ONCE WERE WARRIORS: THE NEED OF RE-EDUCATION IN MATHEMATICS AND COMPUTING FOR LIFE "SCIENTISTICIANS"

Jorge A. Navarro Alberto
Departamento de Ecología
Universidad Autónoma de Yucatán, Yucatán, México
jorge.navarro@uady.mx

This paper discusses the pros and cons of the new computing era and the status of mathematics in Life Sciences in relation to the training in statistics a life scientist is expected to have. Reflections are made about how preferences in the use of software and the topics taught nowadays in mathematics at University level have mined the creative abilities of the life scientists whenever data analysis is an on due task. Contrasts of the Mathematics and Computing taught in the pre-GUI era and the paradoxically diminishing influence of these scientific subjects at present times are given. Examples of the powerful computing concepts and the ubiquitous computing resources useful for the data analyst and avenues of opportunity of applications of mathematical ideas are also presented, in order to make suggestions in the curricula that respond to the changing demands of what life scientists are expected to do.

INTRODUCTION

In the 60's of the XXth century, Arthur L. Samuel, at Watson Research, IBM, was the first to propose the word *personal computer*, that is to say, designed to be used by a person simultaneously. Since the mid 60's a movement supporting the computerization at all the levels was put in action, deriving in the first social networks: computer users began to get connected with colleagues, managers, designers, etc., people who accelerated the processes of complex social alliances and construction of spaces to the not-literate-in-computers. All these people organized the social movements of using computers at a large scale. Coalitions were formed to put computers to the scope to everybody, and originated one of the most powerful business division worldwide nowadays: the development of hardware and software.

The bet of popularizing computers in everyday life detonated the development of computer systems that, between other tasks, allowed the development of *programmed gadgets*, in such a way that the operator be able to *look and feel* the environment by means of symbols, dialogue boxes and icons to gain access to files, menus, procedures, etc. These ways of interacting were named *Graphical User Interface* (*GUIs*). In the beginning of the computer era, operating systems and common programming environments *were not visual*, but *text-type*, so the prevailing computational culture was similar for users and programmers. Gradually this type of interaction human-computer changed, and it began the era of *WYSIWYG* (*What you see is what you get*) in *application* programs (this latter term coined to distinguish them from programming languages). The WYSIWYG concept was then transformed into GUI environments. The GUIs allowed one to relieve the problem that interfaces of the first computers had to rely on: they only interacted through a prompt (the command line indicated by a cursor to locate where to write instructions). Gradually the ambient became *visual* and *friendly* to the user: it was not necessary to be a programmer to interact with the machine. In fact, programmers were employed for interface development, the so-called *Application Program Interfaces* (*APIs*), which propelled the creation of systems with GUIs for problem solving of any nature. In the scientific sphere these visual environments allowed to execute routine tasks like document edition and spreadsheet calculation, making them "*a better experience*". Specialists formed guilds of users on computing which ensured access to computer programs to the non-specialist. This way, for example, scientists abandoned the activity to create their own programs, and they were grateful how easily they could now perform any task in the computer. At present the applications programs (based on GUIs) answer practically to any demand of the scientist. The words of Arthur L. Samuel have been resounding since he published a series of articles in the magazine New Scientist in 1964, named "*The Banishment of Paper Work*": "*While it will be entirely feasible to obtain an education at home, via one's own personal computer, human nature will not have changed, and there will still be a need for schools with laboratories, classrooms, and individual teachers to motivate the students ('The world in 1984')*". There are still

In K. Makar, B. de Sousa, & R. Gould (Eds.), *Sustainability in statistics education. Proceedings of the Ninth International Conference on Teaching Statistics (ICOTS9, July, 2014), Flagstaff, Arizona, USA.* Voorburg, The Netherlands: International Statistical Institute. iase-web.org [© 2014 ISI/IASE]

many reasons why, humans, still need to gain computer-literacy at our universities but there are forces pulling towards the complete independence in the use of computers for everything!

