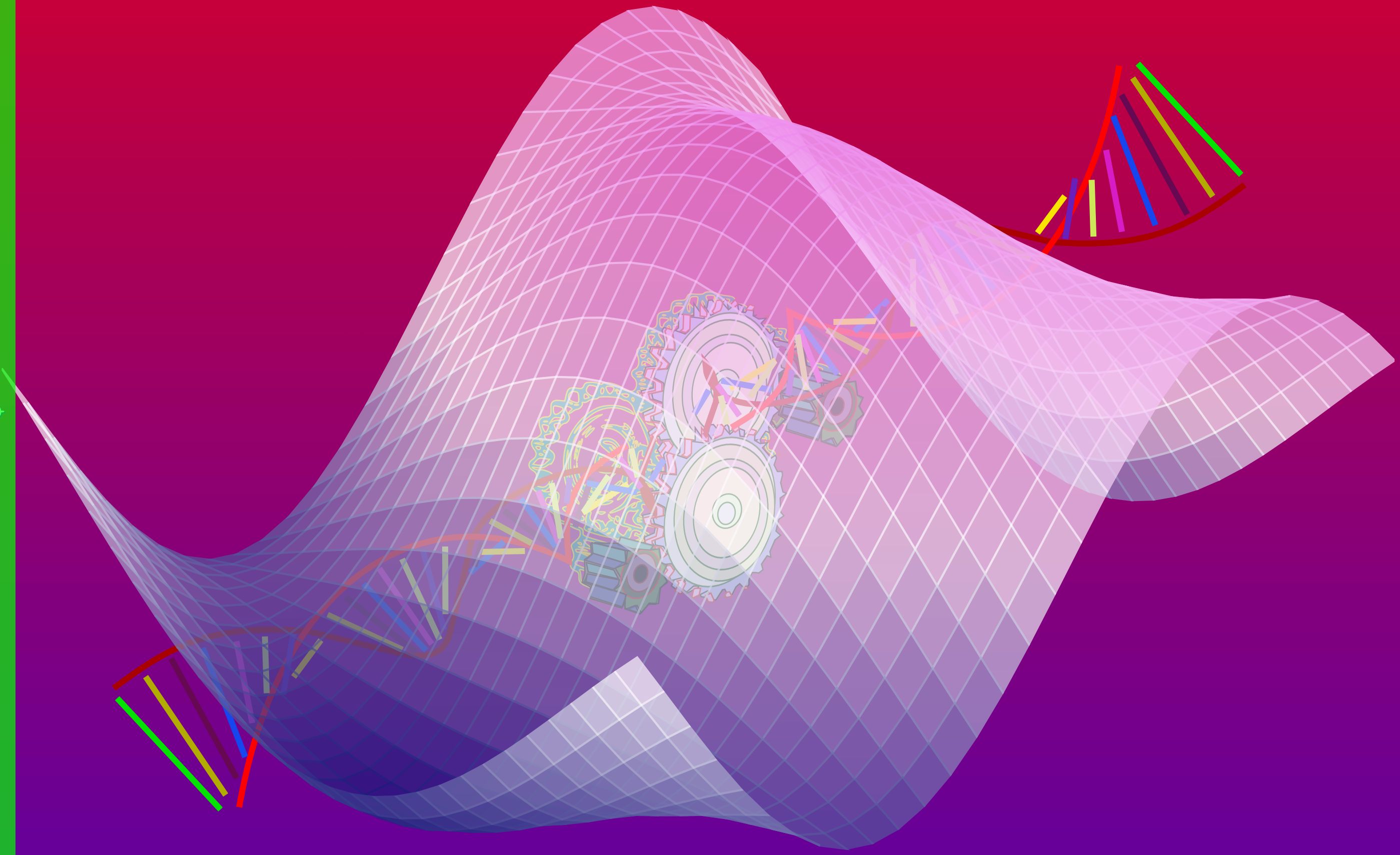
GAFit



User Manual

January 4, 2020
Version 1.6

GAFit
User Manual

Roberto Rodríguez-Fernández
Francisco Baptista Pereira
Jorge M. C. Marques
Saulo Vázquez-Rodríguez
Emilio Martínez-Núñez

January 4, 2020
Version 1.6
Build 383

Contents

Conventions

Symbols

⟶ tabs

␣ blank spaces

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

⤸ wrapped line

⤷ wrapped line continuation

Acronyms

AM1	Austin Model 1	89
BLX-α	Blend Alpha Crossover	112
CHARMM	Chemistry at HARvard Macromolecular Mechanics	12
CPU	Central Processing Unit	11
DNA	deoxyribonucleic acid	104
DPC	Double Point Crossover (or TPX, Two Point Crossover)	106
FPU	Floating Point Unit	33
GA	Genetic Algorithm	12
MOPAC	Molecular Orbital PACkage	12
MPX	Multiple Point Crossover	106
NFS	Network File System	177
PES	Potential Energy Surface	11
SBX	Simulated Binary Crossover	112
SPC	Single Point Crossover (or SPX)	106
VC	vinyl cyanide	87

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:

```
[..]
MODULE INTERMOLECULAR
---------------------
Coordinates:[coord.molden]
Energies:[energies.txt]
Atom2type:[atom2type.txt]
Bounds:[bounds.txt]
Charges:[charges.txt]
Potential read: 1
All coefficients: no, Read and repeat subset
Interactions types: inter
Fitting: relative
[...]
```

- An input or output file:

File 1: Input file example.

```
[job]
coefficients:␣␣5
```

- Source code file:

File 2: C source code

```
int i;

//to print stats every evaluations/1000
int last_evals;
int outputeach = 0;

if (jo->evaluations < STATS_MAX_LINES / 10)   //100
  outputeach = 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 ’**}**’.

License and citation

License

Citation

The main features of **GAFit** are described the following papers:

1. Roberto Rodríguez-Fernández, Francisco B. Pereira, Jorge M.C. Marques, Emilio Martínez-Núñez, and Saulo A. Vázquez. “GAFit: A general-purpose, user-friendly program for fitting potential energy surfaces”. In: *Computer Physics Communications* 217 (2017), pp. 89–98. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2017.02.008. URL: http://www.sciencedirect.com/science/article/pii/S0010465517300607

2. J. M. C. Marques, F. V. Prudente, F. B. Pereira, M. M. Almeida, A. M. Maniero, and C. E. Fellows. “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

Please, cite these articles in every scientific work that reports results obtained with GAFit.

Simplified User Guide

This manual is accompanied by a concise guide:

SimplifiedUserGuide.pdf.

It's recommended to read it before this manual. It's focused in simple and practical examples of use: Fitting pairwise intermolecular potentials, interfacing with the CHARMM program, interfacing with the MOPAC program and fitting a user-defined function to data.

Part I

Short manual

1

GAFit

1.1 Introduction

> 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)[3]. 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 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.

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

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]. 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 intermolecular potentials for the interaction between two fragments–e.g. molecules–, and to the reparametrization of a semiempirical Hamiltonian[2]. 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, Prudente, Pereira, Almeida, Maniero, and Fellows [2] and co-workers.

The functionality of the package was extended separating the core itself from the fitting targets –See Figure 1.1–. 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 –*internal job type*–. A complete set of tools were added to the package to facilitate the creation and configuration of input files.

In addition, a external interface –*external job type*– 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) and Chemistry at HARvard Macromolecular Mechanics (CHARMM), among others.

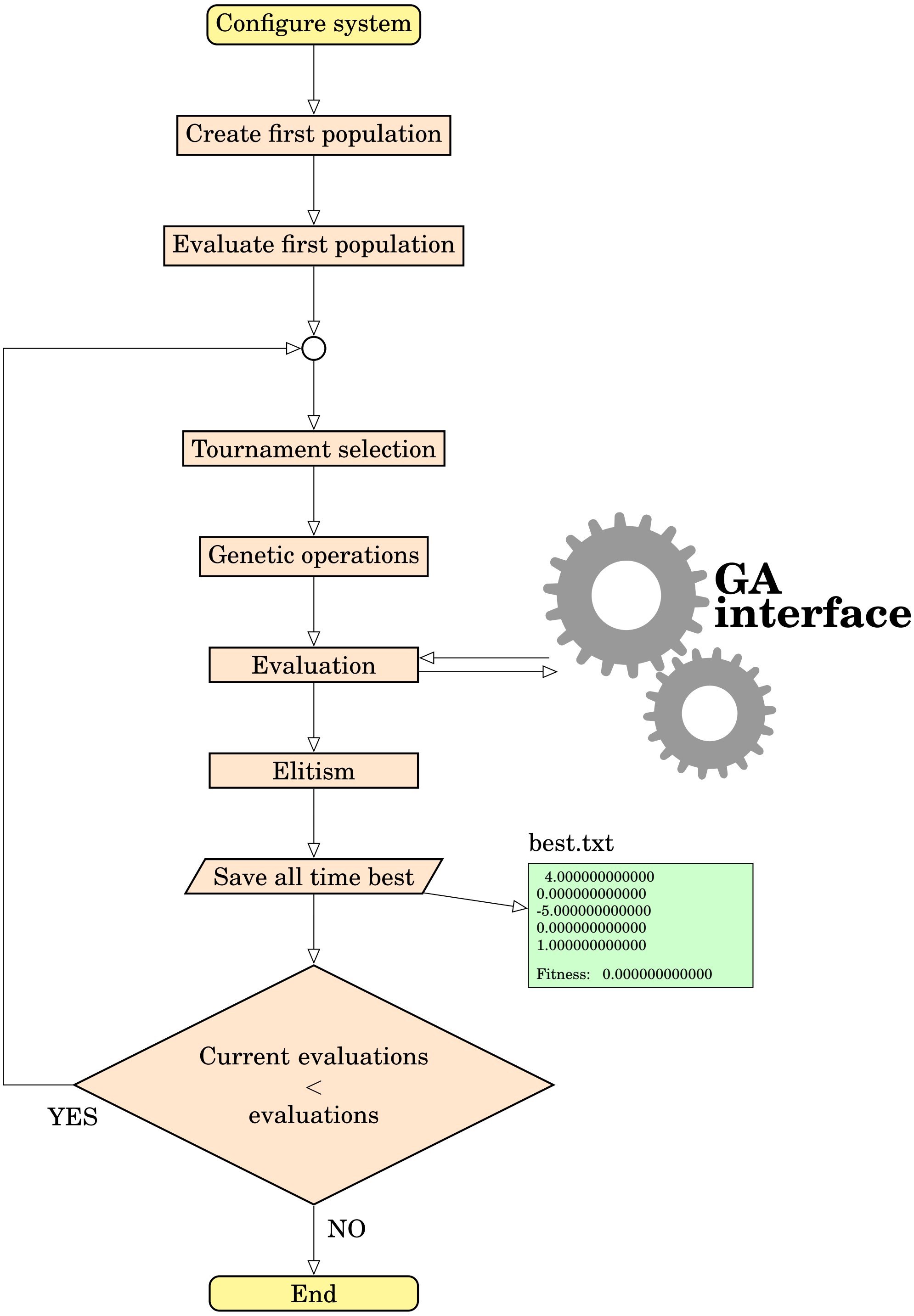
1.2 Installation

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:

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



```
  4.000000000000
0.000000000000
-5.000000000000
0.000000000000
1.000000000000

Fitness:   0.000000000000
```

Figure 1.1: GA main loop

```
./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).

1.3 Configuration

GAFit uses only one configuration file: *job.txt*, divided into logic **[sections]**. Each **[section]** have key/value pairs and all have default values.

File 1.1: job.txt file example

```
[job]
runs:         1
type: external auto
command: external-intermolecular.sh
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
direction:       min

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

- The **[sections]** could be skipped if default values are used.
- Only options applicable to the actual job are processed.
- Options or sections erroneous are omitted.
- Section and options names are case insensitive.

There are three main **[sections]**:

Parameters: Genetic algorithm settings. It is safe to skip this section: default values are good.

Job: Job definition and its options.

Print: Output options.

1.4 Simple configuration

Some of the applications presented here have default values except for a few group of parameteres which must be given from users. A simple configuration method has been developed using a few key directives in the configuration file *job.txt*, as shown in the example File 1.2.

File 1.2: Simple configuration job.txt file example

```
[Job]
Evaluations: 100000
Application: MOPAC
Exec: /usr/programs/mopac/MOPAC2016.exe
```

The application modules with an alternative *simple configuration* are sumarized in table 1.1.

Table 1.1: Modules with a *simple configuration*.

application	notes
intermolecular	Intermolecular potential energy fit.
mopac	Fitting the properties of a molecular system using MOPAC.
charmm	Fitting the properties of a molecular system using CHARMM.
mvariable	multivariable fitting.
generic	Generic module to interface external programs.

The keyword **application** in File 1.2 shown that the *simple configuration module* in use is **mopac**.

1.5 Jobs

This changed from **version 1.3** onwards.

An external program evaluates the individuals generated by **GAFit** like MOPAC or CHARMM. The external program behavior is the target of the fit.

The **type** option value must be set in section **[job]** to select how to communicate with the external program or the intermediate programs between **GAFit** and the target program.

external : One individual is passed in each external program run.

external bulk : The whole generation passed per external program run.

external auto : The external program knows the **GAFit**'s protocol and can configure it as needed.

1.6 Examples included

There are several folders in the package with examples divided in two categories:

Simple configuration method This examples, from the *Simplefied User Guide*, are in the folder **simple-mode-examples** and follow Section 1.4.

- charmm
- intermolecular
- mopac
- mvariable
- generic

Detailed configuration This examples are in the folder **advanced-mode-examples**.

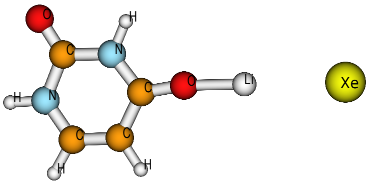
- Forcefield

  charmm Charmm example.

- Intermolecular

  uracil Here the interaction between Xe and the $[Li(Uracil)]^+$ complex is studied.

Figure 1.2: $[Li(Uracil)]^+$- Xe example.



analytical Same as the *uracil example* but using an analytical expression as potential.

$\mathbf{n_2n_2}$ Here the interaction between two nitrogen molecules is studied. A fully custom potential can be implemented using *userpotential.f* file.

- Miscellaneous

external
An example in C with a generic external fit. The given test code supports both *external* and *external bulk* options. 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.

poly-fortran The same as above but written in fortran.

exponential Code in C to fit data from file *exponential.values* to an exponential.

$$\sum_{i=1}^{n} a_i e^{-b_i * x}$$

To use a exponential with **n** terms you must specify in file *job.txt* **coefficients=2*n** and give them a name. The coefficient limits are taken from *bounds.txt* file.

mvariable multivariable example fitting.

- mopac: Change and/or set MOPAC_EXECUTABLE and MOPA C_LICENSE in file *external-mopac2009.sh* to run with MOPAC (any version from)2012 to 2016).

mopac It employs the interface with MOPAC. Source code for the interface tools is in the *src/mopac* folder.

shepherd It employs the enhanced interface with MOPAC.

vc As in the previous one, it uses the enhanced interface with MOPAC. Taken from Homayoon, Vázquez, Rodríguez-Fernández, and Martínez-Núñez [5]

gradient Optimizing aldehyde using gradients respect to Cartesian coordinates in a pm6 model parametrization.

2 Jobs

> 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*

The files needed for a *job* depend on the type of job to be done where an external program or tool evaluates the coefficients vector. 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. . .

Other interfaces have been developed apart from MOPAC: CHARMM and mvariable.

2.1 Job configuration

GAFit can pass the coefficient vectors one per run, **external type**, a whole population per run, **external bulk type**, or as required by the external program, **external auto type**.

- Simple configuration: **external** or **external bulk**. There are six options to configure as shown in File 2.1:

  type: external bulk. Whole population per run.

  command: ./external, **external command** to execute per run. Normally a *shell script*.

  coefficients: 5, number of coeficients.

external input: external.input, file where **GAFit** will write all the population.

external fit: external.fit, file where the **external command** will write the evaluation of each individual to be read by **GAFit**.

bounds: bounds.txt, bounds file.

File 2.1: External job simple configuration example

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

- Automatic configuration: **external auto**. Only two options to configure as shown in File 2.2:

  type: external auto.

  command: ./external,

  GAFit obtains its configuration from **external command**. When **GAFit** calls the **external command** with a command line parameter with value **"0"**, the external command write a file named *"response"* with the requested configuration to **GAFit**.

File 2.2: External job automatic configuration example

```
[job]
evaluations:     50000
type:            external auto
command:         ./external
```

File 2.3: Response file from the external command

```
[job]
type: external bulk
coefficients: 16
external input: mopac.input
external fit: mopac.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

3

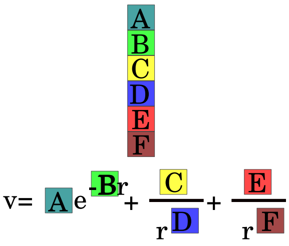
Intemolecular Module

> A mathematician is a device for turning coffee into theorems.
>
> *Alfréd Rényi*

The intermolecular module is intended to parametrize an internal intermolecular potential energy function to fit a set of interaction energies between two fragments (or intra in the same fragment).

In the simple configuration method use the keyword **application: intermolecular**

Figure 3.1: Intermolecular potential pair example.

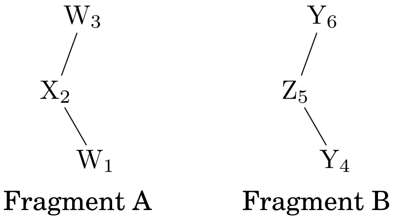


If we use an intermolecular potential pairs like Fig. 3.1:

- the genes are A, B, C, D, E, F.
- the chromosome is: ABCDEF

3.1 An example

Here, we are going to fit intermolecular potential pairs like $V = Ae^{-Br} + \frac{C}{r^D}$ between two fragments (e.g. a molecule). A, B, C, D are the coefficients to fit per each interaction. W, X, Y, Z are the atoms in the fragments.



The data to fit are **geometries** and their correspondent **energies** like the File 3.1 named as *geometries.txt* and the file 3.2 named *energies.txt* respectively.

File 3.1: geometries.txt

```
                6
                  X              Y              Z
   W     −13.694289     −0.182672       0.000000
   X     −13.299638      0.824476       0.000000
   W     −12.403476     −0.960776       0.000000
   Y     −14.263389     −0.348152      −0.831048
   Z     −14.263389     −0.348152       0.831048
   Y     −11.316612      0.153002       0.000000
                6
                  X              Y              Z
   W      −9.694289     −0.182672       0.000000
   X      −9.299638      0.824476       0.000000
   W      −8.403476     −0.960776       0.000000
   Y     −10.263389     −0.348152      −0.831048
   Z     −10.263389     −0.348152       0.831048
   Y      −7.316612      0.153002       0.000000
   ...
```

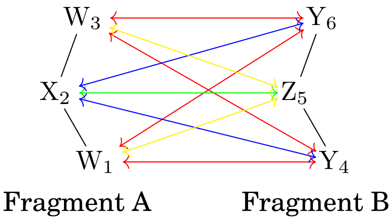
The **geometries** format is the known as “xyz” format. The **energies** are obtained from high level *ab initio* calculations.

File 3.2: energies.txt

```
 −0.016881788   1
 −0.024242894   1
 −0.033981373   1
   ...
```

The second column into *energies.txt* is the weight of the correspondent geometry in the fit.

3.2 Interactions



The interactions in our example are shown above. The atoms W_1 and W_3 are equivalents like Y_4 and Y_6. So, there are 4 different interactions with some redundant coefficients:

$$\begin{aligned} A_{1,4} = A_{1,6} = A_{3,4} &= A_{3,6} \\ B_{1,4} = B_{1,6} = B_{3,4} &= B_{3,6} \\ &\dots \\ A_{2,6} &= A_{2,4} \\ B_{2,6} &= B_{2,4} \\ &\dots \\ &etc \end{aligned}$$

We have to inform about this with a new file, **atom2type**, who maps between *atom number* in the geometry and their *type number*. This file is named *atom2type.txt*, and shown in File 3.3.

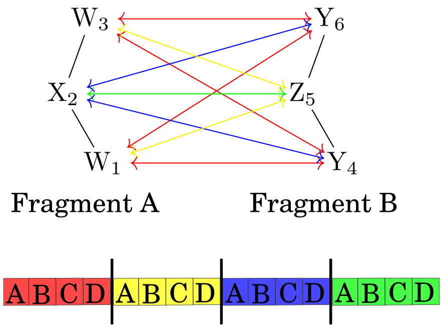
File 3.3: atom2type.txt

```
 3  6
     1      W         1
     2      X         2
     3      W         1
     4      Y         3
     5      Z         4
     6      Y         3
```

The information in the file is as follows:

- In the first line, the number of atoms in the Fragment A and the total number of atoms.
- In each of the following lines, the *atom number* as noted in the *geometry*, the chemical symbol and the *type number*.
- It can be done manually or by a utility included in the **GAFit**'s package called **needle**.

Using this information, **GAFit** knows that there are 4 different interactions, so there are 4 equations of 4 coefficients each. In this case, a **chromosome** from any individual has 16 real values:



3.3 charges

If our potential use partial charges, we must use another file, *charges.txt*:

- Partial charges must be specified using the *atom types* considered in the file *atom2type*.
- A template with *0* values can be generated by the **needle** tool.

One of the included potentials, the fourth, use charges. See Table 3.1.

File 3.3 has four different *atom types*, so File 3.4 has four lines.

File 3.4: charges.txt

```
+0.12
−0.24
−0.08
+0.16
```

3.4 needle

needle is a tool written in Perl to analyze the *geometry* –File 3.1– file building the *atom2type* –File 3.3– and *charges* –File 3.4– files automatically.

```
$ needle -h
 needle v0.5 (c)GAFit toolkit -  2010-2013
    collects sets of equivalent atoms
    input: any geometries input file
        -d        debug
        -p N     fragment A atoms
        -o       creates needed files
```

needle builds bonds and rings from atom Cartesian coordinates and search for equivalent atoms. This only work for F, H, Si, O, N, S, C and Au. You can invoke **needle** as shown below:

```
$ needle -p 3 -o geometries.txt
```

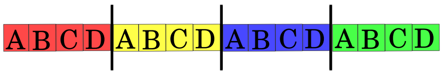
-p 3 there are 3 atoms in **fragment A**.

-o create the files *atom2type* and *charges*.

More on **needle** in section 20.1.

3.5 Bounds

This is our chromosoma:



And now, we have to establish the limits of each coefficient, each gene. This is accomplished with a new file, *bounds.txt*. We can do this giving values to only the first fourth coefficients, or to all of them, setting the option **all coefficients** to **no** or **yes** respectively in the **[job]** section.

- option **all coefficients:no**, 4 bounds:

```
TEXT OR EMPTY
          -100      100.      9
          0.         100.0    9
          -1500.    5000.0    9
          3         5         0
```

- option **all coefficients:yes**, 16 bounds:

```
TEXT OR EMPTY
          -100      100.      9
          0.         100.0    9
          -1500.    5000.0    9
          3         5         0
          0.         100.0    9
          -1500.    5000.0    9
          [...]
```

- The first and second column are the lower and upper bound respectively.

- In the third column:

  0 the gen (coefficient) is handled as integer.

  9 the gen is handled as real number.

  1..8 the gen is handled as real number but using from 1 to 8 decimal places as specified here.

- the first line is skipped, so you can leave it empty or write a comment on it.

3.6 Fitting

There are three types of fitting:

absolute

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

relative

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

user This option route the fitting to a user defined function.

3.7 Defined potentials

The value of **potential** in the section **[job]** selects the defined potential to use as shown in Table 3.1. We have to use the number **1** from table.

Table 3.1: Included potentials.

Value	Coefficients	Potential
-1	any	any user defined in userpotential.f
0	any	any analytical expression defined in an **[analytical]** section
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	2	$V = A\left[\left(\frac{B}{r}\right)^{12} - \left(\frac{B}{r}\right)^{6}\right] + 332.0532\frac{q_i q_j}{r}$

3.8 Final configuration

Now, we can write the configuration like File 3.5. All the options, except the **type**, **command** and **Evaluations**, are defaults, so they could be omitted. The option **Evaluation: 5000000** means that **GAFit** run for **5000000** generations and then finish.

File 3.5: job.txt

```
[job]
type: external auto
command: external—intermolecular.sh
Evaluations:   5000000
Geometries:    geometries.txt
Energies:      energies.txt
Atom2Type:     atom2type.txt
Bounds:        bounds.txt
Potential:     1
```

The program binary which comprises the **intermolecular** module is **intpot**. We have to call it passing the correct options. In the *job.txt* we call *external-intermolecular.sh*–File 3.6– which in turn call **intpot** and sets some options.

the **inpot** binary must be in the PATH!

File 3.6: external-intermolecular.sh

```
#!/bin/sh


export EXTERNAL_INPUT="intpot.input"
export EXTERNAL_FIT="intpot.fit"
export BOUNDS_FILE="bounds.txt"

intpot $1 bulk
```

The simple configuration method equivalent to both File 3.5 and File 3.6 is only 3.7.

File 3.7: simple configuration job.txt

```
[job]
Evaluations:   5000000
Application:   intermolecular
Potential:     1
```

3.9 Results

GAFit runs in the folder where is the configuration file, *job.txt*–, writing the best values found till now to the file *best.txt*. After the **5000000** generations, in this file is the best result obtained. Is this the best result possible? Sure not, but it is the best found in this run.

File 3.8: best.txt

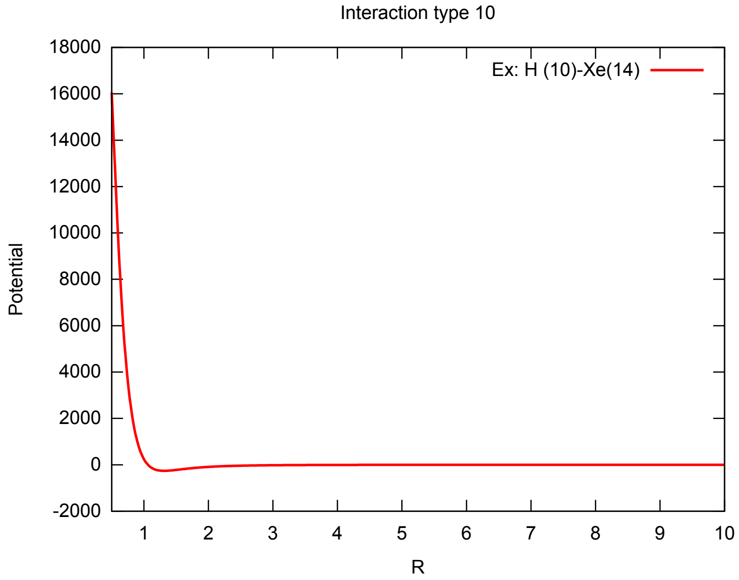
```
671108.383527237223
5.000000000000
−480.511518927649
−522.865043822352
7.000000000000
[...]

Fitness: 7.0634075026 83
```

3.10 Plotting results

Included in the **intermolecular** module is the **fitview** utility –More in section 20.2–.

- **fitview** extracts and create a bunch of plots to view the results using the saved *best.txt* file coefficients.
- you can select the upper and lower limits and the stepping –**delta** option– of the plot.
- it generates two files per plot, one with the data and other with the **gnuplot** commands needed to create graphic files –bmp, jpeg, pdf, etc– or direct plot to a graphical terminal. The data file could be used to load data to a spreadsheet.
- the plots are:
  - one per interaction.
  - an general evaluation with all the geometries.
  - all the interactions in the same plot.

```
$ fitview -h
fitview v0.3 (c)GAFit toolkit - 2010-2013
Usage: fitview [tag] [-l value] [-u value] [-d value] [-h]
           -l lower bound
           -u uper bound
           -d delta
           -e gnuplot supports enhanced terminal
           -h this help
           -g general evaluation only
           default [0.500000,10.000000] delta: 0.010000
```



3.11 FORTRAN interface

As stated before, you can also write your own routines in FORTRAN to add a new intermolecular pair potential. To do so, the **intermolecular** module expose to the user two FORTRAN modules and two routines.

- FORTRAN modules:
  - VGLOBALES: it give access to the program core variables.
  - USERDATA: here, the user can load and customize its own variables and data.
- subroutines and functions:
  - function **ix(i,j,k)**: usefull to access coefficients knowing only which are the atoms involved in the interaction.

    k k=1, coefficient A, k=2, coefficient B, ...
    i first interaction atom.
    j second interaction atom.
  - subroutine **coordinates(geo,atom,x,y,z)**: to obtain the atom's Cartesian coordinates.

    geo geometry number.
    atom atom number.
    x,y,z returning coordinates from subroutine.

There are two options to add a new potential in the code –src/inter folder–:

- Add it into *potential.f*. In this file are coded the potentials 1,2,3 and 4. You can modify one of them or add a new one, assigning it a positive integer number. The subroutines to modify are:

- **setcoefs** to register the number of parameters needed.
- **getcharges** to take into account if the charges are needed.
- **potRouter** to route the new potential to its function.
- **curRouter** to route **fitview** utility to the new potential.

In the **[job]** change potential option to **potential=n**, where **n** is the number choosen in **potRouter** subroutine to route to the new potential.

- Modify userpotential.f: this file is a *template*, you only need to customize it for your needs:
  - you must write the potential to use.
  - modify the fitting function or write a new one.
  - in the **[job]** change potential option to **potential=-1**.

In both cases is mandatory to recompile **GAFit**. We are going to view in detail how to implement a new potential using *userpotential.f*.

- Add your code to userdata module if needed.

```
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
```

- Change the number of coefficients and if you are using the *charges* file.

```
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
```

- Here you have nothing to change...

```
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,k)
       vpot=vpot+userv(d,i,k,x,nmax)
       enddo
      enddo
c ^-----------CHANGE-ME-IF-NEEDED---------------^
      return
      end
```

- Modify or write your potential here.

```
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
```

- Change or write your fitting function.

```
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
```

- Recompile gafit.
- in the **[job]** change potential option to **potential=-1**
- run **GAFit**.

3.12 Analytical expressions

You can use an analytical expression as potential setting **potential=0** in the **[job]** section as shown in File 3.9.

File 3.9: Analytical expression

```
[job]
Evaluations:    5000000
Geometries:     geometries.txt
Energies:       energies.txt
Atom2Type:      atom2type.txt
Bounds:         bounds.txt
Potential:      0

[analytical]
coefficients: a, b, c, d
distance:       dist
expression:     test potential 1
potential:      pot

[test potential 1]
v1=a*exp(-b*dist);
v2=c/dist**d;
pot=v1+v2
```

- Analytical expressions are compiled to *bytecode* once.
- The bytecode is run into a virtual Floating Point Unit (FPU) as needed.

- As interpreted code, It run 10 times slower than FORTRAN potentials compiled into source code.
- They are easy to write and modify.
- There is a utility, **ufpu**, to test them before **GAFit** run.
- The [analytical] section informs **GAFit** which are the names in use for the coefficients, the distance and the potential variables. Also, in this section you tell gafit which analytical expression to use. You can have many analytical expressions defined, each one in its own section as shown in the File 3.10.

File 3.10: Many analytical expressions

```
...

[analytical]
expression: potential 3
distance: dist
potential: pot
coefficients: a, b, 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]
v1=a*exp(-b*dist);
v2=c1/dist**c2;
v3=d1/dist**d2;
v4=e1/dist**e2;
pot=v1+v2+v3+v4

...
```

The analytical expressions compiler supports the operators and functions noted in Table 3.2.

Table 3.2: Operators and functions supported

Operators		Precedence	Example
=	assignment	0	a=b
+	addition	1	a+b
-	subtraction	1	a-b
$\star$	multiplication	2	a$\star$b
/	division	2	a/b
unary +	unary plus	3	+a
unary -	unary minus	3	-a
$\star\star$	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)
sin	sine of a (in radians), $\sin(a)$		sin(a)
cos	cosine of a (in radians), $\cos(a)$		cos(a)

4

MOPAC module

> In mathematics you don't understand things. You just get used to them.
>
> *John von Newmann*

This module was designed for reparameterizations of semiempirical Hamiltonians interfacing MOPAC, which may be useful for direct dynamics simulations of chemical reactions

The MOPAC interface, File 4.1, are based in three tools:

injector configure **GAFit** and create the files needed to run MOPAC.

extractor analyzes MOPAC output to extract and convert useful data –like heats of formation, Cartesian coordinates, etc– to a format suitable for **fitter** . Also controls if there is execution errors. All the extracted information are passed to **fitter**.

fitter evaluates the data and give out the results to **GAFit**.

File 4.1: External command to interface with MOPAC

```
#!/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="mopac.input"
export EXTERNAL_FIT="mopac.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
```

The interface has some features:

- It could be configured by *environment variables*. All tools have notice of them.
- All the *environment variables* have default values. File 4.2.

File 4.2: Minimal external command taking into account defaults

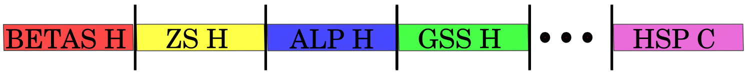
```
#!/bin/sh
export MOPAC_LICENSE=$HOME/mopac2009

export MOPAC_MOP="mopac_input.mop"

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

Three files are needed:

coefficients template: The *COEFS_TEMPLATE* are used by **injector** to build an external file with the semi empirical parameters fed by **GAFit**.

```
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
```

This is our chromosoma using the above coefficients template:



MOPAC template: The *MOPAC_TEMPLATE* are used by **injector** to create the input file for MOPAC where each @ will be replaced with a *COEFS_TEMPLATE* name.

```
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
```

In the example, there are three chained calculations (reactive optimization, frequencies calculation with the reactive optimized geometry and a transition state search).

bounds.txt: The **bounds.txt** file especifies the boundaries of the semiempirical parameters.

