Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

A Computational System for the Calculation of the Pressure Dependence of Ionic Radii in Crystals

Maria Carolina Monard

Alicia Batana

Jorge Bruno

Nº 82

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos

Janeiro/1999

A Computational System for the Calculation of the Pressure Dependence of Ionic Radii in Crystals

Maria Carolina Monard*
Universidade de São Paulo/ILTC
Instituto de Ciências Matemáticas de São Carlos
Departamento de Ciências de Computação e Estatística
Caixa Postal 668, 13560-970 - São Carlos, SP, Brasil
e-mail: mcmonard@icmsc.sc.usp.br

Alicia Batana†
Jorge Bruno†
Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Química Inorgánica Analítica y Química Física
Ciudad Universitaria - Pabellón 2
Buenos Aires - Argentina
e-mail: {batana,jbruno}@q1.fcen.uba.ar

Abstract

A complete computational system has been developed in Maple V that allows for the calculation of anion polarizabilities of ionic crystals (based on the Clausius-Mossoti treatment) and its pressure dependence. It has been applied to alkali halides for which the results may be compared to those obtained from experimental data. It opens the possibility of its further use for the evaluation of the pressure dependence of the dielectric constant from the knowledge of experimental data at $p = 0$.

The system consists of several modules. In this work, the module responsible to obtain the pressures derivatives of anion and cation radii is described in detail through a variety of examples.

*Work partially supported by National and State Research Council — CNPq and FAPESP, Brasil
†Work partially supported by CONICET, Argentina

Contents

List of Figures

List of Tables

1 Introduction

In order to understand ionic crystals it is of value to introduce the concept of ionic properties transferable among families of crystals. In particular, it is important to obtain reliable values of polarizabilities of ions, they occur in the theories of refractive index, dielectric constant and optical properties of ionic solids.

The net ionic polarizability obtained by the standard Clausius-Mossotti treatment can be represented, to a first approximation, as the sum of individual ionic polarizabilities transferable among crystals [16]. Studies by *ab initio* methods [8, 9, 10] allow the understanding of effective polarizabilities of ions, their evaluation and the physical factors that affect them [11]. It is possible to determine cation polarizabilities that are practically invariant among crystals, but anion polarizabilities depend on environment [14].

Therefore the concept of ionic polarizability invariant for different crystals is not valid for anions. Theoretical approaches [17] have introduced an ionic polarizability *function* that depends on the interionic separation. One may assume [5] that a single polarizability function describes the polarizability of an ion in different crystals or in a given crystal under different pressures, being transferable among crystals and structures.

Previous work [1, 2, 3, 4] has studied the pressure dependence of the high and low-frequency dielectric constant by means of models that take account of the deformation of the ions, such as the shell model [7, 13]. The empirical treatment of Dick and Overhauser [5], which has the possibility of being extended to more complex crystals, introduces the evaluation of the pressure dependence of the anion polarizability and will be described in the following sections.

For this purpose a system — named Polariz — has been developed in Maple V. The system also gives a way of obtaining pressure derivatives of dielectric constants for those crystals for which only experimental data at $p = 0$ is available.

The complete system consists of several modules. In this work, the module responsible to obtain the pressures derivatives of anion and cation radii — named PDac — is described in detail through a variety of examples.

This work is organized as follows: in Section 2 the theory and method used are described. In Section 3 the arquitecture, as well as the four modules that constitute the Polariz system are briefly discused. In Section 4 the implementation of the PDac module is described in detail. In order to clarify the notation as well as the corresponding data structure used by this module, two different cases are studied to illustrate. In Section 5 the Maple code that implements the PDac module is explained in detail and we conclude in Section 6.

2 Theory and Method

Using the Clausius-Mossoti relation the anion polarizability in an ionic crystal can be evaluated from experimental data of the high frequency relative permittivity ε_∞ and the interionic separation R. Cation polarizabilities are assumed invariant among different crystals and its values are taken from the literature; due to their insensitivity to crystal environment their pressure derivatives may be considered small and positive. We have found [5] that an adequate analytical fit to the variation of the polarizability of *a given anion*, as R varies by exchanging the cation is given by

$$\alpha_- = a + \frac{b}{R^2}$$

where a and b are fitted from α_- and R values (for different cations and a fixed anion); it even considers changes in crystal structure. On the other hand one can plot α_- *for a fixed cation* as R changes by exchanging anions. We obtain the following adequate analytical fit

$$\alpha_- = a'R^2 + b'R + c'$$

where a', b', c' are fitted from α_- (for different anions and a fixed cation) and R data.

We have therefore confirmed that α_- of a given anion varies smoothly with R for different cations and that α_- of different anions varies smoothly with R for a given cation. Pressure effects can therefore be studied if one postulates [5]

$$\alpha_- = \alpha_-(r_-, r_+)$$

where r_- and r_+ are the anion and cation radii respectively. Considering $R = r_+ + r_-$, one also postulates

$$\left.\frac{\partial \alpha}{\partial r_-}\right)_{r_+} \approx \left.\frac{\partial \alpha}{\partial R}\right)_{fixed\ cation} \tag{1}$$

$$\left.\frac{\partial \alpha}{\partial r_+}\right)_{r_-} \approx \left.\frac{\partial \alpha}{\partial R}\right)_{fixed\ anion} \tag{2}$$

With these values one can therefore calculate

$$\frac{d\alpha_-}{dp} \approx \left.\frac{\partial \alpha_-}{\partial r_-}\right)_{r_+} \frac{dr_-}{dp} + \left.\frac{\partial \alpha_-}{\partial r_+}\right)_{r_-} \frac{dr_+}{dp} \tag{3}$$

where $\frac{dr_-}{dp}$ and $\frac{dr_+}{dp}$ are evaluated by the module PDac as described in section 4.1. First of all, an overview of the complete Polariz system is given in Section 3.

3 Overview of the Polariz System

Figure 1 shows the arquitecture of the system which consists of the following four modules

1. **CMCation** responsible for the evaluation of equation 1
2. **CMAnion** responsible for the evaluation of equation 2
3. **PDac** responsible for the evaluation of $\frac{dr_-}{dp}$ and $\frac{dr_+}{dp}$
4. **PDcalc** responsible for the evaluation of equation 3 combining the outputs of the other three modules

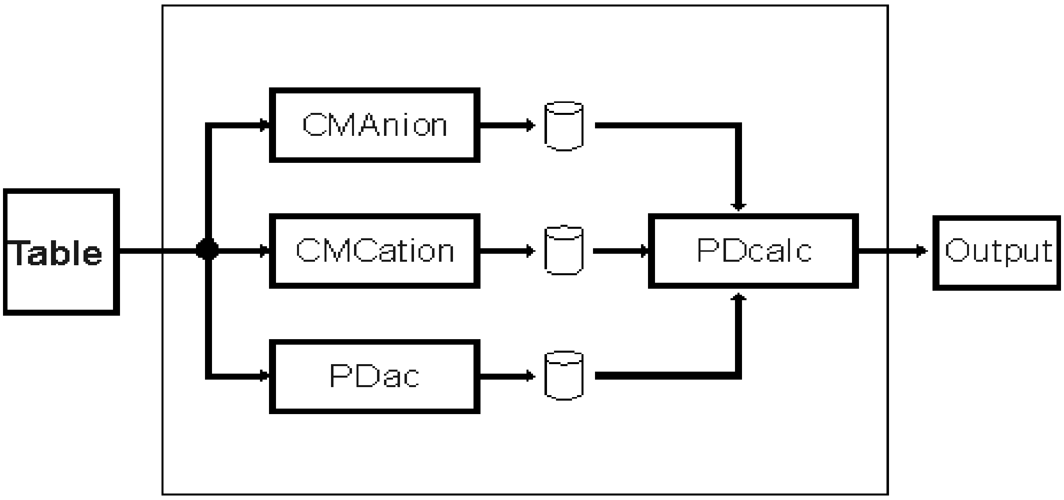

Figure 1: Polariz System Arquitecture

In [15] modules CMCation and CMAnion are described in detail. Both modules, as well as the PDac module, use the same experimental data. Table 6 shows the experimental data used in this work which have been taken from the experimental data at Low Temperatures – LT – and Room Temperatures – RT – described in [12]. It should be noted that some data was not available at LT, in those cases the missing data* has been substituted by data at RT from the same source.

Maple V's extensive mathematical funcionality is easily accessed through its advanced worksheet-based graphical interface. The user interface for every one of the interactive modules of the system implemented is based on the concept of worksheets, making extensive use of region groups to execute logical units. Maple has a wide range of commands for generating both two- and three- dimensional plots. Two- dimensional plots are extensively used by each of the modules of the Polariz system.

Maple has many available data structures. In this system the experimental input data in Table 6 is defined using Maple V table data structure as explained in Section 5.1.