Through my work teaching statistical methods in the biological sciences I have witnessed about how important the programming languages have been to scientists, and later, how the computer applications with GUIs have dominated the scene in the learning process of mathematical and statistical methods. The advantages of the GUIs in the sciences are enormous, but the automation of statistical and mathematical analyses in biological sciences has suffered adverse effects whenever calculations or simulations are performed. It is not uncommon to encounter biologists with a hesitant mood, feeling unhappy with the solution of a problem, because it was achieved with the help of an *intelligent* GUI, while the actual situation is that it was the result of an automated or packed *black box*. This is the price paid of being only a *simple* user. The risks of generating results through *black boxes* can be minimized if the user is familiar with conceptual and procedural principles of programming and computing, the computing of the twenty-first century. All these principles, in addition to a good disciplinary knowledge will confer better abilities for data analysis and modeling. This was the philosophy of computer programming learnt by scientists of the twentieth century, an attitude that has been going down. It is necessary to promote the return of the *computational* culture in such a way that a scientist be more proficient in generating results from computer programs. This paper has the objective to analyze the price that has been paid by the presence of GUIs in the computers and software applications for scientific work. The reflection is illustrated with my personal experience teaching Mathematics and Computing during the pre-GUI era, subjects that, paradoxically, have nowadays a diminishing influence at present times. Examples of the powerful computing concepts and the ubiquitous computing resources useful for the data analyst and avenues of opportunity of applications of mathematical ideas are also presented, in order to make suggestions in the curricula that respond to the changing demands of what life scientists are expected to do.

LET US TALK ABOUT HISTORY...

The hardware of everyday use for the scientist would not be available without the vertiginous sophistication and miniaturization of computerized technology. From vacuum tube computers in the first generation (1951-1957) up to the fifth one (the current generation) based on artificial intelligence, nanotechnologies, superconductors, quantum and molecular calculation, the scientists have regulated and updated their knowledge and training in reply to the implied technological changes. Nowadays, computers and integrated information systems are synonymous of handling large quantities of information and intensive calculations at an unthinkable extent. In the “beginning”, between the first and the third generations, specialists were the leaders of the history, configuring both hardware and software to solve specific tasks: the main role in computing matters was restricted to the experts. In case of the software, the programmers were playing a dual role: programmer and user. In the course of time, and in a similar way to the process of integration of human groups joined by the same language, *technosocial* groups were created in order to share something about computers: a language, an improved integrated circuit, a new algorithm... Remember the scientific and academic sectors between the 50’s and the 80’s of the 20th century characterized by their strong use of computer languages, like FORTRAN, Pascal and, the most popular of all, BASIC. This part of the history of integration of technology and science was distinguished by the absence of sophistication of the interface human-computer. It was enough to have a monitor where to visualize commands and a keyboard in which those commands were transcribed. In this environment, the users somewhat had the control of the operating and file systems. For example, users could copy a specific file from one place to another with a command line. The systems worked with only 32K or 64K of memory RAM! And for the habitual programming tasks it was needed to know the syntax of the language and to write “only” commands lines using a keyboard.

THE *GUI*S DEMYSTIFIED

In my career as a programmer, a teacher at high school and college levels, and mathematician, I have witnessed the disagreements between the *artisan mentality* of whom interacts with the computer via command lines (*the programmer style*) and the attitude of the user

interacting with GUIs. As a bachelor's student in Mathematics, I opened my eyes to the APIs in 1981 during the second Latin American Forum of Informatics and Education held in Toluca City, Mexico (the first one had been held in Spain a year before). As an apprentice of programming I began to think about how to produce moderate-resolution graphics and interact with the user without command lines. It was still the time of the computers of the 4th generation, but we were experiencing the end of a transitional era characterized by the rise of computers *moved* by microprocessors: integrated circuits of high density and with amazing speed. As a historic note we can mention that the production of the first microprocessor, the Intel 4004 occurs in the year 1971 and marks the beginning of the personal computers or microcomputers. Circuits begin to be placed within chips LSI (Large Scale Integration Circuits) and VLSI (Very Large Scale Integration circuit). A simple *chip* contained the control and the arithmetic/logic units. The third component, the primary memory, was operated by other "chips". Computers such as the Altair (marked as the first personal computer, which went on sale in 1975), the Apple II, the Tandy TRS-80 (Radio Shack), PET Commodore and its later versions, were already extremely cheap and small, in such an extent, that they began to be used by non-scientists. During the 4th generation not only microcomputers were the novelty: it marked the beginning of the high-performance supercomputers, and the creation, in 1983, of C++ programming language.

My first confrontation in the battle "command line vs. GUI" arose in the private school in which I initiated my work as a high school teaching assistant in the period 1982-1984, while I was a Maths student. Although at that time the referred high school had no computer labs, the school managers took the initiative to offer computing courses to students so that these would be the "first ones" in learning computer programming. It took only two weeks of stand-by before getting the "new computers"... Surprisingly, we received a multiuser minicomputer with 20 terminals with UNIX! Of course our students were completely disappointed for having to use commands lines. Although some enthusiastic students were daring to learn the "text-style" facet of the computers, the use of GUIs had started to be an everyday experience, especially derived from the fact that in the early 80's the first video games and visual educational programs had been already implemented in cheap microcomputers. It was the apogee of the Ataris!