```
 Lower limit, upper limit, parameter type
     -5.5564      -6.791        9
      1.0692       1.306        9
      2.5940       3.170        9
     11.5632      14.132        9
    -46.8257     -57.231        9
    -35.6528     -43.575        9
    -14.1442     -17.287        9
     -6.9473      -8.491        9
      1.6277       1.989        9
      1.5166       1.853        9
      2.3834       2.913        9
     11.007       13.453        9
     10.323       12.617        9
      9.972       12.188        9
      8.856       10.824        9
      2.187        2.673        9
```

The type of extracted data are shown in the Table 4.1.

Table 4.1: 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
GRADIENTS	7	1	Gradients, x,y,z components per atom
NUMCONF	8	1	Number of states considered in one-electron excitations
DIPXYZ	9	4	Components x, y, z of the effect of dipole operator on states
EEL	10	3	Energies on states

This data is compared to reference values in the file *conditions.txt*. The fit is calculated, taking into account the weight, as:

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

fitter conditions

The conditions that can be used to compare are shown in Table 4.2 and established in the File 4.3.

File 4.3: conditions.txt

```
delt     1   2   100.6      0.1
frequency    2      15     3271.0    1e−4
distance     3       1        7      3.70    100.0
penalty  1e10
```

This interface could be used as a template or code guide to build other different module to face a new external program of interest.

However, in the MOPAC specific case, this approach presents some problems:

- If MOPAC fails, all the process tree could be hang and it is necessary to kill manually the problematic MOPAC process.
- If a file contains some calculations and one of this fail, all the rest fail or have no valid data.
- The design were to process many templates solely by one MOPAC process. But due the problems shown above, it only could process one template at time.

Table 4.2: 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*
gradient	4	calcA N value weight	Gradient number *N* of the calculation *calcA*. N varies from 1 to 3*NUMATOMS.
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*
dipx	4	calcA state value weight	Component *x* of the effect of dipole operator on *state* into calculation *calcA*
dipy	4	calcA state value weight	Component *y* of the effect of dipole operator on *state* into calculation *calcA*
dipz	4	calcA state value weight	Component *z* of the effect of dipole operator on *state* into calculation *calcA*
eel	5	calcA state order value weight	State energy into calculation *calcA*. *State*: 1 for singlet, 2 for doublet and 3 for triplet. *Order* is the order in the listing (eg. 1 for first singlet, 2 for second singlet and so on). If there are no data for this state, a **penalty** is applied.
penalty	1	penalty	Fit if any of the MOPAC calculations failed for a given coefficient vector. If not set, default value is 1.0e10.

4.1 Enhanced interface with MOPAC

To resolve the problems shown, an enhanced interface was developed with a new tool: **shepherd**.

shepherd can:

- Execute, control and maintain a optimal number of MOPAC processes near to the computing resources number –cpus, cores ...– if the output file will be written in a local storage resource.

This only works well using a local storage, not shared resources like NFS.

- Detect and kill hung MOPAC processes automatically.
- Create fake output files for the killed MOPAC processes.
- Gather all output files, including fakes, building a unique file to be processed by **extractor**.

The external command, simplified using default values, are shown in File 4.4.

File 4.4: Simplified external command to use with shepherd

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

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

To use this new interface, you only need replace the call to MOPAC with a call to **shepherd** in the *script*. Compare File 4.2 with File 4.4.

5 CHARMM module

This module interface **GAFit** with the CHARMM program in order to facilitate direct parameterizations of force fields.

6

mvariable module

GAFit can also be employed to fit a user-defined multivariable function to a set of data points using the *mvariable* module.

7

Simple configuration

> Any sufficiently advanced bug is indistinguishable from a feature.
>
> *Rich Kulawiec*

There are five simple configuration modules. The parameters and options for *simple configurations* are sumarized in Table 15.2.

The folder *simple-mod-examples* contains the examples from *SimplifiedUserGuide.pdf*.

7.1 Intermolecular simple configuration

File 7.1: Intermolecular job.txt file.

```

[Job]
Evaluations: 1000000
Application: intermolecular
Potential: 2
```

File 7.1 shows a simple configuration example.

7.2 Mopac simple configuration

Two MOPAC examples are developed in Sections 12 and 13. The interface is explained in Sections 21 and 22.

The module follows the MOPAC enhanced interface. The *shell script* needed for the interface is created on the fly by the module.

File 7.2: Mopac job.txt file.

```
[Job]
Evaluations: 100000
Application: MOPAC
Exec: /usr/programs/mopac/MOPAC2016.exe
```

File 7.2 shows a simple configuration example.

7.3 Charmm simple configuration

The CHARMM interface is explained in Section 25.

File 7.3: Charmm job.txt file.

```
[Job]
Evaluations: 50000
Application: CHARMM
Exec: /usr/programs/charmm/exec/gnu/charmm
Refgeom: geo-1.crd
Calculated energies: 1 3
```

7.4 Mvariable simple configuration

The **mvariable** module is explained in Section 26.

File 7.4: Mvariable job.txt file.

```
[Job]
evaluations: 100000
application: mvariable

...
```

7.5 Generic simple configuration

The **mvariable** module is explained in Section 27.

File 7.5: Generic job.txt file.

```
[job]
evaluations:    5000
application: generic
ncores: 1
executable: ./genericscript.sh
template: template
reference values: reference.values
```

Part II

Step by step examples

8 The examples

> 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 configuration, compilation and installation phases are done by the *GNU autotools* utilities:

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

The source tree from the distribution package –gafit-VERSION.tar.gz– are shown in Figure 8.1.

In this case, you can use the examples directly in their folder: There is a handy “make test” *makefile target* ready to run the example:

```
make test
```

If you install **GAFit**:

```
make install
```

the default installed tree is shown in Figure 8.2.

Once installed, take into account that:

$HOME/bin: where the binaries are installed.

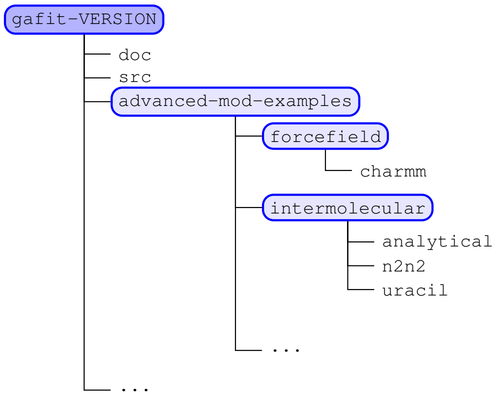


Figure 8.1: Source tree from distribution package, gafit-VERSION.tar.gz

$HOME/share: where the examples, documentation and other files are installed.

You can run the examples expanding the compressed *tar.gz* data file if present and running:

```
$HOME/bin/gafit
```

You also must copy in the folder other binaries needed from **$HOME-/bin** if you have **"."** included in your **PATH** variable, or better, set the environment variable **PATH** pointing to $HOME/bin:

```
export PATH=$PATH:$HOME/bin/gafit
```

The examples in this part are taken from the *advanced-mod-examples* folder.

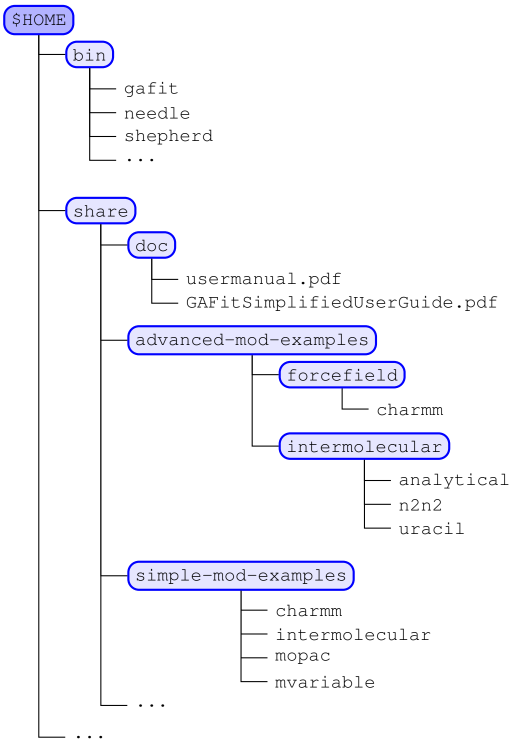
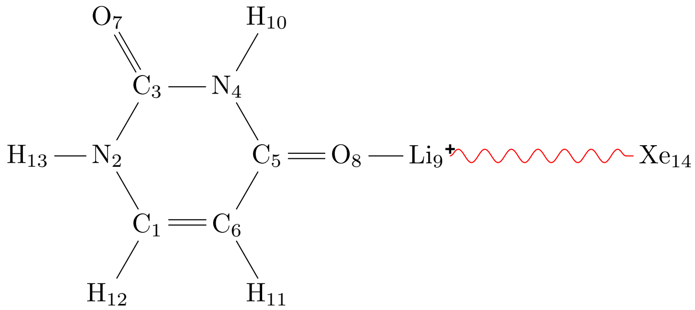


Figure 8.2: Installed tree into $HOME

9 Xe + (Li(Uracil))$^+$

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. In this example, we fit one of the potentials shown in Table 17.2 to the *interaction energies* between Xe and the [Li(Uracil)]$^+$ complex, computed by *ab initio* calculations.

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

```
$ tar -xvzf uracil.tgz
$ gafit > output.txt
```

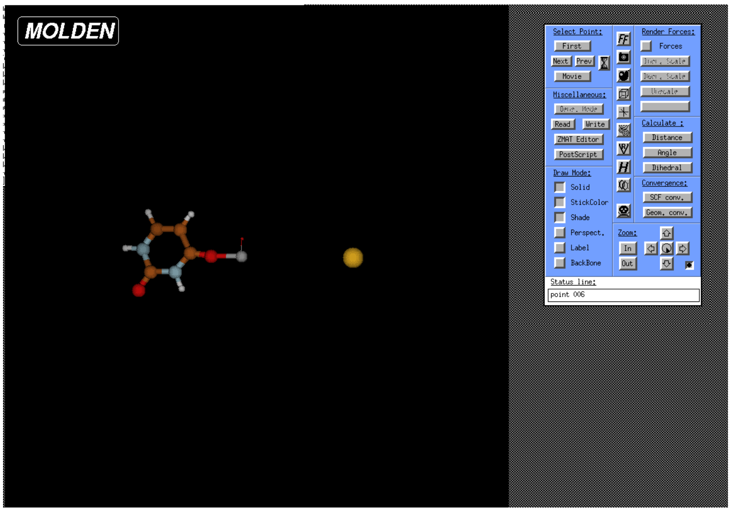
Once these commands are employed, some files are extracted and **GAFit** is run.

9.1 Preparing input files

The input file *coord.molden* contains the set of geometries employed in the *ab initio* calculations to obtain the *interaction energies*. The geometries can be viewed using molden (see Fig. 9.1):

```
$ molden coord.molden
```

Figure 9.1: Viewing the points with Molden.



Status line:
point 006

The very first lines of this file are shown in File 9.1.

File 9.1: 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 9.2). This file follows the specifications described in 17.1.

File 9.2: 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 3.4–.

```
$ 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 9.3– and *charges.txt*–File 9.4– files (see section 17.1). 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 9.3: 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 9.4: 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 17.2–.

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

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 9.5. The first and third coefficients for each interaction are real, and the second and fourth, integers.

File 9.5: bounds.txt file.

```
TEXT TEXT TEXT TEXT
         0.   1000000.       9
         0.       10.0       0
     -1500.         0.       9
        4.0        8.0       0
```

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 9.6[1].

File 9.6: 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]
```

[1] You safely can delete the entire **[parameters]** section. All lines are default values.

```
type: external auto
command: ./external-intpot.sh
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
test: 1488732015

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

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

The different sections and their possible options are discussed in section 15. In the **[job]** section we have **potential: 1** and **All coefficients: no**.

As you can see in Table 17.2, 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 9.5, 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, a real number or a real number with some fixed decimal places[2].

9.2 Running the example

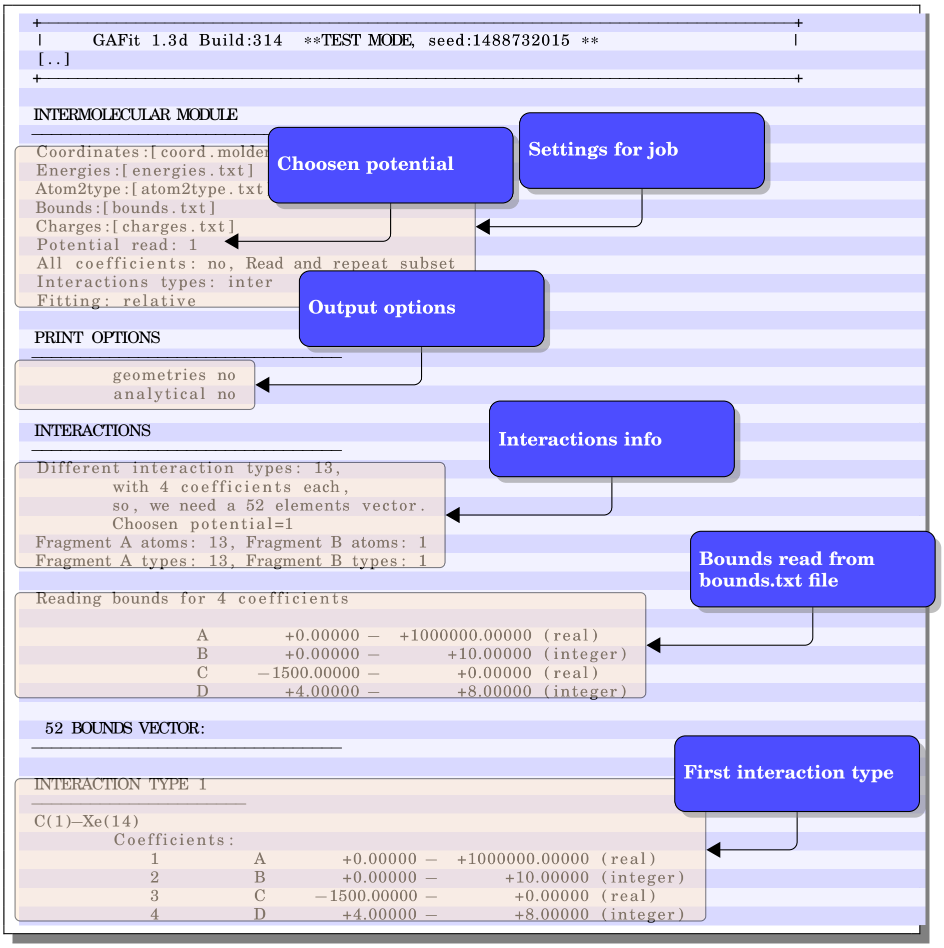
If you run **GAFit** from the folder where all the above files are located you get the output file shown in Files 9.7, 9.8, 9.9, 9.10 and 9.11.

```
$ 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 9.5.

[2] The few decimal places, the few the search domain. This speed up calculations

File 9.7: Uracil example output: output.txt (i)



```
+--------------------------------------------------------------------------------+
|      GAFit 1.3d Build:314   **TEST MODE,  seed:1488732015 **                   |
[..]
+--------------------------------------------------------------------------------+

INTERMOLECULAR MODULE
--------------------------------------
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
Interactions types: inter
Fitting: relative

PRINT OPTIONS
--------------------------------------
        geometries no
        analytical no

INTERACTIONS
--------------------------------------
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

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)

 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 9.8: 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:
              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)
```

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 9.9: Uracil example output: output.txt (iii)

```
INTERACTION TYPE 13
-----------------------------
H(13)-Xe(14)
           Coefficients:
               49          A         +0.00000 -   +1000000.00000 (real)
               50          B         +0.00000 -        +10.00000 (integer)
               51          C      -1500.00000 -         +0.00000 (real)
               52          D         +4.00000 -         +8.00000 (integer)
+--------------------------------------------------------------------------------------------+
|       Settings for GA                                                                      |
+--------------------------------------------------------------------------------------------+
|       runs:                1                                                               |
|       evaluations:         500                                                             |
|       population:          100                                                             |
|       crossover:           sbx                                                             |
|           cossover rate:       0.750000                                                    |
|           blx_alpha:           0.500000                                                    |
|           eta_sbx:             2.000000                                                    |
|       mutation:            sigma                                                           |
|           mutation rate:       0.100000                                                    |
|           sigma:               0.100000                                                    |
|       integer mutation: random                                                            |
|       elitism:             yes                                                             |
|       tournament size:     5                                                               |
|       direction:           min                                                             |
+--------------------------------------------------------------------------------------------+
|       Settings for job                                                                     |
+--------------------------------------------------------------------------------------------+
|       Command:[./external-intpot.sh]                                                       |
|       Bounds:[bounds.txt.internal]                                                         |
|       External input:[intpot.input]                                                        |
|       External fit:[intpot.fit]                                                            |
|       Total coefficients: 52                                                               |
|       Print options: runs yes, ga settings yes                                             |
+--------------------------------------------------------------------------------------------+
|       run: 1                                                                               |
|       TEST MODE seed: 1488732015                                                           |
+--------------------------------------------------------------------------------------------+

Eval.                Best fit.
----------------------------------------------
100                  22.5565
300                  19.0732
500                  7.59589
500                  7.59589


#
#Results
#

INTERACTION TYPE 1
-----------------------------
C(1)-Xe(14)
           Coefficients:
                1 A        +74540.0217736325
                2 B             +5.0000000000
                3 C          -845.7260791565
                4 D             +8.0000000000

INTERACTION TYPE 2
-----------------------------
N(2)-Xe(14)
           Coefficients:
                5 A       +543556.0785643021
                6 B            +10.0000000000
                7 C          -805.1226735632
                8 D             +5.0000000000
```

Used random seed

Here begins results

Second interaction type results

The calculations employ random numbers, so if you take the same seed used in a given run, you will reproduce the whole output. You must activate the option **print runs** to view it in output. In any case, you can retrieve the seed in the file *stats.txt*. The details are in Section 15.2 and it is useful for testing and debugging purposes. Each interaction with the coefficients found are printed. This is the information saved in the file *best.txt*.

File 9.10: Uracil example output: output.txt (iv)

```
INTERACTION TYPE 3
------------------------------
C(3)-Xe(14)
          Coefficients:
               9 A    +501043.7557968706
              10 B         +10.0000000000
              11 C       -1155.6351484105
              12 D          +8.0000000000

INTERACTION TYPE 4
------------------------------
N(4)-Xe(14)
          Coefficients:
              13 A    +158499.3083434265
              14 B          +5.0000000000
              15 C       -1350.4319505465
              16 D          +8.0000000000

INTERACTION TYPE 5
------------------------------
C(5)-Xe(14)
          Coefficients:
              17 A    +430213.6165002815
              18 B          +2.0000000000
              19 C        -465.2443965131
              20 D          +4.0000000000

INTERACTION TYPE 6
------------------------------
C(6)-Xe(14)
          Coefficients:
              21 A     +26522.1955712938
              22 B          +4.0000000000
              23 C        -771.4668786474
              24 D          +6.0000000000

INTERACTION TYPE 7
------------------------------
O(7)-Xe(14)
          Coefficients:
              25 A    +987791.4463235690
              26 B          +3.0000000000
              27 C        -868.7300231100
              28 D          +4.0000000000

INTERACTION TYPE 8
------------------------------
O(8)-Xe(14)
          Coefficients:
              29 A    +496941.4962879005
              30 B          +5.0000000000
              31 C       -1262.0250218901
              32 D          +8.0000000000

INTERACTION TYPE 9
------------------------------
 Li(9)-Xe(14)
          Coefficients:
              33 A    +882006.9378573116
              34 B          +5.0000000000
              35 C       -1270.6122879899
              36 D          +5.0000000000

INTERACTION TYPE 10
------------------------------
H(10)-Xe(14)
          Coefficients:
              37 A    +302068.4610048971
              38 B          +8.0000000000
              39 C       -1051.2509426219
              40 D          +7.0000000000

INTERACTION TYPE 11
------------------------------
H(11)-Xe(14)
          Coefficients:
              41 A    +155814.6940483026
              42 B         +10.0000000000
              43 C        -324.8565324163
              44 D          +5.0000000000

INTERACTION TYPE 12
------------------------------
```

```
H(12)-Xe(14)
           Coefficients:
               45 A      +52853.6256781471
               46 B           +7.0000000000
               47 C        -1500.0000000000
               48 D           +7.0000000000
```

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 9.11: Uracil example output: output.txt (v)

```
INTERACTION TYPE 13
_____________________________
H(13)-Xe(14)
           Coefficients:
               49 A     +950049.8248932150
               50 B           +9.0000000000
               51 C         -787.5269561135
               52 D           +7.0000000000
#
#Evaluation
#
#Geometry       Energy            Calculated          Difference         Weight
#========       ======            ==========          ==========         ======
       1    -0.006436000000   -0.011050577330   +71.70 %           +1.00
       2    -0.012603000000   -0.017997557493   +42.80 %           +1.00
       3    -0.024660000000   -0.031867370776   +29.23 %           +1.00
       4    -0.053662000000   -0.063991213263   +19.25 %           +1.00
       5    -0.151027000000   -0.158047898351    +4.65 %           +1.00
       6    -0.208324000000   -0.207739342432    -0.28 %           +1.00
       7    -0.298249000000   -0.279542061446    -6.27 %           +1.00
       8    -0.443987000000   -0.385691828283   -13.13 %           +1.00
       9    -0.576097000000   -0.473617455187   -17.79 %           +1.00
      10    -0.762092000000   -0.586861719042   -22.99 %           +1.00
      11    -1.031527000000   -0.733570004510   -28.89 %           +1.00
      12    -1.431174000000   -0.924749242688   -35.39 %           +1.00
      13    -2.022694000000   -1.175614113636   -41.88 %           +1.00
      14    -2.554913000000   -1.386332740552   -45.74 %           +1.00
      15    -3.208230000000   -1.641405665602   -48.84 %           +1.00
      16    -3.966854000000   -1.950496945548   -50.83 %           +1.00
      17    -4.767595000000   -2.321014222875   -51.32 %           +1.00
      18    -5.448579000000   -2.744231792499   -49.63 %           +1.00
      19    -5.645469000000   -2.958529426817   -47.59 %           +1.00
      20    -5.658691000000   -3.149037786327   -44.35 %           +1.00
      21    -5.387761000000   -3.274427021287   -39.22 %           +1.00
      22    -4.692701000000   -3.259569971701   -30.54 %           +1.00
      23    -3.377588000000   -2.971079070471   -12.04 %           +1.00
      24    -1.167944000000   -2.176024998738   +86.31 %           +1.00
      25    +2.322455000000   -0.472650935654   -120.35 %           +1.00
      26    +7.633202000000   +2.825651217629   -62.98 %           +1.00
      27    +15.516838000000   +8.883658156418   -42.75 %           +1.00
      28    +27.007602000000   +19.665205957558   -27.19 %           +1.00
      29    +66.979582000000   +70.789210814611    +5.69 %           +1.00
      30    +146.056144000000   +218.641328594868   +49.70 %           +1.00
      31    +297.072019000000   +634.044445712745   +113.43 %           +1.00
```

Geometry fit evaluation

9.3 Examining results

The best individual from the program run is stored in the file *best.txt* –File 9.13–. 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 Section 3.10.

The file *best.txt* is overwritten in each run

File 9.12: 2body-type-1.plt

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

Files 9.12 and 9.14 are the **gnuplot** commands and data file, repectivelly, to plot *Potential* vs *r* for the *interaction type 2* between C(1) and Xe(14), Figure 9.2.

File 9.13: Uracil example best.txt

```
901608.806630330742
4.000000000000
-6.430323296743
5.000000000000
165595.864979834671
7.000000000000
-1138.239454060825
5.000000000000
565031.244248823496
5.000000000000
-215.144199774099
8.000000000000
462307.829517660779
8.000000000000
-70.773260147771
8.000000000000
662752.755474972306
2.000000000000
-311.802009422465
4.000000000000
819468.319292194792
10.000000000000
-1378.714903828626
5.000000000000
702730.873476595385
6.000000000000
-1068.294837888685
8.000000000000
270196.896241575596
9.000000000000
-1426.856074983300
5.000000000000
868175.125098152435
5.000000000000
-1499.764070025773
5.000000000000
321195.104213012499
9.000000000000
-293.802737729408
5.000000000000
211372.727365899016
3.000000000000
-562.671412678537
5.000000000000
93914.834234193142
6.000000000000
-9.016924216977
5.000000000000
520130.035980527289
2.000000000000
-886.907942598062
8.000000000000

Fitness: 4.53655
```

A, B, C, D for interaction type 1, C(1)-Xe(14)

Result from evaluate this coefficients set

In File 9.12 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:

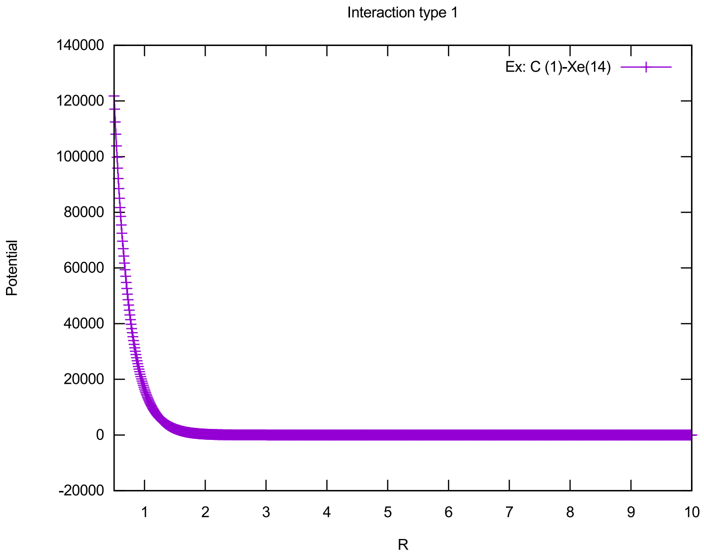
```
$ gnuplot 2body-type-1.plt
```

to obtain a *svg graphic file* named *plot.svg* like Figure 9.2.

File 9.14: 2body-type-1.dat

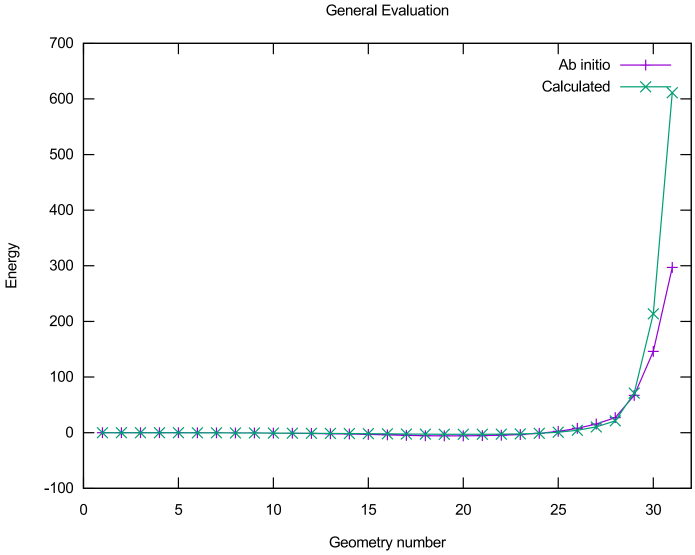
```
#
#INTERACTION TYPE 1
#———————————————
# C(1)—Xe(14)
#         Coefficients:
#              1 A    +901608.8066303307
#              2 B          +4.0000000000
#              3 C          −6.4303232967
#              4 D          +5.0000000000
#
#                             r                  V
                     +0.5000000000       +121813.7128684444
                     +0.5100000000       +117048.6583011130
                     +0.5200000000       +112469.0513077077
                     +0.5300000000       +108067.8097677045
                     +0.5400000000       +103838.1009140337
                     +0.5500000000        +99773.3361867638
                     +0.5600000000        +95867.1654688832
                     +0.5700000000        +92113.4708844208
 [...]
```

Figure 9.2: Interaction type 1 plot.



One of the plots produced by **fitview** is the evaluation of the fit, that can help you to adjust the geometry weights, Figure 9.3.

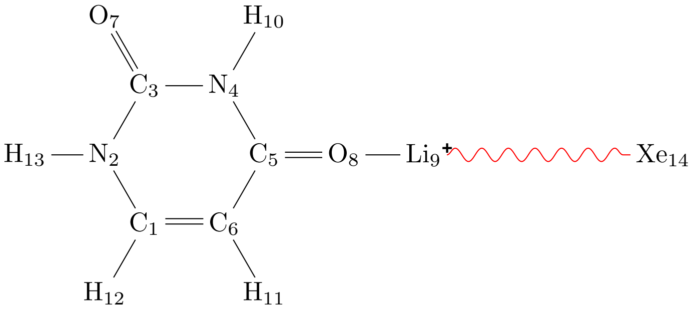
Figure 9.3: General evaluation plot.



10 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 [Roberto Rodriguez-Fernandez, Saulo A. Vazquez, and Emilio Martinez-Nunez. "Collision-induced dissociation mechanisms of [Li(uracil)]+". In: *Phys. Chem. Chem. Phys.* 15 (20 2013), pp. 7628–7637. DOI: 10.1039/C3CP50564B].

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.

10.1 Preparing input files

The files for this example are in the folder *intermolecular/analytical*. The input files are the same than the previous one, except for the *job.txt* file –File 10.1[1]– which is the unique file to modify.

File 10.1: 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]
type: external auto
command: ./external-intpot.sh
runs:             1
evaluations:      500
Geometries:    coord
Energies:   energies
Atom2type: atom2typ
Bounds: bounds.txt
Charges: charges.txt
Potential: 0
All coefficients: no
fitting: relative
test: 1488732015

[print]
geometries: no
runs: no

[analytical]
expression: potential 3
distance: dist
potential: pot
coefficients: a, b, 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]
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 17.2

Potential type, according to Table 17.2, 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 17.2–. *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*.

[1] As in previous section, you safely can delete the entire **[parameters] section.**

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

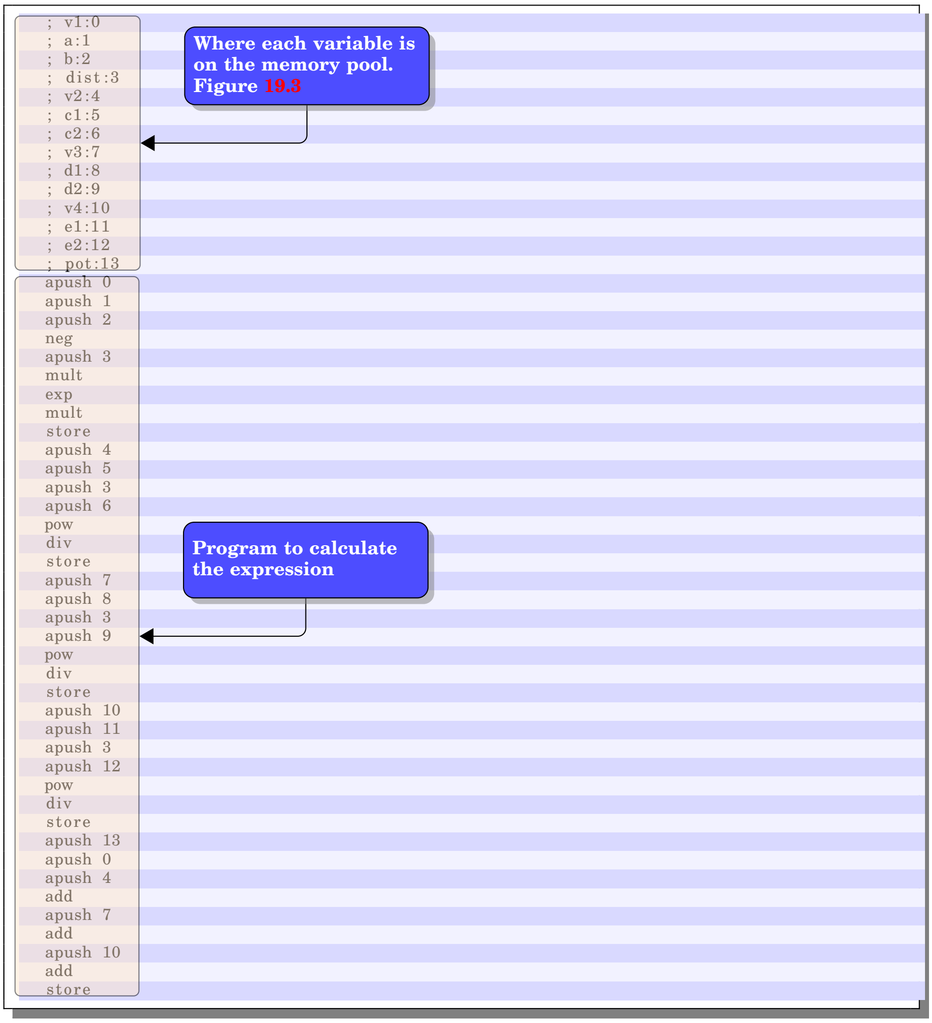
You can test the *job.txt* file using **ufpu** –section 20.3– and type some values to **distance** and **coefficients** and check the calculated **potential**.

```
$ ufpu
uFpu v0.2 (c)GAFit toolkit - 2013

expression name: "potential 3"
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
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 b=1 dist=1 v2=1 c1=1 c2=1 v3=1 d1=1
d2=1 v4=1 e1=1 e2=1 pot=3.36788

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 10.2.

The resulting *job.txt* is shown in File 10.3 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 10.2: Asembler bytecode produced



```
; 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
```

Where each variable is on the memory pool. Figure 19.3

Program to calculate the expression

File 10.3: Analytical expression job

```
[job]
type: external auto
command: ./external-intpot.sh
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 10.4: Analytical expression job output

```
[...]

