Modelování a simulace 2012/13

# Implementace spojitého simulátoru

1. prosince 2012

Autoři: Milan Seitler, xseitl01@stud.fit.vutbr.cz
Martin Šafář, xsafar13@stud.fit.vutbr.cz
Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

# 1 Úvod

Tento text vznikl za účelem zdokumentování činnosti na projektu do předmětu Modelování a simulace (IMS) vyučovaného Fakultou informačních technologiií Vysokého učení technického v Brně. Popisuje proces implementace spojitého simulátoru v jazyce C++, který pro integrátory využívá numerické metody Eulerovu a Runge-Kutta 4. řádu. Funkčnost výsledného programu je demonstrována na dvou příkladech.

## 1.1 Autoři a zdroje

Na projektu spolupracovali studenti FIT VUT Milan Seitler a Martin Šafář. Nepřímo byli do řešení svými přednáškami zapojeni také Ing. Martin Hrubý, Ph.D. a Dr. Ing. Petr Peringer.

Autoři čerpali informace z výše uvedených přednášek a studijních materiálů kurzu Modelování a simulace (dále IMS) [1, 2], z absolvovaného kurzu Numerická matematika a pravěpodobnost [3], ze simulační knihovny SIMLIB [4] a také z referenční příručky jazyka C++ [5].

## 1.2 Prostředí pro ověřování činnosti simulátoru

Výsledný program byl testován na školním serveru merlin.fit.vutbr.cz (CentOS 5.5 64bit Linux).

Výstup programu byl testován na dvou systémech se třemi integrátory, výsledky byly porovnány s programem využívajícím knihovnu SIMLIB. Více v kapitole č. 5.

# 2 Rozbor tématu a použitých metod a technologií

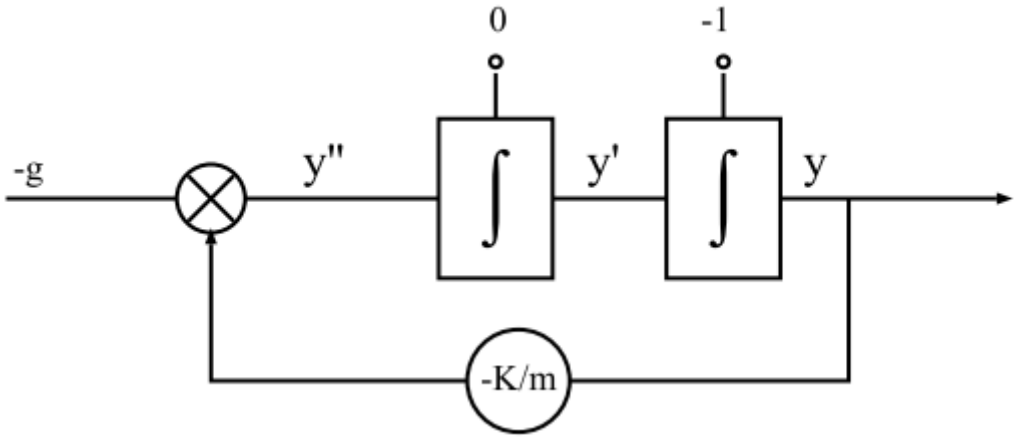
Spojitá simulace je simulace jevů, které jsou popsány spojitými rovnicemi. K popisu se používají zejména rovnice algebraické a difrenciální. (přednáška IMS 2.11.2012, [1, str. 211]). V rámci předmětu IMS a projektu jsme se zabývali řešením soustav obyčejných diferenciálních rovnic. Spojitých simulací využívají například elektrické a elektronické obvody, uplatňují se mj. i v oborech fyziky, astronomie či chemie.

## 2.1 Použité postupy

Pro implementaci byl zvolen jazyk C++ s použitím objektově orientovaného přístupu. Ten oproti klasickému způsobu programování přináší největší výhodu při implementaci bloků spojitého systému, kde lze jednotlivé typy bloků sestavit do hierarchické struktury a využít principy dědičnosti.

