■4.-El algoritmo de ordenación rápida es un algoritmo de ordenación de listas en el que cada elemento es comparado con uno fijo de la lista, por ejemplo el primero. Si un elemento es menor que el primero se guarda en una lista que llamaremos lista1, y si no se guarda en otra que llamaremos lista2. Con cada una de esas listas se repite el proceso; asi sucesivamentehasta que en las particiones sucesivas sólo quede un elemento.

Programar el algoritmo anterior en una función recursiva que se llame Ordena , y que tenga como argumentos una lista de números

■ 5.-A continuación hemos definido la función ProdExt que da el producto hemisimétrico de tensores hemisimétricos. Explica cada instrucción razonadamente.

```
ProdExt[w[i__], w[j__]] :=
Signature[Join[List[i], List[j]]] *
Apply[w, Sort[Join[List[i], List[j]]]];
SetAttributes[ProdExt, Flat];
ProdExt[t1_ + t2_, t3_] := ProdExt[t1, t3] + ProdExt[t2, t3];
ProdExt[a_. x_w, b_. y_w] := Times[a, b, ProdExt[x, y]]
```

CAPÍTULO V: APLICACIONES AL CÁLCULO. ECUACIONES DIFERENCIALES Y SISTEMAS DE ECUACIONES DIFERENCIALES.

Se llama ecuación diferencial a una ecuación del tipo

$$F(x_1,\ldots,x_n,y_1,\ldots,y_m,\frac{\partial y_1}{\partial \mathrm{x1}},\ \ldots.\frac{\partial y_1}{\partial \mathrm{xn}},\ \ldots,\frac{\partial^{|\alpha|}}{\partial x_1^{\alpha_1}\ldots\partial x_n^{\alpha_n}})=0$$

en la que aparecen ligadas n variables independientes $x_1,\ldots,x_n$, m variables dependientes $y_1,\ldots,y_m$, y las derivadas parciales de las variables dependientes respecto de las n variables independientes. Resolver la ecuación es encontrar funciones $y_1=f_1(x_1,\ldots,x_n),\ldots,y_m=f_m(x_1,\ldots,x_n)$ tales que al ser sustituidas en la ecuación esta se verifique idénticamente.

Si sólo aparece una variable independiente diremos que la ecuación es una ecuación diferencial ordinaria (EDO u ODE en inglés). En caso contrario, diremos que la ecuación es una ecuación diferencial en derivadas parciales (EDP o PDE en inglés).

Llamaremos orden de la ecuación al orden de la mayor derivada que aparezca.

Mathematica puede resolver ecuaciones diferenciales y sistemas de ecuaciones diferenciales y dependiendo de su dificultad, se pueden obtener soluciones simbólicas o bien soluciones aproximadas con precisión arbitraria.

A continuación veremos algunos rudimentos sobre la resolución de ecuaciones diferenciales con Mathematica. Puedes encontrar más información en la ayuda y en particular en los tutoriales de la ayuda tutorial/DSolveOverview, tutorial/NDSolveOverview y tutorial/NumericalSolutionOfDifferentialEquations.

5.1.RESOLUCIÓN SIMBÓLICA DE ODE Y PDE

La función de *Mathematica* que resuelve de modo exacto ecuaciones diferenciales es DSolve

Algunas cuestiones respecto a la definición de DSolve son las siguientes:

Respecto de las ecuaciones ordinarias:
• Las ecuaciones lineales con coeficientes constantes se resuelven utilizando exponenciación matricial.
• Lasecuaciones de segundo orden lineales con coeficientes variables, cuyas soluciones se pueden expresar en términos de funciones elementales y sus integrales, se resuelven utilizando el algoritmo de Kovacic
• Las ecuaciones lineales con coeficientes polinomios se resuelven en términos de funciones especiales usando tranformaciones de Mellin.
• Siempre que se pueda , las ecuaciones no lineales se resuelven utilizando reducciones por simetria. Para ecuaciones de primer orden se utilizan técnicas clásicas y para ecuaciones de segundo orden y sistemas se utilizan técnicas de Bocharov.

Para EDP se utilizan separación de variables y reducciones por simetrias. DSolve integra ecuaciones con dos o más variables independientes y una variable dependiente. Puede resolver la mayoria de las ecuaciones de primer orden y un número limitado de ecuaciones de segundo orden, habitualmente aquellas que se encuentran en libros standard de referencia.

DSolve ocupa alrededor de 300 páginas en código de Mathematica y 200 en código C.

5.1.1 RESOLUCION SIMBÓLICA DE EDO

El comando para resolver ecuaciones diferenciales es DSolve y su sintaxis es la siguiente:

DSolve[ecuacion,y[x],x]
o
DSolve[ecuacion,y,x]

La diferencia entre usar DSolve de un modo u otro reside en la forma en que se devuelven las soluciones. En el primero las soluciones se obtienen en forma de regla de transformación, mientras que en la segunda se obtienen en forma de función pura.

■ Veamos un ejemplo:

```
sol1 = DSolve[y'[x] == a y[x] + 1, y[x], x]
```

$$\left\{\left\{y[x] \to -\frac{1}{a} + E^{a\,x}\,C[1]\right\}\right\}$$

Devuelve la solución en forma de regla de tranformación. Si queremos utilizar la solución como una función en cálculos posteriores, lo haremos utilizando la solución como una regla de sustitución local :

```
y[x] + 2 y'[x] /. sol1[[1]]
```

$$-\frac{1}{a} + E^{a\,x}\,C[1] + 2\,y'[x]$$

La regla de sustitución se aplica sólo a y[x] y no a y'[x]

```
sol2 = DSolve[y'[x] == a y[x] + 1, y, x]
```

$$\left\{\left\{y \to \text{Function}\left[x, -\frac{1}{a} + E^{a\,x}\,C[1]\right]\right\}\right\}$$

Devuelve la solución en forma de regla de transformación, pero en modo puro. Esto tiene la ventaja que cuando utilizamos la solución como una regla de sustitución local, se sustituye no sólo y[x], sino también y'[x],y[2]...

```
y[x] + 2 y'[x] /. sol2
```

$$\left\{-\frac{1}{a}+E^{a x} C[1]+2 a E^{a x} C[1]\right\}$$

■ La solución general de una EDO de orden n depende de n constantes arbitrarias. Por defecto estas constantes se denotan C[1], ...,C[n]

■ Podemos indicar condiciones iniciales sobre las soluciones, añadiendolas a la lista de ecuaciones. Si añadimos suficientes condiciones iniciales, los valores de las constantes C[1], ...,C[n] quedan completamente determinados

```
Clear[y]
```

```
sol3 = DSolve[{y'[x] == a y[x] + 1, y[0] == 0},
  y[x], x]
```

$$\left\{\left\{y[x] \to \frac{-1+e^{a x}}{a}\right\}\right\}$$

■ También podemos indicar condiciones de contorno. A diferencia de los problemas de condiciones iniciales, que tienen solución única los de contorno pueden no tener solución o tener infinitas soluciones.

El problema siguiente tiene infinitas soluciones

```
DSolve[{y''[x] + 4 * y[x] == 0, y[0] == 0, y[Pi] == 0}, y, x]
```

```
DSolve::bvsing :
  Unable to resolve some of the arbitrary constants in the general solution using the given boundary
      conditions. It is possible that some of the conditions have
      been specified at a singular point for the equation. ≫
{{y → Function[{x}, C[2] Sin[2 x]]}}
```

■ DSolve puede resolver ODE lineales de coeficientes constantes de cualquier orden. También puede resolver muchas ecuaciones lineales hasta orden 2 con coeficientes no constantes. DSolve incorpora rutinas para resolver la mayoria de las ecuaciones no lineales, cuyas soluciones aparecen en libros de soluciones.

■ Dependiendo de la versión de Mathematica que estemos utilizando es necesario cargar el paquete Calculus`D-Solve`, para incorporar a la función DSolve del núcleo algunas rutinas más avanzadas.

■ Encontrar fórmulas exactas para las soluciones de ecuaciones diferenciales es un problema dificil. De hecho hay unos pocos tipos para los que podemos encontrar soluciones, al menos en términos de funciones elementales.

■ Muchas de las funciones especiales que veremos se definen como soluciones de ecuaciones diferenciales.

■ 5.1.1.1 Ecuaciones diferenciales ordinarias lineales

Unas de las ecuaciones diferenciales más estudiadas son las ecuaciones diferenciales lineales, ecuaciones en que "y" y sus derivadas aparecen sólo linealmente.

■ 5.1.1.1.1 Ecuaciones diferenciales ordinarias lineales de primer orden

La forma más general para una ecuación de este tipo es:

y'[x]+a[x]* y[x]=b[x].

Si b[x]=0 decimos que la ecuación es homogénea.

■ Veamos un ejemplo

```
sol4 = DSolve[y'[x] - x y[x] == 0, y[x], x]
```

$$\left\{\left\{y[x] \to E^{\frac{x^2}{2}}\, C[1]\right\}\right\}$$

■ Si consideramos la misma ecuación no homogenea obtenemos una solución mucho más complicada

```
sol5 = DSolve[y'[x] - x y[x] == 1, y[x], x]
```

$$\left\{\left\{y[x] \to E^{\frac{x^2}{2}}\, C[1] + E^{\frac{x^2}{2}} \sqrt{\frac{\pi}{2}}\, \text{Erf}\left[\frac{x}{\sqrt{2}}\right]\right\}\right\}$$

En este caso la solución se expresa en términos de la función de error (Erf).

Una ecuación lineal de primer orden siempre se puede resolver haciendo integrales.

■ 5.1.1.1.2. Ecuaciones diferenciales lineales de segundo orden

Una ecuación diferencial lineal de segundo orden no siempre se puede resolver haciendo integrales. Pero muchas de las ecuaciones diferenciales lineales de orden 2 se pueden resolver en términos de funciones especiales, funciones introducidas en muchos casos para resolver específicamente tales ecuaciones.

Veamos algunos ejemplos:

■ La ecuación de Airy se resuelve en términos de las funciones de Airy

```
sol6 = DSolve[y''[x] - x y[x] == 0, y[x], x]
```

$$\{\{y[x] \to \text{AiryBi}[x]\, C[1] + \text{AiryAi}[x]\, C[2]\}\}$$

■ La siguiente ecuación se resuelve en términos de funciones de Bessel

```
sol7 = DSolve[y''[x] - Exp[x] y[x] == 0, y[x], x]
```

$$\left\{\left\{y[x] \to \text{BesselI}\left[0, 2\sqrt{E^x}\right] C[1] + 2\,\text{BesselK}\left[0, 2\sqrt{E^x}\right] C[2]\right\}\right\}$$

■ Sin embargo, a veces una ecuación lineal de orden 2 se puede resolver sólo utilizando funciones elementales, como ocurre con la ecuación de Euler

```
sol8 = DSolve[x^2 y''[x] + y[x] == 0, y[x], x]
```

$$\left\{\left\{y[x] \to x^{\frac{1}{2}\left(1-I\sqrt{3}\right)}\, C[1] + x^{\frac{1}{2}\left(1+I\sqrt{3}\right)}\, C[2]\right\}\right\}$$

■ 5.1.1.1.3. Ecuaciones diferenciales lineales de orden superior

Para las ecuaciones de orden superior a dos las funciones necesarias para resolver incluso ecuaciones sencillas, pueden ser muy complicadas. Para las de orden tres la función generalizada MeijerG resuelve el problema en algunos casos. Para ecuaciones de orden cuatro o superior en la mayoria de los casos no es posible en general encontrar funciones adecuadas que resuelvan un número amplio de ecuaciones.

