UNIVERSIDADE DE SANTIAGO DE COMPOSTELA
FACULTADE DE QUÍMICA

PhD Thesis

GAFit: A computational tool kit for parameterizations of potential energy surfaces

Roberto Rodríguez-Fernández

Santiago de Compostela, June 2014

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

FACULTADE DE QUÍMICA

DEPARTAMENTO DE QUÍMICA FÍSICA

PhD Thesis

GAFit: A computational tool kit for parameterizations of potential energy surfaces

Roberto Rodríguez-Fernández

Santiago de Compostela, June 2014

Dr. Emilio Martínez-Núñez e Dr. Saulo A. Vázquez-Rodríguez, Profesores titulares do Departamento de Química Física da Universidade de Santiago de Compostela, autorizan a presentación da tesis doctoral titulada "GAFit: A computational tool kit for parametrizations of potential energy surfaces" realizada por D. Roberto Rodríguez-Fernández, Licenciado en Química pola Universidade de Santiago de Compostela, para optar ao grao de Doctor en Química.

E para que así conste, asinamos o presente documento.

En Santiago de Compostela a 13 de xuño do 2014.

Dr. Emilio Martínez-Núñez   Dr. Saulo A. Vázquez-Rodríguez

Aos meus:
meus pais, Gloria e Primo,
miña irmá, Susana,
miña dona, Marysol,
meus fillos, Lois e Victoria Uxía,
e aos amigos...

GAFit:
A computational tool kit for parametrizations of potential energy surfaces

RESUMO

A superficie de enerxía potencial (**PES**) de un sistema goberna moitas das súas propiedades químicas, e particularmente, a dinámica, isto é, a evolución espacial dos núcleos co tempo. Hoxe en día, moitas das simulacións feitas integran as ecuacións clásicas do movemento, calculando as forzas sobre os átomos a cada paso directamente por cálculos de estruturas moleculares –dinámica directa– ou por **PES** analíticas.

Incluso, en sistemas pequenos, o uso dunha superficie analítica pode ser unha elección axeitada. Sen embargo, ata o que coñecemos, non hai un código xenérico que permita aos usuarios parametrizar superficies analíticas de unha forma fácil. O motivo final deste traballo é escribir un conxunto de programas que axuden aos usuarios no desenvolvemento de superficies analíticas.

GAFit foi inicialmente desenvolvido para facilitar o axuste de potenciais intermoleculares e a reparametrización de hamiltonianos semiempíricos usando un alogritmo xenético. Sen embargo, pode ser facilmente configurado para outros propósitos nos que se necesiten axustar series de parámetros.

A funcionalidade do paquete foi estendido separando o núcleo dos obxetivos a axustar. Os usuarios poden escoller, dependendo dos seus coñecementos de programación, dende introducir os seus propios potenciais directamente no código, usar un esquema fácil de encher co potencial requerido, ou para eses sen coñecementos, empregar unha expresión analítica ou os potenciais máis comúns xa listos para usar. Para facilitar a creación e configuración dos arquivos de entrada engadíronse un conxunto de ferramentas especializadas ao paquete.

A maiores, desenvolveuse unha interface externa para interactuar con programas externos. Usando esta interface creáronse as ferramentas necesarias para parametrizar o **MOPAC**. Co gallo de solucionar certos problemas atopados perante o primeiro estadio de desenvolvemento, creouse unha interface mellorada que incluso permite lanzar procesos **MOPAC** concorrentes acelerando os cálculos.

PALABRAS CHAVE

algoritmo xenético superficie enerxía potencial

SUMMARY

The potential energy surface (**PES**) of a molecular system governs many of its chemical properties, and particularly, the dynamics, that is, the spatial evolution of nuclei with time.

Most of the chemical dynamics simulations performed nowadays involve integration of the classical equations of motion, calculating the forces on atoms at each step either directly by electronic structure calculations –direct dynamics– or from analytical **PES**.

Even for small-size systems, the use of an analytical surface may be a convenient choice. However, to our knowledge, there is not a general code that allows users to parametrize analytical surfaces in a relatively easy way. The aim of the present work was to write a suite of programs to assist users in developing them.

GAFit was initially developed to facilitate fittings of intermolecular potentials and reparametrizations of semiempirical Hamiltonians using a genetic algorithm. However, it can be easily adjusted for other purposes in which fittings of a series of parameters are needed.

The functionality of the package was extended separating the core itself from the fitting targets. Users can choose, upon their programming skills, from introducing their custom potentials directly into code, use an easy pre-coded potential template to do so, or for those with no programming knowledge at all, that can use an analytical expression or the most used potentials coded just ready to use. A complete set of tools were added to the package to facilitate the creation and configuration of input files.

In addition, an external interface was developed to interact with external programs. Using this interface the tools needed to use **GAFit** to parametrize the **MOPAC** program were developed. A further **MOPAC** enhanced interface permits running parallel copies of **MOPAC** to speed up calculations and face up some problems encountered during the first stage development.

KEYWORDS

genetic algorithm potential energy surface

RESUMEN

La superficie de energía potencial (**PES**) de un sistema molecular gobierna muchas de sus propiedades químicas, y particularmente, la dinámica, esto es, la evolución espacial de los núcleos con el tiempo. Muchas de las simulaciones dinámicas realizadas hoy en día involucran la integración de las ecuaciones clásicas del movimiento, calculando las fuerzas sobre los átomos a cada paso mediante cálculos de estructuras electrónicas –dinámica directa– o mediante **PES** analíticas. Incluso para sistemas de pequeño tamaño, el uso de una superficie analítica es una opción conveniente. Sin embargo, de acuerdo a nuestro conocimiento, no hay un código genérico que permita a los usuarios parametrizar superficies analíticas de una forma relativamente fácil. El motivo del presente trabajo es escribir un conjunto de programas para asistir a los usuarios en su desarrollo. **GAFit** fue inicialmente desarrollado para facilitar el ajuste de potenciales intermoleculares y reparametrizaciones de hamiltonianos semiempíricos usando un algoritmo genético. Sin embargo, puede ser fácilmente configurado para otros propósitos dónde sea necesario ajustar series de parámetros.

La funcionalidad del paquete se extendió separando el núcleo de los objetivos. Ahora, los usuarios pueden escoger dependiendo de sus conocimientos de programación, desde introducir sus propios potenciales directamente dentro del código, o usar un esquema a rellenar con lo necesario, o para los que no tienen ningún conocimiento de programación, emplear una expresión analítica o los potenciales más comunes ya implementados. Se añadió un conjunto completo de de utilidades para facilitar la creación y configuración de los ficheros de entrada. También desarrolló una interface externa para interactuar con otros programas. Usando esta inteface se desarrollaron las herramientas necesarias para usar **GAFit** en la parametrización del **MOPAC**. Para solucionar los problemas encontrados durante el primer desarrollo, se escribió una interface mejorada que incluso permite correr copias concurrentes de **MOPAC** para acelerar los cálculos.

PALABRAS CLAVE

algoritmo genético superficie energía potencial

Contents

Acknowledgements

First and foremost, I have to thank my thesis directors, Emilio Martínez Núñez and Saulo A. Vázquez Rodríguez, for their support and understanding over these past years.

I would also like to thank Francisco Baptista Pereira and Jorge M. C. Marques. Without their help this thesis would not have been possible.

Last, but not least, I would like to thank my family and friends for their support.

Conventions

Symbols

⟶ tabs

␣ blank spaces

. . . or [. . .] more output not shown

⤸ wrapped line

⤹ wrapped line continuation

Acronyms

3C	three-centered	218
4C	four-centered	219
AM1	Austin Model 1	187
AM1-SRP	AM1 Specific Reaction Parameters	198
BLX-α	Blend Alpha Crossover	31
BSSE	Basis Set Superposition Error	191
CESGA	"Centro de Supercomputación de Galicia"	208
CID	Collision-Induced Dissociation	187
CPU	Central Processing Unit	5

DNA	deoxyribonucleic acid . . . 18
DPC	Double Point Crossover (or TPX, Two Point Crossover) 21
ESI	Electronic Supplementary Information . . . 190
TR-FTIRES	time-resolved Fourier transform infrared emission spectroscopy . . . 218
FPU	Floating Point Unit . . . 73
GA	Genetic Algorithm . . . 6
GUI	Graphical User Interface . . . 123
IDE	integrated development environment . . . 266
ITPs	Inner Turning Points . . . 193
IVR	Intramolecular Vibrational Energy Redistribution . . . 189
KMC	Kinetic Monte Carlo . . . 193
LJ	Lennard-Jones . . . 194
MEP	minimum energy path . . . 220
MOPAC	Molecular Orbital PACkage . . . 36
MPX	Multiple Point Crossover . . . 21
NFS	Network File System . . . 100
PBS	Portable Batch System . . . 262
PDF	portable document format . . . 263
PES	Potential Energy Surface . . . 5
PNG	portable network graphics . . . 263
QCT	Quasiclassical Trajectory . . . 191
RNA	ribonucleic acid . . . 189
RRKM	Rice–Ramsperger–Kassel–Marcus . . . 189
SBX	Simulated Binary Crossover . . . 30
SPC	Single Point Crossover (or SPX) . . . 21
SRP	Specific Reaction Parameters . . . 208
VC	vinyl cyanide . . . 169

Input, output and files

- A command line interactive shell session:

```
tar -xvzf gafit-VERSION.tar.gz
cd gafit-VERSION
./configure
make
make install
```

- A program output to interactive terminal or redirected to a file:

```
Build: 2122

Settings for job
----------------------------
Geometries:[coord.molden]
Energies:[energies]
Atom2type:[atom2type.txt]
Bounds:[bounds.txt]
Charges:[charges.txt]
Potential read: 2
All coefficients: no, Read and repeat subset
Fitting: absolute
[...]
```

- An input or output file:

File 1: Input file example.

```
[job]
runs:␣␣␣␣␣␣␣␣␣␣1
evaluations:␣␣5000000
Geometries:␣␣␣moldeni.dat
Energies:␣␣␣␣␣energies.dat
Atom2type:␣␣␣␣atom2types.txt
Bounds:␣␣␣␣␣␣␣bounds.txt
Charges:␣␣␣␣␣␣charges.txt
Potential:␣␣␣␣1
All␣coefficients:␣0
```

- Source code file:

File 2: C source code

```

int
```

```
build (void)
{
  char *build = "$Rev: 3521 $";
  return atoi (index (build, ':') + 1);
}
```

- Command line tool syntax:

```
command [-a][-b c] [-d [e]] [-f {g|h|i}] mandatory-argument [optional-argument]
```

options or *flags* consist of '**-**' characters and single letters or digits, such as '**-a**' or '**-1**' which enable a feature. Some of them have an option argument too, like the '**-b c**', where '**c**' is the argument for option '**-b**'. Here '**c**' is used to 'tune' the 'feature' enabled with '**-b**'.

Arguments or option-arguments enclosed in the '**[**' and '**]**' notation are optional and can be omitted like the '**[optional-argument]**' or '**[e]**' or '**[-d [e]]**'. The ones not enclosed like '**mandatory-argument**' must be set.

If the '**-b**' feature is enabled '**c**' must be set, but if the '**-d**' feature is enabled, '**e**' is optional.

'**{**' and '**}**' notation represents a set of options to select. Arguments separated by the '|' bar notation are mutually-exclusive, and only one of them must be chosen from the set enclosed with '**{**' and '**}**'.

- Menu selection sequence: Edit » Tree » Internal job » Analytical

- keystrokes:
  - A + O: A key plus O key at the same time.
  - A, O: A key, then O key.

1 Introduction and objectives

To invent, you need a good
imagination and a pile of junk.

Thomas A. Edison

One of the key concepts in chemistry is that of Potential Energy Surface (PES)[1]. It comes from the Born-Oppenheimer approximation, which facilitates the solution of the time-independent Schrödinger equation for molecular systems. Fortunately, the errors associated with this approximation are negligible for many of the systems and conditions of interest to chemists. The potential energy surface of a molecular system governs many of its chemical properties, and particularly, the dynamics, that is, the spatial evolution of nuclei with time. Most of the chemical dynamics simulations performed nowadays involve integration of the classical equations of motion, calculating the forces on atoms at each step either directly by electronic structure calculations (i.e., "on-the-fly" or direct dynamics) or from analytical PESs. In principle, the direct dynamics approach may be the preferred option for simulations of reactive systems that include a small number of atoms, because one avoids the construction of the analytical surface. The use of analytical PESs, however, has a clear advantage in terms of Central Processing Unit (CPU)-time costs, being mandatory in molecular dynamics simulations of systems composed of thousands of atoms[1]. Even for small-size

[1]In molecular mechanics and molecular dynamics, the analytical potential energy surface of a system is generally known as the force field.

systems, the use of an analytical surface may be a convenient choice. If it is developed with care, it may be almost as accurate as the exact surface corresponding to the electronic structure method used as a reference for its construction.

The development of analytical PESs or force fields may be facilitated by using optimization methods, and many research groups have been using them for their particular purposes. However, to our knowledge, there is not a general code that allows users to parametrize analytical surfaces or force fields in a relatively easy way. The aim of the present work was to write a suite of programs to assist users in developing analytical surfaces. This suite of programs will be called **GAFit**. We used this name because, with this computational tool kit, a Genetic Algorithm (GA) conducts the fitting –Fit– or parametrization of a desired potential energy surface. The genetic algorithm was not developed in this work; rather it was taken from the literature[2][3]. For our purposes, the advantages of a genetic algorithm against other type of optimization methods are detailed later on. In this work, the **GAFit** program is applied to the development of an intermolecular potential for the interaction between Xe and the $[\text{Li(Uracil)}]^+$complex, and to the reparametrization of a semiempirical Hamiltonian[2]. The program, however, can be easily adapted to conduct any type of fittings or parametrizations of analytical surfaces or force fields, as well as other optimization problems in chemistry.

[2]Semiempirical Hamiltonians supplemented with specific reaction parameters were first proposed by Truhlar[4] as a practical method for direct dynamics calculations.

Part I

Maths

2 Optimization methods

A mathematician is a device for turning coffee into theorems.

Alfréd Rényi

2.1 Introduction

An *optimization*, *mathematical optimization* or *mathematical program* is a problem that consists of finding the best element from some possible set, using some criteria. Usually, it implies the maximization or minimization of the so called *objective function*. Here, the term *program* antedates computers and means *preparing a schedule of tasks*.

The generic *mathematical optimization* problem[5] can be expressed as[6]:

$$\text{Optimize} \quad f(x) \tag{2.1.1}$$
$$\text{Subject to the constraints:} \quad \begin{cases} g(x) \in S_1 \\ x \in S_2 \end{cases}$$

where x is a vector of variables which are used to maximize or minimize the function, $f(x)$, that expresses the objective algebraically. S_1 and S_2 are any set to reflect the constraints that must obey $g(x)$ and the variables respectively.

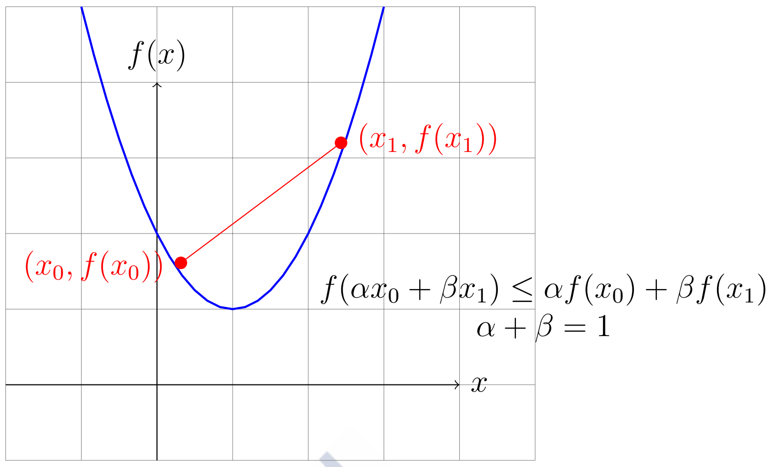


Figure 2.1: Convex function.

By convention, the standard form of an optimization problem is stated in terms of minimization because minimizing $f(x)$ is the same as maximizing an adequate function $h(x)$ and viceversa.

A point x is *feasible* if it satisfies all constraints. The *feasible region* is the set of all feasible points: those for $g(x)$ belongs to S_1 and x belongs to S_2. A *mathematical optimization* is *feasible* if its *feasible region* is not empty.

An important concept to take into account is *convexity* as we see in 2.3. A function is *convex* –see Fig. 2.1– if it satisfies the inequality:

$$f(\alpha x + \beta y) \leq \alpha f(x) + \beta f(y) \qquad \forall \quad x, y \in R^n \tag{2.1.2}$$
$$\begin{cases} \alpha + \beta = 1 \\ \alpha \geq 0, \beta \geq 0 \end{cases} \qquad \forall \quad \alpha, \beta \in R$$

A *linear function* is a special case of *convex* where

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y) \tag{2.1.3}$$

Many different *types of problems* embrace the mathematical programming problem[6]:

- *linear programming problem*: if $f(x)$ and $g(x)$ are linear and the x's are individually non-negative.

- *integer programming problem*: if the $x \in S_2$ restriction requires some x's to take on integer values.

- *nonlinear programming problem*: if $f(x)$ and $g(x)$ are general nonlinear functions with S_2 being nonnegativity conditions.

- etc.

Examining the different parts of the whole problem, we can have different *types of mathematical optimization*[5]:

- Abstract
- Biconvex
- Bilinear
- Composite concave
- Continuous
- Convex
- Discrete
- Disjunctive
- Dynamic
- Factorable
- Fractional
- Geometric
- Integer
- Infinite
- Lattice
- Linear
- Mixed-Integer
- Multilevel
- Nonlinear
- Pseudo-boolean
- Reverse convex
- Semi-definite
- Semi-infinite
- Separable

In behalf of clarity, a simpler and practical classification can be used as in Boyd and Vandenberghe [7] taking into account convexity and linearity.

2.2 Linear programming

A linear programming problem is a problem of minimizing a linear function –(2.1.3)– with linear constraints of the inequality and/or the equality type[8]:

$$\begin{array}{llccccccccc}
Maximize & Z = & c_1x_1 & + & c_2x_2 & + & \dots & + & c_nx_n & \\
subject & to & a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & \leq b_1 \\
 & & \dots & + & \dots & + & \dots & + & \dots & \dots \\
 & & a_{m1}x_1 & + & a_{m2}x_2 & + & \dots & + & a_{mn}x_n & \leq b_m \\
 & & x_1 \geq 0 & & x_2 \geq 0 & & \dots & & x_n \geq 0 &
\end{array}$$

Z is the *objective function* to be minimized, $c_1, c_2, ..., c_n$ are the *cost coefficients* and $x_1, x_2, ..., x_n$ are the *decision variables.* Finally, a_{ij} are the *technological coefficients* forming the *constraint matrix.*

There are effective methods for solving linear programming problems as the **simplex method** or the **interior-point methods**.

2.3 Convex optimization

A *convex optimization problem* is one in which the objective and constraint functions are convex, which means they satisfy the inequality (2.1.2). Linear programming and least-squares problems are special cases of convex problems. Convex programming problems have a well developed theory, and can be solved numerically very reliably and efficiently, using interior-point methods or other special methods for convex optimization[7].

Convex optimization has also found wide application in global optimization, where it is used to find an optimal value as approximate solutions.

2.4 Nonlinear optimization

Nonlinear optimization is the term used to describe an optimization problem when the objective or constraint functions are **not linear, but not known to be convex**[7].

There are no effective methods for solving the general nonlinear programming problem. The simple ones with a dozen of variables are difficult to solve, and those with one hundred variables may become an impossible task.

2.4.1 Local optimization

In local optimization we search for a good feasible point, x^*, that is the best compared with nearby feasible points in the same region:

$$\forall x \quad \exists \delta > 0 \quad \begin{cases} \|x - x^*\| < \delta \\ f(x^*) \leq f(x) \end{cases} \quad \text{e.g., if } x^* \text{ is a minimum} \tag{2.4.1}$$

But it may not coincide with the globally optimal solution.
There are some facts about local optimization methods:

- For most local optimization methods differentiability of the objective and constraint functions, with respect to the variables or parameters, is the only requirement.
- You have to experiment with the choice of algorithm to find a suitable one to the problem at hand.
- They are often sensitive to algorithm parameters and must be adjusted depending on the problem to be solved.
- In many cases, an initial guess –*seed*– is needed. This has a huge influence on the solution.

In figure 2.2 we can see an example of a surface with local minima (green), and an absolute minimum (in red). Using our initial guess, and the gradient to move over the surface, we can find one of the minima, but it may not be the global one.

2.4.2 Global optimization

Global optimization is the task of finding the absolute minimum (or best solution). It is considered "*the hardest part of a subject called nonlinear programming*[9]". In the worst case, complexity grows exponentially with the sizes of parameters and constraints, so it can take a long time to solve.

If the number of variables is big –hundreds–, the cost in computation time can make the problem *intractable*[7]. An *intractable* problem in *complexity theory* is a a problem in which no algorithm can exist computing all instances of it in *polynomial time*[10]: When the execution time of a computation, $m(n)$, is no more than a polynomial function of the problem size, n. More formally $m(n) = O(n^k)$ where k is a constant[11].

Global optimization algorithms can be classified according to the method of operation into two different types[12]: deterministic and probabilistic.

Deterministic At least one way to proceed exists in each execution step. If there is not any way, the algorithm ends. Deterministic algorithms are often used if there is a clear relation to the fitness of the possible solutions. If so, the search space could be efficiently explored to find good solutions.

Probabilistic If the relation is not obvious, if it is difficult or if the search space has a high dimensionality, then probabilistic algorithms are used. In general, to obtain an optimal solution, you must spend time exploring the search space.

Examples of deterministic are *State Space Search*, *Branch and Bound* and *Algebraic Geometry*.

Examples of probabilistic are *Hill Climbing*, *Random Optimization*, *Simulated Annealing*, *Genetic Algorithms* etc..

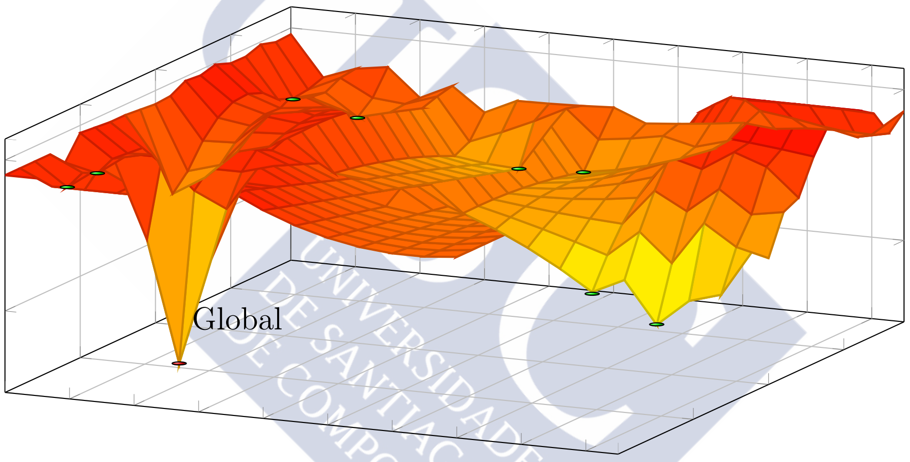


Figure 2.2: Global search.

3 Evolutionary Algorithms

I am turned into a sort of machine for observing facts and grinding out conclusions.

Charles Darwin

Evolutionary algorithms are a good tool in Global Optimization because they make no assumptions about the problem, and therefore, they usually perform very well in all types of problems[12].

These algorithms employ techniques inspired in biology such as reproduction, mutation, recombination and selection applied to a set of candidates used as a population to find optimal ones.

Evolutionary algorithms proceed according to the scheme shown in Figure 3.1. A population is initialized; then, each member is evaluated according to some objective function. And finally, some of the members

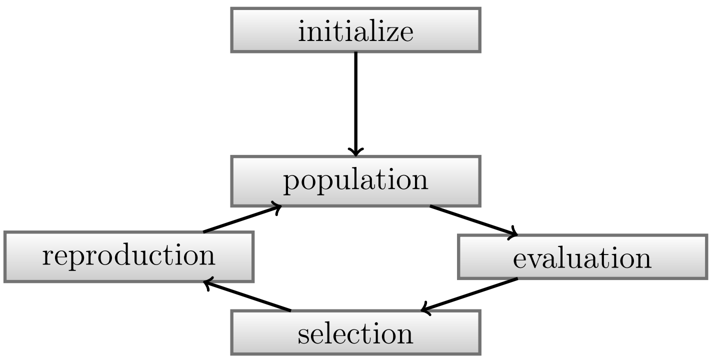


Figure 3.1: Evolutionary algorithms.

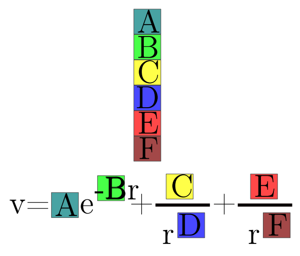


Figure 3.2: Genes and chromosome example: 4th potential from Table 6.2.

are selected to create a new population using reproduction techniques. The process continues until a population member turns out to be a good solution, or a maximum number of populations are reached.

There are many evolutionary algorithm types with distinctive features depending on how the populations are used, how the individuals are represented, how the individuals are selected to reproduction, how the offspring are included in the population of the next generation, etc.

The population of the next generation can be formed from:

- a combination of the current population and its offspring,
- some or all of the offspring, and none of the current generation individuals,
- none or some of the best individuals –known as **elitist algorithm**– are propagated to the next generation.

We describe here two types of evolutionary algorithms of our interest: Genetic Algorithms and Genetic Programming.

3.1 Genetic Algorithms

The individuals are described by an array of elementary types –the *genes*: any suitable representation, including bits and bytes– similar to a deoxyribonucleic acid (DNA) string, and are also called a *chromosome*.

Each *gen* can describe a characteristic, e.g. a double precision polynomial coefficient value like the example in Fig. 3.2 where is represented the 4th potential from Table 6.2.

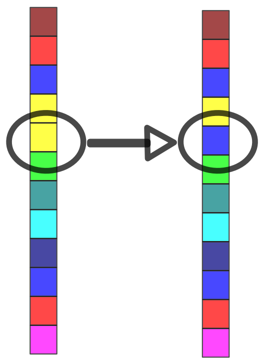


Figure 3.3: Single gene mutation.

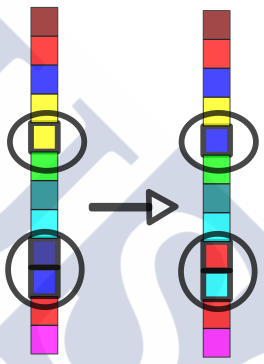


Figure 3.4: Multiple gene mutation.

Chromosomes could be fixed or variable length strings. The type, number, characteristics, etc of genes and how they are related in the chromosoma is a problem type dependent matter.

There are some *genetic operators* which can be applied over a chromosome string: **Mutation**, **permutation** and **crossover**.

3.1.1 Mutation

Mutation randomly changes one or more genes. If the chromosomes are of fixed length, we may have a single gene mutation (Fig. 3.3) or a multiple gene mutation (Fig. 3.4), and if the chromosomes are of variable length, there can be an insertion (Fig. 3.5) or a deletion (Fig. 3.6).

3.1.2 Permutation

Permutation exchanges a pair of genes. Fig. 3.7.

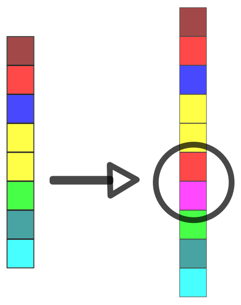


Figure 3.5: Variable lenght insertion.

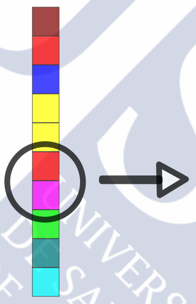


Figure 3.6: Variable lenght deletion.

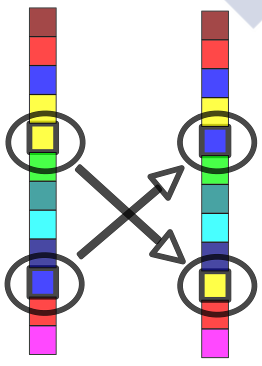


Figure 3.7: Permutation.

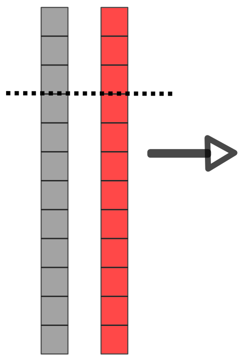


Figure 3.8: Single point crossover.

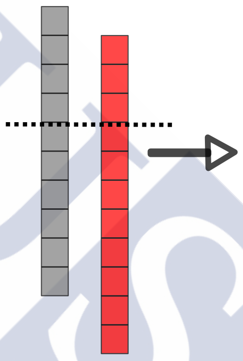


Figure 3.9: Variable lenght single point crossover.

3.1.3 Crossover

Crossover recombines two chromosomes to obtain a new one. Some crossover types are described in the literature as Single Point Crossover (SPC), Double Point Crossover (DPC), and Multiple Point Crossover (MPX). As above, the chromosomes can be of fixed or variable length. See Fig. 3.8, 3.9, 3.10 and 3.11.

3.2 Genetic Programming

We do not use in this Thesis Genetic Programming, but we do not disregard its use in the future as it provides a new means to implement new capabilities in the program. One of those capabilities that would be of great interest to us is the optimization of the functional form of the potentials we employ to fit a set of *ab initio* energies for two interacting species.

Genetic Programming includes all evolutionary algorithms that cre-

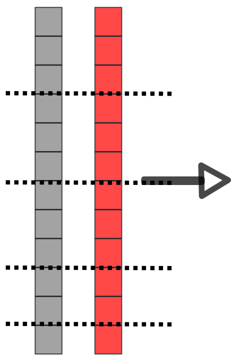


Figure 3.10: Multiple point crossover.

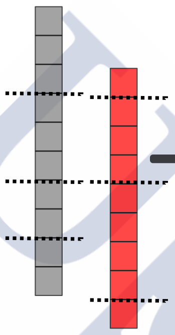


Figure 3.11: Variable lenght multiple point crossover.

ate and modify programs or algorithms. The genes are instructions, inputs or constants, and the chromosomes pieces of interpretable code. The goal is to find a representation of code that, when ran with a known input, shows some kind of desired behavior.

Genetic Programming usually uses tree representations of chromosomes. The Fig.3.12 is a good example where a simple calculation of $\frac{a+b}{c}$ is represented as a tree chromosome. Leaf nodes are inputs or constants, the non leaf nodes are operations.

The potentials implemented in *potentials.f*, shown in Table 6.2, are easily calculated with the routines from **Fpu** –section 11–, and hence suitable to be employed in a search with Genetic Programming using tree chromosomes as represented in Fig.3.13.

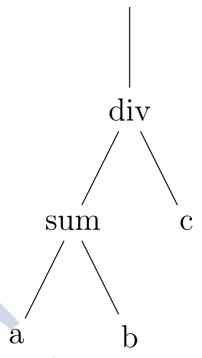
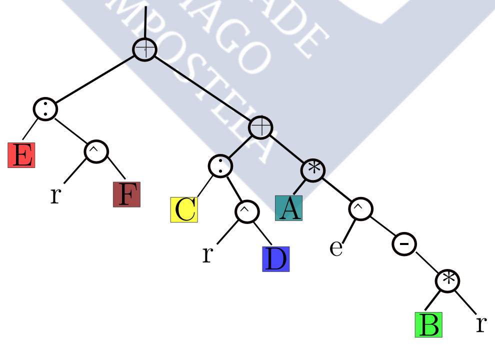


Figure 3.12: Tree choromosome.



$$v = Ae^{-Br} + \frac{C}{r^{D}} + \frac{E}{r^{F}}$$

Figure 3.13: 4th potential from Table 6.2.

4 The Genetic Algorithm

In mathematics you don't understand things. You just get used to them.

John von Newmann

The genetic algorithm used here was developed by Marques [2] and co-workers and slightly modified to support integer parameters in the function employed to fit interaction energies. The GA main loop is shown in File 4.1. As expected, it begins creating and evaluating the first population prior to run into the main loop –a *do-while* between lines 109-179–.

File 4.1: ga.c

```
64
65 void
66 ga (int evaluations, int pop_size, double p_cx, double blx_alpha,
67     double eta_sbx, double p_mt, int elite,
68     int size_k, int cx_type, int mutation_type, int
       mutation_integer,
69     double sigma, p_ind all_time_best, vect_domain bounds, int dir
       )
70 {
71   int generation = 1;
72   int current_evaluations = 0;
73   ind best, new_best, dummy;
74   p_ind population;
75   p_ind new_population, temp;
76   int last_evals;
77   int print;
78   static int flag = 0;
```

```
int i;


// make and evaluate population
population = make_population (pop_size, bounds);
best.genes = malloc (sizeof (double) * N);
new_best.genes = malloc (sizeof (double) * N);

current_evaluations += evaluate_pop (population, pop_size, N);

dummy = get_best (population, pop_size, dir);
copia_ind (&dummy, &best, N);
if (flag == 0)
  {
    flag = 1;
    copia_ind (&best, all_time_best, N);
  }

else
  update_all_time_best (best, all_time_best, N, dir);
stats (population, generation, current_evaluations, pop_size,
  best, N, 1);
last_evals = current_evaluations;

// allocates memory for individuals (to generate the new
  population)
new_population = malloc (sizeof (ind) * pop_size);
for (i = 0; i < pop_size; i++)
  {
    new_population[i].genes = malloc (sizeof (double) * N);
  }
// evolution cycle
do
  {
    generation++;

    // tournament selection
    tournament_selection (population, new_population,
                          pop_size, size_k, dir, N);

    // applies genetic operators
    apply_crossover (new_population, pop_size, N, p_cx,
                     cx_type, blx_alpha, eta_sbx, bounds, dir);
    apply_mutation (new_population, pop_size, N, p_mt,
      mutation_type,
                    mutation_integer, bounds, sigma);

    // evaluate populations
    current_evaluations += evaluate_pop (new_population,
      pop_size, N);

    // GA simples
    temp = population;
```

```
        population = new_population;
        new_population = temp;

        // Get best from new generation
        dummy = get_best (population, pop_size, dir);
        copia_ind (&dummy, &new_best, N);
        if (elite)
          {
            if (dir == 1)
              {
                if (new_best.fitness < best.fitness)
                  {
                    apply_elite (population, pop_size, best, dir, N)
                      ;
                  }

                else if (new_best.fitness > best.fitness)
                  {
                    copia_ind (&new_best, &best, N);
                  }
              }

            else
              {
                if (new_best.fitness > best.fitness)
                  {
                    apply_elite (population, pop_size, best, dir, N)
                      ;
                  }

                else if (new_best.fitness < best.fitness)
                  {
                    copia_ind (&new_best, &best, N);
                  }
              }
          }
        update_all_time_best (best, all_time_best, N, dir);

        // output
        if ((current_evaluations < 100)
            || ((current_evaluations < 1000)
                && (current_evaluations - last_evals > 100))
            || (current_evaluations - last_evals > 250))
          {
            print = 1;
            last_evals = current_evaluations;
          }

        else
          print = 0;
        stats (population, generation, current_evaluations,
               pop_size, best, N, print);
      }
```

```
while (current_evaluations < evaluations);
```

The system is configured reading an input file –Section 6–. Once configured, the GA main loop routine starts and continues till a maximum number of evaluations is reached as shown in Figure 4.1. The GA only comunicates with the external world –internal or external routines or programs– through the evaluation phase and when some subroutines print outputs.

Table 4.1: GA subroutines

subroutine	source	comments
ga	ga.c	main loop
tournament_selection	selection.c	tournament algorithm
apply_elitism	selection.c	elitism algorithm
apply_crossover	crossover.c	crossover
apply_mutation	mutation.c	mutation
evaluate_pop	evaluation.c	this subroutine works as an interface switching the evaluation to the desired type of application
get_best	selection.c	

4.1 Tournament Selection

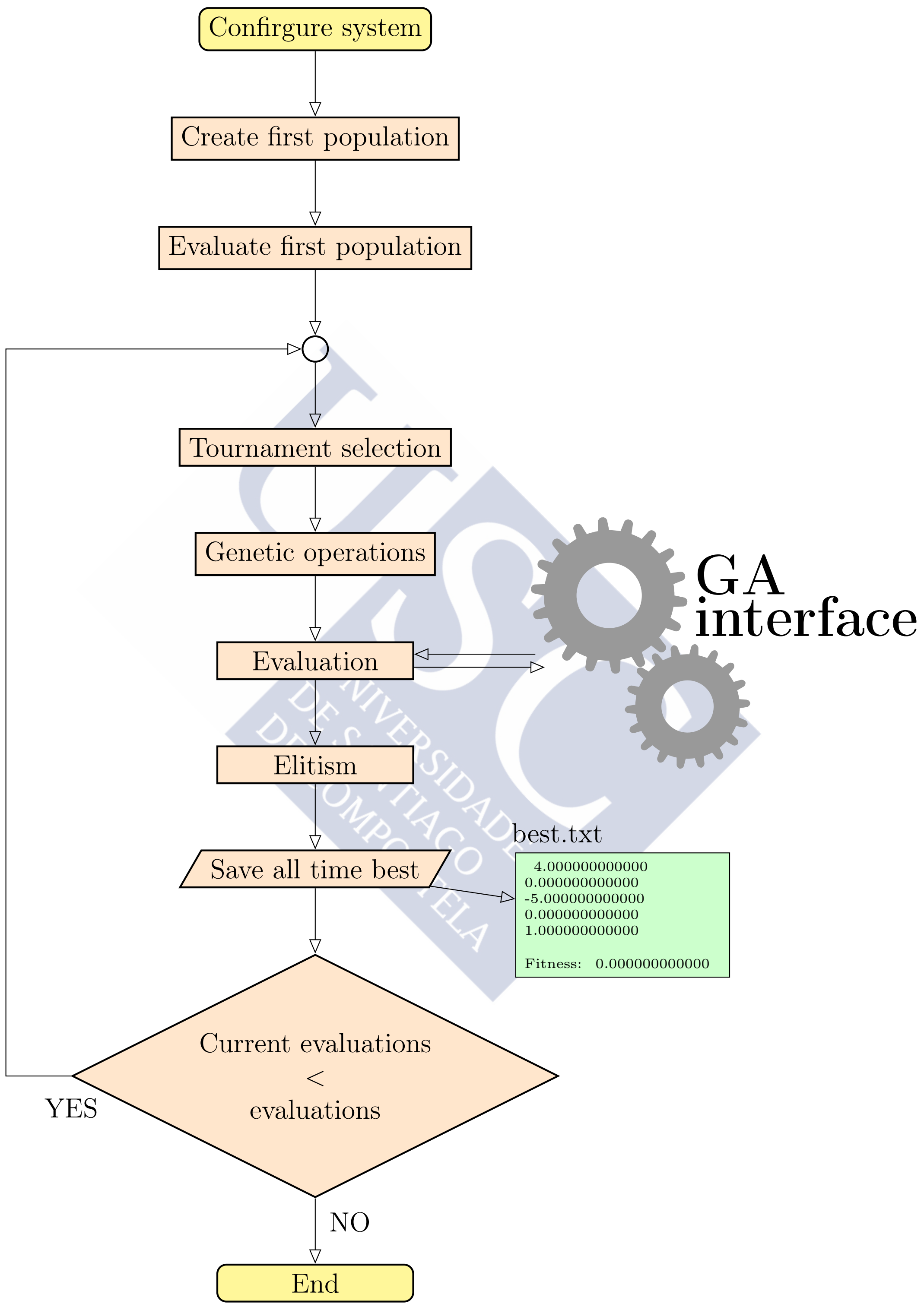
A subset of K individuals are selected randomly from the old population. The best of the set is selected and introduced in the new population. This operation is repeated till the new population is completed. K is the tournament controlling parameter: *Tournament size*.

4.2 Genetic operations

4.2.1 Crossover

For all the population, each two consecutive individuals, a random number between 0 and 1 is obtained and if it is greater than the *crossover rate* a crossover is performed obtaining two new offspring replacing their parents. The type of crossover selects the operator to apply:

- Single point crossover. A random point is selected and the offspring are obtained from the parents by exchanging the tail segments.



```
 4.000000000000
0.000000000000
-5.000000000000
0.000000000000
1.000000000000

Fitness:   0.000000000000
```

Figure 4.1: GA main loop

- Double point crossover. Two random points are selected and the offspring are obtained from the parents by exchanging the center segments.

- Simulated Binary Crossover (SBX)[13]. SBX simulates a SPC operator on binary strings obtaining two offspring having some interesting properties to self-adaptation[14]:

  – high probability to mantain the extend between them like the parents

  – high probability to be near the parents values

  SBX works as follows:

  – A random value between 0 and 1 is selected: $\mu \in [0,1]^1$.

  – Using a uniform distribution we calculate β so the area under probability curve from 0 to β is equal to μ:

$$\beta = (2\mu)^{\frac{1}{\eta+1}} \quad if\ \mu \leq 0.5$$
$$\beta = \left(\frac{1}{2(1-\mu)}\right)^{\frac{1}{\eta+1}} \quad if\ \mu > 0.5$$

  – Now, we obtain the two children, C_1 and C_2, from the parents, P_1 and P_2:

$$C_1 = \frac{1}{2}\left[(1+\beta)P_1 - (1-\beta)P_2\right]$$

$$C_2 = \frac{1}{2}\left[(1-\beta)P_1 + (1+\beta)P_2\right]$$

  The controlling parameter is η –eta_sbx, Table 6.1– which is a real non negative number. Larger values increase probability of children close to their parents while small ones increase probability of distant children[2].

^{1}Really, here the coded implementation is $\mu \in [0, 0.99]$ to avoid a *divide by zero* problem in the calculation of β

- Blend Alpha Crossover (BLX-α)[15]. BLX-α crossover creates new offspring choosing a random value for each gene in the range:

  $$[G_{min} - \Delta\alpha, G_{max} + \Delta\alpha]$$

  Here G_{min} and G_{max} are the smallest and largest of the two parents gene values. Δ is $G_{max} - G_{min}$. The value obtained is checked and limited to the acceptable values for the gene, called the bounds.

  BLX-α crossover has the first interesting self-adaption property of SBX: high probability to mantain the extend between them like the parents[14].

  The controlling parameter is α –blx_alpha, Table 6.1– which determines the degree of variability. It was reported that a value $\alpha = 0.5$[2] performs better than other values for many test problems[14].

SBX and BLX-α are calculated crossovers. In both cases, if an integer gene type is used, they revert to a Single Point crossover.

4.2.2 Mutation

The application is slighty different from the crossover operators. Here *mutation rate* operates over genes while *crossover rate* operates over individuals:

- For all individuals in the population, a call to mutation subroutines is performed obtaining a new offspring replacing the parent.

- For each individual's gene, a random number between 0 and 1 is obtained, and if it is greater than the *mutation rate* the corresponding mutation is performed in the gene[3].

There are four types of mutation to apply upon coefficient nature and user choice:

- Real coefficients: *Random* and *sigma*.

[2]Known as BLX-0.5 crossover

[3]As the *mutation rate* drops to zero, the probability that the parent replaces itself increases.

   - *Random* mutation. The parent gene is replaced by a random number obtained from the acceptable set of values for the gen –bounds–.
   - *Sigma* mutation. The child gene, G_{child}, is replaced by a new value calculated from parent G_{parent} as:

     $$G_{child} = G_{parent} + \sigma(G_{max} - G_{min})N(0,1)$$

     G_{max} and G_{min} are bounds, $N(0,1)$ is a random value sampled from a *standard normal distribution* and σ –sigma, Table 6.1– is the control parameter.

     The value is checked against the bounds, and if in five tries a suitable value between bounds is not found, a *random* mutation is performed.

- Integer coefficients: *Random* and *adjacent*.
   - *Random* mutation. The parent gene is replaced by a random integer number between bounds.
   - *Adjacent* mutation. *Adjacent* changes the parent gene by a unit amount as follows:

$$G_{child} = \begin{cases} G_{min} + 1 & \text{if } G_{parent} = G_{min} \\ G_{max} - 1 & \text{if } G_{parent} = G_{max} \\ \text{otherwise randomly:} & \begin{cases} G_{parent} + 1 \\ G_{parent} - 1 \end{cases} \end{cases}$$

4.3 Elitism

Finally, elitism is applied: A random individual of the new generation is replaced with the best from parent generation ensuring that the quality of the best does not decrease along the time.

Part II

Program Details

5 GAFit

DNA is like a computer program but far, far more advanced than any software ever created.

Bill Gates

5.1 Introduction

GAFit is a package of programs initially developed to facilitate fittings of intermolecular potentials and reparametrizations of semiempirical Hamiltonians. However, it can be easily adjusted for other purposes in which fittings of a series of parameters are needed. The core of the package is the genetic algorithm developed by Marques [2] and co-workers.

The functionality of the package was extended separating the core itself from the fitting targets –See Figure 4.1–. Now, users can choose, upon their programming skills, from introducing their custom potentials directly into code, use an easy pre-coded potential template to do so, or for those with no programming knowledge at all, use an *analytical expression* or the most used potentials coded just ready to use. A complete set of tools were added to the package to facilitate the creation and configuration of input files.

In addition, a external interface was developed to interact with external programs. Using this interface were developed the tools needed to use **GAFit** to parametrize the Molecular Orbital PACkage (MOPAC). A further MOPAC enhanced interface permits running parallel copies of MOPAC to speed up calculations and face up some problems encountered during the first stage development.

5.2 Installation

The configuration, compilation and installation phases are done by the *GNU autotools* utilities.

```
tar -xvzf gafit-VERSION.tar.gz
cd gafit-VERSION
./configure
make
make install
```

The binaries go into $HOME/bin and other files into $HOME/share. To install into /usr/local (note that you need *superuser* permissions.), use:

```
./configure --prefix=/usr/local
make
sudo make install
```

To force a fortran compiler (e.g. *ifort*) use:

```
./configure FC=ifort
```

To force a C compiler (e.g. *icc*) use:

```
./configure CC=icc
```

Or any combination above:

```
./configure --prefix=/usr/local FC=ifort CC=icc
```

To compile with debug options:

```
./configure --enable-debug
```

In addition, the usual targets of *Autotools* apply (i.e. *make distcheck*, *make clean* etc).

5.3 Usage

5.3.1 Overview

Internal job

Internal job means here a job where an internal intermolecular potential energy function is parametrized to fit a set of interaction energies between two fragments.

You must have the files: *geometries*, *energies* and *charges* if needed. See 6.3.

To edit and/or build the *atom2type* file, you can use the **needle** tool with the *geometries* file as input. See 12.1. It is compulsory to check the *atom2type* file if you employed the **needle** tool.

Once you know how many different interactions you have in your system you can write the *bounds* file, page 48.

Write the *job.txt*, see 6.3. Run the **GAFit** executable in the folder where all the files are located.

If you want to run more than one **GAFit** process from the same folder –with the same configuration–, you can use a command line argument to distinguish the output files:

```
$ gafit alpha
Build: 2427
TAG:alpha
        OUTPUT_FILE:   stats.alpha.txt
        OUTPUT_BEST:   best.alpha.txt

Settings for job
-----------------------------
[...]
```

and change their names accordingly to prevent overwriting. See 7.2.

External job

The files needed for an *external job* depend on the type of job to be done. In this case, an external program or tool evaluates the coefficients vector. Of all the files needed in a internal job, only the *bounds.txt* file is needed to execute an *external job*.

In an external job, for instance, an *ab initio*, *density functional theory* or *semiempirical* program can be employed to calculate the properties of our system, that will be employed as targets. So far, scripts and binaries are provided with the program to work with MOPAC, a program for *semiempirical* calculations, fitting the properties of a molecular system: energy barriers for the *unimolecular decomposition channels*, geometries and frequencies of the corresponding *transition states*, etc... See Section 9.

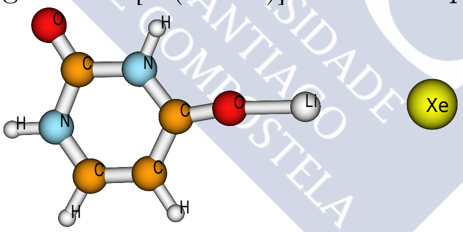
5.3.2 Examples

There are several folders with examples:

- Internal examples

  uracil-example Here the interaction between Xe and the $[Li(Uracil)]^+$complex is studied. Explained in Section 13.

  Figure 5.1: $[Li(Uracil)]^+$- Xe example.

  analytical-example Same as the *uracil-example* but using an analytical expression as potential. Explained in Section 14.

  n_2n_2-example Here the interaction between two nitrogen molecules is studied. A fully custom potential can be implemented using *userpotential.f* file.

- External examples

  Generic external example

  An external-example, with a generic external fit. The given test code supports both *external* and *external bulk* options (section 9.1). This code fits data from file *external.values* –value pairs "(x, f(x))" to fit–, using file *bounds.txt* as upper and lower limits, to a polynomial of degree n.

$$a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$$

The polynomial degree is the number of coefficients minus one. Explained in Section 15.

MOPAC interface

Change and/or set MOPAC_EXECUTABLE and MOPAC_LICENSE in file *external-mopac2009.sh* to run with MOPAC 2012.

- mopac-example. It employs the interface with MOPAC 2009. Source code for the interface tools is in the *src/mopac* folder and explained in the Section 16.
- shepherd-example. It employs the enhanced interface with MOPAC 2009. Explained in Section 17.
- vc-example. As in the previous one, it uses the enhanced interface with MOPAC 2009. Taken from Section 19.

6 Input files

Garbage in, garbage out.

George Fuechsel. IBM instructor.

The input files names are of your choice, except for *job and parameters file*.
The *job and parameters file* was hardcoded as *job.txt*[1].

File 6.1: job.txt. Genetic algorithm parameters and job settings

```
[job]
runs:␣␣␣␣␣␣␣␣␣␣1
evaluations:␣␣5000000
Geometries:␣␣␣moldeni.dat
Energies:␣␣␣␣␣energies.dat
Atom2type:␣␣␣␣atom2types.txt
Bounds:␣␣␣␣␣␣␣bounds.txt
Charges:␣␣␣␣␣␣charges.txt
Potential:␣␣␣␣1
All␣coefficients:␣no
fitting:␣␣␣␣␣␣relative

[parameters]
population:␣␣␣␣␣␣50
crossover␣rate:␣␣0.75
blx_alpha:␣␣␣␣␣␣␣0.5
mutation␣rate:␣␣␣0.1
elitism:␣␣␣␣␣␣␣␣␣yes
tournament␣size:␣5
crossover:␣␣␣␣␣␣␣sbx
mutation:␣␣␣␣␣␣␣␣sigma
sigma:␣␣␣␣␣␣␣␣␣␣␣0.1
```

[1]Defined in *ga.h*

```
direction:       min

[print]
geometries: yes
runs:        yes
```

There are four fixed *sections*, which can be put in any order, have their own parameters, which can also be used in any order; these parameters specify:

parameters These parameters affect the *genetic algorithm* working mode.

job The job to be done.

print Diverse printing options.

analytical Mandatory if an **analytical espression** as potential is chosen. See Section 8, *Specifiying a new potential*.

Each option, including the whole sections, can be avoided[2], but the file *job.txt* itself must be present. In case of omited parameters, the program takes some default values (See table 6.1), so you can write a *job.txt* file like 6.2.

File 6.2: Reduced job.txt.

```
[job]
runs:          1
evaluations:  5000000
Geometries:   moldeni.dat
Energies:     energies.dat
Atom2type:    atom2types.txt
Bounds:       bounds.txt
Charges:      charges.txt
Potential:    1
All coefficients: 0
```

False bool values can be written as "0" or "no". *True bool values* can be written as a "number <>0" or "yes". Some parameters have a set of valid values to choose from. If the chosen parameter is out the set, the default will be taken. Optionally, according to the **potential** value, there could be additional sections to define the *analytical expression* used. Parameters and sections are case-insensitive, but in parameters

[2]Except for the **analytical expression** configuration section if chosen.

names with more than one word whitespace matters! Please, use one space between words.

6.1 Types of jobs

There are two types of jobs. Default is **internal** job, where the potential can be:

- defined by **GAFit** (in *potentials.f* file –File 8.1–, table 6.2).
- user defined (in *potentials.f* or *userpotential.f* –File 8.2– files, sections 8.1.3 and 8.1.4).
- user defined *analytical expression* as potential, section 8.2.

Table 6.1: Job file default value parameters

Section	Parameter	Type	Valid set	Default
parameters				
	population	integer		100
	crossover rate	real		0.75
	crossover	string	{spc, dpc, blax, sbx}	sbx
	blx_alpha	real		0.5
	eta_sbx	real	non negative	2.0
	mutation rate	real		0.1
	mutation	string	{random, sigma}	sigma
	sigma	real		0.1
	integer mutation	string	{random, adjacent}	random
	elitism	bool	{yes, no}	yes
	tournament size	integer		5
	direction	string	{min, max}	min
job				
	type	string	{internal, external, external bulk, external auto}	internal
	internal options			
	test	unsigned integer	0, integer number	0
	runs	integer		1
	evaluations	integer		5000
	geometries	string		geometries.txt
	energies	string		energies.txt
	atom2type	string		atom2type.txt
	bounds	string		bounds.txt
	charges	string		charges.txt
	potential	integer		1
	all coefficients	bool	{yes, no}	no
	fitting	string	{absolute, relative, user}	relative
	auto weights	bool	{yes, no}	no
	external options			
	test	unsigned integer	0, integer number	0
	runs	integer		1
	evaluations	integer		5000
	bounds	string		bounds.txt

Section	Parameter	Type	Valid set	Default
	command	string		./external
	external input	string		external.input
	external fit	string		external.fit
	coefficients	integer		0
print				
	geometries	bool	{yes, no}	yes
	runs	bool	{yes, no}	yes
	ga settings	bool	{yes, no}	no
	analytical	bool	{yes, no}	yes
	auto weights	bool	{yes, no}	yes

An **external** job implies that the user only employs the *genetic algorithm* to fit the parameters and evaluate them by an external program.

6.2 Section [parameters]

The **section [parameters]** contains the genetic algorithm settings.

population Population size

elitism Elistism strategy. Section 4.3.

- no
- yes

tournament size Tournament selection size. Section 4.1.

crossover rate Crossover rate. Section 3.1.3.

blx_alpha BLX-α crossover coefficient

eta_sbx SBX crossover coefficient

crossover Crossover type.

- spc: Single Point Crossover
- dpc: Double Point Crossover
- blax: Blend Alpha Crossover
- sbx: Simulated Binary Crossover

mutation rate Mutation rate. Section 4.2.2.

mutation Mutation type

- random = Random mutation
- sigma = Sigma mutation

sigma Sigma mutation coefficient

integer mutation Mutation operator for integer variables. Section 4.2.2.

- random
- adjacent

direction Search direction

- min: Minimization
- max: Maximization

6.3 Section [job]

This section defines the run parameters for the present job. It also indicates the names of the different files for the calculation.

The job parameters from the *job* section are:

Type type of job: external or internal. In case of external could be:

external Each gene is passed to the external program, one per run.

external bulk All the genes of the same generation are passed to the external program, an entire generation per run, reducing the overall load, speeding up calculations.

external auto **GAFit** is configured by the **external command**. See 9.1.

Test If it is not equal to zero, the integer is used as random seed, breaking the system randomness. This is useful for testing purposes. For as standard job you should use a random number: set to zero this value or do not put anything. The used seed in a job is printed in the output as shown below to recover it when needed .

```
[...]

#seed#1348550422#seed#

[...]
```

Runs Number of runs. Remember to change this setting if you are using **auto weights**, page 52.

Evaluations Number of generations

Geometries Continuous set of **molden** format Cartesian geometries without any empty lines between them.

File 6.3: Geometries file. Molden xyz coordinates

```
␣␣␣␣␣␣␣␣␣␣116
␣␣␣␣␣␣␣␣␣␣␣␣X␣␣␣␣␣␣␣␣␣␣␣Y␣␣␣␣␣␣␣␣␣␣␣Z
␣N␣␣␣-13.694289␣␣␣␣-0.182672␣␣␣␣␣0.000000
␣H␣␣␣-13.299638␣␣␣␣␣0.824476␣␣␣␣␣0.000000
␣C␣␣␣-12.403476␣␣␣␣-0.960776␣␣␣␣␣0.000000
␣H␣␣␣-14.263389␣␣␣␣-0.348152␣␣␣␣-0.831048
␣H␣␣␣-14.263389␣␣␣␣-0.348152␣␣␣␣␣0.831048
␣C␣␣␣-11.316612␣␣␣␣␣0.153002␣␣␣␣␣0.000000
␣H␣␣␣-12.348018␣␣␣␣-1.588139␣␣␣␣-0.892698
␣H␣␣␣-12.348018␣␣␣␣-1.588139␣␣␣␣␣0.892698
␣O␣␣␣-11.719020␣␣␣␣␣1.326881␣␣␣␣␣0.000000
␣...
␣␣␣␣␣␣␣␣␣␣116
␣␣␣␣␣␣␣␣␣␣␣␣X␣␣␣␣␣␣␣␣␣␣␣Y␣␣␣␣␣␣␣␣␣␣␣Z
␣N␣␣␣␣-9.694289␣␣␣␣-0.182672␣␣␣␣␣0.000000
␣H␣␣␣␣-9.299638␣␣␣␣␣0.824476␣␣␣␣␣0.000000
␣C␣␣␣␣-8.403476␣␣␣␣-0.960776␣␣␣␣␣0.000000
␣H␣␣␣-10.263389␣␣␣␣-0.348152␣␣␣␣-0.831048
␣H␣␣␣-10.263389␣␣␣␣-0.348152␣␣␣␣␣0.831048
␣C␣␣␣␣-7.316612␣␣␣␣␣0.153002␣␣␣␣␣0.000000
␣H␣␣␣␣-8.348018␣␣␣␣-1.588139␣␣␣␣-0.892698
␣H␣␣␣␣-8.348018␣␣␣␣-1.588139␣␣␣␣␣0.892698
␣O␣␣␣␣-7.719020␣␣␣␣␣1.326881␣␣␣␣␣0.000000
␣...
␣␣␣␣␣␣␣␣␣␣116
␣␣␣␣␣␣␣␣␣␣␣␣X␣␣␣␣␣␣␣␣␣␣␣Y␣␣␣␣␣␣␣␣␣␣␣Z
␣N␣␣␣␣-6.694289␣␣␣␣-0.182672␣␣␣␣␣0.000000
␣H␣␣␣␣-6.299638␣␣␣␣␣0.824476␣␣␣␣␣0.000000
␣C␣␣␣␣-5.403476␣␣␣␣-0.960776␣␣␣␣␣0.000000
␣H␣␣␣␣-7.263389␣␣␣␣-0.348152␣␣␣␣-0.831048
␣H␣␣␣␣-7.263389␣␣␣␣-0.348152␣␣␣␣␣0.831048
␣C␣␣␣␣-4.316612␣␣␣␣␣0.153002␣␣␣␣␣0.000000
␣H␣␣␣␣-5.348018␣␣␣␣-1.588139␣␣␣␣-0.892698
␣H␣␣␣␣-5.348018␣␣␣␣-1.588139␣␣␣␣␣0.892698
␣O␣␣␣␣-4.719020␣␣␣␣␣1.326881␣␣␣␣␣0.000000
```

```
␣...
```

Energies File with energies and weights for each geometry listed at *geometries file*. It must be in sync with the *geometries file*. Weights are taken into account when the potential is calculated.

File 6.4: Energies file. Energies and weights

```
-0.016881788␣␣1
-0.024242894␣␣1
-0.033981373␣␣1
␣...
```

File 6.5: Energies file. Structure

```
energie_of_first_geometry␣␣first_weight
energie_of_second_geometry␣␣second_weight
energie_of_third_geometry␣␣third_weight
␣...
```

If the option **auto weights** in the **[job]** section is used, each contain the *type*, *tolerance* and *delta* columns needed for the desired automatic weights calculations. See 6.3.

File 6.6: Energies file. Energies with auto weights

```
-0.006436␣1␣1␣.5␣0.5
-0.012603␣1␣1␣␣.5␣0.5
-0.024660␣1␣1␣␣.5␣0.5
...
```

File 6.7: Energies file. Structure of Energies file with auto weights

```
energie_first_geometry␣␣first_weight␣auto␣tolerance␣delta
energie_second_geometry␣␣second_weight␣auto␣tolerance␣delta
energie_third_geometry␣␣third_weight␣auto␣tolerance␣delta
␣...
```

Atom2type File to map atom numbers to type numbers. The first line has the required parameters as integer numbers:

- Number of atoms in *Fragment A*. In this example, 18 (File 6.8).
- Total number of atoms.

The rest of the lines, three columns, specify:

- Atom number. Atom numbering must follow the order given in the *coordinate file*.
- Atom symbol (two character max).
- Atom type number. A positive integer used as a type index.

From these parameters, all the different interactions are calculated. The total number of interactions is obtained from the number of atoms in *Fragment A* times the number of atoms in *Fragment B*. The coefficients of some interactions are repeated: those that correspond to interactions between atoms of the same type.