INTERMOLECULAR MODULE
------------------------------------
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
Interactions types: inter
Fitting: relative

PRINT OPTIONS
------------------------------------
        geometries no
        analytical yes

Analytical expression
------------------------------------
expression name: "potential 3"
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
[...]
```

10.2 Running and examining results

The output is similar to the previous one –section 9–, except for the potential. Here we use the number 3 from Table 17.2 but coded as an *analytical expression*.

11 External Interface

> 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 building a new module to interface with.

11.1 Input files

Whe have some $(x, f(x))$ pair values shown in Table 11.1 to fit to a polynomial of fifth degree. These value pairs are in the input File 11.1.

File 11.1: *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 11.1. Also, we need a *bounds.txt* file to fix upper and lower limits as the included example in File 11.2. In this case, we want integer values, so the righmost column is set to 0.

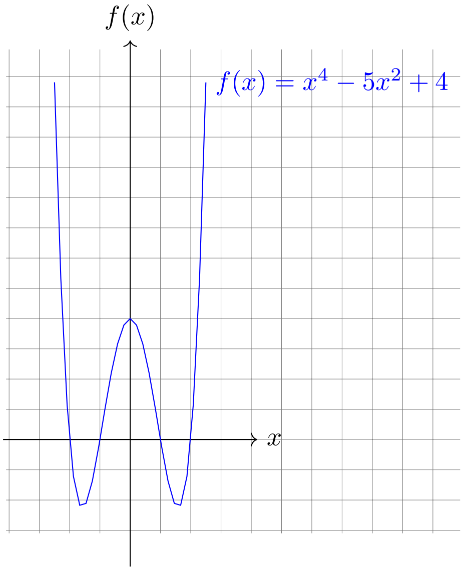
File 11.2: *bounds.txt* file

```
TEXT TEXT TEXT TEXT
        −10.        10.        0
        −10.        10.0       0
        −10.        10.        0
        −10.        10.0       0
        −10.        10.        0
```

Table 11.1: Example values to fit.

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

Figure 11.1: Example polynomial plot

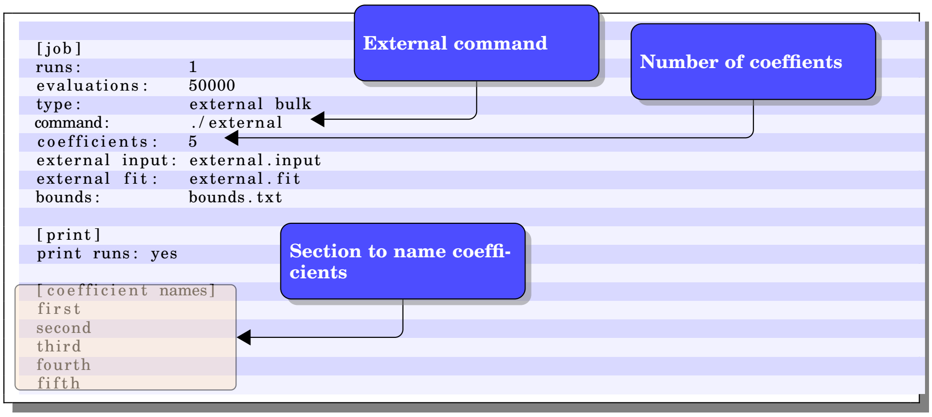


An example is provided in File 11.4. This code inputs the coefficients values provided by **GAFit** and the external known values –like the Table 11.1, File 11.1– to calculate a fit to a generic polynomial of degree n:

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

The given test code supports both *external* and *external bulk* options (Section 21.1): 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 11.3 –the *job.txt* file– the type of job.

File 11.3: External example job.txt: fitting a polynomial



```
[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
```

External command

Number of coeffients

Section to name coefficients

In the configuration file –*job.txt*, File 11.3– 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.

The function to adjust is defined **double func(double x, double a[], int n)** at lines 13-14, File 11.4. In this case is a polynomial of degree n. You can use this code to adjust a different function.

File 11.4: external.c

```
 1 /*
 2  (c)GAFit toolkit $Id: external.c 378 2019-12-04 17:52:09Z 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;
21 }
22
23 int
24 main (void)
25 {
26   char line[MAXLINE + 1];
27   double *coef = NULL;
28   double *valuesx = NULL, *valuesy = NULL;
29   double fit, number0, number1, tmp, div;
30   int i, j, ncoefs, mvalues, tcoefs;
31   int first, ok;
32
33   FILE *f, *out;
```

```c


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);
}
```

11.2 Running the example and examining results

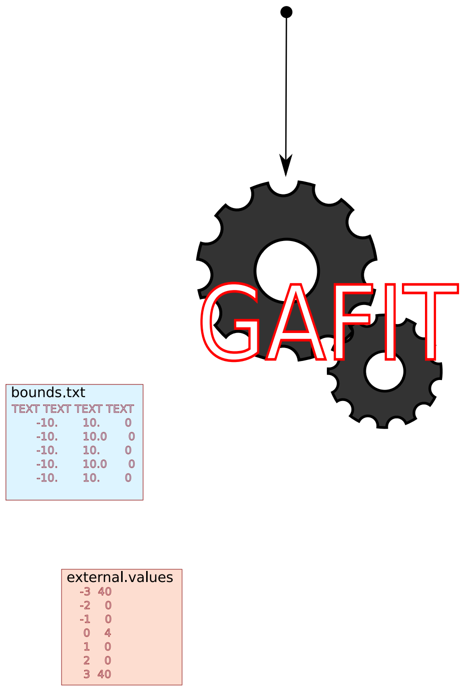
To create the needed files and run the test you only have to type in the **GAFit**'s examples folder –see 52–:

```
$ cd miscellaneuous/external
$ make external
$ gafit > output.txt
```

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*.

Figure 11.2: **Step 1** : **GAFit** is launched



```
TEXT TEXT TEXT TEXT
     -10.     10.     0
     -10.     10.0    0
     -10.     10.     0
     -10.     10.0    0
     -10.     10.     0
```

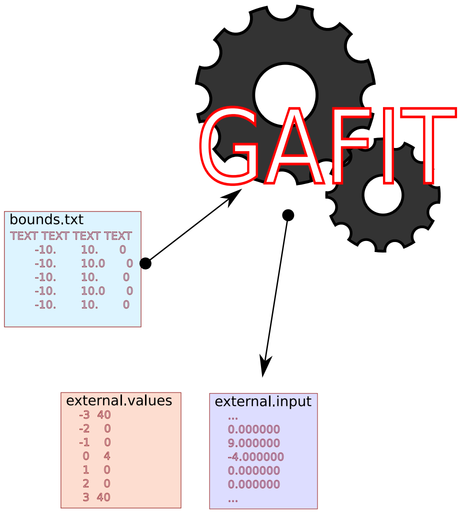
```
 -3  40
 -2   0
 -1   0
  0   4
  1   0
  2   0
  3  40
```

Step 2 **GAFit** writes a whole population of coefficients to be evaluated in the *external.input* file –File 11.5– using as upper and lower bounds those specified in the file *bounds.txt* –File 11.2–. If the file *external.input* 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 11.3.

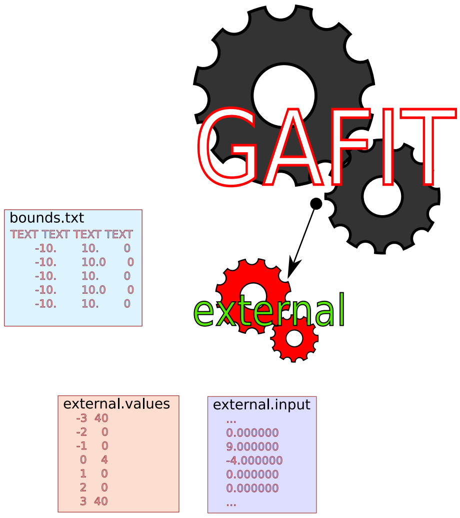
The coefficients must be integers –*bounds.txt* last column set to 1–.

Figure 11.3: **Step 2** : **GAFit** overwrites or creates the *external.input* file.



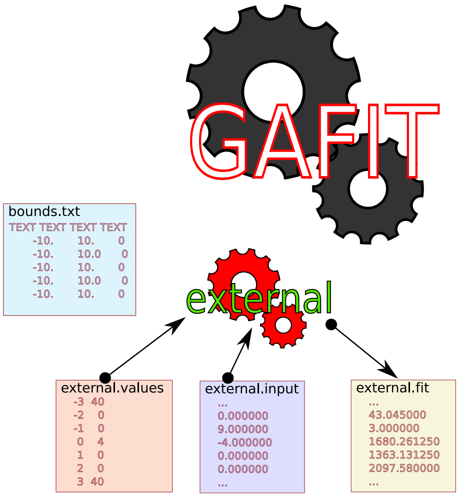
Step 3 **GAFit** launches the **external** binary.

Figure 11.4: **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 11.6–.

Figure 11.5: Step 5 : **external** using *external.input* evaluates the *external.values* and overwrites or creates the *external.fit* file



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 11.6, the 13^{th} value is worse than 6^{th}.

The n^{th} value, (0, 9, -4, 0, 0), from File 11.5 represents the polynomial:

$$p(x) = 0x^4 + 0x^3 - 4x^2 + 9x + 0$$

Table 11.2: n^{th} set of coefficients fit.

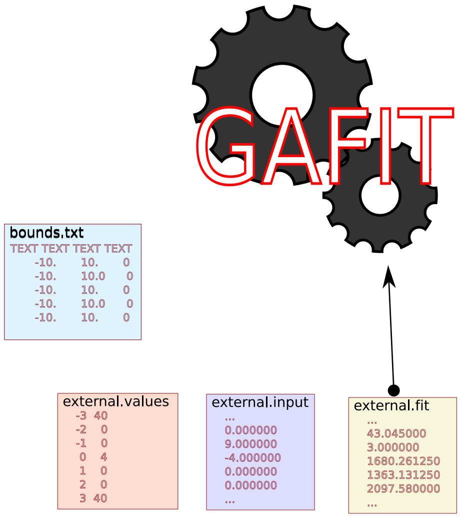
x	$f(x)$	$p(x) = -4x^2 + 9x$	$\frac{[p(x)-f(x)]^2}{f(x)^2}$
-3	40	-63	6.630625
-2	0	-34	1156.000000
-1	0	-13	169.000000
0	4	0	1.00000
1	0	5	25.00000
2	0	2	4.00000
3	40	-9	1.500625
		$\sum \frac{[p(x)-f(x)]^2}{f(x)^2}$	1363.131250

The calculations are shown in Table 11.2 for the n^{th} coefficients set: Files 11.5 and 11.6.

Note that, in the *external.c* program, File 11.4, 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.

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 11.7. Note that this file always

Figure 11.6: **Step 5** : **GAFit** reads the *external.fit* file



```
bounds.txt
TEXT TEXT TEXT TEXT
    -10.    10.     0
    -10.    10.0    0
    -10.    10.     0
    -10.    10.0    0
    -10.    10.     0
```

```
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
  ...
```

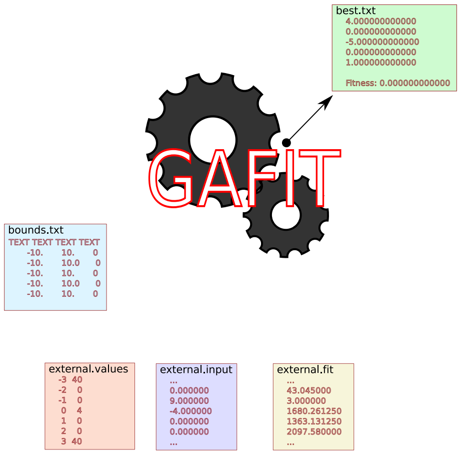
```
external.fit
  ...
  43.045000
  3.000000
  1680.261250
  1363.131250
  2097.580000
  ...
```

will be overwritten: If you have some fit to save, copy it out there or rename it.

The values shown represent the polynomial:

$$f(x) = x^4 - 5x^2 + 4$$

Figure 11.7: **Step 6** : if the fit is the best till now, **GAFit** overwrites or creates the *best.txt* file



```
best.txt
  4.000000000000
  0.000000000000
  -5.000000000000
  0.000000000000
  1.000000000000

  Fitness: 0.000000000000
```

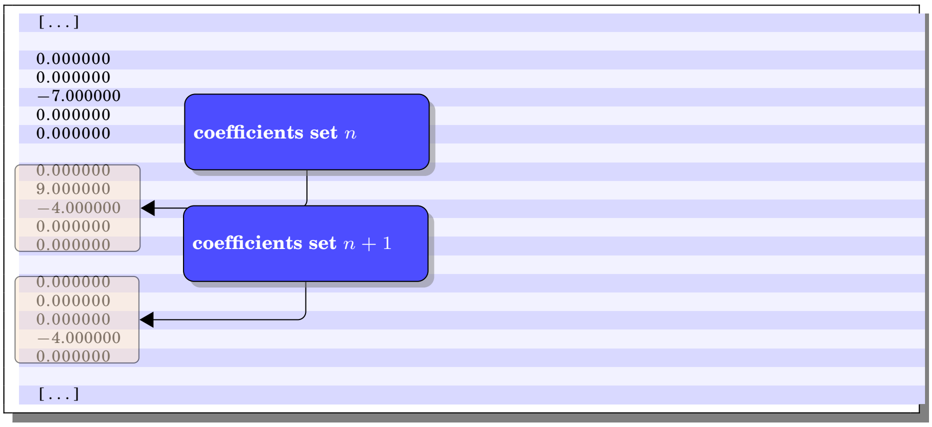
```
bounds.txt
TEXT TEXT TEXT TEXT
    -10.    10.     0
    -10.    10.0    0
    -10.    10.     0
    -10.    10.0    0
    -10.    10.     0
```

```
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
  ...
```

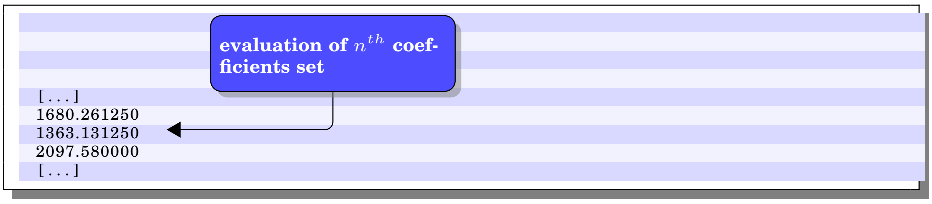
```
external.fit
  ...
  43.045000
  3.000000
  1680.261250
  1363.131250
  2097.580000
  ...
```

File 11.5: external.input file



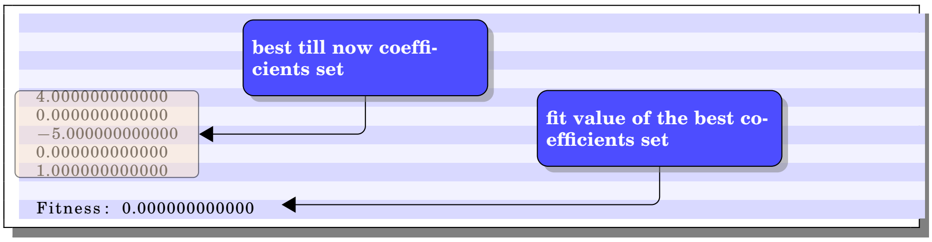
```
[...]

0.000000
0.000000
−7.000000
0.000000
0.000000

0.000000
9.000000
−4.000000
0.000000
0.000000

0.000000
0.000000
0.000000
−4.000000
0.000000

[...]
```

File 11.6: external.fit file



```




[...]
1680.261250
1363.131250
2097.580000
[...]
```

File 11.7: best.txt

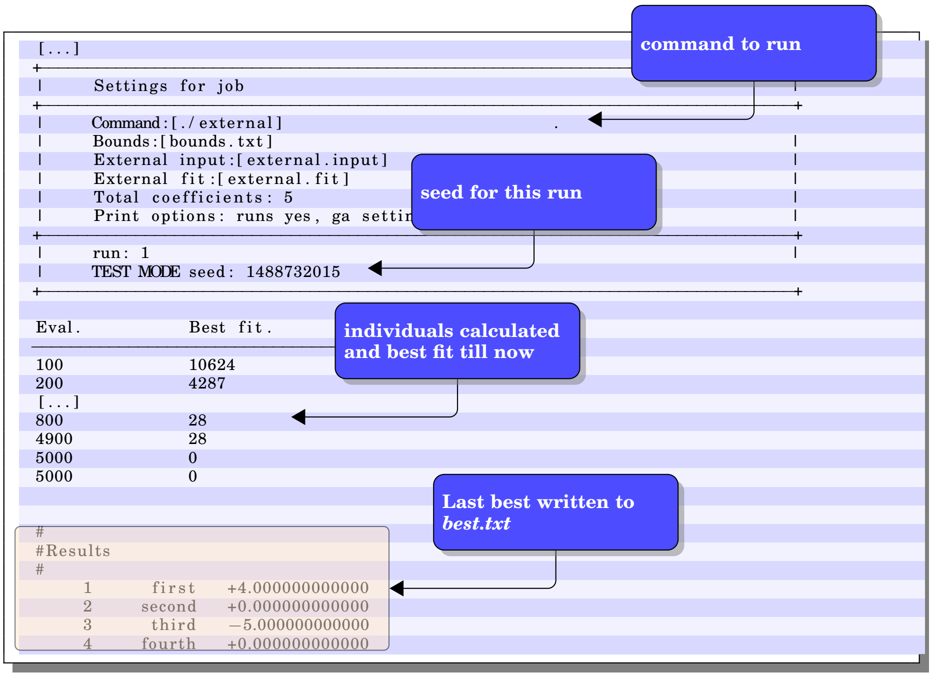

best till now coefficients set
fit value of the best coefficients set

```




4.000000000000
0.000000000000
−5.000000000000
0.000000000000
1.000000000000

Fitness: 0.000000000000
```

The output of the whole process is sumarized in File 11.8.

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 15.2–.

File 11.8: external.output



```
[...]
+------------------------------------------------------------
|      Settings for job                                     |
+-------------------------------------------------------------+
|      Command:[./external]                          .
|      Bounds:[bounds.txt]                                   |
|      External input:[external.input]                       |
|      External fit:[external.fit]                           |
|      Total coefficients: 5                                 |
|      Print options: runs yes, ga settin                    |
+-------------------------------------------------------------+
|      run: 1                                                |
|      TEST MODE seed: 1488732015
+-------------------------------------------------------------+

Eval.            Best fit.
-------------------------------
100              10624
200              4287
[...]
800              28
4900             28
5000             0
5000             0


#
#Results
#
     1      first   +4.000000000000
     2     second   +0.000000000000
     3      third   −5.000000000000
     4     fourth   +0.000000000000
```

command to run

seed for this run

individuals calculated and best fit till now

Last best written to *best.txt*

More information on this subject on 21.1. 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.

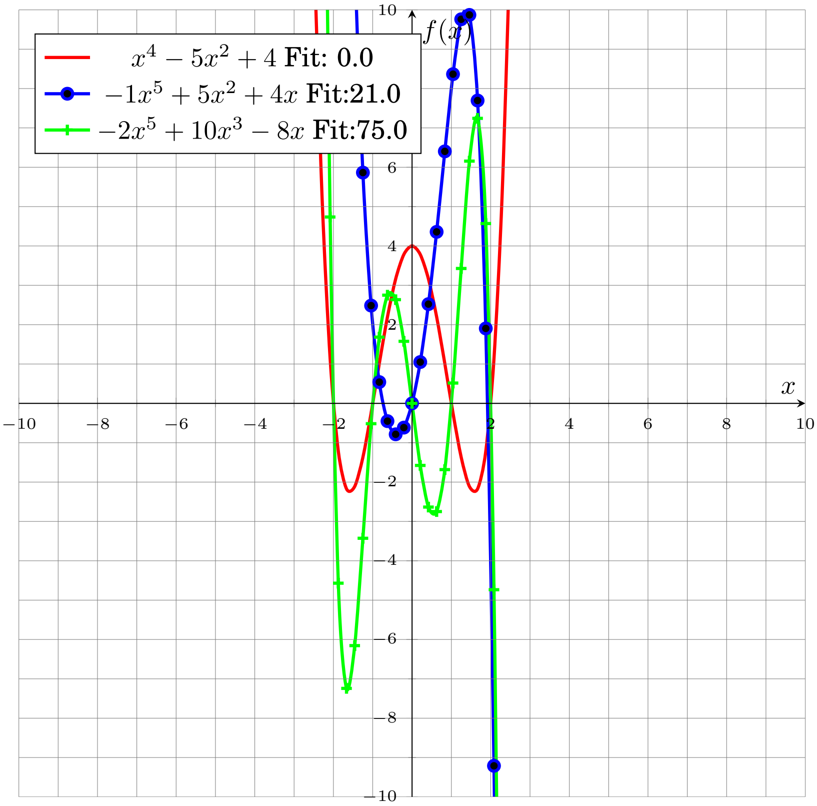
There are distinct results from each run, because the GA explores all the space limited by the bounds and by the type of coefficients: only integers. Some results are shown in Table 11.3 and plotted in Figure 11.8.

Table 11.3: 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 11.8: Table 11.3 polynomial plots.



12 MOPAC Interface

> 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 –taken from [Zahra Homayoon, Saulo A. Vázquez, Roberto Rodríguez-Fernández, and Emilio Martínez-Núñez. "Ab Initio and RRKM Study of the HCN/HNC Elimination Channels from Vinyl Cyanide". In: *The Journal of Physical Chemistry A* 115.6 (2011). PMID: 21261315, pp. 979–985. DOI: 10.1021/jp109843a]–.

12.1 Prerequisites

You must have installed MOPAC in your system –MOPAC 2009, 2012 or 2016–. You must know where it is installed or which is the value of the MO PAC_LICENSE shell variable, to set correctly **external-mopac.sh** –File 12.6–.

12.2 Input and executable files

The complete interface was explained in the Section 21. To create and run the example you must type:

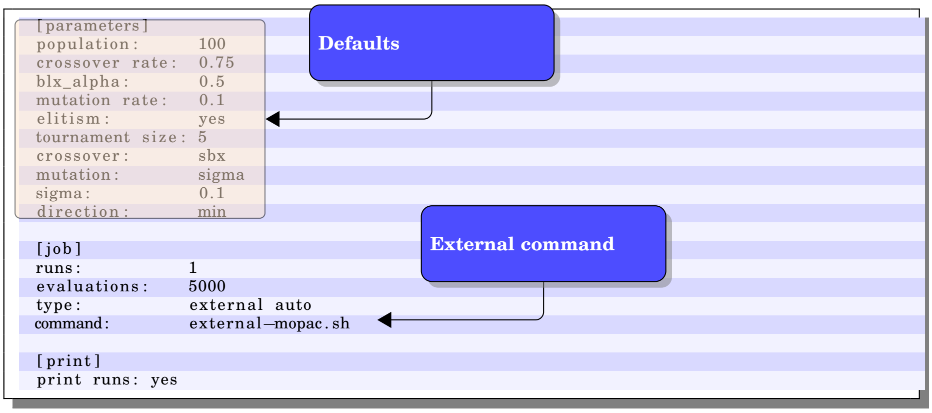
```
$ cd mopac/mopac
$ tar xvzf mopac.tgz
$ gafit > output.txt
```

Some files are extracted from the compressed data file and the example is run. Check that your environment variable **PATH** contains the folder were the **GAFit**'s executables are installed[1].

Table 12.1: Files in the mopac-example folder after uncompress the *mopac.tgz* file.

File	Type
bounds.txt	text file
conditions.txt	text file
external-mopac.sh	shell script
job.txt	job configuration
template.coefs	mopac external coefficients
template.mop	mopac job template

File 12.1: External example *job.txt*: fitting MOPAC 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-mopac.sh

[print]
print runs: yes
```

As shown in File 12.1, the job is declared as **external auto**, so the external scripts and/or binaries must configure the system by themselves.

File 12.2: MOPAC 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
```

[1] default value: $HOME/bin

The objective is to obtain a suitable combination of coefficients, File 12.3, to satisfy the constrains declared in File 12.5 using the MOPAC 2009 task shown in File 12.4 where the @ symbol will be replaced by the name of a copy of File 12.3 where the coefficients are generated by **GAFit** between some limits expressed in the File 12.2.

Note that these randomly generated coefficients are prone to err and could crash MOPAC.

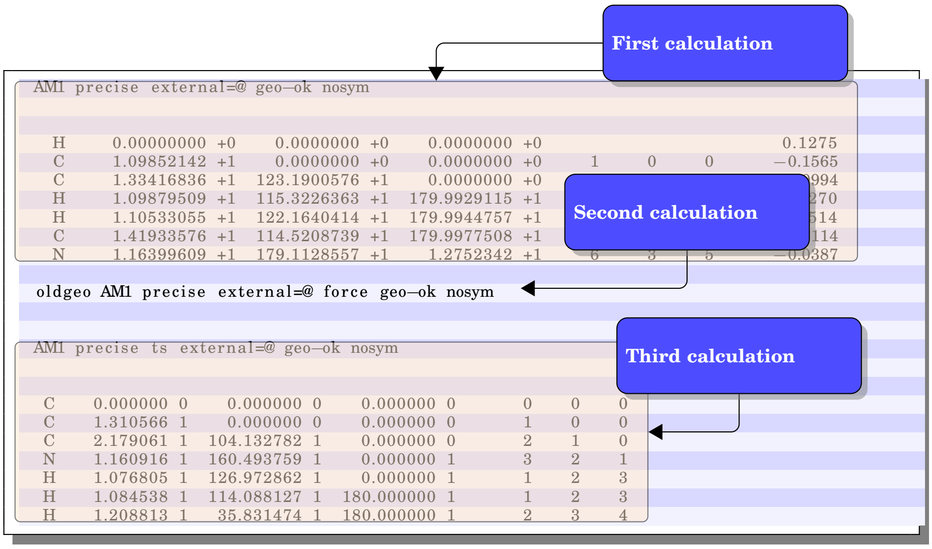
File 12.3: MOPAC 2009 coefficients to fit. *template.coefs* file

```
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 12.3 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 21.2 and explained in section 21.4. Here, the File 12.7 is used to pass to **GAFit** the configuration.

File 12.4: MOPAC 2009 task. *template.mop* file

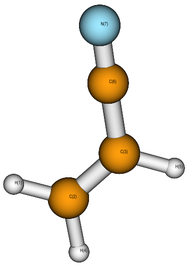


```
AM1  precise  external=@ geo—ok  nosym


  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                            994
  H      1.09879509  +1    115.3226363  +1    179.9929115  +1                            270
  H      1.10533055  +1    122.1640414  +1    179.9944757  +1                            514
  C      1.41933576  +1    114.5208739  +1    179.9977508  +1                            114
  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 12.4 we have three calculations:

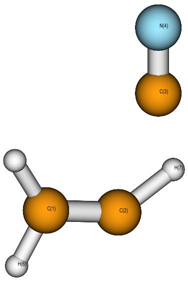
- The first one, an Austin Model 1 (AM1) geometry optimization of the vinyl cyanide. Figure 12.1.

Figure 12.1: Vinyl cyanide drawn using the coordinates of the first calculation (optimization of the minimum energy structure).



- 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 12.2.

Figure 12.2: 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 12.5: 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 21.6. 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:

$$\textbf{fit} = \begin{cases} \begin{array}{r} \textit{if} \\ \textit{calculation} \\ \textit{is} \\ \textit{done:} \end{array} & \begin{cases} \left[100.6 - (\textbf{HEAT}_{[\textbf{1st calculation}]} - \textbf{HEAT}_{[\textbf{3rd calculation}]})\right]^2 * 0.1 \\ \qquad + \\ \left[3271.0 - \textbf{FREQUENCY}_{[\textbf{number 15 from 2nd calculation}]}\right]^2 * 1e^{-4} \\ \qquad + \\ \left[3.700309096 - \textbf{DISTANCE}_{[\textbf{atoms 3-1 from 3rd calculation}]}\right]^2 * 100. \end{cases} \\ \textit{if calculation fails:}\ 1e10 & \end{cases}$$

GAFit shall run to minimize the fit.

12.3 Running the example and examining results

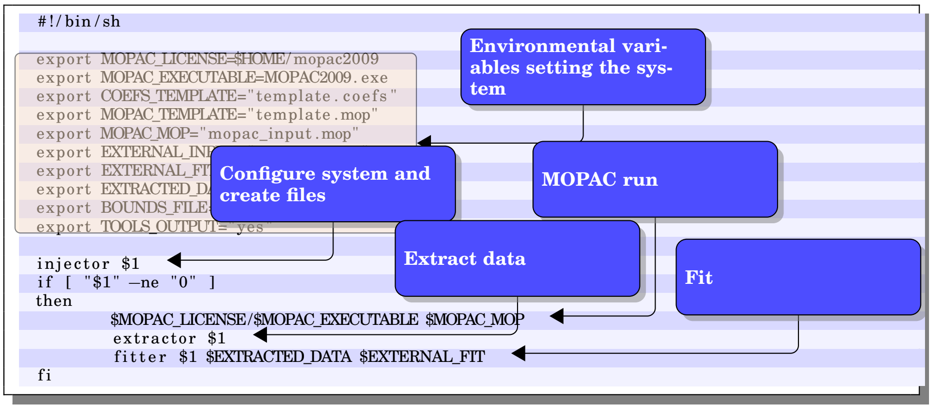
The file *external-mopac.sh* performs all the above operations, as shown in File 12.1.

To run the example, type:

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

The external program provided is shown in File 12.6. The operation mode is similar but slightly more complicated than 11. These are the steps:

File 12.6: **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="mopac_input.mop"
export EXTERNAL_INP
export EXTERNAL_FIT
export EXTRACTED_DA
export BOUNDS_FILE=
export TOOLS_OUTPUT="yes"

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
MOPAC run
Extract data
Fit

Step 1 **GAFit** runs the external program to configure the system as:

```
external-mopac.sh 0
```

A file with the configuration is generated by running **injector 0** in turn. This file is shown in File 12.7. All the options are taken from the environment variables set in File 12.6.

File 12.7: **external auto**: *response* file

```
[job]
type: external
coefficients: 16
external input: mopac.input
external fit: mopac.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 12.7 configures itself.

Step 3 **GAFit** creates a whole population of individuals. Each individual is a coefficient set.

Step 4 **GAFit** writes the file *mopac.input* with one set of coefficients –or a whole population, depending upon configuration–. File 12.8.

File 12.8: *mopac.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-mopac.sh 16
```

Step 6 **external-mopac.sh** launches **injector 16** which create the needed files to run the MOPAC 2009 task:

- *mopac_input.mop*, a copy of File 12.4 where the @ is replaced to point the file below –File 12.8–.
- a copy of File 12.8.

Step 7 **external-mopac.sh** launches MOPAC 2009 on *mopac_input.mop*, as input file, running the task with *mopac_input.out* as output: File 12.9, where near most the lines are omitted and the three individual task are shown.

File 12.9: *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-mopac.sh** launches **extractor** which extracts data from the mopac 2009 output –*mopac_input.out*– writing it to *extracted.data*, File 12.10.

File 12.10: *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 21.5.

Step 9 **external-mopac.sh** launches **fitter** which using the *extracted.data* file evaluate the coefficients –File 12.11– writing the result to *mopac.fit* –File 12.12–.

File 12.11: 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
```

To see this output you need to set the environmental variable TOOLS_OUTPUT to “yes”

Step 10 **external-mopac.sh** finishes, and control returns to **GAFit** which apply the *mopac.fit* values to genetic selection.

File 12.12: *mopac.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 12.13. 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 *mopac.input* and run the external script *external-mopac.sh* with **1** as its argument as shown below:

Don’t forget to set TOOLS OUTPUT to “yes”

```
$ cp best.txt mopac.input
$ ./external-mopac.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 12.13: **GAFit** output

```
[...]
+------------------------------------------------------------------------+
|     Settings for job                                                   |
+------------------------------------------------------------------------+
|     Command:[./external-mopac.sh]                                      |
|     Bounds:[bounds.txt]                                                |
|     External input:[mopac2009.input]                                   |
|     External fit:[mopac2009.fit]                                       |
|     Total coefficients: 16                                             |
|     Print options: runs yes, ga settings no                            |
+------------------------------------------------------------------------+
|     run: 1                                                             |
|     TEST MODE seed: 1488732015                                         |
+------------------------------------------------------------------------+

Eval.           Best fit.
----------------------------------
[...]
100             608.43
[...]
extractor correct/total:0/1
 PENALTY cont=   10000000000.000000
  individual             1  fit=   10000000000.000000
[...]
extractor correct/total:1/1
 DELTA           calc=   1434.4099399999996       ref=   100.59999999999999       we=   0.10000000000000001        cont=   177904.89560428029
 FREQUENCY       calc=   781.64999999999998       ref=   3271.0000000000000       we=   1.0000000000000000E-004  cont=   619.68634224999994
 DISTANCE        calc=   3.7135732432381094       ref=   3.7003090959999998       we=   100.00000000000000       cont=   1.7593760195425176E-002
  individual             1  fit=   178524.59954029048
[...]
500             608.4
[...]