■

```
sol9 = DSolve[y⁽³⁾[x] + x y[x] == 0, y[x], x]
```

$$\left\{\left\{y[x] \to C[1]\,\text{HypergeometricPFQ}\left[\{\}, \left\{\frac{1}{2}, \frac{3}{4}\right\}, -\frac{x^4}{64}\right] + \frac{x\,C[2]\,\text{HypergeometricPFQ}\left[\{\}, \left\{\frac{3}{4}, \frac{5}{4}\right\}, -\frac{x^4}{64}\right]}{2\sqrt{2}} + \frac{1}{8}x^2\,C[3]\,\text{HypergeometricPFQ}\left[\{\}, \left\{\frac{5}{4}, \frac{3}{2}\right\}, -\frac{x^4}{64}\right]\right\}\right\}$$

$$\left\{\left\{y[x] \to \frac{C[1]\,\text{HypergeometricPFQ}\left[\{\}, \left\{\frac{1}{2}, \frac{3}{4}\right\}, -\frac{x^4}{64}\right]}{\sqrt{\pi}\,\text{Gamma}\left[\frac{3}{4}\right]} + \frac{\left(x^4\right)^{1/4} C[2]\,\text{HypergeometricPFQ}\left[\{\}, \left\{\frac{3}{4}, \frac{5}{4}\right\}, -\frac{x^4}{64}\right]}{\pi} + \frac{\sqrt{x^4}\;C[3]\,\text{HypergeometricPFQ}\left[\{\}, \left\{\frac{5}{4}, \frac{3}{2}\right\}, -\frac{x^4}{64}\right]}{4\sqrt{\pi}\,\text{Gamma}\left[\frac{5}{4}\right]}\right\}\right\}$$

■ La siguiente ecuación requiere funciones de MeijerG

```
sol10 = DSolve[y⁽³⁾[x] + Exp[x] y[x] == 0, y[x], x]
```

```
{{y[x] → C[1] HypergeometricPFQ[{}, {1, 1}, -ℯ^x] +
     C[2] MeijerG[{{}, {}}, {{0, 0}, {0}}, -ℯ^x] +
     C[3] MeijerG[{{}, {}}, {{0, 0, 0}, {}}, ℯ^x]}}
```

■ 5.1.1.2. Ecuaciones diferenciales ordinarias no lineales

Sólo unos pocos tipos de ecuaciones no lineales se pueden resolver en términos de funciones elementales. Sin embargo, DSolve incluye procedimientos generales que permiten resolver casi todas las ecuaciones de este tipo cuyas soluciones se pueden encontrar en libros stándar de soluciones.

Veamos algunos ejemplos:

■ Las ecuaciones no lineales de primer orden donde no aparece x son fáciles de resolver.

```
sol11 = DSolve[y'[x] - y[x]^2 == 0, y[x], x]
```

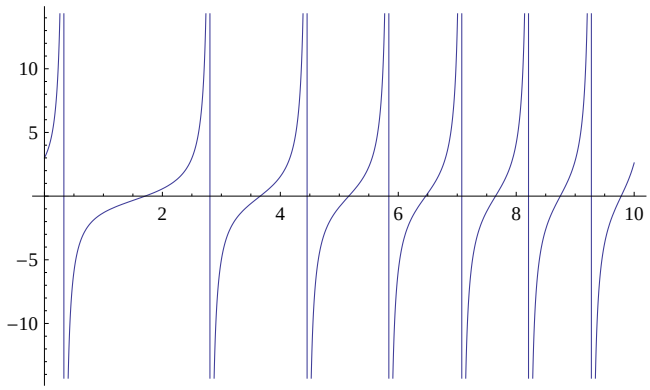
$$\left\{\left\{y[x] \to \frac{1}{-x - C[1]}\right\}\right\}$$

■ La ecuación de Ricatti ya tiene una solución considerablemente complicada y se resuelve en términos de funciones especiales

```
sol12 = DSolve[{y'[x] - y[x]^2 == x, y[0] == 3}, y, x]
```

$$\left\{\left\{y \to \text{Function}\left[\{x\},\ \left(-6\,x^{3/2}\,\text{BesselJ}\left[-\frac{2}{3},\frac{2\,x^{3/2}}{3}\right]\text{Gamma}\left[\frac{1}{3}\right]+3^{1/3}\,x^{3/2}\,\text{BesselJ}\left[-\frac{4}{3},\frac{2\,x^{3/2}}{3}\right]\text{Gamma}\left[\frac{2}{3}\right]+3^{1/3}\,\text{BesselJ}\left[-\frac{1}{3},\frac{2\,x^{3/2}}{3}\right]\text{Gamma}\left[\frac{2}{3}\right]-3^{1/3}\,x^{3/2}\,\text{BesselJ}\left[\frac{2}{3},\frac{2\,x^{3/2}}{3}\right]\text{Gamma}\left[\frac{2}{3}\right]\right)\Big/\left(2\,x\left(3\,\text{BesselJ}\left[\frac{1}{3},\frac{2\,x^{3/2}}{3}\right]\text{Gamma}\left[\frac{1}{3}\right]-3^{1/3}\,\text{BesselJ}\left[-\frac{1}{3},\frac{2\,x^{3/2}}{3}\right]\text{Gamma}\left[\frac{2}{3}\right]\right)\right)\right]\right\}\right\}$$

```
Plot[y[x] /. sol12[[1]], {x, 0, 10}]
```



■ La ecuación de Bernoulli, sin embargo tiene una solución mucho más sencilla.

```
sol12 = DSolve[y'[x] - x y[x]^2 - y[x] == 0, y, x]
```

$$\left\{\left\{y \to \text{Function}\left[\{x\},\ -\frac{e^x}{-e^x + e^x\,x - C[1]}\right]\right\}\right\}$$

■ La ecuación de Abel se puede resolver pero sólo implícitamente

```
sol13 = DSolve[{y'[x] + x y[x]^3 + y[x]^2 == 0,
   y[0] == 0}, y, x]
```

Solve::tdep :
The equations appear to involve the variables to be solved for in an essentially non−algebraic way.

$$\text{Solve}\left[\frac{1}{2}\left(\frac{2\,\text{ArcTanh}\left[\frac{-1-2\,x\,y[x]}{\sqrt{5}}\right]}{\sqrt{5}}+\text{Log}\left[\frac{-1-x\,y[x]\ (-1-x\,y[x])}{x^2\,y[x]^2}\right]\right) == C[1] - \text{Log}[x],\ y[x]\right]$$

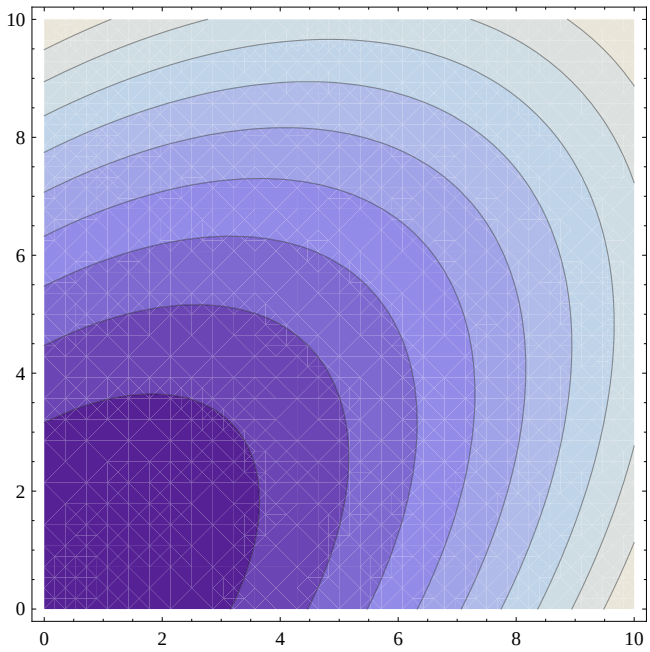
■ 5.1.1.3 . APLICACIONES DE ECUACIONES DIFERENCIALES

■ 5.1.1.3.1. TRAYECTORIAS ORTOGONALES A UNA FAMILIA DE CURVAS

Encontrar la familia de trayectorias ortogonales a la familia de elipses
$x^2+y^2 - x*y = c^2$. Dibujar la familia de elipses y sus trayectorias ortogonales

Para dibujar la familia de elipses utilizamos la instrucción ContourPlot

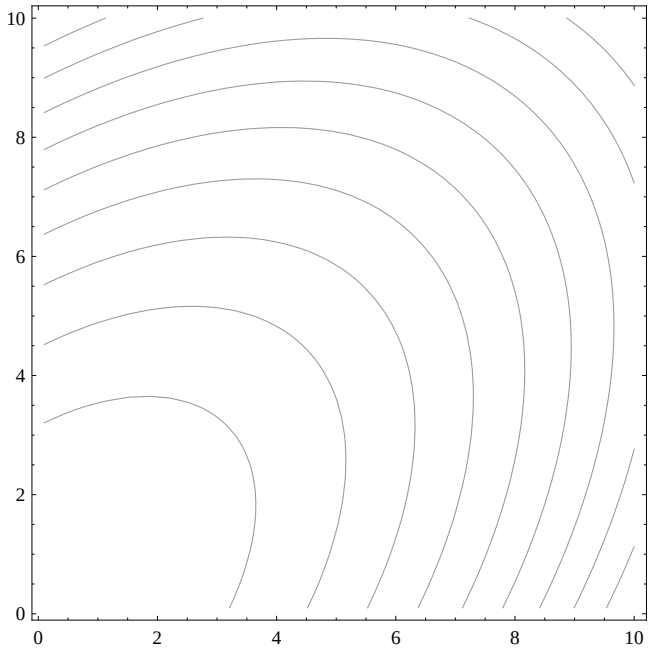
```
dibu = ContourPlot[x^2 + y^2 - x * y, {x, 0, 10}, {y, 0, 10}]
```



Valores más claros corresponden a valores mayores de z

Si queremos eliminar los colores escribimos ContourShading->False

```
fam1 = ContourPlot[x^2 + y^2 - x * y,
  {x, 0.1, 10}, {y, 0.1, 10}, ContourShading -> False]
```



