Równania różniczkowe

Przygotował: Maciej Maliborski
Uzupełnił: Jacek Golak

Wstęp

Bardzo często, chcąc zrozumieć zachowanie układu fizycznego, napotykamy równania, lub układy równań w których występują nie konkretne wartości liczbowe, ale FUNKCJE. Jeśli warunki na te funkcje zadane są z użyciem pochodnych, to mamy do czynienia z **równaniami różniczkowymi**. Jeśli w tych równaniach występują funkcje jednej zmiennej, to są to równania różniczkowe zwyczajne. Jeśli w równaniach występują funkcje wielu zmiennych, to są to równania różniczkowe cząstkowe.

Nie ma fizyki bez równań różniczkowych !

Zasadnicze prawo rządzące ruchem cząstki
(równanie ruchu Newtona) ma postać wektorowego
równania różniczkowego

$$m\frac{d^2\vec{r}(t)}{dt^2} = \vec{F}\left(\vec{r}, \frac{d\vec{r}}{dt}, t\right),$$

gdyż w ogólnym przypadku siła może zależeć w sposób jawny od czasu, ale także od prędkości i położenia. (Zakładamy, że nie zależy od przyspieszenia, ani wyższych pochodnych wektora położenia.)

Równania różniczkowe napotykamy w zasadzie w każdej dziedzinie fizyki: w mechanice punktów materialnych i brył sztywnych, w dynamice ośrodków ciągłych, w elektromagnetyźmie (słynne równania Maxwella), w nauce o cieple (równanie przewodnictwa cieplnego), w nauce o zjawiskach falowych, w mechanice kwantowej, w OTW, ...

Główne narzędzia: `DSolve[]`, `NDSolve[]`

W NOF_2.nb pokazaliśmy, że A*Cos[Sqrt[k/m]*t]+B*Sin[Sqrt[k/m]*t] jest rozwiązaniem równania m*x''[t]+k*x[t]=0, gdzie A i B są zupełnie dowolnymi stałymi. To wymagało jedynie podstawienia funkcji do równania z pochodnymi:

In[1]:=
```
x[t_] := A * Cos[Sqrt[k / m] * t] + B * Sin[Sqrt[k / m] * t]
```
co··· pierwiastek kwadratowy si··· pierwiastek kwadrat

In[2]:=
```
Simplify[m * x''[t] + k * x[t] == 0]
```
uprość

Out[2]=
```
True
```

In[3]:=
```
ClearAll[x, A, B, k, m];
```
wyczyść wszystko

Teraz stawiamy sobie cel bardziej ambitny, bo chcemy znaleźć rozwiązanie dla zadanego równania różniczkowego

Zaczynamy od prostego równania: szukamy funkcji, która jest równa swojej pierwszej pochodnej. Równanie różniczkowe jest pierwszego rzędu, bo mamy w nim tylko pierwszą pochodną.

In[4]:=
```
sol = DSolve[x'[t] == x[t], x[t], t]
```
rozwiązywanie równań różniczkowych

Out[4]=
$$\{\{x[t] \to e^{t} c_1\}\}$$

Rozwiązanie (tzw. całkę ogólną równania różniczkowego) dostajemy w postaci reguły, jako listę, ponieważ w ogólności może istnieć więcej niż jedno rozwiązanie. c_1 oznacza stałą całkowania. Symbol c_i posiada atrybut “Protected”.

In[5]:=
```
(* Tak ...  *)
```

In[6]:=
```
c1 = 2
```
Set: Tag C in c_1 is Protected.

Out[6]= 2

In[7]:=
```
(* ... ani tak nie można nadać stałej całkowania wartości *)
```

In[8]:=
```
C[1] = 2
```
stała

Set: Tag C in c_1 is Protected.

Out[8]= 2

In[9]:=
```
(* Można jednak za stałą coś podstawić i wtedy przypisanie wartości jest dozwolone *)
```

In[10]:=
```
x[t] /. First[sol] /. C[1] → K
```
pierwszy stała

Out[10]=
$$e^{t} K$$

In[11]:=
```
K = 2; e^t K
```

Out[11]=
$$2\, e^{t}$$

Kolejny przykład pokazuje, jak wydobyć rozwiązanie z DSolve

In[12]:=
```
(* Można w dwóch krokach, analogicznie jak to robiliśmy dla Solve ... *)
```
rozwiąż równanie

In[13]:=
```
s2 = DSolve[{x'[t] == 2 x[t] + Cos[t]}, x[t], t]
```
rozwiązywanie równań różnicz··· cosinus

Out[13]=
$$\left\{\left\{x[t] \to e^{2t} c_1 + \frac{1}{5}\left(-2\cos[t] + \sin[t]\right)\right\}\right\}$$

In[14]:=
```
x[t] /. s2[[1]]
```

Out[14]=
$$e^{2t} c_1 + \frac{1}{5}\left(-2\cos[t] + \sin[t]\right)$$

In[15]:=
```
(* ... ale można po prostu tak: *)
```

In[16]:=
```
x[t] /. First@DSolve[{x'[t] == 2 x[t] + Cos[t]}, x[t], t]
```
pierw··· rozwiązywanie równań różnicz··· cosinus

Out[16]=
$$e^{2t} C_1 + \frac{1}{5} (-2 \operatorname{Cos}[t] + \operatorname{Sin}[t])$$

Dodanie **warunków początkowych** prowadzi do jednoznacznego rozwiązania

In[17]:=
```
(* Dla pierwszego prostego równania I rzędu wystarczy wartość funkcji x[t] dla t=0 *)
```
jedność urojona

In[18]:=
```
x[t] /. First@DSolve[{x'[t] == x[t], x[0] == 2}, x[t], t]
```
pierw··· rozwiązywanie równań różniczkowych

Out[18]=
$$2 e^{t}$$

In[19]:=
```
(* Dla równania oscylatora harmonicznego, które jest równaniem II rzędu,
trzeba podać x[0] oraz x'[0], czyli wartość funkcji i pochodnej dla t=0 *)
```

In[20]:=
```
Simplify[
```
uprość
```
 x[t] /. First@DSolve[{x''[t] == -ω^2 x[t], x[0] == x0, x'[0] == v0}, x[t], t], ω > 0]
```
pierw··· rozwiązywanie równań różniczkowych

Out[20]=
$$x0 \operatorname{Cos}[t\,\omega] + \frac{v0 \operatorname{Sin}[t\,\omega]}{\omega}$$

In[21]:=
```
ClearAll[x];
```
wyczyść wszystko

W wielu przypadkach zakładamy, że równanie różniczkowe i warunki początkowe (w danym przypadku położenie początkowe i prędkość początkowa) wyznaczają jednoznaczne rozwiązanie, ale trzeba mieć świadomość, że twierdzenia matematyczne, które to zagadnienie opisują są bardzo skomplikowane.

Kolejny znany i łatwy do rozwiązania przykład: ruch punktu materialnego w jednorodnym ziemskim polu grawitacyjnym. Ruch wystarczy rozpatrywać w płaszczyźnie xy, przy czym zakładamy, że przyspieszenie ziemskie jest skierowane przeciwnie do osi y.
II zasada dynamiki daje tylko

$$m\,\vec{a} = m\,\vec{g}, \text{ czyli } \vec{a} = \vec{g}\,.$$

Dopiero **warunki początkowe** prowadzą do jednoznacznego rozwiązania (rzut pionowy w górę, rzut pionowy w dół, spadek swobodny z zadanej wysokości, rzut poziomy, rzut ukośny). Poniżej pokazuję rozwiązanie dla najbardziej ogólnego przypadku, rozważając rzut ukośny z wysokości H.

In[22]:=
```
r[t_] = {x[t], y[t]}; (* wektor położenia *)
```

In[23]:=
```
v[t_] = r'[t] ;(* wektor prędkości *)
```

In[24]:=
```
a[t_] = r''[t]; (* wektor przyspieszenia *)
```

In[25]:=
```
r0 = {0, H}; (* wektor położenia w chwili t=0 *)
```

In[26]:=
```
v0 = {v0 * Cos[alfa], v0 * Sin[alfa]}; (* prędkość w chwili t=0 *)
```
cosinus sinus

In[27]:=
```
vecg = {0, -g}; (* wektor przyspieszenia ziemskiego, gdzie g > 0 *)
```

In[28]:=
```
rzut = DSolve[{m * a[t] == m * vecg, r[0] == r0, r'[0] == v0}, {x[t], y[t]}, t]
```
rozwiązywanie równań różniczkowych

Out[28]=
$$\left\{\left\{x[t] \to t\,\text{Cos}[\text{alfa}]\,v_0,\ y[t] \to \frac{1}{2}\left(2H - g\,t^2 + 2\,t\,\text{Sin}[\text{alfa}]\,v_0\right)\right\}\right\}$$

In[ ]:=

Zamiast warunków początkowych do określenia stałych całkowania można (próbować) użyć warunków **brzegowych**, to znaczy zadać wartości funkcji dla dwóch różnych argumentów.
Uwaga: Nie zawsze rozwiązanie istnieje !
Przykład

In[29]:=
```
DSolve[{y''[x] == x, y[0] == 1, y[1] == -1}, y[x], x]
```
rozwiązywanie równań różniczkowych

Out[29]=
$$\left\{\left\{y[x] \to \frac{1}{6}\left(6 - 13\,x + x^3\right)\right\}\right\}$$

In[30]:=
```
(* Tu się udało *)
```

In[31]:=
```
DSolve[{y''[x] == -y[x], y[0] == 0, y[2 * Pi] == 1}, y[x], x]
```
rozwiązywanie równań różniczkowych pi

DSolve: For some branches of the general solution, the given boundary conditions lead to an empty solution.

