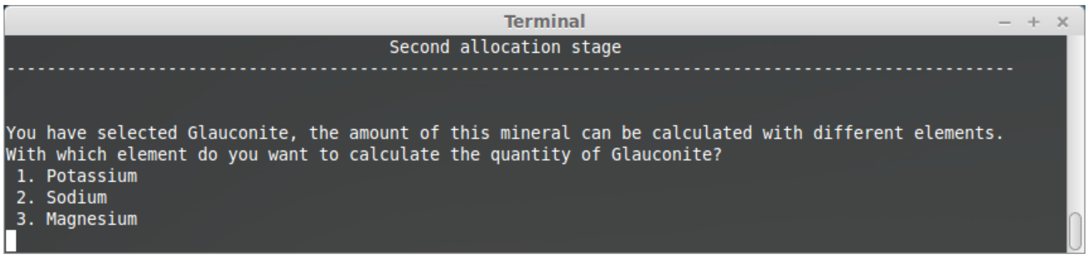
Figure 7.6: Inserting XRF data into Mincomp.



```
Terminal

The following element weights will be used:

Element oxide       wt%        El.Ox. (mg)     El.Ox. (mmol)       Element (mmol)
-------------------------------------------------------------------------------------
Na2O                0.8        8.0             0.129               0.258
MgO                 1.8        18.0            0.446               0.446
Al2O3               15.5       155.0           1.5202              3.0391
SiO2                52.5       525.0           8.736               8.725
P2O5                0.2        2.0             0.0140              0.0281
SO3                 0.04       0.4             0.004               0.004
K2O                 1.2        12.0            0.12                0.25
CaO                 8.9        89.0            1.587               1.587
TiO2                0.8        8.0             0.100               0.100
Fe2O3               4.1        41.0            0.256               0.512
```

Figure 7.7: Summarized data.

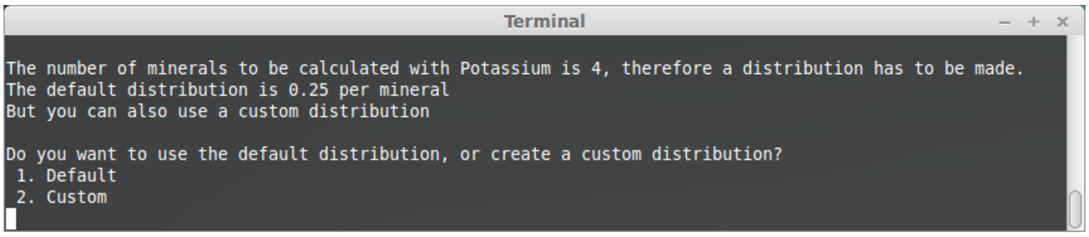
the aluminium-silicates and clay minerals are allocated. In the third and last allocation stage quartz, carbonates and the remainder of minerals. During the second and third allocation stage user input is sometimes necessary. Some minerals can be calculated by using multiple main elements, for example, if you want Glauconite to be calculated according to the available amount of Magnesium, you can set that option, but you can also choose other elements.



```
Terminal
                                      Second allocation stage
-----------------------------------------------------------------------------------------------


You have selected Glauconite, the amount of this mineral can be calculated with different elements.
With which element do you want to calculate the quantity of Glauconite?
 1. Potassium
 2. Sodium
 3. Magnesium
```

Figure 7.8: Choose a main element to calculate Glauconite

Some minerals are much alike, or have almost the same chemical formula, in that case an arbitrary distribution of a specific element has to be made. How much of the available mass is allocated to a specific mineral. The user has two choices, the default option is an equal distribution between the different elements. The second option is to make a custom distribution, this way you can distribute for example 75% of K_2O to Illite, and 25% to Orthoclase. See figure 7.9



```
The number of minerals to be calculated with Potassium is 4, therefore a distribution has to be made.
The default distribution is 0.25 per mineral
But you can also use a custom distribution

Do you want to use the default distribution, or create a custom distribution?
 1. Default
 2. Custom
```

Figure 7.9: Mincomp asks how to distribute a specific element between different minerals.

Final result

After all the allocation stages, Mincomp displays a final result including the weight percentages and volume percentages of the different minerals that were present in the calculation list. The final result, but also the intermediate results, are printed to an output file. In this file everything that has been done with the program is stored. Also, the program draws a graph for quick visualization of the amounts of different minerals.

Appendix B - Python source files

```python
#Start-up prompt

#import numpy as np
from numpy import*
from math import*
from fractions import*
from operator import itemgetter
import matplotlib
matplotlib.use("Agg")
import pylab
import time
import mineral_data
import os
import matplotlib.pyplot as plt

#Declare variable localtime, which is printed in the output file.
localtime = time.asctime( time.localtime(time.time()) )

print "\nThis program works best with a screen width of 105"
print "For best use, maximize your screen \n"

#We start by creating a log file
filename = raw_input("Please give a name for the output file:  ")

#Create output file and print name, reference and date.
file_out = open(filename + ".dat", "w")
file_out.write("-"*100 + "\n")
file_out.write("This log file is automatically created by running the program \n")
file_out.write("File created on  " + str(localtime) +  "\n")

user = raw_input("Please type your name:  ")
user_reference = raw_input("Please type a reference:  ")

file_out.write("This file is created by:  " + str(user) + "\n")
file_out.write("Project reference: " + str(user_reference) + "\n")
file_out.write("-"*100 + "\n \n")


#Print the list of minerals in a formatted way.
print "\nThe following mineral data will be used throughout the program:\n"
print "{0:10} {1:18} {2:42} {3:10} {4:10} {5:10}".format(mineral_data.example[6],
    mineral_data.example[0], mineral_data.example[1], mineral_data.example[2],
    mineral_data.example[3], mineral_data.example[4])
print "-"*100
for i in mineral_data.all_minerals:
  print "{0:10} {1:18} {2:42} {3:10} {4:10} {5:10}".format(i[6],i[0],i[1],i[2],i[3],i
    [4])
print "\n"


#You don't have to edit the mineral data, if you don't want to.
edit_mode = raw_input("Do you want to edit the mineral data? (y/n) ")

#But if you want to, it calls the edit_mineral module.
if edit_mode =="y" or edit_mode=="Y" or edit_mode=="yes":
  import edit_mineral

#declare calculate_minerals as a list.
calculate_minerals = ([])
```

```python

#Needed in order to calculate mineral amounts.
add_minerals_to_list = raw_input("\nDo you want to add minerals to the calculation list? (y/n) ")

add_all = raw_input("\nDo you want to add all minerals to the calculation list, or do you want to select them individually?\n y: I want to add them all to the list\n n: I want to select the minerals individually\n ")
if add_all=="yes" or add_all=="y" or add_all=="Y":
  calculate_minerals =([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21])
  add_minerals_to_list = "n"

#Add all the minerals you want.
while add_minerals_to_list=="yes" or add_minerals_to_list=="y" or add_minerals_to_list=="Y":
  input_minerals = int(raw_input("Please add mineral for calculation (corresponding number) "))
  while (input_minerals-1) in calculate_minerals:
    input_minerals = int(raw_input("The selected mineral is already present in your list, please add mineral for calculation (corresponding number) "))
  while (input_minerals-1) > 21:
    input_minerals = int(raw_input("The selected number is not incorrect, please add mineral for calculation (corresponding number) "))
  calculate_minerals.append(input_minerals-1)
  print "\nCurrent list:"
  for i in calculate_minerals:
    print mineral_data.all_minerals[i][0]
  add_minerals_to_list = raw_input("Do you want to add another mineral? (y/n) ")


calculation_list = ([])

for i in calculate_minerals:
  calculation_list.append(mineral_data.all_minerals[i][0])


#If user is finished, show the complete list of minerals.
print "\nYou have selected the following minerals for calculation: \n"
print "-"*100
for i in calculate_minerals:
  print mineral_data.all_minerals[i][0]
print "-"*100

#Print the selected minerals to output file.
