Implementace řešičů ODR

Riccatiho rovnice

Experimentální řád konvergence

Harmonický oscilátor

Adaptivní volba časového kroku

Soupeřící druhy

Lorenzovy rovnice

Epidemiologie - SIR model

Úloha n-těles

Tomáš Oberhuber

Faculty of Nuclear Sciences and Physical Engineering
Czech Technical University in Prague

Numerická matematika

Zdrojové kódy s příklady jsou na:
https://gitlab.com/oberhuber.tomas/fjfi-num-src.
Stahovat lze pomocí:

```
git clone https://gitlab.com/oberhuber.tomas/fjfi-num-src.git
```

nebo tlačítkem *Download*.
Rozbalíme příkazem

```
tar xvf fjfi-num-src*

cd fjfi-num-src*
```

Řešení ODR

- řešíme úlohu typu:

$$\frac{d\vec{u}}{dt} = \vec{f}(t, \vec{u}(t)) \text{ pro } t \in (0, T\rangle), \quad (1)$$

$$\vec{u}(0) = \vec{u}_{ini}, \quad (2)$$

  kde $\vec{u} : \langle 0, T\rangle \to \mathbb{R}^n$, $\vec{f} : \langle 0, T\rangle \times \mathbb{R}^n \to \mathbb{R}^n$ a $\vec{u}_{ini} \in \mathbb{R}^n$ je okrajová (počáteční) podmínka.
- n se často také označuje jako **DOF**
  - **degrees of freedom** = stupně volnosti
  - znamená to, kolik parametrů použijeme k aproximaci řešení úlohy v daném čase t
- tuto úlohu lze řešit některou z Rungových-Kuttových metod
- pro jednoduchost nejprve zvolíme Eulerovu variantu

$$\vec{k}_1 = \tau \vec{f}(t, \vec{u}(t)) \quad (3)$$

$$\vec{u}(t+\tau) = \vec{u}(t) + \vec{k}_1 \quad (4)$$

Řešení ODR

Algoritmicky lze řešič naší úlohy zapsat takto:

procedure SOLVEODE($\vec{u}_{ini}$, $\vec{f}$, t_{stop}, τ_0)
  $t := 0$
  $\vec{u} := \vec{u}_{ini}$
  while $t < t_{stop}$ **do**
    $\tau := min\{\tau_0, t_{stop} - t\}$
    $\vec{u} := \vec{u} +$ update($\vec{u}$, $\vec{f}$, τ)
    $t := t + \tau$
  end while
end procedure

Algoritmus 1: Algoritmus pro řešení ODR pomocí Rungových-Kuttových metod.

Řešení ODR

procedure UPDATE($\vec{u}$, $\vec{f}$, τ)

$\vec{k}_1 := \tau \vec{f}(t, \vec{u})$

return $\vec{k}_1$

end procedure

Algoritmus 2: Příklad implementace nejjednodušší Rungovy-Kuttovy metody.

Řešení ODR

Naším cílem bude odvodit kód, ve kterém bude snadné měnit:

- úlohu, kterou řešíme
  - $\vec{f}$, $\vec{u}_{ini}$
- použitou Rungovu-Kuttovu metodu
  - v našem případě ji reprezentuje funkce `update`

Popis úlohy

Ve zdrojových kódech je každá úloha odvozována z objektu `ODEProblem`:

```
struct ODEProblem
{
      virtual int getDegreesOfFreedom();

      virtual void getRightHandSide( const double t,
                                     const double* u,
                                     double* fu );

      virtual bool writeSolution( const double t,
                                  const int step,
                                  const double* u );
};
```

Kód lze nalézt v souboru `ode/ODEProblem.h`

Všechny metody jsou virtuální, protože popisují rozhraní obecné úlohy, podobně, jako tomu bylo v případě objektu `struct Function` popisujícím obecnou funkci v předchozí prezentaci.

Popis úlohy

Význam jednotlivých metod:

- `int` **`getDegreesOfFreedom`**`()`
  - vrací počet stupňů volnosti dané úlohy, tj. n pro $\vec{u} \in \langle 0, T \rangle \times \mathbb{R}^n$
- `void` **`getRightHandSide`**`( t, u, fu )`
  - napočítává pravou stranu rovnice (2)
  - vektor $\vec{u}$ a výsledek $\vec{f}(t, \vec{u})$ jsou reprezentovány formou ukazatele na pole
  - velikost těchto polí je vždy rovna počtu DOF, tj. n
- `bool` **`writeSolution`**`( t, step, u)`
  - v předem určených časových bodech provádí uložení stavu řešení u do souboru
    - t odpovídá času simulace
    - *step* odpovídá celkovému počtu již uložených stavů

Popis Ruungovy-Kuttovy metody

Ve zdrojových kódech je každá Rungova-Kuttova metoda odvozována z objektu `ODESolver`:

```
struct ODESolver
{
      virtual bool setup( const int degreesOfFreedom );

      virtual bool solve( const double integrationTimeStep,
                          const double stopTime,
                          double* time,
                          ODEProblem* problem,
                          double* u );
};
```

Kód lze nalézt v souboru `ode/ODESolver.h`
Metody jsou opět označeny jako virtuální, protože jde o popis obecného objektu.