Out[31]=
$$\{\}$$

In[32]:=
```
(* W tym przypadku się nie udało. Dlaczego ? *)
```

In[33]:=

Równania różniczkowe często mają rozwiązania w postaci funkcji specjalnych. W wielu przypadkach ich rozwiązania wręcz definiują funkcje specjalne.

Przykład

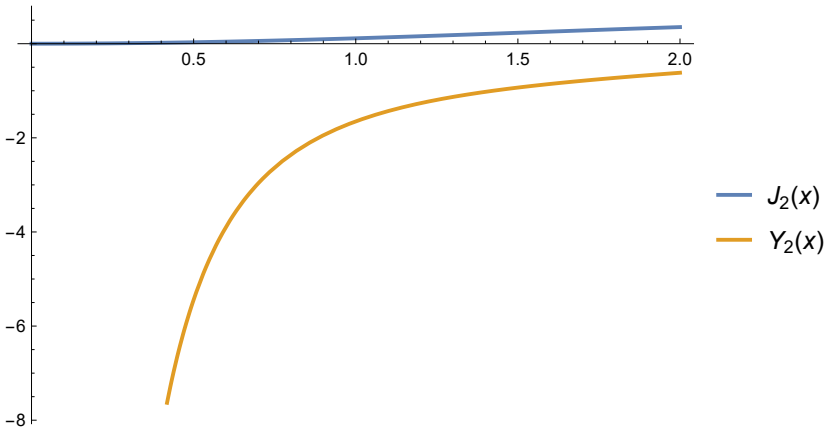
In[34]:=
```
DSolve[x^2 y''[x] + x y'[x] + (x - n) (x + n) y[x] == 0, y[x], x]
```
rozwiązywanie równań różniczkowych

Out[34]=
$$\{\{y[x] \to \text{BesselJ}[n, x]\,C_1 + \text{BesselY}[n, x]\,C_2\}\}$$

In[35]:=
```
(* W tym przypadku ogólne rozwiązanie jest
 dowolną kombinacją liniową funkcji Bessela pierwszego
 (BesselJ[n,x]↔ J_n(z)) i drugiego rodzaju (BesselY[n,x]↔ Y_n(z)). *)
```
funkcja J Bessela funkcja Y Bessela

In[36]:=
```
Plot[{BesselJ[2, x], BesselY[2, x]}, {x, 0, 2}, PlotLegends → "Expressions"]
```
wykres | funkcja J Bessela | funkcja Y Bessela | legenda dla grafik

Out[36]=



Inny przykład

In[37]:=
```
DSolve[y''[x] + Cos[x] y[x] == 0, y, x]
```
rozwiązywanie rów··· | cosinus

Out[37]=

$$\left\{\left\{y \to \text{Function}\left[\{x\}, c_1\, \text{MathieuC}\left[0, -2, \frac{x}{2}\right] + c_2\, \text{MathieuS}\left[0, -2, \frac{x}{2}\right]\right]\right\}\right\}$$

Jeszcze inny przykład

In[38]:=
```
DSolve[y''[x] - x y[x] == 0, y[x], x]
```
rozwiązywanie równań różniczkowych

Out[38]=

$$\{\{y[x] \to \text{AiryAi}[x]\, c_1 + \text{AiryBi}[x]\, c_2\}\}$$

In[39]:=

Jak pokazano dla rzutu ukośnego, przy pomocy funkcji DSolve możemy rozwiązywać **układy równań** różniczkowych zwyczajnych. Proszę przyjrzeć się składni tej komendy używającej “@” !

In[40]:=
```
{x[t], y[t]} /. First@
                  pierwszy
  DSolve[{x'[t] == 2 y[t] - x[t], y'[t] == -x[t] + y[t]}, {x[t], y[t]}, t]
  rozwiązywanie równań różniczkowych
```

Out[40]=

$$\{c_1\,(\text{Cos}[t] - \text{Sin}[t]) + 2\,c_2\,\text{Sin}[t],\ -c_1\,\text{Sin}[t] + c_2\,(\text{Cos}[t] + \text{Sin}[t])\}$$

In[41]:=
```
(* Dla mnie prościej wygląda taki zapis (polecam): *)
```

In[42]:=
```
s13 = DSolve[{x'[t] == 2 y[t] - x[t], y'[t] == -x[t] + y[t]}, {x[t], y[t]}, t]
      rozwiązywanie równań różniczkowych
```

Out[42]=

$$\{\{x[t] \to c_1\,(\text{Cos}[t] - \text{Sin}[t]) + 2\,c_2\,\text{Sin}[t],\ y[t] \to -c_1\,\text{Sin}[t] + c_2\,(\text{Cos}[t] + \text{Sin}[t])\}\}$$

In[43]:=
```
{x[t], y[t]} /. s13[[1]]
```

Out[43]=
$$\{\mathbb{C}_1 (\text{Cos}[t] - \text{Sin}[t]) + 2\,\mathbb{C}_2 \text{Sin}[t],\ -\mathbb{C}_1 \text{Sin}[t] + \mathbb{C}_2 (\text{Cos}[t] + \text{Sin}[t])\}$$

In[44]:=
```
(* Gdybym chciał mieć do dyspozycji te rozwiązania w postaci funkcji, wtedy: *)
```

In[45]:=
```
x[t_] = x[t] /. s13[[1]]
```

Out[45]=
$$\mathbb{C}_1 (\text{Cos}[t] - \text{Sin}[t]) + 2\,\mathbb{C}_2 \text{Sin}[t]$$

In[46]:=
```
y[t_] = y[t] /. s13[[1]]
```

Out[46]=
$$-\mathbb{C}_1 \text{Sin}[t] + \mathbb{C}_2 (\text{Cos}[t] + \text{Sin}[t])$$

In[47]:=
```
ClearAll["Global`*"];
```
wyczyść wszystko

In[ ]:=

Równania w układzie równań mogą mieć charakter różniczkowo-algebraiczny, co oznacza, że nie wszystkie równanie muszą być różniczkowe. (W drugim równaniu nie ma pochodnej !)

In[48]:=
```
DSolve[{y'[x] + 3 z'[x] == 4 y[x] + 1 / x, y[x] + z[x] == 1}, {y[x], z[x]}, x]
```
rozwiązywanie równań różniczkowych

Out[48]=
$$\left\{\left\{y[x] \to \frac{3}{2} + \frac{1}{18}\left(-e^{-2x}\mathbb{C}_1 - 9\,e^{-2x}\left(3\,e^{2x} + \text{ExpIntegralEi}[2x]\right)\right),\right.\right.$$
$$\left.\left.z[x] \to -\frac{1}{2} + \frac{1}{18}\left(e^{-2x}\mathbb{C}_1 + 9\,e^{-2x}\left(3\,e^{2x} + \text{ExpIntegralEi}[2x]\right)\right)\right\}\right\}$$

In[49]:=
```
(* Tu także pojawiają się funkcje specjalne *)
```

In[50]:=

Poprawność rozwiązania analitycznego możemy sprawdzić poprzez różniczkowanie wyniku. Jednak nie zawsze równanie daje się scałkować. Rozwiązania w postaci algebraicznej (inaczej analityczne) istnieją tylko dla wybranej klasy równań. Co więcej, rozwiązanie nietrywialnego równania może być bardzo skomplikowane, a jego analiza może przyprawiać o zawrót głowy. Dlatego czasem trzeba się zadowolić rozwiązaniem w postaci numerycznej.

Wynik procedur numerycznych podawany jest jako **funkcja interpolacyjna** (interpolująca ?). Zobaczmy najpierw, co to oznacza dla znanego przypadku. Funkcja interpFnc ma imitować sinus i jest zbudowana z wartości funkcji sinus w punktach x= 0, Pi/2, Pi, 3*Pi/2, 2*Pi, 5*Pi/2, 3*Pi, 7*Pi/2,4*Pi.

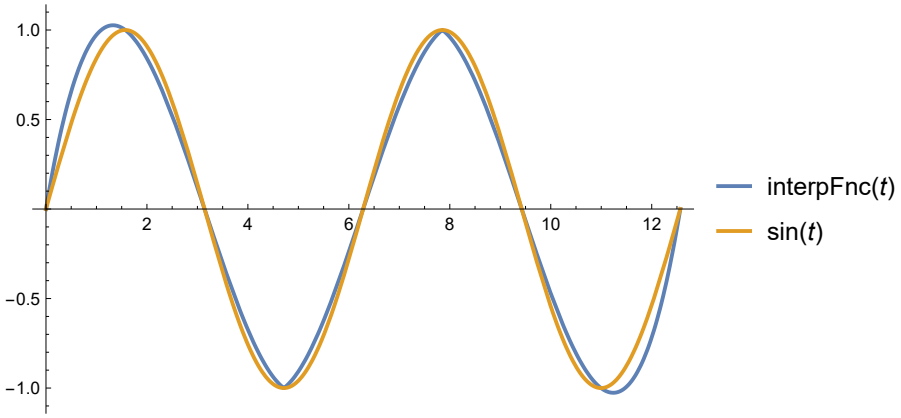
In[51]:=
```
Table[{x, Sin[x]}, {x, 0, 4 Pi, Pi / 2}]
```
tabela sinus pi pi

Out[51]=
$$\left\{\{0, 0\}, \left\{\frac{\pi}{2}, 1\right\}, \{\pi, 0\}, \left\{\frac{3\pi}{2}, -1\right\}, \{2\pi, 0\}, \left\{\frac{5\pi}{2}, 1\right\}, \{3\pi, 0\}, \left\{\frac{7\pi}{2}, -1\right\}, \{4\pi, 0\}\right\}$$