4 Module PDac Description

In this section the implementation of the module PDac, responsible for the calculation of $\frac{\partial \alpha_-}{\partial p}$, is described in details. To do that, and in order to clearly undestand the notation used as well as the corresponding data structure used by this module, two different cases will be ilustrated.

4.1 Evaluation of $\frac{\partial \alpha_-}{\partial p}$

The general notation used by this module is shown in Table 1, assuming we have n_c different cations **c** and n_a different anions **a** with $md = n_c + n_a$

Cations	$\mathbf{c}_1$	$\mathbf{c}_2$	$\mathbf{c}_3$	$\dots$	$\mathbf{c}_{n_c}$
	r_1	r_2	r_3	$\dots$	r_{n_c}
Anions	$\mathbf{a}_{n_c+1}$	$\mathbf{a}_{n_c+2}$	$\mathbf{a}_{n_c+3}$	$\dots$	$\mathbf{a}_{n_c+n_a}$
	r_{n_c+1}	r_{n_c+2}	r_{n_c+3}	$\dots$	$r_{n_c+n_a}$

Table 1: General Notation

the system implements the linear programming problem of the form

$$maximize\ Z = \sum_{k=1}^{md} r_k$$

subject to the restrictions

$$r_i + r_j \leq R_{ij}\ for \begin{cases} i\ is\ any\ value\ in\ \{1..n_c\} \\ j\ is\ any\ value\ in\ \{n_c + 1..n_c + n_a\} \end{cases}$$

all nonnegative. All instances of i and j values are user defined through the corresponding companion matrix, as explained latter on. To solve this problem the user can also fix one or more r_k at zero pressure (p_0) and iteratively solve

$$R'_{ij} = R^0_{ij} - \frac{\chi_{ij} R^0_{ij}}{3}(p' - p_0) \tag{4}$$

such that it must verify for $p'' > p' > p_0$

$$\begin{array}{llll} R''_{ij} & < R'_{ij} & < R_{ij} & all\ i\ and\ j\ defined\ by\ the\ user \\ r''_k & < r'_k & < r_k & k = 1..md \end{array}$$

From p_0 to $p' = 0.001$ GPa a step of 10^{-4} is used and from $p' = 0.001$ GPa to $p'' = 0.01$ GPa a step of 10^{-3} (10 times higher) is used, making a total of 20 iterations. This module allows for a wide range of experimental settings such as the inclusion of fixed values of one or more $r_k, k = 1..md$ at different pressures, where every r_k may be evaluated at each value of p and may then be fixed in each maximization procedure.

To obtain the restriction that for $p'' > p' > p_0$ then $r''_k < r'_k < r_k$, aditional constraints may be used in each iteration $(n + 1)$ of the form

$$r_k^{(n+1)} \leq r_k^{(n)} - \Delta r_k \Delta p$$

where Δr_k can be fixed by the user. The value of the objective function Z being minimized

$$Z^{(n)} = \sum_{k=1}^{md} r_k^{(n)}$$

in each of the $n = 20$ iterations is also plotted. Also, for each one of the total number of equations of the system the error $e_{ij}^{(n)}$ in every iteration n is calculated and plotted, for user's defined i and j

$$e_{ij}^{(n)} = r_i^{(n)} + r_j^{(n)} - R_{ij}^{(n)}$$

as well as

$$e^{(n)} = \sqrt{\sum_{i,j} (e_{ij}^{(n)})^2}$$

Finally, $r_k = f(p)$ for $k = 1..md$ is plotted for the 20 calculated pressures and using leastsquares, a linear function is interpolated for the following two cases: all 20 pressures and only the first 10 pressures. From that the module calculates its output $\frac{\partial r}{\partial p}$ for both cases. Those values are then saved for their final combination, executed by the module PDgeral, with the values calculated by the modules CMCation (see eq. 1) and CMAnion (see eq. 2) to evaluate $\frac{\partial \alpha_-}{\partial p}$ (see eq. 3 pg. 2).

4.1.1 Case Study 1

This case study corresponds to the cations and anions in Table 2.

Cations	Li^+	Na^+	K^+	Rb^+
	r_1	r_2	r_3	r_4
Anions	F^-	Cl^-	Br^-	I^-
	r_5	r_6	r_7	r_8

Table 2: Case1 – Cations and Anions Considered

Thus $n_c = 4$, $n_a = 4$ and $md = 8$ with

$$maximize\ Z = \sum_{k=1}^{8} r_k$$

subject to the restrictions

$$r_i + r_j \leq R_{ij}\ for \begin{cases} i = 1..4 \\ j = 5..8 \end{cases}$$

all nonnegative making 16 inequalities. It is very important to understand the correspondence among the salts being studied and the R_{ij} notation. In this case the correspondence shown in Table 3 is used. In matrix form this can be written as

$$\mathbf{Ar} \leq \mathbf{R} \tag{5}$$

where

LiF	LiCl	LiBr	LiI	NaF	NaCl	NaBr	NaI
$R_{1,5}$	$R_{1,6}$	$R_{1,7}$	$R_{1,8}$	$R_{2,5}$	$R_{2,6}$	$R_{2,7}$	$R_{2,8}$
KF	KCl	KBr	KI	RbF	RbCl	RbBr	RbI
$R_{3,5}$	$R_{3,6}$	$R_{3,7}$	$R_{3,8}$	$R_{4,5}$	$R_{4,6}$	$R_{4,7}$	$R_{4,8}$

Table 3: Case1 – Correspondence Among Salts

$$\mathbf{A} = \begin{pmatrix} 1&0&0&0&1&0&0&0\\ 1&0&0&0&0&1&0&0\\ 1&0&0&0&0&0&1&0\\ 1&0&0&0&0&0&0&1\\ 0&1&0&0&1&0&0&0\\ 0&1&0&0&0&1&0&0\\ 0&1&0&0&0&0&1&0\\ 0&1&0&0&0&0&0&1\\ 0&0&1&0&1&0&0&0\\ 0&0&1&0&0&1&0&0\\ 0&0&1&0&0&0&1&0\\ 0&0&1&0&0&0&0&1\\ 0&0&0&1&1&0&0&0\\ 0&0&0&1&0&1&0&0\\ 0&0&0&1&0&0&1&0\\ 0&0&0&1&0&0&0&1 \end{pmatrix} \quad \mathbf{r} = \begin{pmatrix} r_1\\ r_2\\ r_3\\ r_4\\ r_5\\ r_6\\ r_7\\ r_8 \end{pmatrix} \quad \mathbf{R} = \begin{pmatrix} R_{1,5}\\ R_{1,6}\\ R_{1,7}\\ R_{1,8}\\ R_{2,5}\\ R_{2,6}\\ R_{2,7}\\ R_{2,8}\\ R_{3,5}\\ R_{3,6}\\ R_{3,7}\\ R_{3,8}\\ R_{4,5}\\ R_{4,6}\\ R_{4,7}\\ R_{4,8} \end{pmatrix} \tag{6}$$

In this case we decided to fix the value of r_8 at p_0 as $\frac{R_{1,8}^{(0)}}{\sqrt{2}}$ (assuming that both I^- outermost boundaries are in contact along the diagonal, see Figure 2) and in each maximization procedure as

$$r_8' = \frac{R_{1,8}^0 - \frac{\chi_{18} R_{1,8}^0}{3}(p' - p_0)}{\sqrt{2}}$$

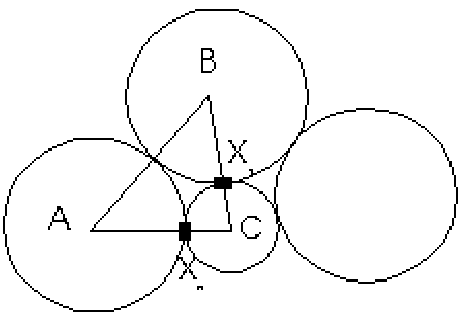

Figure 2: r_8 at p_0 assumed as $\frac{R_{1,8}^{(0)}}{\sqrt{2}}$

To obtain the restriction that for $p'' > p' > p_0$ then $r_k'' < r_k' < r_k$ in each iteration $(n+1)$ of the form

$$r_i^{(n+1)} \leq r_i^{(n)} - \Delta r_i \Delta p$$

we experimented with different values of Δr_i such as $\Delta r_i = constant$, Δr_i proportional to the ionic radii and others. Results presented in this work for both cases 1 and 2, correspond to the first option with $\Delta r_i = 1$.

