1 Úvod

Tato práce implementuje knihovnu pro řešení spojitého simulátoru [IMS, slide 208] pomocí diferenciálních rovnic. Rozhraní a použití této knihovny je inspirováno simulační knihovnou SIMLIB[1]. Jsou v ní zahrnuty bloky pro intergrátory[IMS, slide 214] a bloky[IMS, slide 214] pro aritmetické operace. Pro integrátory se využívájí tři numerické integrační metody[IMS, slide 220] a to Eulerova metoda[IMS, slide 224], Runge-Kutta 4. řádu[IMS, slide 228] a metoda Adams-Bashforh[IMS, slide 232], ze kterých si uživatel může vybrat kterou metodu zvolí. Kontrola rychlých smyček[IMS, slide 244] není součástí této práce. Simulace probíhá na základě uživatelem definovaných rovnic. Uživatel si taktéž volí formát výstupních dat.
Kapitola *Rozbor tématu a použitých metod/technologií* popisuje numerické integrační metody řešení diferenciálních rovnic. Tyto metody jsou implementovány jako výpočetní jádro spojitého simulátoru. *Koncepce* popisuje způsob rozlišení jednotlivých součástí vstupních rovnic, což je základem pro vytvoření jednotlivých tříd, které tyto součásti implementují. To je popsáno v kapitole *Architektura simulátoru*. Ukázka použití, včetně testování validity simulátoru je popsána páté kapitole *Použití a testování simulátoru*. Tento text je ukončen kapitolou *Shrnutí simulačních experimentů a závěr*.

1.1 Autoři

Autory této práce jsou Kryštof Matěj a Vojtěch Kaisler. Důležitými zdroji informací byly přednášky předmětu IMS[2] a dokumentace simulační knihovny SIMLIB[1]. K realizaci také přispěl přednášející Dr. Ing. Petr Peringer, který vysvětlil teoreticky i prakticky principy spojité simulace.

1.2 Validita modelu

V tomto případě se jedná o porovnání výsledků spojitého simulátoru s analytickým řešením vstupních rovnic. Přesnost výsledků je závislá na použité numerické integrační metodě. Při testování musí být tato skutečnost brána v potaz.
Ne vždy existuje analytické řešení problému popsaného vstupními rovnicemi. Z toho důvodu je nutné výsledky takovýchto simulací porovnávat s výsledky jiných existujících simulátorů.

2. Rozbor tématu a použitých metod/technologií

Vyhodnocujeme diferenciální rovnici ve tvaru

$$y' = f(t,y)$$

a hledáme řešení ve tvaru [IMS, slide 221]

$$y(T) = y_0 + \int f(t,y)\,dt$$

Z pohledu spojitého simulátoru rozlišujeme jednokrokové a vícekrokové numerické integrační metody [IMS, slide 222]. Tyto metody nejsou přesné a proto se mohou výdledky dle metody lišit. Při analýze dosažených výsledků je třeba brát tuto skutečnost v potaz.

V této práci využité jednokrokové metody jsou Eulerova a Runge-Kutta 4.řádu. Jediným zástupcem vícekrokové metody je Adams-Bashforh.

2.1 Eulerova metoda[3][5]:

Jedná se o nejjednodušší z imlementovaných numerických metod. [IMS, slide 225]

$$y(t+h)=y(t)+hf(t,y)$$

Přibližná chyba této medody je ale $O(h^2)$, kde h je velikost kroku, což z ní činní metodu s nejmenší přesností [5].

2.2 Runge-Kutta 4.řádu[4][5]:

Velmi důležitá rodina metod používaná k výpočtu diferenciálních rovnic. V našem případě zastoupená svou 4. variantou. [IMS, slide 229]

$$k_1=hf(t,y)$$
$$k_2=hf\left(t+h/2, y(t)+k_1/2\right)$$
$$k_3=hf\left(t+h/2, y(t)+k_2/2\right)$$
$$k_4=hf\left(t+h, y(t)+k_3\right)$$
$$y(t+h)=y(t)+k_1/6+k_2/3+k_3/3+k_4/6$$

Přibližná chyba této metody je $O(h^5)$. Je tedy podstatně přesnější než Eulerova metoda. Toto tvrzení je však třeba brát s rezervou, neboť na velikost chyby má velký vliv složitost reálné trajektorie [5].

2.3 Adams-Bashforh [5]:

Jedniná vícekroková numerická metoda použita v této práci. Pro inicializaci této metody lze použít některou z jednokrokových metod.

$$y_{(n+1)}=y_n+h/24\left(9f_n-59f_{(n-1)}+37f_{(n-2)}-9f_{(n-3)}\right)$$

3 Koncepce

Aby bylo možné diferenciální rovnice vyhodnocovat, je třeba určit ze kterých prvků se mohou tyto rovnice skládat. Po rozložení na jednotlivé prvky je třeba určit vztahy mezi těmito prvky. Z toho důvodu je skládáme dle aretimetických operací do bloků. Vytváříme tedy syntaktický strom[6] nad vstupní rovnicí. Rozlišujeme následující prvky:

3.1 Konstanty [IMS, slide 214]

Neměnné bloky které po celou dobu simulace nabývají pouze jedné hodnoty.