In[52]:=
```
interpFnc = Interpolation[ Table[{x, Sin[x]}, {x, 0, 4 Pi, Pi / 2}]];
```
interpolacja tabela sinus pi pi

In[53]:=
```
Plot[ {interpFnc[t], Sin[t]}, {t, 0, 4 Pi} , PlotLegends → "Expressions"]
```
wykres sinus pi legenda dla grafik

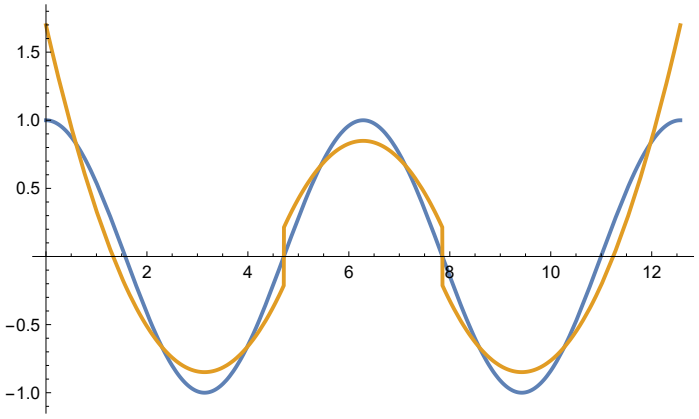
Out[53]=



Interpolującą funkcją można (i należy) posługiwać się w ten sam sposób jak zwykłymi funkcjami.

In[54]:=
```
Plot[ {Cos[t], interpFnc'[t]}, {t, 0, 4 Pi}]
```
wykres cosinus pi

Out[54]=



In[55]:=
```
dwiecałki = Integrate[{Sin[t], interpFnc[t]}, {t, 0, Pi}]
```
całka sinus pi

Out[55]=

$$\left\{2, \frac{2\,\pi}{3}\right\}$$

In[56]:=
```
N[dwiecałki]
```
przybliżenie numeryczne

Out[56]=
```
{2., 2.0944}
```

In[57]:=
```
N[dwiecałki[[2]] / dwiecałki[[1]]]
```
przybliżenie numeryczne

Out[57]=
```
1.0472
```

In[58]:=
```
(* Przy tej całce,
która jest polem powierzchni pod wykresem Sin[t] i interpFnc[t] dla 0≤ x ≤ Pi,
                                          sinus                    pi
mamy około 5 % różnicy  *)
```

Funkcja interpolująca nie daje absolutnie dokładnych wyników. Aby rachunki z taką funkcją były precyzyjne, trzeba znać interpolowaną funkcję w odpowiednio wielu "punktach podparcia".

W jaki sposób uzyskać dostęp do punktów, w których została spróbkowana funkcja podczas interpolacji ?

In[59]:=
```
interpFnc = Interpolation[ Table[{x, Sin[x]}, {x, 0, 4 Pi, Pi / 2}] ]
            interpolacja     tabela     sinus          pi  pi
```

Out[59]=

InterpolatingFunction[Domain: {{0, 4 π}} Output: scalar]

In[60]:=
```
(* Tak dostaniemy pełną informację o interpFnc *)
```

In[61]:=
```
interpFnc // N // FullForm
             pr··· pełna forma
```

Out[61]//FullForm=
```
InterpolatingFunction[List[List[0.`, 12.566370614359172`]],
 List[5, 7, 0, List[9], List[4], 0, 0, 0, 0, Automatic, List[], List[], False],
 List[List[0.`, 1.5707963267948966`, 3.141592653589793`,
   4.71238898038469`, 6.283185307179586`, 7.853981633974483`,
   9.42477796076938`, 10.995574287564276`, 12.566370614359172`]],
 List[Developer`PackedArrayForm, List[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
  List[0.`, 1.`, 0.`, -1.`, 0.`, 1.`, 0.`, -1.`, 0.`]], List[Automatic]]
```

In[62]:=
```
(* Tak wydobywamy punkty x użyte do zbudowania funkcji, ... *)
```

In[63]:=
```
xi = interpFnc[[3, 1]]
```

Out[63]=

$$\left\{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi, \frac{5\pi}{2}, 3\pi, \frac{7\pi}{2}, 4\pi\right\}$$

In[64]:=
```
(* ... a tak wartości funkcji w tych punktach *)
```

In[65]:=
```
fi = N[interpFnc][[4, 3]]
     przybliżenie numeryczne
```

Out[65]=
```
{0., 1., 0., -1., 0., 1., 0., -1., 0.}
```

In[66]:=
```
Sin[xi]
sinus
```

Out[66]=
```
{0, 1, 0, -1, 0, 1, 0, -1, 0}
```

In[67]:=
```
Sin[xi] == fi
sinus
```

Out[67]=
```
True
```

Opcje funkcji **Interpolation[].**

Metoda “Spline” generuje funkcję o ciągłej pochodnej, w przeciwieństwie do metody “Hermite”

In[68]:=
```
Options[Interpolation]
```
opcje interpolacja

Out[68]=
```
{InterpolationOrder → 3, Method → Automatic, PeriodicInterpolation → False}
```

In[69]:=
```
ClearAll["Global`*"];
```
wyczyść wszystko

In[70]:=

Do rozwiązywania numerycznego równań różniczkowych służy NDSolve. Składnia funkcji NDSolve[] różni się od składni DSolve tym, że wymaga podania warunków początkowych (lub brzegowych) oraz zakresu zmiennej niezależnej.

Wszystkie wartości numeryczne parametrów równania muszą być określone.

Przykład 1

In[71]:=
```
(* Tu mamy proste równanie I rzędu z jednym warunkiem początkowym. Dla tego
```
jedność urojona
```
 równania mamy rozwiązanie dokładne i możemy sprawdzić rozwiązanie numeryczne *)
```

In[72]:=
```
ana1 = DSolve[{y'[x] == 3 * y[x], y[0] == 1}, y[x], x]
```
rozwiązywanie równań różniczkowych

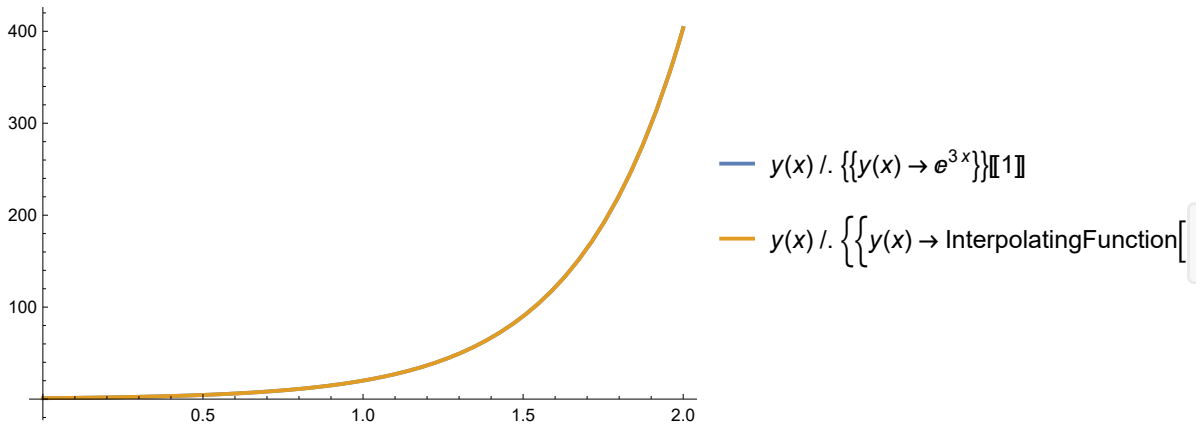
Out[72]=
$$\{\{y[x] \to e^{3x}\}\}$$

In[73]:=
```
num1 = NDSolve[{y'[x] == 3 * y[x], y[0] == 1}, y[x], {x, 0, 2}]
```
rozwiąż numerycznie równanie różniczkowe

Out[73]=
$$\{\{y[x] \to \text{InterpolatingFunction}[\langle\text{Domain: } \{\{0., 2.\}\} \text{ Output: scalar}\rangle][x]\}\}$$

In[74]:=
```
Plot[{(y[x] /. ana1[[1]]), (y[x] /. num1[[1]])}, {x, 0, 2}, PlotLegends → "Expressions"]
```
wykres legenda dla grafik

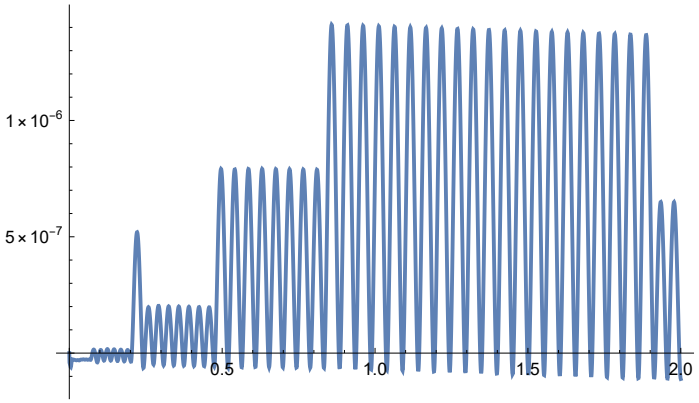
Out[74]=



In[75]:=
```
(* Poniżej rysunek względnej różnicy między rozwiązaniem
 analitycznym (dokładnym) i numerycznym (przybliżonym)*)
```

