

# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ  
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION  
DEPARTMENT OF RADIO ELECTRONICS

## ALGORITMY PRO VÝPOČET DIMENZÍ STAVOVÝCH ATRAKTORŮ

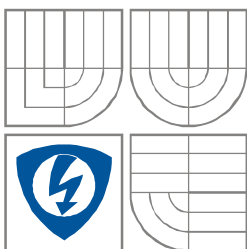
BAKALÁŘSKÁ PRÁCE  
BACHELOR'S THESIS

AUTOR PRÁCE  
AUTHOR

TOMÁŠ GÖTTHANS



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ  
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH  
TECHNologiÍ  
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION  
DEPARTMENT OF RADIO ELECTRONICS

## ALGORITMY PRO VÝPOČET DIMENZÍ STAVOVÝCH ATRAKTORŮ

ALGORITHMS FOR COMPUTATION OF THE DIMENSIONS OF STATE SPACE  
ATTRACTORS

BAKALÁŘSKÁ PRÁCE  
BACHELOR'S THESIS

AUTOR PRÁCE  
AUTHOR

Tomáš Götthans

VEDOUCÍ PRÁCE  
SUPERVISOR

Ing. Jiří Petržela, Ph.D.

BRNO,

2008

## **ORIGINÁLNÍ ZADÁNÍ**

# LICENČNÍ SMLOUVA POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

## 1. Pan/paní

Jméno a příjmení: Tomáš Götthans  
Bytem: bří. Mrštíků 6, Šlapanice, 664 51  
Narozen/a (datum a místo): 26. ledna 1985 v Brně

(dále jen „autor“)

a

## 2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií  
se sídlem Údolní 53, Brno, 602 00  
jejímž jménem jedná na základě písemného pověření děkanem fakulty:  
prof. Dr. Ing. Zbyněk Raida, předseda rady oboru Elektronika a sdělovací technika  
(dále jen „nabyvatel“)

## Čl. 1

### Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):

- ☐ disertační práce
  - ☐ diplomová práce
  - ☒ bakalářská práce
  - ☐ jiná práce, jejíž druh je specifikován jako .....
- (dále jen VŠKP nebo dílo)

Název VŠKP: Algoritmy pro výpočet dimenzí stavových atraktorů

Vedoucí/ školitel VŠKP: Ing. Jiří Petržela, Ph.D.

Ústav: Ústav radioelektroniky

Datum obhajoby VŠKP: \_\_\_\_\_

VŠKP odevzdal autor nabyvateli\* :

- ☒ v tištěné formě – počet exemplářů: 2
- ☒ v elektronické formě – počet exemplářů: 2

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.

3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.

4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

---

\* hodící se zaškrtněte

## Článek 2

### Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
  - **Y** ihned po uzavření této smlouvy
  - 1 rok po uzavření této smlouvy
  - 3 roky po uzavření této smlouvy
  - 5 let po uzavření této smlouvy
  - 10 let po uzavření této smlouvy(z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

## Článek 3

### Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísní a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: 6. června 2008

.....  
Nabyvatel

.....  
Autor

## **Abstrakt**

Geometrie chaotických atraktorů může mnohdy být komplexní a složitá k popisu bez matematických nástrojů. Hlavním předmětem této práce je vytvoření programu pro výpočet dimenzí stavových atraktorů. Pomocí něj dokonce můžeme zjistit, za je velmi systém citlivý na počáteční podmínky. Nejdříve musíme numericky integrovat daný systém diferenciálních rovnic, dále musíme vytvořit datovou posloupnost ze které můžeme určit kapacitu nebo Kaplan-Yorkeho dimenzi. Hlavním cílem programu je analyzovat a rozpoznat chaotické chování systémů a srovnat dosažené výsledky početního systému s teoretickými předpoklady.

## **Abstract**

The geometry of chaotic attractors can be complex and difficult to describe without some mathematical tool. The topic of this contribution is the realization of program for computing the dimensions of state space attractors. We can also find out if the system is highly sensitive to initial conditions. First we need to numerically integrate system of equations, create a data set and finally we can estimate the capacity or Kaplan-Yorke dimension. The main objective of derived program is to analyze and determine chaotic behavior providing a chance to discuss the accuracy of computation engine and theoretical value.

## **BIBLIOGRAFICKÁ CITACE**

GÖTTTHANS, T. ALGORITMY PRO VÝPOČET DIMENZÍ STAVOVÝCH ATRAKTORŮ. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. XY s. Vedoucí bakalářské práce Ing. Jiří Petržela, Ph.D.

# Prohlášení

Prohlašuji, že svou bakalářskou práci na téma Algoritmy pro výpočet dimenzí stavových atraktorů jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 6. června 2008

.....  
podpis autora

# Poděkování

Děkuji vedoucímu bakalářské práce Ing. Jiřímu Petrželovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne 6. června 2008

.....  
podpis autora

# 1 OBSAH

1	OBSAH.....	8
2	ÚVOD DO TEORIE CHAOSU.....	10
	2.1 Chaos.....	11
	2.2 Atraktory .....	11
	2.3 Podivné atraktory .....	12
	2.4 Citlivost na počáteční podmínky .....	12
3	DIMENZE.....	14
	3.1 Euklidovská dimenze .....	14
	3.2 Topologická dimenze .....	14
	3.3 Hausdorffova dimenze .....	14
4	METODY MĚŘENÍ DIMENZÍ.....	15
	4.1 Měření Hausdorffovy dimenze .....	15
	4.2 Mřížková dimenze .....	16
	4.3 Kaplan Yorkeho dimenze .....	18
5	NUMERICKÉ METODY.....	18
	5.1 Numerické řešení diferenciálních rovnic .....	18
	5.2 Eulerova metoda: .....	19
	5.3 Metody Runge-Kutta.....	20
	5.4 Porovnání přesnosti metod .....	20
6	LYAPUNOVY EXPONENTY .....	22
7	ORTOGONÁLNÍ A ORTONORMÁLNÍ BÁZE .....	23
	7.1 Gram-Schmidtova ortonormalizace .....	23
8	PRINCIP NEURČITOSTI - HEISENBERG.....	25
9	KARDIOVASKULÁRNÍ RYTMY .....	26
10	MOZEK A CHAOS.....	27
	10.1 Neuron.....	27
	10.2 Mozek.....	28
11	VZORKOÁNÍ.....	28
12	PROGRAMOVÁ REALIZACE.....	30
	12.1 Programové řešení numerické integrace .....	30
	12.2 Ukládání do souboru .....	33
	12.3 3D Vizualizace dat .....	33
	12.4 2D Vizualizace dat .....	35
	12.5 Mřížková dimenze.....	36
	12.6 Kaplan-Yorkeho dimenze.....	39



13	ZÁVĚR.....	46
14	POUŽITÁ LITERATURA.....	47
15	SEZNAM OBRÁZKŮ:.....	48
16	SEZNAM PŘÍLOH.....	49

## 2 ÚVOD DO TEORIE CHAOSU

Teorie chaosu se zabývá chováním jistých nelineárních dynamických systémů, které (za jistých podmínek) vykazují jev známý jako chaos, nejvýznamněji charakterizovaný citlivostí na počáteční podmínky. V důsledku této citlivosti se chování těchto fyzikálních systémů, vykazujících chaos, jeví jako náhodné, i když model systému je 'deterministický' v tom smyslu, že je dobře definovaný a neobsahuje žádné náhodné parametry. Příklady takových systémů zahrnují atmosféru, solární systém, tektoniku zemských desek, turbulenci tekutin, ekonomii, vývoj populace.

Kořeny teorie chaosu lze datovat k roku 1900, ve studiích Henri Poincarého o problému pohybu 3 objektů se vzájemnou gravitační silou, tzv. problému tří těles. Poincaré objevil, že mohou existovat orbity, které jsou neperiodické, a které nejsou ani neustále vzrůstající ani se neblíží k pevnému bodu. Pozdější studie, také na téma nelineárních diferenciálních rovnic, byly realizovány G. D. Birkhoffem, A. N. Kolmogorovem, M.L. Cartwrightovou, J.E. Littlewoodem, a Stephenem Smalem. Kromě Smaleho, který snad jako první čistý matematik studoval nelineární dynamiku, byly všechny tyto studie přímo inspirovány fyzikou: problém tří těles v případě Birkhoffa, turbulence a astronomické problémy v případě Kolmogorova, a radiové technice v případě Cartwrightové a Littlewooda. Ačkoliv chaotický pohyb planet nebyl pozorován, experimentátoři narazili na turbulenci v pohybu kapalin a neperiodické kmity v radiových obvodech, bez podpory teorie, která by vysvětlila jejich pozorování.

Teorie chaosu rychle postupovala vpřed po polovině minulého století, kdy se stalo pro některé vědce zřejmé, že lineární teorie, převažující teorie systémů v tomto období, prostě nemůže vysvětlit pozorované chování v určitých experimentech, jako jsou logistické mapy. Hlavním katalyzátorem vývoje teorie chaosu byl elektronický počítač. Většina matematických teorií chaosu zahrnuje jednoduché opakované iterace, jejichž vývoj je nepraktické zkoušet ručně. Elektronické počítače výzkum takových systémů velice usnadňují. Jeden z prvních elektronických počítačů, ENIAC, byl použit ke studiu jednoduchých modelů předpovědi počasí. Jedním z prvních pionýrů této teorie byl Edward Lorenz, jehož zájem o chaos vznikl náhodně během jeho práce na předpovědi počasí v roce 1961. Lorenz použil počítač Royal McBee LPG-30 k výpočtu svého modelu simulujícího počasí. Chtěl vidět opět svou sekvenci, a aby ušetřil čas, začal simulaci zprostředka. Měl totiž vytištěna data z minulé simulace a tak je zadal jako vstupní data do svého modelu.

K jeho překvapení bylo předpovídané počasí zcela jiné, než na jeho původním modelu. Lorenz zkoumal, proč tomu tak je, a příčinu objevil ve své sestavě. Sestava zaokrouhlovala proměnné na 3 desetinná místa, zatímco počítač pracoval s 5 desetinnými místy. Tento rozdíl je malý a neměl by mít prakticky na řešení vliv. Avšak Lorenz objevil, že malé změny v počátečních podmínkách vedou k velkým změnám na výstupu z dlouhodobého hlediska.

Pojem chaos, jak je používán v matematice, byl vytvořen aplikovaným matematikem Jamesem A. Yorkem. Mooreův zákon a dostupnost levnějších počítačů rozšířila možnost zkoumání teorie chaosu. V současné době pokračuje velmi aktivně zkoumání této teorie.

Systémy, které vykazují matematický chaos, jsou v jistém smyslu složitě uspořádané. Tím je význam slova v matematice a fyzice v jistém nesouladu s obvyklým chápáním slova chaos jako totálního nepořádku. Původ tohoto slova lze najít v řecké mytologii.

## 2.1 Chaos

Termínem *chaos* je označena taková vlastnost nějakého dynamického a současně i deterministického systému, při jejíž platnosti je nemožné vypočítat budoucí stav systému. Chaos nastává zejména u těch dynamických systémů, které vykazují velkou citlivost na počáteční podmínky. V takových systémech se při volbě minimálně dvou nekonečně blízkých počátečních bodů (reprezentujících počáteční podmínky systému) tyto dva body posléze exponenciálně vzdalují, takže budoucí stav systému není možné žádným způsobem předpovědět.

## 2.2 Atraktory

Jedním způsobem vizualizace chaotického pohybu, nebo opravdu libovolného typu pohybu, je vytvoření fázového diagramu pohybu. V takovém diagramu je čas implicitní a každá osa reprezentuje jednu dimenzi stavu. Například někdo kreslí pozici kyvadla vůči jeho rychlosti. Kyvadlo v klidu bude zobrazeno jako bod a kyvadlo v periodickém pohybu bude nakresleno jako jednoduchá uzavřená křivka. Když takový graf vytváří uzavřenou křivku, křivka se nazývá orbita.

Často je na fázových diagramech vidět, že většina stavových trajektorií se přibližuje a obmotává nějakou obecnou limitu. Systém končí ve stejném pohybu pro všechny počáteční stavy v oblasti okolo tohoto pohybu, téměř jako by byl systém k tomuto pohybu (trajektorii fázového prostoru) přitahován (anglicky „attracted“). Tento „cílový“ pohyb je tedy případně nazýván atraktor systému a je velmi častý pro nucené disipativní systémy.

