Tutorial super rápido do Mathematica (que também serve para o Wolfram)

MA311B - 1s2020 - Prof. Ricardo M. Martins

O comando abaixo resolve equações. Observe que é preciso usar == (dois sinais de igual) para o Mathematica entender que se trata de uma equação, se usar só um sinal de igual ele vai entender que você está atribuindo valor).

Muito importante: no Mathematica, para executar uma linha, você precisa apertar shift+enter e não só enter.

In[∘]:= `Solve[x^2 + x - 1 == 0, x]`

Out[∘]= $\left\{\left\{x \to \frac{1}{2}\left(-1-\sqrt{5}\right)\right\}, \left\{x \to \frac{1}{2}\left(-1+\sqrt{5}\right)\right\}\right\}$

Quando usamos um só sinal de igual acontece isto:

In[∘]:= `a = 3`

Out[∘]= 3

Agora a variável “a” tem o valor de 3, e você pode fazer contas com ela:

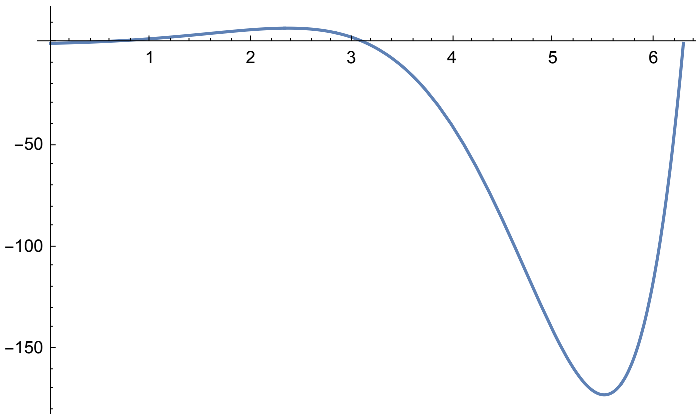
In[∘]:= `a + 2`

Out[∘]= 5

Para plotar gráficos, o Mathematica tem uma série de comandos. O mais simples deles é o Plot, ou Plot3D, conforme o gráfico seja em R2 ou R3.

In[∘]:= `Plot[Exp[x] * Sin[x], {x, 0, 2 * Pi}]`

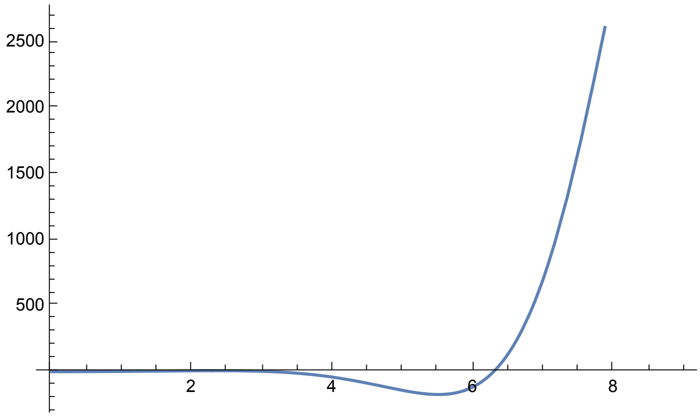
Out[∘]=



Em alguns gráficos, o eixo y fica “grande” demais, então podemos limitar os eixos. Este é um comando muito útil.

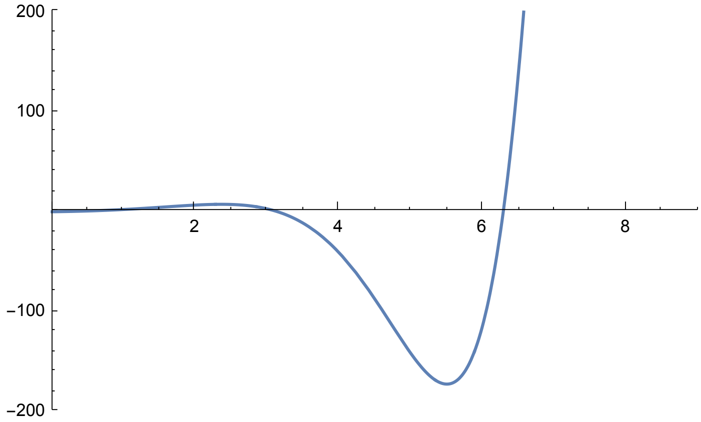
```
In[∘]:= Plot[Exp[x] * Sin[x], {x, 0, 9}]
```