file_out.write("\n\n")
file_out.write("\nYou have selected the following minerals for calculation: \n")
file_out.write("\n" + "{0:10} {1:18} {2:42} {3:10} {4:10} {5:10}".format("Number", "Name", "Formula", "Density", "Mass", "Volume"))
file_out.write("-"*100)
for i in calculate_minerals:
  file_out.write("\n" + "{0:10} {1:18} {2:42} {3:10} {4:10} {5:10}".format(mineral_data.all_minerals[i][6], mineral_data.all_minerals[i][0], mineral_data.all_minerals[i][1],mineral_data.all_minerals[i][2],mineral_data.all_minerals[i][3],mineral_data.all_minerals[i][4]))


#next step, XRF input
import xrf_input


#write xrf_input data to output file
file_out.write("\n\n\n\n")
file_out.write("\nThe following data will be used for calculation:\n")
file_out.write("{0:20} {1:10} {2:15} {3:20} {4:20}".format("Element oxide", "wt%", "Weight (mg)", "Amount (moles)", "Element amount (moles)"))
file_out.write("\n" + "-"*100)
for i in xrf_input.elox_list:
  if float(i[4]) != 0:
    file_out.write("\n" + "{0:20} {1:10} {2:15} {3:20} {4:20}".format(i[0],i[4],i[5],i[6],i[9]))
file_out.write("\n\n\n\n")
file_out.write("The sample weight is " + str(xrf_input.total_weight_sample) + " mg\n")
```

```
#file_out.write("The sum of weight percentages before normalization is " + str(xrf_input
    .total_before_normalization) + " %\n")

print "\n\n\nThe following element weights will be used: \n"
print "{0:20} {1:10} {2:15} {3:20} {4:20}".format("Element oxide", "wt%", "El.Ox. (mg)",
     "El.Ox. (mmol)", "Element (mmol)")
print "-"*90
for i in xrf_input.elox_list:
  if float(i[4]) != 0:
    print "{0:20} {1:10} {2:15} {3:20} {4:20}".format(i[0],i[4],i[5],i[6],i[9])



#--------------------------------------------------------------------------#
#                                                                          #
#                        First Allocation Stage                            #
#                                                                          #
#--------------------------------------------------------------------------#

#mineral_moles = array(["Mineral name", mole amount, weight, wt%, volume, volume
    percentage]
#                              0                1          2      3      4            5
anhydrite_moles = array(["Anhydrite", 0, 0, 0, 0, 0])
pyrite_moles = array(["Pyrite", 0, 0, 0, 0, 0])
gibbsite_moles = array(["Gibbsite", 0, 0, 0, 0, 0])
goethite_moles = array(["Goethite", 0, 0, 0, 0, 0])
hematite_moles = array(["Hematite", 0, 0, 0, 0, 0])
quartz_moles = array(["Quartz", 0, 0, 0, 0, 0])
rutile_moles = array(["Rutile", 0, 0, 0, 0, 0])
halite_moles = array(["Halite", 0, 0, 0, 0, 0])
calcite_moles = array(["Calcite", 0, 0, 0, 0, 0])
dolomite_moles = array(["Dolomite", 0, 0, 0, 0, 0])
magnesite_moles = array(["Magnesite", 0, 0, 0, 0, 0])
siderite_moles = array(["Siderite", 0, 0, 0, 0, 0])
apatite_moles = array(["Apatite",0, 0, 0, 0, 0])
albite_moles = array(["Albite", 0, 0, 0, 0, 0])
anorthite_moles = array(["Anorthite", 0, 0, 0, 0, 0])
chlorite_moles = array(["Chlorite", 0, 0, 0, 0, 0])
glauconite_moles = array(["Glauconite", 0, 0, 0, 0, 0])
muscovite_moles = array(["Muscovite", 0, 0, 0, 0, 0])
orthoclase_moles = array(["Orthoclase", 0, 0, 0, 0, 0])
kaolinite_moles = array(["Kaolinite", 0, 0, 0, 0, 0])
illite_moles = array(["Illite", 0, 0, 0, 0, 0])
montmorillonite_moles = array(["Montmorillonite", 0, 0, 0, 0, 0])



# Trace mineral allocation

oxygen = float(16.00)
hydrogen = float(1.008)
carbon = float(12.01)


#Anhydrite [CaSO4]
if "Anhydrite" in calculation_list:
  if float(xrf_input.elox_so3[9]) != int(0) and float(xrf_input.elox_cao[9]) >= float(
    xrf_input.elox_so3[9]):
    anhydrite_moles[1] = float(xrf_input.elox_so3[9])
    xrf_input.elox_cao[9] = float(xrf_input.elox_cao[9]) - float(anhydrite_moles[1])
    xrf_input.elox_so3[9] = float(xrf_input.elox_so3[9]) - float(anhydrite_moles[1])
    xrf_input.weight_loss = xrf_input.weight_loss - (4.0)*float(anhydrite_moles[1])*
    oxygen


#Apatite [Ca5(PO4)3(OH)]
if float(xrf_input.elox_p[9]) != float(0) or float(xrf_input.elox_p2o5[9]) != float(0):
  if "Apatite" not in calculation_list:
    yn = raw_input("Apatite is not in the calculation list, do you want to add it to the
     calculation list? (y/n) ")
    if yn =="y" or yn == "Y":
      calculation_list.append("Apatite")
```

```python
elif float(xrf_input.elox_p[9]) != float(0) and float(xrf_input.elox_p2o5[9]) != float(0):
  xrf_input.elox_p[9] = float(xrf_input.elox_p[9]) + float(xrf_input.elox_p2o5[9])
  xrf_input.elox_p2o5[9] = 0
  if "Apatite" not in calculation_list:
    yn = raw_input("Apatite is not in the calculation list, do you want to add it to the calculation list? (y/n) ")
    if yn =="y" or yn == "Y":
      calculation_list.append("Apatite")

if "Apatite" in calculation_list:
  if float(xrf_input.elox_p[9]) != int(0) and float(xrf_input.elox_p2o5[9]) == float(0) and float(xrf_input.elox_cao[9]) >= float(Fraction(5,3))*float(xrf_input.elox_p[9]):
    apatite_moles[1] = float(Fraction(1,3))*float(xrf_input.elox_p[9])
    xrf_input.elox_cao[9] = float(xrf_input.elox_cao[9]) - (5.0)*float(apatite_moles[1])
    xrf_input.elox_p[9] = float(xrf_input.elox_p[9]) -(3.0)*float(apatite_moles[1])
    xrf_input.weight_loss = xrf_input.weight_loss - (13.0)*float(apatite_moles[1])*oxygen -(1.0)*float(apatite_moles[1])*hydrogen
  elif float(xrf_input.elox_p[9]) == int(0) and float(xrf_input.elox_p2o5[9]) != int(0) and float(xrf_input.elox_cao[9]) >= float(Fraction(5,3))*float(xrf_input.elox_p2o5[9]):
    apatite_moles[1] = float(xrf_input.elox_p2o5[9])/(3.0)
    xrf_input.elox_cao[9] = float(xrf_input.elox_cao[9]) - (5.0)*float(apatite_moles[1])
    xrf_input.elox_p2o5[9] = float(xrf_input.elox_p2o5[9]) -(3.0)*float(apatite_moles[1])
    xrf_input.weight_loss = xrf_input.weight_loss - (13.0)*float(apatite_moles[1])*oxygen -(1.0)*float(apatite_moles[1])*hydrogen

#Halite [NaCl]
if float(xrf_input.elox_cl[9]) != float(0):
  if "Halite" not in calculation_list:
    yn = raw_input("The amount of chlorine is nonzero, and Halite is not in the calculation list, do you want to add it to the calculation list? (y/n) ")
    if yn == "y" or yn == "Y" or yn == "yes" or yn == "Yes":
      calculation_list.append("Halite")
if "Halite" in calculation_list:
  if xrf_input.elox_cl[9] != int(0) and float(xrf_input.elox_na2o[9]) >= float(Fraction(1,2))*float(xrf_input.elox_cl[9]):
    halite_moles[1] = float(xrf_input.elox_cl[9])
    xrf_input.elox_na2o[9] = float(xrf_input.elox_na2o[9]) - (0.5)*float(halite_moles[1])
    xrf_input.elox_cl[9] = float(xrf_input.elox_cl[9]) - float(halite_moles[1])


#Pyrite [FeS2]
if "Pyrite" in calculation_list:
  if float(xrf_input.elox_s[9]) != int(0) and float(xrf_input.elox_fe2o3[9]) >= (0.5)*float(xrf_input.elox_s[9]):
    pyrite_moles[1] = (0.5)*float(xrf_input.elox_s[9])
    xrf_input.elox_s[9] = float(xrf_input.elox_s[9]) - 2*float(pyrite_moles[1])
    xrf_input.elox_fe2o3[9] = float(xrf_input.elox_fe2o3[9]) - float(pyrite_moles[1])


#Rutile [TiO2]
if "Rutile" in calculation_list:
  if xrf_input.elox_tio2[9] != int(0):
    rutile_moles[1] = float(xrf_input.elox_tio2[9])
    xrf_input.elox_tio2[9] = float(xrf_input.elox_tio2[9]) - float(rutile_moles[1])
    xrf_input.weight_loss = xrf_input.weight_loss - 2*float(rutile_moles[1])*oxygen


#---------------------------------------------------------------------------------


mineral_list = (pyrite_moles, rutile_moles, halite_moles, anhydrite_moles, apatite_moles)

#display the mole amounts of elements after trace mineral allocation
print "\n\n\n"
print "{0:^100}".format("First allocation stage")
print "-"*100
print "\n"
print "{0:20} {1:20}".format("Element oxide", "Element mmol")
print "-"*40
for i in xrf_input.elox_list:
  if i[4] != int(0):
    print "{0:20} {1:<20}".format(i[0],float(i[9]))

#display the amounts of minerals after trace mineral allocation
print "\n\n"
print "{0:20} {1:20}".format("Mineral", "Moles")
print "-"*40
for i in mineral_list:
  print "{0:20} {1:<20}".format(i[0],float(i[1]))

print xrf_input.weight_loss

#write information to output file
file_out.write("\n\n\n\n\n")
file_out.write("After trace mineral allocation\n\n")
file_out.write("-"*100)
file_out.write("\n" + "{0:20} {1:20}".format("Element oxide", "Element mmol"))
file_out.write("\n" + "-"*40)
for i in xrf_input.elox_list:
  if i[4] != int(0):
    file_out.write("\n" + "{0:20} {1:<20}".format(i[0],float(i[9])))

file_out.write("\n\n")
file_out.write("{0:20} {1:20}".format("Mineral", "Moles"))
file_out.write("\n" + "-"*40)
for i in mineral_list:
  file_out.write("\n" + "{0:20} {1:<20}".format(i[0],float(i[1])))




#---------------------------------------------------------------------------#
#                                                                           #
#                        Second Allocation Stage                            #
#                                                                           #
#---------------------------------------------------------------------------#





#In the second allocation stage difficulties show up, when minerals with a great
#similarity are present in the test results. For example, Orthoclase and Muscovite.