So, the number of different interactions is just the different atom types in *Fragment A* multiplied by the number of different atom types in *Fragment B*. See 5.3.2 for an example.

File 6.8: Atom2type. Atom to atom types mapping

```
␣18␣116
␣␣␣␣1␣␣␣␣N␣␣␣␣␣␣1
␣␣␣␣2␣␣␣␣H␣␣␣␣␣␣2
␣␣␣␣3␣␣␣␣C␣␣␣␣␣␣3
␣␣␣␣4␣␣␣␣H␣␣␣␣␣␣2
␣␣␣␣5␣␣␣␣H␣␣␣␣␣␣2
␣␣␣␣6␣␣␣␣C␣␣␣␣␣␣4
␣␣␣␣7␣␣␣␣H␣␣␣␣␣␣5
␣␣␣␣8␣␣␣␣H␣␣␣␣␣␣5
␣␣␣␣9␣␣␣␣O␣␣␣␣␣␣6
␣...
```

File 6.9: Atom2type. Structure

```
␣AtmFrA␣␣AtmTotal
␣␣␣␣AtomNumber1␣␣␣␣AtomSymbol1␣␣␣␣␣␣AtomTypeNumber1
␣␣␣␣AtomNumber2␣␣␣␣AtomSymbol2␣␣␣␣␣␣AtomTypeNumber2
␣␣␣␣AtomNumber3␣␣␣␣AtomSymbol3␣␣␣␣␣␣AtomTypeNumber3
␣␣␣␣AtomNumber4␣␣␣␣AtomSymbol4␣␣␣␣␣␣AtomTypeNumber2
␣␣␣␣AtomNumber5␣␣␣␣AtomSymbol5␣␣␣␣␣␣AtomTypeNumber2
␣␣␣␣AtomNumber6␣␣␣␣AtomSymbol6␣␣␣␣␣␣AtomTypeNumber4
␣␣␣␣AtomNumber7␣␣␣␣AtomSymbol7␣␣␣␣␣␣AtomTypeNumber5
␣␣␣␣AtomNumber8␣␣␣␣AtomSymbol8␣␣␣␣␣␣AtomTypeNumber5
␣␣␣␣AtomNumber9␣␣␣␣AtomSymbol9␣␣␣␣␣␣AtomTypeNumber6
␣...
```

This file can be created with the *needle* tool. See 12.1, page 115.

Bounds The variation range of the coefficients is specified here. The third column specifies if the coefficient will be treated as a real

(0) or integer (1) number. The number of lines depends on **All coefficients** parameter –**[job]** section– and the chosen **potential** in *job file*.

It could be edited by the *bedit* tool. See 12.2, page 115.

File 6.10: Bounds. Variation range of the coefficients

```
TEXT_OR_EMPTY
→−100→100.→0
→0.→␣100.0→0
→−1500.→5000.0→0
→3.5→5.5→0
```

File 6.11: Bounds. All Coefficients=0. Structure

```
TEXT_OR_EMPTY_LINE
1stMinimum→1stMaximum→1stType
2ndMinimum→2ndMaximum→2ndType
3rdMinimum→3rdMaximum→3rdType
4thMinimum→4thMaximum→4thType
 ...
nthMinimum→nthMaximum→nthType
```

File 6.12: Bounds. All Coefficients<>0. Structure

```
TEXT_OR_EMPTY_LINE␣−␣interaction␣1␣coefficients␣set
1stMinimum→1stMaximum→1stType
2ndMinimum→2ndMaximum→2ndType
3rdMinimum→3rdMaximum→3rdType
4thMinimum→4thMaximum→4thType
 ...
nthMinimum→nthMaximum→nthType
TEXT_OR_EMPTY_LINE␣␣−␣interaction␣2␣coefficients␣set
1stMinimum→1stMaximum→1stType
2ndMinimum→2ndMaximum→2ndType
3rdMinimum→3rdMaximum→3rdType
4thMinimum→4thMaximum→4thType
 ...
nthMinimum→nthMaximum→nthType
 ....
TEXT_OR_EMPTY_LINE␣␣−␣interaction␣N␣coefficients␣set
1stMinimum→1stMaximum→1stType
2ndMinimum→2ndMaximum→2ndType
3rdMinimum→3rdMaximum→3rdType
4thMinimum→4thMaximum→4thType
 ...
nthMinimum→nthMaximum→nthType
```

The text line between each interaction is skipped when reading bounds. Using the **bedit** tool, labels can be written automatically as in File 6.13.

Note that BLX-α and SBX revert to SPC crossover for integer coefficients.

File 6.13: Bounds file written with the **bedit** tool

```
TYPE␣1:␣C(1)−Xe(14)
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+0.00000␣␣␣+1000000.00000␣␣␣0
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+0.00000␣␣␣␣␣␣␣␣+10.00000␣␣␣1
␣␣␣␣␣␣␣␣␣␣␣␣−1500.00000␣␣␣␣␣␣␣␣␣+0.00000␣␣␣0
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+4.00000␣␣␣␣␣␣␣␣␣+8.00000␣␣␣0
TYPE␣2:␣N(2)−Xe(14)
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+0.00000␣␣␣+1000000.00000␣␣␣0
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+0.00000␣␣␣␣␣␣␣␣+10.00000␣␣␣0
␣␣␣␣␣␣␣␣␣␣␣␣−1500.00000␣␣␣␣␣␣␣␣␣+0.00000␣␣␣0
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+4.00000␣␣␣␣␣␣␣␣␣+8.00000␣␣␣0
TYPE␣3:␣C(3)−Xe(14)
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+0.00000␣␣␣+1000000.00000␣␣␣0
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+0.00000␣␣␣␣␣␣␣␣+10.00000␣␣␣0
␣␣␣␣␣␣␣␣␣␣␣␣−1500.00000␣␣␣␣␣␣␣␣␣+0.00000␣␣␣0
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+4.00000␣␣␣␣␣␣␣␣␣+8.00000␣␣␣0
TYPE␣4:␣N(4)−Xe(14)
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+0.00000␣␣␣+1000000.00000␣␣␣0
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+0.00000␣␣␣␣␣␣␣␣+10.00000␣␣␣0
␣␣␣␣␣␣␣␣␣␣␣␣−1500.00000␣␣␣␣␣␣␣␣␣+0.00000␣␣␣0
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+4.00000␣␣␣␣␣␣␣␣␣+8.00000␣␣␣0
TYPE␣5:␣C(5)−Xe(14)
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+0.00000␣␣␣+1000000.00000␣␣␣0
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+0.00000␣␣␣␣␣␣␣␣+10.00000␣␣␣0
␣␣␣␣␣␣␣␣␣␣␣␣−1500.00000␣␣␣␣␣␣␣␣␣+0.00000␣␣␣0
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+4.00000␣␣␣␣␣␣␣␣␣+8.00000␣␣␣0
```

Charges This file must include partial charges (in a.u.) for all atoms when potential 4 is selected (see Table 6.2). Partial charges may be specified for atom types (File 6.14 and 6.15).

The types must be the same as those from *Atom2type file*. See 6.8. It depends on the chosen potential. Note that the type number can be any one, as long as they are different between them.

The file can be edited by the *bedit* tool, and generated from *needle*. See 12.1 and 12.2.

File 6.14: Charges. Type to charges mapping

```
␣␣␣␣1␣␣␣0.027
␣␣␣␣2␣␣␣0.113
␣␣␣␣3␣␣−0.057
␣␣␣␣4␣␣−0.01
␣␣␣␣5␣␣␣0.001
␣...
```

File 6.15: Charges. Structure

```
␣␣␣␣AtomType1␣␣␣Charge1
␣␣␣␣AtomType2␣␣␣Charge2
␣␣␣␣AtomType3␣␣␣Charge3
␣␣␣␣AtomType4␣␣␣Charge4
␣␣␣␣AtomType5␣␣␣Charge5
␣...
```

Potential An integer, that specifies the chosen potential as defined in *potentials.f* source fortran file.

Table 6.2: Potential values from *potentials.f* code file

Value	Coefficients	Potential
-1	any	any user defined potential
0	any	any analytical expression as potential
1	4	$V = Ae^{-Br} + \frac{C}{r^D}$
2	6	$V = Ae^{-Br} + \frac{C}{r^D} + \frac{E}{r^F}$
3	8	$V = Ae^{-Br} + \frac{C}{r^D} + \frac{E}{r^F} + \frac{G}{r^H}$
4	3 plus charges	$V = A\left[\left(\frac{B}{r}\right)^{12} - \left(\frac{B}{r}\right)^{6}\right] + 332.0532\frac{q_iq_j}{r}$

Table 6.2 shows the available potentials where:

r is the distance between the two atoms whose interaction is calculated

332.0532 A conversion factor

A, B, C, D, E, F, G The coefficients to be fitted

q_i,q_j Charges

All coefficients Drives the reading mode of *Bounds file*. If this variable is not set, it reads a sequence of coefficients for only one interaction, and then, the program assumes all the interactions have the same bounds. If it is set, it reads the bounds for all the coefficients. See Files 6.10,6.11 and 6.12

Fitting Can be absolute or relative (see below).

absolute

$$\sum\left[(\mathbf{v}_i - \mathbf{Pot}(i))^2\ \mathbf{Weight}(i)\right]$$

relative

$$\sum\left[\frac{(\mathbf{v}_i - \mathbf{Pot}(i))^2}{\mathbf{v}_i^2}\mathbf{Weight}(i)\right]$$

user this option redirects to a user defined fitting function in the *userpotential.f* file. See 8.1.4 section.

Auto weights Boolean parameter. Activate *automatic weights*: At the end of every **run**, the potential for **each geometry** is calculated and compared with the *reference value*. If the difference is larger than *tolerance*, the weight is increased by *delta* as detailed below.

In this case, each **energies** file line must contain:

energy The energy of the geometry

weight The initial weight

type The type of check performed.

0 None, a 0 must be typed.

1 Relative. Weight is incremented by *delta* if *tolerance* is less than the relative value between calculated energy and the energy.
$\frac{[\mathbf{Energy}-\mathbf{Calculated}]^2}{\mathbf{Energy}^2}$

2 Absolute. Weight is incremented by *delta* if *tolerance* is less than the absolute value between calculated energy and the energy.
$|\mathbf{Energy} - \mathbf{Calculated}|$

tolerance The tolerance. A real number.

delta The value to increment weight. A real number.

See page 47 for **energies** file details. Note in file example 6.16 the *runs* and *evaluations* parameters. In this case, 10 times (10 runs) the checks are performed at the end of each run.

File 6.16: Job settings with auto weights

```
[job]
runs:→→␣100
evaluations:→␣50000
auto␣weights:␣yes
```

Command External job, the **command** to be run.

File 6.17: External job settings

```
[job]
runs:␣␣␣␣␣␣␣␣␣␣␣␣1
evaluations:␣␣␣␣500000
type:␣␣␣␣␣␣␣␣␣␣␣␣external␣bulk
command:␣␣␣␣␣␣␣␣␣external.sh
coefficients:␣␣␣5
external␣input:␣external.input
external␣fit:␣␣␣external.fit
bounds:␣␣␣␣␣␣␣␣␣␣bounds.txt
```

External input External job, the input for the external **command**, File 6.18. Here **GAFit** writes a coefficient vector to be evaluated by the external command. If the option *external bulk* is chosen, all the coefficients for a complete generation are passed, separating each one by a blank line, File 6.19.

File 6.18: External input

```
4.894146
0.013449
−6.092118
−0.003859
1.216052
```

File 6.19: External bulk input

```
4.894146
0.013449
−6.092118
−0.003859
1.216052

4.894410
0.013449
−6.091149
−0.003859
1.215979

4.894332
0.013449
```

```
−6.091579
−0.003859
1.216001
...
```

External fit External job, the evaluation of the coefficients returned to **GAFit**. If the option *external bulk* is used, a complete set must be returned. Examples: 6.20 and 6.21.

File 6.20: External fit: one individual fit

```
25647.561250
```

File 6.21: External bulk fit: entire generation fit

```
25647.561250
3.000000
13.011250
6417.651250
3.000000
3.000000
3.000000
18.055000
13.011250
3.000000
25647.561250
7012.161250
4715.805000
[...]
```

Coefficients Number of coefficients to be considered in a external job.

6.4 Section [print]

This section controls how much is printed.

Geometries This parameter controls if the read geometries are printed on standard output. See 7.

Runs This parameter controls if the intermediate results are printed on standard output. See 7.

GA settings Prints genetic algorithm settings.

Analytical Prints output from **analytical expressions** routines.

Auto weights Prints auto weights checks between runs.

6.5 Section [coefficient names]

GAFit coefficient names default to the sequence {A, B, ..., Z, AA, AB, ..., BA, ..., ..., AAA, ...} names and so on. If you want to use your own ones, write a new section **[coefficient names]** with each name in a line. You must specify at least the same number of lines as the number of coefficients to be used; if not, **GAFit** stops. An example can be viewed in File 9.1.

These routines are also used internally to no related tasks like to name temporary files.

6.6 Section [analytical]

The reader is referred to Section 8.2, where this is explained in detail.

7 Output files

On two occasions I have been asked, "Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?" ... I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.

Charles Babbage

standard output The standard output is used to print job results. An example of the output is below. Some of the output is controlled by options into the **[print]** section. See 6.4.

```
Build: 2122

Settings for job
-----------------------------
Geometries:[coord.molden]
Energies:[energies]
Atom2type:[atom2type.txt]
Bounds:[bounds.txt]
Charges:[charges.txt]
Potential read: 2
All coefficients: no, Read and repeat subset
Fitting: absolute

Print options:
     geometries yes
     runs yes
     ga settings yes
     analytical yes

...now reading data
```

```
Different interaction types: 13,
        with 4 coefficients each,
        so, we need a 52 elements vector.
        Choosen potential=2
Fragment A atoms 13, Fragment B atoms 1
13 types in fragment A, 1 in Fragment B
13 different interactions

Reading bounds for 4 coefficients

         1      A        +0.00000 -  +1000000.00000 (real)
         2      B        +0.00000 -       +10.00000 (real)
         3      C     -1500.00000 -        +0.00000 (real)
         4      D        +4.00000 -        +8.00000 (real)


Creating a 52 bounds vector...


 52 BOUNDS VECTOR
======================

INTERACTION TYPE 1
----------------------
C(1)-Xe(14)
        Coefficients:
            1 A        +0.00000 -  +1000000.00000 (real)
            2 B        +0.00000 -       +10.00000 (real)
            3 C     -1500.00000 -        +0.00000 (real)
            4 D        +4.00000 -        +8.00000 (real)

INTERACTION TYPE 2
----------------------
N(2)-Xe(14)
        Coefficients:
            5 A        +0.00000 -  +1000000.00000 (real)
            6 B        +0.00000 -       +10.00000 (real)
            7 C     -1500.00000 -        +0.00000 (real)
            8 D        +4.00000 -        +8.00000 (real)
[...]

#seed#1351155784#seed#

run 1
50      1       2290090179083.717285156250      9.635496575595e+03
200     4       15290009321005.964843750000     3.633024843375e+03
350     7       7630038244343.797851562500      3.275150020838e+03
500     10      1919724358228.594482421875      2.661055597153e+03
650     13      42486949205.559173583984        2.661055597153e+03
800     16      9432708886029.101562500000      2.374854855353e+03

[...]

#
#Results
#

INTERACTION TYPE 1
----------------------
C(1)-Xe(14)
        Coefficients:
            1 A   +560362.1769717000
            2 B        +3.1916051065
            3 C     -1433.0179704333
            4 D        +6.5900203399

INTERACTION TYPE 2
```

```
----------------------
N(2)-Xe(14)
        Coefficients:
            5 A   +985812.3248912474
            6 B        +4.9449172106
            7 C     -1500.0000000000
            8 D        +4.9264827631
[...]
#
#Evaluation
#
#Geometry     Energy        Calculated      Difference
#========     ======        ==========      ==========
     1   -0.006436000000  -0.037892707550  +488.76%
     2   -0.012603000000  -0.061285488548  +386.28%
     3   -0.024660000000  -0.105859323488  +329.28%
     4   -0.053662000000  -0.199490133092  +271.75%
     5   -0.151027000000  -0.422816767398  +179.96%
     6   -0.208324000000  -0.521290713046  +150.23%
[...]
```

If the **runs** option is set in section **[print]**, like above, the number of the current run is printed –just below the random number seed–, and also four columns indicating:

- The number of individuals evaluated up to now, *800* in the last line before *#Results*.
- The current generation, *16* in the same line.
- The average objective function of the current population: *94327088886029.101562500000*.
- And the objective function best value up to now: *2.374854855353e+03*.

When an *analytical expression* is selected, the names of the coefficients selected by the user are printed .

```
[...]

Settings for job
-----------------------------
Geometries:[coord.molden]
Energies:[energies.txt]
Atom2type:[atom2type.txt]
Bounds:[bounds.txt]
Charges:[charges.txt]
Potential read: Analytical expression
All coefficients: no, Read and repeat subset
Fitting: absolute

Print options:
        geometries yes
        runs yes
        ga settings yes
        analytical yes

Analytical expression
-----------------------------
expression name: "potential 5"
potential:       pot
distance:        dist
coefficients:    aaa, bbb, c1, c2, d1, d2, e1, e2

Expression found:

        v1  =  aaa * exp (  -bbb * dist ) ;
        v2  =  c1 / pow ( dist , c2 ) ;
        v3  =  d1 / dist ** d2 ;
        v4  =  e1 / dist ^ e2 ;
        pot  =  v1 + v2 + v3 + v4@

        Variables found in expression: v1 aaa bbb dist v2 c1 c2 v3 d1 d2 v4 e1 e2 pot
        Expression code OK
        pot index 13
        dist index 3
        8 coefficients found
...now reading data
          14
 C    0.0000000       0.0000000       0.0000000
 N    0.0000000       0.0000000       1.3545491
 C    1.1521430       0.0000000       2.1275020

[...]

Creating a 104 bounds vector...


 104 BOUNDS VECTOR
======================

INTERACTION TYPE 1
----------------------
C(1)-Xe(14)
        Coefficients:
            1        aaa        +0.00000 -  +1000000.00000 (real)
            2        bbb        +0.00000 -       +10.00000 (integer)
            3         c1     -1500.00000 -        +0.00000 (real)
            4         c2        +4.00000 -        +8.00000 (integer)
            5         d1        +0.00000 -  +1000000.00000 (real)
            6         d2        +0.00000 -       +10.00000 (integer)
            7         e1     -1500.00000 -        +0.00000 (real)
            8         e2        +4.00000 -        +8.00000 (integer)

INTERACTION TYPE 2
----------------------
N(2)-Xe(14)
        Coefficients:
            9        aaa        +0.00000 -  +1000000.00000 (real)
           10        bbb        +0.00000 -       +10.00000 (integer)
           11         c1     -1500.00000 -        +0.00000 (real)
           12         c2        +4.00000 -        +8.00000 (integer)
           13         d1        +0.00000 -  +1000000.00000 (real)
           14         d2        +0.00000 -       +10.00000 (integer)
           15         e1     -1500.00000 -        +0.00000 (real)
           16         e2        +4.00000 -        +8.00000 (integer)

[...]

run 1
50      1       62550614237885.890625000000     1.415158910538e+10
200     4       1319338879864.184570312500      2.876907390518e+08
```

```
[...]

#
#Results
#

INTERACTION TYPE 1
----------------------
C(1)-Xe(14)
        Coefficients:
            1        aaa   +105671.4794050544
            2        bbb        +4.0000000000
            3         c1     -1085.2285321549
            4         c2        +6.0000000000
            5         d1   +974559.2805542682
            6         d2        +9.0000000000
            7         e1      -990.8034563390
            8         e2        +6.0000000000

INTERACTION TYPE 2
----------------------
N(2)-Xe(14)
        Coefficients:
            9        aaa   +680324.6698314144
           10        bbb        +3.0000000000
           11         c1      -204.2193705419
           12         c2        +4.0000000000
           13         d1        +0.0000000000
           14         d2        +7.0000000000
           15         e1     -1178.8818235065
           16         e2        +4.0000000000
[...]
```

best.txt This file contains the best set of coefficients. It is updated every time **GAFit** finds a better set, and it can be used by **fitview** -see 12.3-to obtain the coefficient values.

NOTE: This file is NOT loaded at the beginning of any run, so it can be overwritten when a new run begins if you do not save it beforehand.

7.1 Other output files

Other intermediate output files are:

- stats.txt

7.2 Using TAGS

You can also use a command line **tag** to run multiple **GAFit** processes changing the output names, as stated in Section 5.3.

```
$ gafit TAG
```

The file names are changed inserting the **tag** before the **.txt** suffix:

- best.TAG.txt
- stats.TAG.txt

You cannot use TAGS with **external potentials**.

8 Specifiying a new interaction potential

Simplicity is the ultimate sophistication.

Apple II pc slogan, 1977

Besides the interaction potentials implemented in this code –See Table 6.2–, the user can specify a new potential to fit the interaction energies of the system. The new potential can be introduced by:

- adding it in the file *potentials.f*
- modifying the file *userpotential.f* using it as a template
- writing an *analytical expression*.

8.1 Modifiying potentials.f and userpotential.f

8.1.1 VGLOBALES module

You can use the variables exported by the ***VGLOBALES*** module in addition to your own variables from the ***USERDATA*** module to customize your potential or your fitting function. These are shown in Table 8.1.

8.1.2 Interface subroutines and functions

For an easy customization, some functions and subroutines are provided in addition to the module ***VGLOBALES***.

Table 8.1: Module VGLOBALES variables

variable	type	dimension	comments
r	double precision	(geometries, nprox, nsam)	Calculated interatomic distances for each interaction
v	double precision	geometries	Potential energy for each geometry. Read from energies file
w	double precision	geometries	Weights. Read from energies file
wdelta	double precision	geometries	Delta for each weight. Read from energies file
wtol	double precision	geometries	Tolerance. Read from energies file
wtype	integer	geometries	Type of weight. Read from energies file
q	double precision	natom	Charges. Read from charges file.
geometries	integer	-	Number of geometries
nprox	integer	-	Number of atoms in fragment A
nsam	integer	-	Number of atoms in fragment B
natom	integer	.	Number total of atoms
ptypes	integer	-	Different types of atoms in fragment A
stypes	integer	-	Different types of atoms in fragment B
potential	integer	-	Type of potential
interactions	integer	-	Number of different interactions
coefficients	integer	-	Number of coefficients
charges	logical	-	If charges file is needed
autoweights	logical	-	If autoweights is active
atom	character*2	natom	Two character atom labels

ix function

The function ***ix(i,j,k)*** organizes the different coefficients into the coefficient vector.

k is the index of a given coefficient, i.e.: k=1 means A, k=2 means B, etc. **k** ranges from 1 to the number of **coefficients**

i, j are the atoms that define a given interaction for which the coefficients are defined.

Atom **i** belongs to *fragment A* and **j** belongs to *fragment B*. The atoms of *Fragment A* range from 1 to **nprox**, and those of *fragment B* range from **nprox**+1 to **natom**. See also the *needle* tool output, page 116.

coordinates subroutine

The ***coordinates(geo,atom,x,y,z)*** subroutine can access the Cartesian coordinates.

geo is the geometry index, ranging from 1 to **geometries**

atom the atom index in the geometry, ranging from 1 to **natom**

x, y, z the coordinates returned by subroutine.

8.1.3 Adding a new potential to potentials.f

Introducing a new potential in the program implies to implement it into *potentials.f* –File 8.1–, to modify **setcoefs** (line 3), **getcharges** (line 28), **potRouter** (line 51) and **curRouter** (line 74) functions, and to write the corresponding potential functions. Finally, the program has to be recompiled.

File 8.1: potentials.f

```fortran
 1 c POTENTIALS
 2 c sets the number of coefs required by potential
 3 c
 4       integer function setcoefs(potential)
 5       implicit none
 6       integer potential
 7       integer angetncoefs
 8       integer usetcoefs
 9       external angetncoefs
10       if (potential .eq. -1) then
11         setcoefs=usetcoefs()
12       else if (potential .eq. 0) then
13         setcoefs=angetncoefs()
14       else if (potential .eq. 1) then
15         setcoefs=4
16       else if (potential .eq. 2) then
17         setcoefs=6
18       else if (potential .eq. 3) then
19         setcoefs=8
20       else if(potential.eq.4) then
21         setcoefs=2
22       else
23         stop 'setcoefs: not implemented'
24       endif
25       end
26
27 c if a charge file is needed
28 c
```

```
      logical function getcharges(potential)
      implicit none
      integer potential
      logical ugetcharges
      if (potential .eq. -1) then
        getcharges=ugetcharges()
      else if (potential .eq. 0) then
        getcharges=.false.
      else if (potential .eq. 1) then
        getcharges=.false.
      else if (potential .eq. 2) then
        getcharges=.false.
      else if (potential .eq. 3) then
        getcharges=.false.
      else if(potential.eq.4) then
       getcharges=.true.
      else
        stop 'getcharges: not implemented'
      endif
      end

c Potential Router, route calculations to the desired potential
c
      subroutine  potRouter(geo,x,nmax,vpot)
      use vglobales
      integer nmax,geo
      double precision vpot, x(nmax)
      if (potential .eq. -1) then
        call userpot(geo,x,nmax,vpot)
      else if (potential .eq. 0) then
        call pot0(geo,x,nmax,vpot)
      else if (potential .eq. 1) then
        call pot1(geo,x,nmax,vpot)
      else if (potential .eq. 2)then
        call pot2(geo,x,nmax,vpot)
      else if (potential .eq. 3) then
        call pot3(geo,x,nmax,vpot)
      else if(potential .eq.4) then
        call pot4(geo,x,nmax,vpot)
      else
        stop 'not implemented potential'
      endif
      end

c Curve Router, route calculations to the desired potential
c
      subroutine  curRouter(d,atom1,atom2,x,nmax,vpot)
      use vglobales
      integer nmax,atom1,atom2,index
      double precision vpot, x(nmax),d
      double precision analytical,userv,v1,v2,v3,v4
      integer ix
      if (potential .eq. -1) then
```

```
            vpot=userv(d,atom1,atom2,x,nmax)
          else if (potential .eq. 0) then
            index=ix(atom1,atom2,1)
            vpot=analytical(d,index,x)
          else if (potential .eq. 1) then
            vpot=V1(d,atom1,atom2,x,nmax)
          else if (potential .eq. 2)then
            vpot=V2(d,atom1,atom2,x,nmax)
          else if (potential .eq. 3) then
            vpot=V3(d,atom1,atom2,x,nmax)
          else if(potential .eq.4) then
            vpot=V4(d,atom1,atom2,x,nmax,q(atom1),q(atom2))
          else
            stop 'not_implemented_potential'
          endif
          end

c Now, each potential calculation down from here.

c       0--------analytical----------------
        subroutine pot0(geo,x,nmax,vpot)
        use vglobales
        integer nmax,geo,i,j,k,index
        double precision d,vpot,analytical
        external analytical
        double precision X(nmax)
        integer ix
        vpot=0.0d0
        do i=1,nprox
         do j=1,nsam
         k=j+nprox
         d=r(geo,i,j)
         index=ix(i,k,1)
         vpot=vpot+analytical(d,index,x)
         enddo
        enddo
        return
        end

c       1-----------------------------------
        subroutine pot1(geo,x,nmax,vpot)
        use vglobales
        integer nmax,geo,i,j,k
        double precision d,vpot,V1
        double precision X(nmax)
        vpot=0.0d0
        do i=1,nprox
         do j=1,nsam
         k=j+nprox
         d=r(geo,i,j)
         vpot=vpot+V1(d,i,k,x,nmax)
         enddo
        enddo
```

```
      return
      end

      FUNCTION V1(r,i,j,x,m)
      implicit none
      integer i,j,m,ix
      dimension x(m)
      double precision x,r,a,b,c,d,v1
      A=x(ix(i,j,1))
      B=x(ix(i,j,2))
      C=x(ix(i,j,3))
      D=x(ix(i,j,4))
      V1=A*EXP(-B*R)+C/R**D
      RETURN
      END

c     2-------------------------------------------------
      subroutine pot2(geo,x,nmax,vpot)
      use vglobales
      integer nmax,geo,i,j,k
      double precision d,vpot,V2
      double precision X(nmax)
      vpot=0.0d0
      do i=1,nprox
       do j=1,nsam
       k=j+nprox
       d=r(geo,i,j)
       vpot=vpot+V2(d,i,k,x,nmax)
       enddo
      enddo
      return
      end

      FUNCTION V2(r,i,j,x,m)
      implicit none
      integer i,j,m,ix
      dimension x(m)
      double precision x,r,a,b,c,d,e,f,v2
      A=x(ix(i,j,1))
      B=x(ix(i,j,2))
      C=x(ix(i,j,3))
      D=x(ix(i,j,4))
      E=x(ix(i,j,5))
      F=x(ix(i,j,6))
      V2=A*EXP(-B*R)+C/R**D+E/R**F
      RETURN
      END


c     3-------------------------------------------------
      subroutine pot3(geo,x,nmax,vpot)
      use vglobales
      integer nmax,geo,i,j,k
      double precision d,vpot,V3
      double precision X(nmax)
      vpot=0.0d0
      do i=1,nprox
       do j=1,nsam
       k=j+nprox
       d=r(geo,i,j)
       vpot=vpot+V3(d,i,k,x,nmax)
       enddo
      enddo
      return
      end

      FUNCTION V3(r,i,j,x,m)
      implicit none
      integer i,j,m,ix
      dimension x(m)
      double precision x,r,a,b,c,d,e,f,g,h,v3
      A=x(ix(i,j,1))
      B=x(ix(i,j,2))
      C=x(ix(i,j,3))
      D=x(ix(i,j,4))
      E=x(ix(i,j,5))
      F=x(ix(i,j,6))
      G=x(ix(i,j,7))
      H=x(ix(i,j,8))
      V3=A*EXP(-B*R)+C/R**D+E/R**F+G/R**H
      RETURN
      END

c     4------------------------------------------
      subroutine pot4(geo,x,nmax,vpot)
      use vglobales
      integer nmax,geo,i,j,k
      double precision d,vpot,V4
      double precision X(nmax)
      vpot=0.0d0
      do i=1,nprox
       do j=1,nsam
       k=j+nprox
       d=r(geo,i,j)
       vpot=vpot+V4(d,i,k,x,nmax,q(i),q(j))
       enddo
      enddo
      return
      end

      FUNCTION V4(r,i,j,x,m,qi,qj)
      implicit none
      integer i,j,m,ix
      dimension x(m)
      double precision x,r,a,b
      double precision v4,qi,qj
      A=x(ix(i,j,1))
      B=x(ix(i,j,2))
      V4=A*((B/R)**12-(B/R)**6)+qi*qj/R*332.0532d0
      RETURN
      END
```

setcoefs returns the number of coefficients used per potential.

getcharges returns **true** if the formula needs the charges file, if not **false**.

potRouter selects the function to calculate.

curRouter is used by **fitview** to plot two body interactions.

Some other variables are loaded into functions via the **use** statement or they are available via interface functions or subroutines –see 8.1.1–.

8.1.4 Changing userpotential.f

The user potential file is a template. Using *potential=-1* in the **[job]** section, the program understands that it has to employ this file. The included template (File 8.2) contains, as an example, potential number 1 (see 6.2 table). To implement a new potential function you only have to:

- change line number 34, the number of coefficients.
- change line 44 if the charges file is needed.
- change lines from 86 to 91 to code the potential formula.
- additionally, you can specify here a *user fitting function* –page 52–.
- if you need to share or load some variables, you can use the **USER-DATA** module.

You can use the **function ix** (see page 64) to access individual coefficients or use the **subroutine coordinates** to access individual atom coordinates.

File 8.2: userpotential.f

```
c USER POTENTIAL
c please change as needed


c USER DATA MODULE

      module userdata
      implicit none
      save
c v----------CHANGE-ME---------------------------v
c define your variables here

c ^----------CHANGE-ME---------------------------^
      end module userdata


c USERREAD SUBROUTINE

      subroutine userread()
      use userdata
c v----------CHANGE-ME---------------------------v
c your code to read external files here


c ^----------CHANGE-ME---------------------------^
      end


C USETCOEFS FUNCTION

      integer function usetcoefs()
c here specify the number of coefficients
c v----------CHANGE-ME---------------------------v
      usetcoefs=4
c ^----------CHANGE-ME---------------------------^
      end


c UGETCHARGES FUNCTION

      logical function ugetcharges()
c specify if you need a charges file
c v----------CHANGE-ME---------------------------v
      ugetcharges=.false.
c ^----------CHANGE-ME---------------------------^
      end

c USERPOT SUBROUTINE

      subroutine userpot(geo,x,nmax,vpot)
      use vglobales
c -----------------------------------------------
```

```
c to use your external data
      use userdata
c ---------------------------------------------
      integer nmax,geo,i,j,k
      double precision d,vpot,userv
      double precision X(nmax)
c v---------CHANGE-ME-IF-NEEDED------------v
      vpot=0.0d0
c note: here all interactions are calculated
      do i=1,nprox
       do j=1,nsam
       k=j+nprox
       d=r(geo,i,j)
       vpot=vpot+userv(d,i,k,x,nmax)
       enddo
      enddo
c ^---------CHANGE-ME-IF-NEEDED------------^
      return
      end


c FUNCTION USER POTENTIAL
c  write userv using ix function to access
c  individual coefficients.
c  use CALL coordinates(geometry,atom,x,y,z)
c  to access individual coordinates.

      double precision FUNCTION userv(r,i,j,x,m)
      implicit none
      integer i,j,m,ix
      dimension x(m)
c note: here ONE interaction is calculated
c v---------CHANGE-ME---------------------v
      double precision x,r,a,b,c,d
      A=x(ix(i,j,1))
      B=x(ix(i,j,2))
      C=x(ix(i,j,3))
      D=x(ix(i,j,4))
      userv=A*EXP(-B*R)+C/R**D
c ^---------CHANGE-ME---------------------^
      RETURN
      END


c USER FITTING FUNCTION
c  write here the user fitting function
c  if you only need the fitting function
c  leave the line "call potRouter..." unchanged
c  and change the line "userfitting=..." with your
c  fitting function.
c  if you have a userv function (above this), you can
c  use it here, or access it via potRouter

```

```
      double precision function userfitting(x,m,geo)
      use vglobales
      use userdata
      double precision x,vpot
      integer m,geo
      dimension x(m)
c v-------------CHANGE-ME-------------------------------v
      call potRouter(geo,x,m,vpot)
      userfitting=(v(geo)-vpot)*(v(geo)-vpot)
c ^-------------CHANGE-ME-------------------------------^
      return
      end
```

The subroutine *userread* is called after reading the job settings and associated data, so it can be used to load data to the *userdata module* for later use in the user potential function or subroutine (*userv* or *userpot*). A complete example can be found in the folder *n2n2-example*.

8.2 Analytical expression

If you do not want to write code, the potential function can be introduced as an **analytical expression** just by writing an *analytic expression* or *analytic formulae* in a file. Note that an **analytical expression** runs about ten times slower compared with the above compiled version.

The *analytical expression* introduced by the user must be checked, compiled to intermediate code, and finally, run in a virtual Floating Point Unit (FPU) with the correct variables loaded. The number of coefficients per interaction is automatically counted from the expression.

First of all, you have to select *potential: 0* in the **[job]** section, and a mandatory **[analytical]** section must be fulfilled with each of its parameters. The table 8.2 shows and explains them.

An example can be seen in File 8.3. It also shows different forms to express the potential.

As you see in File 8.3, "potential 5" is selected so the section **[potential 5]** contains the expression to be calculated.

The **distance** variable is named "dist", and **potential** "pot". The **coefficients** are: "aaa", "bbb", "c1", "c2", "d1", "d2", "e1" and "e2".

The expression is divided in five parts, using intermediate variables "v1", "v2", "v3" and "v4" to hold partial calculations. These variables are automatically defined by the compiler algorithm. In fact, this potential

Table 8.2: Analyltical potential parameters

Section	Parameter	Type	comments
analytical			
	expression	string	Specifies a **whole section** where the expression is defined
	potential	string	Variable used for potential
	distance	string	Variable used for distance between atoms
	coefficients	string	Comma-separated value lists of coefficients used in expression. These, taking in account interactions, build the vector optimized by **GAFit**

is *number 3 standard potential* defined in table 6.2.

The section **[potential 3]** shows a different way to use the same potential. Section **[potential 1]** and **[potential 2]** are the first and second *standard potentials* from table 6.2.

File 8.3: job.txt. Analytical expression options

```
[parameters]
population:⟶␣50
crossover␣rate:→␣0.75
blx_alpha:⟶␣0.5
mutation␣rate:→␣0.1
elitism:⟶␣yes
tournament␣size:␣5
crossover:␣␣␣␣␣␣␣sbx
mutation:␣␣␣␣␣␣␣␣sigma
sigma:→⟶␣0.1
direction:⟶␣min

[job]
runs:→⟶␣1
evaluations:⟶␣5000000
Geometries:␣␣␣coord.molden
Energies:␣␣energies.txt
Atom2type:␣atom2types.txt
Bounds:␣bounds.txt
Charges:␣charges.txt
Potential:␣0
All␣coefficients:␣no
```

```

[print]
geometries: yes
runs: yes
ga settings: yes
analytical: yes

[analytical]
expression: potential 5
distance: dist
potential: pot
coefficients: aaa, bbb, c1, c2, d1, d2, e1, e2

[potential 1]
V=A*EXP(-B*R)+C/R**D;

[potential 2]
v=a*exp(-b*r)+c/r**d+e/r**f;

[potential 3]
enum = 27.182818284e-1 ;
v1 = aaa * pow ( enum , -bbb * dist ) ;
v2 = c1 / pow ( dist , c2 ) ;
v3 = d1 / dist ** d2 ;
v4 = e1 / dist ** e2 ;
pot = v1 + v2 + v3 + v4

[potential 5]
v1 = aaa * exp (  -bbb * dist ) ;
v2 = c1 / pow ( dist , c2 ) ;
v3 = d1 / dist ** d2 ;
v4 = e1 / dist ^ e2 ;
pot = v1 + v2 + v3 + v4
```

Operators and functions supported in expressions are shown in table 8.3. Note that a^b can be input as “a**b”, “a^b” or “pow(a,b)”[1].

Defining constants and using floating point notation is also supported as shown in File 8.3, section **[potential 3]**.

To check your potential definition you can use **ufpu**. See 12.4.

[1]Like fortran, basic or C languages, respectively

Table 8.3: Operators and functions supported in expressions

Operators		Precedence	Example
=	assignment	0	a=b
+	addition	1	a+b
-	subtraction	1	a-b
*	multiplication	2	a*b
/	division	2	a/b
unary +	unary plus	3	+a
unary -	unary minus	3	-a
**	a raised by power b, a^b	4	a**b
^	a raised by power b, a^b	4	a^b
Puntuaction			
( )	change precedence		(a+b)*c
,	comma, separate arguments in functions		pow(a,b)
;	semicolon, separate individual expressions		a=b+c; d=e+f
Functions			
exp	number e raised by power a, e^a		exp(a)
pow	a raised by power b, a^b		pow(a,b)

9 MOPAC interface

To err is human, but to really screw things up you need a computer.

Bill Vaughn

An additional feature of **GAFit** is the possibility of parametrizing a semiempirical Hamiltonian. The current version of **GAFit** supports MOPAC –2009 and 2012– as the external program to compute the PES of our system. In the example given in Section 16 the MOPAC interface is used to parametrize the intramolecular PES of vinyl cyanide.

The details of how **GAFit** works with an external interface –or external potential– are explained in the following.

9.1 External potential

The *external potential* works as follows:

- **GAFit** generates a whole generation, where each individual is a coefficient vector.
- *for each* individual,
  - the coefficients are written in the file named in the **external input** option of the **[job]** section.
  - the external program specified in the option **command** is run.

    * The external program must read the **external input** file,
    * doing its calculations,
    * and writing the file named in the **external fit** option of the **[job]**.
  - **GAFit** reads the **external fit** file.

- **GAFit** using the *fit*, given by the external program, applies the genetic operators to create a new generation.

If the *bulk* option is chosen, an entire generation is written to the **external input** file, and the external **command** must write into the **external fit** file all the individuals fitting values. This option speeds up calculations.

In all cases, the **command** is executed passing one argument in the command line: the number of the individuals that were written to the **external input** file.

For example, if the **command** is *mopac2009.sh*, and the job is an **external bulk** passing an entire generation of 100 coefficient vectors, the command line executed by the shell is:

```
$ mopac2009.sh 100
```

external input examples are given in Files 6.18 and 6.19. **external fit** examples are the Files 6.20 and 6.21

GAFit only evaluates if there is a command processor available –i.e. *sh*– and the **coefficients** value. No other checks are performed.

Note that you cannot use TAGS with **external potentials**.

9.1.1 Autoconfigure

If the option *external auto* is chosen, the external command can configure **GAFit**. At the beginning, **GAFit** executes the external command passing an argument of "0". If the external command is *mopac2009.sh*, the command line executed by the shell is:

```
$ mopac2009.sh 0
```

The external command must answer with a file named "*response*" with the options requested. This file follows the *job.txt* format. An example from the MOPAC interface is shown below.

File 9.1: response

```
[job]
type: external_bulk
coefficients: 16
external_input: mopac2009.input
external_fit: mopac2009.fit
bounds: bounds.txt

[coefficient_names]
BETAS_H
ZS_H
ALP_H
GSS_H
USS_C
UPP_C
BETAS_C
BETAP_C
ZS_C
ZP_C
ALP_C
GSS_C
GSP_C
GPP_C
GP2_C
HSP_C
```

Note that **GAFit** does not check if there is a *response* file before the call. All is ok if it finds one, independently of whether it has been created by the system call or not.

9.1.2 Stopping an external job

You can stop a running job writing a **stop file** in the folder where it is running. The **stop file**'s name is **__STOP__**, and the text it contains is whatever you want.

```
$ echo ‘‘stop job’’> __STOP__
```

A first approach to the general problem of launching an external program is shown as a guideline for development to complement section 9.1 with a useful case: MOPAC 2009.

Later, a better solution –**shepherd**–, specifically designed to solve some problems found while testing these scripts, is developed and discussed in Section 10.

9.2 Interfacing with MOPAC 2009

Interfacing with MOPAC 2009 is achieved using three new tools:

injector Written in **C**, is responsible for:

- answering the **GAFit** *external auto* configuration option.
- creating the MOPAC's external file parameters.
- creating the MOPAC's input file.

extractor Written in **perl** and using **perl**'s special characteristics to extract text, it is in charge of:

- extracting and digesting data from the MOPAC output to a intermediate file with a format for easy retrieve by the next tool.
- dealing with MOPAC's calculation failures.

fitter Written in **fortran**,

- calculates the fitting.
- writes the file with the fits to be read by **GAFit**.

Two templates are used to create the files needed by MOPAC 2009.

coefficients template (**COEFS_TEMPLATE**) is used to extract the coefficients values and replace them with the ones obtained by **GAFit** and to count and assign names to **GAFit** coefficients too.

MOPAC **calculation template** (**MOPAC_TEMPLATE**), contains one or more calculations. For example: one for the reactants, one for the TS and a third one for the products (calculations 1, 2 and 3 respectively).

It is used to generate a continuous and unique file with all calculations, which is employed as input of MOPAC 2009. There are places, marked with an @, where the symbol is replaced by the file name of the *coefficients template*, containing the coefficients obtained by **GAFit**.

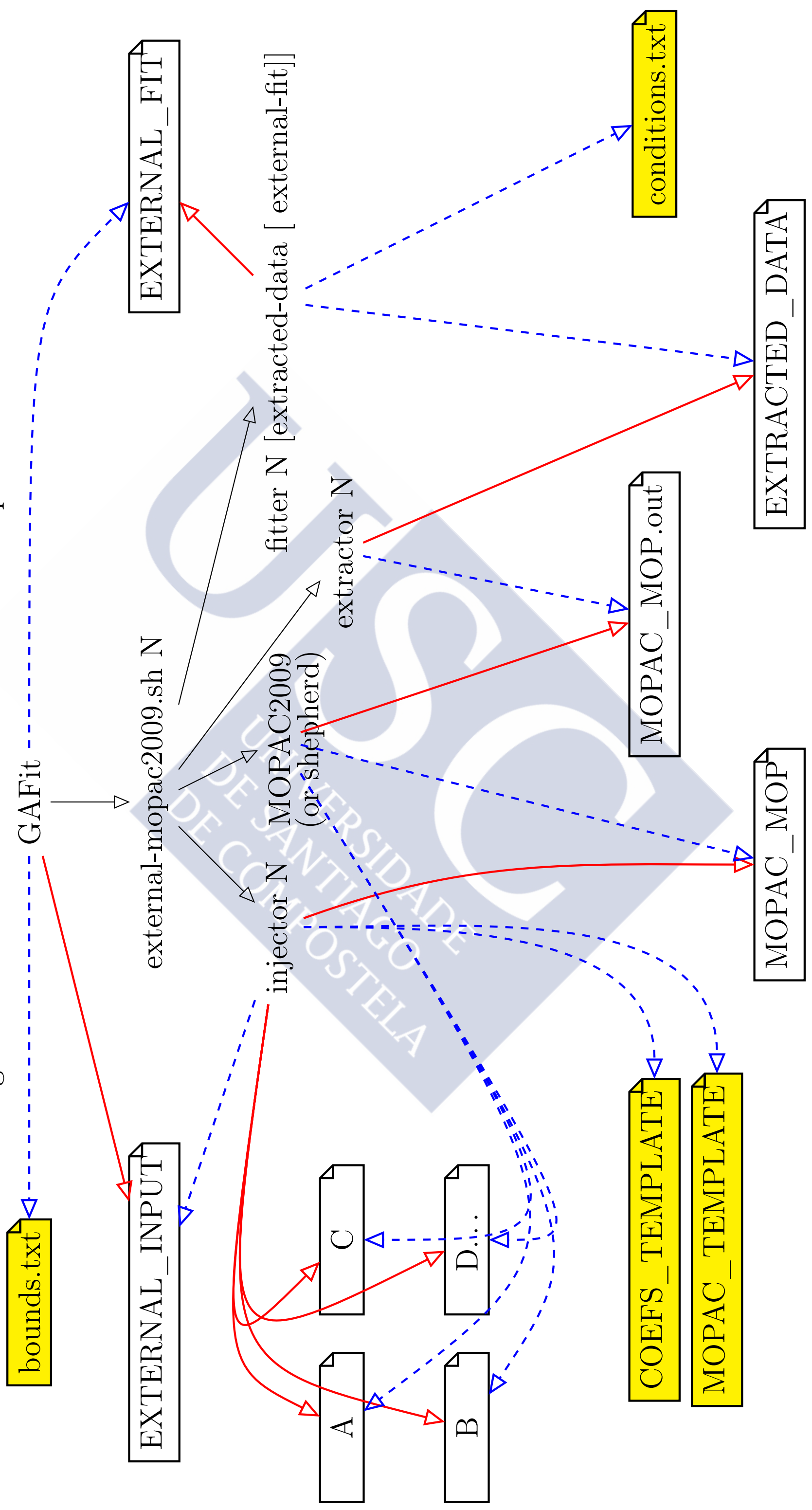
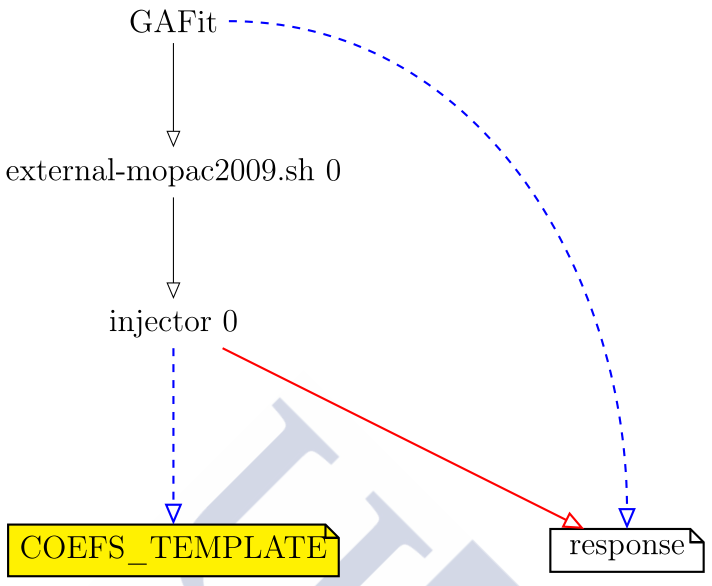


Figure 9.1: MOPAC 2009 interface: normal operation

Figure 9.2: MOPAC 2009 interface: autoconfigure



If there are two calculations in the MOPAC calculation template and **GAFit** exports 100 sets of coefficients per generation, then the unique file generated contains 200 calculations, and also, there are 100 independent files generated from the *coefficients template*, each one with a complete set of coefficients replaced.

These files are named A . . . Z, AA . . . AZ . . . and so on.

Figures 9.1 and 9.2 show the relations between programs and files:

- Dashed blue lines indicate that a tool uses the file as input.
- A red line indicates that a tool creates the file.
- Black lines indicate calls to execute a tool.
- Files fill in yellow indicate that they must be created or given by the user.

There are environmental variables, shown in Table 9.1, which can be set to control the file names.

Notice that for the **fitter** point of view, **EXTERNAL_FIT** and **E XTRACTED_DATA** are command line arguments.

Table 9.1: Environmental variables

Variable	Default value	Tools
COEFS_TEMPLATE	template.coefs	injector
MOPAC_TEMPLATE	template.mop	injector
MOPAC_MOP	mopac_input.mop	injector, MOPAC 2009, extractor, shepherd
EXTERNAL_INPUT	mopac2009.input	**GAFit**, injector
EXTERNAL_FIT	mopac2009.fit	**GAFit**
EXTRACTED_DATA	extracted.data	extractor
BOUNDS_FILE	bounds.txt	**GAFit**, injector

9.3 External command

GAFit only calls an external shell script: *external-mopac2009.sh*, or the name given in job.txt. There is a complete example in the folder *mopac-example* which can be examined in the File 9.2. A minimal implementation due to the defaults could be the one in File 9.4.

File 9.2: external-mopac2009.sh

```
#!/bin/sh
export MOPAC_LICENSE=$HOME/mopac2009

export COEFS_TEMPLATE="template.coefs"
export MOPAC_TEMPLATE="template.mop"
export MOPAC_MOP="mopac_input.mop"
export EXTERNAL_INPUT="mopac2009.input"
export EXTERNAL_FIT="mopac2009.fit"
export EXTRACTED_DATA="extracted.data"
export BOUNDS_FILE="bounds.txt"

injector $1
if [ "$1" -ne "0" ]
then
        $MOPAC_LICENSE/MOPAC2009.exe $MOPAC_MOP
        extractor $1
        fitter $1 $EXTRACTED_DATA $EXTERNAL_FIT
fi
```

9.4 injector

injector is a program written in C. The syntax is

```
injector number-of-vectors [bulk]
```

where **number-of-vectors** and **bulk** are parameters explained below.

9.4.1 Configuration

If the *external auto* option is used, **GAFit** calls the *external command* passing a “0” as first parameter, so the **injector** creates the file *response* and **GAFit** uses this information to configure itself. This file is deleted the first time **injector** runs in the normal operation.

File 9.3: job.txt in mopac-example

```
[parameters]
population:         100
crossover rate:     0.75
blx_alpha:          0.5
mutation rate:      0.1
elitism:            yes
tournament size: 5
crossover:        sbx
mutation:         sigma
sigma:              0.1
direction:          min

[job]
runs:               1
evaluations:        5000
type:            external auto
command:         external-mopac2009.sh

[print]
print runs: yes
```

The data needed to create the *response* file is obtained from environmental variables and from the **COEFS_TEMPLATE** file[1]. If it is not set, there are default values for them (see Table 9.1).

A minimal external script is shown in File 9.4. In this case, the *external auto* option defaults to *external*. To override defaults use *bulk* option to change to *external bulk*.

[1]Number and name of the coefficients.

File 9.4: Minimal external-mopac2009.sh

```
#!/bin/sh
export MOPAC_LICENSE=$HOME/mopac2009

export MOPAC_MOP="mopac_input.mop"

injector $1
if [ "$1" -ne "0" ]
then
        $MOPAC_LICENSE/MOPAC2009.exe $MOPAC_MOP
        extractor $1
        fitter $1
fi
```

9.4.2 Normal operation

If the parameter is not “0”, it must be the number of coefficient vectors, which are written in the file **EXTERNAL_INPUT**.

The injector reads **EXTERNAL_INPUT** and using **COEFS_TEMPLATE** and **MOPAC_TEMPLATE** it creates the **MOPAC_MOP** file and its relative external coefficients files, which are named according to the default option for the coefficients names. See 6.5.

File 9.5: COEFS_TEMPLATE file: template.coefs

```
BETAS_H       -6.173787
ZS   H         1.188078
ALP  H         2.882324
GSS  H        12.848
USS  C       -52.028658
UPP  C       -39.614239
BETAS_C      -15.715783
BETAP_C       -7.719283
ZS   C         1.808665
ZP   C         1.685116
ALP  C         2.648274
GSS  C        12.23
GSP  C        11.47
GPP  C        11.08
GP2  C         9.84
HSP  C         2.43
```

At the configuration stage, the file **COEFS_TEMPLATE** is analyzed; this file provides the number of coefficients and their names.

In a normal operation, the file is replicated to generate the files needed to complement the jobs in **MOPAC_TEMPLATE**.

File 9.6: MOPAC_TEMPLATE file: template.mop

```
AM1 precise external=@ geo-ok nosym


  H     0.00000000 +0    0.0000000 +0    0.0000000 +0                 ↪
       ↪          0.1275
  C     1.09852142 +1    0.0000000 +0    0.0000000 +0     1     0    ↪
       ↪ 0      -0.1565
  C     1.33416836 +1  123.1900576 +1    0.0000000 +0     2     1    ↪
       ↪ 0      -0.0994
  H     1.09879509 +1  115.3226363 +1  179.9929115 +1     2     1    ↪
       ↪ 3        0.1270
  H     1.10533055 +1  122.1640414 +1  179.9944757 +1     3     2    ↪
       ↪ 1        0.1514
  C     1.41933576 +1  114.5208739 +1  179.9977508 +1     3     5    ↪
       ↪ 2      -0.1114
  N     1.16399609 +1  179.1128557 +1    1.2752342 +1     6     3    ↪
       ↪ 5      -0.0387

 oldgeo AM1 precise external=@ force geo-ok nosym


AM1 precise ts external=@ geo-ok nosym


 C    0.000000 0    0.000000 0    0.000000 0       0    0    0
 C    1.310566 1    0.000000 0    0.000000 0       1    0    0
 C    2.179061 1  104.132782 1    0.000000 0       2    1    0
 N    1.160916 1  160.493759 1    0.000000 1       3    2    1
 H    1.076805 1  126.972862 1    0.000000 1       1    2    3
 H    1.084538 1  114.088127 1  180.000000 1       1    2    3
 H    1.208813 1   35.831474 1  180.000000 1       2    3    4
```

MOPAC_MOP is created clonning **MOPAC_TEMPLATE** and replacing the symbol @ with the files obtained changing parametres in the **COEFS_TEMPLATE** file, one per each different coefficient vector.

Therefore, if the *external bulk* option is used, and there are 100 coefficients per generation, one **MOPAC_MOP** file is generated referencing 100 different files, each one being a **COEFS_TEMPLATE** clone with the parameters obtained from **GAFit external input** changed.

9.5 extractor

extractor is a perl script which analyses the MOPAC 2009 output file, the **MOPAC_MOP** file replacing the *.mop* extension by *.out*. I.e. if

MOPAC_MOP is the default *mopac_input.mop* then the MOPAC 2009 output is *mopac_input.out*.

Syntax:

```
extractor number-of-vectors
```

File 9.7: Extractor first lines

```
#!/usr/bin/perl

use strict;

use constant {
      HEATFCAL    => 0,
      HEATFJUL    => 1,
      NUMATOMS    => 2,
      CARTESIAN   => 3,
      NUMFREQ     => 4,
      FREQUENCIES => 5,
      CALCPERIND  => 6,
};

my (%defaults) = (
      'COEFS_TEMPLATE' => "template.coefs",
      'MOPAC_TEMPLATE' => "template.mop",
      'MOPAC_MOP'      => "mopac_input.mop",
      'EXTERNAL_INPUT' => "mopac2009.input",
      'EXTERNAL_FIT'   => "mopac2009.fit",
      'EXTRACTED_DATA' => "extracted.data",
);

my (
      $CoefsTemplate, $MopacTemplate, $MopacMop, $ExternalInput,
      $ExternalFit,   $MopacOut,      $Extracted
);

my (@mopErrors) = (
      "TOO MANY ITERATIONS IN LAMDA BISECT",
      "CALCULATION IS TERMINATED TO AVOID ZERO DIVIDE",
      "GRADIENT IS TOO LARGE TO ALLOW FORCE MATRIX TO BE CALCULATED"
        ,
      "THIS IS A FATAL ERROR, RUN STOPPED IN GMETRY",
      "TS FAILED TO LOCATE TRANSITION STATE",
      "A FAILURE HAS OCCURRED, TREAT RESULTS WITH CAUTION!!",
      "EXCESS NUMBER OF OPTIMIZATION CYCLES",
      "SHEPHERD NON RECOVERABLE ERROR"
);

my (@mopSTOPErrors) = ("EXTERNAL file: '.*' does not exist!");
```

The gathered information is saved in an intermediate file –**EXTRACTED_DATA**– with a suitable format to be processed later.

extractor accepts one command line parameter: the number of individual coefficients vectors used. The rest of the configuration data must be passed through environmental variables or use the defaults. See Table 9.1 and File 9.7, line 15.

extractor also checks for MOPAC 2009 failure, i.e., when MOPAC 2009 is not able to achieve a result with the given parameters. Special care must be taken to test this and, if needed, change the @*mopErrors* array in the line 29 of the script –File 9.7–, adding the new error texts not listed before in the array found in the MOPAC 2009 output.

Also, change the @*mopSTOPErrors* array in line 40 of the script adding the fatal error texts[2] found in the MOPAC 2009 output which must stop the entire job.

File 9.8: extracted.data

```
0 0 6
3
13 0 0
-879.04453
13 0 1
-3677.92230
13 0 2
7
13 0 3
1 H 0.0000 0.0000 0.0000
13 0 3
2 C 50.4746 0.0000 0.0000
13 0 3
3 C 84.8574 36.9379 0.0000
13 0 3
4 H 54.3105 -50.2347 -0.8804
13 0 3
5 H 122.4161 78.0661 -0.2120
13 0 3
6 C 52.1018 1.8440 0.2744
13 0 3
7 N 51.2886 0.9219 0.1372
13 0 4
0
13 1 2
7
13 1 4
15
13 1 5
1 -7.23
13 1 5
2 -7.20
```

[2]They could be a *REGEX* expression as in this case. Note the '.*' in the middle of the string.

```
13␣1␣5
3␣−6.01
13␣1␣5
4␣−5.91
13␣1␣5
5␣−4.20
13␣1␣5
 [...]
```

The **EXTRACTED_DATA** file format takes two lines per each kind of data. The first line indicates:

- the coefficient vector used from **EXTERNAL_INPUT**,
- the number of calculations from **MOPAC_TEMPLATE**, and
- the code type.

The second line has the data itself.

Table 9.2: Extracted data

mnemonic	code	data fields	data
HEATFCAL	0	1	Heat of formation in kcal/mol
HEATFJUL	1	1	Heat of formation in kJ/mol
NUMATOMS	2	1	Number of atoms
CARTESIAN	3	5	Sequence number in structure, atom symbol and x, y, z coordinates
NUMFREQ	4	1	Number of total frequencies
FREQUENCIES	5	2	Sequence number and value in cm^{-1}
CALCPERIND	6	1	Total number of different calculations per coefficient vector

The different types of extracted data are shown in table 9.2 and in the line 5 of the File 9.7. An example is given in File 9.8. Failed calculations are not written to the file.

The tool **lsexdata** can be used to show the contents of the **EXTRACTED_DATA** file.

9.6 fitter

fitter reads the **EXTRACTED_DATA** file to calculate a fit for each coefficient vector using the conditions in the *conditions.txt* file. The variables that can be used to calculate the fit are shown in table 9.3. It is written in fortran and the syntax:

```
fitter number-of-vectors [extracted-data-file [ external-fit-file]]
```

The optional parameters –*extracted-data-file* and *external-fit-file*– defaults to the ones shown in the table 9.1 –**EXTRACTED_DATA** and **EXTERNAL_FIT**, respectively.

Table 9.3: Fitter conditions

Condition	data fields	data	comment
heat	3	calcA value weight	Heat of formation of calculus *calcA*
delta	4	calcA calcB value weight	Difference between heat of formation of calculation *calcA* and *calcB*. $\Delta = (calcA - calcB)$ in kcal/mol
frequency	4	calcA N value weight	Frequency number N of the calculation *calcA*
distance	5	calcA atom1 atom2 value weight	Distance between *atom1* and *atom2* into calculation *calcA*
angle	6	calcA atom1 atom2 atom3 value weight	Angle between *atom1*, *atom2* and *atom3* into calculation *calcA*
dihedral	7	calcA atom1 atom2 atom3 atom4 value weight	Dihedral angle between *atom1*, *atom2*, *atom3*, and *atom4* into calculation *calcA*
penalty	1	penalty	Fit if any of the MOPAC 2009 calculations failed for a given coefficient vector

Each line references the calculation index into the **MOPAC_TEMPLATE** file, atom indexes, frequency numbers, etc, a reference value to check against the calculated one, and a weight.