Out[∘]=




```
In[∘]:= Plot[Exp[x] * Sin[x], {x, 0, 9}, PlotRange → {{0, 9}, {-200, 200}}]
```

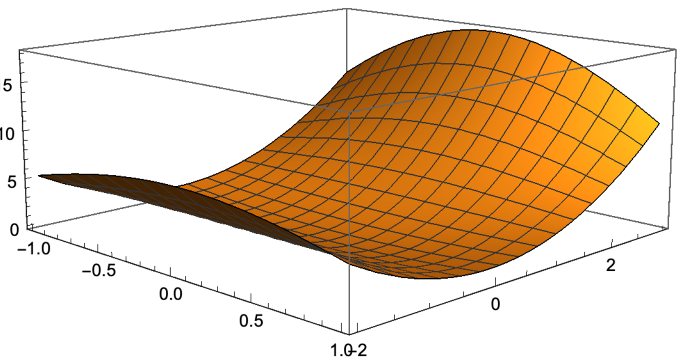
Out[∘]=



Gráficos em 3D são feitos com o comando abaixo. Note que no caso de funções z=f(x,y), você só coloca f(x,y). Precisa também indicar o intervalo de x e de y. No caso abaixo, o domínio é um retângulo.

```
In[∘]:= Plot3D[x^2 + 2 * y^2 * Cos[x], {x, -1, 1}, {y, -2, 3}]
```

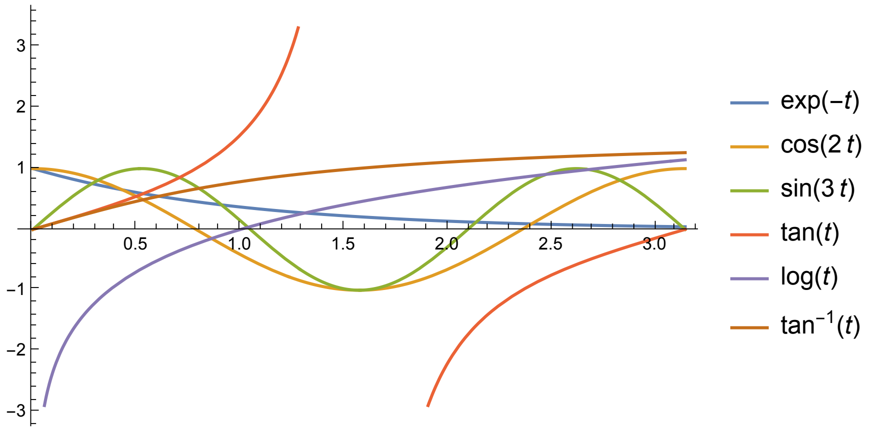
Out[∘]=



Abaixo vamos plotar várias funções em um mesmo sistema de eixos. Para isto, é só usar a notação de conjunto dentro do Plot. As funções mais “comuns” tem notação bem natural, com a ressalva de que a primeira letra é maiúscula e sempre em inglês. Quando plotamos vários gráficos, o uso de legendas é muito útil, e também está no comando abaixo.

In[∘]:=
```
Plot[{Exp[-t], Cos[2 t], Sin[3 t], Tan[t], Log[t], ArcTan[t]},
 {t, 0, Pi}, PlotLegends → LineLegend["Expressions"]]
```

Out[∘]=



Equações diferenciais!

Especificamente para equações diferenciais, o Mathematica tem duas formas de trabalhar: analiticamente e numericamente. O comando DSolve resolve (ou tenta resolver) analiticamente equações diferenciais. Já o NDSolve resolve numericamente. Para fazer gráficos, o NDSolve é muitas vezes mais adequado. Observo que o DSolve às vezes usa expressões de funções integrais no meio da resposta, então é preciso cuidado para interpretar os resultados.

Solução geral de uma EDO. Dá para usar normalmente a notação y’[x], observando o uso de [‘s ao invés de (‘s. As constantes vão aparecer com nomes c1, c2, etc.