In[76]:=
```
Plot[{((y[x] /. ana1[[1]]) - (y[x] /. num1[[1]])) / (y[x] /. ana1[[1]])},
 {x, 0, 2}, PlotLegends → "Expressions"]
```
wykres
legenda dla grafik

Out[76]=



Przykład 2 (S. Wolfram)

In[77]:=
```
(* Tu mamy równanie II rzędu z dwoma warunkami brzegowymi *)
```

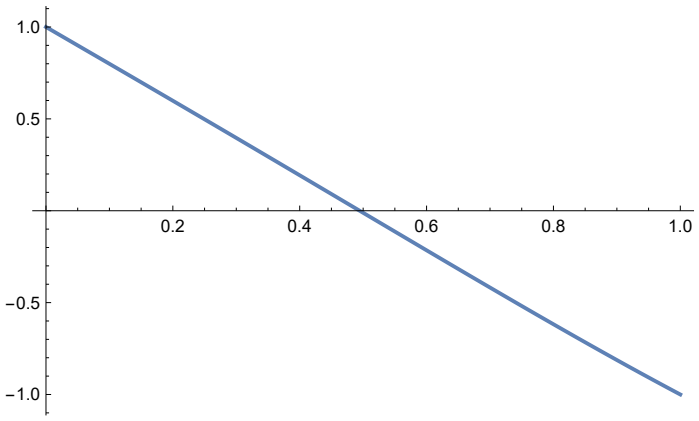
In[78]:=
```
num2 = NDSolve[{y''[x] + x * y[x] == 0, y[0] == 1, y[1] == -1}, y[x], {x, 0, 1}]
```
rozwiąż numerycznie równanie różniczkowe

Out[78]=

$$\left\{\left\{y[x] \to \text{InterpolatingFunction}\left[\begin{array}{l}\text{Domain: } \{\{0.,\ 1.\}\} \\ \text{Output: scalar}\end{array}\right][x]\right\}\right\}$$

In[79]:=
```
Plot[y[x] /. num2[[1]], {x, 0, 1}]
```
wykres

Out[79]=



Dalsze przykłady

In[80]:=
```
(* Teraz pokazuję po kolei składniki problemu *)
```

In[81]:=
```
(* Samo równanie: *)
```

In[82]:=
```
eq = x'[t] + t Cos[Pi t]^2 x[t] == t Cos[t];
```
cosinus

In[83]:=
```
(* Warunki początkowe *)
```

In[84]:=
```
wp = x[0] == 0;
```

In[85]:=
```
(* Przedział, w którym szukamy rozwiązania *)
```

In[86]:=
```
{tmin, tmax} = {0, 10};
```

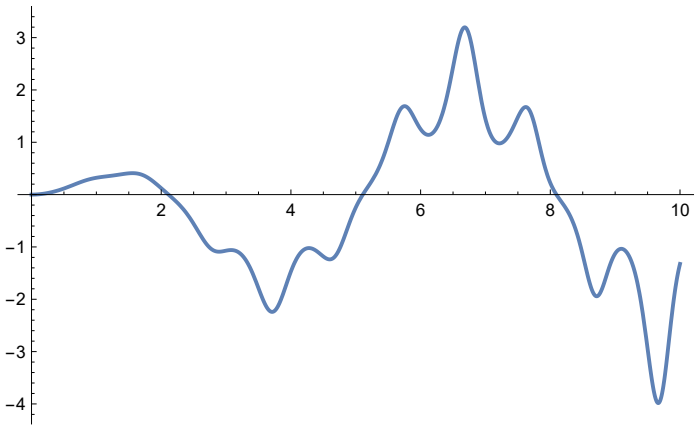
In[87]:=
```
sol = x /. First@NDSolve[{eq, wp}, x, {t, tmin, tmax}]
```
pierw··· rozwiąż numerycznie równanie różniczkowe

Out[87]=

InterpolatingFunction [Domain: {{0., 10.}} Output: scalar]

In[88]:=
```
Plot[Evaluate[sol[t]], {t, 0, 10}]
```
wyk··· oblicz

Out[88]=



Używamy **InterpolatingFunction[]** jako zwyczajnej funkcji. Tym razem użyjemy jej jako argumentu funkcji **FindRoot[]**.

In[89]:=
```
FindRoot[sol[t] == -3, {t, 10}]
```
znajdź pierwiastek

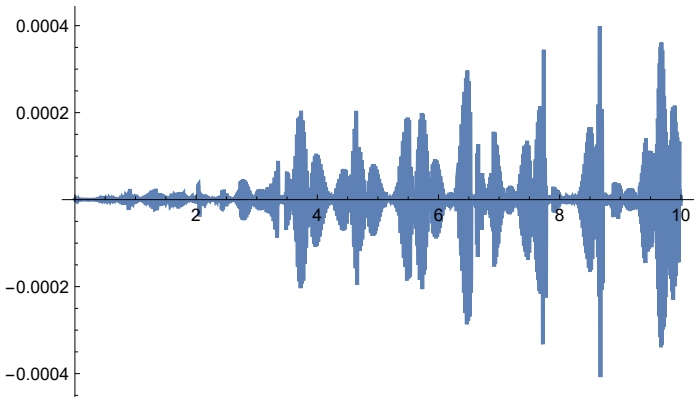
Out[89]=

{t → 9.50599}

Możemy sprawdzić poprawność rozwiązania poprzez wstawienie wyniku funkcji **NDSolve[]** do równania

In[90]:=
```
Plot[eq /. x → sol, {t, tmin, tmax}, Evaluated → True, PlotRange → All]
```
wykres pra··· zakres wykresu wszys

Out[90]=



Domyślną wartością opcji AccuracyGoal (PrecisionGoal) funkcji NDSolve jest (dla precyzji maszynowej) 7 lub 8. Dlatego można się było spodziewać, że uzyskany wynik będzie rozwiązaniem

z dokładnością 10^{-7} czy 10^{-8}. Powyższy wykres wskazuje jednak coś zupełnie innego, bo widzimy znacznie większe wartości.

Okazuje się, że jeśli sprawdzić rozwiązanie w punktach, dla których algorytmy zaimplementowane w NDSolve wyliczyły rozwiązanie, wynik będzie zgodny z oczekiwaniem.

In[91]:=
```
First@sol["Coordinates"] // Short
```
pierwszy krótki zapis

```
 (* punkty siatki w których jest znane rozwiązanie oraz pochodne *)
```

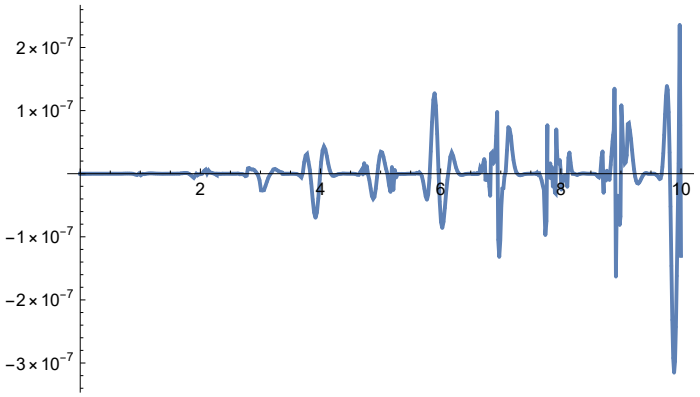
Out[91]//Short=
```
{0., 0.000120972, <<584>>, 9.98921, 10.}
```

In[92]:=
```
Table[{t, x'[t] + t Cos[Pi t]^2 x[t] - t Cos[t] /. x → sol},
```
tabela cosinus

```
  {t, First[sol["Coordinates"]]}];
```
pierwszy

```
ListPlot[%, PlotRange → All, Joined → True, Mesh → All]
```
wykres danyc… zakres wykresu ws… połączone pra… siatka wszystk

Out[93]=



In[94]:=
```
(* Teraz widzimy znacznie mniejsze wartości, tak jak tego oczekiwaliśmy *)
```

In[95]:=
```
ClearAll["Global`*"];
```
wyczyść wszystko

Pułapki obliczeń numerycznych

Typowe błędy w rachunkach numerycznych spowodowane są przez zaokrąglenia, które są efektem ubocznym skończonej precyzji obliczeń. Co więcej, rachunki związane z równaniami różniczkowymi posiadają dodatkową pułapkę, mianowicie małe błędy mogą propagować się w rozwiązaniu. Jeśli problem jest bardzo wrażliwy na tego typu niewielkie zaburzenia, otrzymane wyniki mogą być dalekie od prawdziwych.
Obrazują to poniższe przykłady.