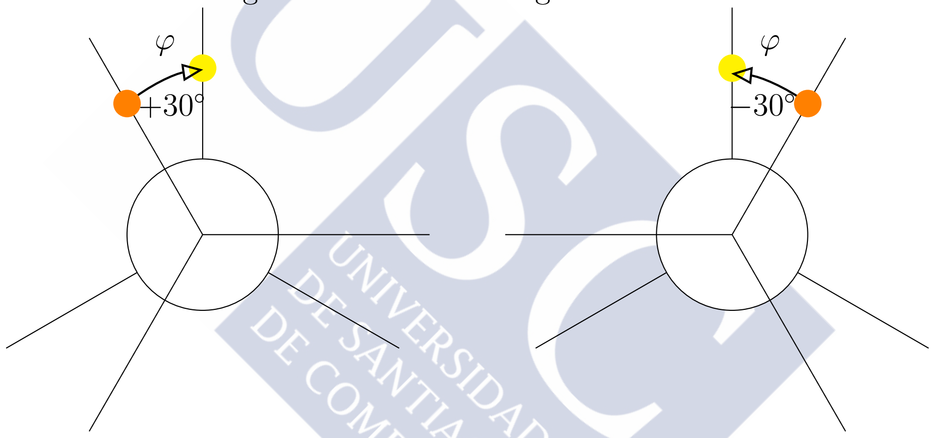
An example of the *conditions.txt* file is shown in the File 9.9. The overall fit per coefficient vector is the sum of relative differences in each line calculation multiplied by its weight.

$$\textbf{fit} = \begin{cases} \sum \left[\frac{(\textbf{Reference value}_i - \textbf{Calculated value}_i)}{\textbf{Reference Value}_i}\right]^2 \textbf{Weight}_i \; \textit{if calculation is done.} \\ \\ \textbf{penalty} \; \textit{if calculation fails.} \end{cases}$$

Due to the fact that distances, angles and dihedral angles are calculated from the Cartesian coordinates, the intervening atoms may not be connected in any other way.

The dihedral angles follow the usual convention, shown in the figure 9.3.

Figure 9.3: Dihedral angles convention



To express a condition, only the four first characters are needed, as shown in bold in table 9.3.

An example of fitter calculations using the file *conditions.txt* shown in File 9.9 is presented in File 9.10, where the type of condition, the calculated value, the reference value, the weight used, and the individual contributions to the final fit were printed.

File 9.9: conditions.txt

```
delt   1  2  100.6    0.1
frequency   2    15   3271.0  1e-4
distance   3     1      7    3.700309096  100.0
penalty 1e10
```

File 9.10: fitter calculations example

```
 DELTA            calc=   317.14201000000003        ref=   100.59999999999999    
     we=  0.10000000000000001        cont=  0.46332780745783869
 FREQUENCY        calc=   1894.7900000000000        ref=   3271.0000000000000    
     we=  1.0000000000000005E-004  cont=  1.7701429112978933E-005
 DISTANCE         calc=   2.8164272438676630        ref=   3.7003090959999998    
     we=   100.00000000000000        cont=   5.7057459091163221
  individual           97  fit=   6.1690914180032737
 DELTA            calc=   3347.9733500000002        ref=   100.59999999999999    
     we=  0.10000000000000001        cont=   104.20018333626695
 FREQUENCY        calc=   4569.0900000000001        ref=   3271.0000000000000    
     we=  1.0000000000000005E-004  cont=  1.5748838169209032E-005
 DISTANCE         calc=   2.2933691656599904        ref=   3.7003090959999998    
     we=   100.00000000000000        cont=   14.456897586931765
  individual           98  fit=   118.65709667203687
 DELTA            calc=  -8.4001199999999994        ref=   100.59999999999999    
     we=  0.10000000000000001        cont=  0.11739726808151489
 FREQUENCY        calc=   1086.3800000000001        ref=   3271.0000000000000    
     we=  1.0000000000000005E-004  cont=  4.4605737294125923E-005
 DISTANCE         calc=   3.8158625315909900        ref=   3.7003090959999998    
     we=   100.00000000000000        cont=  9.7519107516602199E-002
  individual           99  fit=  0.21496098133541122
 DELTA            calc=   2421.2273100000002        ref=   100.59999999999999    
     we=  0.10000000000000001        cont=   53.212643739134158
 FREQUENCY        calc=   1331.9000000000001        ref=   3271.0000000000000    
     we=  1.0000000000000005E-004  cont=  3.5143039809276018E-005
 DISTANCE         calc=   5.1161291627557643        ref=   3.7003090959999998    
     we=   100.00000000000000        cont=   14.639967757020900
  individual          100  fit=   67.852646639194873
```

9.7 Caveats

Some problems may arise when using a long MOPAC input file if the initial parameters are far from the optimized ones:

- If MOPAC crashes, it can freeze the entire job and you have to kill the MOPAC process manually. Alternatively, you may use **shepherd** to control this. See 10.

- It can be worse: a failed MOPAC calculation can spoil all the previous calculations in the input file. These failed calculations are the ones which the **fitter** assigns a *penalty*. See 9.6. You must use the **injector** default option to calculate one vector at once, or use **shepherd** to deal with it.

9.8 MOPAC 2012

MOPAC 2012 output differs a little from that of MOPAC 2009. From our point of view, the most important change is that some cartesian coordinates printout are missing, so internal coordinates must be used and converted to Cartesian. This job must be done by **extractor** using *quaternion maths* to calculate 3D rotations. The Karney [16] article is a good reference about this subject.

10 shepherd

Computers are good at following instructions, but not at reading your mind.

Donald Knuth

shepherd launches and controls the running MOPAC processes. It is written in C. Also, it can deal with the problems shown in section 9.7. It can:

- Detect and kill a MOPAC frozen/crashed process.
- Split the job sent by **GAFit** from one individual once at a time to a bunch of them.

  The default behavior is to send a sole calculation – a MOPAC_TEMPLATE clone– per MOPAC process. You can change defaults modifying the source code and compiling it again: Details in section 10.2.
- Run, control and maintain a suitable number –equal or near to the number of resources available: CPUs, cores or hyperthreads, etc– of parallel MOPAC processes.

  shepherd calculates a good value to this number. It dynamically changes depending on the node load.

Syntax:

```
shepherd
```

The *external command* to be used is slightly different with **shepherd** as shown in File 10.1:

- To use the special characteristics of **shepherd** the line 12 is changed to pass an entire parameters vector (*bulk*) .
- Also line 15 is changed, where **shepherd** replaces the entire "$MOPAC_LICENSE/ MOPAC2009.exe $MOPAC_MOP" line. **shepherd** calls itself the MOPAC executable as needed.

File 10.1: *external-mopac2009.sh* with shepherd

```
1  #!/bin/sh
2  export MOPAC_LICENSE=$HOME/mopac2009
3
4  export COEFS_TEMPLATE="template.coefs"
5  export MOPAC_TEMPLATE="template.mop"
6  export MOPAC_MOP="mopac_input.mop"
7  export EXTERNAL_INPUT="mopac2009.input"
8  export EXTERNAL_FIT="mopac2009.fit"
9  export EXTRACTED_DATA="extracted.data"
10 export BOUNDS_FILE="bounds.txt"
11
12 injector $1 bulk
13 if [ "$1" -ne "0" ]
14 then
15         shepherd
16         extractor $1
17         fitter $1 $EXTRACTED_DATA $EXTERNAL_FIT
18 fi
```

A shorter version of File 10.1 is 10.2 using the default values. **shepherd** is totally configured by the environmental variables.

File 10.2: Shorter *external-mopac2009.sh* with shepherd

```
#!/bin/sh
export MOPAC_LICENSE=$HOME/mopac2009

injector $1 bulk
if [ "$1" -ne "0" ]
then
        shepherd
        extractor $1
        fitter $1
fi
```

10.1 Controling freezes

If a MOPAC 2009 process crashes, it freezes and blocks all the entire job (see 9.7).

In these cases, *glibc* will produce output on the process controlling terminal, so the environment variable LIBC_FATAL_STDERR_=1 must be set to send fatal errors to *stderr* in order to check it.

shepherd forks itself and execs the MOPAC process in an environment with the LIBC_FATAL_STDERR_ variable set, and establishing a *pipe* with the child process to read MOPAC's *stderr*.

If a fatal error is noticed, **shepherd** kills the child process avoiding the freeze and creates a fake MOPAC output file suitable for the **extractor**.

```
[...]
shepherd #flocks:4
shepherd errno 2 forrtl: severe (174): SIGSEGV, segmentation fault occurred
Image              PC        Routine            Line        Source
libc.so.6          B760BEEA  Unknown               Unknown  Unknown
libc.so.6          B7610050  Unknown               Unknown  Unknown
MOPAC2009.exe      08267594  Unknown               Unknown  Unknown
MOPAC2009.exe      08089053  Unknown               Unknown  Unknown
MOPAC2009.exe      0822AA58  Unknown               Unknown  Unknown
MOPAC2009.exe      081E835E  Unknown               Unknown  Unknown
MOPAC2009.exe      0818392E  Unknown               Unknown  Unknown
MOPAC2009.exe      0804A141  Unknown               Unknown  Unknown
libc.so.6          B75B1DB6  Unknown               Unknown  Unknown
MOPAC2009.exe      0804A051  Unknown               Unknown  Unknown

in file BE-BE.out lost sheep:56
shepherd elapsed time:17.611128
[...]
```

In the above example, **shepherd** notices a runtime error, so it kills the MOPAC 2009 process, creates the fake *BE-BE.out* file and continues processing. In the case of MOPAC 2012, the output is the same but with less detail.

10.2 Operating modes

shepherd takes the file MOPAC_MOP as input to build a MOPAC_MOP.out file, suitable for the extractor.

It calculates the number of individuals –how many MOPAC_TEMPLATEs are in the file–, and it can split the input in slices[1] from one individual[2]

[1]Flocks in shepherd parlance
[2]Sheep

to many, running a MOPAC 2009 process on each slice.

The temporary files for the slices are in the form *FIRST-LAST.ext*, where *FIRST* and *LAST* are the first and last individuals in the file using the same naming convention as the *coefficient names* default option –see 6.5–, and *ext* is the extension corresponding to the type of file.

For example:

- *BE-BE.mop* is the MOPAC 2009 input file corresponding from 56th to 56th individuals.

- *A-E.out* is the MOPAC 2009 output file corresponding from 1st to 5th individuals as a result of calculations on *A-E.mop* input file.

The default is to launch a MOPAC 2009 process with an individual –i.e.: *A-A.mop*–, an individual per slice[3].

The other mode –**burst**– is disabled but it can be enabled recompiling the source code changing the line 613 in the *main* function setting **burst** to a value different from zero, File 10.3. **burst** mode is discouraged. See 9.7.

File 10.3: Shepherd, main function.

```
603
604 int
605 main (int argc, char **argv)
606 {
607   int burst = 0;
```

In this mode, the slice can contain more than one individual and it will be calculated by one MOPAC 2009 process.

10.3 Parallel processes

Tracking the minimum time elapsed, processing an entire population and running a fixed number of concurrent MOPAC 2009 processes, yields the blue line shown in figure 10.1.

There is an optimum number from which a further increase in the number of parallel processes provides little gain in performance, or no gain at all. **shepherd** maintains the number of parallel processes around this number.

[3]A sheep per flock

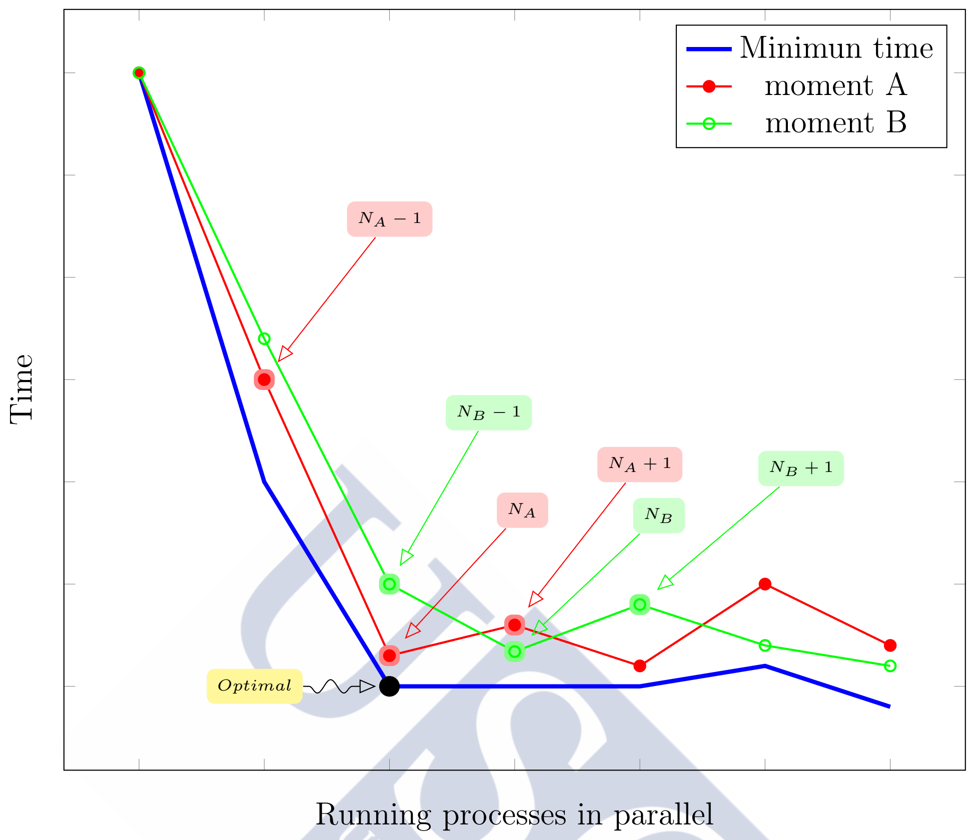


Figure 10.1: Shepherd algorithm: minimum time

Using the *taskset* utility, some experiments were performed. Figure 10.2 shows the results in a real four core CPU running repeatedly the same **GAFit** task –same seed– selecting from one to four cores.

The same experiment was performed in an eight virtual cpu system. The host really had only a four core CPU. The results are shown in figure 10.3. Notice that the algorithm behaves as if there were only four core CPU.

In figure 10.1, the red and green lines represent two different moments in the calculations. In both cases, **shepherd** steps down to find the first minimum. The minimum found is considered the optimum for this run –noted as N_A and N_B–.

shepherd processes entire populations cycling between N, $N+1$ and $N-1$ as the number of concurrent processes and it counts the real time spent. The time recorded changes dynamically, changing N in turn.

The number of times a number of parallel processes are chosen by

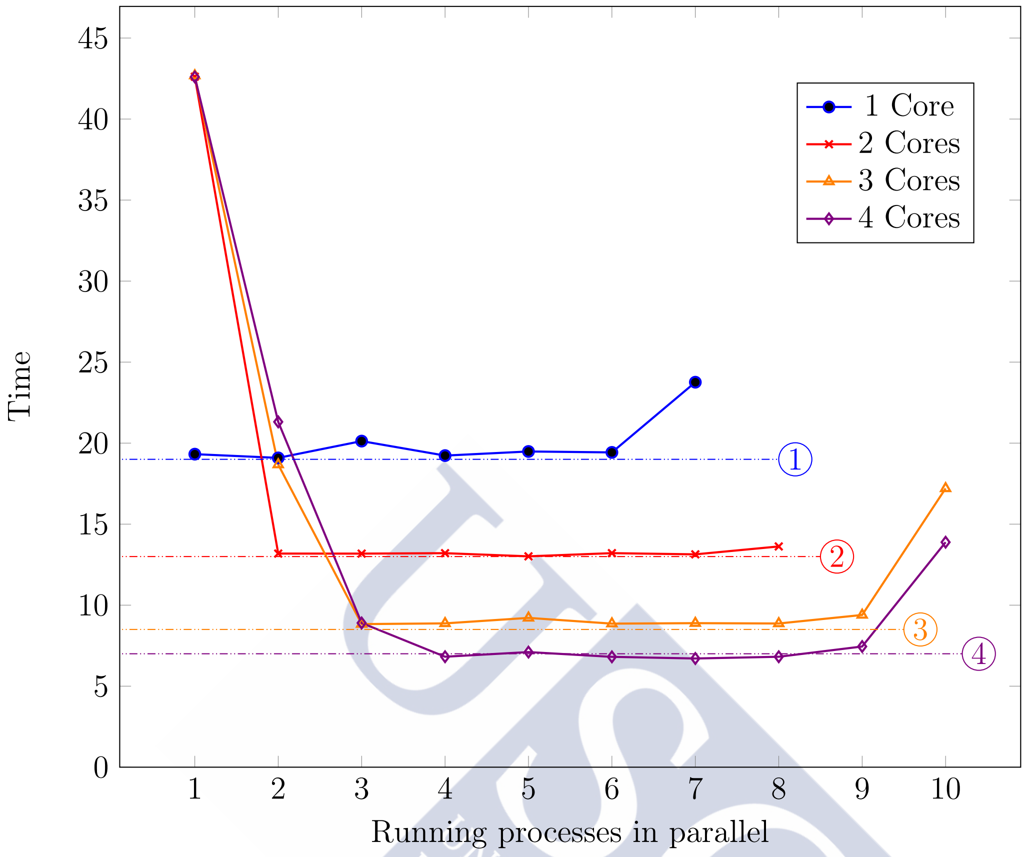


Figure 10.2: Real four core CPU: minimun time vs maximum concurrent parallel processes per run

shepherd are shown in figures 10.4 and 10.5.

This information can be summarized taking into account the average N in both cases, as shown in figure 10.6.

The algorithm presents a weakness: if **shepherd** writes to a local storage, the algorithm works well. However, if it writes to a share, it fails.

Figure 10.7 compares the same job –using the same seed, executed in a one CPU node– writing to a local storage and to a Network File System (NFS) share[4].

As shown, writing to a local storage stabilizes the minimum time from one running process –it is a one core CPU–. But writing to a NFS share, minimum times stabilize over 12 running processes, as if there were 12 core CPUs.

[4]A typical configuration where the user's HOME is shared with all cluster nodes.

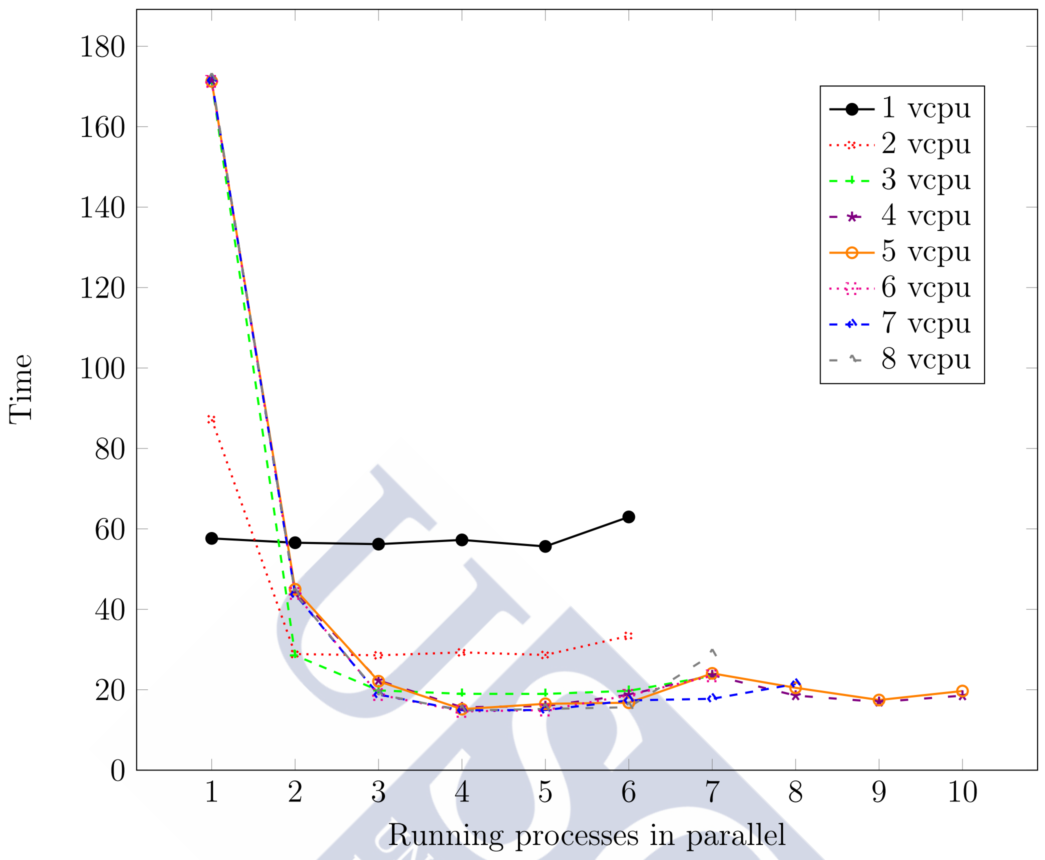


Figure 10.3: Virtual eight core CPU: minimum time vs maximum concurrent parallel processes per run

There are a utility, **lstimes**, to show the current number of parallel processes, the time spend, the number of times the algorithm choose a particular number of processes and the maximum and minimun time.

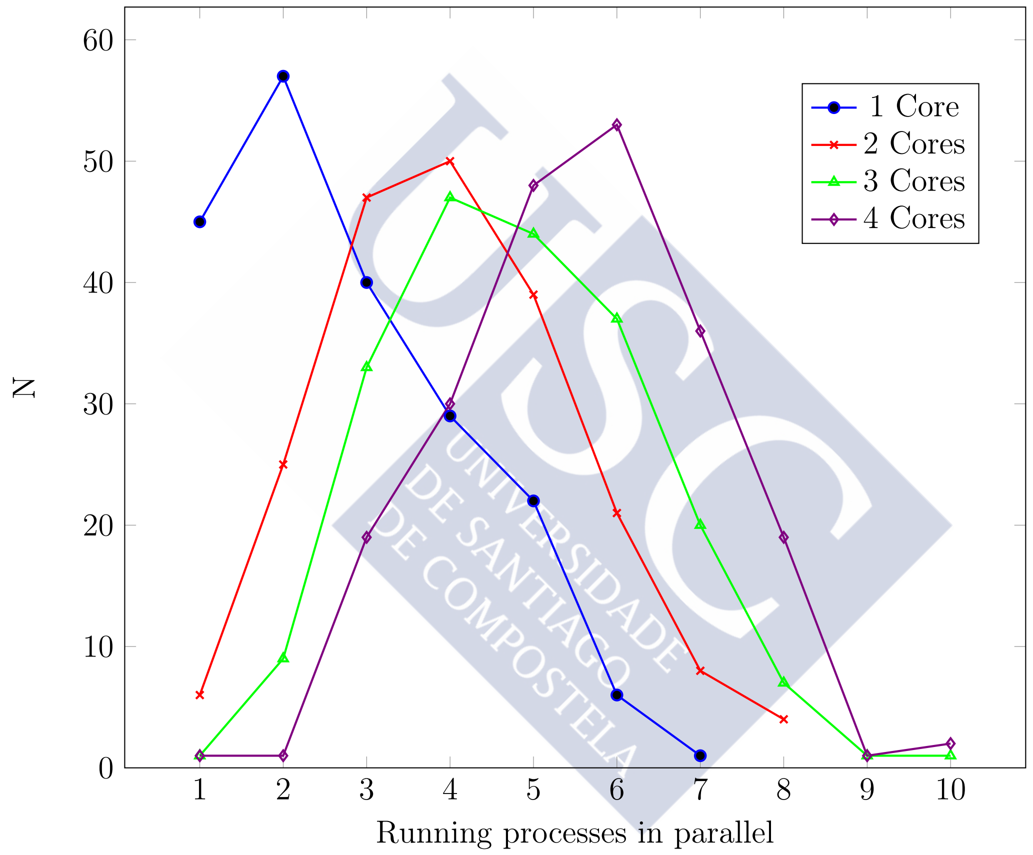


Figure 10.4: Real four core CPU: number of times (N) vs parallel processes per run

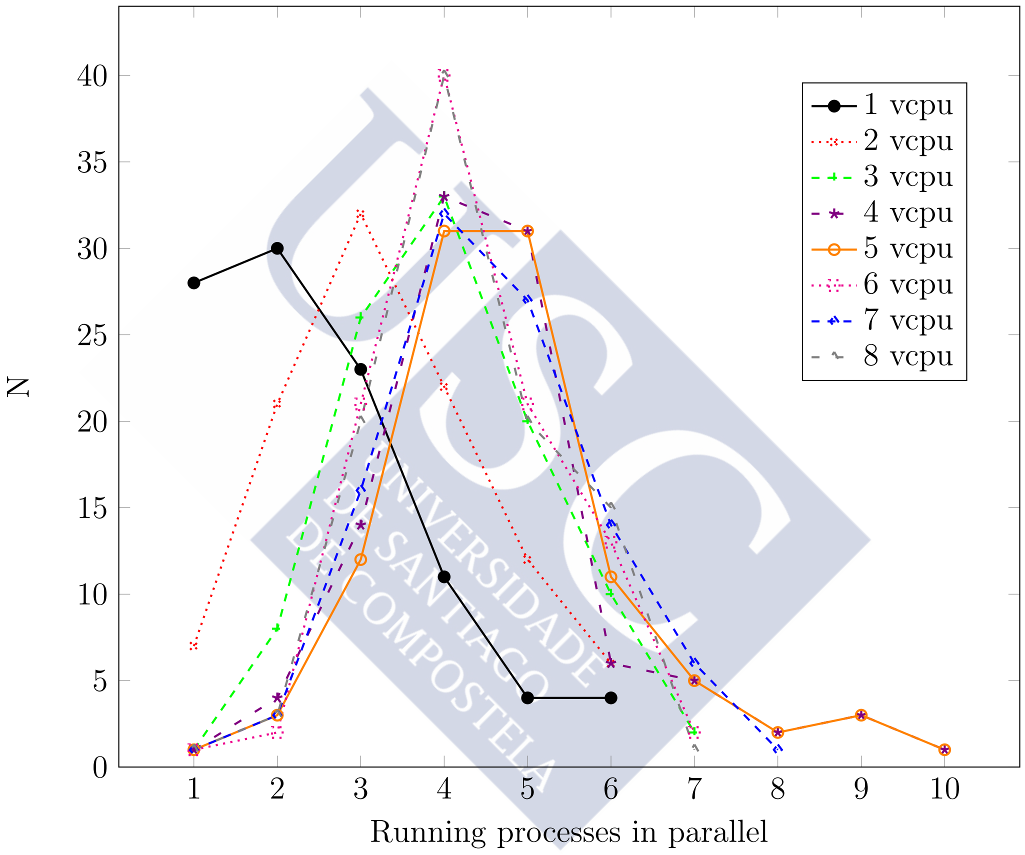


Figure 10.5: Virtual eight core CPU: number of times (N) vs parallel processes per run

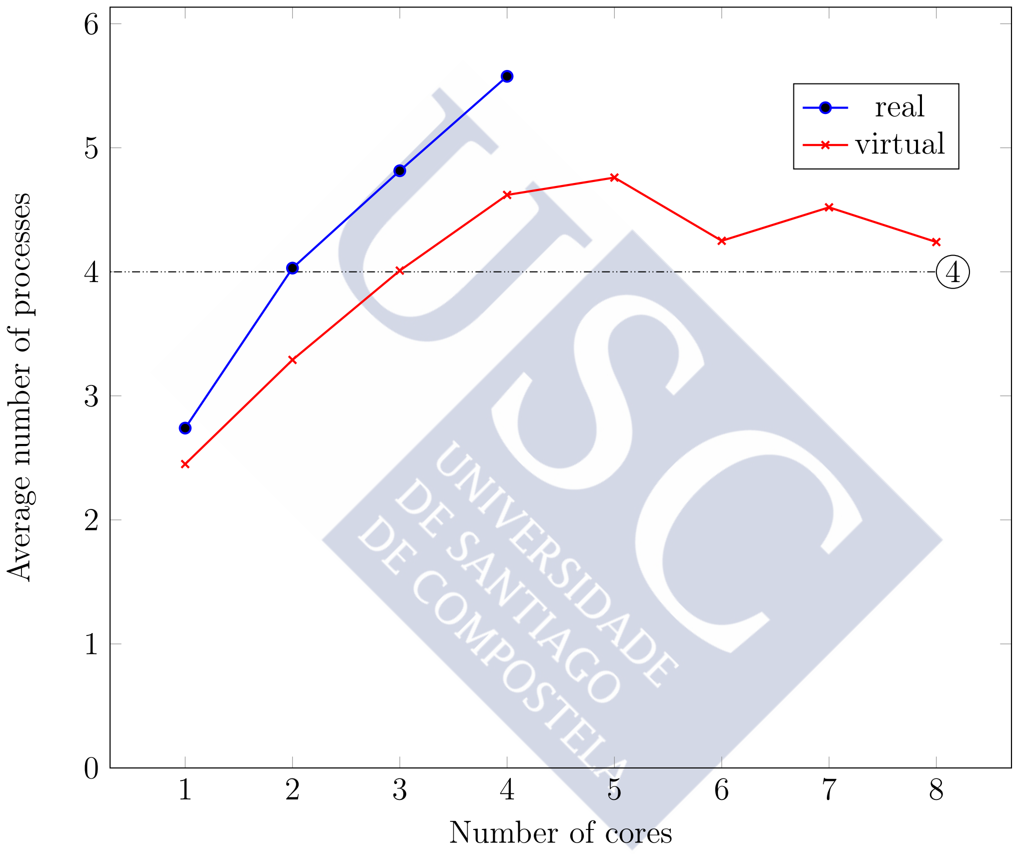


Figure 10.6: Average parallel processes per run. 4 core real CPU vs 8 core virtual CPU (4 real)

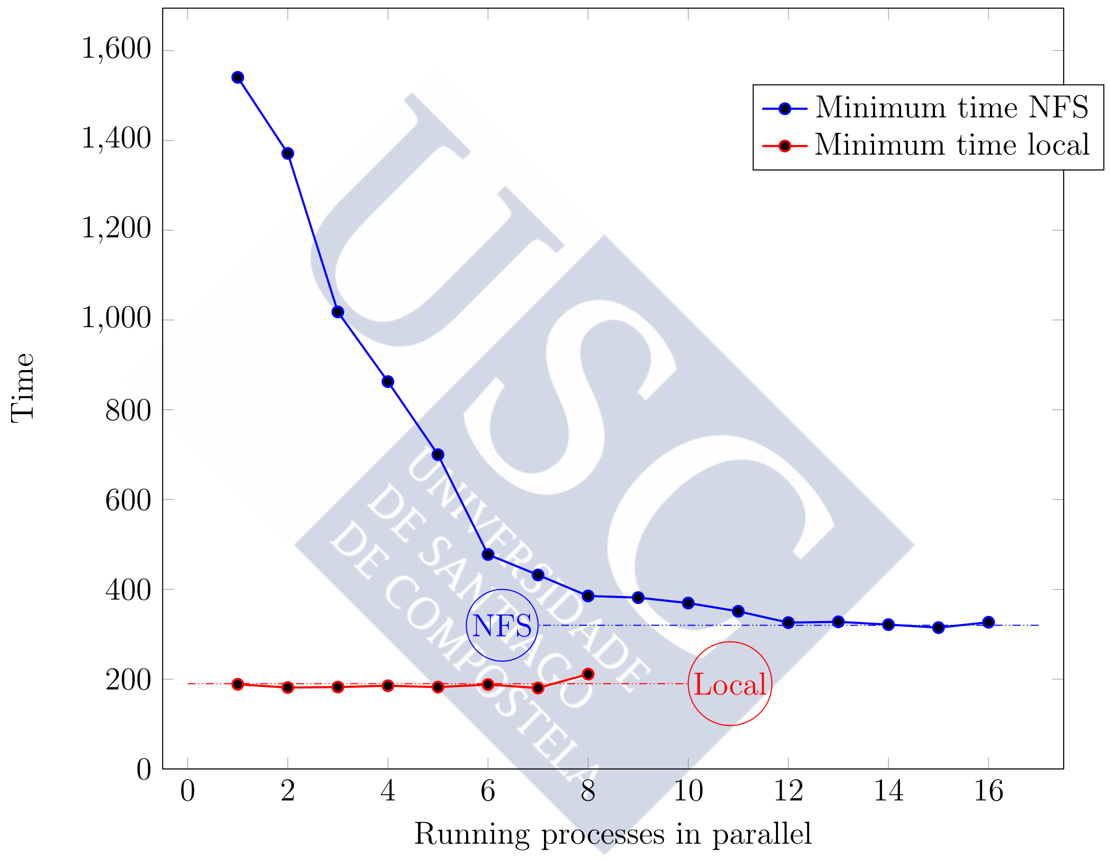


Figure 10.7: Behavior in the same one core CPU writing output to a NFS share vs local storage.

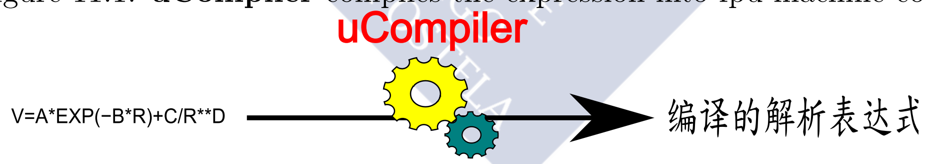
Fpu 11

Сколько языков ты знаешь - столько раз ты человек..

Anonymous

11.1 Fpu overview

Figure 11.1: **uCompiler** compiles the expression into fpu machine code.



Fpu is a function that emulates a Floating Point Unit (FPU) with its own instruction set in order to calculate *analytical expressions*. A related function, **uCompiler**, compile each source expression to **fpu** machine bytecode –Figure 11.1–, so it can be executed by a **Fpu** instance –Figure 11.2–.

Source code is included in the folders *fpu*, *compiler*, *pack*, *bytecodes* and *nullist*. A complete implementation is the **ufpu** tool. See Section 12.4.

Figure 11.3 shows a **Fpu** overview. It contains:

address stack used to operate, like to a real CPU *stack pointer*.

Figure 11.2: **Fpu** load the machine code and process the variables to obtain V value.

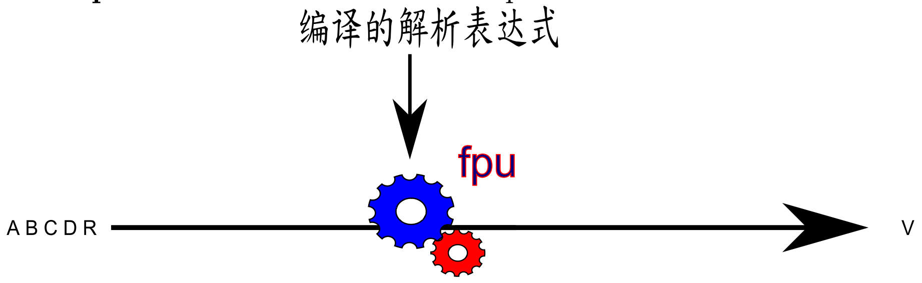


Table 11.1: **Fpu** source code

Folder	Comments
fpu	implements the **Fpu** function
compiler	implements the bytecode compiler
pack	bytecode packaging (as file or in memory)
bytecodes	bytecode instructions helper functions
nullist	implements stacks using null terminated lists of strings

memory pool an array referencing each allocated double, always growing up. There is no mechanism to resize down allocated memory, except resetting or deleting the **Fpu** from memory. It is like a real CPU *stack*.

program counter memory address pointing to the instruction to be processed, like a real CPU *program counter*.

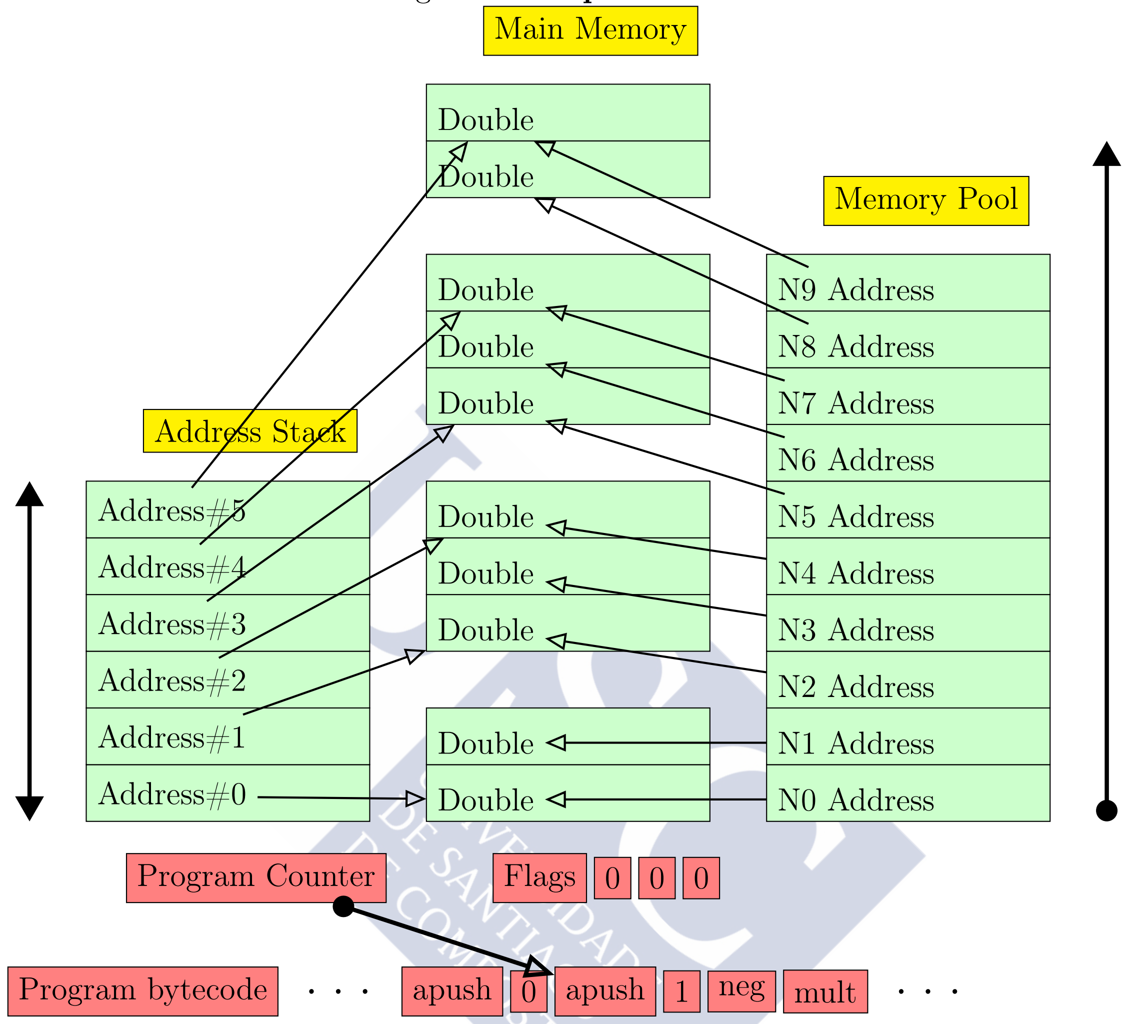
status flags register which is set on error like a real CPU *flags*.

program A continuous memory block containing the loaded program opcodes. The *data* and the *program code* does not share the same *"memory"*, so conceptually this is a virtual machine with a *Harvard* architecture[1].

The supported instruction set is shown in table 11.2.

[1]The opposite is the *von Neumann*'s architecture where data and program code are loaded in the same memory. This is the most widely used if not the unique.

Figure 11.3: **Fpu** overview



11.2 Mode of operation

A program example is shown in File 11.2, which is generated using the *job.txt* file configuration 11.1. Semicolons are interpreted as comments.

File 11.1: Job.txt to generate the File 11.2

```

[analytical]
expression: potential 1
distance: r
potential: v
coefficients: a, b, c, d

[potential 1]
V=A*EXP(-B*R)+C/R**D;
```

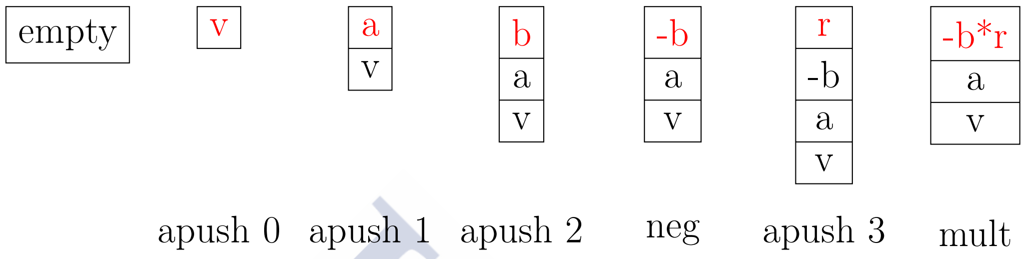
Table 11.2: Fpu instruction set

Instruction	Parameters	Comments
NOP		No operation
APUSH	N	pushes address of *memory pool* N onto *stack*
PUSH	A	allocates memory for value A incrementing *memory pool*, and pushes its address onto *stack*
POP		pops from *stack*
MOVE	N	copies top of stack value to N^{th} *memory pool* reference and leaves *stack* unchanged
STORE		moves value of *top of stack* to allocation referenced by *top of stack - 1*. Pops both addresses from *stack*
CLRF		clears status flags
ADD		adds two top most referenced values of stack, pops both from *stack*, and allocates memory for result pushing its address onto it
SUB		same as add but substracting
MULT		same as add but multiplicating
DIV		same as add but dividing
NEG		pops out top of stack reference, allocating memory for its negated value and pushing onto it
POW		raises power of the two top most values of stack popping them, allocates memory for result and pushes onto it
EXP		allocates memory for the result of $e^{topmoststack}$, pops the top most stack references, and pushes onto it the result reference

File 11.2: Bytecode source example

```
␣;␣v:0
␣;␣a:1
␣;␣b:2
␣;␣r:3
␣;␣c:4
␣;␣d:5
␣apush␣0
␣apush␣1
␣apush␣2
␣neg
␣apush␣3
␣mult
␣exp
␣mult
␣apush␣4
␣apush␣3
␣apush␣5
␣pow
␣div
␣add
␣store
```

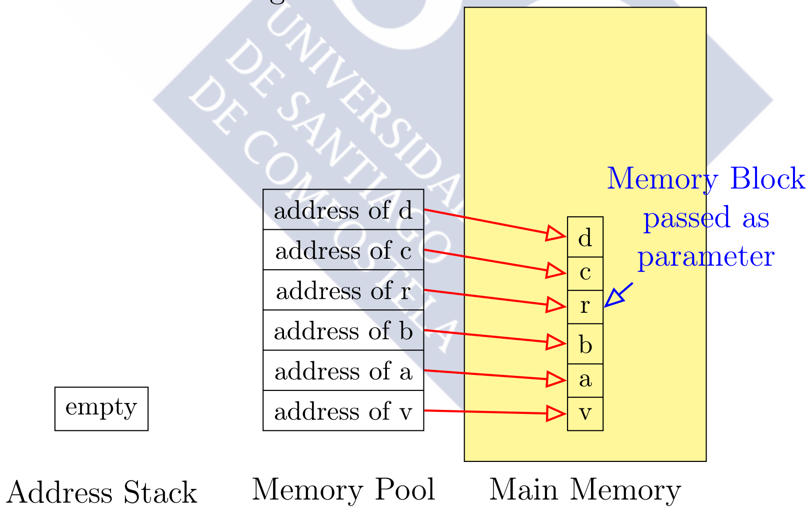
As shown in File 11.2, a memory block must be passed to **Fpu** containing the variables v, a, b, r, c, d in the correct order, as it could be seen in the first lines of the file –comments which are generated by the compiler as a remark–. At this time, the *Address Stack* is empty, so v, a and b are pushed.



Next, the value of the top of the stack is negated $(-b)$. r is pushed and multiplied by $-b$, so on top of the stack we have $-b * r$.

The e^{-br} is calculated and multiplied by a leaving it in the top of stack again.

Figure 11.4: Initial status



From the memory management point of view, the first six operations from File 11.2 are shown in figures 11.4 to 11.8. A memory block with the program variables is passed to **Fpu**.

New intermediate results generate new allocations of memory, all of them are taken into account by the *Memory Pool* array, which always grows. At the end, all of them are freed except the initial memory block with the initial variables returned to the caller.

Figure 11.5: apush 0, apush 1, apush 2

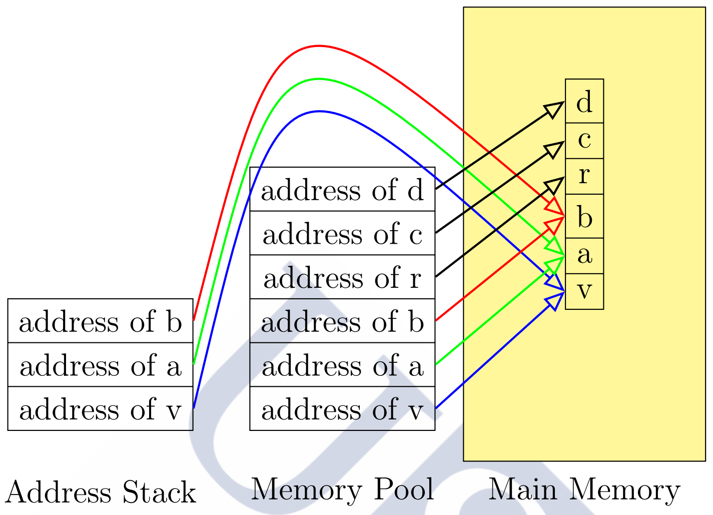


Figure 11.6: neg

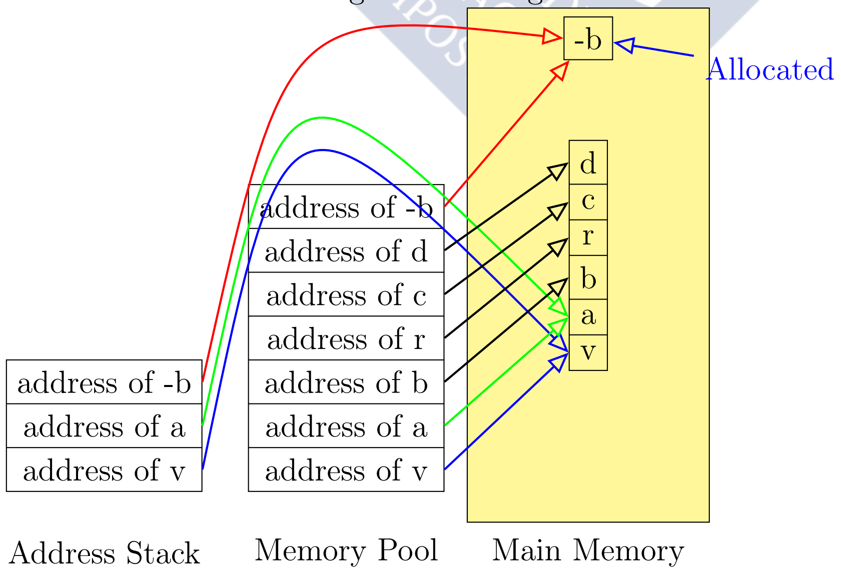


Figure 11.7: apush 3

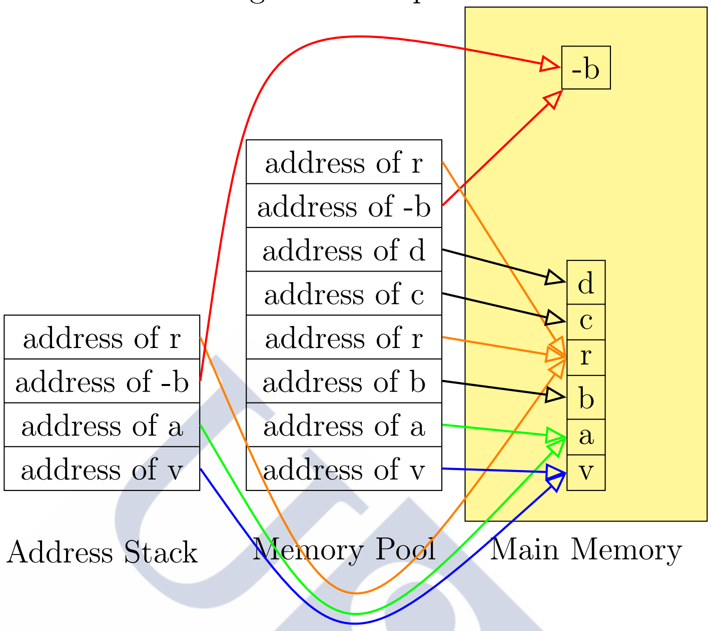
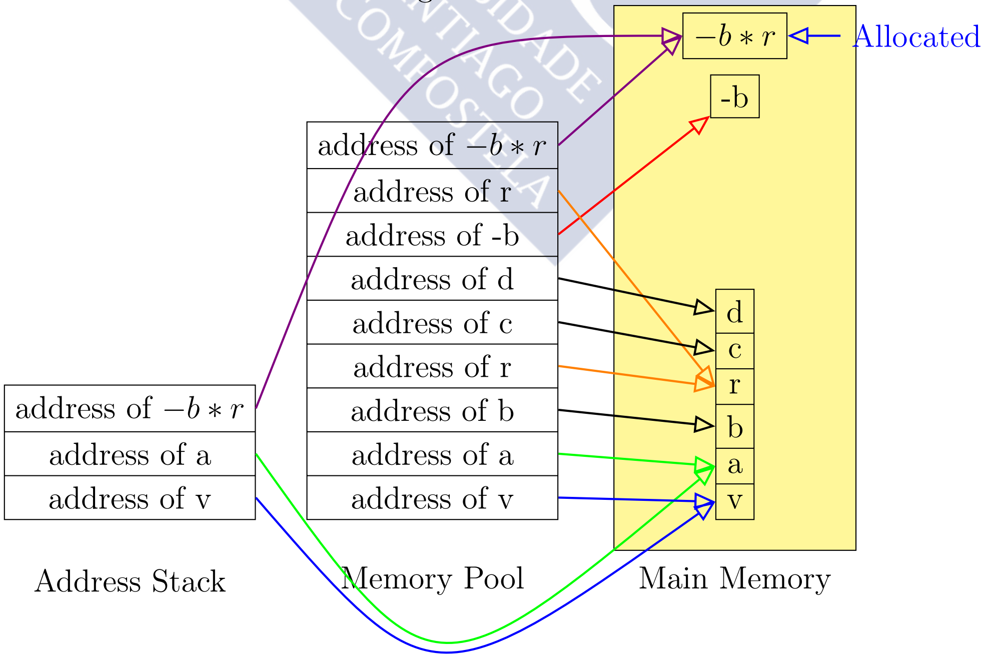


Figure 11.8: mult



12 Tools

Contrary to popular belief, Unix is user friendly. It just happens to be very selective about who it decides to make friends with.

Anonymous

12.1 needle

needle is a perl script used to distinguish different types of atoms, which are needed to calculate the different types of interactions between *Fragment A* and *Fragment B*.

```
$ needle -h
  needle v0.4
    (c) Roberto Rodriguez-Fernandez - 2010-2013
    collects sets of equivalent atoms
    input: any geometries input file
        -d       debug
        -p N    fragment A atoms
        -o      creates needed files
```

The atoms considered are: F, H, Si, O, N, S, C and Au. If any atom is different from those, it must be previously coded.

```
$ needle -p 18 moldeni.dat
  needle v0.2
    (c) Roberto Rodriguez-Fernandez - 2010-2013
    collects sets of equivalent atoms
    input: any geometries input file
```

```
Number(Atom)
1. 1(N)
2. 2(H) 4(H) 5(H)
3. 3(C)
4. 6(C)
5. 7(H) 8(H)
6. 9(O)
7. 10(N)
8. 11(H)
9. 12(C)
10. 13(C)
11. 14(H) 15(H)
12. 16(O)
13. 17(O)
14. 18(H)
15. 19(C) 22(C) 33(C) ...
16. 20(C) 21(C) 34(C) ...
17. 23(F) 24(F) 29(F) ...
18. 25(F) 26(F) 27(F) ...

Results:
1
2 4 5
3
6
7 8
9
10
11
12
13
14 15
16
17
18
19 22 33 ...
20 21 34 ...
23 24 29 ...
25 26 27 ...

Fragment A atoms:18
There are 18 different atom types. Fragment A:14, Fragment B:4, Common types:0
Total diff interactions: a vector of 56 coefs, X(k)
Vector Atom2Type:
Atom2Type(i)={1 2 3 2 2 4 ... 17 17 17 17 }
```

Options:

-d Debug output.

-p N Indicates the number of atoms into fragment A, required if **-o** is used.

-o Creates output files: *atom2type.txt* and *charges.txt* as a template to be modified as desired. Note that *charges.txt* assigns a dummy value of **0** to each type of atom, therefore the file must be manually edited or edited using *bedit* tool. See 6.3.

Notice that *needle* only reads the first molden geometry in the file, so its input can be the *geometries* file used for the job.

The algorithm used in *needle* is not bulletproof, so pay special attention to the *atom2type.txt* file.

12.2 bedit

bedit is an interactive editor to:

- edit atom types and charges
- edit bounds
- copy and clone bounds across the bounds vector

All options are self explained in the example below. The first option changes *atom2type* and *charges* files.

```
Bedit v0.2 Interactive editing
        (c) Roberto Rodriguez Fernandez 2010

====
actual interaction: [1]
====
a Interactive modify atom types
b Interactive modify bounds
c Interactive copy bounds
z Quit
====
Option:
```

```
Option:a
Type   Atoms
------ --------
    1.- N(1)
    2.- H(2) H(4) H(5)
    3.- C(3)
    4.- C(6)
    5.- H(7) H(8)
    6.- O(9)
    7.- N(10)
    8.- H(11)
    9.- C(12)
   10.- C(13)
   11.- H(14) H(15)
   12.- O(16)
   13.- O(17)
   14.- H(18)
   15.- C(19) C(22) C(33) ...
   16.- C(20) C(21) C(34) ...
   17.- F(23) F(24) F(29) ...
   18.- F(25) F(26) F(27) ...

use 'c3=766' to change atom 3 to type 766
```

```
     'c4=100 c5=1000' to change atom 3 to type 100
       and atom 5 to type 1000
     'q3=-0.33' to set type 3 charge to -0.33
     'w' to save types to file
     'z' to quit
Input:
```

The second option changes *bounds* file.

```
Option:b

INTERACTION TYPE 1
----------------------
N(1)-C(19) N(1)-C(22) N(1)-C(33) N(1)-C(40) N(1)-C(45) N(1)-C(49)
N(1)-C(56) N(1)-C(65) N(1)-C(72) N(1)-C(79) N(1)-C(84) N(1)-C(85)
N(1)-C(98) N(1)-C(101)
        Coefficients:
            1 A      -100.00000 -      +100.00000 (real)
            2 B        +0.00000 -      +100.00000 (real)
            3 C     -1500.00000 -     +5000.00000 (real)
            4 D        +3.50000 -        +5.50000 (real)
            5 E     -1500.00000 -     +1500.00000 (real)
            6 F        +5.50000 -        +9.50000 (real)

use 'a:l=1.03' to change A:lower to 1.03
    'c:u=1000' to change C:upper to 1000
    'b:l=100 b:u=1000' to change B:lower and B:upper
    'a:r c:i' to change A to real, C to integer
    'xx' to list and select interaction xx
         where xx is a number
    'w' to save to file (writing labels)
    'z' to quit
Input:
```

The third option can copy and clone the bounds across the *bounds* file.

```
Option:c

INTERACTION TYPE 1
----------------------
N(1)-C(19) N(1)-C(22) N(1)-C(33) N(1)-C(40) N(1)-C(45) N(1)-C(49)
N(1)-C(56) N(1)-C(65) N(1)-C(72) N(1)-C(79) N(1)-C(84) N(1)-C(85)
N(1)-C(98) N(1)-C(101)
        Coefficients:
            1 A      -100.00000 -      +100.00000 (real)
            2 B        +0.00000 -      +100.00000 (real)
            3 C     -1500.00000 -     +5000.00000 (real)
            4 D        +3.50000 -        +5.50000 (real)
            5 E     -1500.00000 -     +1500.00000 (real)
            6 F        +5.50000 -        +9.50000 (real)

use 'c1=2,3,4' to copy 1 to 2,3,4
    'c20=30,40,52 c21=31,41,53' to copy 20 to 30,40 and 52;
      and to copy 21 to 31,41 and 53 too
    'xx' to list interaction xx
         where xx is a number
    'w' to save to file (writing labels)
    'z' to quit'
Input:
```

bedit reads the *job.txt* file and follows the configuration therein.

Note that the last two options can change the *bounds* file, but they cannot modify the parameter **all coefficients** accordingly in the *job.txt* file.

Bedit changes slightly its behavior if an analytical expression is in use, as shown below.

```
Bedit v0.2 Interactive editing
        (c) Roberto Rodriguez Fernandez 2010

====
actual interaction: [1]
====
a Interactive modify atom types
b Interactive modify bounds
c Interactive copy bounds
z Quit
====
Option:b

INTERACTION TYPE 1
----------------------
C(1)-Xe(14)
        Coefficients:
            1(A)        aaa        +0.00000 -  +1000000.00000 (real)
            2(B)        bbb        +0.00000 -       +10.00000 (integer)
            3(C)         c1     -1500.00000 -        +0.00000 (real)
            4(D)         c2        +4.00000 -        +8.00000 (integer)
            5(E)         d1        +0.00000 -  +1000000.00000 (real)
            6(F)         d2        +0.00000 -       +10.00000 (integer)
            7(G)         e1     -1500.00000 -        +0.00000 (real)
            8(H)         e2        +4.00000 -        +8.00000 (integer)
                                   [l]ower          [u]pper

use 'a:l=1.03' to change A:lower to 1.03
    'c:u=1000' to change C:upper to 1000
    'b:l=100 b:u=1000' to change B:lower and B:upper
    'a:r c:i' to change A to real, C to integer
    'xx' to list and select interaction xx
         where xx is a number
    'w' to save to file (writing labels)
    'z' to quit
    NOTE: use the letters in parenthesis
      to select the desired coefficients
Input:
```

The coefficient names are printed, but the operative details remain the same using the A, B, C. . . coefficient letters to access them.

In case of an **external potential**, only the second option (*Interactive modify bounds*) is available.

12.3 fitview

An utility to write and plot data from results. **fitview** generates two files per plot, one contains the data (*file.dat*) and the other (*file.plt*) the **gnuplot**[1] commands to print out the plot. So to plot, you can type:

```
$gnuplot file.plt
```

The plots produced by fitview are one per two body interaction, a general evaluation including all geometries found in the geometry file and all the two body interactions in the same plot for a quick look:

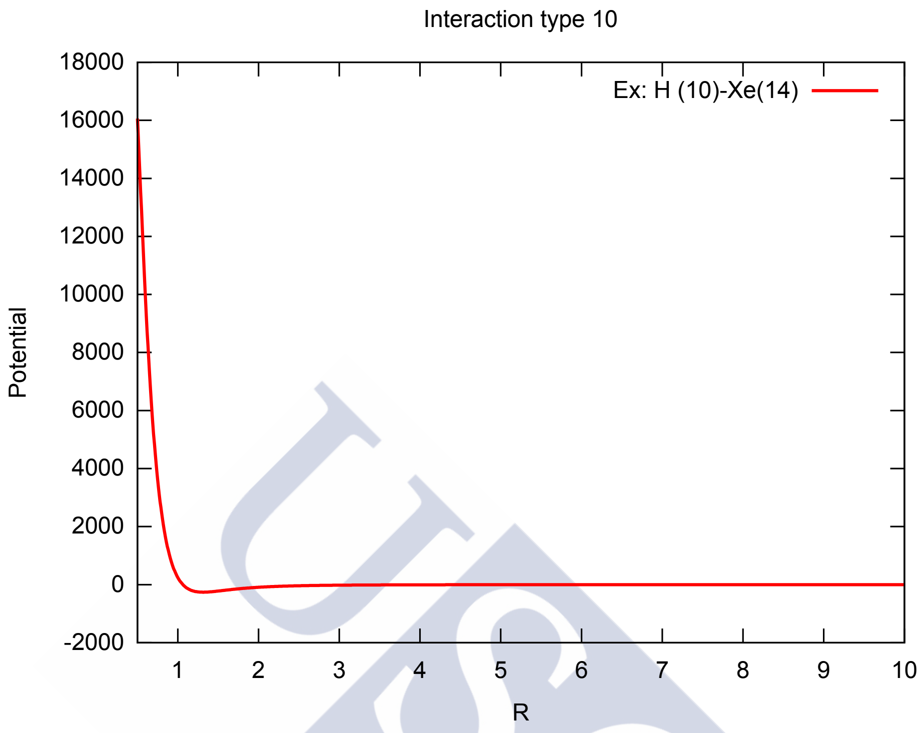
- general_evaluation.plt
- general_evaluation.dat
- 2body-type-1.dat
- 2body-type-1.plt
- 2body-type-2.dat
- 2body-type-2.plt
- ...
- 2body-type-n.dat
- 2body-type-n.plt
- 2body-type-all.plt

```
$ fitview -h
Usage: fitview [tag] [-l value] [-u value] [-d value] [-h]
             -l lower bound
             -u upper bound
             -d delta
             -e gnuplot supports enhanced terminal
             -h this help
             [tag] process this TAG
             default [0.500000,10.000000] delta: 0.010000
```

In the command line you can specify the *lower* and *upper bound*, the increment *delta* and whether your local version of **gnuplot** supports the *enhanced* terminal to print the subscripts needed for the data labels.