#Therefore an arbitrary choice has to be made, what amount of a specific element
#is allocated to a mineral.
#To simplify this we create a subroutine, that checks if these minerals are present.

potassium_list = []
sodium_list = []
magnesium_list = []


mineral_list = (pyrite_moles, rutile_moles, halite_moles, anhydrite_moles, apatite_moles,
        chlorite_moles, glauconite_moles, muscovite_moles, illite_moles,
        montmorillonite_moles, albite_moles, anorthite_moles, orthoclase_moles)

print "\n\n\n\n"
print "{0:^100}".format("Second allocation stage")
print "-"*100

#If glauconite and montmorillonite are present, they can be calculated with different elements, here you can choose
if "Glauconite" in calculation_list:
  glauconite_list = int(raw_input("\n\nYou have selected Glauconite, the amount of this mineral can be calculated with different elements.\nWith which element do you want to calculate the quantity of Glauconite?\n 1. Potassium\n 2. Sodium\n 3. Magnesium\n"))
if "Montmorillonite" in calculation_list:
  montmorillonite_list = int(raw_input("\n\nYou have selected Montmorillonite, the amount of this mineral can be calculated with different elements.\nWith which
    element do you want to calculate the quantity of Montmorillonite?\n 1. Sodium\n 2.
    Magnesium\n"))


#There are multiple minerals that consist of potassium, if they are selected by the user
     they are transferred to a list
if "Orthoclase" in calculation_list:
  potassium_list.append(orthoclase_moles)
if "Muscovite" in calculation_list:
  potassium_list.append(muscovite_moles)
if "Illite" in calculation_list:
  potassium_list.append(illite_moles)
if "Glauconite" in calculation_list and int(glauconite_list) == 1:
  potassium_list.append(glauconite_moles)

#Same as above, but for sodium
if "Albite" in calculation_list:
  sodium_list.append(albite_moles)
if "Glauconite" in calculation_list and int(glauconite_list) == 2:
  sodium_list.append(glauconite_moles)
if "Montmorillonite" in calculation_list and int(montmorillonite_list) == 1:
  sodium_list.append(montmorillonite_moles)

#Same as above, but for magnesium
if "Chlorite" in calculation_list:
  magnesium_list.append(chlorite_moles)
if "Montmorillonite" in calculation_list and int(montmorillonite_list) == 2:
  magnesium_list.append(montmorillonite_moles)
if "Glauconite" in calculation_list and int(glauconite_list) == 3:
  magnesium_list.append(glauconite_moles)

print "\n\nThe list of minerals to be calculated with potassium contains:"
file_out.write("\n\nThe list of minerals to be calculated with potassium contains:\n")
if not potassium_list:
  print "List is empty"
  file_out.write("The list is empty")
else:
  for i in potassium_list:
    print i[0]
    file_out.write(str(i[0]) + "\n")
print "\nThe list of minerals to be calculated with sodium contains:"
file_out.write("\nThe list of minerals to be calculated with sodium contains:\n")
if not sodium_list:
  print "List is empty"
  file_out.write("The list is empty")
else:
  for i in sodium_list:
    print i[0]
    file_out.write(str(i[0]) + "\n")
print "\nThe list of minerals to be calculated with magnesium contains:"
file_out.write("\nThe list of minerals to be calculated with magnesium contains:\n")
if not magnesium_list:
  print "List is empty"
  file_out.write("The list is empty")
else:
  for i in magnesium_list:
    print i[0]
    file_out.write(str(i[0]) + "\n")

print "\n\n\n\n"

if len(potassium_list) > 1:
  print "\nThe number of minerals to be calculated with Potassium is " + str(len(
    potassium_list)) + ", therefore a distribution has to be made."
  print "The default distribution is " + str(float(Fraction(1,len(potassium_list)))) + "
     per mineral"
  print "But you can also use a custom distribution\n"
  potassium_custom = 0
  while potassium_custom == 0:
    potassium_custom = int(raw_input("Do you want to use the default distribution, or
    create a custom distribution?\n 1. Default\n 2. Custom\n"))
    if potassium_custom == 1:
      print "The default distribution will be used"
      for i in potassium_list:
        i[1] = float(Fraction(1,len(potassium_list)))*float(xrf_input.elox_k2o[9])
    elif potassium_custom == 2:
      print "\nYou can make a custom distribution\nPlease make sure the total equals 1.0\n"
      j = 1.0
      for i in potassium_list:
        print "The available percentage is " + str(j)
        i[1] = float(raw_input("Please enter the quantity for " + str(i[0]) + " "))*float(xrf_input.elox_k2o[9])
        j = j - float(i[1])/float(xrf_input.elox_k2o[9])
    else:
      print "The number you entered is invalid, please try again."
      potassium_custom = 0
elif len(potassium_list) == 1:
  for i in potassium_list:
    i[1] = float(xrf_input.elox_k2o[9])

if len(sodium_list) > 1:
  print "\nThe number of minerals to be calculated with Sodium is " + str(len(sodium_list)) + ", therefore a distribution has to be made."
  print "The default distribution is " + str(float(Fraction(1,len(sodium_list)))) + " per mineral"
  print "But you can also use a custom distribution\n"
  sodium_custom = 0
  while sodium_custom == 0:
    sodium_custom = int(raw_input("Do you want to use the default distribution, or create a custom distribution?\n 1. Default\n 2. Custom\n"))
    if sodium_custom == 1:
      print "The default distribution will be used"
      for i in sodium_list:
        i[1] = float(Fraction(1,len(sodium_list)))*float(xrf_input.elox_na2o[9])
    elif sodium_custom == 2:
      print "\nYou can make a custom distribution\nPlease make sure the total equals 1.0\n"
      j = 1.0
      for i in sodium_list:
        print "The available percentage is " + str(j)
        i[1] = float(raw_input("Please enter the quantity for " + str(i[0]) + " "))*float(xrf_input.elox_na2o[9])
        j = j - float(i[1])/float(xrf_input.elox_na2o[9])
    else:
      print "The number you entered is invalid, please try again."
      sodium_custom = 0
elif len(sodium_list) == 1:
  for i in sodium_list:
    i[1] = float(xrf_input.elox_na2o[9])

if len(magnesium_list) > 1:
  print "\nThe number of minerals to be calculated with magnesium is " + str(len(magnesium_list)) + ", therefore a distribution has to be made."
  print "The default distribution is " + str(float(Fraction(1,len(magnesium_list)))) + " per mineral"
  print "But you can also use a custom distribution\n"
  magnesium_custom = 0
  while magnesium_custom == 0:
    magnesium_custom = int(raw_input("Do you want to use the default distribution, or create a custom distribution?\n 1. Default\n 2. Custom\n"))
    if magnesium_custom == 1:
      print "The default distribution will be used"
      for i in magnesium_list:
        i[1] = float(Fraction(1,len(magnesium_list)))*float(xrf_input.elox_mgo[9])
    elif magnesium_custom == 2:
      print "\nYou can make a custom distribution\nPlease make sure the total equals 1.0\n"
      j = 1.0
      for i in magnesium_list:
        print "The available percentage is " + str(j)
        i[1] = float(raw_input("Please enter the quantity for " + str(i[0]) + " "))*float(xrf_input.elox_mgo[9])
        j = j - float(i[1])/float(xrf_input.elox_mgo[9])
    else:
      print "The number you entered is invalid, please try again."
