Usando MAXIMA CAS para resolver ecuaciones diferenciales ordinarias*

Renato Álvarez Nodarse

Dpto. Análisis Matemático, Universidad de Sevilla.

4 de julio de 2010

`http://euler.us.es/~renato/`

Índice

1. ¿Por qué usar el ordenador en las clases de matemáticas? 2

2. Primeros pasos con Maxima 3
2.1. Empezando: primeras operaciones 3
2.2. Conceptos del cálculo diferencial e integral 8

3. Resolviendo EDOs con Maxima 14
3.1. Soluciones analíticas 14
3.2. Soluciones numéricas 16

4. El método de Euler con Maxima 18
4.1. El método de Euler 19
4.2. El método de Euler mejorado 23

5. Resolviendo sistemas de EDOs lineales 24
5.1. Resolviendo sistemas con MAXIMA 27
5.2. Resolviendo EDOs lineales de orden n 33

6. Un ejemplo usando series de potencias 35

Referencias básicas sobre EDOs 38

*A las notas les acompañan las sesiones en en entorno wxMAXIMA. Cada fichero comienza con un distintivo de la sección. Por ejemplo, el `s2` del fichero `s2-maxima-intro.wxm` indica que este fichero es la sesión usada en la sección 2. Así, `s4-metodo-euler.wxm` será el fichero usado en la sección 4, etc. Dichos ficheros se encuentran disponibles en la página web del autor.

1. ¿Por qué usar el ordenador en las clases de matemáticas?

Los computadores, como instrumentos de experimentación matemática, han seguido un lento camino desde que John von Neumann[1] los concibiera; Fermi, Pasta y Ulam[2] lo usaran por primera vez *experimentalmente* en los años 50, hasta el día de hoy, en que constituyen un elemento más de nuestro entorno cotidiano.

No obstante lo que que es deseable, aparte de los muchos usos lúdicos que puedan tener, es que aprendamos a utilizarlo como una herramienta esencial para resolver problemas y proyectos relacionados con las matemáticas y demás ciencias exactas. De esta forma se pueden, entre otras cosas, *comprobar* las predicciones analíticas (teóricas) mediante experimentos numéricos. Es por ello que es interesante aprender a trabajar con un programa de cálculo matemático. En relación a lo anterior se plantea el problema de qué software elegir. La elección de un software libre es, seguramente, la mejor opción ya que no tiene ningún coste adicional como, por ejemplo, MAXIMA (programa basado en LIPS de corte simbólico/numérico) u OCTAVE (escrito en C++, para cálculo numérico esencialmente), programas matemáticos con licencia GNU/GPL (y por tanto gratuitos y de distribución libre) accesibles por internet en

- `http://maxima.sourceforge.net` y
- `http://www.gnu.org/software/octave` ,

respectivamente[3].

En estas notas vamos a describir brevemente como funciona el programa de cálculo simbólico (y numérico) MAXIMA. La elección de MAXIMA no es arbitraria. En primer lugar hay que destacar que es un programa que permite tanto el cálculo *simbólico* (es decir, como si lo hiciésemos con lápiz y papel) como numérico; en segundo lugar, tiene una comunidad de usuarios muy numerosa, con foros de discusión, etc. y por último, pero no por ello menos importante, existen una gran cantidad de manuales de acceso libre muchos de ellos disponibles en la propia web del programa `http://maxima.sourceforge.net`, entre los que se encuentran los siguientes:[4]

1. Jerónimo Alaminos Prats, Camilo Aparicio del Prado, José Extremera Lizana, Pilar Muñoz Rivas y Armando R. Villena Muñoz, *Prácticas de ordenador con* WXMAXIMA, Granada
2. José Manuel Mira Ros, *Elementos para prácticas con* MAXIMA (Murcia, `http://webs.um.es/mira/maxima/manualico.html`)
3. Rafa Rodríguez Galván, MAXIMA *con* WXMAXIMA*: software libre en el aula de matemáticas* (Cadiz, 2007)
4. Mario Rodríguez Riotorto, *Primeros pasos en* MAXIMA (Ferrol, 2008)

[1] En realidad el primer ordenador *digital* fue el ENIAC (Electronic Numerical Integrator And Calculator) pero su programa estaba conectado al procesador y debía ser modificado manualmente, es decir si queríamos resolver un problema distinto había que cambiar las propias conexiones en el ordenador, i.e., cambios de programa implicaban cambios en la estructura de la máquina (*hardware*). Fue el polifacético John von Neumann quien propuso que se separara el programa (software) del hardware, por lo que se le considera el creador de los ordenadores modernos.

[2] A mediados de 1952 el físico Enrico Fermi (1901–1954) propuso resolver numéricamente con ayuda de la computadora MANIAC-I (Mathematical Analyzer Numerical Integrator And Calculator) de los Alamos un problema de un sistema con osciladores no lineales, para lo que se asoció con el especialista en computación John Pasta (1918—1984) y el Matemático Stanislaw Ulam (1909–1984). Aunque nunca se llegó a publicar el trabajo debido a la prematura muerte de Fermi (apareció como preprint titulado "Studies of nonlinear problems. I" y más tarde fue incluido en *Collected Papers of Enrico Fermi*, E. Segré (Ed.), University of Chicago Press (1965) p. 977-988, con una introducción de Ulam) fue un trabajo extremadamente influyente y marcó la fecha del nacimiento de la *matemática experimental* (para más detalle véase el artículo de M.A. Porter, N.J. Zabusky, B. Hu y D.K. Campbell, "Fermi, Pasta, Ulam y el nacimiento de la matemática experimental", *Revista Investigación y Ciencia* **395** Agosto de 2009, p. 72–80).

[3] Una lista bastante completa de las opciones se puede consultar en la wikipedia `http://es.wikipedia.org/wiki/Software_matematico`

[4] Conviene visitar también la web `http://wxmaxima.sourceforge.net/wiki/index.php/Tutorials` que continene además muchos manuales en inglés muy interesantes.

5. José Antonio Vallejo Rodríguez, *Manual de uso de* MAXIMA *y* WXMAXIMA *en asignaturas de cálculo diferencial* (México, 2008)

En estas notas vamos a describir brevemente como usar MAXIMA como complemento al curso de Ampliación de Análisis Matemático de la Diplomatura de Estadística de la Universidad de Sevilla (dicha asignatura tiene como objetivo, esencialmente, el estudio y resolución numérica de ecuaciones diferenciales ordinarias (EDOs) por lo que estas notas se puede usar como complemento a cualquier curso introductorio de EDOs). Por comodidad usaremos la interfaz gráfica WXMAXIMA (otra opción posible es XMAXIMA) que se puede obtener gratuitamente desde la página de MAXIMA o, si se usa LINUX, mediante el instalador de paquetes de la correspondiente distribución, en cuyo caso hay que asegurarse de instalar GNUPLOT y MAXIMA.

2. Primeros pasos con Maxima

2.1. Empezando: primeras operaciones

MAXIMA es un programa que funciona como una calculadora científica. Las operaciones aritméticas elementales son las habituales: + suma, − resta, ∗ multiplicación, / división, ^ potencias:[5]

```
(%i1) 2+2; 3-3; 2*3; 5/10; 3^3;
(%o1) 4
(%o2) 0
(%o3) 6
(%o4) 1/2
(%o5) 27
```

La notación que usa MAXIMA es `(%i`n`)` y `(%o`m`)` para indicar la *entrada* (inchar o input) n-ésima y la *salida* (outchar u output) m-ésima. Usando, por ejemplo, las órdenes `inchar: "input";` y `outchar: "output";` podemos cambiar las `(%i`n`)` y `(%o`m`)` por `(%input`n`)` y `(%output`m`)`, respectivamente.

Dado que MAXIMA es un programa de cálculo *simbólico*, trabaja con variables definidas por letras:

```
(%i6) e;
(%o6) e
```

Las funciones (comandos) tienen el argumento (o los argumentos) que van entre paréntesis, como por ejemplo el comando `float(x)` que nos da como resultado el valor numérico de la variable x. En este ejemplo e representa una variable que no tiene ningún argumento asignado:

```
(%i7) float(e);
(%o7) e
(%i8) pi;
(%o8) pi
(%i9) float(pi);
(%o9) pi
```

por lo que su respuesta es la propia variable. Nótese que si escribimos "pi", (o π, lo que se puede conseguir con la combinación "Esc" p "Esc") el resultado es pi. Para los valores numéricos de las constantes e (base de los logaritmos neperianos) o π MAXIMA usa una notación distinta:

[5] Normalmente para que MAXIMA calcule es necesario apretar al mismo tiempo las teclas "Schift" (mayúsculas) y "Enter".

```
(%i10) %e;
(%o10) %e
(%i11) float(%e);
(%o11) 2.718281828459045
```

Si no se le dice lo contrario, MAXIMA trabaja en precisión simple, i.e., con 16 dígitos. Dado que es un programa simbólico, podemos definirle con cuantas cifras queremos trabajar. Para ello hay que definir la variable `fpprec`. Para asignar valores a las variables dadas se usan los "dos puntos". Por ejemplo si escribimos `e: %e` habremos definido la variable e con el valor de la base de los logaritmos neperianos. Vamos a definir el grado de precisión en 50 cifras significativas:

```
(%i12) fpprec:50;
(%o12) 50
```

y lo comprobamos pidiendo a MAXIMA que nos de los valores de e y π con 50 cifras, para lo cual hay que usar el comando `bfloat`

```
(%i13) bfloat(%e); bfloat(%pi);
(%o13) 2.7182818284590452353602874713526624977572470937b0
(%o14) 3.1415926535897932384626433832795028841971693993751b0
```

Aquí, la notación $b0$ indica $\times 10^0$. Calculemos ahora e^7 y pidamos a MAXIMA que escriba los valores numéricos del mismo:

```
(%i15) float(%e^7);
(%o15) 1096.633158428459
(%i16) bfloat(%e^7);
(%o16) 1.0966331584284585992637202382881214324422191348336b3
(%i17) %e^7;
(%o17) %e^7
```

Así si escribimos `%e^7` la salida es simplemente e^7 pues MAXIMA, como hemos dicho, trabaja simbólicamente. Si usamos `float`, la salida es en precisión normal, y solo si usamos `bfloat` nos devuelve el resultado en la precisión definida previamente de 50 cifras.

El orden de realización de las operaciones es el habitual. Así en la expresión

```
(%i18) (2+3^2)^3*(5+2^2);
(%o18) 11979
```

primero calcula las potencias dentro de cada paréntesis, luego las sumas, luego las potencias externas y finalmente la multiplicación.

Como hemos mencionado, para definir los valores de las variables se usan los dos puntos. En la secuencia que aparece a continuación se definen la x y la y para que valgan 123 y 321, respectivamente, dejándose la z libre.

```
(%i19) x:123; y:321; x*y; x/y; x-y;
(%o19) 123
(%o20) 321
(%o21) 39483
(%o22) 41/107
(%o23) -198
(%i24) 123*321;
(%o24) 39483
(%i25) x;
(%o25) 123
(%i26) z=2;
```

```
(%o26) z=2
(%i27) z;
(%o27) z
(%i28) x;
(%o28) 123
(%i29) y;
(%o29) 321
(%i30) x*y;
(%o30) 39483
(%i31) x+z;
(%o31) z+123
```

Del resultado se sigue, que, por ejemplo, al realizar la división x/y, MAXIMA simplifica al máximo el resultado. Nótese también que el resultado de la multiplicación de $x * y$ es precisamente el valor de $123 * 321$. Como vemos en la salida `(%o26)` la expresión `z=2` no asigna el valor 2 a la variable `z`, ya que en realidad `z=2` es una ecuación (como veremos más adelante).

Es importante destacar, además, que hay que escribir el símbolo de cada multiplicación pues, si por ejemplo escribimos `x y` en vez de `x*y` obtenemos un mensaje de error

```
 (%i32)  x y;
Incorrect syntax: Y is not an infix operator
SpacexSpacey;
   ^
```

También es posible definir funciones. Hay múltiples formas en función de lo queramos hacer. La más sencilla es mediante la secuencia `:=`

```
(%i32) f(x):= x^2 -x + 1;
(%o32) f(x):=x^2-x+1
(%i33) f(%pi);
(%o33) %pi^2-%pi+1
(%i34) float(%);
(%o34) 7.728011747499565
(%i35) float(f(%pi));
(%o35) 7.728011747499565
```

Nótese que a no ser que pidamos a MAXIMA que trabaje numéricamente, sigue usando cálculo simbólico (ver el valor de $f(\pi)$ de la salida 33).

Otro detalle interesante a tener en cuenta es que MAXIMA contiene una ayuda completa que puede ser invocada desde la propia línea de comandos. Para ello preguntamos a MAXIMA con `??` delante del comando *desconocido*

```
(%i36) ??float;
 0: Functions and Variables for Floating Point
 1: bfloat  (Functions and Variables for Floating Point)
 2: bfloatp  (Functions and Variables for Floating Point)
 3: float  (Functions and Variables for Floating Point)
 4: float2bf  (Functions and Variables for Floating Point)

etc.

Enter space-separated numbers, 'all' or 'none':
```

La respuesta es una lista de opciones de funciones de MAXIMA que contienen las letras del comando y el programa se queda esperando que elijamos una opción. Al elegirla nos da una explicación detallada de la sintaxis y de lo que hace dicho comando y, en muchos casos, ejemplos de utilización:

```
Enter space-separated numbers, 'all' or 'none': 0;
10.1 Functions and Variables for Floating Point
===============================================
 -- Function: bffac (<expr>, <n>)
     Bigfloat version of the factorial (shifted gamma) function.
     The second argument is how many digits to retain and
     return, it's a good idea to request a couple of extra.
 -- Option variable: algepsilon
     Default value: 10^8
     'algepsilon' is used by 'algsys'.

etc.

(%o36) true
```

Si no existe ningún comando con ese nombre la respuesta es `false` (en caso contrario, al final de las explicaciones, la salida es `true`).

```
(%i37) ??renato;
(%o37) false
```

Otra de las interesantes opciones de MAXIMA es su potencia gráfica. Para hacer las gráficas MAXIMA usa GNUPLOT, un potente paquete gráfico GNU. El comando más sencillo de usar es `plot2d`. Ahora bien, para usar ese comando tenemos que ser muy cuidadosos. Si hacemos en nuestro caso

```
(%i39) wxplot2d([f(x)], [x,-5,5]);
Bad range: [123,-5,5].
Range must be of the form [variable,min,max]
 -- an error. To debug this try: debugmode(true);
```

obtenemos un error, que simplemente, leyendo nos indica que la variable x ya esta asignada. Eso se resuelve cambiando la variable por otra no usada o, usando una variable *muda*, es decir una variable que solo se use dentro del propio comando `wxplot`. Estas variables se definen colocando delante de la misma el signo `'`, por ejemplo, en vez de `x` usamos `'x`

```
(%i40) wxplot2d([f('x)], ['x,-5,5])$
```

que nos saca en el propio entorno de XWMAXIMA la gráfica. También podemos usar `plot` que saca el resultado en una pantalla aparte (de GNUPLOT). También podemos representar varias gráficas al mismo tiempo. Para ello creamos una *lista*[6] de funciones de la siguiente forma `[ f(x), g(x), ... , h(x) ]`. En el ejemplo representamos las funciones $f(x)$, $\sin(2x)$ y $\arctan(x)$

```
(%i41) wxplot2d([f('x),2*sin('x),atan(x)], ['x,-2,2])$
```

MAXIMA también dispone de una gran cantidad de funciones *elementales*, exponencial, logaritmo, trigonométricas, etc. Además las trata de forma simbólica como se puede ver en la secuencia que sigue. Nótese que la función `log` denota a los logaritmos neperianos (probar con la *función* `ln`).

```
(%i42) log(10);
(%o42) log(10)
(%i43) float(%);
(%o43) 2.302585092994046
(%i44) log(%e);
(%o44) 1
```

[6] Las listas no son más que expresiones de la forma `[objeto1,...,objetoN]`, donde `objeto1, ..., objetoN` son objetos cuales quiera (números, variables, gráficas, listas, ...).