Atraktor (anglicky „attractor“) dynamického systému je tedy množina stavů, do kterých systém směřuje. Je to množina hodnot, kterých může nabývat stavový vektor dynamického systému po dostatečně dlouhém časovém úseku od počátečního impulsu.

Atraktory se dají rozdělit do několika tříd:

atraktorem jsou pevné body

atraktorem jsou periodické body

atraktorem jsou kvaziperiodické body

atraktor je chaotický, tzv. podivný atraktor

Jsou-li *atraktorem* dynamického systému *pevné body* (resp. množina pevných bodů), jedná se o nejjednodušší případ, protože se systém v nekonečném čase ustálil v nějakém stabilním stavu, který je možné dopředu vypočítat.

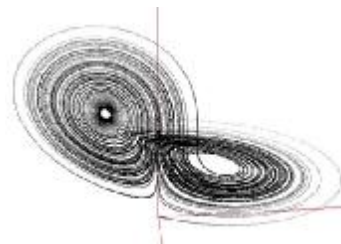
Jsou-li *atraktorem periodické* (resp. *kvaziperiodické*) *body*, jde také o vcelku jednoduchý případ. Systém se totiž po určité době ustálí tak, že osciluje mezi několika stavy. Ty mohou být buď spočitatelné (astabilní klopný obvod), nebo nespočitatelné (dráha osamělé planety okolo slunce).

Je-li *atraktor chaotický*, znamená to, že výsledný stav systému nelze v podstatě nijak dopředu předpovědět. To může být způsobeno mimo jiné tím, že je systém velmi citlivý na počáteční podmínky. Chaotičnost v tomto případě neznamena náhodnost, protože se stále bavíme o systémech deterministických. Podivný atraktor je fraktál s Hausdorffovou dimenzí mezi 2 a 3.

## 2.3 Podivné atraktory

Termín *podivný atraktor* není ještě přesně matematicky definován (definice sice existují, ale nezahrnují všechny typy podivných atraktorů), ale považujeme za něj takový atraktor, který vykazuje vlastnosti do jisté míry shodné s těmi, jaké mají fraktály. Dále platí, že všechny chaotické atraktory jsou současně podivnými atraktory, opačná implikace však neplatí.

Například jednoduchý trojdimenzionální model Edwarda Lorenze vede ke slavnému Lorenzovu atraktoru. Lorenzův atraktor je jeden z nejznámějších diagramů chaotických systémů, protože nejen že byl jeden z prvních popsanych, ale zároveň je jeden z nejsložitějších. Vznikají v něm velmi zajímavé obrazce, které vypadají jako motýlí křídla.



Obr. 1 Lorenzův atraktor popisuje pohyb systému ve stavovém prostoru. Zde pro počáteční hodnoty  $r = 28$ ,  $\sigma = 10$ ,  $b = 8/3$ .

Rovnice popisující Lorenzův atraktor:

$$\begin{aligned}\frac{dx}{dt} &= \sigma(y - x) \\ \frac{dy}{dt} &= x(r - z) - y \\ \frac{dz}{dt} &= xy - bz\end{aligned}$$

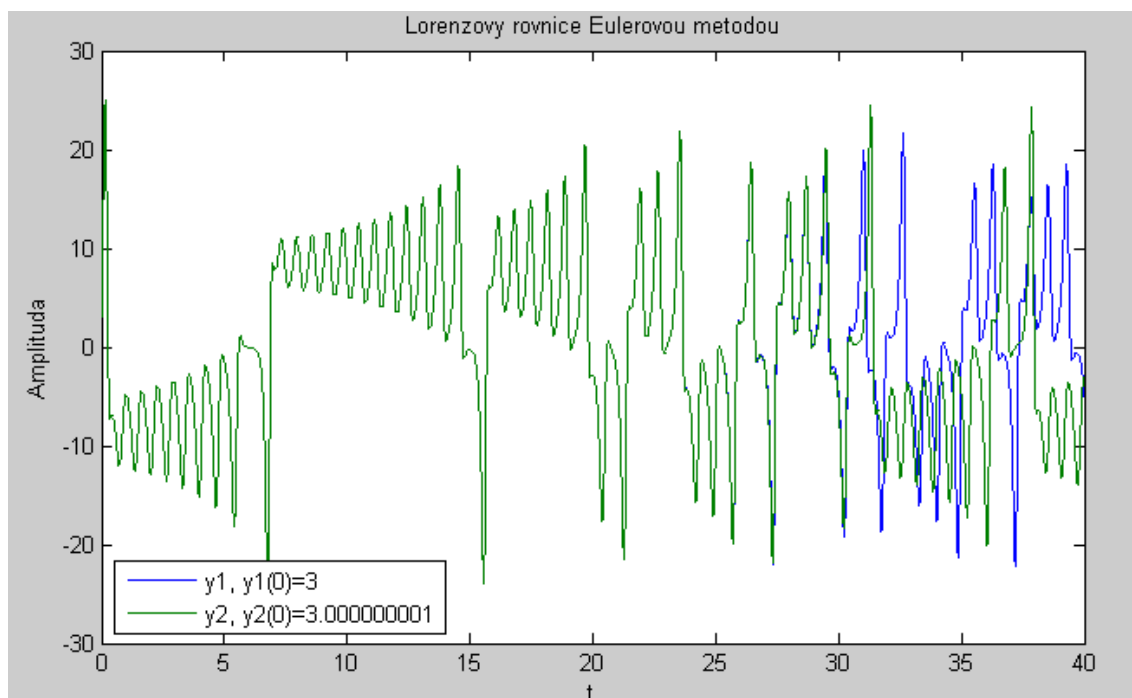
(2.3.1)

Kde  $\sigma$  je Prandtlovo číslo a  $r$  je Rayleighovo číslo (redukované).  $\sigma, r, b > 0$ , ale obvykle  $\sigma = 10$ ,  $b = 8/3$  a  $r$  se mění. Systém vykazuje chaotické chování pro  $r = 28$  ale zobrazuje zamotané periodické orbity pro další hodnoty  $r$ . Například Obr. 1.

Podivné atraktory se objevují jak ve spojitých dynamických systémech (jako je Lorenzův systém), tak i v některých diskrétních systémech jako je Hénonovo zobrazení. Jiné diskrétní dynamické systémy mají odpuzující strukturu nazývanou Juliovy množiny, která tvoří hranici mezi oblastmi přitažlivosti pevných bodů. Juliovy množiny lze pokládat za podivné odpuzovače. Jak podivné atraktory, tak Juliovy množiny typicky mají fraktální strukturu.

## 2.4 Citlivost na počáteční podmínky

Citlivost k počátečním podmínkám znamená to, že dvě blízké trajektorie ve fázovém prostoru se s rostoucím časem rozbíhají. Jinak řečeno, malá změna v počátečních podmínkách vede po čase k velmi odlišnému výsledku. Systém se chová identicky, pouze když jsou jeho počáteční podmínky stejné. Spojený s citlivostí je takzvaný efekt motýlího křídla. Systém počasí může být natolik citlivý, že stačí mávnutí motýlího křídla na jedné straně planety a na druhé (za delší časový úsek) může vyvolat tornáda. Provedeme si zkoušku citlivosti na počáteční podmínky programem Matlab. Vezmeme si Lorenzovi rovnice a vyřešíme je pro různé počáteční podmínky. Potom porovnáme časové průběhy obou řešení.



Obr. 2 Rozdílné výsledky při různých počátečních podmínkách

Koeficienty Lorenzovy rovnice (2.3.1) jsou:  $\sigma = 10$ ,  $b = 8 / 3$  a  $r=28$ .

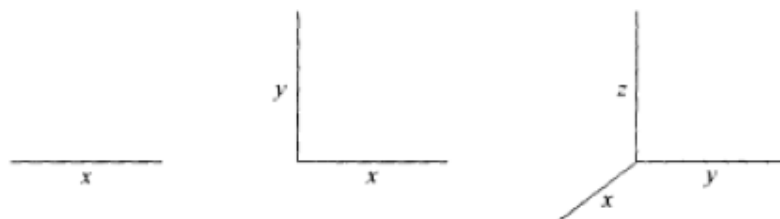
Počáteční podmínky:  $x_1(0)=8$ ,  $y_1(0)=3$ ,  $z_1(0)=4$  a  $x_2(0)=8$ ,  $y_2(0)=3,000000001$ ,  $z_2(0)=4$ .

Do grafu vyneseme jen jednu proměnnou kvůli přehlednosti. V grafu je vidět, že i když je odlišnost počátečních podmínek jen o 0,000000001, dojde během delšího časového úseku ke značným odchylkám.

# 3 DIMENZE

## 3.1 Euklidovská dimenze

Euklidovská dimenze je dimenze známá z klasické geometrie. Její hodnota pro dané těleso udává, jaký je minimální počet údajů pro popis polohy bodu náležícího tělesu. Například změříme-li délku tělesa, která se dá popsat úsečkou, měříme v jedné dimenzi. Dále si můžeme představit hřiště. K jeho základnímu geometrickému popisu používáme délku a šířku. To znamená, že již měříme ve dvojdimenzionálním prostoru.



Obr. 3 Euklidovské dimenze

## 3.2 Topologická dimenze

Topologická dimenze tělesa se určuje jako  $1 +$  Euklidovská dimenze nejjednoduššího tělesa, které dokáže rozdělit dané těleso na řadu menších těles. Taková přímka se skládá z nekonečně mnoha bodů. Vyjmeme-li z řady jeden bod, přímka se rozdělí na dvě. Vyjdeme-li z definice, topologická dimenze přímky se vypočte jako  $1+0=1$ . Jak to ale bude s povrchem? K rozdělení povrchu již bod nebude stačit. Musíme tedy vzít přímku. Podobně výpočet:  $1+1=2$  (přímka má Euklidovskou dimenzi 1) Topologická dimenze je vždy celé číslo a obvykle má stejnou hodnotu jako Euklidovská dimenze.



Obr. 4 Topologická dimenze (Vlevo-přímka, Vpravo- povrch)

Výpočet:  $1+0=1$

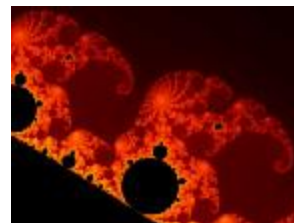
Výpočet:  $1+1=2$

## 3.3 Hausdorffova dimenze

Z předchozích definicí vyplývá závěr, že pro "běžné" útvary vyskytující se v okolním světě si vystačíme s dimenzemi 0, 1, 2 nebo 3. Proto bylo poměrně velkým překvapením, když byly objeveny zvláštní geometrické útvary, pro které toto rozdělení na celočíselné dimenze není dostatečné. Objekty popisované fraktální geometrií mají dimenzi neceločíselnou. Dimenzi fraktálních objektů nazýváme fraktální dimenzí či Hausdorffovou dimenzí, nebo se také nazývá objemová (krychlová) dimenze. Hodnota této dimenze (resp. míra rozdílu mezi fraktální dimenzí a dimenzí topologickou) potom udává úroveň členitosti daného objektu.

Některé tyto útvary nejsou jen abstraktní objekty vzniklé díky fantazii matematiků či umělců, ale mají své vzory přímo v okolní přírodě. Měřením délky geometricky hladké křivky, která má topologickou dimenzi rovnou jedné, dostaneme při pohledu v různých měřítkách vždy stejné konečné číslo. Měřením délky břehu ostrova (což je opět křivka s topologickou dimenzí rovnou jedné) se při zmenšování měřítka toto číslo stává nekonečně velkým. Pobřeží tedy v rovině zabírá více místa než hladká křivka. Nezabírá však všechno místo (přesněji řečeno, nevyplňuje celou rovinu). Jeho skutečná dimenze je tedy větší než topologická dimenze křivky (ta je rovna jedné) a současně je menší než topologická dimenze roviny (ta je rovna dvěma). Z toho jasně vyplývá, že dimenze takového útvaru není celočíslná. Hodnota Hausdorffovy dimenze udává, s jakou rychlostí délka těchto útvarů roste do nekonečna. Jestliže se bude Hausdorffova dimenze a topologická dimenze lišit velmi málo, bude takový objekt málo členitý. Bude-li Hausdorffova dimenze ostře větší než dimenze topologická, bude objekt velmi členitý.

Mezní hodnotou je případ, kdy je Hausdorffova dimenze o jedničku větší než dimenze topologická - tuto vlastnost má například hranice Mandelbroty množiny.