Popis Rungovy-Kuttovy metody

Význam jednotlivých metod:

- `int` **`setup`**`( degreesOfFreedom)`
  - tato metoda slouží pro alokaci pomocných polí, zejména pro vektory $\vec{k}_1, \vec{k}_2 \ldots$
  - jejich velikost opět odpovídá počtu DOFů
- `void` **`solve`**`( ... )`
  - napočítává řešení `u` úlohy `problem` od času daného hodnotou proměnné `time` s časovým krokem τ_0 rovnému `integrationTimeStep` až do času t_{stop} rovnému `stopTime`

Řešení ODR

Funkce pro řešení ODR pak může vypadat takto:

```
1  bool solve( const double initialTime,
2              const double finalTime,
3              const double timeStep,
4              const double integrationTimeStep,
5              ODEProblem* problem,
6              ODESolver* solver,
7              double* u )
8  {
9     solver->setup( problem->getDegreesOfFreedom() );
10    const int timeStepsCount =
11       ceil( max( 0.0, finalTime - initialTime ) / timeStep );
12    double time( initialTime );
13    for( int k = 1; k <= timeStepsCount; k++ )
14    {
15       double progress = (double) k/(double) timeStepsCount;
16       printf( "Solving time step %d / %d => %f %\n",
17          k, timeStepsCount, 100.0*progress );
18       if( ! solver->solve( integrationTimeStep,
19                            time + timeStep,   // stopTime
20                            &time,
21                            problem,
22                            u ) )
23          return false;
24       time += timeStep;
25       problem->writeSolution( time, k, u );
26    }
27    printf( "Done. \n");
28    return true;
29 }
```

Kód lze najít v souboru `ode/ode-solve.h`

Řešení ODR

Všimněme si, že se v kódu objevují dva různé časové kroky:

- `timeStep` - udává, v jakých intervalech se ukládá stav řešení ulohy $\vec{u}(t)$
- `integrationTimeStep` - udává integrační časový krok, který odpovídá parametru τ_0 v algoritmu 1

Implementace Rungovy-Kuttovy metody prvního řádu

Implementace Eulerova ODE řešiče vypadá takto:

Listing 1: "ode/Euler.h"

```
1  class Euler : public ODESolver
2  {
3     public:
4
5        // Constructor
6        Euler();
7
8        bool setup( const int degreesOfFreedom );
9
10       bool solve( const double integrationTimeStep,
11                   const double stopTime,
12                   double* time,
13                   ODEProblem* problem,
14                   double* u );
15
16       // Destructor
17       ~Euler();
18
19    protected:
20
21       double* k1;
22 };
```

Implementace Rungovy-Kuttovy metody prvního řádu

- objekt `Euler` je odvozen od `ODESolver`
- konstruktor `Euler()` se volá automaticky při vytvoření instance objektu
  - v našem případě se v něm nuluje ukazatel `k1`
- destruktor `~Euler()` se volá automaticky při zániku instance objektu
  - v našem případě má na starost dealokaci dynamicky alokovaného pole `k1`

Implementace Rungovy-Kuttovy metody prvního řádu

Listing 2: "ode/Euler.cpp"

```
1  Euler::Euler()
2  {
3     k1 = 0;
4  }
5
6  bool Euler::setup( const int degreesOfFreedom )
7  {
8     k1 = new double[ degreesOfFreedom ];
9     if( ! k1 )
10       return false;
11    return true;
12 }
13
14 Euler::~Euler()
15 {
16    if( k ) delete[] k;
17 }
```

- konstruktor `Euler()` nuluje ukazatel `k1`
- metoda `setup` alokuje do ukazatele `k1` pole o velikosti dané počtem DOFů
- destruktor `~Euler()` dealokuje alokovanou paměť

Implementace Rungovy-Kuttovy metody prvního řádu

Listing 3: "ode/Euler.cpp"

```
bool Euler::solve( const double integrationTimeStep,
                   const double stopTime,
                   double* time,
                   ODEProblem* problem,
                   double* u )
{
   const int dofs = problem->getDegreesOfFreedom();
   while( *time < stopTime )
   {
      // compute current tau
      double tau = min( integrationTimeStep, stopTime - *time );

      // compute k1
      problem->getRightHandSide( *time, u, k1 );

      // update to the next time step
      for( int i = 0; i < dofs; i++ )
         u[ i ] += tau * k1[ i ];
      *time += tau;
   }
   return true;
}
```

Implementace řešičů ODR
Riccatiho rovnice
Experimentální řád konvergence
Harmonický oscilátor
Adaptivní volba časového kroku
Soupeřící druhy
Lorenzovy rovnice
Epidemiologie - SIR model
Úloha n-těles

Implementace Rungovy-Kuttovy metody prvního řádu

Domácí úkol

- do souborů `ode/RungeKutta.h` a `ode/RungeKutta.cpp` dopište kód pro Rungovu-Kuttovu metodu druhého řádu
- zdrojové kódy vytiskněte a odevzdejte

Implementace řešičů ODR
Riccatiho rovnice
Experimentální řád konvergence
Harmonický oscilátor
Adaptivní volba časového kroku
Soupeřící druhy
Lorenzovy rovnice
Epidemiologie - SIR model
Úloha n-těles

Riccatiho rovnice

Jde o rovnici tvaru:

$$\frac{du(t)}{dt} = a_0(t) + a_1(t)u(t) + a_2(t)u^2(t).$$

Konkrétněji např.

$$\frac{du(t)}{dt} = t^{-4}e^t + u(t) + 2e^{-t}u^2(t). \tag{5}$$

Řešení této rovnice je:

$$u(t) = e^t\left[\frac{1}{\sqrt{2}t^2}\tan\sqrt{2}\left(c - \frac{1}{t}\right) - \frac{1}{2t}\right] \tag{6}$$

pro $c \in \mathbb{R}$.

