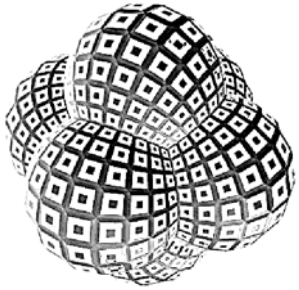


Zborník konferencie o vyučovaní matematiky na VŠ pomocou programového systému firmy

WOLFRAM RESEARCH, INC. USA
MATHEMATICA®

MATHEMATICA '99

Numerické riešenie diferenciálnych rovníc a systémov pomocou numerických funkcií systému *MATHEMATICA®*

Monika Kováčová[1]

Katedra Matematiky, Strojnícka fakulta STU

1.Úvod

Aj napriek dobrým schopnostiam príkazu **`DSolve[]`**, principiálne len veľmi málo diferenciálnych rovníc má analytické riešnie. Ak príkaz **`DSolve[]`**nemôže nájsť riešenie, musíme použiť dostupné numerické techniky na nájdenie približného riešenia. **`NDSolve[]`** je definovaná funkcia programu *MATHEMATICA*, ktorá umožňuje riešiť diferenciálne rovnice numericky. Má tvar:

`NDSolve[{rovnica1, rovnica2, ...},y,{x, xmin, xmax}]`
nájde numerické riešenie pre funkciu y a x z intervalu [xmin, xmax]
`NDSolve[{rovnica1,rovnica2,...},{y1,y2,...},{x, xmin, xmax}]`
nájde numerické riešenie pre viaceré funkcie yi a x z intervalu [xmin, xmax]

Cieľom tohto článku nie je zopakovať úvodnú prednášku základného kurzu matematiky o diferenciálnych rovniciach. Preto nebudeme presne definovať, čo rozumieme diferenciálnou rovnicou, ako je definované jej riešenie, aké základné typy dif. rovníc poznáme a ako vyzerajú algoritmy ich riešenia. Vzhľadom k už uvedenému prosíme čitateľa o prepáčenie niektorých nepresností vo vyjadrovaní v nasledujúcich riadkoch. Chceme skôr poukázať na problémy, ktoré vznikajú pri snahe numericky riešiť dif. rovnice pomocou pg. systému *MATHEMATICA* a na možnosti, ako sa týmto problémom vyhnúť, resp. ako ich odstrániť.

Ak riešime diferenciálnu rovnicu bez hraničných podmienok symbolickými technikami získame všeobecné riešenie tejto rovnice, ktoré obsahuje konštanty. Dosadením

[1] Mgr. Monika Kováčová, Katedra Matematiky, Strojnícka fakulta STU, Nám. Slobody 17, 812 31 Bratislava, kovacova_v@dekan.sjf.stuba.sk, kovacova@cvt.stuba.sk

hraničných podmienok (počiatočných, alebo okrajových) do všeobecného riešenia dostaneme riešenie počiatočnej (okrajovej) úlohy.

Pri numerickom riešení obyčajnej diferenciálnej rovnice hrajú hraničné podmienky významnú úlohu - určujú možnú metódu riešenia dif. rovnice.

Ak hraničné podmienky sú dané len v jednom bode nezávisle premennej x hovoríme o počiatočných podmienkach a úlohu nazývame *počiatočnou úlohou (initial value problem).* Ak zahŕňajú viac ako jeden bod úlohu nazývame *okrajovou úlohou (boundary value problem).*

napr. dif. rovnica s počiatočnými podmienkami y[0]==1, y'[0]==2 je počiatočnou úlohou a rovnica s podmienkami y[0]==1, y[2]==0 je okrajovou úlohou.

Obe triedy úloh sú numericky dôležité. Aktuálna verzia príkazu **NDSolve[]** implementovaná vo verzii *MATHEMATICA* 3.0 podporuje len počiatočné úlohy. V ďalšej časti tohto článku však poskytneme čitateľovi ukážky, ako možno v niektorých prípadoch toto obmedzenie obísť. Niektoré boundary value problem je možné riešiť len použitím dodatočných programových balíkov, podrobnejší popis možností týchto balíkov je možné nájť v Standard Add-On Packages manuále.

Vstupné parametre príkazu **NDSolve[]** musia obsahovať nielen diferenciálnu rovnicu, ktorú treba riešiť, ale tiež postačujúci počet počiatočných podmienok. Vo všeobecnosti diferenciálna rovnica s deriváciami do *n*-tého rádu požaduje počiatočné podmienky do *(n-1)* derivácie. Pre systém dif. rovníc, počet počiatočných podmienok je rovný súčtu rádov podmienok pre jednotlivé rovnice.

Druhý argument v príkaze **NDSolve[]** udáva nezávisle premennú a interval, pre ktorý potrebujeme riešiť počiatočnú úlohu. Pre túto nezávisle premennú je potrebné určiť interval konečnej veľkosti, na ktorom treba hľadať riešenie dif. rovnice.
Jednoduchý príklad použitia tohto príkazu:

In[1]:=

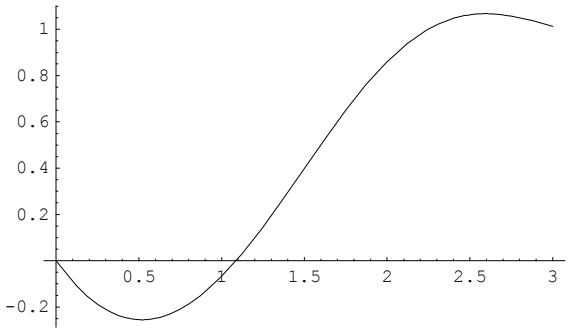
```
NDSolve[{y'''[x] + x y''[x]  + x^2 y'[x] + Sin[x] y[x] ==0, y''[0]==2,
y'[0]==-1, y[0]==0},y[x],{x,0,3}]
```

Out[1]=

```
{{y[x] -> InterpolatingFunction[{{0., 3.}}, <>][x]}}
```

In[2]:=

```
Plot[y[x]/.%,{x,0,3}]
```



Out[2]=

```
-Graphics-
```

2. Interpolačné funkcie

Príkaz **NDSolve[]**vráti riešenie diferenciálnej rovnice ako objekt tvaru `InterpolatingFunction`, ktorý v skutočnosti tvorí funkcia po častiach polynomická. (Tento objekt okrem iných vie vytvoriť aj funkcia **Interpolation[]**.)