3.2 Aritmetické operace

Specifickým prvkem jsou aritmetické operace, které spojují výše zmíněné prvky do bloků pro jednodušší orientaci ve výpočtu. Z těchto operací je součástí řešení sčítání, násobení, odčítání a dělení.

3.3 Integrátory

Všechny ostatní prvky jsou určeny jako funkce, jejichž hodnota se může během běhu simulace měnit. Hodnoty integrátorů jsou určeny vstupní rovnicí, počáteční hodnotou, použitou numerickou metodou a modelovým časem [IMS, slide 21].
Po rozlišení prvků vstupních rovnic se přechází na výpočet nad zadaným vstupním intervalem.
Na začátku simulace je pro každou funkci požadována inicializační hodnota. Zároveň se určí počáteční krok simulace.
Simulace probíhá postupným výpočtem jednotlivých integratorů dle zvolené numerické metody. Jednotlivé integrátory jsou vyhodnocovány v závislosti na pořadí zadání vstupních rovnic.

4. Architektura simulátoru

Simulátor je realizován jako program zapsaný jazykem C++. Základem simulátoru jsou bloky kterými jsou popsány vstupní rovnice. Třída `DSBlock` je abstraktní třídou pro jednotlivé prvky popsané v kapitole *Koncepce*. Ty vytváří syntaktický strom, který zaštiťuje třída `DSEquation`. Tento syntaktický strom je při každém kroku numerické metody přepočítán a vyhodnocen daným integrátorem (`DSIntegratorBlock`). Vzhledem k tomu že je tato třída potomkem třídy `DSBlock`, je možné tuto strukturu libovoně zanořovat.
Běh simulátoru je zajistěn knihovnou `DSFunctions.h`. Konkrétně funkcí `Run`, která zajišťuje přepočet simulačního času a je zodpovědná za výpis výstupních hodnot skrz globální objekt `runSampler` do kterého se přiřadí každá instance třídy `DSSampler`.
Při změně simulačního času se začnou postupně vyhodnocovat jednotlivé integrátory v takovém pořadí, v jakém byly registrovány do pole integrátorů.

4.1 Popis jednotlivých tříd:

Následuje popis výše zmíněných důležitých tříd a funkcí.

4.1.1 DSSampler:

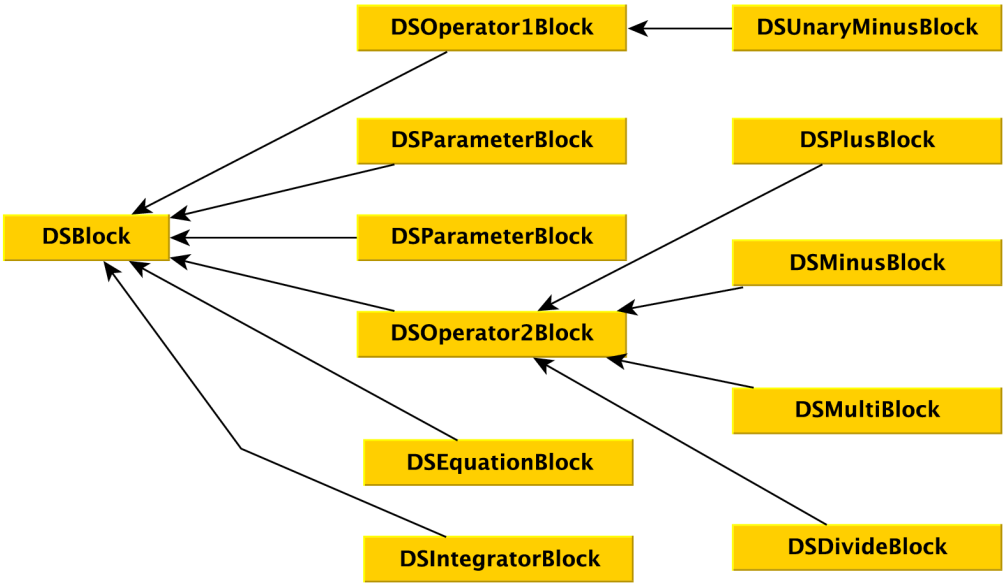
Tato třída tvoří předěl mezi uživatelem a knihovnou simulátoru. Tato třída byla inspirována třídou Sampler z knihovny SIMLIB. Srkz tuto třídu se určuje formát a krok výstupu simulátoru.

4.1.2 DSIntegratorBlock:

Jádro celého výpočtu tvoří instance této třídy. V této třídě jsou implementovány všechny numerické integrační metody. Pokud uživatel nezvolí globální funkcí `setMethod(<method>)` jinou numerickou metodu, tak je implicitně použita Eulerova metoda. Důležitou metodou této třídy je metoda `reset()`. Tato metoda vrací výpočet integrátoru při změně kroku výpočtu. Pokud je změna integrátoru větší než povolená odchylka, nastavuje příznak pro změnu kroku a reset současného výpočtu.

4.1.3 DSEquation:

Touto třídou je popsán syntaktický strom vstupní rovnice složený z výpočetních bloků. Ty mohou být tvořeny kombinací následujích bloků: `DSIntegratorBlock`, `DSMinusBlock`, `DSPlusBlock`, `DSTime`, `DSParametrBlock`, `DSMutiBlock`, `DSDivideBlock`, `DSUnaryMinusBlock`. Tyto bloky jsou spojovány aritmetickými operacemi do bloků podle počtu operatoru:



`DSOperator1Block`, `DSOperator2Block`.