Tomáš
Oberhuber

Implementace řešičů ODR

Riccatiho rovnice

Experimentální řád konvergence

Harmonický oscilátor

Adaptivní volba časového kroku

Soupeřící druhy

Lorenzovy rovnice

Epidemiologie - SIR model

Úloha n-těles

Riccatiho rovnice

Tato úloha je implementována v souborech `ode/RiccatiProblem.h`, `RiccatiProblem.cpp` a `ricatti.cpp`

Riccatiho rovnice

Listing 4: "ode/RicattiProblem.h"

```
class RiccatiProblem : public ODEProblem
{
   public:

      RiccatiProblem();

      int getDegreesOfFreedom();

      void getRightHandSide( const double& t,
                             const double* _u,
                             double* fu );

      double getExactSolution( const double& t ,
                               const double& c = 1.0 );

      bool writeExactSolution( const char*  fileName,
                               const double& initialTime,
                               const double& finalTime,
                               const double& timeStep,
                               const double& c = 1.0 );

      bool writeSolution( const double& t,
                          int step,
                          const double* u );

      double getL1Error( const double timeStep );

      double getL2Error( const double timeStep );

      double getMaxError();

      double l1Error, l2Error, maxError;
};
```

Riccatiho rovnice

Listing 5: "ode/RicattiProblem.cpp"

```
1  int RiccatiProblem::getDegreesOfFreedom()
2  {
3     return 1;
4  }
```

- metoda `getDegreesOfFreedom()` vrací počet stupnů volnosti úlohy
- v našem případě je $u : \langle 0, T \rangle \rightarrow \mathbb{R}^1$, tj. DOF $= 1$

Riccatiho rovnice

Listing 6: "ode/RicattiProblem.cpp"

```
1  void RiccatiProblem::getRightHandSide( const double& t,
2                                          const double* _u,
3                                          double* fu )
4  {
5     // create alias so that we can write u insted of _u[0]
6     const double& u = _u[ 0 ];
7     fu[ 0 ] = pow( t, -4.0 ) * exp( t ) + u +
8               2.0 * exp( -t ) * u * u;
9  }
```

- metoda `getRightHandSide()` napočítává pravou stranu vztahu (5), tj. výraz

$$t^{-4}e^{t} + u(t) + 2e^{-t}u^{2}(t)$$

Riccatiho rovnice

Listing 7: "ode/RicattiProblem.cpp"

```
1 double RiccatiProblem::getExactSolution( const double& t ,
2                                          const double& c )
3 {
4    const double sqrt_2 = sqrt( 2.0 );
5    return exp( t ) * ( 1.0 / ( sqrt_2 * t * t ) *
6                   tan( sqrt_2 * ( c - 1.0 / t ) ) -
7                   1.0 / ( 2.0 * t ) );
8 }
```

- metoda `getExactSolution()` přesné řešení (6), tj. výraz

$$u(t) = e^t \left[\frac{1}{\sqrt{2}t^2} \tan \sqrt{2} \left(c - \frac{1}{t} \right) - \frac{1}{2t} \right]$$

- budeme ho využívat k napočítaní chyby našich numerických metod

Riccatiho rovnice

Listing 8: "ode/RicattiProblem.cpp"

```
1  bool RiccatiProblem::writeExactSolution(
2      const char* fileName,
3      const double& initialTime,
4      const double& finalTime,
5      const double& timeStep,
6      const double& c )
7  {
8     std::fstream file;
9     file.open( fileName, std::ios::out );
10    double t = initialTime;
11    while( t < finalTime )
12    {
13       file << t << " " << this->getExactSolution( t, c )
14            << std::endl;
15       t = std::min( t + timeStep, finalTime );
16    }
17 }
```

- metoda `getExactSolution()` vypisuje přesné řešení na intervalu $\langle initalTime, finalTime \rangle$ s krokem (rozlišením) `timeStep`
- výsledný graf lze zobrazit pomocí programu `gnuplot`

Riccatiho rovnice

Formát dat pro program `gnuplot` při vykreslování funkce $u : \mathbb{R} \rightarrow \mathbb{R}$ je:

```
t0 u(t0)
t1 u(t1)

...

tn u(tn)
```

tj. hodnota proměnné t + mezera + příslušná funkční hodnota + konec řádku.