4.1.2 Case Study 2

The second case study corresponds to the cations and anions in Table 4

Cations	Li^+	Na^+	K^+	Rb^+	Ca^+	Sr^+	Ba^+	Mg^+
	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
Anions	F^-	Cl^-	Br^-	I^-	O^-			
	r_9	r_{10}	r_{11}	r_{12}	r_{13}			

Table 4: Case2 – Cations and Anions Considered

where $n_c = 8$, $n_a = 5$ and $md = 13$ with

$$maximize\ Z = \sum_{k=1}^{13} r_k$$

subject to the restrictions $r_i + r_j \leq R_{ij}$ for several (i, j) pairs as explained afterwards. In this second case, we decided to study the same salts than in case 1 as well as others salts. In this case the correspondence shown in Table 5 is used.

LiF	LiCl	LiBr	LiI	NaF	NaCl	NaBr	NaI
$R_{1,9}$	$R_{1,10}$	$R_{1,11}$	$R_{1,12}$	$R_{2,9}$	$R_{2,10}$	$R_{2,11}$	$R_{2,12}$
KF	KCl	KBr	KI	RbF	RbCl	RbBr	RbI
$R_{3,9}$	$R_{3,10}$	$R_{3,11}$	$R_{3,12}$	$R_{4,9}$	$R_{4,10}$	$R_{4,11}$	$R_{4,12}$
CaF	CaO	SrF	SrO	BaF	Ba O	MgO	
$R_{5,9}$	$R_{5,13}$	$R_{6,9}$	$R_{6,13}$	$R_{7,9}$	$R_{7,13}$	$R_{8,13}$	

Table 5: Case 2 – Correspondence Among Salts

where eq. 5 pg. 5 is now given by

$$
\mathbf{A} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\quad
\mathbf{r} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \\ r_{11} \\ r_{12} \\ r_{13} \end{pmatrix}
\quad
\mathbf{R} = \begin{pmatrix} R_{1,9} \\ R_{1,10} \\ R_{1,11} \\ R_{1,12} \\ R_{2,9} \\ R_{2,10} \\ R_{2,11} \\ R_{2,12} \\ R_{3,9} \\ R_{3,10} \\ R_{3,11} \\ R_{3,12} \\ R_{4,9} \\ R_{4,10} \\ R_{4,11} \\ R_{4,12} \\ R_{5,9} \\ R_{5,13} \\ R_{6,9} \\ R_{6,13} \\ R_{7,9} \\ R_{7,13} \\ R_{8,13} \end{pmatrix}
\tag{7}
$$

In this case we decided to fix the following values at p_0

$$
r_9 = \frac{R^{(0)}_{1,9}}{\sqrt{2}} \qquad r_{10} = \frac{R^{(0)}_{1,10}}{\sqrt{2}} \qquad r_{11} = \frac{R^{(0)}_{1,11}}{\sqrt{2}} \qquad r_{12} = \frac{R^{(0)}_{1,12}}{\sqrt{2}} \tag{8}
$$

Also, at $p = 0$ the value of r_{13} was set up as being $r_{13} \geq 0.1$, since the maximum value of the objetive function Z without this constrain has $r_{13} = 0$ as solution at $p = 0$. In each maximization procedure the following values were fixed as

$$
\begin{aligned}
r'_9 &= \frac{R^0_{1,9} - \frac{\chi_{1,9} R^0_{1,9}}{3}(p' - p_0)}{\sqrt{2}} \qquad & r'_{10} &= \frac{R^0_{1,10} - \frac{\chi_{1,10} R^0_{1,10}}{3}(p' - p_0)}{\sqrt{2}} \\
r'_{11} &= \frac{R^0_{1,11} - \frac{\chi_{1,11} R^0_{1,11}}{3}(p' - p_0)}{\sqrt{2}} \qquad & r'_{12} &= \frac{R^0_{1,12} - \frac{\chi_{1,12} R^0_{1,12}}{3}(p' - p_0)}{\sqrt{2}}
\end{aligned}
\tag{9}
$$

As expalined before, we used in each iteration the same constraints than in case 1 to obtain the restriction that for $p'' > p' > p_0$ then $r''_k < r'_k < r_k$ for $k = 1..13$.

5 Module PDac Implementation

In this section the Maple code that implements the module PCac is explained in detail. Both case study 1 and 2 — Section 4.1.1 and 4.1.2 — will be used to exemplify the user's definition needed in each case. First of all the data structure used by the Polariz system to represent the experimental input data is discussed.

5.1 Data Structure

Maple has many available data structures. In this system the experimental input data in Table 6 pg. 27 is defined using Maple V table data structure. A table is an extension of the concept of the array data structure. The difference between an array and a table is that a table can have anything for index, not just integers. For example, a table named `fcc_LT` represents the low temperature data corresponding to Table 6 such that each index is a name. For each cation index `Li`, `Na`, `K`, `Rb`, `Cs`, etc... each entry is a number representing the value of the corresponding α_+. For example, `fcc_LT[Li]` is 0.030. For indexes related to the salt's chemical symbols each entry is a list representing the row of the corresponding table. For example `fcc_LT[LiF]` is the list `[0.89, 1.996, 8.50, 1.93, 1.43]`. Particular elements of a list can be extracted. For example, `fcc_LT[LiF][1]` refers to `0.89`, `fcc_LT[LiF][2]` to `1.996` and so on. We consider that this rich data structure is more efficient, easier to understand as well as to increment and alter when having new and more data available. A partial Maple print out of table `fcc_LT` is shown bellow.

```
> fcc_LT :=table([Li=0.030,
>                          LiF=[0.89, 1.996, 8.50, 1.93, 1.43, -1, -1],
>                          LiCl=[2.94,2.539, 10.83, 2.79, 3.20, -1, -1],
>                          LiBr=[4.13, 2.713, 11.95, 3.22, 4.20, -1, -1],
>                          LiI= [6.19, 2.951, 15.28, 3.89, 5.7803, -1, -1],
>                 ..........
>                 Na=0.147,
>                          NaF=[1.02, 2.295, 4.73, 1.75, 1.99, -1, -1],
>                          NaCl=[3.14, 2.789,5.43, 2.35, 3.98, -1, -1],
>                          NaBr=[4.28, 2.954, 5.78, 2.64, 4.70, -1, -1],
>                          NaI=[6.35, 3.194, 6.62, 3.08, 6.45, -1, -1],
```

5.2 Data Used by PDac

Although the PDac module has access to the complete table, it should be observed that it only uses the data related to the values of R^0_{ij} and χ_{ij} for the salts treated. For example, for table `fcc_LT` they correspond to the entries

$$\begin{aligned} R^0_{ij} &= \texttt{fcc\_LT[}\ <\textit{salt name}>\ \texttt{][2]} \\ \chi_{ij} &= \texttt{fcc\_LT[}\ <\textit{salt name}>\ \texttt{][5]} \end{aligned}$$

where the mapping $<\textit{salt name}>\Leftrightarrow (i,j)$ is defined by the user as explained in Section 4 pg. 3.

5.3 Worksheet Description

Initially the module is restarted and the necessary Maple packages are loaded by the command `with`

```
> restart:with(linalg):with(simplex):with(stats): with(student):
```

Next the experimental input data — Table 6 pg. 27 — named `fcc_LT` is read in and stored in `fcc_TA` and the variable `temperature` should be set accordingly to indicate this fact. The value of the variable `temperature` is printed in several of the two- dimensional plots.

```
> read `al_fcc_LT.m`: fcc_TA:=fcc_LT: temperature:=`LT `:
```

Next, the user needs to define the specific problem. To exemplify, it follows the necessary user's definitons for case study 1 and 2.

5.3.1 User's Definitions for Case Study 1

In this case — see Section 4.1.1 pg. 5 — $md = 8$ and the number of inequalities is 16. Optionally the user can define the number of digits carried in floats (default is 10) through the `Digits` environment variable. This is defined as follows

```
> MD:=8: MDEQ:=16: Digits:=5:
```

Next, the user has to define the corresponding matrix **A** and vector **R** in eq. 6 pg. 6. The definition of matrix **A** can be done as a straightforward line by line definition as shown bellow

```
> A:=array([[1,0,0,0,1,0,0,0],[1,0,0,0,0,1,0,0],[1,0,0,0,0,0,1,0],
> [1,0,0,0,0,0,0,1],[0,1,0,0,1,0,0,0],[0,1,0,0,0,1,0,0],
> [0,1,0,0,0,0,1,0],[0,1,0,0,0,0,0,1],[0,0,1,0,1,0,0,0],
> [0,0,1,0,0,1,0,0],[0,0,1,0,0,0,1,0],[0,0,1,0,0,0,0,1],
> [0,0,0,1,1,0,0,0],[0,0,0,1,0,1,0,0],[0,0,0,1,0,0,1,0],
> [0,0,0,1,0,0,0,1]]):
```

At any time the user can display any values using the Maple function `print` pretty-printing of expressions, as shown bellow

```
> print(A);
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Next, the user needs to define the function `newp` to calculate vector $\mathbf{R}$ in eq. 6 pg. 6 for this particular problem. In this case is very important to observe the mapping defined by matrix $\mathbf{A}$. It follows the Maple definition of this function where the first line correspond to the value of R_{15}, the second to R_{16} and so on.

It should be noted that the definition of this function — see eq. 4 pg. 4 — is easily achieved with the table data structure defined keeping in mind the mapping used, i.e. R_{15} corresponds to LiF, R_{16} to LiCl and so on, as shown in Table 3 pg. 6.

