# Informce ke cvičení z Programování pro fyziky

Plánoval jsem nejdříve ukázat, jak funguje Eulerova metoda na rovnici $\dot{y} = -y$, tedy řešení konstruovat z lomených čar, kde jejich směrnice se počítá jednoduše v bodě, kde se čára láme.

```
program Priklad1;

const dt=0.001;
var   n : integer;
t,y,dydt: real;

begin
  n =0;

  t = 0;  // jako obvykle nezapomenout na inicializaci, tady ma navic
  y = 1;  // vyznam pocatecnich podminek

  while (t<20) do begin
    if n mod 100 = 0 then writeln(t,' ',y);
    dydt:= -y;
    y    := y+dydt*dt;
    t    := t+dt;
    n    := n+1;
  end;

end.
```

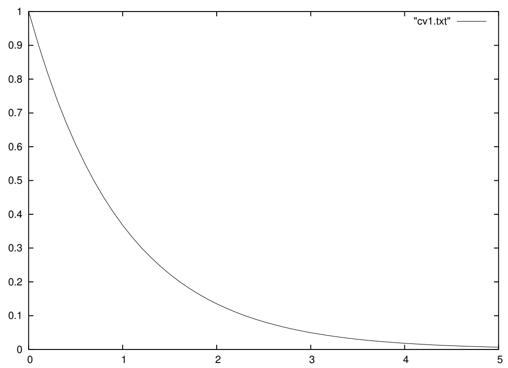


Figure 1: Příklad 1. Numerické řešení rovnice $\dot{y} = -y$.

Totéž si pak lze během krátké chvíle vyzloušet i pro harmonický oscilátor, kde se používá jakási zpackaná Eulerova metoda, kdy se nové polohy počítají již ze změněné rychlosti. Je to trik a funguje dobře u periodických dějů, kde se vyruší nejvyšší chyba. Tato metoda je ale především úvod do toho, co to znamenají Newtonovy pohybové rovnice v počítači, konkrétně pro rovnici $\ddot{y} + y = 0$.

```
program Priklad2;

const dt=0.001;
var   n : integer;
    t,y : real;
   dydt : real;
  d2ydt2: real;

begin
  n :=0;

  t := 0;  // jako obvykle nezapomenout na inicializaci, tady ma navic
  y := 1;  // vyznam pocatecnich podminek
dydt:= 0;  // ted uz je to pocactecni podminka

  while (t<20) do begin
    if n mod 100 = 0 then writeln(t,' ',y);
  d2ydt2:= -y;
    dydt:= dydt+d2ydt2*dt;
    y    := y+dydt*dt;
    t    := t+dt;
    n    := n+1;
  end;

end.
```

Je jasné, že změnou pohybové rovncice na $\ddot{y} + \sin y = 0$ dostáváme neharmonické kmity. Program lze sputit s různými hodnotami počátečního úhlu a koukat, co se stane. Oproti 2. příkladu stačí tedy změnit jen dva řádky:

```
  y := 3.1;  // Kyvadlo temer v horni uvrati.
....
   d2ydt2 := -sin(y);
```

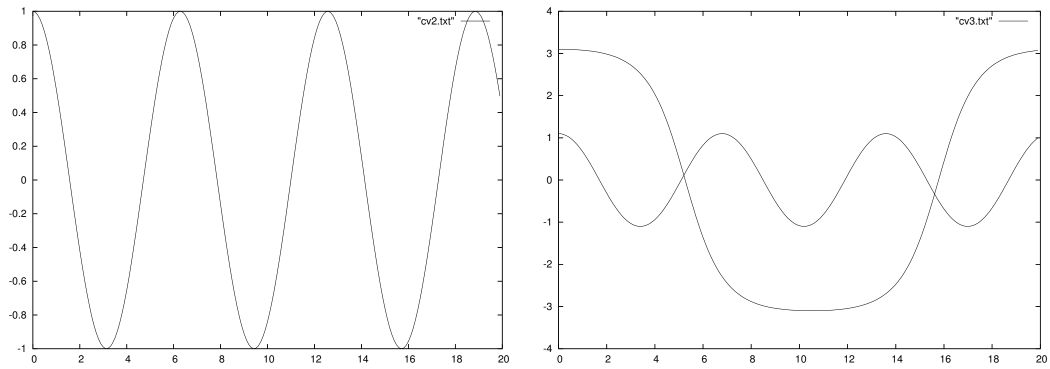


Figure 2: Příklady 2 a 3, tedy rovnice $\ddot{y} + y = 0$ a $\ddot{y} + \sin y = 0$. Pro neharmonický oscilátor stojí za to zkusit různé amplitudy.