#
#Results
#
        1     BETAS H    +8.101808046038
        2        ZS H    +7.087829763576
        3       ALP H    +5.154836427027
        4       GSS H    -5.328341920547
        5       USS C    +18.998979138361
        6       UPP C    +46.181250338527
        7     BETAS C    -31.725560376293
        8     BETAP C    -1.149324435345
        9        ZS C    +3.304215163600
       10        ZP C    -6.055745378163
       11       ALP C    +1.087429221295
       12       GSS C    -5.451587242242
       13       GSP C    +0.110110780275
       14       GPP C    +10.631884089965
       15       GP2 C    +10.075900047122
       16       HSP C    -8.789173820345
```

Enhanced MOPAC Interface 13

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 12, so we shall only show the differences.

13.1 Input and executable files

The complete enhanced interface was explained in the Section 22. To create and run the example you must type:

```
$ cd mopac/shepherd
$ tar xvzf mopac-shepherd.tgz
$ gafit > output.txt
```

After this, the files created are shown in Table 13.1.

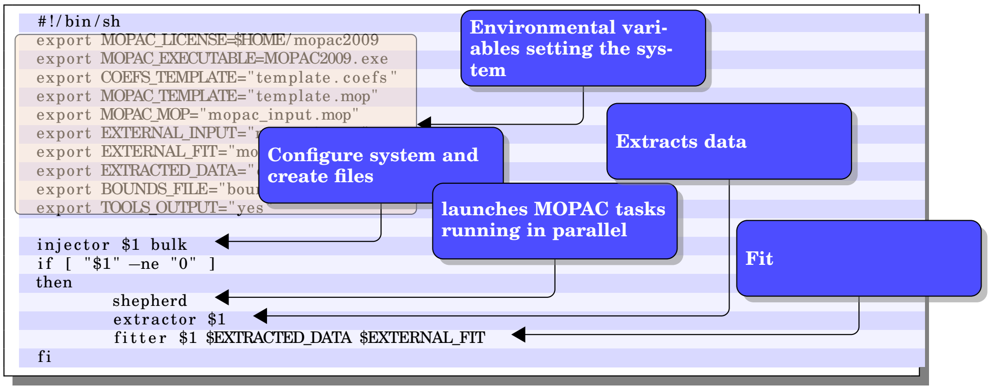
Checking files against the previous section example, you figure out that the *external-mopac.sh* file –13.1– is slighty different:

- the line "**injector $1**" is changed to "**injector $1 bulk**". As stated in 21.4, the option **bulk** brings the system to an *external bulk* configuration.

Table 13.1: Files in the shepherd-example folder.

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

File 13.1: **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="mopac_input.mop"
export EXTERNAL_INPUT="
export EXTERNAL_FIT="mo
export EXTRACTED_DATA="
export BOUNDS_FILE="bou
export TOOLS_OUTPUT="yes

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

Environmental variables setting the system
Extracts data
Configure system and create files
launches MOPAC tasks running in parallel
Fit

Here –See Section 15.2– a whole population of coefficient sets are passed from **GAFit**– Step 4 in page 92–.

- the line "**$MOPAC_LICENSE/$MOPAC_EXECUTABLE $MOPAC_MOP**" is replaced by "**shepherd**" only.

13.2 Running the example

The big difference with Section 12 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.

To see the output shown in File 13.2, the environmental variable **TOOLS_OUTPUT** must be set to **yes** as in File 13.1.

File 13.2: **shepherd** example output

```
 [...]
+-----------------------------------------------------------------+
 |      Settings for job                                           |
+-----------------------------------------------------------------+
 |      Command:[./external-mopac.sh]                              |
 |      Bounds:[bounds.txt]                                        |
 |      External input:[mopac2009.input]                           |
 |      External fit:[mopac?                                       |
 |      Total coefficients:                                        |
 |      Print options: runs                                        |
+-----------------------------------------------------------------+
 |      run: 1
 |      TEST MODE seed: 1488732015
+-----------------------------------------------------------------+
shepherd #flocks:8
shepherd elapsed time:31.718445
extractor correct/total:6/100
 [...]
 DELTA             calc=   -6301.0098500000004        ref=    100.599999
        ↪0.10000000000000001          cont=   4098060.8671617033
 FREQUENCY         calc=    1598.5699999999999        ref=    3271.0000000000000        we=   ↩
        ↪1.0000000000000000E-004  cont=    279.70221049000003
 DISTANCE          calc=    6.3118891919999998        ref=    3.7003090959999998        we=   ↩
        ↪100.00000000000000           cont=    682.03505978233693
  individual                6   fit=    4099022.6044319756
DELTA             calc=   -9881.3282600000002         ref=    100.59999999999999        we=  ↩
        ↪0.10000000000000001          cont=    9963889.1787786633
 FREQUENCY         calc=    1510.0400000000000        ref=    3271.0000000000000        we=   ↩
        ↪1.0000000000000000E-004  cont=    310.0980121600000
 DISTANCE          calc=    3.8634768579999998        ref=    3.                             =   ↩
        ↪100.00000000000000           cont=    2.662371855608865
  individual               41   fit=    9964201.9391626790
 [...]
 PENALTY cont=    10000000000.000000
  individual               23   fit=    10000000000.000000
 [...]
Eval.              Best fit.
______________________________________________
100                2624.58
 [...]
200                2624.58
 [...]

#
#Results
#
        1     BETAS H    -1.768452251222
        2        ZS H    -2.376986291435
        3       ALP H    +9.991399850692
        4       GSS H    +10.914581171663
        5       USS C    +6.828760684854
        6       UPP C    -29.019169662622
        7     BETAS C    +74.815480306193
        8     BETAP C    +2.377750618559
        9        ZS C    +2.854646124344
       10        ZP C    -1.962588155625
       11       ALP C    -5.219131584847
       12       GSS C    +4.110906954906
       13       GSP C    +12.013191111392
       14       GPP C    -14.296990835246
       15       GP2 C    -9.524967982213
       16       HSP C    -3.068438523015
```

A maximum of eight parallel MOPAC task to process this population

Time spent processing this population by eight parallel task

The population is of 100 coefficient sets and only 6 yield correct results

Coefficient set with correct results. Fit: 4099022.6044319756

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 15.4–, each of them containing a unique coefficient set to be used as external file for the *mopac template* –See Step 6 in page 92–. 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 22.2 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 12.

Here, we can use the same trick –Section 12.3– evaluating the *best.txt* to examine the fitting details:

```
$ cp best.txt mopac.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=  6.02010474994563588E-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
$
```

Part III

Reference

14 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[7].

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 14.1. A population is initialized; then, each member is evaluated according to some objective function. And finally, some of the members are selected to create a new population using reproduction techniques. The

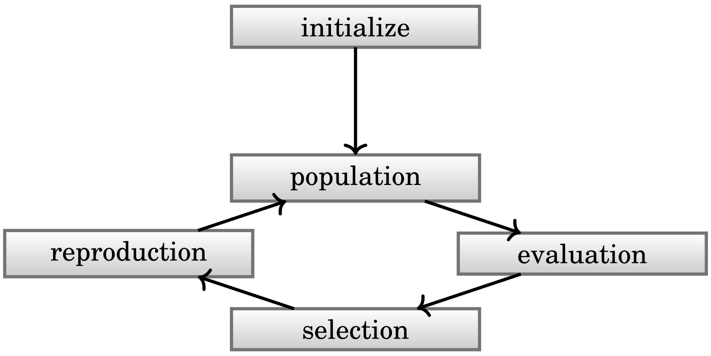


Figure 14.1: Evolutionary algorithms.

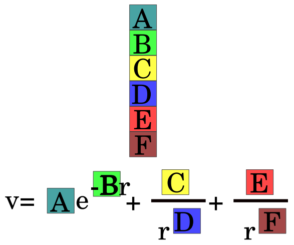


Figure 14.2: Genes and chromosome example: 4th potential from Table 17.2.

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.

14.1 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. 14.2 where is represented the 4th potential from Table 17.2.

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**.

Mutation

Mutation randomly changes one or more genes. If the chromosomes are of fixed length, we may have a single gene mutation (Fig. 14.3) or a multiple gene mutation (Fig. 14.4), and if the chromosomes are of variable length, there can be an insertion (Fig. 14.5) or a deletion (Fig. 14.6).

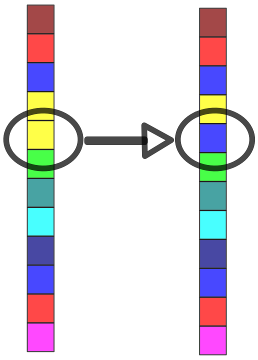


Figure 14.3: Single gene mutation.

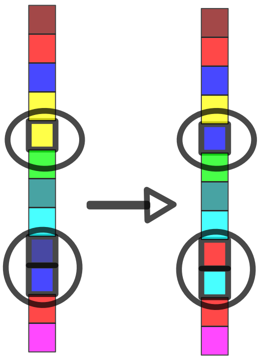


Figure 14.4: Multiple gene mutation.

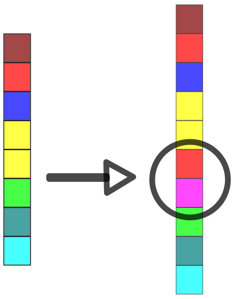


Figure 14.5: Variable lenght insertion.

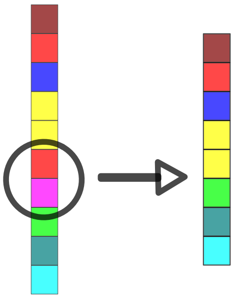


Figure 14.6: Variable lenght deletion.

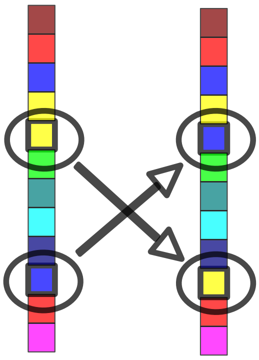


Figure 14.7: Permutation.

Permutation

Permutation exchanges a pair of genes. Fig. 14.7.

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. 14.8, 14.9, 14.10 and 14.11.

14.2 The Genetic Algorithm used in GAFit

The genetic algorithm used here was developed by Marques, Prudente, Pereira, Almeida, Maniero, and Fellows [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 14.1. 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 14.1: core.c

```
64
```

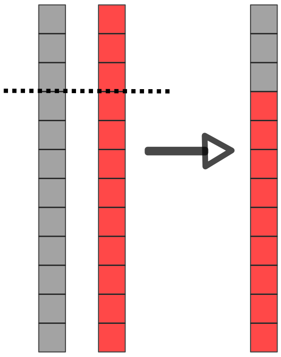


Figure 14.8: Single point crossover.

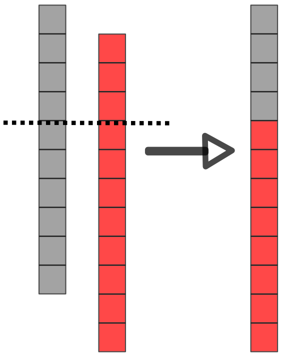


Figure 14.9: Variable lenght single point crossover.

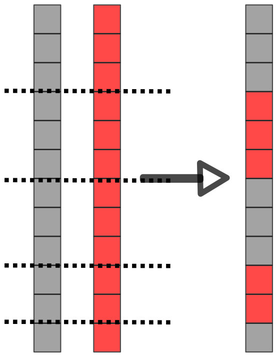


Figure 14.10: Multiple point crossover.

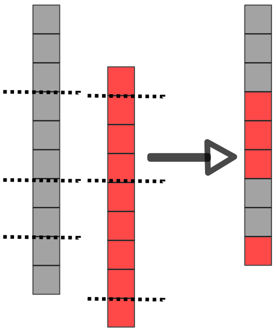


Figure 14.11: Variable lenght multiple point crossover.

```
  // allocates memory for individuals (to generate the new population)
  initializeNewPopulation (jo);

  // evolution cycle
  do
    {
      generation++;
//********************************************************
      current_evaluations += genetic (jo);
//********************************************************
      update_all_time_best (jo);

      // output stats each 'outputeach' evaluations
      if (current_evaluations - last_evals > outputeach)
        {
          last_evals = current_evaluations;
          stats (jo, generation, current_evaluations);
        }
    }
  while (current_evaluations < jo->evaluations);

  //last stats

  stats (jo, generation, current_evaluations);


  for (i = 0; i < jo->pop_size; i++)
    {
      free (jo->population[i].genes);
      free (jo->new_population[i].genes);
    }
  free (jo->population);
  free (jo->new_population);
  free (jo->best.genes);
  free (jo->new_best.genes);

  // release memory of old population
}

void
runJob (JOB * jo)
{
  int run;                            // current run
  FILE *output;
  time_t t1, t2;
  char randtext[TEXT_RANDOM_SIZE];

  initBest (jo);

  time (&t1);

  for (run = 1; run <= jo->runs; run++)
    {
      //************ hook to test mode *********//
      init_rand (jo->test, randtext);
      //****************************************//

      if (jo->print_run || jo->test != 0)
        {
          printRuRa (stdout);
          if (jo->print_run)
            {
              printRun (stdout, run);
            }
          printRand (stdout, randtext);
          printRuRa (stdout);
        }
      jo->last_print = 0;

      // header
      output = fopen (OUTPUT_FILE, "at");
      fprintf (output, "run␣%d\n", run);
      fprintf (output, "%s", randtext);
      fclose (output);

      algorithm (jo);
      fflush (stdout);
    }

  time (&t2);
  avg_stats (jo->runs, t2 - t1, jo->dir);
  free (jo->bounds);
  cleanJob (jo);
}


// main function
int
main (int argc, char **argv)
{
  JOB job;

  initJob (&job);

  banner (stdout, &job);

  // read print options
  ReadPrintOptions (&job);

  // read GA parameters
  ReadGaParameters (&job);

  // setup job
  ReadJobType (&job);

  // read data and set job's parameters
```

```
job.bounds = ReadJobExternal (&job);

if (job.bounds != NULL)
  {
    // Run GA
    runJob (&job);
    if (job.final_evaluation)
      {
        char run_command[STRING_MAX];
        snprintf (run_command, STRING_MAX - 1, "%s␣-1",
```

The system is configured reading an input file –Section 15–. Once configured, the GA main loop routine starts and continues till a maximum number of evaluations is reached as shown in Figure 14.12. 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 14.1: 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	

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*.

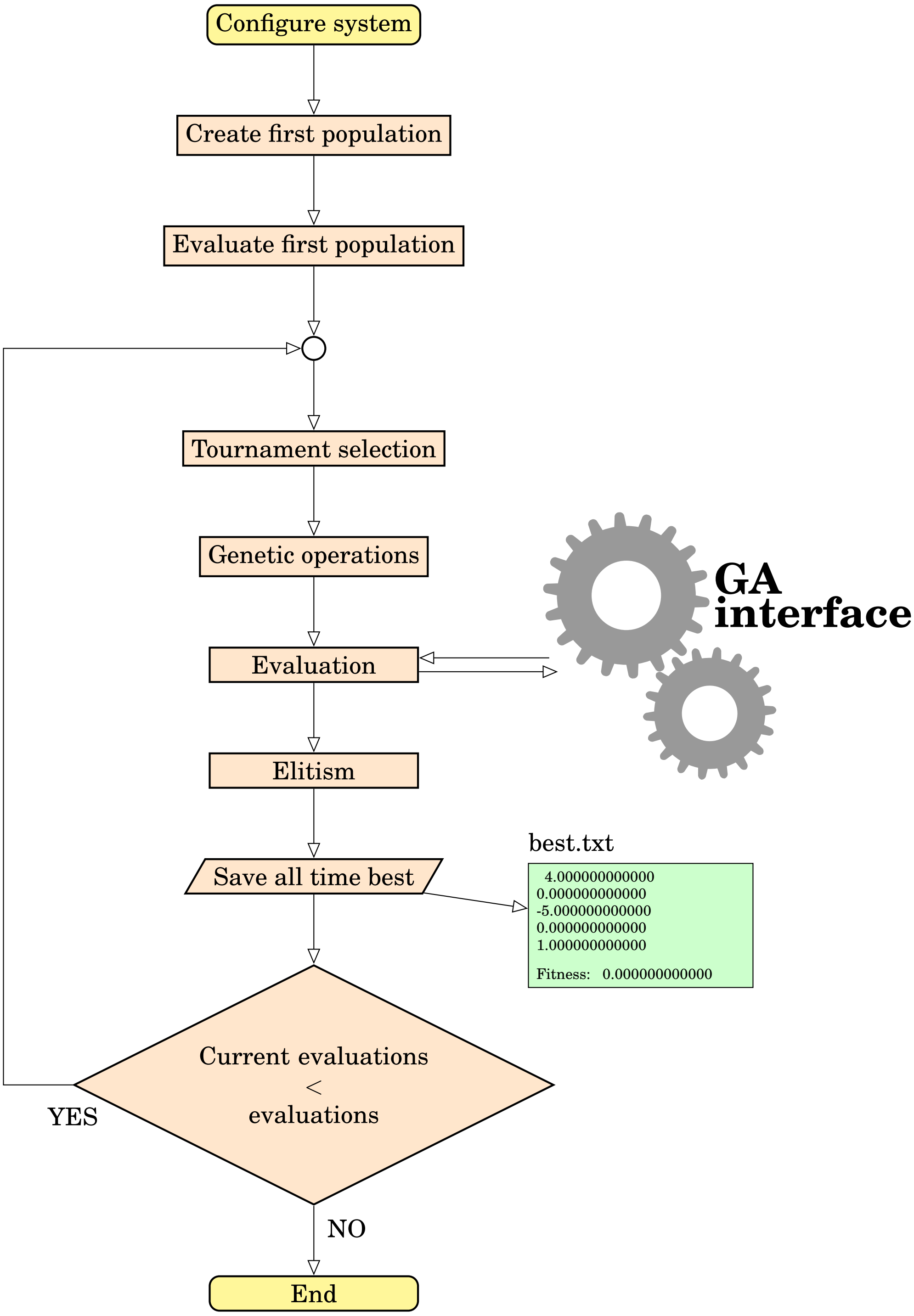
Genetic operations

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.

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



```
  4.000000000000
0.000000000000
-5.000000000000
0.000000000000
1.000000000000

Fitness:   0.000000000000
```

Figure 14.12: GA main loop

- Simulated Binary Crossover (SBX)[8]. SBX simulates a SPC operator on binary strings obtaining two offspring having some interesting properties to self-adaptation[9]:

  - 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:

  $$\begin{array}{rl} \beta = (2\mu)^{\frac{1}{\eta+1}} & if\ \mu\ \leq\ 0.5 \\ \beta = (\frac{1}{2(1-\mu)})^{\frac{1}{\eta+1}} & if\ \mu > 0.5 \end{array}$$

  - 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 15.1– 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].

- Blend Alpha Crossover (BLX-α)[10]. 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[9].

  The controlling parameter is α –blx_alpha, Table 15.1– 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[9].

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

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

[2] Known as BLX-0.5 crossover

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*.
  - *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 15.1– 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:

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

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.

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

15 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 15.1: job.txt. Genetic algorithm parameters and job settings for an intermolecular module job

```
[job]
runs:          1
type: external auto
command: external-intermolecular.sh
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
direction:       min

```

[1] Defined in *ga.h*

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

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

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

job The job to be done.

print Diverse printing options.

coefficient names This section is used to set a user name to each coefficient.

Each option, including the whole sections, can be avoided, but the file *job.txt* itself must be present. In case of omited parameters, the program takes some default values (See table 15.1), so you can write a *job.txt* file like 15.2. This case is included in the *advanced mode examples, miscellaneous, external* example as *minimal-job.txt* file.

File 15.2: Reduced job.txt.

```
[job]
coefficients:␣␣5
```

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. Parameters and sections are case-insensitive, but in parameters names with more than one word whitespace matters! Please, use one space between words.

Table 15.1: 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	{external, external bulk, external auto}	external
	runs	integer		1
	evaluations	integer		5000
	command	string		./external
	external input	string		external.input
	external fit	string		external.fit
	coefficients	integer		0

Section	Parameter	Type	Valid set	Default
print				
	runs	bool	{yes, no}	yes
	ga settings	bool	{yes, no}	no

In the Table 15.1 is summarized all common configuration options and its default values. There are options not shown in the Table applicable to some modules. i.e. the specific parameters for the **intermolecular module** are shown In the Table 17.1.

An alternative configuration, the *simple configuration mode*, was developed using the keyword **application** in the **[job]** section with some selected application modules. These work using the defaults for the module and specifying only the options that must be set by user as shown in Table 15.2.

Table 15.2: Job file, application modules options

Section	Parameter	Type	Valid set	Default
job				
	application module intermolecular			
	application	string	intermolecular	must be set
	evaluations	integer		5000
	potential	integer		1
	interactions	string	inter,all	inter
	application module mopac			
	application	string	mopac	must be set
	evaluations	integer		5000
	exec	string	absolute path to mopac executable including binary	none, must be set
	application module charmm			
	application	string	charmm	must be set
	evaluations	integer		5000
	exec	string	absolute path to charmm executable including binary	none, must be set
	refgeom	string	reference geometry	none
	calculated energies	two integers	which columns are the geometry names and the calculated energies	none, must be set
	application module mvariable			
	application	string	mvariable	must be set
	evaluations	integer		5000
	coefficients	integer	number of coefficients to fit	none, must be set
	application module generic			
	application	string	generic	must be set
	evaluations	integer		5000
	ncores	integer	number of parallel calculations	1
	template	string	templates	template
	executable	string	user provided script	none, must be set
	reference values	string	reference data	reference.values

The examples for this mode were shown in the *SimplifiedUserGuide.pdf*.

15.1 Section [parameters]

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

population Population size

elitism Elistism strategy. Section 14.2.

- no

- yes

tournament size Tournament selection size. Section 14.2.

crossover rate Crossover rate. Section 14.1.

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 14.2.

mutation Mutation type

- random = Random mutation
- sigma = Sigma mutation

sigma Sigma mutation coefficient

integer mutation Mutation operator for integer variables. Section 14.2.

- random
- adjacent

direction Search direction

- min: Minimization
- max: Maximization

15.2 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 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 21.1.

Test If it is not equal to zero, the integer is used as random seed, breaking the system randomness. This is the *test mode*, 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 one per run –if the option **print runs** is activated– in the standard output and in the file *stats.txt* as shown below. The use of this option forces one run despite the value of the **runs** parameter.

```
[...]
run 1
TEST MODE seed: 1488732015
[...]
```

Runs Number of runs. If the *test mode* is activated, only one run is performed.

Evaluations Number of generations

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*.

File 15.3: 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 15.4: 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 15.5: 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.

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

File 15.6: Bounds file

```
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
```

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

File 15.7: 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 15.8. 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 15.9.

File 15.8: External input

```
4.894146
0.013449
-6.092118
-0.003859
1.216052
```

File 15.9: 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: 15.10 and 15.11.

File 15.10: External fit: one individual fit

```
25647.561250
```

File 15.11: 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.

15.3 Section [print]

This section controls how much is printed.

Runs This parameter controls if the intermediate results are printed on standard output. See 16.

GA **settings** Prints genetic algorithm settings.

15.4 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 21.1.

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

16 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 15.3.

```
+------------------------------------------------------------------------------+
|     GAFit 1.3d Build:314  **TEST MODE, seed:1488732015 **                    |
|     Fri Mar  2 16:09:22 2018                                                 |
+------------------------------------------------------------------------------+
|                                                                              |
|     Cite this program as GAFit 1.3d                                          |
[...]
+------------------------------------------------------------------------------+

INTERMOLECULAR MODULE
--------------------------------
Coordinates:[coord.molden]
Energies:[energies.txt]
Atom2type:[atom2type.txt]
Bounds:[bounds.txt]
Charges:[charges.txt]
Potential read: 1
All coefficients: no, Read and repeat subset
Interactions types: inter
Fitting: relative

PRINT OPTIONS
--------------------------------
geometries no
analytical no

INTERACTIONS
--------------------------------
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

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)

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)
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 13
----------------------
H(13)-Xe(14)
Coefficients:
49          A        +0.00000 -  +1000000.00000 (real)
50          B        +0.00000 -       +10.00000 (integer)
51          C     -1500.00000 -        +0.00000 (real)
52          D        +4.00000 -        +8.00000 (integer)
+--------------------------------------------------------------------------------+
|     Settings for job                                                           |
+--------------------------------------------------------------------------------+
|     Command:[./external-intpot.sh]                                             |
|     Bounds:[bounds.txt.internal]                                               |
|     External input:[intpot.input]                                              |
|     External fit:[intpot.fit]                                                  |
|     Total coefficients: 52                                                     |
|     Print options: runs yes, ga settings no                                    |
+--------------------------------------------------------------------------------+
|     run: 1                                                                     |
|     TEST MODE seed: 1488732015                                                 |
+--------------------------------------------------------------------------------+

Eval.                Best fit.
------------------------------------
100               22.5565
200               22.5565
[...]
5000               4.53655


#
#Results
#

INTERACTION TYPE 1
----------------------
C(1)-Xe(14)
Coefficients:
1 A   +901608.8066303307
2 B        +4.0000000000
3 C        -6.4303232967
4 D        +5.0000000000

INTERACTION TYPE 2
----------------------
N(2)-Xe(14)
Coefficients:
5 A   +165595.8679798347
6 B        +7.0000000000
7 C     -1138.2394540608
8 D        +5.0000000000
[...]
```

```
INTERACTION TYPE 13
----------------------
H(13)-Xe(14)
Coefficients:
49 A   +520130.0359805273
50 B        +2.0000000000
51 C      -886.9079425981
52 D        +8.0000000000
#
#Evaluation
#
#Geometry     Energy        Calculated      Difference     Weight
#========     ======        ==========      ==========     ======
1   -0.006436000000  -0.007377743473  +14.63 %       +1.00
2   -0.012603000000  -0.013195973115   +4.71 %       +1.00
[...]
30   +146.056144000000  +213.560874079430  +46.22 %       +1.00
31   +297.072019000000  +611.114367352091  +105.71 %       +1.00
```

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

- The number of individuals evaluated up to now, *5000* in the last line before *#Results*.
- And the objective function best value up to now: *4.53655*.

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 20.2-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.

16.1 Other output files

Other intermediate output files are:

- stats.txt This file show statistical data about the fitting showing the number of evaluations performed, the generation, the average fitting in the generation and the best fit till now.

```
run 1
TEST MODE seed: 1488732015
Eval.        Gen.        Average/population         Best fit.
-------------------------------------------------------
100        1        5.03763e+14                         16027.9
200        2        1.05173e+13                         2624.58
```

17 Intermolecular module: input files

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

Bill Gates

Specific parameters for the **intermolecular module** are shown In the Table 17.1.

Table 17.1: Job file default value for intermolecular module specific parameters

Section	Parameter	Type	Valid set	Default
job				
	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}	yes
	fitting	string	{absolute, relative, user}	relative
print				
	geometries	bool	{yes, no}	yes
	analytical	bool	{yes, no}	yes

File 17.1: job.txt. Genetic algorithm parameters and job settings for an intermolecular module job

```
[job]
runs:         1
type: external auto
command: external-intermolecular.sh
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
direction:␣␣␣␣␣␣␣min

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

17.1 Section [job]

The job parameters from the *job* section are:

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

File 17.2: 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 17.3: Energies file. Energies and weights

```
−0.016881788␣␣1
−0.024242894␣␣1
−0.033981373␣␣1
␣...
```

File 17.4: Energies file. Structure

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

File 17.5: 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 17.6).
- 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*.

File 17.6: 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 17.7: 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 20.1, page 153.

interactions Here you can select the type of interactions to take into account.

inter Only *inter* fragments interactions: Fragment A × Fragment B interactions.

all All interactions between atoms: *inter* fragment interactions plus *intra* fragment interactions. To select only the *intra* interactions, use the *atoms2types.txt* file to specify the same number of atoms in the first fragment and the total. See page 129.

number A user defined number of interactions. You have to write a suitable function to evaluate the coefficients.

Charges This file must include partial charges (in a.u.) for all atoms when potential 4 is selected (see Table 17.2). Partial charges may be specified for atom types (File 17.8 and 17.9).

The types must be the same as those from *Atom2type file*. See 17.6. 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 generated from *needle*. See 20.1.

File 17.8: Charges. Type to charges mapping

```
␣␣␣␣1␣␣␣0.027
␣␣␣␣2␣␣␣0.113
␣␣␣␣3␣␣−0.057
␣␣␣␣4␣␣−0.01
```

```
␣␣␣␣5␣␣␣0.001
␣...
```

File 17.9: 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* file and two options more for setting a potential from source code in FORTRAN.

Table 17.2: Potential values

Value	Coefficients	Potential
-1	any	any user defined in userpotential.f
0	any	any analytical expression defined in an **[analytical]** section
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	2	$V = A\left[\left(\frac{B}{r}\right)^{12} - \left(\frac{B}{r}\right)^{6}\right] + 332.0532\frac{q_i q_j}{r}$

Table 17.2 shows the available potentials in *potentials.f* source file –positive values from table–, 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 15.3,15.4 and 15.5

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 18.1 section.

17.2 Section [print]

This section controls how much is printed.

Geometries This parameter controls if the read geometries are printed on standard output. See 16.

GA settings Prints genetic algorithm settings.

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

17.3 Section [analytical]

The reader is referred to Section 18.2, where this is explained in detail.

18 Intermolecular module: 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 17.2–, 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*. You have to compile the code.
- modifying the file *userpotential.f* using it as a template. As above, you need to compile the code.
- writing an *analytical expression*. Just write your function, no compile needed but slower execution. Useful for testing new intermolecular functions.

18.1 Modifiying potentials.f and userpotential.f

VGLOBALES fortran 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 18.1.

Fortran interface subroutines and functions

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

Table 18.1: Module VGLOBALES variables

variable	type	dimension	comments
r	double precision	(geometries, natom, natom)	Calculated interatomic distances for all atoms pairs
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
intratypes	logical	-	inter and intra interactions
userdefined	logical	-	user defined 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.

There are three cases:

- **inter interactions** 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 154.
- **intra + inter interactions** Atom **i** and Atom **j** are any atom pair.
- **user defined number of interactions** You cannot use *ix* for these types.

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.

Adding a new potential to potentials.f

Introducing a new potential in the program implies to implement it into *potentials.f* –File 18.1–, 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 18.1: potentials.f

```
 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
29       logical function getcharges(potential)
30       implicit none
31       integer potential
32       logical ugetcharges
33       if (potential .eq. -1) then
34         getcharges=ugetcharges()
35       else if (potential .eq. 0) then
36         getcharges=.false.
37       else if (potential .eq. 1) then
38         getcharges=.false.
39       else if (potential .eq. 2) then
40         getcharges=.false.
41       else if (potential .eq. 3) then
42         getcharges=.false.
43       else if(potential.eq.4) then
44        getcharges=.true.
45       else
46         stop 'getcharges:␣not␣implemented'
47       endif
48       end
49
50 c Potential Router, route calculations to the desired potential
51 c
52       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,k)
       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,k)
       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,k)
       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,k)
       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,k)
       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 18.1–.

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 18.2) contains, as an example, potential number 1 (see 17.2 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 132–.
- if you need to share or load some variables, you can use the **USER-DATA** module.

You can use the **function ix** (see page 134) to access individual coefficients or use the **subroutine coordinates** to access individual atom coordinates.

File 18.2: userpotential.f

```
 1 c USER POTENTIAL
 2 c please change as needed
 3
 4
 5 c USER DATA MODULE
 6
 7       module userdata
 8       implicit none
 9       save
10 c v------------CHANGE ME------------------------------v
11 c define your variables here
12
13 c ^------------CHANGE ME------------------------------^
14       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,k)
       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*.

In order to use **fitview** to plot two body interactions you need to provide **curRouter** with a function capable of calculate the potential using the atom pair, the distance between them and the coefficients as arguments.

18.2 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 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 18.2 shows and explains them.

Table 18.2: 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**

An example can be seen in File 18.3. It also shows different forms to express the potential.

As you see in File 18.3, "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 is *number 3 standard potential* defined in table 17.2.

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 17.2.

File 18.3: job.txt. Analytical expression options

```
[job]
runs:			1
command: ./external-intermolecular.sh
evaluations:		5000000
Geometries:   coord.molden
Energies:  energie
Atom2type: atom2ty
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
```

Potential: 0

Analytical expression section

Selected analytical expression

Operators and functions supported in expressions are shown in table 18.3. 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 18.3, section **[potential 3]**.

To check your potential definition you can use **ufpu**. See 20.3.

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

Table 18.3: 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)
sin	sine of a (in radians), $\sin(a)$		sin(a)
cos	cosine of a (in radians), $\cos(a)$		cos(a)

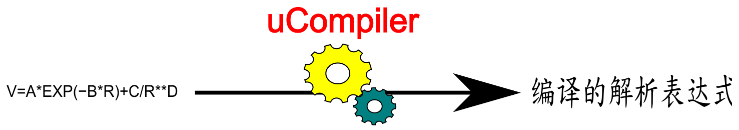
Intermolecular module: Fpu simulator 19

Сколько языков ты знаешь - столько раз ты человек..

А.П.Чехов

19.1 Fpu overview

Figure 19.1: **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 19.1–, so it can be executed by a **Fpu** instance –Figure 19.2–.

Source code is included in the folders *fpu*, *compiler*, *pack*, *bytecodes* and *nullist*. A complete implementation is the **ufpu** tool. See Section 20.3.

Figure 19.3 shows a **Fpu** overview. It contains:

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

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*.

Figure 19.2: **Fpu** load the machine code and process the variables to obtain V value.

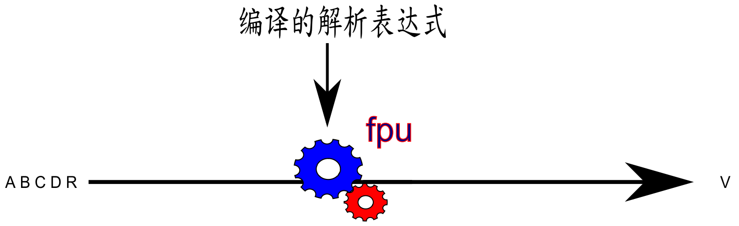


Table 19.1: **Fpu** source code

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

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 19.2.

19.2 Mode of operation

A program example is shown in File 19.2, which is generated using the *job.txt* file configuration 19.1. Semicolons are interpreted as comments.