V skutočnosti sa tento objekt skladá z dvoch častí. Prvá určuje interval hodnôt nadobúdaných nezávisle premennou, pre ktorý príkaz našiel približné riešenie. Druhá časť je tabuľkou koeficientov nutných pre vytvorenie interpolačnej funckie. Len pre úplnosť pripomeňme, že interval hodnôt nie vždy zodpovedá intervalu, ktorý sme zadali vo vstupe funkcie **NDSolve[].**

Príklad: Rovnica $y' = \frac{1}{(2-x)^2}$ s poč. podm. y(0) = $\frac{1}{2}$ obsahuje singularitu na pravej strane v bode x =2. Ak požadujeme numerické riešenie na intervale [0,4], *MATHEMATICA* nás vo výstupe na túto chybu upozorní.

In[3]:=

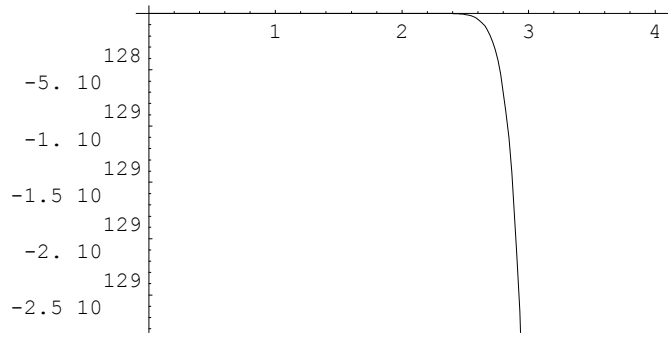
```
NDSolve[{y'[x] == 1/(2-x)^2,y[0]==1/2},y[x],{x,0,4}]

NDSolve::ndsz:
   At x == 2., step size is effectively zero; singularity suspected.
```

Out[3]=

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

In[4]:=

```
Plot[y[x]/.%,{x,0,4}]

InterpolatingFunction::dmval:
   Input value {2.01686} lies outside the range of data in the
     interpolating function. Extrapolation will be used.

InterpolatingFunction::dmval:
   Input value {2.00736} lies outside the range of data in the
     interpolating function. Extrapolation will be used.
```



Out[4]=

```
-Graphics-
```

3. Ďalšie možnosti príkazu `NDSolve[]`

`NDSolve[]` používa "viackrokové metódy " na výpočet riešenia.

Ak máme určené počiatočné podmienky diferenciálnej rovnice, štartujúca hodnota pre výpočet nasledujúcej hodnoty y_1 je daná implicitne. V tomto prípade stačí tretí argument príkazu **`NDSolve[]`** uviesť v tvare $\{x, x_{max}\}$. Ak uvedieme tento argument v tvare $\{x, x_{min}, x_{max}\}$ za štartujúcu hodnotu *MATHEMATICA* zoberie hodnotu x_{min}. Ak sú hodnoty x_0 (v počiatočnej podmienke) a hodnota x_{min} rozdielne, ako štartujúcu hodnotu pre výpočet systém *MATHEMATICA* zoberie hodnotu x_{min} a ignoruje počiatočnú podmienku. Ošetrený je aj prípad $x_0 \in [x_{min}, x_{max}]$, ale $x_0 \neq x_{min}$. V tomto prípade objekt `InterpolatingFunction` vznikne zložením dvoch objektov tohto typu vytvorených na intervaloch $[x_{min}, x_0]$ a $[x_0, x_{min}]$.

In[5]:=

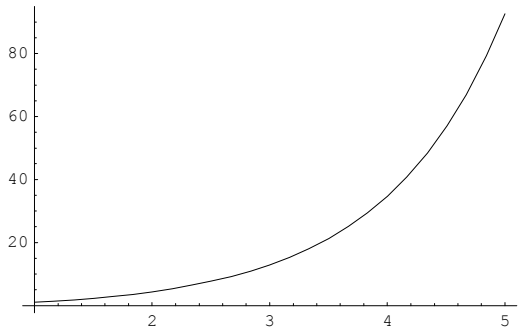
```
ries=NDSolve[{y'[x] ==y[x] + Sin[x], y[1]==1},y,{x,5}]
```

Out[5]=

```
{{y -> InterpolatingFunction[{{1., 5.}}, <>]}}
```

In[6]:=

```
Plot[y[x]/.ries,{x,1,5}]
```



Out[6]=

```
-Graphics-
```

In[7]:=

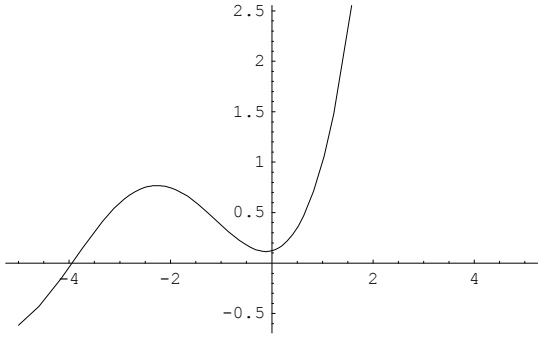
```
ries=NDSolve[{y'[x] ==y[x] + Sin[x], y[1]==1},y,{x,-5, 5}]
```

