Sistemas de Ecuaciones Diferenciales – Resolución por medio de Maple, Matemática, Gauss, Matlab y Macros en Excel

Jorge Mauricio Oviedo [1]

Resumen: El presente trabajo tiene por objetivo brindar un enfoque teórico accesible sobre Ecuaciones Diferenciales y su implementación diversos Softwares. Para llevar a cabo dicha tarea se presenta una revisión teórica de tal tópico y se procede a implementarlo en todo tipo de software tanto algebraico y numérico. Se exponen y se comparan alternativamente las resoluciones numéricas contra las algebraicas evaluando la eficiencia en cada programa. Se proveen además los códigos de programación usados en cada Software.

Palabras clave: Ecuaciones Diferenciales, Modelo Depredador Presa, Solución General y Particular, Métodos Numéricos, Runge-kutha, Maple, Mathematica, Matlab, Gauss, Excel, Macros, Visual Basic

[1] joviedo@eco.unc.edu.ar

EJERCICIO Nº 8

1- INTRODUCCIÓN TEÓRICA[2]

1.1- ECUACIONES DIFERENCIALES

Una ecuación diferencial ordinaria (es decir con una sola variable independiente) es una ecuación en donde los argumentos que intervienen son la variable independiente de una función incógnita desconocida y las sucesivas derivadas de dicha función, es decir:

$$f(x, y, y', y'', ..., y^{(n)}) = 0$$

donde el problema, a diferencia de una ecuación algebraica, consiste en averiguar quien es $y(x)$, es decir, aquí el arte consiste en hallar una ***función*** que satisfaga la condición anterior y no simplemente un valor numérico como en el caso de una ecuación algebraica.

Existen distintas maneras de clasificar a las ecuaciones diferenciales: Una de ellas es la clasificación por el orden de la ecuación (que viene dado por la derivada de mayor orden
que aparezca en la expresión) y otra por la linealidad. Ésta última es quizás una de las clasificaciones más importantes en el sentido de que permite caracterizar inmediatamente la facilidad con que se pueden hallar sus soluciones. En este sentido, se puede decir que las ***ecuaciones diferenciales no lineales de primer orden*** pueden resolverse solo en algunos casos especiales (como lo es el caso de las ecuaciones diferenciales de Bernoulli, las exactas, las separables, las homogéneas, etc.) mientras que las ***ecuaciones diferenciales lineales de primer orden*** siempre pueden resolverse de manera exacta. Pero la distinción es quizás mas relevante cuando se hablan de ecuaciones de orden dos o superior ya que en el caso de ecuaciones lineales existe la posibilidad de hallar una forma manejable de solución (ya sea exacta o en forma de ***serie de potencias***), mientras que en las no lineales, hallar una solución suele ser todo un desafío. Esto no significa que una ecuación diferencial no lineal de orden superior no tenga solución sino más bien que no hay métodos generales para llegar a una solución explícita o implícita. Aunque esto parezca desalentador hay algunas cosas que se pueden hacer tales como analizar cuantitativamente la ecuación no lineal (a través de métodos numéricos) o cualitativamente (a través de diagramas de fases).
Pero aun quedan otras cuestiones que diferencian a una ecuación lineal de una no lineal y es el hecho de que las segundas pueden tener ***soluciones singulares***.

[2] La presente sección a sido elaborada en base al libro "Ecuaciones diferenciales con aplicaciones de modelado" de Dennis Zill.

1.2- SISTEMAS DE ECUACIONES DIFERENCIALES

Los sistemas de ecuaciones diferenciales también pueden clasificarse en lineales y no lineales y de primer orden o de orden superior. Pero en esta situación merece hacerse una consideración ya que "*todo sistema de orden superior a dos puede transformarse en uno de orden uno mediante la simple redefinición y agregado de variables"*. En consecuencia, el tratamiento de sistemas de ecuaciones diferenciales puede enfocarse en los sistemas de primer orden sin perdida de generalidad. Un sistema de ecuaciones de primer orden puede representarse de la siguiente forma:

$$\begin{aligned}
\frac{dx_1}{dt} &= f_1(t, x_1, x_2, ...x_n) \\
\frac{dx_2}{dt} &= f_2(t, x_1, x_2, ...x_n) \\
&\vdots \\
\frac{dx_n}{dt} &= f_n(t, x_1, x_2, ...x_n)
\end{aligned}$$

Como se mencionó en un principio las ecuaciones no lineales no pueden resolverse de una manera general y analítica por lo que este análisis se enfocará en sistemas lineales. Así, si cada una de las funciones $f_1, f_2, ..., f_n$ es lineal en las variables dependientes $x_1, x_2, ..., x_n$ entonces el sistema es lineal de primer orden. Su estructura general es:

$$\begin{aligned}
\frac{dx_n}{dt} &= a_{11}(t)x_1 + a_{12}(t)x_2 + ... + a_{1n}(t)x_n + f_1(t) \\
\frac{dx_n}{dt} &= a_{21}(t)x_1 + a_{22}(t)x_2 + ... + a_{2n}(t)x_n + f_2(t) \\
&\vdots \\
\frac{dx_n}{dt} &= a_{1n}(t)x_1 + a_{2n}(t)x_2 + ... + a_{nn}(t)x_n + f_n(t)
\end{aligned}$$

El cual se puede expresar en forma matricial como:

$$\mathbf{X' = AX + F}$$

donde:

$$\mathbf{X} = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \text{M} \\ x_1(t) \end{bmatrix} \qquad \mathbf{X'} = \begin{bmatrix} x'_1(t) \\ x'_2(t) \\ \text{M} \\ x'_1(t) \end{bmatrix} \qquad \mathbf{A}(t) = \begin{pmatrix} a_{11}(t) & \text{K} & a_{1n}(t) \\ \text{M} & \text{O} & \text{M} \\ a_{n1}(t) & \text{L} & a_{nn}(t) \end{pmatrix} \qquad \mathbf{F}(t) = \begin{bmatrix} f_1(t) \\ f_2(t) \\ \text{M} \\ f_n(t) \end{bmatrix}$$

La resolución general de este sistema no homogéneo se puede expresar como sigue:

$$\mathbf{X = X_P + X_C}$$

donde $\mathbf{X_p}$ es una solución particular de $\mathbf{X' = AX + F}$ y $\mathbf{X_c}$ es la solución del sistema homogéneo formado por $\mathbf{X' = AX}$, donde las funciones $f_i(t)$ se igualan a cero ($\mathbf{F = 0}$).
Esta es una generalización del caso de una ecuación diferencial individual.
En primer lugar se obtendrá $\mathbf{X_c}$, la solución complementaria, para luego tratar el caso más general, cuando el sistema no es homogéneo.

1.2.1- SOLUCIÓN DE UN SISTEMA LINEAL HOMOGÉNEO

Recuérdese que la sencilla ecuación diferencial lineal homogénea de primer orden (*dx/dt=ax,* donde a es una constante) tiene una solución general $x = ce^{at}$. Parece lógico preguntar si se puede definir una matriz exponencial e^{At} tal que el sistema homogéneo **X'=AX**, donde **A** es una matriz (nxn) de constantes, tenga una solución:

$$\mathbf{X} = e^{\mathbf{A}t}\mathbf{C}$$

Dado que **C** debe ser de orden (n *x*1) y esta formada por constantes arbitrarias, se desearía que $e^{\mathbf{A}t}$ sea una matriz de (n *x* n). Una forma de definir $e^{\mathbf{A}t}$ se basa en la representación en serie de potencias de la función escalar exponencial e^{at}:

$$e^{at} = 1 + at + \frac{a^2t^2}{2!} + \ldots + \frac{a^nt^n}{n!} + \ldots = \sum_{n=0}^{\infty} \frac{a^nt^n}{n!}$$

Esta serie converge para todo valor de *t*. Reemplazando 1 por la matriz identidad **I** y a la constante a por la matriz **A** de constantes, llegamos a la definición de la matriz exponencial:

$$e^{\mathbf{A}t} = \mathbf{I} + \mathbf{A}t + \frac{\mathbf{A}^2t^2}{2!} + \ldots + \frac{\mathbf{A}^nt^n}{n!} + \ldots = \sum_{n=0}^{\infty} \frac{\mathbf{A}^nt^n}{n!}$$

Se puede demostrar que esta serie converge a una matriz de (n *x* n) para cualquier valor de *t*. Sin embargo, esta definición no tiene utilidad a los efectos prácticos por lo que se hace mas que necesario hallar una forma alternativa de cálculo. Para ello, haciendo uso de algunos teoremas, se podrá hallar la matriz exponencial por medio de los valores y los vectores propios de la matriz del sistema asociado. El teorema dice que[3]:

> "Sea J la forma canónica de Jordan de una matriz **A** y sea $\mathbf{J} = \mathbf{C}^{-1}\mathbf{AC}$. Entonces $\mathbf{A} = \mathbf{CJC}^{-1}$ y
>
> $$e^{\mathbf{A}t} = \mathbf{C}e^{\mathbf{J}t}\mathbf{C}^{-1}$$

Demostración: Primero se observa que

$$\begin{aligned}\mathbf{A}^n &= (\mathbf{CJC}^{-1})^n = \underbrace{(\mathbf{CJC}^{-1})(\mathbf{CJC}^{-1})\ldots(\mathbf{CJC}^{-1})}_{n \text{ veces}} \\ &= \mathbf{CJ}(\mathbf{C}^{-1}\mathbf{C})\mathbf{J}(\mathbf{C}^{-1}\mathbf{C})\mathbf{J}(\mathbf{C}^{-1}\mathbf{C})\ldots(\mathbf{C}^{-1}\mathbf{C})\mathbf{JC}^{-1} \\ &= \mathbf{CJ}^n\mathbf{C}^{-1}\end{aligned}$$

Se sigue entonces que

$$(\mathbf{A}t)^n = \mathbf{C}(\mathbf{J}t)^n\mathbf{C}^{-1}$$

Así,

[3] Sacado de Grossman, capítulo 6, página 611.

$$e^{\mathbf{A}t} = \mathbf{I} + \mathbf{A}t + \frac{\mathbf{A}^2t^2}{2!} + ... = \mathbf{C}\mathbf{I}\mathbf{C}^{-1} + \mathbf{C}(\mathbf{J}t)\mathbf{C}^{-1} + \frac{\mathbf{C}(\mathbf{J}t)^2\mathbf{C}^{-1}}{2!} + ...$$
$$= \mathbf{C}\left[\mathbf{I} + \mathbf{J}t + \frac{\mathbf{C}(\mathbf{J}t)^2\mathbf{C}^{-1}}{2!} + ...\right]\mathbf{C}^{-1} = \mathbf{C}e^{\mathbf{J}t}\mathbf{C}^{-1}$$

Este teorema establece que para calcular $e^{\mathbf{A}t}$ solo se necesita obtener $e^{\mathbf{J}t}$. Las matrices **A** y **J** son semejantes entre ellas, con la particularidad que al ser **J** la forma canónica de Jordan es una matriz diagonal, cuando **A** tiene n valores propios distintos. Por lo tanto, si la matriz **A** tiene *n* valores propios distintos ($\lambda_1, \lambda_2 ... \lambda_n$), formamos la matriz **J** diagonal, y de ella podemos calcular $e^{\mathbf{J}t}$.

$$J = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix} \Rightarrow e^J = \begin{pmatrix} e^{\lambda 1t} & 0 & \cdots & 0 \\ 0 & e^{\lambda 2t} & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & e^{\lambda nt} \end{pmatrix}$$

En este caso, solo se necesita especificar la matriz **C** particular del ejercicio para poder obtener $e^{\mathbf{A}t}$. Como se tiene n valores propios distintos (los vectores propios de valores propios distintos son linealmente independientes), **C** se construye con un vector propio de cada uno de los distintos valores propios de **A**. Por lo tanto, la obtención de $e^{\mathbf{A}t}$ es bastante sencilla, siendo la única complicación posible la existencia de valores propios con parte imaginaria.

Puede darse el caso en que la matriz **A** no posea *n* valores propios distintos, sino que alguno/s posea/n multiplicidad algebraica mayor a uno. En este caso la matriz **J** ya no se compone únicamente de valores propios. Sabemos que para una ecuación diferencial lineal homogénea de segundo orden (recuérdese que puede ser transformada en un sistema de 2 ecuaciones diferenciales lineales de primer orden) de la forma $dx/dt = b(d^2x/dt) + ax$, cuando sus raíces son reales e iguales, la solución es de la forma:

$$x = c_1e^{at} + tc_1e^{at}$$

buscaremos una solución para el sistema que se le asemeje. En este caso, la matriz **J** ya no será diagonal.

Definiendo a la matriz $\mathbf{N_k}$ como una matriz de orden *k*, la cual posee *unos* sobre la diagonal principal y *ceros* en el resto de la matriz

$$\mathbf{N_k} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & \vdots \\ \vdots & \vdots & \vdots & & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}$$

Para cada valor propio de **A**, se define una matriz de bloques de Jordan $\mathbf{B}(\lambda_i)$ (de orden *k x k*) como:

$$\mathbf{B}(\lambda_i) = \lambda_i \mathbf{I} + \mathbf{N_k} = \begin{pmatrix} \lambda_i & 1 & \cdots & 0 \\ 0 & \lambda_i & \cdots & \vdots \\ \vdots & \vdots & \text{L} & 1 \\ 0 & 0 & \cdots & \lambda_i \end{pmatrix}$$

donde *k* estará dado por la multiplicidad algebraica del valor propio λ.
En este caso, la forma canónica de Jordan de **A**, estará compuesta por una matriz de bloques de Jordan en la diagonal principal y ceros en el resto de ella:

$$\mathbf{J} = diag\left[\mathbf{B}(\lambda_i)\right] = \begin{pmatrix} \mathbf{B}(\lambda_1) & 0 & \cdots & 0 \\ 0 & \mathbf{B}(\lambda_2) & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \mathbf{B}(\lambda_n) \end{pmatrix}$$

Donde los **B** y **0** son matrices de orden *k x k*. Si un valor propio posee multiplicidad algebraica 1, cabe aclarar que la matriz **B** será un escalar igual a dicho valor propio, y cuando todos los valores propios poseen multiplicidad algebraica 1, la matriz de bloques es la misma que la matriz de Jordan anterior (para el caso de n valores propios distintos).
La matriz e^{Jt} será parecida a la que se tenía anteriormente, pero para los valores propios con multiplicidad mayor a uno, se registrará sobre la diagonal principal $te^{\lambda t}$. Para el caso una matriz de 2 *x* 2 con una solo valor propio λ de multiplicidad 2, tenemos un $e^{\mathbf{J}t}$:

$$\begin{pmatrix} e^{\lambda t} & te^{\lambda t} \\ 0 & e^{\lambda t} \end{pmatrix}$$

En cuanto a la construcción de la matriz **C** para matrices **A** con un valor propio real repetido, tenemos dos subcasos: cuando el valor propio de multiplicidad algebraica mayor a uno posee igual multiplicidad geométrica[4], y cuando la primera es mayor a la segunda.
En el primer caso, podemos obtener n vectores propios linealmente independientes y ellos son los que componen (como antes) la matriz **C**. Pero en el segundo caso, ello ya no es posible y se recurre al concepto de vector propio generalizado, el cual se define como aquel vector w que:

$$(A - \lambda I)w = v$$

donde λ es el valor propio y *v* el vector propio de λ. En el caso de que la diferencia entre la multiplicidad geométrica y aritmética del valor propio λ sea de 1, es suficiente con este vector propio generalizado, pero cuando es mayor puede ser necesario recurrir una cadena de vectores propios generalizados, definidos por:

[4] La multiplicidad geométrica de un valor propio λ es mayor o igual a la multiplicidad algebraica y se define como la dimensión del espacio propio correspondiente a dicho valor propio. La dimensión de dicho espacio propio establece cuantos vectores linealmente independientes se pueden asociar a λ.

$$(A-\lambda I)v = 0$$
$$(A-\lambda I)w_1 = v$$
$$(A-\lambda I)w_2 = w_1$$
$$(A-\lambda I)w_3 = w_2 \quad \ldots$$

donde *v* es un único vector propio. Con éstos vectores linealmente independientes construimos la matriz **C** y nuevamente estamos en condiciones de obtener $e^{\mathbf{A}t}$.