Obr. 5 Část Mandelbroty množiny

## 4 METODY MĚŘENÍ DIMENZÍ

### 4.1 Měření Hausdorffovy dimenze

Nejjednodušším příkladem bude zřejmě úsečka. Vytvoříme úsečku, která má jednotkovou délku. Nyní tuto úsečku rozdělíme na  $N$  dílů. To odpovídá tomu, jako bychom se na úsečku podívali s  $N$ -násobným zvětšením. Měřítka nové úsečky se tedy vypočítá takto:

$$s = \frac{1}{N}, \quad (4.1.1)$$

kde  $s$  je měřítko a  $N$  je počet dílů, na které se těleso rozdělí. Pro Hausdorffovu dimenzi obecně platí následující podmínka:

$$N \cdot s^D = 1, \quad (4.1.2)$$

Z toho vyplývá, že Hausdorffova dimenze se pro dané dělení  $N$  a dané měřítko  $s$  se vypočítá jako:

$$D \cdot \log s = -\log N, \quad (4.1.3)$$

$$D = \frac{\log N}{\log \frac{1}{s}}. \quad (4.1.4)$$

Pro úsečku rozdělenou na dvě části tedy platí:

$$D = \frac{\log N}{\log \frac{1}{s}} = \frac{\log 2}{\log \frac{1}{\frac{1}{2}}} = 1 . \quad (4.1.5)$$






Obr. 6 Rozdělení úsečky

Hausdorffova dimenze se tedy rovná dimenzi topologické. Z výše uvedené definice fraktálu tedy vyplývá, že úsečka není fraktál (pro fraktál musí být Hausdorffova dimenze ostře větší než dimenze topologická).

Dále  $D=2$  pro čtverec a  $D=3$  pro krychli.

Tento typ výpočtu dimenze lze aplikovat pouze na struktury, které jsou soběpodobné. Řekneme, že nějaká struktura je ryze soběpodobná, pokud ji lze rozdělit na několik částí, kde každá z těchto částí je zmenšená kopie celku.

Příklady hodnot dimenzí deterministických fraktálů:

Název	Prostorové znázornění	Dimenze
Sierpinského trojúhelník		$D=1,5850$
Lorenzův atraktor		$D=2,06$
Povrch plic		$D=2,97$

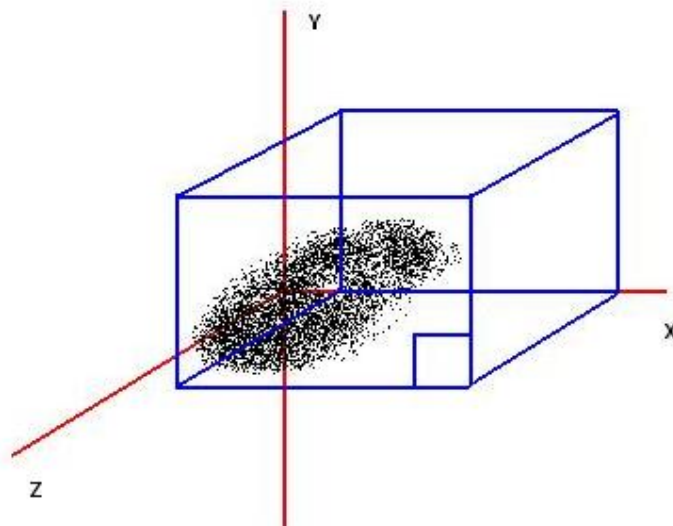
Tab. 4.1.1

## 4.2 Mřížková dimenze

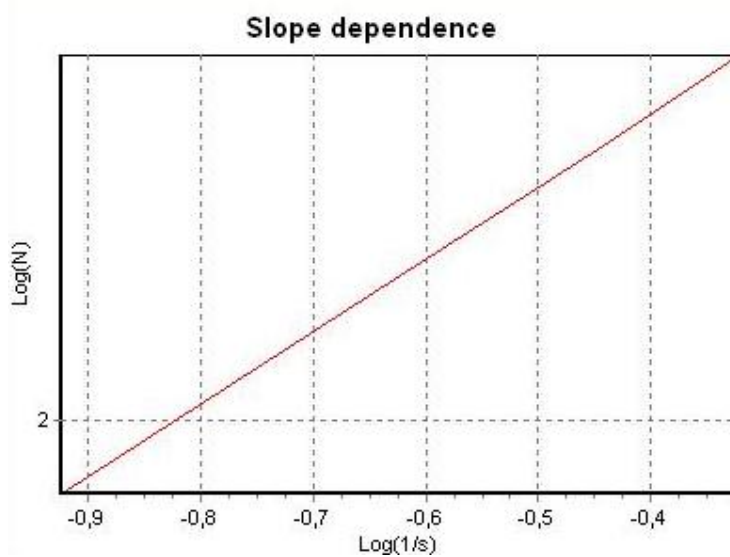
Narozdíl od dimenze soběpodobnostní, která je definována pouze pro ryze soběpodobné útvary, lze mřížkovou dimenzi aplikovat na libovolný útvar.

Zjišťování dimenze: zadaný útvar se umístí na mřížku s velikostí buněk  $s$  a spočítá se, kolik buněk obsahuje alespoň část útvaru. Tím se získá číslo  $N$ , které samozřejmě závisí na volbě buněk  $s$ . Dále se postupně volí menší  $s$  a počítá se  $N(s)$ . Potom se hodnoty vynesou do log/log diagramu (přesněji do  $\log(N(s))/\log(1/s)$  digramu) a získané body se aproximují přímkou. Mřížková dimenze je směrnice této přímky.





Obr. 7 Příklad určování mřížkové dimenze



Obr. 8 Příklad výpočtu mřížkové dimenze ( $D=1.84$ )

Výhodou mřížkové metody je její snadná realizace pomocí výpočetní techniky (konkrétně v jazyce C++ Builder). Více o realizaci v kapitole 6.5

### 4.3 Kaplan Yorkeho dimenze

Samotný výpočet dimenze pomocí této metody je velice jednoduchý. Vychází ze znalosti Lyapunovových exponentů. Abychom však exponenty získali, je třeba provést ortonormalizaci. Pro počítačové řešení problémů se budeme zabývat Gram-Schmidtovou metodou.

$$D_{KY} \equiv j - \frac{I_1 + \dots + I_j}{I_{j-1}}. \quad (4.1.1)$$

Kde hodnota  $j$  znamená maximální počet Lyapunovových exponentů.

## 5 NUMERICKÉ METODY

### 5.1 Numerické řešení diferenciálních rovnic

Diferenciální rovnice je matematická rovnice, ve které jako proměnné vystupují derivace funkcí. Za řešení (integrál) diferenciální rovnice považujeme funkci, která má příslušné derivace a vyhovuje dané diferenciální rovnici. Tedy řešením soustavy diferenciálních rovnic je množina funkcí s derivacemi potřebného řádu, které vyhovují všem rovnicím dané soustavy.

#### Řešení dělíme na:

Obecné

Partikulární – získané přiřazením určité číselné hodnoty každé integrační konstantě obecného řešení.

Singulární – vyskytují se jen u některých rovnic nebo v některých bodech.

Partikulární řešení můžeme v případě jednoduchých diferenciálních rovnic vypočítat analyticky. Nicméně ve velkém množství případů je analytické řešení příliš obtížné a diferenciální rovnice se řeší numericky. Existuje více metod, jak získat numericky dané řešení.

V našem případě se budeme zabývat řešením diferenciálních rovnic v maticovém zápisu:

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \cdot (x \ y \ z) \cdot \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}. \quad (5.1.1)$$

A dále také tímto typem rovnic.

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \frac{1}{2} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \cdot \left( \left| \begin{pmatrix} w_1 & w_2 & w_3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + 1 \right| - \left| \begin{pmatrix} w_1 & w_2 & w_3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + 1 \right| \right). \quad (5.1.2)$$

Je patrné, že složitost pro analytické řešení je velická. Pro řešení vybereme Eulerovu numerickou metodu, která je relativně snadná pro počítačové zpracování a její chyba je přijatelná.

## 5.2 Eulerova metoda:

Je to nejjednodušší z numerických metod pro řešení diferenciálních rovnic. Publikoval ji Leonhard Euler již v roce 1768.

Obvykle je dána rovnice či soustava a počáteční podmínky:

$$y'(t) = f(t, y), \quad (5.2.1)$$

$$y(t_0) = y_0. \quad (5.2.2)$$

A bývá také zadán krok  $h$  (zde můžeme výrazně ovlivnit přesnost řešení). Řešení se dá zapsat ve tvaru:

$$y(i+1) = y(i) + dy(i) \cdot h \quad (5.2.3)$$

Pro soustavu tří rovnic tedy můžeme psát:

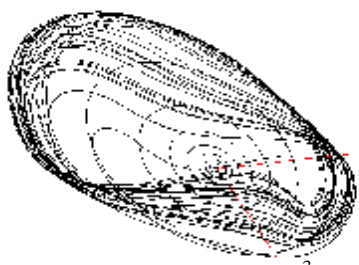
$$\begin{aligned} x(i+1) &= x(i) + dx(i) \cdot h, \\ y(i+1) &= y(i) + dy(i) \cdot h, \\ z(i+1) &= z(i) + dz(i) \cdot h. \end{aligned} \quad (5.2.4)$$

Za  $dx(i)$  dosadíme řešení dané rovnice se zadáním  $x(i), y(i), z(i)$ .

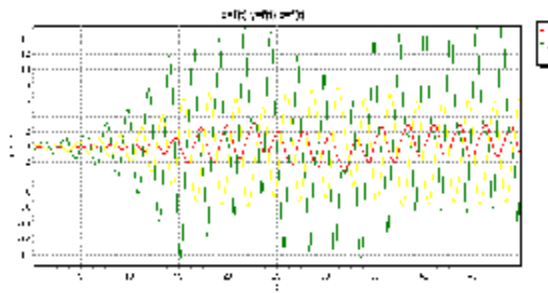
Zkusíme si výpočet pro rovnici (5.1.2). Parametry matic jsou například následující:

$$A: \begin{pmatrix} 0,287 & 1 & 0 \\ 3,214 & 0 & -1 \\ 17,358 & 0 & 0 \end{pmatrix} W: \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} B: \begin{pmatrix} -3,859 \\ 3,86 \\ -52,08 \end{pmatrix}. \quad (5.2.5)$$

Počáteční podmínky:  $x(0)=0,1$ ,  $y(0)=0,1$  a  $z(0)=0,1$ . Při grafickém řešení soustava vypadá následovně:



Obr. 9 Zobrazení v prostoru  $R^3(x,y,z)$



Obr. 10 Vývoj proměnných v čase

Další z možností jak řešit diferenciální rovnice je užití Runge-Kuttových metod.

## 5.3 Metody Runge-Kutta

Obecně lze tyto metody zapsat následovně:

$$y_{n+1} = y_n + h \sum_{i=1}^{i-1} w_i k_i, \quad (5.3.1)$$

$$k_i = f(t + a_i h, y_n + h \sum_{j=1}^{i-1} b_{ij} k_j). \quad (5.3.2)$$

Koeficienty jsou vypočteny tak, aby metoda řádu  $p$  odpovídala Taylorovu polynomu stejného řádu. (dá se tedy říct, že Eulerova metoda je Runge-Kuttova metoda prvního řádu). Nejčastěji se používá metoda 4. řádu. Její náročnost na početní výkon není velká a přesnost je dostačující.