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

Figure 12.1: Two body interaction example plot.



fitview loads the *best.txt* coefficients and honors the job configuration found in the current working directory using the *job.txt* file therein.

If a **tag** is included in the command line, it processes the *best.**tag**.txt* and the output files overwrites the previous ones. Note that the result file names do not change.

In case of an *external potential*, **fitview** refuses to run.

12.4 ufpu

An utility to test analytical expressions configuration, following the next steps:

1. **ufpu** searches the *job file* in the current working directory for an **[analytical]** section[2].
2. Checks and validates the expression if found.
3. Compiles generating two files: *prog.uxe* and *prog.usm*, and extracts the variables to be used. *prog.uxe* is the packed bytecode result of

[2]Regardless the potential value in the **[job]** section.

compilation. *prog.usm* is the result assembler for the same expression.

4. Loads the *prog.uxe* file.

5. Asks for each variable.

6. Runs and shows the results.

7. Resets and goes to 5

The analytical subroutines do the same. At **GAFit** initialization, performs the steps from 1 through 4.

Each time a potential calculation is requested, it loads the **Fpu** with the appropriate values in a memory block, runs it, extracts the result and resets again the **Fpu**. See 11.

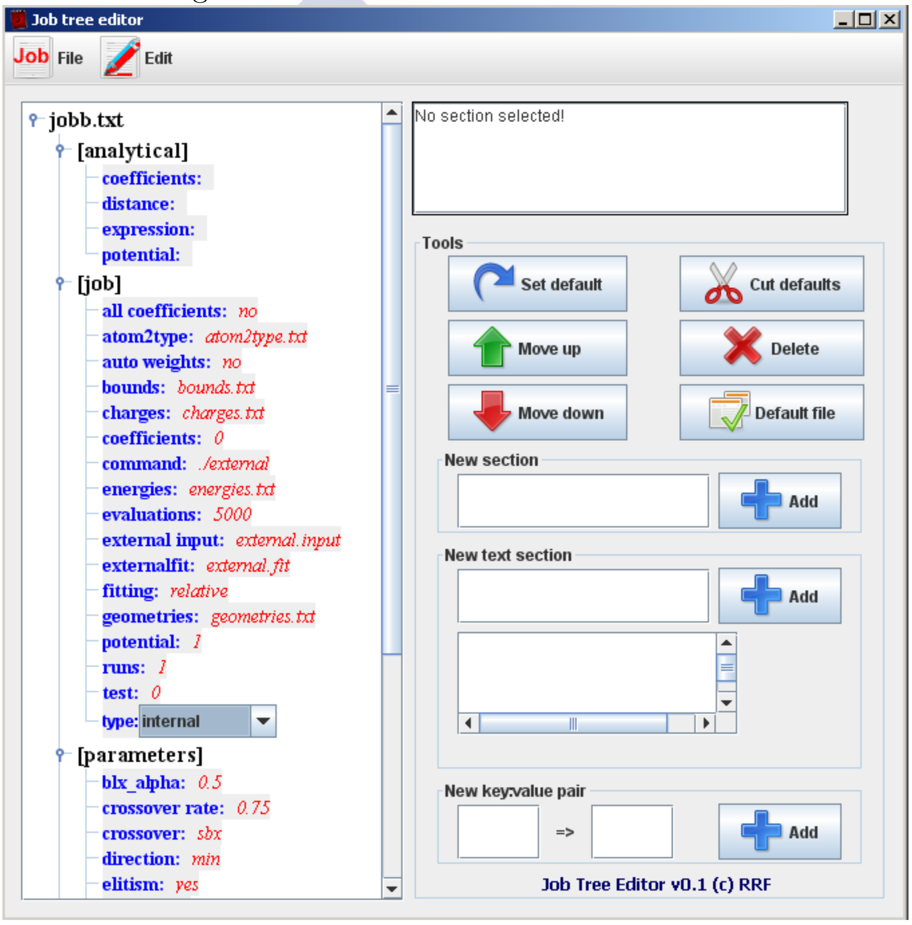
The output shown was generated using File 8.3.

```
uFpu v0.2 (c) Roberto Rodriguez-Fernandez

expression name: "potential 5"
potential:       pot
distance:        dist
coefficients:    aaa, bbb, c1, c2, d1, d2, e1, e2

Expression found:

        v1  =  aaa * exp (  -bbb * dist ) ;
        v2  =  c1 / pow ( dist , c2 ) ;
        v3  =  d1 / dist ** d2 ;
        v4  =  e1 / dist ^ e2 ;
        pot  =  v1 + v2 + v3 + v4

        Variables found in expression: v1 aaa bbb dist v2 c1 c2 v3 d1 d2 v4 e1 e2 pot
        Expression code OK
        pot index 13
        dist index 3
        8 coefficients found
INPUT
        distance variable (dist)=1
        coefficient aaa=1
        coefficient bbb=1
        coefficient c1=1
        coefficient c2=1
        coefficient d1=1
        coefficient d2=1
        coefficient e1=1
        coefficient e2=1

After run:      Memory (total used 27)  v1=0.367879 aaa=1.000000 bbb=1.000000
dist=1.000000 v2=1.000000 c1=1.000000 c2=1.000000 v3=1.000000 d1=1.000000
d2=1.000000 v4=1.000000 e1=1.000000 e2=1.000000 pot=3.367879

RESULT POTENTIAL:3.367879

Press 'q'/INTRO to quit, another key/INTRO to repeat
```

12.5 JobTreeEditor

The **JobTreeEditor** is a little Graphical User Interface (GUI) tool written in Java, useful to create and modify the *configuration and parameters* file: *job.txt* described in Section 6–. It can run in any Java correctly enabled graphical system: MS Windows or any Unix clone –Linux, BSD, Solaris, Mac OS X, etc–. **JobTreeEditor** takes special care with *defaults parameters* summarized in Table 6.1 and *potentials values* from Table 6.2.

Figure 12.2: Job Tree Editor main window.



```
jobb.txt
  [analytical]
    coefficients:
    distance:
    expression:
    potential:
  [job]
    all coefficients: no
    atom2type: atom2type.txt
    auto weights: no
    bounds: bounds.txt
    charges: charges.txt
    coefficients: 0
    command: ./external
    energies: energies.txt
    evaluations: 5000
    external input: external.input
    externalfit: external.fit
    fitting: relative
    geometries: geometries.txt
    potential: 1
    runs: 1
    test: 0
    type: internal
  [parameters]
    blx_alpha: 0.5
    crossover rate: 0.75
    crossover: sbx
    direction: min
    elitism: yes
```

```
No section selected!
```

The *configuration and parameters* file –*job.txt*– is displayed as a tree in the left panel, see Figure 12.2. The tree root is the file name to be edited: *jobb.txt* in this case as shown. In the right panel, there are a text information box, some buttons and text boxes to modify the tree in the selected section or key/value pair in the left panel.

From up to down, left to right:

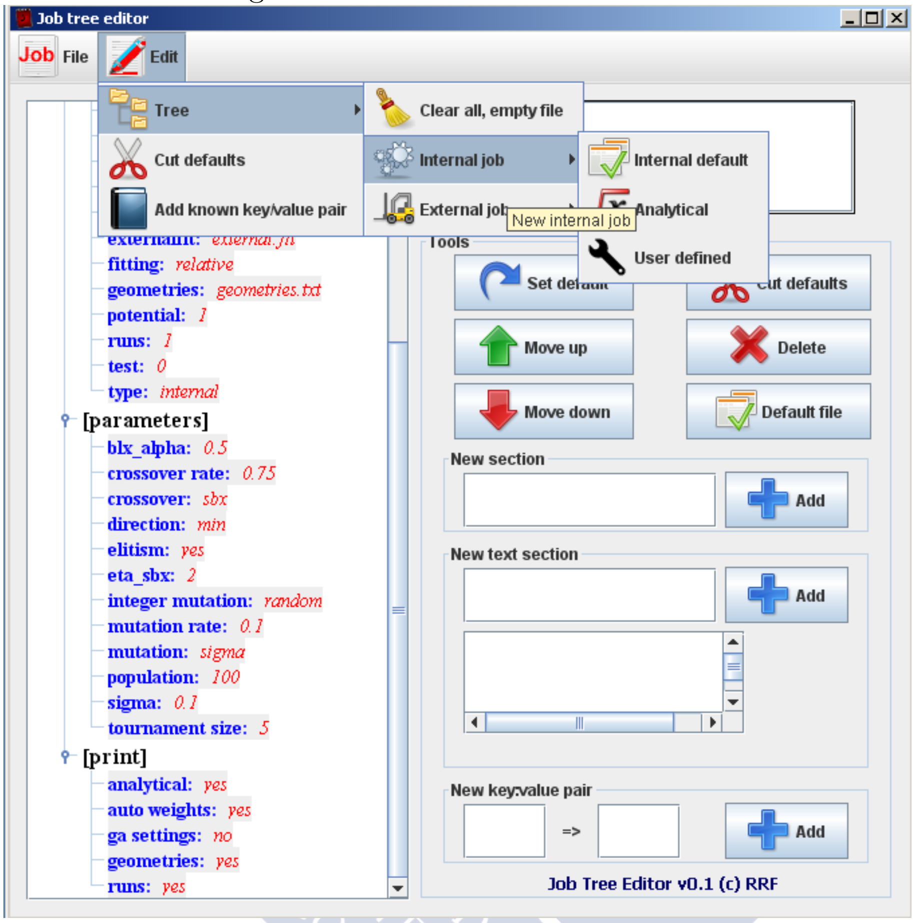
- Set default: Sets the selected key to its default value.
- Cut defaults: Deletes the key/value pairs from tree if the value is the default one. If the key is not present, **GAFit** takes its default value.
- Move up: Moves up a key/value pair or a whole section.
- Delete: Deletes key/value pair or a whole section. The key or the whole section keys take their default values.
- Move down: Moves down a key/value pair or a whole section.
- Default file: Clears all the tree and load a default one: an internal job with potential type 1 and all default values.

  After loading defaults, check the remaining tree inmediatly when you apply the Edit ≫ Cut Defaults option menu or Cut defaults button.
- **New section** Add: Adds a new empty section.
- **New text section** Add: Adds a new text section with the required text. Useful to introduce an analytical expression as potential.
- **New key:value pair** Add: Adds a new key/value pair to the selected section.

Also, there is a menu, see Figure 12.3:

- File ≫ Open file or alt + O: To open a file.
- File ≫ Save or alt + S: To save the current file.
- File ≫ Save as or alt + A: To save the current file changing its name.
- Edit ≫ Tree ≫ Clear all, empty file: To clear the whole tree.
- Edit ≫ Tree ≫ Internal job ≫ Internal default: Same as Default file button.
- Edit ≫ Tree ≫ Internal job ≫ Analytical: Defaults to use an analytical expression as potential. You must introduce the analytical expression by hand using the **new text section** Add button. Section 6.6, 8.2 and 14.

Figure 12.3: Job Tree Editor menu.



externalfit: external.fit
fitting: relative
geometries: geometries.txt
potential: 1
runs: 1
test: 0
type: internal
[parameters]
blx_alpha: 0.5
crossover rate: 0.75
crossover: sbx
direction: min
elitism: yes
eta_sbx: 2
integer mutation: random
mutation rate: 0.1
mutation: sigma
population: 100
sigma: 0.1
tournament size: 5
[print]
analytical: yes
auto weights: yes
ga settings: no
geometries: yes
runs: yes

- Edit 〉 Tree 〉 Internal job 〉 User defined: Defaults to use an user defined potential. You must introduce your potential changing the code: Check Section 8.1.4.

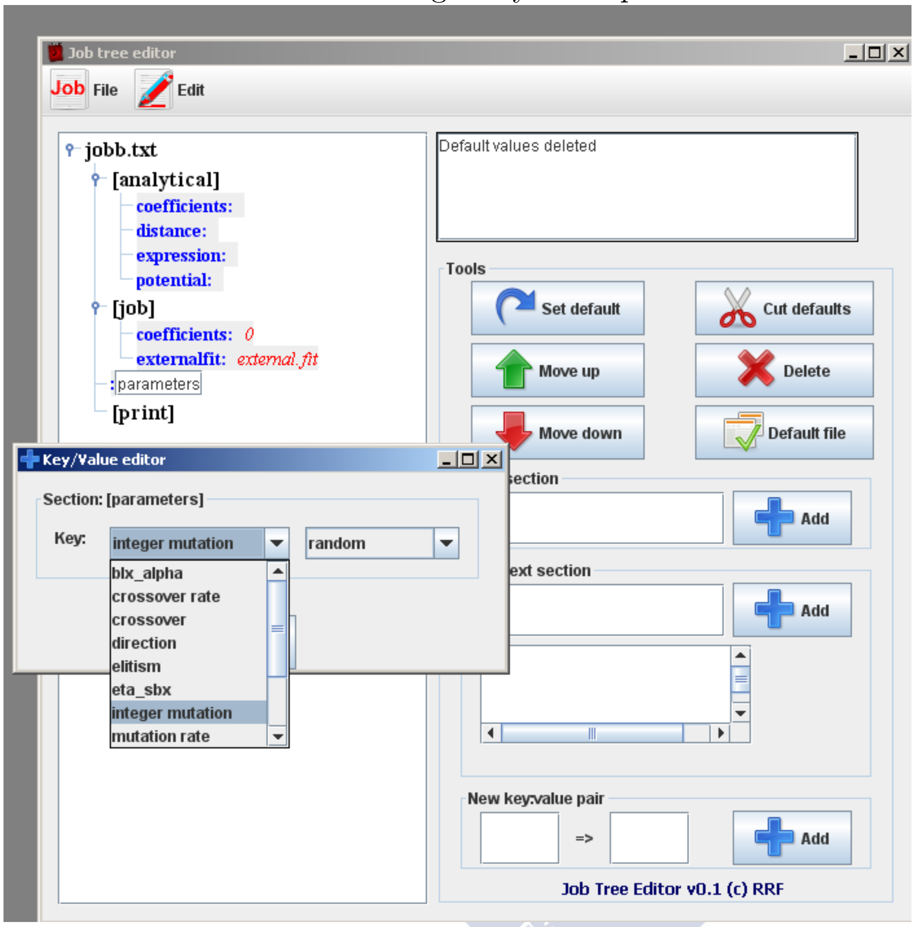
  Note that you can use a new potential adding your own one giving it a number different from those in Table 6.2 as shown in Section 8.1.3. In this case, also change the key **potential** to the newly defined potential.

- Edit 〉 Tree 〉 External job 〉 External, Edit 〉 Tree 〉 External job 〉 External bulk and Edit 〉 Tree 〉 External job 〉 External auto: External jobs configurations. Check Sections 9.1, 9, 10, 15, 16 and 17.

- Edit 〉 Cut Defaults: Same as Cut defaults button.

- Edit 〉 Add known key/value pair: Shows up a menu to pick up one of the not present keys with their default values to just edit and add to the selected section. See Figure 12.4.

Figure 12.4: Job Tree Editor editing a key-value pair with a context menu.



```
jobb.txt
  [analytical]
    coefficients:
    distance:
    expression:
    potential:
  [job]
    coefficients: 0
    externalfit: external.fit
  :parameters
  [print]
```

Default values deleted

Section: [parameters]

Key: integer mutation   random

```
blx_alpha
crossover rate
crossover
direction
elitism
eta_sbx
integer mutation
mutation rate
```

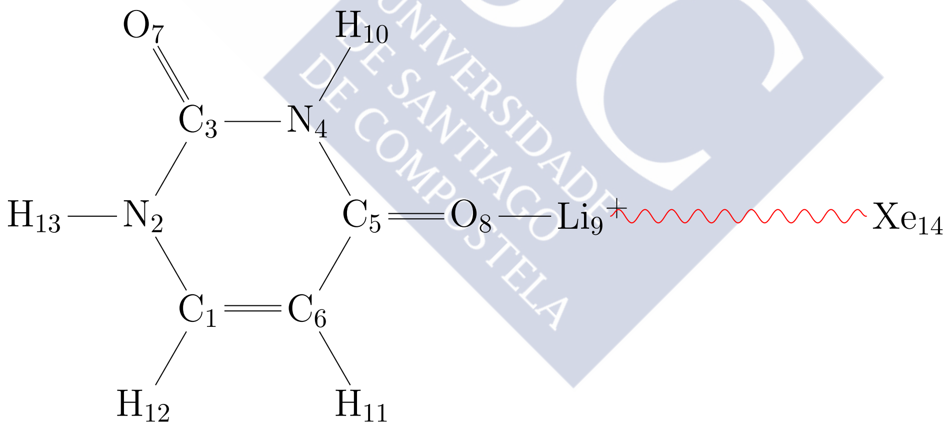
Part III

Step by step examples

13 Xe + $[Li(Uracil)]^+$ example

As a rule, software systems do not work well until they have been used, and have failed repeatedly, in real applications.

Dave Parnas



We shall use the **Xe + $[Li(Uracil)]^+$** system as an example taken from Section 18. In this example, we fit one of the potentials shown in Table 6.2 to the *interaction energies* between Xe and the $[Li(Uracil)]^+$complex, computed by *ab initio* calculations.

These files are included in the **uracil-example** folder. You can run it typing:

```
$ make test
```

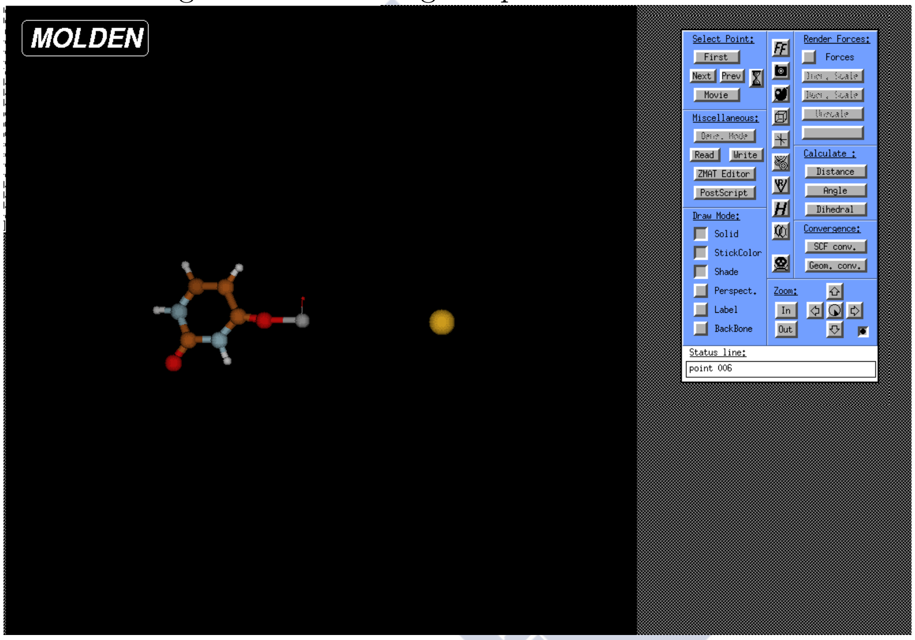
Once this command is employed, some files are extracted and **GAFit** is run.

13.1 Preparing input files

The input file *coord.molden* contains the set of geometries employed in the *ab initio* calculations. The geometries can be viewed using molden (see Fig. 13.1):

```
$ molden coord.molden
```

Figure 13.1: Viewing the points with Molden.



The very first lines of this file are shown in File 13.1.

File 13.1: coord.molden geometries file first lines.

```
 14

 C  0.000000  0.000000  0.000000
 N  0.000000  0.000000  1.354549
 C  1.152143  0.000000  2.127502
 N  2.311655  0.000000  1.343162
 C  2.393034  0.000000  −0.016579
 C  1.152592  0.000000  −0.718330
 O  1.169220  0.000000  3.330930
 O  3.523582  0.000000  −0.559509
 Li  4.968935  0.000000  −1.513449
 H  3.175968  0.000000  1.870824
 H  1.142155  0.000000  −1.793856
 H  −0.971622  0.000000  −0.471648
 H  −0.866367  0.000000  1.874333
 Xe  17.488048  0.000000  −9.776123
 14

 C  0.000000  0.000000  0.000000
 N  0.000000  0.000000  1.354549
 C  1.152143  0.000000  2.127502
 N  2.311655  0.000000  1.343162
 C  2.393034  0.000000  −0.016579
  [...]
```

Also, we need the *interaction energies* corresponding to each geometry in *coord.molden*. These energies are used to fit our model potential and they are listed in the file *energies.txt* (see File 13.2). This file follows the specifications described in 6.3.

File 13.2: energies.txt file.

```
−0.006436 1
−0.012603 1
−0.024660 1
−0.053662 1
−0.151027 1
−0.208324 1
−0.298249 1
−0.443987 1
−0.576097 1
−0.762092 1
−1.031527 1
−1.431174 1
−2.022694 1
−2.554913 1
−3.208230 1
−3.966854 1
−4.767595 1
−5.448579 1
−5.645469 1
−5.658691 1
−5.387761 1
−4.692701 1
−3.377588 1
−1.167944 1
2.322455 1
7.633202 1
15.516838 1
27.007602 1
66.979582 1
146.056144 1
297.072019 1
```

There are two columns, the first one is the *interaction energies* and the second one is the *weight* of each geometry. The order must be the same of the *geometries* file.

Taking into account that some of the atoms in the $[Li(Uracil)]^+$complex (*Fragment A* below) can be equivalent, we have to determine the different atom types. To achieve this, we shall use the *needle* tool–see 12.1–.

```
$ needle -p 13 -o coord.molden

[...]

Fragment A atoms:13
There are 14 different atom types. Fragment A:13,
Fragment B:1, Common types:0
Total diff interactions: a vector of 13 coefs, X(k)
Vector Atom2Type:
Atom2Type(i)={1 2 3 4 5 6 7 8 9 10 11 12 13 14 }
two files created: atom2type.txt and charges.txt
```

When we run *needle* using the **-p** and **-o** switches, we have to provide the number of atoms present in *fragment A*. Additionally, with these options *needle* creates the *atom2type.txt* –File 13.3– and *charges.txt*–File 13.4– files (see section 6.3). As seen above, the output informs that, in this case, there are no equivalent atoms. In our example, there are 14 different atom types, and 13 different interactions between *fragment A* and *fragment B* (Xe)

File 13.3: atom2type.txt file.

```
 13  14
     1    C      1
     2    N      2
     3    C      3
     4    N      4
     5    C      5
     6    C      6
     7    O      7
     8    O      8
     9   LI      9
    10    H     10
    11    H     11
    12    H     12
    13    H     13
    14   XE     14
```

The number of different types of atoms determines the *charges.txt* file with a line per atom type. The generated *charges.txt* file is a dummy file to be used as a template and you need to edit it, if you use a potential with charges.

File 13.4: charges.txt file.

```
     1   0.000000
     2   0.000000
     3   0.000000
     4   0.000000
     5   0.000000
     6   0.000000
     7   0.000000
     8   0.000000
     9   0.000000
    10   0.000000
    11   0.000000
    12   0.000000
    13   0.000000
    14   0.000000
```

We shall use the implemented potential number 1 with four coefficients –from Table 6.2–.

$$V = Ae^{-Br} + \frac{C}{r^D}$$

So we need a file with the lower and upper limits of the coefficients –the bounds–. Here we can specify the same limits for all interactions or different limits per each interaction. We choose the former option, as shown in File 13.5.

File 13.5: bounds.txt file.

```
TEXT TEXT TEXT TEXT
           0.   1000000.        0
           0.        10.0       1
       -1500.         0.        0
           4.0        8.0       1
```

To edit the atom2type and the bounds file we can use the **bedit** tool –12.2– using option a. Option b is employed to edit the *bounds.txt* file, and option c is needed when we want to specify different bounds for the parameters corresponding to the different interaction types.

```
$ bedit

Bedit v0.2 Interactive editing
        (c) Roberto Rodriguez Fernandez 2010


Job type: internal
====
actual interaction: [1]
====
a Interactive modify atom types
b Interactive modify bounds
c Interactive copy bounds
z Quit
====
Option:a
Type   Atoms
------ --------
    1.- C(1)
    2.- N(2)
    3.- C(3)
    4.- N(4)
    5.- C(5)
    6.- C(6)
    7.- O(7)
    8.- O(8)
    9.- Li(9)
   10.- H(10)
   11.- H(11)
   12.- H(12)
   13.- H(13)
   14.- Xe(14)

use 'c3=766' to change atom 3 to type 766
    'c4=100 c5=1000' to change atom 3 to type 100
      and atom 5 to type 1000
    'q3=-0.33' to set type 3 charge to -0.33
    'w' to save types to file
    'z' to quit

Input:z
====
actual interaction: [1]
====
a Interactive modify atom types
b Interactive modify bounds
c Interactive copy bounds
z Quit
====
Option:b

INTERACTION TYPE 1
```

```
----------------------
C(1)-Xe(14)
        Coefficients:
            1(A)          A        +0.00000 -   +1000000.00000 (real)
            2(B)          B        +0.00000 -        +10.00000 (integer)
            3(C)          C     -1500.00000 -         +0.00000 (real)
            4(D)          D        +4.00000 -         +8.00000 (integer)
                       [l]ower            [u]pper

use 'a:l=1.03' to change A:lower to 1.03
    'c:u=1000' to change C:upper to 1000
    'b:l=100 b:u=1000' to change B:lower and B:upper
    'a:r c:i' to change A to real, C to integer
    'xx' to list and select interaction xx
         where xx is a number
    'w' to save to file (writing labels)
    'z' to quit
Input:
```

Next, we have to edit the *job.txt* file to configure **GAFit**. The file *job.txt* that comes with the uracil example is the one shown in the File 13.6.

File 13.6: job.txt file.

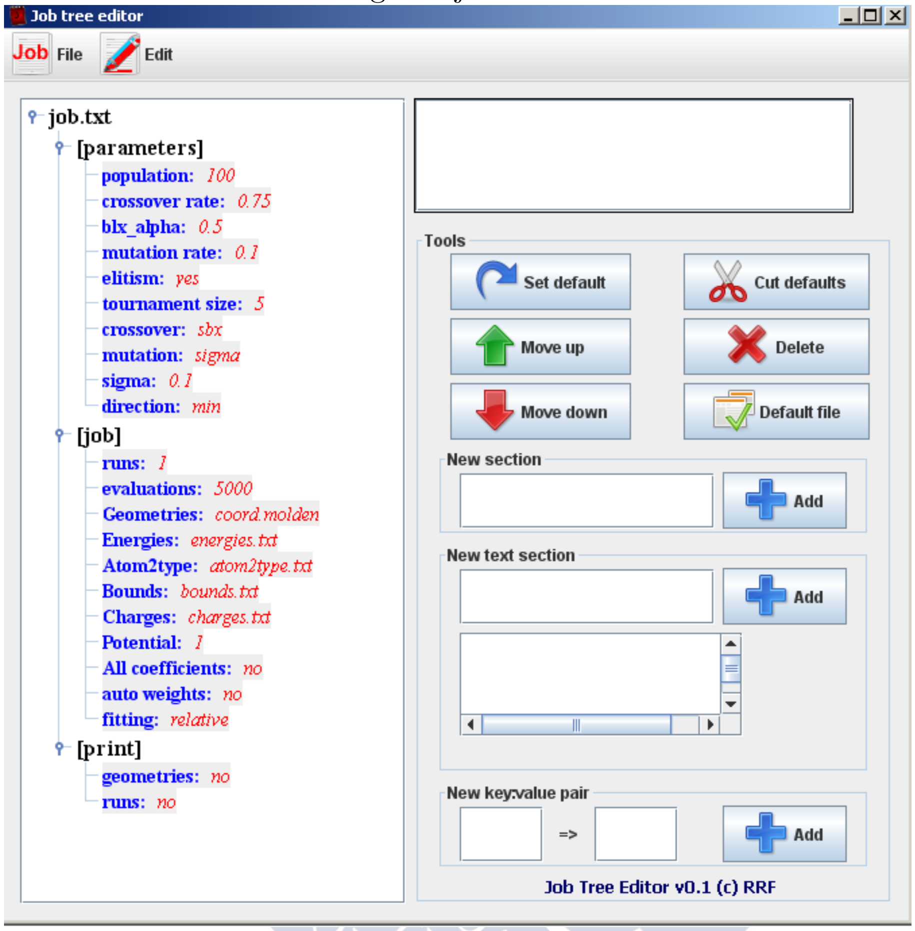
```
[parameters]
population:        100
crossover rate:    0.75
blx_alpha:         0.5
mutation rate:     0.1
elitism:           yes
tournament size:   5
crossover:         sbx
mutation:          sigma
sigma:             0.1
direction:         min

[job]
runs:               1
evaluations:        5000
Geometries:     coord.molden
Energies:    energies.txt
Atom2type:  atom2type.txt
Bounds:  bounds.txt
Charges:  charges.txt
Potential:  1
All coefficients:  no
auto weights:  no
fitting:  relative

[print]
geometries:  no
runs:  no
```

job.txt is split in some sections, the text between square brackets, with options as key-value pairs.

You can also construct this file using the **JobTreeEditor** tool, section 12.5, and use the menu command Edit ≫ Tree ≫ Internal job ≫ Internal default, Figure 13.2, and change some defaults for this job:

- section [parameters]: none to change.
- section [job]: Change the **geometries** key to *coord.molden*.

Figure 13.2: Job Tree Editor editing the 'job.txt' file included in the uracil example.



```
job.txt
  [parameters]
    population: 100
    crossover rate: 0.75
    blx_alpha: 0.5
    mutation rate: 0.1
    elitism: yes
    tournament size: 5
    crossover: sbx
    mutation: sigma
    sigma: 0.1
    direction: min
  [job]
    runs: 1
    evaluations: 5000
    Geometries: coord.molden
    Energies: energies.txt
    Atom2type: atom2type.txt
    Bounds: bounds.txt
    Charges: charges.txt
    Potential: 1
    All coefficients: no
    auto weights: no
    fitting: relative
  [print]
    geometries: no
    runs: no
```

- section [print]: Change the keys **geometries** and **runs** to *no*.

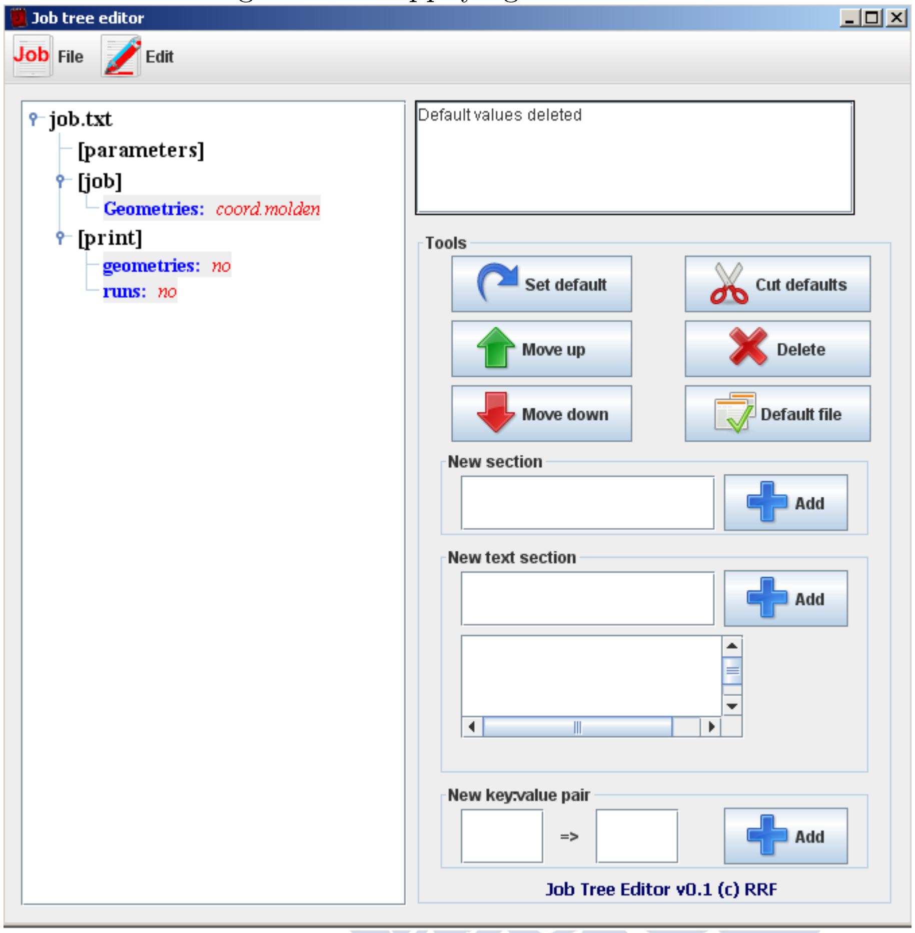
If you select the option Edit ⟩ Cut defaults you get a "standard" file where some options are omitted because they are assigned default values, as you can see in Figure 13.3

The different sections and their possible options are discussed in section 6. In the **[job]** section we have **potential: 1** and **All coefficients: no**.

As you can see in Table 6.2, this potential function has a total of 4 coefficients and we want the same bounds (**All coefficients: no**) for all two-body interactions. This is specified in the *bounds.txt* shown in File 13.5, with only 4 lower and 4 upper bounds for the coefficients.

The last column of this file is employed to specify whether the coefficient is an integer or a real number.

Figure 13.3: Appliying 'cut defaults'.



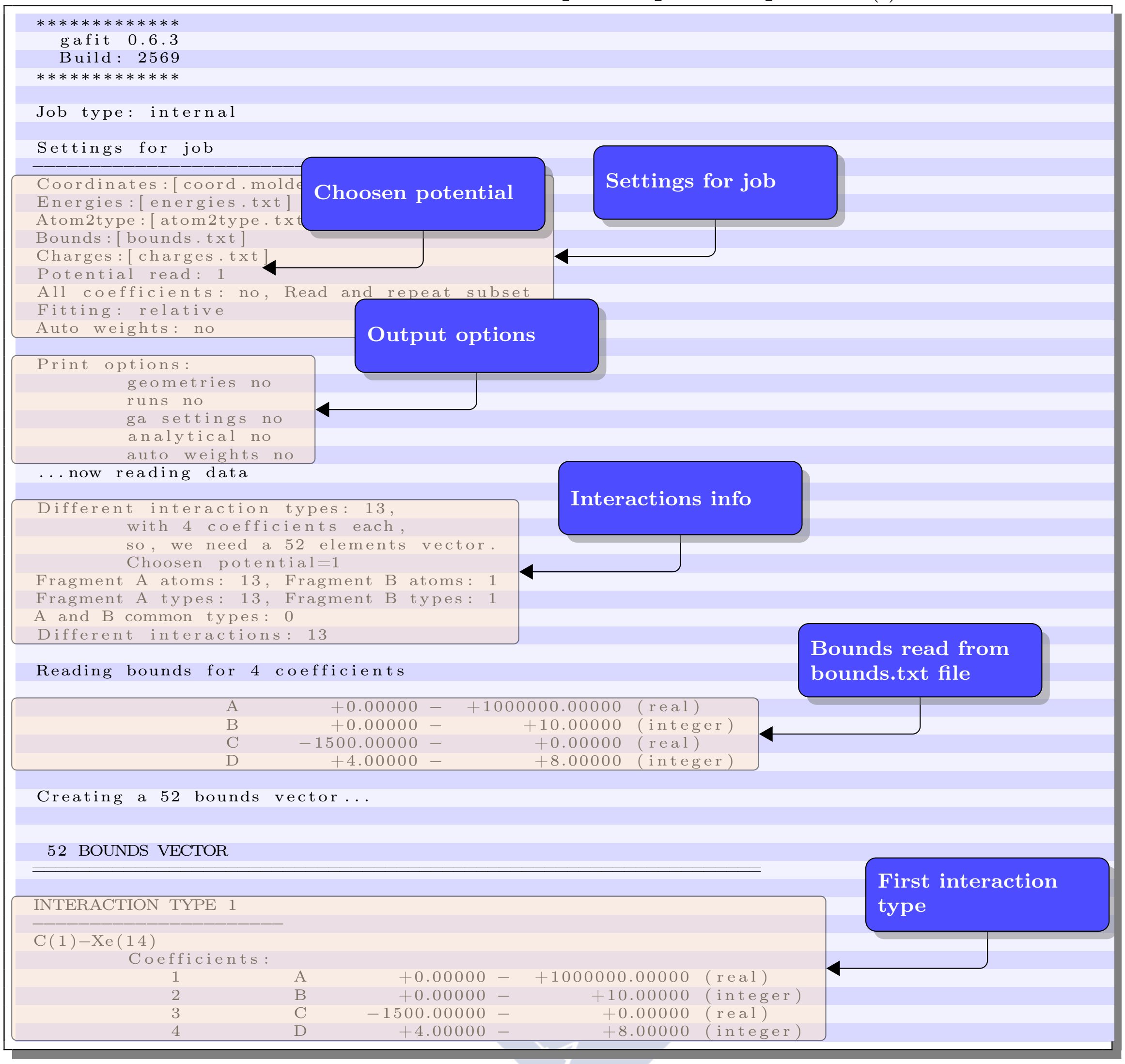
13.2 Running the example

If you run **GAFit** from the folder where all the above files are located you get the output file shown in Files 13.7, 13.8, 13.9, 13.10 and 13.11.

```
$ gafit > output.txt
```

As we mentioned above, there are 13 different two-body interactions with four coefficients each one, so we have a vector of 52 coefficients to optimize. Two of the coefficients, B and D, are integer, as indicated in File 13.5.

File 13.7: Uracil example output: output.txt (i)


Choosen potential
Settings for job
Output options
Interactions info
Bounds read from bounds.txt file
First interaction type

```
*************
   gafit 0.6.3
   Build: 2569
*************

Job type: internal

Settings for job
------------------------------
Coordinates:[coord.molde
Energies:[energies.txt]
Atom2type:[atom2type.txt
Bounds:[bounds.txt]
Charges:[charges.txt]
Potential read: 1
All coefficients: no, Read and repeat subset
Fitting: relative
Auto weights: no

Print options:
            geometries no
            runs no
            ga settings no
            analytical no
            auto weights no
...now reading data

Different interaction types: 13,
            with 4 coefficients each,
            so, we need a 52 elements vector.
            Choosen potential=1
Fragment A atoms: 13, Fragment B atoms: 1
Fragment A types: 13, Fragment B types: 1
A and B common types: 0
Different interactions: 13

Reading bounds for 4 coefficients

                         A          +0.00000 -  +1000000.00000 (real)
                         B          +0.00000 -       +10.00000 (integer)
                         C       -1500.00000 -        +0.00000 (real)
                         D          +4.00000 -        +8.00000 (integer)

Creating a 52 bounds vector...


 52 BOUNDS VECTOR
================================================================

INTERACTION TYPE 1
--------------------------------
C(1)-Xe(14)
            Coefficients:
                 1            A          +0.00000 -  +1000000.00000 (real)
                 2            B          +0.00000 -       +10.00000 (integer)
                 3            C       -1500.00000 -        +0.00000 (real)
                 4            D          +4.00000 -        +8.00000 (integer)
```

File 13.8: Uracil example output: output.txt (ii)

```

INTERACTION TYPE 2
--------------------------------
N(2)-Xe(14)
            Coefficients:
                 5            A          +0.00000 -  +1000000.00000 (real)
                 6            B          +0.00000 -       +10.00000 (integer)
                 7            C       -1500.00000 -        +0.00000 (real)
                 8            D          +4.00000 -        +8.00000 (integer)

INTERACTION TYPE 3
--------------------------------
C(3)-Xe(14)
            Coefficients:
                 9            A          +0.00000 -  +1000000.00000 (real)
                10            B          +0.00000 -       +10.00000 (integer)
                11            C       -1500.00000 -        +0.00000 (real)
                12            D          +4.00000 -        +8.00000 (integer)

INTERACTION TYPE 4
--------------------------------
N(4)-Xe(14)
            Coefficients:
                13            A          +0.00000 -  +1000000.00000 (real)
```

```
              14          B        +0.00000 -        +10.00000 (integer)
              15          C     -1500.00000 -         +0.00000 (real)
              16          D        +4.00000 -         +8.00000 (integer)

INTERACTION TYPE 5
-----------------------
C(5)-Xe(14)
         Coefficients:
              17          A        +0.00000 -   +1000000.00000 (real)
              18          B        +0.00000 -        +10.00000 (integer)
              19          C     -1500.00000 -         +0.00000 (real)
              20          D        +4.00000 -         +8.00000 (integer)

INTERACTION TYPE 6
-----------------------
C(6)-Xe(14)
         Coefficients:
              21          A        +0.00000 -   +1000000.00000 (real)
              22          B        +0.00000 -        +10.00000 (integer)
              23          C     -1500.00000 -         +0.00000 (real)
              24          D        +4.00000 -         +8.00000 (integer)

INTERACTION TYPE 7
-----------------------
O(7)-Xe(14)
         Coefficients:
              25          A        +0.00000 -   +1000000.00000 (real)
              26          B        +0.00000 -        +10.00000 (integer)
              27          C     -1500.00000 -         +0.00000 (real)
              28          D        +4.00000 -         +8.00000 (integer)

INTERACTION TYPE 8
-----------------------
O(8)-Xe(14)
         Coefficients:
              29          A        +0.00000 -   +1000000.00000 (real)
              30          B        +0.00000 -        +10.00000 (integer)
              31          C     -1500.00000 -         +0.00000 (real)
              32          D        +4.00000 -         +8.00000 (integer)

INTERACTION TYPE 9
-----------------------
 Li(9)-Xe(14)
         Coefficients:
              33          A        +0.00000 -   +1000000.00000 (real)
              34          B        +0.00000 -        +10.00000 (integer)
              35          C     -1500.00000 -         +0.00000 (real)
              36          D        +4.00000 -         +8.00000 (integer)

INTERACTION TYPE 10
-----------------------
H(10)-Xe(14)
         Coefficients:
              37          A        +0.00000 -   +1000000.00000 (real)
              38          B        +0.00000 -        +10.00000 (integer)
              39          C     -1500.00000 -         +0.00000 (real)
              40          D        +4.00000 -         +8.00000 (integer)

INTERACTION TYPE 11
-----------------------
H(11)-Xe(14)
         Coefficients:
```

In the output, next lines explain how the interactions are and their per coefficient bounds. In this case, the bounds are equal for any interaction.

File 13.9: Uracil example output: output.txt (iii)

```
               41             A          +0.00000 –    +1000000.00000 (real)
               42             B          +0.00000 –         +10.00000 (integer)
               43             C       –1500.00000 –          +0.00000 (real)
               44             D          +4.00000 –          +8.00000 (integer)

 INTERACTION TYPE 12
 ------------------------
 H(12)–Xe(14)
           Coefficients:
               45             A          +0.00000 –    +1000000.00000 (real)
               46             B          +0.00000 –         +10.00000 (integer)
               47             C       –1500.00000 –          +0.00000 (real)
               48             D          +4.00000 –          +8.00000 (integer)

 INTERACTION TYPE 13
 ------------------------
 H(13)–Xe(14)
           Coefficients:
               49             A                              00.00000 (real)
               50             B                              10.00000 (integer)
               51             C                              +0.00000 (real)
               52             D          +4.00000 –          +8.00000 (integer)

 #seed#1380143828#seed#
```

The seed used in this calculation

Here begins results

```
 #
 #Results
 #

 INTERACTION TYPE 1
 ------------------------
 C(1)–Xe(14)
           Coefficients:
                1 A   +671108.3835272372
                2 B         +5.0000000000
                3 C      –480.5115189276
                4 D         +8.000000000
```

Second interaction type results

```
 INTERACTION TYPE 2
 ------------------------
 N(2)–Xe(14)
           Coefficients:
                5 A   +807732.6068476612
                6 B         +4.0000000000
                7 C      –363.5238706214
                8 D         +6.0000000000

 INTERACTION TYPE 3
 ------------------------
 C(3)–Xe(14)
           Coefficients:
                9 A   +501056.2925643864
               10 B         +4.0000000000
               11 C      –522.8650438224
               12 D         +7.0000000000

 INTERACTION TYPE 4
 ------------------------
 N(4)–Xe(14)
           Coefficients:
               13 A   +441674.4181581688
               14 B        +10.0000000000
               15 C      –554.1423215544
               16 D         +6.0000000000

 INTERACTION TYPE 5
 ------------------------
 C(5)–Xe(14)
           Coefficients:
               17 A  +1000000.0000000000
               18 B         +8.0000000000
               19 C     –1271.2989060403
               20 D         +5.0000000000

 INTERACTION TYPE 6
```

The calculations employ random numbers, so if you take the same seed used in a given run, you will reproduce the whole output. The details are in Section 6.3 and it is useful for testing –for example, adjusting weights– and debugging purposes. Each interaction with the coefficients found are printed. This is the information saved in the file *best.txt*.

File 13.10: Uracil example output: output.txt (iv)

```
INTERACTION TYPE 6
————————————————————————
C(6)−Xe(14)
        Coefficients:
           21 A    +912220.2325137885
           22 B         +7.0000000000
           23 C      −1307.7763470169
           24 D         +7.0000000000

INTERACTION TYPE 7
————————————————————————
O(7)−Xe(14)
        Coefficients:
           25 A     +83922.2858810042
           26 B         +8.0000000000
           27 C       −491.9348306627
           28 D         +6.0000000000

INTERACTION TYPE 8
————————————————————————
O(8)−Xe(14)
        Coefficients:
           29 A    +463280.5928098704
           30 B         +4.0000000000
           31 C       −724.6517550236
           32 D         +4.0000000000

INTERACTION TYPE 9
————————————————————————
 Li(9)−Xe(14)
        Coefficients:
           33 A    +758522.1087732586
           34 B         +5.0000000000
           35 C      −1268.3315068897
           36 D         +7.0000000000

INTERACTION TYPE 10
————————————————————————
H(10)−Xe(14)
        Coefficients:
           37 A     +71043.3370584520
           38 B         +7.0000000000
           39 C       −213.4004863545
           40 D         +6.0000000000

INTERACTION TYPE 11
————————————————————————
H(11)−Xe(14)
        Coefficients:
           41 A    +923403.5894252134
           42 B         +8.0000000000
           43 C      −1142.7749517176
           44 D         +5.0000000000

INTERACTION TYPE 12
————————————————————————
H(12)−Xe(14)
        Coefficients:
           45 A    +586877.5618903312
           46 B         +6.0000000000
           47 C       −673.3113060535
           48 D         +8.0000000000
```

Finally, an objective function is calculated for each geometry:

$$\textbf{Difference} = \frac{(\textbf{Calculated} - \textbf{Energy})}{\textbf{Energy}} * 100$$

Where *Calculated* is the energy calculated using the *best.txt* coefficients, and the geometry energy –*Energy*– from the file *energies.txt*.

File 13.11: Uracil example output: output.txt (v)

```
INTERACTION TYPE 13
--------------------
H(13)-Xe(14)
           Coefficients:
              49 A    +347409.2736768266
              50 B         +10.0000000000
              51 C       -941.8487944828
              52 D          +8.0000000000
#
#Evaluation
#
#Geometry       Energy            Calculated          Difference        Weight
#=========     =======           ===========         ============      =======
       1     -0.006436000000    -0.010530628758    +63.62 %         +1.00
       2     -0.012603000000    -0.017699130419    +40.44 %         +1.00
       3     -0.024660000000    -0.032091913786    +30.14 %         +1.00
       4     -0.053662000000    -0.064429793016    +20.07 %         +1.00
       5     -0.151027000000    -0.149580503099     -0.96 %         +1.00
       6     -0.208324000000    -0.190470092759     -8.57 %         +1.00
       7     -0.298249000000    -0.246469158730    -17.36 %         +1.00
       8     -0.443987000000    -0.325030021284    -26.79 %         +1.00
       9     -0.576097000000    -0.387854637179    -32.68 %         +1.00
      10     -0.762092000000    -0.467207988967    -38.69 %         +1.00
      11     -1.031527000000    -0.569012154888    -44.84 %         +1.00
      12     -1.431174000000    -0.702078775309    -50.94 %         +1.00
      13     -2.022694000000    -0.879834493862    -56.50 %         +1.00
      14     -2.554913000000    -1.032893122931    -59.57 %         +1.00
      15     -3.208230000000    -1.222805807243    -61.89 %         +1.00
      16     -3.966854000000    -1.458467698450    -63.23 %         +1.00
      17     -4.767595000000    -1.745074243069    -63.40 %         +1.00
      18     -5.448579000000    -2.068595213586    -62.03 %         +1.00
      19     -5.645469000000    -2.224172704913    -60.60 %         +1.00
      20     -5.658691000000    -2.347883233893    -58.51 %         +1.00
      21     -5.387761000000    -2.396779068486    -55.51 %         +1.00
      22     -4.692701000000    -2.294965536523    -51.09 %         +1.00
      23     -3.377588000000    -1.910862831798    -43.43 %         +1.00
      24     -1.167944000000    -1.019365059710    -12.72 %         +1.00
      25     +2.322455000000    +0.760929895293    -67.24 %         +1.00
      26     +7.633202000000    +4.071415893533    -46.66 %         +1.00
      27     +15.516838000000    +9.984437384988    -35.65 %         +1.00
      28     +27.007602000000    +20.283675639687    -24.90 %         +1.00
      29     +66.979582000000    +67.773763282783    +1.19 %         +1.00
      30     +146.056144000000    +201.103957775714    +37.69 %         +1.00
      31     +297.072019000000    +564.308380397447    +89.96 %         +1.00
```

Geometry fit evaluation

13.3 Examining results

The best individual from the program run is stored in the file *best.txt*, File 13.13. You must save this file, because it is overwritten in each run, and it is used to load coefficients by some tools. The last line of the file shows the above objective function calculated with the best coefficients. Executing the *fitview* tool in the same folder, it reads the configuration and the *best.txt* file creating some useful graphs. See 12.3.

File 13.12: 2body-type-2.plt

```
set terminal x11
set title "Interaction type 2"
set xrange [0.500000:10.000000]
set xlabel "R"
set ylabel "Potential"
plot "2body-type-2.dat" using 1:2 title "Ex: N (2)-Xe(14)" with linespoints
pause -1
```

Files 13.12 and 13.14 are the **gnuplot** commands and data file, repectivelly, to plot *Potential* vs r for the *interaction type 2* between N(2) and Xe(14), Figure 13.4.

File 13.13: Uracil example best.txt

```
671108.383527237223
5.000000000000
-480.511518927649
8.000000000000
807732.6068476612
4.0000000000
363.5238706214
6.0000000000
501056.292564386385
4.000000000000
-522.865043822352
7.000000000000
441674.418158168846
10.000000000000
-554.142321554391
6.000000000000
1000000.000000000000
8.000000000000
-1271.298906040291
5.000000000000
912220.232513788505
7.000000000000
-1307.776347016855
7.000000000000
83922.285881004180
8.000000000000
-491.934830662740
6.000000000000
463280.592809870373
4.000000000000
-724.651755023580
4.000000000000
758522.108773258631
5.000000000000
-1268.331506889747
7.000000000000
71043.337058452002
7.000000000000
-213.400486354491
6.000000000000
923403.589425213402
8.000000000000
-1142.774951717634
5.000000000000
586877.561890331213
6.000000000000
-673.311306053478
8.000000000000
347409.273676826560
10.000000000000
-941.848794482794
8.000000000000

Fitness:  7.063407502683
```

A, B, C, D for interaction type 2, N(2)-Xe(14)

Result from evaluate this coefficients set

In File 13.12 you can change, for example, *set terminal x11* with *set terminal svg* and add a line with *set output "plot.svg"*. Next, you can run:

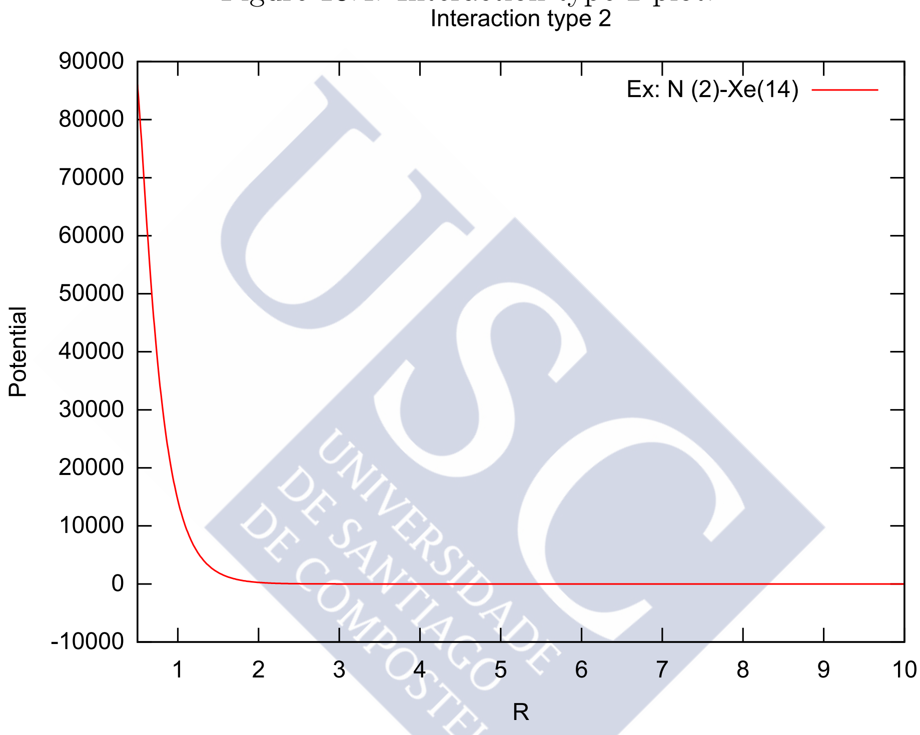
```
$ gnuplot 2body-type-2.plt
```

to obtain a *svg graphic file* named *plot.svg* like Figure 13.4.

File 13.14: 2body-type-2.dat

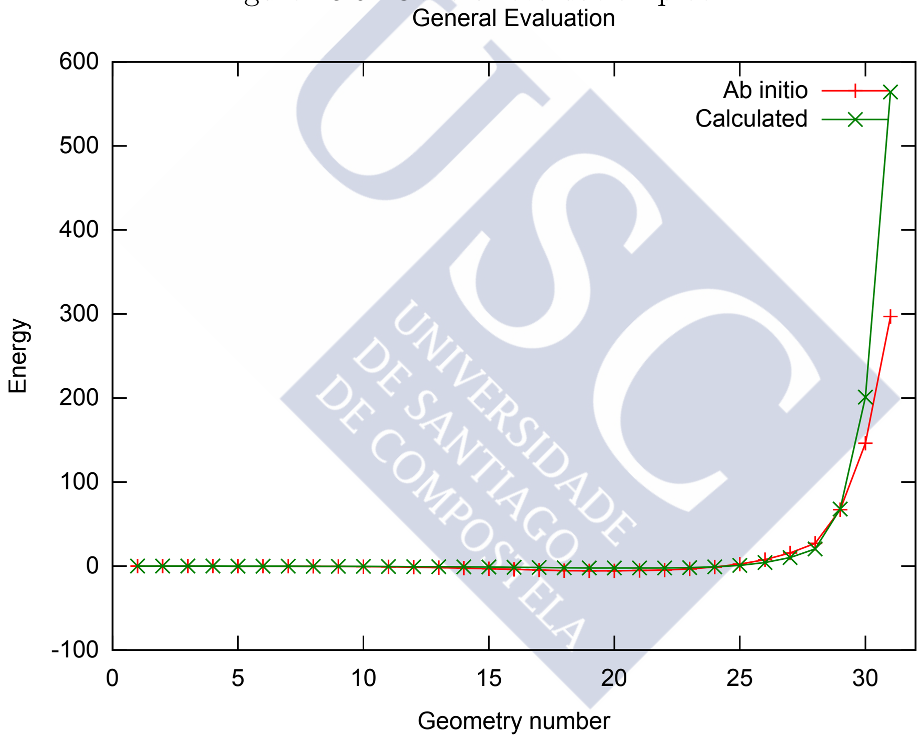
```
#
#INTERACTION TYPE 2
#-----------------------
# N(2)-Xe(14)
#          Coefficients:
#               5 A    +807732.6068476612
#               6 B          +4.0000000000
#               7 C        -363.5238706214
#              8  D            +6.0000000000
#
#                        r                         V
                 +0.5000000000          +86049.1934074035
                 +0.5100000000          +84369.3067962544
                 +0.5200000000          +82523.1214674791
                 +0.5300000000          +80552.1812067841
                 +0.5400000000          +78490.6317286889
                 +0.5500000000          +76366.5400842072
                 +0.5600000000          +74202.9648699068
                 +0.5700000000          +72018.8274757875
                 +0.5800000000          +69829.6238311471
 [...]
```

Figure 13.4: Interaction type 2 plot.



One of the plots produced by **fitview** is the evaluation of the fit, that can help you to adjust the geometry weights, Figure 13.5.

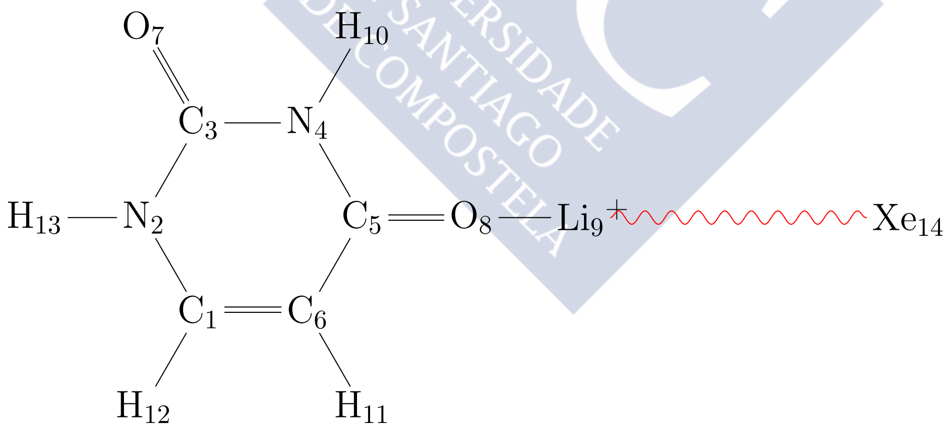
Figure 13.5: General evaluation plot.



14 User designed analytical expressions

The only way for errors to occur in a program is by being put there by the author. No other mechanisms are known. Programs can't acquire bugs by sitting around with other buggy programs.

Harlan D. Mills



Instead of using a potential function already implemented in **GAFit**, the user can type manually a new *analytical expression* directly in the *job.txt* file. We shall use the previous example, **Xe** + **[Li(Uracil)]**$^+$ system, taken from Section 18.

In this example, we fit an analytical expression to the *interaction energies* between Xe and the [Li(Uracil)]$^+$complex, computed by *ab initio* calculations. Next, it is shown how to use this feature using the previous example.

14.1 Preparing input files

The files for this example are in the folder *analytical-example*. The input files are the same than the previous one, except for the *job.txt* file –File 14.1– which is the unique file to modify.

File 14.1: Uracil example with an analytical expression

```
[parameters]
population:        100
crossover rate:    0.75
blx_alpha:         0.5
mutation rate:     0.1
elitism:           yes
tournament size:   5
crossover:         sbx
mutation:          sigma
sigma:             0.1
direction:         min

[job]
runs:              1
evaluations:       500
Geometries:     coord
Energies:    energies
Atom2type:  atom2typ
Bounds:  bounds.txt
Charges:  charges.txt
Potential:  0
All  coefficients:  no
fitting:  relative

[print]
geometries:  no
runs:  no

[analytical]
expression:  potential 3
distance:  dist
potential:  pot
coefficients:  a, b, c1, c2, d1, d

[potential 1]
V=A*EXP(-B*R)+C/R**D;

[potential 2]
v=a*exp(-b*r)+c/r**d+e/r**f;

[potential 3]
v1=a*exp(-b*dist);
v2=c1/dist**c2;
v3=d1/dist**d2;
v4=e1/dist**e2;
pot=v1+v2+v3+v4
```

Analytical expression as a potential

Analytical section

Analytical expressions for internal potentials 1, 2 and 3 from Table 6.2

Potential type, according to Table 6.2, must be changed to **0**. Next we have to write a new section, **[analytical]**, with some configuration data:

expression : This is the expression employed for the potential. In this example it is configured as *potential 3*.