1.2.2- SISTEMAS DE ECUACIONES LINEALES NO HOMOGENEO

En el caso de una ecuación diferencial lineal de primer orden *x'=ax+f(t)* (donde a es una constante) la solución general está dada por:

$$x = x_C + x_P = ce^{at} + e^{at}\int_{t_0}^{t} e^{-as} f(s)\, ds$$

Para un sistema no homogéneo de ecuaciones diferenciales lineales de primer orden, se puede demostrar que la solución general de **X'=AX+F** (donde **A** es una matriz de constantes) es:

$$\mathbf{X} = \mathbf{X_C} + \mathbf{X_P} = e^{\mathbf{A}t}\mathbf{C} + e^{\mathbf{A}t}\int_{t_0}^{t} e^{-\mathbf{A}s}\mathbf{F}(s)\, ds$$

La matriz exponencial $e^{\mathbf{A}t}$ es una matriz siempre no singular y $e^{-\mathbf{A}s} = (e^{\mathbf{A}s})^{-1}$. En la práctica se puede obtener $e^{-\mathbf{A}s}$ a partir de $e^{\mathbf{A}t}$ simplemente si reemplazamos *t* por *–s*.

1.2.3- MÉTODOS NUMÉRICOS

Si existe una solución de una ecuación diferencial, ella representa un conjunto de puntos en el plano cartesiano. A través de los métodos numéricos, es posible emplear procedimientos que emplean la ecuación diferencial para obtener una sucesión de puntos distintos cuyas coordenadas se aproximen a las coordenadas de los puntos de la curva de solución real.
Pero una ecuación diferencial no necesita tener una solución, y aún si la tiene, no siempre se puede expresarla en forma explícita o implícita. En estos casos, tendremos que contentarnos con una aproximación que nos brindan los métodos numéricos.
Los procedimientos numéricos para las ecuaciones lineales de primer orden se pueden adaptar para sistemas de ecuaciones de primer orden. A continuación, se expondrá el método de Runge-Kutta que resulta ser simple de aplicar y muy preciso en sus resultados. Se partirá del caso de una ecuación diferencial de primer orden sencilla y luego se ampliará para considerar sistemas.
Tal vez el método más difundido y más exacto para obtener soluciones aproximadas al problema de valor inicial:

$$\frac{dy}{dx} = f(x, y), \qquad y(x_0) = y_0$$

sea el método de Runge-Kutta de cuarto orden . Los distintos métodos de Runge-Kutta se deducen a partir del desarrollo de $y(x_n+h)$ en serie de Taylor con residuo:

$$y(x_{n+1}) = y(x_n + h) = y(x_n) + \frac{dy(x_n)}{dx} + \frac{h^2}{2!}\frac{d^2 y(x_n)}{dx^2} + \ldots + \frac{h^{k+1}}{(k+1)!}\frac{d^{(k+1)} y(c)}{dx^{(k+1)}}$$

Cuando $K=1$ y el residuo $(h^2/2)(dy(c)/dx)$ es pequeño, se obtiene la fórmula de iteración del procedimiento de Runge-Kutta de primer orden (o método básico de Euler):

$$y_{n+1} = y_n + h\frac{dy(x_n)}{dx} = x_{n+1} + hf(x_n, y_n)$$

siendo el error local de truncamiento $O(h^2)$ y el error global de $O(h^1)$.

El procedimiento de Runge-Kutta de segundo orden(o método de Euler mejorado) consiste en hallar las constantes a, b, α y β tales que la fórmula

$$y_{n+1} = y_n + ak_1 + bk_2, \qquad k_1 = hf(x_n, y_n)$$
$$k_2 = hf(x_n + \alpha h, y_n + \beta h)$$

coincida con un polinomio de Taylor de segundo grado. Se puede demostrar que ello es posible siempre y cuando las constantes cumplan con las siguientes condiciones:

$$a + b = 1 \qquad b\alpha = 1/2 \qquad b\beta = 1/2$$

Este es un sistema de tres ecuaciones con cuatro incógnitas, por lo que tiene una cantidad infinita de soluciones. Se ha comprobado que cuando se asume que $a=b=1/2$ y $\alpha=\beta=1$, se obtienen buenos resultados, siendo el error local de truncamiento $o(h^3)$ y el error global de $O(h^2)$.

Nótese que la suma ak_1+bk_2 es un promedio ponderado de k_1 y k_2, ya que $a+b=1$, y que a su vez k_1 y k_2 son múltiplos aproximados de la pendiente de la curva de solución $y(x)$ en dos puntos distintos en el intervalo (x_n, x_{n+1}).

Por último, el procedimiento de Runge-Kutta de cuarto orden consiste en determinar las constantes adecuadas para que la fórmula

$$y_{n+1} = y_n + \frac{1}{6}(ak_1 + bk_2 + ck_3 + dk_4)$$

$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf(x_n + \alpha_1 h, y_n + \beta_1 k_1)$$
$$k_3 = hf(x_n + \alpha_2 h, y_n + \beta_2 k_1 + \beta_3 k_2)$$
$$k_4 = hf(x_n + \alpha_3 h, y_n + \beta_4 k_1 + \beta_5 k_2 + \beta_6 k_3)$$

coincida con un polinomio de Taylor de cuarto grado. En el sistema tenemos 11 ecuaciones con 13 incógnitas. Su error local de truncamiento $O(h^5)$ y su error global de $O(h^4)$.
El conjunto de valores de las constantes que más se usa produce el siguiente resultado:

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_1)$$
$$k_3 = hf(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_2)$$
$$k_4 = hf(x_n + h, y_n + k_3)$$

Este resultado se puede ampliar y utilizar el método de Runge-Kutta para solucionar sistemas de ecuaciones diferenciales de primer orden (lineales o no). Para nuestro caso, deberemos ser capaces de aplicar el método para un sistema de dos ecuaciones diferenciales. Para aplicar el método de Runge-Kutta de cuarto orden a dicho sistema

$$\frac{dy}{dt} = f_1(t, x, y), \qquad x(t_0) = x_0$$
$$\frac{dy}{dt} = f_2(t, x, y), \qquad y(x_0) = y_0$$

deberemos resolver:

$$x_{n+1} = x_n + \frac{1}{6}(m_1 + 2m_2 + 2m_3 + m_4)$$
$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$m_1 = hf_1(t_n, x_n, y_n) \qquad k_1 = hf_2(t_n, x_n, y_n)$$
$$m_2 = hf_1(t_n + \tfrac{1}{2}h, x_n + \tfrac{1}{2}m_1, y_n + \tfrac{1}{2}k_1) \qquad k_2 = hf_2(t_n + \tfrac{1}{2}h, x_n + \tfrac{1}{2}m_1, y_n + \tfrac{1}{2}k_1)$$
$$m_3 = hf_1(t_n + \tfrac{1}{2}h, x_n + \tfrac{1}{2}m_2, y_n + \tfrac{1}{2}k_2) \qquad k_3 = hf_2(t_n + \tfrac{1}{2}h, x_n + \tfrac{1}{2}m_2, y_n + \tfrac{1}{2}k_2)$$
$$m_4 = hf_1(t_n + h, x_n + m_3, y_n + k_3) \qquad k_4 = hf_2(t_n + h, x_n + m_3, y_n + k_3)$$

1.2.4- MÉTODOS ADAPTATIVOS

La exactitud de un método numérico como del de Runge-Kutta puede ser mejorado disminuyendo el tamaño del paso h.- Esta mayor exactitud se obtiene a un costo: mayor tiempo de cálculo y mayores posibilidades de error de redondeo. En general, en el intervalo de aproximación pueden existir subintervalos en que baste un tamaño mayor de paso para mantener el error de truncamiento dentro de cierto límite deseado. Los métodos numéricos que emplean variables de paso se llaman métodos adaptativos y uno

de los más difundidos para aproximar las soluciones de ecuaciones diferenciales es el algoritmo de Runge-Kutta-Fehlberg.

1.2.5- ERRORES EN LOS MÉTODOS NUMÉRICOS

Para tener una idea más clara de cual es el grado de precisión que se obtiene al aplicar esta técnica numérica comparada con la solución analítica, se define el error absoluto como:

$$\text{Error} \quad \text{Absoluto} = \left|\text{valor exacto} - \text{valor aproximado}\right|$$

donde el valor exacto está dado por la solución analítica y el valor aproximado por la solución numérica. De forma similar, puede definirse un error relativo y el error relativo porcentual son, respectivamente:

$$Error \quad \text{Re}\, lativo = \frac{\left|valor\ exacto - valor\ aproximado\right|}{\left|valor\ exacto\right|}$$

$$Error\ \text{Re}\, lativo\ Porcentual = \frac{\left|valor\ exacto - valor\ aproximado\right|}{\left|valor\ exacto\right|} x100$$

Éstas son medidas que se pueden utilizar para comparar las bondades de la aproximación por Runge-Kutta. Para elegir y aplicar un método numérico en la solución de un problema de valores iniciales, debemos estar conscientes de las diversas fuentes de errores. Para algunos problemas, la acumulación de errores podría reducir la exactitud de una aproximación, e incluso provocar que los resultados obtenidos sean considerablemente incorrectos.
Una fuente perpetua de error en los cálculos es el error de redondeo. No importa que calculadora o computadora que se utilice para realizar los cálculos, sólo es posible representar los números con una cantidad finita de dígitos[5]. Cabe aclarar que para este ejercicio se cometen errores de redondeo tanto en la solución numérica como en la solución analítica, ya que este tipo de error depende principalmente del programa computacional que se utilice[6]. Este error es impredecible, difícil de analizar y no será motivo de nuestro análisis.
Al iterar un algoritmo iterativo se obtiene una sucesión de valores $y_1\ y_2\ ...y_n$ y en general dichos valores no coincidirán con el de la solución exacta $y(t_i)$ debido a que el algoritmo solo proporciona una aproximación por polinomio de Taylor de un orden determinado

[5] Sin embargo, desde el advenimiento de los programas computacionales simbólicos (Maple y Mathematica) los problemas de errores de redondeo desaparecen. Esto es así, ya que los mismos trabajan algebraicamente cuando se hayan las soluciones analíticas y mantienen los valores exactos de los pasos de cada algoritmo al acumular en cada iteración la expresión exacta representada por medio de expresiones simbolico-algebraicas mediante una composición de funciones iterativas.
[6] Incluso es posible que el método de la matricial exponencial cometa errores de redondeo mayores, ya que implica el cálculo de una matriz inversa y los mismos pueden ser considerablemente grandes en casos donde el determinante se aproxime a cero. Al calcular los valores y vectores propios por aproximación, dichos errores también pueden ser importantes.

(que para Runge-Kutta de cuarto orden esta dado por 4). La diferencia se conoce como ***error local de truncamiento, error de fórmula o error de discretización***, el cual se produce en cada una de las iteraciones.
Para obtener una formula para el error local de truncamiento, partimos de una serie de Taylor con residuo:

$$y(x_{n+1}) = y(x_n + h) = y(x_n) + \frac{dy(x_n)}{dx} + \frac{h^2}{2!}\frac{d^2y(x_n)}{dx^2} + \ldots + \frac{h^{k+1}}{(k+1)!}\frac{d^{(k+1)}y(c)}{dx^{(k+1)}}$$

Las aproximaciones de Runge-Kutta de órdenes 1, 2 y 4 utilizan los 2, 3 y 5 primeros términos respectivamente. El resto de los elementos del polinomio corresponden al error local de truncamiento. Se suele toma como referencia de dicho error al más importante de los términos no tenidos en cuenta, despreciando el resto de los términos. Los errores de truncamiento para los métodos de Runge-Kutta de órdenes 1, 2 y 4 son:

$$O(h^2) = \frac{h^2}{2!}\frac{d^2y(x_n)}{dx^2}, \qquad O(h^3) = \frac{h^3}{3!}\frac{d^3y(x_n)}{dx^3}, \qquad O(h^5) = \frac{h^5}{5!}\frac{d^5y(x_n)}{dx^5}$$

Al describir los errores originados en el empleo de métodos numéricos se utilizó la notación $O(h^n)$. Si denominamos $e(h)$ al error en un cálculo numérico que dependa de h, entonces se dice que $e(h)$ es de orden h^n (lo que se representa como $O(h^n)$) si existe una constante C y un entero positivo n tales que $|e(h)|<Ch^n$ cuando h es lo suficientemente pequeña. En general, si un método numérico tiene un orden h^n y h se reduce a la mitad, el nuevo error es, aproximadamente $C(h/2)^n=C(h/2)^n$, o sea, se reduce en un factor de $1/2^n$.
En el análisis del error local de truncamiento se supone o se parte de que el valor estimado de y_n (y los otros rezagos que incluye el método) conque se calcula y_{n+1} es exacto; pero no lo es, ya que posee errores de truncamiento locales debido a los pasos anteriores. En y_{n+1} el error total cometido se denomina ***error global de truncamiento***.
En general, se puede demostrar que "***si un método de solución numérica de una ecuación diferencial tiene un error local de truncamiento $O(h^{n+1})$, el error global de truncamiento es $O(h^n)$***".

Estabilidad de los métodos numéricos:
Un aspecto importante del uso de métodos numéricos para aproximar la solución de un problema de valor inicial es la estabilidad de los mismos. En términos sencillos, un método numérico es estable si cambios pequeños en la condición inicial solo generan pequeñas modificaciones en la solución calculada. Se dice que un método numérico es inestable si no es estable. La importancia de la estabilidad radica en que en cada paso subsecuente de una técnica numérica, en realidad se comienza de nuevo con un nuevo problema de valor inicial en que la condición inicial es el valor aproximado de la solución calculado en la etapa anterior. Debido a la presencia de errores de redondeo, caso con seguridad este valor varía respecto del valor real de la solución, cuando menos un poco. Además del error de redondeo, otra fuente común de error se presenta en la condición inicial misma; con frecuencia, en las aplicaciones físicas los datos se obtienen con mediciones imprecisas.
Un posible método para detectar la inestabilidad de la solución numérica de cierto problema de valor inicial, es comparar las soluciones aproximadas que se obtienen al disminuir los tamaños de etapa utilizados. Si el método numérico es inestable, el error

puede aumentar con tamaños menores del paso. Otro modo de comprobar la estabilidad es observar qué sucede a las soluciones cuando se perturba ligeramente la condición inicial.

1.2.6- FUENTES DE ERRORES Y COMPORTAMIENTO CAÓTICO

Para resumir puede decirse que existen tres fuentes de errores que pueden ser capaces de ocasionar comportamientos inestables en las funciones halladas. Estas causas pueden provenir de:

- Errores inherentes al método numérico en si mismo tales como los errores locales y globales
- Errores de redondeo en cada paso debido a las calculadoras o programas informáticos que utilizan punto flotante. Esta fuente de error puede eliminarse trabajando con programas simbólicos tales como Mathematica o Maple
- Errores de medición en la condición inicial

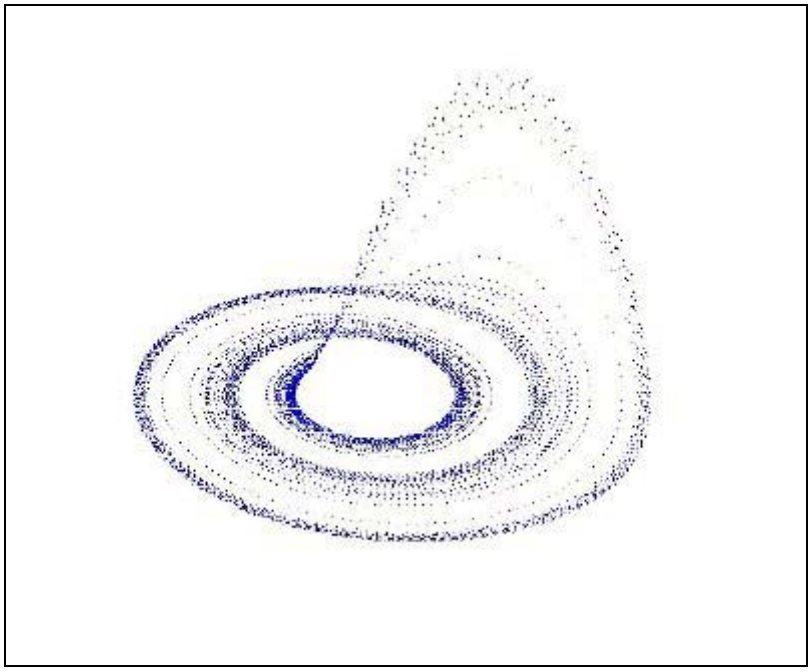
Estas consideraciones son validas en el sentido de que estas tres fuentes de errores pueden permitir la introducción de comportamientos caóticos. A modo de ejemplo, se presenta a continuación un sistema de ecuaciones diferenciales no lineales junto a su representación gráfica mediante el Software Matlab 5.3.
Una de las construcciones geométricas más sencillas de CAOS en sistemas contínuos es el sistema de ecuaciones diferenciales de **Rössler:**

$$x' = -(y + z)$$
$$y' = x + ay$$
$$z' = b + xz - cz$$

donde *a,b* y *c* son constantes ajustables.
Si se grafican los valores en el plano ***x,y,z*** paramétricamente con el tiempo, se obtiene una gráfica, en la que se puede observar que el sistema no converge ni a un punto fijo, ni a un ciclo, sino sigue "dando vueltas" alrededor del atractor.

```
clear all
%DECLARACION DE CONSTANTES
a=.2;b=.2;c=5.7;
%PASO DE TIEMPO PARA INTEGRACION
dt=.04;dt2=dt/2;
%CONDICIONES INICIALES
xn=0;yn=0;zn=0;
for i=1:10000
   xfn=-yn-zn;
   yfn=xn+a*yn;
   zfn=b+xn*zn-c*zn;
   xn0=xn;yn0=yn;zn0=zn;
   xfn0=xfn;yfn0=yfn;zfn0=zfn;
   %trapezoide
   xn=xn0+dt*xfn0;
   yn=yn0+dt*yfn0;
   zn=zn0+dt*zfn0;
   xfn=-yn-zn;
   yfn=xn+a*yn;
   zfn=b+xn*zn-c*zn;
```

```
    %extrapolación de Richardson
    xn=xn0+dt2*(xfn+xfn0);
    yn=yn0+dt2*(yfn+yfn0);
    zn=zn0+dt2*(zfn+zfn0);
         if i==1
    plot3([xn0 xn],[yn0 yn],[zn0 zn])
       hold
              end
    plot3([xn0 xn],[yn0 yn],[zn0 zn])
end
```