Derivando implícitamente en la ecuación de las elipses obtenemos la ecuación diferencial que verifican

```
eq1 = Dt[x^2 + y^2 - x * y, x] /. y -> y[x] /. (Dt[y[x], x] -> D[y[x], x])
```

$$2\,x - y[x] - x\,y'[x] + 2\,y[x]\,y'[x]$$

Despejamos en eq1 el valor de y'[x]

```
Solve[eq1 == 0, y'[x]]
```

$$\left\{\left\{y'[x] \to \frac{-2\,x + y[x]}{-x + 2\,y[x]}\right\}\right\}$$

La familia de trayectorias ortogonales a fam1 tiene pendiente -1/m1 , si la de fam1 es m1. Por tanto la ecuación diferencial que verifican las trayectorias ortogonales es

$$y'[x] == \frac{x-2y[x]}{-2x+y[x]}$$

Resolvemos la ecuación diferencial

```
sol = DSolve[y'[x] == (x - 2 y[x])/(-2 x + y[x]), y, x]
```

$$\left\{\left\{y \to \text{Function}\left[\{x\}, \frac{1}{2}\left(-2x - \frac{e^{2C[1]}}{3^{1/3}\left(18 e^{2C[1]} x + \sqrt{3}\sqrt{e^{6C[1]} + 108 e^{4C[1]} x^2}\right)^{1/3}} + \frac{\left(18 e^{2C[1]} x + \sqrt{3}\sqrt{e^{6C[1]} + 108 e^{4C[1]} x^2}\right)^{1/3}}{3^{2/3}}\right)\right]\right\},$$

$$\left\{y \to \text{Function}\left[\{x\}, -x + \frac{\left(1 + i\sqrt{3}\right) e^{2C[1]}}{4\,3^{1/3}\left(18 e^{2C[1]} x + \sqrt{3}\sqrt{e^{6C[1]} + 108 e^{4C[1]} x^2}\right)^{1/3}} - \frac{1}{4\,3^{2/3}}\left(1 - i\sqrt{3}\right)\left(18 e^{2C[1]} x + \sqrt{3}\sqrt{e^{6C[1]} + 108 e^{4C[1]} x^2}\right)^{1/3}\right]\right\},$$

$$\left\{y \to \text{Function}\left[\{x\}, -x + \frac{\left(1 - i\sqrt{3}\right) e^{2C[1]}}{4\,3^{1/3}\left(18 e^{2C[1]} x + \sqrt{3}\sqrt{e^{6C[1]} + 108 e^{4C[1]} x^2}\right)^{1/3}} - \frac{1}{4\,3^{2/3}}\left(1 + i\sqrt{3}\right)\left(18 e^{2C[1]} x + \sqrt{3}\sqrt{e^{6C[1]} + 108 e^{4C[1]} x^2}\right)^{1/3}\right]\right\}\right\}$$

```
sol[[1]]
```

$$\left\{y \to \text{Function}\left[\{x\}, -x + \frac{e^{2C[1]}}{3^{1/3}\left(-9 e^{2C[1]} x + \sqrt{3}\sqrt{-e^{6C[1]} + 27 e^{4C[1]} x^2}\right)^{1/3}} + \frac{\left(-9 e^{2C[1]} x + \sqrt{3}\sqrt{-e^{6C[1]} + 27 e^{4C[1]} x^2}\right)^{1/3}}{3^{2/3}}\right]\right\}$$

```
solu = y[x] /. sol[[1]]
```

$$\frac{1}{2}\left(-2x - \frac{e^{2C[1]}}{3^{1/3}\left(18 e^{2C[1]} x + \sqrt{3}\sqrt{e^{6C[1]} + 108 e^{4C[1]} x^2}\right)^{1/3}} + \frac{\left(18 e^{2C[1]} x + \sqrt{3}\sqrt{e^{6C[1]} + 108 e^{4C[1]} x^2}\right)^{1/3}}{3^{2/3}}\right)$$

```
solu1 = solu //. {E^x_ → x}
```

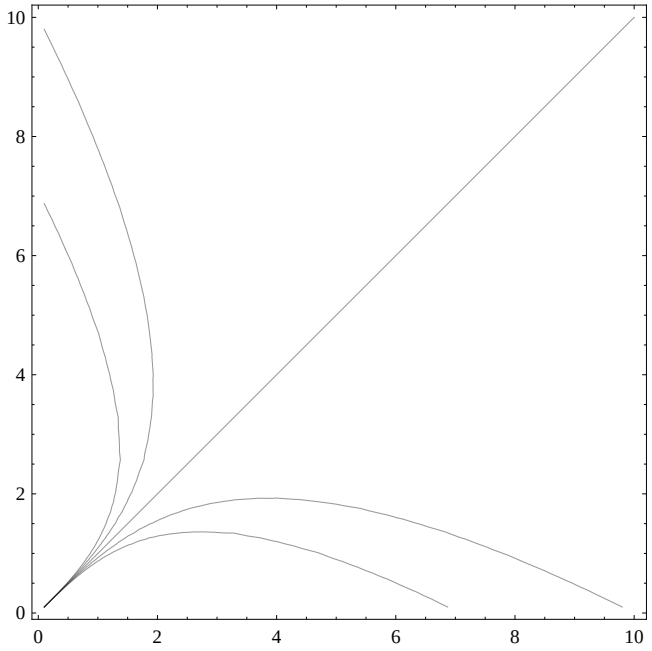
$$\frac{1}{2}\left(-2x-\frac{2\,C[1]}{3^{1/3}\left(36x\,C[1]+\sqrt{3}\,\sqrt{6\,C[1]+432x^2\,C[1]}\right)^{1/3}}+\frac{\left(36x\,C[1]+\sqrt{3}\,\sqrt{6\,C[1]+432x^2\,C[1]}\right)^{1/3}}{3^{2/3}}\right)$$

Despejo C[1] para dibujarlo como un gráfico de contorno

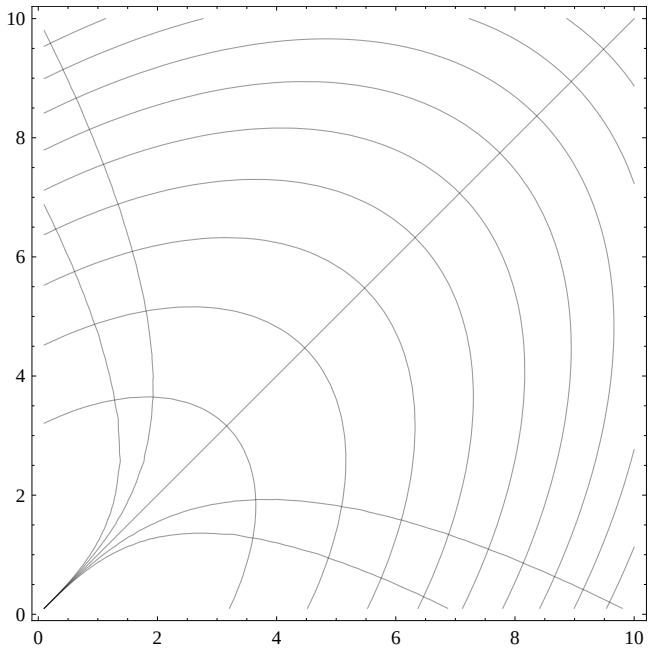
```
soludes = Solve[y == solu, C[1]]
```

```
$Aborted
```

```
fam2 = ContourPlot[(y - x)/(y + x)^3, {x, 0.1, 10}, {y,
    0.1, 10}, ContourShading -> False]
```



```
Show[fam1, fam2]
```



■ **5.1.1.3.2. CAMPOS DE DIRECCIONES**

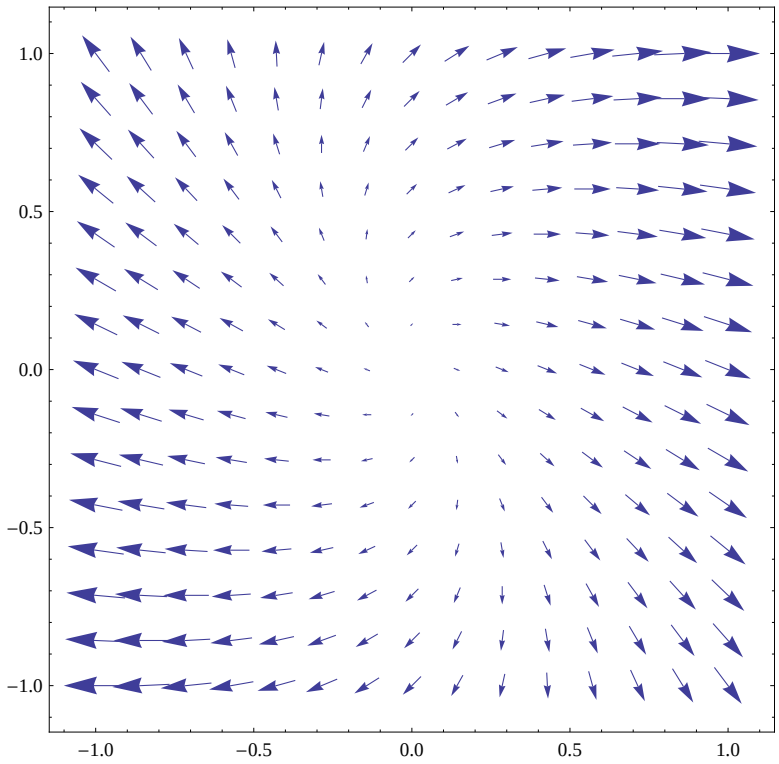
Considerar el sistema de ecuaciones de primer orden

x'[t]==5x[t]+2y[t]
y'[t]==-2x[t]+2y[t]

Este sistema define un campo de direcciones en R^2 dado por (x'[t],y'[t]).
Resolver el sistema de ecuaciones diferenciales equivale a encontrar las
curvas tangentes a ese campo de direcciones.

Vamos a resolver el sistema y a dibujar el campo y las soluciones del sistema, y comprobaremos que efectivamente son tangentes.

```
dibu1 = VectorPlot[{5 x + 2 y, -2 x + 2 y}, {x, -1, 1}, {y, -1, 1}]
```



■ Resolvemos el sistema

```
sol = DSolve[{x'[t] == 5 x[t] + 2 y[t], y'[t] == -2 x[t] + 2 y[t]}
, {x[t], y[t]}, t]
```

$$\left\{\left\{x[t] \to \frac{4\, e^{7t/2} C[2] \operatorname{Sin}\left[\frac{\sqrt{7}\, t}{2}\right]}{\sqrt{7}} + \frac{1}{7} e^{7t/2} C[1] \left(7 \operatorname{Cos}\left[\frac{\sqrt{7}\, t}{2}\right] + 3\sqrt{7}\, \operatorname{Sin}\left[\frac{\sqrt{7}\, t}{2}\right]\right),\right.$$

$$\left. y[t] \to -\frac{4\, e^{7t/2} C[1] \operatorname{Sin}\left[\frac{\sqrt{7}\, t}{2}\right]}{\sqrt{7}} + \frac{1}{7} e^{7t/2} C[2] \left(7 \operatorname{Cos}\left[\frac{\sqrt{7}\, t}{2}\right] - 3\sqrt{7}\, \operatorname{Sin}\left[\frac{\sqrt{7}\, t}{2}\right]\right)\right\}\right\}$$

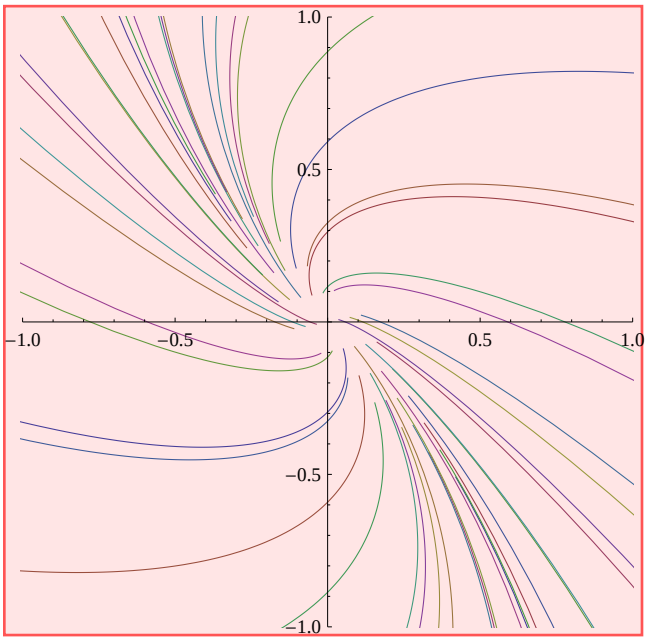
Vamos a dibujar unas cuantas curvas solución para algunos valores de las constantes C[1],C[2]