```

```python
      magnesium_custom = 0
elif len(magnesium_list) == 1:
  for i in magnesium_list:
    i[1] = float(xrf_input.elox_mgo[9])

#Anorthite [CaAl2Si2O8]
if "Anorthite" in calculation_list:
  if float(xrf_input.elox_cao[9]) != int(0) and float(xrf_input.elox_al2o3[9]) >= (2.0)*float(xrf_input.elox_cao[9]) and float(xrf_input.elox_sio2[9]) >= (2.0)*float(xrf_input.elox_cao[9]):
    anorthite_moles[1] = float(xrf_input.elox_cao[9])
    xrf_input.elox_cao[9] = float(xrf_input.elox_cao[9]) - float(anorthite_moles[1])
    xrf_input.elox_al2o3[9] = float(xrf_input.elox_al2o3[9]) -(2.0)*float(anorthite_moles[1])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) - (2.0)*float(anorthite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) -(8.0)*float(anorthite_moles[1])*oxygen
  elif min(float(xrf_input.elox_cao[9]), float(xrf_input.elox_al2o3[9])/(2.0), float(xrf_input.elox_sio2[9])/(2.0)) < float(anorthite_moles[1]):
    anorthite_moles[1] = float(xrf_input.elox_cao[9])
    xrf_input.elox_cao[9] = float(xrf_input.elox_cao[9]) - float(anorthite_moles[1])
    xrf_input.elox_al2o3[9] = float(xrf_input.elox_al2o3[9]) -(2.0)*float(anorthite_moles[1])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) - (2.0)*float(anorthite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) -(8.0)*float(anorthite_moles[1])*oxygen


#Albite [NaAlSi3O8]
if "Albite" in calculation_list:
  if float(xrf_input.elox_na2o[9]) >= float(albite_moles[1]) and float(xrf_input.elox_al2o3[9]) >= float(albite_moles[1]) and float(xrf_input.elox_sio2[9]) >= (3.0)*float(albite_moles[1]):
    xrf_input.elox_na2o[9] = float(xrf_input.elox_na2o[9]) - float(albite_moles[1])
    xrf_input.elox_al2o3[9] = float(xrf_input.elox_al2o3[9]) - float(albite_moles[1])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) - (3.0)*float(albite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (8.0)*float(albite_moles[1])*oxygen
  elif min(float(xrf_input.elox_na2o[9]), float(xrf_input.elox_al2o3[9]), float(xrf_input.elox_sio2[9])/(3.0)) < float(albite_moles[1]):
    albite_moles[1] = min(float(xrf_input.elox_na2o[9]), float(xrf_input.elox_al2o3[9]), float(xrf_input.elox_sio2[9])/(3.0))
    xrf_input.elox_na2o[9] = float(xrf_input.elox_na2o[9]) - float(albite_moles[1])
    xrf_input.elox_al2o3[9] = float(xrf_input.elox_al2o3[9]) - float(albite_moles[1])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) - (3.0)*float(albite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (8.0)*float(albite_moles[1])*oxygen




#Chlorite [FeMg4Al(Si3Al)O10(OH)8]
if "Chlorite" in calculation_list:
  if float(xrf_input.elox_fe2o3[9]) >= float(chlorite_moles[1]) and float(xrf_input.elox_mgo[9]) >= (4.0)*float(chlorite_moles[1]) and float(xrf_input.elox_al2o3[9]) >= (2.0)*float(chlorite_moles[1]) and float(xrf_input.elox_sio2[9]) >= (3.0)*float(chlorite_moles[1]):
    xrf_input.elox_fe2o3[9] = float(xrf_input.elox_fe2o3[9]) - float(chlorite_moles[1])
    xrf_input.elox_mgo[9] = float(xrf_input.elox_mgo[9]) - (4.0)*float(chlorite_moles[1])
    xrf_input.elox_al2o3[9] = float(xrf_input.elox_al2o3[9]) - (2.0)*float(chlorite_moles[1])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) - (3.0)*float(chlorite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (18.0)*float(chlorite_moles[1])*oxygen - (8.0)*float(chlorite_moles[1])*hydrogen
  elif min(float(xrf_input.elox_fe2o3[9]), float(xrf_input.elox_mgo[9])/(4.0), float(xrf_input.elox_al2o3[9])/(2.0), float(xrf_input.elox_sio2[9])/(3.0)) < float(chlorite_moles[1]):
    chlorite_moles[1] = min(float(xrf_input.elox_fe2o3[9]), float(xrf_input.elox_mgo[9])/(4.0), float(xrf_input.elox_al2o3[9])/(2.0), float(xrf_input.elox_sio2[9])/(3.0))
    xrf_input.elox_fe2o3[9] = float(xrf_input.elox_fe2o3[9]) - float(chlorite_moles[1])
    xrf_input.elox_mgo[9] = float(xrf_input.elox_mgo[9]) - (4.0)*float(chlorite_moles[1])
    xrf_input.elox_al2o3[9] = float(xrf_input.elox_al2o3[9]) - (2.0)*float(chlorite_moles[1])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) - (3.0)*float(chlorite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (18.0)*float(chlorite_moles[1])*oxygen - (8.0)*float(chlorite_moles[1])*hydrogen


#Illite [KAl2Si4O10(OH)2]
if "Illite" in calculation_list:
  if float(xrf_input.elox_k2o[9]) >= float(illite_moles[1]) and float(xrf_input.elox_al2o3[9]) >= (2.0)*float(illite_moles[1]) and float(xrf_input.elox_sio2[9]) >= (4.0)*float(illite_moles[1]):
    xrf_input.elox_k2o[9] = float(xrf_input.elox_k2o[9]) - float(illite_moles[1])
    xrf_input.elox_al2o3[9] = float(xrf_input.elox_al2o3[9]) - (2.0)*float(illite_moles[1])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) -(4.0)*float(illite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (12.0)*float(illite_moles[1])*oxygen -(2.0)*float(illite_moles[1])*hydrogen
  elif min(float(xrf_input.elox_k2o[9]), float(xrf_input.elox_al2o3[9])/(2.0), float(xrf_input.elox_sio2[9])/(4.0)) < float(illite_moles[1]):
    illite_moles[1] = min(float(xrf_input.elox_k2o[9]), float(xrf_input.elox_al2o3[9])/(2.0), float(xrf_input.elox_sio2[9])/(4.0))
    xrf_input.elox_k2o[9] = float(xrf_input.elox_k2o[9]) - float(illite_moles[1])
    xrf_input.elox_al2o3[9] = float(xrf_input.elox_al2o3[9]) - (2.0)*float(illite_moles[1])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) -(4.0)*float(illite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (12.0)*float(illite_moles[1])*oxygen -(2.0)*float(illite_moles[1])*hydrogen


#Muscovite [K2Al4Si8O20(OH)4]
if "Muscovite" in calculation_list:
  muscovite_moles[1] = (0.5)*float(muscovite_moles[1])
  if float(xrf_input.elox_k2o[9]) >= (2.0)*float(muscovite_moles[1]) and float(xrf_input.elox_al2o3[9]) >= (4.0)*float(muscovite_moles[1]) and float(xrf_input.elox_sio2[9]) >= (8.0)*float(muscovite_moles[1]):
    xrf_input.elox_k2o[9] = float(xrf_input.elox_k2o[9]) - (2.0)*float(muscovite_moles[1])
    xrf_input.elox_al2o3[9] = float(xrf_input.elox_al2o3[9]) - (4.0)*float(muscovite_moles[1])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) - (8.0)*float(muscovite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (24.0)*float(muscovite_moles[1])*oxygen -(4.0)*float(muscovite_moles[1])*hydrogen
  elif min(float(xrf_input.elox_k2o[9]), float(xrf_input.elox_al2o3[9])/(2.0), float(xrf_input.elox_sio2[9])/(4.0)) < float(muscovite_moles[1]):
    muscovite_moles[1] = min(float(xrf_input.elox_k2o[9]), float(xrf_input.elox_al2o3[9])/(2.0), float(xrf_input.elox_sio2[9])/(4.0))
    xrf_input.elox_k2o[9] = float(xrf_input.elox_k2o[9]) - (2.0)*float(muscovite_moles[1])
    xrf_input.elox_al2o3[9] = float(xrf_input.elox_al2o3[9]) - (4.0)*float(muscovite_moles[1])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) - (8.0)*float(muscovite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (24.0)*float(muscovite_moles[1])*oxygen -(4.0)*float(muscovite_moles[1])*hydrogen


#Orthoclase [KAlSi3O8]
if "Orthoclase" in calculation_list:
  if float(xrf_input.elox_k2o[9]) >= float(orthoclase_moles[1]) and float(xrf_input.elox_al2o3[9]) >= float(orthoclase_moles[1]) and float(xrf_input.elox_sio2[9]) >= (3.0)*float(orthoclase_moles[1]):
    xrf_input.elox_k2o[9] = float(xrf_input.elox_k2o[9]) - float(orthoclase_moles[1])
    xrf_input.elox_al2o3[9] = float(xrf_input.elox_al2o3[9]) - float(orthoclase_moles[1])
```

```python
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) - (3.0)*float(orthoclase_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (8.0)*float(orthoclase_moles[1])*oxygen