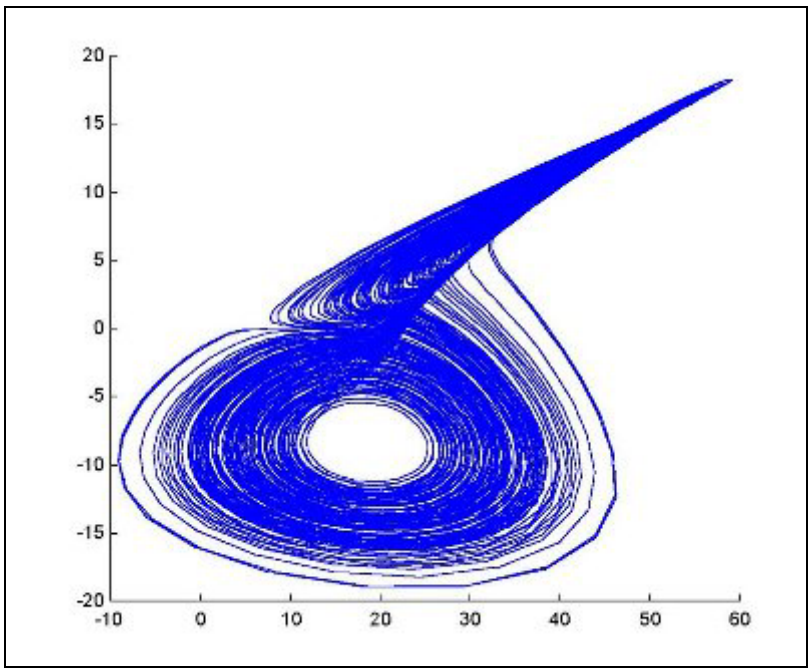
Otro sistema de ecuaciones diferenciales que presenta CAOS es el de **Lorenz**, el cual es una simplificación de las ecuaciones originales que encontró en la predicción del tiempo.

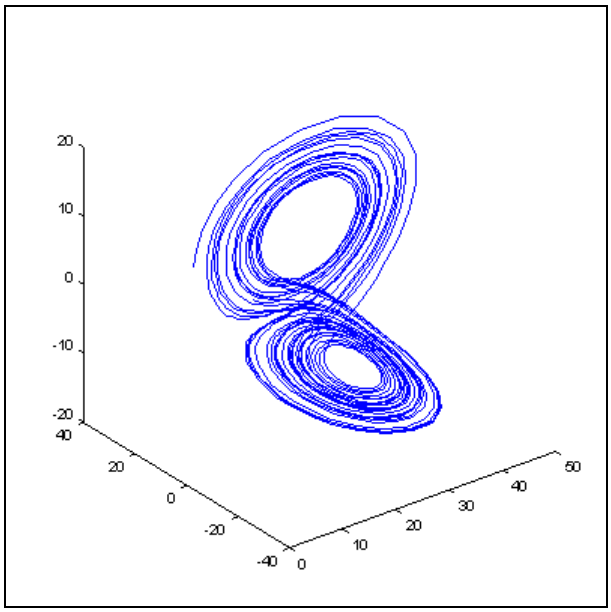
$$x' = 10(y - x)$$
$$y' = x(28 - z) - y$$
$$z' = xy - 8z/3$$

Animación de la posición (*x,y,z*) con el paso del tiempo

```
% Calcula un atractor extraño
% Resolviendo el sistema de ecuaciones diferenciales por el método de
integración ...
% trapezoidal, con extrapolación de Richarson
clear all          %limpia la memoria

%   CONSTANTES
a=10;
b=2.667;
c=28;
```

```
%PASO DE TIEMPO
dt=.02;
dt2=dt/2;
%ESTO LE DA AL SISTEMA TIEMPO PARA ACERCARSE AL ATRACTOR
temp=1000;
%CONDICIONES INICIALES (SE PUEDEN CAMBIAR A VALORES CERCANOS)
xn=0.1;yn=1;zn=0.1;

for i=1:2500
   zfn=a*(yn-zn);
   yfn=c*zn-yn-zn*xn;
   xfn=-b*xn+zn*yn;
   xn0=xn;yn0=yn;zn0=zn;
   xfn0=xfn;yfn0=yfn;zfn0=zfn;
   %trapezoide
   xn=xn0+dt*xfn0;
   yn=yn0+dt*yfn0;
   zn=zn0+dt*zfn0;
   zfn=a*(yn-zn);
   yfn=c*zn-yn-zn*xn;
   xfn=-b*xn+zn*yn;
   %extrapolación de Richardson
   xn=xn0+dt2*(xfn+xfn0);
   yn=yn0+dt2*(yfn+yfn0);
   zn=zn0+dt2*(zfn+zfn0);
   if i==temp
        plot3([xn0 xn],[yn0 yn],[zn0 zn])
        hold
    end
   if i>=temp
      plot3([xn0 xn],[yn0 yn],[zn0 zn])

   end
end
```





1.2.7- APLICACIONES DE LOS SISTEMAS DE ECUACIONES DIFERENCIALES

Una sola ecuación diferencial puede describir una población en un ambiente, pero si nos interesa estudiar a dos poblaciones en un medio, el modelo que utilicemos debe tener en cuenta que ambas especies interactúan y compiten en el mismo ambiente. El modelo demográfico de sus poblaciones x(t) t y(t) podría se un sistema de dos ecuaciones diferenciales de primer orden, como:

$$\frac{dx}{dt} = g_1(t, x, y)$$
$$\frac{dy}{dt} = g_2(t, x, y)$$

A continuación, se diferencian tres casos de este tipo de modelos, según cual es el tipo de relación que existe entre las dos especies.
Las aplicaciones económicas de este tipo de modelos puede incluir el modelado de mercados de productos de bienes complementarios o sustitutos entre sí, el efecto de la presencia de firmas transnacionales dentro de economías latinoamericanas en la tasa de crecimiento de las empresas locales y las ventajas y desventajas dinámicas[7] de la integración económica entre países.

MODELO LOTKA-VOLTERRA: DEPREDADOR PRESA

[7] En el análisis de la integración económica solo suele tratarse las ventajas y desventajas estáticas (creación del comercio y desviación del comercio) de la formación de acuerdos preferenciales de comercio, áreas de libre comercio y uniones monetarias. El estudio de las cuestiones dinámicas para el caso de los países latinoamericanos (Mercosur, Alca, etc.) es una materia pendiente.

Supongamos que dos especies animales interactúan en el mismo ambiente o ecosistema: la primera se alimenta de vegetación y la segunda se alimenta de la primera. En otras palabras, una especie es depredador ($x(t)$) y la otra es presa($y(t)$).
Si no hubiera presas (estuviera extinta) cabría de esperar que los depredadores disminuyeran en número al carecer de alimento. Utilizando el modelo de crecimiento exponencial, se tendrá:

$$\frac{dx}{dt} = -ax, \qquad a > 0$$

Cuando hay presas en el sistema, supondremos que la cantidad de encuentros o iteracciones por unidad de tiempo entre ambas especies es proporcional de forma simultánea a ambas poblaciones (o sea, es proporcional al producto xy, nuevamente siguiendo un comportamiento parecido al descripto por un modelo de crecimiento exponencial. Cuando hay presas, hay alimento para los depredadores y éstos crecen a una tasa $bxy>0$. Por lo tanto, para tener una descripción completa del crecimiento de los depredadores, debemos sumar los dos efectos anteriores. Se obtiene un modelo demográfico para los depredadores descripto por:

$$\frac{dx}{dt} = -ax + bxy, \qquad a > 0 \quad y \quad b > 0$$

Para las presas, cuando no hay depredadores, supondremos que las reservas de alimentos son ilimitadas y que el modelo de crecimiento exponencial sería adecuado para describir su modelo, por lo que las presas crecerían con una rapidez proporcional a su población en el momento t:

$$\frac{dy}{dt} = hy, \qquad h > 0$$

Pero cuando hay depredadores, el modelo demográfico para las presas debe incluir el efecto que causa en su población la caza por parte de los depredadores. Nuevamente supondremos que la cantidad de interacciones entre ambas especies es proporcional simultáneamente a sus poblaciones, pero esta vez de forma negativa. Cuando hay depredadores, las presas decrecen a una tasa de $(-kxy)<0$. Sumando ambos efectos en el modelo demográfico de las presas, tenemos:

$$\frac{dy}{dt} = hy - kxy, \qquad h > 0 \quad y \quad k > 0$$

La ecuación última y la antepenúltima conforman un sistema de ecuaciones diferenciales no lineales, el cual suele denominársele el modelo depredador-presa de Lotka-Volterra. El sistema no lineal no puede resolverse en término de funciones elementales, sino que se debe recurrirse a métodos numéricos para resolver ecuaciones diferenciales ordinarias.

Modelo de Competencia

Ahora consideremos que hay dos especies animales distintas que ocupan el mismo ecosistema, pero que la relación entre ambas especies no es ya depredador y presa, sino como competidores en el uso del mismo recurso (como alimento o espacio vital). Cuando falta una de las especies, supongamos que su crecimiento demográfico se encuentra bien descrito por el modelo de crecimiento exponencial, por lo tanto:

$$\frac{dx}{dt} = ax, \qquad a > 0 \qquad\qquad \frac{dy}{dt} = hy, \qquad h > 0$$

Como las dos especies compiten, la presencia de ambas provoca que sus poblaciones disminuyan por la influencia de la otra especie. Suponiendo nuevamente una proporcionalidad simultánea a sus dos poblaciones, los modelos demográficos para las dos especies queda definido como:

$$\frac{dx}{dt} = ax - bxy, \qquad a > 0 \quad y \quad b > 0$$
$$\frac{dy}{dt} = hy - kxy, \qquad h > 0 \quad y \quad k > 0$$

Nuevamente obtenemos un sistema no lineal. Este sistema no lineal se parece al de la sección anterior, pero se diferencia claramente en la relación entre las dos especies.

Modelo de Cooperación de Especies: Simbiosis

En este caso, las especies se asocian, viven juntas y se favorecen mútuamente en su desarrollo. Por lo tanto, la existencia de la otra especie provoca un efecto positivo en la tasa de crecimiento de ambas. Supondremos que dicho efecto positivo provoca un crecimiento positivo proporcional a las poblaciones de ambas especies de forma simultánea.
A su vez, ante la ausencia de la otra especie, el crecimiento es negativo a una tasa constante. Por lo tanto, el modelo queda especificado como:

$$\frac{dx}{dt} = -ax + bxy, \qquad a > 0 \quad y \quad b > 0$$
$$\frac{dy}{dt} = -hy + kxy, \qquad h > 0 \quad y \quad k > 0$$

1.2.8- EXTENSIONES

Los modelos de crecimiento exponencial pueden ser adaptados para que las poblaciones aisladas no crezcan según el modelo exponencial, sino que las tasas reflejen que cada población crece en forma logística (la población permanece acotada).
Para el caso del modelo depredador-presa, tendríamos:

$$\frac{dx}{dt} = -a_{11}x + a_{12}x^2 + b_1xy = x(-a_{11} + a_{12}x + b_1y)$$
$$\frac{dy}{dt} = h_{11}y - h_{12}y^2 - k_1xy = y(h_{11} - h_{12}y - k_1x)$$

Para el caso del modelo de competencia:

$$\frac{dx}{dt} = a_{21}x - a_{22}x^2 - b_2xy = x(a_{211} - a_{22}x - b_2y)$$
$$\frac{dy}{dt} = h_{21}y - h_{22}y^2 - k_2xy = y(a_{211} - a_{22}y - b_2x)$$

Por último, podría modelarse un nuevo sistema que tuviera en cuenta la existencia de tres especies (como dos depredadores y una presa) pero se omitirá esta clase de modelo ya que el análisis de los dos anteriores es suficiente para estudiar las relaciones existentes entre ellos y no agrega nada nuevo a consideración.

2- RESOLUCIÓN DEL EJERCICIO

El ejercicio establece que resolvamos un modelo depredador-presa a través del método de la matriz exponencial (resolución de forma analítica) como del método numérico de Runge-Kutta (resolución de forma numérica) para poder comparar los resultados. Sin embargo, a raíz de que el método de la matriz exponencial solo puede aplicarse a sistemas de ecuaciones diferenciales lineales, el crecimiento poblacional no se comportará de manera proporcional a ambas poblaciones para resolverlo por este método exacto. En consecuencia no queda mas remedio que recurrir a tratar un modelo depredador-presa lineal con coeficientes constantes a los efectos de contar con la solución exacta.
En el caso de la especie presa (*y(t)*), se supondrá que su población disminuye si la población de la especie depredadora (*x(t)*) aumenta (efecto cruzado negativo), mientras que la especie depredadora incrementa su población cuando las presas son más abundantes. También se supondrá que el crecimiento relativo de ambas especies depende de forma positiva de su población actual. El sistema general de ecuaciones que gobierna el crecimiento relativo de las dos especies será:

$$\frac{dx}{dt} = ax + by, \qquad x(t=0) = x_0$$
$$\frac{dy}{dt} = -kx + hy, \qquad y(t=0) = y_0$$

siendo todos los parámetros (*a, b, h y k*) positivos.