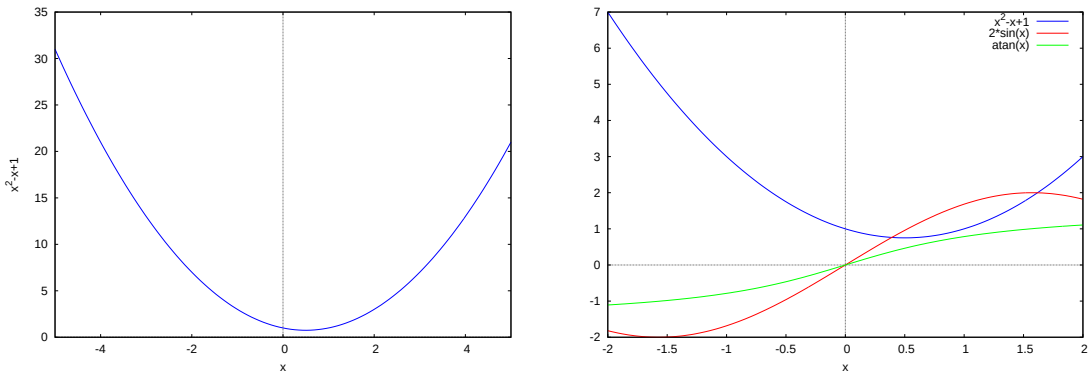

Figura 1: Los gráficos de la función de la salida `%o32` $f(x) = x^2 - x + 1$ (izquierda) y el de las tres funciones de la salida `%o41` (derecha)

También podemos calcular factoriales

```
(%i45) 15!;
(%o45) 1307674368000
```

En el próximo apartado se incluye una lista de algunas de las funciones más comunes que usa MAXIMA.

Otra de las bondades de MAXIMA es que cuenta con un gran repertorio de comandos de manipulación algebraica, basta pinchar en la pestaña "Simplificar" y veremos las muchas posibilidades. A modo de ejemplo factorizaremos el factorial anterior (que al ser un número, MAXIMA lo que hace es encontrar sus factores primos).

```
(%i46) factor(%);
(%o46) 2^11*3^6*5^3*7^2*11*13
```

Podemos factorizar expresiones algebraicas:

```
(%i47) factor(x^2+x+6);
(%o47) 2*3*2543
```

¡Ops! ¿qué es esto?

```
(%i48) x;
(%o48) 123
```

Claro, x estaba asignada al valor 123. Una opción es usar variables mudas

```
(%i49) factor('x^2 + 'x -6);
(%o49) (x-2)*(x+3)
```

o bien, limpiar el valor de una variable, para lo cual usamos el comando `kill`

```
(%i50) kill(x); x;
(%o51) done
```

Ahora ya podemos usar sin problemas los comandos de simplificación usando la variable x (como la simplificación racional)

```
(%i52) radcan((x^2-1)/(x-1));
(%o52) x+1
```

Las ecuaciones se definen en MAXIMA de forma *habitual*, mediante el signo igual, i.e., miembro izquierdo = miembro derecho, por ejemplo `x^2-4=0` define la ecuación $x^2 - 4 = 0$. Lo interesante es que, definida una ecuación, podemos resolverla *analíticamente* con el comando `solve`:

```
(%i53) solve(x^2-4=0,x);
(%o53) [x=-2,x=2]
```

Además, MAXIMA lo hace simbólicamente, como ya hemos mencionado más de una vez. Incluso da las soluciones complejas si es necesario

```
(%i54) solve(x^2-4=a,x);
(%o54 [x=-sqrt(a+4),x=sqrt(a+4)]
(%i55) solve(x^2=-1);
(%o55) [x=-%i,x=%i]

(%i56) solve(x^5=1,x);
(%o56) [x=%e^((2*%i*%pi)/5),x=%e^((4*%i*%pi)/5),
        x=%e^(-(4*%i*%pi)/5),x=%e^(-(2*%i*%pi)/5),x=1]
(%i60) rectform(%);
(%o60) [x=%i*sin((2*%pi)/5)+cos((2*%pi)/5),
                x=%i*sin((4*%pi)/5)+cos((4*%pi)/5),
                x=cos((4*%pi)/5)-%i*sin((4*%pi)/5),
                x=cos((2*%pi)/5)-%i*sin((2*%pi)/5),x=1]
```

Como ejercicio resolver la ecuación $x^n = 1$.

Si queremos limpiar todas las variables[7] de nuestra sesión debemos usar el comando `kill(all)`

```
(%i61) kill(all);
(%o0) done
```

Nótese que el número del output es 0.

2.2. Conceptos del cálculo diferencial e integral

Antes de comenzar a mostrar como MAXIMA trabaja con los conceptos del cálculo diferencial e integral vamos a incluir una lista con las funciones más comunes que tiene MAXIMA:

- `abs(exp)` valor absoluto o módulo de la expresión `exp`
- `entier(x)` parte entera de x
- `round(x)` redondeo de x
- `exp(x)` función exponencial de x
- `log(x)` logaritmo neperiano de x
- `max(x1, $x$2, ...), min(x1, x2, ...)` máximo y mínimo de los números $x_1, x_2, \ldots$
- `sign(exp)` signo de la expresión `exp`. Devuelve una de las respuestas siguientes: pos (positivo), neg (negativo), zero, pz (positivo o cero), nz (negativo o cero), pn (positivo o negativo), o pnz (positivo, negativo, o cero, i.e. no se sabe).
- `sqrt(x)` raíz cuadrada de x, i.e., *sqrtx*.

[7] Esto es muy recomendable cuando comenzemos a trabajar en un nuevo problema o queremos repetir los cálculos para asegurarnos que no hemos cometidos errores. Otra opción es ir a la pestaña "Maxima" y elegir "Reiniciar".

- `acos(x)` arco coseno de x
- `acosh(x)` arco coseno hiperbólico de x
- `acot(x)` arco cotangente de x
- `acoth(x)` arco cotangente hiperbólico de x
- `acsc(x)` arco cosecante de x
- `acsch(x)` arco cosecante hiperbólico de x
- `asec(x)` arco secante de x
- `asech(x)` arco secante hiperbólico de x
- `asin(x)` arco seno de x
- `asinh(x)` arco seno hiperbólico de x
- `atan(x)` arco tangente de x
- `atan2(y,x)` proporciona el valor de $\arctan(y/x)$ en el intervalo $[-\pi, \pi)$
- `atanh(x)` arco tangente hiperbólico de x
- `cos(x)` coseno de x
- `cosh(x)` coseno hiperbólico de x
- `cot(x)` cotangente de x
- `coth(x)` cotangente hiperbólica de x
- `csc(x)` cosecante de x
- `csch(x)` cosecante hiperbólica de x
- `sec(x)` secante de x
- `sech(x)` secante hiperbólica de x
- `sin(x)` seno de x
- `sinh(x)` seno hiperbólico de x
- `tan(x)` tangente de x
- `tanh(x)` tangente hiperbólica de x
- `gamma(x)` función gamma de Euler
- `beta(x,y)` función beta de Euler
- `erf(x)` la función error, i.e. $\frac{2}{\pi}\int_0^x e^{-u^2}du$
- `isqrt(x)` parte entera de la raíz cuadrada de x, que debe ser entero
- `binomial(x,y)` número combinatorio $\binom{n}{m} = \frac{n!}{m!(n-m)!}$

Veamos como MAXIMA trabaja con los conceptos del cálculo diferencial e integral. Ante todo, MAXIMA *sabe* calcular límites $\lim_{x\to x_0\pm} f(x)$, para ello usa el comando `limit` cuya sintaxis es

```
limit(f(x),x,x0,dirección)
```

```
(%i1) limit(sin(a*x)/x,x,0);
(%o1) a
```

Incluso aquellos que dan infinito

```
(%i2) limit(log(x),x,0);
(%o2) infinity
(%i3) limit(log(x),x,0,plus);
(%o3) -inf
(%i4) limit(log(x),x,0,minus);
(%o4) infinity
```

Debemos destacar que `infinity` es entendido por MAXIMA como el infinito complejo, mientras que `inf` lo entiende como el $+\infty$ real. Gracias a esto podemos calcular límites infinitos

```
(%i5) limit((1+2/x)^(x^(1/3)),x,inf);
(%o5) 1
```

Como ejercicio calcula los límites $\lim_{x\to 0}\log(3x)/x$, $\lim_{n\to\infty}(1+2/n)^{3/n}$ y $\lim_{x\to 2}\frac{x^2-4}{x-2}$.

El comando `diff` permite calcular derivadas de cualquier orden de una función. Su sintaxis es

```
diff(f(x),x,k)
```

donde `f(x)` es la función a la que le vamos a calcular la derivada `k`-ésima respecto a la variable `x`:

```
(%i6) diff(sin(x^2+2),x);
(%o6) 2*x*cos(x^2+2)
(%i7) diff(x^x, x,3);
(%o7) x^(x-1)*(log(x)+(x-1)/x)+x^x*(log(x)+1)^3
           +2*x^(x-1)*(log(x)+1)(%i8) (sin(x))^x;
(%o8) sin(x)^x
(%i9) diff(sin(x)^x, x);
(%o9) sin(x)^x*(log(sin(x))+(x*cos(x))/sin(x))
```

Además, como vemos también funciona para funciones definidas por el usuario

```
(%i10) g(x):= sin(x*log(1+x^2));
(%o10) g(x):=sin(x*log(1+x^2))
(%i11) diff(g(x),x,1);
(%o11) (log(x^2+1)+(2*x^2)/(x^2+1))*cos(x*log(x^2+1))
```

MAXIMA recuerda todas las salidas. Para invocar la última salida bastante usar el `%`. Por ejemplo, la orden `ratsimp(%)` simplifica la salida anterior (que en nuestro ejemplo era la derivada de $g(x)$

```
(%i12) ratsimp(%);
(%o12) (((x^2+1)*log(x^2+1)+2*x^2)*cos(x*log(x^2+1)))/(x^2+1)
(%i13) radcan(%);
(%o13) (((x^2+1)*log(x^2+1)+2*x^2)*cos(x*log(x^2+1)))/(x^2+1)
(%i14) factor(%);
(%o14) ((x^2*log(x^2+1)+log(x^2+1)+2*x^2)*cos(x*log(x^2+1)))/(x^2+1)
(%i15) expand(%);
(%o15) (x^2*log(x^2+1)*cos(x*log(x^2+1)))/(x^2+1)+
              (log(x^2+1)*cos(x*log(x^2+1)))/(x^2+1)+
              (2*x^2*cos(x*log(x^2+1)))/(x^2+1)
(%i16) trigsimp(%);
(%o16) (((x^2+1)*log(x^2+1)+2*x^2)*cos(x*log(x^2+1)))/(x^2+1)
```

Podemos recuperar la n-ésima salida simplemente usando `%n`.

El comando `integrate(f(x),x)` calcula una primitiva de la función `f(x)`

```
(%i17) integrate(sin(2*x), x);
(%o17) -cos(2*x)/2
```

Vamos ahora a aprender como definir funciones a partir de las salidas de MAXIMA. Por ejemplo, definamos la función int(x) como la salida de la orden `integrate`

```
(%i18) int(x):=integrate(x+2/(x -3), x);
(%o18) int(x):=integrate(x+2/(x-3),x)
(%i19) int(x);
(%o19) 2*log(x-3)+x^2/2
```

En efecto, vemos que podemos dibujarla

```
(%i20) wxplot2d([int('x)], ['x,2,7])$
plot2d: expression evaluates to non-numeric value somewhere
in plotting range.
(%t20)  << Graphics >>
```

Nótese que MAXIMA nos previene que hay valores para los cuales la función int(x) no está definida. Además podemos comprobar que la derivada de int(x) es precisamente nuestra función de partida:

```
(%i21) diff(int(x),x,1);
(%o21) x+2/(x-3)
```

Sin embargo hay que ser cuidadoso a la hora de trabajar con las funciones definidas a partir de las salidas de MAXIMA. El siguiente ejemplo[8] muestra que *no siempre* MAXIMA *funciona bien*:

```
(%i22) vnum1:integrate(x+2/(x -3), x, 0,1);
(%o22) -2*log(3)+2*log(2)+1/2
(%i23) int(1)-int(0);
Attempt to integrate wrt a number: 1
#0: int(x=1)
 -- an error. To debug this try: debugmode(true);
(%i24) int(1);
Attempt to integrate wrt a number: 1
#0: int(x=1)
 -- an error. To debug this try: debugmode(true);
```

La razón es que MAXIMA al calcular `int(1)` intenta sustituir la x por 1 en la expresión `integrate(x+2/(x-3),x,0,1)`, lo que causa el error, es por ello que hay que proceder de forma diferente, concretamente usando el comando `define` de la siguiente forma[9]:

```
(%i25) define(intprim(x),integrate(x+2/(x-3),x));
(%o25) intprim(x):=2*log(x-3)+x^2/2
(%i26) intprim(1);
(%o26) 2*log(-2)+1/2
(%i27) intprim(1)-intprim(0);
(%o27) 2*log(-2)-2*log(-3)+1/2
```

[8] El comando `integrate(expr, x, a, b)` calcula la integral definida de la expresión `expr` (usualmente una función) en la variable `x`, e.g. `integrate(f(x), x, a, b)` calcula $\int_a^b f(x)dx$.

[9] También valdría `define(intprim(x),int(x))`

lo que nos permite, efectivamente, usar la salida de MAXIMA para definir la nueva función `intprim`, que ahora si está bien definida. Podemos finalmente comprobar que `intprim` da, efectivamente, una primitiva de nuestra función

```
(%i28) %-vnum1;
(%o28) 2*log(3)-2*log(2)+2*log(-2)-2*log(-3)
(%i29) rectform(%);
(%o29) 0
```

Hemos de destacar que en la expresión anterior aparece `log(-2)` y `log(-3)` lo que en principio no tiene sentido ya que el logaritmo no esta definido para los negativos a no ser que ... sea el logaritmo complejo,

```
(%i30) log(-2); rectform(%);
(%o30) log(-2)
(%o31) log(2)+%i*%pi
```

que como vemos es precisamente el caso.