## 5.4 Porovnání přesnosti metod

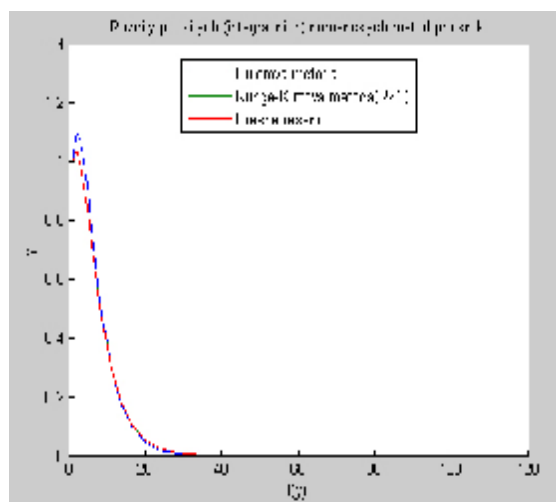
Porovnáme Eulerovu metodu a Runge-Kuttovu metodu 2. řádu s analytickým řešením. Vybereme si tedy diferenciální rovnici, kde známe analytické řešení:

$$y'(t) + 2y(t) = 3 \cdot \exp(-4t). \quad (5.4.1)$$

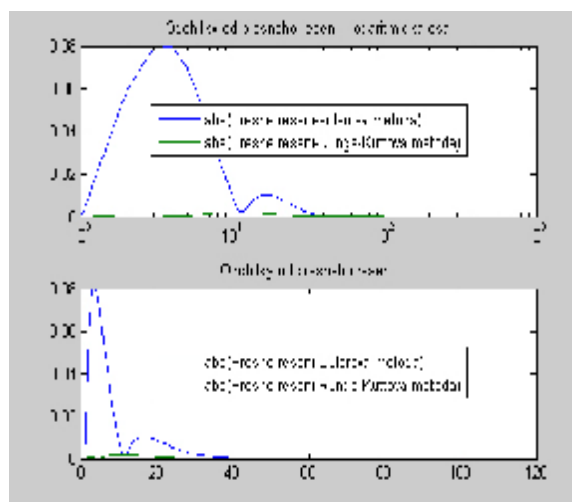
Analytické řešení této rovnice je:

$$y(t) = 2 \cdot \exp(-2t) - 1,5 \cdot \exp(-4t). \quad (5.4.2)$$

Nejprve si zvolíme krok obou metod  $h=0,1$ .

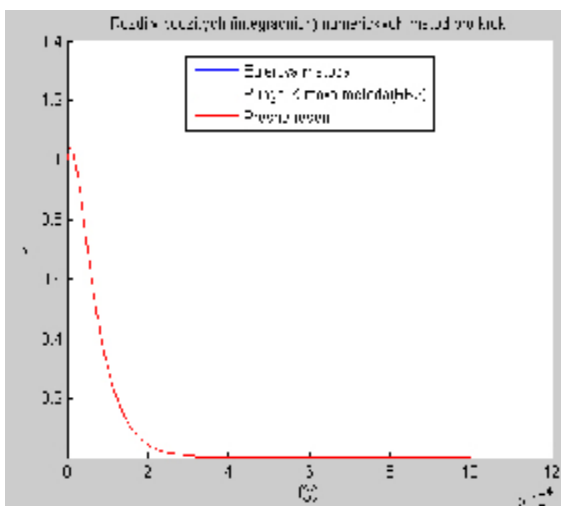


Obr. 11 Rozdíly v řešení pro  $h=0,1$

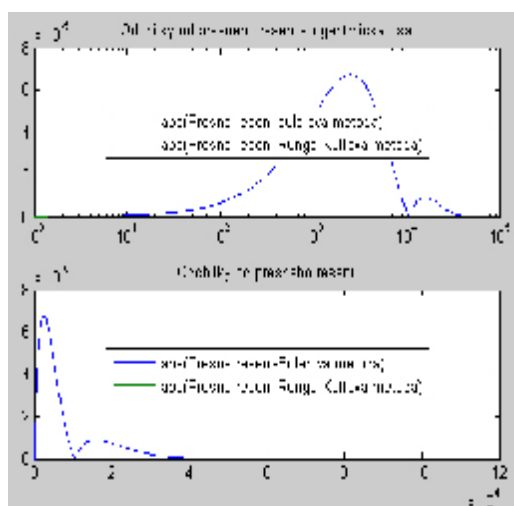


Obr. 12 Vynesení odchylek

Z grafů je patrné, že Runge-Kuttova metoda má mnohem vyšší přesnost pro daný krok. Nyní řešení vyšetříme pro krok  $h=0,0001$



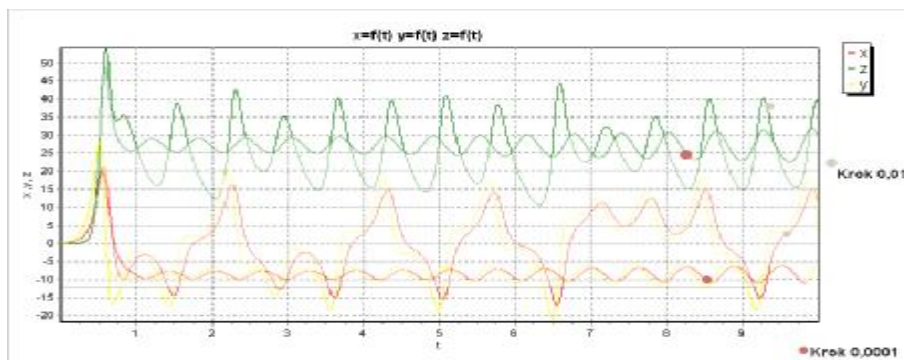
Obr. 13 Rozdíly v řešení pro  $h=0,0001$



Obr. 14 Vynesení odchylek

Vidíme, že nejvyšší chyba Eulerovy metody je řádově  $7E-5$ , což je proti Runge-Kuttově metodě 2. řádu vysoká odchylka, avšak zanedbatelná (musíme brát ohled i na rychlost řešení).

U Eulerovy metody tedy může nastat problém při špatné volbě kroku  $h$ . Například pro Lorenzovy rovnice má volba zásadní vliv na dané řešení:



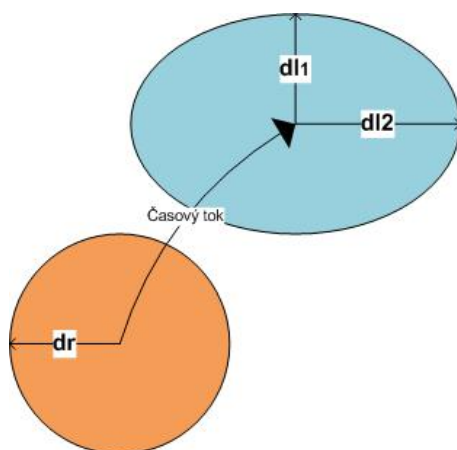
Obr. 15 Porovnání vlivu volby kroku  $h$  pro Lorenzovy rovnice

## 6 LYAPUNOVY EXPONENTY

Jedno z měřítek citlivosti na počáteční podmínky jsou Lyapunovovy exponenty  $\lambda_n$ . Je to průměrná hodnota divergence (nebo konvergence) dvou sousedních trajektorií. V matematice je Lyapunovův exponent nebo Lyapunovův charakteristický exponent dynamického systému veličina, která charakterizuje hodnotu separace nekonečně blízkých trajektorií. Tedy k popisu chování trajektorií v okolí libovolné trajektorie  $\Gamma$  se používají Lyapunovovy exponenty, které jsou zobecněním vlastních čísel nebo multiplikátorů. Lyapunovovy exponenty (LE) jsou reálná čísla, která lze výhodně použít pro klasifikaci nechaotických i chaotických atraktorů. Je-li systém v nestabilním stavu lze ukázat, že se dvě vzájemně blízké trajektorie vzdalují rychleji než polynomiálně. Vzdálenost  $l$  dvou bodů na blízkých trajektoriích lze aproximovat vztahem

$$l(t) = d_r e^{l_t}, \quad (6.1)$$

kde  $\lambda$  je tzv. lokální Lyapunovův exponent.



Obr. 13 Transformace kruhové trajektorie na eliptickou

Volíme-li v tečném prostoru jen jeden vektor, dostaneme jednorozměrný Lyapunovův exponent. Ve vícerozměrném fázovém prostoru je definováno tzv. globální spektrum Lyapunovových exponentů (každé stavové proměnné odpovídá jeden). Pro jeden zvolený vektor v tečném prostoru platí:

$$l_i = \lim_{t \rightarrow \infty} \frac{1}{t} \ln\left(\frac{dl_i(t)}{dr}\right). \quad (6.2)$$

Známe-li toto spektrum, pak můžeme konstatovat, že jsou-li všechny exponenty nekladné, jedná o stabilní chování systému a je-li alespoň jeden exponent kladný, pak se systém chová chaoticky.

Lyapunovovy exponenty jsou řazeny k časovým kvantifikátorům chaosu. Více je popsáno v tabulce 6.1.1:

Atraktor	Lyapunovy exponenty
Stabilní bod	$0 > I_1^1 \geq \dots \geq I_N^1$
Limitní cyklus	$I_1^1 = 0, 0 > I_3^1 \geq \dots \geq I_N^1$
Prstenec se 2 periodami	$I_1^1 = I_2^1 = 0, 0 > I_3^1 \geq \dots \geq I_N^1$
Prstenec s k periodami	$I_1^1 = \dots = I_k^1 = 0, 0 > I_{k+1}^1 \geq \dots \geq I_N^1$
Chaotický	$I_1^1 > 0, 0 \geq I_2^1 \geq \dots \geq I_N^1, \sum_{i=1}^N I_i^1 < 0$
Hyperchaotický	$I_1^1 \geq I_2^1 > 0, 0 \geq I_3^1 \geq \dots \geq I_N^1, \sum_{i=1}^N I_i^1 < 0$

Tabulka 6.1.1 Identifikace atraktoru podle významu jeho Lyapunovových exponentů.

## 7 ORTOGONÁLNÍ A ORTONORMÁLNÍ BÁZE

V lineární algebře, dva vektory  $\mathbf{v}$  a  $\mathbf{w}$  v prostoru s definovaným skalárním součinem jsou ortonormální, pokud jsou ortogonální a mají jednotkovou délku, tedy jejich skalární součin je roven nule a jejich norma (velikost) je rovna 1. Báze, kde jsou všechny vektory navzájem ortonormální se nazývá ortonormální báze. Ortogonální bázi vektorového prostoru  $V$  vybaveného skalárním součinem nazýváme takovou bázi, jejíž vektory jsou navzájem kolmé, tzn. jejich skalární součin je nulový. Mají-li navíc všechny vektory báze jednotkovou velikost (neboli jsou normovány k 1), mluvíme o ortonormální bázi. Je-li  $B \equiv \{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n\}$  ortonormální bázi, pak platí pro skalární součin libovolných dvou jejích prvků  $\mathbf{b}_i, \mathbf{b}_j$  vztah  $\mathbf{b}_i \cdot \mathbf{b}_j = \delta_{ij}$ . Symbol na pravé straně, tzv. Kroneckerovo delta, je roven 1 pro  $i = j$ , jinak je roven 0.

Nejpoužívanější ortonormální bázi v prostoru  $\mathbb{R}^3$  je báze tvořená vektory  $\mathbf{i} = (1, 0, 0)$ ,  $\mathbf{j} = (0, 1, 0)$ ,  $\mathbf{k} = (0, 0, 1)$ .

### 7.1 Gram-Schmidtova ortonormalizace

Proces vytváření ortogonální báze se nazývá Gram-Schmidtova ortogonalizace (Gram-Schmidtův ortogonalizační proces). Kde vzít ortogonální bázi? Z každé báze  $\zeta = (\mathbf{f}_1, \dots, \mathbf{f}_k)$  prostoru  $V$  lze sestavit ortogonální bázi

$$\varepsilon = (\mathbf{e}_1, \dots, \mathbf{e}_n). \quad (7.1.1)$$

Na začátku si všimneme, že  $\mathbf{f}_1 \neq 0$ , a položíme  $\mathbf{e}_1 = \mathbf{f}_1$ . Předpokládejme, že máme ortogonální vektory  $\mathbf{e}_1, \dots, \mathbf{e}_k$  takové, že pro každé  $i \in \{1, \dots, k\}$  je vektor  $\mathbf{e}_i$  lineární kombinací vektorů  $\mathbf{f}_1, \dots, \mathbf{f}_i$ . Najdeme koeficienty  $\alpha_1, \dots, \alpha_k$  tak, aby

$$\mathbf{e}_{k+1} = \mathbf{f}_{k+1} - \alpha_1 \mathbf{e}_1 - \dots - \alpha_k \mathbf{e}_k \quad (7.1.2)$$

byl ortogonální k  $\mathbf{e}_1, \dots, \mathbf{e}_k$ .