El sistema de ecuaciones particular (junto con los valores iniciales) que se utilizará será[8]:

$$\frac{dx}{dt} = 4x + 2y$$

$$\frac{dy}{dt} = -x + y$$

donde x(t) representa la población de la especie depredadora y y(t) la población de las presas, siendo las condiciones iniciales x_0=100 y y_0=400. Su forma matricial **X'=AX** está dada por:

$$\begin{pmatrix} dx/dt \\ dx/dt \end{pmatrix} = \begin{pmatrix} 4 & 2 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Solución Analítica:

Solución Manual:

Para la solución analítica de este sistema, debemos encontrar la matriz exponencial $e^{\mathbf{A}t}$. La matriz de coeficientes **A** posee dos valores propios reales y distintos, las cuales se obtienen al resolver el polinomio característico de dicha matriz (*det(***A**-λ**I**))

$$\left|A - \lambda I\right| = (4 - \lambda)(1 - \lambda) + 2$$

Las raíces λ_1 y λ_2 de dicho polinomio (valores propios de **A**) son 2 y 3. Para obtener los vectores propios correspondientes a cada valor propio, debemos resolver:

$$\left[A - \lambda_i I\right] v = 0 \quad \Rightarrow \quad \begin{pmatrix} 4 - \lambda_i & 2 \\ -1 & 1 - \lambda_i \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

donde i=1, 2. Los vectores propios obtenidos son:

$$v_1 = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \qquad v_2 = \begin{pmatrix} -2 \\ 1 \end{pmatrix}$$

donde v_1 se corresponde con λ_1 y v_2 con λ_2 respectivamente. Por lo tanto, conocemos como son **J** y **C** y estamos en condiciones de obtener $e^{\mathbf{A}t}$:

$$\mathbf{J} = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix} \Rightarrow e^{\mathbf{J}t} = \begin{pmatrix} e^{2t} & 0 \\ 0 & e^{3t} \end{pmatrix} \qquad \mathbf{C} = \begin{pmatrix} -1 & -2 \\ 1 & 1 \end{pmatrix} \Rightarrow \mathbf{C^{-1}} = \begin{pmatrix} 1 & 2 \\ -1 & -1 \end{pmatrix}$$

[8] La especificación de este ejercicio coincide con la presentada por Paulo Regis en su trabajo "Ejercicios complementarios" a efectos de comparar resultados. Adelantándome, destaco que las soluciones a las que arribaré se apartan en algunos casos considerablemente.

$$e^{At} = \mathbf{C}e^{\mathbf{J}t}\mathbf{C}^{-1} = \begin{pmatrix} -1 & -2 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} e^{2t} & 0 \\ 0 & e^{3t} \end{pmatrix} \begin{pmatrix} 1 & 2 \\ -1 & -1 \end{pmatrix} = \begin{pmatrix} (-e^{2t} + 2e^{3t}) & (-2e^{2t} + 2e^{3t}) \\ (e^{2t} - e^{3t}) & (2e^{2t} - e^{3t}) \end{pmatrix}$$

Por lo tanto, la solución analítica del sistema está dada por:

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = e^{\mathbf{A}t} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} (-e^{2t} + 2e^{3t}) & (-2e^{2t} + 2e^{3t}) \\ (e^{2t} - e^{3t}) & (2e^{2t} - e^{3t}) \end{pmatrix} \begin{pmatrix} 100 \\ 400 \end{pmatrix}$$
$$= \begin{pmatrix} -900e^{2t} + 1000e^{3t} \\ 900e^{2t} - 500e^{3t} \end{pmatrix}$$

Solución Numérica:

El algoritmo numérico por el método de Runge-Kutta (de cuarto orden) está dada por la siguiente formulación

$$x_{n+1} = x_n + \frac{1}{6}(m_1 + 2m_2 + 2m_3 + m_4)$$
$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$m_1 = hf_1(t_n, x_n, y_n) \qquad k_1 = hf_2(t_n, x_n, y_n)$$
$$m_2 = hf_1(t_n + \tfrac{1}{2}h, x_n + \tfrac{1}{2}m_1, y_n + \tfrac{1}{2}k_1) \qquad k_2 = hf_2(t_n + \tfrac{1}{2}h, x_n + \tfrac{1}{2}m_1, y_n + \tfrac{1}{2}k_1)$$
$$m_3 = hf_1(t_n + \tfrac{1}{2}h, x_n + \tfrac{1}{2}m_2, y_n + \tfrac{1}{2}k_2) \qquad k_3 = hf_2(t_n + \tfrac{1}{2}h, x_n + \tfrac{1}{2}m_2, y_n + \tfrac{1}{2}k_2)$$
$$m_4 = hf_1(t_n + h, x_n + m_3, y_n + k_3) \qquad k_4 = hf_2(t_n + h, x_n + m_3, y_n + k_3)$$

donde f_1 y f_2 están dadas por:

$$f_1 = 4x + 2y$$
$$f_2 = -x + y$$

y h es una variable de longitud de paso. Mientras más pequeña sea esta longitud de paso más exactas serán las soluciones numéricas acercándose cada vez más a las analíticas, pero mayor será el volumen de cálculos y mayores posibilidades de errores de redondeo, aunque en general las ganancias de precisión por un menor h superan con creces a las pérdidas por redondeo. En este caso, se optó por $h = 0{,}025$.

Implementación del Algoritmo en diversos Software

A continuación se efectuará un análisis exhaustivo en cuanto a las distintas maneras de implementar un algoritmo numérico y algebraico para la resolucion de sistemas de ecuaciones diferenciales. En principio se utilizaran los siguientes Software:

- **Gauss Light 4.0**
- **Matlab 5.3**

- **Mathematica 4.0**
- **Maple 6**
- **Excel 2002**

- **A su vez dentro de cada uno de ellos se exploraran las diversas herramientas enlatadas para la resolución y graficación de tales problemas como así también las diversas formas de programación que permiten los distintos lenguajes.**

- GAUSS LIGHT 4.0

A raíz de que este programa no cuenta con métodos enlatados predeterminados para la solución de este tipo de problemas se crearon 2 procedimientos: exact y rungekutta, donde el primero proporciona la solución analítica del sistema y el segundo la solución numérica.
La matriz A de coeficientes es un dato y el programa le solicita al operador que introduzca las poblaciones iniciales de ambas especies y el momento tk en que el operador desea estimar las poblaciones de ambas especies.
El programa utilizado se describe a continuación:

```
/*Resolución de un sistema lineal, un modelo depredador-presa "lineal"
 (SEGUN GROSSMAN) para valores propios reales.
 La variable x es la población del depredador e y la población de la
presa.*/


Cls;
Format 7,5;
/*Procedimiento para obtener la solución analítica a través de la
matriz exponencial*/


proc(1)=exact(t,n,V,C,X0);
    local Jexp, Aexp, solucion, g, gg;
    if V[1,1]==V[2,1];
        g=exp(t*V[1,1]); Jexp=zeros(2,2);
        Jexp[1,1]=g; Jexp[1,2]=t*g; Jexp[2,2]=g;
    else;
        Jexp=diagrv(zeros(n,n),exp(t*V));
    endif;
    Aexp=C*Jexp*inv(C);              /*Matriz Exponencial exp(At)*/
    solucion=Aexp*X0;
    retp(solucion);
endp;
```

```
/*Procedimiento para obtener la solución numérica a través del método
de Runge-Kutta de cuarto orden*/

proc(1)=rungekutta(t,h,&fx,&fy,Xant);
    local fx:proc, fy:proc;
    local m1, m2, m3, m4, k1, k2, k3, k4, xn, yn, solucion;
    xn=Xant[1,1]; yn=Xant[2,1];
    m1=h*fx(t,xn,yn);
```

```
    k1=h*fy(t,xn,yn);
    m2=h*fx(t+h/2,xn+m1/2,yn+k1/2);
    k2=h*fy(t+h/2,xn+m1/2,yn+k1/2);
    m3=h*fx(t+h/2,xn+m2/2,yn+k2/2);
    k3=h*fy(t+h/2,xn+m2/2,yn+k2/2);
    m4=h*fx(t+h,xn+m3,yn+k3);
    k4=h*fy(t+h,xn+m3,yn+k3);
    xn=xn+(1/6)*(m1+2*m2+2*m3+m4);
    yn=yn+(1/6)*(k1+2*k2+2*k3+k4);
    solucion=xn|yn;
    retp(solucion);
endp;
```

```
A={4 2,-1 1};                    /*Matriz de Coeficientes*/
{valp,vecp}=eigv(A);             /*Valores y Vectores propios*/


"Sistema a estimar";
"x' =    " a[1,1] "x  +    " a[1,2] "y";
"y' =    " a[2,1] "x  +    " a[2,2] "y";
?; "Ingresar los valores iniciales";
"Población inicial de depredadores:";
x0=con(1,1);
"Población inicial de presas:";
y0=con(1,1);
"Período tk a estimar las poblaciones:";
tk=con(1,1);
?; "          Solución Analítica       Solución numérica
Errores Absolutos                 Errores Relativos";
 "Período   Depredador   Presa   Depredador   Presa    Depredador
Presa   Depredador    Presa";

fn f1(t,x,y)=A[1,1]*x+A[1,2]*y;     /*Ecuación Diferencial
Depredador*/
fn f2(t,x,y)=A[2,1]*x+A[2,2]*y;     /*Ecuación Diferencial Presa*/
xanal=x0;
xnum=x0;
yanal=y0;
ynum=y0;
h=0.025;
t=0;
tt=t;
{inic}=exact(t,rows(A),valp,vecp,x0|y0);
Xant=x0|y0;

do until t>tk;      /*Cuadro de 9 columnas*/
    t=t+h;
    tt=tt|t;
    {sol1}=exact(t,rows(A),valp,vecp,inic);
    {sol2}=rungekutta(t,h,&f1,&f2,Xant);
    t~sol1'~sol2'~abs(sol1-sol2)'~(abs(sol1-sol2)./sol1)';
    xanal=xanal|sol1[1,1];
    xnum=xnum|sol2[1,1];
    yanal=yanal|sol1[2,1];
    ynum=ynum|sol2[2,1];
    Xant=sol2;
endo;
```

```
t=0; h=0.000001;
yn=x0;              /*Extinción de la presa*/
do until yn<0;
    {sol}=exact(t,rows(A),valp,vecp,x0|y0);
    t=t+h;
    yn=sol[2,1];
endo;
?; "Período en que la población de la presa se extingue:" t;
output off;
```

La generalidad del programa es casi total en el sentido que permite cambiar las funciones a resolver (pudiendo para el caso de rungue-kuta ser no lineales), la longitud de paso, el tiempo inicial, el tiempo final, las condiciones iniciales y el número de iteraciones. Con respecto a la solución analítica se destaca que ante la inexistencia de un procedimiento que genere matrices exponenciales se vió en la necesidad de crearlas vía el lenguaje de programación. Esta programación no es del todo general en el sentido de que es incapaz de halar la verdadera matriz exponencial en el caso de raíces repetidas y con multiplicidad algebraica mayor que la geométrica como así también en el caso de valores propios complejos aunque extenderla para esos casos no es difícil, en este trabajo dicha extensión no se efectúa[9].
Tomando una longitud de paso (h) de 0.025, el programa proporciona 9 columnas, donde se registran los cálculos realizados para cada paso. La primera establece el período (para ser más exactos, el subperíodo) a que corresponden las sucesivas iteraciones del procedimiento rungekutta y, de forma auxiliar para fines de comparación, el mismo cálculo pero por medio del procedimiento exact.
Las columnas 2 y 3 corresponden a las resoluciones analíticas para la población en el momento t de la especie depredador y presa respectivamente, mientras que en las columnas 4 y 5 se encuentran las estimaciones por Runge-Kutta para el mismo momento t de los depredadores y las presas respectivamente. Las cuatro columnas siguientes corresponden a medidas de los errores cometidos por el método Runge-Kutta, en comparación con la resolución analítica (columnas 2 y 3). En las columnas 6 y 8 se registran los errores absolutos y relativos respectivamente para la especie depredador y en las columnas 7 y 9 tenemos los mismos conceptos para la especie presa.
Cabe aclarar que las soluciones analíticas provistas por GAUSS no son exactas, ya que el programa utiliza distintos procedimientos numéricos para el cálculo de varios conceptos, como en el caso de la inversa y valores y vectores propios.

Por último, el programa calculará a través del método de la exponencial matricial el momento en que la especie presa se extingue ($y(t)=0$). Es necesario tener presente este dato, ya que no sería realista proponer un modelo de crecimiento poblacional donde una de las especies asumiera valores negativos.

Los resultados obtenidos por el programa se presentan a continuación:

```
Sistema a estimar
x' =     4.0000 x  +     2.0000 y
y' =    -1.0000 x  +     1.0000 y
```

[9] Para una extensión al caso de multiplicidad algebraica mayor que la geométrica véase Regis, Paulo José anteriormente citado.

```
Ingresar los valores iniciales
Población inicial de depredadores:
? 100
Población inicial de presas:
? 400
Período tk a estimar las poblaciones:
? 0.5
```

Solución Analítica		Solución numérica		Errores Absolutos		Errores Relativos	
Perío	Depre	Presa	Depre	Presa	Depredado	Presa	Depredado	Presa
0.0250	131.74	407.20	131.74	407.20	1.7662e-005	7.6492e-006	1.3407e-007	1.8785e-008
0.0500	167.18	413.74	167.18	413.74	3.8201e-005	1.6616e-005	2.2850e-007	4.0160e-008
0.0750	206.67	419.49	206.67	419.49	6.1962e-005	2.7063e-005	2.9981e-007	6.4514e-008
0.100	250.60	424.33	250.60	424.33	8.9329e-005	3.9173e-005	3.5646e-007	9.2316e-008
0.1250	299.37	428.13	299.37	428.13	0.00012072	5.3145e-005	4.0326e-007	1.2413e-007
0.1500	353.44	430.72	353.44	430.72	0.00015661	6.9202e-005	4.4311e-007	1.6067e-007
0.1750	413.30	431.93	413.30	431.93	0.00019751	8.7588e-005	4.7789e-007	2.0278e-007
0.2000	479.48	431.58	479.48	431.58	0.00024398	0.00010858	5.0886e-007	2.5158e-007
0.2250	552.55	429.46	552.55	429.46	0.00029666	0.00013247	5.3690e-007	3.0844e-007
0.2500	633.15	425.35	633.15	425.35	0.00035624	0.00015959	5.6264e-007	3.7519e-007
0.2750	721.95	18.99	721.95	418.99	0.00042347	0.00019030	5.8656e-007	4.5419e-007
0.3000	819.70	410.11	819.70	410.11	0.00049919	0.00022501	6.0899e-007	5.4868e-007
0.3250	927.18	398.40	927.18	398.40	0.00058432	0.00026417	6.3021e-007	6.6307e-007
0.3500	1045.30	383.55	1045.30	383.55	0.00067988	0.00030825	6.5044e-007	8.0367e-007
0.3750	1174.9	365.19	1174.9	365.19	0.00078699	0.00035780	6.6982e-007	9.7976e-007
0.4000	1317.1	342.93	1317.1	342.93	0.00090686	0.00041340	6.8851e-007	1.2055e-006
0.4250	1473.0	316.33	1473.0	316.33	0.0010408	0.00047571	7.0661e-007	1.5038e-006
0.4500	1643.8	284.93	1643.8	284.93	0.0011904	0.00054545	7.2420e-007	1.9143e-006
0.4750	1830.7	248.21	1830.7	248.21	0.0013572	0.00062339	7.4136e-007	2.5115e-006
0.5000	2035.2	205.61	2035.2	205.61	0.0015430	0.00071040	7.5815e-007	3.4551e-006

```
Período en que la población de la presa se extingue:0.58779
```

Como vemos, los resultados analíticos y numéricos son muy parecidos (idénticos en el formato de redondeo presentado en las columnas 2, 3, 4 y5), siendo los errores absolutos y relativos en todos los casos muy pequeños (del orden de 10^{-7} y 10^{-6} o menos, respectivamente). Los resultados obtenidos indican que las estimaciones de Runge-Kutta en GAUSS son tan buenas como las soluciones por la matriz exponencial proporcionadas por el mismo programa.

Como conclusión de esta sección, podemos decir que el método de Runge-Kutta ha demostrado poseer una gran precisión, siendo sus resultados casi idénticos a los obtenidos de forma analítica.

Por lo tanto, los métodos numéricos no solo muestran una performance muy buena en cuanto a la exactitud de las soluciones encontradas (para una variable de paso *h* aceptable), sino que son más generales y permiten aplicar programas más sencillos y rápidos para la resolución e cualquier sistema similar a 8.3.
Aún así, la mayor ventaja de los métodos numéricos que se obtiene de su generalidad no es la resolución de problemas lineales ya que casi no tiene sentido aproximar una solución si se cuenta con un medio de hallarla en forma exacta (aquí se lo hizo solo a los efectos de comparar la precisión del método). Sin embargo ante sistemas no lineales, no existe la posibilidad de contar con una solución analítica y por lo tanto se debe recurrir forzosamente a un método numérico.