Otra opción interesante que tiene MAXIMA es que se pueden hacer suposiciones sobre las variables. Por ejemplo si pedimos a MAXIMA que calcule la integral

```
(%i32) integrate (x^a*exp(-x), x, 0, inf);
Is  a+1  positive, negative, or zero?p;
(%o32) gamma(a+1)
```

este nos pregunta si la a es positiva, negativa, etc. Al responder nos da la solución (¿preguntar a MAXIMA qué significa `gamma`?) Si sabemos que, por ejemplo, la a es positiva, etc. podemos usar el comando `assume` que indica que al calcular la integral la a es un número mayor que uno.

```
(%i33) assume (a > 1)$
       integrate (x^a*exp(-x), x, 0, inf);
(%o34) gamma(a+1)
```

Veamos otros dos comandos interesantes. El primero es el comando `taylor` que permite calcular el polinomio de Taylor de orden n alrededor del punto x_0 de la función $f(x)$. Formalmente MAXIMA lo que hace es calcular el polinomio

$$P_n(x, x_0) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2!}f''(x_0)(x - x_0)^2 + \cdots + \frac{1}{n!}f^{(n)}(x_0)(x - x_0)^n.$$

La sintaxis es `taylor(f(x),x,x0,n)`

```
(%i42) taylor(%e^x,x,0,5);
(%o42)/T/ 1+x+x^2/2+x^3/6+x^4/24+x^5/120+...
```

El segundo `powerseries` permite a MAXIMA tratar con series infinitas. En primer lugar la orden `powerseries(f(x),x,x0)` devuelve la serie de potencias de f alrededor del punto x_0.

```
(%i43) powerseries(%e^x,x,0);sere:niceindices(%);
(%o43) sum(x^i1/i1!,i1,0,inf)
(%o44) sum(x^i/i!,i,0,inf)
```

Lo interesante es que MAXIMA es capaz de integrar y diferenciar las series de potencias

```
(%i45) ser:niceindices(powerseries(1/(1+x),x,0));
(%o45) sum((-1)^i*x^i,i,0,inf)
(%i46) integrate(ser,x);
(%o46) sum(((-1)^i*x^(i+1))/(i+1),i,0,inf)
(%i47) diff(%,x);
(%o47) sum((-1)^i*x^i,i,0,inf)
```

Para terminar esta introducción debemos decir que MAXIMA también dibuja gráficas 3D con el comando `plot3d` cuya sintaxis es similar a la de `plot2d`

```
(%i35) kill(f,x,y)$
       f(x,y):= sin(x) + cos(y);
       wxplot3d(f(x,y), [x,-5,5], [y,-5,5])$
```

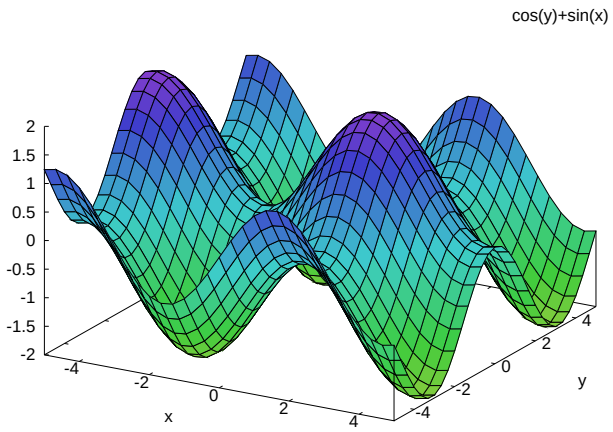

Figura 2: La gráfica de la función $f(x, y) = \sin(x) + \cos(x)$.

Para que MAXIMA reconozca el directorio local de trabajo (lo que es conveniente para importar y exportar ficheros) es conveniente definir la variable `file_search_maxima`. En este ejemplo (con LINUX) el directorio de búsqueda es `/home/renato/`. La forma más sencilla de definirlo es mediante la opción "Añadir a ruta" en la pestaña "Maxima" del menú del programa:

```
(%i38) file_search_maxima : cons(sconcat(
         "/home/renato/###.{lisp,mac,mc}"), file_search_maxima)$
```

Otra opción muy útil de Máxima es la de exportar los gráficos como ficheros `eps` (postcript encapsulado) o `jpg` (jpeg). En el primer caso podemos usar la orden

```
(%i39) plot3d(f(x,y), [x,-5,5], [y,-5,5],
            [psfile, "/home/renato/nuevos/gra3dnew.eps"])$
(%i40) plot3d(f(x,y), [x,-5,5], [y,-5,5], [psfile, "gra3d.eps"])$
```

que nos genera en el directorio `/home/renato/nuevos/`, el primero, o en el directorio por defecto (en este caso el `/home/renato/`). Si queremos otros formatos como `jpg, png, eps, eps_color, pdf, gif, animated_gif`, etc., hay que usar las opciones de GNUPLOT. En este ejemplo generamos un gráfico en formato `jpg`

```
(%i41) plot3d(f(x,y), [x,-5,5], [y,-5,5],
[gnuplot_preamble, "set terminal jpeg; set output 'grafico3d.jpg'"])
```

Finalmente, para grabar la sesión, podemos usar la opción "Guardar" en la pestaña "Archivo" de `wxMaxima`.

3. Resolviendo EDOs con Maxima

3.1. Soluciones analíticas

MAXIMA cuenta con varios comandos para resolver analíticamente las ecuaciones diferenciales ordinarias (EDOs).

Comencemos comentando la orden `desolve` que resuelve sistemas de EDOs mediante la transformada de Laplace. Su sintaxis es

```
desolve([eq1,eq2,...,eqn],[y1,y2,...,yn])
```

donde `eq1,..., eqn` denota las ecuaciones diferenciales (lineales) e `y1,...,yn` las funciones desconocidas. Para definir la ecuación debemos usar la sintaxis correcta. Por ejemplo la ecuación $y'(x) = y$ la escribimos de la siguiente forma:

```
(%i1) edo:diff(y(x),x,1) = y(x);
(%o1) y(x)=y(0)*%e^x
```

Entonces hacemos

```
(%i2) desolve(edo,y(x));
(%o2) y(x)=y(0)*%e^x
```

Si ahora queremos resolver el problema de valores iniciales (PVI) $y' = f(x, y)$, $y(x_0) = y_0$, usamos el comando `atvalue` para indicar el valor de la función y en $x = x_0$:

```
(%i3) atvalue(y(x),x=0,1);
       desolve(edo,y(x));
(%o3) 1
(%o4) y(x)=%e^x
```

Nótese que tenemos que especificar para las funciones desconocidas sus correspondientes variables pues en caso contrario MAXIMA da error:

```
(%i5) kill(y)$
       edo1:diff(y,x,1) = y;
       desolve(edo,y);
(%o6) 0=y
       length: argument cannot be a symbol; found y
       -- an error. To debug this try: debugmode(true);
```

puesto que entiende que y es una constante y por tanto $y' = 0$.

Resolvamos ahora la ecuación $y' = -1/2y + \sin(x)$ con la condición $y(0) = 1$ y dibujemos su solución. Si usamos sólo el comando `desolve` obtenemos la solución general de la ODE pero no la solución del PVI

```
(%i8) kill(y)$
      edo2:diff(y(x),x,1) = -1/2*y(x)+sin(x);
      desolve(edo2,y(x))$
      expand(%);
(%o9) 'diff(y(x),x,1)=sin(x)-y(x)/2
(%o11) y(x)=(2*sin(x))/5-(4*cos(x))/5+y(0)*%e^(-x/2)+(4*%e^(-x/2))/5
```

así que usamos la combinación con `atvalue`

```
(%i12) atvalue(y(x),x=0,1);
       desolve(edo2,y(x));
(%o12) 1
(%o13) y(x)=(2*sin(x))/5-(4*cos(x))/5+(9*%e^(-x/2))/5
```

Para definir la función solución y luego trabajar con ella usamos, como ya hemos visto, la orden `define`

```
(%i14) define(soledo2(x),second(%));
(%o14) soledo2(x):=(2*sin(x))/5-(4*cos(x))/5+(9*%e^(-x/2))/5
```

Además hemos usado un comando muy útil `second` que, dada una ecuación `miembro izq. = miembro der.`, extrae el miembro de la derecha de la misma[10] Finalmente, dibujamos la solución:

```
(%i15) wxplot2d(soledo2(x),[x,0,18])$
(%t15)  << Graphics >>
```

Este comando no siempre funciona. Si lo aplicamos a la ecuación separable $y' = 1 + y^2$, nos da

```
(%i16) edo3:diff(w(x),x,1) = 1+(w(x))^2;
       desolve(edo3,w(x));
(%o16) 'diff(w(x),x,1)=w(x)^2+1
(%o17) w(x)=ilt(((laplace(w(x)^2,x,g36165)+
             w(0))*g36165+1)/g36165^2,g36165,x)
```

donde `ilt` significa trasformada inversa de Laplace (en sus siglas inglesas *inverse Laplace transform*), pero no nos permite obtener ningún valor de la misma. En este caso es conveniente utilizar otro comando: el comando `ode2` cuya sintaxis es

```
ode2(eqn, variable dependiente, variable independiente)
```

y que resuelve EDOs de primer y segundo orden intentando reconocer alguna de las ecuaciones tipo más usuales: lineales, separables, de Bernoulli, etc.

Por ejemplo, resolvamos la EDO $z' = -z + x$:

```
(%i18) 'diff(z,x)=x-z;
        ode2('diff(z,x)=x-z,z,x)$
        expand(%);
(%o18) 'diff(z,x,1)=x-z
(%o20) z=%c*%e^(-x)+x-1
```

Si queremos resolver el PVI $z' = -z + x$, $y(0) = 1$ hay que usar el comando `ic1` cuya sintaxis es

```
ic1(solución, valor de x, valor de y)
```

donde `solución` es la solución general que da el comando `ode2` y el `valor de x` y el `valor de y`, son los valores que toma la y cuando $x = x_0$, i.e., los valores iniciales. Así tenemos

```
(%i21) expand(ic1(%,x=1,z=2));
(%o21) z=2*%e^(1-x)+x-1
(%i22) define(s1(x),second(%));
(%o22) s1(x):=2*%e^(1-x)+x-1
(%i23) wxplot2d(s1(x),[x,1,5])$
(%t23)  << Graphics >>
```

Si lo aplicamos a nuestro PVI $y' = 1 + y^2$, $y(0) = 0$ tenemos

[10] Si queremos el izquierdo podemos usar el comando `first`.

```
(%i23) kill(w)$
       'diff(w,x)=1+w^2;
        ode2('diff(w,x)=1+w^2,w,x)$
        sol:expand(%);
        expand(ic1(sol,x=0,w=0));
(%o24) 'diff(w,x,1)=w^2+1
(%o26) atan(w)=x+%c
(%o27) atan(w)=x
(%i28) sol:solve(%,w);
(%o28) [w=tan(x)]
```

La última salida (entre "`[ ]`")es una lista. Las listas son especialmente importantes en MAXIMA aunque por ahora sólo nos interesa saber sacar un elemento dado lo que se hace con el comando `part(lista,nº del elemento)` (o bien, en nuestro caso escribiendo `sol[1]`) así que hacemos

```
(%i29) part(%,1);
(%o29) w=tan(x)
(%i30) define(solw(x),second(%));
(%o30) solw(x):=tan(x)
```

3.2. Soluciones numéricas

Esta orden `ode2` no siempre funciona como es el caso de la EDO $z' = x - \sin z$, en cuyo caso la salida es "false"

```
(%i31) ode2('diff(z,x)=x-sin(z),z,x);
(%o31) false
```

En ese caso hay que usar algún método numérico. Más adelante explicaremos algunos métodos muy sencillos, no obstante aquí usaremos el comando `runge1` que permite resolver numéricamente PVI de primer orden del tipo $y' = f(x, y)$, $y(x_0) = y_0$ con MAXIMA usando en método de Runge-Kutta. Para ello lo primero que tenemos que hacer es cargar el paquete numérico `diffeq` y luego invocar dicho comando. La sintaxis de `runge1` es la siguiente

```
runge(f, x0, x1, h, y0)
```

donde f es la función $f(x, y)$ de la ecuación $y' = f(x, y)$, `x0` y `x1` los valores inicial, x_0, y final, x_1, de la variable independiente, respectivamente, `h` es la la longitud (o paso) de los subintervalos e `y0` es el valor inicial y_0 que toma y en x_0. El resultado es una lista que a su vez contiene tres listas: la primera contiene las abscisas x, la segunda las ordenadas y y tercera las correspondientes derivadas y'.

Como ejemplo consideremos el PVI $y' = 1 + y$, $y'(0) = 1$. Ante todo limpiaremos todas las variables y cargaremos el paquete `diffeq`

```
(%i32) kill(all);
(%o0) done
(%i1) load(diffeq);
(%o1) /usr/share/maxima/5.20.1/share/numeric/diffeq.mac
```

A continuación definimos la función f, y el paso h, para, a continuación, invocar la orden `runge1`

```
(%i2) f(x,y):=1+y; h:1/20;
(%o2) f(x,y):=1+y
(%o3) 1/20(%i4) solnum:runge1(f,0,1,h,1)
(%o4) [[0.0,0.05,...,0.95],[1.0,1.102104166666667,...,4.150987618241528],
```

```
       [2.0,2.102104166666667,...,5.150987618241528]]
(%i5) wxplot2d([discrete,solnum[1],solnum[2]])$
(%t5)  << Graphics >>
```

Como esta ecuación es exactamente resoluble podemos comparar sus gráficas. Para ello primero usamos `ode2` e `ice1` para resolver analíticamente el PVI:

```
(%i6) ode2('diff(w,x)=1+w,w,x)$
       sol:expand(%);
       expand(ic1(sol,x=0,w=1));
       define(solw(x),second(%));
(%o7) w=%c*%e^x-1
(%o8) w=2*%e^x-1
(%o9) solw(x):=2*%e^x-1
```

Y ahora dibujamos ambas gráficas

```
(%i10) wxplot2d([[discrete,solnum[1],solnum[2]],solw(x)],[x,0,1])$
(%t10)  << Graphics >>
```

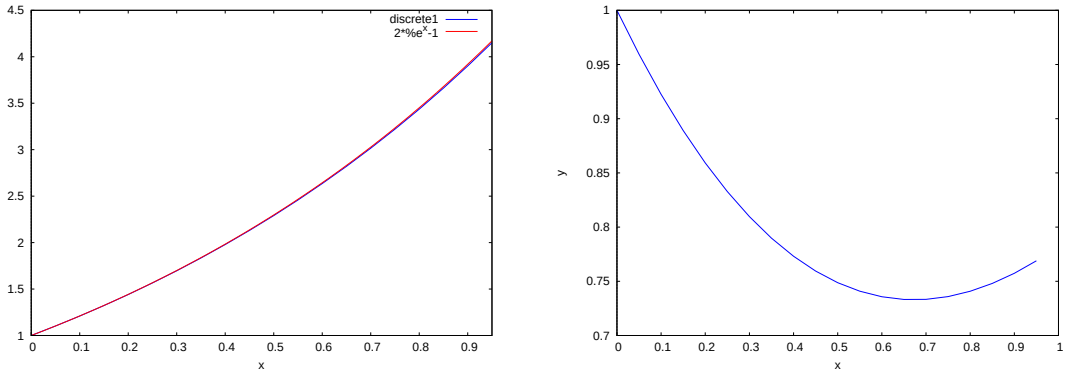

Figura 3: Comparación de la solución numérica y analítica del PVI $y' = 1 + y$, $y(0) = 1$ (izquierda) y $y' = x - \sin(y)$, $y(0) = 1$ (derecha).