Obr. č.1 Třídní dědičnosti pro dceřiné třídy DSBlock

4.2 Globální funkce `run()`:

Odpovídá za velikost výpočetního kroku, simulační čas a správu příznaků. Pro každý časový krok je volána funkce `runSimulation()` která zajistí propočítání jednotlivých integrátorů.

4.3 Globální funkce `SetMethod()`:

Tato funkce nastavuje numerickou integrační metodu. Vstupem je hodnota typu `IntegratorType` která nabývá hodnot `EULER, RUNGEKUTT, ADAMB` dle odpovídající metody.

5. Použití a testování simulátoru

Testováním chceme ověřit korekní vyhodnocení zadaných vstupních funkcí, nikoli jejich správné zadání.
Výsledky zde prezentovaných testů se mají co nejlépe příblížit výsledkům stejných vstupních funkcí pří použití simulační knihovny SIMLIB. Kromě jiného jsou zde uvedeny příklady použití simulátoru.

5.1 Použití simulátoru:

- make – přeložení aplikace `lorenz` a `shilnikov`, které reprezentují testované příklady uvedené níže
- make run - spustí vytvořené aplikace. Pokud není uveden výstupní soubor, je výsledek vytisknut na standardní výstup. Předvytvořené aplikace uvedou výstup do příslušných souborů, viz. níže
- make clean – smaže všechny soubory vygenerované příkazem make

V rámci ověření validity simulátoru jsme testovali simulaci Lorenzova atraktoru a Shilnikovy rovnice.

5.2 Testování Lorenzova atraktoru[4]

Vstupem tohoto testu jsou následující rovnice reprezentující Lorenzův atraktor[4]

$$dx/dt = \sigma(y - x)$$
$$dx/dt = x(\tau - z) - y$$
$$dz/dt = xy - \beta z$$

Přepsáním do programovacího jazyka dostáváme následující strukturu, která je základem simulace

```
struct Lorenz {
    DSIntegratorBlock x1, x2, x3;
    DSParameterBlock p1, p2, p3, p4;
    Lorenz(double sigma, double lambda, double b) :
    p1(sigma), p2(lambda), p3(b), p4(1),
    x1(p1*(x2 - x1), 1),           // dx1/dt = sigma * (x2 - x1)
    x2((p4 + p2 - x3)*x1 - x2, 1), // dx2/dt = (1 + lambda - x3) * x1 - x2
    x3(x1*x2 - p3*x3, 1){}          // dx1/dt = sigma * (x2 - x1)
};
```

kde `sigma`, `lambda`, `d` jsou hodnoty koeficientů.
Tyto koeficienty jsou jsme nastavili na hodnoty:

```
Lorenz L(10, 24, 2);
```

Hodnoty těchto koeficientů jsou shodné se stejnojmenným příkladem z dokumentace knihovny SIMLIB, neboť výsledky budou s tímto příkladem porovnávány.
K výpisu hodnot je definována funkce

```
void Sample() {
    print("%6.2f\t %g\t %g\n", t.value(), L.x1.value(), L.x2.value());
}
```

Tato funkce je předána třídě DSSampler k vyhodnocení.

```
DSSampler S(Sample, 0.05);
```

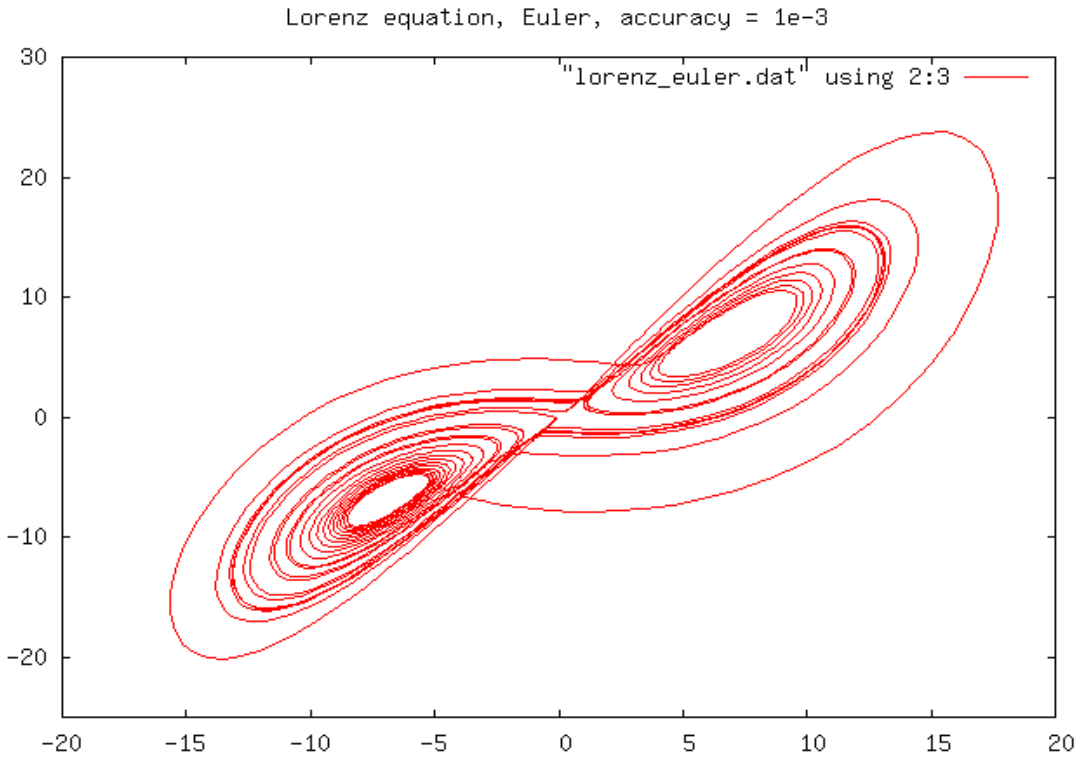
Výstup je uložen do souboru lorenz_euler.dat s krokem výpisu 0.05.

```
setOutput("lorenz_euler.dat");
init(0,30);
setAccuracy(1e-3);
```