File 19.1: Job.txt to generate the File 19.2

```

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

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

[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 19.3: **Fpu** overview

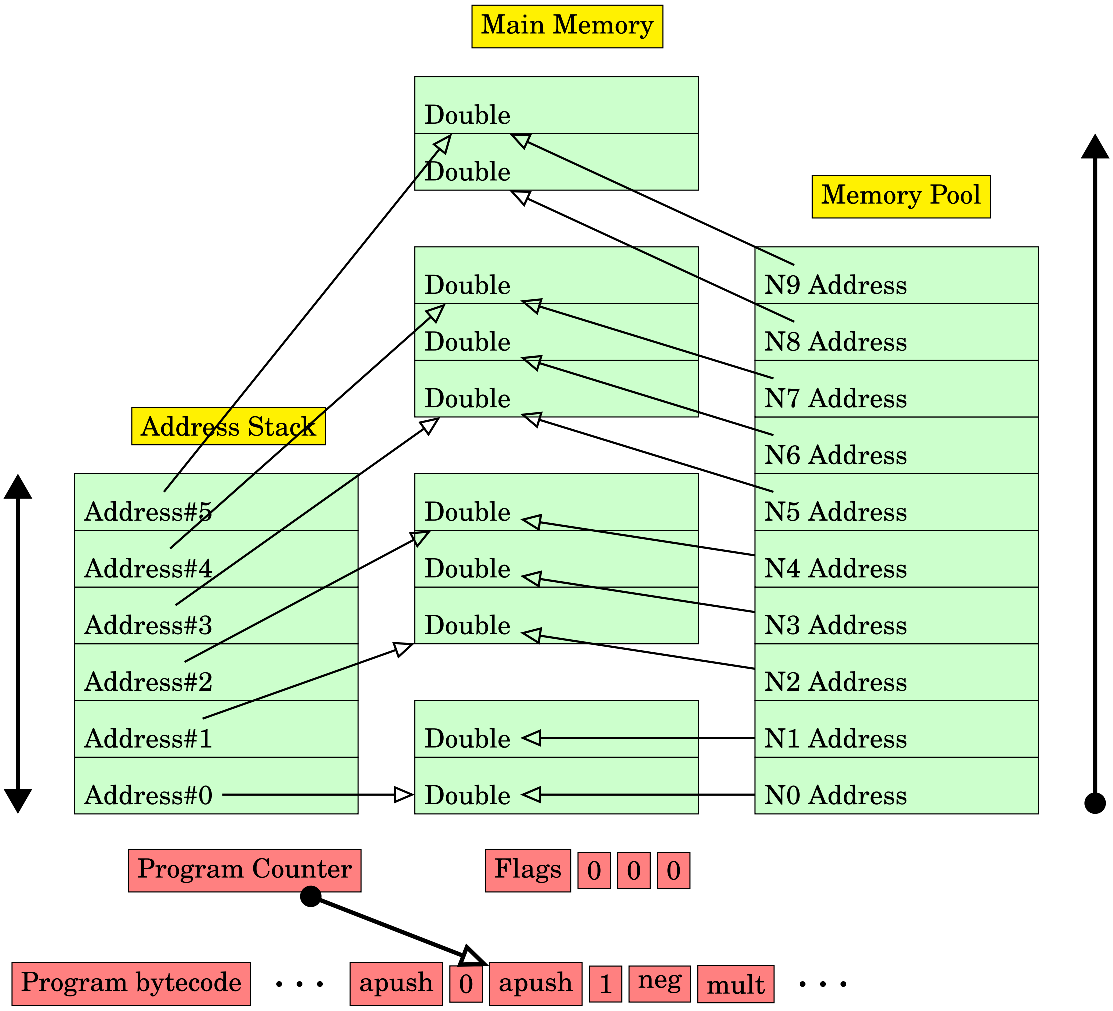


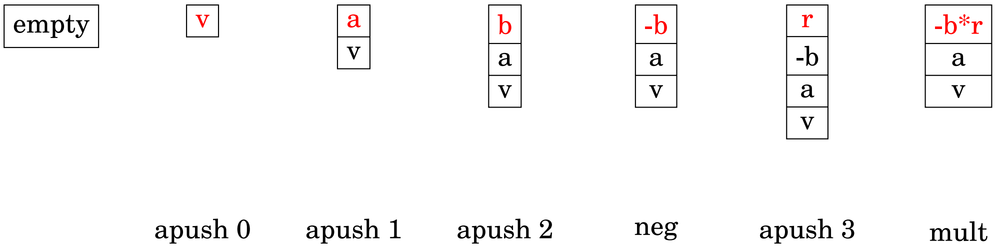
Table 19.2: Fpu instruction set

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
SIN		allocates memory for the result of $sin(topmoststack)$, pops the top most stack references, and pushes onto it the result reference
COS		allocates memory for the result of $cos(topmoststack)$, pops the top most stack references, and pushes onto it the result reference

File 19.2: 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
```

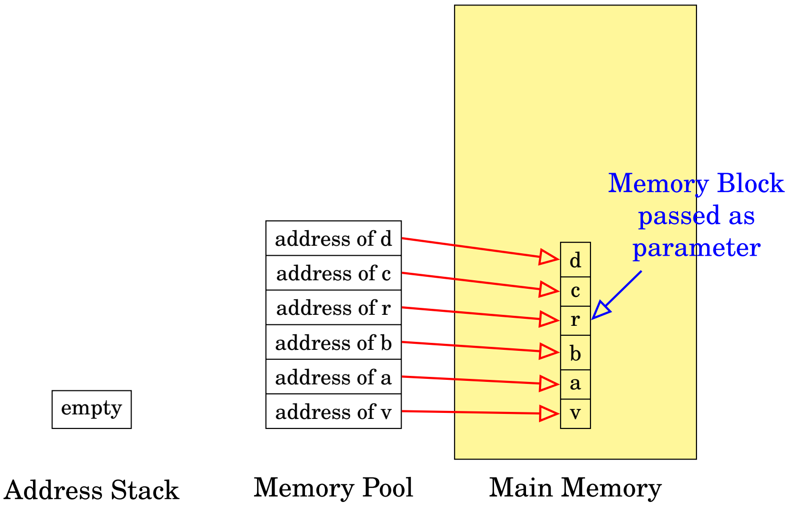
As shown in File 19.2, a memory block must be passed to **Fpu** containing the variables v, a, b, r, c, d in the correct order, as it could be seen in the first lines of the file –comments which are generated by the compiler as a remark–. At this time, the *Address Stack* is empty, so v, a and b are 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 19.4: Initial status



From the memory management point of view, the first six operations from File 19.2 are shown in figures 19.4 to 19.8. 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 19.5: apush 0, apush 1, apush 2

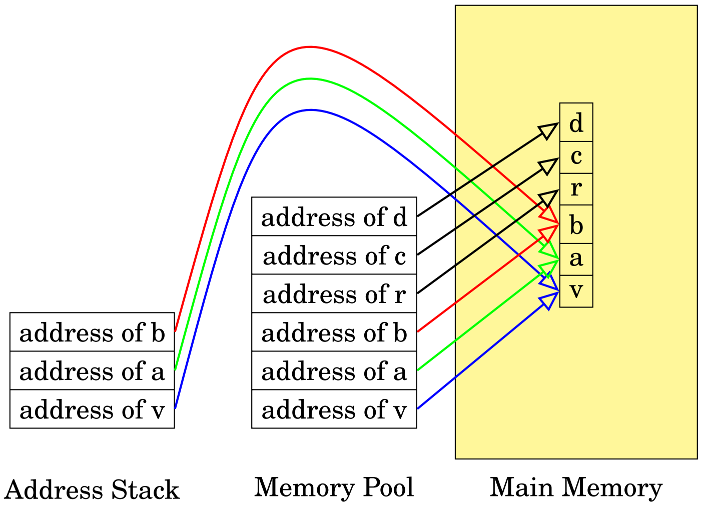


Figure 19.6: neg

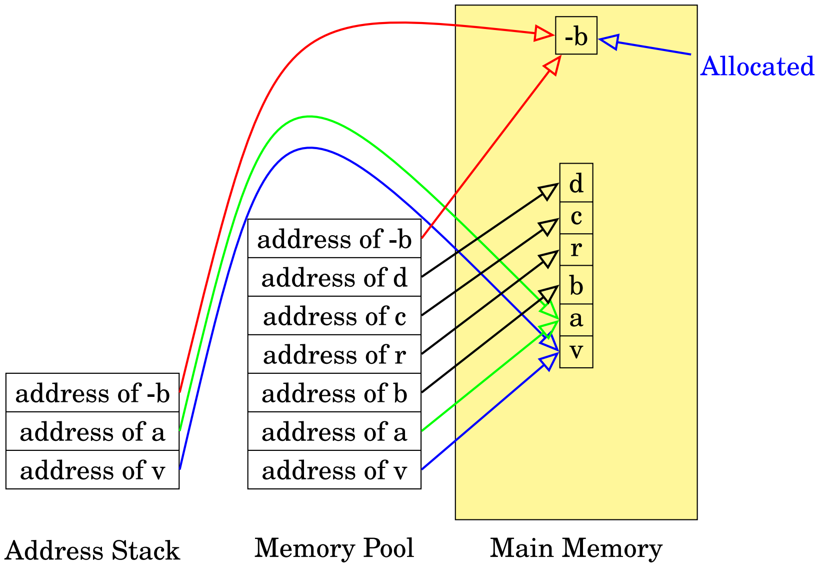


Figure 19.7: apush 3

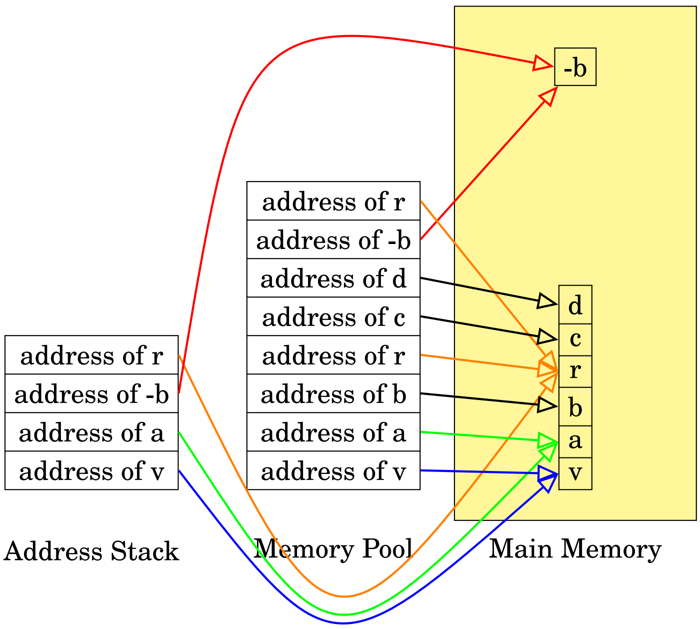
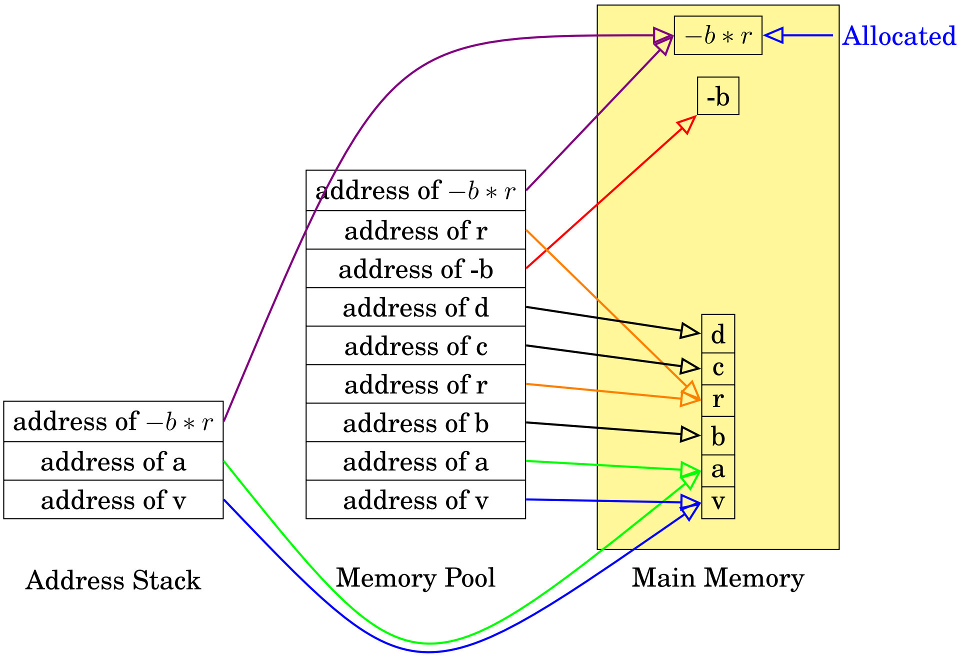


Figure 19.8: mult



20 Intermolecular module: 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*

20.1 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 v0.5 (c)GAFit toolkit -  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 v0.5 (c)GAFit toolkit -  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. See 17.1.

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.

20.2 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:

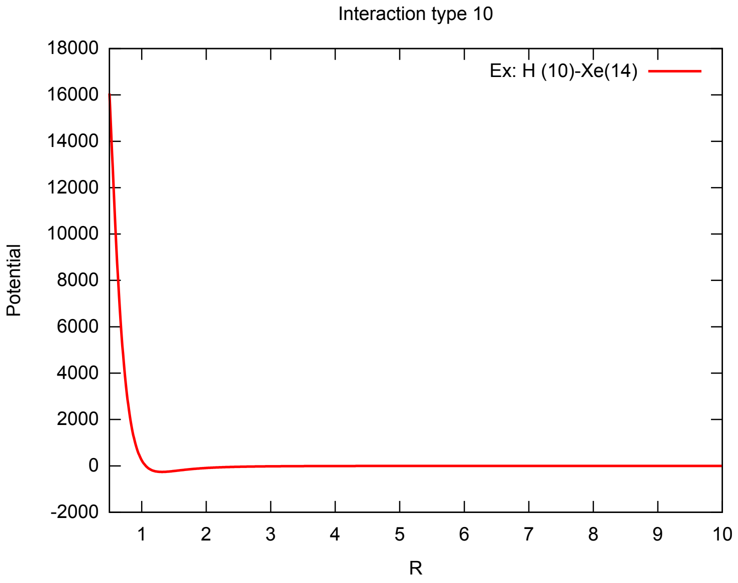
[1] Home page: http://www.gnuplot.info/. **Gnuplot** is a portable command-line driven graphing utility for Linux, OS/2, MS Windows, OSX, VMS, and many other platforms.

- 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
fitview v0.3 (c)GAFit toolkit - 2010-2013
Usage: fitview [tag] [-l value] [-u value] [-d value] [-h]
           -l lower bound
           -u uper bound
           -d delta
           -e gnuplot supports enhanced terminal
           -h this help
           -g general evaluation only
           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.

Figure 20.1: 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. Take special care using a potential of your own: See 18.1.

20.3 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 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 19.

The output shown was generated using File 18.3.

```
uFpu v0.2 (c)GAFit toolkit - 2013

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
```

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

```
        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
```

21 MOPAC module

> 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 –from 2009 to 2016– as the external program to compute the PES of our system. In the example given in Section 12 the MOPAC interface is used to parametrize the intramolecular PES of vinyl cyanide.

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

21.1 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 15.8 and 15.9. **external fit** examples are the Files 15.10 and 15.11

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

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 21.1: response

```
[job]
type: external bulk
coefficients: 16
external input: mopac.input
external fit: mopac.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.

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 21.1 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 22.

21.2 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.

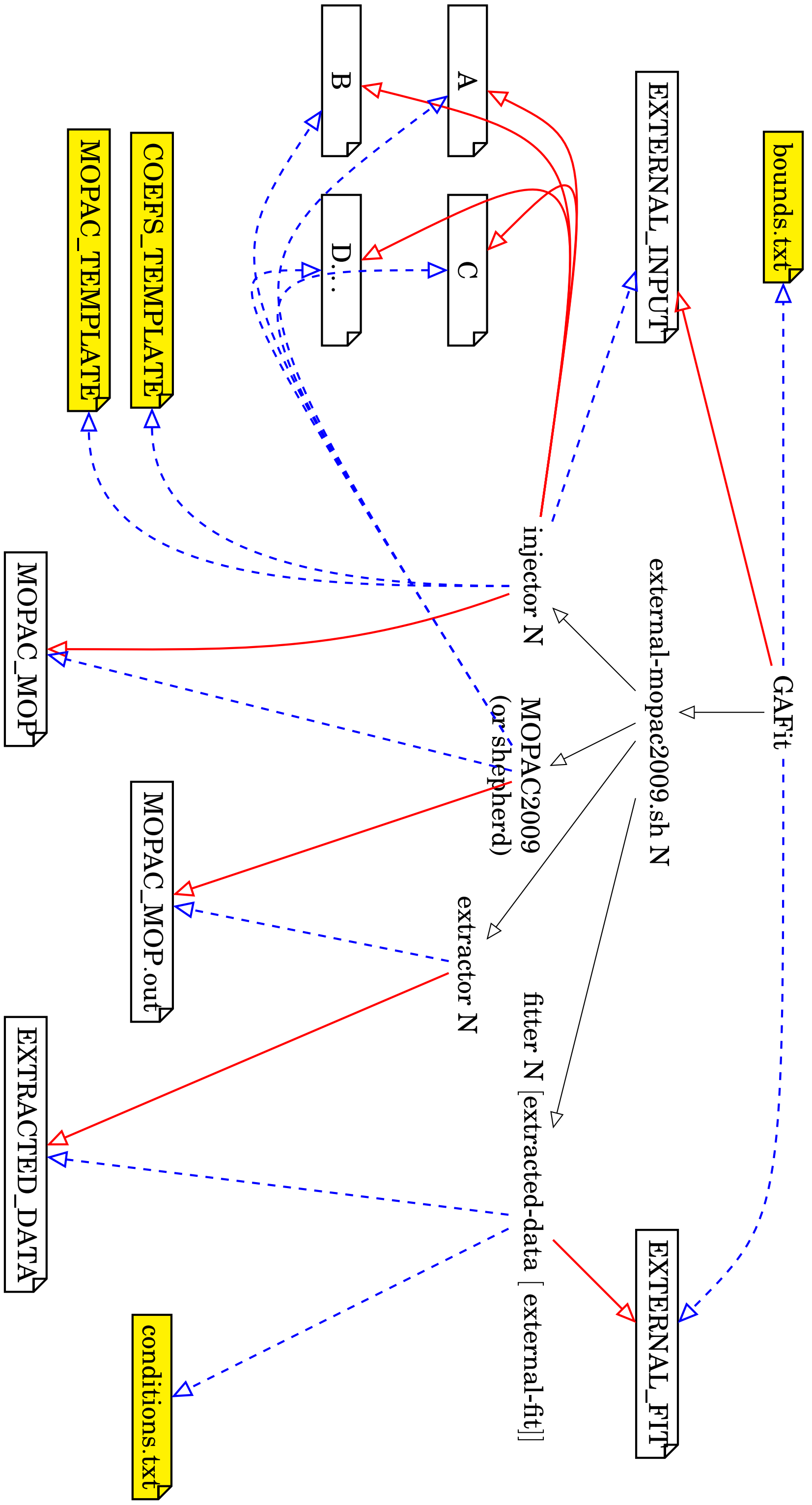
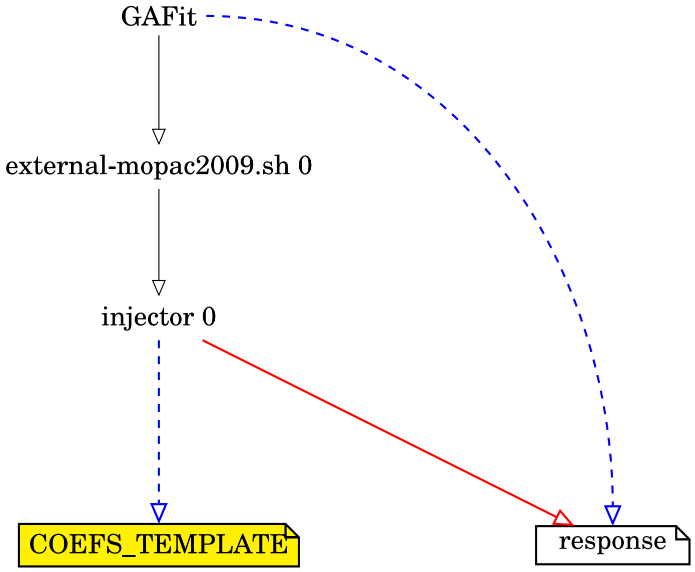


Figure 21.1: MOPAC 2009 interface: normal operation

Figure 21.2: MOPAC 2009 interface: autoconfigure



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**.

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 21.1 and 21.2 show the relations between programs and files:

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

There are environmental variables, shown in Table 21.1, which can be set to control the file names.

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

Table 21.1: 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	mopac.input	**GAFit**, injector
EXTERNAL_FIT	mopac.fit	**GAFit**
EXTRACTED_DATA	extracted.data	extractor
BOUNDS_FILE	bounds.txt	**GAFit**, injector

21.3 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 21.2. A minimal implementation due to the defaults could be the one in File 21.4.

File 21.2: 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="mopac.input"
export EXTERNAL_FIT="mopac.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
```

21.4 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.

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 21.3: 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 21.1).

A minimal external script is shown in File 21.4. In this case, the *external auto* option defaults to *external*. To override defaults use *bulk* option to change to *external bulk*.

File 21.4: 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
```

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 15.4.

[1] Number and name of the coefficients.

File 21.5: 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 21.6: 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 **COE FS_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.

21.5 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 21.7: Extractor first lines

```
#!/usr/bin/perl

use strict;

use constant {
     HEATFCAL    => 0,
     HEATFJUL    => 1,
     NUMATOMS    => 2,
     CARTESIAN   => 3,
     NUMFREQ     => 4,
     FREQUENCIES => 5,
     CALCPERIND  => 6,
     GRADIENTS   => 7,
     NUMCONF     => 8,
     DIPXYZ      => 9,
     EEL         => 10,
};

my($CALS_TO_JOULES)=4.1868;

my (%defaults) = (
      'COEFS_TEMPLATE' => "template.coefs",
      'MOPAC_TEMPLATE' => "template.mop",
      'MOPAC_MOP'      => "mopac_input.mop",
      'EXTERNAL_INPUT' => "mopac.input",
      'EXTERNAL_FIT'   => "mopac.fit",
      'EXTRACTED_DATA' => "extracted.data",
      'CONDITIONS_FIT' => "conditions.txt",
      'TOOLS_OUTPUT'   => "no",
);

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

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
         ",
42        "THIS␣IS␣A␣FATAL␣ERROR,␣RUN␣STOPPED␣IN␣GMETRY" ,
43        "TS␣FAILED␣TO␣LOCATE␣TRANSITION␣STATE" ,
44        "A␣FAILURE␣HAS␣OCCURRED,␣TREAT␣RESULTS␣WITH␣CAUTION!!" ,
45        "EXCESS␣NUMBER␣OF␣OPTIMIZATION␣CYCLES" ,
46        "SHEPHERD␣NON␣RECOVERABLE␣ERROR"
47  );
```

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 21.1 and File 21.7, 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 36 of the script –File 21.7–, 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 21.8: 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 21.2: 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
GRADIENTS	7	1	Gradients, x,y,z components per atom
NUMCONF	8	1	Number of states considered in one-electron excitations
DIPXYZ	9	4	Components x, y, z of the effect of dipole operator on states
EEL	10	3	Energies on states

The different types of extracted data are shown in table 21.2 and in the line 5 of the File 21.7. An example is given in File 21.8. Failed calculations are not written to the file.

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

21.6 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 21.3. 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 21.1 –**EXTRACTED_DATA** and **EXTERNAL_FIT**, respectively.

Table 21.3: 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*
gradient	4	calcA N value weight	Gradient number *N* of the calculation *calcA*. N varies from 1 to 3*NUMATOMS.
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*
dipx	4	calcA state value weight	Component *x* of the effect of dipole operator on *state* into calculation *calcA*
dipy	4	calcA state value weight	Component *y* of the effect of dipole operator on *state* into calculation *calcA*
dipz	4	calcA state value weight	Component *z* of the effect of dipole operator on *state* into calculation *calcA*
eel	5	calcA state order value weight	State energy into calculation *calcA*. *State*: 1 for singlet, 2 for doublet and 3 for triplet. *Order* is the order in the listing (eg. 1 for first singlet, 2 for second singlet and so on). If there are no data for this state, a **penalty** is applied.
penalty	1	penalty	Fit if any of the MOPAC calculations failed for a given coefficient vector. If not set, default value is 1.0e10.

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 21.10. The overall fit per coefficient vector is the sum of relative differences in each line calculation multiplied by its weight.

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

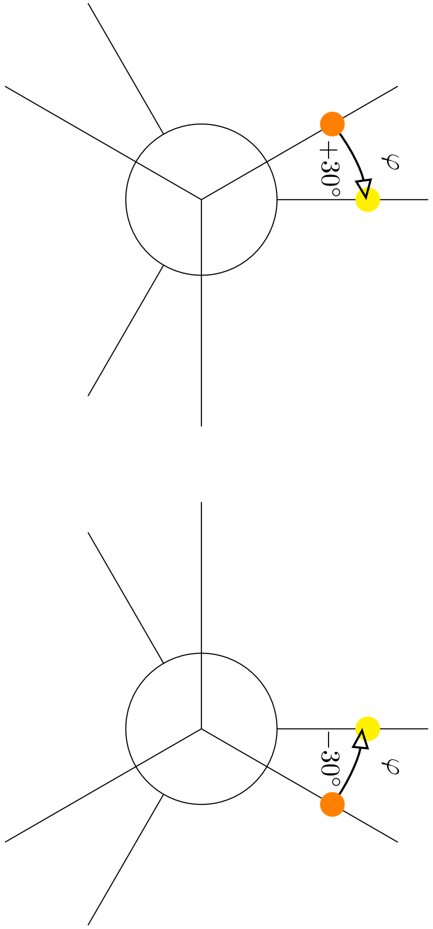
Due to the fact that distances, angles and dihedral angles are calculated from the Cartesian coordinates, the intervening atoms may not be connected in any other way.

The dihedral angles follow the usual convention, shown in the figure 21.3.

To express a condition, only the four first characters are needed, as shown in bold in table 21.3.

An example of fitter calculations using the file *conditions.txt* shown in File 21.10 is presented in File 21.9, where the type of condition, the calculated value, the reference value, the weight used, and the individual contributions to the final fit were printed. Default output is none but to activate it you must set the **TOOLS_OUTPUT** environmental variable to **yes** as shown in File 21.11.

Figure 21.3: Dihedral angles convention



File 21.9: 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=␣␣1.77014291129788933E−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=␣␣1.57488381692090322E−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=␣␣4.46057372941259230E−005
␣DISTANCE␣␣␣␣␣␣␣␣calc=␣␣␣3.8158625315909900␣␣␣␣␣␣␣ref=␣␣␣3.7003090959999998␣␣␣␣␣␣␣we=␣␣␣↩
     ↪100.00000000000000␣␣␣␣␣␣␣cont=␣␣9.75191075166021992E−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=␣␣3.51430398092760188E−005
␣DISTANCE␣␣␣␣␣␣␣␣calc=␣␣␣5.1161291627557643␣␣␣␣␣␣␣ref=␣␣␣3.7003090959999998␣␣␣␣␣␣␣we=␣␣␣↩
     ↪100.00000000000000␣␣␣␣␣␣␣cont=␣␣␣14.639967757020900
␣␣individual␣␣␣␣␣␣␣␣␣␣100␣␣fit=␣␣␣67.852646639194873
```

File 21.10: conditions.txt

```
delt␣␣␣1␣␣2␣␣100.6␣␣␣␣0.1
frequency␣␣␣2␣␣␣␣15␣␣␣3271.0␣␣1e−4
distance␣␣␣3␣␣␣␣␣1␣␣␣␣␣␣7␣␣␣␣3.70␣␣100.0
penalty␣1e10
```

File 21.11: Minimal external-mopac2009.sh with the tools output active

```
#!/bin/sh
export MOPAC_LICENSE=$HOME/mopac2009
export MOPAC_MOP="mopac_input.mop"
export TOOLS_OUTPUT="yes"

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

21.7 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 22.

- 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 21.6. You must use the **injector** default option to calculate one vector at once, or use **shepherd** to deal with it.

21.8 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 [11] article is a good reference about this subject.

21.9 MOPAC 2016

There are no significant difference with this interface, but there are some output to *stderr* which make the *enhanced interface* –see Section 22– think that something is going wrong and kill the process. These are now filtered by **shepherd**.

22 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 21.7. 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 22.2.
- 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 22.1:

- 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 22.1: *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="mopac.input"
8  export EXTERNAL_FIT="mopac.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 22.1 is 22.2 using the default values. **shepherd** is totally configured by the environmental variables.

File 22.2: 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
```

File 22.3: Short script for MOPAC 2012

```
#!/bin/bash
export MOPAC_LICENSE=$HOME/mopac2012
export MOPAC_EXECUTABLE=MOPAC2012.exe
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/lib/saa/:$MOPAC_LICENSE

injector $1 bulk

if [ "$1" —ne "0" ]
then
shepherd
extractor $1
fitter $1
fi
```

22.1 Controling freezes

If a MOPAC 2009 process crashes, it freezes and blocks all the entire job (see 21.7).

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.

22.2 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] 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 15.4–, 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.

[1] Flocks in shepherd parlance

[2] Sheep

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

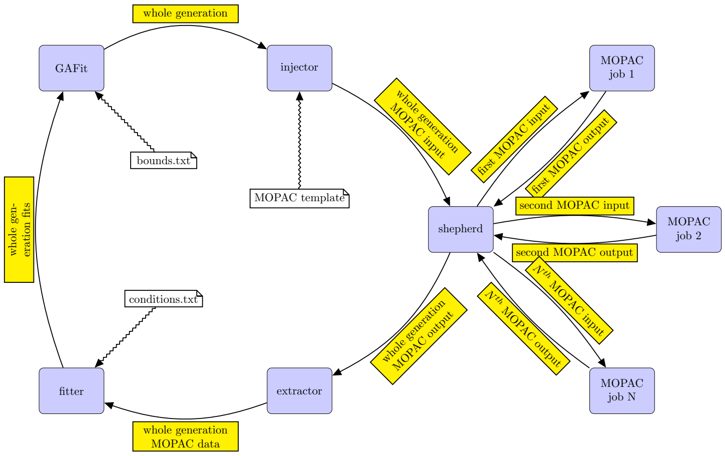
The other mode –**burst**– is disabled but it can be enabled recompiling the source code changing the line 640 in the *main* function setting **burst** to a value different from zero, File 22.4. **burst** mode is discouraged. See 21.7.

File 22.4: Shepherd, main function.

```
637 int
638 main (int argc, char **argv)
639 {
640   int burst = 0;
```

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

Figure 22.1: Data flow between **GAFit** and **shepherd**.



22.3 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 22.2.

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.

Using the *taskset* utility, some experiments were performed. Figure 22.3 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 22.4. Notice that the algorithm behaves as if there were only four core CPU.

[3] A sheep per flock

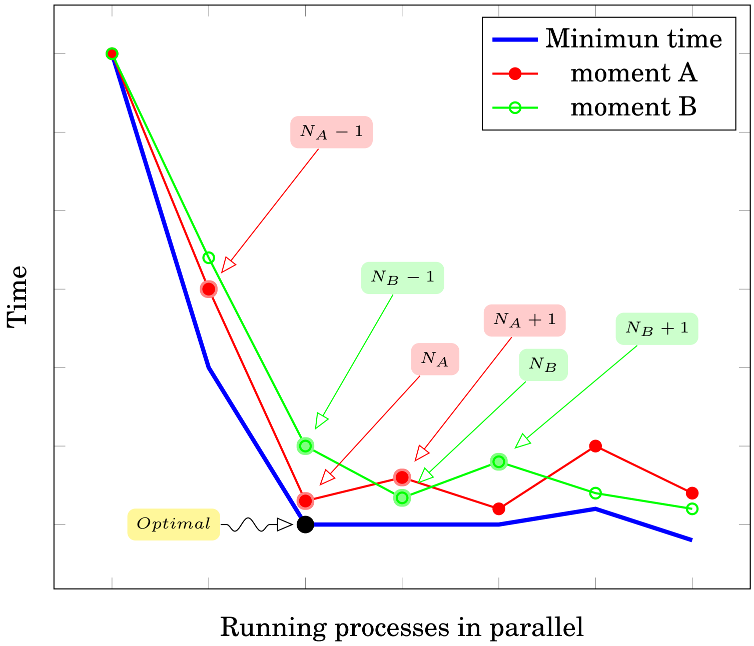


Figure 22.2: Shepherd algorithm: minimum time

In figure 22.2, 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 **shepherd** are shown in figures 22.5 and 22.6.

This information can be summarized taking into account the average N in both cases, as shown in figure 22.7.

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 22.8 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.

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.

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

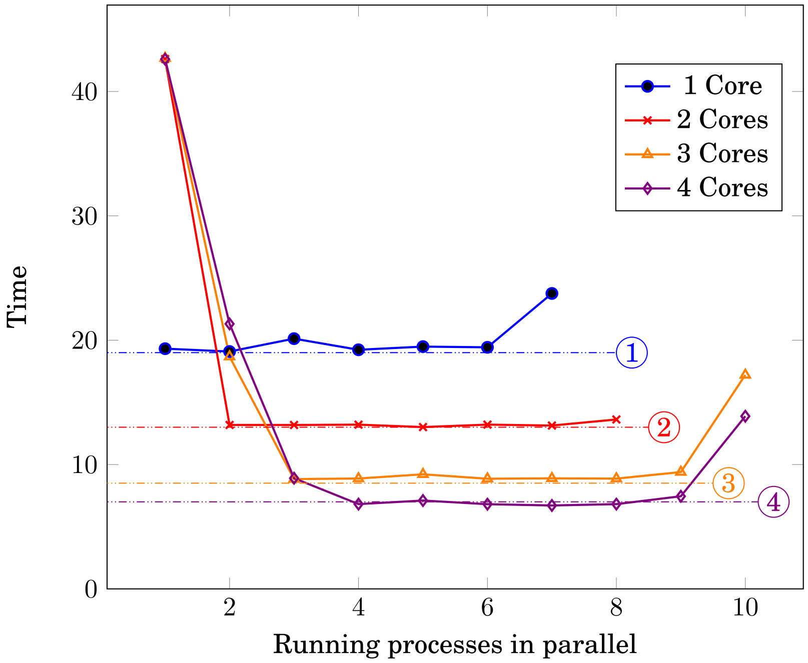


Figure 22.3: Real four core CPU: minimun time vs maximum concurrent parallel processes per run

Some interesting utilities, like **lstimes**, created to help with the MOPAC interface are commented in the Section 23.