```
> newp:=x->evalf([fcc_TA[LiF][2]*(1-fcc_TA[LiF][5]*x/3),
> fcc_TA[LiCl][2]*(1-fcc_TA[LiCl][5]*x/3),
> fcc_TA[LiBr][2]*(1-fcc_TA[LiBr][5]*x/3),
> fcc_TA[LiI][2]*(1-fcc_TA[LiI][5]*x/3),
> fcc_TA[NaF][2]*(1-fcc_TA[NaF][5]*x/3),
> fcc_TA[NaCl][2]*(1-fcc_TA[NaCl][5]*x/3),
> fcc_TA[NaBr][2]*(1-fcc_TA[NaBr][5]*x/3),
> fcc_TA[NaI][2]*(1-fcc_TA[NaI][5]*x/3),
> fcc_TA[KF][2]*(1-fcc_TA[KF][5]*x/3),
> fcc_TA[KCl][2]*(1-fcc_TA[KCl][5]*x/3),
```

```
> fcc_TA[KBr][2]*(1-fcc_TA[KBr][5]*x/3),
> fcc_TA[KI][2]*(1-fcc_TA[KI][5]*x/3),
> fcc_TA[RbF][2]*(1-fcc_TA[RbF][5]*x/3),
> fcc_TA[RbCl][2]*(1-fcc_TA[RbCl][5]*x/3),
> fcc_TA[RbBr][2]*(1-fcc_TA[RbBr][5]*x/3),
> fcc_TA[RbI][2]*(1-fcc_TA[RbI][5]*x/3)]):
```

After executing this region the user can eventually verify its definition by printing several function values as shown bellow

```
> print(newp(0));
```

$$[1.996, 2.539, 2.713, 2.951, 2.295, 2.789, 2.954, 3.194, 2.648, 3.116, 3.262, 3.489, 2.789, 3.259, 3.410, 3.628]$$

```
> print(newp(0.01));
```

$$[1.98649, 2.51192, 2.67502, 2.89414, 2.27978, 2.75200, 2.90772, 3.12533, 2.62064, 3.06137, 3.19491, 3.39887, 2.75358, 3.18948, 3.32259, 3.51336]$$

5.3.2 User's Definitions for Case Study 2

In this case — see Section 4.1.2 pg. 7 — $md = 13$ and the number of inequalities is 23, then the user should define

```
> MD:=13: MDEQ:=23: Digits:=5:
```

The corresponding matrix **A** in eq. 7 pg. 8 can be defined as follows

```
> A:=array([
> [1,0,0,0,0,0,0,0,1,0,0,0,0],[1,0,0,0,0,0,0,0,0,1,0,0,0],
> [1,0,0,0,0,0,0,0,0,0,1,0,0],[1,0,0,0,0,0,0,0,0,0,0,1,0],
> [0,1,0,0,0,0,0,0,1,0,0,0,0],[0,1,0,0,0,0,0,0,0,1,0,0,0],
> [0,1,0,0,0,0,0,0,0,0,1,0,0],[0,1,0,0,0,0,0,0,0,0,0,1,0],
> [0,0,1,0,0,0,0,0,1,0,0,0,0],[0,0,1,0,0,0,0,0,0,1,0,0,0],
> [0,0,1,0,0,0,0,0,0,0,1,0,0],[0,0,1,0,0,0,0,0,0,0,0,1,0],
> [0,0,0,1,0,0,0,0,1,0,0,0,0],[0,0,0,1,0,0,0,0,0,1,0,0,0],
> [0,0,0,1,0,0,0,0,0,0,1,0,0],[0,0,0,1,0,0,0,0,0,0,0,1,0],
> [0,0,0,0,1,0,0,0,1,0,0,0,0],[0,0,0,0,1,0,0,0,0,0,0,0,1],
> [0,0,0,0,0,1,0,0,1,0,0,0,0],[0,0,0,0,0,1,0,0,0,0,0,0,1],
> [0,0,0,0,0,0,1,0,1,0,0,0,0],[0,0,0,0,0,0,1,0,0,0,0,0,1],
> [0,0,0,0,0,0,0,1,0,0,0,0,1]]):
```

Next, the user needs to define the function `newp` to calculate vector **R** in eq. 7 pg. 8 for this particular problem. It follows the Maple definition of this function where the first line correspond to the value of $R_{1,9}$ (LiF), the second to $R_{1,10}$ (LiCl) and so on up to $R_{8,13}$ (MgO) as shown in Table 5 pg. 7.

```
> newp:=x->evalf([fcc_TA[LiF][2]*(1-fcc_TA[LiF][5]*x/3),
> fcc_TA[LiCl][2]*(1-fcc_TA[LiCl][5]*x/3),
> fcc_TA[LiBr][2]*(1-fcc_TA[LiBr][5]*x/3),
> fcc_TA[LiI][2]*(1-fcc_TA[LiI][5]*x/3),
> fcc_TA[NaF][2]*(1-fcc_TA[NaF][5]*x/3),
> fcc_TA[NaCl][2]*(1-fcc_TA[NaCl][5]*x/3),
> fcc_TA[NaBr][2]*(1-fcc_TA[NaBr][5]*x/3),
> fcc_TA[NaI][2]*(1-fcc_TA[NaI][5]*x/3),
> fcc_TA[KF][2]*(1-fcc_TA[KF][5]*x/3),
> fcc_TA[KCl][2]*(1-fcc_TA[KCl][5]*x/3),
> fcc_TA[KBr][2]*(1-fcc_TA[KBr][5]*x/3),
> fcc_TA[KI][2]*(1-fcc_TA[KI][5]*x/3),
> fcc_TA[RbF][2]*(1-fcc_TA[RbF][5]*x/3),
> fcc_TA[RbCl][2]*(1-fcc_TA[RbCl][5]*x/3),
> fcc_TA[RbBr][2]*(1-fcc_TA[RbBr][5]*x/3),
> fcc_TA[RbI][2]*(1-fcc_TA[RbI][5]*x/3),
> fcc_TA[CaF][2]*(1-fcc_TA[CaF][5]*x/3),
> fcc_TA[CaO][2]*(1-fcc_TA[CaO][5]*x/3),
> fcc_TA[SrF][2]*(1-fcc_TA[SrF][5]*x/3),
> fcc_TA[SrO][2]*(1-fcc_TA[SrO][5]*x/3),
> fcc_TA[BaF][2]*(1-fcc_TA[BaF][5]*x/3),
> fcc_TA[BaO][2]*(1-fcc_TA[BaO][5]*x/3),
>  fcc_TA[MgO][2]*(1-fcc_TA[MgO][5]*x/3)
> ]):
```

After having executed this region, any function values can be printed by the user as shown bellow

```
> print(newp(0));
```

$$[1.996, 2.539, 2.713, 2.951, 2.295, 2.789, 2.954, 3.194, 2.648, 3.116, 3.262, 3.489, 2.789, 3.259, 3.410, 3.628, 2.30, 2.3995, 2.51, 2.5698, 2.69, 2.8129, 2.1066]$$

```
> print(newp(0.01));
```

$$[1.9865, 2.5119, 2.6750, 2.8941, 2.2798, 2.7520, 2.9077, 3.1253, 2.6206, 3.0614, 3.1949, 3.3989, 2.7536, 3.1895, 3.3226, 3.5134, 2.2911, 2.3923, 2.4980, 2.5602, 2.6743, 2.8005, 2.1023]$$

In this case, as well as the previous case, we fixed $\Delta r_i = 1$ to obtain the restriction that for $p'' > p' > p_0$ then $r_k'' < r_k' < r_k$, in each iteration $(n+1)$ using the additional constraints $r_i^{(n+1)} \leq r_i^{(n)} - \Delta r_i \Delta p$. The values of Δr_i are defined as follows

```
> delta_r1:= 1 : delta_r2:= 1 : delta_r3:= 1 : delta_r4:= 1:
> delta_r5:= 1 : delta_r6:= 1 : delta_r7:= 1 : delta_r8:= 1:
> delta_r9:= 1 : delta_r10:=1: delta_r11:=1: delta_r12:=1: delta_r13:=1:
```

This ends the user's necessary definitions to run his particular problem using this module, except for the initial and other constraints to be considered in each iteration of the maximization procedure, described in detail in next section.

5.3.3 General Description

In the following description user's definitions for case study 2, Section 5.3.2 pg. 12, are considered.

The value of the variable `delta_initial` is the initial value of Δp used by the constraints $r_i^{(n+1)} \leq r_i^{(n)} - \Delta r_i \Delta p$ from p_0 to $p' = 0.001$ GPa where a step of 10^{-4} is used. From $p' = 0.001$ GPa to $p' = 0.0001$ Gpa a step of 10^{-3} (10 times higher) is used and Δp (represented by the variable `delta`) is also set 10 times higher. Eventually, the value of the variable `delta_initial` can be changed if necessary. Still, we have experimented with several values and found that this initial value is quite appropriate.

