

Numerical Calculations Using Maple: Why & How?

E.V. Corrêa Silva^{*}, *L.G.S. Duarte*[†], *L.A. da Mota*[‡] and *J.E.F. Skea*[§]

Abstract: The possibility of interaction between Maple and numeric compiled languages in performing extensive numeric calculations is exemplified by the `Ndynamics` package, a tool for studying the (chaotic) behavior of dynamical systems. Programming hints concerning the construction of `Ndynamics` are presented. The `system` command, together with the application of the black-box concept, is used to implement a powerful cooperation between Maple code and some other numeric language code.

Keywords: Dynamical systems, chaos, numeric calculations, non-symbolic compiled languages, black-box

Introduction

In this paper we explore the possibility of interaction between Maple and numeric languages, exemplified by the `Ndynamics` package [8] — a tool for studying the (chaotic) behavior of dynamical systems.

Maple interaction facilities may benefit users and programmers of both symbolic and numeric languages. As far as the implementation of extensive *numerical* calculations is concerned, non-symbolic compiled languages like FORTRAN, PASCAL or C are still more popular than any symbolic environment; the existing collections of efficient programs written in such languages could not be simply overlooked.

This paper is organized as follows: first, the concept of dynamical systems is introduced to the reader, as well as to the concepts of fractal dimension and boundary [4], extensively used in the paper; the commands of the package are then briefly described and an example of utilization follows. Finally we point out some useful considerations about programming design, as to the possibility of interaction between Maple and other languages.

Dynamical Systems

Systems of ordinary differential equations (or *dynamical systems*) play a central role in a large number of problems in all areas of scientific research, mainly in physics. Hence the importance of developing tools to study these systems — in particular, nonlinear systems. A n -dimensional dynamical system can be generically represented by

$$\frac{d\mathbf{X}}{dt} = \mathbf{F}(\mathbf{X}, t), \quad (1)$$

where $\mathbf{X} = (X_1(t), X_2(t), \dots, X_n(t))$ and $\mathbf{F} = (F_1(\mathbf{X}, t), F_2(\mathbf{X}, t), \dots, F_n(\mathbf{X}, t))$. The variables $X_i(t)$, where $i = 1, 2, \dots, n$, represent the relevant quantities in some physical model, and t is a continuous parameter (time, for instance). Each $F_i(\mathbf{X}, t)$ is an arbitrary function of the variables \mathbf{X} and of the parameter t .

Roughly speaking, “chaoticity” means extreme sensitivity to small changes in the initial conditions. This is the typical case of 3-dimensional nonlinear dynamical systems, among which chaotic behavior is a rule, rather than an exception: due to nonlinearity, small fluctuations in the initial conditions propagate dramatically, so that two neighbouring initial conditions may yield orbits with completely distinct asymptotic behavior.

Chaotic systems have a richer structure than “well-behaved” ones, hence their interest. Strange attractors and repellers, as well as fractal boundaries are examples of peculiarities of chaotic systems. A measure of the degree of chaoticity of a system can be obtained by the evaluation of the fractal dimension of the boundaries.

In the next section, we are going to elaborate further the concept of fractals and introduce a method to calculate the associated fractal dimension.

Fractals

Fractal Dimension

The idea of fractal will always lack a precise definition [1]. However, when one refers to a set F as a fractal, one typically has the following in mind:

1. F has a “fine structure”; i.e., has complex details on arbitrarily small scales;
2. F is too irregular to be described in the traditional geometrical language, both locally and globally;

^{*}Centro Brasileiro de Pesquisas Físicas, R. Dr. Xavier Sigaud, 150, Urca, CEP 22290-180, Rio de Janeiro, RJ, Brazil. E-mail: ecorrea@cat.cbpf.br

[†]Universidade do Estado do Rio de Janeiro, Instituto de Física, Departamento de Física Teórica, R. São Francisco Xavier, 524, Maracanã, CEP 20550-013, Rio de Janeiro, RJ, Brazil. E-mail: lduarte@dft.if.uerj.br

[‡]Idem. E-mail: damota@dft.if.uerj.br

[§]Idem. E-mail: jimsk@dft.if.uerj.br

3. often, F has some form of self-similarity, perhaps approximate or statistical;
4. usually, the “fractal dimension” of F (defined in some way) is greater than its “topological dimension”;
5. in most cases of interest, F is defined in a very simple way, perhaps recursively.