Przykład 1

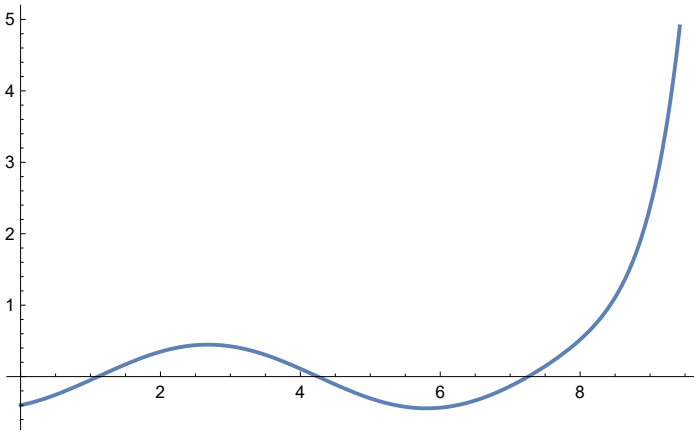
In[96]:=
```
ode = {x'[t] == 2 x[t] + Cos[t], x[0] == -2 / 5};
```
cosinus

In[97]:=
```
xs = x /. First@NDSolve[ode, x, {t, 0, 3 π}];
```
pierw… rozwiąż numerycznie równanie różniczk

In[98]:=
```
(* Wykres dla rozwiązania numerycznego *)
```

In[99]:=
```
Plot[xs[t], {t, 0, 3 π}, PlotRange → All]
```
wykres zakres wykresu wszys

Out[99]=



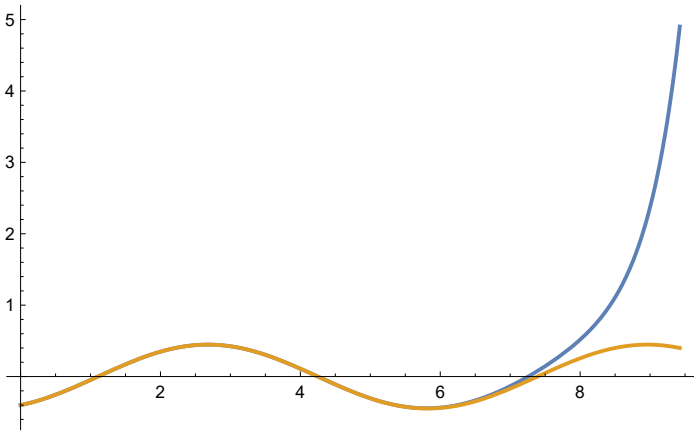
Dla tego problemu, możemy znaleźć rozwiązanie analityczne

In[100]:=
```
X = x /. First @@ DSolve[ode, x, t];
```
pierwszy rozwiązywanie równań róż

In[101]:=
```
(* Możemy więc porównać wynik numeryczny
 (niebieski) z analitycznym (pomarańczowym) *)
```

In[102]:=
```
Plot[{xs[t], X[t]}, {t, 0, 3 π}, PlotRange → All]
```
wykres zakres wykresu wszys

Out[102]=



Olbrzymia rozbieżność wyników jest związana z wartością początkową `-(2/5)` (do przeanalizowania).

To niezwykle szczęśliwy przypadek dla którego znamy rozwiązanie analityczne. W ogólnym przypadku, kiedy takiego rozwiązania nie mamy, należy stosować i polegać na innych metodach, aby zdobyć pewność poprawności przybliżonego numerycznego rozwiązania.

Przykład 2: równanie Duffinga

Czteroparametrowe równanie pojawiające się przy analizie ruchu oscylatora wymuszonego

$$x''(t) - a\,x(t) + b\,x(t)^3 + c\,x'(t) = d\cos(t)$$

Dla pewnych wartości początkowych zachowanie rozwiązania jest bardzo skomplikowane.

In[103]:=
```
Clear[ode, x, xs]
```
wyczyść

In[104]:=
```
odeDuff[a_, b_, c_, d_] := x''[t] - a x[t] + b x[t]^3 + c x'[t] == d Cos[t]
```
cosinus

In[105]:=
```
ode = {odeDuff[1, 1, 15 / 100, 3 / 10], x[0] == 1, x'[0] == -1};
xs = x /. First@NDSolve[ode, x, {t, 0, 100}];
```
pierw··· rozwiąż numerycznie równanie różniczkowe

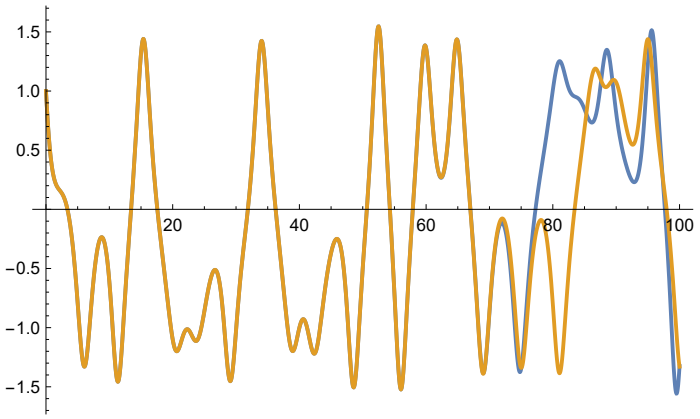
In[107]:=
```
(* Teraz wprowadzamy bardzo niewielką zmianę (eps) w
 warunkach początkowych i sprawdzamy, jak zmieni się rozwiązanie *)
```

In[108]:=
```
eps = 10^-8;
ode = {odeDuff[1, 1, 15 / 100, 3 / 10], x[0] == 1 + eps, x'[0] == -1 + eps};
xseps = x /. First@NDSolve[ode, x, {t, 0, 100}];
```
pierw··· rozwiąż numerycznie równanie różniczkowe

In[111]:=
```
(* Porównujemy wyniki dla przypadków,
które bardzo nieznacznie różnią się warunkami początkowymi *)
```

In[112]:=
```
Plot[ {xs[t], xseps[t]}, {t, 0, 100} ]
```
wykres

Out[112]=



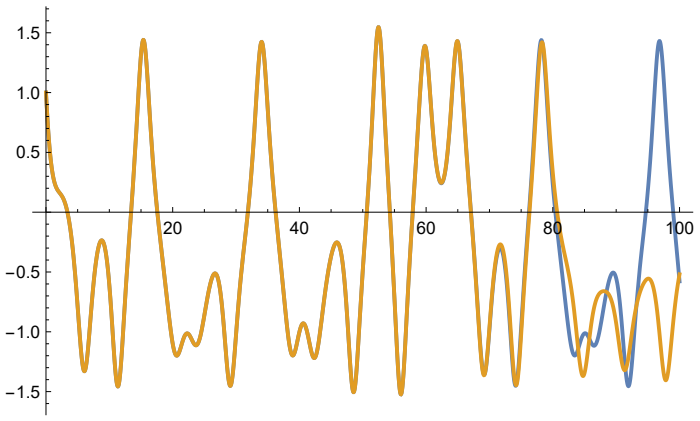
Począwszy od około t=75 zaburzone rozwiązanie różni się znacząco od oryginalnego rozwiązania. To mocny dowód na to, że oba rozwiązania są nieprawidłowe dla dużych t. Co więcej, zgodność obu rozwiązań do tego punku może być potraktowana jako dowód na to, że oba rozwiązania są poprawne.

In[113]:=
```
(* Mamy sposób, poprzez WorkingPrecision,
```
dokładność robocza
```
by uczynić rozwiązanie bardziej dokładnym *)
```

In[114]:=
```
ode = {odeDuff[1, 1, 15 / 100, 3 / 10], x[0] == 1, x'[0] == -1};
xs = x /. First@NDSolve[ode, x, {t, 0, 100}, WorkingPrecision → 20];
```
pierw··· rozwiąż numerycznie równanie różni··· dokładność robocza
```
eps = 10^-8;
ode = {odeDuff[1, 1, 15 / 100, 3 / 10], x[0] == 1 + eps, x'[0] == -1 + eps};
xseps = x /. First@NDSolve[ode, x, {t, 0, 100}, WorkingPrecision → 20];
```
pierw··· rozwiąż numerycznie równanie różni··· dokładność robocza

In[119]:=
```
Plot[ {xs[t], xseps[t]}, {t, 0, 100} ]
```
wykres

Out[119]=



Zwiększając precyzję obliczeń (poprzez ustawienie parametru **WorkingPrecision→20**) możemy wydłużyć czas, po jakim oba rozwiązania się rozbiegają. Jest to pesymistyczny przypadek, dla którego całkowanie dla dużych czasów wymaga coraz większej precyzji.

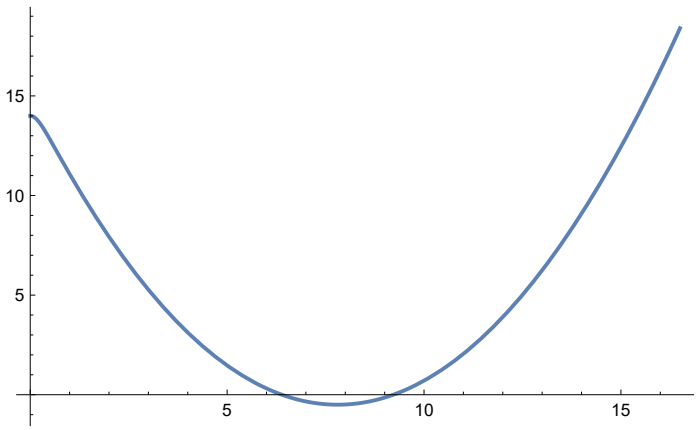
Przykład 3

In[120]:=
```
ode = {x''[t] == x'[t]^2 - x[t], x[0] == 14, x'[0] == 0};
```

In[121]:=
```
xs = x /. First@NDSolve[ode, x, {t, 0, 16.5}];
```
pierw··· rozwiąż numerycznie równanie różniczkow

In[122]:=
```
Plot[xs[t], {t, 0, 16.5}, PlotRange → All]
```
wykres zakres wykresu wszys

Out[122]=

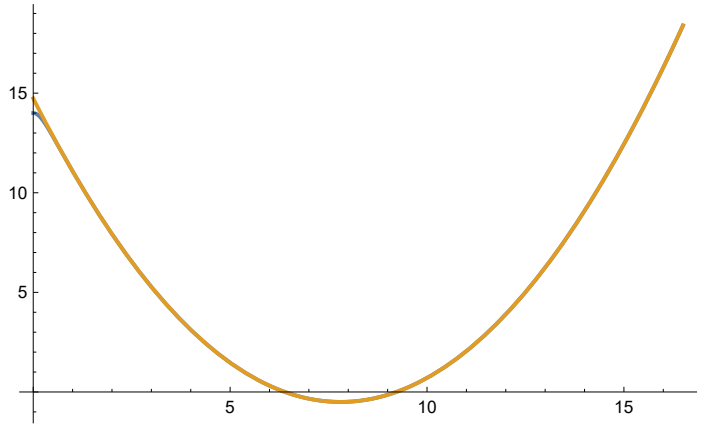


Częstym sposobem sprawdzenia poprawności otrzymanych wyników numerycznych jest całkowanie równania wstecz.