Finalmente resolvemos numéricamente el PVI $y' = x - \sin(y)$, $y(0) = 1$.

```
(%i11) g(x,y):=x-sin(y);
       sol2:runge1(g,0,1,1/20,1);
       wxplot2d([discrete,sol2[1],sol2[2]])$
(%o11) g(x,y):=x-sin(y)
(%o12) [[0.0,0.05,..., 0.9,0.95],
       [1.0,0.95944094204897, ... ,0.75740012409521,0.7689229050892],
       [-0.8414709848079,-0.76887080939197, ... ,0.25463842295662]]
(%t13)  << Graphics >>
```

Antes de continuar tenemos que comentar que hay un comando alternativo para resolver numéricamente una ecuación diferencial pero que además es aplicable a sistemas de ecuaciones de primer orden lo cual nos será de utilidad más adelante. Se trata de comando `rk` del paquete `dynamics` que no es más que otra implementación de un método de Runge-Kutta. Su sintaxis, para el caso del PVI $y' = f(x, y)$, $y(x_0) = y_0$ es la siguiente

```
rk(f,y,y0,[x,x0,x1,h])
```

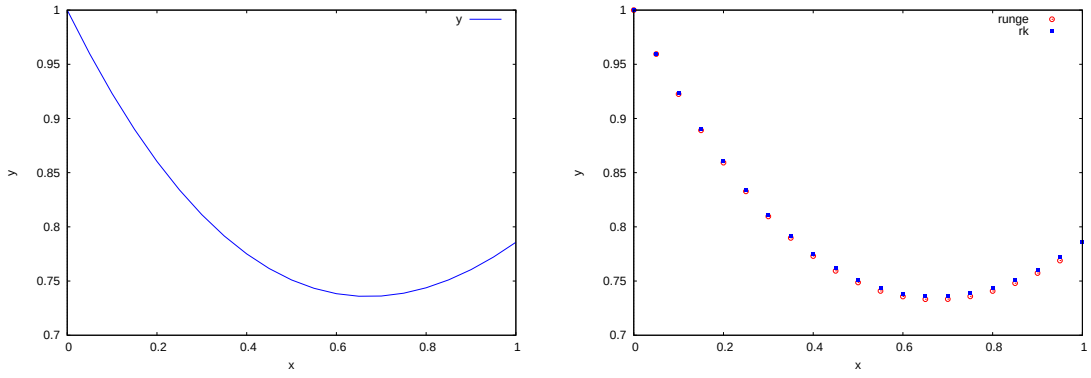

Figura 4: A la izquierda vemos la solución numérica usando el comando `rk` mientras que a la derecha están representados los resultados usando ambos métodos para el PVI $y' = x - \sin(y)$, $y(0) = 1$

donde f es la función $f(x, y)$ de la ecuación $y' = f(x, y)$, `x0` y `x1` los valores inicial, x_0, y final, x_1, de la variable independiente, respectivamente, `h` es la la longitud de los subintervalos e `y0` es el valor inicial y_0 que toma y en x_0. El resultado es una lista con los pares $[x, y]$ de las abscisas x y las ordenadas y.

Así tenemos la secuencia de órdenes

```
(%i15) load(dynamics)$
(%i16) h:1/20;kill(x,y);
       numsolrk:rk(x-sin(y),y,1,[x,0,1,h])$
(%o16) 1/20
(%o17) done
(%i19) numsolrk;
(%o19) [[0,1],[0.05,0.95973997169251],[0.1,0.92308155305544], ...
       [0.95,0.77210758398484],[1.0,0.78573816934072]]
```

La salida de `rk` es justo una lista que entiende perfectamente el comando `plot2d` por lo que podemos dibujar la solución y comparar ambas salidas numéricas.

```
(%i20) wxplot2d([discrete,numsolrk],[legend, "y"],
                [xlabel, "x"], [ylabel, "y"],[color,blue])$
(%t20)  << Graphics >>
(%i21) wxplot2d([[discrete,sol2[1],sol2[2]],[discrete,numsolrk]],
       [x,0,1],[style,[points,2,2,2],[points,1,1,6]],
       [xlabel,"x"],[ylabel,"y"],[legend,"runge","rk"])$
(%t21)  << Graphics >>
```

4. El método de Euler con Maxima

Como ya hemos visto son pocas las EDOs que se pueden resolver analíticamente, es por ello que se necesita de métodos fiables para obtener la solución de una EDO numéricamente. Supongamos que queremos resolver el problema de valores iniciales

$$\frac{d\,y(x)}{dx} = f(x, y), \qquad y(x_0) = y_0. \tag{1}$$

Obviamente usando un ordenador sólo podremos resolver el problema en un intervalo acotado, digamos $[x_0, x_0 + l]$. Para ello vamos a dividir el intervalo en N subintervalos $[x_0, x_1] \cup [x_1, x_2] \cup \cdots \cup [x_{N-1}, x_N]$, $x_N = x_0 + l$. Supongamos que hemos encontrado los valores de y en los puntos $x_0, x_1, \ldots, x_N$, que denotaremos por $y_0, y_1, \ldots, y_N$. Entonces, para encontrar una solución aproximada $\widehat{y}(x)$ podemos unir los puntos (x_i, y_i), $i = 0, 1, \ldots, N$ mediante líneas rectas (ver figura 5). Es evidente que si el valor y_i es bastante cercano al valor real $y(x_i)$ para todos los $i = 0, 1, \ldots, N$, entonces, al ser $\widehat{y}$ e y funciones continuas, la solución aproximada $\widehat{y}(x)$ estará "*muy cercana*" a la solución real $y(x)$ en cada uno de los intervalos $[x_i, x_{i+1}]$.

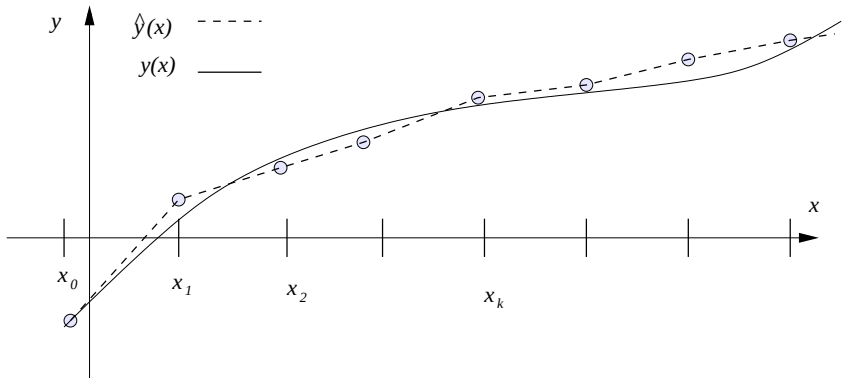

Figura 5: Construcción de un esquema numérico.

Vamos a usar por simplicidad intervalos iguales, es decir, vamos a escoger los *nodos* x_i equidistantes. Lo anterior se conoce en la teoría de métodos numéricos como una red equiespaciada o uniforme de paso $h = l/N$. Así pues tendremos las siguientes ecuaciones $x_k = x_0 + kh = x_0 + k\left(\frac{l}{N}\right)$, $k = 0, 1, \ldots, N$, $x_{k+1} = x_k + h$. Obviamente la única información que tenemos para calcular los valores y_i es la EDO que satisface nuestra incógnita $y(x)$. ¿Cómo encontrar entonces los valores y_i? La idea es como sigue:

1. Usando la EDO y la condición inicial calculamos el valor de y_1 en el punto $x_1 = x_0 + h$
2. A continuación usando los valores de y_0 e y_1 calculamos el valor aproximado y_2 de $y(x)$ en x_2, y así sucesivamente.
3. Conocido los valores y_0, y_1, …, y_k encontramos el valor y_{k+1} de $y(x)$ en x_{k+1}.

4.1. El método de Euler

Entre las muchas posibilidades para resolver el problema de encontrar en valor de $y(x_{k+1})$ conocidos los valores anteriores $y(x_j)$, $j = 0, \ldots, k$, podemos optar, por ejemplo, por usar el teorema de Taylor

$$y(x_{k+1}) = y(x_k + h) = y(x_k) + y'(x_k)h + \frac{y''(x_k)}{2!}h^2 + \cdots . \tag{2}$$

Como $y'(x_k) = f(x_k, y(x_k))$, entonces

$$\begin{aligned} y''(x_k) = & \left.\frac{d}{dx}\left(\frac{dy}{dx}\right)\right|_{x=x_k} = \left.\frac{d}{dx}\left(f(x, y(x))\right)\right|_{x=x_k} = \left.\frac{\partial f(x,y)}{\partial x} + \frac{\partial f(x,y)}{\partial y}\frac{\partial y}{\partial x}\right|_{(x,y)=(x_k,y(x_k)} \\ = & \left.\frac{\partial f(x,y)}{\partial x} + f(x,y)\frac{\partial f(x,y)}{\partial y}\right|_{(x,y)=(x_k,y(x_k)} . \end{aligned}$$

Luego, (2) nos da

$$y(x_{k+1}) = y(x_k) + hf(x_k, y(x_k)) + \left[\frac{\partial f}{\partial x}(x_k, y(x_k)) + f(x_k, y(x_k))\frac{\partial f}{\partial y}(x_k, y(x_k))\right]\frac{h^2}{2!} + \cdots .$$

La aproximación más sencilla es por tanto cuando nos quedamos en la serie anterior con el término de primer orden, o sea, cuando tenemos el esquema númerico

$$y_1 = y_0 + hf(x_0, y_0), \quad y_2 = y_1 + hf(x_1, y_1), \quad \ldots, \quad y_{k+1} = y_k + hf(x_k, y_k), \tag{3}$$

donde obviamente $y_0 = y(x_0)$.

El esquema anterior se conoce por el nombre de *esquema o método de Euler* y es, quizá, el método más sencillo para resolver númericamente una EDO de primer orden. Nótese que dicho esquema necesita en cada paso del valor $y(x_k)$, por tanto cuanto más cercano sea el valor y_k calculado del $y(x_k)$ real más preciso será el método. Obviamente en cada paso arrastramos el error del cáculo del paso anterior. En efecto, para calcular y_1 usamos el valor real y_0 pero cuando calculamos y_2, sustituimos el valor exacto $y(x_1)$ desconocido por su valor aproximado y_1, para calcular y_3 sustituimos el valor $y(x_2)$ por su valor aproximado y_2, y así sucesivamente.

Veamos algunos ejemplos.

Comenzaremos con una ecuación que sepamos resolver exactamente. Por ejemplo, estudiemos el problema de valores iniciales

$$y' + y = x, \qquad y(0) = 1, \quad x \in [0, 1],$$

cuya solución exacta es $y(x) = 2e^{-x} - 1 + x$. Escogeremos una discretización equidistante con paso $h = 1/20$ (20 subintervalos iguales).

Vamos a programarlo con MAXIMA. Para ello usaremos la notación `x[n]` que es esencialmente la forma de definir sucesiones con MAXIMA.

Comenzaremos definiendo el intervalo $[x_0, x_n]$, cuya longitud denotaremos por $l = x_n - x_0$ y que en nuestro caso será igual a 1.

```
 (%i1) x[0]:0;
       l:1;Nu:20;h:l/Nu;
       x[n]:=x[0]+n*h;
(%o1) 0
(%o2) 1
(%o3) 20
(%o4) 1/20
(%o5) x[n]:=x[0]+n*h
```

A continuación limpiaremos la variable `f` que definirá nuestra función f del PVI (1)

```
 (%i6) kill(f)$ y[0]:1;
         define(f[x,y],-y+x);
(%o7) 1
(%o8) f[x,y]:=x-y
```

Ya estamos listos para pedirle a MAXIMA que encuentre nuestra solución numérica. Para ello le pediremos que calcule los valores de $y(x_k)$ mediante la fórmula (3) y luego crearemos una lista que podamos representar gráficamente. Para crear la lista usamos el comando `makelist` cuya sintaxis es

```
makelist( expr , k , ki , kf )
```

que lo que hace general una lista donde la entrada k-ésima es el resultado de evaluar la expresión `expr` para cada `k` desde `k0` inicial hasta `kf` final

```
(%i9) y[k]:=float(y[k-1]+h*f[x[k-1],y[k-1]]);
          sol:makelist(float([x[k],y[k]]),k,0,Nu);
(%o9) y[k]:=float(y[k-1]+h*f[x[k-1],y[k-1]])
(%o10) [[0.0,1.0],[0.05,0.95],[0.1,0.905],[0.15,0.86475], ...,
              [0.95,0.70470720507062],[1.0,0.71697184481708]]
```

Los resultados están escritos en la tabla 1 o dibujados en la gráfica 6 (izquierda). Para dibujarlos hemos usado el comando `plot2d`

```
(%i11) wxplot2d([[discrete,sol]],[style, [points,2,2]],
                    [legend,"y(x)"],[xlabel,"x"], [ylabel,"y"])$
(%t11)  << Graphics >>
```

k	x_k	$\widehat{y}(x_k)$	$y(x_k)$	$\widehat{y}(x_k) - y(x_k)$
0	0	1.	1.	0
1.	0.05	0.95	0.952459	-0.00245885
2.	0.1	0.905	0.909675	-0.00467484
3.	0.15	0.86475	0.871416	-0.00666595
4.	0.2	0.829012	0.837462	-0.00844901
5.	0.25	0.797562	0.807602	-0.0100397
6.	0.3	0.770184	0.781636	-0.0114527
7.	0.35	0.746675	0.759376	-0.0127016
8.	0.4	0.726841	0.74064	-0.0137992
9.	0.45	0.710499	0.725256	-0.0147575
10.	0.5	0.697474	0.713061	-0.0155874
11.	0.55	0.6876	0.7039	-0.0162994
12.	0.6	0.68072	0.697623	-0.0169031
13.	0.65	0.676684	0.694092	-0.0174074
14.	0.7	0.67535	0.693171	-0.0178206
15.	0.75	0.676582	0.694733	-0.0181506
16.	0.8	0.680253	0.698658	-0.0184046
17.	0.85	0.686241	0.70483	-0.0185892
18.	0.9	0.694429	0.713139	-0.0187107
19.	0.95	0.704707	0.723482	-0.0187748
20.	1.	0.716972	0.735759	-0.018787

Cuadro 1: Solución númerica de la ecuación $y' + y = x$, $y(0) = 1$ en el intervalo $[0, 1]$ para $N = 20$ según el esquema de Euler.