You can fix the number of concurrent MOPAC processes setting the environmental variable SHEPHERD_CORES . Also, if ***using the simple configuration***, you can use "**ncores: number**" into *job.txt* configuration file.

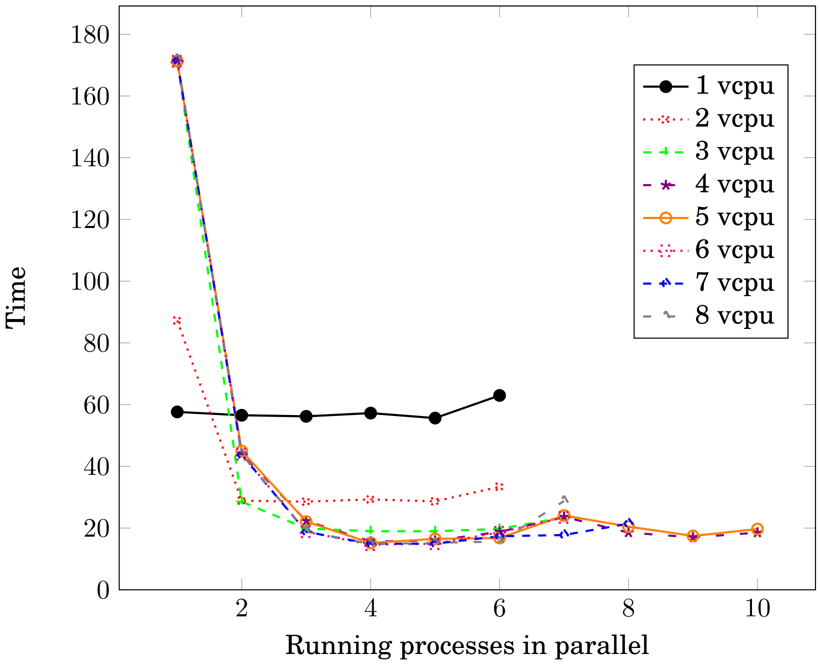


Figure 22.4: Virtual eight core CPU: minimum time vs maximum concurrent parallel processes per run

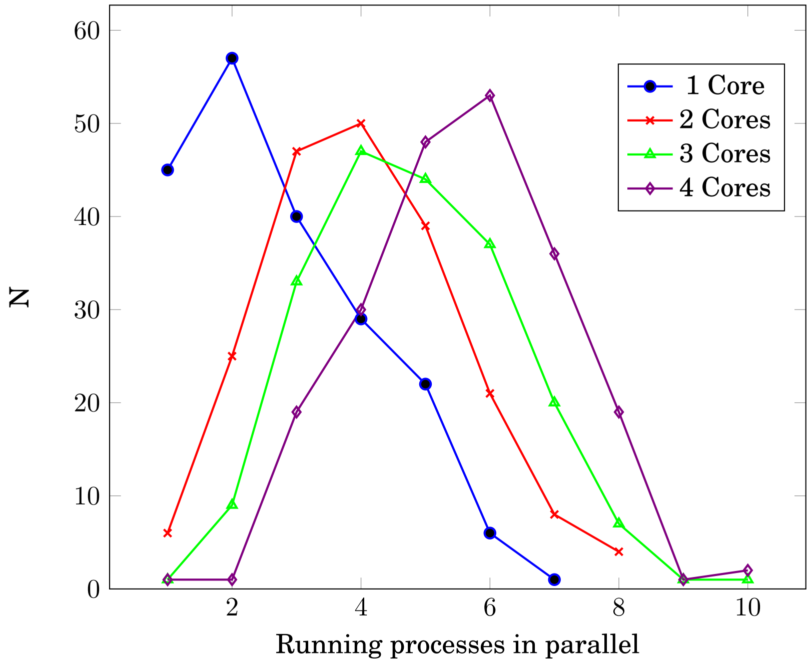


Figure 22.5: Real four core CPU: number of times (N) vs parallel processes per run

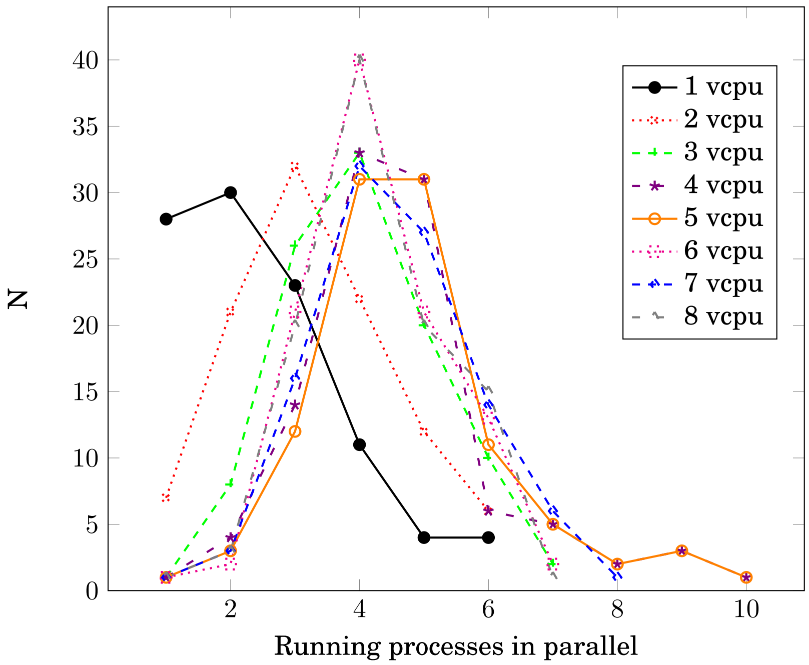


Figure 22.6: Virtual eight core CPU: number of times (N) vs parallel processes per run

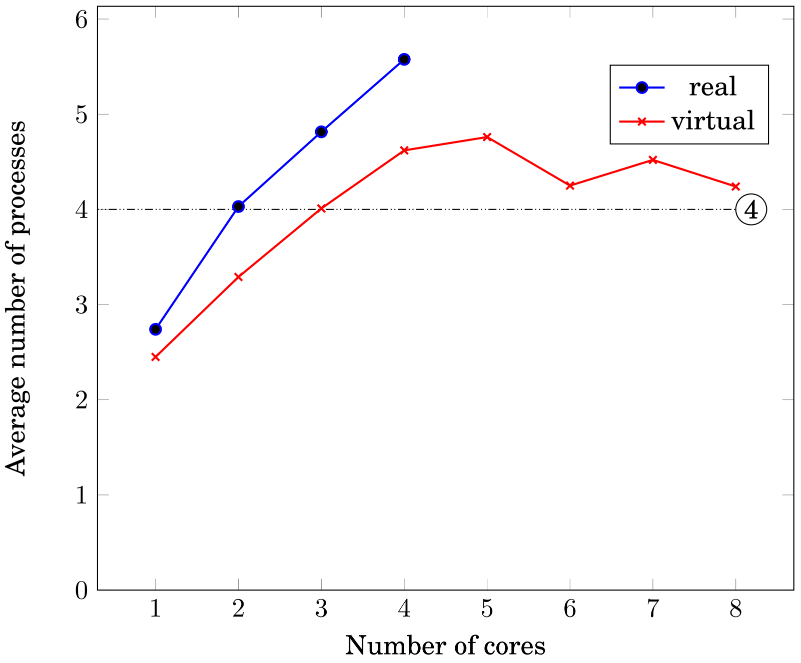


Figure 22.7: Average parallel processes per run. 4 core real CPU vs 8 core virtual CPU (4 real)

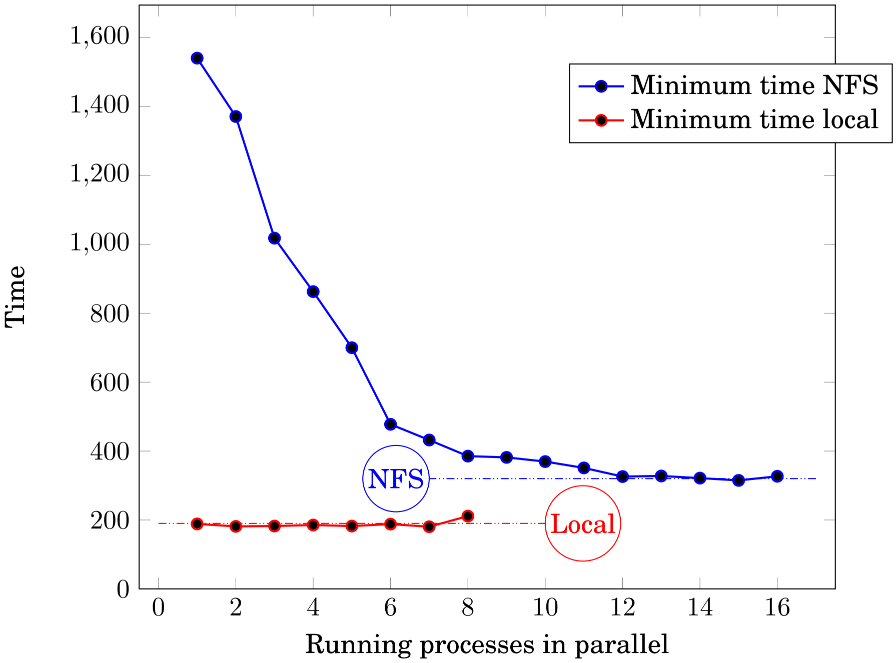


Figure 22.8: Behavior in the same one core CPU writing output to a NFS share vs local storage.

23 Mopac module tools

Give a man a fish, and you feed him for a day. Teach a man to fish, and he'll invite himself over for dinner.

Calvin Keegan

23.1 lsexdata

Utility to view the data extracted and saved by **extractor** in an intermediate file.

```
$ lsexdata
  lsexdata v0.1(c)GAFit toolkit -  2014
  Usage: lsexdata #individuals [file-data]
```

lstimes

Used to list statistical information about the processes managed by **shepherd** while running. The asterisk show the optimal point, the arrow the current number of concurrent processes.

```
$ lstimes
(c)GAFit toolkit 2014
 -- ---- ----------- ----- ----------- -----------
 PR slot current t     n      min         max
 -- ---- ----------- ----- ----------- -----------
 --    1  103.296173     5   32.095946  177.990252
 --    2   66.666929    19   10.797143  177.528340
 --    3   57.509871    33   14.297250  169.456044
 *-    4   56.051490    52   19.642072  164.538096
 ->    5   56.279652    51   23.727960  153.558413
 --    6   50.923629    42   25.808633  152.090570
 --    7   57.991970    44   20.538284  148.776540
 --    8   58.542538    26   22.370231  146.897739
 --    9   24.146727    29   20.527292  145.649365
 --   10   56.634926    23   17.946090  145.636910
 --   11   25.284372    34   19.549988  145.340982
 --   12   27.906675    27   18.232288  146.541943
 --   13   20.938043    12   20.345970  146.325395
--   14   21.070755     9   21.070755  141.851327
--   15  137.002666     1  137.002666  137.002666
--   16    0.000000     0    0.000000    0.000000
-------------------------------------------------
last:5 total:407 average: 7.02
```

23.2 mkbounds

Useful to create the *bounds* file from the *coefficients template* varying the values a %percent up and down before run **shepherd**.

```
$ mkbounds
mkbounds v0.1 (c)GAFit toolkit - 2014
        Create bounds file from coefs.template
        Usage: mkbounds %percent
```

Using the default values, to create a *bounds.txt* file from *template.coefs* with the upper bounds increased 10% and the lower bounds decreased 10% from the *template.coefs* values:

```
$mkbounds 10
```

File 23.1 is an example using **mkbounds** taken from the gradient-example included with the code.

File 23.1: External command with mkbounds

```
#!/bin/sh
export MOPAC_LICENSE=$HOME/MOPAC
export MOPAC_EXECUTABLE=MOPAC2012.exe
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MOPAC_LICENSE
export COEFS_TEMPLATE="template.coefs"
export MOPAC_TEMPLATE="template.mop"
export MOPAC_MOP="mopac_input.mop"
export EXTERNAL_INPUT="mopac.input"
export EXTERNAL_FIT="mopac.fit"
export EXTRACTED_DATA="extracted.data"
export BOUNDS_FILE="bounds.txt"

injector $1 bulk

if [ "$1" -ne "0" ]
then
        shepherd
        extractor $1
        fitter $1 $EXTRACTED_DATA $EXTERNAL_FIT
else
        mkbounds 10
fi
```

24 AT expressions

One thing that you can't fake is chemistry.

Blake Shelton

You can build a *input file* from a *template* using @**expressions**. These are places where the symbol @ and the following characters are replaced with the coefficient values obtained by **GAFit**. The convention is as shown in Table 24.1.

Table 24.1: @expressions convention

@expression	example	description
@name(float valueA , float valueB)	@bondlenght(1.0,2.1)	replace with float values between valueA and valueB.
@name(float valueC , float valueD/dp)	@energy(1.0,2.1/3)	replace with float values between valueC and valueD using dp decimal places.
@name(integer valueA, integer valueB)	@option1(1,5)	replace with integer values between valueA and valueB.
@name(float valueA; float valueB; ...)	@angle(0.0;90.0;180.0;270.0)	pick one value from the list: valueA, valueB, ...

There are three types of @**expressions**:

d float, like $@distance(1.0, 2.3)$ to be replaced by a float from the interval: $[1.0, 2.3]$. Note the decimal point and the comma –and the optional slash for the decimal places–.

i integer, like $@index(1, 4)$ to be replaced by a integer value from the interval: $[1, 4]$. Note that there is not decimal point but there is a comma.

c choice, like $@choosefrom(0; 90; 180.0; 270)$ to be replaced by one float value picked up from the set $\{0, 90, 180, 270\}$. Note the semi colons.

An optional *format* completes the @**expressions** to resolve the problem to output fortran-like files with fixed formats:

- $@distance(\#\#.\#\#\#, 1.0, 2.3)$ to use a format like ***F6.3***.
- $@index(\#\#\#\#\#, 1, 4)$ to use a format like ***I5***.
- $@choosefrom(\#\#\#.\#, 0; 90; 180.0; 270)$ to use a format like ***F5.1***.

25 CHARMM module

> Research is what I'm doing when I don't know what I'm doing.
>
> *Wernher Von Braun*

Another feature of **GAFit** is the possibility to parametrize CHARMM –at least tested with version *c37b1*– as the external program.

The details of how **GAFit** works with an external interface –or external potential– are the same as explained in Section 21.1 with a final alternative approach. For clarity, details are printed again with specific modifications for this case.

25.1 External Interface

The *external interface* works as follows:

- **GAFit** generates a whole generation, where each individual is a coefficient vector.
- 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, and
  - *for each* individual,
    - 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.

This implementation uses the *bulk* option. So, 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. See Section 25.2.

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 *chmm.sh*, and the job is passing an entire generation of 100 coefficient vectors, the command line executed by the shell is:

```
$ chmm.sh 100
```

external input examples are given in Files 15.8 and 15.9. **external fit** examples are the Files 15.10 and 15.11

Note, as stated before in previous sections that: **GAFit** only evaluates if there is a command processor available –i.e. *sh*– and the **coefficients** value. No other checks are performed.

Autoconfigure

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

```
$ chmm.sh 0
```

With a "0" parameter, 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 CHARMM interface is shown below, File 25.1.

File 25.1: response generated by chmconfigurator

```

[job]
type:␣external␣bulk
coefficients:␣3
external␣input:␣charmm.input
external␣fit:␣charmm.fit
bounds:␣bounds.txt

[coefficient␣names]
tor1
mult1
phase1
```

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.

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__
```

The launching of the external program follows the guidelines developed for the MOPAC case, sections 21 and 22. Only the final details and the tools developed are distinct.

25.2 Interfacing with CHARMM

Interfacing with CHARMM is achieved using three tools, all of them written in **C**: **chmconfigurator**, **chmreference** and **chmrunner**.

The first two are used to configure the system in the first stage, Figure 25.3. The last, **chmrunner**, create the files needed, runs CHARMM and calculate the fits, Figure 25.2.

The trick here, is to use the CHARMM capabilities to write a suitable output to be processed by only one simple binary, **chmrunner**. We don't need here to extract data from complicated output files and process it to calculate the fit.

chmconfigurator is responsible for:

- answering the **GAFit** *external auto* configuration option as an *external bulk* type job.
- prepare the calculations analyzing the parameters template (**CHARMM_TEMPLATE**) and writing the results to the file *template-analysis*.
- create the *bounds.txt* file for **GAFit** use.
- create the *chmfinal-hint* file for **chmfinal** use.

chmreference is in charge of:

- extracting data from the CHARMM geometry files (**CHARMM_GEOMETRIES**) to a intermediate file, *reference-table*, with a format for easy retrieve by **chmrunner**.

  CHARMM_GEOMETRIES is the folder name where **chmreference** will search for geometry files, Figure 25.1.

  This step could be omitted if the *reference-table* file exist either hand made or from previous runs for exactly the same problem to fit.

chmrunner must:

- create the CHARMM's input file from the template.
- launch the calculations.
- evaluate the fitting.

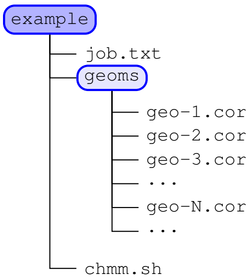


Figure 25.1: CHARMM GEOMETRIES folder.

- write the file with the fits to be read by **GAFit**.

chmfinal is in charge of:

Only one template is used to generate the files needed by CHARMM.

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

There are places, marked with an @**expression**, where the symbol @ and the following characters are replaced with the values obtained by **GAFit**. The convention is as shown in Table 25.1.

Table 25.1: @expressions convention

@expression	example	description
@name(float valueA , float valueB)	@bondlenght(1.0,2.1)	replace with float values between valueA and valueB.
@name(float valueC , float valueD/dp)	@energy(1.0,2.1/3)	replace with float values between valueC and valueD using dp decimal places.
@name(integer valueA, integer valueB)	@option1(1,5)	replace with integer values between valueA and valueB.
@name(float valueA; float valueB; ...)	@angle(0.0;90.0;180.0;270.0)	pick one value from the list: valueA, valueB, ...

See **AT expressions**, Chapter 24.

The File 25.2 is an example.

File 25.2: CHARMM_TEMPLATE: template.prm with formats

```
[...]
*␣␣type␣␣alpha−i␣␣␣␣␣N−i␣␣␣␣␣␣␣␣A−i␣␣␣␣␣␣␣␣G−i␣DA␣Symb␣␣␣Origin
[...]
*──────────────────────────────────────────────────────────────────────
␣␣␣␣1␣␣␣␣␣@alp1(#.###,0.9,1.5␣␣)␣␣␣␣␣@ni1(#.###,2.2,2.8)␣␣␣␣␣↩
      ↪@ai1(#.###,3.6,4.0)␣␣␣␣␣@gi1(#.###,1.0,1.5)␣−␣CR␣␣␣␣␣E94
␣␣␣␣2␣␣␣␣␣1.350␣␣␣␣␣2.490␣␣␣␣␣3.890␣␣␣␣␣1.282␣−␣C=C␣␣␣␣E94
␣␣␣␣3␣␣␣␣␣1.100␣␣␣␣␣2.490␣␣␣␣␣3.890␣␣␣␣␣1.282␣−␣C=O␣␣␣␣E94
[...]
```

Figure 25.2: CHARMM interface: normal operation

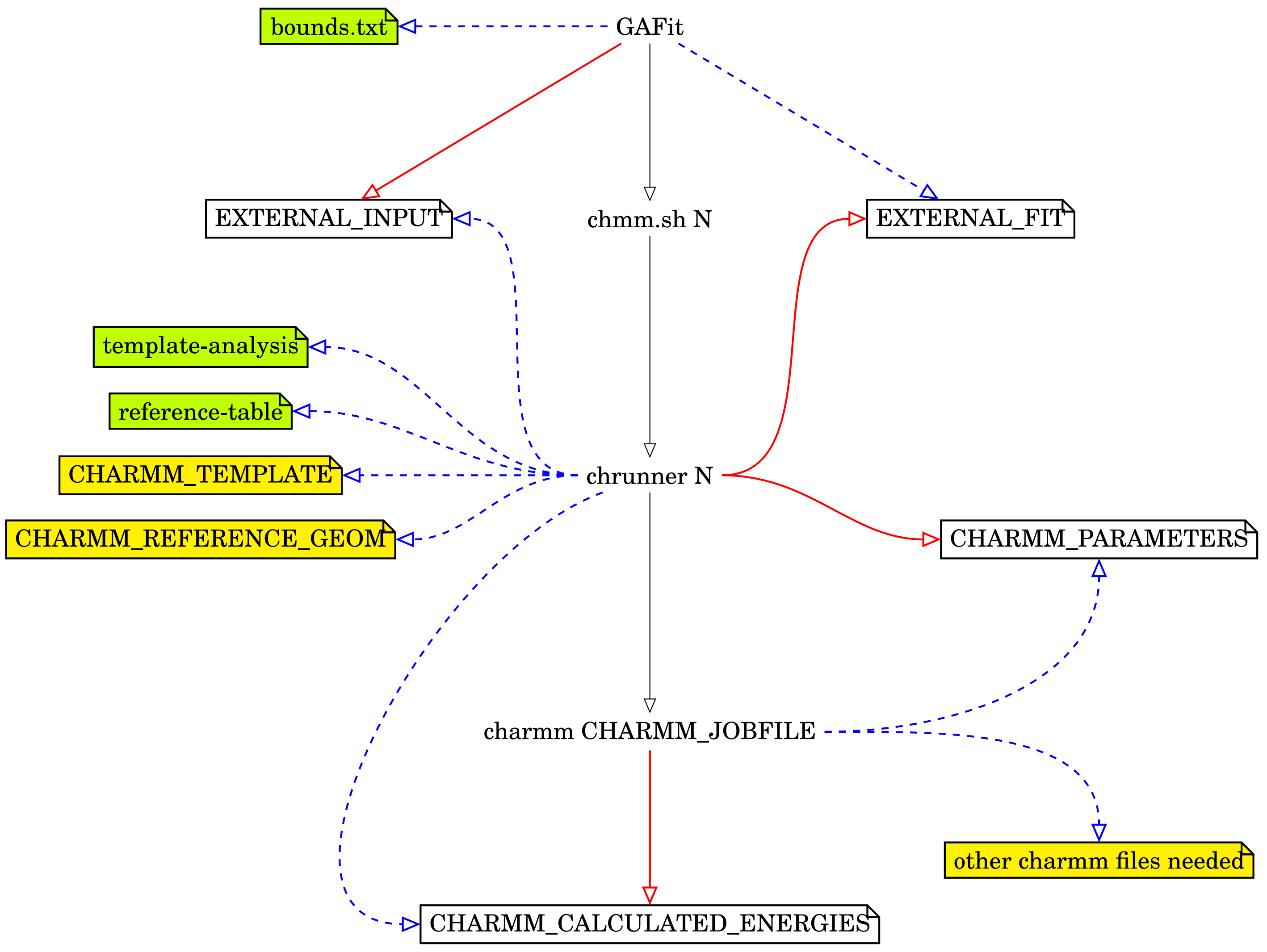
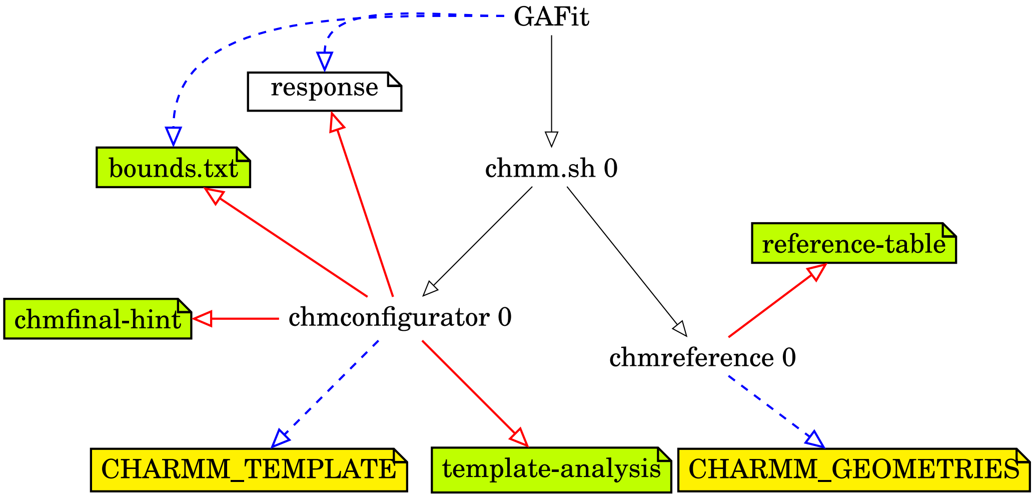


Figure 25.3: CHARMM: autoconfigure and job preparation



Here, is important to highlight the fact that the *bounds file* is generated from the template using the the values from the @**expression**'s. Also, the @**expression**'s names are used to create the **[coefficient names]** section in the *response* file –File 25.1–.

Figures 25.2 and 25.3 show the relations between programs and files:

- Dashed blue lines indicate that a tool uses the file as input.
- red lines indicates that a tool creates the file.
- **Black** lines indicate calls to execute a tool.
- Files filled in *yellow* indicate that they must be created or given by the user.
- Files filled in *lime* indicate that they are created in the first call to the **external program** –*chmm.sh* in this case– but used without modification along the rest of the calculations.

File 25.3: CHARMM_TEMPLATE: template.prm in charmm-example

```
*>CHARMM22 All-Hydrogen Parameter File for Proteins and Lipids <<
*>>>>> Includes phi, psi cross term map (CMAP) correction <<<<<<<
*>>>>>>>>>>>>>>>>>>>>>>>>>>   July, 2003   <<<<<<<<<<<<<<<<<<<<<<<<<<
* All comments to ADM jr. via the CHARMM web site: www.charmm.org
*                 parameter set discussion forum
*

[...]

C    CT1  NH1  C        0.2000  1   180.00 ! ALLOW PEP
                 ! ala dipeptide update for new C VDW Rmin
C    CT2  NH1  C        @tor1(0.,1.0) @mult1(1,5) @phase1(0.;180.00) ! parametrization
                 ! ala dipeptide update for new C VDW Rmin, adm jr., 3/3/93c
C    N    CP1  C        0.8000  3     0.00 ! ALLOW PRO PEP
                 ! 6-31g* AcProNH2, ProNH2, 6-31g*//3-21g AcProNHCH3 RLD 4/23/93
CA   CA   CA   CA       3.1000  2   180.00 ! ALLOW   ARO
                 ! JES 8/25/89
CA   CPT  CPT  CA       3.1000  2   180.00 ! ALLOW   ARO
                 ! JWK 05/14/91 fit to indole
[...]
```

@expressions here

There are environmental variables, shown in Table 25.2, which can be set to control the file names.

Table 25.2: Environmental variables

Variable	Default value	Tools
EXTERNAL_INPUT	charmm.input	**GAFit**, chmrunner
EXTERNAL_FIT	charmm.fit	**GAFit**, chmrunner
BOUNDS_FILE	bounds.txt	**GAFit**, chmconfigurator
CHARMM_TEMPLATE	template.prm	chmconfigurator
CHARMM_PARAMETERS	parameters.prm	chmrunner
CHARMM_JOBFILE[†]	fitting	chmrunner
CHARMM_EXECUTABLE	charmm	chmrunner
CHARMM_GEOMETRIES	geometries	chmreference
CHARMM_REFERENCE_GEOM	none	chmreference, chmrunner
CHARMM_CALCULATED_ENERGIES	calculated.energies	chmrunner

[†] the CHARMM_JOBFILE variable is used to generate a CHARMM_JOBFILE**.dat** file as input for CHARMM and a CHARMM_JOBFILE**.out** file for output. The command executed is:

charmm < CHARMM_JOBFILE**.dat** > CHARMM_JOBFILE**.out**

25.3 External command

File 25.4: job.txt in charmm-example

```
[job]
runs:            1
evaluations:     5000
type:            external auto
command:         chmm.sh

[print]
print runs: yes
```

GAFit only calls an external shell script: the name given in *job.txt*. In this case: *chmm.sh*, File 25.4.

There is a complete example in the folder *charmm-example* which can be examined in the File 25.5.

File 25.5: External: chmm.sh

```
#!/bin/sh

export EXTERNAL_INPUT="charmm.input"
export EXTERNAL_FIT="charmm.fit"
export BOUNDS_FILE="bounds.txt"
export CHARMM_TEMPLATE="template.prm"
export CHARMM_PARAMETERS="parameters.prm"

export CHARMM_GEOMETRIES="geoms"
export CHARMM_REFERENCE_GEOM="GEO-9.COR"
export CHARMM_CALCULATED_ENERGIES="calculated.energies"
export CHARMM_JOBFILE="fitting"
export CHARMM_EXECUTABLE="$HOME/CHARMM/c37a1dev/exec/gnu/charmm"


if [ "$1" -ne "0" ]
then
        chmrunner $1 1 4
```

```
else
        chmconfigurator $1
        chmreference $1
fi
```

A minimal implementation to *chmm.sh* due to the defaults, could be the File 25.6.

File 25.6: Minimal external chmm.sh

```
#!/bin/sh
export CHARMM_GEOMETRIES="geoms"
export CHARMM_EXECUTABLE="$HOME/CHARMM/c37a1dev/exec/gnu/charmm"

if [ "$1" -ne "0" ]
then
        chmrunner $1 1 4
else
        chmconfigurator $1
        chmreference $1
fi
```

25.4 chmconfigurator

chmconfigurator is a program written in C. The syntax is

```
chmconfigurator number-of-vectors
```

When **GAFit** calls the *external command* passing a “0” as first parameter, the **chmconfigurator** creates the file *response* and **GAFit** uses this information to configure itself –File 25.1–. This file is deleted the first time **chmrunner** runs. Also **chmconfigurator** creates the *bounds.txt* and *template-analysis* files.

The *template-analysis* is a summary from the CHARMM_TEMPLATE file. An example is File 25.7 and the format is shown in Table 25.3.

Table 25.3: template-analysis format

name	@expression	format	type	limits string	lower limit	upper limit
tor1	@tor1 (0. ,1.0)		d	0. ,1.0	0.	1.0
mult1	@mult1(1 ,5)		i	1,5	1	5
					GAFit integer	choice value
phase1	@phase1(0.;180.00)		c	0.;180.00	1	0.
					2	180.00

The **choice** type is handled by **GAFit** as integers. So, a set like 0;45;90;180 are translated to a **GAFit** integer coefficient with *bounds* between 1 and 4. There are a utility to do automatically the translation: **chmfinal**. See Section 25.7.

File 25.7: template-analysis file

```
tor1|@tor1(0.,1.0)||d|0.,1.0|0.|1.0|
mult1|@mult1(1,5)||i|1,5|1|5|
phase1|@phase1(0.;180.00)||c|0.;180.00|1|2|0.␣180.00
```

The data needed to create the *response* file is obtained from environmental variables and from the **CHARMM_TEMPLATE** file[1].

chmconfigurator expects a “0” as argument, if not it refuses to work.

25.5 chmreference

chmreference is a program written in C. The syntax is

```
chmreference number-of-vectors
```

chmreference creates the file *table-reference*–File 25.8– extracting data from geometry files. There are three columns: the geometry file name, the reference energy and the weight of the energy. Table 25.4 shows the three first lines from File 25.8.

File 25.8: table-reference file

```
GEO−1.COR␣−21.422200␣1.000000
GEO−10.COR␣−30.643500␣1.000000
GEO−11.COR␣−30.151300␣1.000000
[...]
GEO−48.COR␣−34.625700␣1.000000
GEO−49.COR␣−34.876600␣1.000000
GEO−5.COR␣−28.488700␣1.000000
GEO−6.COR␣−23.643300␣1.000000
GEO−7.COR␣−18.127800␣1.000000
GEO−8.COR␣−26.037500␣1.000000
GEO−9.COR␣−28.996000␣1.000000
```

Table 25.4: table-reference format

geometries file	reference energy	weight
GEO-1.COR	-21.422200	1.000000
GEO-10.COR	-30.643500	1.000000
GEO-11.COR	-30.151300	1.000000
...	...	...

In order to **chmreference** works, its is necessary to modify the geometry files to include data for reference energy and weight in the first line after a colon, as shown in Files 25.9 and 25.10. The first number after the first colon is interpreted as the energy –21.4222, in the example shown– and after the second colon as the weight. If there is no second colon and/or weight present, it is taken as 1.

File 25.9: geo-1.cor file

```
∗␣1␣:␣−21.4222
∗␣␣DATE:␣␣␣␣␣2/␣9/15␣␣␣␣␣17:33:51␣␣␣␣␣␣CREATED␣BY␣USER:␣user
∗
␣␣␣29
␣␣␣␣1␣␣␣␣1␣GLY␣␣CAY␣␣␣−2.20096␣␣−0.38688␣␣␣0.68947␣GLY3␣1␣␣␣␣␣␣↩
        ↪0.00000
```

[1] Number and name of the coefficients.

