Reproduction / Analytical Chemistry

[Rp] Gifa V.4: A complete package for NMR data set processing

Marc-André Delsuc[1,2, ID]

[1]Institut de Génétique Biologie Moléculaire et Cellulaire - UMR 7104 - Inserm U 1258, Illkirch, France – [2]CASC4DE, SAS, Illkirch, France

1 Introduction

I present here the reproduction of the 1996 article entitled "Gifa V.4: A complete package for NMR data set processing" [1].

The *Gifa* project is a career-long effort. It was started in 1987 as a way to gather various codes and ideas I had developed to handle and process NMR datasets in the early days of 2D and 3D NMR[1]. Since the beginning, *Gifa* has been built over a careful memory management, a tuned data organisation, and a large set of elementary functions meant to be pipelined. The version 4 published in 1996 was a major improvement over the previously published version (1988) as it brought a complete scripting language, a complete graphic user interface based on the Motif library, and a database management system. This program was not open-source at that time, but nevertheless profited from a large number of users, and some are still using it nowadays, despite the difficulty to maintain it. This was an additional motivation to try to renew this program and to make it freely available on a code forge.

2 Historical context

When I entered the *ten years challenge*, I first thought of reproducing a later avatar of the same project, a 2007 program implementation called at that time NPK [2]. The NPK version was an effort to move away from the monolithic environment of *Gifa* which was nearly impossible to port to MacOs or Windows. This had lead us to move the graphic from X/Motif to Java/Swing and the scripting language to python 2.1 (thanks to the excellent `jython` package recently released then). However the mathematical core was unchanged and still written in FORTRAN77 with strong 32 bit limitations. Because of these limitations, NPK was abandoned, and a much extended program has been developed in python since then [3].

After more consideration, I realized that reproducing a program written as a mix of FORTRAN77 , Java, and Python 2.1 was not worth the effort, in particular considering

---

[1]I started doing NMR as early as 1979, for my French DEA project. 2D NMR was in its infancy (the initial paper from R.Ernst group is dated 1976). NMR spectrometers, computers, and programs were badly prepared for this new approach, where complex pulse sequences had to be prepared and launched in automation, generated large datasets, which had to be transformed, visualized and quantified while not fitting in the 64kB central memory, and even barely on the removable 14 inch disk pack that stored 2.5MB of data !

my poor knowledge of Java and the fact that nobody was using this program any more. This is why I decided to fall back on the 1996 version of the same project.

Gifa was started in the eighties, when the first 32 bit architectures (Vax/VMS) were co-existing with older 16 bit ones (PDP-11). It was also a time were 64MB of memory was plenty, and the challenge was to fit the 2D NMR datasets in memory, in contrast with other programs at that time which were doing lengthy processes by swapping the data on disk. *Gifa* is thus based on a carefully carved memory organisation, built on many FORTRAN77 COMMON and EQUIVALENCE structures, which allowed to use part of the memory for certain actions, but also to use the whole memory when needed. Due to FORTRAN77 limitations, this allocation is purely static, and set at compile time - for this reason, several version of *Gifa* were distributed, ranging from a very small buffer (total 1MB), to the largest one (256MB).

Gifa is a data analysis program meant for NMR. In its 1996 avatar, it presents a complete set of fundamental mathematical operations (Fourier transforms, data-set arithmetic, model fitting, ...) and some advanced tools (Linear Prediction tools, Maximum Entropy optimisation, DOSY processing by Inverse Laplace transform...). It is able to process 1D, 2D and in a less optimized manner 3D data-sets. It comes with a complete scripting language with variable scoping, (limited) recursion, arrays, dictionaries, file I/O, and some database capabilities. This macro language, complemented with a set of graphic primitives is used to build a complete dynamic GUI, which covers basic and advanced operations, an NMR assignment tool, and an NMR relaxation module. As mentioned above, the memory model is static and monolithic in order to allow an optimal use of the resources. This organisation insures little dependencies with respect to external tools, and a good control of the development.

3 Retrieval of the software

The source files of the historical *Gifa* was still stored somehow on my current hard drive and has been through two laboratory relocations (from Montpellier to Lyon and then to Strasbourg), many computer upgrades (with migration of the accounts), and one OS change (from MacOsX to Linux Ubuntu).

No version control system was used at that time, however I could find several versions of the source in tar archives from version 4.31 dated Aug 2000, to version 4.5 dated Mars 2004. In the absence of the article version dated 1996, I decided to use these versions, and moved them to github.com as successive commits. The code is about 54 000 lines of FORTRAN77, 18 000 lines of C and 27 000 lines of the macro language.