Primero generamos el array en la forma {{x1[t],y1[t]},{x2[t],y2[t]},...} usando Table y eliminamos las llaves que sobran con Flatten

```
curarray = Flatten[Table[{x[t], y[t]} /. sol /. {C[1] -> i, C[2] -> j},
        {i, -6, 6, 2}, {j, -6, 6, 2}], 2];
```

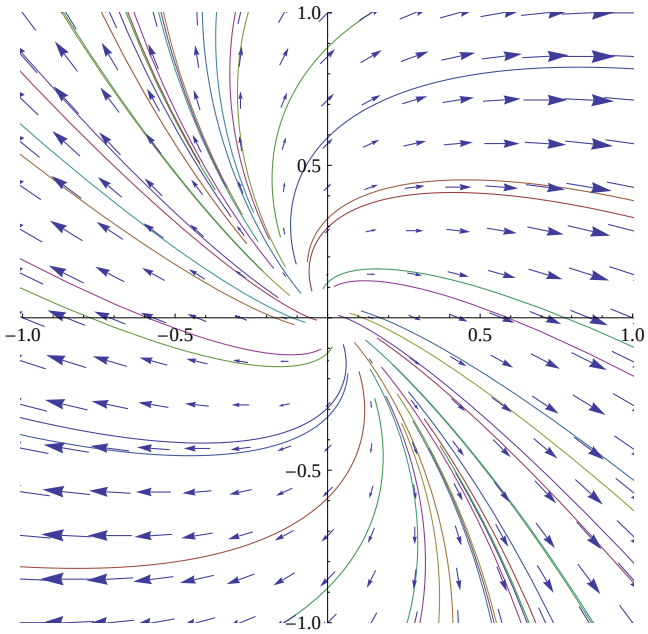
Dibujamos la s curvas {x[t],y[t]} con ParametricPlot

```
dibu2 =
 ParametricPlot[Evaluate[curarray], {t, -1, 1}, PlotRange -> {{-1, 1}, {-1, 1}}]
```



Le pedimos que nos muestre juntos las curvas solución y el campo

```
Show[dibu2, dibu1]
```



Se observa como efectivamente el campo y las curva solución son tangentes

■ 5.1.1.3.3. DEPENDENCIA DE LAS CONDICIONES INICIALES

```
rule = DSolve[{y'[x] - y[x]^2 == x, y[0] == a}, y, x]
```

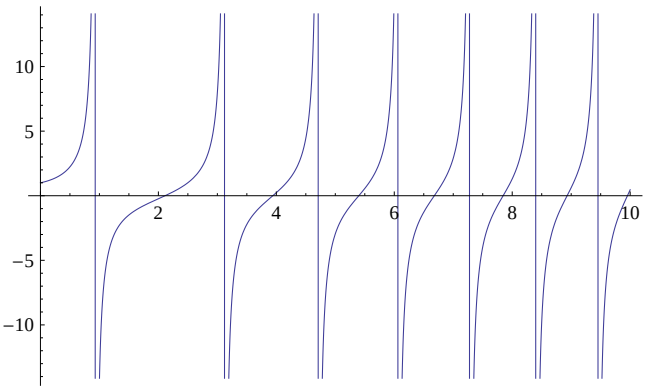
$$\left\{\left\{y \to \text{Function}\left[\{x\},\right.\right.\right.$$
$$\left(-2\,a\,x^{3/2}\,\text{BesselJ}\left[-\frac{2}{3},\ \frac{2\,x^{3/2}}{3}\right]\text{Gamma}\left[\frac{1}{3}\right] + 3^{1/3}\,x^{3/2}\,\text{BesselJ}\left[-\frac{4}{3},\ \frac{2\,x^{3/2}}{3}\right]\text{Gamma}\left[\frac{2}{3}\right] +\right.$$
$$3^{1/3}\,\text{BesselJ}\left[-\frac{1}{3},\ \frac{2\,x^{3/2}}{3}\right]\text{Gamma}\left[\frac{2}{3}\right] - 3^{1/3}\,x^{3/2}\,\text{BesselJ}\left[\frac{2}{3},\ \frac{2\,x^{3/2}}{3}\right]\text{Gamma}\left[\frac{2}{3}\right]\Big) \Big/$$
$$\left.\left.\left.\left(2\,x\left(a\,\text{BesselJ}\left[\frac{1}{3},\ \frac{2\,x^{3/2}}{3}\right]\text{Gamma}\left[\frac{1}{3}\right] - 3^{1/3}\,\text{BesselJ}\left[-\frac{1}{3},\ \frac{2\,x^{3/2}}{3}\right]\text{Gamma}\left[\frac{2}{3}\right]\right)\right)\right]\right\}\right\}$$

```
a = 1
```

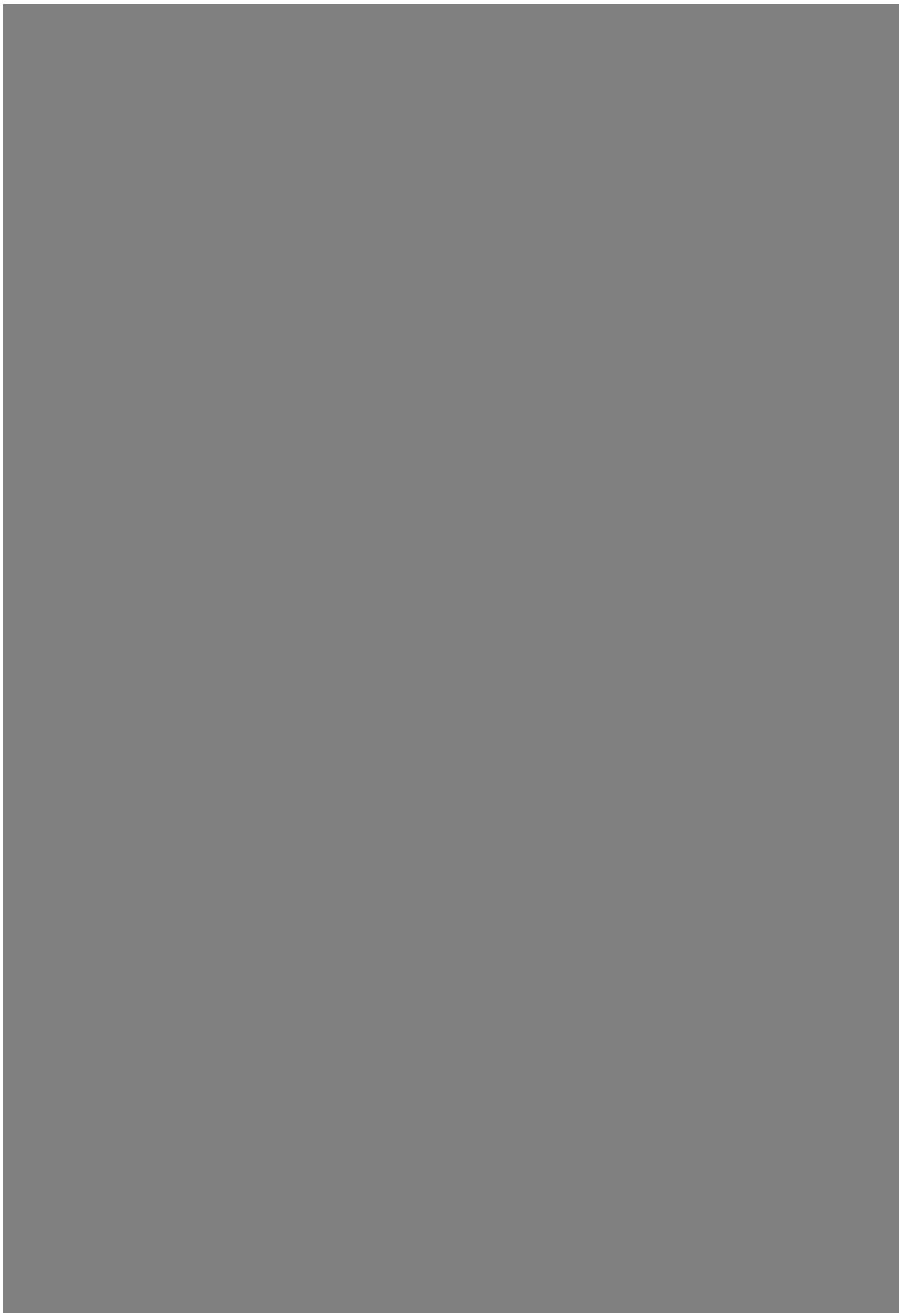
```
1
```

```
Clear[a]
```

```
Plot[y[x] /. rule[[1]], {x, 0, 10}]
```




```
Manipulate[
 Plot[(-2 a x^(3/2) BesselJ[-2/3, (2 x^(3/2))/3] Gamma[1/3] + 3^(1/3) x^(3/2) BesselJ[-4/3, (2 x^(3/2))/3] Gamma[2/3] +
     3^(1/3) BesselJ[-1/3, (2 x^(3/2))/3] Gamma[2/3] - 3^(1/3) x^(3/2) BesselJ[2/3, (2 x^(3/2))/3] Gamma[2/3]) /
   (2 x (a BesselJ[1/3, (2 x^(3/2))/3] Gamma[1/3] - 3^(1/3) BesselJ[-1/3, (2 x^(3/2))/3] Gamma[2/3])),
  {x, 0, 5}, PlotRange -> {-20, 20}], {{a, 0, "y(0)"}, 0, 20}]
```



5.1.1.5 EJERCICIOS

■ 1.-Resolver las siguientes ecuaciones diferenciales ordinarias y calcular la derivada de la solución en el punto t=0.

a) x'[t]=E^t/(1+E^t) x[t]

b) x'[t]+x[t]=t* x[t]

■ 2.-
a) Resolver la ecuación diferencial y'[x]-2y[x]=4x con condición inicial y[0]=0.

b) Dibujar la curva solución.

■3.- Calcular las trayectorias orotogonales a la familia de curvas x^3+y^3·=a^3. Dibujar ambas familias.

■ 4.- Calcular las trayectorias ortogonales a la familia de parábolas y=ax^2. Dibujar ambas familias.

■ 5.- Considerar la ecuación y'[x]+2y[x]==Exp[-x].

a) Dibujar el campo asociado a la ecuación.
b) Resolver la ecuación.
c) Dibujar las curvas solución de la ecuación para algunos valores de C[1].
d) Observar la tangencia entre el campo asociado y las curvas solución.

5.1.2. RESOLUCION SIMBÓLICA DE EDP

5.2.1. GENERALIDADES SOBRE EDP's

• DSolve resuelve también ecuaciones diferenciales en derivadas parciales. La sintaxis es la misma que para EDO.

DSolve[ecuacion,y[x1,x2,..],{x1,x2,...,xn}]
o
DSolve[ecuacion,y,{x1,x2,...,xn}]

En la tabla de abajo se pueden encontrar algunos ejemplos bien conocidos de PDE' s. DSolve da soluciones simbólicas para estas ecuaciones, con ciertas limitaciones, en especial para ecuaciones de segundo orden.