1985 marked the start of the Windows 1.0 operating system and also the year I leaped as a Maths and Stats teacher for Biology students at the University of Yucatan, Mexico. At that time, non-GUI interactions with the computer were still important, especially for file management (MSDOS operating system was still current). However, certain tasks such as writing papers or presentations, calculations and analyses performed in spreadsheets or statistical packages, and even handling information in database managers (in 1987 was in vogue DBase III, and a year later appeared the Dbase IV) already involved GUI environments. The course I taught in 1987 was called "Informatics" and adopted this hybrid approach. The students covered aspects of computer architecture, operating systems, and programming languages (BASIC, DBase), devoted to understand the basics of database programming, and statistical application programs. We were in a transitional stage, and the shift toward application programs with GUI was de facto, though not a recognized shift yet.

Something obvious for the user of the GUIs was the remarkable increase in requirements or resources of RAM for the execution of GUIs. By its very nature, graphical information tends to be larger and more complex than the text-based information. Consider, for example, the sizes of your favorite text editor documents: typically are measured in kilobytes, whereas the image files are measured in megabytes. The growing demands of the resources of the computer by the GUIs partly obeyed to increases in the size of the operating systems seeking to be "smarter". In the race to establish themselves in computational tasks, the GUIs allowed new users to learn things quick'n easy, in comparison with a command line interface. For example, statistical programs like SPSS, MINITAB, GENSTAT, which in their first versions relied on command lines, adopted the graphical interfaces. As a result, they expanded their market of users and established themselves as tools of data analysis in diverse areas of the sciences and humanities. However, the referred programs lost versatility, and with a few exceptions they did not get updated to changes in programming languages and computer sciences. Concepts that began to be coined since the 70's in the computer, such as event-driven or object oriented programming; these paradigms are applied

directly to the GUIs, but the user did not acknowledged them, mainly because such concepts were exclusively used for the programmer of APIs and not for the "final" user.

COMPUTING IN THE 21ST CENTURY

Object-oriented programming (OOP) languages have their origin in Simula 67 (Holmevik, 1994), a language designed for simulations created by Ole-Johan Dahl and Kristen Nygaard, at the Norwegian Computing Center in Oslo. This Center was working on simulations of ships, which were confounded by the combinatorial explosion of how the different attributes from different ships could affect one another. The strategy in this sort of problems was to group different types of ships into different classes of objects, each class of objects to define its own data and behaviors. The OOP works with objects, lists (sets of objects), the objects are classified by types, modes, etc. The OOP took up position as the style of programming predominantly in the mid-1980s, largely due to the influence of C, and prevailed due to the boom of the GUIs, but exclusively on the side of application developers. Very few *scientisticians* (scientists skilled in the application of statistical methods) adopted the OOP approach; the exceptions were the users of the statistical computing language *S*, developed by John Chambers of Bell Laboratories. *S* gave place to a commercial implementation of the program, *S Plus*. In the early 90's of last century, Ross Ihaka and Robert Gentleman of the University of Auckland, New Zealand, experimented with an improved version of the *S* language and called it *R* (R Core Team, 2014). R is an open-source language and statistical programming environment that manipulates objects and allows OOP. The strength of R is more than an object manipulation program: it is a computational system that includes a software environment for statistical computing and plotting. R is the realization of a technosocial attitude towards computers and software at present times: the distribution of free-access multiplatform applications (i.e. for different operating systems).

What has struck to the most of the scientisticians about the R programming environment is the GUI implemented in Windows, Mac OS or Linux. This GUI, compared to commercial statistical packages is visually poor and seems to reflect one of the bad impressions that the user has about R: its programming interface is not a "friendly" one. This unexciting feature of R is caused by the expectation that R should behave like other software packages. Another people's complain is that R is much more complex than other programs for statistical analysis. However, R actually integrates a wide range of statistical methods: from the simplest procedure to the most sophisticated. On the other hand, when one begins to learn to use software as SAS, MINITAB or SPSS, it is enough to have the ability of running a few commands to read and to analyze information. Nevertheless, when one becomes bolder in analyzing information of a novel way, it is the moment to learn new procedures or languages. For example, in SAS it is necessary to learn to construct macros or, in case of the production of high quality graphs, it is necessary to learn to use SAS ODS. And if it is desirable to manipulate information in matrix form, one will have to learn SAS IML with its own set of commands and rules. With R it is not necessary to add complexity because it integrates all its capacities in itself. As you learn R, you will realize that there is less complexity and more power. Definitely, those scientisticians who have crossed the border between statistical programs with GUIs and those programs involving some kind of programming (e.g. R, Matlab, etc.), are rewarded in obtaining conscious results, minimizing the *black boxes* in their daily routine, as far as the analysis of data is concerned. These programming skills are surprisingly common in established disciplines of contemporary biology. Areas like Bioinformatics have few implementations of programs with GUIs, so most of the time the user has to write their own programs.