There are a lot of manners to define the dimension of a fractal. An important one is called the *Hausdorff dimension* [2], one possible generalization of the “primitive” notion of dimension. Suppose we have a hypercube of edge a , its hypervolume V being given by

$$V = a^d, \quad (2)$$

where d is the hyperspace dimension. Let us divide the hypercube into N hypercubic cells of edge ϵ , we have

$$V = N\epsilon^d. \quad (3)$$

Dividing (3) by (2) we have:

$$1 = N\left(\frac{\epsilon}{a}\right)^d \quad (4)$$

Defining $\delta = \frac{\epsilon}{a}$ and expressing the number of cells as a function of δ (i.e., $N = N(\delta)$),

$$1 = N(\delta)\delta^d \quad (5)$$

Solving (5) for d , we obtain the standard definition of dimension:

$$d = -\frac{\ln(N(\delta))}{\ln(\delta)} \quad (6)$$

Let us now measure the dimension of a Cantor set, constructed through a recursive procedure pictured in figure

In figure 1 we take a segment of length ϵ_0 and divide it into three parts of equal length $\epsilon_1 = \epsilon_0/3$; the middle segment is then replaced by two segments of length ϵ_1 . We then apply the same procedure to each segment of length ϵ_1 , dividing them into segments of length $\epsilon_2 = \epsilon_1/3$, and so on. After M iterations, we will have a total of $N = 4^M$ segments of length $\epsilon_M = \epsilon_0/3^M$ each. Defining

$$\delta \equiv \frac{\epsilon_M}{\epsilon_0} = \frac{1}{3^M}, \quad (7)$$

we may write

$$\ln(N(\delta)) = -\frac{\ln(\delta) \ln(4)}{\ln(3)}. \quad (8)$$

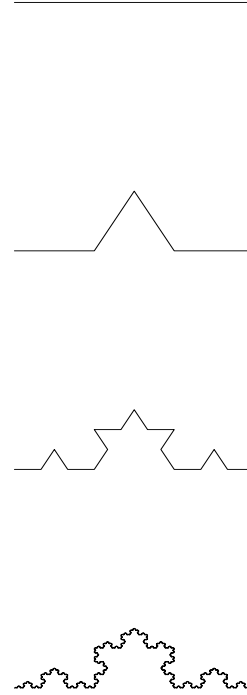


Figure 1: Cantor set after 0, 1, 2 and 6 iterations, respectively.

Taking the limit $M \rightarrow \infty$ (or $\delta \rightarrow 0$), the Hausdorff dimension (6) of the Cantor set is defined by the limit

$$d = -\lim_{\delta \rightarrow 0} \frac{\ln(N(\delta))}{\ln(\delta)} \quad (9)$$

which, in the case considered, yields

$$d = \frac{\ln(4)}{\ln(3)} \sim 1.26 \quad (10)$$

Fractal Dimension of Boundaries

Let R be a D -dimensional finite region — which we will take to be a hypercube of edge a , without loss of generality — divided into N hypercubic “cells” of edge ϵ . Let us choose an arbitrary cell C . If there are any pair of points (P, Q) in C so that the orbits of P and Q have distinct asymptotic behavior, C is said to be a boundary-cell. In the limiting case where N is infinite ($\epsilon \rightarrow 0$), the union of all boundary-cells constitute a *boundary*.

Let N_B be the number of boundary-cells in R . According to eq.(9), the fractal dimension of the boundary [4] can be inferred from the behavior of N_B as $\epsilon \rightarrow 0$ or, equivalently, as $\delta \rightarrow 0$, where $\delta = \epsilon/a$.