```
> delta_initial:=0.00001:
```

Next, several data variables used by this module are defined. Only the ones that will hold data which is interesting for the user are described

- `val_r` is a vector {1..MD} that holds, in each of the 20 iterations the values of r_k from $k = 1..md$ that maximizes the objective function Z
- `Calc_pressures` is a vector {1..20} that holds the pressure values used in each of the 20 iterations.
- `Calc_rs` is a matrix {1..20,1..MD} where every line i=1..20 holds the corresponding values of `val_r[j]` j=1..MD calculated at pressure `Calc_pressure[i]`
- `Calc_error_linear` is matrix {1..20,1..MD} where every line $n = 1..20$ holds the linear error $e_{ij}^{(n)} = r_i^{(n)} + r_j^{(n)} - R_{ij}^{(n)}$ of every {1..MD} equation of the system
- `Calc_error_total` is a vector {1..20} such that `Calc_error_total[n]` holds the total error $e^{(n)} = \sqrt{\sum_{i,j} (e_{ij}^{(n)})^2}$ found at the n^{th} iteration using pressure `Calc_pressure[n]`

- `Calc_objf` is a vector $\{1..20\}$ such that `Calc_objf[n]` holds the value of the objective function Z being maximized $Z^{(n)} = \sum_{k=1}^{md} r_k^{(n)}$ in each of the $n = \{1..20\}$ iterations.

```
> c:=array([[seq(1,i=1..MD)]]):
> val_r:=vector(MD):
> Calc_pressures:=vector(20):
> Calc_rs:=array(1..20,1..MD):
> Calc_error_linear:=array(1..20,1..MDEQ):
> Calc_error_total:=vector(20):
> Calc_objf:=vector(20):
```

The next worksheet's region is responsible for the calculation in every of the 20 iterations. In this region the user has to set up any new constraints at $p = 0$ or other pressures.

```
> p:='p':
> for p from 0 to 19 do
> Z:='Z':  r:='r': b:='b':  i:='i': prod:='prod': result:='result':
> sols:='sols': constraints:='constraints': k:='k': tot_constr:='tot_constr':
> delta:='delta':more_constr:='more_constr':
> r:=vector(MD):
> delta:=delta_initial:
```

Up to here, the necessary variables have been cleaning of all prior definition and initialized. Next the variable `pressure` is set to the corresponding pressure value by the following two commands.

```
> if p<=10 then pressure:=p*0.0001 else  pressure:=(p-9)*0.001 fi:
> if p>10 then delta:=delta_initial*10 fi:
```

Vector `b` is set up to hold the values of vector $\mathbf{R}$ eq. 7 pg. 8 by the following command

```
> b:=vector(newp(pressure)):
```

A vector `Z`, such that the objective function $Z = \sum_{k=1}^{MD} r_k$ corresponds to `Z[1]`, as well as $\mathbf{Ar} \leq \mathbf{R}$ is defined by the following three commands

```
> Z:=multiply(c,r);
> prod:=multiply(A,r):
> constraints:={seq(prod[i] <=b[i], i=1..MDEQ)}:
```

The following command is responsible for setting additional constraints at $p = 0$ as well as fixing any other values in each maximization step. In this case, as explained in Section 4.1.2

pg. 7, it was decided to fix some values at $p = 0$ (see eq. 8) and some in each maximization procedure (see eq. 9) as well as $r_{13} \geq 0.1$

```
> if p=0 then more_constr:={ r[9] = evalf(fcc_TA[LiF][2]/sqrt(2)) ,
>                            r[10] = evalf(fcc_TA[LiCl][2]/sqrt(2)),
>                            r[11] = evalf(fcc_TA[LiBr][2]/sqrt(2)),
>                            r[12] = evalf(fcc_TA[LiI][2]/sqrt(2)),
>                            0.1 <= r[13]}
>        else more_constr:={
>        r[1] <= val_r[1] -delta_r1*delta, r[2] <= val_r[2] -delta_r2*delta,
>        r[3] <= val_r[3] -delta_r3*delta, r[4] <= val_r[4] -delta_r4*delta,
>        r[5] <= val_r[5] -delta_r5*delta, r[6] <= val_r[6] -delta_r6*delta,
>        r[7] <= val_r[7] -delta_r7*delta, r[8] <= val_r[8] -delta_r8*delta,
>        r[9] = evalf(fcc_TA[LiF][2]*(1-fcc_TA[LiF][5]*pressure/3)/sqrt(2)),
>        r[10] = evalf(fcc_TA[LiCl][2]*(1-fcc_TA[LiCl][5]*pressure/3)/sqrt(2)),
>        r[11] = evalf(fcc_TA[LiBr][2]*(1-fcc_TA[LiBr][5]*pressure/3)/sqrt(2)),
>        r[12] = evalf(fcc_TA[LiI][2]*(1-fcc_TA[LiI][5]*pressure/3)/sqrt(2)),
>        r[13] <= val_r[13] -delta_r13*delta} fi:
```

Finally, the total constraints used in each maximization is the union of both, the ones corresponding to the values of vector **R** (variable `constraints`) and these new ones (variable `more_constr`) aditionally defined by the user.

```
> tot_constr:=`union`(constraints,more_constr):
```

In order to illustrate, the actual value of `tot_constr` at $p = 0$ is show bellow

$$\begin{aligned}\Big\{&.1 \leq r_{13}, r_1 + r_9 \leq 1.996, r_2 + r_9 \leq 2.295, r_1 + r_{10} \leq 2.539,\\ &r_1 + r_{11} \leq 2.713, r_1 + r_{12} \leq 2.951, r_5 + r_{13} \leq 2.3995, r_3 + r_{10} \leq 3.116,\\ &r_2 + r_{10} \leq 2.789, r_2 + r_{11} \leq 2.954, r_2 + r_{12} \leq 3.194, r_3 + r_9 \leq 2.648,\\ &r_3 + r_{11} \leq 3.262, r_3 + r_{12} \leq 3.489, r_4 + r_9 \leq 2.789, r_4 + r_{10} \leq 3.259,\\ &r_4 + r_{11} \leq 3.410, r_4 + r_{12} \leq 3.628, r_5 + r_9 \leq 2.30, r_6 + r_9 \leq 2.51,\\ &r_6 + r_{13} \leq 2.5698, r_7 + r_9 \leq 2.69, r_7 + r_{13} \leq 2.8129, r_8 + r_{13} \leq 2.1066,\\ &r_{10} = 1.7953, r_9 = 1.4114, r_{12} = 2.0867, r_{11} = 1.9184\Big\}\end{aligned}$$

The next command calls function `feasible` that returns true if a feasible solution to the linear system `tot_constr` exists, and false otherwise. Non-negativity constraints on all the variables are indicated by the second argument, NONNEGATIVE[‡]

If the function returns a true value then the function `maximize` is called where the expression `Z[1]` is the linear objective function to be maximized subject to the linear constraints `tot_constr` NONNEGATIVE.

[‡]The Maple simplex package is a collection of routines for linear optimization using the simplex algorithm. The particular implementation of the simplex algorithm used is based on the initial chapters of *Linear Programming* by Chvatal, 1983, W.H. Freeman and Company, New York.

```
> if(feasible(tot_constr,NONNEGATIVE))
> then sols:=maximize(Z[1],tot_constr,NONNEGATIVE):
```

The equations returned by `maximize` can be substituted back into the objective function `Z[1]` to obtain the value of the objective function at the optimal solution. This value is stored in `Calc_objf[p+1]`, the solution values r_k from $k = 1..MD$ are assigned to the vector `val_r` as well as stored in `Calc_rs[p+1,k]` and the pressure value for this iteration is stored in `Calc_pressures[p+1]`

```
> result:=subs(sols,Z[1]):
> Calc_objf[p+1]:=result:
> k:='k':
> for k from 1 to MD do val_r[k]:=subs(sols,r[k]) od:
> k:='k':
> for k from 1 to MD do Calc_rs[p+1,k]:=val_r[k] od:
> Calc_pressures[p+1]:=pressure:
```

Next, the function `multiply(A,val_r)` calculates the matrix (a vector in this case) product `A` by `val_r`. The call `scalarmul(b, -1)` computes the vector which is the result of multiplying every entry in `b` by the scalar value `-1`. Finally the function `add` computes the vector sum of both ressults.