Riccatiho rovnice

Listing 9: "ode/RicattiProblem.cpp"

```
bool RiccatiProblem::writeSolution( const double& t,
                                    int step,
                                    const double* u )
{
   const double& u = _u[ 0 ];
   fstream file;
   if( step == 0 )
   {
      /****
       * In the first step, we want to rewrite the file
       */
      file.open( "riccati.txt", ios::out );
      if( !file ) return false;
   }
   else
   {
      /****
       * In later steps, we just append new time steps
       */
      file.open( "riccati.txt", ios::out | ios::app );
      if( !file ) return false;
   }
   file << t << " " << u << endl;

   /****
    * Evaluate errors of the approximation
    */
   const double diff = fabs( getExactSolution(t) - u );
   l1Error += diff;
   l2Error += diff * diff;
   maxError = std::max( maxError, diff );
}
```

Riccatiho rovnice

- při vypisování prvního časového kroku (`step == 0`) chceme výstupní soubor přepsat
- proto ho otevíréme v módu `ios::out`
- při dalších časových krocích již chceme do tohoto souboru jen připisovat
- proto ho otevíráme v módu `ios::out || ios::app`

Implementace řešičů ODR

Riccatiho rovnice

Experimentální řád konvergence

Harmonický oscilátor

Adaptivní volba časového kroku

Soupeřící druhy

Lorenzovy rovnice

Epidemiologie - SIR model

Úloha n-těles

Riccatiho rovnice

```
double RiccatiProblem::getL1Error( const double timeStep )
{
   return timeStep * l1Error;
}

double RiccatiProblem::getL2Error( const double timeStep )
{
   return sqrt( timeStep * l2Error );
}

double RiccatiProblem::getMaxError()
{
   return maxError;
}
```

Zbývající metody napočítávají celkové normy chyb.

Riccatiho rovnice

Chyby aproximace

Jsou definovány jako:

$$\|e\|_{L_1\langle 0,T\rangle} := \int_0^T |e(t)|\,dt \approx \sum_{i=0}^{N} |e(i\Delta t)|\,\Delta t,$$

$$\|e\|_{L_2\langle 0,T\rangle} := \left(\int_0^T |e(t)|^2\,dt\right)^{\frac{1}{2}} \approx \left(\sum_{i=0}^{N} |e(i\Delta t)|^2\,\Delta t\right)^2,$$

$$\|e\|_{L_\infty\langle 0,T\rangle} := \sup_{t\in\langle 0,T\rangle} |e(t)| \approx \max_{i\in 0\dots N} |e(i\Delta t)|,$$

kde $\Delta t := \frac{T}{N}$ a

$$e(t) := u(t) - u_\tau(t),$$

pro u označující přesné řešení a u_τ přibližné řešení.

Riccatiho rovnice

Samotný řešič Riccatiho rovnice pak vypadá takto:

```
#include <cstdlib>
#include "RiccatiProblem.h"
#include "Euler.h"
#include "ode-solve.h"

using namespace std;

const double initialTime( 0.0 );
const double finalTime( 0.15 );
const double timeStep( 1.0e-4 );
const double integrationTimeStep( 1.0e-4 );

int main( int argc, char** argv )
{
    RiccatiProblem problem;
    Euler integrator;

    /***
     * Set initial condition
     */
    double u  = problem.getExactSolution( initialTime );

    if( ! solve( initialTime,
                 finalTime,
                 timeStep,
                 integrationTimeStep,
                 &problem,
                 &integrator,
                 &u ) )
       return EXIT_FAILURE;

    cout << "L1 error: " << problem.getL1Error( timeStep ) << endl
         << "L2 error: " << problem.getL2Error( timeStep ) << endl
         << "Max error: " << problem.getMaxError() << endl;

    return EXIT_SUCCESS;
}
```

Implementace řešičů ODR
Riccatiho rovnice
Experimentální řád konvergence
Harmonický oscilátor
Adaptivní volba časového kroku
Soupeřící druhy
Lorenzovy rovnice
Epidemiologie - SIR model
Úloha n-těles

Riccatiho rovnice

Parametry na řádcích 8 - 11 mají tento význam:

- úloha je řešena na interval $\langle \mathit{initialTime}, \mathit{finalTime} \rangle$
- stav se ukládá v časech $\mathit{initialTime} + i \cdot \mathit{timeStep}$ pro $i = 0, 1, \ldots$
- příslušná Rungova-Kuttova metoda aproximuje řešení úlohy s krokem *integrationTimeStep*

- na řádcích 15 a 16 vytváříme instance úlohy a příslušného řešiče.
- na řádku 21 nastavujeme počáteční/okrajovou podmínku.
- na řádcích 23 - 30 voláme řešič úlohy
- na řádcích 32 - 34 vypisujeme normy chyb aproximace

Riccatiho rovnice

V prostředí linuxu pak postupujeme následujícím způsobem:

```
1 cd fjfi-num-src*
2
3 make install
4
5 ~/.local/bin/riccati
6
7 gnuplot
8
9 plot 'riccati.txt'
```

Riccatiho rovnice

Domácí úkol

- zvolte vhodný interval, kde je možné napočítat numericky řešení Ricattiho rovnice
- napočítejte *experimentální řád konvergence - EOC* pro Eulerův a vámi implementovaný Rungův-Kuttův řešič
- napište report k numerické analýze
  - řešená rovnice
  - interval, na kterém ji řešíme
  - počáteční podmínka
  - ~~parametry úlohy~~
  - zvolená numerická metoda
  - parametry numerické metody - integrační časový krok,...
  - tabulka EOC