In[∘]:=
```
DSolve[y''[x] + y'[x] == Sin[x], y, x]
```

Out[∘]= $\left\{\left\{y \to \text{Function}\left[\{x\},\, c_2 + \frac{1}{2}\left(-2\, e^{-x}\, c_1 - \text{Cos}[x] - \text{Sin}[x]\right)\right]\right\}\right\}$

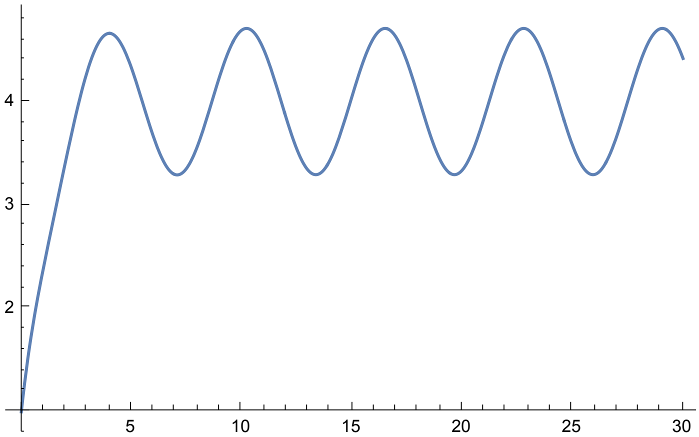
Para resolver um PVI, basta adicionar as condições inicia, lembrando de usar a notação de conjunto. O sol1 na linha de baixo é só para dar um nome a esta linha, para usarmos depois.

In[∘]:=
```
sol1 = DSolve[{y''[x] + y'[x] == Sin[x], y[0] == 1, y'[0] == 2}, y, x]
```

Out[∘]= $\left\{\left\{y \to \text{Function}\left[\{x\},\, -\frac{1}{2}\, e^{-x}\left(5 - 8\, e^{x} + e^{x}\, \text{Cos}[x] + e^{x}\, \text{Sin}[x]\right)\right]\right\}\right\}$

Abaixo fazemos o gráfico da solução obtida acima. A parte y[x]/.sol1 faz o seguinte: ela substitui y[x] pelo y[x] obtido no comando anterior, ou seja, pela solucao da equação. O intervalo em que queremos ver a solucao é o [0,30], e no caso do somando DSolve, podemos esticar este intervalo o quanto quisermos.

In[•]:=
```
Plot[y[x] /. sol1, {x, 0, 30}]
```

Out[•]=


Veja agora o caso abaixo. O Mathematica não consegue resolver esta equação analiticamente.

In[•]:=
```
DSolve[{t^2*y''[t] + t^3*y'[t] == Exp[t^2], y[1] == 1, y'[1] == 2}, y, t]
```

Solve: Inconsistent or redundant transcendental equation. After reduction, the bad equation is

$$4\, e^{\frac{1}{2}-\frac{K[1]^2}{2}} + 2\, e^{\frac{3}{2}-\frac{K[1]^2}{2}} - e^{-\frac{1}{2}K[1]^2}\sqrt{6\pi}\,\mathrm{Erfi}\left[\sqrt{\frac{3}{2}}\right] + e^{-\frac{1}{2}K[1]^2}\sqrt{6\pi}\,\mathrm{Erfi}\left[\sqrt{\frac{3}{2}}\,K[1]\right] - \frac{2\, e^{K[1]^2}}{K[1]} - 2\,\mathrm{InverseFunction}$$

$$[\mathrm{Inactive}[\mathrm{Integrate}], 1, 2]\left[\int_1^1 \left(e^{\mathrm{Times}[\ll 2 \gg]}\, c_1 + e^{\mathrm{Times}[\ll 2 \gg]}\,(\mathrm{Times}[\ll 2 \gg] + \mathrm{Times}[\ll 3 \gg])\right) dK[1], \{K[1], 1, 1\}\right] == 0.$$

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

Out[•]=
$$\left\{\left\{y \to \mathrm{Function}\left[\{t\},\, 1 + \int_1^t \left(\frac{1}{2}\, e^{-\frac{1}{2}K[1]^2}\left(4\sqrt{e} + 2\, e^{3/2} - \sqrt{6\pi}\,\mathrm{Erfi}\left[\sqrt{\frac{3}{2}}\right]\right) + e^{-\frac{1}{2}K[1]^2}\left(\sqrt{\frac{3\pi}{2}}\,\mathrm{Erfi}\left[\sqrt{\frac{3}{2}}\,K[1]\right] - \frac{e^{\frac{3K[1]^2}{2}}}{K[1]}\right)\right) dK[1]\right]\right\}\right\}$$

Teremos que partir para uma solução numérica, com o comando NDSolve. Abaixo resolvemos a equação para t no intervalo [1,3].