The second step is to remember the file organisation and how to build the program. As the program was not open-source, this was not described in the official documentation. A central make program controls the building process - the Makefile contains some documentation, and guided me in the process.

3.1 Steps by steps

All the development work was performed in a virtual machine running Ubuntu 16.04 32 bit with 4GB of memory hosted by a MacBookPro with a Intel Core i5-7360U processor at 2.30GHz, running Ubuntu 19.10.

At the beginning of this project, I had three main concerns: i) the 32 bit organisation of the program; ii) the maintainability of the production pipeline; and iii) external libraries, in particular the X/Motif library used for display.

A 32 bit program — As a consequence of the memory management, the 32 bit organisation is intrinsic to the program. Not sure on how to handle this, I decided to work in a Virtual Machine (VM) based on the last Ubuntu version which proposes a 32 bit version: 16.04

LTS. Despite this environment being more than 20 years posterior to the one in which *Gifa* was developed, everything was built without any particular problems.

The production pipeline – Thanks to the already long development history, the program had been ported to VMS and many Unix flavours (Linux, SunOs, HP-UX, IRIX, IRIX64, AIX). This was possible through the use of a home-built production pipeline able to adapt the variations of each architecture, using a mixture of `make` and `perl` scripts and the `f2c` FORTRAN77 to C translator [4]. Fortran source code first goes through a home-made Fortran code preprocessor that permits to include files, conditional code, and allows to tune and adapt the produced code. Fortran code is then translated to C by `f2c` and finally compiled with `gcc`. This scheme simplifies the choice of the FORTRAN77 compiler and allows additional features, such as recursivity (not allowed in FORTRAN77) and simple link with C code.

This set-up, which was optimized to ease the porting to a new architecture, has helped a lot in the reproduction work. Only a few small modifications was needed to adapt the code to a recent version of Linux, running in a Virtual environment, as the basic languages (`make`, `perl`, and C) on which everything is based have not significantly changed.

External libraries – The program uses very few external libraries apart for the X/Motif graphic user interface, and the `ndbm` database engine. Most mathematical algorithms (Fourier transform, Brent or Levenberg-Marquardt optimisers, ...) are natively coded, inspired from the excellent *Numerical Recipes* book [5]. Linear algebra operations, use the LINPACK [6] routines, which sources were integrated in the code itself. Basic vector operations are constructed from a home-made library (`vect.for`) implementing 60+ basic operations, in an atomic manner, easy to optimize for the compiler.

The X/Motif library had been a major hurdle during the *Gifa* early development in particular while porting to the various available versions, and I was fearing it would still be a problem. It turned out to be extremely simple, just an installation from the repository.

Actual changes –

libraries: I had to install X/Modif from the repository (`libmotif-dev`, `libxext-dev`) and readline from `libreadline6-dev`. The `ndbm` database engine is not available for the distribution 16.04, but the GNU Project proposes gdbm which has a `ndbm` compatibility mode, activated with a flag at link time: `-lgdbm_compat`.

Linux and environment: A few constructs were obsolete or not adapted to a modern system and had to be adapted.

- At that time, `perl` was in version 4 (`perl` 5 was released in 1994 and not widely adopted then). Ubuntu 16.04 uses `perl` 5.22 and a few simple syntactic changes were needed to C have a working pipeline.

- the timing code was using the `CLK_TCK` macro which seems to be obsolete, this was modified to use `sysconf (_SC_CLK_TCK)`;

- temporary files were created on `/usr/tmp`. I had to make a link to `/tmp` to solve this problem with no modification in the code;

- I struggled a bit with the linker step - the simpler appeared to be the better !

- The HTML documentation is created with a `perl` script, HTML comments were using the syntax "`<!-- comments >`" which is not understood any-more and had to be changed to "`<!-- comments -->`"

- HTML display was using Netscape™ or Mozilla™, this was changed to Firefox™.

licensing: At the time of the publication, the program was not open-source, and had to be activated with a licence key distributed upon request. A C code was checking the validity of the key at launch time, this code has been deactivated, and a CeCILL 2.1 license [7] was associated to the code.

All the modifications are reflected as commits in the repository.