However, the total number of cells

$$N(\delta) = \left(\frac{1}{\delta}\right)^D \quad (11)$$

rapidly increases as $\delta \rightarrow 0$, and the computation of $N_B(\delta)$ results unpractical. An alternative approach is that of picking up N^* random cells in R and counting the number of boundary-cells N_B^* . If N^* is large enough to be statistically meaningful, we expect that

$$\frac{N_B^*}{N^*} \rightarrow \frac{N_B}{N}. \quad (12)$$

Supposing that, indeed,

$$\frac{N_B^*}{N^*} = \frac{N_B}{N} \quad (13)$$

we have

$$-\underbrace{\frac{\ln(N_B)}{\ln \delta}}_{d_B} = -\frac{\ln(N_B^*/N^*)}{\ln \delta} - \underbrace{\frac{\ln(N)}{\ln \delta}}_D. \quad (14)$$

Defining

$$\alpha \equiv D - d_B = \frac{\ln(N_B^*/N^*)}{\ln \delta} \quad (15)$$

we have

$$\ln\left(\frac{N_B^*}{N^*}\right) = \alpha \ln \delta \quad (16)$$

Both δ and N_B^*/N^* can be measured, allowing us to determine α and the fractal dimension d_B of the boundary.

Commands of the Package

In this section we present a brief description of the commands of the `Ndynamics` package. A more complete description can be found in the on-line help, provided with the package ¹.

- **Nsolve** allows the user to specify a system of differential equations and initial conditions, calculating trajectories in phase space and generating 2D/3D plots. Also, random initial conditions may be chosen from a user-defined region of phase space.
- **View** allows quick and comfortable visual inspection of the trajectories calculated by **Nsolve**, and the identification of regions of interest in phase space (e.g., chaos). Zooming in and out is supported, as well as choosing the variables to be shown (e.g., 2D or 3D graphs).
- **Boxcount** performs small random perturbations in the initial conditions and analyzes their effect upon the evolution of trajectories. Perturbed initial conditions are chosen from small regions sized by a user-defined perturbation parameter, around each unperturbed initial condition. In the neighborhood of a boundary, two small but distinct perturbations of the same initial condition may yield radically different perturbed trajectories; **Boxcount** determines the fraction of the total number of initial conditions for which this is so.
- **Fdimension** manages the execution of **Boxcount** for a row of values of the perturbation parameter, and analyzes the results obtained to evaluate the fractal dimension of the boundary.

Examples

Commands of the package

`Ndynamics` contains helpful tools for the detection of interesting regions (such as strange attractors, repellers, and fractal boundaries) in the phase space of a dynamical system. We will take the well-known Lorenz system as an example.

The package is loaded by the standard command `with`. The precision of calculations will be set to 16 digits, using Maple environment variable `Digits` ².

```
> with(Ndynamics);
> Digits := 16;
```

We define the Lorenz system, and assign some values to its constants:

¹ <http://www.dft.if.uerj.br/symbcomp.html>

²In what follows, the output of command lines has been occasionally omitted.

```

> Lorenz := {diff(x(t),t)=sigma*(y(t)-x(t)),
>           diff(y(t),t)=-x(t)*z(t)+R*x(t)-y(t),
>           diff(z(t),t)=x(t)*y(t)-b*z(t)};
> sigma:=10;b:=8/3;R:=28;

```

The command `Nsolve` generates a set of random initial conditions within a user-defined region in phase space, and then calculates the orbit of each individual initial condition. The number of initial conditions to be generated is controlled by the global variable `number_ic`. In addition, the color of each individual orbit can be defined by a boolean expression regarding the final condition of the orbit, to be assigned to the global variable `Colouring`. According to the value of that expression, an orbit may be plotted in one out of two predefined colors.

```

> number_ic:=8;
> Colouring:=x(t)<-4 and x(t) > -11;