In[•]:=
```
sol2 =
 NDSolve[{t^2*y''[t] + t^3*y'[t] == Exp[t^2], y[1] == 1, y'[1] == 2}, y, {t, 1, 3}]
```

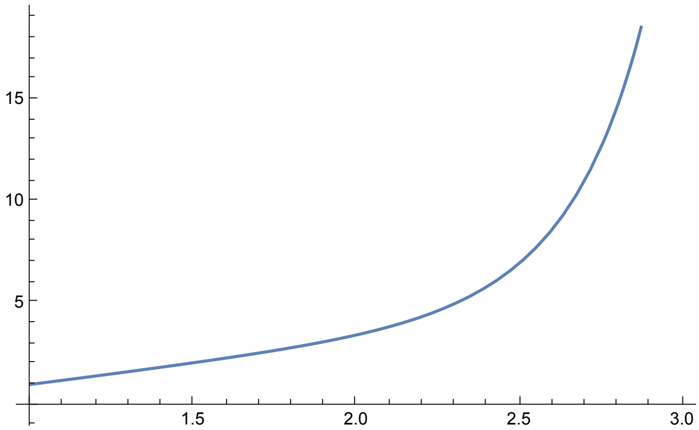
Out[•]=
```
{{y → InterpolatingFunction[ Domain: {{1., 3.}} Output: scalar ]}}
```

Com a solução numérica, temos os valores da função y(t) para todos os valores de t entre 1 e 3. Se quisermos saber y(2), por exemplo, o comando é

In[•]:=
```
y[2] /. sol2
```
Out[•]=
```
{3.43036}
```

Para plotar estes valores, usamos novamente o comando Plot:

In[•]:=
```
Plot[y[t] /. sol2, {t, 1, 3}]
```
Out[•]=

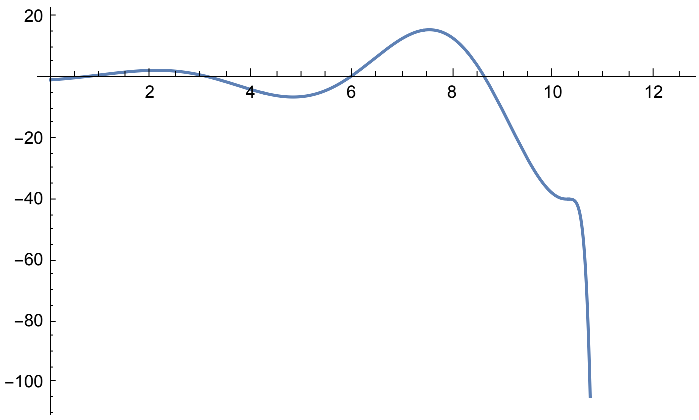


Vamos resolver mais uma EDO numericamente. O intervalo de solução é [0,10].

In[•]:=
```
sol3 = NDSolve[{y'''[t] + y'[t] + y[t] == Exp[-t] * Cos[t],
   y[1] == 1, y'[1] == 2, y''[1] == 0}, y, {t, 0, 10}]
```
Out[•]=
```
{{y → InterpolatingFunction[ Domain: {{0., 10.}} Output: scalar ]}}
```

Se formos plotar o gráfico da solução da EDO acima entre 0 e 4*Pi (note que 4*Pi é maior que 12), acontecerá o seguinte:

In[•]:=
```
Plot[y[t] /. sol3, {t, 0, 4 * Pi}]
```
Out[•]=



O gráfico acima está errado. O erro? Nossa solução do NDSolve só é válida entre 0 e 10, que foi o intervalo que pedimos para ele. Então não podemos usar um intervalo maior para ver o gráfico. Podemos corrigir isto resolvendo num intervalo maior, ou então usando o DSolve (quando possível), como fazemos abaixo:

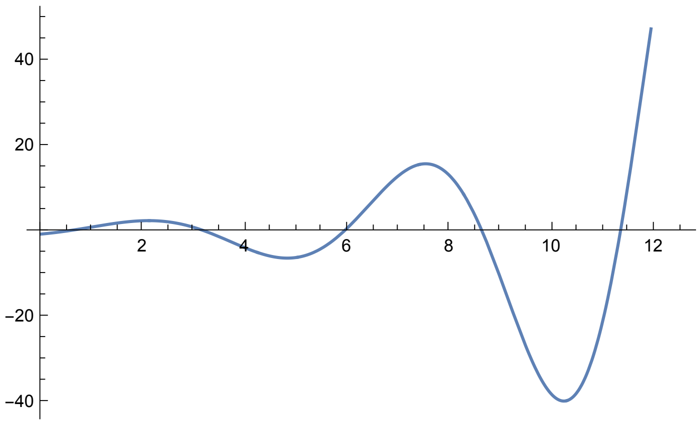
In[•]:=
```
sol4 = NDSolve[{y'''[t] + y'[t] + y[t] == Exp[-t] * Cos[t],
   y[1] == 1, y'[1] == 2, y''[1] == 0}, y, {t, 0, 4 * Pi}]
```

Out[•]=
```
{{y → InterpolatingFunction[ Domain: {{0., 12.6}}
                             Output: scalar ]}}
```

Veja como o gráfico abaixo (correto) é diferente do gráfico acima (errado). Cuidado ao usar softwares!

In[•]:=
```
Plot[y[t] /. sol4, {t, 0, 4 * Pi}]
```

Out[•]=



Vamos obter a solução analítica.

In[•]:=
```
sol5 = DSolve[
  {y'''[t] + y'[t] + y[t] == Exp[-t] * Cos[t], y[1] == 1, y'[1] == 2, y''[1] == 0}, y, t]
```

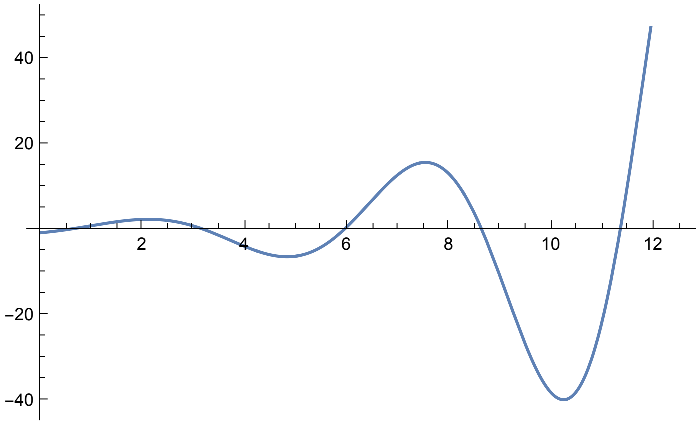
Out[•]=
```
{{y → Function[{t}, (e^(-1- 0.341...-1.16... i + t 0.341...-1.16... i) (... 1 ...)) /
    (((-0.682... - 0.341... - 1.16... i)^2 ... 3 ...
      (-1 + 0.341... - 1.16... i)^2 + 2 -0.682... 0.341... + 1.16... i)) +
    ... 1 ... + ... 1 ... + (e^(... 1 ... ... 1 ... ... 1 ...) (... 1 ...)) /
    ((- -0.682... + 0.341... - 1.16... i) ... 6 ... (... 1 ...))]}}
```

large output | show less | show more | show all | set size limit...

Plotando esta solução, vemos que bate com o resultado numérico quando fizemos da forma certa.

In[∘]:=
```
Plot[y[t] /. sol5, {t, 0, 4 * Pi}]
```

Out[∘]=



Abaixo os dois comandos fundamentais para calcular transformadas de Laplace. O t,s no final diz que a entrada é uma função de t e a saída é uma função de s (as variáveis tradicionais).