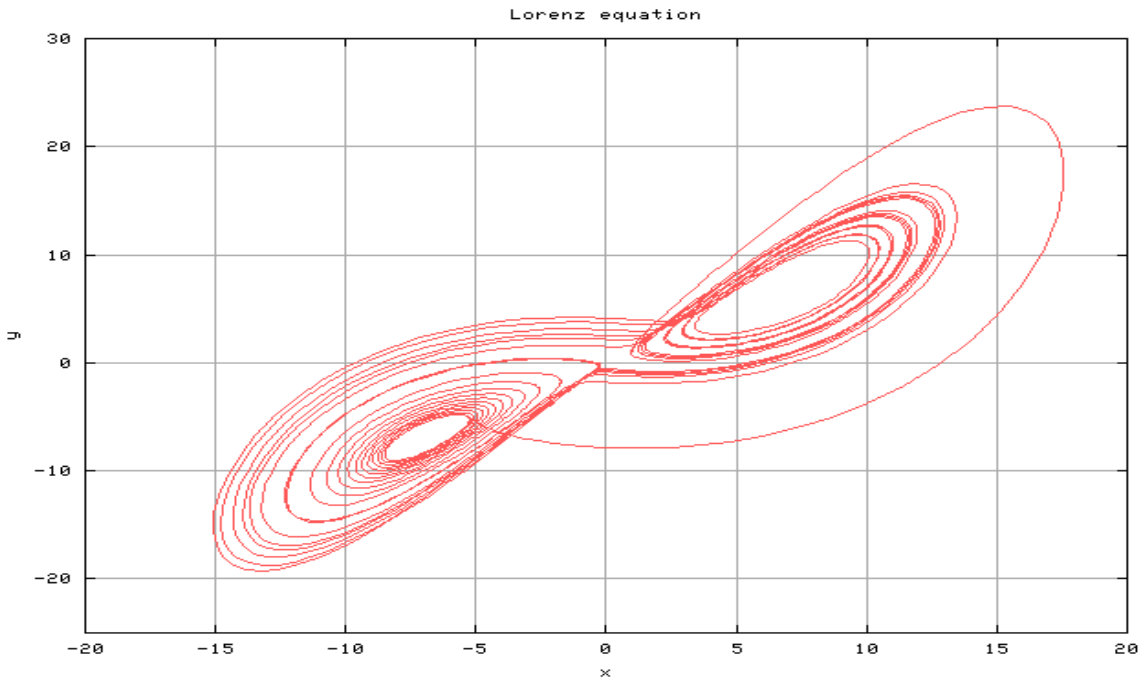
Pro výpočet integrátorů je implicitně nastavena Eulerova metoda.
Výpočet je zahájen voláním funkce

```
run();
```

Z takto získaných dat jsme sestavili graf č.1. Pro ověření validity simulátoru je třeba porovnat tento výsledek s výsledkem již existujícího simulátoru, v tomto případě simulátoru SIMLIB, viz. Graf č.2.



Graf č.1 Leronzův atraktor vypočítán Eulerovou metodou s přesností kroku 0.001



Graf č.2 Lorenzův atraktor použitím knihovny SIMLIB[1]

Můžeme vidět, že výsledky jsou velmi podobné, což lze v rámci přesnosti eulerovy metody považovat za správnou funkčnost. Výsledky použitím metod Runge-Kutta 4.řádu a Adams-Bashforh, které jsou uvedeny v příloze (grafy 5 a 6), dávají přesnější výsledky.

5.3 Testování Shilnikovy rovnice[8]

Základem této simulace je třída

```
class Silnikov {
public:
    DSParameterBlock a,b,c,d;
    DSIntegratorBlock x1, x2, x3;
    Silnikov(double _a, double _b, double _c, double _d) :
    a(_a), b(_b), c(_c), d(_d),
    x1(x2, 0.1234),
    x2(x3, 0.2),
    x3(-a*x3 - x2 + b*x1*(1 - c*x1 - d*x1*x1), 0.1) {}
};
```

Hodnoty parametrů jsou nastaveny následovně:

```
Silnikov e(0.4, 0.65, 0, 1);
```

Stejně jako v předchozím příkladu budeme výsledky našeho simulátoru porovnávat s výsledky simulační knihovny SIMLIB. Z toho důvodu je třeba zachovat stejné koeficienty, rozsah i přesnost.
K výpisu je použita funkce `Sample()`:

```
void Sample() {
    print("%6.2f\t %g\t %g\n", t.value(), e.x1.value(), e.x2.value());
}
```