V následujícím příkladě č. 4 se demonstruje, jak numerickou metodu přestěhovat do dedikované procedury. Zároveň se ukazuje, že ODE 2. řádu jsou dvě ODE prvního řádu. Soustava rovnic vyžaduje zavést typ pole. Jako ukázku, k čemu je dobré mít výčtové typy, zavademe typ `tIndex = ( iUhel,iUhlovaRychlost)`. Dále jako přípravu na další budeme meze cyklů určovat pomocí `low(),high()`.

```
program Priklad4;

type tIndex = (iUhel, iUhlovaRychlost);
     tPole = array[tIndex] of real;

procedure EulerStep( const dUdt: tPole ; var U : tPole; var  t : real; const dt : real);
var     i : tIndex;
begin
 for i:=Low(U) to High(U) do U[i] := U[i] + dUdt[i]*dt;
 t := t+dt;
end;

const dt=0.001;

var  t : real = 0;          // pocatecni cas
     V : tPole = (3.1,0);   // pocatecni podminky jako inicializovana promenna
  dVdt: tPole;
     n : integer = 0;

begin

  while (t<20) do begin
    if n mod 100 = 0 then writeln(t,' ',V[iUhel]);
    dVdt[iUhel]             :=  V[iUhlovaRychlost];
    dVdt[iUhlovaRychlost] := -sin(V[iUhel]);
    EulerStep(dVdT,V,t,dt);
    n    := n+1;
  end;

end.
```

Výše uvedená metoda počítá polohu ze staré rychlosti a tak nedojde ke zázračnému vyrušení chyby a kyvadlo se pro $dt = 0.001$ přehoupne, kam nemá. Proto lze vyskoušet s menším $dt$.

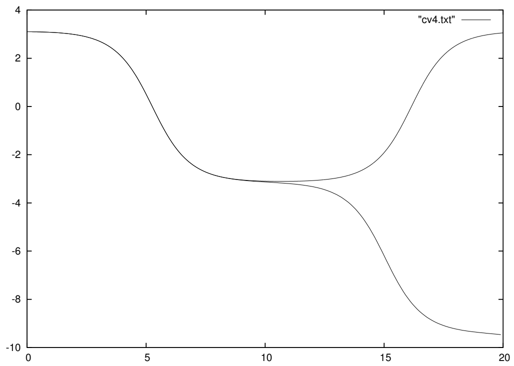


Figure 3: Příklady 4. Opět rovnice $\ddot{y} + \sin y = 0$. Chyba správné Eulerovy metody způsobí, že je třeba menší $dt$ aby se kyvadlo nepřehouplo.

Takto si ale procedura EulerStep nemůže požádat o výpočet pravé stranu ODE, kdy se jí zachce a musí se spoléhat na to, co dostane. Proto v dalším příkladě zavedeme parametr typu funkce.

```
type tProc = procedure( var dUdt : tPole; const U : tPole);
```

Je dobré zdůraznit, že aktuální parametr musí mít stejnou hlavičku jaká je uvedena v deklaraci formálního parametry typu odkaz na funkci/proceduru, jen názvy parametrů se mohou lišit. Tak dostáváme Příklad č.5:

```
program Priklad5;

type tIndex = (iUhel, iUhlovaRychlost);
     tPole = array[tIndex] of real;
     tProc = procedure( var dUdt : tPole; const U : tPole);

procedure EulerStep( Spocti_dUdt : tProc ; var U : tPole; var  t : real; const dt : real);
var    i : tIndex;
    dUdt : tPole;
begin
 Spocti_dUdt(dUdt,U);
 for i:=Low(U) to High(U) do U[i] := U[i] + dUdt[i]*dt;
 t := t+dt;
end;

procedure PohyboveRovniceKyvadla( var dVdt : tPole; const V : tPole);
begin
    dVdt[iUhel]            :=  V[iUhlovaRychlost];
    dVdt[iUhlovaRychlost] := -sin(V[iUhel]);
end;

const dt=0.001;

var  t : real = 0;         // pocatecni cas
     V : tPole = (3.1,0); // pocatecni podminky jako inicializovana promenna
     n : integer = 0;

begin

  while (t<20) do begin
    if n mod 100 = 0 then writeln(t,' ',V[iUhel]);
    EulerStep(PohyboveRovniceKyvadla,V,t,dt);
    n    := n+1;
  end;

end.
```

Kromě přestěhování rovnic do procedury PohyboveRovniceKyvadla nenastala žádná změna.