MATHEMATICA 4.0

En esta sección se expondrá la versatilidad de un programa simbólico en comparación con un programa numérico. Las ganancias vienen por el lado de la exactitud de sus cálculos al eliminar problemas de redondeo mientras que las pérdidas provienen por el mayor insumo de tiempo que demanda la resolución de problemas.
En el caso de Mathematica, se cuentan con dos formas analíticas para resolver un problema de ecuaciones diferenciales en forma exacta en la medida que tales soluciones analíticas existan. Vales destacar que al no ser un programa numérico las soluciones a las que arriba son totalmente algebraico-simbólicas conservando por ende total exactitud.
Se cuentan con comandos esenciales para estos objetivos:

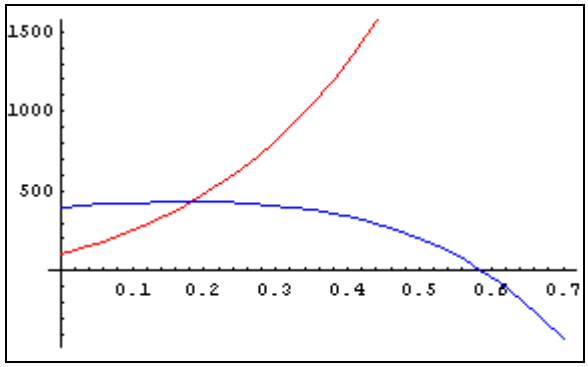
- "`MatrixExp`" que permite la obtención exacta y simbólica de una matriz exponencial de cualquier tipo y tamaño sin presentar problemas de valores propios complejos o repetidos y con multiplicidad algebraica mayor que la geométrica. Como se mencionó anteriormente el método de la matriz exponencial solo sirve para sistemas lineales.
- "`Dsolve`" Que permite resolver ecuaciones diferenciales en forma simbólica sean lineales o no siempre y cuando exista una vía analítica para resolverla. Es decir si existe algún medio Matemático para resolverla en forma analítica y exacta Mathematica 4.0 o hará mediante este comando. El mismo es también aplicables a sistemas de ecuaciones diferenciales**.**

A continuación se exponen las programaciones usando estos dos poderosos comandos. Se incluye también su representación grafica.

```
<< MatrixExp1`
```

```
A = {{4, 2}, {-1, 1}};
K = {{100}, {400}};
S = Eigenvalues[A];
T = Collect[Expand[MatrixExp[A t].K],
  Table[Exp[S[[i]] t], {i, 1, Length[A]}]];
Collect[Expand[MatrixExp[A t].K],
  Table[Exp[S[[i]] t], {i, 1, Length[A]}]] //
 MatrixForm
Plot[{T[[1, 1]], T[[2, 1]]}, {t, 0, 0.7},
 PlotStyle → {RGBColor[1, 0, 0],
   RGBColor[0, 0, 1]}]
```

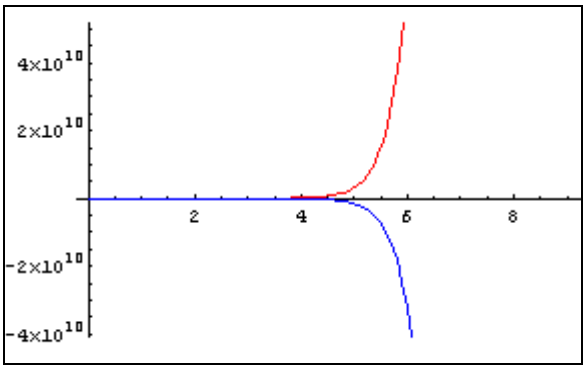
$$\begin{pmatrix} -900\,e^{2t} + 1000\,e^{3t} \\ 900\,e^{2t} - 500\,e^{3t} \end{pmatrix}$$



- Graphics -

```
T = DSolve[{y'[x] == 4 y[x] + 2 z[x],
    z'[x] == -y[x] + z[x], y[0] == 100,
    z[0] == 400}, {y[x], z[x]}, x];
T = {y[x], z[x]} /. T;
DSolve[{y'[x] == 4 y[x] + 2 z[x],
     z'[x] == -y[x] + z[x], y[0] == 100,
     z[0] == 400}, {y[x], z[x]}, x] //
  Transpose // MatrixForm
Plot[{T[[1, 1]], T[[1, 2]]}, {x, 0, 9},
 PlotStyle → {RGBColor[1, 0, 0],
   RGBColor[0, 0, 1]}]
```

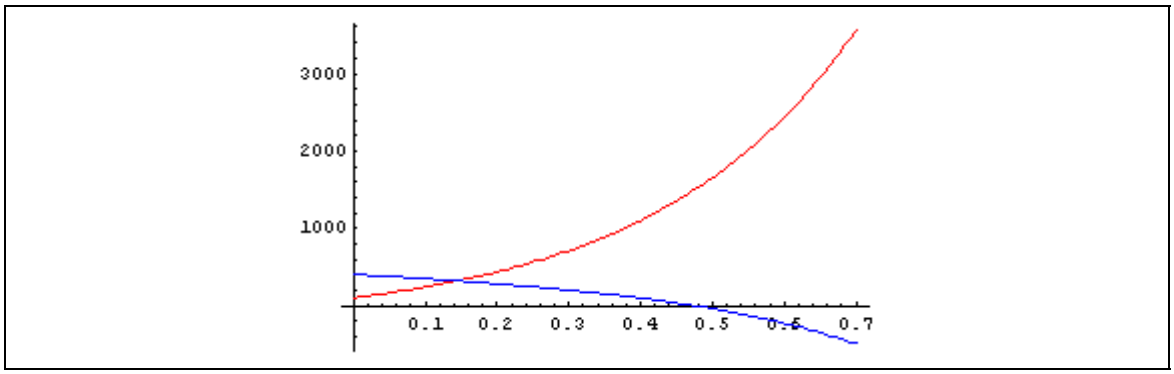
$$\begin{pmatrix} y[x] \to 100\,e^{2x}\,(-9 + 10\,e^{x}) \\ z[x] \to -100\,e^{2x}\,(-9 + 5\,e^{x}) \end{pmatrix}$$



- Graphics -

Con respecto a los comandos para obtener soluciones mediante aproximaciones numéricas, Mathematica cuenta con NDsolve. El mismo concede una solución numérica devolviendo los resultados no con una tabla o una lista para cada paso iterativo sino que lo devuelve internamente como una función interpolada por diversos métodos a los efectos de que el usuario simplemente pueda evaluar dicha solución en el instante *t* que desee. Esto es una gran ventaja para el usuario de Mathematica 4.0.
A continuación se muestra el uso de este comando en una sencilla programación en donde además se incluye la graficación de los resultados.

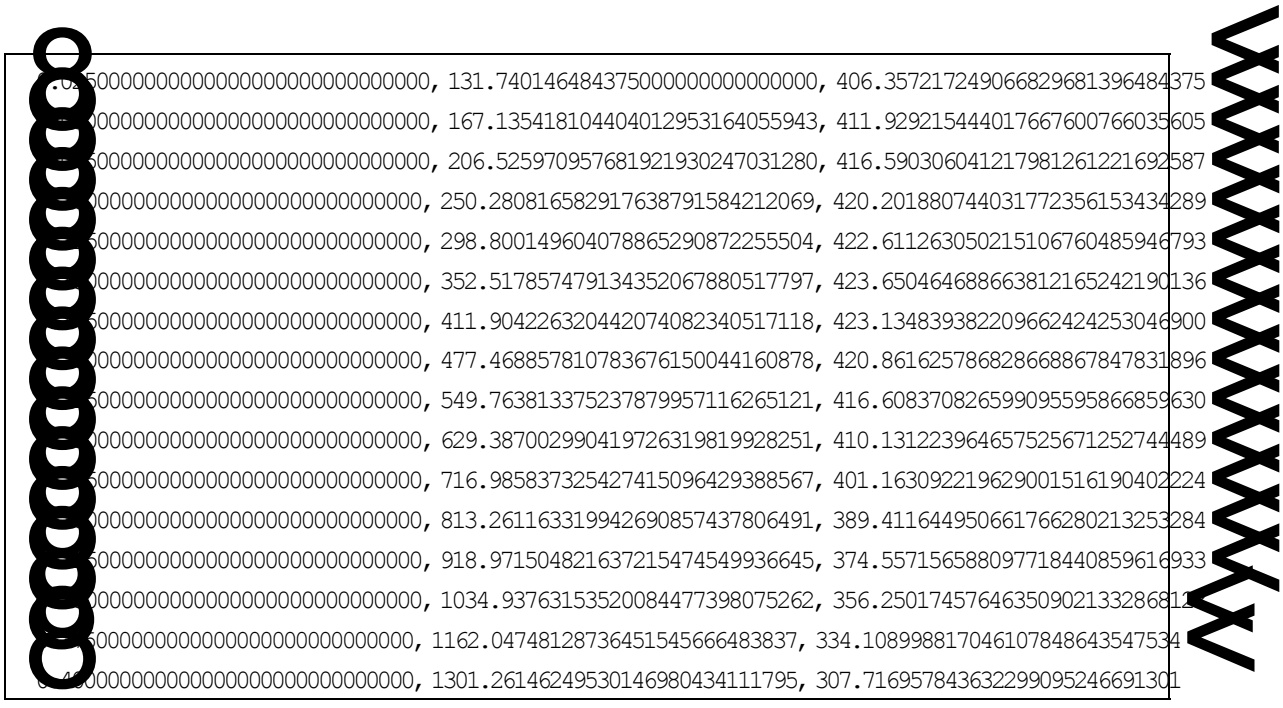
```
solution =
 NDSolve[{x'[t] == 4 x[t] + 2 y[t], x[0] == 100,
   y'[t] == -x[t] - +y[t], y[0] == 400},
  {x, y}, {t, 0, 25}, Method → RungeKutta,
  MaxSteps → 500];
Plot[Evaluate[{x[t], y[t]} /. solution],
 {t, 0, 0.7}, PlotPoints → 100, PlotRange → All,
 PlotStyle → {RGBColor[1, 0, 0],
   RGBColor[0, 0, 1]}];
```



Adicionalmente, para quienes no le agradan los comandos enlatados (ya que se pierde control sobre lo que uno está haciendo o simplemente quiere generar su propio método de solución numérico) se expone a continuación una programación mediante el comando **`For`**

La misma es totalmente general ya que permite la adecuación a cualquier sistema de ecuaciones diferenciales, permite el cambio de condiciones iniciales, de tiempo inicial y final, de iteraciones, el numero de dígitos de precisión con que se muestren los resultados.

```
iteraciones = 20;
x0 = 100;
y0 = 400;
t0 = 0;
tf = 1  2;
$Max
$Max                = Infinity;
h=                  iones;
f1                  2y;
f2                  y;
For                 i < iteraciones,
  x=
   x+
      2* h* f1
        y+ h*
      h* f1
          y+h*
        y+ h* f2
          y+h*
  y=
   y+
      2* h* f2
        y+ h*
      h* f2
          y+h*
Print
```


.025000000000000000000000000000, 131.74014648437500000000000000, 406.35721724906682968139648437
000000000000000000000000000000, 167.13541810440401295316405594 3, 411.92921544401766760076603560 5
500000000000000000000000000000, 206.52597095768192193024703128 0, 416.59030604121798126122169258 7
000000000000000000000000000000, 250.28081658291763879158421206 9, 420.20188074403177235615343428 9
500000000000000000000000000000, 298.80014960407886529087225550 4, 422.61126305021510676048594679 3
000000000000000000000000000000, 352.51785747913435206788051779 7, 423.65046468866381216524219013 6
500000000000000000000000000000, 411.90422632044207408234051711 8, 423.13483938220966242425304690 0
000000000000000000000000000000, 477.46885781078367615004416087 8, 420.86162578682866886784783189 6
500000000000000000000000000000, 549.76381337523787995711626512 1, 416.60837082659909559586685963 0
000000000000000000000000000000, 629.38700299041972631981992825 1, 410.13122396465752567125274448 9
500000000000000000000000000000, 716.98583732542741509642938856 7, 401.16309221962900151619040222 4
000000000000000000000000000000, 813.26116331994269085743780649 1, 389.41164495066176628021325328 4
500000000000000000000000000000, 918.97150482163721547454993664 5, 374.55715658809771844085961693 3
000000000000000000000000000000, 1034.9376315352008447739807526 2, 356.25017457646350902133286811
500000000000000000000000000000, 1162.0474812873645154566648383 7, 334.10899881704610784864354753 4
000000000000000000000000000000, 1301.2614624953014698043411179 5, 307.71695784363229909524669130 1
```
```

```
...0000000000000000000000000000, 1453.618165749487403710882406 82, 276.619465831468671803481139580
...0000000000000000000000000000, 1620.240515596934593753984002 9, 240.320843320146354226782277175
...0000000000000000000000000000, 1802.342395947912277291627669 05, 198.280883219495650144351617167
...0000000000000000000000000000, 2001.235785040880762829187255 24, 149.911142256772954747249129762
```

MAPLE 6

Maple 6 es un programa simbólico muy parecido a Mathematica 4.0. Básicamente resuelven los mismos problemas aunque Maple 6 tiene quizás comandos mas simples e intuitivos que Mathematica. Las consideraciones hechas en cuanto a la forma en que estos programas trabajan es idéntica a la descripta en la sección sobre Mathematica por lo que es innecesaria su repetición. Acontinuación se expondrán las programaciones analíticas usando los comandos **exponential** y **dsolve** y las numéricas usando **for** y **dsolve** con la opción **type = numeric.**

```
restart: with(linalg, coldim, diag, eigenvals, equal,
exponential, multiply, rowdim) :

A := matrix(2, 2, [4, 2, -1, 1]):
C := matrix(2, 1, [100, 400]):
S := simplify(multiply(exponential(A,t),C )):
matrix(2,1,[x(t),y(t)]) =
matrix(2,1,[collect(S[1,1],exp),collect(S[2,1],exp)]);
```

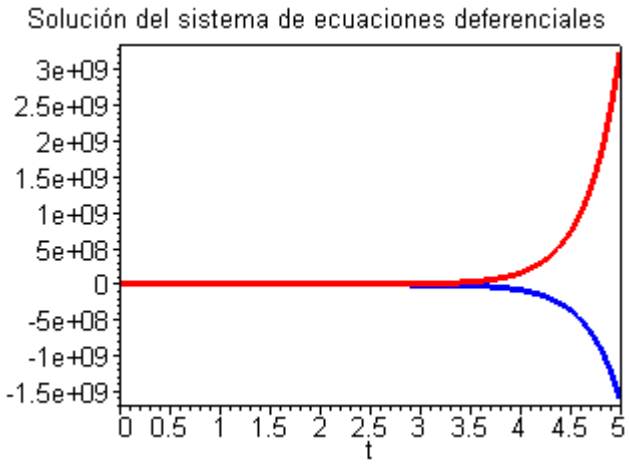
$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} -900\,e^{(2t)} + 1000\,e^{(3t)} \\ -500\,e^{(3t)} + 900\,e^{(2t)} \end{bmatrix}$$

```
sys := {diff(x(t),t)=4*x(t)+2*y(t),diff(y(t),t)=-
x(t)+y(t)}:
IC := {x(0)=100,y(0)=400}:
```

```
ans := combine(dsolve(sys union IC,{x(t),y(t)}),trig):
expr1 := subs(ans, x(t));
expr2 := subs(ans, y(t));
plot( [expr1,expr2],
t=0..5,color=[red,blue],tickmarks=[20,10],
style=[line,line],thickness=3, axes=BOXED,
title=`Solución del sistema de ecuaciones deferenciales` );
```

$$expr1 := 1000\,\mathbf{e}^{(3t)} - 900\,\mathbf{e}^{(2t)}$$

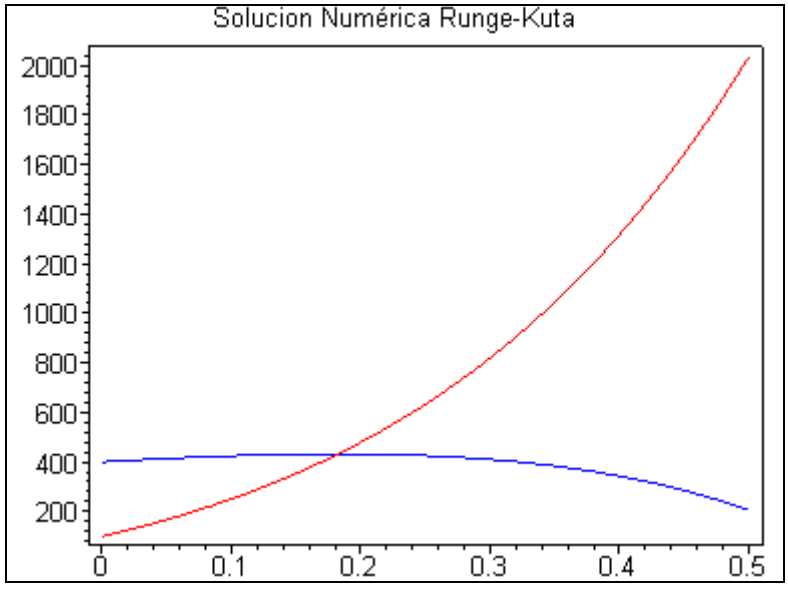
$$expr2 := 900\,\mathbf{e}^{(2t)} - 500\,\mathbf{e}^{(3t)}$$




```
q:=dsolve({diff(x(t),t)=4*x(t)+2*y(t),diff(y(t),t)=-
x(t)+y(t),x(0)=100,y(0)=400}, {x(t),y(t)}, type=numeric,
output=listprocedure):
q(0.3);
with(plots):
A:=odeplot(q,[t,x(t)],0..0.5,color=red):
B:=odeplot(q,[t,y(t)],0..0.5,color=blue):
```

```
display({A,B},axes=boxed,title=`Solucion Numérica Runge-
Kuta`);
```

$$[t(.3) = .3,\ y(t)(.3) = 410.1053655832982,\ x(t)(.3) = 819.6961890753631]$$



>

```
iteraciones := 20:
precision :=20:
x0 := 100:
y0 := 400:
t0 := 0:
tf := 1/2:
h := (tf - t0)/iteraciones:
f1:= 4*x + 2*y:
f2:= -x + y:
```

```
with(linalg):
a:=[t0,x0,y0]:

for i from 0 by 1 while i < iteraciones do
t0 := t0 + h:
u1 := h*eval(f1,[t=t0,x=x0, y=y0]):
v1 := h*eval(f2,[t=t0, x=x0, y=y0]):
u2 := h*eval(f1,[t=t0+h/2, x=x0 + u1/2, y=y0 + v1/2]):
v2 := h*eval(f2,[t=t0+h/2, x=x0 + u1/2, y=y0 + v1/2]):
u3 := h*eval(f1,[t=t0 + h/2, x=x0 + u2/2, y=y0 + v2/2]):
v3 := h*eval(f2,[t=t0 + h/2, x=x0 + u2/2, y=y0 + v2/2]):
u4 := h*eval(f1,[t=t0 + h, x=x0 + u3, y=y0 + v3]):
v4 := h*eval(f2,[t=t0 + h, x=x0 + u3, y=y0 + v3]):
x0 := x0 + (1/6)*(u1 + 2*u2 + 2*u3 + u4):
y0 := y0 + (1/6)*(v1 + 2*v2 + 2*v3 + v4):
b := evalf([t0,x0,y0],precision):
a := concat(a,b):
print(evalf([t0,x0,y0],precision))
od:
pl1 := plot([seq([a[1,i], a[3,i]], i = 1 .. iteraciones)],
axes = boxed, color = [blue], labels = ["tiempo",
"poblaciones"], title = "Solucion Numérica RK(4)"):
pl2 := plot([seq([a[1,i], a[2,i]], i = 1 .. iteraciones)],
axes = boxed, color = [red], labels = ["tiempo",
"poblaciones"], title = "Solucion Numérica RK(4)"):
plots[display]([pl1, pl2]);
```

```
[.025000000000000000000, 131.74014648437500000, 407.20191894531250000]

[.050000000000000000000, 167.18037825958108902, 413.73672151965403557]

[.075000000000000000000, 206.67183577427910085, 419.48948742261057610]

[.100000000000000000000, 250.59623590313512444, 424.33311772870051084]

[.125000000000000000000, 299.36841887634292970, 428.12722085458797760]

[.150000000000000000000, 353.43910206047671746, 430.71690327488274304]

[.175000000000000000000, 413.29785713619360104, 431.93145713286975637]

[.200000000000000000000, 479.47632852924582435, 431.58293625852906052]
```