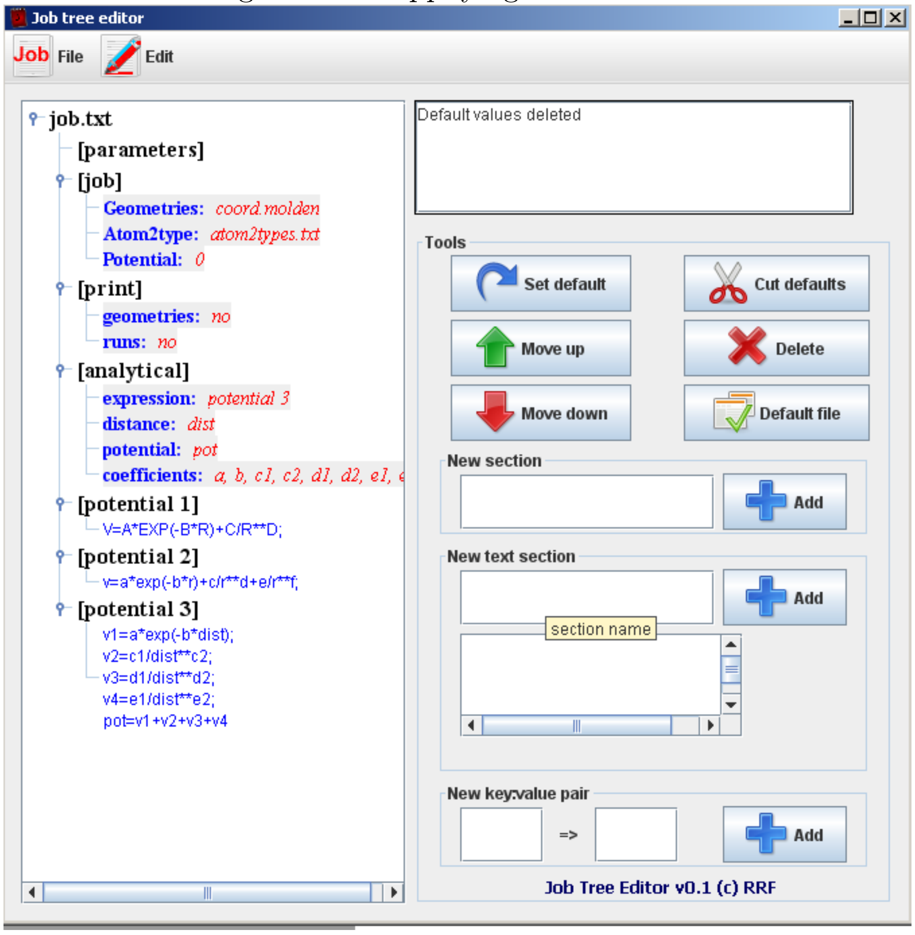
distance : Name of the variable distance –r in the formula from Table 6.2–. *dist* in the example.

potential : Name of the variable potential energy. In the example *pot*.

coefficients : The names of the coefficients to be optimized. In the example *a, b, c1, c2, d1, d2, e1 and e2.*

This can be done with the **JobTreeEditor**–12.5– too: openning the *job.txt* from *analytical-example* and doing Edit 》Cut defaults; Figure 14.1 shows the changes needed from default configuration. You can create a new *job.txt* file by Edit 》Tree 》Iternal job 》Analytical also.

Figure 14.1: Appliying 'cut defaults'.



Default values deleted

```
job.txt
  [parameters]
  [job]
    Geometries: coord.molden
    Atom2type: atom2types.txt
    Potential: 0
  [print]
    geometries: no
    runs: no
  [analytical]
    expression: potential 3
    distance: dist
    potential: pot
    coefficients: a, b, c1, c2, d1, d2, e1, e
  [potential 1]
    V=A*EXP(-B*R)+C/R**D;
  [potential 2]
    v=a*exp(-b*r)+c/r**d+e/r**f;
  [potential 3]
    v1=a*exp(-b*dist);
    v2=c1/dist**c2;
    v3=d1/dist**d2;
    v4=e1/dist**e2;
    pot=v1+v2+v3+v4
```

We have to change or write the key/value pairs:

- section [job]
  - **geometries** to *coord.molden*
  - **atom2type** to *atom2types.txt* –note the final 's'–
  - **potential** to *0*

- section [analytical]
  - **expression** to *potential 3*. It could be any text, but it must be equal letter by letter, including spaces, to the section containing the desired potential.
  - **distance** to the variable used as the distance between atoms, in this case *dist*.
  - **potential** to the variable used as the potential.
  - **coefficients** to the list of coefficients that **GAFit** must deal with.
- section [potential 3] write the analytical expression using the variables *potential*, *dist*, *a*, *b*, *c1*, *c2*, *d1*, *d2*, *e1* and *e2*, like:

  ```
  v1=a*exp(-b*dist);
  v2=c1/dist**c2;
  v3=d1/dist**d2;
  v4=e1/dist**e2;
  pot=v1+v2+v3+v4
  ```

  Do not worry about the intermediate variables *v1*, *v2*, *v3* and *v4*; they will be created as needed by the compiler subroutines. See 6.6.

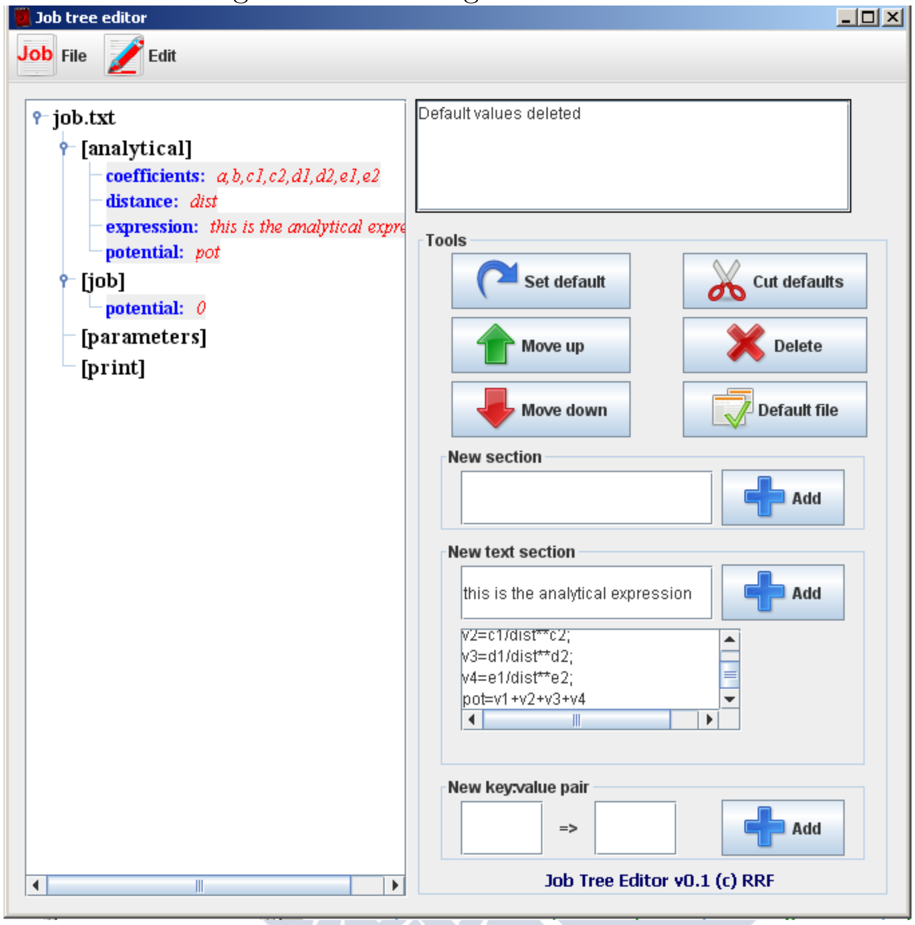
If you want to use other potential like **potential 1** or **potential 2** you must change the whole **[analytical]** section accordingly.

You can use the **JobTreeEditor** tool too: Edit ›Tree ›Internal job ›Analytical to create a default analytical job. Edit ›Cut defaults to clean configuration cutting off keys with default values. Edit the **coefficients** key, writing *a,b,c1,c2,d1,d2,e1,e2*.

Edit the **distance** key writing *dist* and the **potential** key writing *pot*.

Using the **new text section** in the right panel, you can type the analytical expression. In this case the section containing the expression is named **this is the analytical expression**. See Figures 14.2 and 14.3.

Figure 14.2: Adding new text section.



job.txt
[analytical]
coefficients: a,b,c1,c2,d1,d2,e1,e2
distance: dist
expression: this is the analytical expre
potential: pot
[job]
potential: 0
[parameters]
[print]

Default values deleted

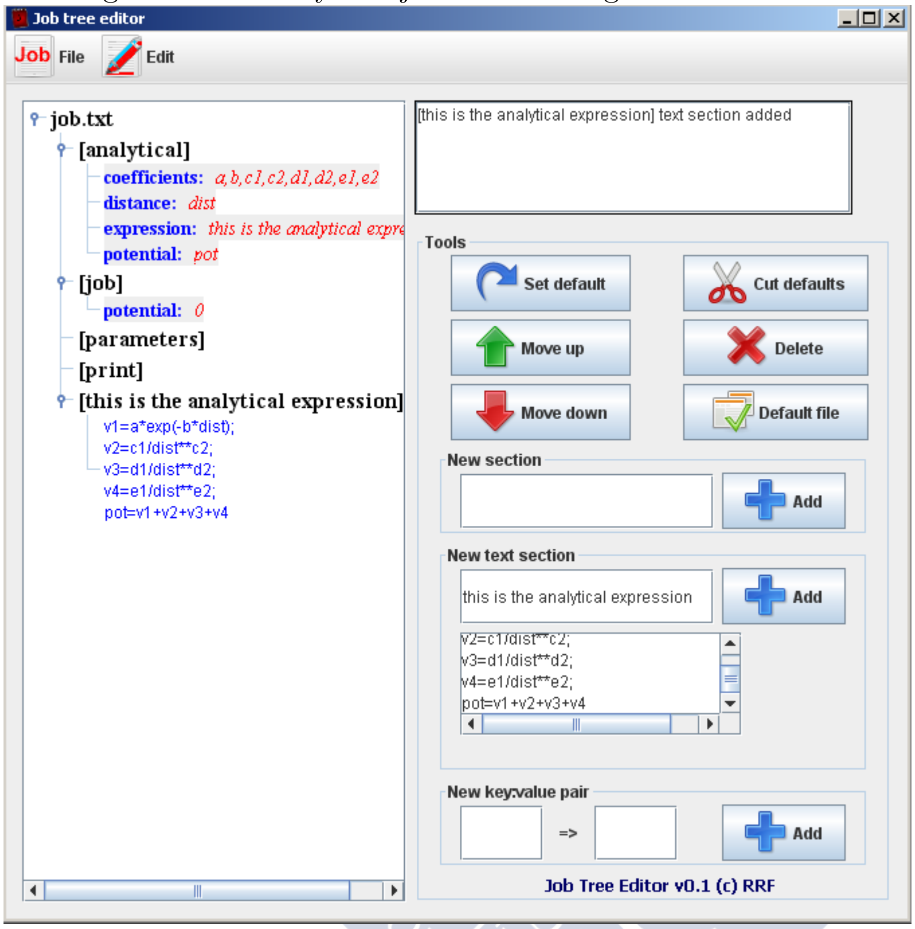
this is the analytical expression

```
v2=c1/dist**c2;
v3=d1/dist**d2;
v4=e1/dist**e2;
pot=v1+v2+v3+v4
```

You can test the *job.txt* file using **ufpu** –section 12.4– and type some values to **distance** and **coefficients** and check the calculated **potential**.

```
$ ufpu
uFpu v0.2 (c) Roberto Rodriguez-Fernandez

expression name: "this is the analytical expression"
potential:       pot
distance:        dist
coefficients:    a,b,c1,c2,d1,d2,e1,e2

Expression found:

        v1 = a*exp(-b*dist);
        v2 = c1/dist**c2;
        v3 = d1/dist**d2;
        v4 = e1/dist**e2;
        pot = v1+v2+v3+v4

        Variables found in expression: v1 a b dist v2 c1 c2 v3 d1 d2 v4 e1 e2 pot
```

Figure 14.3: Analytical job after adding the text section.



```
job.txt
  [analytical]
    coefficients: a,b,c1,c2,d1,d2,e1,e2
    distance: dist
    expression: this is the analytical expre
    potential: pot
  [job]
    potential: 0
  [parameters]
  [print]
  [this is the analytical expression]
    v1=a*exp(-b*dist);
    v2=c1/dist**c2;
    v3=d1/dist**d2;
    v4=e1/dist**e2;
    pot=v1+v2+v3+v4
```

[this is the analytical expression] text section added

New text section: this is the analytical expression

```
v2=c1/dist**c2;
v3=d1/dist**d2;
v4=e1/dist**e2;
pot=v1+v2+v3+v4
```

```
        Expression code OK
        pot index 13
        dist index 3
        8 coefficients found
INPUT
        distance variable (dist)=1
        coefficient a=1
        coefficient b=1
        coefficient c1=1
        coefficient c2=1
        coefficient d1=1
        coefficient d2=1
        coefficient e1=1
        coefficient e2=1

After run:      Memory (total used 27)  v1=0.367879 a=1.000000 b=1.000000
dist=1.000000 v2=1.000000 c1=1.000000 c2=1.000000 v3=1.000000 d1=1.000000
d2=1.000000 v4=1.000000 e1=1.000000 e2=1.000000 pot=3.367879

RESULT POTENTIAL:3.367879

Press 'q'/INTRO to quit, another key/INTRO to repeat
```

The bytecode result of compiling the analytical expression is shown in File 14.2.

The resulting *job.txt* is shown in File 14.3 after adjusting the **geometries** and **atom2type** files. Also a *bounds.txt* file, with 8 bounds like the one included in the example, must be used.

File 14.2: Asembler bytecode produced

Where each variable is on the memory pool. Figure 11.3

Program to calculate the expression

```
 ; v1:0
 ; a:1
 ; b:2
 ; dist:3
 ; v2:4
 ; c1:5
 ; c2:6
 ; v3:7
 ; d1:8
 ; d2:9
 ; v4:10
 ; e1:11
 ; e2:12
 ; pot:13
apush 0
apush 1
apush 2
neg
apush 3
mult
exp
mult
store
apush 4
apush 5
apush 3
apush 6
pow
div
store
apush 7
apush 8
apush 3
apush 9
pow
div
store
apush 10
apush 11
apush 3
apush 12
pow
div
store
apush 13
apush 0
apush 4
add
apush 7
add
apush 10
add
store
```

File 14.3: Analytical expression job

```
[job]
geometries: coord.molden
atom2type: atom2types.txt
potential: 0

[analytical]
coefficients: a,b,c1,c2,d1,d2,e1,e2
distance: dist
expression: this is the analytical expression
potential: pot

[parameters]

[print]

[this is the analytical expression]
v1=a*exp(-b*dist);
v2=c1/dist**c2;
v3=d1/dist**d2;
v4=e1/dist**e2;
pot=v1+v2+v3+v4
```

Analytical expression named section

Analytical expression

File 14.4: Analytical expression job output

```
*************
  gafit 0.6.3
  Build: 2569
*************

Job type: internal

Settings for job
------------------------------
Coordinates:[coord.molden]
Energies:[energies.txt]
Atom2type:[atom2types.txt]
Bounds:[bounds.txt]
Charges:[charges.txt]
Potential read: Analytical expression
All coefficients: no, Read and repeat subset
Fitting: relative
Auto weights: no

Print options:
        geometries yes
        runs yes
        ga settings no
        analytical yes
        auto weights no

Analytical expression
------------------------------
expression name: "this is the analytical expression"
potential:       pot
distance:        dist
coefficients:    a,b,c1,c2,d1,d2,e1,e2

Expression found:

        v1 = a*exp(-b*dist);
        v2 = c1/dist**c2;
        v3 = d1/dist**d2;
        v4 = e1/dist**e2;
        pot = v1+v2+v3+v4@

        Variables found in expression: v1 a b dist v2 c1 c2 v3 d1 d2 v4 e1 e2 pot
        Expression code OK
        pot index 13
        dist index 3
        8 coefficients found
...now reading data

[...]
```

14.2 Running and examining results

The output is similar to the previous one –section 13–, except for the potential. Here we use the number 3 from Table 6.2 but coded as an *analytical expression*.

15 External Interface example

The nice thing about standards is that you have so many to choose from.

Andrew S. Tanenbaum

Before examining the MOPAC interface, we are going to study a simple case: fitting a polynomial to a set of values.

15.1 Input files

Whe have some $(x,\ f(x))$ pair values shown in Table 15.1 to fit to a polynomial of fifth degree. These value pairs are in the input File 15.1.

File 15.1: *external.values* file

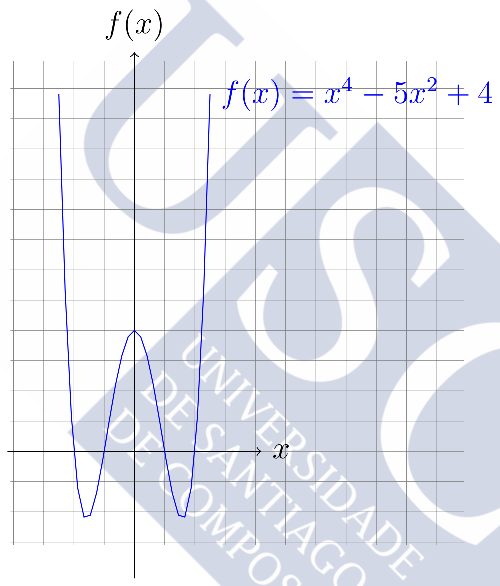
```
−3 40
−2 0
−1 0
0 4
1 0
2 0
3 40
```

Obviously, the data fits to any polynomial who has roots at -2, -1, 1 and 2 like the one shown in Figure 15.1. Also, we need a *bounds.txt* file to fix upper and lower limits as the included example in File 15.2. In this case, we want integer values, so the righmost column is set to 1.

Table 15.1: Example values to fit.

x	$f(x)$
-3	40
-2	0
-1	0
0	4
1	0
2	0
3	40

Figure 15.1: Example polynomial plot



File 15.2: *bounds.txt* file

```
  TEXT TEXT TEXT TEXT
           -10.       10.        1
           -10.       10.0       1
           -10.       10.        1
           -10.       10.0       1
           -10.       10.        1
```

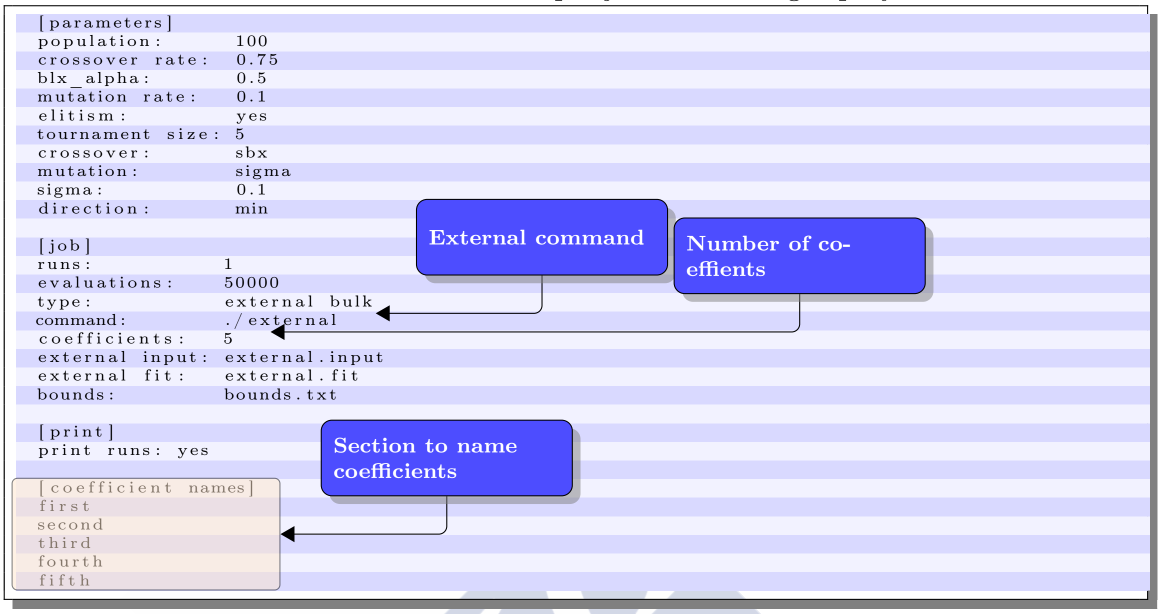
An external tool example is provided in File 15.4. This code inputs the coefficients values provided by **GAFit** and the external known values –Table 15.1, File 15.1– to calculate a fit to a generic polynomial of degree n:

$$a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$$

The given test code supports both *external* and *external bulk* options (section 9.1): It can read to evaluate a set of coefficients –a individual–

or a whole population set of coefficients. To test each one change in File 15.3 –the *job.txt* file– the type of job.

File 15.3: External example job.txt: fitting a polynomial



```
[parameters]
population:          100
crossover rate:      0.75
blx_alpha:           0.5
mutation rate:       0.1
elitism:             yes
tournament size:     5
crossover:           sbx
mutation:            sigma
sigma:               0.1
direction:           min

[job]
runs:              1
evaluations:       50000
type:              external bulk
command:           ./external
coefficients:      5
external input:    external.input
external fit:      external.fit
bounds:            bounds.txt

[print]
print runs: yes

[coefficient names]
first
second
third
fourth
fifth
```

In the configuration file –*job.txt*, File 15.3– is included a **[coefficients names]** section to name each coefficient with a user provided string. So, a_0 becomes $first$, a_1 becomes *second* and so on –note how is defined **double func(double x, double a[], int n)** at lines 13-14, File 15.4–.

File 15.4: external.c

```
1  /*
2  (c) baaden@gmail.com $Id: external.c 3323 2013-11-15 09:33:52Z ro
     $
3  */
4  #if HAVE_CONFIG_H
5  #include <config.h>
6  #endif
7  #include <stdio.h>
8  #include <math.h>
9  #include <stdlib.h>
10
11 #define MAXLINE 100
12
13 double
14 func (double x, double a[], int n)
15 {
16   double ret = 0;
17   int i;
18   for (i = 0; i < n; i++)
19     ret += a[i] * pow (x, (double) i);
20   return ret;
```

```c
}

int
main (void)
{
  char line[MAXLINE + 1];
  double *coef = NULL;
  double *valuesx = NULL, *valuesy = NULL;
  double fit, number0, number1, tmp, div;
  int i, j, ncoefs, mvalues, tcoefs;
  int first, ok;

  FILE *f, *out;


  mvalues = 0;
  f = fopen ("external.values", "r");
  while (fgets (line, MAXLINE, f) != NULL)
    {
      sscanf (line, "%lf%lf", &number0, &number1);
      valuesx = (double *) realloc (valuesx, sizeof (double) * (
        mvalues + 1));
      valuesy = (double *) realloc (valuesy, sizeof (double) * (
        mvalues + 1));
      valuesx[mvalues] = number0;
      valuesy[mvalues] = number1;
      mvalues++;
    }
  fclose (f);

  ok = 1;
  first = 1;
  ncoefs = 0;
  out = fopen ("external.fit", "w");
  f = fopen ("external.input", "r");
  if(!f)
    {
    printf("no file external.input\n");
    exit(EXIT_FAILURE);
    }
  while (ok)
    {
      while (fgets (line, MAXLINE, f) != NULL)
        {
          char *p = line;
          while (*p == ' ' || *p == '\t')
            p++;
          if (*p == '\r' || *p == '\n')
            break;

          sscanf (line, "%lf", &number0);
          ncoefs++;

          if (first)
            {
              coef = (double *) realloc (coef, sizeof (double) * (
                ncoefs));
              tcoefs=ncoefs;
            }
          coef[ncoefs - 1] = number0;
        }
      if(feof(f))
        ok=0;
      first = 0;
      ncoefs = 0;
      fit = 0;
      for (i = 0; i < mvalues; i++)
        {
          tmp = func (valuesx[i], coef, tcoefs);
          //check div by zero
          if (valuesy[i] == 0)
            div = 1;             //use absolute
          else
            div = valuesy[i] * valuesy[i];      //use relative
          fit += (tmp - valuesy[i]) * (tmp - valuesy[i]) / div;
        }

      fprintf (out, "%lf\n", fit);
    }
  fclose (out);
  fclose (f);
}
```

15.2 Running the example and examining results

To create the needed files and run the test you only have to type in the **GAFit**'s distribution folder:

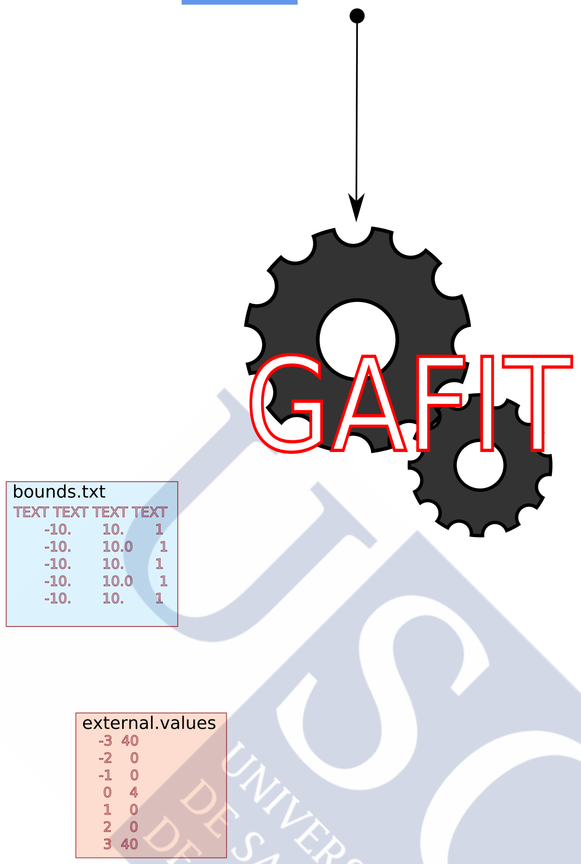
```
$ cd external-example
$ make test
```

Some things happen, e.g. compiling *external.c* source code to produce **external** binary, and the example begins to run. What is on way?

Step 1 **GAFit** is launched. It finds two input files: *bounds.txt* and *external.values*.

Step 2 **GAFit** writes a whole population of coefficients to be evaluated in the *external.input* file –File 15.5– using as upper and lower bounds

Figure 15.2: Step 1: **GAFit** is launched



```
TEXT TEXT TEXT TEXT
      -10.     10.     1
      -10.     10.0     1
      -10.     10.     1
      -10.     10.0     1
      -10.     10.     1
```

```
 -3  40
 -2   0
 -1   0
  0   4
  1   0
  2   0
  3  40
```

those specified in the file *bounds.txt* –File 15.2–. If the file exists, it is overwritten.

If we want only one coefficients set at a time, the **type** of job must be changed from **external bulk** to **external** in File 15.3.

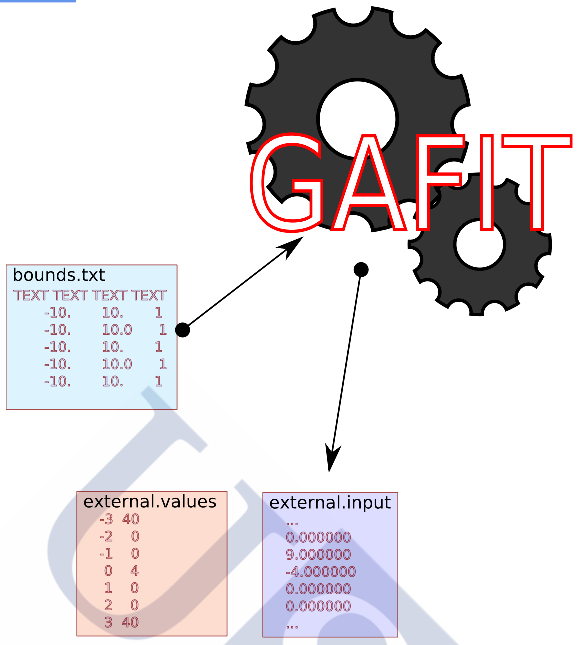
The coefficients must be integers –*bounds.txt* last column set to 1–.

Step 3 **GAFit** launches the **external** binary.

Step 4 **external** using *external.input* evaluates the *external.values* and overwrites if the file exists, or it creates the *external.fit* file –File 15.6–.

Step 5 **GAFit** reads the *external.fit* file with the results. If minimizing, the lesser best, so a 0, or near it, means a very good fit. I the file shown, File 15.6, the 13th value is worse than 6th.

Figure 15.3: Step 2 : **GAFit** overwrites or creates the *external.input* file.



```
bounds.txt
TEXT TEXT TEXT TEXT
     -10.     10.      1
     -10.     10.0     1
     -10.     10.      1
     -10.     10.0     1
     -10.     10.      1
```

```
external.values
  -3  40
  -2   0
  -1   0
   0   4
   1   0
   2   0
   3  40
```

```
external.input
  ...
  0.000000
  9.000000
  -4.000000
  0.000000
  0.000000
  ...
```

The n^{th} value, (0, 9, -4, 0, 0), from File 15.5 represents the polynomial:

$$p(x) = 0x^4 + 0x^3 - 4x^2 + 9x + 0$$

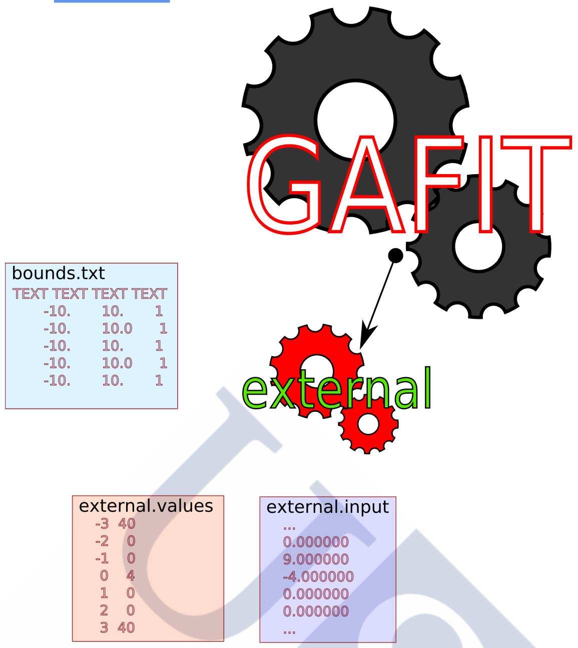
Table 15.2: n^{th} set of coefficients fit.

x	$f(x)$	$p(x) = -4x^2 + 9x$	$\frac{[p(x)-f(x)]^2}{f(x)^2}$
-3	40	-63	6.630625
-2	0	-34	1156.000000
-1	0	-13	169.000000
0	4	0	1.00000
1	0	5	25.00000
2	0	2	4.00000
3	40	-9	1.500625
		$\sum \frac{[p(x)-f(x)]^2}{f(x)^2}$	1363.131250

The calculations are shown in Table 15.2 for the n^{th} coefficients set: Files 15.5 and 15.6.

Note that, in the *external.c* program, File 15.4, lines 88-92, we do a trick to avoid dividing by zero: we use a relative fit, but if divisor equals zero, we use 1 for the divisor which in the other hand it is converted in an absolute fit.

Figure 15.4: Step 3 : **GAFit** launches the **external** binary



Step 6 if **GAFit** finds it in the *external.fit* file, the best fit is overwritten if exists, or creates the *best.txt* file –File 15.7. Note that this file always will be overwritten: If you have some fit to save, copy it out there or rename it.

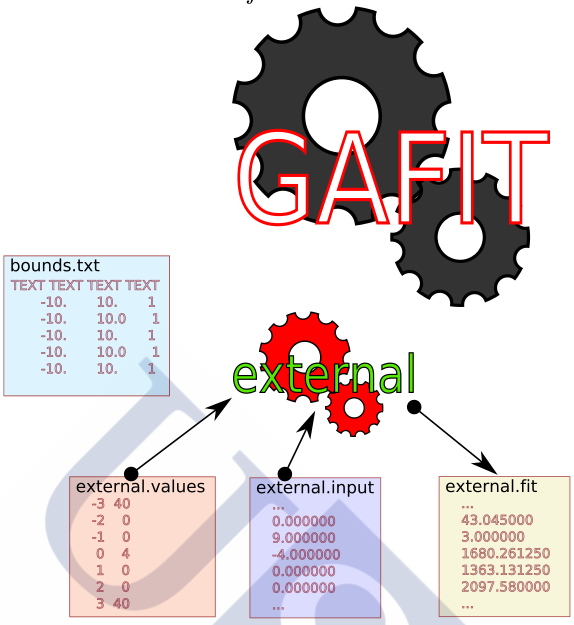
The values shown represent the polynomial:

$$f(x) = x^4 - 5x^2 + 4$$

File 15.5: external.input file

```
[...]

0.000000
0.000000
-7.000000
0.000000
0.000000        coefficients set n
0.000000
9.000000
-4.000000
0.000000        coefficients set n + 1
0.000000
0.000000
0.000000
0.000000
-4.000000
0.000000

[...]
```

Figure 15.5: Step 5 : **external** using *external.input* evaluates the *external.values* and overwrites or creates the *external.fit* file



```
bounds.txt
TEXT TEXT TEXT TEXT
  -10.    10.     1
  -10.    10.0    1
  -10.    10.     1
  -10.    10.0    1
  -10.    10.     1
```

```
external.values
 -3  40
 -2   0
 -1   0
  0   4
  1   0
  2   0
  3  40
```

```
external.input
 ...
 0.000000
 9.000000
 -4.000000
 0.000000
 0.000000
 ...
```

```
external.fit
 ...
 43.045000
 3.000000
 1680.261250
 1363.131250
 2097.580000
 ...
```

File 15.6: external.fit file

evaluation of n^{th} coefficients set

```
[...]
1680.261250
1363.131250
2097.580000
[...]
```

File 15.7: best.txt

best till now coefficients set

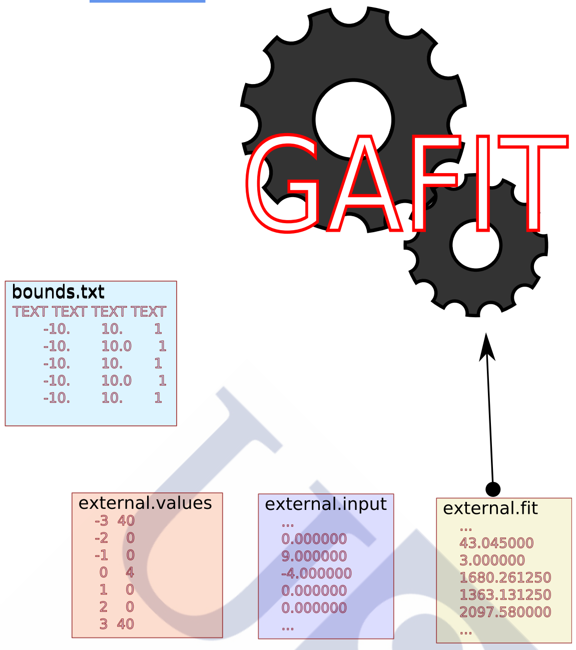
fit value of the best coefficients set

```
4.000000000000
0.000000000000
−5.000000000000
0.000000000000
1.000000000000

Fitness:  0.000000000000
```

The output of the whole process is sumarized in Files 15.8 and 15.9.

Configuring **GAFit** to work with an external program is a complex task. You can begin with this example changing the code and the configuration until it covers all your needs. A good tip is to use the **test** option in the **[job]** section of the *job.txt* file to set always the same *seed* and compare between changes –See 6.3–.

Figure 15.6: Step 5 : **GAFit** reads the *external.fit* file



File 15.8: external.output

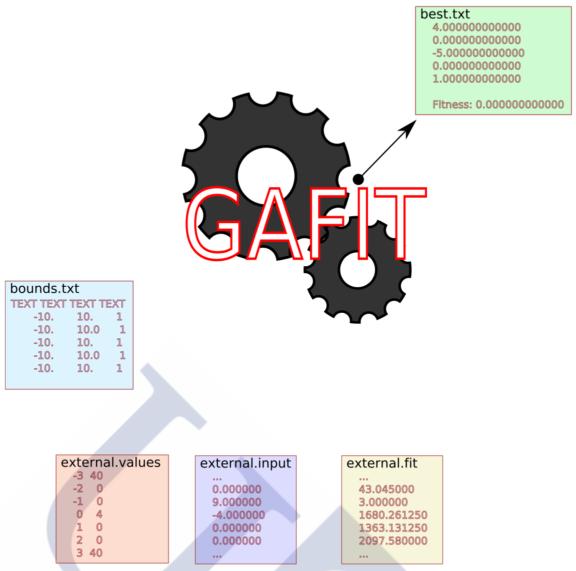
```
 ../src/gafit

 *************
   gafit  0.6.3
   Build:  2569
 *************

 Job  type:  external  bulk
 Command  :  ./external

 Settings  for  job
 --------------------------------
 Command:[./external]
 Bounds:[bounds.txt]
 External  input:[external.input]
 External  fit:[external.fit]
 Coefficients:  5

 Print  options:
          runs  yes
          ga  settings  no
 ...now  reading  data

 Reading  bounds  for  5  coefficients

                1       first       -10.00000  -       +10.00000  (integer)
                2       second      -10.00000  -       +10.00000  (integer)
                3       third       -10.00000  -       +10.00000  (integer)
                4       fourth      -10.00000  -       +10.00000  (integer)
                5       fifth       -10.00000  -       +10.00000  (integer)

 [...]
```

job type and external command to run

settings for job: command and number of coefficients provided. The rest are defaults

coefficients name, limits and type

Figure 15.7: Step 6 : if the fit is the best till now, **GAFit** overwrites or creates the *best.txt* file



File 15.9: external.output (cont.)

```
 [...]
  5 BOUNDS VECTOR
 ==========================================================================
                     1       first                               10.00000  (integ
                     2      second                               10.00000  (integ
                     3       third                               10.00000  (integ
                     4      fourth                               10.00000  (integ
                     5       fifth            −10.00000  −      +10.00000  (integ

#seed#1384731417#seed#

run 1
100      1      28162.612712500002      5.707075000000e+02
300      3      4235.713475000000       5.896375000000e+01
500      5      2559.175325000000       4.068250000000e+01
700      7      2715.567500000000       3.189250000000e+01
900      9      3434.814662500000       1.805500000000e+01

 [...]

49500    495    2798.209387500000       0.000000000000e+00
49800    498    4389.045750000000       0.000000000000e+00

#
#Results
#
       1      first    +4.000000000000
       2     second    +0.000000000000
       3      third    −5.000000000000
       4     fourth    +0.000000000000
       5      fifth    +1.000000000000
```

seed for this run

individuals calculated till now, number of generations, best evaluation and best fit

Last best written to *best.txt*

You can use also the **JobTreeEditor** –See 12.5– application to build a default *job.txt*, issuing an Edit 〉Tree 〉External job 〉External, modifiying the key/value pairs and at last File 〉Save as.

More information on this subject on 9.1. To test further this example, we can do some modiffcations:

- change the number of coefficients to 6
- add a new name to **[coefficients names]** section
- add a new line to the *bounds.txt* file.

and run it some times.

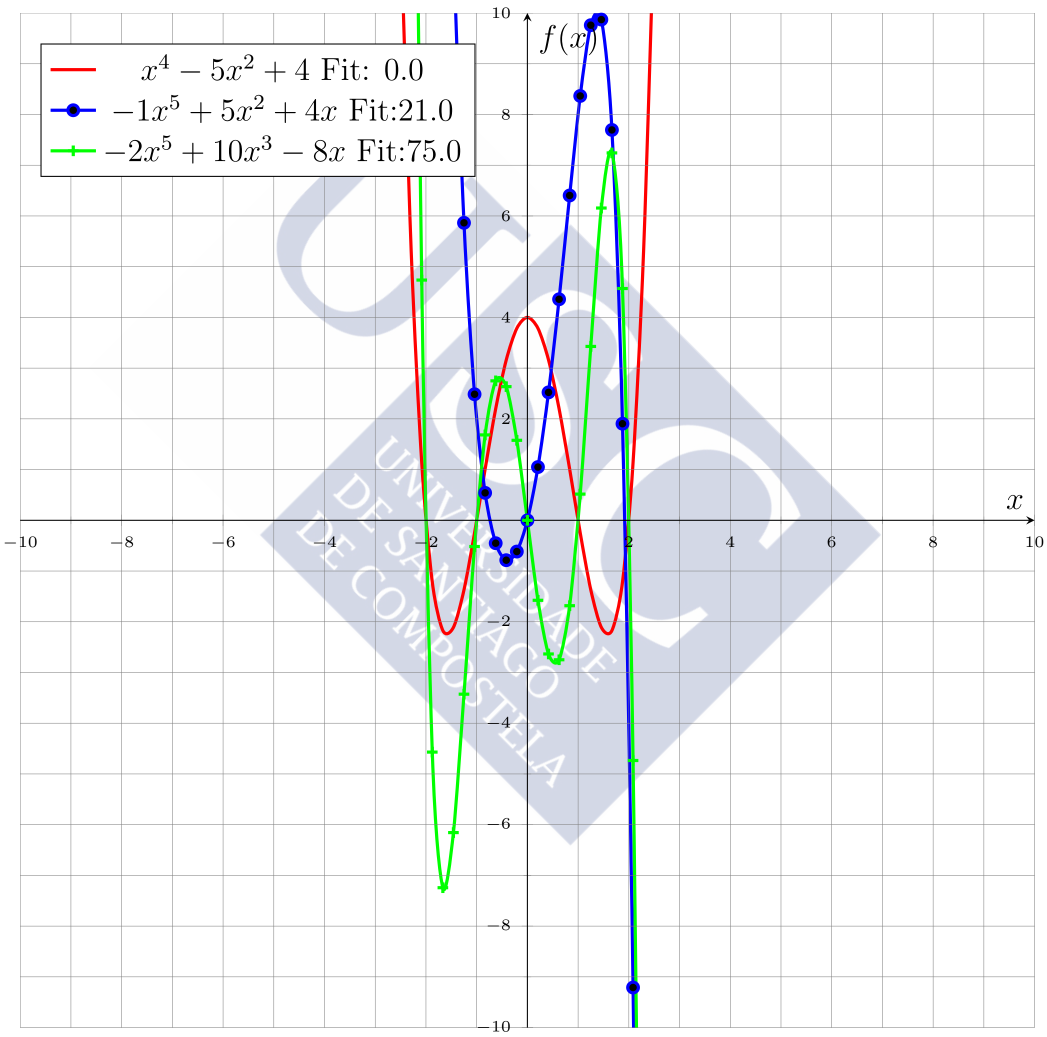
There are distinct results from each run, because the GA explores all the space limited by the bounds and by the type of coefficients: only integers. Some results are shown in Table 15.3 and plotted in Figure 15.8.

Table 15.3: Some results running the example with 6 coefficients.

a_0	a_1	a_2	a_3	a_4	a_5	fit[a]
0	0	0	0	0	0	3.0
4	0	-5	0	1	0	0.0
0	4	5	0	0	-1	21.0
0	-8	0	10	0	-2	75.0

[a] The lesser best.

Figure 15.8: Table 15.3 polynomial plots.



16 MOPAC Interface example

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones.

Linus Torvalds. 1991

In this Section, a semiempirical Hamiltonian is reparametrized to fit the energetics and also geometries and frequencies for a decomposition channel of vinyl cyanide (VC). The *ab initio* calculations for this system are shown below –see Section 19–.

16.1 Prerequisites

You must have installed MOPAC 2009 in your system. You must know where it is installed or which is the value of the MOPAC_LICENSE shell variable.

16.2 Input and executable files

The complete interface was explained in the Section 9. To create and run the example you must type:

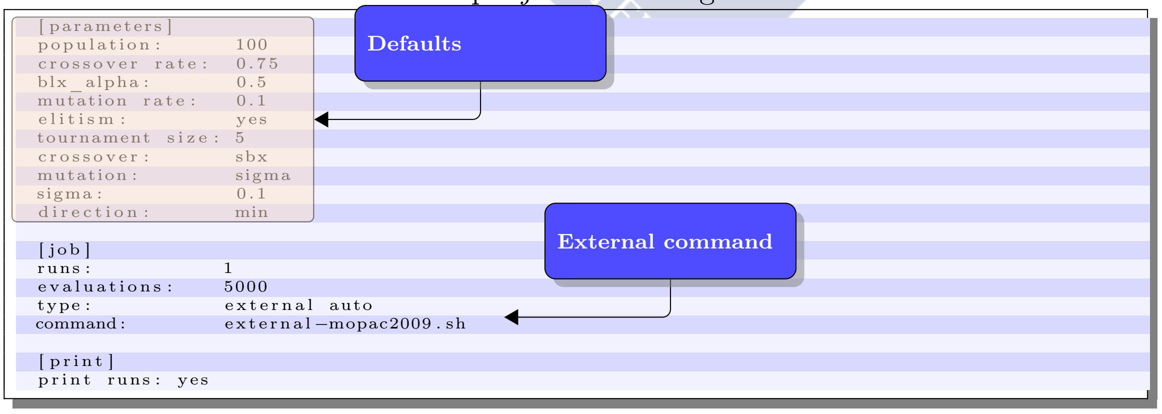
```
$ cd mopac-example
$ tar xvzf mopac.tgz
$ make test
```

Some files are extracted from the compressed example and the executables are copied from the **src** folder where they were compiled if needed. This is a small piece from the Section 19 application.

Table 16.1: Files in the mopac-example folder after run *make test*.

File	Type	Provided by
bounds.txt	text file	example
conditions.txt	text file	example
external-mopac2009.sh	shell script	example
job.txt	job configuration	example
template.coefs	mopac2009 external coefficients	example
template.mop	mopac2009 job template	example
extractor	Perl script	**GAFit**
fitter	binary	**GAFit**
injector	binary	**GAFit**

File 16.1: External example *job.txt*: fitting MOPAC 2009 coefficients



```
[parameters]
population:          100
crossover rate:      0.75
blx_alpha:           0.5
mutation rate:       0.1
elitism:             yes
tournament size:     5
crossover:           sbx
mutation:            sigma
sigma:               0.1
direction:           min

[job]
runs:                1
evaluations:         5000
type:                external auto
command:             external-mopac2009.sh

[print]
print runs: yes
```

As shown in File 16.1, the job is declared as **external auto**, so the external scripts and/or binaries must configure the system by themselves.

File 16.2: MOPAC 2009 coefficient limits: *bounds.txt* file

```
TEXT TEXT TEXT TEXT
        -10.        10.       0
        -10.        10.0      0
        -10.        10.       0
        -20.        20.0      0
       -100.       100.       0
       -100.       100.       0
       -100.       100.0      0
        -10.        10.       0
        -10.        10.0      0
        -10.        10.       0
        -10.        10.       0
        -20.        20.0      0
        -20.        20.       0
        -20.        20.0      0
        -20.        20.       0
        -10.        10.       0
```

The objective is to obtain a suitable combination of coefficients, File 16.3, to satisfy the constrains declared in File 16.5 using the MOPAC 2009 task shown in File 16.4 where the @ symbol will be replaced by the name of a copy of File 16.3 where the coefficients are generated by **GAFit** between some limits expressed in the File 16.2.

Note that these randomly generated coefficients are prone to err and crash MOPAC.

You can edit the *job.txt* using the **JobTreeEditor** –Section 12.5–. The *bounds.txt* can be edited using the **bedit** tool in the example folder. A *Makefile* rule is provided to do it:

```
$ make bedit
```

```
Bedit v0.2 Interactive editing
        (c) Roberto Rodriguez Fernandez 2010

autoconfiguring...

Job type: external
Command : external-mopac2009.sh
====
actual interaction: [1]
====
b Interactive modify bounds
z Quit
====
Option:b
            1(A)    BETAS H      -10.00000 -       +10.00000 (real)
            2(B)       ZS H      -10.00000 -       +10.00000 (real)
            3(C)      ALP H      -10.00000 -       +10.00000 (real)
            4(D)      GSS H      -20.00000 -       +20.00000 (real)
            5(E)      USS C     -100.00000 -      +100.00000 (real)
            6(F)      UPP C     -100.00000 -      +100.00000 (real)
            7(G)    BETAS C     -100.00000 -      +100.00000 (real)
            8(H)    BETAP C      -10.00000 -       +10.00000 (real)
            9(I)       ZS C      -10.00000 -       +10.00000 (real)
           10(J)       ZP C      -10.00000 -       +10.00000 (real)
           11(K)      ALP C      -10.00000 -       +10.00000 (real)
           12(L)      GSS C      -20.00000 -       +20.00000 (real)
           13(M)      GSP C      -20.00000 -       +20.00000 (real)
           14(N)      GPP C      -20.00000 -       +20.00000 (real)
           15(O)      GP2 C       -20.00000 -       +20.00000 (real)
           16(P)      HSP C       -10.00000 -       +10.00000 (real)
                       [l]ower            [u]pper

use 'a:l=1.03' to change A:lower to 1.03
    'c:u=1000' to change C:upper to 1000
    'b:l=100 b:u=1000' to change B:lower and B:upper
    'a:r c:i' to change A to real, C to integer
    'w' to save to file (writing labels)
    'z' to quit
Input:
```

As an external job, **bedit** only can edit the bounds file.

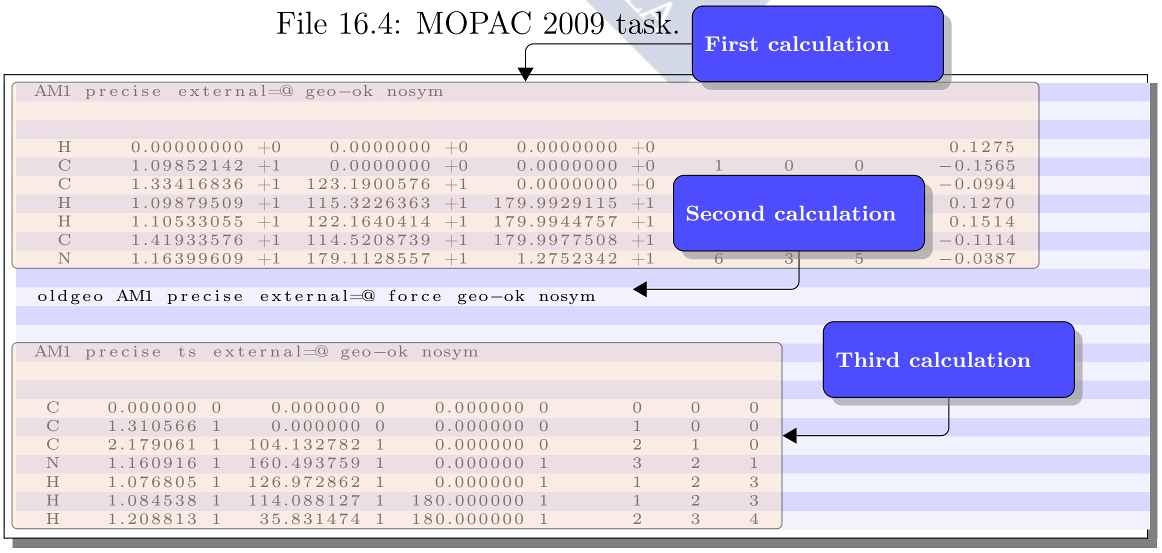
File 16.3: MOPAC 2009 coefficients to fit. *template.coefs* file

```
BETAS H         -6.173787
ZS H         1.188078
ALP H          2.882324
GSS   H         12.848
USS   C         -52.028658
UPP   C         -39.614239
BETAS   C       -15.715783
BETAP   C       -7.719283
ZS    C         1.808665
ZP    C         1.685116
ALP   C         2.648274
GSS   C         12.23
GSP   C         11.47
GPP   C         11.08
GP2   C         9.84
HSP   C         2.43
```

Here, File 16.3 only a small set of coefficients to fit. The whole coefficients list and their default values can be obtained from the MOPAC source.

The interface utilities count the number of coefficients to fit and configure **GAFit** accordingly as shown in Figure 9.2 and explained in section 9.4.1. Here, the File 16.7 is used to pass to **GAFit** the configuration.

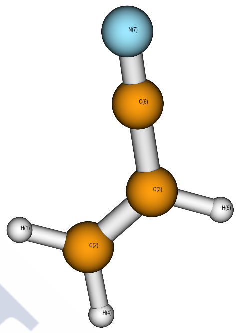
File 16.4: MOPAC 2009 task.



```
AM1 precise external=@ geo-ok nosym


  H    0.00000000 +0    0.0000000 +0    0.0000000 +0                    0.1275
  C    1.09852142 +1    0.0000000 +0    0.0000000 +0    1     0     0  -0.1565
  C    1.33416836 +1  123.1900576 +1    0.0000000 +0                   -0.0994
  H    1.09879509 +1  115.3226363 +1  179.9929115 +1                    0.1270
  H    1.10533055 +1  122.1640414 +1  179.9944757 +1                    0.1514
  C    1.41933576 +1  114.5208739 +1  179.9977508 +1                   -0.1114
  N    1.16399609 +1  179.1128557 +1    1.2752342 +1    6     3     5  -0.0387

oldgeo AM1 precise external=@ force geo-ok nosym

AM1 precise ts external=@ geo-ok nosym


  C    0.000000 0    0.000000 0    0.000000 0      0    0    0
  C    1.310566 1    0.000000 0    0.000000 0      1    0    0
  C    2.179061 1  104.132782 1    0.000000 0      2    1    0
  N    1.160916 1  160.493759 1    0.000000 1      3    2    1
  H    1.076805 1  126.972862 1    0.000000 1      1    2    3
  H    1.084538 1  114.088127 1  180.000000 1      1    2    3
  H    1.208813 1   35.831474 1  180.000000 1      2    3    4
```

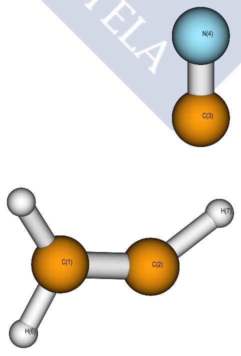
In File 16.4 we have three calculations:

Figure 16.1: Vinyl cyanide drawn using the coordinates of the first calculation (optimization of the minimum energy structure).



- The first one, an AM1 geometry optimization of the vinyl cyanide. Figure 16.1.
- The second one, using the optimized geometry from first one (keyword **oldgeo**), it calculates vibrational frequencies (keyword **force**)
- The third one, a transition state search (keyword **ts**). Figure 16.2.

Figure 16.2: Three-centered transition state drawn using the coordinates of the last calculation.



The number of calculations presents in the task are detected parsing MOPAC output. Some semiempirical parameters are taken at run time by use of **EXTERNAL=@**, where **GAFit** will replace all @ with the name of a file which contains the generated parameters to fit as explained before. For those parameters not in file, MOPAC take its defaults.

File 16.5: Constrains: *conditions.txt* file

```
delt     3   1   100.6      0.1
frequency    2      15     3271.0    1e−4
distance     3       1        7      3.700309096    100.0
penalty  1e10
```

Constrains are explained in Section 9.6. Here, we have:

delt 3 1 100.6 0.1 Difference of heat of formation between calculation 3 (optimized transition state) and calculation 1 (optimized geometry) must be 100.6 kcal/mol and it has a weight of 0.1.

frequency 2 15 3271.0 1e-4 Vibrational frequency number 15, obtained from calculation 2, must be 3271.0 and it has a weight of 0.0001.

distance 3 1 7 3.700309096 100.0 Distance in calculation 3 between atom 1 and atom 7 must be 3.700309096 and having a weight of 100.0.

penalty 1e10 If any of the calculations in the template fails, it be assigned a penalty of 10.000.000.000.

Each set of semiempirical parameters is evaluated taking into account MOPAC output with:

$$\mathbf{fit} = \begin{cases} \textit{if calculation is done:} & \begin{cases} \left[\dfrac{\left(100.6-(\mathbf{HEAT}_{[\mathbf{1st\ calculation}]}-\mathbf{HEAT}_{[\mathbf{3rd\ calculation}]})\right)}{100.6}\right]^2 * 0.1 \\ + \\ \left[\dfrac{\left(3271.0-\mathbf{FREQUENCY}_{[\mathbf{number\ 15\ from\ 2nd\ calculation}]}\right)}{3271.0}\right]^2 * 1e^{-4} \\ + \\ \left[\dfrac{\left(3.700309096-\mathbf{DISTANCE}_{[\mathbf{atoms\ 3\text{-}1\ from\ 3rd\ calculation}]}\right)}{3.700309096}\right]^2 * 100. \end{cases} \\ \textit{if calculation fails:}\ 1e10 \end{cases}$$

GAFit shall run to minimize the fit.

16.3 Running the example and examining results

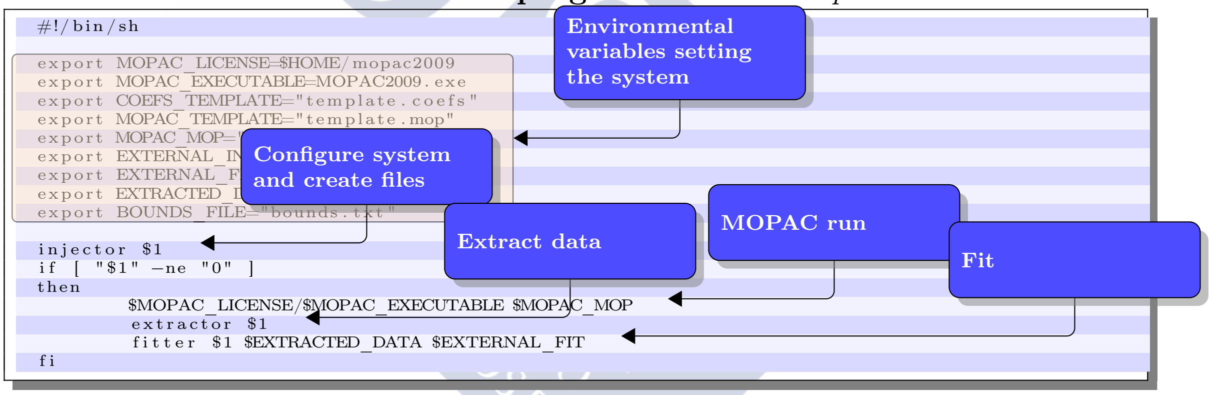
The file *external-mopac.sh* performs all the above operations, as shown in File 16.1.

To run the example, type:

```
$ cd mopac-example
$ make test
```

The external program provided is shown in File 16.6. The operation mode is similar but slightly more complicated than 15. These are the steps:

File 16.6: **external program**: *external-mopac.sh* file



```
#!/bin/sh

export MOPAC_LICENSE=$HOME/mopac2009
export MOPAC_EXECUTABLE=MOPAC2009.exe
export COEFS_TEMPLATE="template.coefs"
export MOPAC_TEMPLATE="template.mop"
export MOPAC_MOP="
export EXTERNAL_IN
export EXTERNAL_F
export EXTRACTED_I
export BOUNDS_FILE="bounds.txt"

injector $1
if [ "$1" -ne "0" ]
then
        $MOPAC_LICENSE/$MOPAC_EXECUTABLE $MOPAC_MOP
        extractor $1
        fitter $1 $EXTRACTED_DATA $EXTERNAL_FIT
fi
```

Environmental variables setting the system
Configure system and create files
Extract data
MOPAC run
Fit

Step 1 **GAFit** runs the external program to configure the system as:

```
external-mopac2009.sh 0
```

A file with the configuration is generated by running **injector 0** in turn. This file is shown in File 16.7. All the options are taken from the environment variables set in File 16.6.

File 16.7: **external auto**: *response* file

```
[job]
type: external
coefficients: 16
external input: mopac2009.input
external fit: mopac2009.fit
bounds: bounds.txt

[coefficient names]
BETAS H
ZS H
ALP H
GSS H
USS C
UPP C
BETAS C
BETAP C
ZS C
ZP C
ALP C
GSS C
GSP C
GPP C
GP2 C
HSP C
```

Step 2 **GAFit** using the information from File 16.7 configures itself.

Step 3 **GAFit** creates a whole population of individuals. Each individual is a coefficient set.

Step 4 **GAFit** writes the file *mopac2009.input* with one set of coefficients –or a whole population, depending upon configuration–. File 16.8.

File 16.8: *mopac2009.input* file

```

3.963742
4.707052
8.613357
-13.268145
-30.000657
-74.414557
-22.103403
-4.673270
4.940829
-1.073867
2.199698
-14.336436
-8.429824
-3.522071
-10.090874
-8.412029
```

Step 5 **GAFit** launches the external program with one parameter: the number of coefficients.

```
external-mopac2009.sh 16
```

Step 6 **external-mopac2009.sh** launches **injector 16** which create the needed files to run the MOPAC 2009 task:

- *mopac_input.mop*, a copy of File 16.4 where the @ is replaced to point the file below –File 16.8–.

- a copy of File 16.8.

Step 7 **external-mopac2009.sh** launches MOPAC 2009 on *mopac_input.mop*, as input file, running the task with *mopac_input.out* as output: File 16.9, where near most the lines are omitted and the tree individual task are shown.