EOC

Definition 1

Buď $\tau_1 > \tau_2$, u přesné řešení úlohy, u_{τ_1} a u_{τ_2} přibližná řešení získaná integrací s časovými kroky τ_1 a τ_2. Nechť $e_{\tau_1}(t) := u(t) - u_{\tau_1}(t)$ a $e_{\tau_2}(t) := u(t) - u_{\tau_2}(t)$. Dále $E_{\tau_1} := \|e_{\tau_1}\|_{L_*\langle 0,T\rangle}$, $E_{\tau_2} := \|e_{\tau_2}\|_{L_*\langle 0,T\rangle}$ ve vhodné normě. Pak definujeme

$$EOC(\tau_1, \tau_2) := \frac{\log(E_{\tau_1}/E_{\tau_2})}{\log(\tau_1/\tau_2)}.$$

EOC

Remark 2

Je-li $\tau_1/\tau_2 = 2$, *tj* $\tau_2 = \frac{1}{2}\tau_1$, *pak*

$$EOC(\tau_1, \tau_2) := \frac{\log(E_{\tau_1}/E_{\tau_2})}{\log(\tau_1/\tau_2)} = \log_2(E_{\tau_1}/E_{\tau_2}).$$

Pokud $E_{\tau_1}/E_{\tau_2} = 2 \Rightarrow EOC(\tau_1, \tau_2) = 1$.
Pokud $E_{\tau_1}/E_{\tau_2} = 4 \Rightarrow EOC(\tau_1, \tau_2) = 2$.
Pokud $E_{\tau_1}/E_{\tau_2} = 8 \Rightarrow EOC(\tau_1, \tau_2) = 3$.
Pokud $E_{\tau_1}/E_{\tau_2} = 16 \Rightarrow EOC(\tau_1, \tau_2) = 4$.

EOC

Example 3

Příklad EOC tabulky

N	τ	$L_1(0,T)$		$L_2(0,T)$		$L_\infty(0,T)$	
		Err.	EOC	Err.	EOC	Err.	EOC
17	0.5	2.1		4		19	
33	0.25	0.46	**2.2**	0.64	**2.6**	2.9	**2.7**
65	0.125	0.15	**1.6**	0.34	**0.92**	1.9	**0.59**
129	0.0625	0.06	**1.3**	0.16	**1.1**	1.1	**0.74**
257	0.03125	0.025	**1.2**	0.08	**0.99**	0.72	**0.66**
513	0.015625	0.012	**1.05**	0.041	**0.96**	0.42	**0.77**

Implementace řešičů ODR
Riccatiho rovnice
Experimentální řád konvergence
Harmonický oscilátor
Adaptivní volba časového kroku
Soupeřící druhy
Lorenzovy rovnice
Epidemiologie - SIR model
Úloha n-těles

Harmonický oscilátor

Řešíme úlohu:

$$\vec{F} = -k\vec{x}.$$

Z fyziky víme, že

$$\vec{F} = m\vec{a} = m\ddot{\vec{x}},$$

tj.

$$\ddot{\vec{x}} + \frac{k}{m}\vec{x} = 0.$$

My budeme řešit úlohu tzv. tlumeného oscilátoru v $\mathbb{R}^1$

$$\ddot{u}(t) + \epsilon u^2(t)\dot{u}(t) + u(t) = 0.$$

Implementace řešičů ODR
Riccatiho rovnice
Experimentální řád konvergence
Harmonický oscilátor
Adaptivní volba časového kroku
Soupeřící druhy
Lorenzovy rovnice
Epidemiologie - SIR model
Úloha n-těles

Harmonický oscilátor

Tuto rovnici druhého řádu převedeme na soustavu dvou rovnic prvního řádu:

$$\dot{u}_1 := u_2, \tag{7}$$

$$\dot{u}_2 := -u_1 - \epsilon u_1^2 u_2, \tag{8}$$

s počáteční podmínkou

$$\vec{u}\mid_{t=0} = \left(u_{1,ini}, u_{2,ini}\right)^T.$$

Harmonický oscilátor

Řešič je implementován v souborech `ode/HyperbolicProblem.h`, `ode/HyperbolicProblem.cpp` a `ode/hyperbolic.cpp`.

Listing 10: "ode/HyperbolicProblem.cpp"

```
 1  int HyperbolicProblem::getDegreesOfFreedom()
 2  {
 3     return 2;
 4  }
 5
 6  void HyperbolicProblem::getRightHandSide( const double& t,
 7                                            const double* _u,
 8                                            double* fu )
 9  {
10     const double& u1 = _u[ 0 ];
11     const double& u2 = _u[ 1 ];
12     fu[ 0 ] = u2;
13     fu[ 1 ] = -u1 - epsilon * u1 * u1 * u2;
14  }
```

Harmonický oscilátor

Listing 11: "ode/hyperbolic.cpp"

```
1  #include "HyperbolicProblem.h"
2  #include "Euler.h"
3  #include "ode-solve.h"
4
5  const double initialTime( 0.0 );
6  const double finalTime( 100.0 );
7  const double timeStep( 1.0e-1 );
8  const double integrationTimeStep( 1.0 );
9
10 int main( int argc, char** argv )
11 {
12     HyperbolicProblem problem;
13     problem.setEpsilon( 0.0 );
14     Euler integrator;
15
16     double u[ 2 ] = { 0.0, 1.0 };
17
18     if( ! solve( initialTime,
19                  finalTime,
20                  timeStep,
21                  integrationTimeStep,
22                  &problem,
23                  &integrator,
24                  u ) )
25        return EXIT_FAILURE;
26     return EXIT_SUCCESS;
27 }
```

Implementace řešičů ODR
Riccatiho rovnice
Experimentální řád konvergence
Harmonický oscilátor
Adaptivní volba časového kroku
Soupeřící druhy
Lorenzovy rovnice
Epidemiologie - SIR model
Úloha n-těles

Harmonický oscilátor

Example 4

Zmenšujte intergační krok pro simulaci harmonického oscilátoru.