[.22500000000000000000, 552.55171236544850111, 429.46461142170342097]

[.25000000000000000000, 633.15051674577942476, 425.34929490925340821]

[.27500000000000000000, 721.95262578339690040, 418.98752371720030053]

[.30000000000000000000, 819.69569161946661744, 410.10558978754997194]

[.32500000000000000000, 927.17988054944309912, 398.40340478970462943]

[.35000000000000000000, 1045.2730014568033335, 383.55218594269880664]

[.37500000000000000000, 1174.9160469789842620, 365.19194829111171017]

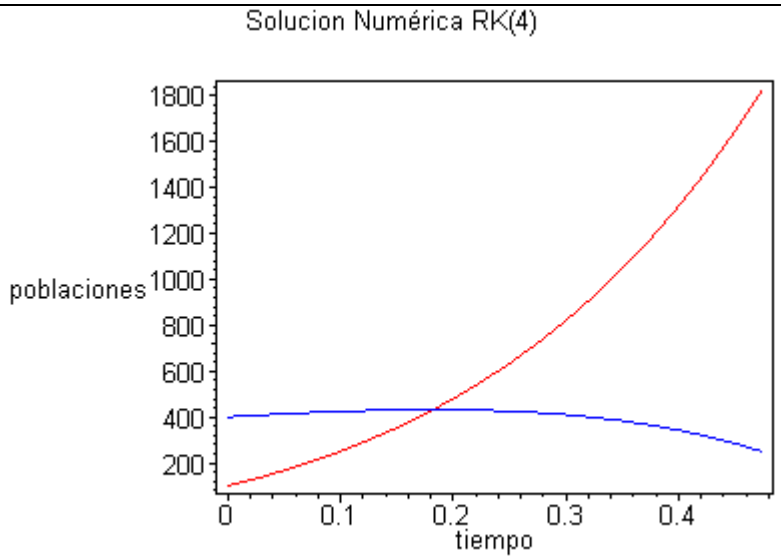
[.40000000000000000000, 1317.1291802331595706, 342.92878767812224987]

[.42500000000000000000, 1473.0182025212277339, 316.33193739700450452]

[.45000000000000000000, 1643.7815402287686136, 284.93058014009619035]

[.47500000000000000000, 1830.7177921477130873, 248.21039539402300942]

[.50000000000000000000, 2035.2338817042593634, 205.60982084322059903]



>

MATLAB 5.3

Matlab posee un gran versatilidad en su lenguaje ya que permite encarar este problema de sistemas dinámicos por diversos medios:

- Mediante el uso de Matrices Exponenciales
- Mediante el uso de comandos enlatados ODE Solvers
- Mediante la creación de un propio programa que implemente el algoritmo de Runge-Kuta. Esto último puede hacerse de dos formas: una es definiéndolo en un mismo programa y otra es creando una función Runge-Kuta que pueda ser utilizada como un nuevo comando en cualquier programa (es decir se puede crear una nueva función que de ahora en mas sea reconocida por el lenguaje)

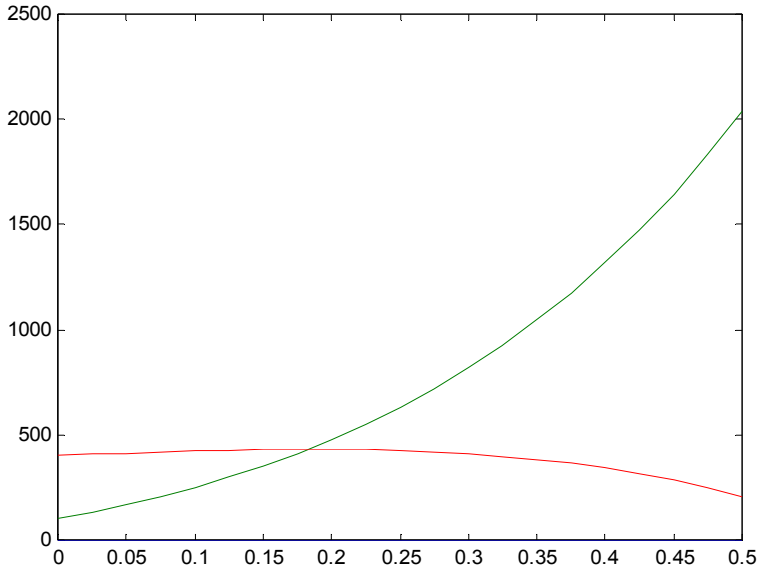
A diferencia de Gauss este programa cuenta con comandos ya programados que habilitan el cálculo de matrices exponenciales en la resolución de sistemas de ecuaciones diferenciales. Sin embargo merecen que se realicen algunas consideraciones al respecto:
En primer lugar el comando puede presentar fallas cuando la multiplicidad geométrica es menor que la multiplicidad algebraica de los valores propios repetidos
En segundo lugar al usar algoritmos numéricos tanto en el cálculo de la inversa como en el cálculo de los valores y vectores propios se pueden cometer errores de redondeo sumamente importantes los que se podrían acumular aun más si dicha matriz exponencial interviene en procesos iterativos.
Por último, vale destacar la mayor velocidad con que se realizan las operaciones en comparación a otros Software como Mathematica o Maple e incluso también con Gauss (aunque con respecto a este último la diferencias de velocidades no es muy significativa).
Se expone a continuación el programa utilizado mediante el comando `expm`.

```
%Resolucion de un sistema de ecuaciones por el método de la matriz
exponencial%
%_____________________________________________________________________%

A=[4 2; -1 1];                  %matriz de coeficientes%
C=[100 400];                    %condiciones iniciales%
t0=0;                           %valor inicial del tiempo%
tf=0.5;                         %valor final del tiempo%
h=0.025;                        %longitud de paso%
k=(tf-t0)/h;                    %calculo del numero de iteraciones%
g=C;
tab=[t0,C];
for i= 1:k
   t0=t0+h;
   w=(expm(A*(t0))*C')';
   g=[g;w];
   tab2=[t0,w];
   tab=[tab;tab2];
end
u = 0:h:tf;                     %dominio para graficacion%
plot(u',g)                      %grafica de ambas poblaciones en (0, t)%
tab
```

tab =

0	100	400
0.025	131.74016414621	407.201911296106
0.05	167.1804164602	413.736704903941
0.075	206.67189773641	419.489460359523
0.1	250.59632523185	424.333078556152
0.125	299.368539599234	428.127167709867
0.15	353.439258671766	430.716834073319
0.175	413.29805464516	431.931369544386
0.2	479.476572513365	431.582827681888
0.225	552.552009028696	429.464478956229
0.25	633.150872982559	425.349135323778
0.275	721.953049248648	418.987333416004
0.3	819.696190805492	410.105364772983
0.325	927.1804648701	398.403140621203
0.35	1045.27368133974	383.551877691848

```
0.375          1174.91683396662          365.191590492393

0.4            1317.13008709332          342.928374274946

0.425          1473.01924336819          316.331461682602

0.45           1643.78273065572          284.930034692768

0.475          1830.71914937175          248.209772006257

0.5            2035.23542472492          205.609110444109
```



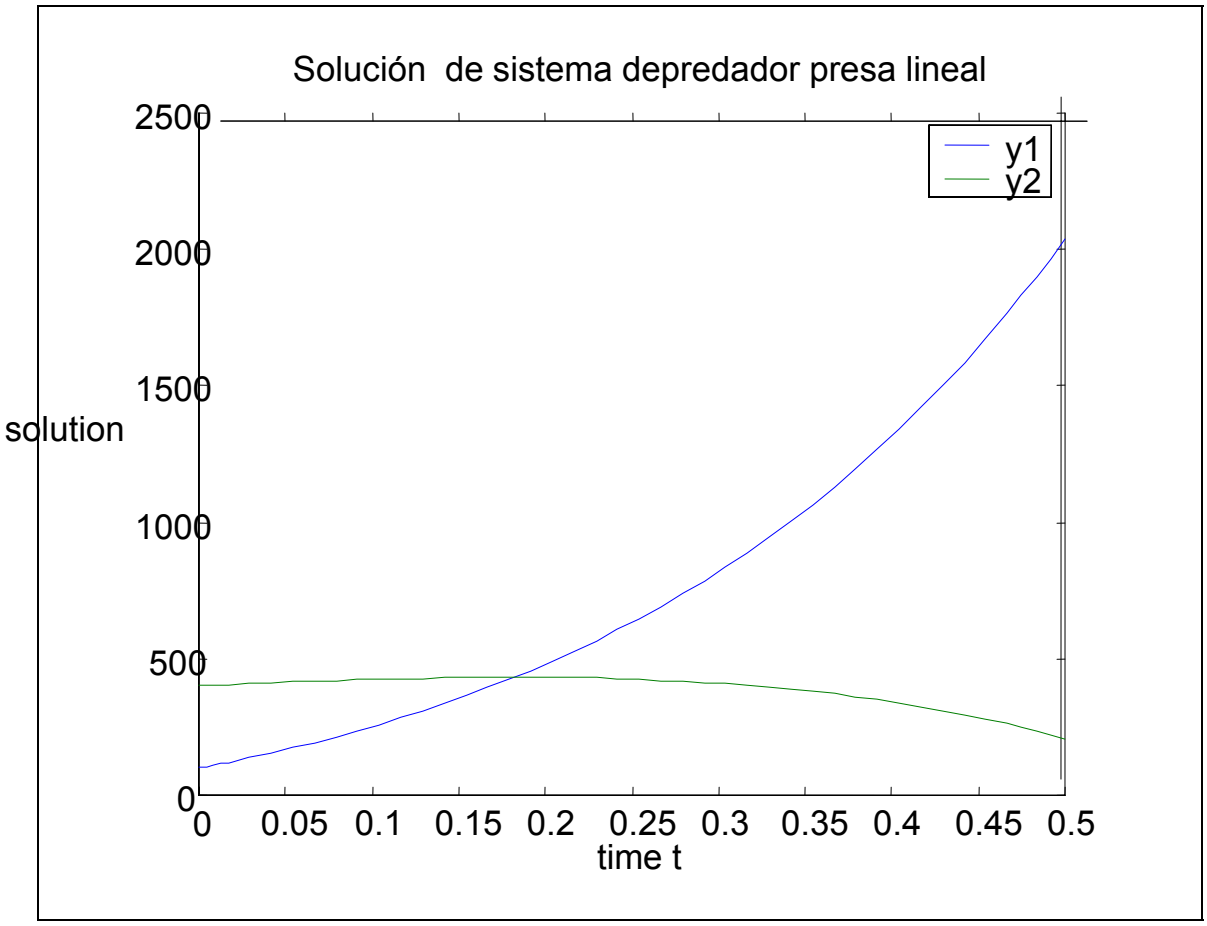
Matlab cuenta además con comandos predeterminados para resolver cualquier tipo de ecuaciones diferenciales y sistemas en forma numérica por medio del comando `ode45` que requiere la definición previa del sistema de ecuaciones diferenciales en un archivo del mismo nombre que el comando con una extensión *.m. Para este caso el caso se utilizó el nombre dpl para crear el archivo que contiene la definición del sistema. A continuación se exponen los comandos utilizados tanto para la creación del sistema como para la utilización del mismo junto a su representación gráfica de la solución:

```
function dy = dpl(t,y)                    %crea el ODE file.m%
dy = [4*y(1)+2*y(2); -y(1)+y(2)];
```

```
[t,y] = ode45('dpl',[0 0.5],[100; 400]);          %solucion%
```

```
plot(t,y(:,1),t,y(:,2))                          %grafica%
title('Solution de sistema depredador presa lineal');
xlabel('time t');
ylabel('solucion y');
legend('y1','y2')

sol = [t,y]
```

0	100	400
0.0041864773858493	105.071337754293	401.247978827619
0.0083729547716986	110.238783873389	402.479716779759
0.0125594321575479	115.503822148739	403.694739446056
0.0167459095433972	120.867957595892	404.89256406742
0.0292459095433972	137.489104493118	408.361789713007
0.0417459095433972	155.049208551019	411.659742813136
0.0542459095433972	173.591829610744	414.77219167813
0.0667459095433972	193.162432643268	417.684146381758
0.0792459095433972	213.808415529899	420.379846321161
0.0917459095433972	235.579105693167	422.842761259667
0.104245909543397	258.525967864002	425.055500917081
0.116745909543397	282.702702240996	426.999772045435
0.129245909543397	308.165276931081	428.65636381133
0.141745909543397	334.971923962462	430.005149080249
0.154245909543397	363.183382202585	431.02497784851
0.166745909543397	392.863012100501	431.693626674825
0.179245909543397	424.076833559181	431.987781515389
0.191745909543397	456.89352122283	431.883039296404
0.204245909543397	491.384688394848	431.353782412137
0.216745909543397	527.625021120359	430.373119201839
0.229245909543397	565.692322388302	428.912863806236
0.241745909543397	605.667506567241	426.943538083371
0.254245909543397	647.634931135992	424.434223931119
0.266745909543397	691.682553317314	421.352493279236
0.279245909543397	737.901981655249	417.664384464092
0.291745909543397	786.388469452287	413.334404552727
0.304245909543397	837.241302246535	408.325355704937
0.316745909543397	890.563980735721	402.598252894013
0.329245909543397	946.464280946685	396.112296212923
0.341745909543397	1005.05424650196	388.824873682982
0.354245909543397	1066.45064108196	380.691357237429
0.366745909543397	1130.775161993	371.665006085916
0.379245909543397	1198.15451034521	361.69693426793
0.391745909543397	1268.72038194772	350.7361140359
0.404245909543397	1342.60999551218	338.729136304422
0.416745909543397	1419.96634193357	325.620097225712
0.429245909543397	1500.9382661247	311.35056019261
0.441745909543397	1585.6804563054	295.859559901705
0.454245909543397	1674.35406046983	279.083321252759
0.466745909543397	1767.12697728036	260.955126297126
0.475059432157548	1831.18401515532	248.115657169
0.483372954771699	1897.1845121398	234.626182618668
0.491686477385849	1965.18300333522	220.464828825142
0.5	2035.23551843932	205.60906618047