Jelikož pro  $i \in \{1, \dots, k\}$  by mělo platit:

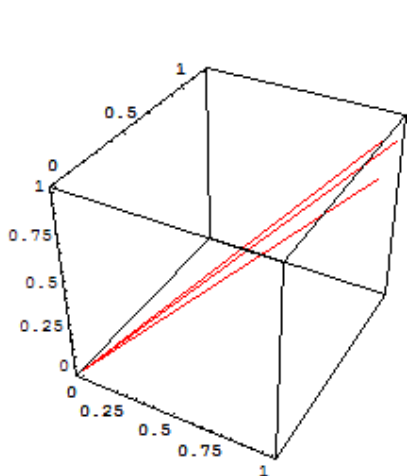
$$0 = (e_k + 1, e_i) = (f_k + 1 - \alpha_1 e_1 - \dots - \alpha_k e_k, e_i) = (f_k + 1, e_i) - \alpha_i \|e_i\|^2, \quad (7.1.3)$$

stačí položit:

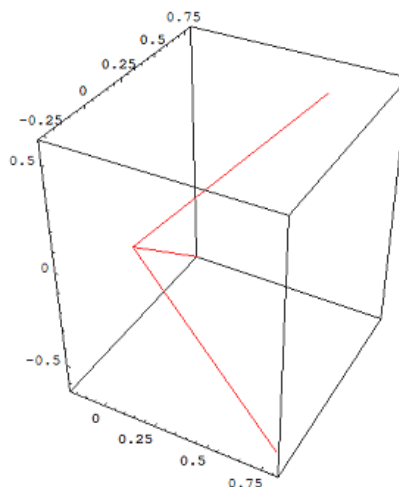
$$\alpha_i = (f_k + 1, e_i) / \|e_i\|^2. \quad (7.1.4)$$

Normalizací vektorů báze  $\varepsilon = (e_1, \dots, e_n)$  dostaneme ortonormální bázi  $\Gamma = (g_1, \dots, g_n)$  s vektory

$$g_1 = \frac{e_1}{\|e_1\|}, \dots, g_n = \frac{e_n}{\|e_n\|}. \quad (7.1.5)$$



Obr. 14 Vizualizace vektorů (resp. Báze)



Obr.15 Vizualizace vektorů po Gram-Schmidtově ortonormalizaci

Příklad: Necht'  $A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$ . Budeme hledat ortogonální bázi  $\mathbf{R}^3$  vzhledem ke skalárnímu součinu  $(x, y)_A = x^T A y$ . Bázi sestavíme ortonormalizačním procesem ze standardní báze  $S = (s_1^I, s_2^I, s_3^I)$ .

$$e_1 = s_1^I = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

Položíme tedy  $e_2 = s_2^I - \alpha e_1$  a určíme aby platilo

$$0 = (e_1, e_2)_A = e_1^T A s_2^I - \alpha e_1^T A e_1 = -1 - 2\alpha, \quad (7.1.6)$$

odtud platí

$$\alpha = -\frac{1}{2}a, \quad (7.1.7)$$

$$e_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} - \left(-\frac{1}{2}\right) \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ 1 \\ 0 \end{bmatrix}. \quad (7.1.8)$$

Dále položíme  $e_3 = s_3^I - \alpha_1 e_1 - \alpha_2 e_2$  a určíme  $\alpha_1, \alpha_2$ , aby platilo:

$$0 = (e_1, e_3)_A = e_1^T A s_3^I - \alpha_1 e_1^T A e_1 - 2\alpha_1, \quad (7.1.9)$$

$$0 = (e_2, e_3)_A = e_2^T A s_3^I - \alpha_2 e_2^T A e_2 = -1 - \frac{3}{2}\alpha_2. \quad (7.1.10)$$

Řešením této soustavy dostaneme  $\alpha_1 = 0, \alpha_2 = -\frac{2}{3}a$ .



$$e_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - 0 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} - \left(-\frac{2}{3}\right) \begin{bmatrix} \frac{1}{2} \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \\ 1 \end{bmatrix}. \quad (7.1.11)$$

## 8 PRINCIP NEURČITOSTI - HEISENBERG

Tento princip říká, že přesnost měření je v mikrosvětě omezena. Podle klasické mechaniky je možné určit polohu a hybnost v každém časovém okamžiku současně. Heisenbergův princip neurčitosti však tvrdí, že v mikrosvětě je nemožné zjistit současně polohu a hybnost elektronu absolutně přesně, a že je to dáno vlastnostmi přírody. Jestliže budeme chtít určit hybnost, ztratíme veškerou informaci o poloze, a naopak. Tento princip v sobě odráží dualitu vlna-částice. Poloha je částicová vlastnost, kdežto vlna je v prostoru rozložena spojitě. Pokud budeme měřit polohu, bude se chovat elektron jako částice a nezměříme již vlnové vlastnosti, což je právě hybnost. Heisenbergův princip neurčitosti říká, že čím více toho zjistíme o jedné veličině, tím méně můžeme zjistit o druhé. Konstanta úměrnosti je známá Planckova konstanta. Její velikost právě způsobuje, že se tento jev neprojevuje v makrosvětě, ale jen u částic, jako jsou elektrony, fotony a podobně.

Heisenbergův princip neurčitosti se dá formálně vyjádřit vztahem:

$$\Delta x \Delta p \geq \frac{\hbar}{2}. \quad (8.1)$$

Součin neurčitosti v poloze ( $x$ ) a neurčitosti v hybnosti ( $p$ ) je tedy větší nebo roven polovině Planckovy konstanty.

Hybnost a poloha nejsou jediné dvě veličiny, pro které toto omezení platí. Podobné omezení platí i pro různé další dvojice fyzikálních veličin. Souvisí to s tím, že v kvantové mechanice je stav systému popsán zcela jinak než v mechanice klasické (kde je určen souborem poloh a hybností všech částic) a fyzikální veličiny jsou do značné míry definovány způsobem jejich měření. Existují dvojice veličin, které nikdy (na žádném stavu systému) nemohou dát obě zároveň jednoznačně předpověditelné výsledky měření. Čím jistější je výsledek měření jedné veličiny, tím neurčitější je výsledek měření druhé.

Princip neurčitosti je důležitý v případě, že operátory dvou pozorovatelných veličin spolu nekomutují.

Dále platí pro určení času a energie:

$$\Delta t \Delta E \geq \frac{\hbar}{2}, \quad (8.2)$$

úhel a moment setrvačnosti objektu:

$$\Delta O_i \Delta J_i \geq \frac{\hbar}{2}, \quad (8.3)$$

pro dvě ortogonální složky operátoru celkového momentu setrvačnosti:

$$\Delta J_i \Delta J_j \geq \frac{\hbar}{2} |\langle J_k \rangle|, \quad (8.4)$$

kde  $i, j, k$  jsou různé a  $J_i$  označuje úhlový moment vzhledem k ose  $x_i$ .

Princip neurčitosti má přímočaré matematické odvození. Klíčovým krokem je uplatnění Cauchy-Schwarzovy nerovnosti, jeden z nejužitečnějších téoremů lineární algebry.

## 9 KARDIOVASKULÁRNÍ RYTMY

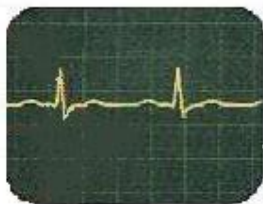
Rytmické změny krevního tlaku, srdečního poměru a další kardiovaskulární hodnoty signalizují význam dynamických hledisek v pochopení kardiovaskulárních rytmů. Několik studií poukazuje na skutečnost, že jisté srdeční arytmie jsou příkladem chaosu. Toto je důležité, protože to může pomoci určit léčebné postupy. Patří sem například srdeční arytmie, fibrilace síní, bradykardie, zrychlení srdeční činnosti, dutinové arytmie a jiné.

Komorové arytmie, jak fibrilace komor, tak komorová tachykardie, jsou nejzávažnější z chorob. Tyto arytmie způsobují mnoho úmrtí. Srdeční nestabilita může být chápána jako spontánní asynchronní stahování srdečních vláken svalu. Fibrilace komor se stává spontánní a neregulérní zmatek srdečního rytmu. Toho může rychle postupovat, až se rytmus srdce stane neslučitelný s životem.

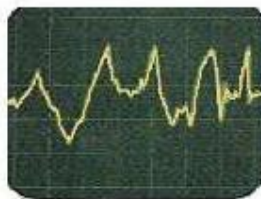
Srdeční rytmus je jedním z nejlepších ukazatelů arytmiických událostí nebo může vést k náhlému úmrtí po infarktu myokardu. Srdeční rytmus je částečně řízen autonomním nervstvem. Autonomní systém (nedá se ovlivnit vůlí člověka) - řízení autonomních funkcí má povahu reflexní, nezávisle na našem vědomí. Autonomní nervový systém je rozdělen do podsystémů; soucitné (SNS) a parasympatické (PNS) nervové soustavy. Krátkodobá proměnlivost je zprostředkována parasympatickým nervovým systémem, zatímco dlouhodobá proměnlivost oběma: soucitnou (SNS) a parasympatickou (PNS) nervovou soustavou. Jak jistě víme, srdeční tep se může značně lišit dokonce za nepřítomnosti fyzického nebo duševního tlaku.

Několik studií poukazuje na vztah mezi srdečními arytmii a chaosem. Toto souvisí s deterministickými charakteristickými rysy některých z těchto arytmií. Klinické arytmie mají největší potenciál pro léčebné aplikace teorie chaosu na aperiodické tachykardie, včetně sínových a komorových fibrilací. Takovýto postup při vyhodnocování by mohl být implementovaný do stimulatorů. Tím bychom se mohli vyhnout se například fibrilaci komor.

Existuje již několik zajímavých srovnání mezi dynamickými charakteristikami zdravých osob a pacientů s vysokým rizikem náhlé srdeční fibrilace. Srdce s vysokým rizikem náhlé srdeční arytmie vykazuje chaotický průběh signálů. Tyto metody by mohly představovat důležitý diagnostický nástroj pro klinické účely.



Obr. 16 Normální ECG



Obr. 17 ECG při infarktu

Můžeme situaci zjednodušit tak, že srdce je určitý typ oscilátoru. Je-li oscilátor, platí pro něj nelineární dynamické rovnice. Problém je však v tom, že tyto rovnice obsahují mnoho stavových proměnných – tedy existuje mnoho faktorů ovlivňujících rytmus srdce. V klinických podmínkách ale můžeme situaci zjednodušit a omezit se na méně proměnných. Existuje šance, že půjde do jisté míry (tedy s krátkým časovým předstihem a omezenými počátečními podmínkami) předpovídat srdeční rytmus. Takovýto přístroj by se stal velice důležitým pomocníkem na operačních sálech. Operatéri by se mohli na určité situace připravovat s časovým předstihem.

# 10 MOZEK A CHAOS

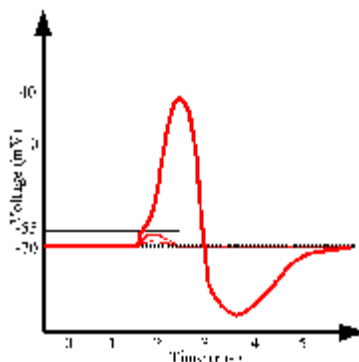
## 10.1 Neuron

A neuron je zvláštní elektricky- aktivní buňka, která může přenášet pulsy elektrického napětí podél své délky. Každý puls vypadá jako "špička" na křivce elektrického napětí v čase. Pulsy jsou vytvořené jako záplavy iontů ženoucí se sem a tam skrz dvě skupiny kanálů položených neuronovou membránou: ty s rychlými změnami a ty s pomalými. Samozřejmě, opravdový žijící neuron je komplikovaná záležitost. To však ukazuje, že tok elektrické energie membránou může být dobře modelován systémem tří vzájemně závislých diferenciálních rovnic.



Obr. 19 Obrázek neuronu.

Tyto popisují přepnutí v čase, rychlost, u které elektrické napětí projde přes plochu membrány. Existují však další dvě proměnné: rychlý proud a pomalý proud kanálem.



Obr. 19 Různé průběhy akčního potenciálu

Membrána, na které převažují chemicky řízené iontové kanály (tj. postsynaptická membrána), je drážditelná chemickými podněty, především mediátorem. Odpověď na toto podráždění může být dvojí: Depolarizace - zvýšení permeability pro sodné, draselné a chloridové ionty, nebo Hyperpolarizace - zvýšení permeability pro draselné a chloridové ionty.

To způsobí lokální změnu membránového potenciálu (místní podráždění) membrány. Chemicky řízené iontové kanály jsou rozhodující pro vzrušivost neuronu, díky nim dochází k modulaci signálu. Membrána, na které převažují elektricky řízené iontové kanály, reaguje na podráždění podle zákona „vše nebo nic“. Buď reaguje vzruchem, nebo ne, a pokud ano, tak s nejvyšší možnou intenzitou. Ke vzniku vzruchu musí dojít k místní depolarizaci membrány, a náhlému rychlému poklesu membránového potenciálu. Vlna depolarizace membrány, která nastává otevřením iontových kanálů, šířící se postupně po povrchu neuronu, se nazývá akční potenciál a je podstatou přenosu informací neurony.