Nyní se nabízí zkusit klíčový problém pro Newtonovu pohybovou rovnici - dráhy pod vlivem gravitace

$$\ddot{\vec{x}} = -\kappa \frac{\vec{x}}{|\vec{x}|^3} \ .$$

V jednotkách času a délky rok resp. AU je $\kappa = 4\pi^2$ a oběžná rychlost Země je asi $2\pi$. Potřebujeme 4 dimenzionální pole pro $x, y, v_x$ a $v_y$, indexy tentokrát definujeme prostřednictvým výčtu.

```
program Priklad6;

type tIndex = (ix, iy, ivx, ivy);
     tPole = array[tIndex] of real;
     tProc = procedure( var dUdt : tPole; const U : tPole);

procedure EulerStep( Spocti_dUdt : tProc ; var U : tPole; var  t : real; const dt : real);
var    i : tIndex;
    dUdt : tPole;
begin
 Spocti_dUdt(dUdt,U);
 for i:=Low(U) to High(U) do U[i] := U[i] + dUdt[i]*dt;
 t := t+dt;
end;

procedure PohyboveRovnicePlanety( var dVdt : tPole; const V : tPole);
const G = (4*Pi*Pi); // G v jednotkach AU, rok
var r,r3 : real;
begin
    r := sqrt(V[ix]*V[ix]+V[iy]*V[iy]);
    r3:= r*r*r;

    dVdt[ix] := V[ivx];
    dVdt[iy] := V[ivy];

    dVdt[ivx]:= -G*V[ix]/r3;
    dVdt[ivy]:= -G*V[iy]/r3;
end;

const dt=1/365.25;

var  t : real = 0;              // pocatecni cas
     V : tPole = (1,0,  0,2*Pi); // pocatecni rychlost Zeme je 2Pi AU/rok
```

```
    n : integer = 0;

begin

  while (t<1.2) do begin // 1.2 roku
    if n mod 10 = 0 then writeln(t,'␣',V[ix],'␣',V[iy]); // data pro trajektorii
    EulerStep(PohyboveRovnicePlanety,V,t,dt);
    n    := n+1;
  end;

end.
```

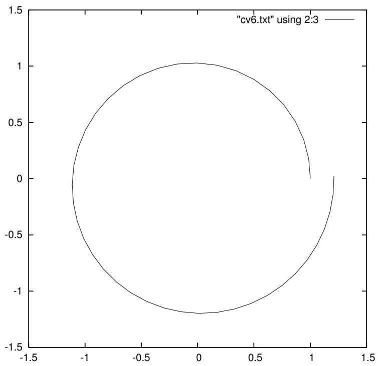


Figure 4: Příklad 6. Trajektorie pohybu Země, kdyby se řídila podle Eulera a vždy jeden den běžela rovně a pak skokem změnila rychlost. (Stojí za to srovnat s obrázky z Newtonových Principií.) Změna kroku z jdenoho dne na 1 hodinu samozřejmě pomůže.

Na závěr lze pak vyložit nejjednodušší Rungeovu-Kuttovu metodu – tzv. midpoint:

```
procedure MidpointStep( Spocti_dUdt : tProc ; var U : tPole; var  t : real; const dt : real);
var    i : tIndex;
    dUdt : tPole;
    Upul : tPole;
begin
 Spocti_dUdt(dUdt,U);
 for i:=Low(U) to High(U) do Upul[i] := U[i] + dUdt[i]*dt*0.5;

 Spocti_dUdt(dUdt,Upul);
 for i:=Low(U) to High(U) do U[i] := U[i] + dUdt[i]*dt;
 t := t+dt;
end;
```

S její pomocí je jeden den použitelný časový krok. Pokud se nastaví počáteční rychlost jiná, než kruhová dostáváme stáčenjící se eliptické dráhy. Zmenšení kroku vede samozřejmě k potlačení stáčení. (Pozn. Kdyby pravá strana ODE závisela na $t$, bylo by potřeba dát $t$ jako parametr GetdUdt a také posunout $t$ o $dt/2$ před jejím druhým zavoláním.)

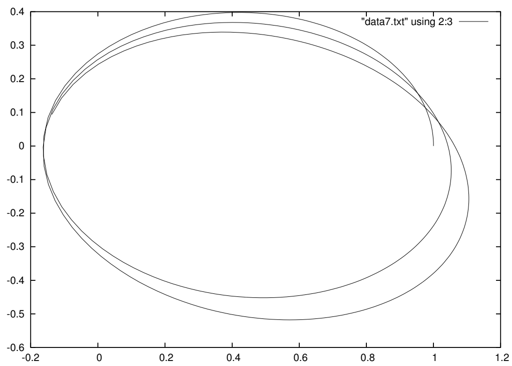


Figure 5: Příklad 7. Numerická trajektorie pohybu Země při počáteční rychlosti $2\pi - 3$ a kroku 1 den.