```

For the sake of syntax clarity only, the following assignments are performed: the variable `init_region` defines ranges for each dependent variable, thus limiting the region in phase space for initial conditions; `t_range` sets the independent variable range; steps to the independent variable are defined by `t_calc_step` (for calculation purposes) and `t_plot_step` (for plotting purposes).

```

> init_region := [x=-0.37717..-0.37716,
>                y=0.48685..0.48686, z=-0.29894..-0.29893];
> t_range := 0..37;
> t_calc_step := .005;
> t_plot_step := .01;

```

The command `Nsolve` may now be run. The standard Maple function `time` is employed to give the reader an estimate computation time (which is machine-dependent of course).

```

> t0:=time():
> graph := Nsolve(Lorenz, [init_region,
> [t=t_range,t_plot_step]], [x(t),z(t)],
> random,method=[rk5C,t_calc_step]):
> tf := time()-t0;

```

(...some omitted output ...)

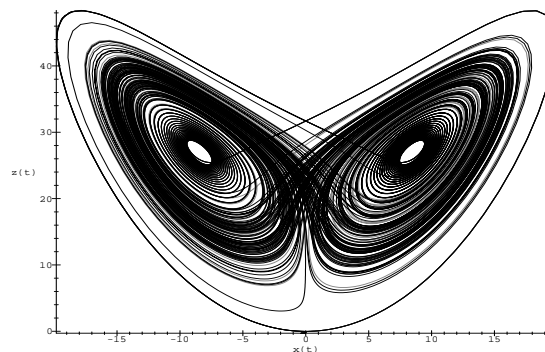
`tf := 123.880`

We have used the option `method=[rk5C,...]`, i.e., we have used the ‘plugged module’ in C to perform the numeric calculations. The result can be plotted, for instance, by using the command `display` of the standard Maple package `plots` (previously loaded by `Ndynamics`):

```

> display(graph);

```

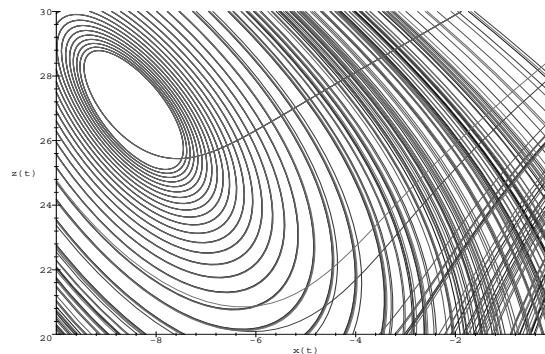


If the user is interested in detailing some particular region of phase space, the command `View` of `Ndynamics` is recommended:

```

> t0 := time();
> View([x=-10..0,z=20..30]); tf := time()-t0;

```



`tf := 51.569`

In order to exemplify the use of the commands `Boxcount` and `Fdimension`, let us slightly modify the Lorenz system parameters:

```

> sigma:=10;b:=8/3;R:=20;

```

and choose another region of initial conditions, as well as other range and steps for the independent variable:

```

> init_region2 := [x= -1.001 .. 1.001,
>                 y = -1.001 .. 1.001,
>                 z = 21.999 .. 22.001];
> t_range2 := 0..16;
> t_calc_step2 := 0.01;
> t_plot_step2 := 0.05;

```

Also, we will choose a much large number of initial conditions, and a different coloring criteria. Besides, all

three dependent variables (x, y, z) are to be shown in the resulting 3-D plot:

```
> number_ic:=10000;
> Colouring:= x(t)<0;
> frame := [x(t),y(t),z(t)];
```

Once more `Nsolve` generates random initial conditions and calculates their orbits:

```
> t0 := time():
> Nsolve(Lorenz,[init_region2,[t=t_range2,
>     t_plot_step2]],frame,initial,random);
> tf := time()-t0;
```

(...some deleted output ...)