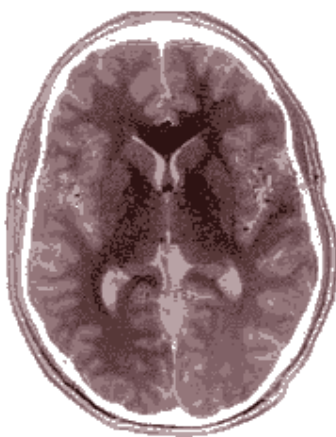
Takovýto jednoduchý model přesně reprodukuje chování. Model tedy dokáže předpovídat chování izolovaného neuronu pozorovaného v laboratoři.

## 10.2 Mozek

Základním prvkem, z něhož se skládá stavba mozku, je nervová buňka či neuron. Lidský mozek je tvořen ze 100 miliard neuronů, které jsou jako všechny buňky organismu vymezeny membránou. Liší se však svými velmi složitými tvary.

Jeden neuron vytváří synapse průměrně s 10 000 jinými neurony. V mozku člověka tedy existuje 100 miliard krát 10 000 synapsí, tedy asi milion miliard synapsí. Jedná se o biochemické reakce, které produkují i elektrické signály.

Existuje-li matematický model jednoho neuronu, šlo by vytvořit matematický model celého mozku. Z pohledu teorie chaosu je mozek velice složitý systém, který reaguje citlivě na změny jednotlivých neuronů. Vidíme tedy, že je velice citlivý na počáteční podmínky. Znamená to, že pravděpodobně by šlo vypočítat (kdybychom znaly všechny stavy synapsí a neuronů), jak se mozek rozhodne. Neznamená to ale bohužel, že budeme moci přesně určit chování daného jedince. Protože každý další okamžik se stává podmínkou pro změnu synapsí. Bude však existovat určitá pravděpodobnost, že jedinec bude reagovat dle výpočtů. Čím méně informací budeme vědět, tím menší bude pravděpodobnost predikce.



Obr. 20 Magnetická rezonance mozku

## 11 VZORKOVÁNÍ

Pro většinu aplikací je důležité zpracovávat signál digitálně. Analogově digitální převodník (zkratky A/D, v angličtině i ADC) je elektronická součástka určená pro převod spojitého (neboli analogového) signálu na signál diskretní (neboli digitální). Důvodem tohoto převodu je umožnění zpracování původně analogového signálu na číslicových počítačích. Mezi nimi v současnosti převažují digitální signální procesory DSP, které jsou právě na zpracování takových signálů specializované. V digitální podobě se také dají signály daleko kvalitněji zaznamenávat a přenášet. Opačný převod z digitálního signálu na analogový zajišťuje D/A převodník.

Pro správné vzorkování musí být splněn Shannon-Kotelnikovův vzorkovací teorém. Podle něj platí:

$$\frac{f_{vz}}{2} \geq f_{signálu} \quad (11.1)$$

Pro splnění této podmínky je důležité znát spektrum signálu, ze kterého se dá stanovit potřebná rychlost převodníku. Vidíme, že je nutný nejprve teoretický rozbor problému.

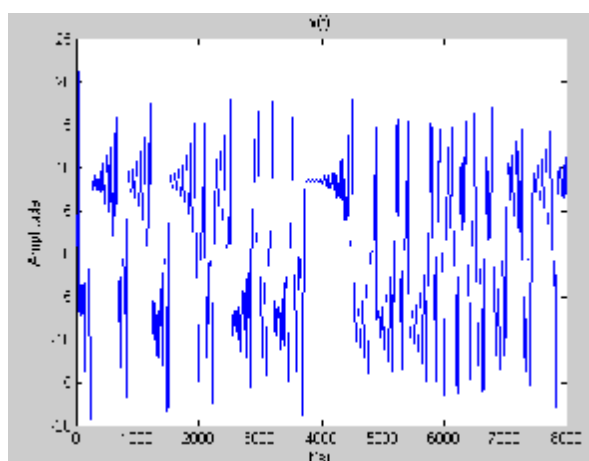
Pro simulaci si vybereme známý Lorenzův atraktor. Ten se tvoří třemi diferenciálními rovnicemi.

Pro určení vzorkovací frekvence, budeme vycházet z předpokladu, že jedna jednotka je jedna sekunda. Je-li integrační krok například  $h=0,003$  (tedy vzdálenost vzorků – jedná se o diskrétní body, ne o analogový signál) platí:

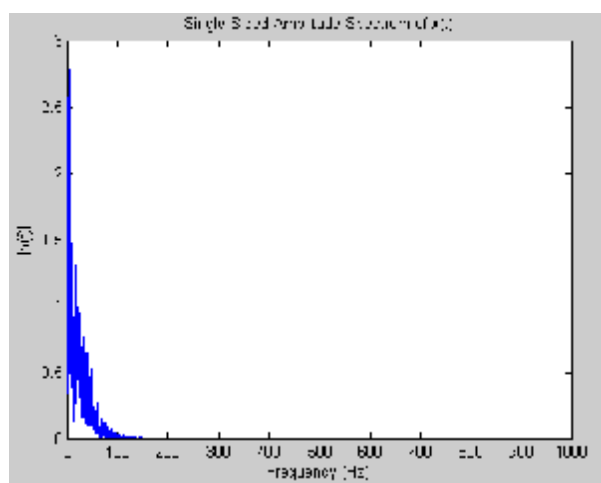
$$\frac{1}{2} f_{vz} = h \Rightarrow f_{vz} \geq 666\text{Hz} \quad (11.2)$$

Zmenšíme-li však krok  $h=0,001$  platí:

$$f_{vz} \geq 2000\text{Hz} \quad (11.3)$$



Obr. 21 Stavová proměnná x v čase t

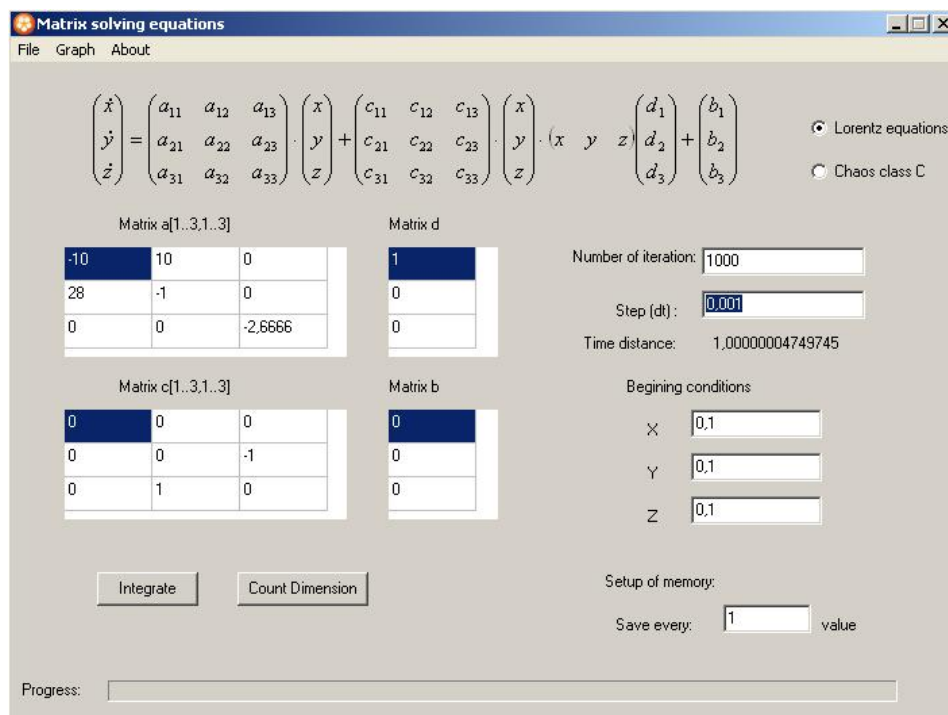


Obr. 22 Spektrum Lorenzova atraktoru pro stavovou proměnnou x

Výsledné spektrum bude obdobné pro další stavové proměnné. V simulaci byl použit integrační krok  $h=0,01$ . Ze získaného spektra se dá však vzorkovací frekvence ještě upravit.

# 12 PROGRAMOVÁ REALIZACE

Program je napsán pro operační systém Microsoft Windows v jazyce C++ Builder 6.0, numericky integruje systém 3 diferenciálních rovnic, pracuje se vstupními daty uloženými v souborech, ukládá do souboru a vizualizuje data pomocí grafických knihoven OpenGL a TChart. Pro spuštění programu jsou zapotřebí mít v systému nainstalované knihovny Borland C++.



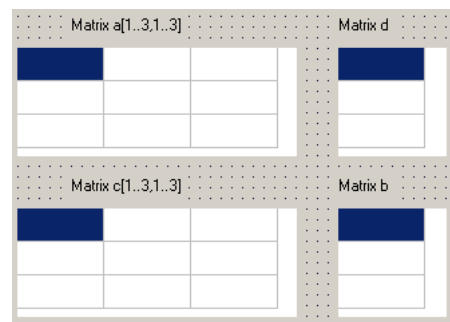
Obr. 23 Ukázka hlavního okna programu

## 12.1 Programové řešení numerické integrace

Nejdůležitější součástí programu je numerické řešení diferenciálních rovnic. Ty jsou v programu zapsané pomocí matic. Každé políčko matice dostane svoji vlastní proměnnou pro jednodušší operace. Proměnná je typu double. Jedná se o 64 bitové racionální číslo s dvojitou přesností.

Ukázka zdrojového kódu:

```
//prevod matic na konstanty
a11= StrToFloat(MatA->Cells[0][0]);
a21= StrToFloat(MatA->Cells[0][1]);
a31= StrToFloat(MatA->Cells[0][2]);
a12= StrToFloat(MatA->Cells[1][0]);
a22= StrToFloat(MatA->Cells[1][1]);
a32= StrToFloat(MatA->Cells[1][2]);
a13= StrToFloat(MatA->Cells[2][0]);
a23= StrToFloat(MatA->Cells[2][1]);
a33= StrToFloat(MatA->Cells[2][2]);
```



Obr. 24 Ukázka matic

Tímto způsobem pokračujeme a nastavíme i jiné parametry. Před samotnou integrací Eulerovou metodou (více o metodě v předchozí kapitole) musíme nastavit do matic všechny hodnoty, i když jen nulové, aby nedošlo k výpočtům s nahodilou hodnotou z operační paměti. Výsledky řešení ukládáme také do matice, protože využití datového typu pole není možné vzhledem k velikosti. Jednodušší by bylo ukládat výsledky do objektu typu TMemor, které však při zápisu ukládá na pevný disk, což výrazně zhoršuje časovou náročnost.

Pro výsledky výpočtu jsou zvoleny datové typy long double (80 bitové velmi dlouhé racionální číslo). Celý výpočet se provádí ve while cyklu. Dále jsou pomocí podmínek vybrány rovnice.

Máme tedy zadané dva typy rovnic v maticovém tvaru. Abychom algoritmus urychlili, je důležité provést násobení matic a do programu zapsat samotné řešení.