Another hot topic that is worth of mention regarding GUIs *vis a vis* programming tools in Science is the confirmation of how easy is nowadays to solve mathematical (theoretical) problems using GUIs implementing symbolic computation. In our traditional 20th century Biomaths courses, the student had to treat the underlying theory completely and algorithmic examples had to be studied in all details, using paper and pencil. This strategy changed since the early 90's, when it was possible to execute a black box phase in which problems were solved using symbolic computation software systems (Buchberger, 1990). In our current courses, GUIs like Maple and Mathematica are good examples of symbolic computation software, quite valuable in showing graphically and easily mathematical derivations that would take much time to get analytically. The

dangers of these implementations are not the programs themselves but to forget that the "white box" facet of learning mathematics (the theory and manual execution of algorithms) must be integrated iteratively to the black box phase. Again, it is suggested that learning a programming language would be more fruitful in both phases, as the programmer will need to express in a computer code the algorithmic thinking.

The message to all Biology prospects is that computational problems in Bioinformatics and Biomathematics involves learning a programming language; a task that requires a proper time and space, and that unfortunately is has been losing in the curricula; at least this is what I have witnessed in Life Sciences careers in Mexico. The computing course referred above (Informatics) disappeared from the curriculum of the Universidad Autónoma de Yucatán, Mexico in 1996, and no course of that sort has returned to take its place (Navarro & Barrientos, 2013). An interpretation on this elimination is that from the last two decades of the 20th century, and up to this second decade of the 21st century, GUI-type application programs relegated the Scientific Computing to more and more limited groups. All this has had an adverse effect on the upgrading of university curricula, mainly in Biosciences. Since a tool (the computer) was already in common use (as the television or the phone has become), it was thought that in the university education scientific computing courses are not needed. This was and has been a regrettable mistake, since it can be shown that, for example, generations of university students trained with basic computing courses at the end of the 20th century developed better skills for problem-solving using algorithmic thinking and programming, in comparison with our new generations. Once Bioscientists were warriors!

It is perceived that curricula of universities in which Computing courses for life sciences have been kept, there are more chances of integration of disciplinary problem-solving with a contemporary quantitative/technological approach. Object-oriented languages, concurrency of computational processes, network systems, portability of programs (e.g., JAVA), are only some examples of the topics to be covered in the basic courses for Bioscientists today, along with the conventional topics about architecture, databases, or statistical computing. Regardless of the contents, an element that needs to be kept as the core of Scientific Computing, is the resolution of problems using algorithms, a useful strategy in the Biosciences.

CONCLUSION

Anyone observing how the interaction man-computer has evolved, will notice that the concept of *user* has changed as a consequence of the GUIs. The programming languages and GUI-based application programs are undeniably powerful nowadays and allow the resolution of increasingly complex problems in the Life Sciences. This makes the user unintentionally be less concerned about the algorithms or the routines used for their operation, and exaggeratedly relies on computers and programs that are rather colorful with respect to the graphical interfaces they use, *virtual reality style*, but that at the same time are black boxes (or *transparent* boxes) for us, the users. It is urgent that the scientists, and generally speaking, *the users*, be re-educated in Scientific Computing matters. One way to accomplish this is through the instrumentation of basic courses in Scientific Computing. Another way, quite useful for the scientist, is through the learning of programming languages implemented in Matlab (The MathWorks Inc, 2014) or R. In particular, R brings together the qualities of being a language supported by current paradigms of computing (e.g., object-oriented programming), and being universal in scope as an open-source (free) language.

REFERENCES

Buchberger, B. (1990). Should Students Learn Integration Rules? *Association for Computing Machinery. Special Interest Group on Symbolic and Algebraic Manipulation Bulletin, 24*(1), 10-17.

Holmevik, J. R. (1994). Compiling Simula: A historical study of technological genesis. *IEEE Annals in the History of Computing, 16*(4), 25–37.

Jansen, B. J. (1998). The Graphical User Interface: An Introduction. *SIGCHI Bulletin, 30*(2), 22-26.

Navarro, J., & Barrientos, R. (2013). [In Spanish] ¿Dónde quedó el cómputo científico? Avances y retrocesos de las herramientas computacionales en las ciencias biológicas. *Revista*

Iberoamericana para la Investigación y el Desarrollo Educativo (Online). Publicación 10, Enero-Junio 2013.

R Core Team (2014). *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org

The MathWorks, Inc. (2014). *Matlab 8.3 (R2014a).* Natick, MA.