  elif min(float(xrf_input.elox_k2o[9]), float(xrf_input.elox_al2o3[9]), float(xrf_input.elox_sio2[9])/(3.0)) < float(orthoclase_moles[1]):
    orthoclase_moles[1] = min(float(xrf_input.elox_k2o[9]), float(xrf_input.elox_al2o3[9]), float(xrf_input.elox_sio2[9])/(3.0))
    xrf_input.elox_k2o[9] = float(xrf_input.elox_k2o[9]) - float(orthoclase_moles[1])
    xrf_input.elox_al2o3[9] = float(xrf_input.elox_al2o3[9]) - float(orthoclase_moles[1])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) - (3.0)*float(orthoclase_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (8.0)*float(orthoclase_moles[1])*oxygen


#Glauconite [K0.6Na0.05Fe1.5Mg0.4Al0.3Si3.8O10(OH)2]
if "Glauconite" in calculation_list:
  if float(xrf_input.elox_k2o[9]) >= (0.6)*float(glauconite_moles[1]) and float(xrf_input.elox_na2o[9]) >= (0.05)*float(glauconite_moles[1]) and float(xrf_input.elox_fe2o3[9]) >= (1.5)*float(glauconite_moles[1]) and float(xrf_input.elox_mgo[9]) >= (0.4)*float(glauconite_moles[1]) and float(xrf_input.elox_al2o3[9]) >= (0.3)*float(glauconite_moles[1]) and float(xrf_input.elox_sio2[9]) >= (3.8)*float(glauconite_moles[1]):
    xrf_input.elox_k2o[9] = float(xrf_input.elox_k2o[9]) - (0.6)*float(glauconite_moles[1])
    xrf_input.elox_na2o[9] = float(xrf_input.elox_na2o[9]) - (0.05)*float(glauconite_moles[1])
    xrf_input.elox_fe2o3[9] = float(xrf_input.elox_fe2o3[9]) - (1.5)*float(glauconite_moles[1])
    xrf_input.elox_mgo[9] = float(xrf_input.elox_mgo[9]) - (0.4)*float(glauconite_moles[1])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) - (3.8)*float(glauconite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (12.0)*float(glauconite_moles[1])*oxygen - (2.0)*float(glauconite_moles[1])*hydrogen
  elif min((0.6)*float(xrf_input.elox_k2o[9]), float(xrf_input.elox_na2o[9])/(0.05), float(xrf_input.elox_fe2o3[9])/(1.5), float(xrf_input.elox_mgo[9])/(0.4), float(xrf_input.elox_al2o3[9])/(0.3), float(xrf_input.elox_sio2[9])/(3.8)) < float(glauconite_moles[1]):
    glauconite_moles[1] = min(float(xrf_input.elox_k2o[9])/(0.6), float(xrf_input.elox_na2o[9])/(0.05), float(xrf_input.elox_fe2o3[9])/(1.5), float(xrf_input.elox_mgo[9])/(0.4), float(xrf_input.elox_al2o3[9])/(0.3), float(xrf_input.elox_sio2[9])/(3.8))
    xrf_input.elox_k2o[9] = float(xrf_input.elox_k2o[9]) - (0.6)*float(glauconite_moles[1])
    xrf_input.elox_na2o[9] = float(xrf_input.elox_na2o[9]) - (0.05)*float(glauconite_moles[1])
    xrf_input.elox_fe2o3[9] = float(xrf_input.elox_fe2o3[9]) - (1.5)*float(glauconite_moles[1])
    xrf_input.elox_mgo[9] = float(xrf_input.elox_mgo[9]) - (0.4)*float(glauconite_moles[1])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) - (3.8)*float(glauconite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (12.0)*float(glauconite_moles[1])*oxygen - (2.0)*float(glauconite_moles[1])*hydrogen


#Montmorillonite [Ca0.17Na0.31Mg0.33Al1.67Si4O10(OH)2.61]
if "Montmorillonite" in calculation_list:
  if float(xrf_input.elox_cao[9]) >= (0.17)*float(montmorillonite_moles[1]) and float(xrf_input.elox_na2o[9]) >= (0.31)*float(montmorillonite_moles[1]) and float(xrf_input.elox_mgo[9]) >= (0.33)*float(montmorillonite_moles[1]) and float(xrf_input.elox_al2o3[9]) >= (1.67)*float(montmorillonite_moles[1]) and float(xrf_input.elox_sio2[9]) >= (4.0)*float(montmorillonite_moles[1]):
    xrf_input.elox_cao[9] = float(xrf_input.elox_cao[9]) - (0.17)*float(montmorillonite_moles[1])
    xrf_input.elox_na2o[9] = float(xrf_input.elox_na2o[9]) - (0.31)*float(montmorillonite_moles[1])
    xrf_input.elox_mgo[9] = float(xrf_input.elox_mgo[9]) - (0.33)*float(montmorillonite_moles[1])
    xrf_input.elox_al2o3[9] = float(xrf_input.elox_al2o3[9]) - (1.67)*float(montmorillonite_moles[1])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) - (4.0)*float(montmorillonite_moles[1])
    xrf_input.weight_loss = xrf_input.weight_loss - (12.61)*float(montmorillonite_moles[1])*oxygen - (2.61)*float(montmorillonite_moles[1])*hydrogen
  elif min(float(xrf_input.elox_sio2[9])/(4.0), float(xrf_input.elox_al2o3[9])/(1.67), float(xrf_input.elox_al2o3[9])/(0.17), float(xrf_input.elox_mgo[9])/(0.33), float(xrf_input.elox_na2o[9])/(0.31)) < float(montmorillonite_moles[1]):
    montmorillonite_moles[1] = min(float(xrf_input.elox_sio2[9])/(4.0), float(xrf_input.elox_al2o3[9])/(1.67), float(xrf_input.elox_al2o3[9])/(0.17), float(xrf_input.elox_mgo[9])/(0.33), float(xrf_input.elox_na2o[9])/(0.31))
    xrf_input.elox_cao[9] = float(xrf_input.elox_cao[9]) - (0.17)*float(montmorillonite_moles[1])
    xrf_input.elox_na2o[9] = float(xrf_input.elox_na2o[9]) - (0.31)*float(montmorillonite_moles[1])
    xrf_input.elox_mgo[9] = float(xrf_input.elox_mgo[9]) - (0.33)*float(montmorillonite_moles[1])
    xrf_input.elox_al2o3[9] = float(xrf_input.elox_al2o3[9]) - (1.67)*float(montmorillonite_moles[1])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) - (4.0)*float(montmorillonite_moles[1])
    xrf_input.weight_loss = xrf_input.weight_loss - (12.61)*float(montmorillonite_moles[1])*oxygen - (2.61)*float(montmorillonite_moles[1])*hydrogen


#display the mole amounts of elements after trace mineral allocation
print "Mole amounts after second allocation\n"
print "{0:20} {1:20}".format("Element oxide", "Element mmol")
print "-"*40
for i in xrf_input.elox_list:
  if i[4] != int(0):
    print "{0:20} {1:<20}".format(i[0],i[9])

#display the amounts of minerals after second mineral allocation stage
print "\n\n"
print "{0:20} {1:20}".format("Mineral", "Moles")
print "-"*40
for i in mineral_list:
  print "{0:20} {1:20}".format(i[0],i[1])

print xrf_input.weight_loss


#write information to output file
file_out.write("\n\n\n\n\n")
file_out.write("After second allocation stage\n\n")
file_out.write("\n" + "{0:20} {1:20}".format("Element oxide", "Element mmol"))
file_out.write("\n" + "-"*40)
for i in xrf_input.elox_list:
  if i[4] != int(0):
    file_out.write("\n" + "{0:20} {1:<20}".format(i[0],i[9]))

file_out.write("\n\n")
file_out.write("{0:20} {1:20}".format("Mineral", "Moles"))
file_out.write("\n" + "-"*40)
for i in mineral_list:
  file_out.write("\n" + "{0:20} {1:<20}".format(i[0],i[1]))



#-----------------------------------------------------------------------------#
#                                                                             #
#                       Third Allocation Stage                                #
#                                                                             #
#-----------------------------------------------------------------------------#

print "\n\n"
print "{0:^100}".format("Third allocation stage")
print "-"*100
print "\n\n"


```

```python

calcium_list = []
magnesium_list = []
aluminium_list = []



if "Dolomite" in calculation_list:
  dolomite_list = int(raw_input("\n\nYou have selected Dolomite, the amount of this mineral can be calculated with different elements.\nWith which element do you want to calculate the quantity of Dolomite?\n 1. Calcium\n 2. Magnesium\n"))

if "Calcite" in calculation_list:
  calcium_list.append(calcite_moles)
if "Dolomite" in calculation_list and int(dolomite_list) == 1:
  calcium_list.append(dolomite_moles)

if "Magnesite" in calculation_list:
  magnesium_list.append(magnesite_moles)
if "Dolomite" in calculation_list and int(dolomite_list) ==2:
  magnesium_list.append(dolomite_moles)


if len(calcium_list) > 1:
  print "\nThe number of minerals to be calculated with calcium is " + str(len(calcium_list)) + ", therefore a distribution has to be made."