$tf := 75.725$

The command `Boxcount` is now able to count the number of hypercubes over the boundary. Basically, this command takes perturbed initial conditions (those generated by `Nsolve`) according to a user-defined parameter, say, `epsilon`, and calculates their orbits. The final value of the independent variable is represented here by `final_time`, and the integration step by `integ_step`. (Once more, these variables are assigned just for the sake of clarity of the `Boxcount` command line.)

```
> epsilon := 0.0000002;
> final_time := 16;
> integ_step := 0.02;
> t0 := time():
> Boxcount(epsilon,final_time,method=
>     [rk5C,integ_step]);
> tf := time()-t0;
```

reading rk5 output

*From the, 10000, points (that were testable), , 200,
of them were close to the boundary.*

[200, 10000]

$tf := 170.560$

In its turn, the command `Fdimension` calculates the fractal dimension of the boundary, by picking up a given number of distinct values of the perturbation parameter within a user-defined range, applying `Boxcount` for each case.

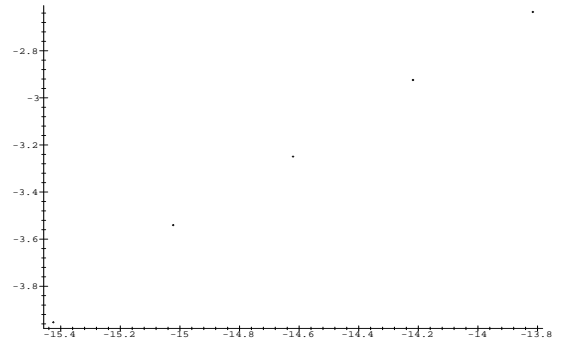
```
> epsilon_range := 0.0000002..0.000001;
> n_epsilon := 5;
> t0 := time():
> Fdimension(epsilon_range, final_time, n_epsilon,
>     method=[rk5C,integ_step]);
> tf := time()-t0;
```

(...some deleted output...)

Fractal dimension = ,2.213198116174276,

statistical error = 1.868128109494962, %

linear correlation = ,.9993933454816381



$tf := 964.940$

Where the slope of the straight line above (α) is defined by 16.

Performance of the Package

In the calculations above, the option `method=[rk5C, ...]` allowed us to employ the `Ndynamics` “plugged module” — the file `rk5.c` — containing the C source code for the algorithm of the 5-th order Runge-Kutta method.

To point out the advantages of our hybrid symbolic/numeric approach, which uses Maple to manage the source code generation and compilation for the number crunching, while maintaining Maple’s flexibility, table 1 presents a comparison of the elapsed time taken to perform the same calculation using the C interface and performing the entire calculation in Maple, using some of its numerical integration routines. In order to obtain a fair comparison, particular care was taken to use methods of the same order and adjusting parameters to produce solutions with the same precision. The following procedure was used: for a given set of initial conditions, Maple’s built-in high-precision Taylor Series integrator was used to integrate the Lorenz system in the interval $0 \leq t \leq 11$, until convergence was obtained to an accuracy of 13 decimal places; subsequently Maple’s global variable `Digits` was set equal to 13 (higher values of `Digits` seemed to cause problems with Maple’s integrator), and Maple’s inbuilt 5th-order Runge-Kutta-Fehlberg integrator, `rkf45`, was used up to $t = 11$. This integration was found to be accurate to 4 decimal places. The step size in the C routine was then adjusted to give the same³ precision, resulting in a step size of 0.002. Having matched the precision of both results, the averages of computation times for both methods was then taken for 200 integrations.

³in fact, the C routine was slightly more precise

The results below were obtained with Maple V.4 running in Windows 98 on an AMD-K6 266 with 64Mb of SDRAM. The C compiler used was Delorie's implementation of GNU's gcc⁴, with level 3 optimization.

Integration Method	C-interface rk5C	Maple rkf45
Seconds per Trajectory	0.1	7.0
Ratio to the Fastest Case	1	70

Table 1: Comparative performances of Maple's inbuilt numerical integrators and Ndynamics C interface, for the Lorenz system.