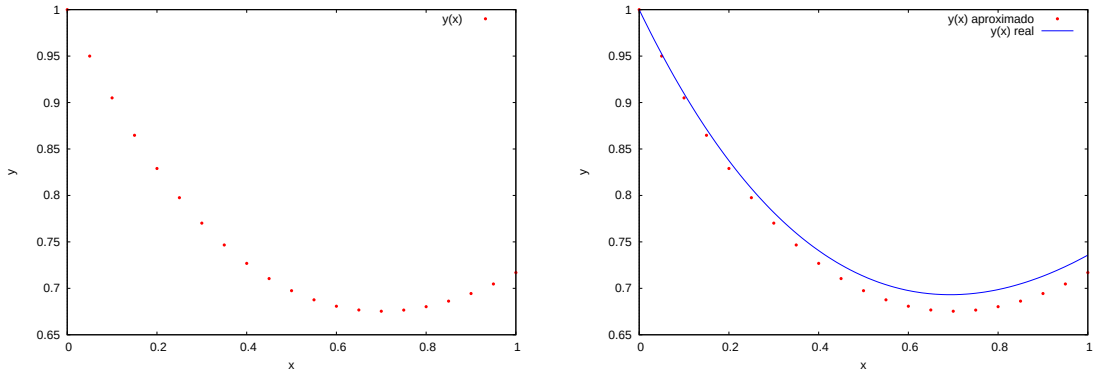

Figura 6: Solución numérica para $N = 20$ (•) (izquierda) y comparación con la solución exacta $y' + y = x$, $y(0) = 1$ (línea) (derecha)

Comparemos ahora la solución exacta $y(x) = 2e^{-x} - 1 + x$ del PVI con los valores numéricos que nos da en método de Euler y dibujemos ambas

```
(%i12) expon(x):=-1+2*exp(-x)+x$
              wxplot2d([[discrete,sol],expon(x)],[x,0,1],
                    [style, [points,2,2], [lines,1,1]],
                    [legend,"y(x) aproximado","y(x) real"],
                     [xlabel,"x"], [ylabel,"y"])$
(%t13)  << Graphics >>
```

Finalmente podemos calcular los errores cometidos en cada paso y representarlos gráficamente –ver figura 6(derecha)–.

```
(%i14) compsol:makelist(float([x[k],abs(y[k]-expon(x[k]))]),k,0,Nu)$
             wxplot2d([[discrete,compsol]],[style, [points,2,3]],
                                [legend,false],[xlabel,"x"], [ylabel,"y(x)-y"])$
(%t15)  << Graphics >> -->
```

Si hacemos un simple cambio como aumentar en número de subintervalos hasta 40 (el doble) vemos que la precisión mejora notablemente. Como ejercicio realizar el programa para $N = 40$ y comparar los resultados con el caso anterior (dibuja las correspondientes gráficas).

Consideremos ahora el problema

$$y' - 1 - y^2 = 0, \qquad y(0) = 0, \qquad x \geq 0.$$

Vamos a usar un esquema de 80 puntos en el intervalo $[0, 2]$ y representemos la solución numérica en la gráfica 7 (izquierda) en el intervalo $[0, 0{,}6]$ mediante $\bullet$. Como la solución exacta de la ecuación es $y(x) = \tan x$, vamos a dibujar ambas en la gráfica 7(izquierda) siendo la línea la solución exacta. Se puede ver que prácticamente coinciden.

```
(%i16) kill(x,y)$
       x[0]:0$ l:2$ Nu:80$ h:l/Nu$
       x[n]:=x[0]+n*h$ kill(f)$ y[0]:0$
       define(f[x,y],1+x*x);
       y[k]:=y[k-1]+h*f[x[k-1],y[k-1]]$
       solt:makelist(float([x[k],y[k]]),k,0,Nu)$
       wxplot2d([[discrete,solt],tan(x)],[x,0,.6],
            [style, [points,2,2], [lines,1,1]],
            [legend,"y(x) aproximado","y(x) real"],
            [xlabel,"x"], [ylabel,"y"])$
(%o17) done
(%o24) f[x,y]:=x^2+1
(%t27)  << Graphics >>
```

¿Qué ocurre si dibujamos las soluciones hasta llegar a $x = 2$?

```
(%i28) wxplot2d([[discrete,solt],tan(x)],[x,0,1.5],
         [style, [points,2,2], [lines,1,1]],
         [legend,"y(x) aproximado","y(x) real"],
         [xlabel,"x"], [ylabel,"y"])$
(%t28)  << Graphics >>
```

El resultado está representado en la gráfica 7 (derecha). La principal razón de la divergencia es obvia: la solución $\tan x$ no está definida en $x = \pi/2 \approx 1{,}55$. Una pregunta natural es, por tanto, ¿cómo decidir a priori si el método de Euler nos está dando la soución correcta? y ¿en qué intervalo podemos asegurar que $\widehat{y}$ es una buena aproximación de y? Eso nos conduce a preguntarnos condiciones suficientes que nos garanticen la existencia y unicidad de las soluciones de una EDO.

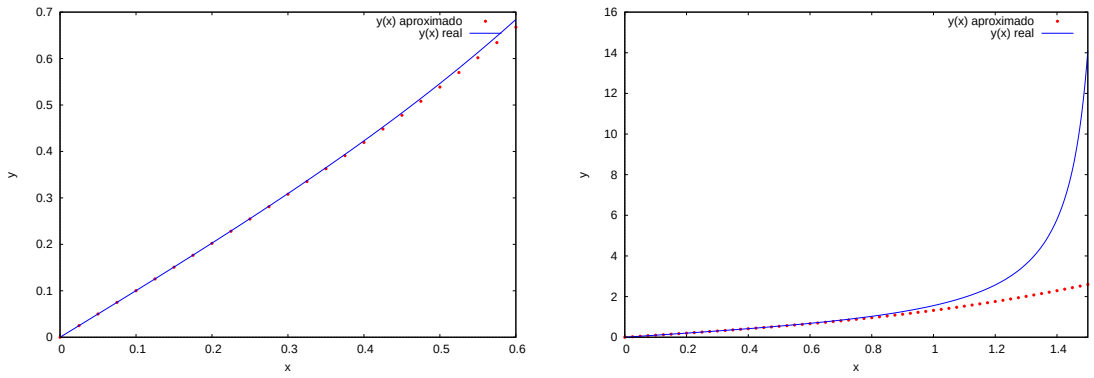

Figura 7: Comparación de las soluciones numérica y exacta del PVI $y' - 1 - y^2 = 0$, $y(0) = 0$ en los intervalos $x \in [0, 0{,}6]$ (izquierda) y $x \in [0, 2]$ (derecha).

4.2. El método de Euler mejorado

El método de Euler es el más sencillo pero tiene un problema, es un método de orden uno, o sea, es *"poco preciso"*. ¿Cómo mejorarlo?

Una posibilidad es truncar la serie de Taylor (2) en el tercer orden, de forma que tengamos

$$y_{k+1} = y_k + hf(x_k, y_k) + \left[\frac{\partial f}{\partial x}(x_k, y_k) + f(x_k, y_k)\frac{\partial f}{\partial y}(x_k, y_k)\right]\frac{h^2}{2}, \qquad y_0 = y(x_0).$$

La ecuación anterior se conoce como el método de la serie de Taylor de tres términos y, aunque es más preciso que el de Euler (se puede probar que es un método de orden 2), es algo incómodo sobre todo si f es una función "*complicada*". Por ello se suele usar una modificación del mismo.

Para ello escribamos la EDO original $y' = f(x, y)$ en el intervalo $x_k, x_k + h$ en su forma integral

$$y(x_{k+1}) = y(x_k) + \int_{x_k}^{x_k+h} f(x, y(x))dx. \tag{4}$$

Para resolver este problema aproximamos la integral mediante un rectángulo de altura $f(x_k, y_k)$ (ver figura 8 izquierda)

$$\int_{x_k}^{x_k+h} f(x, y(x))dx \approx hf(x_k, y_k),$$

lo que nos conduce nuevamente al esquema de Euler (3)

Esta aproximación es muy burda. Una mejor aproximación es usar la regla de los trapecios (ver figura 8 derecha) para aproximar la integral, es decir

$$\int_{x_k}^{x_k+h} f(x, y(x))dx \approx \frac{h}{2}\left[f(x_k, y_k) + f(x_{k+1}, y_{k+1})\right],$$

de donde se deduce el esquema *implícito*

$$y_{k+1} = y_k + \frac{h}{2}\left[f(x_k, y_k) + f(x_{k+1}, y_{k+1})\right], \qquad k = 0, 1, \ldots, N, \quad y_0 = y(x_0).$$

Obviamente este esquema es muy incómodo pues hay que resolver la ecuación implícita para hallar y_{k+1}. Una forma de obviar esta dificultad es usar la predicción que da el método de Euler $y_{k+1} = y_k + hf(x_k, y_k)$, de esta forma obtenemos el *método de Euler mejorado*

$$y_{k+1} = y_k + \frac{h}{2}\left[f(x_k, y_k) + f(x_{k+1}, y_k + hf(x_k, y_k))\right], \qquad k = 0, 1, \ldots, N, \quad y_0 = y(x_0). \tag{5}$$

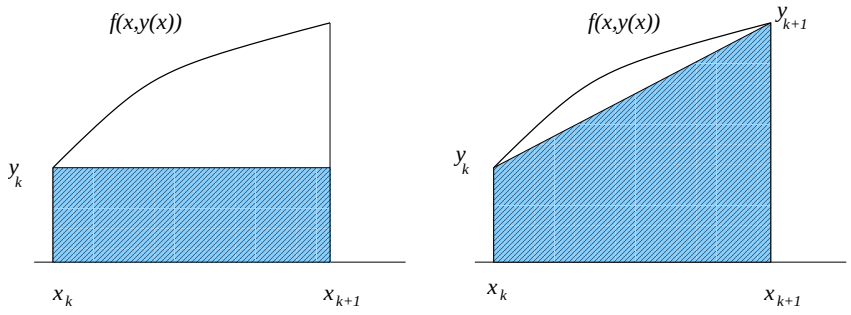

Figura 8: Regla del rectángulo (izquierda) y del trapecio (derecha) para aproximar una integral

Como ejemplo resolvamos nuevamente el PVI

$$y' + y = x, \qquad y(0) = 1, \quad x \in [0, 1],$$

cuya solución exacta es $y(x) = 2e^{-x} - 1 + x$. Escogeremos una discretización equidistante con paso $h = 1/20$ (20 subintervalos iguales).

```
(%i29) kill(x,ym);
       x[0]:0$ ym[0]:1$ ym[0]:1$ l:1$ Nu:20$ h:l/Nu$
       x[n]:=x[0]+n*h$
       define(f[x,y],-y+x)$
       ym[k]:=float(ym[k-1]+(h/2)*(f[x[k-1],ym[k-1]]+
              f[x[k], ym[k-1]+h*f[x[k-1],ym[k-1]]]))$
       solm:makelist(float([x[k],ym[k]]),k,0,Nu)$
       wxplot2d([[discrete,sol],[discrete,solm],expon(x)],[x,0,1],
         [style, [points,2,2], [points,2,5],  [lines,1,1]],
         [legend,"y(x) euler","y(x) euler mejorado","y(x) real"],
         [xlabel,"x"], [ylabel,"y"])$
(%o29) done
(%t40)  << Graphics >>
```

Los resultados los vemos en la gráfica 9.

Como ejercicio, construir la matriz de errores similar a la de la tabla 1 pero para el caso del método de Euler mejorado (ver resultado en la tabla 2).

5. Resolviendo sistemas de EDOs lineales

Vamos a dedicar este apartado a los sistemas de EDOs lineales, i.e., los sistemas de la forma:

$$\begin{cases} y_1'(x) = a_{11}(x)y_1(x) + a_{12}(x)y_2(x) + \cdots + a_{1n}(x)y_n(x) + b_1(x), \\ y_2'(x) = a_{21}(x)y_1(x) + a_{22}(x)y_2(x) + \cdots + a_{2n}(x)y_n(x) + b_2(x), \\ \qquad\qquad \vdots \\ y_n'(x) = a_{n1}(x)y_1(x) + a_{n2}(x)y_2(x) + \cdots + a_{nn}(x)y_n(x) + b_n(x). \end{cases} \tag{6}$$

El sistema anterior se suele escribir en la forma matricial

$$Y'(x) = A(x)Y(x) + B(x), \tag{7}$$

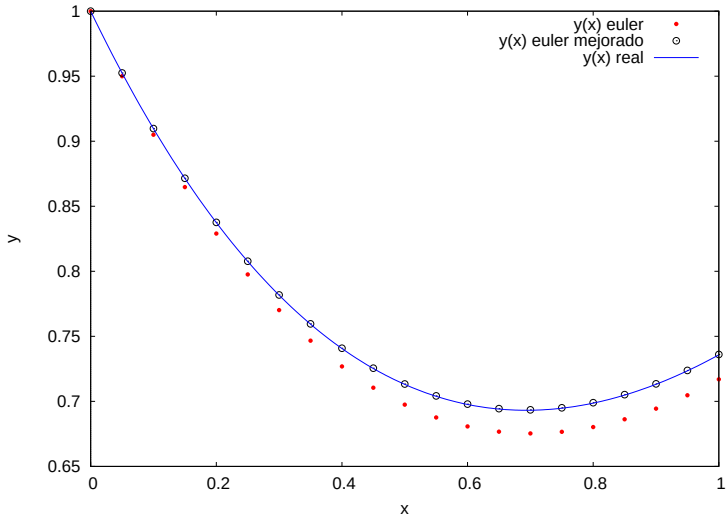

Figura 9: Comparación gráfica de las soluciones numéricas y exacta (línea) de $y' + y = x$, $y(0) = 1$ para $N = 20$ usando los métodos de Euler ($\bullet$) y Euler mejorado ($\circ$).

donde

$$A(x) = \begin{pmatrix} a_{11}(x) & a_{12}(x) & a_{13}(x) & \cdots & a_{1n}(x) \\ a_{21}(x) & a_{22}(x) & a_{23}(x) & \cdots & a_{2n}(x) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1(x)} & a_{n2}(x) & a_{n3}(x) & \cdots & a_{nn}(x) \end{pmatrix}, \qquad B(x) = \begin{pmatrix} b_1(x) \\ b_2(x) \\ \vdots \\ b_n(x) \end{pmatrix},$$

Cuando $B(x) = 0$, o sea cuando todas las componentes del vector B son cero, diremos que SEDO es homogéneo, y si al menos una componente de B es no nula, no homogéneo. En el caso de un sistema homogéneo se tiene el siguiente resultado:

Teorema 1 *Si $Y_1(x)$, ..., $Y_n(x)$ son soluciones linealmente independientes del sistema $Y' = A(x)Y$, entonces toda solución de $Y' = A(x)Y$ se puede expresar como una única combinación lineal de dichas soluciones. O sea, existen unos únicos coeficientes c_1, ... c_n tales que*

$$Y(x) = c_1 Y_1(x) + c_2 Y_2(x) + \cdots + c_n Y_n(x).$$

En particular, la solución del PVI, $Y' = A(x)Y$, $Y(x_0) = Y_0$ se expresa de manera única como una combinación lineal de las soluciones del sistema homogéneo correspondiente.