In[∘]:=
```
LaplaceTransform[Sin[t], t, s]
```

Out[∘]= $\frac{1}{1+s^2}$

In[∘]:=
```
LaplaceTransform[Sin[t] * Exp[t] * t, t, s]
```

Out[∘]= $\frac{2(-1+s)}{\left(2-2s+s^2\right)^2}$

Agora a transformada inversa:

In[∘]:=
```
InverseLaplaceTransform[1 / s, s, t]
```

Out[∘]= 1

In[∘]:=
```
InverseLaplaceTransform[1 / (s + 1) + 3 / (s - 1), s, t]
```

Out[∘]= $e^{-t} + 3\,e^{t}$

Por último, soluções de sistemas de equações diferenciais, que veremos mais para frente. O único detalhe diferente dos anteriores é que quando temos um sistema, por exemplo envolvendo x’[t] e y’[t], iremos plotar a solução também como uma curva parametrizada.

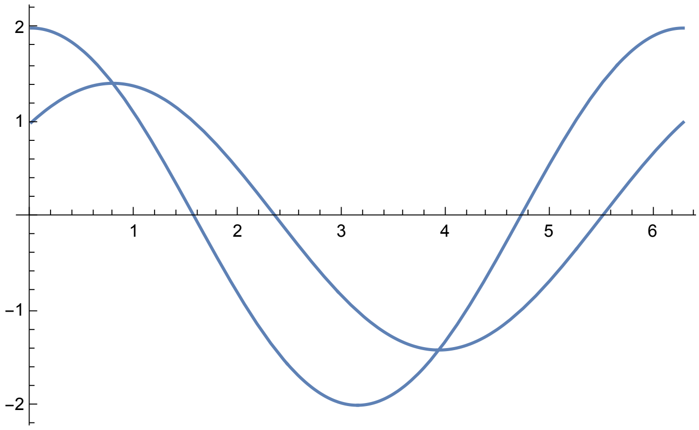
In[∘]:=
```
sol6 = DSolve[
  {x'[t] == -x[t] + y[t], y'[t] == y[t] - 2 x[t], x[0] == 1, y[0] == 2}, {x[t], y[t]}, t]
```

Out[∘]= {{x[t] → Cos[t] + Sin[t], y[t] → 2 Cos[t]}}

Encontrada a solução, podemos plotar normalmente como funções “independentes”, x[t] e y[t]:

In[∘]:=
```
Plot[{x[t], y[t]} /. sol6, {t, 0, 2 * Pi}]
```

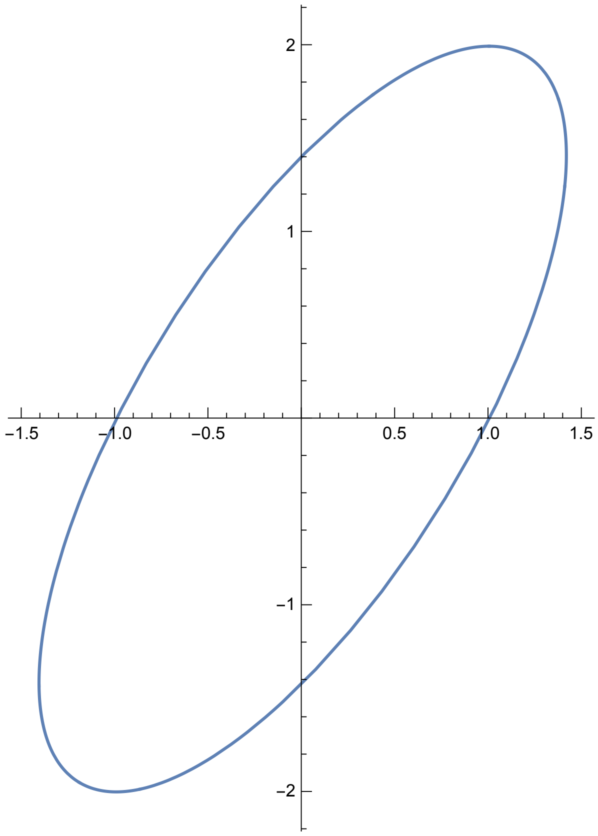
Out[∘]=



Ou então podemos plotar como uma curva parametrizada, o que faz mais sentido para sistemas autônomos.