Nombre de la ecuación	Forma General	Clasificación
Ecuación del transporte	$\frac{\partial u}{\partial x} + c\,\frac{\partial u}{\partial y} = 0$, con c constante	EDP lineal de primer orden
Ecuación de Burger	$\frac{\partial u}{\partial t} + u\,\frac{\partial u}{\partial x} = 0$	EDP quasilineal de primer orden
Ecuación Eikonal	$\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 = 1$	EDP no linear de primer orden
Ecuación de Laplace	$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$	EDP lineal eliptica de segundo orden
Ecuación de ondas	$\frac{\partial^2 u}{\partial x^2} = c^2\,\frac{\partial^2 u}{\partial t^2}$ donde c es una constante	EDP lineal hiperbólica de segundo orden
Ecuación del calor	$\frac{\partial^2 u}{\partial x^2} = k\,\frac{\partial u}{\partial t}$ donde k es la difusividad térmica	EDP lineal parabólica de segundo orden

Veamos algunos ejemplos:

■

```
DSolve[∂x1 y[x1, x2] + ∂x2 y[x1, x2] == 1/(x1 * x2),
    y[x1, x2], {x1, x2}]
```

```
{{y[x1, x2] → 1/(x1 - x2) (-Log[x1] + Log[x2] + x1 C[1][-x1 + x2] - x2 C[1][-x1 + x2])}}
```

■ O la misma ecuación pero expresando la solución en forma de función pura.

```
DSolve[∂x1 y[x1, x2] + ∂x2 y[x1, x2] == 1/(x1 x2), y, {x1, x2}]
```

```
{{y → Function[{x1, x2},
     1/(x1 - x2) (-Log[x1] + Log[x2] + x1 C[1][-x1 + x2] - x2 C[1][-x1 + x2])]}}
```

• Las matemáticas necesarias para resolver EDP son considerablemente más complicadas que las de EDO. Estudiar la teoría para resolverlas se escapa de nuestro objetivo en estas notas.

• Como ya hemos visto la solución de una EDO de orden n depende de n constantes arbitrarias que por defecto se llaman C[1],...,C[n]. En el caso de EDP, si podemos encontrar una solución general, esta puede depender, además de constantes, de funciones arbitrarias que también por defecto se denotan C[1],...

■ Veamos por ejemplo la ecuación de ondas unidimensional:

```
DSolve[c^2 ∂x,x y[t, x] - ∂t,t y[t, x] == 0, y[t, x], {t, x}]
```

```
{{y[t, x] → C[1][-√c^2 t + x] + C[2][√c^2 t + x]}}
```

La solución depende de dos funciones arbitrarias.

• Para una EDO siempre existe una solución general con la propiedad que añadiendo condiciones inciales elegimos valores concretos para las constantes arbitrarias. Para EDP esto deja de ser cierto, sólo podemos encontrar soluciones generales para EDP lineales y otros pocos tipos más.

• Otras EDP solo se pueden resolver cuando se dan condiciones iniciales o condiciones sobre la frontera, y aún así en la mayoria de los casos no se pueden dar las soluciones en términos de funciones elementales.

■ 5.2.2. ECUACIONES DE PRIMER ORDEN

Veamos algunos ejemplos:

■ "y" y sus derivadas aparecen solo linealmente, por tanto podemos encontrar una solución general.

```
DSolve[x1 ∂x1 y[x1, x2] + x2 ∂x2 y[x1, x2] == Exp[x1 x2],
    y[x1, x2], {x1, x2}]
```

$$\left\{\left\{y[x1, x2] \to \frac{1}{2}\left(\text{ExpIntegralEi}[x1\, x2] + 2\, C[1]\left[\frac{x2}{x1}\right]\right)\right\}\right\}$$

La solución general se obtiene en términos de funciones especiales

■ A veces podemos encontrar la solución general solo en términos de funciones elementales

```
DSolve[x1 ∂x1 y[x1, x2] + x2 ∂x2 y[x1, x2] == Exp[y[x1, x2]],
    y[x1, x2], {x1, x2}]
```

```
Solve::ifun :
 Inverse functions are being used by Solve, so some solutions may not
   be found; use Reduce for complete solution information.

Solve::ifun :
 Inverse functions are being used by Solve, so some solutions may not
   be found; use Reduce for complete solution information.
```

$$\left\{\left\{y[x1, x2] \to -\text{Log}\left[-\text{Log}[x1] - C[1]\left[\frac{x2}{x1}\right]\right]\right\}\right\}$$

■ Un ejemplo de ecuación no lineal de primer orden

```
DSolve[D[u[x, y], x] * D[u[x, y], y] == 1, u, {x, y}]
```

```
DSolve::nlpde :
  Solution requested to nonlinear partial differential equation. Trying to build a special solution.
```

$$\left\{\left\{u \to \text{Function}\left[\{x, y\}, C[1] + \frac{x}{C[2]} + y\, C[2]\right]\right\}\right\}$$

En este caso no es posible encontrar la solución general pero si se puede encontrar fácilmente una integral completa (una familia de soluciones dependiente de dos parámetros).

■ 5.2.3. ECUACIONES DE SEGUNDO ORDEN.

■ La ecuación de Laplace

```
solution = DSolve[{∂x,x u[x, y] + ∂y,y u[x, y] == 0}, u[x, y], {x, y}]
```

```
{{u[x, y] → C[1][i x + y] + C[2][-i x + y]}}
```

■ La ecuación de ondas

```
DSolve[{∂x,x u[x, t] - ∂t,t u[x, t] == 0}, u[x, t], {t, x}]
```

```
{{u[x, t] → C[1][-t + x] + C[2][t + x]}}
```

■ La ecuación del calor

```
DSolve[{∂x,x u[x, t] - ∂t u[x, t] == 0}, u[x, t], {t, x}]
```

```
DSolve[{-u^(0,1)[x, t] + u^(2,0)[x, t] == 0}, u[x, t], {t, x}]
```

5.2. RESOLUCION NUMERICA DE EDO Y EDP

Ya hemos visto que en muchos casos no es posible encontrar una solución general de EDP o EDO. Puede ocurrir también que los datos de la función y sus derivadas se obtengan de modo experimental. En estos casos debemos optar por obtener una solución aproximada mediante un método númerico. El comando para resolver numéricamente ecuaciones diferenciales es NDSolve. Encontrarás información más avanzada y más ejemplos en el tutorial de la ayuda NDSolve overview.

■ 5.2.1. EL COMANDO NDSOLVE Y ALGUNOS EJEMPLOS

• El comando de Mathematica para resolver ecuaciones diferenciales númericamente es NDSolve. La sintaxis es la siguiente:

NDSolve[{ecuación,c ondiciones iniciales},y,{x,xmin,xmax}]
encuentra una solución número ica para una ecuación con variable independiente x, con x en el rango xmin a xmax

Si la ecuación no es ordinaria, obviamente debemos especificar el rango de variación para las otras variables independientes.

- NDSolve da el resultado en forma de funciones de interpolación.
- NDSolve resuelve un amplio abanico de EDO y algunas EDP.
- La ecuación diferencial debe contener suficientes condiciones iniciales o condiciones frontera para poder determinar completamente las soluciones.

Condiciones iniciales o de frontera son tipicamente condiciones del tipo y[x0]==c0,y'[x0]==d0,.. pero pueden consistir en condiciones más complicadas.

- El punto x0 para el que escribimos las condiciones iniciales no tiene porque estar en el rango xmin a xmax.

■ Veamos un ejemplo:

```
DSolve[{x''[t] + x[t] == 0, x[0] == 1, x'[0] == 0}, x, t]
```

```
{{x → Function[{t}, Cos[t]]}}
```

En este caso la ecuación se resuelve de modo exacto, pero podemos resolverla de modo aproximado

```
sol = NDSolve[{x''[t] + x[t] == 0, x[0] == 1, x'[0] == 0}, x, {t, 0, 6 π}]
```

```
{{x → InterpolatingFunction[{{0., 18.8496}}, <>]}}
```

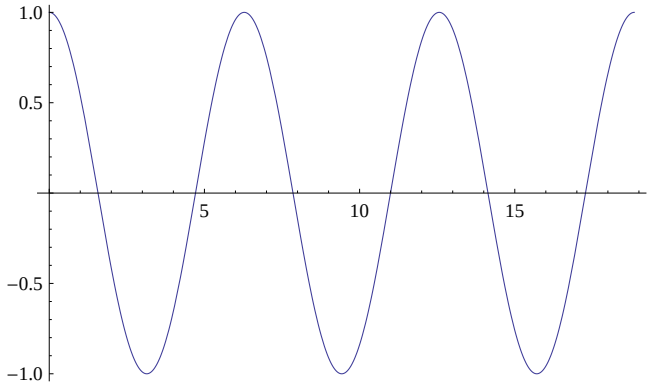
Si queremos obtener un valor concreto

```
x[π] /. sol
```

```
{-1.}
```

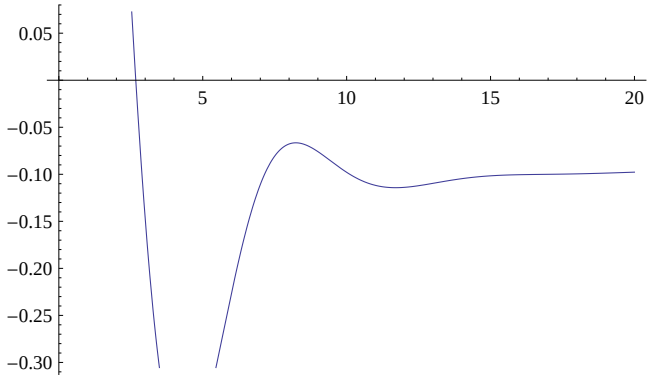
Podemos por ejemplo dibujarla

```
Plot[x[t] /. sol, {t, 0, 6 π}]
```



■ Otro ejemplo

```
solution = NDSolve[
   {y⁽³⁾[x] + y″[x] + y′[x] == -y[x]³, y[0] == 1, y′[0] == y″[0] == 0}, y, {x, 0, 20}];
Plot[Evaluate[y[x] /. solution], {x, 0, 20}]
```



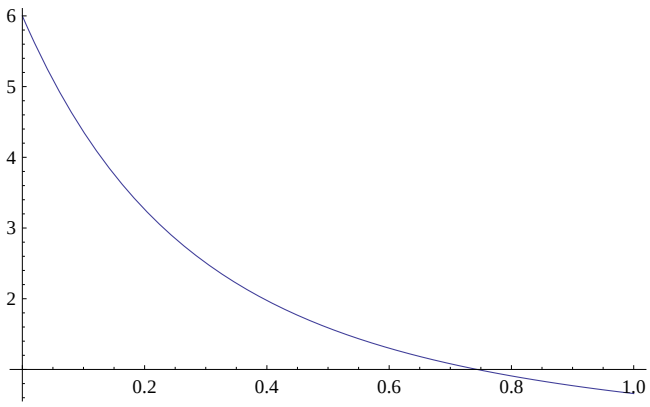
■ Un ejemplo de una ecuación de orden 3. Debemos dar condiciones iniciales hasta las derivadas de orden 2

```
NDSolve[{y'''[x] + 8 y''[x] + 17 y'[x] + 10 y[x] == 0,
  y[0] == 6, y'[0] == -20, y''[0] == 84}, y, {x, 0, 1}]
```

```
{{y → InterpolatingFunction[{{0., 1.}}, <>]}}
```