Mersonova-Rungova-Kuttova metoda

procedure MERSONRUNGEKUTTA($\vec{u}$, $\vec{f}$, t_{stop}, τ_0)

$t := 0$

$\vec{u} := \vec{u}_{ini}$

while $t \leq t_{stop}$ **do**

$\tau := min\{\tau_0, t_{stop} - t\}$

$\vec{k}_1 := \tau \vec{f}(t, \vec{u})$

$\vec{k}_2 := \tau \vec{f}\left(t + \frac{1}{3}\tau, \vec{u} + \frac{1}{3}\vec{k}_1\right)$

$\vec{k}_3 := \tau \vec{f}\left(t + \frac{1}{3}\tau, \vec{u} + \frac{1}{6}\vec{k}_1 + \frac{1}{6}\vec{k}_2\right)$

$\vec{k}_4 := \tau \vec{f}\left(t + \frac{1}{2}\tau, \vec{u} + \frac{1}{8}\vec{k}_1 + \frac{3}{8}\vec{k}_3\right)$

$\vec{k}_5 := \tau \vec{f}\left(t + \tau, \vec{u} + \frac{1}{2}\vec{k}_1 - \frac{3}{2}\vec{k}_3 + 2\vec{k}_4\right)$

$e := \frac{1}{3}\left\|\frac{1}{5}\vec{k}_1 - \frac{9}{10}\vec{k}_3 + \frac{4}{5}\vec{k}_4 - \frac{1}{10}\vec{k}_5\right\|_{\ell_\infty}$

if $e < \epsilon$ **then**

$\vec{u} := \vec{u} + \frac{1}{6}\left(\vec{k}_1 + 4\vec{k}_4 + \vec{k}_5\right)$

$t := t + \tau$

end if

$\tau := \min\left\{\tau \cdot \frac{4}{5}\left(\frac{\epsilon}{e}\right)^{\frac{1}{5}}, t_{stop} - t\right\}.$

end while

end procedure

Tomáš Oberhuber

Implementace řešičů ODR

Riccatiho rovnice

Experimentální řád konvergence

Harmonický oscilátor

Adaptivní volba časového kroku

Soupeřící druhy

Lorenzovy rovnice

Epidemiologie - SIR model

Úloha n-těles

Mersonova-Rungova-Kuttova metoda

Tato metoda je implementována v souborech `ode/Merson.h` a `ode/Merson.cpp`.

- parametr ϵ se nastavuje pomocí metody `setAdaptivity( eps )`
- tento parametr se nastavuje na hodnotu $10^{-3} - 10^{-12}$

Example 5

Proveďte simulaci harmonického oscilátoru s pomocí Mersonovy metody.

Implementace řešičů ODR
Riccatiho rovnice
Experimentální řád konvergence
Harmonický oscilátor
Adaptivní volba časového kroku
Soupeřící druhy
Lorenzovy rovnice
Epidemiologie - SIR model
Úloha n-těles

Soupeřící druhy

Jde o úlohu typu:

$$
\begin{aligned}
\dot{u}_1 &= u_1 - a u_1^2 - c u_1 u_2, \\
\dot{u}_2 &= u_2 - b u_2^2 + d u_1 u_2,
\end{aligned}
$$

kde a, b, c a d jsou konstanty.

- jde o typ úloh zvané Volterrovy-Lotkovy úlohy
- simuluje soupeření dvou živočišných druhů - predátor a oběť
- $u : \langle 0, T \rangle \rightarrow \mathbb{R}^2$, tj. DOFs = 2
- úloha je implementována v souborech `SpeciesProblem.h`, `SpeciesProblem.cpp` a `species.cpp`

Soupeřící druhy

Domácí úkol

Proveďte numerickou analýzu, tj. pokuste se najít zajímavou kombinaci parametrů a, b, c a d a napište opět report doplněný obrázky.

- řešená rovnice
- interval, na kterém ji řešíme
- počáteční podmínka
- parametry úlohy
- zvolená numerická metoda
- parametry numerické metody
- obrázky s výsledky

Lorenzovy rovnice

Řešíme úlohu:

$$\begin{aligned}
\dot{u}_1 &= \sigma(u_2 - u_1), \\
\dot{u}_2 &= \rho u_1 - u_2 - u_1 u_3, \\
\dot{u}_3 &= -\beta u_3 + u_1 u_2,
\end{aligned}$$

kde $\sigma, \rho, \beta > 0$ jsou konstanty.