```
␣␣␣␣2␣␣␣␣1␣GLY␣␣HY1␣␣␣−2.03907␣␣−0.96864␣␣−0.24205␣GLY3␣1␣␣␣␣␣␣↩
        ↪0.00000
␣␣␣␣3␣␣␣␣1␣GLY␣␣HY2␣␣␣−2.76856␣␣␣0.53806␣␣␣0.45524␣GLY3␣1␣␣␣␣␣␣↩
        ↪0.00000
␣␣␣␣[...]
```

File 25.10: geo-1.cor file with weight set

```
*␣1␣:␣−21.4222␣:␣1.2
*␣␣DATE:␣␣␣␣␣2/␣9/15␣␣␣␣␣17:33:51␣␣␣␣␣␣CREATED␣BY␣USER:␣user
*
␣␣␣29
␣␣␣␣1␣␣␣␣1␣GLY␣␣CAY␣␣␣−2.20096␣␣−0.38688␣␣␣0.68947␣GLY3␣1␣␣␣␣␣␣↩
        ↪0.00000
␣␣␣␣2␣␣␣␣1␣GLY␣␣HY1␣␣␣−2.03907␣␣−0.96864␣␣−0.24205␣GLY3␣1␣␣␣␣␣␣↩
        ↪0.00000
␣␣␣␣3␣␣␣␣1␣GLY␣␣HY2␣␣␣−2.76856␣␣␣0.53806␣␣␣0.45524␣GLY3␣1␣␣␣␣␣␣↩
        ↪0.00000
␣␣␣␣[...]
```

You can set a reference energy using the environment variable CHARMM_REFERENCE_GEOM, so the energies are normalized as shown in File 25.11.

File 25.11: table-reference file normalized with geom-9.cor

```
GEO−1.COR␣7.573800␣1.000000
GEO−10.COR␣−1.647500␣1.000000
GEO−11.COR␣−1.155300␣1.000000
[...]
GEO−48.COR␣−5.629700␣1.000000
GEO−49.COR␣−5.880600␣1.000000
GEO−5.COR␣0.507300␣1.000000
GEO−6.COR␣5.352700␣1.000000
GEO−7.COR␣10.868200␣1.000000
GEO−8.COR␣2.958500␣1.000000
GEO−9.COR␣0.000000␣1.000000
```

chmreference like **chmconfigurator** expects a “0” as argument, if not it refuses to work.

25.6 chmrunner

chmrunner is a program written in C. The syntax is

```
chmrunner number-of-vectors index-column energy-column
```

The parameters must be:

number-of-vectors the number of coefficient vectors, which are written in the file EXTERNAL_INPUT.

index-column the column in the file CHARMM_CALCULATED_ENERGIES which is the index: A string equal to the first column string in the file *reference-table*. We use here the geometry file names. Column 1 in File 25.12.

energy-column the column number in the file CHARMM_CALCULATED_ENERGIES corresponding to the calculated energy. Column 4 in File 25.12.

When it is called, **chmrunner**:

- loads the EXTERNAL_INPUT file.
- for each vector, **chmrunner**:
  - creates a CHARMM_PARAMETERS file from CHARMM_TEMPLATE replacing the @**expressions** by the vector values
  - launches a CHARMM job using CHARMM_EXECUTABLE and CHARMM_JOBFILE as a parameter. Using the example configuration, the system call is like:

    ```
    $HOME/CHARMM/c37a1dev/exec/gnu/charmm  < fitting.dat > fitting.out
    ```

    The extensions *.dat* and *.out* are added by **chmrunner**. An example is the File 25.13.
  - examines the results loading the file CHARMM_CALCULATED_ENERGIES created by the CHARMM job.
- finally, after processing all the coefficient vectors, writes the EXTERNAL_FIT file with all the fits.

File 25.12: calculated-energies file example

```
␣GEO-1.COR␣-2.735957E-03␣-3.231801E-04␣-22.221
␣GEO-2.COR␣0.426072␣30.3838␣-24.2362
␣GEO-3.COR␣0.36839␣60.3622␣-27.9524
␣GEO-4.COR␣8.464627E-02␣90.0958␣-30.0975
␣GEO-5.COR␣-0.281997␣119.692␣-29.2872
␣GEO-6.COR␣-0.655632␣149.257␣-24.4397
␣GEO-7.COR␣-6.443053E-03␣179.996␣-18.9246
␣GEO-8.COR␣30.7522␣0.315584␣-26.8306
␣GEO-9.COR␣30.6181␣30.2866␣-29.788
[...]
```

The CHARMM_JOBFILE –File 25.13– must be coded to write the CHARMM_CALCULATED_ENERGIES file in each run – File 25.12– with a column to use as index to check against the *table-reference* file and the energy. As shown, other data can be printed too in this file. In this example, **chmrunner** reads the first –geometry file name used as index– and the fourth column –energy value–.

If is set the *reference geometry* –CHARMM_REFERENCE_GEOM–, its calculated value is used to normalize the calculated values like as the reference geometry energy is used to normalize the *table-reference* file.

The fit is calculated as:

$$fit = \sum_i \left[NormalizedCalculatedE_i - NormalizedTableReferenceE_i\right]^{2.0} * weight_i$$

File 25.13: charmm job example:fitting.dat

```
* gly3 : fitting torsional terms fir phi (C-N-CA-C) and psi (N-CA-C=O) dihedrals in last residue.
* C-N-CA-C
* N-CA-C=O

bomlev -5

open unit 1 card read name top_all36_prot_lipid.rtf
read RTF card unit 1

open unit 2 card read name parameters.prm
read PARA card unit 2

! read the psf and coordinate file
read psf card name gly3.psf
!read coor card name gly3.optc.crd

set CTR 1
set loopsize 49
! Loop for generating conformations around phi and psi
set 1 19
set 2 21
set 3 23
set 4 26
set 5 27

!----loop through the geometries------

!------set up a file to keep track of energies----------
OPEN WRITE CARD UNIT 21 name calculated.energies

LABEL LOOP

! overwrite the coordinates by reading a new set - this requires bomlev is set appropriately!
bomlev 0
open unit 29 card read name geoms/geo-@CTR.cor
read coor card unit 29
close unit 29

    energy                ! recompute the energy without restraints

    quick @1 @2 @3 @4
    set psiangle ?phi
    quick @2 @3 @4 @5
    set phiangle ?phi
    set name geo-@CTR.cor

    WRITE TITLE UNIT 21   ! write out the current restraint distance and energy
    * @name @psiangle @phiangle ?ENER
    *

INCR CTR
IF @CTR LT @loopsize.5  GOTO LOOP

close unit 21

stop
```

```
!------set up a file to keep track of energies and dihedral angles ---------
OPEN WRITE CARD UNIT 21 name  energies.dat

! Loop for generating conformations around phi and psi
set 1 19
set 2 21
set 3 23
set 4 26
set 5 27
set delta 60. ! increment for rotational angle
set i 1
set apsi 0.

label looppsi
  set aphi 0.
  label loopphi

    cons dihe bynum @1 @2 @3 @4 force 1000. min @apsi peri 1
    cons dihe bynum @2 @3 @4 @5 force 1000. min @aphi peri 1

    ! Minimization
    mini sd nstep 200
    mini abnr nstep 1000 nprint 500 tolg 0.01

    cons cldh

    energy                ! recompute the energy without restraints

    quick @1 @2 @3 @4
    set psiangle ?phi
    quick @2 @3 @4 @5
    set phiangle ?phi

    WRITE TITLE UNIT 21   ! write out the current restraint distance and energy
    * @psiangle @phiangle ?ENER
    *

!ioform extended
! Optimized geometry
!open write unit 10 card name geo-@i.pdb
!write coor unit 10 pdb

    open unit 10 card write name geo-@i.cor
    write unit 10 COOR card
    * @i : ?ener
    *

    close unit 10

    incr i by 1
    incr aphi by @delta
    if @aphi .le. 180. then goto loopphi
  incr apsi by @delta
  if @apsi .le. 180. then goto looppsi

close unit 21

stop
```

chmrunner refuses to work if any of the parameters passed is zero or a negative number.

25.7 chmfinal

chmfinal is a program written in C. The syntax is

```
chmfinal
```

chmfinal analyzes the *best.txt* file to print the results translating the **GAFit** integer coefficients to the corresponding *choice* values and run once **CHARMM** using the best coefficients to compare energies. To do this, the file *chmfinal-hint* must be present. This file is created or ***overwritten*** by **chmconfigurator**.

```
$ cat best.txt
0.171372950995
1.000000000000
2.000000000000

Fitness: 0.052448000000

$ chmfinal
#
#FINAL EVALUATION
#
                          COEFFICIENTS

        0                 ang1:        0.171372950995
        1                  per:        1.000000000000
        2                phase:        180

                       Fitness: 0.052448000000
#
# EVALUATION
#
#
#   Geometry     Reference   Calculated   Difference
#==============  ==========  ===========  ===========
  GEO-1.COR        7.573800     7.577900     -0.0541%
  GEO-10.COR      -1.647500    -1.647300      0.0121%
  GEO-11.COR      -1.155300    -1.155100      0.0173%
  GEO-12.COR       2.582800     2.583000     -0.0077%
  GEO-13.COR       7.320600     7.320800     -0.0027%
  GEO-14.COR       4.219200     4.219000      0.0047%
  GEO-15.COR      -2.859800    -2.870100     -0.3602%
  GEO-16.COR      -4.086100    -4.096400     -0.2521%
  GEO-17.COR      -4.005300    -4.015600     -0.2572%
  GEO-18.COR      -2.446100    -2.456400     -0.4211%
  GEO-19.COR       0.128900     0.118500      8.0683%
[...]
```

26 Mvariable module

Beware of bugs in the above code; I have only proved it correct, not tried it.

Donald Knuth

The **mvariable** module is a sole C program with all the needed features to run multivariate fitting using an *analytical formula* and a file with the associated data to fit. This module is an application of the FPU code. See Section 19.

26.1 External interface

The *external interface* works as shown in Sections 21.1, 21.2, 25.1 and 25.2. This implementation uses the *autoconfigure* feature (Sections 21.1 and 25.1). See File 26.1.

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__
```

26.2 Interfacing with mvariable

Figures 26.1 and 26.2 shown the relations between programs and files:

- Dashed blue lines indicate that a tool uses the file as input.
- red lines indicates that a tool creates the file.

Figure 26.1: Mvariable: autoconfigure

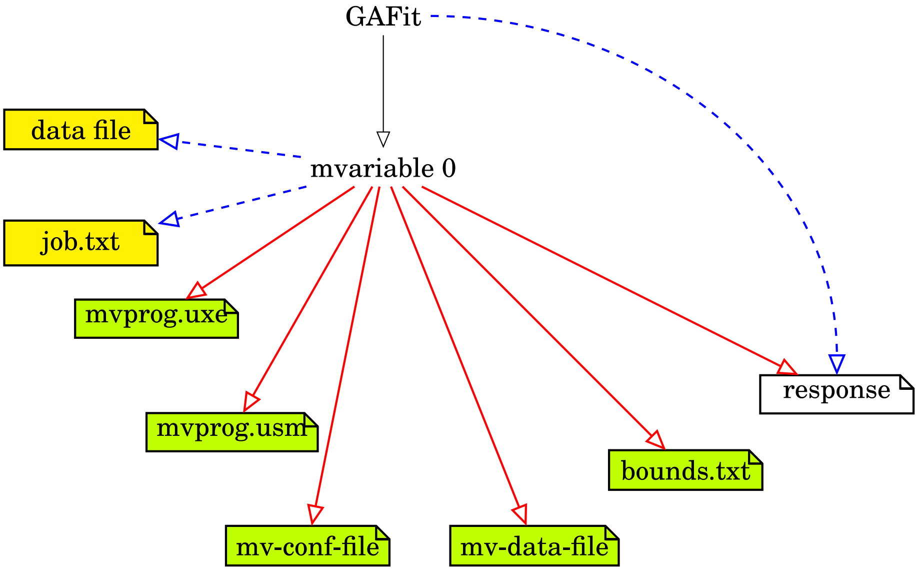
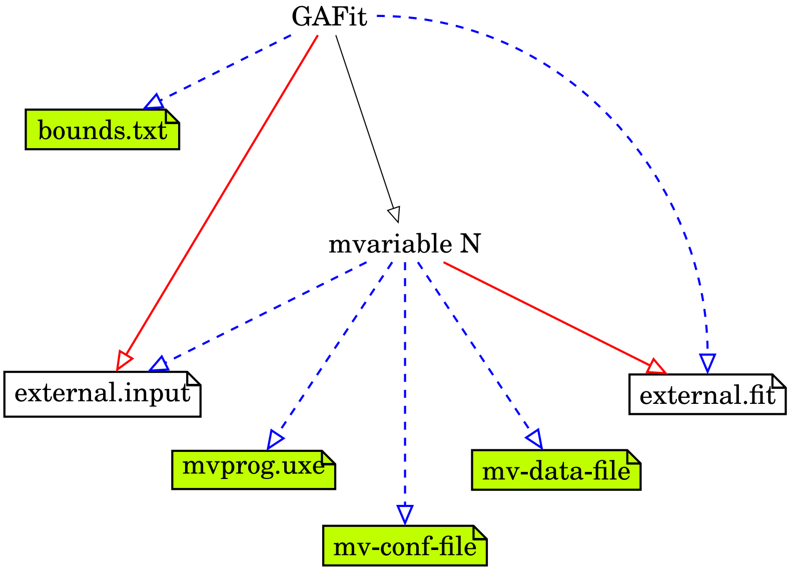


Figure 26.2: Mvariable: normal operation



- **black** lines indicate calls to execute a tool.

- Files filled in *yellow* indicate that they must be created or given by the user.

- Files filled in *lime* indicate that they are created in the first call to **mvariable** but used without modification along the rest of the calculations.

mvariable uses the *job.txt* for configuration. Configuration for File 26.1 and data for File 26.2 are borrowed from the Multiple Regression example, http://simon.cs.vt.edu/SoSci/converted/MRegression/ This is an example problem for Social Sciences taken from Virginia Tech's SABLE [12] where some data are fit to the *multiple regression* equation:

$$\textbf{Predictedsalesperformance} = a + b * Intelligence + c * Extraversion$$

File 26.1: mvariable job.txt file

```
[job]
runs:→→1
evaluations:→5000
type:␣␣␣␣␣␣␣␣␣␣␣external␣auto
command:␣␣␣␣␣␣␣␣mvariable

[print]
print␣runs:␣yes

[multi␣variable]
coefficients:␣a(0.0,2000.0),␣b(0.0,100.0)␣,c(0.0,100.0)
fit␣variable:␣fit

data␣file:␣predictedsalesp.txt
data␣columns:␣salesperson,␣intelligence,␣extroversion,␣sales
data␣headers:␣2

expression:␣mv␣test

[mv␣test]
psperform␣=␣a␣+␣b*intelligence␣+c␣*␣extroversion;

fit=(psperform-sales)^2
```

The *job.txt* is shared between **GAFit** and the **mvariable** program. It has a new section **[multi variable]** read by **mvariable** to configure the problem to resolve. The configuration parameters are summarized in the Table 26.1. All of them must be set.

File 26.2: mvariable data file

```
#salesperson␣Intelligence␣Extroversion␣sales
#
1→89→21→2625
2→93→24→2700
3→91→21→3100
4→122→23→3150
5→115→27→3175
6→100→18→3100
7→98→19→2700
8→105→16→2475
9→112→23→3625
10→109→28→3525
11→130→20→3225
12→104→25→3450
13→104→20→2425
14→111→26→3025
15→97→28→3625
16→115→29→2750
17→113→25→3150
```

```
18———→88———→23———→2600
19———→108———→19———→2525
20———→101———→16———→2650
```

Table 26.1: Multi variable section parameters

Parameter	Type	Comment
coefficients	string	List of coefficients with their limits. The syntax is the same as shown in page 185 and in Table 25.1 without the @ symbol and without the *format* part.
data file	string	Name of the data file to fit
data columns	string	List of column names present into the data file. This names can be used in the *expression*.
data headers	integer	Number of lines to skip into data file
expression	string	Name of the section where is the *expression* used to fit data from *data file* using the *coefficient names*, the *data file column names*, the *fit variable* and any intermediate variables.
fit variable	string	name of the variable into the expression section used as the calculation result.

Using this information, **mvariable** configures **GAFit** and build the *bounds.txt* file. For each line from *data file* the *expression section* is evaluated and the fit *variable* is obtained. The *fit* is the sum of all the *fit variables* over the whole data file.

An example output running **GAFit** with the Files 26.1 and 26.2 is shown below:

```
+-------------------------------------------------------------------------------+
|     GAFit 1.3d Build:314                                                      |
|     Fri Mar  9 16:20:27 2018                                                  |
+-------------------------------------------------------------------------------+

[...]

Mvariable Analysis
============================
external inp: external.input
external fit: external.fit
bounds file : bounds.txt
coefficients: a(0.0,2000.0), b(0.0,100.0) ,c(0.0,100.0)
fit variable: fit
data file    : predictedsalesp.txt
columns      : salesperson, intelligence, extroversion, sales
headers      : 2
expression   : mv test
print code   : no

psperform = a + b * intelligence + c * extroversion;

fit = (psperform - sales)^2

+-------------------------------------------------------------------------------+
|     Settings for job                                                          |
+-------------------------------------------------------------------------------+
|     Command:[mvariable]                                                       |
|     Bounds:[bounds.txt]                                                       |
|     External input:[external.input]                                           |
|     External fit:[external.fit]                                               |
|     Total coefficients: 3                                                     |
|     Print options: runs yes, ga settings no                                   |
+-------------------------------------------------------------------------------+
|     run: 1                                                                    |
|     this run's seed:1520608828                                                |
+-------------------------------------------------------------------------------+

Eval.                  Best fit.
------------------------------------
100                  2.29229e+06
```

```
200                2.17067e+06
300                1.96761e+06
400                1.93755e+06
500                1.93755e+06
600                1.90796e+06
700                1.901e+06
[...]
4800                1.87477e+06
4900                1.87477e+06
5000                1.87477e+06
5000                1.87477e+06


#
#Results
#
1          a   +1010.454994214965
2          b   +8.235963226679
3          c   +48.923735630054
```

26.3 mvtest

A command **mvtest** is provided to test the *best.txt* –the default *coefficient file* parameter– coefficients with the data from *data file*. The configuration is taken from the *job.txt* file.

The syntax is:

```
$ mvtest -h

mvtest v0.1 (c)GAFit toolkit - 2015
        Usage: mvtest [coefficient-file]
```

Below is a **mvtest** run using the above results for the *best.txt* file. There are 20 points in the *data file*, each of one is used to calculate the *fit variable* –third column– and the sum is the overall *fit* shown above –1.874780946116e+06–. A better result is obtained using a higher number of *evaluations*.

```
$ mvtest
point     0:          20633.27276
point     1:          64004.74303
point     2:          99217.54889
point     3:            103.94610
point     4:          12025.52765
point     5:         154400.77546
point     6:           1699.85762
point     7:          29553.01349
point     8:         321534.59538
point     9:          57077.50398
point    10:          29354.08717
point    11:         127051.03437
point    12:         172950.47541
point    13:          31112.05990
point    14:         191076.30652
point    15:         404158.77638
point    16:            296.46434
point    17:          68399.11741
point    18:          88848.19284
point    19:           1283.64736
sum of fits:        1874780.94606
```

27 Generic module

I don't believe in astrology; I'm a Sagittarian and we're skeptical.

Arthur C. Clarke

The **generic module** is intended to generalize that we have already learned in the previous modules. Its target is to interface a broad range of external programs with a litte effort from the user.

Some of the key features are:

- It can parameterize more than one input file at once using many templates. The coefficients to parameterize could be sparse and repeteated along the templates.
- It can run a fixed[1] number of individual calculations in parallel.
- Each individual calculation could be run in its own folder and deleted afterwards.
- Fitting data and other interesting information were extracted out from the run folder by **gfitter** tool before its deletion.
- The fit compares a list of reference values with a list of calculated values using any valid implementation. Could be test points or some of cases or specific problems... The implementation is open but allways a comparison of two lists of floating point numbers are done.
- The information gathered about the whole calculation is shown in the file *report_best.txt* where the best result into each distance slice –from the current best set of coefficients– are presented.

[1] No so as the MOPAC module.

27.1 External interface

The *external interface* works as shown in Sections 21.1, 21.2, 25.1 and 25.2. This implementation uses the *autoconfigure* feature, Sections 21.1, 25.1.

Stopping an external job

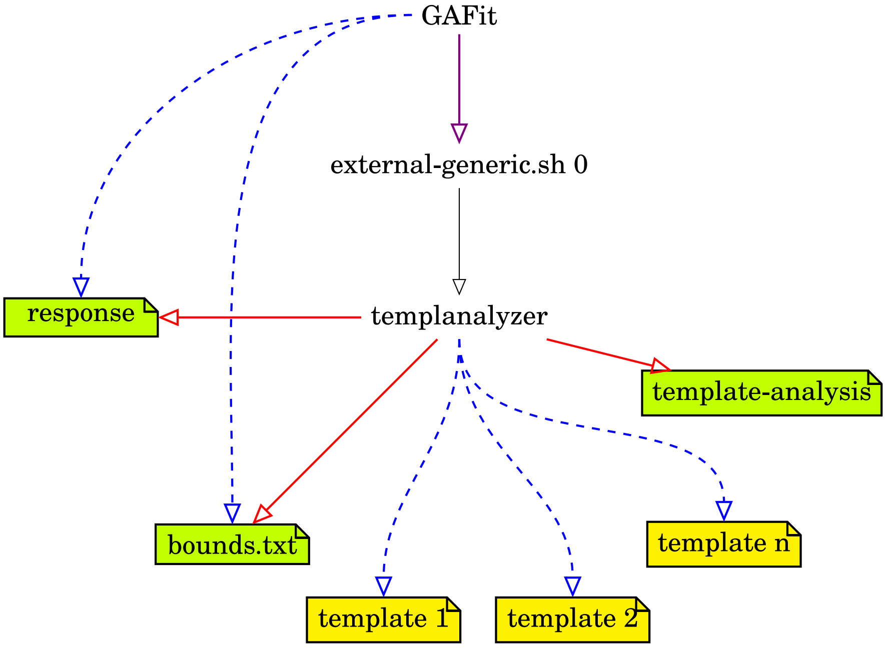
You can stop a running job writing a **stop file** in the folder where it is running. The **stop file**'s name is **__STOP__**, and the text it contains is whatever you want.

```
$ echo ``stop job''> __STOP__
```

27.2 Interfacing with generic

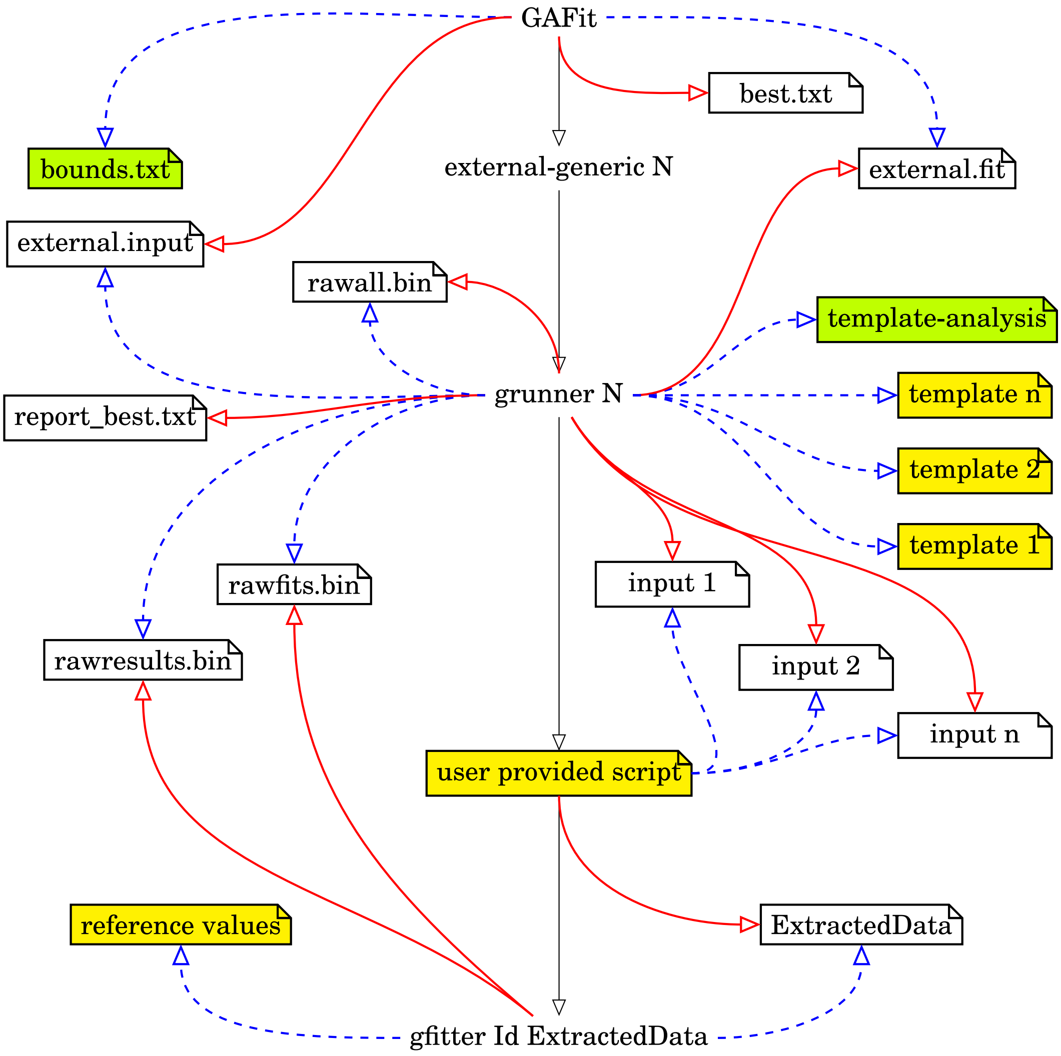
Figure 27.1 –during the autoconfigure phase– and Figure 27.2 –running the calculations– show the relations between programs and files.

Figure 27.1: Generic module: autoconfigure



- Dashed blue lines indicate that a tool uses the file as input.
- red lines indicates that a tool creates the file.
- **black** lines indicate calls to execute a tool.
- **violet** lines show that **GAFit** creates the file and intermediately execute it.

Figure 27.2: Generic module: normal operation



- Files filled in *yellow* must be created or given by the user.
- Files filled in *lime* are created in the first call but used without modification along the rest of the calculations.

The Table 27.1 shows the files used in the above figures.

27.3 The example

This example is included from the distribution in the *generic* folder into *simple-mode-examples* directory. We shall use this example to explain the **generic module**'s behaviour.

Table 27.1: Generic module files. User provided files are in yellow , one time files created by **GAFit** in lime .

File	Description
job.txt	Configuration file. Some of the generated files are built using this information.
external-generic.sh	**Automatically generated by GAFit**. It glues together the tools needed to acomplish the task.
templanalyzer	It analyses templates and builds a combined *template-analysis* file with the results.
grunner	It actually runs a bunch of individual jobs –*a user provided script*– in parallel. It builds the input files needed from templates using info from *template-analysis* file.
user provided script	It runs each individual calculations on input data and generates output extracting usable information to *ExtractedData* file and calls **gfitter** to evaluate it.
gfitter	Using *ExtractedData* and the *reference values* it evaluates the results and save some interesting data for statistical accounting.
response	A **generated file** to automatically configure **GAFit**.
template-analysis	It is a **generated file** by **templanalyzer**, summarizing the types and diverse information about the coefficients to take into account.
bounds.txt	It is a **generated file** to establish the bounds for each variable.
template 1, template 2, . . . , template n	Template sources for one individual job input files.
external.input	The individual coefficients values generated by **GAFit**.
external.fit	Evaluation results.
input 1, input 2, . . . , input n	Input files to one individual job.
reference values	A list of reference values to compare to.
ExtractedData	A list of useful data from calculations, ready to be evaluated against *reference values*.
report_best.txt	Information about the distribution of individuals.
rawall.bin	Intermediate results between generations.
rawfits.bin	Fits for each individual in machine binary format. $fit = (calculated - reference\ value)^2 * weight$
rawresults.bin	Evaluation of each individual: test points with their calculated values in machine binary format.

Job configuration

The example *job.txt* file is shown in File 27.1 where the specific configuration options from Table 27.2 are used. This is a simple job file with only one template.

If there is more than one template, the coefficient set names and their limits are taken from the @expressions and passed to the genetic algorithm.

For example, if we have 3 templates with 2, 5 and 3 @expressions respectively, we have 10 coefficients in each individual from the genetic algorithm. This @expressions generate also the *bounds.txt* file with the upper and lower limit of each coefficient and their own type.

More in @expressions: Section 24.

Taking into account this, the Job's configuration options meanings are:

application Module class. To use this module it must be **generic**.

ncores Number of *individual* calculations running in parallel.

executable Shell script or a program –provided by the user– to run for each *individual* calculation.

template List of templates. A set of templates generates a set of input files to the **executable** program or shell script, so in these input files this module will accommodate the whole coefficient set as stated from the @expressions used into them.

reference values List of values to compare with a test input to evaluate the fit.

Table 27.2: Configuration options to generic module in job.txt file

Option	Default value	Meaning
ncores	1	Number of parallel jobs. One per set of coeffcients.
template	template	One or more templates to process separated by white space.
executable	must be set, no default value	A user provided script to run per coefficient set (individual).
reference values	reference.values	Reference data to compare with.

The parameter **population** was not specified, so its value is 100 –See Section 15.1–.

File 27.1: job.txt, using generic module

```
[job]
runs:		1
evaluations:	5000
application: generic
ncores: 1
executable: ./genericscript.sh
template: template
reference_values: reference.values

[print]
print_runs: yes
```

In this file we configure **GAFit**. You can see it in File 27.1.

The template

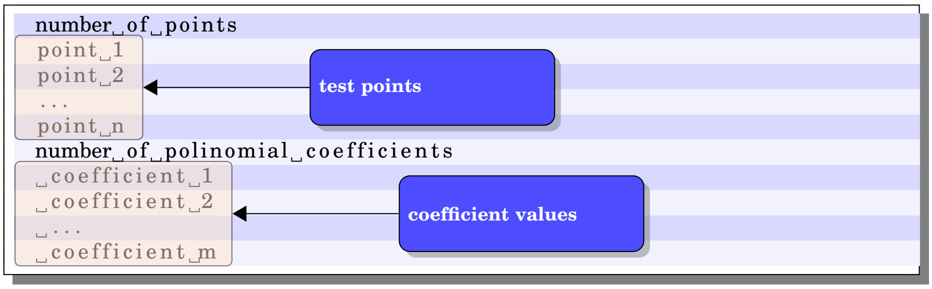
The File 27.2 shows the template used in this example where a piece of the text are represented by @expressions. See Section 24. This expressions will be replaced by **GAFit** with the coefficient values from genetic algorithm in any template included in the *job.txt* file.

File 27.2: template

```
7
-3
-2
-1
0
1
2
3
5
 @a(-10.,10.0/3)
 @b(-10.,10.0/3)
 @c(-10.,10.0/3)
 @d(-10.,10.0/3)
 @e(-10.,10.0/3)
```

The input file generated from this template shall be read by the program **testgeneric**. **testgeneric** expects the format of File 27.3. The values calculated, using the test points as input, will be compared with the reference values as the reference values are the expected values for those test points.

File 27.3: input file format



```
number_of_points
point_1
point_2
...
point_n
number_of_polinomial_coefficients
 coefficient_1
 coefficient_2
 ...
 coefficient_m
```

We need some well known **test points** with their *expected* values –the reference values– to calculate the fit. The **generic module** doesn't impose where this values must be due the fact that they are not necessary to calculate the fit. **GAFit** only needs their *expected* values –the reference values– and the values calculated by **testgeneric** using the **test points**.

Table 27.3: Test points and their corresponding reference values from the example.

	Reference values file		
Test Point	name	value	weight
-3	p1	40	1
-2	p2	0	1
-1	p3	0	1
0	p4	4	1
1	p5	0	1
2	p6	0	1
3	p7	40	1

In this example, we include the test points in the template where they are read from the test program to obtain the calculated values. Note that the test points are in the input file generated from the template and their expected values are in the *reference.values* file. They must be correlated: first test point expected value with the first reference value, second test point with the second reference value, and so on.

Other options are possible upon the type of program to parameterize behavior.

The reference values

The reference values are the values obtained from the known set of test points to check the goodness of the model. An example is shown in File 27.4 and the format is in File 27.5.

There are three columns:

- the reference value
- the weight
- the name of the reference

File 27.4: *reference.values* file

```
␣40␣␣␣␣␣1␣␣␣␣␣p1
␣0␣␣␣␣␣␣1␣␣␣␣␣p2
␣0␣␣␣␣␣␣1␣␣␣␣␣p3
␣4␣␣␣␣␣␣1␣␣␣␣␣p4
␣0␣␣␣␣␣␣1␣␣␣␣␣p5
␣0␣␣␣␣␣␣1␣␣␣␣␣p6
␣40␣␣␣␣␣1␣␣␣␣␣p7
```

File 27.5: reference values file format

```
value1		weight1		name1
value2		weight2		name2
value4		weight3		name3
 ...		...		...
valuen		weightn		namen
```

The configuration and calculation phases

GAFit calls a script named **external-generic.sh** –File 27.6– which is automatically generated by **GAFit** and is used, for the first time, to configure itself –Figure 27.1– and for every generation of coefficients –Figure 27.2– to do the calculations.

File 27.6: external-generic.sh

```sh
#!/bin/sh
export INPUT_TEMPLATE="template"
export EXTERNAL_INPUT=external.input
export EXTERNAL_FIT=external.fit
export N_CORES=4
export EXTERNAL_EXECUTABLE=./genericscript.sh
export REFERENCE_VALUES=reference.values

if [ "$1" -eq "0" ]
then
	templanalyzer
else
	grunner $1
fi
```

templanalyzer

This script is called with a "0" as argument on the very first moment running the **templanalyzer** program which configures **GAFit** and it analyzes the templates creating the files needed for the long run.

The templates go through the environment variable INPUT_TEMPLA TE as shown in File 27.6 or in the command line arguments. The second takes precedence.

grunner

After the first run, the script –which is called with an argument distinct of "0": the number of individuals in the generation– executes the **grunner** program which is in the duty to calculate each generation of individuals –sets of coefficients– generated by **GAFit**.

grunner prepares, from the templates, the input files replacing each @expression with the correspondent coefficient for every template and for every individual.

The input files are in the form "X.template", where "X" is one or more upper case letters –see 15.4– and "template" is the name of the template used to create the file.