Así que, para encontrar la solución gereral del SEDO $Y' = A(x)Y$, basta encontrar n soluciones independientes del mismo. Una de las opciones más comúnes es buscar la solución de $Y' = A(x)Y$ en la forma

$$Y(x) = e^{\lambda x} v, \tag{8}$$

donde λ es cierta constante y v un vector constante. Sustituyendo (8) en (7) tenemos, como v es constante[11]

$$Y'(x) = (e^{\lambda x} v)' = \lambda e^{\lambda x} \quad \Longrightarrow \lambda e^{\lambda x} v = A\, e^{\lambda x} v.$$

[11] Entenderemos que la derivada de un vector o una matriz es la derivada término a término.

k	x_k	$\widehat{y}(x_k)$	$y(x_k)$	$\widehat{y}(x_k) - y(x_k)$
0	0	1.	1.	0
1.	0.05	0.95125	0.952459	-0.00120885
2.	0.1	0.907314	0.909675	-0.00236077
3.	0.15	0.867958	0.871416	-0.00345845
4.	0.2	0.832957	0.837462	-0.00450443
5.	0.25	0.8021	0.807602	-0.00550115
6.	0.3	0.775186	0.781636	-0.00645092
7.	0.35	0.75202	0.759376	-0.00735595
8.	0.4	0.732422	0.74064	-0.00821835
9.	0.45	0.716216	0.725256	-0.00904012
10.	0.5	0.703238	0.713061	-0.00982318
11.	0.55	0.69333	0.7039	-0.0105693
12.	0.6	0.686343	0.697623	-0.0112803
13.	0.65	0.682134	0.694092	-0.0119578
14.	0.7	0.680567	0.693171	-0.0126034
15.	0.75	0.681515	0.694733	-0.0132186
16.	0.8	0.684853	0.698658	-0.0138047
17.	0.85	0.690467	0.70483	-0.0143632
18.	0.9	0.698244	0.713139	-0.0148955
19.	0.95	0.708079	0.723482	-0.0154026
20.	1.	0.719873	0.735759	-0.0158858

Cuadro 2: Solución númerica de la ecuación $y' + y = x$, $y(0) = 1$ con $N = 20$ usando el método de Euler mejorado.

Es decir, (8) es solución del sistema homogéneo si y sólo si λ es un autovalor de A y v su autovector asociado, por tanto para resolver nuestro sistema lineal basta encontrar n autovectores $v_1, \ldots, v_n$ linealmente independientes en cuyo caso la solución general es

$$Y(x) = \sum_{k=1}^{n} c_k e^{\lambda_k x} v_k, \tag{9}$$

donde $c_1, \ldots, c_n$ son constantes arbitrarias y λ_k, $k = 1, \ldots, n$, son los autovalores asociados a los autovectores v_k, $k = 1, \ldots, n$.

En caso no homógeneo es algo más complicado pues se precisa del concepto de exponencial de una matriz: Sea A una matriz $n \times n$ y $x \in \mathbb{R}$. Definiremos la función $\exp(xA)$ mediante la serie formal

$$\exp(xA) = \sum_{k=0}^{\infty} \frac{x^k A^k}{k!} = I_n + xA + \frac{x^2 A^2}{2} + \cdots + \frac{x^n A^n}{n!} + \cdots, \tag{10}$$

donde la convergencia[12] la entenderemos elemento a elemento. La serie anterior converge para todo $x \in \mathbb{R}$ quienquiera sea la matriz A (de hecho converge en todo $\mathbb{C}$).

Teorema 2 *La función* $\exp(xA)$ *satisface las siguientes propiedades:*

1. *Para toda matriz* A, $\exp(0A) = I_n$ *y para todo* $x \in \mathbb{R}$ $\exp(xO_n) = I_n$, *donde* O_n *es la matriz nula.*
2. *Para toda matriz* A, $\frac{d}{dx}\exp(xA) = A\exp(xA) = \exp(xA)A$.
3. *Para todos* $x, t \in \mathbb{R}$ *y toda matriz* A, $\exp[(x+t)A] = \exp(xA)\exp(tA)$.
4. *Para toda matriz* A, *la inversa* $[\exp(xA)]^{-1} = \exp(-xA)$.
5. *Para todas las matrices* A, B *con* $AB = BA$, $\exp[x(A+B)] = \exp(xA)\exp(xB)$.
6. *Para todo* $x \in \mathbb{R}$, $\exp(xI_n) = e^x I_n$.

[12] Es más apropiado definir el correspondiente espacio normado pero eso se sale de nuestros propósitos.

Dado cualquier vector *constante* $v \in \mathbb{R}^n$ se cumple

$$\frac{d}{dx}[\exp(xA)v] = A[\exp(xA)v],$$

por tanto $\exp(xA)v$ es solución de la ecuación homogénea $Y' = AY$ y $Y(0) = v$. Si escogemos v sucesivamente como e_i, $i = 1, 2, \ldots, n$ obtenemos n soluciones v_1, $\ldots$, v_n del PVI, $Y' = AY$, $Y(0) = e_i$ que además son linealmente independientes y por tanto constituyen una base del espacio de soluciones del sistema homogéneo correspondiente.

Otro concepto importante es el matriz fundamental: Una matriz $V \in \mathbb{R}^{n\times n}$ es una matriz fundamental del sistema $Y' = AY$ si sus n columnas son un conjunto linealmente independiente de soluciones de $Y' = AY$. Obviamente la matriz exponencial es una matriz fundamental del sistema $Y' = AY$.

Vamos a usar MAXIMA para encontrar la solución de un sistema homogéneo así como la exponencial de una matriz. Por sencillez nos centraremos en el caso de matrices de 2x2.

5.1. Resolviendo sistemas con Maxima

Comenzaremos de la forma más simple: usando el comando `desolve` (ver el inico de la sección 3.1) podemos resolver cualquier sistema de ecuaciones lineales de forma simbólica (analítica). Lo primero que haremos es introducir la matriz A que necesitaremos más adelante. Para ello se usa la orden `matrix` cuya sintaxis es

```
matrix(fila1,fila2,...,finaN)
```

donde `fila1`, `fila2`, etc. son los vectores filas (listas de números `a1, a2, ...` de la forma `[a1,a2,...,aM]`)[13]

Veamos como ejemplo la resolución del sistema

$$Y' = \begin{pmatrix} 1 & 12 \\ 3 & 1 \end{pmatrix} Y.$$

Para ello hacemos

```
(%i1) A:matrix([1,12],[3,1]);
(%o1) matrix([1,12],[3,1])
(%i2) kill(y1,y2,x)$
      desolve(
      ['diff(y1(x),x)=y1(x)+12*y2(x),'diff(y2(x),x)=3*y1(x)+y2(x)],
      [y1(x),y2(x)]);
(%o3) [y1(x)=((2*y2(0)+y1(0))*%e^(7*x))/2-((2*y2(0)-y1(0))*%e^(-5*x))/2,
      y2(x)=((2*y2(0)+y1(0))*%e^(7*x))/4+((2*y2(0)-y1(0))*%e^(-5*x))/4]
```

y obtenemos la solución general. Nótese que en la salida de MAXIMA aparecen los valores $y_1(0)$ e $y_2(0)$ que apriori son desconcidos. Si queremos resolver el PVI

$$Y' = \begin{pmatrix} 1 & 12 \\ 3 & 1 \end{pmatrix} Y, \quad Y(0) = \begin{pmatrix} 1 \\ 2 \end{pmatrix},$$

antonces debemos usar además el comando `atvalue`

[13] La salida de `matrix([1,12],[3,1])` es en realidad de la forma $\begin{bmatrix} 1 & 12 \\ 3 & 1 \end{bmatrix}$.

```
(%i4) kill(y1,y2,x)$
      atvalue(y1(x),x=0,1)$
      atvalue(y2(x),x=0,2)$
      desolve(
      ['diff(y1(x),x)=y1(x)+12*y2(x),'diff(y2(x),x)=3*y1(x)+y2(x)],
      [y1(x),y2(x)]);
(%o7) [y1(x)=(5*%e^(7*x))/2-(3*%e^(-5*x))/2,
       y2(x)=(5*%e^(7*x))/4+(3*%e^(-5*x))/4]
```

Para encontrar la matriz exponencial usaremos la propiedad de que las columnas E_k de la misma son las soluciones de los PVI $E_k'(x) = AE_k(x)$, $E_k(0) = e_k$, donde e_k es la base canónica de $\mathbb{R}^n$. Así resolvemos primero el PVI

$$Y' = \begin{pmatrix} 1 & 12 \\ 3 & 1 \end{pmatrix} Y, \quad Y(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

```
(%i8) kill(y1,y2,x)$
      atvalue(y1(x),x=0,1)$
      atvalue(y2(x),x=0,0)$
      col1:desolve(
      ['diff(y1(x),x)=y1(x)+12*y2(x),'diff(y2(x),x)=3*y1(x)+y2(x)],
      [y1(x),y2(x)]);
(%o11) [y1(x)=%e^(7*x)/2+%e^(-5*x)/2,y2(x)=%e^(7*x)/4-%e^(-5*x)/4]
(%i12) ec1:makelist(second(col1[k]),k,1,length(col1));
(%o12) [%e^(7*x)/2+%e^(-5*x)/2,%e^(7*x)/4-%e^(-5*x)/4]
```

y luego el PVI

$$Y' = \begin{pmatrix} 1 & 12 \\ 3 & 1 \end{pmatrix} Y, \quad Y(0) = \begin{pmatrix} 0 \\ 1 \end{pmatrix},$$

```
(%i13) kill(y1,y2,x)$
       atvalue(y1(x),x=0,0)$
       atvalue(y2(x),x=0,1)$
       col2:desolve(
       ['diff(y1(x),x)=y1(x)+12*y2(x),'diff(y2(x),x)=3*y1(x)+y2(x)],
       [y1(x),y2(x)]);
(%o16) [y1(x)=%e^(7*x)-%e^(-5*x),y2(x)=%e^(7*x)/2+%e^(-5*x)/2]
(%i17) ec2:makelist(second(col2[k]),k,1,length(col2));
(%o17) [%e^(7*x)-%e^(-5*x),%e^(7*x)/2+%e^(-5*x)/2]
```

Dado que las salidas `ec1` y `ec2` son vectores filas, los convertimos en comumna simplemente transponiendo la matriz con el comando `transpose`

```
(%i18) define(expA(x),transpose(matrix(ec1,ec2)));
(%o18) expA(x):=matrix([%e^(7*x)/2+%e^(-5*x)/2,%e^(7*x)-%e^(-5*x)],
       [%e^(7*x)/4-%e^(-5*x)/4,%e^(7*x)/2+%e^(-5*x)/2])
```

que nos da como salida la matriz exponencial:

$$e^{x\left(\begin{smallmatrix} 1 & 12 \\ 3 & 1 \end{smallmatrix}\right)} = \begin{pmatrix} \dfrac{e^{7x}}{2} + \dfrac{e^{-5x}}{2} & e^{7x} - e^{-5x} \\ \dfrac{e^{7x}}{4} - \dfrac{e^{-5x}}{4} & \dfrac{e^{7x}}{2} + \dfrac{e^{-5x}}{2} \end{pmatrix}$$

Finalmente comprobamos que $(e^{xA})' = Ae^{xA}$ y que $e^{0A} = 0$

```
(%i19) expA(0);
(%o19) matrix([1,0],[0,1])
(%i20) ratsimp(diff(expA(x),x)-A.expA(x));
(%o20) matrix([0,0],[0,0])
```

Resolvamos el problema ahora usando la técnica descrita al principio de esta sección, es decir usando los autovalores y autovectores. Para ello usamos el comando `eigenvectors` cuya sintaxis es

```
eigenvectors(matriz)
```

donde la variable `matriz` es una matriz NxN. La salida de este comando son dos listas, la primera de ellas son dos vectores fila, el primero contiene los autovectores y el segundo la correspondiente multiplicidad algebraica de los mismos. La segunda lista es una lista de vectores fila que se corresponden con los autovalores. Así, para nuestra matriz $\begin{pmatrix} 1 & 12 \\ 3 & 1 \end{pmatrix}$ tenemos

```
(%i21) vec:eigenvectors(A);
(%o21) [[[-5,7],[1,1]],[[1,-1/2]],[[1,1/2]]]
(%i22) vec[1]; vec[2];
(%o22) [[-5,7],[1,1]]
(%o23) [[[1,-1/2]],[[1,1/2]]]
```

La primera lista $[[-5,7],[1,1]]$ nos indica que el autovalor $\lambda = -5$ tiene multiplicidad 1 y que el $\lambda = 7$ tiene multiplicidad 1. Además, a $\lambda = -5$ le corresponde el autovector $(1,-1/2)^T$ y a $\lambda = 7$ el autovector $(1,1/2)^T$, respectivamente. Para extraer los autovalores y autovectores por separado usamos las "[ ]". Así, si hacemos `vec[1]` obtenemos la primera de las dos listas (la de los autovaores y multiplicidades), `vec[1][1]` nos da el primer elemento de esa lista, o sea la de los autovalores, y `vec[1][1][2]` nos imprime el segundo autovalor. De forma similar sacamos los correspondientes autovectores lo que nos permitirá definir la dos soluciones linealmente independientes `s1(x)` y `s2(x)`

```
(%i24) av1:vec[1][1][1]; av2:vec[1][1][2];
       v1:vec[2][1][1];  v2:vec[2][2][1];
(%o24) -5
(%o25) 7
(%o26) [1,-1/2]
(%o27) [1,1/2]
(%i28) s1(x):=%e^(av1*x)*v1$ s1(x);
       s2(x):=%e^(av2*x)*v2$ s2(x);
(%o29) [%e^(-5*x),-%e^(-5*x)/2]
(%o31) [%e^(7*x),%e^(7*x)/2]
```

es decir

$$Y_1(x) = \begin{pmatrix} e^{-5x} \\ -\frac{1}{2}e^{-5x} \end{pmatrix}, \quad Y_2(x) = \begin{pmatrix} e^{7x} \\ \frac{1}{2}e^{7x} \end{pmatrix}.$$

Lo anterior nos permite definir una matriz fundamental

$$V(x) = [Y_1(x)Y_2(x)] = \begin{pmatrix} e^{-5x} & e^{7x} \\ -\frac{1}{2}e^{-5x} & \frac{1}{2}e^{7x} \end{pmatrix}.$$

```
(%i32) define(v(x),transpose(matrix(s1(x),s2(x))));
(%o32) v(x):=matrix([%e^(-5*x),%e^(7*x)],[-%e^(-5*x)/2,%e^(7*x)/2])
```