## 2.2 Původ použitých metod a technologií

Inspirací při tvorbě projektu byla knihovna SIMLIB, která umožňuje simulaci spojitých i diskrétních systémů, studijní opora předmětu IMS a algoritmus pro numerickou integrační




Obrázek 2.1: Grafový popis systému

metodu Runge-Kutta 4. řádu [6].

## 2.3 Popis spojitého systému

Při popisu spojitého systému diferenciálními rovnicemi lze využít maticového zápisu: [2, str. 57]

$$\frac{d}{dt}\vec{w}(t)=A(t)\vec{w}(t)+B(t)\vec{x}(t)$$
$$\vec{y}(t)=C(t)\vec{w}(t)+D(t)\vec{x}(t)$$

kde:
$\vec{x}$ je vektor $m$ vstupů
$\vec{w}$ je vektor $n$ stavových proměnných (výstupy integrátorů)
$\vec{y}$ je vektor $k$ výstupů
$A, B, C, D$ jsou matice koeficientů.

Názorným popisem spojitých systémů je grafový popis. Diferenciální rovnici [2, str. 58]

$$y'' + \frac{K}{m}y + g = 0$$

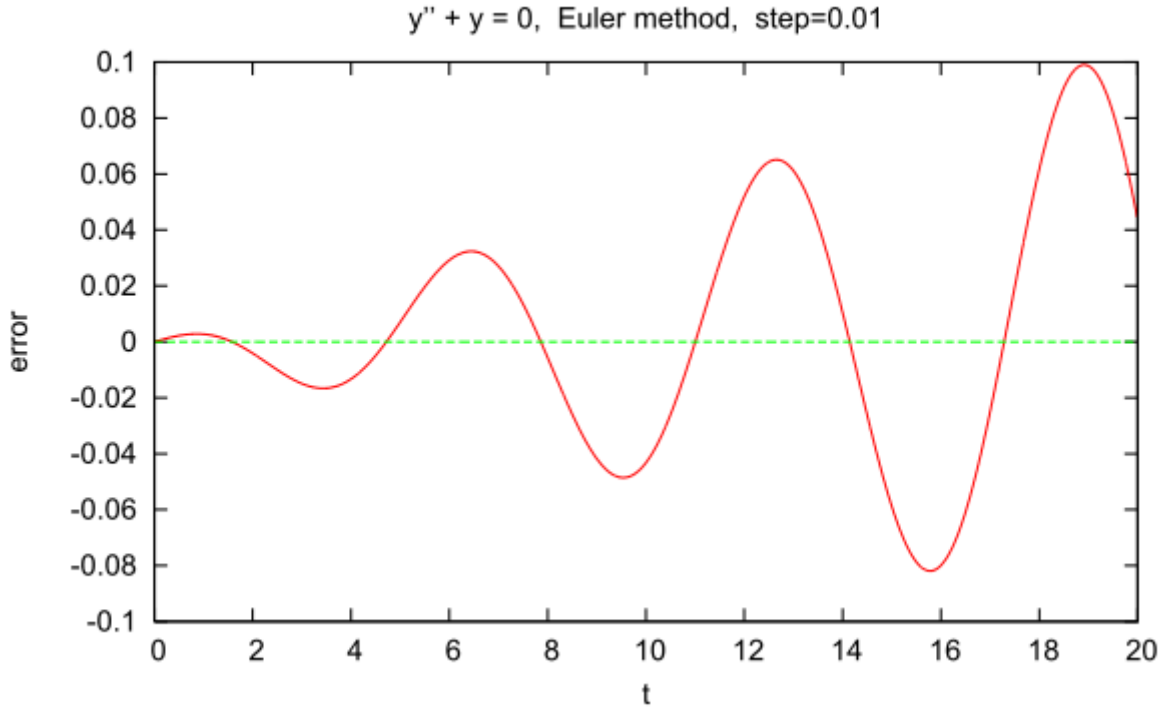
je možné zobrazit jako obr. 2.1. Na tomto obrázku rozlišujeme dva typy bloků – funkční (konstanta, násobení konstantou, aritmetické operace) a stavové (integrátor). V rámci projektu jsou pro integrátory implementovány dvě numerické integrační metody - Eulerova a Runge-Kutta 4. řádu.