Programming and Design Considerations

Building computer programs demands methodical approach, if time and effort of construction are to be saved, and coherence to be kept. Furthermore, debugging and modification of a program relates directly to how apparent the *purpose* of each piece of code turns out. *Software design* is, in itself, a rich subject and field of research, barely touched here. Yet, we hope the reader may profit from reading our present considerations.

Interface with Compiled Languages

The numerical integration of a system of differential equations exemplifies a typical situation involving extensive numerical calculations. As far as the efficiency (basically, memory usage and processing time) of numerical algorithms is concerned, general-purpose symbolic computing systems (based on *interpreted* languages) generally offer less attractive means of implementation than non-symbolic *compiled* languages such as C, FORTRAN, and PASCAL. The long history of non-symbolic languages has already produced a large collection of efficient programs for a wide variety of specific problems — well-known examples are the NAG library of FORTRAN programs, and the collection of C programs in [7]. Using a symbolic environment for numerical calculations, discarding existing solutions (based on compiled languages) might sound like an attempt to “reinvent the wheel”.

The point is that symbolic languages provide flexibility and interactivity which lacks in “numeric-oriented” languages. In other words, symbolic code and data structures tend to be more friendly and versatile than numeric ones. For instance, in our specific case, the identification of chaotic regions in phase space is made

⁴<http://www.delorie.com>

much comfortable by the interactive user-interface of Maple. (Changing parameters and checking out results is faster and easier.)

In one aspect, the question is analogue to “Why using a high-level language instead of Assembly ?” No matter how fast Assembly programs may be, the kind of details involved in their construction (most of them totally unrelated to the problem at hand) certainly discourage its use. The more a given language allows its programmer to focus on “problem-related” details and *ignore* the rest, the most suitable the language becomes to find a solution to the problem. The very nature of ever-evolving problems in scientific research recommends the use of “higher-level” languages.

It would be highly desirable to combine the features of symbolic and numeric languages, for the task in view; that was one of the guidelines to the construction of our package. “High-level” tasks, so to speak, are handled by code written in Maple, whereas numerical calculations may be (optionally) performed by a piece of code written in some other language. The present implementation of the program contains an extension written in C; future versions will allow the user to simply “plug” his own (numerical) code. The implementation of these ideas in Maple was achieved through the utilization of the standard command `system`⁵ — which allows one to send commands directly to the operational system — as well as the possibility of exchanging data through files.

In the current implementation of Ndynamics, the command `system` is used for:

- compiling⁶ two source code files: `derivs.c`, containing the definition of the dynamical system; and `rk5.c`, containing the code for the 5-th order Runge-Kutta method. The executable file `rk5.exe` is then produced.
- executing `rk5.exe` to perform calculations. The files `rk5.in` and `rk5.out` are, respectively, the input and output files for `rk5.exe`.

All these operations are performed automatically, entirely from within Maple. The user needs not to be aware of any such processes.

The attentive Maple programmer will notice that the *utilization* of `rk5.exe` — assuming that `rk5.in` and `rk5.out` have the proper formats — does not depend whether its original source code was in C or any other language. This is an example of a “black-box”, a concept to be explored in what follows.

⁵Please refer to the Maple on-line help, by typing `?system`.

⁶Currently, the DJGPP-32 GNU compiler is used. Please refer to Ndynamics on-line help for more details.

Black-Boxes and Parameter Passing Techniques

A well-known concept in software design is that of a black-box [5]. For our present purposes, it can be pictured as a Maple procedure or an executable file which can be *used* without any assumption on its internal logic: all we need to know relates either to input and output data (meaning, form, values) or to *what* the procedure does (rather than *how* it is done). A good example of a black-box is the standard Maple function `sin`: all one needs to know to use `sin` is that it accepts a single input argument (a valid algebraic expression, assumed to be expressed in radians) and that its output is, also, an algebraic expression. Using the function `sin` does not require the knowledge of details concerning actual computations.