Out[7]=

```
{{y -> InterpolatingFunction[{{-5., 5.}}, <>]}}
```

In[8]:=

```
Plot[y[x]/.ries,{x,-5,5}]
```



Out[8]=

```
-Graphics-
```

Nie je neobvyklé, ak diferenciálne rovnice obsahujú singularity. Príkaz **NDSolve[]** má z programátorského hľadiska ošetrený aj tento problém. Ak systém *MATHEMATICA* detekuje singularitu, upravuje dĺžku kroku výpočtu. Keď je krok príliš malý, výpočet je (aj predčasne) ukončený. Minimálnu hodnotu dĺžky kroku dokážeme, tak ako u ostatných N funkcií programu *MATHEMATICA*, ovplyvniť parametrami `WorkingPrecision, AccuracyGoal a PrecisionGoal.` [5]

Riešenie nasledujúcej rovnice má na intervale [0,5] singularitu v prvom nulovom bode funkcie *Cos[x]*. Vidíme to aj na nakreslenom grafe riešenia.

In[9]:=

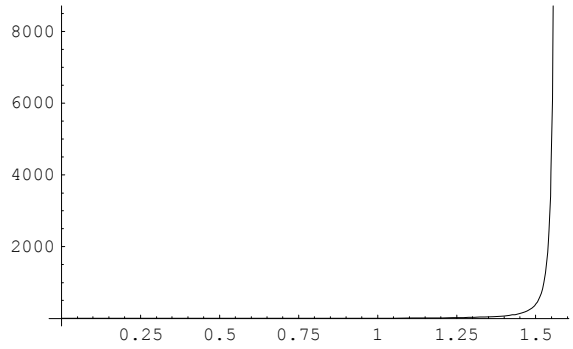
```
ries=NDSolve[{y'[x] ==Cos[x] y[x]^2, y[0]==1},y,{x,5}]

NDSolve::ndsz:
   At x == 1.56645, step size is effectively zero; singularity
suspected.
```

Out[9]=

```
{{y -> InterpolatingFunction[{{0., 1.56645}}, <>]}}
```

In[10]:=

```
Plot[y[x]/.ries,{x,0,1.566}]
```



Out[10]=

```
-Graphics-
```

Parametre *AccuracyGoal a PrecisionGoal* kontrolujú očakávanú správnosť a presnosť výpočtu. Väčšiu správnosť a presnosť výpočtu dosiahneme nastavením hodnôt týchto parametrov na vyššie hodnoty. V práci [5] môže čitateľ nájsť podrobnejšie vysvetlenie súvisu týchto parametrov s option *WorkingPrecision*. Default hodnoty pre *AccuracyGoal* a *PrecisionGoal* sú nastavené ako Automatic, čo znamená *w-10*, kde *w* je *WorkingPrecision*. [1]

Riešme diferenciálnu rovnicu, u ktorej poznáme explicitné riešenie a porovnajme toto riešenie s jeho numerickým riešením. Použime počítačovú aritmetiku.

In[11]:=

```
ries=NDSolve[{y'[x]== y[x], y[0]==1},y,{x,2}]
```

[1] Rozne počítače majú nastavené rôzne počítačové aritmetiky. Ak systém nutne nedefinuje vlatnú aritmetiku, hodnota Automatic sa rovná 6. Výnimkou je tzv. aritmetika veľkých čísel, ktorá by vyžadovala samostatný príspevok

Out[11]=

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

In[12]:=

```
Exp[0.51]-y[0.51]/.ries[[1]]
```

Out[12]=

```
                -6
-4.87479 10
```

Porovnaním výsledkov vidíme, že výsledky sú vypočítané s presnosťou na 6 platných číslic.

Zopakujme výpočet, ale nastavne väčšie hodnoty *PrecisionGoal* a *AccuracyGoal*

In[13]:=

```
ries=NDSolve[{y'[x] ==y[x], y[0]==1},y,{x,2},
     AccuracyGoal->10, PrecisionGoal->10]
```

Out[13]=

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

In[14]:=

```
Exp[0.51]-y[0.51]/.ries[[1]]
```

Out[14]=

```
                -9
-1.35491 10
```

Teraz má riešenie presnosť na 9 miest

Radi by sme pripomenuli, že hodnoty parametrov *AccuracyGoal* a *PrecisionGoal* sú len cieľom. Aktuálna presnosť a správnosť sa mení v závislosti od problému. Ak zväčšíme hodnoty *AccuracyGoal,* alebo *PrecisionGoal*, vzrastie aj hodnota *WorkingPrecision*.

Odporúčame, aby parameter *WorkingPrecision* bol väčší aspoň o niekoľko platných číslic než parametre *AccuracyGoal* alebo *PrecisionGoal*. V nasledovnom príklade je príliš nízko nastavená hodnota *WorkingPrecision*.

In[15]:=

```
NDSolve[{y'[x]==y[x],y[0]==1},y,{x,5}]
```

Out[15]=

```
{{y -> InterpolatingFunction[{{0., 5.}}, <>]}}
```

Tu je už hodnota tohoto parametra postačujúca:

In[16]:=

```
NDSolve[{y'[x]==y[x],y[0]==1},y,{x,5},AccuracyGoal->14,
       PrecisionGoal->14,WorkingPrecision->20]
```

Out[16]=

```
{{y -> InterpolatingFunction[{{0, 5.000000000000000000}}, <>]}}
```

Dĺžku kroku ovplyvňuje aj nastaviteľný parameter *MaxStep*. Výpočet sa ukončí ak je dĺžka kroku príliš malá, alebo ak počet opakovaní prekročí default hodnotu tohto parametra. Vo verzii 2.2 je default hodnota nastavená na 300, vo verzii 3.0 a vyššie na 500. Hodnota *MaxStep -> Infinity* veľmi efektívne imituje limitu. Treba však vždy zvážiť, či takto

nastavená hodnota kroku nespôsobí nekonečné delenie kroku v blízkosti singularity. Obvykle je default hodnota postačujúca pre bežne riešené úlohy.