Para tener la exponencial usamos la fórmula

$$\exp(xA) = V(x)V^{-1}(0). \tag{11}$$

Además comprobamos que efectivamente $(e^{xA})' = Ae^{xA}$ y $e^{0A} = 0$ y que obviamente es la misma matriz que obtuvimos antes

```
(%i33) define(e(x),v(x).invert(v(0)));
(%o33) e(x):=matrix([%e^(7*x)/2+%e^(-5*x)/2,%e^(7*x)-%e^(-5*x)],
             [%e^(7*x)/4-%e^(-5*x)/4,%e^(7*x)/2+%e^(-5*x)/2])
(%i34) ratsimp(diff(e(x),x)-A.e(x));
(%o34) matrix([0,0],[0,0])
(%i35) expA(x)-e(x);
(%o35) matrix([0,0],[0,0])
```

Dado que ya sabemos calcular la matriz exponencial ahora podemos resolver facilmente el problema no homogéneo. Para ello usamos el resultado

Teorema 3 *La solución del problema de valores iniciales* $Y'(x) = AY(x) + B(x)$, $Y(x_0) = Y_0$, *cualquiera sea* $Y_0 \in \mathbb{R}^n$ *existe y es única y se expresa mediante la fórmula*

$$Y(x) = \exp[(x - x_0)A]Y_0 + \int_{x_0}^{x} \exp[(x - t)A]B(t)dt. \tag{12}$$

Vamos por tanto a resolver el sistema

$$Y' = \begin{pmatrix} 1 & 12 \\ 3 & 1 \end{pmatrix} Y + \begin{pmatrix} e^{-x} \\ 0 \end{pmatrix}, \quad Y(0) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Ante todo definimos el término independiente B (usaremos la `b`), luego multiplicamos $e^{(x-t)A} \cdot B(t)$ y definimos la función $\int_0^x \exp[(x-t)A]B(t)dt$. Aquí es importante destacar que la multiplicación[14] habitual de matrices en MAXIMA se ejecuta con ".".

```
(%i36) b:transpose([exp(-t),0]);
(%o36) matrix([%e^(-t)],[0])
(%i37) ratsimp(e(x-t).b); expand(%);
       define(sol(x),expand(integrate(%, t, 0, x)));
(%o37) matrix([(%e^(-5*x-8*t)*(%e^(12*x)+%e^(12*t)))/2],
              [(%e^(-5*x-8*t)*(%e^(12*x)-%e^(12*t)))/4])
(%o38) matrix([%e^(7*x-8*t)/2+%e^(4*t-5*x)/2],
              [%e^(7*x-8*t)/4-%e^(4*t-5*x)/4])
(%o39) sol(x):=matrix([%e^(7*x)/16+%e^(-x)/16-%e^(-5*x)/8],
               [%e^(7*x)/32-(3*%e^(-x))/32+%e^(-5*x)/16])
```

Luego calculamos $\exp[(x)A]Y_0$ y definimos la solución según la fórmula (12)

```
(%i40) e(x).transpose([0,1]);
(%o40) matrix([%e^(7*x)-%e^(-5*x)],[%e^(7*x)/2+%e^(-5*x)/2])
(%i41) define(solt(x),sol(x)+e(x).transpose([0,1]));
(%o41) solt(x):=matrix([(17*%e^(7*x))/16+%e^(-x)/16-(9*%e^(-5*x))/8],
                [(17*%e^(7*x))/32-(3*%e^(-x))/32+(9*%e^(-5*x))/16])
```

[14] No se puede confundir la operación `A.B` con `A*B`. Si bien la primera es la multiplicación habitual de matrices, la segunda es una multiplicación elemento a elemento. Por ejemplo

$$A = \begin{pmatrix} 1 & 12 \\ 3 & 1 \end{pmatrix}, C = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \quad A.B = \begin{pmatrix} 13 & 13 \\ 4 & 4 \end{pmatrix} \text{ pero } A * B \begin{pmatrix} 1 & 12 \\ 3 & 1 \end{pmatrix}.$$

Más aún si v es un vector 2x2, $A.v$ esta bien definido pero $A * v$ no.

Para terminar comprobamos que nuestra solución efectivamente satisface la ecuación origonal con las condiciones iniciales.

```
(%i42) solt(0);
(%o42) matrix([0],[1])
(%i43) ratsimp(diff(solt(x),x)-A.solt(x));
(%o43) matrix([%e^(-x)],[0])
```

El caso de autovalores múltiples es más complicado y lo omitiremos ya que desde el punto de vista de MAXIMA no aporta nada nuevo y se puede resolver por el primer método sin problemas.

Para terminar este apartado resolvamos el siguiente problema de valores iniciales

$$\frac{du}{dx} = u(1-v), \quad \frac{dv}{dx} = \alpha v(u-1), \qquad u(x_0) = u_0,\ v(x_0) = v_0. \tag{13}$$

Ante todo notémos que dividiendo ambas ecuaciones se tiene que

$$\frac{du}{dv} = \alpha \frac{v(u-1)}{u(1-v)},$$

cuya solución es $\alpha u + v - \log u^\alpha v = H = const$. Si dibujamos los valores de u respecto a v obtenemos las *trajectorias de las soluciones del sistema*. Así pues, si $H > 1+\alpha$ (que es el mínimo de H que se alcanza para $u = v = 1$), entonces las *trayectorias* definidas por $\alpha u + v - \log u^\alpha v = H$ son cerradas lo que implica que u y v son funciones periódicas. Dichas trayectorias están representadas en el gráfico 10. Para ello necesitamos el comando `implicit_plot` que está en el paquete `implicit_plot`

```
(%i44) load(implicit_plot)$
(%i45) implicit_plot ([
       0.5*u+v-log(v*u^0.5)=1.6, 0.5*u+v-log(v*u^0.5)=2, 0.5*u+v-log(v*u^0.5)=2.3,
       0.5*u+v-log(v*u^0.5)=2.5, 0.5*u+v-log(v*u^0.5)=3, 0.5*u+v-log(v*u^0.5)=3.5],
       [u,0.01, 7], [v,0.01, 6], [legend,"H1=1.6","H2=2.0","H3=2.3","H4=2.5",
                                          "H5=3.0","H6=3.5"])$
(%o45) done
```

Como vemos la sintaxis es

```
implicit_plot(F(u,v)=0, [u, uini, ufin], [v, vini, vfin])
```

donde $F(u,v) = 0$ es la ecuación implícita que queremos dibujar en los intervalos de las variables `u` desde `uini` hasta `ufin` y `v` desde `vini` hasta `vfin`.

Para resolver el sistema usamos el comando `rk` que ya vimos en la sección 3.2.

```
(%i6) load(dynamics)$
(%i7) kill(a,u,v,sol)$ a:0.5;
      sol:rk([u*(1-v),a*v*(u-1)],[u,v],[0.4,0.2],[t,0,40,0.02])$
(%o8) 0.5
```

A continuación definimos las listas que vamos a representar, a saber los valores de (x_k, u_k) y (x_k, v_k). Como antes, usamos el comando `makelist`.

```
(%i10) u:makelist([sol[k][1],sol[k][2]],k,1,length(sol))$
v:makelist([sol[k][1],sol[k][3]],k,1,length(sol))$
ciclo:makelist([sol[k][2],sol[k][3]],k,1,length(sol))$
```

Finalmente representamos las gráficas de $u(x)$, $v(x)$,

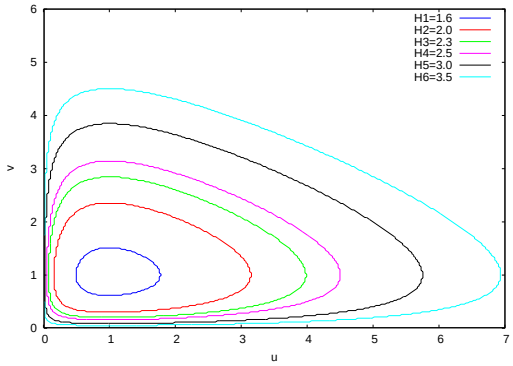

Figura 10: Trayectorias definidas por el sistema (13) para $\alpha = 1/2$ y $H = 1{,}6$, 2, 2.3, 2.5, 3 y 3.5.

```
(%i13) wxplot2d([discrete,u],[legend, "u"],
         [xlabel, "t"], [ylabel, "u"],[color,blue])$
       wxplot2d([discrete,v],[legend, "v"],
         [xlabel, "t"], [ylabel, "v"],[color,red])$
(%t13)  << Graphics >>
(%t14)  << Graphics >>
```

las comparamos,

```
(%i15) wxplot2d([[discrete,u],[discrete,v]],[legend, "u", "v"],
         [xlabel, "t"], [ylabel, "u,v"] )$
(%t15)  << Graphics >>
```

y culminamos dibujando las trajectorias $u(v)$

```
(%i16)  wxplot2d([discrete,ciclo],[legend, "u vs v"],
   [xlabel, "u"], [ylabel, "v"],[color,magenta] )$
(%t16)  << Graphics >>
```

Las dos últimas gráficas están representadas en la figura (11).

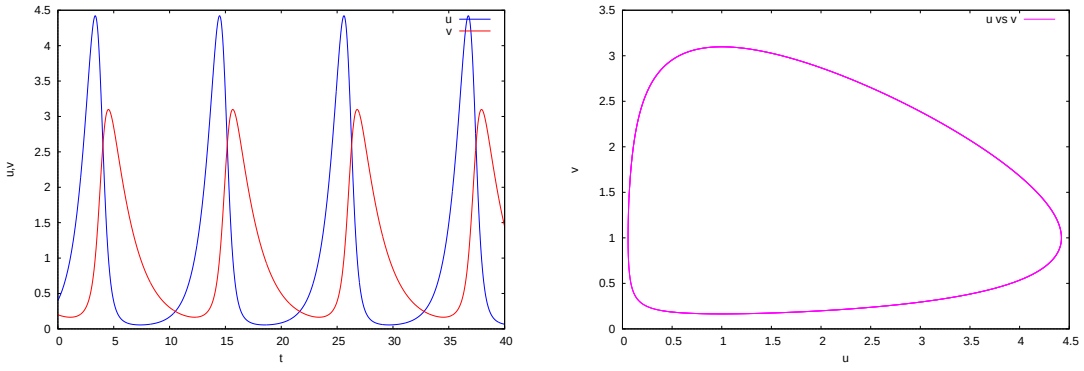

Figura 11: Las soluciones u y v del sistema (13) para $\alpha = 1/2$, $u(0) = 0{,}4$ y $v(0) = 0{,}2$.

5.2. Resolviendo EDOs lineales de orden n

Consideremos ahora otro grupo muy importante de EDOs: las EDOs lineales con coeficientes constantes de orden mayor que 1 definidas por la ecuación:

$$y^{(n)}(x) + a_{n-1}(x)y^{(n-1)}(x) + \cdots + a_1(x)y'(x) + a_0(x)y(x) = f(x). \tag{14}$$

Cuando $f(x) \equiv 0$, diremos que la EDO (14) es homógenea, en caso contrario es no homogénea. Si además imponemos que la solución y sea tal que

$$y(x_0) = y_0, \quad y'(x_0) = y_0', \quad \ldots \quad y^{(n-1)}(x_0) = y_0^{(n-1)},$$

entonces diremos que y es la solución del correspondiente problema de valores iniciales.

Para estudiar las propiedades de las EDOs lineales de orden n vamos a introducir unas nuevas funciones $z_1(x), \ldots z_n(x)$ de forma que

$$\left.\begin{array}{rcl} z_1(x) &=& y(x) \\ z_2(x) &=& y'(x) \\ z_3(x) &=& y''(x) \\ &\vdots& \\ z_{n-1}(x) &=& y^{(n-2)}(x) \\ z_n(x) &=& y^{(n-1)}(x) \end{array}\right\} \Rightarrow \begin{array}{rcl} \dfrac{dz_n}{dx} &=& -a_{n-1}z_n - a_{n-2}z_{n-1} - \cdots - a_0 z_1 + f, \\ \dfrac{dz_{n-1}}{dx} &=& z_n, \\ &\vdots& \\ \dfrac{dz_1}{dx} &=& z_2. \end{array}$$

Es decir, una EDO del tipo (14) es equivalente al siguiente sistema lineal de ecuaciones

$$\frac{d}{dx}\begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_{n-1} \\ z_n \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{n-2} & -a_{n-1} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_{n-1} \\ z_n \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ f(x) \end{pmatrix}, \tag{15}$$

o en forma matricial $Z'(x) = AZ(x) + F(x)$. Nótese que cuando $f(x) \equiv 0$, el sistema anterior se transforma en un sistema homogéneo.

Lo anterior tiene una implicación evidente: toda la teoría de las ecuaciones lineales (14) se puede construir a partir de la teoría de los sistemas lineales y por tanto el estudio con MAXIMA del apartado anterior se puede usar para resolver este tipo de ecuaciones.

Veamos algunos ejemplos sencillos. Comencemos resolviendo el PVI $y'' + 2y' + y = x^2 - 1$, $y(0) = 0$, $y'(0) = 2$. Para ello podemos usar el comando `ode2` junto a la orden `ic2` cuya sintaxis es

```
ic2(solución, valor de x, valor de y, valor de y')
```

donde `solución` es la solución general que da el comando `ode2` y `valor de x`, `valor de y` y `valor de y`, son los valores que toma la y y la y' cuando $x = x_0$, i.e., los valores iniciales. Esta combinación resuelve completamente el problema.

```
(%i1) eq1:'diff(y,x,2)-2*'diff(y,x,1)+2*y=x^2-1;
(%o1) 'diff(y,x,2)-2*('diff(y,x,1))+2*y=x^2-1
(%i2) ode2(eq1,y,x);
(%o2) y=%e^x*(%k1*sin(x)+%k2*cos(x))+(x^2+2*x)/2
(%i3) ic2(%,x=0,y=0,'diff(y,x)=2);
(%o3) y=%e^x*sin(x)+(x^2+2*x)/2
```

También podemos usar el comando `desolve`, aunque en este caso tenemos que escribir explícitamente tanto la variable dependiente como la independiente. En cojunto con la orden `atvalue` también nos da la correspondiente solución.

```
(%i4) eq2:'diff(y(x),x,2)-2*'diff(y(x),x,1)+2*y(x)=x^2-1;
(%o4) 'diff(y(x),x,2)-2*('diff(y(x),x,1))+2*y(x)=x^2-1
(%i5) atvalue(y(x),x=0,0)$
      atvalue('diff(y(x),x),x=0,2)$
      desolve(eq2,y(x));
(%o7) y(x)=%e^x*sin(x)+x^2/2+x
```

Intentemos lo mismo con la EDO $x(y')^2 - (1-xy)y' + y = 0$. Si usamos el comando `ode2` el sistema reconoce que no es una EDO lineal y da error.

```
(%i8) eq3:x*'diff(y,x)^2-(1+x*y)*'diff(y,x)+y=0;
(%o8) x*('diff(y,x,1))^2-(x*y+1)*('diff(y,x,1))+y=0
(%i9) ode2(eq3,y,x);
(%t9) x*('diff(y,x,1))^2-(x*y+1)*('diff(y,x,1))+y=0
"first order equation not linear in y'"
(%o9) false
```