## 2.4 Eulerova metoda

Eulerova metoda [2, str. 64] je jednoduchá jednokroková metoda. Počítá novou hodnotu řešení v čase $y(t+h)$ přímo bez pomocných bodů. Její vzorec je následující:

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

Implementace Eulerovy metody je velmi jednoduchá, z důvodu neefektivity však není příliš používaná. Obr. 2.2 zobrazuje postupné zvětšování chyby při řešení rovnice $y''+y = 0$ Eulerovou metodou.




Obrázek 2.2: chyba Eulerovy metody[2, str. 66]

## 2.5 Metoda Runge-Kutta 4. řádu

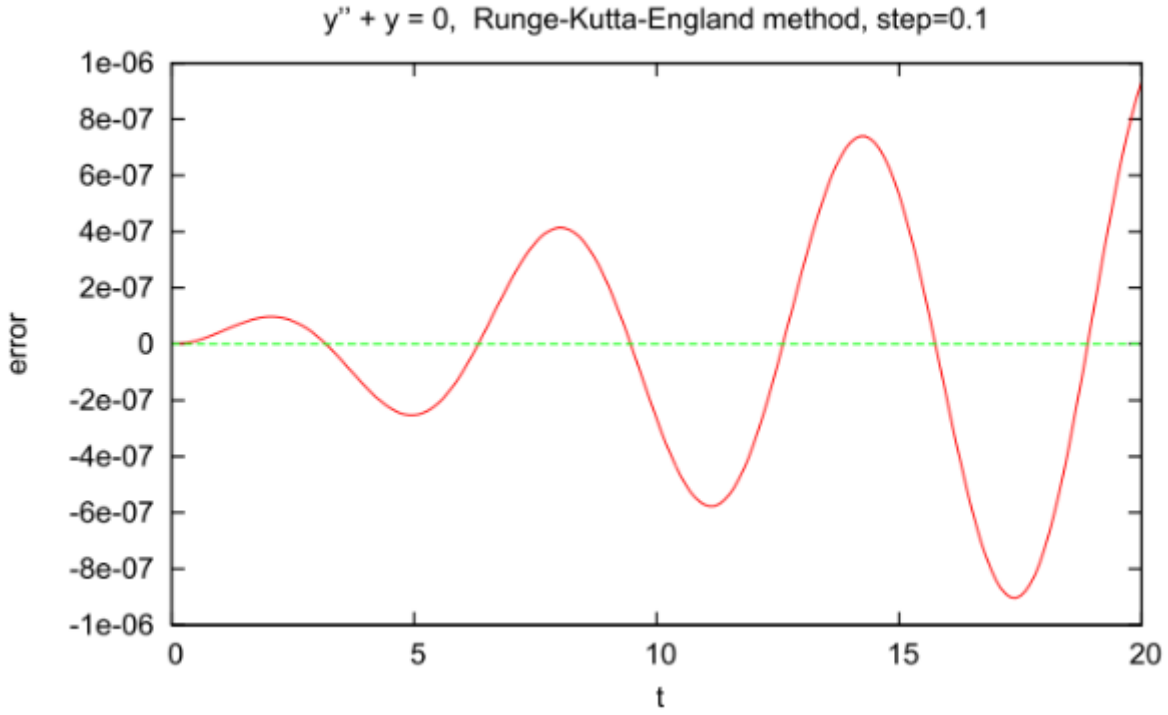
Metody Runge Kutta jsou jednokrokové [2, str. 66], oproti Eulerově metodě provádí více výpočtů a jsou tedy přesnější. Konkrétně metoda RK4 před výpočtem výsledného řešení vyhodnocuje ještě čtyři koeficienty k1-k4. Její vzorce jsou tyto:

$$
\begin{aligned}
k_1 &= hf(t, y(t)) \\
k_2 &= hf(t + \frac{h}{2}, y(t) + \frac{k_1}{2}) \\
k_3 &= hf(t + \frac{h}{2}, y(t) + \frac{k_2}{2}) \\
k_4 &= hf(t + h, y(t) + k_3) \\
y(t + h) &= y(t) + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}
\end{aligned}
$$