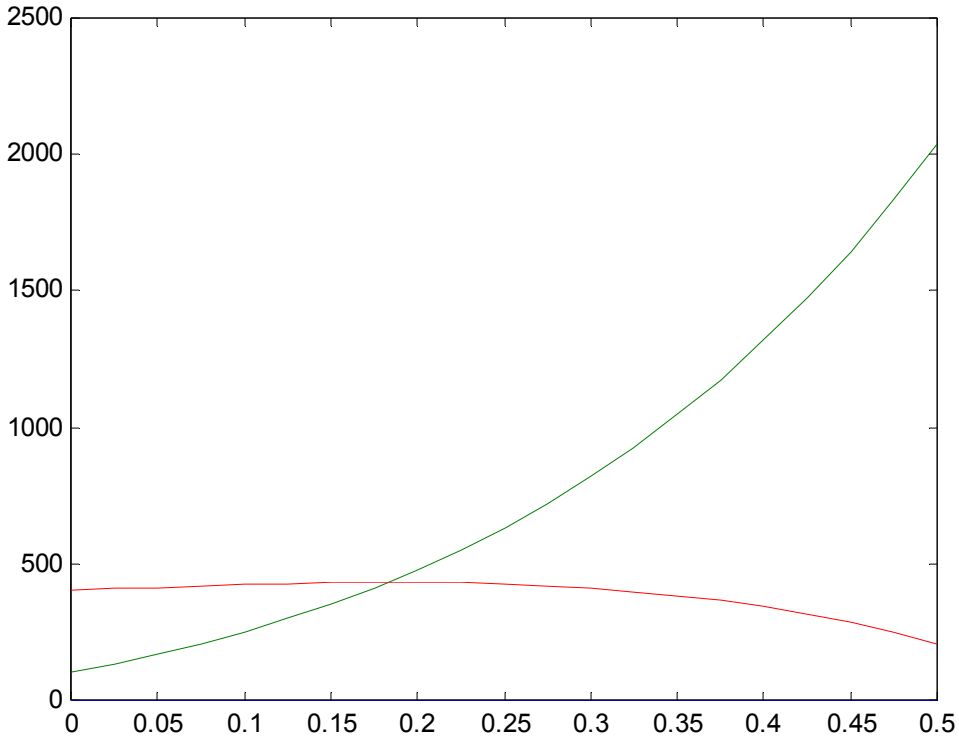


Como se anticipó existen además dos formas adicionales de resolver un sistema de ecuaciones diferenciales con Matlab. Una de ellas es creando una rutina que actúa dentro de un mismo programa sin necesidad de crear una función auxliar mediante un archivo *:m. A continuación se exponen los comandos empleados en esta programación

```
h=0.025
xx=inline('4*x+2*y','t','x','y');
yy=inline('-x+y','t','x','y');          %esto crea una funcion dentro
del programa sin la necesidad de definirla previamente mediante un m-
file%

t0=0;
tf=0.5;
x=100;
y=400;

t = t0;
g=[t,x,y];
tab1=[t,x,y];
while(t<tf)
     t = t+h;
     u1=h*eval('xx(t,x,y)');
     v1=h*eval('yy(t,x,y)');
```

```
      u2=h*eval('xx(t+h/2,x+u1/2,y+v1/2)');
      v2=h*eval('yy(t+h/2,x+u1/2,y+v1/2)');
      u3=h*eval('xx(t+h/2,x+u2/2,y+v2/2)');
      v3=h*eval('yy(t+h/2,x+u2/2,y+v2/2)');
      u4=h*eval('xx(t+h,x+u3,y+v3)');
      v4=h*eval('yy(t+h,x+u3,y+v3)');
      x=x+(1/6)*(u1+2*u2+2*u3+u4);
      y=y+(1/6)*(v1+2*v2+2*v3+v4);
      p=[t,x,y];
      g=[g;p];
      tab2=[t,x,y];
      tab1=[tab1;tab2];
end

tab1

u = 0:h:tf;                %dominio para graficacion%
plot(u',g)
```

0	100	400
0.025	131.740146484375	407.201918945313
0.05	167.180378259581	413.736721519654
0.075	206.671835774279	419.489487422611
0.1	250.596235903135	424.333117728701
0.125	299.368418876343	428.127220854588
0.15	353.439102060477	430.716903274883
0.175	413.297857136194	431.93145713287
0.2	479.476328529246	431.582936258529
0.225	552.551712365448	429.464611421703
0.25	633.150516745779	425.349294909253
0.275	721.952625783397	418.9875237172
0.3	819.695691619467	410.10558978755
0.325	927.179880549443	398.403404789705
0.35	1045.2730014568	383.552185942699
0.375	1174.91604697898	365.191948291112
0.4	1317.12918023316	342.928787678122
0.425	1473.01820252123	316.331937397005
0.45	1643.78154022877	284.930580140096
0.475	1830.71779214771	248.210395394023
0.5	2035.23388170426	205.609820843221

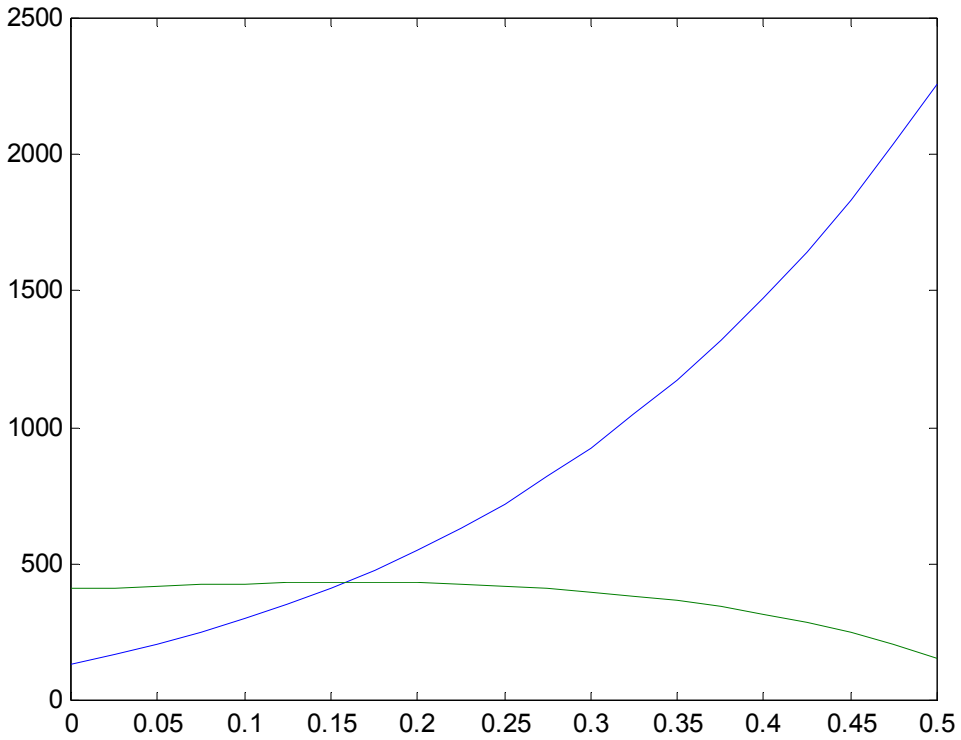


Tambien puede resolverse este problema creando una función auxiliar en un archivo aparte con el nombre de la función. En este caso se optó por llamarla Runge-Kuta. La misma es una función que recibe 6 argumentos: dos funciones definidas previamente (ya sea por medio de un m-file o por medio del comando `inline`), los valores iniciales de las funciones, el tiempo inicial y la longitud de paso; mientras que el resultado que devuelve es un vector con los valores de la función solución en el periodo inicial mas la longitud de paso (t_0+h). La ventaja de esto es que la función quedará habilitada de ahora en mas para ser usada en cualquier otro programa, es decir de ahora en más es parte del lenguaje Matlab. Se expone a continuación esta metodología:

```
function q = rungekuta(xx,yy,x,y,t,h)

u1=h*eval('xx(t,x,y)');
v1=h*eval('yy(t,x,y)');
u2=h*eval('xx(t+h/2,x+u1/2,y+v1/2)');
v2=h*eval('yy(t+h/2,x+u1/2,y+v1/2)');
u3=h*eval('xx(t+h/2,x+u2/2,y+v2/2)');
v3=h*eval('yy(t+h/2,x+u2/2,y+v2/2)');
u4=h*eval('xx(t+h,x+u3,y+v3)');
v4=h*eval('yy(t+h,x+u3,y+v3)');
xx1=x+(1/6)*(u1+2*u2+2*u3+u4);
yy1=y+(1/6)*(v1+2*v2+2*v3+v4);

q = [xx1,yy1];
```

```
r=inline('4*x+2*y','t','x','y');
s=inline('-x+y','t','x','y');
t0=0;
tf=0.5;
h=0.025;


t = t0;
x0=100;
y0=400;
a = rungekuta(r,s,x0,y0,t,h);
tab = [t,a];
g = a;
while(t<tf)
   x0 = a(1);              %valor de x en t%
   y0 = a(2);              %valor de y en t%
   t = t+h;
   tab=[tab;t,x0,y0];      %concatenacion para formar una matriz
[t,x(t),y(t)]%
   a = rungekuta(r,s,x0,y0,t,h);
   g = [g;a];              %concatenacion para formar un vector
[x(t),y(t)] a usarse en graficas%
end

end
u = 0:h:tf;               %dominio para graficacion%
plot(u',g)                     %grafica de ambas poblaciones en (0, t)%
tab
```

```
            0          131.740146484375          407.201918945313
        0.025           131.740146484375           407.201918945313
         0.05           167.180378259581           413.736721519654
        0.075           206.671835774279           419.489487422611
          0.1          250.596235903135          424.333117728701
        0.125           299.368418876343           428.127220854588
         0.15           353.439102060477           430.716903274883
        0.175           413.297857136194            431.93145713287
          0.2          479.476328529246          431.582936258529
        0.225           552.551712365448           429.464611421703
         0.25           633.150516745779           425.349294909253
        0.275           721.952625783397              418.9875237172
          0.3          819.695691619467           410.10558978755
        0.325           927.179880549443           398.403404789705
         0.35            1045.2730014568           383.552185942699
        0.375           1174.91604697898           365.191948291112
          0.4          1317.12918023316          342.928787678122
        0.425           1473.01820252123           316.331937397005
         0.45           1643.78154022877           284.930580140096
        0.475           1830.71779214771           248.210395394023
          0.5          2035.23388170426          205.609820843221
```



EXCEL 2002

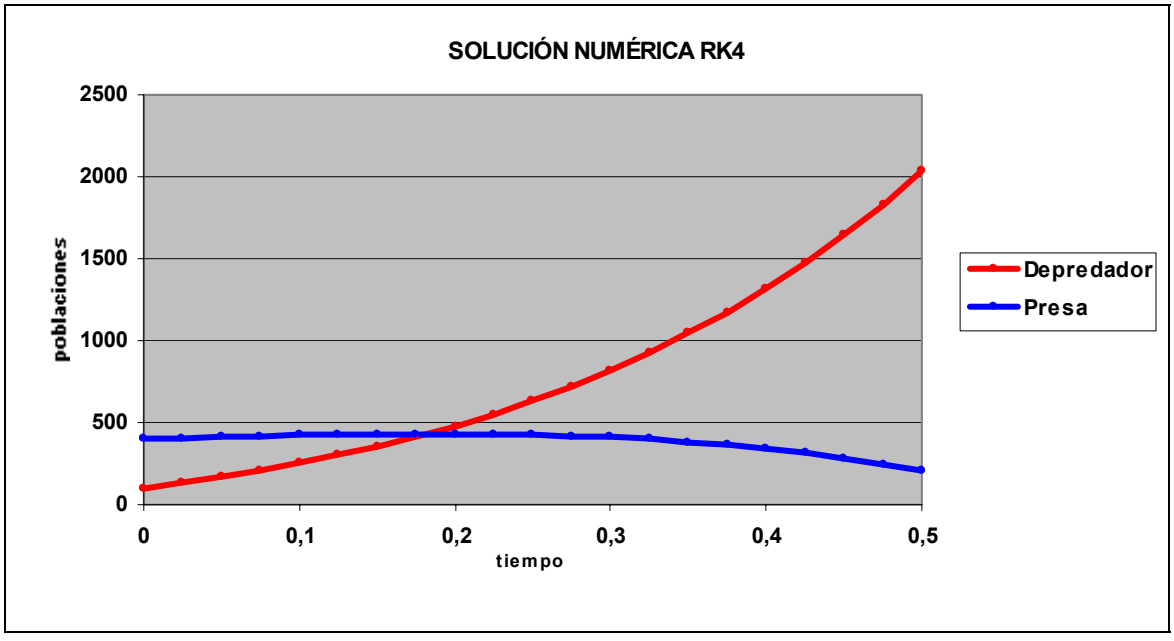
Finalmente se verá como con una simple planilla de cálculos se puede resolver un sistema de ecuaciones diferenciales y obtener una representación gráfica de las soluciones de las mismas. Para ello se recurrirá a definir dos nuevas funciones en el editor de los módulos de Visual Basic de Excel 2002 mediante los comandos `Function`. Esto facilitará la programación en la planilla de cálculo.
Los comandos que se exponen a continuación corresponden a la declaración de funciones en los módulos de Visual Basic y pueden ser modificados y readaptados para resolver cualquier tipo de ecuación diferencial sea línea o no.

```
Function XX(t, x, y)
XX = 4 * x + 2 * y
End Function

Function YY(t, x, y)
YY = -x + y
End Function
```

```
Private Sub Workbook_Open()

End Sub



Private Sub Worksheet_SelectionChange(ByVal Target As
Range)

End Sub
```

Longitud de paso	0,025											
				Cuatro evaluaciones de XX(t,x,p)					Cuatro evaluaciones de YY(t,x,p)			
				para tiempo t, t+h t+h/2 (dos veces)					para tiempo t, t+h t+h/2 (dos veces)			
t	**x(t)**	**p(t)**		k1	k2	k3	k4		k1	k2	k3	k4
0	100	400		1200	1267,5	1270,569	1341,445		300	288,75	287,7656	275,4299
0,025	131,7399	407,2019		1341,364	1415,318	1418,656	1496,281		275,462	262,1382	261,0472	246,5218
0,05	167,1799	413,7367		1496,193	1577,167	1580,795	1665,759		246,5568	230,9364	229,729	212,7802
0,075	206,671	419,4895		1665,663	1754,267	1758,21	1851,15		212,8185	194,6579	193,3234	173,6963
0,1	250,5951	424,3332		1851,047	1947,942	1952,226	2053,834		173,7381	152,7717	151,2985	128,7149
0,125	299,3668	428,1273		2053,722	2159,627	2164,28	2275,304		128,7605	104,6985	103,0739	77,23034
0,15	353,437	430,717		2275,182	2390,873	2395,926	2517,175		77,28006	49,80629	48,01673	18,58233
0,175	413,2951	431,9317		2517,044	2643,362	2648,848	2781,2		18,63653	-12,5936	-14,5629	-47,9487
0,2	479,4729	431,5832		2781,058	2918,914	2924,868	3069,274		-47,8897	-83,2515	-85,4167	-123,147
0,225	552,5475	429,465		3069,12	3219,499	3225,961	3383,448		-123,082	-162,985	-165,364	-207,866
0,25	633,1453	425,3498		3383,281	3547,25	3554,261	3725,942		-207,796	-252,684	-255,295	-303,034
0,275	721,9463	418,9882		3725,762	3904,476	3912,081	4099,161		-302,958	-353,317	-356,181	-409,665
0,3	819,6882	410,1065		4098,966	4293,674	4301,922	4505,704		-409,582	-465,939	-469,077	-528,857
0,325	927,1709	398,4045		4505,493	4717,548	4726,492	4948,385		-528,7	-591,6	-595,1	-661,8

									66	95	32	07
0,35	1045,262	383,5536		4948,157	5179,022	5188,718	5430,249		-661,709	-731,832	-735,595	-809,817
0,375	1174,904	365,1936		5430,002	5681,259	5691,769	5954,588		-809,71	-887,706	-891,822	-974,3
0,4	1317,115	342,9309		5954,32	6227,682	6239,072	6524,963		-974,184	-1060,79	-1065,29	-1156,79
0,425	1473,001	316,3345		6524,674	6821,991	6834,334	7145,227		-1156,67	-1252,68	-1257,6	-1358,97
0,45	1643,762	284,9336		7144,915	7468,19	7481,562	7819,546		-1358,83	-1465,13	-1470,49	-1582,63
0,475	1830,695	248,2141		7819,208	8170,607	8185,092	8552,424		-1582,48	-1700	-1705,86	-1829,75
0,5	2035,208	205,6142		8552,059	8933,922	8949,611	9348,732		-1829,59	-1959,36	-1965,76	-2102,48



Soluciones Numéricas: Sistemas no Lineales[10]

Cuando nos encontramos con una ecuación o un sistema de ecuaciones diferenciales no lineales, no podemos encontrar una solución analítica al problema, por lo que debemos

[10] La siguiente sección fue extraida del libro "Ecuaciones Diferenciales con aplicaciones de modelado" de Dennis Zill, utilizando también el ejemplo gráfico que aparece en pag.101 como sistema depredador – presa no lineal.