  print "The minerals to be calculated with calcium are: "
  for i in calcium_list:
    print i[0]
  print "The default distribution is " + str(float(Fraction(1,len(calcium_list)))) + " per mineral"
  print "But you can also use a custom distribution\n"
  calcium_custom = 0
  while calcium_custom == 0:
    calcium_custom = int(raw_input("Do you want to use the default distribution, or create a custom distribution?\n 1. Default\n 2. Custom\n"))
    if calcium_custom == 1:
      print "The default distribution will be used"
      for i in calcium_list:
        i[1] = float(Fraction(1,len(calcium_list)))*float(xrf_input.elox_cao[9])
    elif calcium_custom == 2:
      print "\nYou can make a custom distribution\nPlease make sure the total equals 1.0\n"
      j = 1.0
      for i in calcium_list:
        print "The available percentage is " + str(j)
        i[1] = float(raw_input("Please enter the quantity for " + str(i[0]) + " "))*float(xrf_input.elox_cao[9])
        j = j - float(i[1])/float(xrf_input.elox_cao[9])
    else:
      print "The number you entered is invalid, please try again."
      calcium_custom = 0
elif len(calcium_list) == 1:
  for i in calcium_list:
    i[1] = float(xrf_input.elox_cao[9])

if len(magnesium_list) > 1:
  print "\nThe number of minerals to be calculated with magnesium is " + str(len(magnesium_list)) + ", therefore a distribution has to be made."
  print "The default distribution is " + str(float(Fraction(1,len(magnesium_list)))) + " per mineral"
  print "But you can also use a custom distribution\n"
  magnesium_custom = 0
  while magnesium_custom == 0:
    magnesium_custom = int(raw_input("Do you want to use the default distribution, or create a custom distribution?\n 1. Default\n 2. Custom\n"))
    if magnesium_custom == 1:
      print "The default distribution will be used"
      for i in magnesium_list:
        i[1] = float(Fraction(1,len(magnesium_list)))*float(xrf_input.elox_mgo[9])
    elif magnesium_custom == 2:
      print "\nYou can make a custom distribution\nPlease make sure the total equals 1.0\n"
      j = 1.0
      for i in magnesium_list:
        print "The available percentage is " + str(j)
        i[1] = float(raw_input("Please enter the quantity for " + str(i[0]) + " "))*float(xrf_input.elox_mgo[9])
        j = j - float(i[1])/float(xrf_input.elox_mgo[9])
    else:
      print "The number you entered is invalid, please try again."
      magnesium_custom = 0
elif len(magnesium_list) == 1:
  for i in magnesium_list:
    i[1] = float(xrf_input.elox_mgo[9])

#Siderite [FeCO3]
if "Siderite" in calculation_list:
  if xrf_input.elox_fe2o3 != int(0):
    siderite_moles[1] = float(xrf_input.elox_fe2o3[9])
    xrf_input.elox_fe2o3[9] = float(xrf_input.elox_fe2o3[9]) - float(siderite_moles[1])
    xrf_input.weight_loss = xrf_input.weight_loss - (3.0)*float(siderite_moles[1])*oxygen -float(siderite_moles[1])*carbon


#Calcite [CaCO3]
if "Calcite" in calculation_list:
  if float(xrf_input.elox_cao[9]) >= float(calcite_moles[1]):
    xrf_input.elox_cao[9] = float(xrf_input.elox_cao[9]) - float(calcite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (3.0)*float(calcite_moles[1])*oxygen - float(calcite_moles[1])*carbon


#Dolomite [CaMg(CO3)2]
if "Dolomite" in calculation_list:
  if float(xrf_input.elox_cao[9]) >= float(dolomite_moles[1]) and float(xrf_input.elox_mgo[9]) >= float(dolomite_moles[1]):
    xrf_input.elox_cao[9] = float(xrf_input.elox_cao[9]) - float(dolomite_moles[1])
    xrf_input.elox_mgo[9] = float(xrf_input.elox_mgo[9]) - float(dolomite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (6.0)*float(dolomite_moles[1])*oxygen - (2.0)*float(dolomite_moles[1])*carbon


#Magnesite [MgCO3]
if "Magnesite" in calculation_list:
  if float(xrf_input.elox_mgo[9]) >= float(magnesite_moles[1]):
    xrf_input.elox_mgo[9] = float(xrf_input.elox_mgo[9]) - float(magnesite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (3.0)*float(magnesite_moles[1])*oxygen - float(magnesite_moles[1])*carbon


#Hematite [Fe2O3]
if "Hematite" in calculation_list:
  if xrf_input.elox_fe2o3 != int(0):
    hematite_moles[1] = (0.5)*float(xrf_input.elox_fe2o3[9])
    xrf_input.elox_fe2o3[9] = float(xrf_input.elox_fe2o3[9]) - (2.0)*float(hematite_moles[1])
    xrf_input.weight_loss = xrf_input.weight_loss - (3.0)*float(hematite_moles[1])*oxygen


#Goethite [FeO(OH)]
if "Goethite" in calculation_list:
  if xrf_input.elox_fe2o3 != int(0):
    goethite_moles[1] = float(xrf_input.elox_fe2o3[9])
    xrf_input.elox_fe2o3[9] = float(xrf_input.elox_fe2o3[9]) - float(goethite_moles[1])
    xrf_input.weight_loss = xrf_input.weight_loss - (2.0)*float(goethite_moles[1])*oxygen - 1*float(goethite_moles[1])*hydrogen

print calculation_list

if "Gibbsite" in calculation_list:
  aluminium_list.append(gibbsite_moles)
if "Kaolinite" in calculation_list:
  aluminium_list.append(kaolinite_moles)

```

```

if len(aluminium_list) > 1:
  print "\nThe remainder of Aluminium can be divided between Kaolinite and Gibbsite, or can be allocated to one mineral."
  print "\nThe number of minerals to be calculated with aluminium is " + str(len(aluminium_list)) + ", therefore a distribution has to be made."
  print "The default distribution is " + str(float(Fraction(1,len(aluminium_list)))) + " per mineral"
  print "But you can also use a custom distribution\n"
  aluminium_custom = 0
  while aluminium_custom == 0:
    aluminium_custom = int(raw_input("Do you want to use the default distribution, or create a custom distribution?\n 1. Default\n 2. Custom\n"))
    if aluminium_custom == 1:
      print "The default distribution will be used"
      for i in aluminium_list:
        i[1] = float(Fraction(1,len(aluminium_list)))*float(xrf_input.elox_al2o3[9])
    elif aluminium_custom == 2:
      print "\nYou can make a custom distribution\nPlease make sure the total equals 1.0\n"
      j = 1.0
      for i in aluminium_list:
        print "The available percentage is " + str(j)
        i[1] = float(raw_input("Please enter the quantity for " + str(i[0]) + " "))*float(xrf_input.elox_al2o3[9])
        j = j - float(i[1])/float(xrf_input.elox_al2o3[9])
    else:
      print "The number you entered is invalid, please try again."