Dibujamos la solución

```
Plot[Evaluate[y[x] /. %], {x, 0, 1}]
```



■ Con una ecuación de orden 3 también se pueden dar condiciones iniciales en tres puntos

```
NDSolve[{y'''[x] + Sin[x] == 0, y[0] == 4, y[1] == 7, y[2] == 0}, y, {x, 0, 2}]
```

```
{{y → InterpolatingFunction[{{0., 2.}}, <>]}}
```

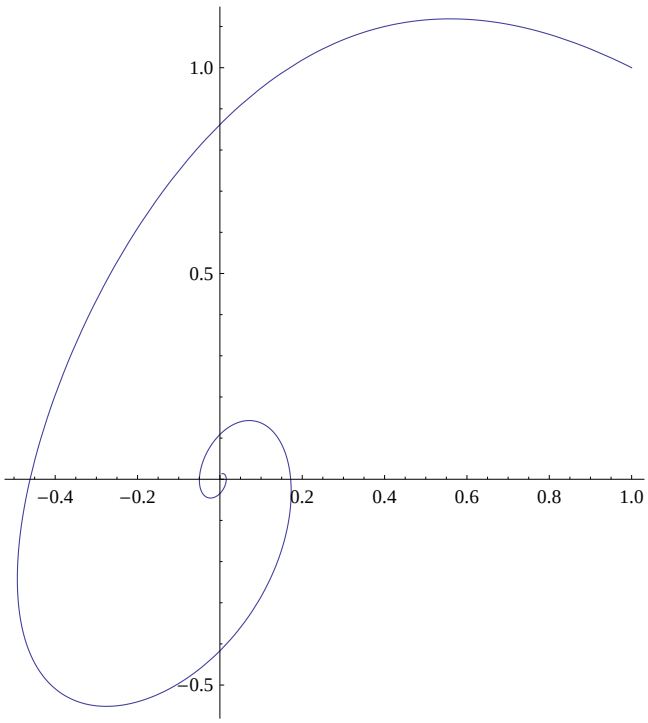
■ Podemos resolver también sistemas de ecuaciones diferenciales

```
sol = NDSolve[{x'[t] == -y[t] - x[t]^2,
   y'[t] == 2 x[t] - y[t], x[0] == y[0] == 1}, {x, y}, {t, 10}]
```

```
{{x → InterpolatingFunction[{{0., 10.}}, <>],
  y → InterpolatingFunction[{{0., 10.}}, <>]}}
```

Dibujamos la curva solución {x(t),y(t)} utilizando ParametricPlot

```
ParametricPlot[Evaluate[{x[t], y[t]} /. sol], {t, 0, 10}, PlotRange -> All]
```



■ Podemos resolver también resolver problemas de contorno y de condiciones iniciales para ecuaciones en derivadas parciales. Por ejemplo, la ecuación del calor unidimensional

```
rule = NDSolve[{∂t u[t, x] == ∂x,x u[t, x], u[0, x] == 0, u[t, 0] == Sin[t], u[t, 5] == 0},
  u, {t, 0, 10}, {x, 0, 5}]
```

```
{{u → InterpolatingFunction[{{0., 10.}, {0., 5.}}, <>]}}
```

Dibujamos la gráfica de evolución de la solución con el tiempo t

```
Manipulate[Plot[Evaluate[u[x, t] /. First[rule]],
  {x, 0, 5}, PlotRange → {-1, 1}], {t, 0, 10}]
```

■ 5.2.2. EL COMANDO INTERPOLATINGFUNCTION

InterpolatingFunction[dominio, tabla] representa una función aproximada cuyos valores se encuentran por interpolación

• dominio especifica el dominio de los datos con los que se construye la función de interpolación
• En el Out , sólo se devuelve explicitamente el dominio de InterpolatingFunction. El resto de los elementos se indican con <>
• Las funciones de interpolación se derivan de manera ordinaria usando D o Derivative
• InterpolatingFunction usa diferencias divididas para construir los polinomios interpoladores de Lagrange o Hermite
• `InterpolatingFunction[ … ][x]` devuelve el valor de la función de interpolación en un valor particular x.

■ Veamos un ejemplo

```
data = Table[{y^2, y}, {y, 0., 10.}]
```

```
{{0., 0.}, {1., 1.}, {4., 2.}, {9., 3.}, {16., 4.},
 {25., 5.}, {36., 6.}, {49., 7.}, {64., 8.}, {81., 9.}, {100., 10.}}
```

```
sqrt3 = Interpolation[data]
```

```
InterpolatingFunction[{{0., 100.}}, <>]
```

El resultado se devuelve en forma de funciones de interpolación

```
Length[sqrt3]
```

```
4
```

```
sqrt3[[1]]
```

```
{{0., 100.}}
```

```
sqrt3[[2]]
```

```
{1, 0, True, Real, {3}, {0}}
```

```
sqrt3[[3]]
```

```
{{0., 1., 4., 9., 16., 25., 36., 49., 64., 81., 100.}}
```

La siguiente longitud cuenta el número de pasos

```
Length[sqrt3[[3, 1]]]
```

```
11
```

```
sqrt3[[4]]
```

```
{{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}, {0., 1., 2., 3., 4., 5., 6., 7., 8., 9., 10.}}
```

■ 5.2.3. Opciones en NDSolve

En NDSolve podemos especificar opciones que nos permiten controlar el proceso númerico utilizado para obtener la solución .

AccuracyGoal	Automatic	digitos de precisión absoluta most
DependentVariables	Automatic	la lista de todas las variables de
EvaluationMonitor	None	□
InterpolationOrder	Automatic	orden de interpolación
MaxStepFraction	1 / 10	fración máxima de rango cubierta e
MaxSteps	10 000	máximo número de pasos
MaxStepSize	Automatic	tamaño máximo de cada paso
Method	Automatic	método
NormFunction	Automatic	norma usada para estimar el error
PrecisionGoal	Automatic	digitos de precisión (relativa) mo
StartingStepSize	Automatic	tamaño del paso inicial
StepMonitor	None	expression to evaluate when a step
WorkingPrecision	MachinePrecision	precisión en cálculos internos

Debemos tener en cuenta lo siguiente respecto la construcción de NDSolve

- Si AccuracyGoal y PrecisionGoal tienen como valor Automatic, ambos son iguales al valor WorkingPrecision /2.
- $MachinePrecision es igual a 16 digitos
- Para EDO MaxSteps , vale por defecto 1000 y este valor debería ser suficiente para la mayoria de las ecuaciones con soluciones diferenciables.

• La opción Method permite especificar el método utilizado por NDSolve para calcular la solución.

"Adams"	método Adams con ordenes de 1 a 12
"BDF"	formulas de diferenciación implícita con ordenes de 1 a 5
"ExplicitRungeKutta"	métodos Runge-Kutta explícitos
"ImplicitRungeKutta"	métodos Runge-Kutta implícitos de orden arbitrario
"SymplecticPartitionedRungeKutta"	Runge-Kutta para sistemas hamiltonianos separables

■ 5.2.4. ECUACIONES DIFERENCIALES ORDINARIAS

Veamos algunos ejemplos

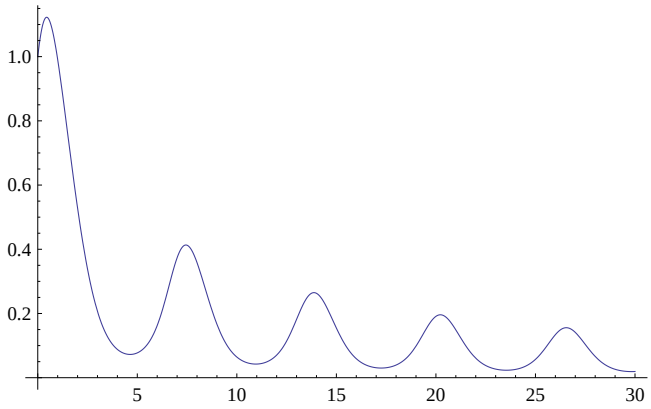
■ Una ecuación implícita de primer orden

```
s = NDSolve[{y'[x] == y[x] Cos[x + y[x]], y[0] == 1}, y, {x, 0, 30}]
```

```
{{y → InterpolatingFunction[{{0., 30.}}, <>]}}
```

Dibujamos la solución

```
Plot[Evaluate[y[x] /. s], {x, 0, 30}, PlotRange → All]
```



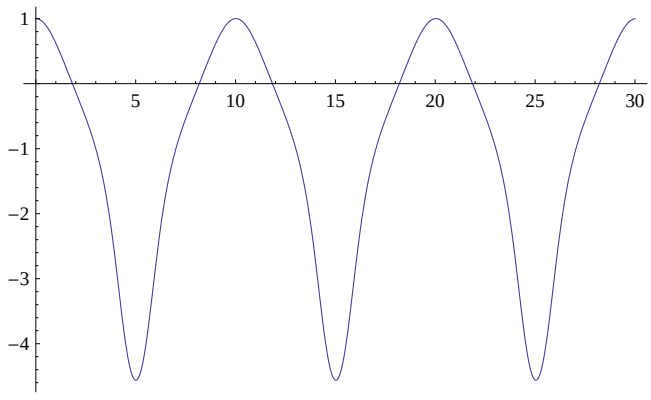
■ Una ecuación no lineal de segundo orden

```
s = NDSolve[{y''[x] + Sin[y[x]] y[x] == 0, y[0] == 1, y'[0] == 0}, y, {x, 0, 30}]
```

```
{{y → InterpolatingFunction[{{0., 30.}}, <>]}}
```

Dibujamos la solución

```
Plot[Evaluate[y[x] /. s], {x, 0, 30}, PlotRange → All]
```



NDSolve sigue el principio general de ir reduciendo el tamaño del paso hasta alcanzar una solución con la precisión deseada. Sin embargo hay un problema cuando la solución verdadera tiene una singularidad. En este caso el proceso para cuando MaxSteps alcanza el valor máximo por defecto (10000). Este valor máximo debería ser suficiente para la mayoria de las ecuaciones ordinarias, pero si las soluciones son complicadas se puede ocasionalmente poner MaxSteps->Infinity.

```
NDSolve[{y'[x] == -1 / x^2, y[-1] == -1}, y[x], {x, -1, 0}]
```

NDSolve::mxst: Maximum number of 10000 steps reached at the point x == -1.83136×10^{-172}. ≫

```
{{y[x] → InterpolatingFunction[{{-1., -1.83136 × 10^-172}}, <>][x]}}
```