In[17]:=

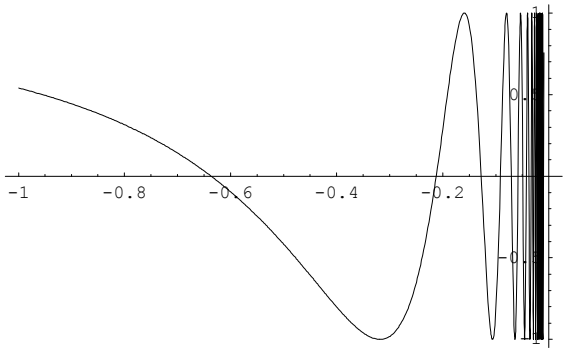
```
ries=NDSolve[{y'[x]==Sin[1/x] /x^2,y[-1]==Cos[1]},y,{x,0}]

NDSolve::mxst:
   Maximum number of 1000 steps reached at the point x == -0.00413118.
```

Out[17]=

```
{{y -> InterpolatingFunction[{{-1., -0.00413118}}, <>]}}
```

In[18]:=

```
Plot[Evaluate[y[x]/. ries[[1]]],{x,-1,-0.0093},PlotPoints->500]
```



Out[18]=

```
-Graphics-
```

Ukážme si riešenie van der Polovho systému rovníc. Tento problém vyžaduje zvýšiť hodnotu parametra *MaxStep*

In[19]:=

```
NDSolve[{y'[x]==z[x],z'[x]==(1-y[x]^2) z[x] -y[x],y[0]==20, z[0]==0},
       {y[x],z[x]},{x,0,220}]

NDSolve::mxst:
   Maximum number of steps reached at the point 209.664.
```

Out[19]=

```
{{y[x] -> InterpolatingFunction[{{0., 209.664}}, <>][x],

  z[x] -> InterpolatingFunction[{{0., 209.664}}, <>][x]}}
```

In[20]:=

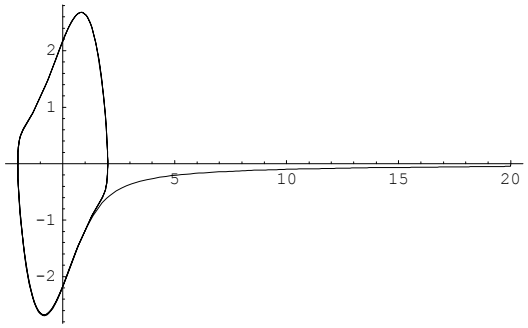
```
ries=NDSolve[
    {y'[x]==z[x],z'[x]==(1-y[x]^2)z[x]-y[x],y[0]==20,z[0]==0},
    {y[x],z[x]},{x,0,220}, MaxSteps->1000]
```

Out[21]=

```
{{y[x] -> InterpolatingFunction[{{0., 220.}}, <>][x],

  z[x] -> InterpolatingFunction[{{0., 220.}}, <>][x]}}
```

In[22]:=

```
ParametricPlot[Evaluate[{y[x],z[x]}/.ries[[1]]],{x,0,220},
PlotPoints->200,PlotRange->All]
```



Out[23]=

```
-Graphics-
```

Iným typom singularity je singularita derivácie vyskytujúcej sa v diferenciálnej rovnici. Program *MATHEMATICA* na takúto singularitu odpovedá zmenšením dĺžky kroku a extrapoláciou riešenia, až do toho momentu, kým sa riešenie opäť nestane dostatočne hladkým. Riešme dif. rovnicu, s nespojitou deriváciou v bode $x = 0$.

In[24]:=

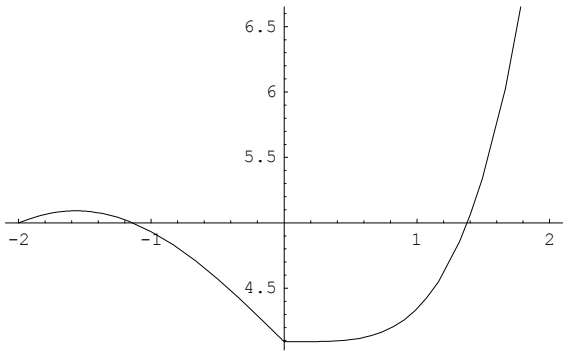
```
ries=NDSolve[{y'[x] ==If[x>0,x^3,-Cos[x]], y[-2]==5},y,{x,2}]
```

Out[24]=

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

In[25]:=

```
Plot[y[x]/.ries,{x,-2,2}]
```



Out[25]=

```
-Graphics-
```

Niektoré diferenciálne rovnice nespĺňajú podmienky vety o jednoznačnosti riešenia dif. rovnice, môžu mať viac riešení spĺňajúcich tú istú počiatočnú podmienku. Ak nastane táto situácia prog. systém *MATHEMATICA* vo verzii 3.0 a viac obvykle nájde všetky riešenia. Staršie verzie ešte nemali implementované nové algoritmy, ktoré vytvorili až v roku 1996 Wedenskyi a Keiper a preto obvykle našli len jedno riešenie diferenciálnej rovnice a opäť len v niektorých prípadoch upozornili, že riešení môže existovať viac. Náš príklad ukazuje dve, od seba sa vzďaľujúce riešenia dif. rovnice.