Rozdíl metody Runge-Kutta oproti Eulerově metodě demonstruje obr. 2.3, kde je zobrazena chyba metody Runge-Kutta-England (varianta RK4) při řešení stejné rovnice. Dle měřítka osy y (velikost chyby) lze vypozorovat, že chyba se v obou případech pohybuje ve zcela jiném řádu.

# 3 Koncepce

Cílem projektu bylo implementovat simulační jádro pro spojitou simulaci, které, podobně jako SIMLIB, poskytuje funkce pro modelování a simulaci spojitých systémů. Jedná se tedy o knihovnu, kterou uživatel zahrne do svého programu, ve kterém namodeluje požadovaný spojitý systém a následně spustí simulaci. Knihovna zpracovává soustavy diferenciálních rovnic prvního řádu, v případě rovnic vyšších řádů je nutné je nejprve převést [2, str. 59].




Obrázek 2.3: chyba metody Runge-Kutta-England[2, str. 67]

Při modelování systému je možné použít konstanty, aritmetické operace +-*/, unární minus a integrátory. Před započetím simulace je potřeba nastavit parametry simulace, tedy počáteční a koncový čas, délku časového kroku či počáteční podmínky integrátorů. Knihovna umožňuje nastavit výpis do uživatelem zvoleného souboru či na standardní výstup.

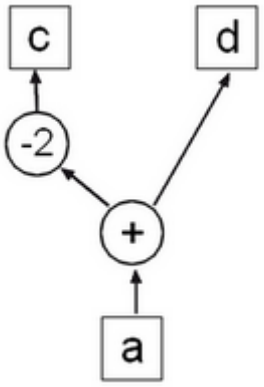
## 3.1 Vytvoření modelu

Spojité systémy se skládají z jednotlivých vzájemně propojených bloků. Klíčovými bloky při spojité simulaci jsou integrátory, jejichž vstupním výrazem může být kombinace dalších bloků systému a aritmetických operací. Jednotlivé bloky mezi sebou odkazují a vytváří tak hierarchickou strukturu pro jednotlivé výrazy. Například výraz $a = -2c + d$ lze graficky zobrazit viz obr. 3.1

## 3.2 Algoritmus spojitého simulátoru

Algoritmus spojitého simulátoru vypadá takto:

1. Namodeluj simulovaný systém
2. Nastav parametry simulace, počáteční podmínky a integrační metodu
3. Začni v nastaveném počátečním čase a dokud se čas nerovná zadanému koncovému, prováděj následující:
   (a) vyhodnoť vstupy všech bloků systému
   (b) proveď jeden krok numerickou metodou (vypočítej nové stavy integrátorů)
   (c) posuň čas simulace o zadanou délku kroku




Obrázek 3.1: výraz $a = -2c + d$ ve stromové struktuře bloků

## 3.3 Výstup programu

Standardním výstupem knihovny je výpis času a stavů všech integrátorů v každém kroku simulace. Uživatel může dále před zahájením simulace doplnit například záhlaví výstupu.

# 4 Architektura simulátoru

Architektura simulátoru vychází přepokladů uvedených v kapitole 3 a inspiruje se také simulační knihovnou SIMLIB. Mimo funkce vázající se k samotné simulaci jsou uživateli poskytnuty funkce `SetOutput` a `Print`, které slouží k nastavení výstupního souboru (výchozí je standardní výstup) a zápisu do něj.