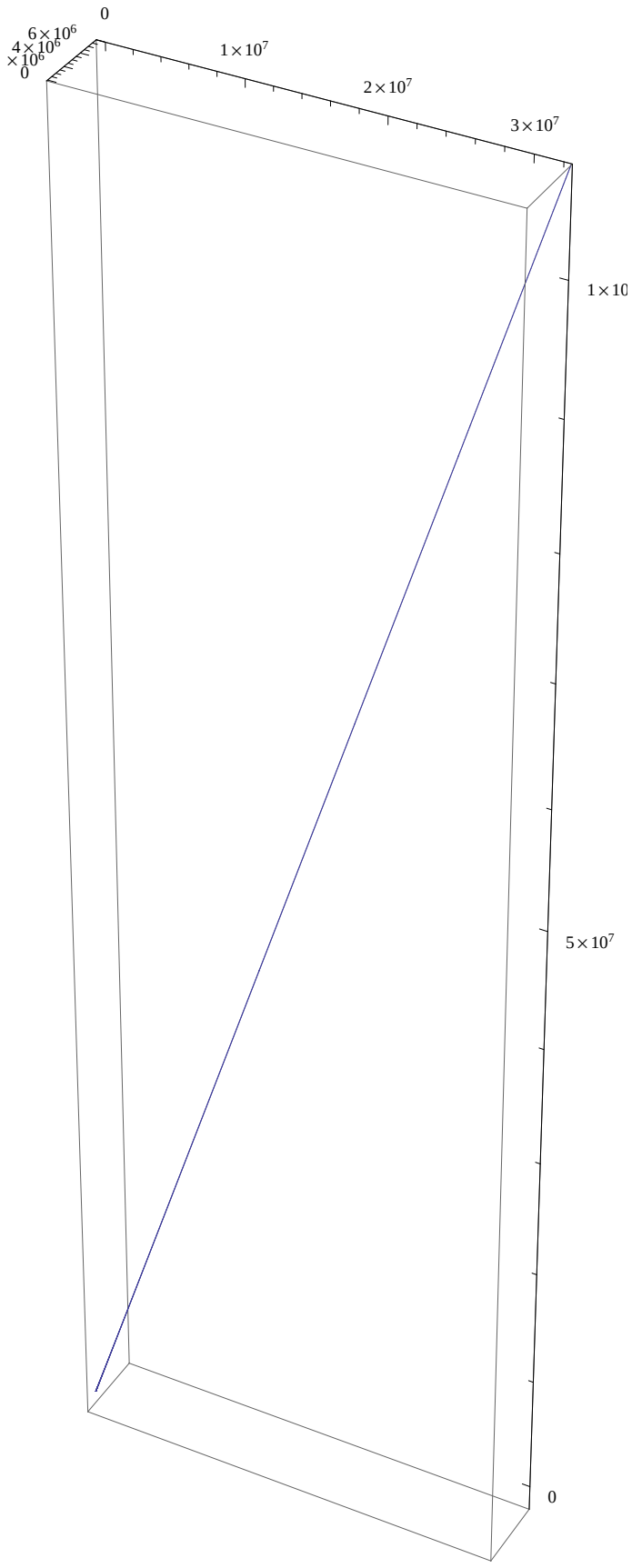
■ Resolvemos las ecuaciones de Lorenz con valores MaxSteps->10000 y MaxSteps->Infinity

```
rule1 = NDSolve[ {x'[t] == -3 (x[t] - y[t]),
    y'[t] == -x[t] z[t] + 26.5 x[t] - y[t],
    z'[t] == x[t] y[t] - z[t],
    x[0] == z[0] == 0, y[0] == 1},
   {x, y, z}, {t, 0, 200}]
```

NDSolve::mxst: Maximum number of 10000 steps reached at the point t == 125.88200678617493`. ≫

```
{{x → InterpolatingFunction[{{0., 125.882}}, <>],
  y → InterpolatingFunction[{{0., 125.882}}, <>],
  z → InterpolatingFunction[{{0., 125.882}}, <>]}}
```

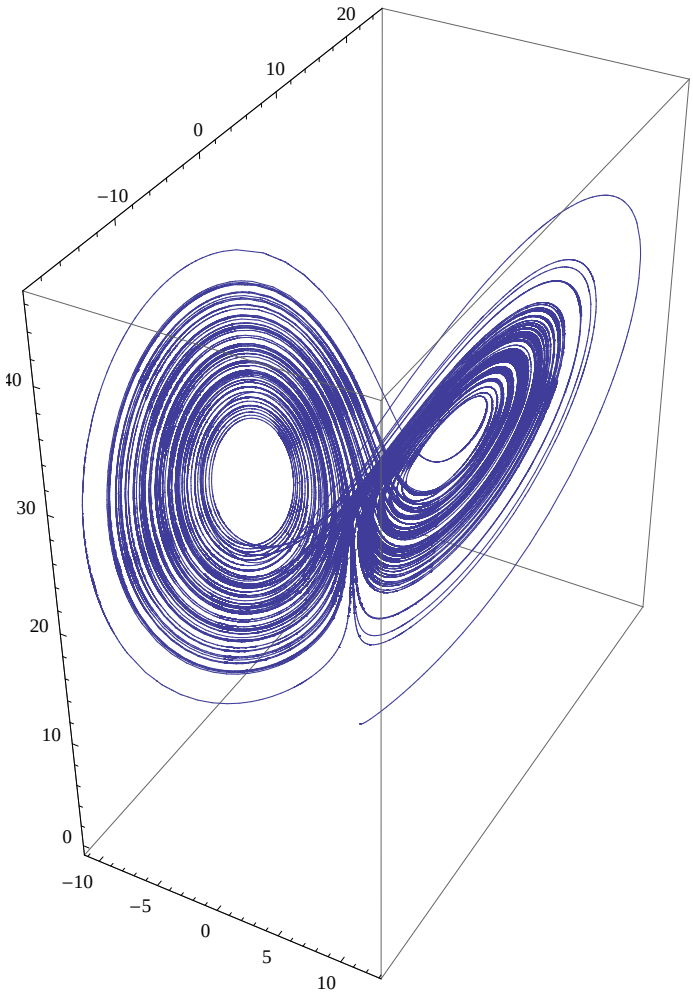
```
ParametricPlot3D[Evaluate[{x[t], y[t], z[t]} /. rule1],
 {t, 0, 200}, PlotPoints -> 10 000]
```



```
rule2 = NDSolve[{x'[t] == -3 (x[t] - y[t]),
    y'[t] == -x[t] z[t] + 26.5 x[t] - y[t],
    z'[t] == x[t] y[t] - z[t],
    x[0] == z[0] == 0, y[0] == 1},
   {x, y, z}, {t, 0, 200}, MaxSteps → Infinity]
```

```
{{x → InterpolatingFunction[{{0., 200.}}, <>],
  y → InterpolatingFunction[{{0., 200.}}, <>],
  z → InterpolatingFunction[{{0., 200.}}, <>]}}
```

```
ParametricPlot3D[Evaluate[{x[t], y[t], z[t]} /. rule2],
 {t, 0, 200}, PlotPoints -> 10 000]
```



■ Hay que ser cuidadoso al usar valores demasiado grandes para WorkingPrecision, porque cuando WorkingPrecision crece, no sólo crece el tiempo que se emplea en cada operación aritmética , sino que también el número de pasos empleados por el método crece.

A continuación comparamos un resultado exacto conocido con resultados aproximados obtenidos con distintos valores de WorkingPrecision

```
DSolve[{y''[x] + y[x] == 0, y[0] == 1, y'[0] == 0}, y, x]
```

```
{{y → Function[{x}, Cos[x]]}}
```

```
s16 = y /. First[NDSolve[{y''[x] + y[x] == 0, y[0] == 1, y'[0] == 0}, y, {x, 0, 2 π}]]
s24 = y /. First[NDSolve[{y''[x] + y[x] == 0, y[0] == 1, y'[0] == 0},
      y, {x, 0, 2 π}, WorkingPrecision → 24]];
s32 = y /. First[NDSolve[{y''[x] + y[x] == 0, y[0] == 1, y'[0] == 0},
      y, {x, 0, 2 π}, WorkingPrecision → 32]];
```

```
InterpolatingFunction[{{0., 6.28319}}, <>]
```

```
TableForm[{{16, Abs[1 - s16[2 π]], Length[s16[[3, 1]]]},
  {24, Abs[1 - s24[2 π]], Length[s24[[3, 1]]]},
  {32, Abs[1 - s32[2 π]], Length[s32[[3, 1]]]}},
 TableHeadings → {None, {"WorkingPrecision", "error", "Número de pasos"}}]
```

WorkingPrecision	error	Número de pasos
16	6.29902×10^{-10}	78
24	$9.244582180220 \times 10^{-12}$	126
32	$1.968008458816077 \times 10^{-16}$	191

■ 5.2.5. ECUACIONES EN DERIVADAS PARCIALES

Limitaciones en el uso de NDSolve

NDSolve usa un método para resolver PDE con dos variables independientes que consiste en discretizar una variable para conseguir un sistema de ODE. El sistema se resuelve utilizando las rutina programadas en NDSolve para resolver ecuaciones ordinarias

Para que el método funcione, se tienen que especificar una función inicial para una de las variables y valores frontera para las otras variables. La función inicial se usa para encontrar condiciones iniciales para el sistema de ODE.

■ El método tiene la ventaja que puede resolver una cantidad importante de clases de ecuaciones. Sin embargo hay ecuaciones que no se pueden resolver: por ejemplo ecuaciones elipticas, como la ecuación de Laplace

```
solution = u /. First[NDSolve[{∂x,x u[x, y] + ∂y,y u[x, y] == 0, u[x, 0] == Cos[2 π x],
      u^(0,1)[x, 0] == 0, u[0, y] == 1, u[1, y] == 1}, u, {x, 0, 1}, {y, 0, 1}]];
Plot3D[solution[x, y], {x, 0, 1}, {y, 0, 1}];
```

```
NDSolve::eerr:
  Warning: Scaled local spatial error estimate of 679.6743666409138` at y = 1.` in the direction of
      independent variable x is much greater than prescribed error tolerance. Grid
      spacing with 25 points may be too large to achieve the desired accuracy
      or precision.  A singularity may have formed or you may want to specify a
      smaller grid spacing using the MaxStepSize or MinPoints method options. ≫
```

Los problemas elipticos están mal condicionados a no ser que se especifiquen valores en todos los lados de la región.

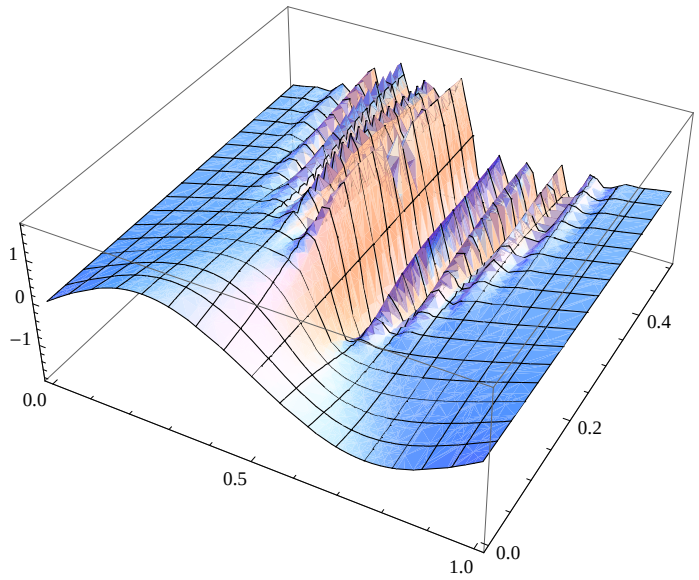
■ Otra clase de problemas en el que el método puede no funcionar correctamente son aquellos en los que la solución tiene singularidades. Por ejemplo la ecuación de Berger

```
rule =
 NDSolve[{∂t u[x, t] == -u[x, t] ∂x u[x, t], u[x, 0] == Sin[2 π x], u[0, t] == u[1, t]},
  {u}, {x, 0, 1}, {t, 0, .5}, MaxSteps → Infinity, MaxStepSize → Infinity]
```

```
NDSolve::eerr:
  Warning: Scaled local spatial error estimate of 7098.067652671562` at t = 0.5` in the direction of
        independent variable x is much greater than prescribed error tolerance. Grid
        spacing with 27 points may be too large to achieve the desired accuracy
        or precision.  A singularity may have formed or you may want to specify a
        smaller grid spacing using the MaxStepSize or MinPoints method options. ≫
```

```
{{u → InterpolatingFunction[{{0., 1.}, {0., 0.5}}, <>]}}
```

```
Plot3D[u[x, t] /. rule, {x, 0, 1}, {t, 0, .5}]
```



Opciones en EDP

Muchas de las opciones que controlan la resolución de ODE con NDSolve también se aplican a ecuaciones diferenciales en derivadas parciales. Las soluciones para EDP se calculan en dos etapas: primero se discretiza la ecuación y después se resuelve el sistema resultante de ODE
Se pueden especificar opciones diferentes para cada una de las dos etapas (el orden de las varibles independientes determina que opción se aplica a cada una de las variables)

En la etapa de discretización se usa por defecto el método de diferencias finitas de cuarto orden. En algunos casos el método de cuarto orden no es óptimo. Para especificar ordenes mayores se utiliza la opción DifferenceOrder.