      aluminium_custom = 0
elif len(aluminium_list) == 1:
  for i in aluminium_list:
    i[1] = float(xrf_input.elox_al2o3[9])

#Gibbsite [Al(OH)3]
if "Gibbsite" in calculation_list:
  if xrf_input.elox_al2o3[9] != int(0):
    xrf_input.elox_al2o3[9] = float(xrf_input.elox_al2o3[9]) - float(gibbsite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (3.0)*float(gibbsite_moles[1])*oxygen - (3.0)*float(gibbsite_moles[1])*hydrogen

#Kaolinite [Al2Si2O5(OH)4]
if "Kaolinite" in calculation_list:
  if float(xrf_input.elox_al2o3[9]) != int(0) and float(xrf_input.elox_sio2[9]) != int(0):
    kaolinite_moles[1] = float(0.5)*float(kaolinite_moles[1])
    xrf_input.elox_al2o3[9] = float(xrf_input.elox_al2o3[9]) - (2.0)*float(kaolinite_moles[1])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) - (2.0)*float(kaolinite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (9.0)*float(kaolinite_moles[1])*oxygen - (4.0)*float(kaolinite_moles[1])*hydrogen
  elif min(float(xrf_input.elox_al2o3[9])/(2.0), float(xrf_input.elox_sio2[9])/(2.0)) < float(kaolinite_moles[1]):
    kaolinite_moles[1] = min(float(xrf_input.elox_al2o3[9])/(2.0), float(xrf_input.elox_sio2[9])/(2.0))
    xrf_input.elox_al2o3[9] = float(xrf_input.elox_al2o3[9]) - (2.0)*float(kaolinite_moles[1])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) - (2.0)*float(kaolinite_moles[1])
    xrf_input.weight_loss = float(xrf_input.weight_loss) - (9.0)*float(kaolinite_moles[1])*oxygen - (4.0)*float(kaolinite_moles[1])*hydrogen

#Quartz [SiO4]
if "Quartz" in calculation_list:
  if xrf_input.elox_sio2 != int(0):
    quartz_moles[1] = float(xrf_input.elox_sio2[9])
    xrf_input.elox_sio2[9] = float(xrf_input.elox_sio2[9]) - float(quartz_moles[1])
    xrf_input.weight_loss = xrf_input.weight_loss - (4.0)*float(quartz_moles[1])*oxygen


mineral_list = (pyrite_moles, hematite_moles, rutile_moles, gibbsite_moles,
        goethite_moles, halite_moles, calcite_moles, dolomite_moles,
        magnesite_moles, siderite_moles, anhydrite_moles, apatite_moles,
        chlorite_moles, glauconite_moles, muscovite_moles, kaolinite_moles,
        illite_moles, montmorillonite_moles, quartz_moles,
        albite_moles, anorthite_moles, orthoclase_moles)

#display the mole amounts of elements after trace mineral allocation
print "\nAfter third allocation stage\n"
print "{0:20} {1:20}".format("Element oxide", "Element mmol")
print "-"*40
for i in xrf_input.elox_list:
  if i[4] != int(0):
    print "{0:20} {1:<20}".format(i[0],float(i[9]))

#display the amounts of minerals after third mineral allocation stage
print "\n\n"
print "{0:20} {1:20}".format("Mineral", "Moles")
print "-"*40
for i in mineral_list:
  print "{0:20} {1:<20}".format(i[0],float(i[1]))

print xrf_input.weight_loss

#write information to output file
file_out.write("\n\n\n\n\n")
file_out.write("After third allocation stage\n\n")
file_out.write("\n" + "{0:20} {1:20}".format("Element oxide", "Element mmol"))
file_out.write("\n" + "-"*40)
for i in xrf_input.elox_list:
  if i[4] != int(0):
    file_out.write("\n" + "{0:20} {1:<20}".format(i[0],float(i[9])))

file_out.write("\n\n")
file_out.write("{0:20} {1:20}".format("Mineral", "Moles"))
file_out.write("\n" + "-"*40)
for i in mineral_list:
  file_out.write("\n" + "{0:20} {1:<20}".format(i[0],float(i[1])))


#Calculate weight of mineral from molar mass, and molar quantity
for (i,j) in zip(mineral_list, mineral_data.all_minerals):
  i[2] = float(i[1]) * float(j[3])

#Calculate back to weight percentages
for i in mineral_list:
  i[3] = (float(i[2])/float(xrf_input.total_weight_sample))*float(100)

#Calculate volume of mineral
for (i,j) in zip(mineral_list, mineral_data.all_minerals):
  i[4] = float(i[1]) * float(j[4])

total_volume = (0.0)
#Calculate total volume
for i in mineral_list:
  total_volume = float(total_volume) + float(i[4])

#Calculate percentage of total volume
for i in mineral_list:
  i[5] = (float(i[4])/float(total_volume))*float(100)



#display weight and volume percentages
print "\n\n\n\n"
print "{0:^100}".format("Final result")
print "-"*100
print "\n\n"
print "{0:20} {1:20} {2:20}".format("Mineral", "wt%", "Vol. %")
print "-"*60
for i in mineral_list:
  print "{0:20} {1:<20} {2:20}".format(i[0], str(float(i[3])) + " %", str(float(i[5])) +
    " %")

svalues = ([])
slabels = ([])
```

```
sindex = ([])
for i in mineral_list:
  svalues.append(float(i[3]))
  slabels.append(i[0])

sindex = ([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21])


plt.barh(sorted(sindex, reverse=True), svalues, height=0.25)
pylab.yticks(sorted(sindex, reverse=True), slabels)
plt.title(user_reference)
pylab.xlim([0,100])
pylab.xlabel("Weight percentages")
ax = plt.gca()
ax.grid(True)
plt.savefig(filename + ".pdf", bbox_inches="tight")




#write weight and volume percentages to file
file_out.write("\n\n\n\n")
file_out.write("{0:100}".format("Final result"))
file_out.write("\n\n")
file_out.write("{0:20} {1:20} {2:20}".format("Mineral", "wt%", "Vol. %"))
file_out.write("\n" + "-"*60)
for i in mineral_list:
  file_out.write("\n" + "{0:20} {1:<20} {2:20}".format(i[0], str(float(i[3])) + " %", str(float(i[5])) + " %"))
file_out.write("\n\n")



file_out.close()
```

```
###########################################################################
# XRF data input module                                                   #
# This module handles the input of the element-oxides, and gives a       #
# message when you reached 100%                                           #
#                                                                         #
# A small explanation, the total available amount is of course 100%      #
# weight. Assigning weight to a particular element oxide will cause      #
# subtraction of that amount of the total. Therefore, it checks for      #
# each element oxide if there is mass available, if there isn't, it      #
# will return that you have used all available mass.                     #
###########################################################################

from numpy import*

#First we will create the element oxide arrays, in which the data will be stored.
#Note that numpy arrays fields will be formatted as 'numpy-string'.
#elox_xx = array([name, molweight elox, element, molweight element, wt%, true weight, elox moles, conversion factor, element weight, element moles])
#                    0              1            2                3             4        5        6                7                   8                  9

elox_f = array(["F", 19.00, "F", 19.00, 0, 0, 0, 1, 0, 0])
elox_na2o = array(["Na2O", 61.98, "Na", 22.99, 0, 0, 0, 0.742, 0, 0])
elox_mgo = array(["MgO", 40.31, "Mg", 24.31, 0, 0, 0, 0.603, 0, 0])
elox_al2o3 = array(["Al2O3", 101.96, "Al", 26.98, 0 ,0, 0, 0.529, 0, 0])
elox_sio2 = array(["SiO2", 60.09, "Si", 28.09, 0, 0, 0, 0.467, 0, 0])
elox_p2o5 = array(["P2O5", 141.94, "P", 30.97, 0, 0, 0, 0.436, 0, 0])
elox_p = array(["P", 30.97, "P", 30.97, 0, 0, 0, 1, 0, 0])
elox_so3 = array(["SO3", 80.07, "S", 32.07, 0, 0, 0, 0.401, 0, 0])
elox_s = array(["S", 32.07, "S", 32.07, 0, 0, 0, 1, 0, 0])
elox_cl = array(["Cl", 35.45, "Cl", 35.45, 0, 0, 0, 1, 0, 0])
elox_k2o = array(["K2O", 94.20, "K", 39.10, 0, 0, 0, 0.83, 0, 0])
elox_cao = array(["CaO", 56.08, "Ca", 40.08, 0, 0, 0, 0.715, 0, 0])
elox_tio2 = array(["TiO2", 79.87, "Ti", 47.87, 0, 0, 0, 0.599, 0, 0])
elox_fe2o3 = array(["Fe2O3", 159.70, "Fe", 55.85, 0, 0, 0, 0.699, 0, 0])
elox_h2o = array(["H2O", 18.0018, "H", 1.008, 0, 0 , 0, 0.112, 0, 0])
elox_co2 = array(["CO2", 44.01, "CO2", 44.01, 0, 0, 0, 0.364, 0, 0])
elox_o = array(["O2", 32.00, "O2", 32.00, 0, 0, 0, 0, 0, 0])
```

```
elox_list=(elox_f, elox_na2o, elox_mgo, elox_al2o3, elox_sio2, elox_p2o5, elox_p,
    elox_so3, elox_s, elox_cl, elox_k2o, elox_cao, elox_tio2, elox_fe2o3)

elox_total = float(100)

weight_loss = float(0)

print "\nPlease fill in the wt% for the element-oxides, without wt%"

if elox_total > 0:
  print "\nAvailable mass = " + str(elox_total)
  elox_f[4] = float(raw_input("Please fill in the value for F  "))
  elox_total = elox_total - float(elox_f[4])

if elox_total > 0:
  print "\nAvailable mass = " + str(elox_total)
  elox_na2o[4] = float(raw_input("Please fill in the value for Na2O  "))
  elox_total = elox_total - float(elox_na2o[4])

if elox_total > 0:
  print "\nAvailable mass = " + str(elox_total)
  elox_mgo[4] = float(raw_input("Please fill in the value for MgO  "))
  elox_total = elox_total - float(elox_mgo[4])

if elox_total > 0:
  print "\nAvailable mass = " + str(elox_total)
  elox_al2o3[4] = float(raw_input("Please fill in the value for Al2O3  "))
  elox_total = elox_total - float(elox_al2o3[4])

if elox_total > 0:
  print "\nAvailable mass = " + str(elox_total)
  elox_sio2[4] = float(raw_input("Please fill in the value for SiO2  "))
  elox_total = elox_total - float(elox_sio2[4])

if elox_total > 0:
  print "\nAvailable mass = " + str(elox_total)
  elox_p2o5[4] = float(raw_input("Please fill in the value for P2O5  "))
  elox_total = elox_total - float(elox_p2o5[4])

if elox_total > 0:
  print "\nAvailable mass = " + str(elox_total)
  elox_p[4] = float(raw_input("Please fill in the value for P  "))
  elox_total = elox_total - float(elox_p[4])

if elox_total > 0:
  print "\nAvailable mass = " + str(elox_total)
  elox_so3[4] = float(raw_input("Please fill in the value for SO3  "))
  elox_total = elox_total - float(elox_so3[4])

if elox_total > 0:
  print "\nAvailable mass = " + str(elox_total)
  elox_s[4] = float(raw_input("Please fill in the value for S  "))
  elox_total = elox_total - float(elox_s[4])

if elox_total > 0:
  print "\nAvailable mass = " + str(elox_total)
  elox_cl[4] = float(raw_input("Please fill in the value for Cl  "))
  elox_total = elox_total - float(elox_cl[4])

if elox_total > 0:
  print "\nAvailable mass = " + str(elox_total)
  elox_k2o[4] = float(raw_input("Please fill in the value for K2O  "))
  elox_total = elox_total - float(elox_k2o[4])

if elox_total > 0:
  print "\nAvailable mass = " + str(elox_total)
  elox_cao[4] = float(raw_input("Please fill in the value for CaO  "))
  elox_total = elox_total - float(elox_cao[4])

if elox_total > 0:
  print "\nAvailable mass = " + str(elox_total)
  elox_tio2[4] = float(raw_input("Please fill in the value for TiO2  "))
  elox_total = elox_total - float(elox_tio2[4])

if elox_total > 0:
  print "\nAvailable mass = " + str(elox_total)
  elox_fe2o3[4] = float(raw_input("Please fill in the value for Fe2O3  "))
  elox_total = elox_total - float(elox_fe2o3[4])



#if the available amount reaches zero, you can't assign mass to another element-oxide.
