Sistema de Lorenz

L.M. Duque Valencia,[1] L. Del Castillo Detoeuf,[2] J.F. Reyes Botero[3]

[1,2,3]Universidad Nacional de Colombia
Departamento de Física, Facultad de Ciencias
Programación e Introducción a los Métodos Numéricos
{[1]lmduquev, [2]ldeld, [3]jfreyesb}@unal.edu.co

Este proyecto se propone estudiar gráficamente el sistema de ecuaciones diferenciales conocido como sistema de Lorenz haciendo uso del método de aproximación de Runge-Kutta 4 y de herramientas computacionales estudiadas durante el semestre. En primer lugar se modeló el sistema usando un programa en lenguaje en C++, se graficaron las soluciones obtenidas en 3D y las proyecciones sobre los tres planos principales usando el programa Gnuplot. Finalmente, se modificaron ligeramente las condiciones iniciales del modelo para comprobar que se trata de un sistema caótico.

Introducción

El sistema de Lorenz, introducido por Edward Lorenz en 1963, es un modelo tridimensional de la dinámica atmosférica terrestre expresado en sistema no lineal y determinista de tres ecuaciones diferenciales. El modelo se basa en tres parámetros y las condiciones iniciales del sistema de ecuaciones: σ (número de Prant), ρ (número de Rayleigh) y β. El sistema se expresa de la siguiente forma:

$$\dot{X} = \sigma(Y - X) \tag{1}$$

$$\dot{Y} = X(\rho - Z) \tag{2}$$

$$\dot{Z} = XY - \beta Z \tag{3}$$

Donde X es la velocidad de la corriente de convección, Y es proporcional a la diferencia de temperatura entre corrientes de convección ascendetes y descendentes y Z es la desviación de la temperaratura respecto a la dependencia lineal de la altura. Este sistema es de gran interés matemático pues para ciertos valores de los parámetros muestra un comportamiento caótico con un atractor conocido como "*Mariposa de Lorenz*".
Para resolver este sistema de ecuaciones diferenciales, se utiliza el mtodo de "*Runge-Kutta 4*", el cual consiste en obtener la solución del problema inicial planteado, por medio de la aproximación en cada punto de la malla, basándose en el resultado obtenido para el punto anterior. Teniendo como problema inicial por ejemplo:

$$\frac{dy}{dt} = f(t, y), \quad t_0 < t < t_n \tag{4}$$

$$y(0) = c \tag{5}$$

Donde la malla est determinada por $t_0, t_1, ..., t_n$ de paso h, pasa obtener un resultado aproximado en estos puntos.
El mtodo de RK4 es una generalización de la fórmula básica de Euler $y_{i+1} = y_i + hf(t_i, y_i)$, en los que los valores de f se reemplazan por un promedio ponderado de f en $t_i < t < t_{i+1}$; donde el orden del método es la cantidad de términos que se usan en el promedio ponderado, en este caso ser de orden 4.

Programación

El programa implementado para aproximar las soluciones del sistema de Lorenz consiste en 5 funciones y una estructura, e imprime los valores sucesivos de X, Y, y Z cada 10^{-4} segundos en un rango de tiempo de 0 a 50. En primer lugar, se declara una estructura con tres variables *double* x_0, y_0, y z_0 que representan los valores de las ecuaciones en un tiempo t:

```
struct ecuaciones {
  double x0;
  double y0;
  double z0;
} ecuacion;
```

En seguida, implementamos tres funciones *dx, dy* y *dz*, una para cada ecuación diferencial del sistema de Lorenz. Usamos los siguientes valores para los parámetros del sistema:

$\sigma = 10$, $\rho = \frac{8}{3}$, $\beta = 28$.

```
double dx(double xx, double yy, double zz) {
  double ff = 10 * (yy - xx);
  return ff;
}

double dy(double xx, double yy, double zz) {
  double gg = xx* (28 - zz) - yy;
  return gg;
}

double dz(double xx, double yy, double zz) {
  double hh = xx*yy - zz*(8.0/3.0);
  return hh;
}
```