File 16.9: *mopac_input.out* file

```
 [...]

  ***************************************************************************
  **                                                                       **
  **                              MOPAC2009                                **
  **                                                                       **
  ***************************************************************************

 [...]

AM1 precise external=A geo-ok nosym
  Sheep #A#

   ATOM    CHEMICAL      BOND LENGTH      BOND ANGLE      TWIST ANGLE
  NUMBER    SYMBOL      (ANGSTROMS)       (DEGREES)       (DEGREES)
   (I)                     NA:I            NB:NA:I        NC:NB:NA:I      NA    NB    NC
     1        H          0.00000000       0.0000000       0.0000000
     2        C          1.09852142  *    0.0000000       0.0000000       1     0     0

 [...]

  TOTAL CPU TIME:              0.08 SECONDS

  == MOPAC DONE ==

 [...]

 oldgeo AM1 precise external=A force geo-ok nosym

 [...]

  TOTAL CPU TIME:              0.16 SECONDS

  == MOPAC DONE ==

 [...]

AM1 precise ts external=A geo-ok nosym

 [...]

  TOTAL CPU TIME:              0.24 SECONDS

  == MOPAC DONE ==
```

Step 8 **external-mopac2009.sh** launches **extractor** which extracts data from the mopac 2009 output –*mopac_input.out*– writing it to *extracted.data*, File 16.10.

File 16.10: *extracted.data* file

```
0 0 6
3
0 0 0
-285.89460
0 0 1
-1196.18301
0 0 2
7
0 0 3
1 H 0.0000 0.0000 0.0000
0 0 3
2 C 7.5565 0.0000 0.0000
0 0 3
[...]
```

The structure is described in Section 9.5.

Step 9 **external-mopac2009.sh** launches **fitter** which using the *extracted.data* file evaluate the coefficients –File 16.11– writing the result to *mopac2009.fit* –File 16.12–.

File 16.11: Output: **fitter** evaluation

```
 DELTA            calc=    140.22725000000014        ref=    100.59999999999999        we=   ↩
        ↪0.10000000000000001          cont=   1.55164336304490329E-002
 FREQUENCY        calc=    98.739999999999995        ref=    3271.0000000000000        we=   ↩
        ↪1.0000000000000005E-004   cont=   9.40538249390785976E-005
 DISTANCE         calc=    3.6829195484017840        ref=    3.7003090959999998        we=    ↩
        ↪100.00000000000000         cont=   2.20851605509992830E-003
   individual             1    fit=   1.78190035104880407E-002
```

Step 10 **external-mopac2009.sh** finishes, and control returns to **GAFit** which apply the *mopac2009.fit* values to genetic selection.

File 16.12: *mopac2009.fit* file

```
   1.78190035104880407E-002
```

Step 11 **GAFit** runs steps from **Step 4** to here for each coefficient set to evaluate.

Step 12 if **GAFit** does not meet a condition to stop, it jumps to **Step 3**.

A reduced output example is shown in File 16.13. At the end, there are the best coefficients set, which also can be found in the file *best.txt*.

A trick to evaluate the *best.txt* again and examine the fitting details is to copy *best.txt* over *mopac2009.input* and run the external script *external-mopac2009.sh* with **1** as its argument as shown below:

```
$ cp best.txt mopac2009.input
$ ./external-mopac2009.sh 1
extractor correct/total:1/1
  DELTA           calc=   22.545130000005884       ref=   100.59999999999999
we=  0.10000000000000001      cont=  6.02010474994563588E-002
  FREQUENCY       calc=   2117.2900000000000       ref=   3271.0000000000000
we=  1.00000000000000005E-004 cont=  1.24403393046421793E-005
  DISTANCE        calc=   3.6747770408556764       ref=   3.7003090959999998
we=   100.00000000000000        cont=  4.76097105301748029E-003
   individual            1  fit=  6.49744588917784832E-002
$
```

File 16.13: **GAFit** output

```
*************
  gafit 0.6.3
  Build: 2569
*************
autoconfiguring...

Job type: external
Command : external-mopac2009.sh

Settings for job
------------------------------
Command:[external-mopac2009.sh]
Bounds:[bounds.txt]
External input:[mopac2009.input]
External fit:[mopac2009.fit]
Coefficients: 16
Print options:
        runs yes
        ga settings no
...now reading data

Reading bounds for 16 coefficients

             1    BETAS H       -10.00000 -       +10.00000 (real)
             2       ZS H       -10.00000 -       +10.00000 (real)
             3      ALP H       -10.00000 -       +10.00000 (real)
             4      GSS H       -20.00000 -       +20.00000 (real)
             5      USS C      -100.00000 -      +100.00000 (real)
             6      UPP C      -100.00000 -      +100.00000 (real)
             7    BETAS C      -100.00000 -      +100.00000 (real)
             8    BETAP C       -10.00000 -       +10.00000 (real)
             9       ZS C       -10.00000 -       +10.00000 (real)
            10       ZP C       -10.00000 -       +10.00000 (real)
            11      ALP C       -10.00000 -       +10.00000 (real)
            12      GSS C       -20.00000 -       +20.00000 (real)
            13      GSP C       -20.00000 -       +20.00000 (real)
            14      GPP C       -20.00000 -       +20.00000 (real)
            15      GP2 C       -20.00000 -       +20.00000 (real)
            16      HSP C       -10.00000 -       +10.00000 (real)
 16 BOUNDS VECTOR

[...]

#seed#1387754553#seed#

run 1

[...]
external evaluation
-------------------
extractor correct/total:0/1
 PENALTY cont=   10000000000.000000
  individual            1  fit=   10000000000.000000

[...]
external evaluation
-------------------
extractor correct/total:1/1
 DELTA            calc=   83.978449999999953        ref=   100.59999999999999        we= ⤸
      ⤷0.10000000000000001        cont=  2.72990214184575886E-003
 FREQUENCY        calc=   434.80000000000001        ref=   3271.0000000000000        we= ⤸
      ⤷1.0000000000000005E-004  cont=  7.51817823005893499E-005
 DISTANCE         calc=   3.7084044655350095        ref=   3.7003090959999998        we=  ⤸
      ⤷100.00000000000000        cont=  4.78627171380651393E-004
  individual            1  fit=  3.28371109552699986E-003
[...]

#
#Results
#
       1    BETAS H   -5.452765783429
       2       ZS H   +1.812760345726
       3      ALP H   +5.837755188701
       4      GSS H   +15.317101171297
       5      USS C   +2.854251382320
       6      UPP C   +35.398050378277
       7    BETAS C   +45.582866967845
       8    BETAP C   -9.090224794790
       9       ZS C   -8.294661718821
      10       ZP C   -3.351313187157
      11      ALP C   +6.549632365488
      12      GSS C   +0.829685862335
      13      GSP C   +15.745782081940
      14      GPP C   +14.404053853550
      15      GP2 C   -16.514534230484
      16      HSP C   -5.122967116645
```

17 Enhanced MOPAC Interface example

Giving the Linus Torvalds Award to the Free Software Foundation is a bit like giving the Han Solo Award to the Rebel Fleet.

Richard Stallman

This example is the same as the Section 16, so we shall only show the differences.

17.1 Input and executable files

The complete enhanced interface was explained in the Section 10. To create and run the example you must type:

```
$ cd shepherd-example
$ tar xvzf mopac-shepherd.tgz
$ make test
```

After this, the files created are shown in Table 17.1.

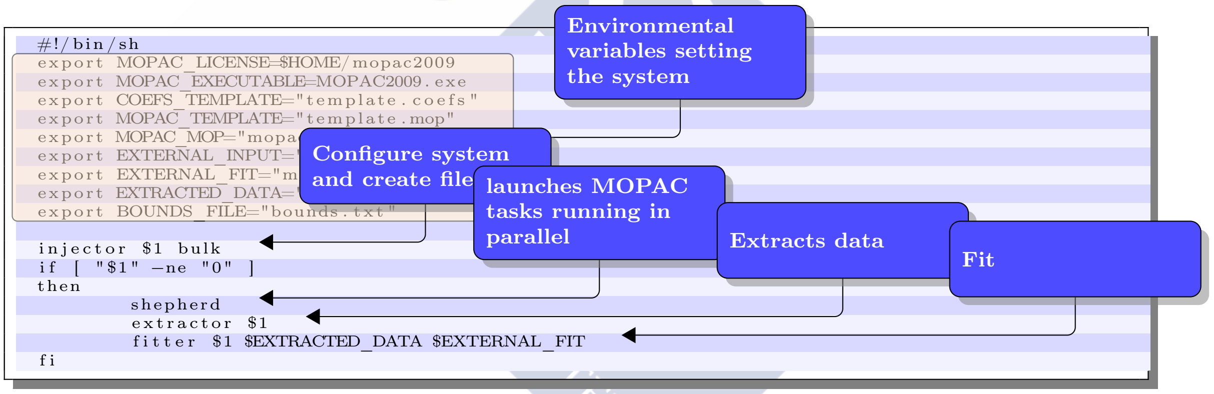
Checking files against the previous section example, you figure out that the *external-mopac.sh* file –17.1– is slighty different:

- the line "**injector $1**" is changed to "**injector $1 bulk**". As stated in 9.4, the option **bulk** brings the system to an *external bulk* configuration.

Table 17.1: Files in the shepherd-example folder after run *make test*.

File	Type	Provided by
bounds.txt	text file	example
conditions.txt	text file	example
external-mopac2009.sh	shell script	example
job.txt	job configuration	example
template.coefs	mopac2009 external coefficients	example
template.mop	mopac2009 job template	example
shepherd	binary	**GAFit**
extractor	Perl script	**GAFit**
fitter	binary	**GAFit**
injector	binary	**GAFit**

File 17.1: **external program**: *external-mopac.sh* file


```
#!/bin/sh
export MOPAC_LICENSE=$HOME/mopac2009
export MOPAC_EXECUTABLE=MOPAC2009.exe
export COEFS_TEMPLATE="template.coefs"
export MOPAC_TEMPLATE="template.mop"
export MOPAC_MOP="mopa
export EXTERNAL_INPUT="
export EXTERNAL_FIT="m
export EXTRACTED_DATA="
export BOUNDS_FILE="bounds.txt"

injector $1 bulk
if [ "$1" -ne "0" ]
then
        shepherd
        extractor $1
        fitter $1 $EXTRACTED_DATA $EXTERNAL_FIT
fi
```
Environmental variables setting the system
Configure system and create file
launches MOPAC tasks running in parallel
Extracts data
Fit

Here –See Section 6.3– a whole population of coefficient sets are passed from **GAFit**–Step 4 in page 176–.

- the line "**$MOPAC_LICENSE/$MOPAC_EXECUTABL E $MOPAC_MOP**" is replaced by "**shepherd**" only.

17.2 Running the example

The big difference with Section 16 is Step 7 where **shepherd** launches and controls MOPAC 2009 tasks running in parallel feeding them with one or various coefficient sets. The time spent processing each population is used to calculate the optimal number of concurrent tasks which varies around some optimal one.

File 17.2: **shepherd** example output

A maximum of four parallel MOPAC task to process this population

Time spent processing this population by four parallel task

The population is of 100 coefficient sets and only 9 yield correct results

```
 [...]

run  1
shepherd #flocks:4
shepherd elapsed time:53.160538
extractor correct/total:9/100
 PENALTY cont=    10000000000.000000
  individual              1   fit=    10000000000.000000
 [...]
 DELTA              calc=    7181.9406700000000          ref=    100.59999999999999          we= ↩
       ↪0.10000000000000001         cont=    495.49013755040392
 FREQUENCY          calc=    1973.5500000000000          ref=    3271.0000000000000          we= ↩
       ↪1.0000000000000005E-004  cont=  1.5733312632874934 1E-005
 DISTANCE           calc=    3.8125655102568405          ref=    3.7003090959999998          we=  ↩
       ↪100.00000000000000          cont=  9.2033581885933121 7E-002
  individual             12   fit=    495.58218686560247
 [...]
 PENALTY cont=    10000000000.000000
  individual            100   fit=    10000000000.000000
100      1        9100000596.463308334351  8.602835950342e+00
shepherd #flocks:3
shepherd elapsed time:59.095997
extractor correct/total:15/100
 DELTA              calc=   -4814.1148300000004          ref=    100.59999999
       ↪0.10000000000000001         cont=    238.67156761441231
 FREQUENCY          calc=   -8564.2099999999991          ref=    3271.0000000000000          we= ↩
       ↪1.0000000000000005E-004  cont=  1.30915433567136 81
 DISTANCE           calc=    2.2739708397426739          ref=                                 we=  ↩
       ↪100.00000000000000          cont=  14.858297674795789
  individual              1   fit=    253.53117444354376
 [...]
 PENALTY cont=    10000000000.000000
  individual            100   fit=    10000000000.000000
shepherd #flocks:3
shepherd elapsed time:37.518653
 [...]
 DELTA              calc=    60.117649999999998          ref=    100.59999999999999          we= ↩
       ↪0.10000000000000001         cont=  1.6193304008182515 8E-002
 FREQUENCY          calc=    1896.5699999999999          ref=    3271.0000000000000          we= ↩
       ↪1.0000000000000005E-004  cont=  1.7655668412022650 1E-005
 DISTANCE           calc=    3.6998361990769268          ref=    3.7003090959999998          we=  ↩
       ↪100.00000000000000          cont=  1.6332661827924222 3E-006
  individual            100   fit=  1.6212592942777329 8E-002


#
#Results
#
          1     BETAS  H    +3.742722536450
          2        ZS  H    +2.452440978714
          3       ALP  H    +2.008292084018
          4       GSS  H    -7.976934072834
          5       USS  C    -8.767498674896
          6       UPP  C   -17.380794123604
          7     BETAS  C   -15.645160881834
          8     BETAP  C    -2.524336159684
          9        ZS  C    +3.400840391719
         10        ZP  C    -0.531486822561
         11       ALP  C    +5.087837269451
         12       GSS  C   -10.748657465697
         13       GSP  C    -1.973897569441
         14       GPP  C    -7.256002543392
         15       GP2  C    +7.265166397026
         16       HSP  C    +5.160470194181
```

Coefficient set with correct results. Fit: 253.53117444354376

Coefficient set failed

The best of all in this run. Saved into *best.txt* file

So there are a lot of files named A, B, C, ..., AA, AB, ...–following the **GAFit**'s *automatic coefficient names* convention, as explained in Section 6.5–, each of them containing a unique coefficient set to be used as external file for the *mopac template* –See Step 6 in page 176–. In the example, 100 sets comprised from A to CV.

Also, the *mopac template* file is cloned to a file named taking into account the first and last coefficient set to calculate in the task. For example, if the first coefficient set is the first of all –A coefficient set file– and the last the 29th –AB coefficient set file–, the file cloned would be *A-AB.mop*. This is a "*flock*" of 29 "*sheep*".

This behaviour is restricted in the code to a one set only: one set per MOPAC 2009 task –a *sheep* per *flock*–, so the *mopac template* file is cloned to files like *A-A.mop*, *B-B.mop*, ..., *CV-CV.mop*. See Section 10.2 about **burst** mode if you want to change this behaviour.

After processing an entire population by **shepherd**, **extractor** extracts the data and **fitter** evaluates it as shown in Section 16.

Here, we can use the same trick –Section 16.3– evaluating the *best.txt* to examine the fitting details:

```
$ cp best.txt mopac2009.input
$ ./external-mopac2009.sh 1
  shepherd #flocks:1
  shepherd elapsed time:0.338015
  extractor correct/total:1/1
  DELTA           calc=   22.545130000005884       ref=   100.59999999999999
we=  0.10000000000000001      cont=  6.0201047499456358 8E-002
  FREQUENCY       calc=   2117.2900000000000       ref=   3271.0000000000000
we=  1.0000000000000005E-004 cont=  1.2440339304642179 3E-005
  DISTANCE        calc=   3.6747770408556764       ref=   3.7003090959999998
we=   100.00000000000000        cont=  4.7609710530174802 9E-003
   individual            1  fit=  6.4974458891778483 2E-002
$
```

Part IV

Applications

18 Collision-induced dissociation mechanisms of $[Li(uracil)]^+$

Chemistry is a class you take in high school or college, where you figure out two plus two is 10, or something.

Dennis Rodman, ex NBA player

Roberto Rodríguez-Fernández, Saulo A. Vázquez and Emilio Martínez-Núñez

Departamento de Química Física and Centro Singular de Investigación en Química Biológica y Materiales Moleculares, Campus Vida, Universidad de Santiago de Compostela, 15782 Santiago de Compostela, Spain

18.1 Abstract

The Collision-Induced Dissociation (CID) of the $[Li(uracil)]^+$complex with Xe is studied by means of quasi-classical trajectory calculations. The potential energy surface is obtained "on the fly" from Austin Model 1 (AM1) semiempirical calculations, supplemented with two-body analytical potentials to model the intermolecular interactions. The simulations show that Li^+production is the primary channel, in agreement with a previous experimental study [M.T. Rodgers and P.B. Armentrout. In: *J. Am. Chem. Soc.* 122 (2000), pp. 8548–8558]. Collision-induced isomerization of $[Li(uracil)]^+$was found to be very important as well in the

2.5–10 eV collision energy range. Three minor channels are also identified: complex formation between Xe and $[Li(uracil)]^+$, ligand exchange to form $LiXe^+$, and fragmentations of the uracil ring, which are strongly nonstatistical. Additional quasi-classical trajectory calculations carried out to investigate in more detail the fragmentations of the uracil ring reveal the presence of bifurcations in the potential energy surface, as trajectories starting from a transition state give rise to four different product channels. The integral cross sections for Li^+production calculated in this work agree well with those obtained in the experiments only for the lowest collision energies, being ~ 20 times greater than the experimental values for a collision energy of 10 eV. Finally, the initial translational energy is transferred preferentially to the $[Li(uracil)]^+$vibrational degrees of freedom, with energy transfer to rotation being modest. The amount of energy transfer to the different degrees of freedom as a function of the collision energy follows quite nicely a model recently proposed in our group.

18.2 Introduction

Collision-Induced Dissociation (CID) is an experimental technique consisting of an initial energy activation of a molecular ion (or projectile), through collisions with a neutral target (usually a rare gas atom), and a subsequent dissociation of the projectile. CID studies have helped elucidate the structure, bond energies and kinetics of molecular ions, ranging from small molecules to peptides.[18–39]

The initial step in CID involves energy transfer to the molecular ion. For collision energies below 100 eV, only the ro-vibrational states of the ion are involved in the energy transfer process, i.e., no electronic excitation is expected.[38,40–42] Previous theoretical studies have shown that the initial relative translational energy can be preferentially transferred to the vibrational or rotational degrees of freedom of the molecular ion, depending on its structure.[43–48] Rotational excitation of the projectile usually occurs when it presents a planar structure[43], whereas for spherically shaped molecules, translational to vibrational energy transfer is usually preferred.[43,45]

The dissociation process can take place by two limiting mechanisms.

One is a direct, or "shattering"[49,50] mechanism, in which one or more bonds of the ion break within one vibrational period. The other limiting mechanism can be described by the Rice–Ramsperger–Kassel–Marcus (RRKM) or related statistical theories[51], as fragmentation occurs after Intramolecular Vibrational Energy Redistribution (IVR) redistribution takes place. The collisions with the target activate just a subset of normal modes of the projectile and, if a complete IVR does not take place, the molecular ion can dissociate following pathways that are not among the lowest energy ones.[52,53] The importance of "shattering" and non-RRKM mechanisms to understand CID in small molecules, amino acid and peptides has been highlighted in a number of previous experimental and simulation studies.[45,52–58]

In this paper, the CID of $[Li(uracil)]^+$with Xe atoms is studied by means of quasi-classical trajectory calculations. Uracil is one of the four nucleobases of ribonucleic acid (RNA), and its fragmentation mechanisms in gas phase induced by ion, electron or proton impact, or by photoionization have been thoroughly investigated.[59–68] All these studies agree in that the initial step of the fragmentation gives rise to isocyanic acid (HCNO) and the C_3H_3NO fragment through a retro Diels-Alder[69,70] mechanism. Li^+forms stable complexes with uracil, binding more strongly at the oxygen atoms. The $O — Li^+$ bond dissociation energy was derived using guided ion beam mass spectrometry and *ab initio* calculations, and is ~ 48 kcal/mol.[17] In their study, Rodgers and Armentrout studied the CID of a number of $[M(L)]^+$ complexes with Xe, with $M^+ = Li^+$, Na^+ and K^+, and L=uracil, tymine and adenine.[17] The primary channel found in their study is the endothermic loss of the neutral molecule, but some of the $[M(L)]^+$ complexes experience ligand exchange to form MXe^+. For $[Li(uracil)]^+$, only $Li^+$was detected, though.[17]

The present simulation study complements the previous experimental study of Rodgers and Armentrout on the CID of $[Li(uracil)]^+$with Xe.[17] The integral cross sections for Li^+production obtained in our simulation will be compared with those obtained in the experiments. Additionally, the differential cross sections for Li^+production, not measured experimentally, will be reported here. Our quasi-classical trajectory calculations identify minor fragmentation products of $[Li(uracil)]^+$, not detected in the experiments. The present paper will also show how some of these new mechanisms can only be described by dynamical model like those employed here. Finally, the efficiency of energy transfer in the CID process is studied with the help of a recently developed energy transfer model.[71,72]

18.3 Computational details

18.3.1 Potential energy surface

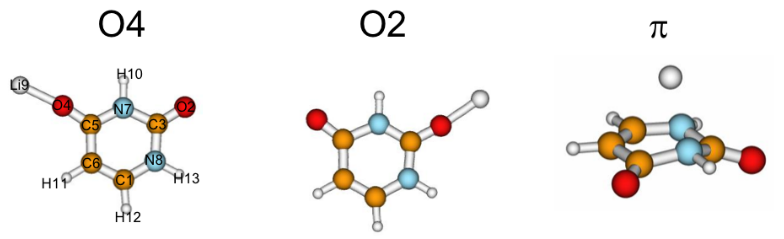
The potential energy surface of the system is expressed as:

$$V = V_{intra} + V_{inter} \tag{18.3.1}$$

where V_{intra} is the intramolecular energy of $[Li(uracil)]^+$and V_{inter} models the interaction energy between Xe and $[Li(uracil)]^+$. The intramolecular potential is calculated "on the fly" from semiempirical calculations. In particular, the AM1 Hamiltonian[73] is employed with parameters that have been reparametrized to reproduce the main features of the $[Li(uracil)]^+$energy landscape. Specifically, Li^+can bind to the oxygen atoms (labeled here as O_4 and O_2) or to the π-electrons of uracil (see Figure 18.1). O_4 is the preferred binding site, being the complex formed with Li^+bound to O_2 $\sim$ 4 kcal/mol less stable. When Li^+binds to the π-electrons, the geometry of uracil is strongly distorted and the energy of this complex is $\sim$ 30 kcal/mol above the O_4 global minimum.[17] MP2(FC)/6-31+G(d) calculations, carried out in this work, serve as a benchmark for the reparametrization of the AM1 Hamiltonian, which is described in detail in the Electronic Supplementary Information (ESI). The MP2(FC)/6-31+G(d) level of theory affords energy values for the stationary points of Figure 18.1 and for the uracil + Li^+dissociation limit that agree very well with previous MP2(full)/6-311+G(2d,2p)//MP2(full)/6-31G(d) calculations (see Table 18.6 of the ESI).[17]

The $Xe/[Li(uracil)]^+$interaction potential V_{inter} was obtained from RI-MP2(FC)/def2-QZVPP[74] single point energy calculations for the twelve relative orientations of Xe and $[Li(uracil)]^+$shown in Figure 18.10

Figure 18.1: Atomic labeling and geometries of [Li(uracil)]$^+$optimized in this work at the MP2/6-31+G(d) level of theory.



of the ESI. The *ab initio* calculations, carried out with Turbomole[75], include the counterpoise correction to account for the Basis Set Superposition Error (BSSE).[76] The analytical function employed to fit the fp-CCSD(T)/CBS interaction energies is a sum of generalized two-body Buckingham potentials:

$$V_{inter} = \sum_i A_i e^{-B_i R_i} + \frac{C_i}{R_i^{D_i}} + \frac{E_i}{R_i^{F_i}} \tag{18.3.2}$$

where i represents each of the atoms of [Li(uracil)]$^+$, R_i is the Xe-i interatomic distance and $A_i, B_i, \ldots F_i$ are the parameters. During the fit, which was conducted using a genetic algorithm[2], some restrictions were imposed to the parameters to avoid the Buckingham catastrophe, namely, all E_i parameters are positive and $F_i > D_i + 3$. The resulting parameters and the fit are shown in Table 18.7 and Figure 18.10 of the ESI. The stratified root-mean-square error of the fit is 0.2 kcal/mol for energies lower than 2 kcal/mol, and 2.6 kcal/mol for energies in the range 2-200 kcal/mol.

18.3.2 Chemical dynamics simulations

Quasiclassical Trajectory (QCT) calculations for the CID of [Li(uracil)]$^+$with Xe were carried out with the general molecular-dynamics package VENUS05.[77] The initial conditions were selected to simulate the experimental CID study of Rodgers and Armentrout.[17] In particular, the internal energies of [Li(uracil)]$^+$are described by a Maxwell-Boltzmann distribution of states at 300 K. The [Li(uracil)]$^+$ions were randomly oriented and the initial separation between its center of mass and Xe was 15 Å. The trajectories are integrated using a sixth-order Adams–Moulton predictor-corrector algorithm with a fixed time step of 0.02 fs, which ensured an average energy conservation of 99.997% of the total energy. Each trajectory was halted when Xe is 20 Å away from [Li(uracil)]$^+$, which corresponds to an integration time between 0.6 and 1.5 ps, depending on the collision energy. Then, the geometries are checked for possible O–Li$^+$bond dissociations, O4 $\longrightarrow$ O2 isomerizations, Li$^+$Xe formation, and other possible fragmentations channels of [Li(uracil)]$^+$. From the final geometries and the momenta, the relative translational energies of the Xe and [Li(uracil)]$^+$fragments, and the rotational and vibrational energies of [Li(uracil)]$^+$are also computed.

Table 18.1: Computational details of the quasi-classical trajectory simulations carried out in this work to calculate the cross sections and energy transfer.

	N_{traj}[a]	Collision energies (eV)	b_{max} (Å)
Cross section	10^4	2.5	3.0
		3	4.0
		4	4.5
		5	5.2
		6	5.3
		7	5.3
		8	5.4
		9	5.4
		10	5.5
Energy transfer	10^3	3, 4, 5, 6, 7, 8, 9, 10	12[b]

[a] Number of trajectories in each ensemble.
[b] The maximum impact parameter is 12 Å for all collision energies.

Two different QCT simulations are carried out in this work. One simulation type was devised to calculate cross sections for Li$^+$production. In the other one, the focus is on calculating average energy transfer efficiencies. The details of each calculation are summarized in Table 18.1. In the cross sections calculation, the impact parameters b were chosen randomly from the equation $b = b_{max}R^{1/2}$, where R is a random number and the maximum impact parameters b_{max} are selected in separate runs using batches of 10^3 trajectories. Then, the O–Li$^+$dissociation cross sections are obtained from

$$\sigma = \frac{N_{diss}}{N_{tot}}\pi b^2_{max} \tag{18.3.3}$$

where N_{diss} and N_{tot} are the number of trajectories leading to uracil + $Li^+$and the total number of trajectories, respectively. A particular type of dissociations are those that occur in a very short time scale after the collision; this mechanism has been observed in previous work[45,52] and is called "shattering". "Shattering" dissociations are identified by the number of O–Li^+ Inner Turning Points (ITPs) after the collision, i.e., this mechanism takes place if the number of ITPs is ≤ 1.

In order to make a detailed comparison with the experiments, possible O–Li^+ dissociations taking place in the experimental time scale of $10^{-4}s$ should be considered in our study.[17] This was accomplished by running Kinetic Monte Carlo (KMC) simulations[78,79] for the $[Li(uracil)]^+$ions that remain undissociated after the QCT simulation and have vibrational energies greater than the O–Li^+dissociation energy (including the zero-point energy). KMC is a very useful Monte Carlo simulation for modeling the transient behavior of various molecular species that participate in highly coupled chemical reactions. The method is an alternative to the traditional procedure of numerically solving the deterministic reaction rate equations. The chemical processes employed in our KMC simulations are the O4 $\longrightarrow$ O2 and O2 $\longrightarrow$ O4 isomerizations, and the O–Li^+dissociations from the O2 and O4 isomers. To calculate the dissociation-isomerization probabilities needed in the KMC calculations, RRKM rate constants are employed. For the dissociation reactions, variational RRKM calculations[51] were carried out, using an average of 20 Hessians[80] per dissociation path to obtain the corresponding sums of states. The $[Li(uracil)]^+$ions are monitored for $10^{-4}s$, and the dissociations taking place in this time window are regarded as statistical or following an RRKM mechanism. The inclusion of other channels in this analysis, like other fragmentations of $[Li(uracil)]^+$, does not change the KMC simulation results.

Additionally, the energy transfer efficiencies are calculated in separate QCT runs (see Table 18.1 for the details). Since low impact parameter trajectories contribute more significantly to energy transfer, the initial impact parameter is chosen with importance sampling[81] from the distribution

$$f(b)\,\mathrm{d}b = \frac{\mathrm{d}b}{b_{max}} \tag{18.3.4}$$

The average amount of energy transfer $\langle \Delta E^{traj} \rangle$ can then be calculated as

$$\Delta E^{traj} = \frac{1}{N}\sum_{i=1}^{N} \frac{2b_i}{b_{max}} \Delta E_i \tag{18.3.5}$$

with ΔE_i and b_i being the amount of energy transfer and impact parameter of trajectory i, respectively. The maximum impact parameter b_{max} was 12 Å for all collision energies, for which the change in the internal energy of [Li(uracil)]$^+$ is less than 30 cm^{-1}. The energy transfer values calculated from the trajectories $\langle \Delta E^{traj} \rangle$ can be related to experimental obtained quantities $\langle \Delta E \rangle$ by the ratio of the collision cross sections.[82,83]

$$\Delta E = \frac{\pi b_{max}^2}{\pi \sigma_{LJ}^2 \Omega^{(2,2)*}} \Delta E \tag{18.3.6}$$

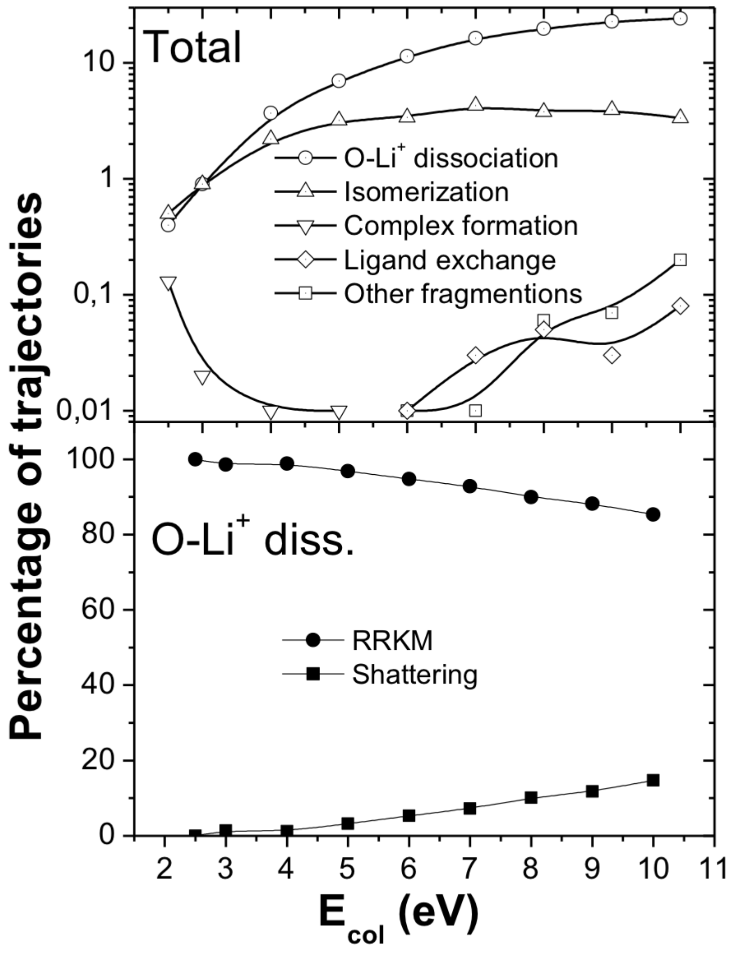
where the product $\pi \sigma_{LJ}^2 \Omega^{(2,2)*}$ is the Lennard-Jones (LJ) collision cross-section that has been obtained using the program COLRATE.[84] The Lennard-Jones parameters for Xe are taken from the literature[85,86] and those for [Li(uracil)]$^+$are obtained employing the Lorentz-Berthelot combining rules[87] after fitting a Lennard Jones potential for the interaction of Xe with [Li(uracil)]$^+$to the intermolecular potential energy surface obtained above using eqn (18.3.2). The intermolecular function of eqn (18.3.2) has been properly averaged over thousands of configurations for each distance between Xe and the center of mass of the cation (see the ESI for further details).

18.4 Results and Discussion

18.4.1 Trajectory types

Figure 18.2 shows the percentage of the different processes found in our CID study as a function of the collision energy. As seen in the top panel of the figure, O–Li$^+$bond dissociations dominate at most of the

Figure 18.2: Percentage of the different channels found in this study, with respect to the total number of trajectories (upper panel) and percentage of RRKM and "shattering" found in the O–Li$^+$dissociations (lower panel).

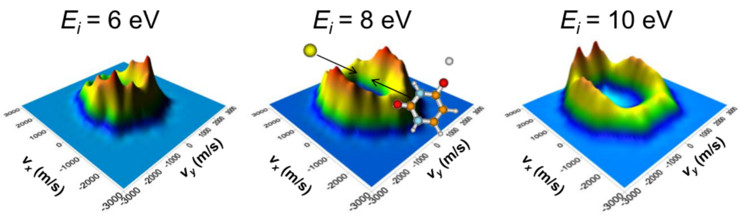


collision energies. This process experiences a 50-fold increase in the collision energy range of our study. As seen in the lower panel of Figure 18.2, the vast majority (more than 85%) of the O–Li$^+$bond dissociations follow an RRKM mechanism. The percentage of prompt or "shattering" O–Li$^+$dissociations is modest and increases with collision energy. The maximum percentage of "shattering" is 15% and occurs for the highest collision energy of 10 eV.

The amount of "shattering" dissociations found in this study is significantly lower than that found in previous CID studies in our group.[45,52] In particular, when $Cr(CO)_6^+$ collides with Xe at $E_{col} = 5\ eV$, "shattering" amounts 63% of the total Cr–O dissociations.[45] At a collision energy of 21.7 eV, "shattering" accounts for more than 80% of the CH_3SH^+ fragmentations, induced by collisions with Ar; for this ion the fragmentation products are CH_2^+, CH_3^+ and CH_3S^+.[52]

The small percentage of "shattering" found in the present study, compared with our previous CID studies, can be explained by looking at the different sizes/number of bonds of the projectiles. $[Li(uracil)]^+$is rather big and the O–Li^+bond is located in one end of the ion, which means that the probability of Xe hitting this part of the molecule is small. By contrast, CH_3SH^+ is much smaller and each of the bonds is susceptible to suffer prompt dissociations. Also, in $Cr(CO)_6^+$ all the collisions with Xe affect at least one Cr–O bond.

Figure 18.3: Scattering maps for uracil obtained for the collision energies of 6, 8 and 10 eV.



Additional analysis of the CID dynamics can be obtained from the angle-velocity distributions of the scattered uracil molecules. In Figure 18.3 we depict polar uracil scattering maps at the collision energies of 6, 8 and 10eV; cylindrical symmetry is assumed in the scattering process.[33] These plots are obtained from the angular distributions of the uracil fragments at the end of the reactive trajectories, i.e., those leading to O–Li^+dissociations. It should be noted that these plots only reflect the distributions of the prompt O–Li^+dissociations and do not consider the dissociations that occur through an RRKM mechanism. The uracil fragments are scattered anisotropically, with a change from backward scattering to forward scattering as the collision energy increases; at $E_{col} = 8\ eV$ the products are more sideways scattered. This trend has also been observed in other CID studies and can be explained by the fact that, as the collision energy increases, there are more grazing collisions leading to O–Li^+dissociation.[45,52]

After O–Li$^+$dissociations, the second major process observed in our study is the O4 $\longrightarrow$ O2 isomerization reaction (see Figure 18.2 top panel). Even though the isomerization barrier is well below the O–Li$^+$ dissociation energy (25.7 $kcal/mol$ vs 49.1 $kcal/mol$, respectively), the O–Li$^+$dissociation is a barrierless process and its RRKM rate constants are much larger than those for the isomerization, because the entropic factor favors dissociation over isomerization. The collision-induced isomerizations are enhanced with collision energy plateauing at around 4% for energies above $5 - 6$ eV.

Three minor channels have been identified in this study. The formation of complexes between Xe and [Li(uracil)]$^+$is one of them. Although previous CID studies pointed out the importance of complex-mediated mechanisms[45], in the present study, though, formation of complexes is unimportant. The percentage of trajectories forming those complexes is about 0.1% at the lowest collision energy, being the average lifetime of the complexes 3.6 ps. For $E_{col} > 5$ eV all collisions are direct, without complex formation. The observation of a negligible number of collision complexes is consistent with the above anisotropic angle-velocity distribution plots, since the existence of transiently bound rotating collision complexes lead to isotropic angular distributions. On the other hand, in Figure 18.10 of the ESI it can be seen that the deepest potential well in the intermolecular potentials is ~ -6 $kcal/mol$ and corresponds to Xe interacting with Li$^+$. This value is less than half that obtained for the interaction between Xe and Cr$^+$, which amounts -13.5 $kcal/mol$.[45] This and direct ejection of one (sometimes two) CO ligands upon collision with Xe explain why in the Xe + Cr(CO)$_6^+$ CID study the formation of $\left[\text{XeCr(CO)}_\text{n}\right]^+$ ($n < 6$) complexes was very important.

Ligand exchange to form Li$^+$Xe is another minor channel found in this study. The collision energy dependence of ligand exchange is at variance with that found above for complex formation. In particular, ligand exchange does not take place for collision energies lower than 6 eV. Nevertheless, for high collision energies it is also a negligible channel. Actually, Rodgers and Armentrout did not observe this process for [Li(uracil)]$^+$, although for other complexes like [Na(uracil)]$^+$, it represents a feasible mechanism, whose cross sections increase with E_{col}.[17]

According to our simulations, ligand exchange to form Li$^+$Xe has a cross section $\sim$ 200 times lower than those for Li$^+$production, which might difficult the experimental detection. Even though this channel is insignificant, two different mechanisms leading to Li$^+$Xe have been identified in our simulations. In the first one, Xe hits the Li$^+$side of the molecular ion and directly strips away the Li$^+$cation. The second mechanism is an indirect process, where the O–Li$^+$bond is broken first, with Li$^+$scattering in the same direction as Xe, allowing Li$^+$Xe to be formed.

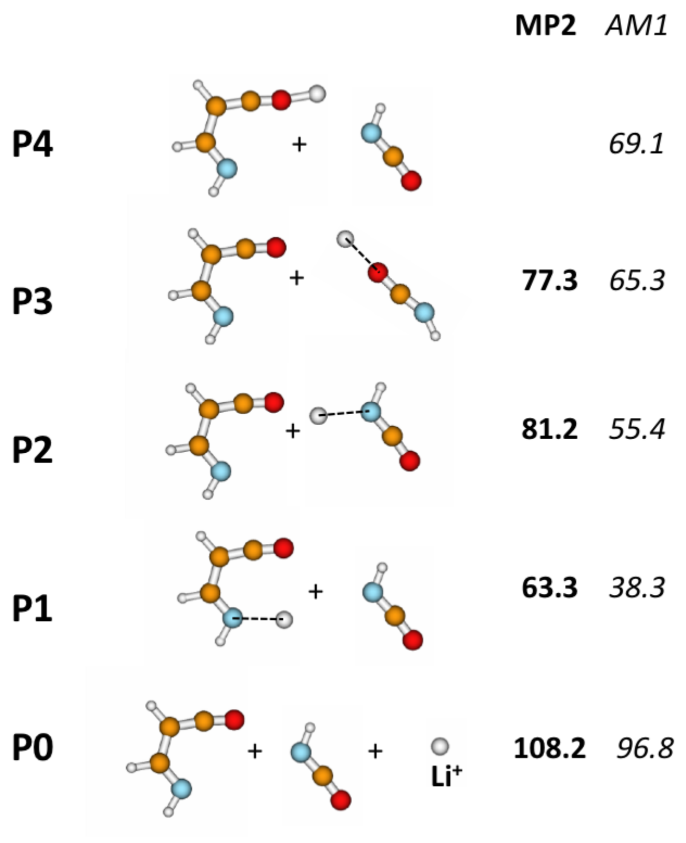
Finally, a small number of trajectories experience other fragmentations of [Li(uracil)]$^+$, where the uracil ring is broken. These additional channels are analyzed in more detail in the next section.

18.4.2 Aditional fragmentation channels of [Li(uracil)]$^+$

The fragmentation channels of the ion that involve ring opening increase in importance as the collision energy increases. At $E_{col} = 10\ eV$, besides O–Li$^+$dissociations, which amounts 24% of the total number of trajectories, 0.21% of all trajectories experience additional fragmentation channels of the [Li(uracil)]$^+$complex. All these channels involve the rupture of the C5—N7 bond (see Figure 18.1 for the atom labeling), and in many of them the C3—N8 bond is also broken.

As seen in Figure 18.4, cleavage of the C5—N7 and C3—N8 bonds produce isocyanic acid (HCNO) and the C_3H_3NO fragment, with Li$^+$dissociated (channel P0), attached to a nitrogen atom (channels P1 and P2), or to an oxygen atom (channels P3 and P4) of any of the fragments. A recent DFT study of the fragmentation channels of uracil reveals that the C5—N7 and C3—N8 bonds are more easily cleaved than any other one in the molecule.[88]The same pattern is observed here for [Li(uracil)]$^+$. Figure 18.4 also collects the dissociation energies calculated for each of the channels at the MP2/6-31+G(d) and AM1 Specific Reaction Parameters (AM1-SRP) levels of theory. Since the AM1 Hamiltonian was not reparametrized to reproduce these additional fragmentation channels, it provides energies that are systematically lower than those obtained with MP2. However, both methods agree that the lowest energy channel is P1 and the highest energy channel is P0,

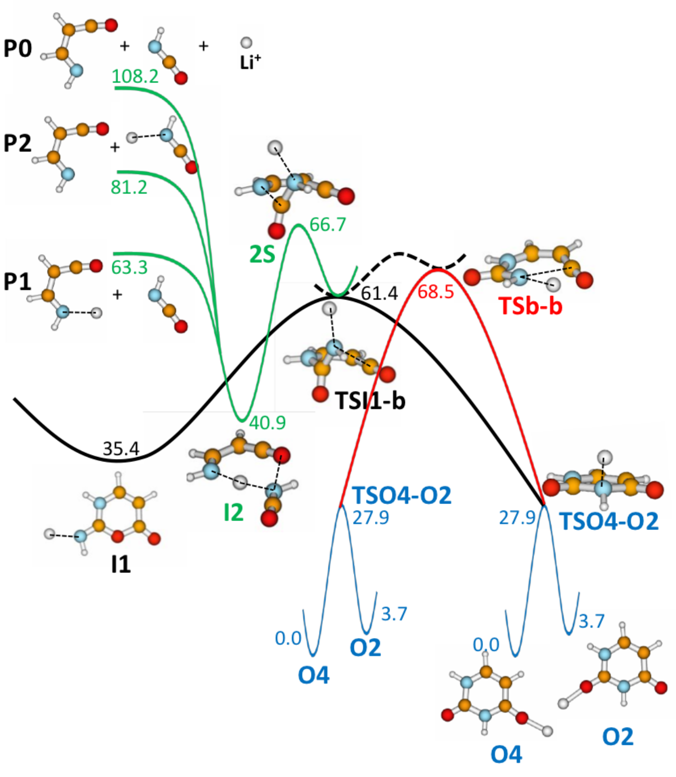
Figure 18.4: Fragmentation channels of [Li(uracil)]$^+$involving ring opening. The MP2/6-31+G(d) and AM1-SRP dissociation energies are indicated in kcal/mol.



the energy difference between both channels being ∼ 45(58) with the MP2(AM1) method.

Of the 21 reactive trajectories mentioned above, 2/1/11/1/1 trajectories gave rise to P0/P1/P2/P3/P4, respectively, with the remaining 5 trajectories breaking only the C5—N7 bond. Quite interestingly, the vast majority of the trajectories give rise to P2, which is not the lowest energy channel. This nonstatistical behavior is a consequence of weak couplings between the reaction coordinates involved in these fragmentation channels and the remaining internal degrees of freedom of the molecule and/or to non-IRC dynamics.[57] The combination of collisional activation, which results in a nonrandom vibrational excitation, with weak couplings among the various degrees of freedom, explains why nonstatistical behavior is sometimes manifested in CID.[52,53]

Figure 18.5: Potential energy profiles involved in the [Li(uracil)]$^+$ring opening channels P0-P2. The profiles are computed at the MP2/6-31+G(d) level of theory and energies are given in kcal/mol.



A close inspection of the 21 trajectories revealed that, in all of them, the C5——N7 bond is broken first, with Li$^+$remaining close to N7 for quite a long time, which suggests the presence of a flat area of the surface associated to these geometries. To investigate this in detail some snapshots were taken from the 21 trajectories to be subsequently employed as input geometries in MP2/6-31+G(d) optimizations. Using this procedure, the transition states TSI1-b and TSb-b of Figure 18.5 were obtained. Specifically, the transition state TSI1-b connects, according to an intrinsic reaction coordinate (IRC) calculation,[89] the 6-membered ring I1 with the lowest energy isomer O4. The reaction coordinate is primarily composed of a rotation about the C3——N8 bond. However, additional trajectory calculations started at TSI1-b (*vide infra*) indicate the presence of a bifurcation,[90,91] in the reaction path, i.e., TSI1-b is actually connected to TSO4-O2, as seen in Figure 18.5. Additionally, from the TSI1-b structure, the molecule can break the C3——N8 bond, after surmounting a second order saddle point S2, which lies only 5.3$kcal/mol$ above TSI1-b. A geometry optimization from a geometry close to 2S leads to the I2 complex, which can go on and dissociate to either P0, P1 or P2. I2 is not the only possible complex between Li$^+$, the isocyanic acid and C_3H_3NO and several were optimized in this work but, for simplicity, in Figure 18.5 only I2 is depicted.

The reaction coordinate associated with the other transition state TSb-b, which is nearly planar, primarily involves wagging of Li$^+$. An IRC calculation from TSb-b shows that going downhill in either direction leads to the O4 minimum. However, quasi-classical trajectory calculations (*vide infra*) that start from this transition state show that a bifurcation exists in the reaction path, as drawn in Figure 18.5. Just like for TSI1-b, a second order saddle point that connects TS1b-b with the P0-P2 fragmentation products could be located, but only with the HF method; at the HF/6-31+G(d) level of theory this saddle point lies only 6.6$kcal/mol$ above TSb-b. Finally, TSI1-b and TSb-b differ by only $\sim 7kcal/mol$ and a path obtained by interpolation of the geometries of both transition states shows a small barrier of 2.7$kcal/mol$ from TSb-b. Attempts to locate a saddle point connecting these two transition states were unsuccessful.

Table 18.2: Simulation results for the trajectories starting from the TSI1-b and TSb-b transition states.

		Results[b]			
Initialization	N_{traj}[a]	I1	P0-P4	O4	O2
TSI1-b	157	58	56	18	25
TSb-b	160	–	105	46	9

[a] Number of trajectories in each ensemble.
[b] Number of trajectories that finish in each of the stationary points indicated at the top. The trajectories labeled under P0-P4 indicate that they either finish in the I2 complex or in a related complex or dissociate to the P0-P4 products (see text).

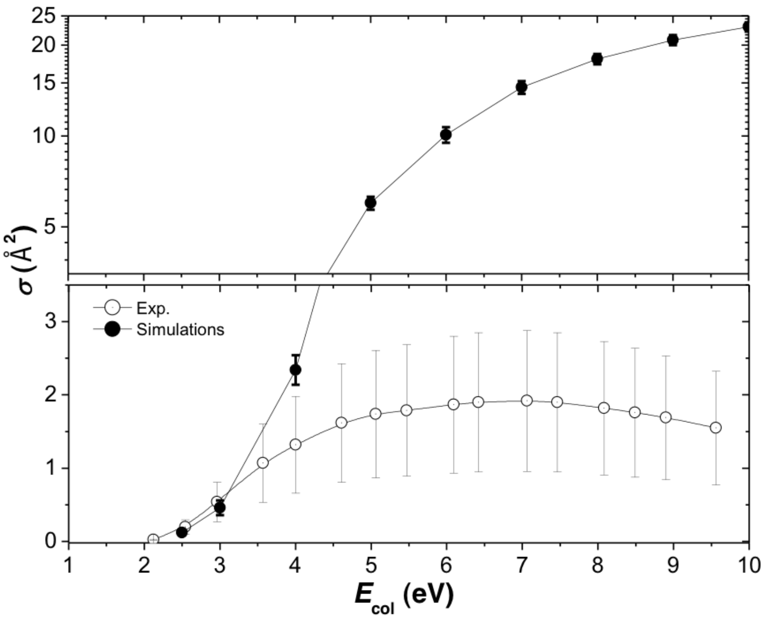
To further investigate the fragmentation dynamics of [Li(uracil)]$^+$, quasi-classical trajectory simulations, starting from either TSI1-b or TSb-b, were run "on the fly" at the HF/3-21G level of theory, using an interface between VENUS[77] and NWCHEM.[92,93] The trajectories were integrated for 500 fs with a Hessian-based predictor-corrector algorithm[94] with a step size of 0.2 fs, using the quasiclassical normal mode sampling method[95,96] to prepare a quantum mechanical microcanonical ensemble with a total energy of $100kcal/mol$ above the corresponding transition state. The summary of the trajectory results is shown in Table 18.2. As indicated above, I2 is not the unique complex found in this study and others exist with similar energies and low interconversion barriers. The results under the column P0-P4 involve those trajectories ending up in the I2 complex or in any other complex between Li^+, isocyanic acid and C_3H_3NO. A small fraction (1%) of the trajectories that followed this mechanism dissociated to P1, the lowest energy dissociation channel. Following the trajectories for longer times would give more fragmentations.

Starting from TSI1-b, 37% of the trajectories led to I1, 36% to the P0-P4 dissociation products and 27% to the O2/O4 isomers. The O2 isomer is 1.4 times more abundant than O4, even though, as indicated above, the IRC calculation connects TSI1-b with O4. The 160 trajectories initiated at TSb-b led primarily to dissociation (66%) with 34% leading to O2/O4, in this case the O4:O2 ratio being 5:1. Even though the P0-P4 dissociation channels involve a barrier (second order saddle point), dissociation is the second most important channel for the trajectories that start from TSI1-b, and the most important one by far for those starting from TSb-b. These results point out very weak couplings between the torsion about the C3–N8 bond and the C3–N8 stretching, and also between Li^+out-of-plane and the C3 — N8 stretching.

The potential energy surface involved in the P0-P4 fragmentation channels is very flat and makes necessary the study of this system by dynamical models like quasi-classical trajectories. It is actually very interesting to see how trajectories that start from a transition state lead to four different products (like those starting from TSI1-b), a result which is difficult to anticipate by a simple inspection of the PES. The presence of plateaus on the potential energy surface like those found here around TSI1-b and TSb-b and/or bifurcations precludes the use of kinetic models like RRKM or transition state theory, as they cannot predict the product ratio.[57,90,91,97]

18.4.3 Comparison with experiment

Figure 18.6: Comparison between QCT and experimental[17] integral cross sections for Li$^+$production. The error bars in the simulation results represent the 95% confidence limits. The experimental error bars are taken from Rodgers and Armentrout [17].



In Figure 18.6 the cross sections for Li$^+$production calculated in this work are compared with the experimental results of Rodgers and Armentrout.[17] While the agreement between the experimental and the simulation results is very good for the lowest collision energies ($E_{col} < 3.5\ eV$), for higher energies though, the simulation cross sections are an order of magnitude greater than the experimental ones. The experimental values of σ present a maximum at around $7\ eV$ and then decrease with E_{col}, while those obtained in the simulations steadily increase with the collision energy.

Previous CID studies in our group show that the simulation cross sections are systematically higher than the experimental ones in the high energy region.[45,52,53] This trend is clearly enhanced in the system that is being studied here. Much of the disagreement between the experimental data and the simulation results might arise from the inherent difficulties of experimental CID techniques to collect efficiently product ions as the collision energies increases. This is because the electric field generated by the octopole (in a typical CID apparatus) does not trap very efficiently ions resulted from sideways scattering, especially when the ions are light, as in the present case.[98,99] In fact, Anderson and coworkers[56,99] advised for the need of care in interpreting guided-ion-beam cross sections at high collision energies. However, the $\sim$ 30-fold difference between simulations and experiment observed here can not be explained based on the above arguments alone, and possible inaccuracies of the potential energy surface lead to large errors in the σ computed values.

18.4.4 Energy transfer

A phenomenological model for energy transfer has been recently developed in our group.[71,72] The model is an adaptation of two limiting models of energy transfer, originally developed for atom-diatom collisions. One limit of the model refers to the adiabatic (*ad*) or low-collision energy regime and the other one is applied to impulsive (*imp*) collisions. The model was successfully employed for gas-surface collisions,[71,72] just by replacing the original diatom by a projectile, and with the surface playing the role of the atom. The amount of energy transfer to each of the degrees of freedom of the projectile is thus:

$$\begin{aligned}\Delta E &= \Delta E^{ad} + \Delta E^{imp} \\ &= a_0 + a_1 \exp\left(\frac{-b_1}{\sqrt{E_{col}}}\right) + a_2 \operatorname{cosech}^2\left(\frac{b_2}{\sqrt{E_{col}}}\right)\end{aligned} \tag{18.4.1}$$

with a_0 determined assuming complete accommodation of the projectile at low energies and a_1, b_1, a_2 and b_2 being adjustable parameters. Eqn (18.4.1) predicts for $\langle \Delta E \rangle$ a linear dependence on E_{col} for high collision energies; the $E_{col} \to \infty$ limit of $\langle \Delta E_{int} \rangle / E_{col} = a_2/b_2$.[26] Fairly constant values of $\langle \Delta E_{int} \rangle / E_{col}$ (for high E_{col}) were previously found for a number of gas-surface and gas-molecule systems,[43,71,100–102] which lends support to the model. The energy transfer results of the present work also show the same trend, with $\langle \Delta E_{int} \rangle / E_{col}$ being nearly constant and in the range $0.22 - 0.24$.

Figure 18.7: Average energy transfer values computed in this work for Xe + [Li(uracil)]$^+$according to eqn (18.3.6).

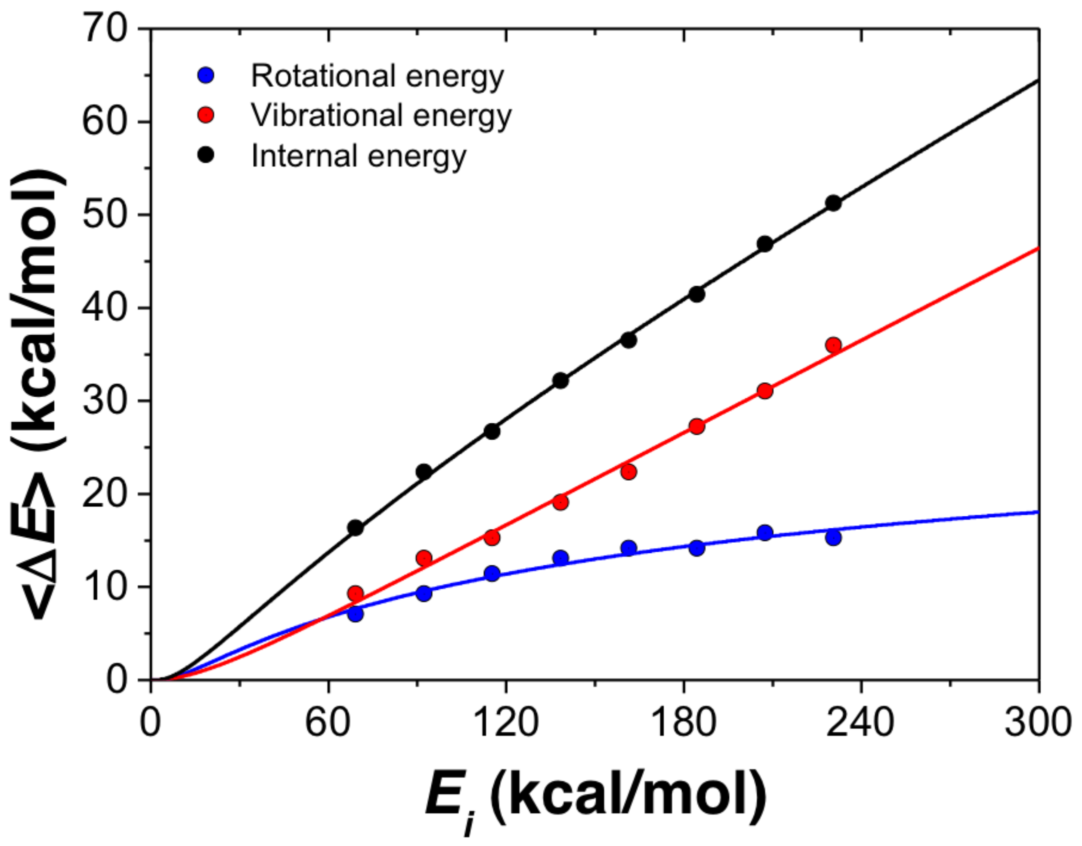


Table 18.3: Parameters of the energy transfer model (eqn 18.4.1) fitted to our simulation data.

	Parameters[a]			
	a_1	b_1	a_2	b_2
Rotation	39.8	13.7	0.0	0.0
Vibration	6.5	10.0	20.2	11.1

[a] a_1 and a_2 are in kcal/mol and b_1 and b_2 in $(kcal/mol)^{1/2}$

It is of interest to see if the above model can predict energy transfer efficiencies, irrespective of the nature of the collision partners. For that reason, the model is fit here to the energy transfer values in Xe + [Li(uracil)]$^+$collisions. Figure 18.7 and Table 18.3 show the fit of eqn (18.4.1) to the trajectory data and the adjusted parameters, respectively. As seen in the figure, except for the lowest collision energies, $\langle E_{vib} \rangle$ is greater than $\langle E_{rot} \rangle$. The percentages of energy transferred to vibration (rotation) are 13 (10); 14 (10); 13 (10); 14 (9); 14 (9); 15 (8); 15 (8); 16 (7) for $E_{col} = 3, 4, 5, 6, 7, 8, 9$ *and* 10 *eV* respectively. The values of $\langle E_{vib} \rangle / E_{col}$ remain fairly constant with E_{col}, while those of $\langle E_{rot} \rangle / E_{col}$ diminish with the collision energy. In previous work, it has been shown that the relative percentages of $\langle E_{vib} \rangle / E_{col}$ and $\langle E_{rot} \rangle / E_{col}$ depend on the structure of the projectile.[43–48] Energy transfer to rotation (R) was found to be very important for Ar + protonated urea[48] and Ar + planarAl clusters, namely, Al_6 (C_{2h}), and Al_{13} (D_{2h} and D_{6h}).[43] On the contrary, collisions with spherically shaped molecules, like $Cr(CO)_6^+$and Al_6 (O_h) or Al_{13} (D_{3d}), translational (T) to vibrational (V) energy transfer is preferred.[43,45]The presence of low vibrational frequencies is very important to enhance $T \rightarrow V$ energy transfer.[43] In fact, while $T \rightarrow R$ dominates in Ar + protonatedurea collisions,[48] $Ar + \left[Ca(urea)\right]^{2+}$ simulation results show the opposite trend, with $T \rightarrow V$ energy transfer being preferred.[47] The results of the present paper seem to resemble those of $Ar + \left[Ca(urea)\right]^{2+}$, with $T \rightarrow V$ being clearly the predominant energy transfer pathway, and are at odds with previous work on Ar + planarAl clusters.[43] This is a surprising result, as being $[Li(uracil)]^+$planar, one would expect, on account of the previous CID studies,[43] an enhancement of $T \rightarrow R$ energy transfer. Cleary more work is needed to understand the dynamics of energy transfer in CID studies.

18.5 Conclusions

Quasi-classical trajectory calculations are employed in this work to study the dynamics of CID of the $[Li(uracil)]^+$complex with Xe. The AM1 semiempirical Hamiltonian is reparametrized to reproduce the main features of the $[Li(uracil)]^+ \rightarrow uracil + Li^+$ potential energy surface. Additionally, two-body Buckingham potentials are employed to model the interaction of $[Li(uracil)]^+$with Xe.