Opět je tato funkce předána třídě `DSSampler` k vyhodnocení, tentokrát s krokem 0.01

```
DSSampler S(Sample, 0.01);
```

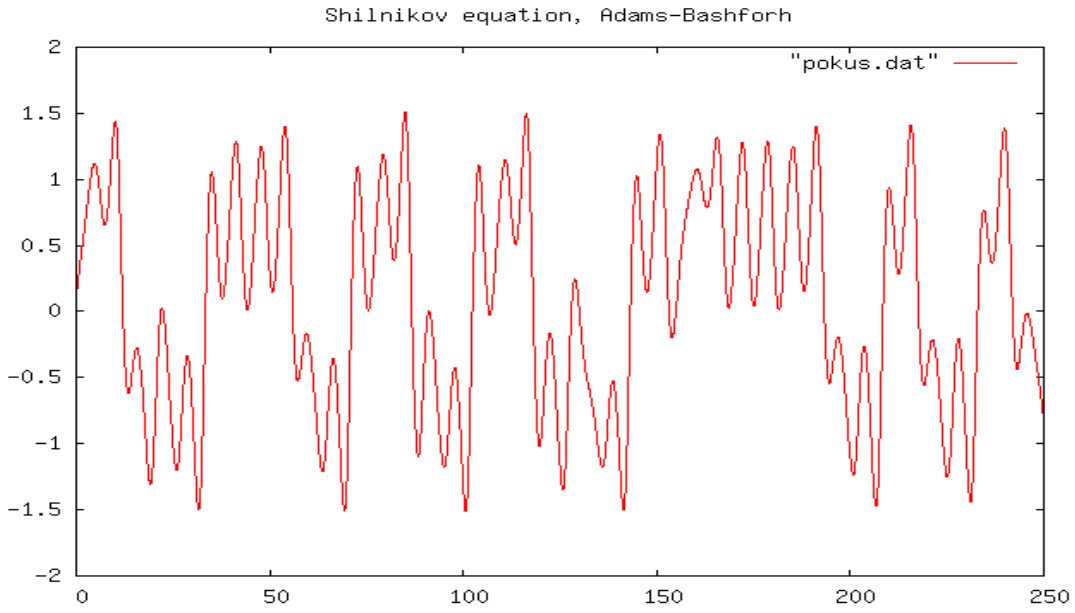
Tentokrát nastavíme metodu Adams-Bashforh jako numerickou metodu. Výsledek simulace bude zapsán do souboru Shilnikov_adamb.dat. Nejmenší simulační krok nastavíme na 10^{-8} a největší krok simulace na 0.001. Interval simulace je určen na (0, 250).

```
setMethod(ADAMB);
setOutput("Shilnikov_adamb.dat");
setAccuracy(1e-8,1e-3);
init(0, 250);
```

Simulaci spustíme příkazem

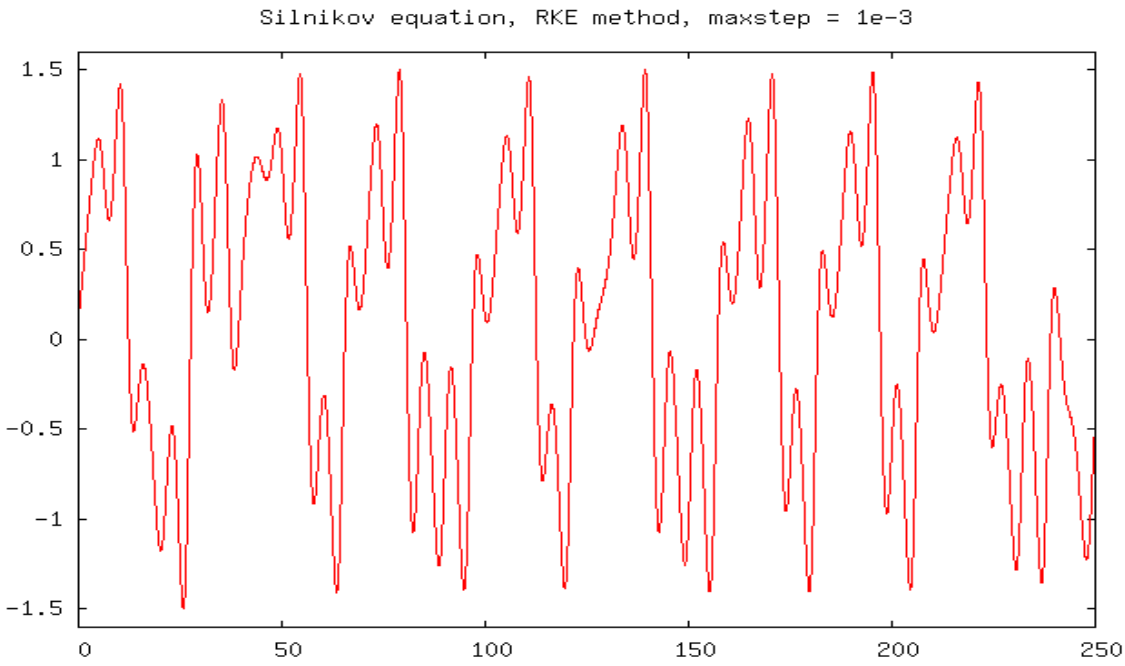
```
run();
```

Přenesením výsledných hodnot simulace do grafu dostáváme graf č. 3.



