Informce ke cvičení z Programování pro fyziky

Přikládám zdrojáky, jak mi chodí tady na Linuxu, v MSVC je potřeba vytvořit nový konzolový projekt a zvolit Hello-world sample. Pak nejspíš ještě je třeba přidat include hlavičky math.h.

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.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

using namespace std;


//priklad 1. integrace dif. rovnice y' = - y, y(0) = 1

int main (int argc, char * argv[])
{

  const double dt=0.001;
  int n =0;

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

  while (t<20) {
    if (n++ % 100 == 0) printf( "%g\t%g\n", t,y);
        // zvlaste druhe %g ve fromatu je mnohem lepsi nez %f (at si to vyzkouseji)
    dydt = -y;
    y   += dydt*dt;
    t   += dt;
  }

  return 0;
}
```

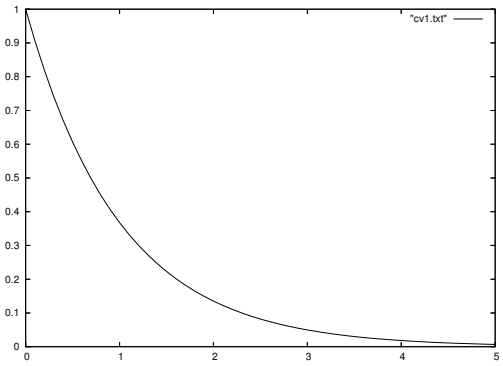

Figure 1: Příklad 1. Numerické řešení rovnice $\dot{y} = -y$.

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 $\ddot{y} + y = 0$.

```
//priklad 2. integrace dif. rovnice y'' = - y, y(0) = 1

int main (int argc, char * argv[])
{

  const double dt=0.001;
  int n =0;

  double  t = 0;     // jako obvykle nezapomenout na inicializaci, tady ma navic
  double  y = 1;     // vyznam pocatecnich podminek pro y
  double  dydt = 0; // a zde je pocactecni podminka pro y'
  double  d2ydt2;

  while (t<20) {     // pár period
    if (n++ % 100 == 0) printf( "%g\t%g\n", t,y);

    d2ydt2 = -y;
    dydt  += d2ydt2*dt;
    y     += dydt*dt;
    t     += dt;
  }
```

```
  return 0;
}
```

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

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

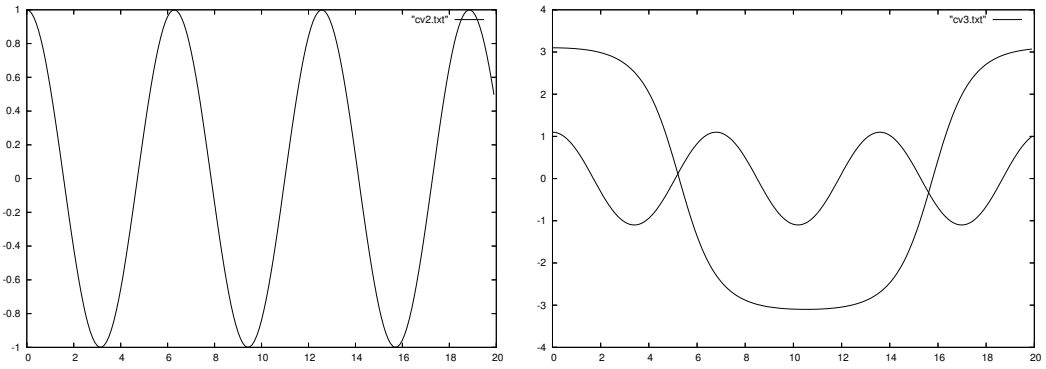

Figure 2: Příklady 2 a 3, tedy rovnice $\ddot{y} + y = 0$ a $\ddot{y} + \sin y = 0$. Pro neharmonický oscilátor stojí za to zkusit různé amplitudy.

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. Indexy si pojmenuji iUhel a iUhlovaRychlost. Na cvičení nepoužíváme pointery, ale jen odkazy, a tak kód vypadá následovně:

```
const int Dim = 2;
typedef double tVektor[Dim];

void EulerStep( tVektor dUdt, tVektor &U, double &t, double dt)
{
  int i;

  for (i=0; i<Dim; i++) U[i] += dUdt[i]*dt;
  t += dt;
}

int main (int argc, char * argv[])
{

  const double dt=0.001;
  int n =0;
  const int iUhel = 0;
  const int iUhlovaRychlost = 1;

  double  t = 0;              // pocatecni cas
  tVektor dVdt,V = {3.1,0}; // pocatecni poloha

  while (t<20) {
    if (n++ % 100 == 0) printf( "%g\t%g\t%g,\n", t,V[iUhel],V[iUhlovaRychlost]);
    dVdt[iUhel]           = V[iUhlovaRychlost];
    dVdt[iUhlovaRychlost] = -sin(V[iUhel]);
    EulerStep(dVdt, V, t, dt);
  }

  return 0;
}
```

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.

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. To se ve standardním C dělá přes pointer na funkce, ale opět, kvůli nebezpečí a kvůli srovnání s Pascalem použijeme parametr odkaz na proceduru, který má tvar

```
void EulerStep( void (&GetdUdt)( tVektor& , tVektor)  , tVektor &U, double &t, double dt)
```

Závorky okolo (&GetdUdt) jsou nezbytné kvůli prioritě, bez nich by to byl pokus o referenci na void, což je nesmysl. Navíc je dobré zdůraznit, že aktuální parametr musí mít stejnou hlavičku jaká je uvedena v deklaraci formálního parametry typu odkaz na funkci/proceduru. Nyní dostáváme Příklad č.5:

```
const int Dim = 2;
typedef double tVektor[Dim];
```

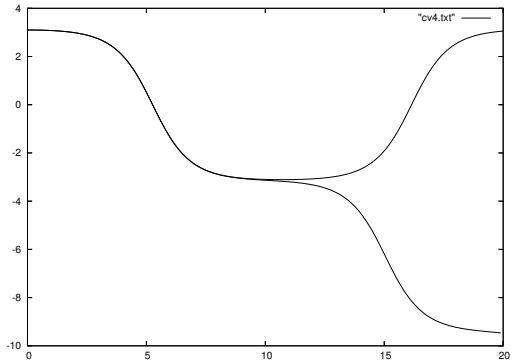

Figure 3: Příklady 4. Opět rovnice $\ddot{y} + \sin y = 0$. Chyba správné Eulerovy metody způsobí, že je třeba menší dt aby se kyvadlo nepřehouplo.

```
const int iUhel = 0;
const int iUhlovaRychlost = 1;

void PohyboveRovnice( tVektor &dVdt, tVektor V) {
    dVdt[iUhel]           = V[iUhlovaRychlost];
    dVdt[iUhlovaRychlost] = -sin(V[iUhel]);
}

void EulerStep( void (&GetdUdt)( tVektor& , tVektor)  , tVektor &U, double &t, double dt)
{
  int i;

  tVektor dUdt;
  GetdUdt( dUdt, U);

  for (i=0; i<Dim; i++) U[i] += dUdt[i]*dt;
  t += dt;
}

int main (int argc, char * argv[])
{

  const double dt=0.001;
  int n =0;

  double  t = 0;              // pocatecni cas
  tVektor dVdt,V = {3.1,0}; // pocatecni poloha

  while (t<20) {
    if (n++ % 100 == 0) printf( "%g\t%g\t%g,\n", t,V[iUhel],V[iUhlovaRychlost]);
    EulerStep(PohyboveRovnice, V, t, dt);
  }

  return 0;
}
```

Kromě přestěhování rovnic do procedury PohyboveRovnice a nutného přestěhování deklarací konstant iUhel a iUhlovaRychlost 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 2π. Potřebujeme 4 dimenzionální pole pro x, y, v_x a v_y, indexy tentokrát definujeme prostřednictvým výčtu.

```
const double Pi = 3.14159265358979324;

const int Dim = 4;
typedef double tVektor[Dim];

typedef enum {ix,iy,ivx,ivy} tDim;

void PohyboveRovnice( tVektor &dVdt, tVektor V) {
    double r = sqrt(V[ix]*V[ix]+V[iy]*V[iy]);
    double r3= r*r*r;
    const double G = (4*Pi*Pi); // G v jednotkach AU,rok

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

```
    dVdt[ivx] = -G*V[ix]/r3;
    dVdt[ivy] = -G*V[iy]/r3;
}

void EulerStep( void (&GetdUdt)( tVektor& , tVektor)  , tVektor &U, double &t, double dt)
{
  int i;

  tVektor dUdt;
  GetdUdt( dUdt, U);

  for (i=0; i<Dim; i++) U[i] += dUdt[i]*dt;
  t += dt;
}

int main (int argc, char * argv[])
{

  const double dt=1.0/356; // 1 den bezi Zeme rovne
  int n =0;

  double  t = 0;              // pocatecni cas
  tVektor dVdt,V = {1,0, 0,2*Pi}; // pocatecni poloha a rychlost

  while (t<1.2) {
    if (n++ % 10 == 0) printf( "%g\t%g\t%g,\n", t,V[ix],V[iy]);
    EulerStep(PohyboveRovnice, V, t, dt);
  }

  return 0;
}
```

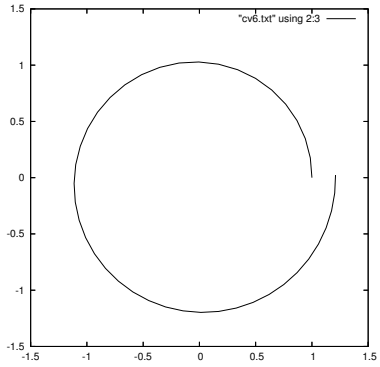

Figure 4: Příklady 6. Trajektorie pohybu Země, kdyby se řídila podle Eulera a vždy jeden den běžela rovně a pak skokem změnila rychlost. (Stojí za to srovnat s obrázky z Newtonových Principií.) Změna kroku z jdenoho dne na 1 hodinu samozřejmě pomůže.

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

```
void MidpointStep( void (&GetdUdt)( tVektor& , tVektor)  , tVektor &U, double &t, double dt)
{
  int i;

  tVektor dUdt,Upul;
  GetdUdt( dUdt, U);
  for (i=0; i<Dim; i++) Upul[i] = U[i] + 0.5*dUdt[i]*dt;

  GetdUdt( dUdt, Upul);
  for (i=0; i<Dim; i++) U[i] += dUdt[i]*dt;
  t += dt;
}
```

S její pomocí je jeden den použitelný časový krok. Pokud se nastaví počáteční rychlost jiná, než kruhová dostáváme stáčející se eliptické dráhy. Opět zmenšení kroku vede 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.)