4 Reproduction

After performing the modifications above, the program runs directly with no particular problem, and the internal test suit is validated with no error. The display system is up and the graphic User Interface is correctly functioning. Figures 1 and 2a in the original study, which present the macro language are not reproduced here. Listing 1 reproduces Figure 2b, and Figures 1 and 2 reproduce Figures 2c and 3 from the original study. The data-set which is shown in the original work could not be recovered. In consequence, I chose to present a TOCSY spectrum of Artemisine, obtained at 700MHz in a recent study [8]

```
; this text reproduces Figure 2a from the original paper

; This macro creates a form box which permits to modify the
; parameters of the contour-mode display window.
; The user can - open / close the contour-mode display window
;               - choose to display positive / negative / both sides
;               - change the number and spacing of contour levels.
;Preset parameters:
if ($CDISP2D==1) then     ;$cdisp2d describes graphic states.
    set cdef = on
else
    set cdef = off
endif
if ($SIGN==1) then        ;Describes which levels are displayed
    set pndef = positive  ; (positive, negative, both)
elsif ($SIGN == 0) then
    set pndef = both
else
    set pndef = negative
endif
;Build the form
formbox "Display control" \
    "dispcont_doit $cont $posneg $lev $log" \    ;Callback
    "Contour Display (cdisp2d)" \
        enum "on,off" cont $cdef \                ;Define fields
    "Display mode (sign)" \
        enum "positive,negative,both" posneg $pndef \
    "Number of levels (level)" int lev $LEVEL \
    "Level Algo : (loga)" real log $LOGA \
    "(x=1 : equidistant; x>1 : by power of x)" message \
    *               ; * is end of list
exit
```

Listing 1. Text of the macro `dispcont`, which builds the display control form box.

Slight differences can be observed between the original figures and the current ones. Most of the differences come to simple improvements in the program after the initial release, for instance the button 'Cancel' changed to 'Close', or the reorganisation, and extension of the user menus.

The benchmark programs proposed in the original publication were run, the results are shown in Table 1.
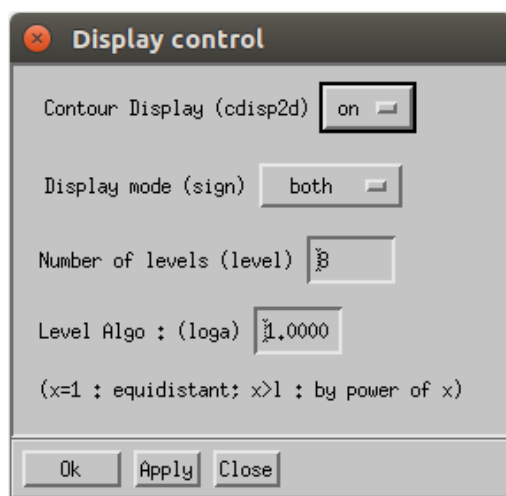
Figure 1. The form resulting from the execution of the macro in Listing 1 is a static graphic box with an action button 'Apply', with which the macro command dispcont_doit (*not show*) is associated.
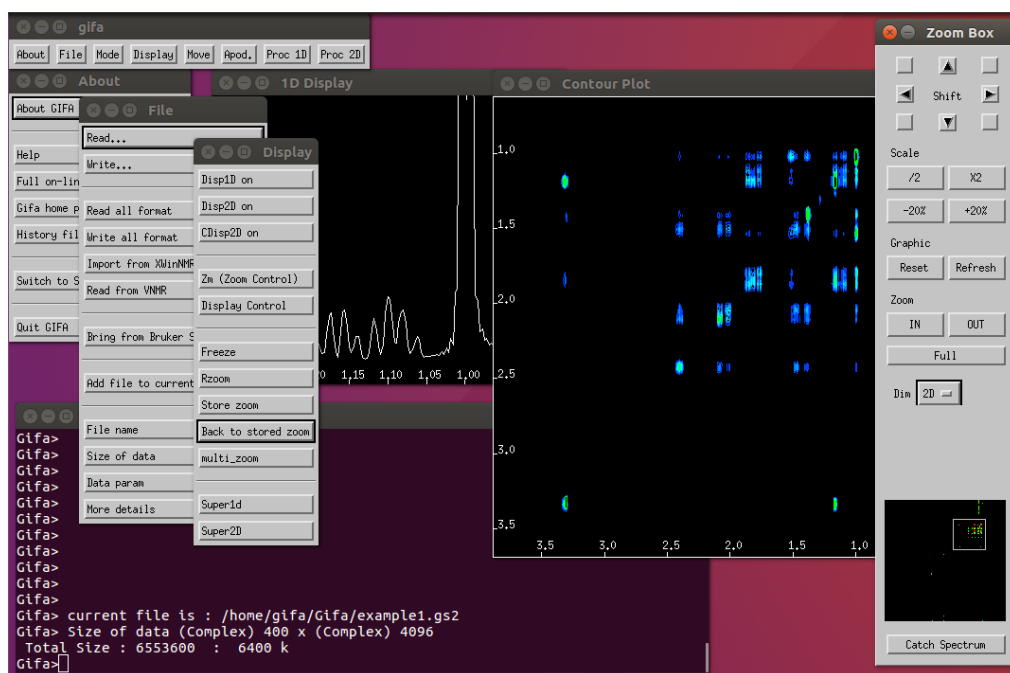
Figure 2. The default graphic environment as invoked during start-up of Gifa. All the basic commands can be run from this graphic environment. The control box called Zoom Box permits interaction with the graphic window. The data are currently displayed in a contour plot 2D window and in a 1D window. The small vignette seen in the Zoom Box is a view of the complete data set held into the 2D buffer, the square in it indicates the current zoom window.