```
> error_linear:=add(multiply(A,val_r),scalarmul(b,-1)):
```

Next the `map` function applies the function $f(x) = x^2$ to each element of the vector `error_linear`. The call to `sum` computes the sum of the elements of `error_quadrado` over the given range and `sqrt` computes the square root

```
> error_quadrado:=map(x->x^2,error_linear):
> sum_error_quadrado:=sum(error_quadrado[m],m=1..MDEQ):
> error_total:=sqrt(sum_error_quadrado):
```

Finally, the linear and total error are stored for further use

```
> k:='k':
> Calc_error_total[p+1]:=error_total:
> for k from 1 to MDEQ do Calc_error_linear[p+1,k]:=error_linear[k] od:
> od:
```

To llustrate, the current values of variables `sols pressure` and `error_total` at the begining ($p = 0$) and end ($p = 0.01$) of this iteration are shown bellow

$$sols =, \{r_7 = 1.2786, r_5 = .88860, r_{10} = 1.7953, r_9 = 1.4114, r_{12} = 2.0867,$$
$$r_{11} = 1.9184, r_3 = 1.2366, r_1 = .58460, r_{13} = 1.4712, r_4 = 1.3776,$$
$$r_8 = .63540, r_2 = .88360, r_6 = 1.0986\}$$

$$pressure =, 0$$

$$error\_total =, .57590$$

$$sols =, \{r_{12} = 2.0465, r_3 = 1.2159, r_6 = 1.0933, r_7 = 1.2696, r_8 = .63440,$$
$$r_2 = .87510, r_{13} = 1.4669, r_4 = 1.3489, r_5 = .88640, r_1 = .58180,$$
$$r_{10} = 1.7762, r_{11} = 1.8915, r_9 = 1.4047\}$$

$$pressure =, .010$$

$$error\_total =, .51778$$

At this point, all necessary calculations have been performed and results stored. From now on the user has access to any of them.

Thus, using the data stored in the variables described at the begining of this section (pg. 14) several results can be plotted for inspection. For example, the following region produces the graph in Figure 3 pg. 19, showing the value of the objective function Z in every iteration

```
> with(plots):
> display(plot([seq([Calc_pressures[i], Calc_objf[i]],i=1..20)],style=POINT),
> labels=['pres.','Z'],title='Z=f(p)');
```

The total error $e^{(n)} = \sqrt{\sum_{i,j}(e_{ij}^{(n)})^2}$ in each iteration is plotted by the following region — see Figure 4 pg. 20.

```
> display(plot([seq([Calc_pressures[i], Calc_error_total[i]],i=1..20)],
> style=POINT),labels=['pres.','Error'],title='Total Error=f(p)');
```

The next region prepares, in each entry of the variable `err_eq[k]` from `k=1..MDEQ`, the necessary data to plot the linear error in each iteration of every inequation in the system, i.e. $e_{ij}^{(n)} = r_i^{(n)} + r_j^{(n)} - R_{ij}^{(n)}$, in function of the corresponding pressure.

```
> k:='k':
> for k from 1 to MDEQ do
> err_eq[k]:=[seq([Calc_pressures[i], col(Calc_error_linear,k)[i]],i=1..20)];
> od:
```

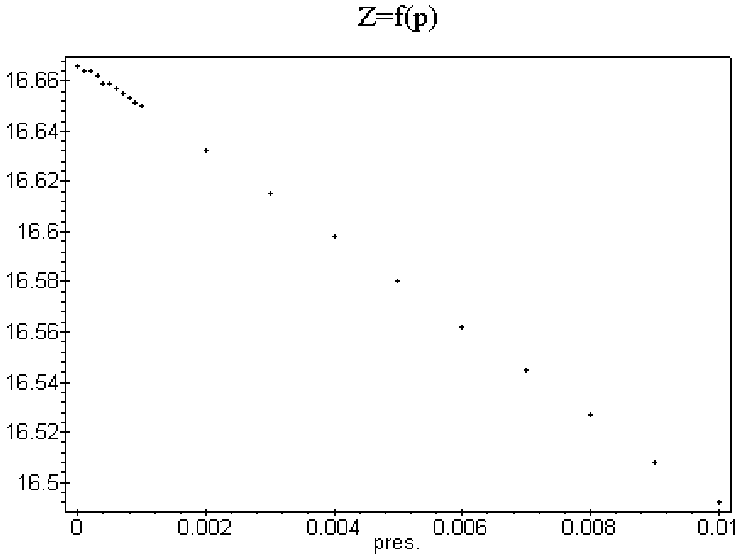

Figure 3: Plot of $Z = f(p), p = 0..0.01$

Afterwards, the user may plot any of those `1..MDEQ` (23 in this case study) errors. Again, the only thing to keep in mind is the mapping defined for the correspondent salts (see Table 5 pg. 7).

To illustrate Figure 5 pg. 21 shows the plot correspondent to the execution of the following region. In this case the erro $e_{1,9}^{(n)}$ is zero in all iterations (in this run were also zero in all iterations $e_{2,9}^{(n)}$, $e_{3,9}^{(n)}$, $e_{4,9}^{(n)}$, $e_{6,9}^{(n)}$, $e_{6,13}^{(n)}$, $e_{7,9}^{(n)}$).

```
> display(plot(err_eq[1],style=LINE),labels=['pres.','eq1'],
> title='Error eq1:  r1+r9');
```

The plot related to the execution of the following region is shown in Figure 6 pg. 21

```
> display(plot(err_eq[10],style=LINE),labels=['pres.','eq10'],
> title='Error eq10: r3+r10');
```

The plot related to the execution of the following region is shown in Figure 7 pg. 22

```
> display(plot(err_eq[17],style=LINE),labels=['pres.','eq17'],
> title='Error eq17: r5+r9');
```

The plot related to the execution of the following region is shown in Figure 8 pg. 22

```
> display(plot(err_eq[23],style=LINE),labels=['pres.','eq23'],
> title='Error eq23: r8+r13');
```

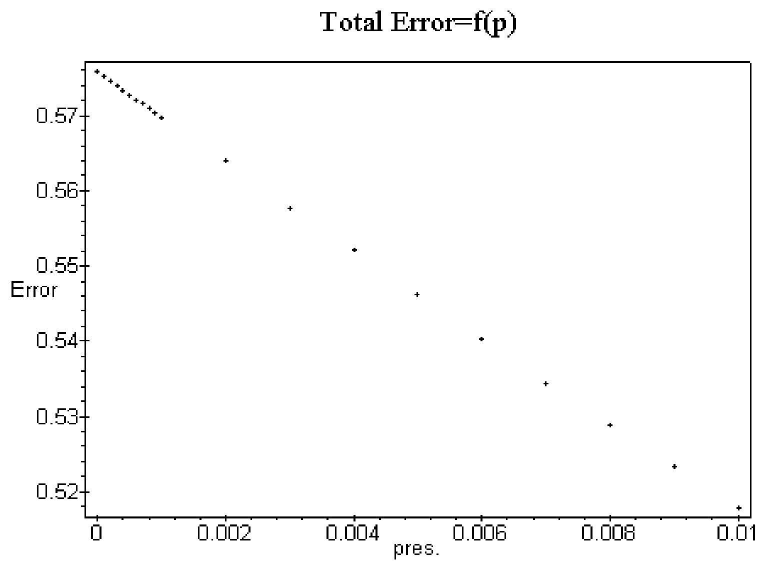

Figure 4: Plot of $e^{(n)} = \sqrt{\sum_{i,j}(e_{ij}^{(n)})^2}$

In general, the methodology used to build a feasible linear programimg system is to add additional constaints step by step. Most of the time, there are several solutions which minimize Z. We have found that the analysis of those errors while studying the behavior of the system for different additional constraints is quite useful.

After having found the $r's$ values that satisfy the objective function Z the next region prepares, in each entry of the variable `r_plots[k]` from `k=1..MD`, the necessary data to plot, in function of p, the solutions values of r_k $k = 1..md$ in each of the 20 iterations.

```
> k:='k':
> for k from 1 to MD do
> r_plots[k]:=[seq([Calc_pressures[i], col(Calc_rs,k)[i]],i=1..20)] ;
> od:
```

Next, similar data is prepared in `r_values[k]` for the 20 iterations as well as `r_values_10[k]` for the first 10 iterations. This preparation is necessary due to the statistical list data structure needed to apply leastsquare.