Después introducimos una funciń que realice los cálculos del método RK4. Esta función toma 4 argumentos: las variables x_0, y_0, y z_0 en un tiempo t_i y un Δt que llamamos *step*. Esta función usa un total de 12 variables (los k, l, y m necesarios en el método RK4) y llama a las tres funciones de las ecuaciones para aproximar los valores del sistema en el tiempo $t_{i+1} = t_i + \Delta t$. La función finaliza modificando los valores x_0, y_0, y z_0 de la estructura por los calculados con el método RK4:

```
//funcion que calcula valores de rk4
void rk4(double x0, double y0,double z0, double step) {

  double k0 = step * dx(x0, y0, z0);
  double l0 = step * dy(x0, y0, z0);
  double m0 = step * dz(x0, y0, z0);
  double k1 = step * dx(x0 + 0.5*k0, y0 + 0.5*l0, z0 + 0.5*m0);
  double l1 = step * dy(x0 + 0.5*k0, y0 + 0.5*l0, z0 + 0.5*m0);
  double m1 = step * dz(x0 + 0.5*k0, y0 + 0.5*l0, z0 + 0.5*m0);
  double k2 = step * dx(x0 + 0.5*k1, y0 + 0.5*l1, z0 + 0.5*m1);
  double l2 = step * dy(x0 + 0.5*k1, y0 + 0.5*l1, z0 + 0.5*m1);
  double m2 = step * dz(x0 + 0.5*k1, y0 + 0.5*l1, z0 + 0.5*m1);
  double k3 = step * dx(x0 + k2, y0 + l2, z0 + m2);
  double l3 = step * dy(x0 + k2, y0 + l2, z0 + m2);
  double m3 = step * dz(x0 + k2, y0 + l2, z0 + m2);

  ecuacion.x0 += (1./6)*(k0 + 2*k1 + 2*k2 + k3);
  ecuacion.y0 += (1./6)*(l0 + 2*l1 + 2*l2 + l3);
```

```
  ecuacion.z0 += (1./6)*(m0 + 2*m1 + 2*m2 + m3);

}
```

Finalmente, usamos la función *main* para asignar los valores de las condiciones iniciales a las variables de la estructura y el valor de Δt, que tomamos como 10^{-6}. Luego iteramos la función RK4 desde $t = 0$ a $t = 50$, con incremento Δt (la variable *step*), imprimiendo a cada paso los resultados de x_0, y_0, y z_0. Estos datos son guardados en un archivo de texto para ser graficados usando **Gnuplot**:

```
int main(void) {
  //para modificar las condiciones iniciales, descomentar "+pow(10, 6)"

  ecuacion.x0 = 0;
  ecuacion.y0 = 1 //+pow(10, -6);
  ecuacion.z0 = 0;
  double step = 0.0001;

  for (double t=0 ; t < 50; t+=step) {
    rk4(ecuacion.x0, ecuacion.y0, ecuacion.z0, step);
    std::cout<< ecuacion.x0 << "\t" << ecuacion.y0 << ecuacion.z0 << "\n";

    //para imprimir datos de los planos (descomentar slo una de las lneas):
    //std::cout << ecuacion.x0 << "\t" << ecuacion.y0 << "\n";
    //std::cout << ecuacion.x0 << "\t" << ecuacion.z0 << "\n";
    //std::cout << ecuacion.y0 << "\t" << ecuacion.z0 << "\n";

    //para imprimir Y en funcin de t
    //std::cout << t << "\t" << ecuacion.y0 << "\n";
  }
  return 0;
}
```

Como se puede observar en los comentarios, para obtener los datos y graficar las proyecciones sobre los tres planos XY, XZ y YZ, y la gráfica de Y en función de t, usamos el mismo programa modificando únicamente la última línea del ciclo *for* para que imprima sólo los valores necesarios (X y Y, X y Z, Y y Z, y t y Y, respectivamente).

Graficar los resultados

Al graficar los resultados en **Gnuplot**, sólo se usaron uno de cada 10 datos para que la talla de los archivos no fuera demasiado grande. Usamos el siguiente código para generar los pdf:

```
gnuplot> set term pdfcairo
gnuplot> set output "lorenz.pdf"
gnuplot> set title "Sistema de Lorenz"
gnuplot> set xlabel "x"
gnuplot> set ylabel "y"
gnuplot> set zlabel "z"
gnuplot> splot "datos.txt" every 10 notitle pointtype 0 lt -1
gnuplot> set output
```

Obtuvimos la siguiente "Mariposa de Lorenz":

Podemos también observar el comportamiento caótico del sistema graficando las ecuaciones por aparte en función del tiempo y modificando solo una de las condiciones iniciales.