Ideal packages, made out of black-box-like procedures, would be most easily understood, and hence most easily tested, debugged and modified.

Exchange of data among procedures is a crucial aspect in the construction of a package. A Maple procedure may exchange data by

- using global or environment variables (input and output);
- using the result of its last executed line (output);
- using the RETURN command (output);
- reading from or writing to a file (input and output);
- using its parameters or arguments (input and output).

Of all these ways, the last one appears to us as the most interesting, as far the implementation of black-boxes is concerned. Using arguments of a procedure as a means of conveying *input* data is a common practice, needing no further comments. We would like to invite the reader to attend more closely to the possibility of passing *output* data through the arguments of a procedure — which is also a known technique [6, ?] found, for instance, in the standard commands `iquo` and `member`:

```
> iquo(11,2,'r');
> # r is an output parameter, to be
> # internally assigned to the
> # remainder of the integer division
> # 11/2
                    5

> r;
                    1
```

```
> member(x,[y,y,x,y],'pos');
> # pos is an output parameter, to be
> # assigned to the position of the
> # first occurrence of x in [y,y,x,y]
                    true

> pos;
                    3
```

Another example is the command `irem`, similar to `iquo`. Let us make our point clear with the help of a “toy procedure” `f`:

```
> f := proc(a::posint,sm_,pr_)
> local i;
> sm_ := convert([seq(i,i=1..a)],'+');
> pr_ := convert([seq(i,i=1..a)],'*');
> NULL;
> end;
```

Given a positive integer `a`, we compute the sum ($1 + 2 + \dots + a$) and the product ($1 * 2 * \dots * a$), which are assigned to the parameters `sm_` and `pr_`, respectively. An example of usage is

```
> f(5,'s,p'); # null output
> s,p;
                    15, 120
```

Now compare the procedure `f` to a slightly (but significantly) modified version `g`, where output parameters are not used:

```
> g := proc(a::posint)
> local i, s, p;
> s := convert([seq(i,i=1..a)],'+');
> p := convert([seq(i,i=1..a)],'*');
> s,p;
> end;
```

This version employs the “last executed line” technique. It would run as

```
> g(5);
                    15, 120
```

What is the practical difference between the two procedures? The reader should notice that, in order to interpret (and evaluate the correctness of) the output of `g`, he must hold the *extra* information that the *first* operand of the output sequence corresponds to $1 + 2 + 3 + 4 + 5$, and the *second* one corresponds to $1 * 2 * 3 * 4 * 5$ — i.e., the interpretation of the output depends on its particular (and arbitrary) form. In this example, it is a sequence, but it might as well be a list, or even a table. Were it a sequence or a list, the first operand might be the product, instead of the sum ... Well, whatever the choice as to the output form — which *has to* be made —, it would certainly have little to do with the actual purpose

of the procedure (i.e., calculating sums and products), and thus would be superfluous detail. A procedure calling `g` (and its poor programmer) would have to know all about these details, though; it (or he, or she) would be free from such concern, if `f` were used instead. The procedure `f` can be used as a black-box, whereas `g` can't.

When it comes to developing a large number of integrated Maple procedures, the black-box is a valuable concept to keep in mind, and so is the "output parameter" technique. That doesn't mean that all procedures should be written in that way, though. The "last line" and "RETURN" techniques, for instance, seem to be the most suitable to user-level procedures. The "output parameter" technique, in its turn, fits best on internal procedures, where all the "hard work" is carried out.

Conclusion

The powerful combination of Maple flexibility and interactivity with the high speed of numeric languages has been used in the implementation of `Ndynamics`, a package for the numerical study of (chaotic) behaviour in dynamical systems. A more detailed description of the functions and physical applications of `Ndynamics` can be found on [8].