Table 1. Different processing times observed using the *Gifa* program *(see reference [1] for details)*

Processing Time (s)	SUN	IBM	SGI	HP	This study
2D	17.5	12.6	10.5	7.5	0.05
3D	1064	329	193	172	1.10

At this stage, the reproduction is considered as successful, even though a complete check of all functionalities has not been fully performed.

5 Conclusion

The *Gifa* program was successfully reproduced, in a relatively easy manner for a software of this size, developed about 25 years ago. The version of the code originally published was lost, but a later version of the program was used and fully reproduced on a recent computer. The reproduction was performed in a simple Virtual Machine, running the 32 bit version of Ubuntu 16.04, which should be supported until April 2024.

The ease of porting is mostly coming from the use of standard tools and programming languages, still well supported today (`f2c`, `perl`, `make`, `gzip`, `tar`, ...). The program is one large code with very little external dependencies, as it internally contains all mathematical routines, an internal macro language, and its own GUI system. The only external libraries which had to be installed are the X/Motif graphic and `ndbm`, a flat database engine. While the X/Motif library could be installed and linked with no modification, `ndbm` was not available. By chance `gdbm`, the GNU simple database engine, has a compatibility mode for `ndbm` which could be readily used. Also, the diversity of the Unix environments in the nineties, imposed the development of a robust production pipeline easy to modify and to tune, which has been a valuable tool in this reproduction. The needed corrections have mostly consisted in finding and installing the correct software packets or libraries. Some tools and some structures had to be modified because of obsolete constructs (Netscape™ or HTML comments for instance!).

Reviving the *Gifa* program was an interesting experience, it reminded me the qualities - and flaws - of this program, and let me appreciate the efficiency of a monolithic program, with a tight memory management and an algorithmics optimized to the limited resources available 20 years ago. Despite its limitations, such a program can still have many applications today, for instance embedded in a larger environment.

This work also showed that a monolithic program, written with simple standard tools, with little dependencies on the environment, is quite easy to maintained and can see its life extended beyond 20 years.

References

1. J.-L. Pons, T. E. Malliavin, and M.-A. Delsuc. "Gifa V. 4: A complete package for NMR data set processing." In: **Journal of Biomolecular NMR** 8.4 (1996).
2. D. Tramesel, V. Catherinot, and M.-A. Delsuc. "Modeling of NMR processing, toward efficient unattended processing of NMR experiments." In: **Journal of Magnetic Resonance** 188.1 (2007), pp. 56–67.
3. L. Chiron, M.-A. Coutouly, J.-P. Starck, C. Rolando, and M.-A. Delsuc. "SPIKE a Processing Software dedicated to Fourier Spectroscopies." In: **arXiv** (2016), p. 1608.06777. arXiv: `1608.06777 [physics.chem-ph]`.
4. **f2c - FORTRAN to C convertor**. https://www.netlib.org/f2c/.
5. W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. **Numerical recipes in Fortran 77**. Cambridge University Press, 1996.
6. **LINPACK - LINear algebra package**. https://www.netlib.org/linpack/.
7. **CeCILL 2.1 license**. https://cecill.info/index.en.html. 2013.

8. L. Margueritte, P. Markov, L. Chiron, J.-P. Starck, C. Vonthron-Sénécheau, M. Bourjot, and M.-A. Delsuc. "Automatic differential analysis of NMR experiments in complex samples." In: **Magnetic Resonance in Chemistry** 56.6 (2018), pp. 469–479.