Graf č.3, Shilnikova rovnice metodou Adams-Bashforh.

Pro určení validity simulátoru při tomto testu porovnáme námi získaný výsledek s výsledkem knihovny SIMLIB (graf č.4).



Graf č.4. Shilnikova rovnice výsledkem knihovny SIMLIB[1]

Porovnáním těchto dvou výsledků dostáváme přibližně stejné výsledky, což potvrzuje validitu simulace, neboť výsledky jsou pro prvních několik desítek záznamů shodné, až posléze se začínají lišit. To může být způsobeno rozdílnou metodou výpočtu.
Výsledky použitím Eulerovy metody a Runge-Kutta 4.řádu jsou uvedeny v příloze (grafy 7 a 8). Obě dvě metody dávají podobné výsledky.

6. Shrnutí simulačních experimentů a závěr

Validita simulátoru byla ověřena na základě testů které porovnávají výstup tohoto simulátoru a výsledek za použití simulační knihovny SIMLIB. Tyto testy vykazovaly značnou podobnost výsledků. To že se nejednalo o úplnou shodu může být způsobeno rozdílnými integračními metodami, neboť tento simulátor využívá rozdílné metody oproti knihovně SIMLIB.

7. Reference

[1] PERINGER, Petr, David MARTINEK a David LEŠKA. *SIMLIB* [online]. 2011 [cit. 2013-12-06]. Dostupné z: http://www.fit.vutbr.cz/~peringer/SIMLIB/doc/html/index.html.en

[2] PERINGER, Petr. *Modelování a simulace* [online]. 2012 [cit. 2013-12-06]. Dostupné z: https://www.fit.vutbr.cz/study/courses/IMS/public/prednasky/IMS.pdf

[3]LAKOBA a TARAS. *Simple Euler method and its modifications (Lecture notes for MATH334, University of Vermont)* [online]. 2012 [cit. 2013-12-06]. Dostupné z: http://www.cems.uvm.edu/~tlakoba/math337/notes_1.pdf

[4]TAN, Delin a Zheng CHEN. *Simple Euler method and its modifications (Lecture notes for MATH334, University of Vermont)* [online]. 2012 [cit. 2013-12-06]. Dostupné z: http://msme.us/2012-2-1.pdf

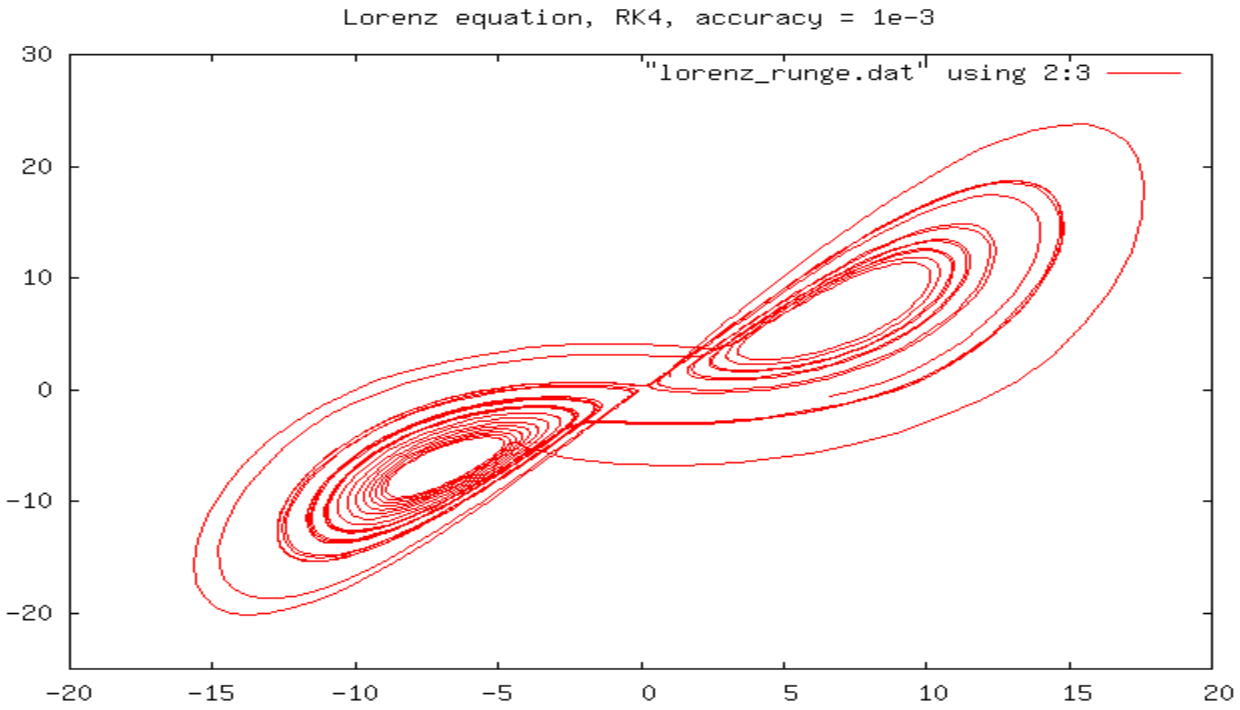
[5] *Numerické metody* [online]. 2013 [cit. 2013-12-06]. Dostupné z: http://www.fce.vutbr.cz/studium/materialy/Dynsys/kap7/kap7.htm