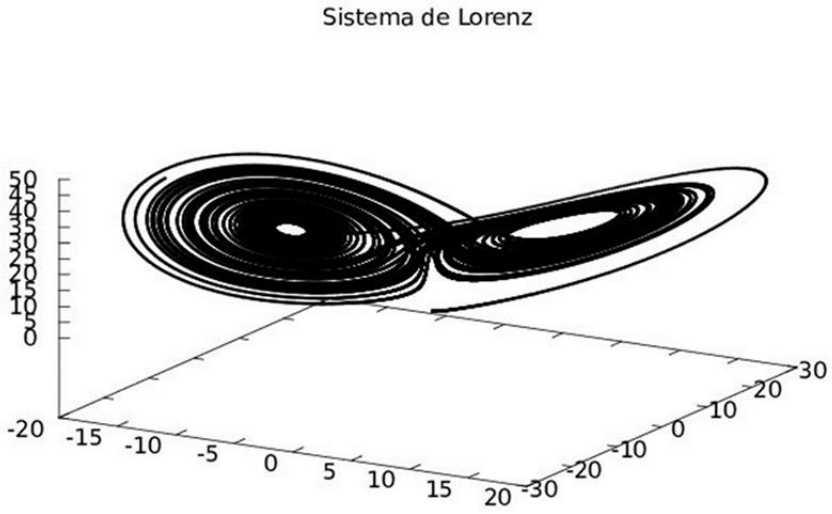
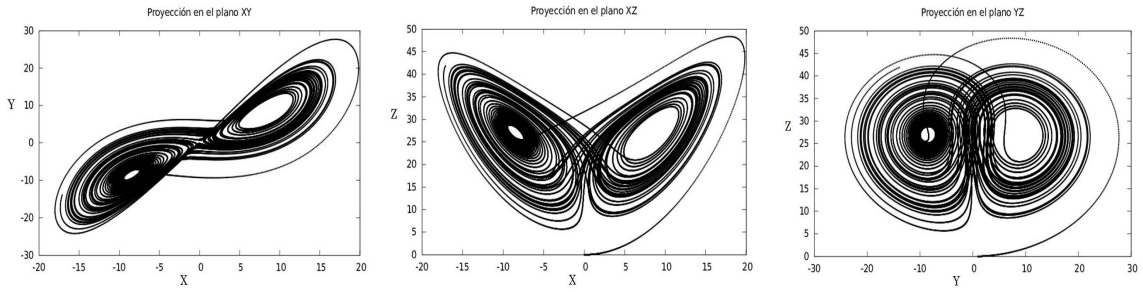


Figure 1: Gráfica tridimensional de las soluciones del sistema de Lorenz, conocidas como "Mariposa de Lorenz"



(a) En el plano XY **(b)** En el plano XZ **(c)** En el plano YZ

Figure 2: Proyecciones en los tres planos

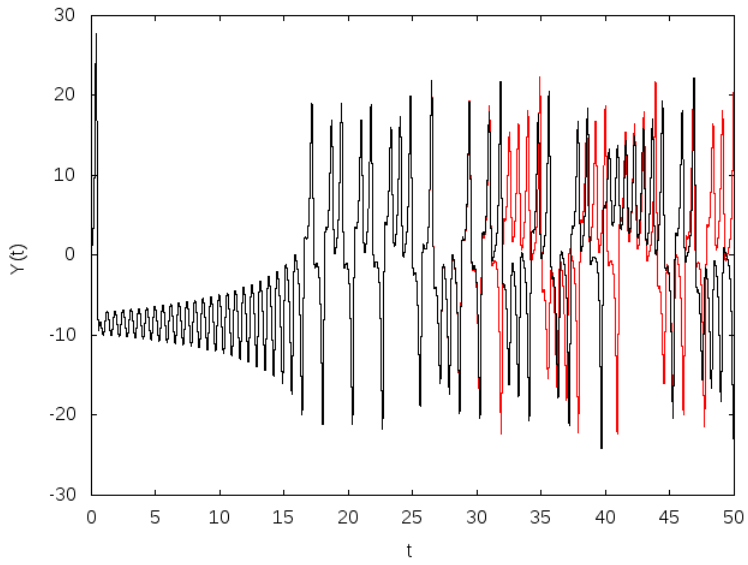


Figure 3: Gráficas de Y en función del tiempo. En negro se tiene la gráfica de la solución de Y usando las mismas condiciones iniciales que para la "Mariposa de Lorenz" ($X(0) = 0, Y(0) = 1, Z(0) = 0$). En rojo, se tiene la solución al alterar ligeramente las condiciones iniciales: en este caso, tomamos $Y(0) = 1 + \epsilon$ con $\epsilon = 10^{-6}$. Las dos soluciones son muy similares al inicio, pero alrededor de $t = 30$ comienzan a desviarse y resultan siendo completamente diferentes, lo que muestra el comportamiento caótico del sistema.