## 4.1 Rozdělení tříd

### 4.1.1 Simulátor

Jádro simulátoru se nachází v třídě `Simulator`, která obsahuje parametry simulace jako jsou počáteční čas či délka kroku. Dále uchovává seznam integrátorů `integrators` a ukazatel `method` na metodu použitou pro numerickou integraci. Všechny atributy třídy kromě seznamu integrátorů jsou privátní a pro přístup k nim slouží veřejné metody `SetStart, GetStart` atd. analogicky. Pro přidání integrátoru do seznamu je možné použít metodu `AddIntegrator`. Třída `Simulator` poskytuje také metody pro inicializaci (`InitAll`) a vyhodnocení vstupů (`EvalAll`) integrátorů. Samotnou simulaci spouští metoda `Simulate`.

### 4.1.2 Bloky

Jak již bylo vysvětleno v kapitole 3, simulovaný systém lze modelovat pomocí bloků. Hlavním blok reprezentuje třída `Block`, jejími přímými potomky jsou třídy `Constant, Variable, Integrator, Time, Operation`. Všechny tyto musejí implementovat virtuální metodu `Value`, která vrací aktuální hodnotu daného bloku. Na třídě `Block` je nezávislá třída `Input`, ta v sobě obaluje ukazatel na jednotlivé bloky, čímž umožňuje univerzální použití bloků ve všech výrazech. Třída `Constant` obsahuje číselnou konstantu typu double, třída `Variable` ukládá libovolný výraz, ve třídě `Time` se nachází ukazatel na globální čas simulace, je tedy možné do výpočtu zahrnout i hodnotu času (použito při integraci metodou RK4).

#### 4.1.2.1 Integrátor

Třída `Integrator` je potomkem třídy `Block`, oproti ostatním potomkům však nabízí více atributů a metod. Kromě vstupního výrazu obsahuje také jeho poslední vypočítanou hodnotu, dále ukládá počáteční podmínku integátoru a jeho stav. Metody třídy umožňují nastavit vstup integrátoru, přístup k atributům, vyhodnocení aktuální hodnoty vstupního výrazu a počáteční inicializaci stavu integrátoru.

#### 4.1.2.2 Aritmetické operace

Základní třídou aritmetických operací je třída `Operation`, která je potomkem třídy `Block` a obsahuje dva výrazy, každý pro jeden z operandů (v případě unárního mínus je využit pouze jeden výraz). Aktuální hodnoty výrazu lze získat metodami `Input1Value` a `Input2Value`. Potomky této třídy jsou jednotlivé aritmetické operace `Add, Sub, Mul, Div, Minus`, které provádí danou operaci nad dvěma uloženými operandy ve formě výrazů v instancích třídy `Input`. Tyto funkce jsou volány prostřednictvím přetížených aritmetických operátorů.

### 4.1.3 Numerické integrační metody

Nadřazenou třídou pro obě metody je třída `Method` která pouze deklaruje virtuální metodu `Integrate`, kterou obě třídy potomků implementují. Třídy `RK4` a `Euler` obsaují referenci na hlavní simulační třídu `Simulator`, skrze kterou přistupují k seznamu integrátorů. Dále v metodě `Integrate` implementují danou numerickou metodu dle vzorců uvedených v kapitole 2.

# 5 Podstata simulačních experimentů a jejich průběh

Činnost simulátoru byla ověřena na dvou příkladech s diferenciálními rovnicemi, první z nich byl převzat z demonstračního cvičení IMS, druhým příkladem je Lorenzův atraktor.