[6]MEDUNA, Alexander, Roman LUKÁŠ. *Formální jazyky a překladače* [online]. 2012 [cit. 2013-12-07]. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/IFJ-IT/texts/OporaIFJ.pdf?cid=8649

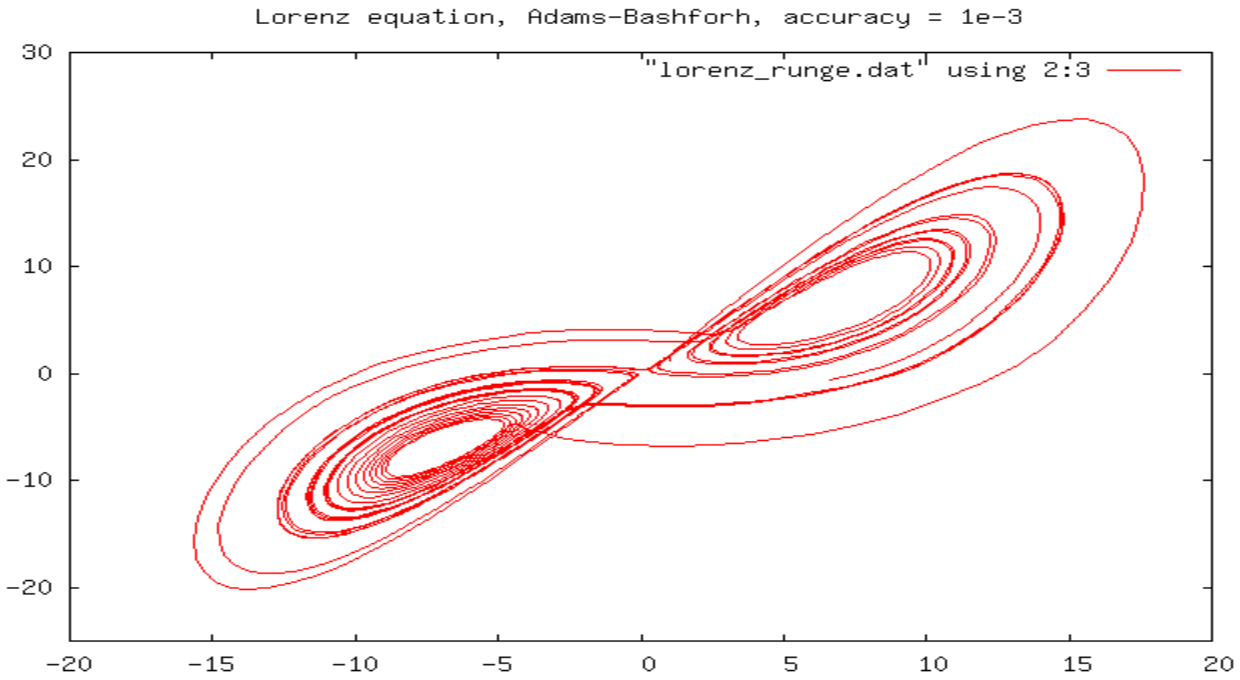
[7] *Lorenz Equation* [online]. 2006 [cit. 2013-12-06]. Dostupné z: http://planetmath.org/LorenzEquation

[8] PERINGER, Petr, David MARTINEK a David LEŠKA. *SIMLIB* [online]. 2011 [cit. 2013-12-06]. Dostupné z: http://www.fit.vutbr.cz/~peringer/SIMLIB/examples/silnikov.html.cs