Para visualizar mejor la forma tridimensional de la "*mariposa*", usamos la terminal de *gif* de Gnuplot para animar la rotación sobre el eje z de la gráfica (ver "mariposa.gif" en el repositorio). Usamos los siguientes comandos de Gnuplot para siular la rotación:

```
gnuplot> set term gif animate
gnuplot> set output "Mariposa.gif"
gnuplot> set title "Mariposa de Lorenz"
gnuplot> set xlabel "x"
gnuplot> set ylabel "y"
gnuplot> set zlabel "z"
gnuplot> do for [a = 0 : 360] {
more> splot "datos.txt" every 10 notitle pointtype 0 lt -1
more> }
gnuplot> set output
```

Conlusiones

Se diseó y desarrolló un programa en lenguaje C++ para resolver el sistema de Lorenz con las condiciones iniciales dadas. El programa fue escrito en el editor de texto "*Emacs*" y compilado con "*g++*", con la implementación de una estrucutura para el valor de las condiciones inciales, y el cual itera a travs del mtodo RK4, para la solución de un sistema de ecuaciones diferenciales, de 1 a 50 con un intervalo de 10^{-4}. Una vez obtenidos las aproximaciones numéricas del sistema, se realizó una gráfica por medio del programa "*Gnuplot*", imprimiendo cada 10 datos, es decir 50000 datos en total, para que el progrma no se saturara de información y pudiese correr de manera eficiente; para cada gráfica se tomaron los datos necesarios: gráfica tridimensional x vs y vs z y planos x vs y, x vs z, y vs z. Luego, se graficó el mismo sistema incial pero con una ligera modificación en una de las condiciones iniciales, tomado en este caso $Y(0)$, y con la gráfica de este respecto al tiempo t se puede observar que con una mnima alteración en las condiciones inciales el sistema puede cambiar su comportamiento en una diferencia bastante notable, mostrando así el comportamiento caótico característico del sistema. Adicionalmente, para complementar el trabajo realizado, por medio de "*Gnuplot*" se implementó la rotación de la grfica tridimensional sobre el eje z obtenida anteriormente a manera de "*gif*".

Formatting Citations

Citations can be handled in one of three ways. The most straightforward (albeit labor-intensive) would be to hardwire your citations into your LaTeX source, as you would if you were using an ordinary word processor. Thus, your code might look something like this:

```
However, this record of the solar nebula may have been
partly erased by the complex history of the meteorite
parent bodies, which includes collision-induced shock,
thermal metamorphism, and aqueous alteration
({\it 1, 2, 5--7\/}).
```

Compiled, the last two lines of the code above, of course, would give notecalls in *Science* style:

> ... thermal metamorphism, and aqueous alteration (*1, 2, 5–7*).

Under the same logic, the author could set up his or her reference list as a simple enumeration,

```
{\bf References and Notes}

\begin{enumerate}
\item G. Gamow, {\it The Constitution of Atomic Nuclei
and Radioactivity\/} (Oxford Univ. Press, New York, 1931).
\item W. Heisenberg and W. Pauli, {\it Zeitschr.\ f.\
Physik\/} {\bf 56}, 1 (1929).
\end{enumerate}
```

yielding

References and Notes

1. G. Gamow, *The Constitution of Atomic Nuclei and Radioactivity* (Oxford Univ. Press, New York, 1931).
2. W. Heisenberg and W. Pauli, *Zeitschr. f. Physik* **56**, 1 (1929).

That's not a solution that's likely to appeal to everyone, however — especially not to users of BIBTEX [**?**]. If you are a BIBTEX user, we suggest that you use the `Science.bst` bibliography style file and the `scicite.sty` package, both of which we are downloadable from our author help site (http://www.sciencemag.org/about/authors/prep/TeX_help/). You can also generate your reference lists by using the list environment {`thebibliography`} at the end of your source document; here again, you may find the `scicite.sty` file useful.

Whether you use BIBTEX or {`thebibliography`}, be very careful about how you set up your in-text reference calls and notecalls. In particular, observe the following requirements:

1. Please follow the style for references outlined at our author help site and embodied in recent issues of *Science*. Each citation number should refer to a single reference; please do not concatenate several references under a single number.