## 5.1 Postup experimentování

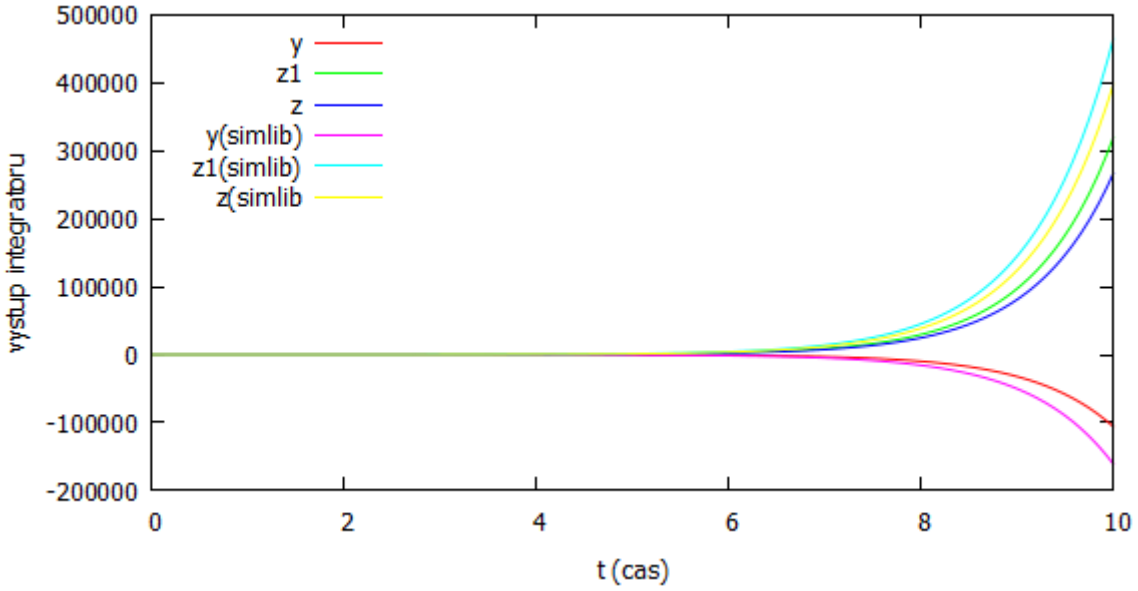
Výchozí rovnice nejprve byly upraveny do tvaru, který je možné zadat jako vstup integrátoru. Pro každý příklad byly vytvořeny modely pro náš program i pro SIMLIB, byly nastaveny stejné parametry simulace, aby bylo možné porovnat výsledky. Výpočty integrátorů byly zobrazeny v podobě grafů, kde jsou do jednoho grafu pro srovnání vyneseny výstupy obou programů.

## 5.2 Experimenty

### 5.2.1 Soustava diferenciálních rovnic vyššího řádu

Tento příklad byl převzat z demonstračního cvičení předmětu IMS z 28.11.2012. Je dána soustava diferenciálních rovnic

$$8y' + 3y + 5z = t$$
$$2z'' + z + 5y' = 0$$




Obrázek 5.1: hodnoty integrátorů prvního příkladu v čase t

Parametry simulace byly následující: počáteční čas = 0, koncový čas = 10, délka kroku = 0.01, numerická integrační metoda byla Eulerova. Vstupy integrátorů byly nastaveny takto:

$$y : \frac{-3y - 5z + t}{8}, y(0) = 1$$
$$z' : \frac{-z - 5y'}{2}, z'(0) = 2$$
$$z : z', z(0) = 3$$