```
> pressure_values:=[seq(Calc_pressures[i],i=1..20)]:
> pressure_values_10:=[seq(pressure_values[i],i=1..11)]:
> k:='k':
> for k from 1 to MD do
> r_values[k]:=[seq(col(Calc_rs,k)[i],i=1..20)];
> r_values_10[k]:=[seq(col(Calc_rs,k)[i],i=1..11)];
> od:
```

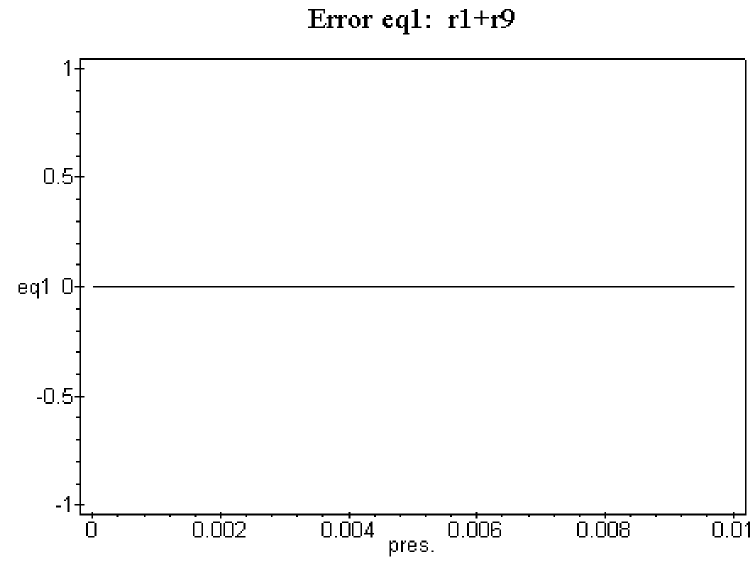

Figure 5: Plot of $e_{1,9}^{(n)} = r_1^{(n)} + r_9^{(n)} - R_{1,9}^{(n)}$

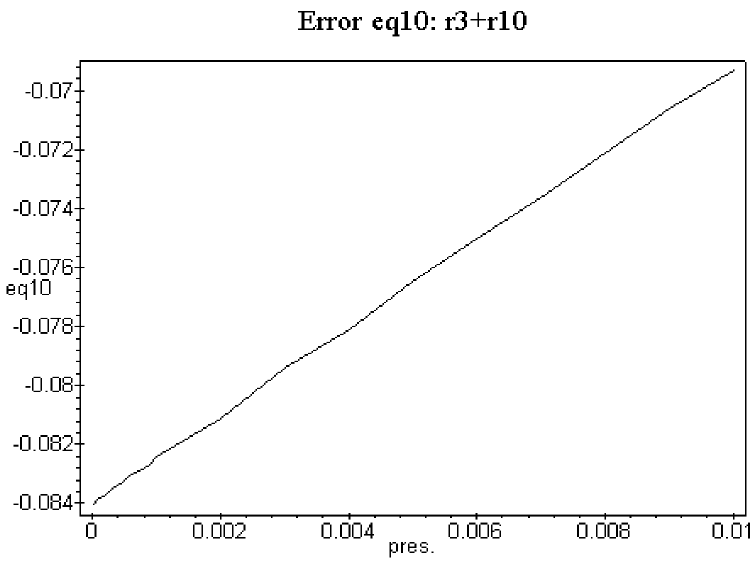

Figure 6: Plot of $e_{3,10}^{(n)} = r_3^{(n)} + r_{10}^{(n)} - R_{3,10}^{(n)}$

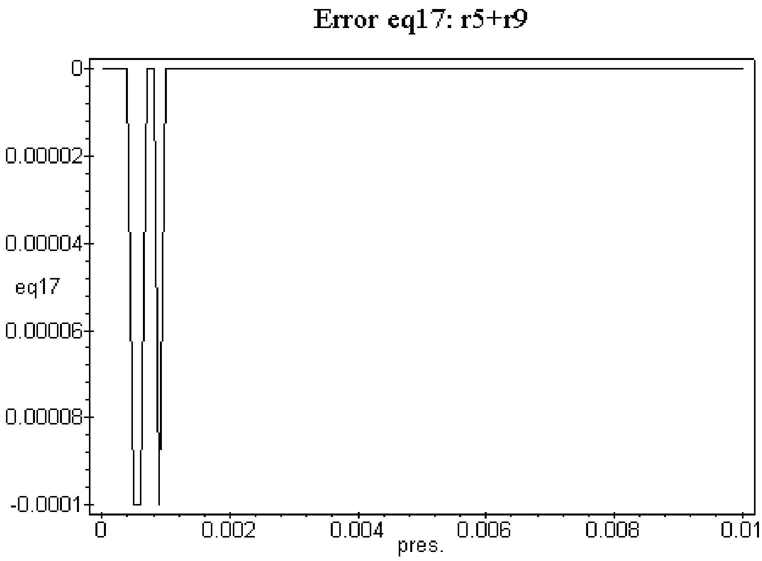

Figure 7: Plot of $e_{5,9}^{(n)} = r_5^{(n)} + r_9^{(n)} - R_{5,9}^{(n)}$

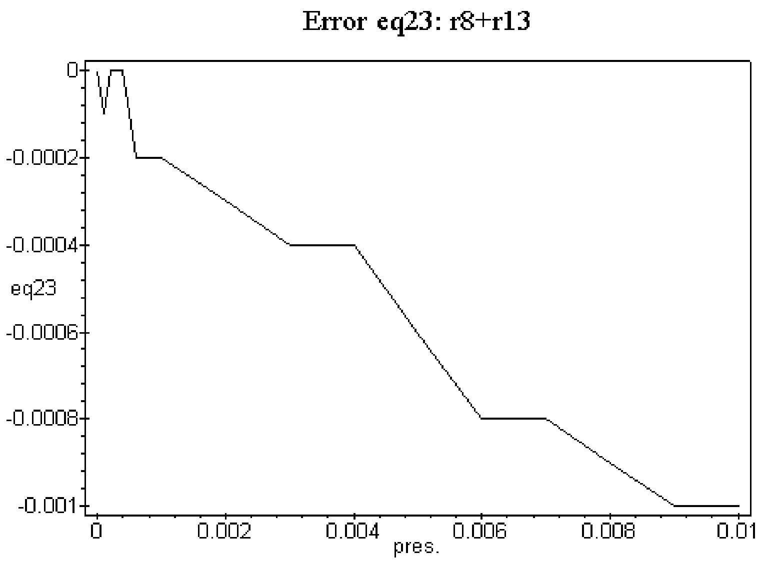

Figure 8: Plot of $e_{8,13}^{(n)} = r_8^{(n)} + r_{13}^{(n)} - R_{8,13}^{(n)}$

At this point data is ready to fit a linear equation to the data (r_k values found during the first 10 iterations as well as the total 20 iterations) using the calling sequence function `fit[leastsquare[vars]](data)` where `vars` is a list of variables, corresponding, in order, to the lists in `data`, and `data` is a list of statistical lists already prepared by previous region.

Also, in both cases, the coefficient of linear correlation between the two statistical lists is computed by the calling sequence function `describe[linearcorrelation](data1,data2)` and stored for future user's investigation since it measures how well a linear function explains the relationship between the two data lists (the coefficient of linear correlation is computed through this function by dividing the covariance by the square root of the product of the variance of the two data lists).

The result of `unapply(expr,x)` is a functional operator. Applying this operator to `x` gives the original expression. These results will be used to plot the straight lines found by applying leastsquare. Finally, the calling sequence `slope(equation)` computes the slope of a line determined by an equation.

```
> k:='k':
> for k from 1 to MD do
> leastsq_r[k]:=fit[leastsquare[[x,y]]]([pressure_values, r_values[k]]);
> lincorr_r[k]:=describe[linearcorrelation](pressure_values, r_values[k]);
> leastsq_rf[k]:=unapply(rhs(leastsq_r[k]),x):
> derivative_r[k]:=slope(leastsq_r[k]);
> leastsq_r_10[k]:=fit[leastsquare[[x,y]]]([pressure_values_10, r_values_10[k]]);
> lincorr_r_10[k]:=describe[linearcorrelation](pressure_values_10, r_values_10[k]);
> leastsq_rf_10[k]:=unapply(rhs(leastsq_r_10[k]),x):
> derivative_r_10[k]:=slope(leastsq_r_10[k]);
> od:
```

All those previous ressults have been stored, so at this point they can be printed by the user for further investigation, for example

```
>  print(derivative_r); print(derivative_r_10);
```

table([
$1 = -.28221$
$13 = -.45153$
$12 = -3.9932$
$4 = -2.8503$
$9 = -.66319$
$8 = -.10435$
$7 = -.89601$
$3 = -2.0672$
$6 = -.52914$
$5 = -.21166$
$10 = -1.9049$
$11 = -2.6739$
$2 = -.83003$
])

table([
1 = −.40911
13 = −1.3637
12 = −3.6366
4 = −2.2729
9 = −.90914
8 = −.40911
7 = −.90914
3 = −2.2729
6 = −.45457
5 = −.45457
10 = −1.8183
11 = −2.7274
2 = −.95460
])

In fact, the calculation of those derivatives is the main output of this module PDac. The values of this derivatives, together with the corresponding values calculated by modules CMAnion and CMCation, should be combined by module PDcalc to calculate the final values related to eq. 3 pg. 2 giving the final output of the complete system Polariz — see Figure 1 pg. 3.

The calculated derivatives can be saved in a Maple file named `deriv_r.ms` through the `save` command as shown bellow

```
> save derivative_r , derivative_r_10,'deriv_r.ms':
```

Finally, all this results related to r_i, $i = 1..md$ can be plotted by the user. As an example, Figure 9 shows the output produced by the following region which correspond to the calculated values of r_4 (RB^+) in each of the 20 iteration as well as the straight lines found by applying leastsquare using all the values and the first 10 values.