2. Please cite your references and notes in text *only* using the standard LATEX `\cite` command, not another command driven by outside macros.

3. Please separate multiple citations within a single `\cite` command using commas only; there should be *no space* between reference keynames. That is, if you are citing two papers whose bibliography keys are `keyname1` and `keyname2`, the in-text cite should read `\cite{keyname1,keyname2}`, *not* `\cite{keyname1, keyname2}`.

Failure to follow these guidelines could lead to the omission of the references in an accepted paper when the source file is translated to Word via HTML.

Handling Math, Tables, and Figures

Following are a few things to keep in mind in coding equations, tables, and figures for submission to *Science*.

In-line math. The utility that we use for converting from LaTeX to HTML handles in-line math relatively well. It is best to avoid using built-up fractions in in-line equations, and going for the more boring "slash" presentation whenever possible — that is, for `$a/b$` (which comes out as a/b) rather than `$\frac{a}{b}$` (which compiles as $\frac{a}{b}$). Likewise, HTML isn't tooled to handle certain overaccented special characters in-line; for $\hat{\alpha}$ (coded `$\hat{\alpha}$`), for example, the HTML translation code will return [ˆ(α)]. Don't drive yourself crazy — but if it's possible to avoid such constructs, please do so. Please do not code arrays or matrices as in-line math; display them instead. And please keep your coding as TeX-y as possible — avoid using specialized math macro packages like `amstex.sty`.

Displayed math. Our HTML converter sets up TeX displayed equations using nested HTML tables. That works well for an HTML presentation, but Word chokes when it comes across a nested table in an HTML file. We surmount that problem by simply cutting the displayed equations out of the HTML before it's imported into Word, and then replacing them in the Word document using either images or equations generated by a Word equation editor. Strictly speaking, this procedure doesn't bear on how you should prepare your manuscript — although, for reasons best consigned to a note [**?**], we'd prefer that you use native TeX commands within displayed-math environments, rather than LaTeX sub-environments.

Tables. The HTML converter that we use seems to handle reasonably well simple tables generated using the LaTeX {`tabular`} environment. For very complicated tables, you may want to consider generating them in a word processing program and including them as a separate file.

Figures. Figure callouts within the text should not be in the form of LaTeX references, but should simply be typed in — that is, `(Fig. 1)` rather than `\ref{fig1}`. For the figures themselves, treatment can differ depending on whether the manuscript is an initial submission or a final revision for acceptance and publication. For an initial submission and review copy, you can use the LaTeX `{figure}` environment and the `\includegraphics` command to include your PostScript figures at the end of the compiled PostScript file. For the final revision, however, the `{figure}` environment should *not* be used; instead, the figure captions themselves should be typed in as regular text at the end of the source file (an example is included here), and the figures should be uploaded separately according to the Art Department's instructions.

What to Send In

What you should send to *Science* will depend on the stage your manuscript is in:

- **Important:** If you're sending in the initial submission of your manuscript (that is, the copy for evaluation and peer review), please send in *only* a PostScript or PDF version of the compiled file (including figures). Please do not send in the TeX source, `.sty`, `.bbl`, or other associated files with your initial submission. (For more information, please see the instructions at our Web submission site, http://www.submit2science.org/ .)
- When the time comes for you to send in your revised final manuscript (i.e., after peer review), we require that you include all source files and generated files in your upload. Thus, if the name of your main source document is `ltxfile.tex`, you need to include:
  - `ltxfile.tex`.
  - `ltxfile.aux`, the auxilliary file generated by the compilation.

– A PostScript file (compiled using `dvips` or some other driver) of the `.dvi` file generated from `ltxfile.tex`, or a PDF file distilled from that PostScript. You do not need to include the actual `.dvi` file in your upload.

– From BIBTEX users, your bibliography (`.bib`) file, *and* the generated file `ltxfile.bbl` created when you run BIBTEX.

– Any additional `.sty` and `.bst` files called by the source code (though, for reasons noted earlier, we *strongly* discourage the use of such files beyond those mentioned in this document).

Fig. 1. Please do not use figure environments to set up your figures in the final (post-peer-review) draft, do not include graphics in your source code, and do not cite figures in the text using LaTeX `\ref` commands. Instead, simply refer to the figure numbers in the text per *Science* style, and include the list of captions at the end of the document, coded as ordinary paragraphs as shown in the `scifile.tex` template file. Your actual figure files should be submitted separately.