La orden `desolve` devuelve su tipíco `ilt(...)` cuando no es capaz de resolverlo. En este caso podemos recurrir al paquete `contrib_ode` y la orden homónima que nos encuentra dos posibles soluciones:

```
(%i10) load('contrib_ode)$
define: warning: redefining the built-in function lcm
(%i11) contrib_ode(eq3,y,x);
(%t11) x*('diff(y,x,1))^2-(x*y+1)*('diff(y,x,1))+y=0
        "first order equation not linear in y'"
(%o11) [y=log(x)+%c,y=%c*%e^x]
```

Algo similar pasa con la EDO $(1-x)\,x\left(\frac{d^2}{dx^2}\,y\right) - 2\,x\left(\frac{d}{dx}\,y\right) + 12\,y = 0$. Aquí el comando `contrib_ode` da como salida dos funciones `gauss_a` y `gauss_b`, que no son más que las funciones y_1 e y_2 definidas por

$$y_1(x) = {}_2\mathrm{F}_1\left(\begin{array}{c}\alpha,\beta\\ \gamma\end{array}\,\middle|\,x\right) = \sum_{k=0}^{\infty}\frac{(\alpha)_k(\beta)_k}{(\gamma)_k}\frac{x^k}{k!}, \quad y_2(x) = x^{1-\gamma}{}_2\mathrm{F}_1\left(\begin{array}{c}\alpha-\gamma+1,\beta-\gamma+1\\ 2-\gamma\end{array}\,\middle|\,x\right),$$

respectivamente, donde por $(a)_n$ se denota el producto

$$(a)_0 = 1, \qquad (a)_n = a(a+1)\cdots(a+n-1), \quad n \geq 1.$$

```
(%i12) eq4:x*(1-x)*'diff(y,x,2)+(-2*x)*'diff(y,x)+4*(3)*y=0;
(%o12) (1-x)*x*('diff(y,x,2))-2*x*('diff(y,x,1))+12*y=0
(%i13) ode2(eq4,y,x);
(%o13) false
(%i14) contrib_ode(eq4,y,x);
(%o14) [y=gauss_b(-3,4,0,x)*%k2+gauss_a(-3,4,0,x)*%k1]
```

Hay que destacar que MAXIMA no trabaja con las funciones `gauss_a` y `gauss_b`, simplemente las reconoce. Si queremos trabajar con ellas tenemos que definir la correspondiente serie de potencias, o bien trabajar numéricamente, como sudece con la ecuación $y'' - 2xy' + 2y = 0$, que es imposible resolver usando los tres comandos anteriores.

```
(%i15) eq5:'diff(y(x),x,2)-2*x*'diff(y(x),x,1)+2*y(x)=0;
(%o15) 'diff(y(x),x,2)-2*x*('diff(y(x),x,1))+2*y(x)=0
(%i16) atvalue(y(x),x=0,1)$
```

```
       atvalue('diff(y(x),x),x=0,0)$
       desolve(eq5,y(x));
(%o18) y(x)=ilt( ... ,x)
(%i19) contrib_ode(eq5,y,x);
(%o19) false
```

En este caso, al igual que en todos, podemos recurrir al comando `rk` del paquete `dynamics` pero antes hay que convertir nuestra EDO de orden 2 en un sistema lineal. Para ello hacemos el cambio $z = y'$ de donde se tiene

$$y'' - 2xy' + 2y = 0 \quad \Leftrightarrow \quad \begin{cases} y' = z, \\ z' = 2xy - y. \end{cases}$$

Así pues tenemos

```
(%i20) load(dynamics)$
(%i21) sol:rk([z,2*x*y-y],[y,z],[1,0],[x,0,2,0.02])$
       valy:makelist([sol[k][1],sol[k][2]],k,1,length(sol))$
       wxplot2d([discrete,valy],[legend, "y"],
           [xlabel, "x"], [ylabel, "x"],[color,blue])$
(%t23)  << Graphics >>
```

donde la solución está representada en la gráfica 12

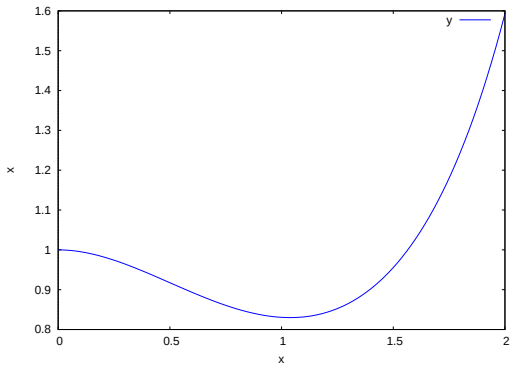

Figura 12: La solución del PVI $y'' - 2xy' + 2y = 0$ con $y(0) = 1$ e $y'(0) = 0$.

6. Un ejemplo usando series de potencias

Sea la EDO

$$y'' + p(x)y' + q(x)y = f(x). \tag{16}$$

El objetivo es resolver la EDO anterior cuando las funciones $p(x)$, $q(x)$ y $f(x)$ dependen de x. Para ello vamos a suponer que dichas funciones son "razonablemente" buenas en un entorno de cierto punto x_0 de la recta real y buscaremos la solución como una *serie de potencias*, es decir en la forma

$$y(x) = \sum_{k=0}^{\infty} a_k (x - x_0)^k, \qquad a_k \in \mathbb{R}. \tag{17}$$

Las funciones que se pueden expresar mediante una serie de potencias convergentes en todo un entorno de $x = x_0$ como la anterior se denominan funciones analíticas en $x = x_0$. Por sencillez

supondremos $x_0 = 0$, ya que en caso que $x_0 \neq 0$ siempre podemos realizar el cambio de variable $z = x - x_0$ de forma que en la nueva variable $z_0 = 0$.

La idea del método es sustituir la serie $y(x)$ (17) en (16) e igualar los coeficientes de las potencias. De esta forma obtenemos un sistema de ecuaciones para los coeficientes a_n en (17). Es importante que la serie converja en todo un entorno de x_0, de esta forma tenemos asegurada la existencia de la solución al menos en cierta región.

Como ejemplo resolvamos la EDO $y'' - 2xy' - 2y = 0$. Para ello buscamos la solución $y(x) = \sum_{k=0}^{\infty} a_k x^k$, la sustituimos en la EDO y usamos que

$$y'(x) = \frac{d}{dx}\sum_{k=0}^{\infty} a_k x^k = \sum_{k=0}^{\infty}(a_k x^k)' = \sum_{k=0}^{\infty} k a_k x^{k-1} = \sum_{n=0}^{\infty}(n+1)a_{n+1}x^n,$$
$$y''(x) = \frac{d}{dx}y'(x) = \frac{d}{dx}\sum_{k=0}^{\infty} k a_k x^{k-1} = \sum_{k=0}^{\infty} k(k-1)a_k x^{k-2} = \sum_{n=0}^{\infty}(n+1)(n+2)a_{n+2}x^n$$

lo que nos da

$$\sum_{k=0}^{\infty} k(k-1)a_k x^{k-1} - 2\sum_{k=0}^{\infty} k a_k x^k - 2\sum_{k=0}^{\infty} a_k x^k = 0,$$

que equivale a

$$\sum_{n=0}^{\infty}\left[(n+1)(n+2)a_{n+2} - (2n+2)a_n\right]x^n = 0.$$

Como $(x^n)_n$ es una sucesión de funciones linealmente independientes la igualdad a cero tiene lugar si y sólo si $(n+1)(n+2)a_{n+2} - (2n+2)a_n = 0$, de donde tenemos

$$a_{n+2} = \frac{2}{n+2}a_n, \qquad n \geq 0. \tag{18}$$

Obviamente la ecuación anterior define todos los valores de a_n en función de a_0 y a_1. En efecto, si sabemos a_0, la recurrencia anterior permite calcular los valores a_2, a_4, $\ldots a_{2k}$, $k \in \mathbb{N}$ y si conocemos a_1 entonces podemos calcular a_3, a_5, $\ldots a_{2k+1}$, $k \in \mathbb{N}$. Así, tenemos

$$a_{2n} = \frac{2}{2n}\,a_{2n-2} = \left(\frac{2}{2n}\right)\left(\frac{2}{2n-2}\right)a_{2n-4} = \cdots = \frac{2^n}{(2n)(2n-2)\cdots 2}a_0 = \frac{a_0}{n!},$$

$$a_{2n+1} = \frac{2}{2n+1}\,a_{2n-1} = \left(\frac{2}{2n+1}\right)\left(\frac{2}{2n-1}\right)a_{2n-3} = \cdots = \frac{2^n}{(2n+1)(2n-1)\cdots 3\cdot 1}\,a_1,$$

es decir $2^n/(2n+1)!!\,a_1$, donde $(2n+1)!! = 1\cdot 3\cdot 5\cdots(2n+1)$. De esta forma obtenemos dos soluciones linealmente independientes (una tiene solo potencias pares y la otra solo impares)

$$y(x) = a_0\sum_{n=0}^{\infty}\frac{x^{2n}}{n!} + a_1\sum_{n=0}^{\infty}\frac{2^n x^{2n+1}}{(2n+1)!!}.$$

Obviamente la primera suma es fácilmente reconocible como la serie de potencias de la función e^{x^2}, no así la segunda que en general no se expresa como combinación de funciones elementales. De la expresión explícita de las dos sumas anteriores es fácil comprobar que el radio de convergencia es infinito.

Resolvámoslo ahora con MAXIMA. Para ello definimos la sucesión `bn[n]` solución de (18) cuando $a_0 = 1$ y $a_1 = 0$.

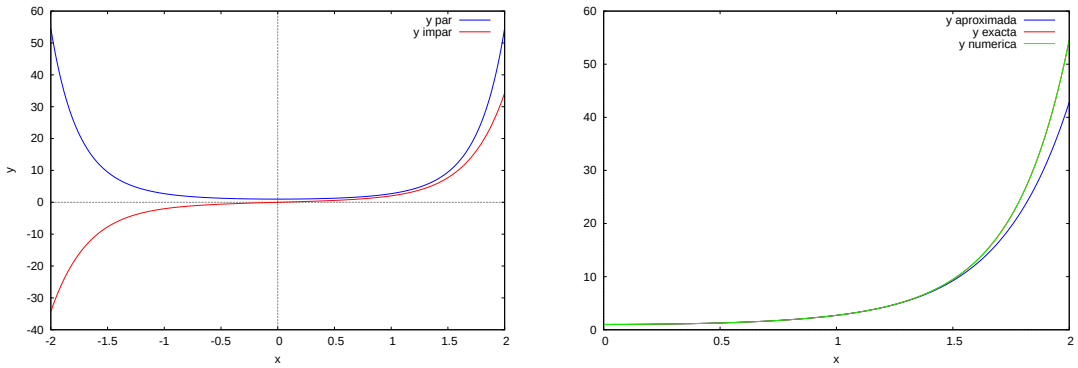

Figura 13: Las dos soluciones linealmente independientes de la EDO $y''-2xy'-2y=0$ (izquierda) y la comparación de las soluciones aproximada, exacta y numérica, respectivamente, del PVI cuando $y(0)=1$ e $y'(0)=0$. .

```
(%i1) bn[0]:1;bn[1]:0;
bn[n]:=2/(n)*bn[n-2];
(%o1) 1
(%o2) 0
(%o3) bn[n]:=2/n*bn[n-2]
```

Luego definimos la solución aproximada hasta el orden que queramos:

```
(%i4) solaprox(x,p):=sum(bn[n]*x^n,n,0,p);
(%o4) solaprox(x,p):=sum(bn[n]*x^n,n,0,p)
(%i5) solaprox(x,10);
(%o5) x^10/120+x^8/24+x^6/6+x^4/2+x^2+1
```

Ahora repetimos el proceso para obtener la solución de (18) cuando $a_0=0$ y $a_1=1$

```
(%i6) bi[0]:0;bi[1]:1;
bi[n]:=2/(n)*bi[n-2];
(%o6) 0
(%o7) 1
(%o8) bi[n]:=2/n*bi[n-2]
```

que nos conduce a la segunda solución linealmente independiente:

```
(%i9) solaproxi(x,p):=sum(bi[n]*x^n,n,0,p);
(%o9) solaproxi(x,p):=sum(bi[n]*x^n,n,0,p)
(%i10) solaproxi(x,10);
(%o10) (16*x^9)/945+(8*x^7)/105+(4*x^5)/15+(2*x^3)/3+x
```

Ahora podemos dibujar ambas soluciones

```
(%i11) wxplot2d([solaprox(x,10),solaproxi(x,10)],[x,-2,2],
   [legend,"y par","y impar"],[xlabel,"x"],[ylabel,"y"]);
(%t11)  << Graphics >>
```

En el caso que nos ocupa el comando `ode2` es capaz de resolver la EDO. En particular resolveremos el PVI cuando $y(0)=1$ e $y'(0)=0$

```
(%o11) (%i12) ed:'diff(y,x,2)-2*x*'diff(y,x)-2*y=0;
(%o12) 'diff(y,x,2)-2*x*('diff(y,x,1))-2*y=0
(%i13) ode2(ed,y,x);
       ic2(%,x=0,y=1,diff(y,x)=0);
(%o13) y=(sqrt(%pi)*%k1*%e^x^2*erf(x))/2+%k2*%e^x^2
(%o14) y=%e^x^2
```

que nos conduce a la primera de las dos soluciones:

```
(%i15) define(soana(x),rhs(%));
(%o15) soana(x):=%e^x^2
```

Dibujamos las soluciones aproximada y exacta, respectivamente:

```
(%i16) wxplot2d([solaprox(x,10),soana(x)],[x,-2,2],
          [legend,"y aproximada","y exacta"],[xlabel,"x"],[ylabel,"y"]);
(%t16)  << Graphics >>
```

Finalmente, resolvemos el problema numéricamente con el comando `rk` que ya hemos usado varias veces

```
(%o16) (%i17) load(dynamics)$
(%i18) solnum: rk([z,2*x*z+2*y],[y,z],[1,0],[x,0,2,0.02])$
       soly:create_list([solnum[i][1],solnum[i][2]],i,1,length(solnum))$
```

y dibujamos las soluciones aproximada, exacta y numérica, respectivamente:

```
(%i20) wxplot2d([solaprox(x,10),soana(x),[discrete,soly]],
         [x,0,2],[legend,"y aproximada","y exacta","y numerica"])$
(%t20)  << Graphics >>
```

El primer y tercer gráficos están representados en la figura 13.

Referencias

[1] Boiarchuk A.K., Golovach G.P. *Problemas Resueltos: Ecuaciones diferenciales* (Anti-Demidovich) Vol, 8, 9 y 10, Editorial URSS, 2002.

[2] Braun M. *Differential Equations and their Applications*. Springer Verlag, 1993.

[3] A. F. Filipov *Problemas de ecuaciones diferenciales.* Editorial URSS, 2007

[4] Rainville E.D., Bedient P.E. y Bedient R.E. *Ecuaciones diferenciales.* Prentice Hall, 8ª edición, 1998.

[5] Simmons, G.F. *Ecuaciones diferenciales: con aplicaciones y notas históricas.* McGraw-Hill, 1993.