- jde o jednoduchý model používaný v meteorologii
- úloha dala vzniknout teorii determinstického chaosu
- řešením může být tzv. Lorenzův atraktor

Implementace řešičů ODR
Riccatiho rovnice
Experimentální řád konvergence
Harmonický oscilátor
Adaptivní volba časového kroku
Soupeřící druhy
Lorenzovy rovnice
Epidemiologie - SIR model
Úloha n-těles

Lorenzovy rovnice

Domácí úkol

Proveďte numerickou studii Lorenzových rovnic a pokuste se najít Lorenzův atraktor.

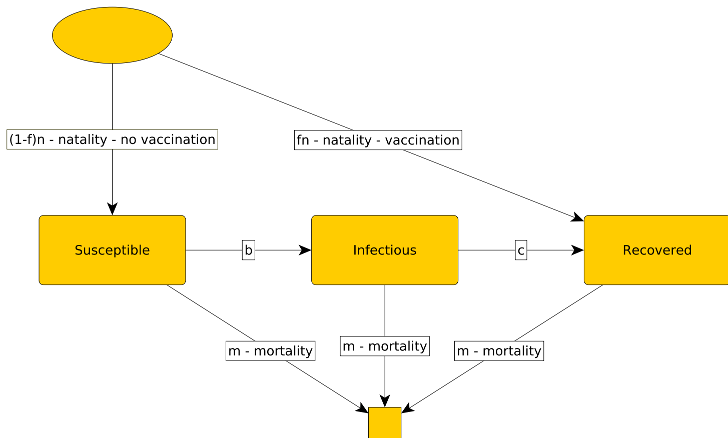
Implementace řešičů ODR
Riccatiho rovnice
Experimentální řád konvergence
Harmonický oscilátor
Adaptivní volba časového kroku
Soupeřící druhy
Lorenzovy rovnice
Epidemiologie - SIR model
Úloha n-těles

Epidemiologie - SIR model

Tento modelo popisuje šíření infekčních nemocí.

- **S**usceptible - skupina lidí, kteří mohou být nakaženi
- **I**nfectious - skupina infikovaných lidí, kteří mohou šířit nákazu
- **R**ecovered - skupina imuních lidí díky očkování nebo překonání infekce, kteří nemohou být infikováni a nemohou šířit nákazu

Epidemiologie - SIR model



Epidemiologie - SIR model

$$
\begin{aligned}
\frac{dS(t)}{dt} &= n(1-f)N - \frac{bI(t)S(t)}{N} - mS(t) \\
\frac{dI(t)}{dt} &= \frac{bI(t)S(t)}{N} - cI(t) - mI(t) \\
\frac{dR(t)}{dt} &= cI(t) + nfN - mR(t),
\end{aligned}
$$

kde

- N je velikost populace,
- n je míra porodnosti,
- m je míra úmrtnosti, $m \approx n$,
- b je riziko nakažení neimuních jedinců,
- c je míra uzdravení se nakažených jedinců,
- f je míra pročkování populace
- $\frac{bI(t)S(t)}{N}$ udává počet jedinců nakažených za jednotku času.

Implementace řešičů ODR
Riccatiho rovnice
Experimentální řád konvergence
Harmonický oscilátor
Adaptivní volba časového kroku
Soupeřící druhy
Lorenzovy rovnice
Epidemiologie - SIR model
Úloha n-těles

Epidemiologie - SIR model

Domácí úkol:

- implementujte tento model
- proveďte výpočetní studii, tj. různá nastevní parametrů
- (na internetu najděte parametry pro reálné epidemie a porovnejte s výpočtem)

Úloha n-těles

Řešíme úlohu:

$$m_i \frac{d^2 \vec{p}_i}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{G m_i m_j}{|\vec{p}_j - \vec{p}_i||} \left(\vec{p}_j - \vec{p}_i \right), i = 1, \ldots n$$

tj.

$$\frac{d^2 \vec{p}_i}{dt} = \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{G m_j}{|\vec{p}_j - \vec{p}_i||} \left(\vec{p}_j - \vec{p}_i \right), i = 1, \ldots n$$

tj.

$$\begin{aligned} \frac{d \vec{v}_i}{dt} &= \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{G m_j}{|\vec{p}_j - \vec{p}_i||} \left(\vec{p}_j - \vec{p}_i \right), i = 1, \ldots n, \\ \frac{d \vec{p}_i}{dt} &= \vec{v}_i, i = 1, \ldots n. \end{aligned}$$

Úloha n-těles

- každá částice je popsána vektorem pozice $\vec{p}_i$ a rychlosti $\vec{v}_i$
- tj. $2d$ DOFů, kde d je dimenze prostoru
- celkem dostáváme $2dn$ DOFů
- $$\vec{u} = \left(\underbrace{u_1, \ldots u_{dn}}_{\vec{v}_1,\ldots,\vec{v}_n}, \underbrace{u_{dn+1}, \ldots, u_{2dn}}_{\vec{p}_1,\ldots,\vec{p}_n} \right)^T$$
- počáteční podmínka vlastně nastavuje pozici a rychlost každé částice na začátku
- úloha je implementovaná v souborech `NBodyProblem.h`, `NBodyProblem.cpp` a `nbody.cpp`

Implementace řešičů ODR
Riccatiho rovnice
Experimentální řád konvergence
Harmonický oscilátor
Adaptivní volba časového kroku
Soupeřící druhy
Lorenzovy rovnice
Epidemiologie - SIR model
Úloha n-těles