In[26]:=

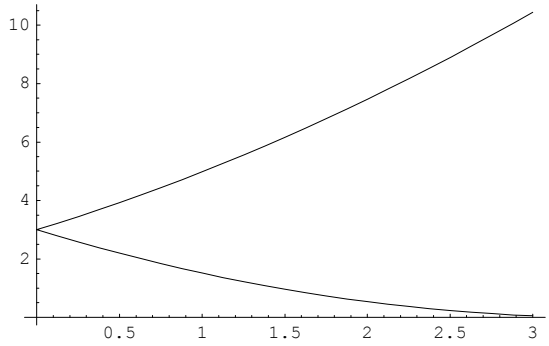
```
ries=NDSolve[{y'[x]^2 ==y[x], y[0]==3},y,{x,3}]
```

Out[26]=

```
{{y -> InterpolatingFunction[{{0., 3.}}, <>]},
  {y -> InterpolatingFunction[{{0., 3.}}, <>]}}
```

In[27]:=

```
Plot[{y[x]/.ries[[1]],y[x]/.ries[[2]]},{x,0,3}]
```



Out[27]=

```
-Graphics-
```

4.Riešenie diferenciálnych systémov

Možnosti systému *MATHEMATICA* pri riešení diferenciálnych systémov sú veľmi široké a tak pre nedostatok miesta uvádzame v tomto príspevku len dva príklady z pedagogickej praxe.

Riešme systém diferenciálnych rovníc s komplexnými koeficientami a skúmajme spolu reálnu a imaginárnu časť riešenia tohto systému

Čitateľ isto vie, že počiatočné úlohy môžeme rozdeliť na "stiff" a "non-stiff" úlohy. Poznamenajme len, že programový systém *MATHEMATICA* používa metódu Adams predictor - corrector pre non-stiff úlohy a spätnú diferenčnú formulu (Gear's method) pre stiff úlohy. Celý proces klasifikácie úlohy a výberu vhodnej metódy sa deje nezávisle od užívateľa. Ukážme si riešenie stiff systému diferenciálnych rovníc. Funkcia *y[x]* klesá veľmi rýchlo, preto vo výslednom grafe, keď volíme spoločný rozsah pre obe riešenia vidíme len funkciu *y[x]*.

In[28]:=

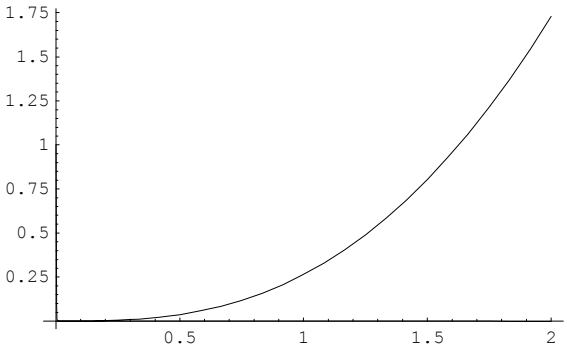
```
ries=NDSolve[ {y'[x] == -1000 y[x] + Cos[x], z'[x] == - z[x] + x^2,
y[0]==1, z[0]==0 },{y[x],z[x]},{x,2}]
```

Out[28]=

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

  z[x] -> InterpolatingFunction[{{0., 2.}}, <>][x]}}
```

In[29]:=

```
Plot[Evaluate[{y[x],z[x]}/.ries[[1]]],{x,0,2},PlotRange->All]
```



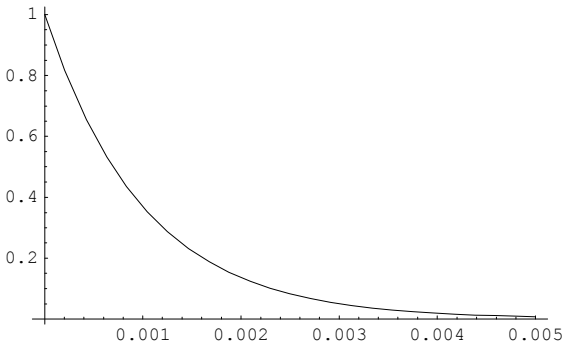
Out[29]=

```
-Graphics-
```

Pre úplnosť si nakreslime aj graf funkcie *y[x]*.

In[30]:=

```
Plot[Evaluate[y[x]/.ries[[1]]],{x,0,0.005},PlotRange->All]
```



Out[30]=

```
-Graphics-
```

Druhým príkladom je nelineárny diferenciálny systém. Vznikol ako čiastkový výsledok pri riešení strojníckeho problému (úlohy z praxe). Pre ďalší postup bolo dôležité vyriešiť ho a vhodným spôsobom výsledky vizualizovať. Nedokázali sme nájsť jeho explicitné, alebo implicitné riešenie . Preto sme sa rozhodli riešiť ho numerickými metódami.

Ukážeme si niektoré výsledky, ktoré sme pri riešení problému získali. Uvažujme nelineárny diferenciálny systém rovníc. Problém bol vo svojom pôvodnom zadaní ešte trochu zložitejší, tu uvádzame jeho zjednodušený tvar (bez straty základných čŕt riešenia).

$$x' = \mu.x + y - x.(x^2 + y^2)$$

$$y' = \mu.y - x - y.(x^2 + y^2)$$

Skúmajme situáciu pre μ = 2, 1, 1/2 a – 1/2 . Pre každú hodnotu μ nájdite periodické riešenie spĺňajúce počiatočné podmienky x(0) = 0 a y(0) = β

Pri riešení použijeme aj standard package PlotField a pre rôzne zvolené μ nakreslíme vektorové pole nášho systému na intervale [-2,2] x [-2,2]. Jednotlivé výsledky nebudeme zobrazovať samostatne.

In[31]:=