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \cdot \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \quad (12.1.1)$$

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \frac{1}{2} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \cdot \left( \left| \begin{pmatrix} w_1 & w_2 & w_3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + 1 \right| - \left| \begin{pmatrix} w_1 & w_2 & w_3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + 1 \right| \right) \quad (12.1.2)$$

Pro rovnici 13.1.1 je řešení:

$$\begin{pmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} \end{pmatrix}^T = \begin{pmatrix} a_{11} \cdot x + a_{12} \cdot y + a_{13} \cdot z + x^2 \cdot d_{11} + x \cdot d_{12} \cdot y + x \cdot d_{13} \cdot z + y^2 \cdot d_{21} + y \cdot d_{22} \cdot z + z^2 \cdot d_{31} + z \cdot d_{32} \cdot y + z^2 \cdot d_{33} + b_1 \\ a_{21} \cdot x + a_{22} \cdot y + a_{23} \cdot z + x^2 \cdot d_{21} + x \cdot d_{22} \cdot y + x \cdot d_{23} \cdot z + y^2 \cdot d_{22} + y \cdot d_{23} \cdot z + z \cdot d_{31} \cdot x + z \cdot d_{32} \cdot y + z^2 \cdot d_{33} + b_2 \\ a_{31} \cdot x + a_{32} \cdot y + a_{33} \cdot z + x^2 \cdot d_{31} + x \cdot d_{32} \cdot y + x \cdot d_{33} \cdot z + y^2 \cdot d_{32} + y \cdot d_{33} \cdot z + z \cdot d_{31} \cdot x + z \cdot d_{32} \cdot y + z^2 \cdot d_{33} + b_3 \end{pmatrix} \quad (12.1.3)$$

Pro rovnici 13.1.2 je řešení:

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{pmatrix} = \begin{pmatrix} a_{11} \cdot x + a_{12} \cdot y + a_{13} \cdot z + \frac{1}{2} \cdot b_1 \cdot (|w_1 \cdot x + w_2 \cdot y + w_3 \cdot z + 1| - |w_1 \cdot x + w_2 \cdot y + w_3 \cdot z - 1|) \\ a_{21} \cdot x + a_{22} \cdot y + a_{23} \cdot z + \frac{1}{2} \cdot b_2 \cdot (|w_1 \cdot x + w_2 \cdot y + w_3 \cdot z + 1| - |w_1 \cdot x + w_2 \cdot y + w_3 \cdot z - 1|) \\ a_{31} \cdot x + a_{32} \cdot y + a_{33} \cdot z + \frac{1}{2} \cdot b_3 \cdot (|w_1 \cdot x + w_2 \cdot y + w_3 \cdot z + 1| - |w_1 \cdot x + w_2 \cdot y + w_3 \cdot z - 1|) \end{pmatrix} \quad (12.1.4)$$

Je vidět, že pro výpočet druhé rovnice je zapotřebí absolutní hodnota. Problém je v tom, že v jazyce C++ funkce Abs() provádí operace jen s celými čísly. Musíme si tedy napsat svoji vlastní funkci, kterou nazveme abs2():

```
long double abs2(long double a)
{
    if (a<0) a=a*-1;
    return a;
}
```

Nyní ukázka zdrojového kódu samotného výpočtu:

```
while (i<times) {    //prováděj výpočet, je-li i menší než proměnná times (volí uživatel)

    i++;
    if (pocitatpodle==1){
        //vlastni zjednodušené řešení matice LORENZ

        dx=a11*x+a12*y+a13*z+x*x*d1*c11+x*d1*c12*y+x*d1*c13*z+y*d2*c11*x+y*y*d2*c12+y*d2*
        c13*z+z*d3*c11*x+z*d3*c12*y+z*z*d3*c13+b1;

        dy=a21*x+a22*y+a23*z+x*x*d1*c21+x*d1*c22*y+x*d1*c23*z+y*d2*c21*x+y*y*d2*c22+y*d2*
        c23*z+z*d3*c21*x+z*d3*c22*y+z*z*d3*c23+b2;

        dz=a31*x+a32*y+a33*z+x*x*d1*c31+x*d1*c32*y+x*d1*c33*z+y*d2*c31*x+y*y*d2*c32+y*d2*
        c33*z+z*d3*c31*x+z*d3*c32*y+z*z*d3*c33+b3;
    }
    //Chaos Class C
    if (pocitatpodle==2){
        mezivypocet=(abs2(d1*x+d2*y+d3*z+1)-abs2(d1*x+d2*y+d3*z-1));
        dx= a11*x + a12*y +a13*z +(0.5)*b1*mezivypocet;
        dy= a21*x + a22*y +a23*z +(0.5)*b2*mezivypocet;
        dz= a31*x + a32*y +a33*z +(0.5)*b3*mezivypocet;
    }
    //Samotná Eulerova Integrační Metoda
    x = x + dx * dt;
    y = y + dy * dt;
    z = z + dz * dt;
```

Všechny výsledky, včetně časového údaje, uložíme do matice. Abychom dodrželi přesnost a snížili výpočetní náročnost je v programu možnost zvolit, kolikátý výsledek se uloží.

```
//Uložení do pole s vypuštěním prvku
    krok++;
    if (krok==StrToFloat(Edit6->Text))
    {

        Pole->Cells[0][pamet]=FloatToStr(x);
        Pole->Cells[1][pamet]=FloatToStr(y);
        Pole->Cells[2][pamet]=FloatToStr(z);
```



```

Pole->Cells[3][pamet]=FloatToStr(i*dt);
pamet++;
krok=0;
}

```

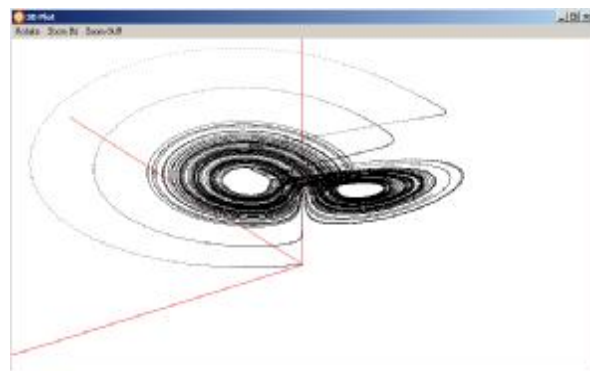
Z algoritmu je evidentní, že se uložení provede, když se krok rovná uživatelem volené konstantě (konstanta však musí být celé číslo).

## 12.2 Ukládání do souboru

Program ukládá a načítá výsledky vzniklé integrací. Vše je prováděno přes komponentu TMemo a komponenty TOpenDialog a SaveDialog. Příponu souboru si může uživatel zvolit vlastní, takže při zpětném načítání není aplikován žádný filtr. Formát souboru volíme takový, že se pod sebe stále ukládají hodnoty x,y,z a časový údaj. Program také ukládá a načítá tzv. simulační profily. Zde se do souboru uloží patřičné informace o všech nastavených parametrech důležitých pro řešení rovnic. Soubor má koncovku \*.prj, takže při načítání aplikujeme filtr, který zobrazuje jen soubory typu \*.prj. Zdrojové kódy jsou v souborech Unit1.cpp a Unit1.h.

## 12.3 3D Vizualizace dat

Další důležitou součástí programu je vizualizace dat. Uživatel má možnost volby 2D nebo 3D vizualizace. Pro 3D vizualizaci využijeme grafické knihovny OpenGL, kterou připojíme ve druhém formuláři. Abychom v programovém prostředí mohli využít těchto knihoven, musíme připojit následující hlavičkové soubory.



Obr. 25 Ukázka programu (3D vizualizace)

```

#include <gl/glu.h>
#include <gl/glaux.h>

```

Při prvním spuštění je však zapotřebí nastavit hloubku barev, buffer a formát pixelů. Další problém nastane při změně velikosti okna. Je zapotřebí okno znovu nastavit. Vše je programově ošetřeno. Celý zdrojový kód je přiložen v souboru Unit2.cpp.

Nejdůležitější součástí zdrojového kódu je vykreslení os x,y,z a samotných bodů řešení.

```

void TForm2::DrawScene(void)
{
    glShadeModel(GL_SMOOTH);
    glClearColor(1, 1, 1, 1); //barva pozadí
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity(); // reset pohledu
    glTranslatef(0.0f,-2.0f,-20.9f); // posune počátek
    glRotatef(-45, 0.1f,0.0f,0.0f);
}

```

```

glRotatef(-45, 0.0f,0.0f,1.0f);
glRotatef(status,0.0f,0.0f,1.0f);

glBegin(GL_LINES);    //osy x,y,z
glColor3f(1,0, 0);
glVertex3f(0 , 0, 0);
glVertex3f(0, 0, 10);
glEnd();
glBegin(GL_LINES);
glColor3f(1,0, 0);
glVertex3f(0 , 0, 0);
glVertex3f(0, 10, 0);
glEnd();
glBegin(GL_LINES);
glColor3f(1,0, 0);
glVertex3f(0 , 0, 0);
glVertex3f(10, 0, 0);
glEnd();
long double x = 0;
long double y = 0;
long double z = 0;
int i=0,barva=0;
bool stop = false;
    glBegin(GL_POINTS);
    while (!stop) {                                //načítání dat z matice ve formuláři 1
        x=StrToFloat(Form1->Pole->Cells[0][i]);
        y=StrToFloat(Form1->Pole->Cells[1][i]);
        z=StrToFloat(Form1->Pole->Cells[2][i]);
        i++;
        glColor3f(0,0,0);                          //barva bodů
        glVertex3f(x/meritko , y/meritko, z/meritko); //zmena meritka
        if (i >= Form1->pamet) stop = true;    //ochrana
    }
    glEnd();
glFlush();
}

```

Další funkcí je rotace vytvořeného objektu. To je realizováno pomocí časovače (kvůli plynulosti otáčení). Zmáčknutí tlačítka řídí jen spuštění resp. vypnutí časovače.

```

void __fastcall TForm2::rotace1Click(TObject *Sender)
{
    if (Timer1->Interval == 0)
        Timer1->Interval = 300;    //START STOP ROTACE
                                    //300 Mmezicas rotace.
    else
        Timer1->Interval = 0;

    DrawScene();
}

```

```

Invalidate();
}

```

Celé natočení objektu je realizováno funkcí časovače. Do globální proměnné status se stále přičítá hodnota 10. Když dosáhne hodnoty 360, nastaví ze znovu na 0 (probíhá tedy natáčení o  $10^\circ$ ).

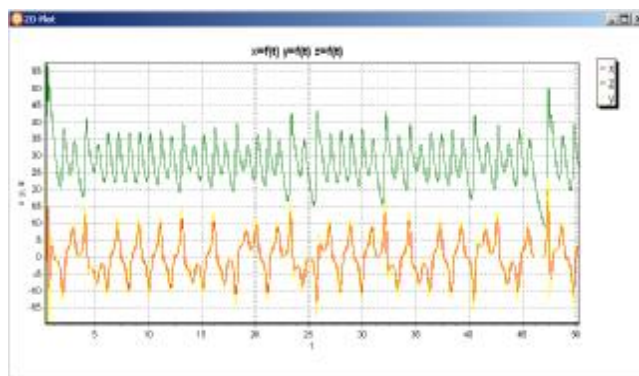
```

void __fastcall TForm2::Timer1Timer(TObject *Sender)
{
if (status==360) status=0;
status=status+10;
Invalidate();
}

```

## 12.4 2D Vizualizace dat

Dalším krokem, který program realizuje, je 2D vizualizace dat v závislosti na čase. Ta se provádí za pomoci standardní komponenty TChart, která je součástí programového prostředí. Komponenta umožňuje dokonce animovaný zoom. Po zobrazení třetího formuláře dojde k vykonání následující části zdrojového kódu.



Obr. 25 Ukázka 2D vizualizace

```

void __fastcall TForm3::FormShow(TObject *Sender)
{
long double x = 0;
long double y = 0;
long double z = 0;
long double dt = 0;
int i=0;
bool stop = false;
TFastLineSeries *ser=new TFastLineSeries(Char1); //vytvoření série
TFastLineSeries *ser2=new TFastLineSeries(Char1);
TFastLineSeries *ser3=new TFastLineSeries(Char1);
Chart1->AddSeries(ser);
Chart1->AddSeries(ser2);
Chart1->AddSeries(ser3);
ser->Name="x"; //pojmenování série
ser2->Name="z";
ser3->Name="y";
}

```

```

while (!stop) {    //načítání dat do grafu

x=StrToFloat(Form1->Pole->Cells[0][i]);
y=StrToFloat(Form1->Pole->Cells[1][i]);
z=StrToFloat(Form1->Pole->Cells[2][i]);
dt=StrToFloat(Form1->Pole->Cells[3][i]);
i++;
ser->AddXY(dt, x,""); //přiřazování dat do sérii pro graf
ser2->AddXY(dt, z,"");
ser3->AddXY(dt, y,"");
if (i >= Form1->pamet) stop = true;    //ochrana, aby nedošlo k přetečení
}
}

```

Po ukončení okna musí dojít k vymazání sérií. Více ve zdrojovém kódu v souboru Unit3.cpp.