So, if we have two templates "t0" and "t1", the input files are: A.t0, A.t1, B.t0, B.t1 . . . AA.t0, AA.t1, . . . and so on. Below you can see some of the files generated by **grunner** in the example:

```
$ ls
A.data          AW.data         BS.data         CO.data         R.data
A.output        AW.output       BS.output       CO.output       R.output
A.template      AW.template     BS.template     CO.template     R.template
AA.data         AX.data         BT.data         CP.data         S.data
AA.output       AX.output       BT.output       CP.output       S.output
AA.template     AX.template     BT.template     CP.template     S.template
[...]
```

Once the input files are ready, **grunner** calls for each individual the **user provided script** –Table 27.2– passing the input file names –A, B, C, . . . , AA, AB, . . . – as argument.

The user provided script

The example's **user provided script** is shown in File 27.7. The script uses its argument to access the files produced for each individual executing the commands needed to accomplish the task and lately the **gfitter** program to account the results.

File 27.7: Example's user provided script: genericscript.sh

```
#!/bin/sh
#echo "-------------------- begin $1 ----------------------"
./testgeneric < $1.template  > $1.output
./extractdata < $1.output > $1.data
gfitter $1 $1.data
rm $1.template $1.output $1.data
#echo "------ end $1 ------ "
```

If the argument is AA, the **genricscript.sh** executes the commands:

```
./testgeneric < AA.template > AA.output
./extactdata  < AA.output > AA.data
gfitter AA AA.data
rm  AA.template AA.output AA.data
```

The **testgeneric** program reads the file *AA.template* created by **grunner** from the template "*template*" –File 27.2–, producing the file *AA.output*.

File 27.8: *AA.template* file

```
7
−3
−2
−1
0
1
2
3
5
␣−4.877000
␣−3.636000
␣−2.432000
␣0.752000
␣0.833000
```

File 27.9: *AA.output* file

```
processing␣input
31.312000
-0.021000
-3.592000
-4.877000
-9.360000
-2.533000
50.104000
done
```

extractdata –File 27.10– processes this file and produces *AA.data* stripping out the first and last line. This was unnecessary if **testgeneric** does not generate those lines. **extractdata** is used only to increase the difficulty of a very simple system.

File 27.10: extractdata.c

```c
/*
   (c) GAFit Toolkit $Id: extractdata.c 378 2019-12-04 17:52:09Z ro $
   */

#if HAVE_CONFIG_H
#include <config.h>
#endif

#include <stdio.h>
#include <stdlib.h>

#define MAXLINE 200

char *
newLine ()
{
  return malloc (sizeof (char) * MAXLINE);
}

int
main (void)
{
  int counter = 0;
  char **list = NULL;
  char *line = newLine ();

  while (1)
    {
      fgets (line, MAXLINE, stdin);
      if (feof (stdin))
        break;
      counter++;
      list = realloc (list, sizeof (char *) * (counter));
      list[counter - 1] = line;
      line = newLine ();
    }
  for (int i = 1; i < counter - 1; i++)
    {
      printf ("%s", list[i]);
    }
}
```

File 27.11: *AA.data* file

```
31.312000
-0.021000
-3.592000
-4.877000
-9.360000
-2.533000
50.104000
```

The example program: testgeneric

The program used in the example is shown in File 27.12. This program reads from *standard input* and writes to *standard output*.

It calculates a polynomial:

$$y = \sum_{0}^{i} a_i x^i$$

File 27.12: testgeneric.c

```c
/*
 (c)GAFit toolkit $Id: testgeneric.c 378 2019-12-04 17:52:09Z ro $
*/
#if HAVE_CONFIG_H
#include <config.h>
#endif

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

#define MAXLINE 100

#define NINT 1
#define NDOUBLE 2

#define OUTPUT_EXT ".output"
#define INPUT_EXT ".input"

union Number
{
  int n;
  double d;
};

typedef union Number NUMBER;

NUMBER
getThing (FILE * f, int type)
{
  char line[MAXLINE + 1];
  NUMBER number;
  number.n = 0;
  while (fgets (line, MAXLINE, f) != NULL)
    {
      char *p = line;
      while (*p == ' ' || *p == '\t')
        p++;
      if (*p == '\r' || *p == '\n')
        continue;
      switch (type)
        {
        case NINT:
          sscanf (line, "%d", &number.n);
          return number;
        case NDOUBLE:
          sscanf (line, "%lf", &number.d);
          return number;
        }
      break;
    }
  return number;
}

int
getInt (FILE * f)
{
  return getThing (f, NINT).n;
}

double
getDouble (FILE * f)
{
  return getThing (f, NDOUBLE).d;
}

double
func (double x, double a[], int n)
{
  double ret = 0;
  int i;
  for (i = 0; i < n; i++)
    {
      ret += a[i] * pow (x, (double) i);
    }
  return ret;
}

int
main (int argc, char **argv)
{
  double *coefs;
  double *points;
  double *yf;

  int ncoefs, npoints;

  FILE *f;

  f = stdin;
  npoints = getInt (f);
  points = (double *) malloc (npoints * sizeof (double));
  yf = (double *) malloc (npoints * sizeof (double));

  for (int i = 0; i < npoints; i++)
    {
      points[i] = getDouble (f);
```

```
 99     }
100
101   ncoefs = getInt (f);
102   coefs = (double *) malloc (ncoefs * sizeof (double));
103
104   for (int i = 0; i < ncoefs; i++)
105     {
106       coefs[i] = getDouble (f);
107     }
108
109   f = stdout;
110   fprintf (f, "processing␣input\n");
111   for (int i = 0; i < npoints; i++)
112     {
113       yf[i] = func (points[i], coefs, ncoefs);
114       fprintf (f, "%lf\n", yf[i]);
115     }
116   fprintf (f, "done\n");
117 }
```

The **testgeneric** program:

1. reads from input the number of points to calculate, line 92.
2. reserves memory for the points and the results, lines 93-94.
3. reads the points from input into memory, lines 96-99.
4. reads the number of coefficients from input, line 101.
5. reserves memory for the coefficients, line 102.
6. reads the coefficients from input, lines 104-107.
7. calculates the result and prints it to output, lines 110-116.

The gfitter program

gfitter has two arguments: the input file name and the calculated data –in this example: *AA* and *AA.data* file–.

gfitter calculates the fit as:

$$\mathbf{fit} = \sqrt{\sum_{i=1}^{n} \left[(\mathbf{ReferenceValues}_i - \mathbf{calculated}_i)^2 * \mathbf{weight}_i \right]}$$

n is the number of individuals (or coefficient sets)

reference is the reference value, File 27.4

calculated is the calculated value from the coefficient set

weight is the weight of each reference value, File 27.4

gfitter also works as a probe, extracting data from the actual calculation. This information is summarized by **grunner** and written to a dynamical report: *report_best.txt*.

The first argument, the input file name, is to know where to account the data.

The second argument is a list of the results of the actual calculation –File 27.9– to compare with the reference values –27.4– line by line.

The report

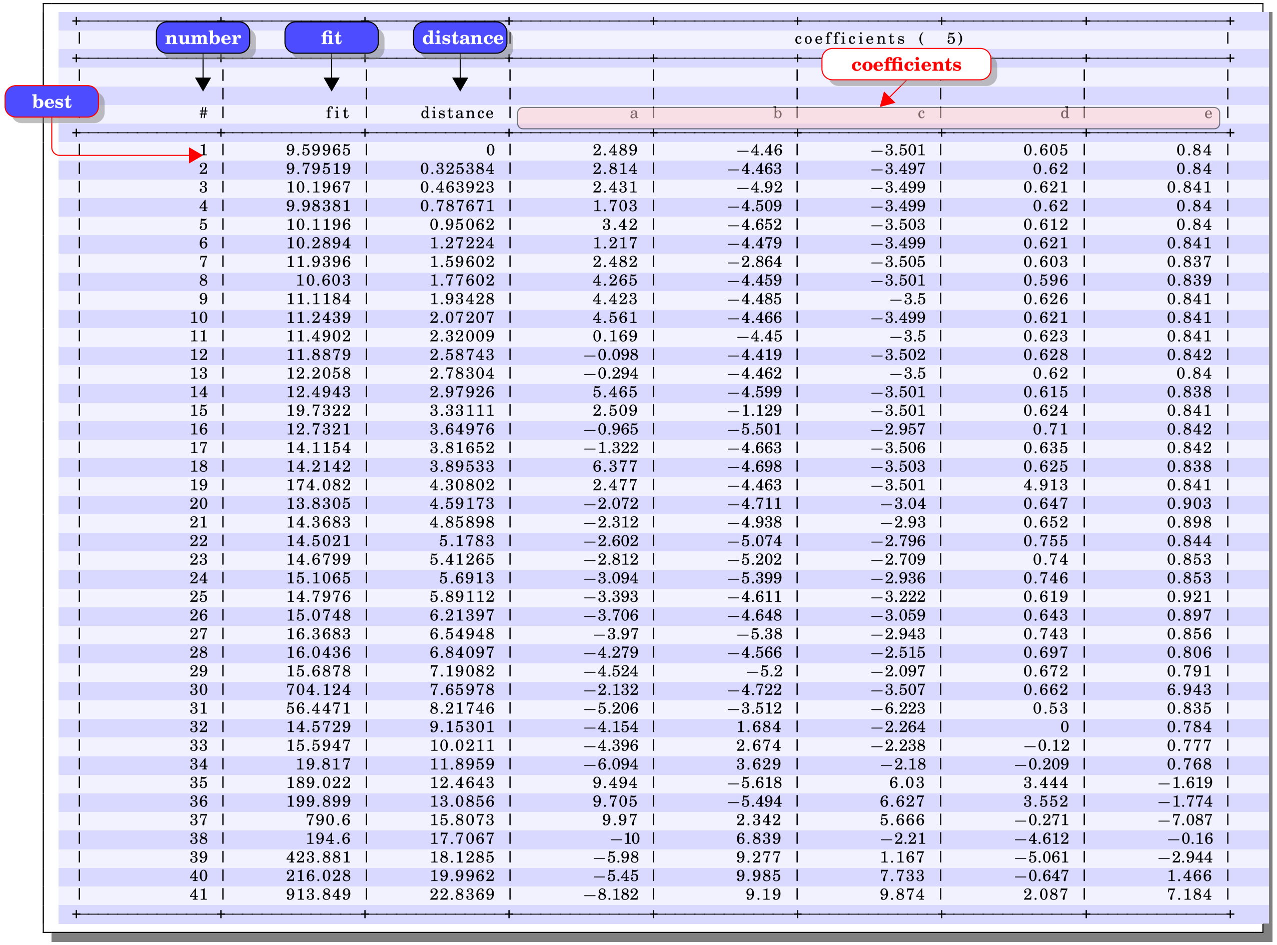
Every generation calculation, **GAFit** updates the file *report_best.txt* with information about the process.

This file contains in the very first line the best result found till now, and compares the rest of generation individuals with it.

The module groups the individuals by slices using the distance from the best, and print the best result found into each slice. Take into account that these slices are dynamically set as just the best found till now is varying as time goes by.

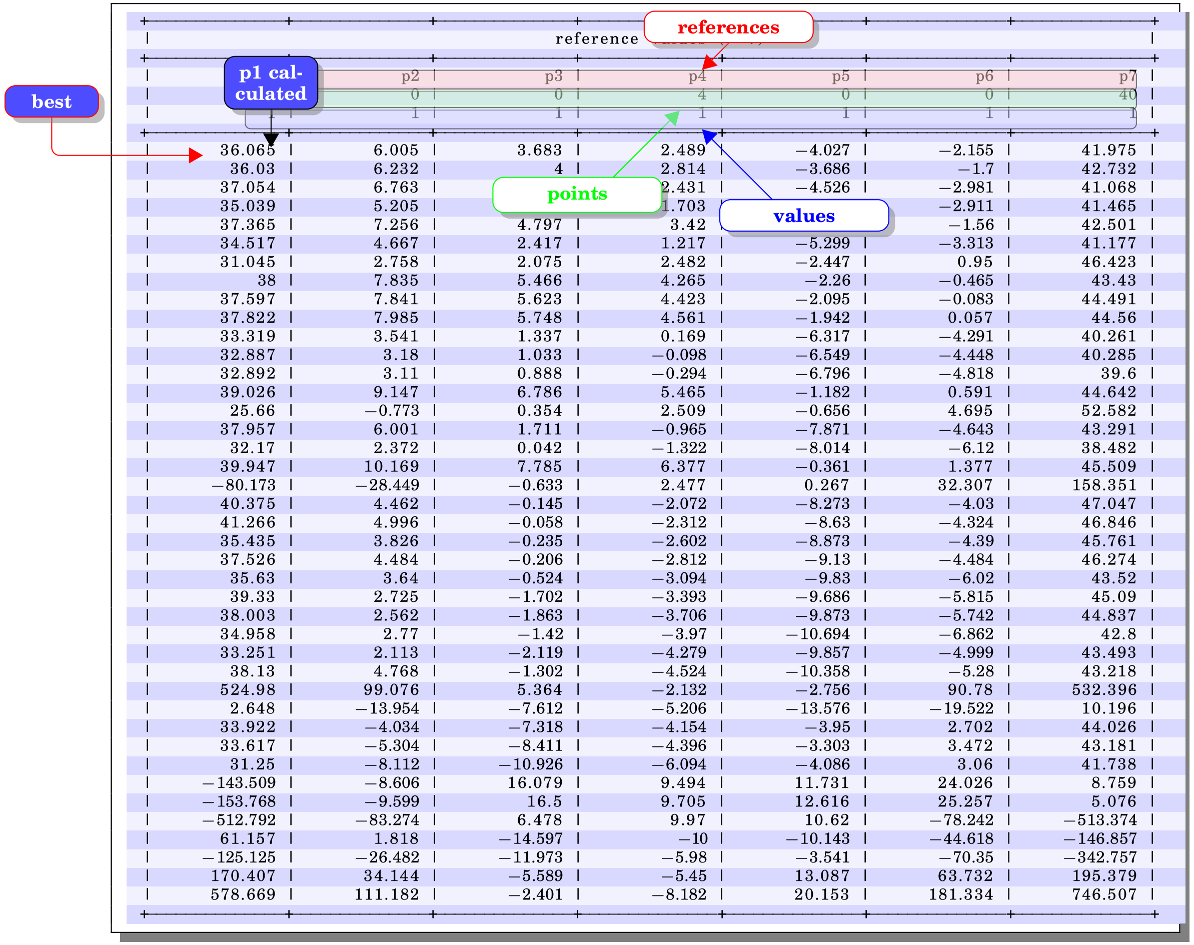
For each individual selected –the best in its slice–, the file includes the distance from best, the coefficients and the calculated reference values.

File 27.13: report best.txt left side


number
fit
distance
coefficients
best

```
+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+
|             |             |             |                        coefficients (  5)                         |
+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+
|             |             |             |             |             |             |             |             |
|           # |         fit |    distance |           a |           b |           c |           d |           e |
+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+
|           1 |     9.59965 |           0 |       2.489 |       −4.46 |      −3.501 |       0.605 |        0.84 |
|           2 |     9.79519 |    0.325384 |       2.814 |      −4.463 |      −3.497 |        0.62 |        0.84 |
|           3 |     10.1967 |    0.463923 |       2.431 |       −4.92 |      −3.499 |       0.621 |       0.841 |
|           4 |     9.98381 |    0.787671 |       1.703 |      −4.509 |      −3.499 |        0.62 |        0.84 |
|           5 |     10.1196 |     0.95062 |        3.42 |      −4.652 |      −3.503 |       0.612 |        0.84 |
|           6 |     10.2894 |     1.27224 |       1.217 |      −4.479 |      −3.499 |       0.621 |       0.841 |
|           7 |     11.9396 |     1.59602 |       2.482 |      −2.864 |      −3.505 |       0.603 |       0.837 |
|           8 |      10.603 |     1.77602 |       4.265 |      −4.459 |      −3.501 |       0.596 |       0.839 |
|           9 |     11.1184 |     1.93428 |       4.423 |      −4.485 |        −3.5 |       0.626 |       0.841 |
|          10 |     11.2439 |     2.07207 |       4.561 |      −4.466 |      −3.499 |       0.621 |       0.841 |
|          11 |     11.4902 |     2.32009 |       0.169 |       −4.45 |        −3.5 |       0.623 |       0.841 |
|          12 |     11.8879 |     2.58743 |      −0.098 |      −4.419 |      −3.502 |       0.628 |       0.842 |
|          13 |     12.2058 |     2.78304 |      −0.294 |      −4.462 |        −3.5 |        0.62 |        0.84 |
|          14 |     12.4943 |     2.97926 |       5.465 |      −4.599 |      −3.501 |       0.615 |       0.838 |
|          15 |     19.7322 |     3.33111 |       2.509 |      −1.129 |      −3.501 |       0.624 |       0.841 |
|          16 |     12.7321 |     3.64976 |      −0.965 |      −5.501 |      −2.957 |        0.71 |       0.842 |
|          17 |     14.1154 |     3.81652 |      −1.322 |      −4.663 |      −3.506 |       0.635 |       0.842 |
|          18 |     14.2142 |     3.89533 |       6.377 |      −4.698 |      −3.503 |       0.625 |       0.838 |
|          19 |     174.082 |     4.30802 |       2.477 |      −4.463 |      −3.501 |       4.913 |       0.841 |
|          20 |     13.8305 |     4.59173 |      −2.072 |      −4.711 |       −3.04 |       0.647 |       0.903 |
|          21 |     14.3683 |     4.85898 |      −2.312 |      −4.938 |       −2.93 |       0.652 |       0.898 |
|          22 |     14.5021 |      5.1783 |      −2.602 |      −5.074 |      −2.796 |       0.755 |       0.844 |
|          23 |     14.6799 |     5.41265 |      −2.812 |      −5.202 |      −2.709 |        0.74 |       0.853 |
|          24 |     15.1065 |      5.6913 |      −3.094 |      −5.399 |      −2.936 |       0.746 |       0.853 |
|          25 |     14.7976 |     5.89112 |      −3.393 |      −4.611 |      −3.222 |       0.619 |       0.921 |
|          26 |     15.0748 |     6.21397 |      −3.706 |      −4.648 |      −3.059 |       0.643 |       0.897 |
|          27 |     16.3683 |     6.54948 |       −3.97 |       −5.38 |      −2.943 |       0.743 |       0.856 |
|          28 |     16.0436 |     6.84097 |      −4.279 |      −4.566 |      −2.515 |       0.697 |       0.806 |
|          29 |     15.6878 |     7.19082 |      −4.524 |        −5.2 |      −2.097 |       0.672 |       0.791 |
|          30 |     704.124 |     7.65978 |      −2.132 |      −4.722 |      −3.507 |       0.662 |       6.943 |
|          31 |     56.4471 |     8.21746 |      −5.206 |      −3.512 |      −6.223 |        0.53 |       0.835 |
|          32 |     14.5729 |     9.15301 |      −4.154 |       1.684 |      −2.264 |           0 |       0.784 |
|          33 |     15.5947 |     10.0211 |      −4.396 |       2.674 |      −2.238 |       −0.12 |       0.777 |
|          34 |      19.817 |     11.8959 |      −6.094 |       3.629 |       −2.18 |      −0.209 |       0.768 |
|          35 |     189.022 |     12.4643 |       9.494 |      −5.618 |        6.03 |       3.444 |      −1.619 |
|          36 |     199.899 |     13.0856 |       9.705 |      −5.494 |       6.627 |       3.552 |      −1.774 |
|          37 |       790.6 |     15.8073 |        9.97 |       2.342 |       5.666 |      −0.271 |      −7.087 |
|          38 |       194.6 |     17.7067 |         −10 |       6.839 |       −2.21 |      −4.612 |       −0.16 |
|          39 |     423.881 |     18.1285 |       −5.98 |       9.277 |       1.167 |      −5.061 |      −2.944 |
|          40 |     216.028 |     19.9962 |       −5.45 |       9.985 |       7.733 |      −0.647 |       1.466 |
|          41 |     913.849 |     22.8369 |      −8.182 |        9.19 |       9.874 |       2.087 |       7.184 |
+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+
```

File 27.14: report best.txt right side


references
best
p1 cal-
culated
points
values

```
+---------+---------+---------+---------+---------+---------+---------+
|                         reference                                   |
+---------+---------+---------+---------+---------+---------+---------+
|         |      p2 |      p3 |      p4 |      p5 |      p6 |      p7 |
|         |       0 |       0 |       4 |       0 |       0 |      40 |
|       1 |       1 |       1 |       1 |       1 |       1 |       1 |
+---------+---------+---------+---------+---------+---------+---------+
|   36.065 |   6.005 |   3.683 |   2.489 |  −4.027 |  −2.155 |  41.975 |
|    36.03 |   6.232 |       4 |   2.814 |  −3.686 |    −1.7 |  42.732 |
|   37.054 |   6.763 |         |   2.431 |  −4.526 |  −2.981 |  41.068 |
|   35.039 |   5.205 |         |   1.703 |         |  −2.911 |  41.465 |
|   37.365 |   7.256 |   4.797 |    3.42 |         |   −1.56 |  42.501 |
|   34.517 |   4.667 |   2.417 |   1.217 |  −5.299 |  −3.313 |  41.177 |
|   31.045 |   2.758 |   2.075 |   2.482 |  −2.447 |    0.95 |  46.423 |
|       38 |   7.835 |   5.466 |   4.265 |   −2.26 |  −0.465 |   43.43 |
|   37.597 |   7.841 |   5.623 |   4.423 |  −2.095 |  −0.083 |  44.491 |
|   37.822 |   7.985 |   5.748 |   4.561 |  −1.942 |   0.057 |   44.56 |
|   33.319 |   3.541 |   1.337 |   0.169 |  −6.317 |  −4.291 |  40.261 |
|   32.887 |    3.18 |   1.033 |  −0.098 |  −6.549 |  −4.448 |  40.285 |
|   32.892 |    3.11 |   0.888 |  −0.294 |  −6.796 |  −4.818 |    39.6 |
|   39.026 |   9.147 |   6.786 |   5.465 |  −1.182 |   0.591 |  44.642 |
|    25.66 |  −0.773 |   0.354 |   2.509 |  −0.656 |   4.695 |  52.582 |
|   37.957 |   6.001 |   1.711 |  −0.965 |  −7.871 |  −4.643 |  43.291 |
|    32.17 |   2.372 |   0.042 |  −1.322 |  −8.014 |   −6.12 |  38.482 |
|   39.947 |  10.169 |   7.785 |   6.377 |  −0.361 |   1.377 |  45.509 |
|  −80.173 | −28.449 |  −0.633 |   2.477 |   0.267 |  32.307 | 158.351 |
|   40.375 |   4.462 |  −0.145 |  −2.072 |  −8.273 |   −4.03 |  47.047 |
|   41.266 |   4.996 |  −0.058 |  −2.312 |   −8.63 |  −4.324 |  46.846 |
|   35.435 |   3.826 |  −0.235 |  −2.602 |  −8.873 |   −4.39 |  45.761 |
|   37.526 |   4.484 |  −0.206 |  −2.812 |   −9.13 |  −4.484 |  46.274 |
|    35.63 |    3.64 |  −0.524 |  −3.094 |   −9.83 |   −6.02 |   43.52 |
|    39.33 |   2.725 |  −1.702 |  −3.393 |  −9.686 |  −5.815 |   45.09 |
|   38.003 |   2.562 |  −1.863 |  −3.706 |  −9.873 |  −5.742 |  44.837 |
|   34.958 |    2.77 |   −1.42 |   −3.97 | −10.694 |  −6.862 |    42.8 |
|   33.251 |   2.113 |  −2.119 |  −4.279 |  −9.857 |  −4.999 |  43.493 |
|    38.13 |   4.768 |  −1.302 |  −4.524 | −10.358 |   −5.28 |  43.218 |
|   524.98 |  99.076 |   5.364 |  −2.132 |  −2.756 |   90.78 | 532.396 |
|    2.648 | −13.954 |  −7.612 |  −5.206 | −13.576 | −19.522 |  10.196 |
|   33.922 |  −4.034 |  −7.318 |  −4.154 |   −3.95 |   2.702 |  44.026 |
|   33.617 |  −5.304 |  −8.411 |  −4.396 |  −3.303 |   3.472 |  43.181 |
|    31.25 |  −8.112 | −10.926 |  −6.094 |  −4.086 |    3.06 |  41.738 |
| −143.509 |  −8.606 |  16.079 |   9.494 |  11.731 |  24.026 |   8.759 |
| −153.768 |  −9.599 |    16.5 |   9.705 |  12.616 |  25.257 |   5.076 |
| −512.792 | −83.274 |   6.478 |    9.97 |   10.62 | −78.242 | −513.374 |
|   61.157 |   1.818 | −14.597 |     −10 | −10.143 | −44.618 | −146.857 |
| −125.125 | −26.482 | −11.973 |   −5.98 |  −3.541 |  −70.35 | −342.757 |
|  170.407 |  34.144 |  −5.589 |   −5.45 |  13.087 |  63.732 | 195.379 |
|  578.669 | 111.182 |  −2.401 |  −8.182 |  20.153 | 181.334 | 746.507 |
+---------+---------+---------+---------+---------+---------+---------+
```

Appendices

A

Source code

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*
aplication.c	simple configuration	
aplication.h	simple configuration	
autoweights.c	stuff to use automatic weights	
autoweights.h	autoweights header	
bounds.c	stuff to read bounds	
bounds.h	bounds header	
bytecodes	defines bytecodes for fpu	
charmmm	interface with charmm	
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	
finput.c	read variables and setup system	
fitview.c	plots data	
flyctl	rutines to stop running jobs	
fpu	virtual FPU	
ga.c	main program	

File/Directory	Description	Comments
ga.h	ga header	
generic	generic module stuff	
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	
inter	inter module stuff	
interface.f	glue to link all together	
interface.h	interface header	
job.txt	job configuration	modify as per job basis
literals	subroutines to support automatic coefficient names	
mvariable	external interface to deal with multivariate calculus	
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 Application

Simple configuration shortcuts. C language.

- application.h
- application.c

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.7, A.4, A.7, A.7, A.6 and A.8. See section 14.2 and Figure 1.1.

- ga.h
- ga.c

A.6 Genetic Algorithm Core

GA routines. C language.

Crossover

- crossover.h
- crossover.c

Mutation

- mutation.h
- mutation.c

Selection

- selection.h
- selection.c

Stats

- stats.h
- stats.c

Utils

- utils.h
- utils.c

A.7 MODULES

Here is implemented the interface with external programs. C, Fortran and Perl languages.

Module inter

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

Flyctl

This code addresses the external job stopping problem.

- flyctl.h
- flyctl.c

MOPAC module

Interface with MOPAC.

- extractor
- fitter.f
- injector.c
- mopac.h
- mopac.c
- shepherd.c
- lstimes.c
- lsexdata.f

CHARMM module

Interface with CHARMM.

- chmconfigurator.c
- chmreference.c
- chmrunner.c
- charmm.h
- charmm.c
- chmbest.c

Multivariate module

The fpu routines are used to deal with multivariate calculus.

- mvariable.h
- mvariable.c

Generic module

See Section 27.

- common.h
- common.c
- grunner.c
- gfitter.c
- templanalyzer.c

A.8 Miscellaneous

Arguments

Program arguments stuff. C language.

- arguments.h
- arguments.c

Bounds

Custom routines to read bounds files. C language.

- bounds.h
- bounds.c

Cnames

Coefficient names stuff. C language.

- cnames.h
- cnames.c

Inputline

Custom routines to read lines and text from configuration and data files. C language.

- line.h
- line.c

Integer

Code to support integer coefficients. C language.

- integer.h
- integer.c

Literals

Routines to support automatic coefficient names. C language.

- literals.h
- literals.c

Parameters

Code to deal with program parameters. C language.

- parameters.h
- paramenters.c

Rand

Random stuff. C language.

- rand.h
- rand.c

Rangecf

Coefficients with range. C language.

- rangecf.h
- rangecf.c

Rstrings

C strings custom routines. C language.

- rstrings.h
- rstrings.c

A.9 Tools

C, Java and Perl languages.

Fitview

Tool to create some **gnuplot** plots.

- fitview.c

Needle

Perl tool to create the *atom2type* file from a geometry file.

- needle

Ufpu

Utility to test analytical expressions as potentials.

- ufpu.c

B

License

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. `http://fsf.org/`

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

   A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

   The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

   The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

   The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

   "Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

   When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

   Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

   a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

   b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

    A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

    The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

    If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

    Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS

AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

References

[1] Roberto Rodríguez-Fernández, Francisco B. Pereira, Jorge M.C. Marques, Emilio Martínez-Núñez, and Saulo A. Vázquez. “GAFit: A general-purpose, user-friendly program for fitting potential energy surfaces”. In: *Computer Physics Communications* 217 (2017), pp. 89–98. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2017.02.008. URL: http://www.sciencedirect.com/science/article/pii/S0010465517300607.

[2] J. M. C. Marques, F. V. Prudente, F. B. Pereira, M. M. Almeida, A. M. Maniero, and C. E. Fellows. “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] Ira N. Levine. *Quantum Chemistry*. 7th ed. Pearson Education Inc, Boston, Feb. 2014, p. 720.

[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] Zahra Homayoon, Saulo A. Vázquez, Roberto Rodríguez-Fernández, and Emilio Martínez-Núñez. “Ab Initio and RRKM Study of the HCN/HNC Elimination Channels from Vinyl Cyanide”. In: *The Journal of Physical Chemistry A* 115.6 (2011). PMID: 21261315, pp. 979–985. DOI: 10.1021/jp109843a.

[6] Roberto Rodriguez-Fernandez, Saulo A. Vazquez, and Emilio Martinez-Nunez. “Collision-induced dissociation mechanisms of [Li(uracil)]+”. In: *Phys. Chem. Chem. Phys.* 15 (20 2013), pp. 7628–7637. DOI: 10.1039/C3CP50564B.

[7] 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/.

[8] Kalyanmoy Deb and Ram Bhushan Agrawal. "Simulated binary crossover for continuous search space". In: *Complex Systems* 9 (1994), pp. 1–34.

[9] 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.

[10] Larry J. Eshelman and J. David Schaffer. "Real-Coded Genetic Algorithms and Interval-Schemata". In: *FOGA*. 1992, pp. 187–202.

[11] Charles FF Karney. "Quaternions in molecular modeling". In: *Journal of Molecular Graphics and Modelling* 25.5 (2007), pp. 595–604.

[12] Virginia Tech's SABLE. *Statistics Activity-Based Learning Environment (SABLE)*. URL: http://simon.cs.vt.edu/SoSci/converted/.

Other interesting references to the reader

Marcos M Almeida, Frederico V Prudente, Carlos E Fellows, Jorge MC Marques, and Francisco B Pereira. “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.

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.

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.

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.

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.

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.

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.

James Newkirk, Jesús García Molina, Robert C. Martin, and Martin Fowler. *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, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *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. *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.

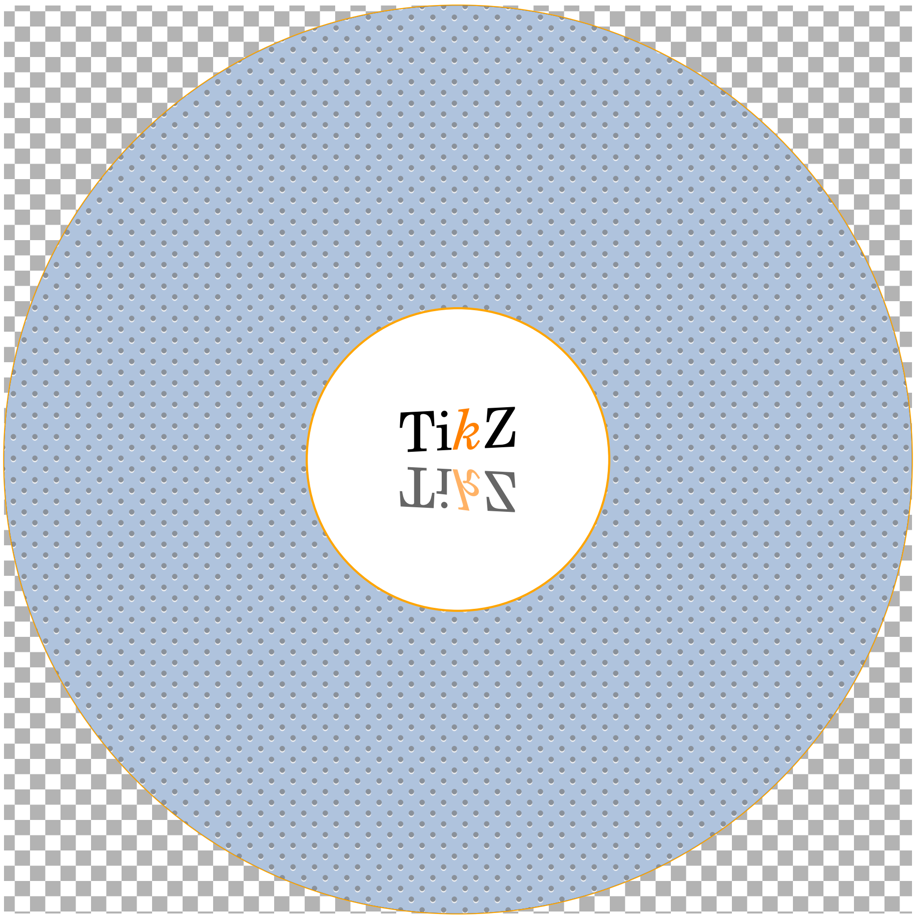
Stephen Wright and Jorge Nocedal. *Numerical Optimization*. Springer Verlag, 2006, 664 pages. ISBN: 9780387303031.

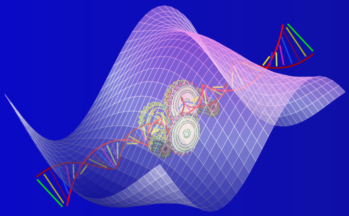
List of tables

List of figures

List of files

This manual was typeset using the
LaTeX typesetting system.


TikZ



GAFit

User Manual

✦