```
<<Graphics`PlotField`
```

In[32]:=

```
pole[mi_]:=PlotVectorField[
{mi x + y - x (x^2 + y^2), mi y - x - y (x^2 + y^2)}, {x,-2,2},{y,-
2,2}, ScaleFunction->(1&),Axes->True,AxesOrigin->{0,0},PlotPoints->20,
DisplayFunction->Identity];
```

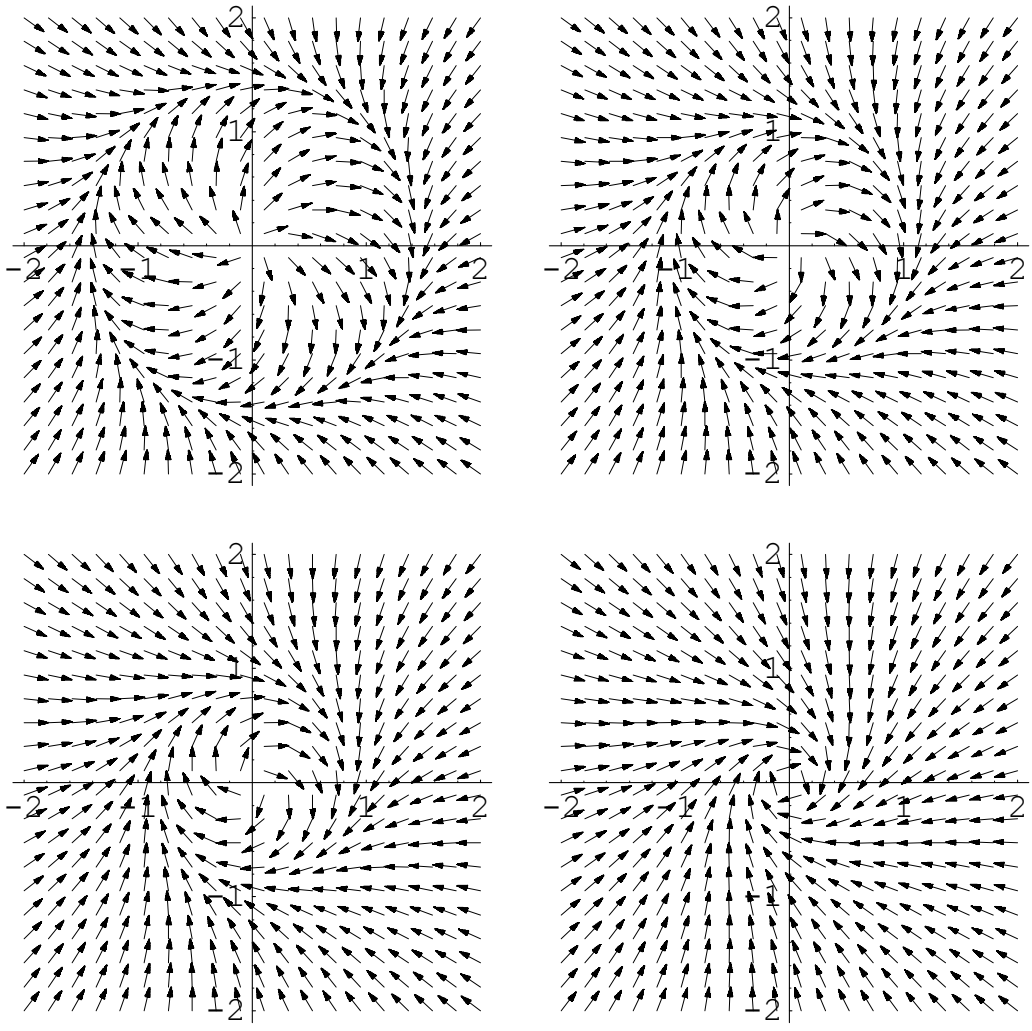
Označme jednotlivé vektorové polia a nakreslime ich

In[33]:=

```
f1= pole[2];f2= pole[1];f3=pole[1/2];f4=pole[-1/2];
```

In[37]:=

```
Show[GraphicsArray[{{f1,f2},{f3,f4}}],DisplayFunction-
>$DisplayFunction]
```



Out[37]=

```
-GraphicsArray-
```

Pomocou príkazu **NDSolve** nájdime numerickú aproximáciu riešenia počiatočnej úlohy pre uvedené $\mu = 2, 1, 1/2, -1/2$

In[38]:=

```
system={x'[t]==mi x[t] + y[t] - x [t] (x[t]^2 + y[t]^2), y'[t]==mi
y[t] - x[t] - y[t] (x[t]^2 + y[t]^2),x[0]==0, y[0]==1/2};
```

```
mi=2;
```

In[40]:=

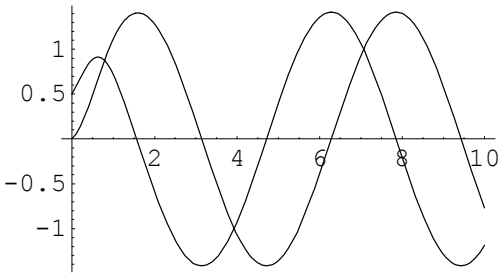
```
riesenie= NDSolve[system,{x[t],y[t]},{t,0,10}]
```

Out[40]=

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

In[41]:=