In[123]:=
```
ode = {x''[t] == x'[t]^2 - x[t], x[16.5] == xs[16.5], x'[16.5] == xs'[16.5]};
xb = x /. First@NDSolve[ode, x, {t, 16.5, 0}];
```
pierw··· rozwiąż numerycznie równanie różniczkowe

In[125]:=
```
Plot[{xs[t], xb[t]}, {t, 0, 16.5}, PlotRange → All]
```
wykres zakres wykresu wszys

Out[125]=



In[126]:=
```
{xs[0], xb[0]}
```

Out[126]=
```
{14., 14.7341}
```

In[127]:=
```
(* Widać, że te wyniki nie zgadzają się
 idealnie i dają informację o dokładności obliczeń. *)
```

In[128]:=
```
ClearAll["Global`*"];
```
wyczyść wszystko

Informacje o problemach związanych z numerycznym rozwiązywaniem równań różniczkowych nie powinny Państwa zniechęcać. Jako fizycy mamy własne sposoby, by sprawdzić jakość rozwiązań - zasady zachowania !
Pokazuję to dla nieco ambitniejszego problemu, w którym szukamy rozwiązania układu równań różniczkowych II rzędu z czterema funkcjami. Chodzi o tzw. problem Keplera.

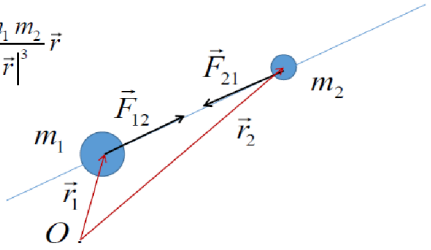
Dwie masy, m_1 i m_2 , poruszają się w ustalonej płaszczyźnie pod wpływem wzajemnej siły grawitacji. Opisujemy ruch tego układu, rozwiązując numerycznie równanie Newtona dla wektorów położenia $\vec{r}_1$ i $\vec{r}_2$.

m_1= 4 MS, m_2= **MS**

$$m_1 \frac{d^2\vec{r}_1}{dt^2} = \vec{F}_{12} = G\frac{m_1 m_2}{|\vec{r}|^3}\vec{r}$$

$$m_2 \frac{d^2\vec{r}_2}{dt^2} = \vec{F}_{21} = -G\frac{m_1 m_2}{|\vec{r}|^3}\vec{r}$$

$$\vec{r} \equiv \vec{r}_2 - \vec{r}_1$$



In[129]:=
```
r1 = {x1[t], y1[t]};
```

In[130]:=
```
r2 = {x2[t], y2[t]};
```

In[131]:=
```
r = r2 - r1;
```

In[132]:=
```
(* II zasada dynamiki dla masy m1 *)
```

In[133]:=
```
eq1 = m1 * D[r1, {t, 2}] == G * m1 * m2 / (Sqrt[r.r]) ^ 3 * r
```
oblicz pochodną pierwiastek kwadratow

Out[133]=
$$\{m1\,x1''[t],\ m1\,y1''[t]\} == \left\{\frac{G\,m1\,m2\,(-x1[t]+x2[t])}{\left((-x1[t]+x2[t])^2+(-y1[t]+y2[t])^2\right)^{3/2}},\ \frac{G\,m1\,m2\,(-y1[t]+y2[t])}{\left((-x1[t]+x2[t])^2+(-y1[t]+y2[t])^2\right)^{3/2}}\right\}$$

In[134]:=
```
(* II zasada dynamiki dla masy m2 *)
```

In[135]:=
```
eq2 = m2 * D[r2, {t, 2}] == -G * m1 * m2 / (Sqrt[r.r]) ^ 3 * r
```
oblicz pochodną pierwiastek kwadratow

Out[135]=
$$\{m2\,x2''[t],\ m2\,y2''[t]\} == \left\{-\frac{G\,m1\,m2\,(-x1[t]+x2[t])}{\left((-x1[t]+x2[t])^2+(-y1[t]+y2[t])^2\right)^{3/2}},\ -\frac{G\,m1\,m2\,(-y1[t]+y2[t])}{\left((-x1[t]+x2[t])^2+(-y1[t]+y2[t])^2\right)^{3/2}}\right\}$$

In[136]:=
```
(* MS=2*10^(30) *) (* kg , masa naszego Słońca *)
```

In[137]:=
```
MS = 1;
```

In[138]:=
```
m1 = 4 * MS;
```

In[139]:=
```
m2 = MS;
```

In[140]:=
```
AU = 1; (* jednostka astronomiczna,
czyli 150 milionów kilometrów jest teraz jednostką długości ! *)
```

In[141]:=
```
rmin = 24 * AU;
```

In[142]:=
```
rmax = 36 * AU;
```

In[143]:=
```
a = (rmin + rmax) / 2;
```

In[144]:=
```
(* ziemski rok jest teraz jednostką czasu *)
```

In[145]:=
```
rok = 1;
```

In[146]:=
```
T = 80 * rok;
```

In[147]:=
```
(* Po wyborze okresu T oraz parametru elipsy w ruchu względnym,
wartość stałej G jest już określona i możemy ją znaleźć z III prawa Keplera *)
```

In[148]:=
```
solG = Solve[2 * Pi * a^(3 / 2) / Sqrt[G * (m1 + m2)] == T, G];
       rozwiąż ···  pi              pierwiastek kwadratowy
```

In[149]:=
```
G = G /. solG[[1]];
```

In[150]:=
```
tmax = 1500;
```

In[151]:=
```
(* Warunki początkowe w odpowiednich jednostkach. Są one tak dobrane,
by środek masy układu pozostawał w spoczynku *)
```

In[152]:=
```
WP = {x1[0] == -912 / 100, y1[0] == 0, x2[0] == 3648 / 100, y2[0] == 0,
   x1'[0] == 0, y1'[0] == -48 / 100, x2'[0] == 0, y2'[0] == 192 / 100};
```

In[153]:=
```
(* Używam funkcji NDSolve, aby numerycznie rozwiązać układ równań różniczkowych *)
                    rozwiąż numerycznie równanie różniczkowe
```

In[154]:=
```
numerycznie = NDSolve[{eq1, eq2, WP}, {x1[t], y1[t], x2[t], y2[t]}, {t, 0, tmax}];
              rozwiąż numerycznie równanie różniczkowe
```

In[155]:=
```
(* Wydobywam rozwiązania w postaci funkcji *)
```

In[156]:=
```
x1[t_] := Evaluate[x1[t] /. Flatten[numerycznie]]
          oblicz             spłaszcz
```

In[157]:=
```
y1[t_] := Evaluate[y1[t] /. Flatten[numerycznie]]
          oblicz             spłaszcz
```

In[158]:=
```
x2[t_] := Evaluate[x2[t] /. Flatten[numerycznie]]
          oblicz             spłaszcz
```

In[159]:=
```
y2[t_] := Evaluate[y2[t] /. Flatten[numerycznie]]
          oblicz             spłaszcz
```

In[160]:=
```
ClearAll[r1, r2, r];
wyczyść wszystko
```

In[161]:=
```
(* Na podstawie numerycznych rozwiązań tworzę
 wektory położenia obu mas oraz wektor położenia względnego *)
```

In[162]:=
```
r1[t_] := {x1[t], y1[t]};
```

In[163]:=
```
r2[t_] := {x2[t], y2[t]};
```

In[164]:=
```
r[t_] := r2[t] - r1[t];
```

In[165]:=
```
(* całkowita (dla obu mas) energia kinetyczna *)
```

In[166]:=
```
Ekin[t_] = (1 / 2) * m1 * D[r1[t], t].D[r1[t], t] + (1 / 2) * m2 * D[r2[t], t].D[r2[t], t];
```
oblicz pochodną oblicz pochodną oblicz pochodną oblicz pochodną

In[167]:=
```
(* całkowita (dla obu mas) energia potencjalna *)
```

In[168]:=
```
Epot[t_] = -G * m1 * m2 / Sqrt[r[t].r[t]];
```
pierwiastek kwadratowy

In[169]:=
```
(* Całkowita energia mechaniczna układu dwóch mas *)
```

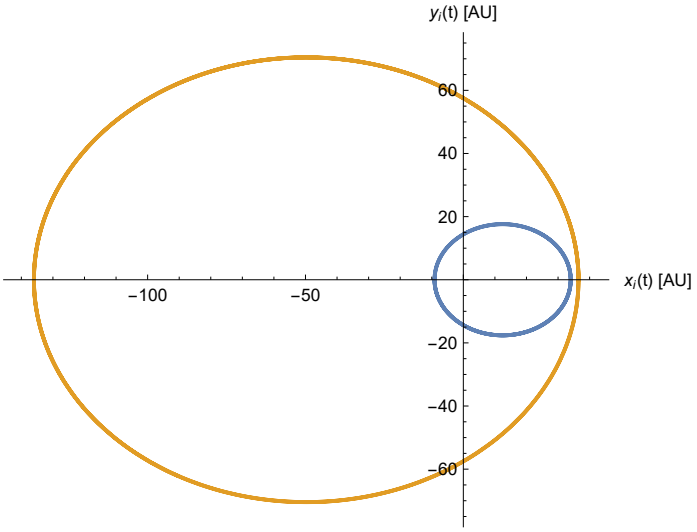
In[170]:=
```
Etot[t_] = Ekin[t] + Epot[t];
```