#therefore, it will ask for the other test data.
elif elox_total == 0:
  print "\nYou have used all available mass."

#The other case, if weight percentage exceeds 100 percent. Data has to be filled in again.
elif elox_total < 0:
  print "\nThe input is invalid, please fill in correct amounts."
  import xrf_input

#Sample weight, needed to convert weight percentage to actual weight.
weight = raw_input("\nPlease fill in the total weight of the sample (in mg)  ")
if weight == "":
  total_weight_sample = 1000.0
elif weight > 0:
  total_weight_sample = float(weight)


if elox_total > 0:
  weight_loss = ((float(elox_total)/100) * total_weight_sample) + float(weight_loss)


#If data is not normalized, the data will not add up to 100%, the difference is considered as weight-loss.
#weight_loss = weight_loss + total_before_normalization

###########################################################################

#Data input is correct, the next step is to convert weight percentage to actual weight, and molar quantities.
#elox_xx = array([name, molweight elox, element, molweight element, wt%, true weight, elox moles, conversion factor, element weight, element moles])
#                     0              1           2                 3             4         5        6                 7                    8                  9


#convert weight percentage to actual weight (in mg)
for i in elox_list:
  i[5] = (float(i[4]) / 100)*total_weight_sample

#convert elox weight to elox moles
for i in elox_list:
  i[6] = float(i[5])/float(i[1])

#convert elox weight to mass of specific element by using conversion factor
for i in elox_list:
  i[8] = float(i[5])*float(i[7])

#convert element weight to element moles
for i in elox_list:
  i[9] = float(i[8])/float(i[3])

#add the oxygen in the element oxides to weight loss
for i in elox_list:
  weight_loss = ((1-float(i[7]))*float(i[5])) + float(weight_loss)

print "\nThe total weight loss is: " + str(weight_loss)
```

```
#mineral data for calculation, the inputs are density, mass and volume.


from numpy import*

```

```
#mineralname_data = array([density, mass, volume])

example = (["Name", "Chemical formula", "Density", "Mass", "Volume","Calculation Element","Number"])
#              0                 1                  2          3          4                  5              6


Pyrite = array(["Pyrite", "FeS2", 5.01, 119.99, 23.95, "S,Fe", 1])
Hematite = array(["Hematite", "Fe2O3", 5.3, 159.7, 30.13, "Fe", 2])
Rutile = array(["Rutile", "TiO2", 4.25, 79.87, 18.79, "Ti", 3])
Gibbsite = array(["Gibbsite", "Al(OH)3", 2.34, 78.004, 33.34, "Al", 4])
Goethite = array(["Goethite", "FeO(OH)", 3.8, 88.858, 23.38, "Fe", 5])
Halite = array(["Halite", "NaCl", 2.17, 58.44, 26.93, "Cl,Na", 6])
Calcite = array(["Calcite", "CaCO3", 2.71, 100.09, 36.93, "Ca", 7])
Dolomite = array(["Dolomite", "CaMg(CO3)2", 2.84, 184.41, 64.39, "Ca,Mg", 8])
Magnesite = array(["Magnesite", "MgCO3", 3, 84.32, 28.11, "Mg", 9])
Siderite = array(["Siderite", "FeCO3", 3.96, 115.86, 29.26, "Fe", 10])
Anhydrite = array(["Anhydrite", "CaSO4", 2.97, 136.95, 46.11, "Ca,S", 11])
Apatite = array(["Apatite", "Ca5(PO4)3(OH)", 3.19, 506.318, 158.72, "P,Ca", 12])
Chlorite = array(["Chlorite", "FeMg4Al(Si3Al)O10(OH)8", 2.65, 587.384, 221.65, "Mg,Fe",
    13])
Glauconite = array(["Glauconite", "K0.6Na0.05Fe1.5Mg0.4Al0.3Si3.8O10(OH)2", 2.67,
    426.93, 159.90, "", 14, 0])
Muscovite = array(["Muscovite", "K2Al4(Si6Al2)O20", 2.82, 796.652, 282.50, "K", 15, 0])
Kaolinite = array(["Kaolinite", "Al2Si2O5(OH)4", 2.6, 258.172, 99.30, "Al", 16])
Illite = array(["Illite", "KAl2(Si3Al)O10(OH)2", 2.75, 398.326, 144.85, "K", 17, 0])
Montmorillonite = array(["Montmorillonite", "(Ca0.17Na0.31Mg0.33Al1.67)Si4O10(OH)2.61",
    2.35, 383.77, 163.30, "Ca,Na,Mg", 18])
Quartz = array(["Quartz", "SiO2", 2.62, 60.09, 22.94, "Si", 19])
Albite = array(["Albite", "NaAlSi3O8", 2.62, 262.24, 100.09, "Na", 20])
Anorthite = array(["Anorthite", "CaAl2Si2O8", 2.73, 279.02, 102.21, "Ca", 21])
Orthoclase = array(["Orthoclase", "KAlSi3O8", 2.56, 278.32, 108.73, "K", 22, 0])


all_minerals = array([Pyrite, Hematite, Rutile, Gibbsite, Goethite, Halite,
          Calcite, Dolomite, Magnesite, Siderite, Anhydrite, Apatite,
          Chlorite, Glauconite, Muscovite, Kaolinite, Illite, Montmorillonite,
          Quartz, Albite, Anorthite, Orthoclase])
```

```
import mineral_data

edit_mode = "y"
while edit_mode == "y":
  edit_mineral = int(raw_input("\n For which mineral do you want to edit the data? \n 1.
    Pyrite\n 2. Hematite\n 3. Rutile\n 4. Gibbsite\n 5. Goethite\n 6. Halite\n 7.
    Calcite\n 8. Dolomite\n 9. Magnesite\n 10. Siderite\n 11. Anhydrite\n 12. Apatite\n
    13. Chlorite\n 14. Glauconite\n 15. Muscovite\n 16. Kaolinite\n 17. Illite\n 18.
    Montmorillonite\n 19. Quartz\n 20. Albite\n 21. Anorthite\n 22. Orthoclase\n"))
  edit_mineral_data = int(raw_input("\n What do you want to change? \n 1.Density \n 2.
    Mass \n 3.Volume \n"))
  mineral_data.all_minerals[edit_mineral -1][edit_mineral_data+1] = float(raw_input("
    Please type in the new value "))
  print mineral_data.all_minerals[edit_mineral -1]
  edit_mode = raw_input("\n Do you want to keep on editing? (y/n) ")
```