The analysis of our simulations indicates that the vast majority of the reactive trajectories either dissociate to give uracil + $Li^+$and/or result in isomerization from O4 to O2. Three minor but interesting channels are also found here: Formation of complexes between Xe and $[Li(uracil)]^+$, $LiXe^+$production and fragmentations of the uracil ring. All of them account for less than 0.25% of the trajectories. While $LiXe^+$production and fragmentations of the uracil ring are increasingly important with collision energy, complex formation is more frequent as the collision energy decreases. Ligand exchange to produce $LiXe^+$ can be achieved by two different mechanisms: a direct collision of Xe with Li^+, which results in an immediate Li^+—Xe bond formation. The second mechanism involves O—Li^+ dissociation with Li^+scattering side-by-side with Xe.

The uracil ring can be broken to give isocyanic acid + C_3H_3NO with Li^+bound to one of the fragments. The most abundant channel is that where $Li^+$is bound to the N atom of icocyanic acid. The fragmentations of the uracil ring present interesting dynamical effects, namely, weak couplings between the normal modes of the molecular ion and the presence of bifurcations in the potential energy surface. Slow IVR is manifested in the CID dynamics, as the major fragmentation product of the uracil ring is not the lowest energy channel. The presence of bifurcations in the PES has been identified in additional quasi-classical trajectory calculations at the HF/3-21G level of theory, as trajectories that start from a transition state lead to four different channels.

Computed integral cross sections only agree with the experimental values at the lowest collision energies. At the highest collision energy of 10 eV, the simulation cross sections are two orders of magnitude higher than the experimental ones. The differential cross sections computed here indicate strong side-ways scattering, which might affect the experimental detection of ions.

Even though $[Li(uracil)]^+$is planar, energy is preferentially transferred from translation to vibration, rather than to rotation. This result is at odds with a previous theoretical CID study, which shows that the rotational degrees of freedom are preferentially excited when the projectile has a planar structure. A model recently proposed in our research group fits very well the simulation energy transfer data as a function of the collision energy.

18.6 Acknowledgements

The authors thank "Centro de Supercomputación de Galicia" (CESGA) for the use of their facilities

18.7 Electronic Supplementary Information

18.7.1 Intramolecular [Li(uracil)]$^+$ Potential Energy Surface

Figure 18.8 shows a contour plot for Li$^+$orbiting around uracil calculated at the MP2(FC)/6-31+G(d) level of theory. The plot clearly shows the two minima at O4 (lower right corner, $\theta \sim 120°$) and O2 (lower left corner $\theta \sim 0°$) and the transition state connecting both ($\theta \sim 60°$. The AM1 contour plot fails to describe the main features of the energy landscape. Therefore the AM1 Hamiltonian is reparametrized to fit MP2(FC)/6-31+G(d) calculations. The resulting AM1 Hamiltonian is termed AM1 with Specific Reaction Parameters (SRP). Only the parameters for H, Li and O have been optimized in the fitting. In practice the following function is minimized using the general optimization program SUPOPT:

$$f = \sum_i \left(X_i - X_i^{target}\right)^2 \omega_i \tag{18.7.1}$$

where X_i is a semiempirical molecular property of our system (an energy difference, the geometry of a stationary point or the frecuencies) and X_i^{target} is the corresponding target value, taken in our case from MP2(FC)/6-31+G(d) calculations. In our case we selected the energies for several values of R and θ (see Figure 18.8 and Table 18.4 and also some properties of the O4 and O2 minima (distances, angles and frequencies). The values of X in Table 18.4 correspond to the AM1-SRP optimized values. The AM1-SRP optimized parameters are collected in Table 18.5. Figure 18.8 shows how the reparametrized potential (AM1-SRP) nicely reproduces the MP2(FC)/6-31+G(d) features. Table 18.6 collects the energies of the main stationary points in the PES. Quite interestingly, the high energy isomer π and the transition state connecting O4 and O2 are also nicely reproduced by our AM1-SRP Hamiltonian, even though those structures are not included in the optimization procedure. Figure 18.9 compares the main geometrical features of the stationary points of Table 1. Overall, the AM1-SRP potential seems to account for the most important details of the $[\mathrm{Li(uracil)}]^+$ PES.

18.7.2 LJ parameters for $[\mathrm{Li(uracil)}]^+$

The LJ paremters σ and ϵ of the ion, needed in eqn (18.3.6), are obtained by fitting a LJ potential for the interaction between Xe and $[\mathrm{Li(uracil)}]^+$ to the intermolecular potential of Figure 18.10. In particular, for each distance between Xe and the center of mass of the cation a total of 10^4 configurations are generated by randomly rotating the molecular over its Euler angles to obtain an average intermolecular potential between both species. Figure 18.11 shows both the averaged intermolecular potential (black line), and the fitted LJ potential (red line). The resulting LJ parameters are: $\epsilon =$ 12.7 $kcal/mol$ and $\sigma =$ 5.6 Å.

Once the LJ parameters are determined, the collision integral $\Omega^{(2,2)*}$, which is a function of the reduced temperature $T^* = k_B T/\epsilon$, is approximated by:[103]

$$\Omega^{(2,2)*} = (0.636 + 0.567 \log_{10} T^*)^{-1} \tag{18.7.2}$$

$$\Omega^{(2,2)*} = (0.697 + 0.518 \log_{10} T^*)^{-1} \tag{18.7.3}$$

$$\Omega^{(2,2)*} = 1.161(T^*)^{-0.14874} + 0.525 \exp(-0.773T^*) + 2.162 \exp(-2.438T^*) \tag{18.7.4}$$

Equation (18.7.2 is accurate within ±7% in the range $0.3 \leq T^* \leq 500$, Eqn (18.7.3) within ±2.5% in the range $3 \leq T^* \leq 300$, and Eq. (18.7.4) within ±0.16% in the range $0.3 \leq T^* \leq 100$.

18.7.3 Tables

Table 18.4: Molecular properties selected for the reparametrization of the AM1 Hamiltonian.

Property[a]	X	X^{target}	ω_i
D_e of O4	49.1	48.9	1
D_e of O4	45.0	45.5	1
d_{Li-O4} in O4	1.803	1.755	100
d_{Li-O2} in O2	1.811	1.760	100
$a_{Li-O4-C}$ in O4	171.8	171.9	0.5
$a_{Li-O2-C}$ in O2	174.3	173.8	0.5
$Freq(30^{th})$ of O4	3098	3100	0.01
$Freq(31^{st})$ of O4	3206	3200	0.01
$Freq(32^{nd})$ of O4	3396	3400	0.01
$Freq(33^{th})$ of O4	3402	3400	0.01
$E(4, 40)$	42.5	42.0	1
$E(4.3, 40)$	43.3	41.7	1
$E(4.5, 40)$	44.4	42.1	1
$E(7, 40)$	47.3	47.1	1
$E(10, 40)$	48.7	49.0	1
$E(4, 50)$	53.8	53.9	1
$E(4.3, 50)$	50.2	50.2	1
$E(4.5, 50)$	49.6	49.1	1
$E(7, 50)$	47.9	48.0	1
$E(10, 50)$	48.6	49.0	1
$E(4, 60)$	56.2	57.0	1
$E(4.3, 60)$	51.1	52.0	1
$E(4.5, 60)$	50.1	50.4	1
$E(7, 60)$	47.7	47.9	1
$E(10, 60)$	48.6	48.9	1
$E(4, 70)$	48.1	48.6	1
$E(4.3, 70)$	45.9	45.9	1
$E(4.5, 70)$	46.0	45.3	1
$E(7, 70)$	46.9	46.9	1
$E(10, 70)$	48.3	48.6	1
$E(4, 80)$	31.3	32.6	1
$E(4.3, 80)$	34.2	33.5	1
$E(4.5, 80)$	36.6	34.8	1
$E(7, 80)$	45.4	45.1	1
$E(10, 80)$	47.9	48.1	1

[a] D_e is the dissociation energy in kcal/mol, d is a distance in Å, a is an angle in degrees, $Freq$(ith) is the ith frequency in cm^{-1} of the O4 minimum and $E(R, \theta)$ is an energy in kcal/mol (with respect to the O4 minimum) of a geometry defined by R (in Å), and θ (in degrees) defined in Figure 18.8.

Table 18.5: AM1-SRP optimized parameters.

Parameter	Atom	Value
USS	H	-11.481457
ZS	H	1.178108
BETAS	H	-6.060738
GSS	H	13.038821
ALP	H	2.920401
USS	Li	-3.938666
UPP	Li	-4.680952
ZS	Li	0.732507
ZP	Li	0.823119
BETAS	Li	-1.124105
BETAP	Li	-1.282833
GSS	Li	12.136146
GSP	Li	5.721065
GPP	Li	5.626132
GP2	Li	5.679185
HSP	Li	0.859623
ALP	Li	1.367705
USS	O	-96.991556
UPP	O	-80.094474
ZS	O	3.108050
ZP	O	2.549183
BETAS	O	-29.014009
BETAP	O	-29.202568
GSS	O	15.258495
GSP	O	13.592248
GPP	O	16.023733
GP2	O	13.051632
HSP	O	0.997777
ALP	O	4.428909

Table 18.6: Computed relative energies (in kcal/mol) of the main stationary points of the $[Li(uracil)]^+$system.

	MP2(full)[a]	MP2/6-31+G(d)	AM1-SRP
O4	0.0	0.0	0.0
O2	3.7	3.7	4.1
π	33.3	31.6	28.9
TSO4-O2[b]		27.9	25.7
Uracil + Li^+	49.6	48.9	49.1

[a] MP2(full)/6-311+G(2d,2p) single point calculations at the MP2(full)/6-31G(d) optimized geometries from Rodgers and Armentrout [17].
[b] Transition state connecting the O4 and O2 isomers.

Table 18.7: Parameters for the two-body intermolecular potentials[a].

	A	B	C	D	E	F
Xe -C1	14225.136	2.5347848	-595.35486	4.3007982	20.837055	15.909242
Xe-N8	65267.206	3.0366646	-1178.7832	5.3567748	78.400985	13.299376
Xe-C3	28869.129	2.9870266	-780.21341	4.945167	36.73045	12.22095
Xe-N7	54992.446	2.9437494	-1128.5838	5.2205507	173.21272	15.460329
Xe-C5	40173.943	3.2052366	-2396.0069	6.3649598	65.69321	17.478756
Xe-C6	37601.016	2.8349989	-954.5515	6.0897463	56.451637	15.054894
Xe-O2	58784.833	3.1539858	-1369.1143	6.3463582	92.669704	18.552539
Xe-O4	45122.726	2.9442868	-2398.5717	7.1839886	88.763898	18.26193
Xe-Li9	54362.191	3.2662061	-1288.3261	4.5772667	23.394276	20.90295
Xe-H10	46485.505	4.1310207	-626.04702	6.8741713	54.815288	18.079876
Xe-H11	28641.731	3.5712266	-805.23429	6.6524242	138.38928	11.721251
Xe-H12	28010.424	3.6458042	-597.21566	6.9939478	253.36613	13.223736
Xe-H13	52490.15	4.0446817	-950.87406	6.5572047	895.17585	16.716528

[a] Parameter A is given in $kcal\ mol^{-1}$, B in Å^{-1}, D and F are dimensionless. The units for C and E are such that the potential energy is in $kcal\ mol^{-1}$, with R in Å.

Figure 18.8: Contour plots for Li$^+$around uracil computed at the MP2/6-31+G(d), AM1 and AM1-SRP levels of theory.

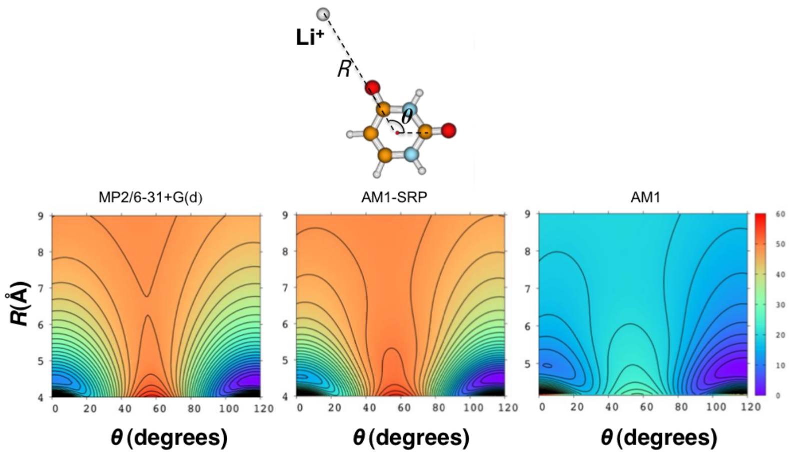


Figure 18.9: Stationary points found in our study for $[Li(uracil)]^+$at the MP2/6-31+G(d) and AM1-SRP levels of theory. Distances are given in Å.

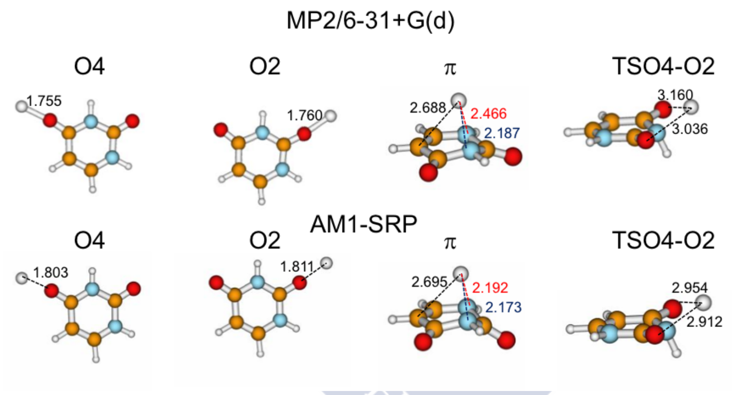


Figure 18.10: Analytical potential of eqn (18.3.2) (solid lines) fitted to the RI-MP2(FC)/def2-QZVPP *ab initio* calculations (circles) for different orientations of Xe and $[\mathrm{Li(uracil)}]^+$.

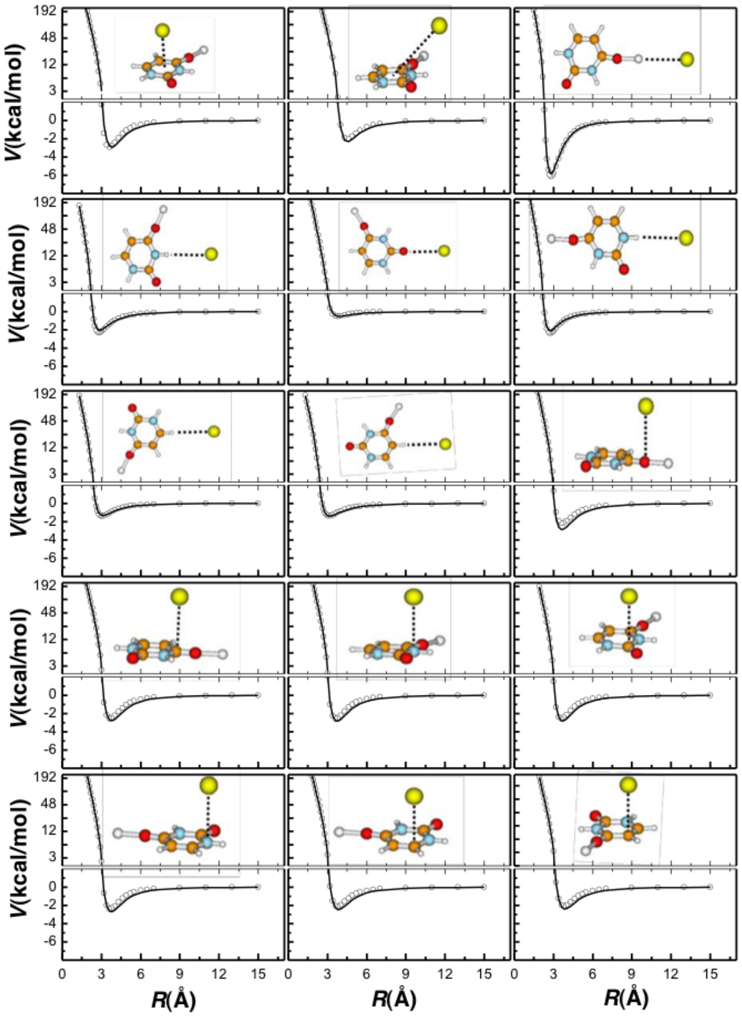
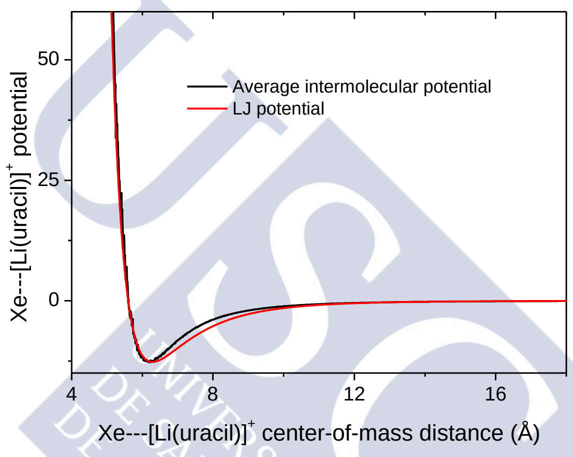


Figure 18.11: Average intermolecular potential and fitted LJ intermolecular potential for different Xe-[Li(uracil)]$^+$distances.



19 Ab initio and RRKM study of the HCN/HNC elimination channels from vinyl cyanide

Research is what I'm doing when I don't know what I'm doing.

Wernher Von Braun

Zahra Homayoon,[a,b] Saulo A. Vázquez,[a] Roberto Rodríguez-Fernández,[a] Emilio Martínez-Núñez,[a]

[a]Departamento de Química Física y Centro Singular de Investigación en Química Biológica y Materiales Moleculares, Campus Vida, Universidad de Santiago de Compostela, 15782 Santiago de Compostela, Spain and [b]Department of Chemistry, College of Sciences, Shiraz University, Shiraz 71454, Iran.

Ab initio CCSD and CCSD(T) calculations with the 6-311+G(2d,2p) and the 6-311++G(3df,3pd) basis sets were carried out to characterize the VC dissociation channels leading to hydrogen cyanide (HCN) and its isomer hydrogen isocyanide (HNC). Our computations predict three elimination channels giving rise to HCN and another four channels leading to HNC formation. The relative HCN/HNC branching ratios as a function of internal energy of VC were computed using RRKM theory and the KMC method. At low internal energies (120 $kcal/mol$), the total HCN/HNC ratio is about 14, but at 148$kcal/mol$ (193nm) this ratio becomes 1.9, in contrast with the value 124 obtained in a previ-

ous *ab initio*/RRKM study at 193 *nm* [A. Derecskei-Kovacs and S.W. North. In: *J. Chem. Phys.* 110 (1999), p. 2862]. Moreover, our theoretical results predict a ratio of rovibrationally excited acetylene over total acetylene of 3.3, in perfect agreement with very recent experimental measurements [M. J. Wilhelm et al. In: *J. Chem. Phys.* 130 (2009), p. 044307].

19.1 Introduction

The eliminations of hydrogen halides (HX, X = F, Cl and Br) in photofragmentations of haloethylenes have been the subject of numerous theoretical and experimental studies.[106–130] Photoexcitation of these compounds at 193*nm* leads to several competing channels for HX formation. HX eliminations may proceed either via three-centered or four-centered transition states. The analysis of the product energy partitioning in the photodissociation of haloethylenes indicates that HX eliminations take place on the ground electronic state PES after internal conversion from the electronic excited state[113,114,119,120,130]

The photodissociation of the more complex VC chemical species has also been studied recently.[104,105,131–135] Like for the haloethylenes, VC exhibits radical and molecular elimination channels when it is photoexcited at 193 *nm*. Fahr and Laufer[133] suggested that the formation of triplet vinylidene ($:CCH_2$) and HCN is a dominant channel in the photodissociation of the molecule at 190 *nm* using time-resolved UV absorption spectroscopy. Blank et al.[132], later on, suggested that all dissociation channels occur on the ground electronic state S_0 after internal conversion from the electronically excited S_1 state. The reaction mechanism leading to triplet vinylidene and HCN was ruled out based on energy conservation arguments.[132] In a more recent study, Wilhelm et al.[105] rendered further support to the exclusion of the triplet vinylidene + HCN reaction path, using time-resolved Fourier transform infrared emission spectroscopy (TR-FTIRES).

Derecskei-Kovacs and North[104] performed *ab initio* calculations of the dissociation channels of VC and found two different pathways for HCN formation. The first one proceeds via a three-centered (3C) transition state forming singlet vinylidene and hydrogen cyanide (HCN) and the second one proceeds via a four-centered (4C) transition state giving rise to acetylene (HCCH) and hydrogen isocyanide (HNC) as coproduct. According to their RRKM calculations, the 3C process is 124 times faster than the 4C process for an energy corresponding to a photon wavelength of 193 *nm*. This means that, according to the HCN(HNC) reaction paths described above, the HCN/HNC ratio would be 124; i.e., HNC formation is negligible. Later on Letendre and Dai[134], using the TR-FTIRES technique, found significant IR emission from acetylene and HNC, which is in contradiction with the *ab initio* results of Derecskei-Kovacs and North.[104] However, the relative importance of the HCN and HNC reaction mechanisms was not determined in the experimental study.

In the most recent experimental study on the photodissociation of VC and perdeuterovinyl cyanide at 193 *nm*, Wilhelm et al.[105] were able to discern the relative abundance of the HCN and HNC products. They deduced the HCN/HNC ratio assuming that the *ab initio* calculations of Derecskei-Kovacs and North[104] mentioned above were correct. In particular they assumed that HCN is formed alongside with vinylidene and HNC is formed with acetylene. Additionally, if vinylidene is formed as a coproduct of HCN, the rapid vinylidene-acetylene isomerization process will lead to highly rovibrationally excited acetylene, since the process is very exothermic.[126] When acetylene is formed as a coproduct of HNC, no isomerization takes place and its rovibrational energy distribution will be colder. Following these arguments and their measured ratio of excited acetylene to total acetylene formation (0.77) they obtained a value for the HCN/HNC branching ratio of 3.3 (0.77/0.23). Wilhelm et al.[105] claimed that the *ab initio* calculations of the transition states for the HCN and HNC formation processes should be re-examined since the RRKM branching ratio calculated in the *ab initio* study is 124,[104] in sharp contrast with their experimentally determined value of 3.3. In particular, Wilhelm et al.[105] suggest that the 4C transition state should be $\sim$ 10.5 $kcal/mol$ below the 3C transition state to reconcile theory with experiment. However, the previous *ab initio* study predicts the 4C transition state to lie $\sim$ 9 $kcal/mol$ above the 3C transition state.[104] In our opinion, it seems more plausible that new HCN(HNC) reaction paths, not found in the previous *ab initio* study,[104] may contribute significantly to the formation of HCN or its isomer HNC.

In this paper further electronic structure calculations were performed to characterize the relevant regions of the ground state PES associated with the HCN and HNC eliminations from VC. Microcanonical $k(E)$ rate constants were also computed as a function of the internal energy of VC using RRKM theory. With the calculated $k(E)$ and modeling the kinetics using a Monte Carlo technique the HCN and HNC product abundances were evaluated for the different reaction paths and compared with the recent experimental results of Wilhelm et al.[105]

19.2 Computational details

A. Electronic structure calculations.

Ab initio calculations were carried out to model the ground state PES of the HCN and HNC elimination channels from VC cyanide. The calculations involve CCSD/6-311+G(2d,2p) optimizations and frequency analyses to characterize the stationary points as minima or saddle points, and to evaluate zero-point vibrational energies (ZPE). The minimum energy path (MEP)[136] was followed at the MP2/6-31+G(d,p) level of theory to make sure that a transition structure connects with the expected minima.

In order to obtain accurate energies, we also performed CCSD(T)/6 -311++G(3df,3pd) single point calculations at the CCSD/6-311+G(2d, 2p) optimized geometries [these calculations are referred as CCSD(T)/ 6-311++G(3df,3pd)//CCSD/6-311+G(2d,2p)].

The GAUSSIAN09 program package.[137] was employed for all the electronic structure calculations.

B. Kinetic calculations.

In this study, the following elementary steps associated with the elimination of HCN (paths I-III) and HNC (paths IV-VII) were taken into account (see also Figures 19.1 and 19.2):

Path I:

$$\text{VC} \xrightarrow{k_1} \text{:CCH}_2 + \text{HCN} \qquad (1)$$

Path II:

$$\mathrm{VC} \xrightarrow{k_2} \mathrm{HCCH} + \mathrm{HCN} \tag{2}$$

Path III:

$$\mathrm{VC} \overset{k_3}{\underset{k_4}{\longleftrightarrow}} \mathrm{Intl\text{-}III} \tag{3}$$

$$\mathrm{Intl\text{-}III} \xrightarrow{k_5} \mathrm{HCCH} + \mathrm{HCN} \tag{4}$$

Path IV:

$$\mathrm{VC} \overset{k_6}{\underset{k_7}{\longleftrightarrow}} \mathrm{Intl\text{-}IV} \tag{5}$$

$$\mathrm{Intl\text{-}IV} \xrightarrow{k_8} \mathrm{HCCH} + \mathrm{HNC} \tag{6}$$

Path V:

$$\mathrm{VC} \overset{k_9}{\underset{k_{10}}{\longleftrightarrow}} \mathrm{Intl\text{-}V} \tag{7}$$

$$\mathrm{Intl\text{-}V} \xrightarrow{k_{11}} \mathrm{:CCH_2} + \mathrm{HNC} \tag{8}$$

Path VI:

$$\mathrm{Intl\text{-}III} \xrightarrow{k_{12}} \mathrm{:CCH_2} + \mathrm{HNC} \tag{9}$$

Path VII:

$$\mathrm{Intl\text{-}III} \xrightarrow{k_{13}} \mathrm{HCCH} + \mathrm{HNC} \tag{10}$$

Rate coefficients $k_8(E)$ were calculated assuming that Int1-IV is connected with the products via TS3-IV, since the energy of Int2-IV, after adding the ZPEs, is above that of TS2-IV (see Figure 19.2). Additionally, for the rate coefficients $k_{11}(E)$, the assumption is that once TS2-V is overcome, vinylidene and HNC are obtained. Both assumptions seem reasonable but nevertheless, as will be seen below, the contribution of paths IV and V to the formation of HNC and HCN is negligible in the whole range of energies studied here, in comparison with the others paths.

The microcanonical rate coefficients $k_i(E)$ (with $i = 1 - 13$) were calculated using RRKM theory:[138]

$$k_i(E) = \sigma \frac{\sum_n P_i(E - \epsilon_n^{tS,i}}{h\rho_i(E)} \tag{19.2.1}$$

where σ is the reaction path degeneracy, $\rho_i(E)$ is the density of states at the reactant, $P_i(E)$ is the one-dimensional tunneling probability as a function of energy E and $\epsilon_n^{tS,i}$ are the vibrational energy levels of the transition state for elementary step i. In the classical limit of no tunneling, the numerator of eqn 19.2.1 tends to the total number of states at the transition state with energy less than or equal to E. A generalized Eckart potential was used to calculate $P_i(E)$, and the density of states were evaluated by direct count of the harmonic vibrational states using the Beyer-Swinehart algorithm. The CCSD(T)/6-311++G (3df,3pd)//CCSD/6-311+G(2d,2p) energies and the CCSD/6-311+G(2 d,2p) frequencies were employed for these calculations. The vibrational frequencies for all the stationary states are collected in Table 19.2 of the Supporting Information.

The method used here to model the time evolution of reactants, intermediates and products is KMC.[78,79] KMC is a very useful Monte Carlo simulation for modeling the transient behavior of various molecular species that participate in many highly coupled chemical reactions. The method is an alternative to the traditional procedure of numerically solving the deterministic reaction rate equations.

To calculate the populations of all the species involved in the photodissociation of VC as a function of time, the above reaction paths were considered and the RRKM rates calculated as above were employed. To obtain an average value of these populations, 10 KMC runs were performed for each internal energies of VC with the same initial conditions. Each of the 10 KMC runs differ in the random number seed used in the stochastic procedure. Using a higher number of KMC runs for each energy does not change the branching ratios obtained in this study.

19.3 Results and Discussion

A. Electronic structure calculations.

The different paths found in our electronic structure calculations for the HCN and HNC eliminations from VC are shown graphically in Figures 19.1 and 19.2, which include relative energies and ZPE contributions. A total of three HCN elimination channels (paths I-III in Figure 19.1) and four HNC elimination channels (paths IV-VII in Figure 19.2) were found in this study. The geometries of all the stationary points found in this study is collected in section 19.6.2 of the Supporting Information.

Path I proceeds via a three-centered transition state (TS1-I) and is the same as that found by Derecskei-Kovacs and North[104] (see Figure 3 of their paper) in their *ab initio* study. The energies of the stationary points found in our study agree very well with their QCISD(T)/6-311 ++G(d,p)//MP2/6-31G(d,p) calculations.[104] In particular, TS1-I lies 100.6 $kcal/mol$ above VC according to our CCSD(T)/6-311++G(3df, 3pd)//CCSD/6-311+G(2d,2p) calculations, in comparison with 100.8 $kcal/mol$, found by Derecskei-Kovacs and North (their transition state structure is called A2 in Figure 3 of their paper).[104]

Figure 19.3 shows the path connecting vinylidene ($:CCH_2$) and acetylene (HCCH). The CCSD(T)/6-311++G(3df,3pd)//CCSD/6-311+G(2 d,2p) calculations predict an isomerization transition state that lies only 1 $kcal/mol$ above vinylidene, suggesting a rapid isomerization process. In a previous chemical dynamics simulation study, conducted in our research group, a lifetime of only 37 fs was found for vinilydene.[126] This value is in agreement with that estimated by Ervin et al.,[139] $40-200$ fs, based on negative ion photodetachment spectral linewidths. Considering the vinylidene-acetylene potential energy profile, it was also suggested that part of the reverse isomerization barrier (of 43.3 $kcal/mol$) may be released as translational energy of the fragments in the photodissociation of vinyl chloride.[114] This suggestion was confirmed in our previous chemical dynamics study and the snapshots of the trajectories showed a concerted mechanism with isomerization (of vinylidene to acetylene) and (HCl) elimination occurring at the same time in the dissociation of vinyl chloride.[126] A similar mechanism was invoked by Blank et al.[132] to explain the product energy partitioning found in the photodissociation of VC at 193 nm.

The other two paths that generate HCN (II and III in Figure 1) were not found in the previous *ab initio* study.[104] Path II proceeds via a four-centered planar transition state (TS1-II) that lies 118.0 $kcal/mol$ above VC. HCN is produced with acetylene (instead as vinylidene) as coproduct. Path III first involves the isomerization of VC to vinyl isocyanide (Int1-III).[140] Then, vinyl isocyanide gives rise to HCN and acetylene, a process that goes through a five-centered transition state (TS2-III). The energy of this transition state is 115.3 $kcal/mol$ with respect to VC.

Paths IV−VII in Figure 19.2 form hydrogen isocyanide (HNC). Path IV proceeds via a planar three-centered transition state (TS1-IV), which is the same as transition state A1 in the previous *ab initio* study.[104] They refer to this path as a four-centered mechanism, because they claim it connects the reactant with acetylene and HNC directly (see Figure 4 of their paper).[104] However, MEP calculations performed here show that TS1-IV actually connects the reactant with cycloprop-2-enimine (int1-IV in Figure 2), which lies 49.8 $kcal/mol$ above the reactant. Cycloprop-2-enimine can give rise to acetylene and hydrogen isocyanide (HNC) after isomerizing via the unstable intermediate Int2-IV. The energy of TS1-IV is 107.2 $kcal/mol$ with respect to VC, in comparison with the 109.6 $kcal/mol$ found in the previous *ab initio* study.[104]

Three more mechanisms that form HNC have been identified in this study. Path V proceeds through allen-1-imine (Int1-IV), an intermediate formed after H transfer between the C and N atoms of VC. The H transfer transition state TS1-V is planar and lies 105.0 $kcal/mol$ above VC. Allen-1-imine can react back to the reactant or dissociate to vinylidene and HNC via the Van der Waals minimum Int2-V.

Paths VI and VII share the first part (the isomerization of VC to vinyl isocyanide) with path III. Vinyl isocyanide (Int1-III) can dissociate via either three-centered or four-centered planar transition states giving rise to vinylidene + HNC(channel VI) and acetylene + HNC(channel VII), respectively (see Figure 19.2). The energies of these two transition states are 110.1 and 115.2 $kcal/mol$ with respect to the global minimum (VC), respectively. These two transition states resemble the corresponding three-centered and four-centered transition states from VC (TS1-I and TS1-II). The main difference is the HCN(HNC) isomer formed in each process; TS1-I and TS1-II form HCN, whereas TS1-VI and TS1-VII form HNC. The energies of the three-centered transition states (TS1-I and TS1-VI) are $5-7$ $kcal/mol$ lower than those of the corresponding four-centered transition states (TS1-II and TS1-VII). As will be seen below, the three-centered and four-centered mechanisms from VC (paths I and II) and vinyl isocyanide (paths VI and VII) are major channels for the HCN and HNC formation, respectively.

B. Kinetics calculations.

As indicated above, the population of the various species involved in paths I-VII can be modeled using combined RRKM and KMC calculations. The RRKM calculations are used for the rate coefficients and the KMC ones for the relative population of all the molecular species. In particular, we are interested in the $t \longrightarrow$ HCN and HNC relative populations for different VC internal energies. Figure 19.4 shows these populations as a function of time and reaction path for three different energies: 120, 148 and 200 $kcal/mol$. The energy of 148 $kcal/mol$ corresponds to a photon wavelength of 193 nm, and was selected to compare with the experimental results of Wilhelm et al.[105]

At the lowest energy of 120 $kcal/mol$, Path I is, by far, the most important one and HCN is formed via this mechanism with a probability of 93% (see also Table 19.1). Paths II and III contribute to the HCN abundance only 0.6%. The total HNC yield amounts 6.3%, with path V being slightly favored over the other HNC paths. The total HCN/HNC branching ratio, which reaches a constant value after 300 ns (see Figure 19.4), is 13.7 at this energy.

At 148 $kcal/mol$ (193 nm), the percentage of HCN formed via path I decreases to 55%, with respect to the 93% calculated at 120 $kcal/mol$. At this energy HCN is also formed significantly via the four-centered mechanism II (8.2%) and to a less extent via path III (2.2%). HNC is formed primarily through paths VI (17.6%) and VII (12.0%). The fact that paths VI and VII, which proceed via relatively high-energy transition states become increasingly important as the internal energy increases is due to the presence of very low vibrational frequencies for the associated transition states (TS1-VI and TS1-VII; see Table 19.2 of the Supporting Information) that makes these processes entropically more favorable. The total HCN/HNC branching ratio at 193 nm obtained in our study decreases (with respect to the value at 120 $kcal/mol$) to 1.9, supporting the recent experimental papers of Dai and co-workers[105,134] that the relative presence of HNC is more important than expected from the previous *ab initio* calculations.[104] This point will be discussed in more detail below.

At the highest energy considered in this study (200 $kcal/mol$), HCN formation via the four-centered mechanism (path II) becomes the dominant channel with a percentage of 32% vs 28% obtained for the three-centered mechanism (path I). As discussed above, the increasing importance of path II with internal energy can be understood in terms of entropic effects. As the energy increases the entropic factor becomes more important than the relative magnitude of the energy barriers (enthalpic factor). The presence of low vibrational frequencies in TS1-II makes the numerator of eqn 19.2.1 to increase more rapidly with energy for path II than for path I. Again this entropic factor explains why HNC produced via path VII is more important at the highest energy than that produced via path VI, even though the corresponding transition state for path VII is 5 $kcal/mol$ higher in energy than that for path VI.

Finally, the relative abundance of the HCN and HNC isomers produced in each path is depicted graphically in Figure 19.5 for a range of internal energies. For all energies except the lowest ones, only paths I, II, VI and VII contribute significantly to the abundance of HNC and HCN isomers. As seen in the figure, the three-centered HCN formation mechanism (Path I) dominates up to an energy of 195 $kcal/mol$. For higher energies, the four-centered mechanism (Path II) is the major one. In the whole range of energies, the HCN abundance is always greater than the HNC abundance; the HCN/HNC ratio however, shows a minimum value of 1.3 at 180 $kcal/mol$. This minimum ratio coincides with the maximum in the curve for the HNC abundance produced in channel VII. At this energy the relative importance of Paths I, II, VI and VII is: 1, 0.75, 0.62 and 0.71, respectively. These results indicate that the dissociation channels that produce HNC are major dissociation channels that compete with those that generate HCN. This results is in agreement with the most recent experimental results.[105]

C. Comparison with experiment.

The HCN/HNC branching ratio calculated by Derecskei-Kovacs and North[104] at 148 $kcal/mol$ (193 nm) was 124. As mentioned above, the previous *ab initio* results are incorrect since what they call four-centered dissociation (Fig. 4 of their paper[104] or TS1-IV in this paper) does not actually connect VC with acetylene and HNC; this path is more complex and involves the intermediate cycloprop-2-enimine, as explained above. Nevertheless, taking only paths I and IV into account, as the previous *ab initio* study predicted,[104] the HCN/HNC branching ratio, calculated with our computed paths and frequencies, would be 79. This value is lower than that obtained using by Derecskei-Kovacs and North[104] of 124 because the corresponding transition state (TS1-IV in our paper and A1 in their paper) is 2.4 $kcal/mol$ higher in energy in their calculations compared to our results and also because path IV is different in their study. In any case, paths I and IV alone are not enough to explain the experimental results of Wilhelm et al.[105] as they claim in their paper. Taking into account all the channels found in our study, the computed HCN/HNC branching ratio is 1.9 at 193 nm. This value is much smaller than 124 (or 79) and tells us about the importance of the new channels found in this study. Moreover, as detailed below, our computed paths and kinetic results are in perfect agreement with the recent experimental data of Wilhelm et al.[105]

Wilhelm et al.[105] estimated the HCN/HNC branching ratio indirectly. They measured the ratio of highly rovibrationally excited acetylene population to the total acetylene population formed as a function of time in the photodissociation of VC at 193 nm. Analyzing their time-resolved infrared spectra, they obtained a value of 0.77 for the nascent population ratio of excited/total acetylene molecules. This means that the ratio of rovibrationally excited acetylene to minimally excited acetylene is $0.77/0.23 = 3.3$. To interpret this value, they suggest that highly excited acetylene comes from vinylidene, since the vinylidene-acetylene isomerization process adds an additional 42.3 $kcal/mol$ (see Figure 19.3) of internal excitation on top of any energy originally partitioned to vinylidene. In addition, the fraction of minimally excited acetylene is formed directly in the photodissociation of VC as one of the photofragments.[105] In addition, based on the two mechanisms proposed in the *ab initio* study of Derecskei-Kovacs and North,[104] Wilhelm et al.[105] regarded the total population of vinylidene as the HCN population, since these are the two photofragments of the 3C process from VC (Path I here and see also Figure 3 of Derecskei-Kovacs and North [104]), whereas acetylene is formed directly (without a previous isomerization) alongside with HNC according to Figure 4 of Derecskei-Kovacs and North [104]. Therefore, they equated the population of highly rovibrationally excited acetylene (or vinylidene) to that of HCN and the population of minimally rovibrationally excited acetylene to that of HNC, determining a HCN/HNC ratio of 3.3.

As indicated above, the present *ab initio* calculations show that HCN may be formed either with vinylidene as coproduct (Path I) or with acetylene as coproduct (Paths II and III). Moreover, HNC can be formed as well alongside with acetylene (Paths IV and VII) or vinylidene (Paths V and VI). Assuming, as Wilhelm et al.[105] did, that rovibrationally excited acetylene comes from vinylidene and that the population of rovibrationally unexcited acetylene is a consequence of the direct formation of this molecule, one can recalculate theoretically, using our *ab initio* and kinetic modeling results, the vinylidene/acetylene ratio to compare with that measured by Wilhelm et al.[105] Using the populations of Table 19.1 at 148 $kcal/mol$, the vinylidene/acetylene ratio (calculated as the sum of the populations of Paths I, V and VI divided by the populations of Paths II, III, IV and VII) is 3.3, in perfect agreement with the experimental measured value of Wilhelm et al.[105]

Table 19.1: Final relative populations of HCN and HNC obtained in channels I-VII.

E[a]	HCN			HNC				HCN/HNC
	I	II	III	IV	V	VI	VII	
120	92.7	0.5	0.1	0.4	3.8	2.4	0.1	13.7
148	55.3	8.2	2.2	0.7	4.0	17.6	12.0	1.9
200	28.1	31.8	3.6	0.6	2.6	14.1	19.2	1.7

[a] Energy in kcal/mol.

19.4 Conclusions

The *ab initio* calculations performed in this work provide seven channels for HCN(HNC) elimination from VC, five of which are new; previous *ab initio* calculations found only two channels. Three of these new paths are extremely important to explain the experimentally observed HCN/HNC branching ratios. Among the new paths found here the major ones to generate HCN or HNC are:

1. A four-centered transition state that leads to acetylene and hydrogen cyanide (Path II).

2. Paths VI and VII that involve, as a first step, the isomerization of VC to vinyl isocyanide.

Then, from vinyl isocyanide there is a three-centered mechanism leading to vinylidene and hydrogen isocyanide (Path VI) and a four-centered mechanism leading to acetylene and hydrogen cyanide (Path VII).

Additionally, one of the paths reported in a previous *ab initio* study[104] was found to be wrong, since the transition state does not really connect the expected chemical species (VC and HCCH+ HNC). This path (IV in our study) proceeds via cycloprop-2-enimine and is more complex, as it involves more steps than previously thought. This path was found to be unimportant to calculate the HCN/HNC branching ratios. The remaining two paths are also not very important and involve isomerizations of VC to vinyl isocyanide (Path III) and allen-1-imine (Path V).

With this new *ab initio* picture of the HCN(HNC) elimination channels, the HCN/HNC branching ratios were calculated from the RRKM rate coefficients and KMC simulations. The predicted HCN/HNC branching ratio at 193 *nm* is 1.9, which differs markedly from the previous theoretical value of 124. Moreover the theoretical calculations of the present work also explain the recently measured value for the ratio of rovibrationally excited acetylene to total acetylene in the photodissociation of VC at 193 *nm*. The value obtained experimentally is 3.3, in perfect agreement with that calculated in this work using our *ab initio*/RRKM/KMC results.

19.5 Acknowledgement

The authors thank CESGA for the use of their computational devices. S.A.V. and E.M.N. thank "Xunta de Galicia" for financial support ("Axuda para a Consolidación e Estructuración de unidades de investigación competitivas do Sistema Universitario de Galicia, 2007/50, cofinanciada polo FEDER 2007-2013"). Z. H. acknowledges financial support from the Ministry of Science, Research and Technology of Iran.

Figure 19.1: Schematic potential energy diagram for paths I-III (leading to HCN formation). The values are the relative energies (in kcal/mol).

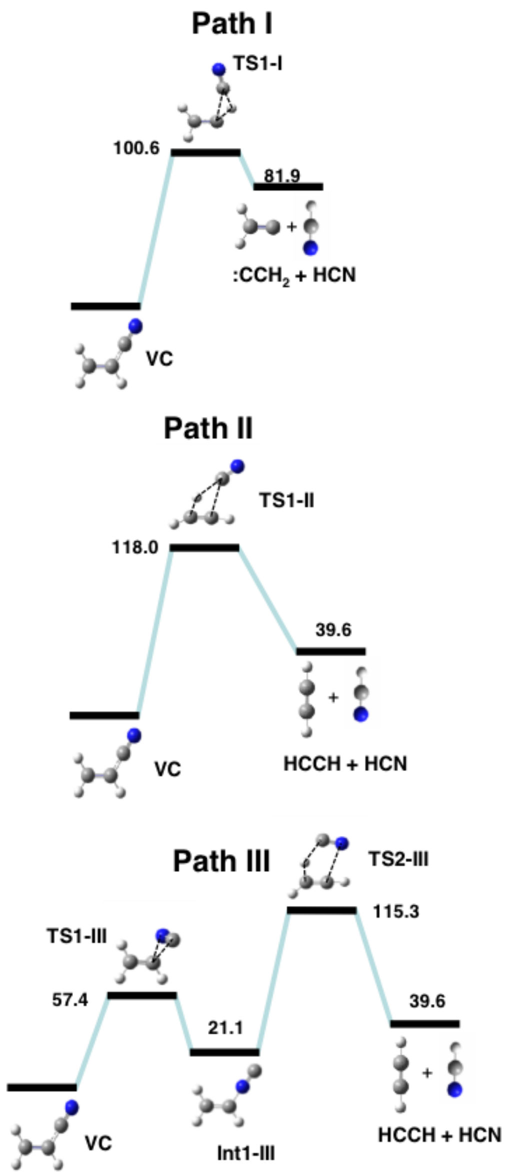


Figure 19.2: Schematic potential energy diagram for paths IV-VII (leading to HNC formation). The values are the relative energies (in kcal/mol).

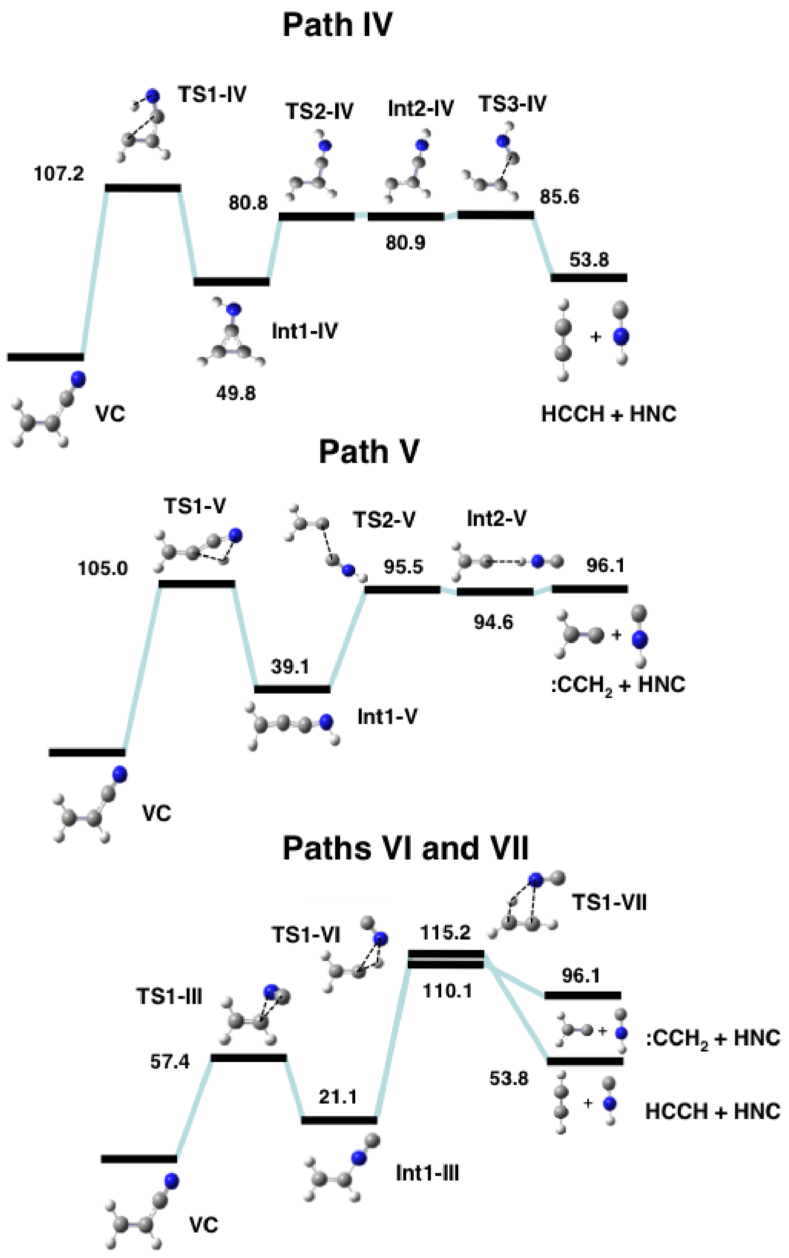


Figure 19.3: Schematic potential energy diagram for the vinylidene-acetylene isomerization process. The values are the relative energies (in kcal/mol).

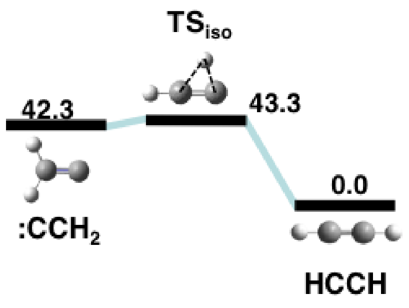


Figure 19.4: Populations, as a function of time, of HCN and HNC obtained in paths I-VII for three different energies and total HCN/HNC ratio.

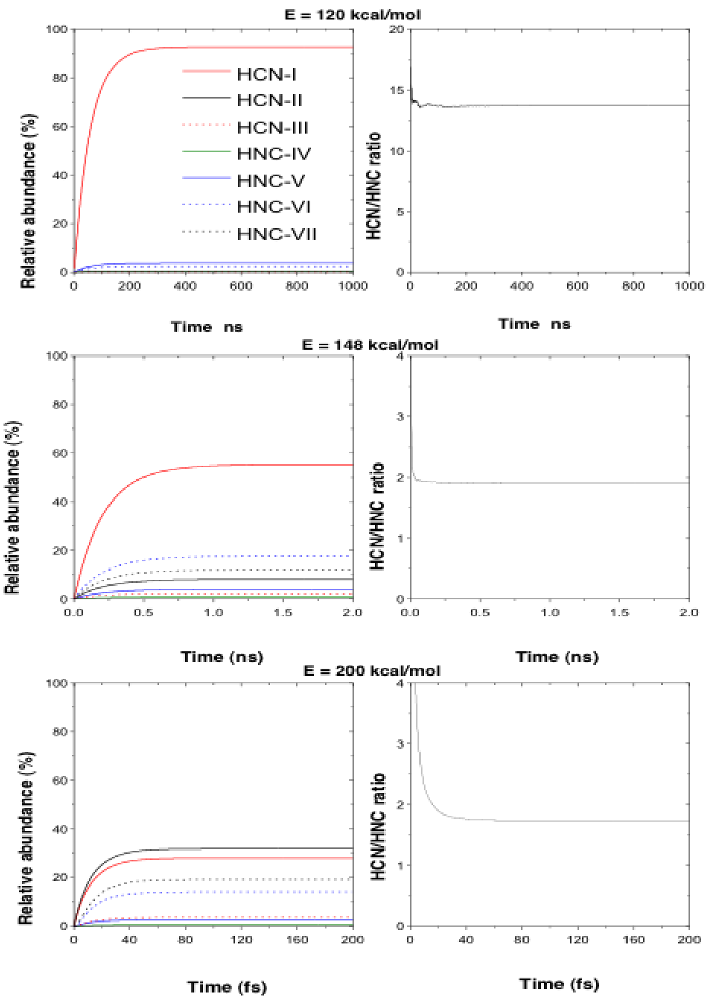
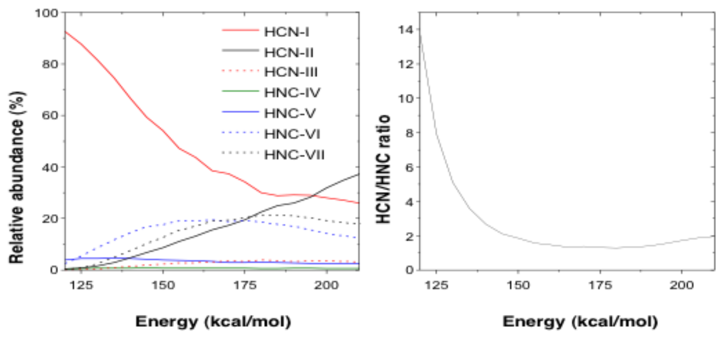


Figure 19.5: Final populations of HCN and HNC obtained for paths I-VII and total HCN/HNC ratio as a function of energy.



19.6 Supporting Information

Supporting Information Available. Vibrational frequencies and geometries of all stationary points. This material is available free of charge via the Internet at http://pubs.acs.org/.

19.6.1 Frequencies of all stationary points found in this study

Table 19.2: Frequencies of all stationary points found in this study

Species	Frequencies (cm^{-1})
VC	235; 344; 570; 696; 872; 980; 1005; 1118; 1331; 1467; 1689; 2322; 3173; 3214; 3271
TS1-I	1417i; 127; 184; 276; 521; 596; 778; 898; 953; 1353; 1615; 2187; 2272; 3153; 3271
:CCH_2	417; 765; 1253; 1680; 3141; 3235
HCN	736(2); 2158; 3451
TS2-I	976i ; 573; 915; 1836; 2552; 3377
HCCH	607(2); 747(2); 2029; 3415; 3511
TS1-II	1634i; 49; 106; 301; 384; 627; 696; 760; 790; 907; 1853; 2085; 2168; 3353; 3427
TS1-III	451i; 237; 413; 604; 685; 958; 1013; 1044; 1303; 1415; 1652; 1979; 3175; 3264; 3280
Int1-III	192; 237; 520; 707; 893; 942; 988; 1141; 1345; 1458; 1698; 2208; 3182; 3230; 3284
TS2-III	1316i; 124; 253; 338; 405; 600; 678; 891; 900; 976; 1759; 1877; 2123; 3303; 3403
TS1-IV	1436i; 411; 469; 570; 597; 753; 954; 970; 1076; 1272; 1365; 1778; 2120; 3047; 3211
Int1-IV	452; 457; 713; 844; 845; 890; 930; 1048; 1146; 1281; 1576; 1832; 3245; 3281; 3516
TS2-IV	461i; 346; 392; 503; 633; 715; 917; 989; 1088; 1278; 1444; 2113; 3050; 3124; 3718
Int2-IV	229; 344; 516; 591; 670; 739; 941; 990; 1156; 1254; 1414; 2122; 2981; 3116; 3692
TS3-IV	474i; 78; 282; 525; 630; 632; 687; 871; 882; 1156; 1708; 1837; 3320; 3364; 3625
HNC	419(2); 2091; 3842
TS1-V	2036i; 208; 255; 428; 584; 718; 881; 972; 1081; 1481; 1686; 1974; 2338; 3126; 3220
Int1-V	162; 221; 379; 589; 866; 904; 953; 1073; 1147; 1489; 1707; 2206; 3132; 3214; 3498
TS2-V	147i; 84; 115; 272; 401; 436; 474; 664; 803; 1289; 1651; 2113; 3128; 3284; 3839
Int2-V	74; 76; 148; 157; 164; 436; 743; 772; 779; 1258; 1704; 2093; 3137; 3230; 3579
TS1-VI	1263i; 111; 150; 210; 447; 466; 835; 853; 920; 1308; 1624; 2053; 2176; 3143; 3260
TS1-VII	1332i; 40; 132; 330; 395; 615; 682; 855; 877; 935; 1813; 2060; 2097; 3329; 3409

19.6.2 Cartesian Coordinates (in Å) of all stationary points found in this study

Table 19.3: :CCH2.xyz

	x	y	z
C	0.000000	0.000000	-0.481254
C	0.000000	0.000000	0.822130
H	0.000000	0.938709	-1.022626
H	0.000000	-0.938709	-1.022626

Table 19.4: HCCH.xyz

C	0.000000	0.000000	0.601899
C	0.000000	0.000000	-0.601899
H	0.000000	0.000000	1.663563
H	0.000000	0.000000	-1.663563

Table 19.5: HCN.xyz

N	0.000000	0.000000	0.652578
C	0.000000	0.000000	-0.500525
H	0.000000	0.000000	-1.564896

Table 19.6: Int1-III.xyz

H	1.728243	1.892979	0.000000
C	1.320636	0.894959	0.000000
C	0.000000	0.725214	0.000000
H	-0.701092	1.544974	0.000000
C	-1.137810	-1.574041	0.000000
N	-0.588772	-0.538465	0.000000
H	1.997293	0.054512	0.000000

Table 19.7: Int1-IV.xyz

C	-0.979481	-0.651941	-0.000072
H	1.999891	0.786895	0.000095
H	-1.587159	-1.539544	0.000111
C	-0.962037	0.681350	-0.000129
H	-1.546591	1.584752	0.000154
C	0.303023	-0.015183	0.000299
N	1.566404	-0.131066	-0.000135

Table 19.8: Int1-V.xyz

C	0.088169	1.909407	0.000000
H	-0.798096	2.531221	0.000000
H	1.047627	2.411050	0.000000
C	0.000000	0.593153	0.000000
H	0.681248	-2.437335	0.000000
C	-0.019514	-0.684988	0.000000
N	-0.191816	-1.915767	0.000000

Table 19.9: Int2-IV.xyz

C	0.752231	0.512909	0.010709
H	-2.363819	-0.456989	0.594247
H	0.979149	1.574844	0.056239
C	1.592936	-0.610264	-0.006023
H	2.632464	-0.247184	0.036993
C	-0.566987	0.155356	0.006348
N	-1.702410	-0.174096	-0.107669

Table 19.10: Int2-V.xyz

H	0.941823	-3.205479	0.000000
C	0.002627	-2.663768	0.000000
H	-0.936309	-3.205959	0.000000
C	0.002093	-1.366377	0.000000
C	-0.003832	2.924519	0.000000
N	-0.001548	1.756679	0.000000
H	0.000000	0.748448	0.000000

Table 19.11: TS1-III.xyz

C	1.425012	-0.263427	0.079283
C	0.354129	0.498225	-0.121163
C	-1.129426	0.188334	0.593364
N	-1.145832	-0.412558	-0.440426
H	2.404215	0.125561	-0.158428
H	0.372692	1.481893	-0.554674
H	1.345628	-1.258340	0.487184

Table 19.12: TS1-II.xyz

C	-1.123017	1.530547	0.000000
C	-1.251470	0.297002	0.000000
C	0.965619	-0.541190	0.000000
N	1.618519	-1.510187	0.000000
H	-1.450019	2.546888	0.000000
H	-1.426411	-0.756012	0.000000
H	0.000000	1.062279	0.000000

Table 19.13: TS1-IV.xyz

C	0.098171	-1.392816	0.000000
H	-1.270267	-0.014155	0.000000
H	0.644486	-2.342959	0.000000
C	0.951278	-0.306501	0.000000
H	2.022457	-0.173053	0.000000
C	0.000000	0.737394	0.000000
N	-1.099053	1.185958	0.000000

Table 19.14: TS1-I.xyz

C	-0.044176	-1.730040	0.000000
H	-1.094048	-1.490714	0.000000
H	0.256536	-2.772055	0.000000
C	0.957042	-0.884377	0.000000
H	1.162350	0.306873	0.000000
C	0.000000	1.073269	0.000000
N	-0.828862	1.886112	0.000000

Table 19.15: TS1-VI.xyz

C	1.293549	1.121106	0.000000
C	0.000000	1.074190	0.000000
N	-1.165554	-0.833953	0.000000
C	-0.364829	-1.706840	0.000000
H	1.720937	2.119168	0.000000
H	-1.073535	0.530804	0.000000
H	1.939154	0.256966	0.000000

Table 19.16: TS1-V.xyz

C	1.791655	0.127867	0.000020
H	1.992513	1.193394	0.000108
H	2.647592	-0.532279	0.000039
C	0.555741	-0.367625	-0.000073
H	-0.970507	-1.029273	0.000255
C	-0.716863	0.187548	-0.000026
N	-1.921829	0.097346	0.000011

Table 19.17: TS2-III.xyz

H	-1.916255	1.816458	0.000000
C	-1.185312	1.030689	0.000000
C	-1.149921	-0.223557	0.000000
H	-1.076098	-1.289589	0.000000
C	1.394753	0.124243	0.000000
N	1.233604	-1.039622	0.000000
H	0.000000	1.162233	0.000000

Table 19.18: TS2-IV.xyz

C	0.819819	0.591947	-0.009475
H	-2.180138	-0.827002	0.384548
H	1.153138	1.611530	0.158940
C	1.368336	-0.662794	-0.042733
H	2.443028	-0.710931	0.161078
C	-0.493515	0.170351	0.001644
N	-1.654838	-0.095803	-0.057311

Table 19.19: TSiso.xyz

C	0.078325	0.519890	0.000000
C	0.078325	-0.732774	0.000000
H	-1.041882	-0.311646	0.000000
H	0.101977	1.588950	0.000000

Table 19.20: TS2-V.xyz

H	1.786035	-1.336957	0.000000
C	0.835235	-1.839086	0.000000
H	0.801039	-2.926625	0.000000
C	-0.371884	-1.319233	0.000000
C	0.000000	1.028607	0.000000
N	-0.599762	2.028314	0.000000
H	-1.168847	2.843656	0.000000

Table 19.21: TS3-IV.xyz

C	-1.008648	-0.552746	-0.000053
H	2.489365	0.123182	0.000350
H	-1.393511	-1.550668	-0.000283
C	-1.447043	0.635924	0.000000
H	-1.114538	1.652427	0.000339
C	0.698849	-0.543933	0.000218
N	1.508534	0.362797	-0.000201

Table 19.22: VC.xyz

C	-0.583152	-0.536271	0.000000
N	-1.068238	-1.586302	0.000000
C	0.000000	0.785121	0.000000
H	-0.694962	1.611652	0.000000
C	1.321786	0.978464	0.000000
H	2.013999	0.149805	0.000000
H	1.726828	1.978777	0.000000

Part V

Conclusions

20 Conclusions

There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.