```
r1=Plot[Evaluate[{x[t],y[t]}/.riesenie],{t,0,10}]
```



Out[41]=

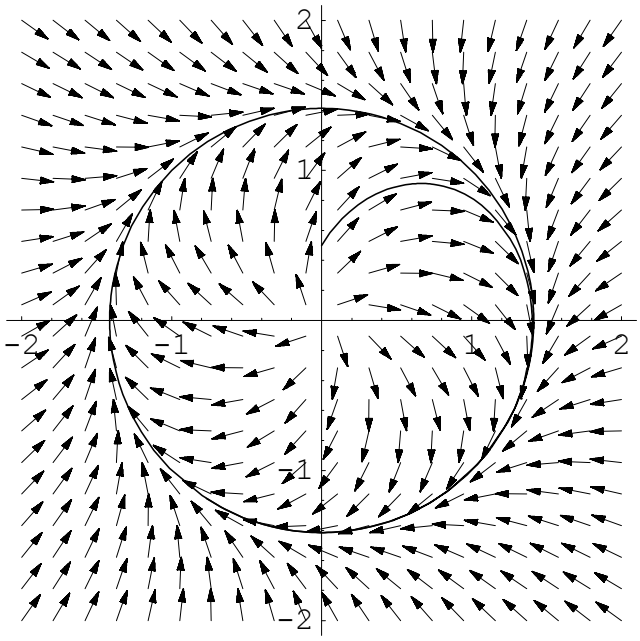
```
-Graphics-
```

In[42]:=

```
p1=ParametricPlot[{x[t],y[t]}/.riesenie,{t,0,10},
    Compiled->False,PlotRange->{{-2,2},{-2,2}},
    AspectRatio->1,DisplayFunction->Identity];
```

In[43]:=

```
Show[f1,p1, DisplayFunction->$DisplayFunction]
```



Out[43]=

```
-Graphics-
```

Pre ostatné konštanty μ postupujeme podobne

In[44]:=

```
Clear[mi]
```

In[45]:=

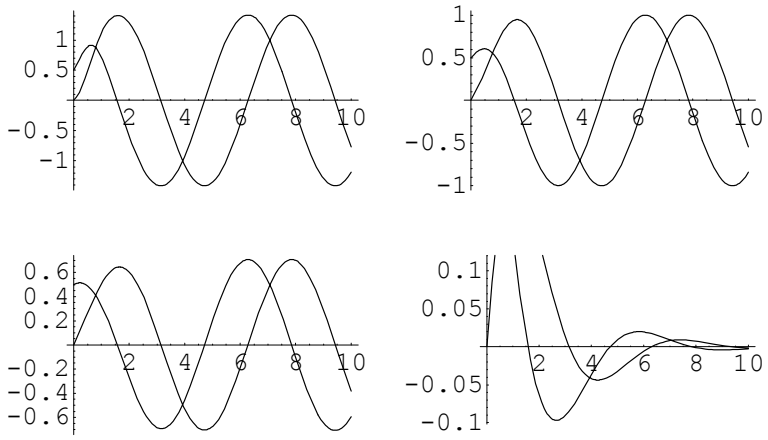
```
sys[mi_]:= NDSolve[{x'[t]==mi x[t] + y[t] - x [t] (x[t]^2 + y[t]^2),
y'[t]==mi y[t] - x[t] - y[t] (x[t]^2 + y[t]^2),x[0]==0,
y[0]==1/2},{x[t],y[t]},{t,0,10}]
```

In[46]:=

```
r2=Plot[Evaluate[{x[t],y[t]}/.sys[1]],{t,0,10},
     DisplayFunction->Identity];
r3=Plot[Evaluate[{x[t],y[t]}/.sys[1/2]],{t,0,10},
     DisplayFunction->Identity];
r4=Plot[Evaluate[{x[t],y[t]}/.sys[-1/2]],{t,0,10},
     DisplayFunction->Identity];
```

In[49]:=

```
Show[GraphicsArray[{{r1,r2},{r3,r4}}],
    DisplayFunction->$DisplayFunction]
```



Out[49]=

```
-GraphicsArray-
```

In[50]:=

```
sys[1]
p2=ParametricPlot[{x[t],y[t]}/. %,{t,0,10},Compiled->False,
PlotRange->{{-2,2},{-2,2}}, AspectRatio->1,DisplayFunction->Identity];
```

Out[50]=

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

In[52]:=

```
sys[1/2]
p3=ParametricPlot[{x[t],y[t]}/.%,{t,0,10}, Compiled->False,
PlotRange->{{-2,2},{-2,2}}, AspectRatio->1,DisplayFunction->Identity];
```

Out[52]=

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

In[54]:=

```
sys[-1/2]
p4=ParametricPlot[{x[t],y[t]}/. %,{t,0,10}, Compiled->False,
PlotRange->{{-2,2},{-2,2}}, AspectRatio->1,DisplayFunction->Identity];
```

Out[54]=

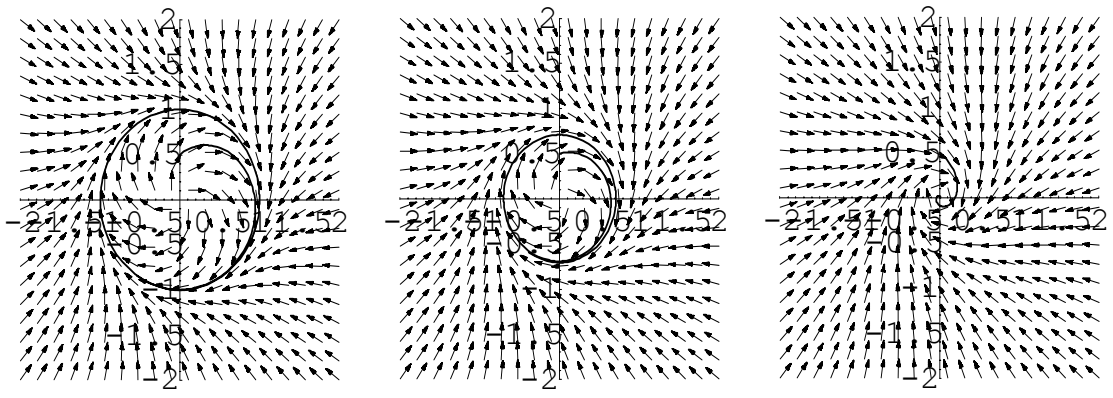
```
{{x[t] -> InterpolatingFunction[{{0., 10.}}, <>][t],
  y[t] -> InterpolatingFunction[{{0., 10.}}, <>][t]}}
```

In[56]:=