```
> Draw:= plot(r_plots[4],style=POINT):
> Drawls_r:=plot(leastsq_rf[4](x),x=0..0.01,color=green):
> Drawls_r_10:=plot(leastsq_rf_10[4](x),x=0..0.01,color=red):
> display({Draw,Drawls_r,Drawls_r_10},labels=['pres.','r'],title='r4=f(p)');
```

6 Conclusions

The module PDac, described in detail in this work, allows for the evaluation of ionic radii and their pressure dependence. An important fact is that it has flexibility so that a variable number of anions and cations may be considered, also varying the number of salts. Variable number of physical restrictions are allowed for; this and the study of errors make the selection of final results more reliable

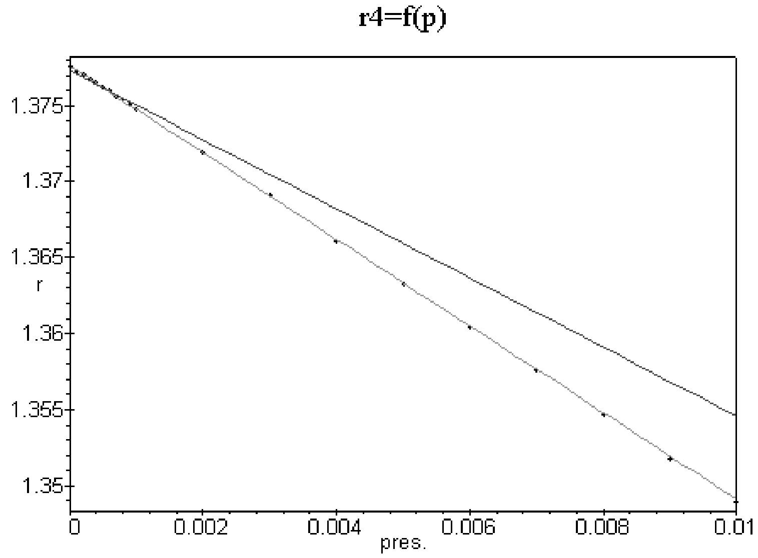

Figure 9: Plot of r_4^n and Leastquare Approximation

The PDac module, together with the previous ones — CMAnion and CMCation — allows for the calculation of the anion polarizability (based on a Clausius Mossoti treatment) and its pressure dependence for ionic crystals. Values of the pressure dependence of the dielectric constant may then be evaluated starting only from dapa at $p = 0$, which is particularly important for more complex crystals when experimental data at high pressures are not available.

References

[1] Barron, T.H.K.; Batana, A. Pressure Dependence of the Optical Properties of Alkali Halide Crystals. *Phil. Mag.* **1969**, 20(165), 619–719.

[2] Batana, A.; Soriano, M.R. On the Pressure Dependence of the Effective Ionic Charge of Alkaline Earth Fluorides. *J. Phys. Chem. Solids.* **1983**, 44(8), 741–746.

[3] Batana, A.; Faour, J. Pressure Dependence of the Effective Charge of Ionic Crystals. *J. Phys. Chem. Solids.* **1984**, 45(5), 571-577.

[4] Batana, A.; Castillo, A.D.; Fracchia, R.M. High Order Pressure Derivatives of the Static Dielectric Constants for Ionic Crystals within the Framework of the Exchange-Charge Model. *J. Phys. Chem. Solids.* **1992**, 53(10), 1309–1316.

[5] Batana, A.; Bruno, J.; Munn, R.W. Anion Polarizability in Alkali Halide Crystals. *Molecular Physics,* (in press).

[6] Char, B.W.; Geddes, K.O.; Gonnet, G.O.; Leong, B.L.; Monagan, M.B.; Watt, S.M. *Maple V Language Reference Manual.* Springer-Verlag, **1991**

[7] Dick, B.C.Jr.; Overhauser, A.W. Theory of the Dielectric Constants of Alkali Halide Crystals. *Phys. Rev.* **1958**, 112(1), 90–103.

[8] Fowler, P.W.; Madden, P.A. The In-crystal Polarizability of the Fluoride Ion. *Molecular Physics.* **1993**, 49(4), 913-923.

[9] Fowler, P.W.; Madden, P.A. In-crystal Polarizabilities of Alkali and Halide Ions. *Physical Review B.* **1984**, 29(2), 1035–1042.

[10] Fowler, P.W.; Madden, P.A. In-crystal Polarizability of O^{2-}. *J. Phys. Chem.* **1985**, 89, 2581–2585.

[11] Fowler, P.W.; Pyper, N.C. In Crystal Ionic Polarizabilities Derived by Combining Experimental and *ab initio* Results. *Proc. Roy. Soc. A398.* **1985**, 377, 377–391.

[12] Fracchia, R.M. *A Theoretical Study of Anharmonic Properties of Solids.* PhD Thesis, (in spanish), Facultad de Ciencias Exactas y Naturales, University of Buenos Aires, Argentina, **1995**.

[13] Havinga, E.E. Contribution to the Theory Of the Dielectric Properties of the Alkali Alides. *Phys. Rev.* **1960**, 119(4), 1193–1198.

[14] Mahan, G.D. Dielectric Parameters of Alkali Halides. *Phys. Rev.* **1986**, 34, 4235–4239.

[15] Monard, M.C.; Bruno, J.; Batana, A. A Computational System for the Calculation of Polarizabilities and their Pressure Dependence. Tech. Rep. ICMSC-USP, No. 51, **1997**.

[16] Tessman, J.R.; Kahan, A.H.; Shockley, W. Electronic Polarizabilities of Ions in Chrystals. *Phys. Rev.* **1953**, 92, 890–895.

[17] Wilson, M.; Madden, P.A. Polarization Effects in Ionic Systems from First Principles. *J. Phys.: Condens. Matter.* **1993**, 5, 2687–2706.

7 Appendix - Experimental Data

Experimental data used in this work has been taken from the experimental data at Low Temperature – LT – and Room Temperature – RT – described in [12]. It should be noted that some data was not available at LT, in those cases the missing data $*$ has been substituted by data at RT from the sane source.

Struct.	Salt	α_+ $10^{-24}cm^3$	α_- $10^{-24}cm^3$	r_0 $10^{-8}cm$	ε_0	ε_∞	χ_T $10^{-12}cm^2dyn^{-1}$
fcc	Li	0.030					
C=2	LiF		0.89	1.996	8.50	1.93	1.43
	LiCl		2.94	2.539	10.83	2.79	3.20
	LiBr		4.13	2.713	11.95	3.22	*4.20
	LiI		6.19	2.951	15.28	3.89	*5.7803
	Na	0.147					
	NaF		1.02	2.295	4.73	1.75	1.99
	NaCl		3.14	2.789	5.43	2.35	3.98
	NaBr		4.28	2.954	5.78	2.64	4.70
	NaI		6.35	3.194	6.62	3.08	6.45
	K	0.81					
	KF		1.20	2.648	5.11	1.86	3.10
	KCl		3.36	3.116	4.49	2.20	5.26
	KBr		4.54	3.262	4.52	2.39	6.17
	KI		6.66	3.489	4.66	2.68	7.75
	Rb	1.35					
	RbF		1.18	2.789	5.99	1.94	*3.81
	RbCl		3.45	3.259	4.53	2.20	*6.40
	RbBr		4.58	3.410	4.51	2.36	*7.69
	RbI		6.80	3.628	4.55	2.61	*9.48
	Cs	2.34					
	CsF		1.27	2.9827	7.27	2.17	*4.25
bcc	Cs	2.34					
C= $\frac{8}{3\sqrt{3}}$	CsCl		3.55	3.5150	6.68	2.67	4.5387
	CsBr		4.66	3.6645	6.38	2.83	5.4476
	CsI		6.81	3.9018	6.29	3.09	6.9396
	CaF			2.30			1.1628
	CaO			2.3995			0.9009
	SrF			2.51			1.4320
	SrO			2.5698			1.1198
	BaF			2.69			1.7513
	BaO			2.8129			1.3263
	MgO			2.1066			0.606

Table 6: Experimental Data at Low Temperatures