Sir Charles Antony Richard Hoare

The main conclusions of this Thesis are enumerated as follows:

1 Core GA routines were isolated and GA was modified to deal with integer parameters in the functions employed to fit intermolecular interactions.

2 A flexible interface was designed with a well known logic to help adding new features sharing code between distinct package utilities to maintain the oneness of the whole. In deep coding was done to increase portability between different systems.

3 Robust routines to read input files were coded. Over these reading routines, a configuration system was created. This system supports configuration sections featuring key/value pairs ready to expand new capabilities with no effort.

4 Specifying a new intermolecular potential energy function can be done modifying clear and explained source code or using a file template to fill up with user code.

5 To use an analytical expression, a complete virtual FPU was developed with its own machine instruction set, a compiler to translate the expression to binary executable code or to its assembler representation. Also a new utility was developed to test analytical expressions compiling and running the resulting executable.

6 To interface with external programs a communication protocol was developed, so with a minimum effort, the external programs can configure **GAFit** behavior. This characteristic was used to build the two MOPAC interfaces but it is general to deal with any external program.

7 A first MOPAC interface was built coding a set of tools and gluing them all together in a shell script, the **external-mopac2009.sh**: The **injector**, in C, to configure **GAFit** and create MOPAC input files, the MOPAC binary executable to run each job, the **extractor**, in Perl, to process the output files and write results to a file, *extracted.data*, with a well known file format, and the **fitter**, in Fortran, to evaluate the results upon the user requirements. If the variables to control and fit are the same as in MOPAC, it only must be changed the **injector** and **extractor** tools to use a new external program.

In the case of MOPAC 2012, where some output in Cartesian coordinates are missing, the **extractor** utility also translates internal coordinates to Cartesian coordinates using *quaternion maths* to calculate 3D rotations.

8 An enhanced MOPAC interface was coded replacing the MOPAC executable with a new tool: **shepherd**, which can launch and control running MOPAC jobs maintaining an optimal number of parallel MOPAC processes, depending on available resources, to speed up calculations.

9 A perl tool, **needle**, was developed to identify types of atoms, which are needed to calculate the different types of interactions between two fragments and automatically build the *atom2type* file from Cartesian coordinates.

10 The **bedit** tool, source code in C and Fortran, was designed to help modifying atom types, charges and bounds in the corresponding input files.

11 A GUI program, **JobTreeEditor**, was developed in Java using the swing widget toolkit to create and modify the job configuration file *job.txt*.

12 **fitview**, written in C and Fortran, an utility to write and plot data from results generating **gnuplot** files was developed.

13 A complete set of case examples is included with code and explained.

14 The package was automated with the *GNU build* system[1]. It can be deployed, compiled, installed and run in many systems, including MacOSX.

[1]Also known as the **autotools**: http://www.gnu.org/software/automake/manual/html_node/Autotools-Introduction.html

Appendices

Source code

A

A.1 Source files

Source files are listed in the table A.1. All files are related to each other. Same functions and subroutines are called from any compiled executables. So, a behaviour change in one means a change in the others.

Table A.1: Source files

File/Directory	Description	Comments
analytical	interface between potential stuff and analytical expressions subsystem	it has dependencies on *nullist*, *pack*, *fpu*, *compiler* and *bytecodes*
arguments.c	stuff to add TAGS	
arguments.h	arguments header	
autoweights.c	stuff to use automatic weights	
autoweights.h	autoweights header	
bedit.c	bounds and types editor	
bounds.c	stuff to read bounds	
bounds.h	bounds header	
bytecodes	defines bytecodes for fpu	
cnames.c	coefficient names stuff	
cnames.h	cnames header	
crossover.c	crossover code	
crossover.h	crossover header	
compiler	compiles expressions into bytecode	
eval.f	fortran entry point	
evaluation.c	evaluation	
evaluation.h	evaluation header	
final.c	prints results	
final.h	final header	

File/Directory	Description	Comments
finput.c	read variables and setup system	
fitview.c	plots data	
flyctl	rutines to stop running jobs	
fpu	virtual FPU	
ga.c	main program	
ga.h	ga header	
global.h	C common variables	
InputLine	subroutines to read files from C	it heavily depends on the **libc** function *getdelim*
integer.c	helper functions to integer coefficients	
integer.h	integer header	
interface.f	glue to link all together	
interface.h	interface header	
job.txt	job configuration	modify as per job basis
JobTreeEditor	job configuration gui	java: it runs in linux, mac, windows...
literals	subroutines to support automatic coefficient names	
mopac	MOPAC interface stuff	
mutation.c	mutation code	
mutation.h	mutation header	
needle	analise system structure	use it to generate *atom2type* and *charges* files
nullist	implements null-terminated list	
pack	code and decode bytecode fpu programs	
parameters	parameters and settings code	
potentials.f	potentials stuff	modify to introduce new potentials
rand.c	random stuff code	
rand.h	random header	
rstrings	strings generic functions	
selection.c	selection code	
selection.h	selection header	
stats.c	stats stuff, prints intermediate results	
stats.h	stats header	
ufpu.c	ufpu code	
userpotential.f	user potential fortran template	modify to introduce a fully custom potential
utils.c	helper functions	
utils.h	utils header	

A.2 Analytical job

This code deal with expressing potentials as analytical expressions. It depends on A.4. C language.

- analytical.h

- analytical.c

A.3 Potential base routines

Potential base routines like the implemented internal and user-coded potentials. C and Fortran languages.

- eval.f
- final.h
- final.c
- finput.c
- global.h
- interface.h
- interface.f
- potentials.f
- userpotential.f

A.4 Fpu routines

This code implements a virtual calculator: it compiles analytical expressions to packed chunks of bytecode, and run the bytecode in a virtual FPU. C language.

- bytecodes.h
- bytecodes.c
- nllist.h
- nllist.c
- pack.h
- pack.c
- ucompiler.h
- ucompiler.c

A.5 GAFit

Entry routines and main loop. C language. It depends on A.2. A.3, A.4, A.7, A.7.2, A.6 and A.8. See section 4 and Figure 4.1.

- ga.h
- ga.c

A.6 Genetic Algorithm Core

GA routines. C language.

A.6.1 Crossover

- crossover.h
- crossover.c

A.6.2 Mutation

- mutation.h
- mutation.c

A.6.3 Selection

- selection.h
- selection.c

A.6.4 Stats

- stats.h
- stats.c

A.6.5 Utils

- utils.h
- utils.c

A.7 External job

Here is implemented the interface with external programs. C, Fortran and Perl languages.

A.7.1 Flyctl

This code addresses the external job stopping problem.

- flyctl.h
- flyctl.c

A.7.2 Mopac

Interface with MOPAC.

- extractor
- fitter.f
- injector.c
- mopac.h
- mopac.c
- shepherd.c
- lstimes.c
- lsexdata.f

A.8 Miscellaneous

A.8.1 Arguments

Program arguments stuff. C language.

- arguments.h
- arguments.c

A.8.2 Autoweights

Code to implement the auto-weights feature. C language.

- autoweights.h
- autoweights.c

A.8.3 Bounds

Custom routines to read bounds files. C language.

- bounds.h
- bounds.c

A.8.4 Cnames

Coefficient names stuff. C language.

- cnames.h
- cnames.c

A.8.5 InputLine

Custom routines to read lines and text from configuration and data files. C language.

- line.h
- line.c

A.8.6 Integer

Code to support integer coefficients. C language.

- integer.h
- integer.c

A.8.7 Literals

Routines to support automatic coefficient names. C language.

- literals.h
- literals.c

A.8.8 Parameters

Code to deal with program parameters. C language.

- parameters.h
- paramenters.c

A.8.9 Rand

Random stuff. C language.

- rand.h
- rand.c

A.8.10 Rstrings

C strings custom routines. C language.

- rstrings.h
- rstrings.c

A.9 Tools

C, Java and Perl languages.

A.9.1 Bedit

Terminal utiltiy to edit *atom2type*, *bounds* and *charges* files.

- bedit.c

A.9.2 Fitview

Tool to create some **gnuplot** plots.

- fitview.c

A.9.3 JobTreeEditor

Java graphical interface to create and edit the configuration file *job.txt*.

- DoubleString.java
- EditorComboBox.java
- EditorRoot.java
- EditorSectionTextField.java
- EditorTextArea.java
- EditorTextField.java
- FilterTxt.java
- FIve.java
- IniData.java
- IniEditor.java
- IniModel.java
- IniRenderer.java
- JobTreeEditor.java
- JobTxtConf.java
- KnownValues.java
- RoFileFilters.java

A.9.4 Needle

Perl tool to create the *atom2type* file from a geometry file.

- needle

A.9.5 Ufpu

Utility to test analytical expressions as potentials.

- ufpu.c

B Resumo

Un dos conceptos chaves na química é o da superficie de enerxía potencial, –Potential Energy Surface (PES), en inglés– a cal vén da aproximación de Born-Oppenheimer que facilita a solución da ecuación de Schrödinger independente do tempo para sistemas moleculares, e afortunadamente, os erros asociados con esta aproximación son depreciábeis para a maioría dos sistemas e condicións de interese para os químicos. A PES dun sistema molecular goberna a maioría das súas propiedades químicas, e particularmente a dinámica, é dicir, a evolución espacial dos núcleos co tempo. Moitas das simulacións dinámicas realizadas hoxe en día envolven a integración das ecuacións clásicas do movemento, calculando as forzas sobre os átomos en cada paso, directamente mediante cálculos da estrutura electrónica –*dinámica directa*, cálculos *"ao voo"*– ou por PES *analíticas*.

Nun principio, o enfoque da *dinámica directa* pode ser a opción preferida para a simulación de sistemas reactivos que inclúen un pequeno número de átomos, porque se evita a construción da *superficie analítica*. Sen embargo, o uso das PES *analíticas* ten unha clara vantaxe en termos de custos en tempos de CPU –Central Processing Unit (CPU), unidade de proceso central, sinónimo do microprocesador ou procesador dos modernos computadores–, sendo obrigada en simulacións dinámicas de sistemas compostos por miles de átomos. Incluso para sistemas de pequeno tamaño, o uso dunha *superficie analítica* —tamén xeralmente chamada *campo de forzas* en *mecánica molecular* e *dinámica molecular*–

pode ser unha boa elección. Se se fai con coidado, pode ser, como mínimo, tan boa como a superficie exacta correspondente ao método cálculo da estrutura electrónica usado como referencia para a súa construción.

O desenvolvemento das PES *analíticas* ou *campos de forza* pódese facilitar usando métodos de optimización, e varios grupos de investigación xa as usaron para diversos obxectivos do seu interese. Sen embargo, ata o que sabemos, non hai un programa xeral que permita aos usuarios parametrizar *superficies analíticas* ou *campos de forza* dunha forma relativamente fácil. O motivo do presente traballo é escribir unha *suite* de ferramentas que axuden aos usuarios a desenvolver *superficies analíticas*. Esta *suite* de programas chámase **GAFit**. Usamos este nome porque neste *kit* de ferramentas un GA –Algoritmo xenético, Genetic Algorithm (GA)– conduce o axuste —*fit*—- ou parametrización da superficie de enerxía potencial desexada. O algoritmo xenético non foi desenvolvido neste traballo, senón que foi recollido da literatura: [F V; Pereira F B; Almeida M M; Maniero A M; Fellows C E Marques J M C; Prudente. “A new genetic algorithm to be used in the direct fit of potential energy curves to ab initio and spectroscopic data”. In: *Journal of Physics B: Atomic, Molecular and Optical Physics* 41.8 (2008), p. 085103. URL: http://stacks.iop.org/0953-4075/41/i=8/a=085103] e [Marcos M Almeida et al. “Direct fit of spectroscopic data of diatomic molecules by using genetic algorithms: II. The ground state of RbCs”. In: *Journal of Physics B: Atomic, Molecular and Optical Physics* 44.22 (2011), p. 225102].

Para os nosos obxectivos, a vantaxe dun algoritmo xenético fronte a outro tipo de método de optimización é dado polo tipo de problema a resolver. Por unha banda, é necesario un algoritmo que realice unha optimización global atopando unha posible resposta dentro dun tempo razoable. Por outra banda, outra característica desexable é non ter que adaptar o algoritmo ao tipo de problema. Neste caso, os algoritmos xenéticos posúen as dúas: exploran todo o posible espazo de solucións co engadido de que non fan ningún tipo de asuncións *a priori* sobre a tarefa a realizar, e polo tanto traballan ben con calquera tipo de problema. Pódese comparar, en contraposición, cun algoritmo tipo, dos usados normalmente para as optimizacións, o cal necesitaría de información inicial: a semente ou o punto de comezo, a partires do cal empeza a optimización. Dependendo da semente, é obvio que posiblemente atopará un mínimo próximo, que non ten por que ser o mínimo global. E polo tanto, é necesario ter un coñecemento *a priori* do sistema a axustar.

Neste traballo, o programa **GAFit** aplícase ao desenvolvemento de un potencial intermolecular para a interacción entre o **Xe** e o complexo $[\mathrm{Li(Uracil)}]^+$, e para a reparametrización de un Hamiltoniano semiempírico. Os Hamiltonianos semiempíricos modificados con parámetros de reacción específicos –Specific Reaction Parameters (SRP)– foron propostos por primeira vez por [Angels Gonzalez-Lafont, Thanh N Truong, and Donald G Truhlar. "Direct dynamics calculations with NDDO (neglect of diatomic differential overlap) molecular orbital theory with specific reaction parameters". In: *The Journal of Physical Chemistry* 95.12 (1991), pp. 4618–4627] como un método práctico de cálculo de *dinámicas directas*. O programa, sen embargo, pode ser facilmente adaptado para conducir calquera tipo de axuste ou parametrización de superficies analíticas ou campos de forza, así como tamén, para outros problemas de optimización na química.

No **GAFit** a funcionalidade do núcleo do GA foi estendida en varias direccións: illando o núcleo do resto do código, modificando o mesmo para permitir o uso de valores enteiros nos coeficientes a parametrizar das funcións de axuste para as interaccións intermoleculares, engadindo utilidade e usabilidade coas novas características e úteis de axuda, e por último, unha codificación coidadosa para permitir a portabilidade entre distintos sistemas operativos.

No caso do illamento do *core*, isto facilitou o deseño dunha interface flexible e ben estruturada que permite o engadido de características novas de forma sinxela compartindo código entre as distintas partes, incluíndo os programas illados de utilidade co fin de que as distintas partes se interrelacionan facendo o todo unha única unidade fácil de manter.

As características engadidas permiten agora aos usuarios escoller, dependendo das súas habilidades de programación, desde codificar directamente os seus potenciais no propio código fonte do **GAFit** ata, para eses usuarios que non teñen ningún coñecemento, poder usar os potenciais máis comúns na literatura xa codificados no propio programa. Outra opción para os que non teñen coñecementos de programación é o uso de expresións analíticas que son compiladas ao código máquina dunha FPU virtual que o interpreta como o potencial a parametrizar.

Para usuarios intermedios que queiran codificar un potencial axustado ás súas necesidades, pero que teñan medo de modificar o código fonte do **GAFit**, inclúese un modelo en Fortran a cubrir polo usuario co seu potencial e a súa función de axuste. En ámbolos dous casos, o código fonte do propio **GAFit** ou o modelo, están claramente codificados e comentados para ir conducindo ao usuario na dirección axeitada.

Ademais, engádese, como complemento para facilitar o traballo, unha completa colección de ferramentas para a creación e configuración dos arquivos de entrada e examinar os resultados: **needle**, **bedit**, **JobTreeEditor**, **fitview** e **ufpu**.

Para o uso de expresións analíticas como potenciais, desenvolveuse unha Floating Point Unit (FPU) virtual co seu propio conxunto de instrucións máquina, un compilador para traducir as expresións analíticas a un binario executable pola propia FPU –ou a súa representación na linguaxe ensambladora da FPU para facilitar a detección de erros–. Co fin de testar as expresións analíticas, a compilación e execución das mesmas, construíuse unha nova utilidade: **ufpu**, que usa directamente o arquivo de configuración do traballo de cálculo para realizar os tests.

Codificáronse rutinas robustas para a lectura de arquivos de entrada. Sobre estas rutinas creouse un sistema de configuración que soporta a división lóxica das distintas partes a configurar en seccións, podendo introducir os parámetros individuais de cada sección mediante un método sinxelo de pares chave/valor. Este sistema de división lóxica e o feito de usar pares chave/valor facilitan o engadido de novas características sen moito traballo. Estas facilidades permitirían ao propio usuario parametrizar os seus propios módulos usando o mesmo arquivo de configuración xenérico do programa ou outro calquera ao seu gusto.

Usando a nova interface co *core*, deseñouse como nova característica unha interface de segundo nivel para interactuar con programas externos e usar os seus resultados como axuste dos parámetros obtidos polo algoritmo xenético. O protocolo establecido permite que calquera utilidade externa configure o comportamento do propio **GAFit** de forma automática.

Usando este interface, desenvolvéronse as ferramentas necesarias para parametrizar algún dos Hamiltonianos semiempíricos implementados no programa de cálculo de estrutura electrónica MOPAC co **GAFit** facendo dúas interfaces. A primeira sería o exemplo a seguir polo usuario para interactuar cun programa externo diferente ao MOPAC. Sen embargo, atopáronse certas deficiencias inherentes ao MOPAC, co cal fíxose de seguido unha interface máis sofisticada para solventar estas eivas.

A primeira foi feita codificando un conxunto de ferramentas e xuntándoas nun *shell script*, o **external-mopac2009.sh**: O **injector**, escrito en C, para configurar o **GAFit** e crear traballos MOPAC, o executable binario do MOPAC para correr cada traballo, o **extractor**, feito en Perl, para procesar os arquivos de saída, e o **fitter**, en Fortran, para avaliar os resultados dependendo dos requirimentos do usuario. Se se usa un novo programa externo que use as mesmas variables co MOPAC so é necesario reescribir o **injector** e o **extractor** para adaptar o sistema ao novo programa.

O **injector** usando os conxuntos de coeficientes pasados polo **GAFit** crea os arquivos necesarios para que sirvan de entrada de datos ao MOPAC. Tamén é responsable de configurar ao **GAFit** se se usa a configuración automática –*autoconfigure*–.

O **extractor** le os arquivos de saída e recolle toda a información interesante gravándoa nun arquivo intermedio cunha estrutura ben coñecida. O **fitter** usa este arquivo para coller a información necesaria tendo en conta o especificado polo usuario nun arquivo de texto no que especifica que variables hai que ter en conta para o axuste e que peso van ter. No caso do MOPAC 2012, onde parte dos resultados en coordenadas cartesianas xa non se escriben no arquivo de saída, o **extractor** traduce as coordenadas internas a cartesianas usando *operacións con quaternions* para calcular as rotacións 3D necesarias para esta tarefa.

O **fitter** recolle os requirimentos do usuario para os axustes e usa os datos recopilados polo **extractor** para calcular o resultado do axuste. Entre as distintas variables que se poden usar están as calores de formación ou a súa diferenza entre distintos cálculos –entre un produto e un estado de transición, por exemplo–, distancias entre átomos, ángulos entre tres átomos e ángulos diedros entre catro átomos. En todos os casos –distancias, ángulos e ángulos diedros– non teñen que estar unidos por enlaces para podelos usar como variables de axuste. Outras variables típicas neste tipo de axustes son as frecuencias de vibración tanto dos mínimos como dos estados de transición que poidan presentar o noso sistema obxecto de estudo. Comunmente os pesos que teñen estas variables son totalmente diferentes.

As deficiencias detectadas son de dous tipos: Por unha banda, algúns dos traballos lanzados quedaban colgados por mor de parámetros non axeitados, sendo necesario matar os procesos á man. Por outra banda, os traballos fallados inflúen nos seguintes gravados dentro do mesmo ficheiro estragándoos. Iso fixo que fora necesario crear unha interface mellorada, da que se falou arriba, que foi codificada cambiando o MOPAC executable no *shell script* cunha nova ferramenta: **shepherd**.

O **shepherd** pode lanzar e controlar traballos MOPAC, incluíndo terminar os procesos colgados automaticamente, mantendo un número óptimo de procesos MOPAC paralelos, tendo en conta os recursos dispoñibles, e polo tanto acelerando os cálculos. Este sistema foi pensado tendo en conta o uso de sistemas de execución en *batch* como o Portable Batch System (PBS) de amplo uso en *clusters* de computación. **shepherd** ten un rendemento óptimo sempre e cando o espazo para os arquivos do cálculo, incluídos os temporais, estean en dispositivos de almacenamento locais ao nodo onde corre o cálculo, xa que, ao basearse o algoritmo de cálculo do número de procesos concorrentes óptimo nos tempos de execución das tarefas, sistemas de arquivos en rede, como o NFS, distorsionan o resultado pola latencia debida a transferencia de datos a través da rede.

Co motivo de calcular os diferentes tipos de interaccións entre os fragmentos e automaticamente construír o arquivo **atom2type**, que correlaciona os átomos individuais co seu tipo, a partir das coordenadas cartesianas desenvolveuse unha ferramenta, **needle**, en Perl para identificar os tipos diferentes de átomos equivalentes. Esta ferramenta, constrúe os enlaces tendo en conta as distancias entre átomos e as valencias. Posteriormente, desenreda secuencias lineais de átomos conectados onde as cadeas máis longas, compáranse para determinar os átomos equivalentes a partir dos seus veciños. Esta configuración escríbese no arquivo **atom2type** que permite ao sistema recoñecer e aplicar potenciais iguais a interaccións atómicas iguais. É dicir, empregar os mesmos coeficientes do potencial para interaccións entre átomos que son equivalentes dous a dous.

Para axudar a editar os arquivos de entrada, escribiuse a utilidade **bedit**, permitindo modificar o arquivo *atom2type*, as cargas e os límites dos coeficientes nos correspondentes arquivos de entrada. O código fonte é en C e Fortran.

JobTreeEditor é un programa con interface gráfica, Graphical User Interface (GUI), feito en Java e usando a biblioteca gráfica para Java **swing** que inclúe *widgets* gráficos para crear e modificar o arquivo de configuración *job.txt* dunha forma visual e sinxela empregando como soporte da información unha arbore onde as pólas son as distintas seccións e as follas os pares chave/valor.

fitview, escrito en C e Fortran, é unha utilidade que escribe un par de arquivos por cada gráfica: un arquivo **gnuplot** coas ordes para representar graficamente os resultados e outro de datos Estes pódense editar para permitir escribir arquivos gráficos directamente: PNG –portable network graphics (PNG), gráfico de rede portábel– ou PDF –portable document format (PDF), formato de documento portábel–, por mencionar algúns dos máis coñecidos. Os gráficos xerados inclúen todas as interaccións entre pares de átomos, todas as interaccións nun só gráfico e unha avaliación do axuste incluíndo todas as xeometrías usadas para a parametrización. Esta utilidade é de grande axuda para os usuarios no caso de que estean interesados no desenvolvemento de potenciais intermoleculares xa que é importante comprobar se existe un equilibrio entre os distintos tipos de interacción, é dicir, que estean compensadas. Ademais no caso de que o usuario faga uso de funcións baseadas en potenciais con termos exponenciais para representar as interaccións de corto alcance, é importante para evitar a denominada *catástrofe de Buckingham*.

ufpu, codificado en C, é unha ferramenta para testar as expresións analíticas. Usa o propio arquivo de configuración do **GAFit** para ler a expresión analítica configurada, compilala a código máquina, e cargar e executar o resultado da compilación nunha instancia da FPU virtual. A compilación xera dous arquivos: o código binario e o correspondente en ensamblador da FPU virtual. Se hai variables na expresión, pide ao usuario que entre os valores necesarios para realizar o cálculo.

O paquete de software foi automatizado co *GNU build system*, tamén coñecido como *autotools*. Pode ser implantado, compilado, instalado e executado en multitude de sistemas, incluíndo ao Mac OS X, así como distintas distribucións de Linux e sistemas tipo Unix.

Co código fonte, inclúese un conxunto completo de casos de exemplo cos seus arquivos de datos e código correspondentes, que son explicados paso á paso. No primeiro, úsase o sistema $\mathbf{Xe} + \mathbf{[Li(Uracil)]^{+}}$ para ilustrar o uso máis sinxelo: a parametrización dun potencial dos máis usados e xa incluído no programa. Cos arquivos de datos empréganse as distintas ferramentas da *suite* para ir construíndo os arquivos de entrada, e finalmente executar o **GAFit**. Amósanse tamén os arquivos de saída e a súa interpretación.

O segundo exemplo enfoca o uso dunha expresión analítica cos mesmos arquivos de entrada que a sección anterior. A expresión analítica vai ser introducida no arquivo de configuración usando unha ferramenta visual como o **JobTreeEditor** e testada co **ufpu**.

No terceiro exemplo, faise un axuste dun polinomio de grado N para ilustrar o interface con programas externos. Proporciónase o código en C do programa encargado de calcular o valor do polinomio segundo os valores dos coeficientes proporcionados por **GAFit** e discútese polo miúdo a mecánica da interface.

No cuarto, reparametrízase un Hamiltoniano semiempírico para axustar as enerxías, xeometrías e frecuencias para a descomposición dun canle do cianuro de vinilo –ou acronitilo, vinyl cyanide (VC)–, traído da aplicación do estudio *ab initio* e RRKM –Rice–Ramsperger–Kassel–Marcus (RRKM)– dos canles de eliminación de HCN/HNC do VC, para ensinar paso a paso a aplicación do interface externo co MOPAC e a interconexión entre as distintas ferramentas que compoñen o interface: **injector**, **extractor** e **fitter**.

No derradeiro exemplo exponse o interface mellorado usando o cuarto exemplo como base e destacando as diferenzas, móstrase o uso da última utilidade do interface co MOPAC: **shepherd**. No código fonte inclúese outro exemplo baseado no VC, tamén traído da mesma aplicación, onde se fai fincapé noutras condicións de axuste onde inclúe moitas mais variables na parametrización da PES, sendo polo demais, igual co anterior.

Na primeira aplicación estúdase a disociación inducida por colisión –Collision-Induced Dissociation (CID)– do complexo $[\mathrm{Li(uracil)}]^+ + \mathrm{Xe}$. A dinámica faise *"ao voo"*, tomando a enerxía e os gradientes de cálculos semiempíricos Austin Model 1 (AM1) complementados con potenciais analíticos de dous corpos para modelar as interaccións intermoleculares. Para a parametrización do potencial intermolecular, tomáronse como valores de referencia enerxías de interacción obtidas co nivel de cálculo CCSD(T)/CBS para un total de 13 orientacións distintas entre os fragmentos. No axuste co **GAFit** impuxéronse algunhas restricións aos valores posibles dos parámetros para evitar a *catástrofe de Buckingham*. O axuste final obtido foi moi bo con valores do erro cuadrático medio de 0,2 kcal/mol e 2,6 kcal/mol para enerxías de interacción no rango -6 – 2 kcal/mol e 2 – 100 kcal/mol, respectivamente.

Na segunda aplicación, lévanse a cabo cálculos *ab initio* para modelar a PES do estado fundamental das canles da eliminación do HCN e HNC desde o VC. Os cálculos comprenden optimizacións CCSD/6-311+G(2d,2p) e análises de frecuencias para caracterizar os puntos estacionarios como mínimos ou puntos cadeira e para avaliar as enerxías vibracionais do ZPE –punto cero, zero-point vibrational energies (ZPE)–. Os resultados obtidos neste traballo, serven para a parametrización dun Hamiltoniano semiempírico, parte dos cales empréganse nesta tese para testar o interface co MOPAC 2009 e 2012, e tamén como exemplos cara ao usuario final. A idea é agora complementar estes traballos previos seleccionado o semiempírico máis axeitado para facer o axuste global da PES do VC co gallo de levar a cabo cálculos de traxectorias clásicas intensivas ás enerxías empregadas nos experimentos de fotodisociación.

Dada a complexidade da tarefa a desenvolver, e das facilidades que presentan as diferentes linguaxes de programación para resolver certo tipo de problemas, a codificación das utilidades fíxose usando a linguaxe máis axeitada en cada caso.

Por exemplo, o Fortran empregouse naquelas partes onde era necesario facer cálculos en coma flotante, dada a súa facilidade de programación para esta tarefa, e ademais a meirande parte dos posibles usuarios teñen coñecementos en maior ou menor medida desta linguaxe de programación. Por esta razón, as partes a modificar polos usuarios están codificadas en Fortran.

O Perl usouse naquelas partes onde se examinaron arquivos de texto e extraéronse datos necesarios para guiar o proceso de parametrización, dada a súa tremenda capacidade para este tipo de tarefas. Non é necesario subliñar de que unha das linguaxes cunha implementación máis completa e potente das *regex –regular expressions*, expresións regulares— usadas para a busca e extracción de texto, aparte do Tcl, é precisamente Perl.

A linguaxe C usouse como o esqueleto do **GAFit**, de todas as subrutinas de compilación e execución de código da máquina virtual, do sistema de procesado dos arquivos de configuración, parte dos úteis da interface externa e o propio core.

Java e as súas librerías gráficas xunto cunha boa IDE –integrated development environment (IDE), contorno de desenvolvemento integrado– de programación como a *netbeans* facilitan a construción de programas GUI.

O *shell script* sirve para executar os procesos máis complexos que requiren de varias utilidades da suite á vez.

E por fin, as *autotools*, que inclúen linguaxes esotéricos como o *m4*, facilitan a tarefa de portar o paquete a outros contornos.

References

[1] Ira N. Levine. *Quantum Chemistry*. 7th ed. Pearson Education Inc, Boston, Feb. 2014, p. 720.

[2] F V; Pereira F B; Almeida M M; Maniero A M; Fellows C E Marques J M C; Prudente. "A new genetic algorithm to be used in the direct fit of potential energy curves to ab initio and spectroscopic data". In: *Journal of Physics B: Atomic, Molecular and Optical Physics* 41.8 (2008), p. 085103. URL: http://stacks.iop.org/0953-4075/41/i=8/a=085103.

[3] Marcos M Almeida et al. "Direct fit of spectroscopic data of diatomic molecules by using genetic algorithms: II. The ground state of RbCs". In: *Journal of Physics B: Atomic, Molecular and Optical Physics* 44.22 (2011), p. 225102.

[4] Angels Gonzalez-Lafont, Thanh N Truong, and Donald G Truhlar. "Direct dynamics calculations with NDDO (neglect of diatomic differential overlap) molecular orbital theory with specific reaction parameters". In: *The Journal of Physical Chemistry* 95.12 (1991), pp. 4618–4627.

[5] A. Holder, ed. *Mathematical Programming Glossary*. Originally authored by Harvey J. Greenberg, 1999-2006. Accessed 1 April 2014. http://glossary.computing.society.informs.org: INFORMS Computing Society, 2006–08.

[6] Bruce A. McCarl and Thomas H. Spreen. *Applied Mathematical Programming using Algebraic Systems*. Accessed 1 April 2014. Texas A&M University, 2011. URL: http://agecon2.tamu.edu/people/faculty/mccarl-bruce/mccspr/thebook.pdf.

[7] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Accessed 1 April 2014. Cambridge university press, 2004. ISBN: ISBN 9780521833783. URL: http://www.stanford.edu/~boyd/cvxbook/bv%5C_cvxbook.pdf.

[8] MS Bazaraa and John J Jarvis. *Linear programming and network flows*. Wiley (New York), 1977.

[9] Arnold Neumaier. *Introduction to Global Optimization.* Accessed 1 April 2014. Self-Published, May 2013. URL: http://www.mat.univie.ac.at/~neum/glopt/intro.html.

[10] Paul M. Sant. *"intractable"*. in *Dictionary of Algorithms and Data Structures* [online], Vreda Pieterse and Paul E. Black eds. accessed 1 April 2014 (Modified December 17 2004). Dec. 2004. URL: http://www.nist.gov/dads/HTML/intractable.html.

[11] Paul M. Sant. *"big-O notation"*. in *Dictionary of Algorithms and Data Structures* [online], Vreda Pieterse and Paul E. Black eds. accessed 1 April 2014 (Modified August 31 2012). Aug. 2012. URL: http://www.nist.gov/dads/HTML/bigOnotation.html.

[12] Thomas Weise. *Global Optimization Algorithms - Theory and Application.* en. Second. Online available at http://www.it-weise.de/ Accessed 1 April 2014. Self-Published, June 2009. URL: http://www.it-weise.de/.

[13] Kalyanmoy Deb and Ram Bhushan Agrawal. "Simulated binary crossover for continuous search space". In: *Complex Systems* 9 (1994), pp. 1–34.

[14] Kalyanmoy Deb and Hans-georg Beyer. "Self-Adaptive Genetic Algorithms with Simulated Binary Crossover". In: *Evol. Comput.* 9 (2 June 2001), pp. 197–221. ISSN: 1063-6560. DOI: http://dx.doi.org/10.1162/106365601750190406.

[15] Larry J. Eshelman and J. David Schaffer. "Real-Coded Genetic Algorithms and Interval-Schemata". In: *FOGA*. Ed. by L. Darrell Whitley. Morgan Kaufmann, 1992, pp. 187–202. ISBN: 1-55860-263-1.

[16] Charles FF Karney. "Quaternions in molecular modeling". In: *Journal of Molecular Graphics and Modelling* 25.5 (2007), pp. 595–604.

[17] M.T. Rodgers and P.B. Armentrout. In: *J. Am. Chem. Soc.* 122 (2000), pp. 8548–8558.

[18] P.B. Armentrout. In: *Top. Curr. Chem.* 225 (2003), p. 233.

[19] P.B. Armentrout. In: *J. Am. Soc. Mass Spectrom.* 13 (2002), p. 419.

[20] P.B. Armentrout, D.A. Hales, and L. Lian. *Advances in Metal and Semiconductor Clusters.* Greenwich, CT: JAI, 1994.

[21] P.B. Armentrout, H. Koizumi, and M. MacKenna. In: *J. Phys. Chem. A* 109 (2005), p. 11365.

[22] W. Buchmann et al. In: *J. Mass Spectrom.* 42 (2007), p. 517.

[23] D.R. Carl, R.M. Moision, and P.B. Armentrout. In: *Int. J. Mass Spectrom.* 256 (2007), p. 308.

[24] R. Chawla, A. Shukla, and J. Futrell. "Collision-induced dissociation of nitrobenzene molecular cations at low energies by crossed-beam tandem mass spectrometry". In: *J. Phys. Chem. A* 105.2 (2001), p. 349.

[25] R. G. Cooks. *Collision Spectroscopy.* New York: Plenum, 1978.

[26] E.R. Fisher, B.L. Kickel, and P.B. Armentrout. In: *J. Phys. Chem.* 97 (1993), p. 10204.

[27] Y.J. Fu, J. Laskin, and L.S. Wang. In: *Int. J. Mass Spectrom.* 255 (2006), p. 102.

[28] N. Hallowita et al. In: *J. Phys. Chem. A* 112 (2008), p. 7996.

[29] A.L. Heaton and P.B. Armentrout. In: *J. Phys. Chem. A* 112 (2008), p. 10156.

[30] J. Laskin, E. Denisov, and J. H. Futrell. "Fragmentation energetics of small peptides from multiple-collision activation and surface-induced dissociation in FT-ICR MS". In: *Int. J. Mass. Spectr.* 219.1 (2002), p. 189.

[31] J. Laskin and J. H. Futrell. "Collisional activation of peptide ions in FT-ICR mass spectrometry". In: *Mass Spectrometry Reviews* 22.3 (2003), pp. 158–181.

[32] J. Laskin and J. H. Futrell. "On the efficiency of energy transfer in collisional activation of small peptides". In: *J. Chem. Phys.* 116.10 (2002), pp. 4302–4310.

[33] F Muntean and P.B. Armentrout. In: *J. Chem. Phys.* 115 (2001), pp. 1213–1228.

[34] C.-Y. Ng. In: *J. Phys. Chem. A* 106 (2002), p. 5953.

[35] M.T. Rodgers and P.B. Armentrout. In: *Mass. Spectrom. Rev.* 19 (2000), p. 215.

[36] J.-Y. Salpin and J. Tortajada. In: *J. Mass Spectrom.* 37 (2002), p. 379.

[37] A. K. Shukla and J. H. Futrell. "Tandem mass spectrometry: Dissociation of ions by collisional activation". In: *Journal of Mass Spectrometry* 35.9 (2000), p. 1069.

[38] A. K. Shukla and J. H. Futrell. "Collisional activation and dissociation of polyatomic ions". In: *Mass Spectrometry Reviews* 12.4 (1993), p. 211.

[39] R. E. Tosh, A. K. Shukla, and J. H. Futrell. "Energy transfer, scattering and dissociation in ion atom collisions: CO2 +/Ar". In: *J. Chem. Phys.* 114.7 (2001), p. 2986.

[40] D.J. Douglas. In: *J. Phys. Chem.* 86 (1982), p. 185.

[41] V.H. Wysocki, H.I. Kenttämaa, and R. G. Cooks. In: *Int. J. Mass Spectrom. Ion Processes* 75 (1987), pp. 181–208.

[42] H. Yamaoka. In: *J. Phys. Chem.* 86 (1982), p. 185.

[43] P. DeSainteClaire, G. H. Peslherbe, and W. L. Hase. "Energy transfer dynamics in the collision-induced dissociation of Al6 and Al13 clusters". In: *J. Phys. Chem.* 99.20 (1995), pp. 8147–8161.

[44] Y. Jeanvoine et al. In: *Int. J. Mass Spectrom.* 308 (2011), pp. 289–298.

[45] E. Martínez-Núñez et al. "Quasiclassical dynamics simulation of the collision-induced dissociation of Cr (CO)6 + with Xe". In: *J. Chem. Phys.* 123 (2005), p. 154311.

[46] O. Meroueh and W. L. Hase. "Energy transfer pathways in the collisional activation of peptides". In: *Int. J. Mass. Spectr.* 201.1-3 (2000), pp. 233–244.

[47] R. Spezia et al. In: *Phys. Chem. Chem. Phys.* 14 (2012), pp. 11724–11736.

[48] R. Spezia et al. In: *J. Phys. Chem. A* 113 (2009), pp. 13853–13862.

[49] T. Raz and R. D. Levine. In: *J. Chem. Phys.* 105 (1996), p. 8097.

[50] D.G. Schultz and L. Hanley. In: *J. Chem. Phys.* 109 (1998), p. 10976.

[51] T Baer and W. L. Hase. *Unimolecular Reaction Dynamics*. Oxford, 1996.

[52] E. Martinez-Nunez et al. "Quasiclassical trajectory study of the collision-induced dissociation dynamics of Ar+CH3SH+ using an ab initio interpolated potential energy surface". In: *J. Phys. Chem. A* 110.4 (2006). 010MF Times Cited:1 Cited References Count:22, pp. 1225–1231. ISSN: 1089-5639. DOI: Doi10.1021/Jp052325d.

[53] E. Martinez-Nunez, S. A. Vazquez, and J. M. C. Marques. "Quasiclassical trajectory study of the collision-induced dissociation of CH3SH++Ar". In: *J. Chem. Phys.* 121.6 (2004). 840JX Times Cited:3 Cited References Count:32, pp. 2571–2577. ISSN: 0021-9606. DOI: Doi10.1063/1.1769364.

[54] Y.-J. Chen et al. In: *J. Phys. Chem. A* 106 (2002), p. 9729.

[55] P.T. Fenn et al. In: *J. Phys. Chem. A* 101 (1997), p. 6513.

[56] J. Liu et al. "Direct dynamics study of energy transfer and collision-induced dissociation: Effects of impact energy, geometry, and reactant vibrational mode in H2CO+ - Ne collisions". In: *J. Chem. Phys.* 119.6 (2003), p. 3040.

[57] U. Lourderaj and W. L. Hase. "Theoretical and Computational Studies of Non-RRKM Unimolecular Dynamics". In: *J. Phys. Chem. A* 113.11 (2009). Lourderaj, Upakarasamy Hase, William L., pp. 2236–2253. ISSN: 1089-5639. DOI: 10.1021/jp806659f.

[58] S. O. Meroueh, Y. Wang, and W. L. Hase. "Direct dynamics simulations of collision- and surface-induced dissociation of n-protonated glycine. Shattering fragmentation". In: *J. Phys. Chem. A* 106.42 (2002), p. 9983.

[59] B. Coupier et al. In: *Eur. J. Phys. J. D* 20 (2002), p. 459.

[60] M. Imhoff, Z. Deng, and M. Huels. In: *Int. J. Mass Spectrom.* 154 (2007), p. 262.

[61] M. Imhoff, Z. Deng, and M. Huels. In: *Int. J. Mass Spectrom.* 245 (2005), p. 68.

[62] A. Le Padellec et al. In: *J. Phys.:Conf. Ser.* 101 (2008), p. 012007.

[63] T. Schlatholter et al. In: *ChemPhysChem* 7 (2006), p. 2339.

[64] T. Schlatholter, F. Alvarado, and R. Hoekstra. In: *Nucl. Instrum. Methods Phys. Res. Sect. B* 233 (2005), p. 62.

[65] J. Tabet et al. In: *Int. J. Mass Spectrom.* 292 (2010), p. 53.

[66] J. Tabet et al. In: *Phys. Rev. A* 82 (2010), p. 022703.

[67] J. Tabet et al. In: *Phys. Rev. A* 81 (2010), p. 012711.

[68] J. de Vries et al. In: *Eur. Phys. J. D* 24 (2003), p. 161.

[69] H. Budzikiewicz, J.I. Brauman, and C. Djerass. In: *Tetrahedron* 21 (1965), p. 1855.

[70] R.C. Dougherty. In: *J. Am. Chem. Soc.* 90 (1968), p. 5780.

[71] M Monge-Palacios, J. J. Nogueira, and E Martinez-Nunez. In: *J. Phys. Chem. C* 116 (2012), pp. 25454–25464.

[72] J. J. Nogueira et al. In: *J. Phys. Chem. C* 115 (2011), pp. 23817–23830.

[73] M.J.S. Dewar, E.G. Zoebisch, and E.F. Healey. In: *J. Am. Chem. Soc.* 107 (1985), pp. 3902–3909.

[74] F. Weigend, F. Furche, and R. Ahlrichs. In: *J. Chem. Phys.* 119 (2003), pp. 12753–12762.

[75] *TURBOMOLE v6.4 2012, a development of University of Karlsruhe and Forschungszentrum Karlsruhe GmbH, 1989-2007, TURBOMOLE GmbH.* Computer Program. URL: http://www.turbomole.com.

[76] S. F. Boys and F. Bernardi. In: *Mol. Phys.* 19 (1970), pp. 553–566.

[77] W. L. Hase et al. In: *QCPE* 16 (1996), p. 671.

[78] A.B. Bortz, M. H. Kalos, and J.L. Lebowitz. In: *J. Comput. Phys.* 17 (1975), pp. 10–18.

[79] D.T. Gillespie. In: *J. Comput. Phys.* 22 (1976), pp. 403–434.

[80] X. Hu and W. L. Hase. In: *J. Chem. Phys.* 95 (1991), p. 8073.

[81] R.N. Porter and L.M. Raff. *Dynamics of Molecular Collisions.* Vol. B. New York: Plenum Press, 1976.

[82] A. Linhananta and K.F. Lim. In: *Phys. Chem. Chem. Phys.* 4 (2002), pp. 577–585.

[83] J. M. C. Marques et al. “Trajectory dynamics study of the Ar+CH4 dissociation reaction at high temperatures: the importance of zero-point-energy effects”. In: *J. Phys. Chem. A* 109.24 (2005). 936PC Times Cited:11 Cited References Count:64, pp. 5415–5423. ISSN: 1089-5639. DOI: Doi10.1021/Jp044707+.

[84] K.F. Lim. In: *Quantum Chem. Program Exchange Bull.* 14.1 (1994), p. 3.

[85] J. Hippler, J. Troe, and H.J. Wendelken. In: *J. Chem. Phys.* 78 (1983), p. 6709.

[86] F.M. Mourits and F.H.A. Rummens. In: *Can. J. Chem.* 55 (1977), p. 3007.

[87] A.J. Stone. *The Theory of Intermolecular Forces.* Oxford: Oxford University Press, 1996.

[88] L. Sadr Arani et al. In: *Phys. Chem. Chem. Phys.* 14 (2012), pp. 9855–9870.

[89] K. Fukui. In: *Acc. Chem. Res.* 14 (1981), p. 363.

[90] D.H. Ess et al. In: *Angew. Chem. Int. Ed.* 47 (2008), p. 7592.

[91] J. Rehbein and B.K. Carpenter. In: *Phys. Chem. Chem. Phys.* 13 (2011), p. 20906.

[92] E. J. Bylaska et al. *NW Chem, A Computational Chemistry Package for Parallel Computers, Version 5.1.* Computer Program. 2007.

[93] R. A. Kendall et al. In: *Comput. Phys. Chem.* 128 (2000), p. 260.

[94] U. Lourderaj et al. "Direct dynamics simulations using Hessian-based predictor-corrector integration algorithms". In: *J. Chem. Phys.* 126.4 (2007), p. 044105.

[95] W. L. Hase et al. "Translational and vibrational energy dependence of the cross section for H + C2H4 â†' C2H5". In: *J. Phys. Chem.* 85.8 (1981), p. 958.

[96] C.S. Sloane and W. L. Hase. In: *J. Chem. Phys.* 66 (1977), p. 1523.

[97] B.K. Carpenter. In: *J. Phys. Org. Chem.* 16 (2003), p. 858.

[98] D. Gerlich. In: *Adv. Chem. Phys.* 82 (1992), p. 1.

[99] J. Liu, B. van Devener, and S.L. Anderson. In: *J. Chem. Phys.* 116 (2002), p. 5530.

[100] K. Song, O. Meroueh, and W. L. Hase. "Dynamics of Cr(CO)6 + collisions with hydrogenated surfaces". In: *J. Chem. Phys.* 118.6 (2003), pp. 2893–2902.

[101] J. Wang et al. "Efficiency of energy transfer in protonated diglycine and dialanine SID: Effects of collision angle, peptide ion size, and intramolecular potential". In: *Int. J. Mass. Spectr.* 230.1 (2003), pp. 57–63.

[102] L. Yang et al. In: *J. Phys. Chem. C* 112 (2008). Yang, Li Mazyar, Oleg A. Lourderaj, U. Wang, Jiangping Rodgers, M. T. Martinez-Nunez, Emilio Addepalli, Srirangam V. Hase, William L., pp. 9377–9386. ISSN: 1932-7447. DOI: 10.1021/jp712069b.

[103] J. Troe. In: *J. Chem. Phys.* 66 (1977), p. 4758.

[104] A. Derecskei-Kovacs and S.W. North. In: *J. Chem. Phys.* 110 (1999), p. 2862.

[105] M. J. Wilhelm et al. In: *J. Chem. Phys.* 130 (2009), p. 044307.

[106] S. A. Abrash et al. In: *J. Phys. Chem.* 99 (1995), p. 2959.

[107] M. Bahou and Y. P. Lee. "Photodissociation dynamics of vinyl chloride investigated with a pulsed slit-jet and time-resolved Fourier-transform spectroscopy". In: *Aust. J. Chem.* 57.12 (2004), pp. 1161–1164. ISSN: 0004-9425. DOI: Doi10.1071/Ch04117.

[108] M. Barbatti, A.J.A. Aquino, and H. Lischka. In: *J. Phys. Chem. A* 109 (2005), p. 5168.

[109] D. A. Blank et al. In: *J. Chem. Phys.* 108 (1998), p. 5414.

[110] T.R. Fletcher and S.R. Leone. In: *J. Chem. Phys.* 88 (1988), p. 4720.

[111] J. Gonzalez-Vazquez et al. “Dissociation of difluoroethylenes. I. Global potential energy surface, RRKM, and VTST calculations”. In: *J. Phys. Chem. A* 107.9 (2003), pp. 1389–1397. ISSN: 1089-5639. DOI: Doi10.1021/Jp021901s.

[112] J. Gonzalez-Vazquez et al. “Dissociation of difluoroethylenes. II. Direct Classical Trajectory Study of the HF elimination from 1,2-difluoroethylene”. In: *J. Phys. Chem. A* 107.9 (2003), pp. 1398–1404. ISSN: 1089-5639. DOI: Doi10.1021/Jp021902k.

[113] G. He et al. In: *J. Chem. Phys.* 103 (1995), p. 5488.

[114] Y. Huang et al. In: *J. Chem. Phys.* 103 (1995), p. 5476.

[115] W. A. Jalenak and N. S. Nogar. In: *J. Chem. Phys.* 79 (1983), p. 816.

[116] S. Kato and K. Morokuma. In: *J. Chem. Phys.* 74 (1981), p. 6285.

[117] H.S. Ko et al. In: *J. Chem. Phys.* 117 (2002), p. 6038.

[118] Y.R. Lee et al. In: *J. Chem. Phys.* 113 (2000), p. 5331.

[119] S. R. Lin et al. “I. Three-center versus four-center HCl-elimination in photolysis of vinyl chloride at 193 nm: Bimodal rotational distribution of HCl ($v <= 7$) detected with time-resolved Fourier-transform spectroscopy”. In: *J. Chem. Phys.* 114.1 (2001), pp. 160–168. ISSN: 0021-9606.

[120] S. R. Lin et al. “Three-center versus four-center elimination in photolysis of vinyl fluoride and vinyl bromide at 193 nm: Bimodal rotational distribution of HF and HBr ($v <= 5$) detected with time-resolved Fourier transform spectroscopy”. In: *J. Chem. Phys.* 114.17 (2001), pp. 7396–7406. ISSN: 0021-9606.

[121] E. Martínez-Núñez et al. “Product energy distributions for the four-center HF elimination from 1,1-difluoroethylene. a direct dynamics study”. In: *Chem. Phys. Lett.* 348.1-2 (2001), p. 81.

[122] E. Martínez-Núñez and S. Vázquez. “Rotational distributions of HBr in the photodissociation of vinyl bromide at 193 nm: An investigation by direct quasiclassical trajectory calculations”. In: *Chem. Phys. Lett.* 425.1-3 (2006), p. 22.

[123] E. Martínez-Núñez and S. Vázquez. “Quasiclassical trajectory calculations on the photodissociation of C F2 CHCl at 193 nm: Product energy distributions for the HF and HCl eliminations”. In: *J. Chem. Phys.* 122.10 (2005), p. 1.

[124] E. Martínez-Núñez and S. Vázquez. “Rovibrational distributions of HF in the photodissociation of vinyl fluoride at 193 nm: A direct MP2 quasiclassical trajectory study”. In: *J. Chem. Phys.* 121.11 (2004), p. 5179.

[125] E. Martínez-Núñez and S. A. Vázquez. “Three-center vs. four-center HF elimination from vinyl fluoride: A direct dynamics study”. In: *Chem. Phys. Lett.* 332.5-6 (2000), p. 583.

[126] E. Martínez-Núñez et al. “Further investigation of the HCl elimination in the photodissociation of vinyl chloride at 193 nm: A direct MP2/6-31G(d,p) trajectory study”. In: *Chem. Phys. Lett.* 386.4-6 (2004), p. 225.

[127] J.-F. Riehl and K. Morokuma. In: *J. Chem. Phys.* 100 (1994), p. 8976.

[128] T. Takayanagi and A. Yokoyama. In: *Bull. Chem. Soc. Japan* 68 (1995), p. 245.

[129] T. Tarrazo-Antelo, E. Martinez-Nunez, and S. A. Vazquez. “Ab initio and RRKM study of the elimination of HF and HCl from chlorofluoroethylene”. In: *Chem. Phys. Lett.* 435.4-6 (2007). 140AR Times Cited:0 Cited References Count:35, pp. 176–181. ISSN: 0009-2614. DOI: DOI10.1016/j.cplett.2006.12.075.

[130] M. Umemoto et al. In: *J. Chem. Phys.* 83 (1985), p. 1657.

[131] C.A. Bird and D.J. Donaldson. In: *Chem. Phys. Lett.* 249 (1996), p. 40.

[132] D. A. Blank et al. In: *J. Chem. Phys.* 108 (1998), p. 5784.

[133] A. Fahr and A.H. Laufer. In: *J. Phys. Chem.* 96 (1992), p. 4217.

[134] L. Letendre and H.-L. Dai. In: *J. Phys. Chem. A* 106 (2002), p. 12035.

[135] S. W. North and G. E. Hall. In: *Chem. Phys. Lett.* 263 (1996), p. 143.

[136] C Gonzalez and H. B. Schlegel. In: *J. Phys. Chem.* 94 (1990), p. 5523.

[137] M. J.; Frisch et al. *et. al. Gaussian 09.* Computer Program. 2009.

[138] W. H. Miller. In: *J. Am. Chem. Soc.* 101 (1979), p. 6810.

[139] K.M. Ervin, J. Ho, and W.C. Lineberger. In: *J. Chem. Phys.* 91 (1989), p. 5974.

[140] J.B. Moffat. In: *J. Phys. Chem.* 81 (1977), p. 82.

Other interesting references to the reader

Erling D. Andersen. *Linear optimization: Theory, methods, and extensions*. Accessed 1 April 2014. Self-Published, 1998. URL: http://plato.asu.edu/ftp/linopt.pdf.

M.S. Bazaraa, H.D. Sherali, and C.M. Shetty. *Nonlinear Programming: Theory and Algorithms*. Wiley, 2006. ISBN: 9780471787761.

Kent Beck. *Una explicación de la programación extrema*. Pearson Educación, 2002, 189 pages. ISBN: 8478290559.

John Calcote. *Autotools: A Practioner's Guide to GNU Autoconf, Automake, and Libtool*. 1st. San Francisco, CA, USA: No Starch Press, 2010. ISBN: 1593272065, 9781593272067.

Bruce Eckel. *Piensa en Java*. Pearson Educación, 2002, 906 pages. ISBN: 9788420531922.

Brian Foy, Tom Phoenix, and Randal Schwartz. *Learning Perl*. "O'Reilly Media, Inc.", 2011, 363 pages. ISBN: 9781449303587.

Daniel Gilly and O'Reilly & Associates. *UNIX in a nutshell*. O'Reilly & Associates, 1992. ISBN: 9781565920019.

H. J. Greenberg. *Myths and Counterexamples in Mathematical Programming*. (ongoing, first posted October 2008, Accessed 1 April 2014.) http://glossary.computing.society.informs.org: INFORMS Computing Society, Feb. 2010.

David Gunter and Jack Tackett. *Utilizando Linux*. Prentice Hall, 1996, 846 pages. ISBN: 9788489660557.

Francisco Herrera, Manuel Lozano, and Jose L. Verdegay. "Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis". In: *Artificial intelligence review* 12.4 (1998), pp. 265–319.

Jarkko Hietaniemi, John Macdonald, and Jon Orwant. *Mastering Algorithms with Perl*. O'Reilly Media, Inc., 1999, 684 pages. ISBN: 9781565923980.

C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations.* Frontiers in Applied Mathematics 16. Accessed 1 April 2014. SIAM, 1995. URL: http://www.siam.org/books/textbooks/fr16_book.pdf.

C. T. Kelley. *Iterative Methods for Optimization.* Vol. 19. Accessed 1 April 2014. SIAM Frontiers in Applied Mathematics, 1999. URL: http://www.siam.org/books/textbooks/fr18_book.pdf.

Olaf Kirch. *Linux.* O'Reilly Media, 1995, 335 pages. ISBN: 9781565920873.

Donald Ervin Knuth. *The art of computer programming.* Vol. 1,2,3,4A. Pearson Education, 1968-2011.

Jesse Liberty. *C++ para principiantes.* Pearson Educación, 2000, 422 pages. ISBN: 9789701704165.

H. A. Luther, James O. Wilkes, and Brice Carnahan. *Cálculo numérico.* Rueda, 1979, 639 pages. ISBN: 8472070131.

Félix García Merayo. *Programación en FORTRAN 77.* Paraninfo, 1991, 399 pages. ISBN: 9788428318181.

James Newkirk et al. *La programación extrema en la práctica.* Pearson Educación, 2002, 200 pages. ISBN: 8478290575.

Francisco José Baptista Pereira. "Estudo das interacções entre evolução e aprendizagem em ambientes de computação evolucionária". PhD thesis. 2002. URL: http://hdl.handle.net/10316/1744.

Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming.* (With contributions by J. R. Koza. Accessed 1 April 2014.) Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008. URL: http://www.gp-field-guide.org.uk.

William H. Press et al. *Numerical Recipes 3rd Edition: The Art of Scientific Computing.* Cambridge University Press, 2007. ISBN: 0521880688.

Eric S. Raymond. *The Art of UNIX Programming.* Pearson Education, 2003. ISBN: 0131429019.

Herbert Schildt. *C.* Osborne MacGraw-Hill, 1989, 358 pages. ISBN: 9788476153819.

Fco. Javier Ceballos Sierra and Francisco Javier Ceballos Sierra. *C/C++.* RA-MA S.A. Editorial y Publicaciones, 2001, 704 pages. ISBN: 9788478974801.

Kathy Sierra and Bert Bates. *Head First Java, 2nd Edition.* O'Reilly Media, 2005. ISBN: 0596009208.

Nick Sofroniou, Apostolos Syropoulos, and Antonis Tsolomitis. *Digital Typography Using LaTeX.* Springer, 2003, 510 pages. ISBN: 9780387952178.

James C. Spall. *Introduction to Stochastic Search and Optimization.* Wiley-Interscience, 2003, 595 pages. ISBN: 9780471330523.

S. Srinivasan. *Advanced Perl programming.* A Nutshell handbook. O'Reilly, 1997. ISBN: 9781565922204.

Johan Vromans. *Perl 5 pocket reference.* O'Reilly Media, 2000, 90 pages. ISBN: 9780596000325.

Kurt Wall. *Programación en Linux con ejemplos.* Prentice-Hall, 2000, 541 pages. ISBN: 9789879460092.

L. Wall, T. Christiansen, and J. Orwant. *Programming Perl.* O'Reilly Series. O'Reilly, 2000. ISBN: 9780596000271.

L. Darrell Whitley, ed. *Proceedings of the Second Workshop on Foundations of Genetic Algorithms. Vail, Colorado, USA, July 26-29 1992.* Morgan Kaufmann, 1993. ISBN: 1-55860-263-1.

Stephen Wright and Jorge Nocedal. *Numerical Optimization.* Springer Verlag, 2006, 664 pages. ISBN: 9780387303031.

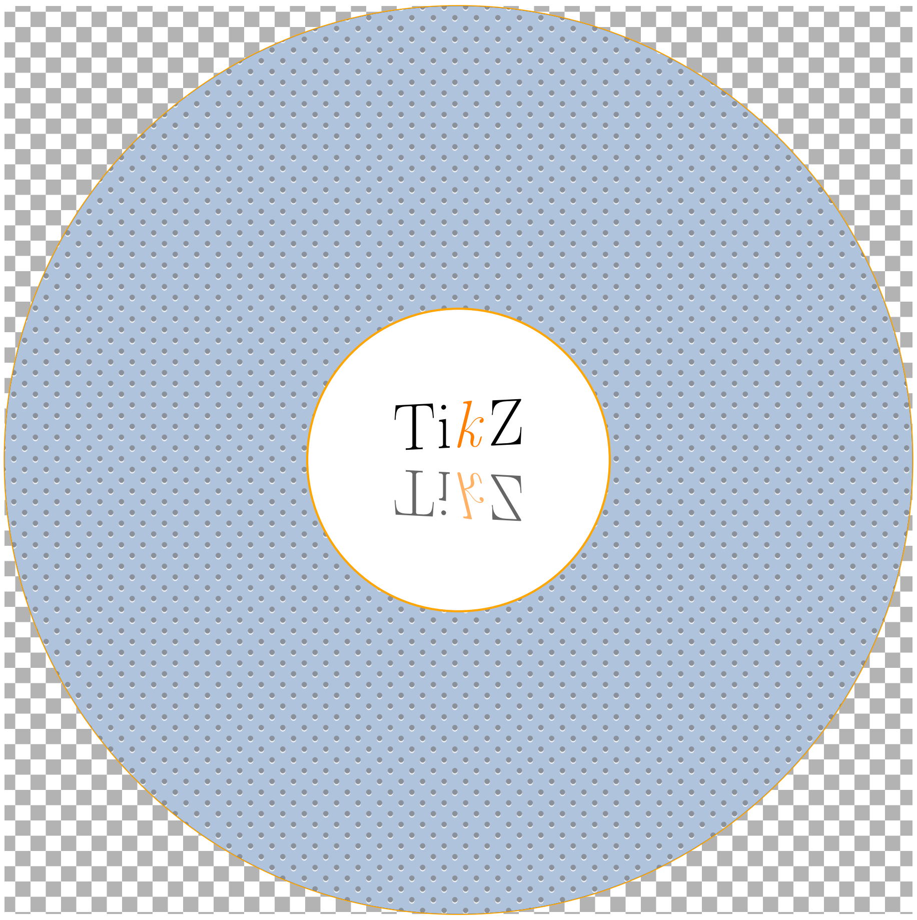
Index

List of Tables

List of Figures

File list

This thesis was typeset using the LaTeX typesetting system.


TikZ

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

FACULTADE DE QUÍMICA