```
obr2= Show[p2,f2];
obr3= Show[p3,f3];
obr4= Show[p4,f4];
```

In[59]:=

```
Show[GraphicsArray[{obr2,obr3,obr4},
    DisplayFunction->$DisplayFunction,Ticks->None]]
```



Out[59]=

```
-GraphicsArray-
```

Čo sme vlastne zistili o správaní systému ?

Ak μ=2 , vidíme, že riešenie systému spĺňajúce počiatočné podmienky x(0) = 0 , y(0) = Sqrt[2] bude periodickým riešením. Podobne ak μ = 1/2 potom periodické bude riešenie pre x(0) = 0 , y(0) = 0,482. Na druhej strane ak μ = 1 , nepotrebujeme aproximovať riešenie . Z grafu vidíme, že riešenie spĺňajúce x(0) = 0 , y(0) = 1 bude periodické. A je veľmi jednoduché overiť, že týmto riešením je {x[t]=Sin[t], y[t]=Cos[t]}.

5. Okrajové úlohy

Pripomeňme len, že **NDSolve[]** nedokáže presne riešiť okrajové úlohy. Môžeme však použiť nasledovú formu substitúcie.
Definujme si funckiu, ktorá rieši počiatočnú úlohu, pričom jej argumentom je práve počiatočná podmienka. Variáciou tohto parametra sa potom snažíme nájsť taký stav, aby obe hraničné podmienky boli splnené. Na riešenie tejto úlohy je vhodná funkcia **FindRoot[]**

Riešme okrajovú úlohu *y´´ = - y , y(0)=1 a y(5)=2*. Funkcia *f[ ]* bude riešiť náš počiatočný problém

In[60]:=

```
f[yp_] :=First[ries=y/. NDSolve[{y[0]==1,y'[0]==yp,y''[t]==-y[t]},{y},
   {t,0,5}]] [5]
```

Riešenie spĺňajúce podmienku *y´(0) = - 1* má hodnotu *y(5)* príliš malú a nastavenie *y´(0)=-2* má zas hodnotu *y(5)* príliš veľkú.

In[61]:=

```
f[-1]
```

Out[61]=

```
1.24257
```

In[62]:=

```
f[-2]
```

Out[62]=

```
2.2015
```

Riešme rovnicu *y(5)=2* a riešenie potom nakreslime

In[63]:=

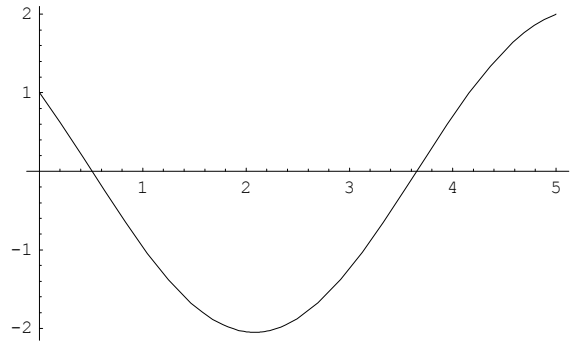
```
FindRoot[f[yp]==2,{yp,-1,2}]
```

Out[63]=

```
{yp -> -1.78987}
```

In[64]:=

```
Plot[ First[ries][t],{t,0,5},PlotRange->All]
```



Out[64]=

```
-Graphics-
```

Literatúra:

[1]Halada L.: *Stabilita úloh a algoritmov vo výučbe numerickej matematiky na SJF STU,* Proceedings of the scientific conference with international participation, INFORMATICS AND ALGHORITHMS ’98, Prešov 3. - 4. sept. 1998, pp.147-151

[2]Kolesárová A.: *Výučba Fourierových radov s podporou programového systému Mathematica,* ***MATHEMATICA* 99,** Bratislava 29.6.-2.7. 1999, pp.

[3]Kováčová M., Halada L. : *Experimentálna výučba numerickej matematiky pomocou programového systému Mathematica na Sjf STU,* Matematická štatistika a Numerická matematika, Kálnica 1. -5. júna 1998, pp.142-150

[4]Kováčová M.: *N Function of MATHEMATICA and Some Notices to Numerical Quadrature,* Matematická štatistika a Numerická matematika, Kočovce 14. - 18. 6. 1999, pp.

[5] Kováčová M.: *Možnosti nového prístupu ku výučbe dif. rovníc na technických univerzitách,* Proceedings of the scientific conference with international participation, INFORMATICS AND ALGORITHMS ’98, Prešov 3. - 4. sept. 1998, pp.287-291

[6]Omachelová M.: *Zisťovanie priebehu funkcie jednej reálnej premennej s podporou pg. systému MATHEMATICA ,* ***MATHEMATICA* 99**, Bratislava 29.6.-2.7. 1999, pp.

[7]Omachelová M.: Skúsenosti s výučbou extrémov funkcie viac premenných s využitím programového systému *MATHEMATICA,* INFORMATICS AND ALGHORITHMS '99, Prešov 9. - 10. sept. 1999, pp.
[8]Záhonová V.: *Lineárne diferenciálne rovnice n-tého rádu s konštantnými koeficientami a program. systém MATHEMATICA ,* ***MATHEMATICA* 99,** Bratislava 29.6.-2.7. 1999, pp.