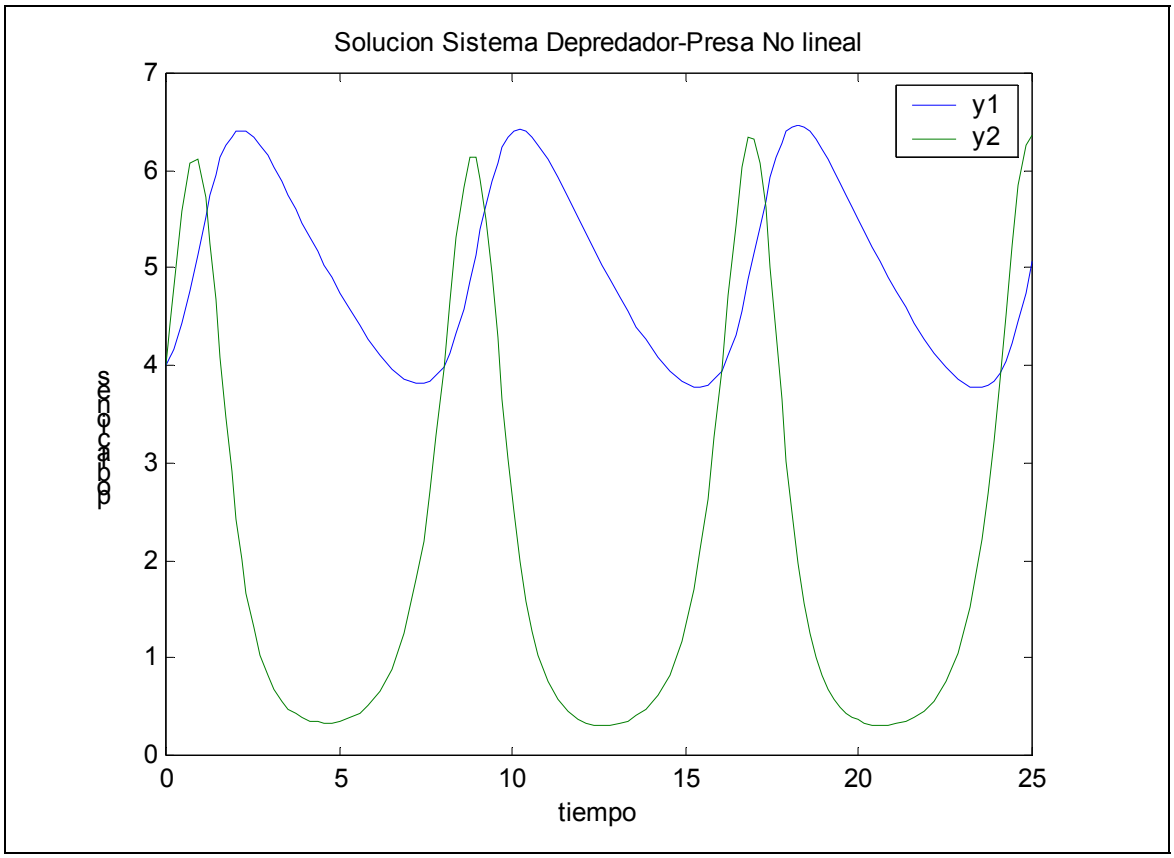
contentarnos con aproximaciones y hacer todo lo posible para que dichas aproximaciones sean lo más exactas posibles.
Al comienzo del presente problema, se presentó una serie de modelos demográficos con el objeto de hacer una correcta descripción de la evolución natural en el modelado del crecimiento demográfico. En una primera instancia se mostraron dos modelos univariantes, donde los valores presentes de la población es perfectamente determinado por los valores pasados de dicha población. El primero de ellos (modelo de crecimiento exponencial) supone que la tasa de crecimiento es constante en el tiempo. El segundo (modelo de crecimiento logístico) supera la rigidez de la tasa de crecimiento constante y le impone una cota superior al tamaño poblacional, por lo que se puede analizar los problemas de sobrepoblación.
A continuación, se presentaron tres nuevos modelos que permiten la iteración entre 2 especies, lo que agrega nuevas consideraciones al crecimiento demográfico. Todos son no lineales y se diferencian por la relación entre ambas especies que supone el sistema. El primero (modelo depredador presa de Lotka-Volterra) es un modelo clásico de sistema de ecuaciones: es el sistema no lineal por excelencia. Es el primer modelo no lineal que suele presentarse a los estudiantes para su análisis. Como no se puede llegar a una solución analítica es un caso en que la única opción es su resolución por métodos numéricos.

$$\frac{dx}{dt} = -0{,}16x + 0{,}08xy, \qquad x_0 = 4$$

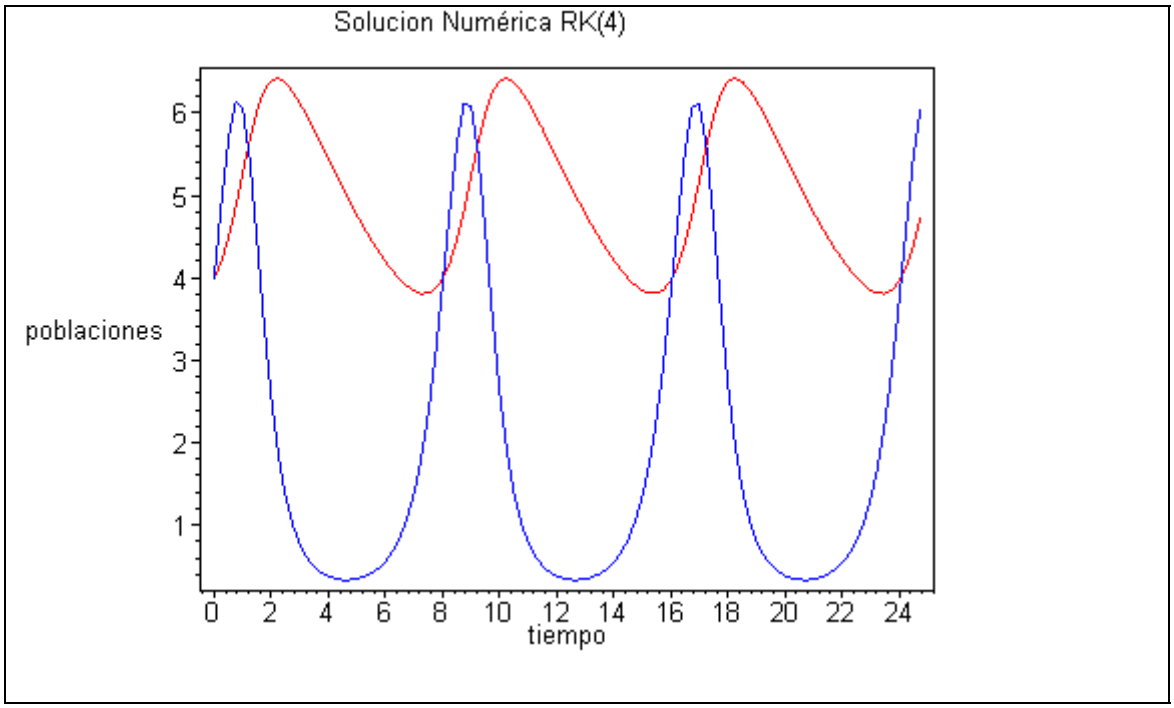
$$\frac{dy}{dt} = -0{,}9xy + 4{,}5y \qquad y_0 = 4$$

Se expone a continuación las representaciones graficas de las soluciones del sistema no lineal que se obtuvieron mediante las programaciones en los Softwares Matlab 5.3 y Maple 6.0 que se presentaron en la sección anterior. Para evitar una extensión innecesaria de líneas se omiten las salidas numéricas de tales programas como así también las salidas graficas de Mathematica y Excel.

Matlab 5.3


Solucion Sistema Depredador-Presa No lineal
y1
y2
tiempo
0
1
2
3
4
5
6
7
0
5
10
15
20
25

Maple 6.0



Como se aprecia, el sistema resulta ser bastante interesante. Claramente se ve la iteración entre ambas especies y como el medio se mantiene en un equilibrio dinámico que implica un comportamiento ciclico de las dos poblaciones, sin llegar a la extinción de ninguna de las dos especies. Los depredadores mantienen una población que oscila entre 4 y 6,5, repitiéndose el ciclo cada 8 períodos aproximadamente. En un principio, ante la abundancia de presas, su población aumenta hasta alcanzar un máximo de $x(t)$=6,4207 (en t=2,175), pero cuando ésta empieza a escasear en relación con la población de depredadores, los depredadores entran en una etapa de sobrepoblación y comienzan a experimentar un crecimiento negativo hasta alcanzar $x(t)$=3,8061 (en t=2,75).

Las presas presentan una variabilidad mayor. En una primera instancia, su población aumenta hasta alcanzar una población un poco mayor a 6 (siendo el máximo en 0,825 con $y(t)$=6,1674). Pero antes de alcanzar el primer período, el incremento en la población de depredadores alcanza tal magnitud que empieza a disminuir la cantidad de presas y dicho crecimiento negativo es tal que en el período 5 el número de presas es menor a 0,5 (siendo el mínimo de $y(t)$=0,3336 en el período 4,625). Sin embargo, el modelo la especie no llega al punto de extinción, sino que a partir del período 5 empieza a crecer nuevamente hasta que alrededor del período 8 llega al nivel de la población inicial (siendo 3,994 en el período 8,025) y alcanzando un nuevo máximo en 8,825 con $y(t)$=6,1673. A partir de 8,025, comienza un nuevo ciclo que sorprendentemente muestra un comportamiento similar de $y(t)$ que dura hasta t=16,05 (con $y(t)$=3,9888), y del 16,05 al 24,075, etc...

Los depredadores poseen ciclos de la misma duración de 8,025 que las presas y comienzan en los mismos períodos. La población de depredadores vuelve a ser de 4 en los mismos períodos ya mencionados para las presas. Sin embargo, los máximos y mínimos de ambas especies no ocurren en los mismos momentos.

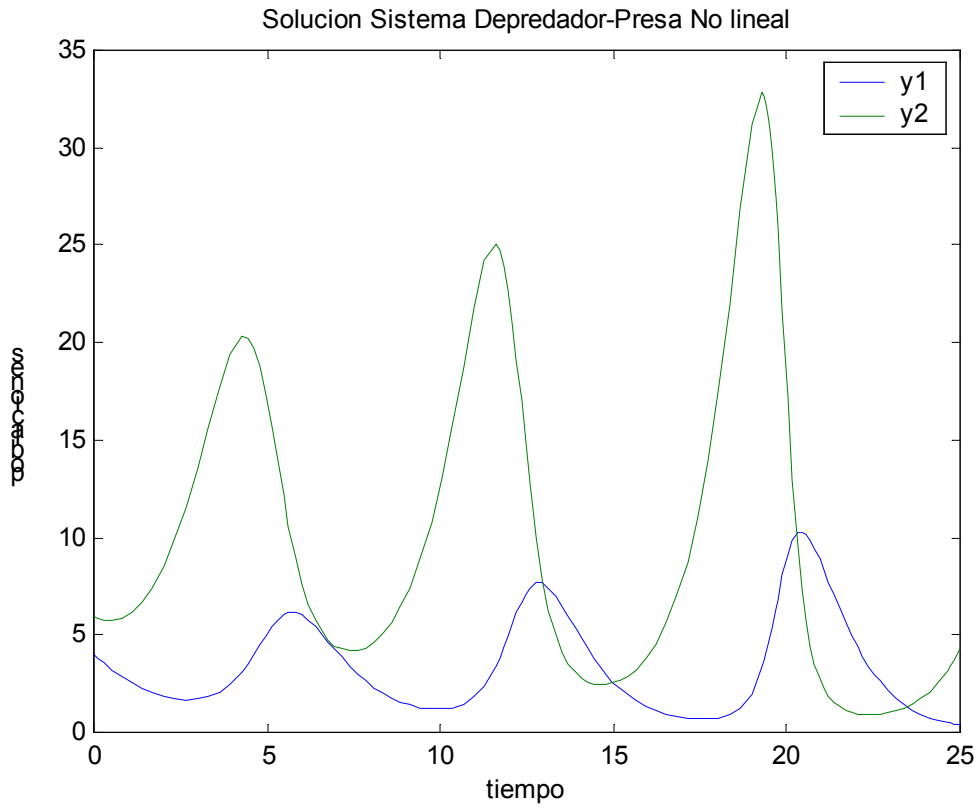
Claramente, este modelo permite un análisis de la relación entre las especies mucho más interesante que el logrado con el sistema linealizado propuesto anteriormente.

En cuanto a la inclusión dentro del modelo de la ecuación logística, se analizará el caso de:

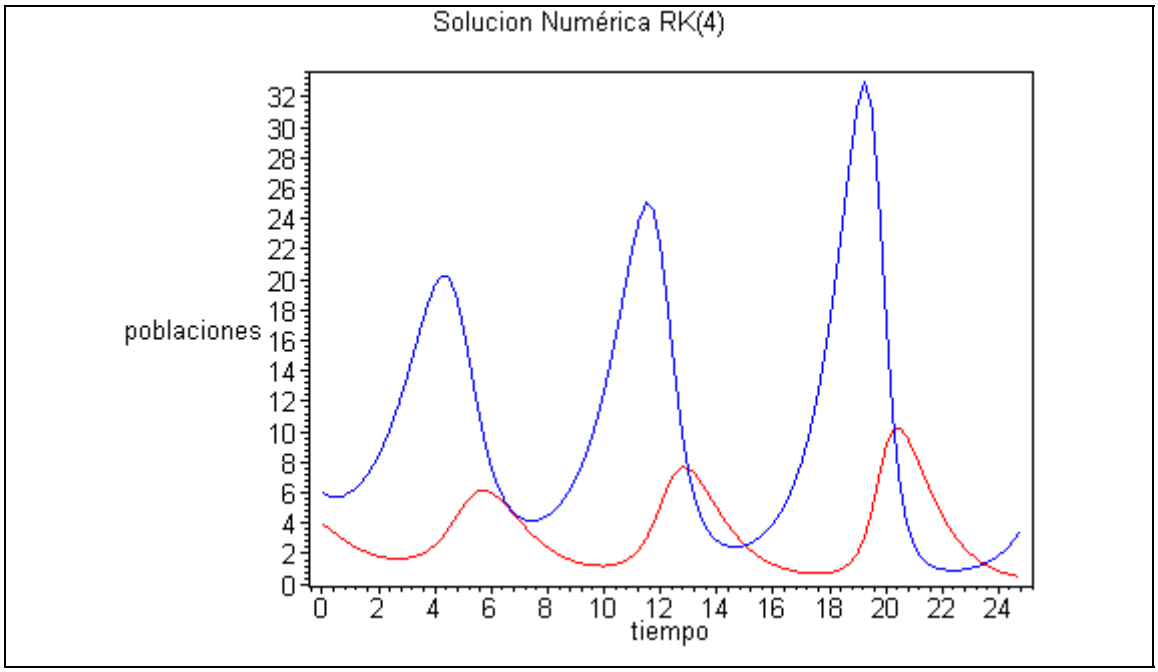
$$\frac{dx}{dt} = -x(1 - 0,035x) + 0,08xy, \qquad x_0 = 4$$
$$\frac{dy}{dt} = -0,3xy + y(1 - 0,0015y) \qquad y_0 = 6$$

El gráfico de este nuevo modelo es:

Matlab 5.3



Maple 6.0



Donde la línea roja representa a las presas y la línea verde a los depredadores. Si bien es cierto que a simple vista el modelo no parece aportar nada nuevo al caso anterior, cabe destacar que ahora es más sencillo encontrar parámetros adecuados para que el modelo esté bien especificado. Si la parte que corresponde a la función logística en cada una de las ecuaciones diferenciales está bien especificada de tal forma que no pueda darse una población negativa para ninguna de las dos especies, será muy difícil que el sistema lleve a la extinción de una especie[11].

Es en esta sección donde se aprecia en realidad la utilidad de los métodos numéricos para la solución de sistemas dinámicos. En los apartados anteriores, siempre existía una solución analítica permitía llegar a la solución exacta del problema. Las aproximaciones numéricas no eran exactas, si bien simplificaban el trabajo, y no eran la única opción. Pero en modelos como lo de la presente sección no existe otra alternativa que el uso de métodos numéricos.

[11] Es más solo la presa puede extinguirse, para cierto conjunto de casos especiales.