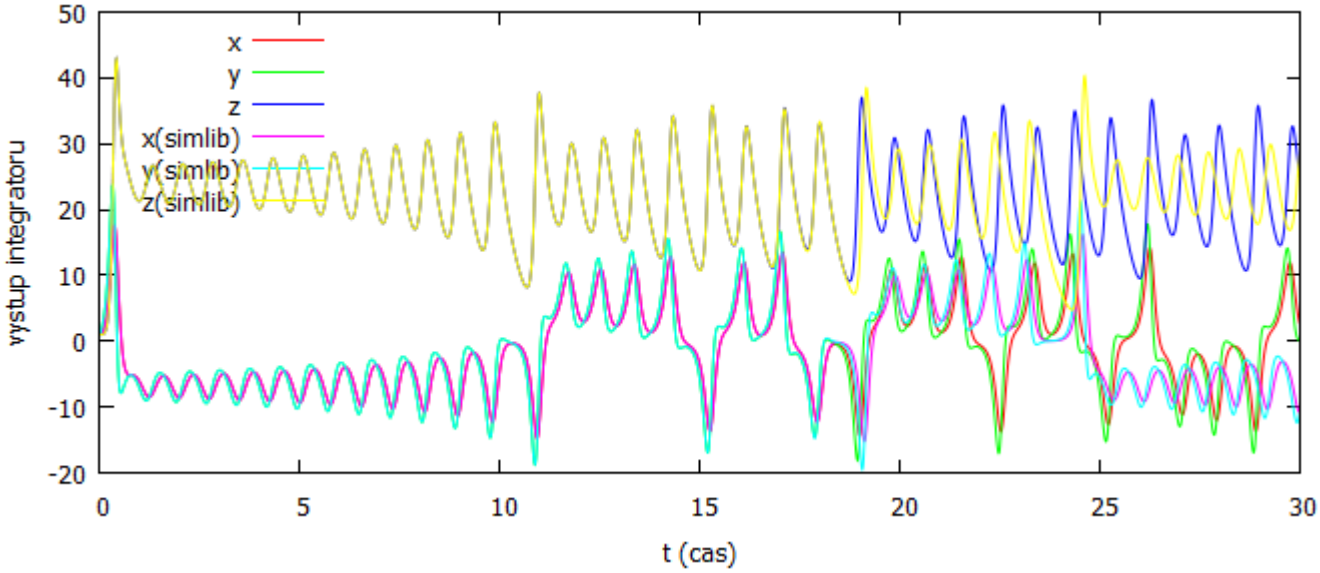
Průběh hodnot integrátorů zachycuje graf 5.1.

### 5.2.2 Lorenzův atraktor

Tento příklad byl převzat z ukázkových příkladů knihovny SIMLIB. Lorenzův atraktor popisují tři diferenciální rovnice:

$$x' = \sigma(y - x)$$
$$y' = x(1 + \lambda - z) - y$$
$$z' = x * y - b * z$$

Parametry simulace byly následující: počáteční čas = 0, koncový čas = 30, délka kroku = 0.01, numerická integrační metoda byla Runge-Kutta. Vstupy integrátorů jsou analogické výše uvedeným rovnicím. Průběh hodnot integrátorů je zobrazen na grafu 5.2.




Obrázek 5.2: hodnoty integrátorů druhého příkladu v čase t

## 5.3 Závěry experimentů

V obou experimentech byly srovnány výstupy implementovaného programu a knihovny SIMLIB. Z grafu lze vyčíst, že hodnoty nejsou zcela stejné, neboť SIMLIB používá upravené verze obou numerických metod, v naší implementaci jsou použity základní varianty. U prvního příkladu lze vidět, jak se rozdíl mezi hodnotami postupně zvětšuje.

# 6 Shrnutí simulačních experimentů a závěr

V rámci projektu byla vytvořena knihovna v jazyce C++ umoňující simulaci spojitých systémů. Byly implementovány dvě numerické integrační metody, Eulerova a Runge-Kutta 4. řádu. Při experimentování se simulátorem a porovnání výsledků s knihovnou SIMLIB byly zjištěny drobné odchylky způsobené použitím úpravených variant numerických metod v SIMLIBu. V dalším vývoji simulátoru by bylo možné zahrnout výpočet chyby či detekci rychlých smyček.

# Literatura


[1] P. Peringer: přednášky předmětu *Modelování a simulace*
https://www.fit.vutbr.cz/study/courses/IMS/public/prednasky/IMS.pdf

[2] P. Peringer: studijní opora *Modelování a simulace*

[3] B. Fajmon, I. Růžičková: studijní opora *Matematika 3*

[4] Simulační knihovna SIMLIB
http://www.fit.vutbr.cz/~peringer/SIMLIB/

[5] Referenční příručka jazyka C++
www.cplusplus.com/reference/

[6] Implementace metody Runge-Kutta 4. řádu v C++
http://www.fit.vutbr.cz/study/courses/IMS/public/priklady/rk4-test.c.ht