In[171]:=
```
(*
px1=Plot[x1[t],{t,0,tmax},PlotRange→All,AxesLabel→{"t [rok]","x₁(t) [AU]"}];
px1=Plot[x1[t],{t,0,tmax},PlotRange→All,AxesLabel→{"t [rok]","x₁(t) [AU]"}];
py1=Plot[y1[t],{t,0,tmax},PlotRange→All,AxesLabel→{"t [rok]","y₁(t) [AU]"}];
py1=Plot[y1[t],{t,0,tmax},PlotRange→All,AxesLabel→{"t [rok]","y₁(t) [AU]"}];
px2=Plot[x2[t],{t,0,tmax},PlotRange→All,AxesLabel→{"t [rok]","x₂(t) [AU]"}];
px2=Plot[x2[t],{t,0,tmax},PlotRange→All,AxesLabel→{"t [rok]","x₂(t) [AU]"}];
py2=Plot[y2[t],{t,0,tmax},PlotRange→All,AxesLabel→{"t [rok]","y₂(t) [AU]"}];
py2=Plot[y2[t],{t,0,tmax},PlotRange→All,AxesLabel→{"t [rok]","y₂(t) [AU]"}];
*)
```
wykres zakres wyk··· w··· oznaczenia osi

In[172]:=
```
(* Tory obu mas, niebieski dla większej masy m1 *)
```

In[173]:=
```
p1 = ParametricPlot[{{x1[t], y1[t]}, {x2[t], y2[t]}},
                    wykres parametryczny
  {t, 0, tmax}, AxesLabel → {"x_i(t) [AU]", "y_i(t) [AU]"}]
                              oznaczenia osi
```

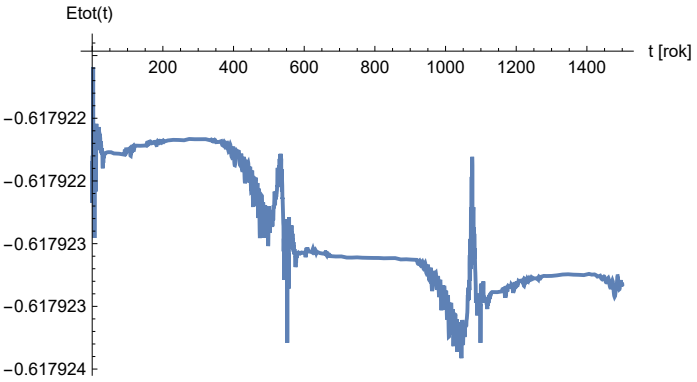
Out[173]=



In[174]:=
```
(* Sprawdzam jakość
 rozwiązania: całkowita energia mechaniczna nie może się zmieniać w czasie *)
```

In[175]:=
```
Plot[Etot[t], {t, 0, tmax}, PlotRange → All, AxesLabel → {"t [rok]", "Etot(t) "}]
wykres                      zakres wykresu ws···  oznaczenia osi
```

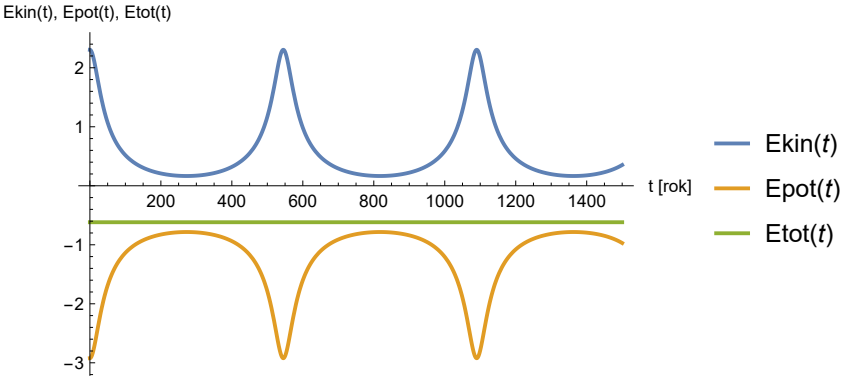
Out[175]=



In[176]:=
```
(* Numeryczne zmiany całkowitej energii są bardzo niewielkie,
ale warto je dodatkowo porównać z wartościami energii kinetycznej,
potencjalnej i całkowitej *)
```

In[177]:=
```
Plot[{Ekin[t], Epot[t], Etot[t]}, {t, 0, tmax}, PlotRange → All,
 AxesLabel → {"t [rok]", "Ekin(t), Epot(t), Etot(t) "}, PlotLegends → "Expressions"]
```
wykres zakres wykresu wszystko
oznaczenia osi legenda dla grafik

Out[177]=



In[178]:=
```
(* Na tym rysunku widać,
że spełniona jest zasada zachowania całkowitej energii mechanicznej *)
```

Modelowanie ruchu układu przy pomocy wbudowanych obiektów graficznych

In[179]:=
```
masa1[t_] = Disk[{x1[t], y1[t]}, m1];
```
koło

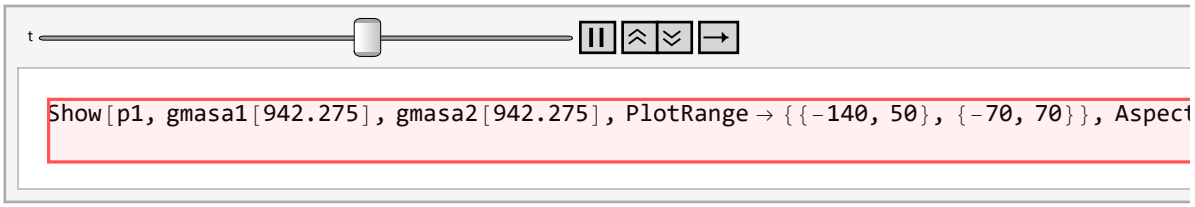
In[180]:=
```
masa2[t_] = Disk[{x2[t], y2[t]}, (1 / 2) * m1];
```
koło

In[181]:=
```
gmasa1[t_] := Graphics[masa1[t]]
```
grafika

In[182]:=
```
gmasa2[t_] := Graphics[masa2[t]]
```
grafika

In[183]:=
```
animacja = Animate[Show[p1, gmasa1[t], gmasa2[t],
 PlotRange → {{-140, 50}, {-70, 70}}, AspectRatio → Automatic], {t, 0, tmax}]
```
animuj pokaż
zakres wykresu format obrazu automatyczny

Out[183]=



In[184]:=
```
(* SetDirectory[NotebookDirectory[]]; Export["2masy_2.avi",animacja]; *)
```
ustaw katalog r··· katalog notatnika eksportuj

In[185]:=
```
ClearAll["Global`*"];
```
wyczyść wszystko

Równania różniczkowe cząstkowe

DSolve służy nie tylko do rozwiązywania równań i układów równań różniczkowych zwyczajnych dla funkcji jednej zmiennej, ale także równań różniczkowych cząstkowych, gdzie występują funkcje wielu zmiennych.

W tym przypadku uzyskanie jednoznacznego rozwiązania wymaga zwykle podania dodatkowych warunków (tzw. warunków początkowych i brzegowych). Rozwiązanie ogólne może zależeć nie od dowolnych parametrów, ale wręcz od dowolnych funkcji !

Przykład 1 : równanie I rzędu na funkcję dwóch zmiennych
(S. Wolfram)

In[186]:=
```
DSolve[D[y[x1, x2], x1] + D[y[x1, x2], x2] == 1 / (x1 x2), y[x1, x2], {x1, x2}]
```
rozwią··· oblicz pochodną oblicz pochodną

Out[186]=
$$\left\{\left\{y[x1, x2] \to \frac{-\text{Log}[x1] + \text{Log}[x2] + (x1 - x2)\, C_1[-x1 + x2]}{x1 - x2}\right\}\right\}$$

In[187]:=
```
(* Rozwiązanie ogólne zależy od dowolnej funkcji różnicy x2-
 x1. Mathematica nazywa ją C[1]. *)
```

Przykład 2 : problem dotyczący pewnego rodzaju sił, dla których ma istnieć energia potencjalna.
Mamy sprawdzić, czy siły F={2*x*z*z-2*y,-2*x-6*y*z,2*x*x*z-3*y*y} i G={x*x*z,-x*y,5} należą do tej kategorii i jeśli tak, znaleźć odpowiadające im energie potencjalne.