In[∘]:=
```
ParametricPlot[{x[t], y[t]} /. sol6, {t, 0, 2 * Pi}]
```

Out[∘]=



Note que para sistemas não-lineares de equações, raramente o DSolve vai funcionar, e teremos que partir para o NDSolve se quisermos plotar as soluções.

O comando abaixo, tentativa de resolver usando o DSolve, no meu computador, não teve retorno após alguns minutos rodando..

In[∘]:=
```
sol7 = DSolve[{x'[t] == -y[t] + x[t] * y[t] ^ 2,
   y'[t] == x[t] - x[t] ^ 3, x[0] == 0, y[0] == 0.5}, {x[t], y[t]}, t]
```

Out[∘]=
```
$Aborted
```

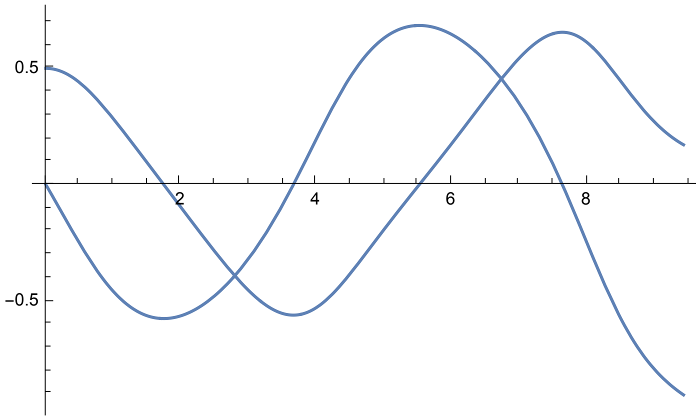
Se usarmos o NDSolve, a solução numérica é calculada.

In[∘]:=
```
sol8 = NDSolve[{x'[t] == -y[t] + x[t] * y[t] ^ 2,
   y'[t] == x[t] - x[t] ^ 3, x[0] == 0, y[0] == 0.5}, {x[t], y[t]}, {t, 0, 3 * Pi}]
```

Out[∘]=
```
{{x[t] → InterpolatingFunction[ Domain: {{0., 9.42}} Output: scalar ][t],
  y[t] → InterpolatingFunction[ Domain: {{0., 9.42}} Output: scalar ][t]}}
```

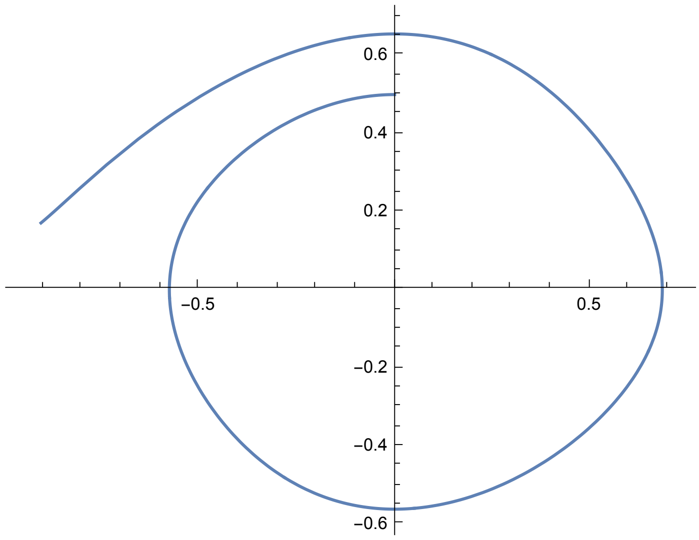
Podemos plotar as soluções individualmente..

In[∘]:=
```
Plot[{x[t], y[t]} /. sol8, {t, 0, 3 * Pi}]
```

Out[∘]=


.. ou então podemos plotar como uma curva parametrizada.

In[∘]:=
```
ParametricPlot[{x[t], y[t]} /. sol8, {t, 0, 3 * Pi}]
```

Out[∘]=


Enfim, é isto. O Help do Mathematica é muito bom, e lá vocês conseguirão muitas outras informações. Recomendo também o link abaixo, sobre o DSolve:
https://reference.wolfram.com/language/tutorial/DSolveWorkingWithDSolve.html