Úloha n-těles

```
1  void NBodyProblem::setInitialCondition( double* u )
2  {
3     const int n = this->particlesCount;
4     for( int i = 0; i < this->particlesCount; i++ )
5     {
6        /****
7         * Initial velocity
8         */
9        u[ 2 * i ] = 0;
10       u[ 2 * i + 1 ] = 0;
11
12       /****
13        * Initial position
14        */
15       u[ n + 2 * i ] =
16          ( double ) ( rand() % 10000 ) / 100.0 - 50.0;
17       u[ n + 2 * i + 1 ] =
18          ( double ) ( rand() % 10000 ) / 100.0 - 50.0;
19
20       /****
21        * Particle mass
22        */
23       this->masses[ i ] = ( double ) ( rand() % 100 ) /
24          100.0 + 1.0;
25    }
26 }
```

Implementace řešičů ODR
Riccatiho rovnice
Experimentální řád konvergence
Harmonický oscilátor
Adaptivní volba časového kroku
Soupeřící druhy
Lorenzovy rovnice
Epidemiologie - SIR model
Úloha n-těles

Úloha n-těles

```cpp
void NBodyProblem::getRightHandSide( const double& t,
                                     const double* _u,
                                     double* fu )
{
   const double epsilon = 1.0e-1;
   const int n = this->particlesCount;
   for( int i = 0; i < n; i++ )
   {
      /****
       * Positions
       */
      fu[ 2 * ( n + i ) ] = _u[ 2 * i ];
      fu[ 2 * ( n + i ) + 1 ] = _u[ 2 * i + 1 ];

      /****
       * Velocities
       */
      fu[ 2 * i ] = 0.0;
      fu[ 2 * i + 1 ] = 0.0;
      const double& q_i_x = _u[ 2 * ( n + i ) ];
      const double& q_i_y = _u[ 2 * ( n + i ) + 1 ];
      for( int j = 0; j < n; j++ )
      {
         if( i == j )
            continue;
         const double& q_j_x = _u[ 2 * ( n + j ) ];
         const double& q_j_y = _u[ 2 * ( n + j ) + 1 ];
         const double q_ij_x = q_i_x - q_j_x;
         const double q_ij_y = q_i_y - q_j_y;
         double dist =
                sqrt( q_ij_x * q_ij_x + q_ij_y * q_ij_y + epsilon );
         double coeff =
                this->g * this->masses[ j ] / ( dist * dist * dist );
         fu[ 2 * i ] += coeff * ( q_j_x - q_i_x );
         fu[ 2 * i + 1 ] += coeff * ( q_j_y - q_i_y );
      }
   }
}
```

Implementace řešičů ODR
Riccatiho rovnice
Experimentální řád konvergence
Harmonický oscilátor
Adaptivní volba časového kroku
Soupeřící druhy
Lorenzovy rovnice
Epidemiologie - SIR model
Úloha n-těles

Úloha n-těles

```
bool NBodyProblem::writeSolution( const double& t,
                                  int step,
                                  const double* u )
{
   /****
    * Filename with step index
    */
   std::stringstream str;
   str << "nbody-" << std::setw( 5 )
       << std::setfill( '0' ) << step << ".txt";

   /****
    * Open file
    */
   std::fstream file;
   file.open( str.str(), std::ios::out );
   if( ! file )
   {
      std::cerr << "Unable to open the file "
                << str.str() << std::endl;
      return false;
   }

   /****
    * Write particles positions
    */
   const int n = this->particlesCount;
   const int d = this->dimension;
   for( int i = 0; i < n; i++ )
   {
      for( int j = 0; j < d; j++ )
         file << u[ d * ( n + i ) + j ] << " ";
      for( int j = 0; j < d; j++ )
         file << u[ d * ( n + i ) + j ] + u[ d * i + j ]
              << " ";
      file << std::endl;
   }
}
```

Implementace řešičů ODR
Riccatiho rovnice
Experimentální řád konvergence
Harmonický oscilátor
Adaptivní volba časového kroku
Soupeřící druhy
Lorenzovy rovnice
Epidemiologie - SIR model
Úloha n-těles

Úloha n-těles

```
1  #include <cstdlib>
2  #include "NBodyProblem.h"
3  #include "Euler.h"
4  #include "Merson.h"
5  #include "ode-solve.h"
6
7  using namespace std;
8
9  const double initialTime( 0.0 );
10 const double finalTime( 1.0e3 );
11 const double timeStep( 4.0e-2 );
12 const double integrationTimeStep( 1.0e-6 );
13 const int particlesCount( 100 );
14
15 int main( int argc, char** argv )
16 {
17    NBodyProblem problem( particlesCount );
18
19    Merson integrator;
20    integrator.setAdaptivity( 1.0e-8 );
21
22    double* u = new double[ problem.getDegreesOfFreedom() ];
23    problem.setInitialCondition( u );
24
25    if( ! solve( initialTime,
26                 finalTime,
27                 timeStep,
28                 integrationTimeStep,
29                 &problem,
30                 &integrator,
31                 u ) )
32    {
33       delete[] u;
34       return EXIT_FAILURE;
35    }
36    delete[] u;
37    return EXIT_SUCCESS;
38 }
```