In[188]:=
```
F := {2 * x * z * z - 2 * y, -2 * x - 6 * y * z, 2 * x * x * z - 3 * y * y};
```

$$\text{rot}\,\vec{F} \equiv \nabla \times \vec{F} = \begin{vmatrix} \hat{x} & \hat{y} & \hat{z} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ F_x & F_y & F_z \end{vmatrix} =$$

$$\left(\frac{\partial F_z}{\partial y} - \frac{\partial F_y}{\partial z}\right)\hat{x} + \left(\frac{\partial F_x}{\partial z} - \frac{\partial F_z}{\partial x}\right)\hat{y} + \left(\frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y}\right)\hat{z}$$

In[189]:=
```
rotF = Curl[F, {x, y, z}]
```
rotacja

Out[189]=
```
{0, 0, 0}
```

In[190]:=
```
G := {x * x * z, -x * y, 5};
```

In[191]:=
```
rotG = Curl[G, {x, y, z}]
```
rotacja

Out[191]=
$$\{0, x^2, -y\}$$

$$\operatorname{rot}\vec{F}(\vec{r}) = 0$$

$$\rightarrow \exists V(\vec{r}) : \vec{F}(\vec{r}) = -\nabla V(\vec{r}) \equiv -\operatorname{grad} V(\vec{r})$$

$$\vec{F}(\vec{r}) = -\nabla V(\vec{r}) \leftrightarrow F_x = -\frac{\partial V}{\partial x}, F_y = -\frac{\partial V}{\partial y}, F_z = -\frac{\partial V}{\partial z}.$$

In[192]:=
```
(* Gradient funkcji skalarnej to wektor,
   wektor gradientowy
którego składowymi są pochodne odpowiednio po x, y i z. W ogólnym zapisie mamy: *)
```

In[193]:=
```
Grad[f[x, y, z], {x, y, z}]
gradient
```

Out[193]=
```
{f^(1,0,0)[x, y, z], f^(0,1,0)[x, y, z], f^(0,0,1)[x, y, z]}
```

In[194]:=
```
(* Powyższy zapis może nie być zbyt czytelny,
dlatego sprawdzam bezpośrednio, czym jest gradient *)
```

In[195]:=
```
Grad[f[x, y, z], {x, y, z}] == {D[f[x, y, z], x], D[f[x, y, z], y], D[f[x, y, z], z]}
gradient                       oblicz pochodną     oblicz pochodną     oblicz pochodną
```

Out[195]=
```
True
```

In[196]:=
```
(* Szukamy energii potencjalnej dla siły F *)
```

In[197]:=
```
DSolve[Grad[VF[x, y, z], {x, y, z}] == -F, VF[x, y, z], {x, y, z}]
rozwią··· gradient
```

Out[197]=
```
{{VF[x, y, z] → 2 x y + z (3 y^2 - x^2 z) + C[1]}}
```

In[198]:=
```
(* Rozwiązanie na energię potencjalną jest określone z dokładnością do stałej *)
```

In[199]:=
```
DSolve[Grad[VG[x, y, z], {x, y, z}] == G, VG[x, y, z], {x, y, z}]
rozwią··· gradient
```

Out[199]=
```
DSolve[{VG^(1,0,0)[x, y, z], VG^(0,1,0)[x, y, z], VG^(0,0,1)[x, y, z]} == {x^2 z, -x y, 5},
 VG[x, y, z], {x, y, z}]
```

In[200]:=
```
(* Zgodnie z oczekiwaniem, dla siły G Mathematica nie znalazła rozwiązania *)
```

In[201]:=

Przykład 3: jednowymiarowe (jedna współrzędna przestrzenna x) równanie falowe. Mamy do czynienia ze struną rozciągającą się wzdłuż osi x, której wychylenie z położenia równowagi zależy od położenia wzdłuż osi x i od czasu.

In[202]:=
```
ClearAll["Global`*"];
```
wyczyść wszystko

In[203]:=
```
$Assumptions = c > 0;
```
domyślne założenia

In[204]:=
```
eq3 = {D[u[x, t], {x, 2}] - (1 / c^2) * D[u[x, t], {t, 2}] == 0}
```
oblicz pochodną oblicz pochodną

Out[204]=
$$\left\{-\frac{u^{(0,2)}[x, t]}{c^2} + u^{(2,0)}[x, t] == 0\right\}$$

In[205]:=
```
s3 = Simplify[DSolve[eq3, u[x, t], {x, t}]]
```
uprość rozwiązywanie równań różniczkowych

Out[205]=
$$\left\{\left\{u[x, t] \to C_1\left[t - \frac{x}{c}\right] + C_2\left[t + \frac{x}{c}\right]\right\}\right\}$$

W równaniu powyższym nie występowały ani warunki początkowe, ani warunki brzegowe. Dlatego rozwiązaniem jest suma dwóch dowolnych funkcji jednej zmiennej, C_1 i C_2, z argumentami równymi odpowiednio t-x/c oraz t+x/c. Te dwie części rozwiązania odpowiadają fali biegnącej zgodnie ze zwrotem osi x (argument t-x/c) oraz przeciwnie do zwrotu osi x (t+x/c).

Teraz wprowadzamy warunki początkowe i brzegowe.
Warunki początkowe dla równania drugiego rzędu w czasie oznaczają, że musimy podać postać funkcji u i jej pierwszej pochodnej w chwili t=0.

Dla fali na strunie o skończonej długości, x ∈ [-L,L], odpowiada to podaniu kształtu struny w chwili t=0 oraz rozkładu prędkości punktów struny w chwili t=0.

In[206]:=
```
wp3 = {u[x, 0] == Exp[-x^2], Derivative[0, 1][u][x, 0] == 0}
```
funkcja ek··· pochodna

Out[206]=
$$\left\{u[x, 0] == e^{-x^2}, u^{(0,1)}[x, 0] == 0\right\}$$

Warunki brzegowe: zakładamy, że fala występuje na strunie, która jest sztywno umocowana na obu końcach. Konkretne wartości wybieram tak, by warunki początkowe były zgodne z warunkami brzegowymi.

In[207]:=
```
L = 1;
```

In[208]:=
```
wb3 = {u[-L, t] == E^(-1), u[L, t] == E^(-1)}
```
liczba Eulera liczba Euler

Out[208]=
$$\left\{u[-1, t] == \frac{1}{e}, u[1, t] == \frac{1}{e}\right\}$$

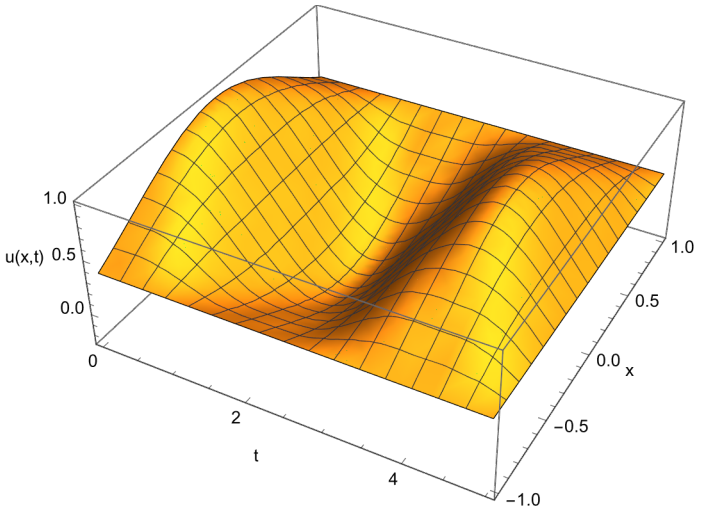
In[209]:=
```
c = 1; tmax = 5;
```

In[210]:=
```
s31 = NDSolve[{eq3, wp3, wb3}, u, {x, -L, L}, {t, 0, tmax}]
```
rozwiąż numerycznie równanie różniczkowe

Out[210]=
```
{{u → InterpolatingFunction[ Domain: {{-1., 1.}, {0., 5.}}
                             Output: scalar ]}}
```

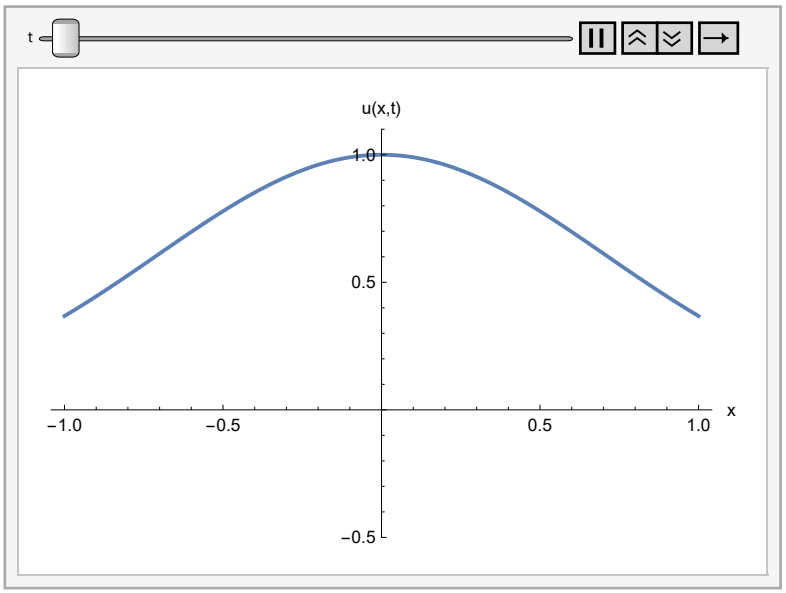
In[211]:=
```
Plot3D[Evaluate[u[x, t] /. s31[[1]]],
 {t, 0, tmax}, {x, -L, L}, AxesLabel → {"t", "x", "u(x,t)"}]
```
wykres ⋯ oblicz — oznaczenia osi

Out[211]=



In[212]:=
```
Animate[Plot[Evaluate[u[x, t] /. s31[[1]]], {x, -L, L},
 PlotRange → {-1 / 2, 1.1}, AxesLabel → {"x", "u(x,t)"}], {t, 0, 4}]
```
animuj — wyk⋯ oblicz — zakres wykresu — oznaczenia osi

Out[212]=



Dalsze przykłady są dostępne na stronie przedmiotu
http://users.uj.edu.pl/~golak/zestawyNOF.html

oraz na stronie
http://users.uj.edu.pl/~golak/zestawyF.html