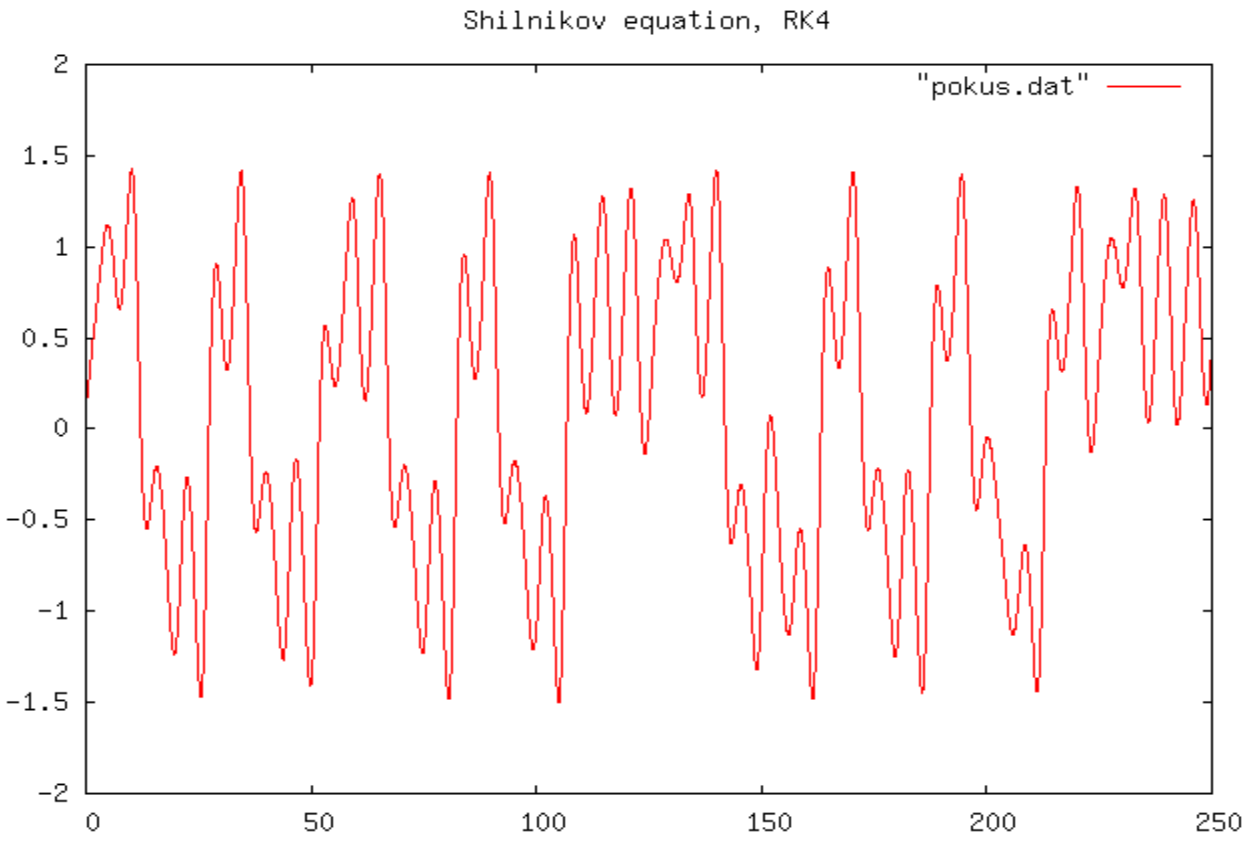
8. Příloha



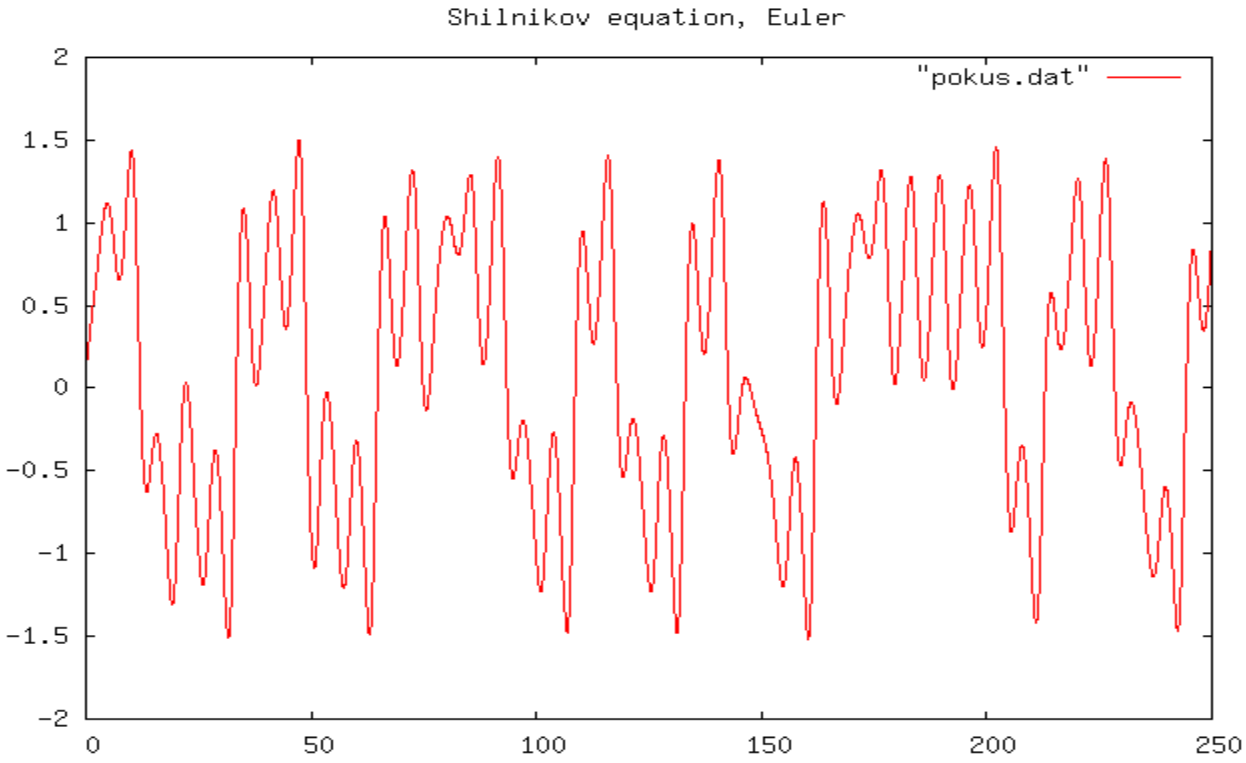
Graf č.5, Lorenzův atraktor řešen metodou Runge-Kutta 4.řádu



Graf č.6, Lorenzův atraktor řešen metodou Adams-Bashforh



Graf č.7, Šilnikova rovnice řešena metodou Runge-Kutta 4.řádu



Graf č.8, Šilnikova rovnice řešena Eulerovou metodou