We have also presented some of the programming hints and strategies used in its construction, hoping to have shown that there can be a stronger link between users (and programmers) of two kinds: those who are interested in heavy numerical calculations, and those whose interest lies in symbolic computations.

Part of the current implementation of `Ndynamics` has been written in C and has been "plugged", so to speak, into the main package. Currently, the user himself can write and use his own integration routine in C, as long as the same form of input and output data is kept⁷. Future versions of `Ndynamics` will allow the user to "plug and play" his own numerical piece of code and run the compiler corresponding to the language of his preference, provided that the input and output data follow the proper conventions.

Biographies

- **E.V. Corrêa Silva** is currently a Ph.D. student of Physics (Quantum Field Theories) at the Centro Brasileiro de Pesquisas Físicas, in Rio de Janeiro, Brazil. He obtained his Master degree in Physics at the Universidade Federal do Rio de Janeiro, in 1994. His research interests include the Quantum Hall Effect, Quantum Field Theory and Chaos, as well as System Design, Artificial Intelligence,

⁷For more detailed instructions, please refer to the on-line documentation.

Computational Physics and its relation to the teaching of Physics.

- **L.G.S.Duarte** is currently a member of staff at the Universidade Estadual do Rio de Janeiro (UERJ). He obtained his PhD in Physics at the Universidade Federal do Rio de Janeiro, in 1997. His interests concentrate on mathematical physics and he has recently participated on a project that lead to the creation of a differential equation solver, written on Maple, that was finally introduced on the comercial release number 5 of such mathematical package.
- **L.A.C.P.da Mota** is currently a member of staff at the Universidade Estadual do Rio de Janeiro (UERJ). He obtained his PhD in Physics at the University of Oxford, U.K, in 1993. He then held an one year position as a visiting researcher at the same University. His interests were then Elementary Particle Phenomenology. Most recently, his work has concentrated on mathematical physics and participated on the same project (mentioned on the biography above).
- **J.E.F.Skea** graduated from the University of Glasgow with a joint honours degree in Physics and Astronomy. He first worked with computer algebra as a research tool at the University of Sussex where, under the supervision of Roger Tayler and John Barrow, he completed his Ph.D. in the areas of general relativity and cosmology. Subsequently he worked for 3 years as a postdoc at the School of Mathematical Sciences of Queen Mary College London with Malcolm MacCallum, mainly on the CA systems SHEEP and REDUCE. He left science for a couple of years to work in Science Policy with Ben Martin at SPRU in the University of Sussex, on the analysis of performance indicators, but returned to physics for his third postdoc, this time at CBPF (Brazilian Centre for Physics Research) in Rio. He moved back to Scotland when he finally got a teaching position in the Department of Mathematics of the University of Aberdeen, where he initiated a course on the theory of computer algebra, using Maple in practical sessions. In 1996 he moved to the Theoretical Physics department of UERJ (Rio State University) where he seems to have settled down ... for now.

References

- [1] B. B. Mandelbrot, *The Fractal Geometry of Nature*. Freeman, San Francisco, 1982.

-
- [2] F. Hausdorff, *Dimension und äusseres Mass*, Math. Annalen **79**, 157 (1918).
- [3] E. N. Lorenz, *Deterministic Nonperiodic Flow*, J. Atmos. Sci. **20**, 130 (1963).
- [4] E. Ott, *Chaos in Dynamical Systems*, Cambridge University Press, 1997.
- [5] Yourdon, E. and Constantine, L.: *Structured Design*. Yourdon Press, 2nd ed. New York, 1978.
- [6] Heck, A.: *Introduction to Maple*. Springer-Verlag, New York, 1993.
- [7] Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., *Numerical Recipes in C*, Cambridge University Press, second edition.
- [8] L. G. S. Duarte, L. A. C. P. da Mota, H. P. de Oliveira, R. O. Ramos and J. E. F. Skea: *Numerical Analysis of Dynamical Systems and the Fractal Dimension of Boundaries*. Submitted to Computer Physics Communications, **chao-dyn/9812029**.