Estos casos son tipicamente aquellos en que aparecen derivadas espaciales de orden superior.

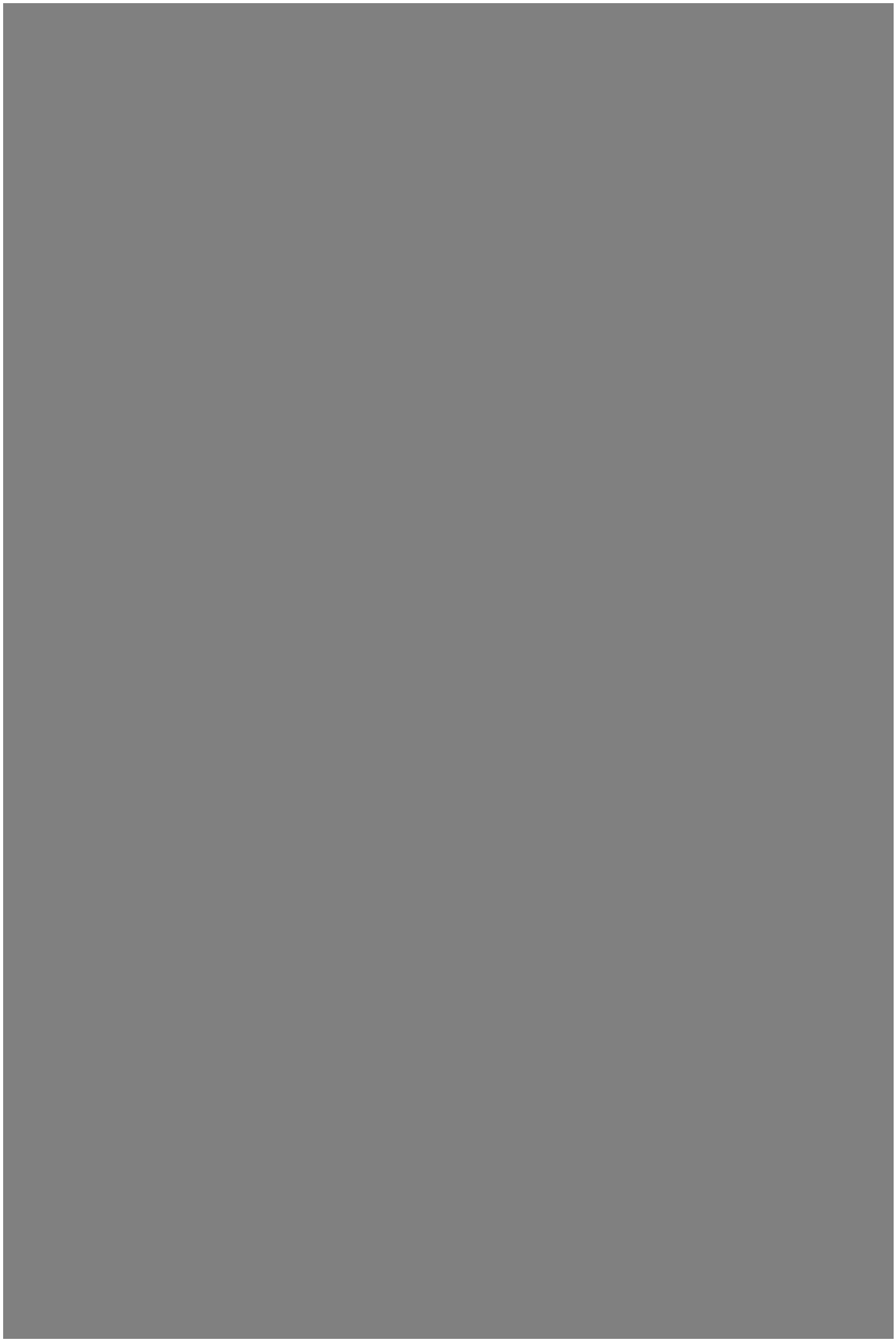
■ Por ejemplo, la ecuacion de Airy

```
rule = NDSolve[{∂t u[x, t] == -∂{x,3} u[x, t], u[x, 0] == Exp[-x^2], u[-5, t] == u[5, t]},
  {u}, {t, 0, 20}, {x, -5, 5}]
```

```
{{u → InterpolatingFunction[{{-5., 5.}, {0., 20.}}, <>]}}
```

```
solution = u /. First[rule];
```

```
Manipulate[Plot[solution[x, t], {x, -5, 5}, PlotRange → {-1, 1}], {t, 0, .4, .025}]
```



■ 5.2.6. PAQUETES DE UTILIDADES PARA ECUACIONES DIFERENCIALES

El paquete DifferentialEquations`NDSolveUtilities` permite comparar los resultados obtenidos con diferentes métodos.

CompareMethods[sys, refsol, methods, opts] devuelve estadisticas para diferentes métodos aplicados al sistema sys.
FinalSolutions[sys, sols] devuelve los valores de las soluciones al final de la integración númerica para varias soluciones sols del sistema sys.
InvariantErrorPlot[invts, dvars, ivar, sol, opts] devuelve un dibujo del error en los invariantes invts para la solución sol.
StepDataPlot[sols, opts] devuelve dibujos de los tamaños de los pasos para las soluciones sols en una escala logaritmica

```
Needs["DifferentialEquations`NDSolveUtilities` "]
```

■ Las ecuaciones de Rössler se resuelven en menos de la mitad de pasos eligiendo el método de RungeKutta en vez de el método por defecto

```
solution1 = NDSolve[{x'[t] == -y[t] - z[t], x[0] == -0.04, y'[t] == x[t] + .425 y[t],
   y[0] == -0.30, z'[t] == 2 - z[t] (4 - x[t]), z[0] == 0.52}, {x, y, z}, {t, 0, 25}]
```

```
{{x → InterpolatingFunction[{{0., 25.}}, <>],
  y → InterpolatingFunction[{{0., 25.}}, <>],
  z → InterpolatingFunction[{{0., 25.}}, <>]}}
```

```
solution2 = NDSolve[{x'[t] == -y[t] - z[t], x[0] == -0.04, y'[t] == x[t] + .425 y[t],
    y[0] == -0.30, z'[t] == 2 - z[t] (4 - x[t]), z[0] == 0.52},
   {x, y, z}, {t, 0, 25}, Method → ExplicitRungeKutta];
```

```
solution1[[1, 1, 2]]
```

```
InterpolatingFunction[{{0., 25.}}, <>]
```

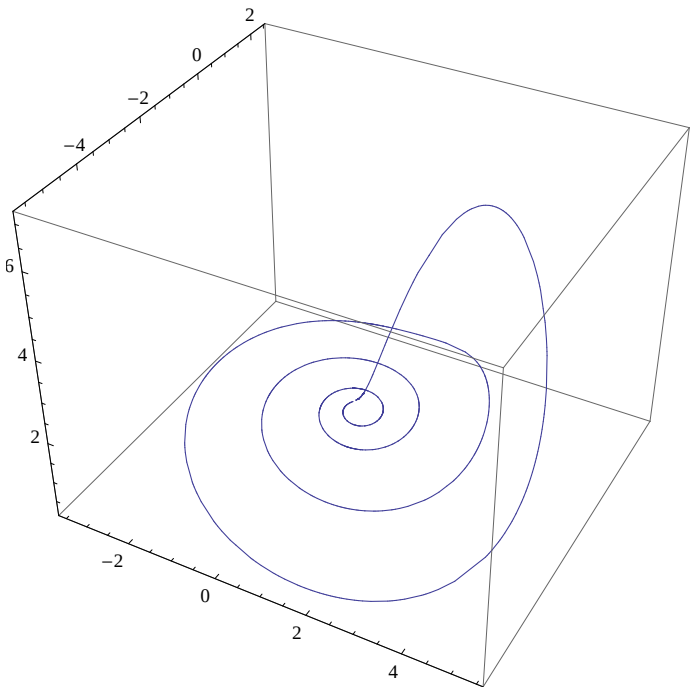
```
numeropasos1 = Length[solution1[[1, 1, 2, 3, 1]]]
```

```
410
```

```
numeropasos2 = Length[solution2[[1, 1, 2, 3, 1]]]
```

```
156
```

```
ParametricPlot3D[Evaluate[{x[t], y[t], z[t]} /. solution2],
 {t, 0, 25}, PlotRange → All]
```



5.2.7 EJERCICIOS

■ 1.-Resolver númericamente

a) x'[t]=(t^2-x[t])/(t+x[t]^2) con condición inicial x[0]=1
b) Obtener x'[0]
c) Representarla gráficamente
d)Obtener el número de pasos empleados por el método para resolver la ecuación
e)Resolver la ecuación númericamente utilizando el método de Runge-Kutta y obtener el número de pasos empleados. Dibujar la solución.
f)¿Con cuál de los dos métodos se resuelve en menos pasos la ecuación?

■ 2.-a) Resolver númericamente la siguiente ecuación de ondas
D[y[x, t], t, t] == D[y[x, t], x, x] es la ecuación de ondas con condiciones y de contorno
y[x, 0] == Exp[-x^2], Derivative[0,1][y][x, 0] == 0,y[-5, t] == y[5, t].
b) Dibujar la solución como superficie de R^3y como curva en R^2para diferentes valores de t.

■ 3.- Idem para la ecuación de Sine-Gordon
D[y[x, t], t, t] == D[y[x, t], x, x] + Sin[y[x, t]] con condiciones iniciales y de contorno
y[x, 0] == Exp[-x^2], Derivative[0,1][y][x, 0] == 0, y[-5, t] == y[5, t] .

■ 4.-a) Resolver de forma exacta el sistema
x'[t]=x-y+1
y'[t]=x+3y+e^(-t)
con condiciones iniciales x[0]=0,y[0]=1.
b) Dibujar la curva solución
c) Resolverlo de forma aproximada
d) Dibujar la solución

CAPITULO VI
APLICACIONES AL CÁLCULO: FUNCIONES ESPECIALES

Mathematica incluye definiciones de funciones especiales de física-matemática, tales como la función gamma, la función beta, funciones de Bessel.....