## 12.5 Mřížková dimenze

Cílem programu bylo vytvořit algoritmy pro numerické řešení diferenciálních rovnic a dále připravit data pro stanovení dimenze objektu. Stanovení dimenze touto metodou je velice časově náročné. Pro stanovení hranic prostoru je potřeba najít největší a nejmenší hodnotu dat. Tímto krokem umístíme celý objekt do kvádrů. Problém je v tom, že algoritmus počítá s krychlí. Určíme-li však největší hranu kvádrů, můžeme vytvořit krychli, do které celý kvádr patří. Budeme tedy prohledávat kvádr, ale pro samotný výpočet a volbu hrany budeme počítat s krychlí. Tento na první pohled složitý krok má veliký dopad na urychlení výpočtu. Celý algoritmus prohledává kvádr (prostor  $\mathbf{R}^3$ ) po malých krychlích (hrana je stanovena z faktoru zmenšení a z hrany krychle) pomocí tří FOR cyklů. V posledním cyklu se v každém kroku prohledává pomocí while cyklu tabulka s daty. Hledají se taková data, která patří do malé krychle. V případě úspěchu dojde k předčasnému ukončení while cyklu. Je důležité si uložit počet krychlí, které obsahují data. Celý výpočet provádíme opakovaně v dalším FOR cyklu se změněným faktorem zmenšení. Abychom určili dimenzi, musíme vynést závislost:

$$f(\log(\text{úspěš}) = \log\left(\frac{1}{\text{Faktor\_zmenšení}}\right) \quad . \quad (12.5.1)$$

Je důležité vložit spojnicí trendu a z její směrnice určíme dimenzi objektu. Program také vykreslí graf závislosti volby velikosti hrany malé krychle (resp. faktoru zmenšení) na dimenzi.

Pro výpočet je volba faktoru zmenšení klíčová. Při volbě veliké hrany dojde k nepřesnému určení dimenze. Avšak při volbě příliš malé hrany, kromě časové náročnosti, může dojít také k chybě. Bude-li hrana menší, než volba integračního kroku, bude chyba velká.

Program také obsahuje okénko komponenty TMemo, ve kterém se zobrazí doplňkové informace o probíhající výpočtu (kvůli časové náročnosti má uživatel možnost kontrolovat průběh výpočtu).

```

void __fastcall TForm4::Button1Click(TObject *Sender)
{
    //deklarace
    long double i;

```

```

long double maxx,maxy,maxz,minx,minz,miny,a,b,c,vysledek;
long double hranka,r,s,t,faktor;
long double obsazeni=0;
bool stop;
int opakovani=0;
double off, slop;
int multi;
float ukazatel,ukazatel2;
for(int i=Chart1->SeriesCount()-1;i>=0;i--) //Vymazání grafu
{ delete Chart1->Series[i];
}
for(int i=Chart2->SeriesCount()-1;i>=0;i--)
{ delete Chart2->Series[i];
}
maxx=StrToFloat(Form1->Pole->Cells[0][1]); //nastaveni na 1. znak pole
maxy=StrToFloat(Form1->Pole->Cells[1][1]);
maxz=StrToFloat(Form1->Pole->Cells[2][1]);
minx=StrToFloat(Form1->Pole->Cells[0][1]);
miny=StrToFloat(Form1->Pole->Cells[1][1]);
minz=StrToFloat(Form1->Pole->Cells[2][1]);
for (i=0;i<=(Form1->pamet-1);i++) //hledání maxim a minim v R^3
{
if (maxx<StrToFloat(Form1->Pole->Cells[0][i]))
maxx=StrToFloat(Form1->Pole->Cells[0][i]);
if (maxy<StrToFloat(Form1->Pole->Cells[1][i]))
maxy=StrToFloat(Form1->Pole->Cells[1][i]);
if (maxz<StrToFloat(Form1->Pole->Cells[2][i]))
maxz=StrToFloat(Form1->Pole->Cells[2][i]);
if (minx>StrToFloat(Form1->Pole->Cells[0][i]))
minx=StrToFloat(Form1->Pole->Cells[0][i]);
if (miny>StrToFloat(Form1->Pole->Cells[1][i]))
miny=StrToFloat(Form1->Pole->Cells[1][i]);
if (minz>StrToFloat(Form1->Pole->Cells[2][i]))
minz=StrToFloat(Form1->Pole->Cells[2][i]);
}
a=maxx-minx; //určení vzdáleností
b=maxy-miny;
c=maxz-minz;
if (a>=b) vysledek=a; //nejvetsi vzdálenost
if (b>=c) vysledek=b;
if (c>=a) vysledek=c;
faktor=StrToFloat(Edit1->Text);vysledek/delitel; //volba hrany z nejmensi vzdálenosti
multi=StrToFloat(Edit2->Text);
Label1->Caption=FloatToStr(faktor*multi);
//-----
//vlastni algoritmus pro pocitani BOX-COUNTING METHOD
//pocitani v kvadru
TFastLineSeries *ser=new TFastLineSeries(Chart1); //nastaveni grafu
Chart1->AddSeries(ser);
ser->Name="log";

```

```

TFastLineSeries *ser2=new TFastLineSeries(Chart2); //nastaveni grafu2
Chart2->AddSeries(ser2);
ser2->Name="Box";
ukazatel=0; //reset ukazatele %
for (opakovani=1;opakovani<=multi;opakovani++)
    //opakovani pro vyneseni do grafu
{
    hranka=vysledek/(faktor*opakovani); //hrana krychle
    obsazeni=0; //reset obsazeni krychli
    for (r=minx;r<=(maxx);r=r+hranka)
    { //souradnice x
        ukazatel=ukazatel+(100/(multi*(a/hranka))); //krok ukazatele %
        StatusBar1->SimpleText=("Done:"+IntToStr(int(ukazatel))+"% ");
        //ukazatel %

        for (s=miny;s<=(maxy);s=s+hranka)
        //Počítáme s krychli, ale vsazujeme do kvádru
        { //souradnice y
            for (t=minz;t<=(maxz);t=t+hranka)
            { //souradnice z
                stop=false;
                //prohledani pole
                i=0;
                Memo1->Clear(); //kontrolní okénko
                Memo1->Lines->Add("----Check----");
                Memo1->Lines->
                    Add("Containing_Boxes:"+FloatToStr(obsazeni));
                Memo1->Lines->Add("Scale:"+IntToStr(int(faktor*opakovani)));
                while (!stop){

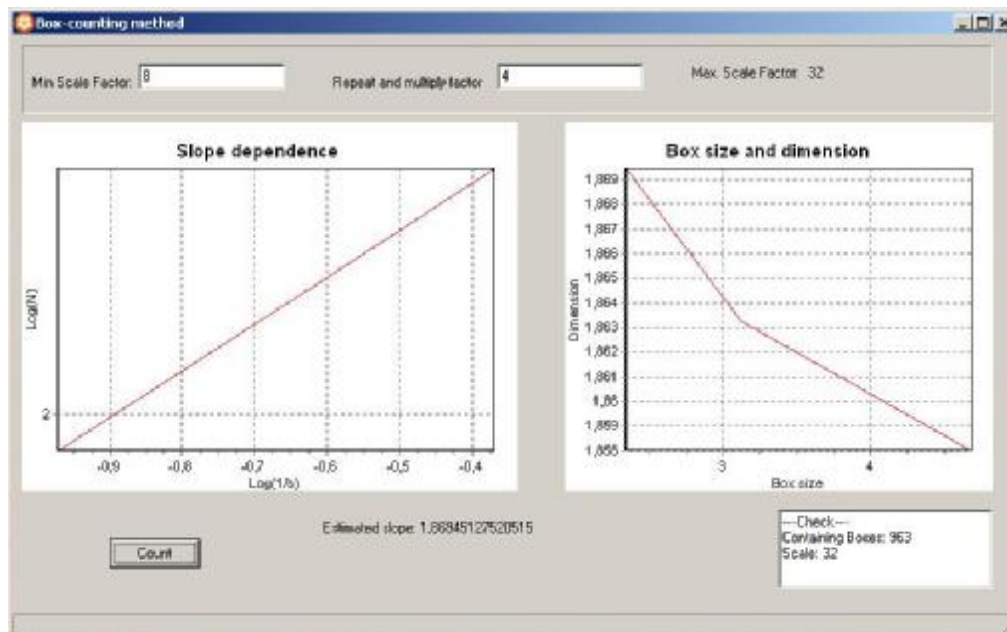
                    i++;
                    if (StrToFloat(Form1->Pole->Cells[0][i])>=r)
                    if (StrToFloat(Form1->Pole->Cells[0][i])<(r+hranka))
                    {
                        if (StrToFloat(Form1->Pole->Cells[1][i])>=s)
                        if (StrToFloat(Form1->Pole->Cells[1][i])<(s+hranka))
                        {
                            if (StrToFloat(Form1->Pole->Cells[2][i])>=t)
                            if (StrToFloat(Form1->Pole->Cells[2][i])<(t+hranka))
                            {
                                obsazeni++;
                                stop = true;
                            }
                        }
                    }
                }
                if (i>=(Form1->pamet-1))stop = true; //ochrana
            }
        }
    }
}

```

```

ser->AddXY(log10(1/hranka), log10(obsazeni), "");
if (opakovani>1){
getTrendLine(ser,off,slop); //funkce pro určení spojnice trendu a spojnice
ser2->AddXY(hranka, slop, "");
}
} //konec opakovani

```



Obr. 26 Výsledek pro počítání dimenze Lorenzova atraktoru

Teoretická hodnota dimenze je 2,06, program hodnotu stanovil na 1,87. Tuto hodnotu můžeme považovat za přesnou. Je pravděpodobné, že při jiné volbě faktoru zmenšení a menší integrační konstantě  $dt$  by výsledek byl přesnější.

## 12.6 Kaplan-Yorkeho dimenze

Tato část programu je realizovaná pomocí nového vývojového prostředí – CodeGear RAD Studio 2007 – Builder C++ 2007. Změna programovacího prostředí byla provedena, kvůli lepší podpoře jak ze strany výrobce IDE, tak ze strany operačních systémů typu Windows. Další změna, která byla na programu provedena, je to, že vytvořený program jde spustit na libovolném PC (se systémem typu Windows – podpora i Windows Vista 32b a 64b) bez nutné instalace dalších doplňujících knihoven. Dalším důvodem změny vývojového nástroje je lepší překladač, tedy výsledný program je efektivnější.

Abychom dostali potřebné vektory, musíme nejprve vynásobit dané rovnice příslušnou Jacobiho maticí.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \cdot (x \ y \ z) \cdot \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \cdot \begin{pmatrix} A & D & G \\ B & E & H \\ C & F & I \end{pmatrix} = \begin{pmatrix} J \\ K \\ L \\ M \\ N \\ O \\ P \\ Q \\ R \\ S \\ T \\ U \end{pmatrix}, \quad (12.1)$$

Kde matice s vektory A-I reprezentuje Jacobiho matici, která je tvořena příslušnými derivacemi.

Například vektor A bude mít tvar:

$$A = a_{11} + 2xd_1c_{11} + d_1c_{12}y + d_1c_{13}z + yd_2c_{11} + zd_3c_{11}. \quad (12.2)$$

Ostatní vektory budou mít obdobný tvar.

Program dále řeší situaci pro jiný typ rovnice. Řešení problému si jen naznačíme:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \frac{1}{2} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \cdot \left( \left| \begin{pmatrix} w_1 & w_2 & w_3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + 1 \right| - \left| \begin{pmatrix} w_1 & w_2 & w_3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + 1 \right| \right) \cdot \begin{pmatrix} A & D & G \\ B & E & H \\ C & F & I \end{pmatrix} = \begin{pmatrix} J \\ K \\ L \\ M \\ N \\ O \\ P \\ Q \\ R \\ S \\ T \\ U \end{pmatrix} \quad (12.3)$$

Opět pro vektor A platí:

$$A = a_{11} + \frac{1}{2}b_1|d_1x + d_2y + d_3z + 1|d_1 - |d_1x + d_2y + d_3z - 1|d_1. \quad (12.4)$$



Je potřeba si uvědomit, že vektory J-U jsou rovnice v diferenciálním tvaru. Pro jejich řešení tedy potřebujeme jednu z uvedených metod v předchozích kapitolách. Uvedeme si část zdrojového kódu, kde se potřebné operace realizují:

```

/* Start iterating... */
for (int d=1; d<imax; d++)
{ /* Linear and Non-Linear Equations + MATICOVE operace */
    if (Form1->pocitatpodle==1){
        dx[1]=a11*x+a12*y+a13*z+x*x*d1*c11+x*d1*c12*y+x*d1*c13*z+y*d2*c11*x+y*y*d2*c
12+y*d2*c13*z+z*d3*c11*x+z*d3*c12*y+z*z*d3*c13+b1;
        dx[2]=a21*x+a22*y+a23*z+x*x*d1*c21+x*d1*c22*y+x*d1*c23*z+y*d2*c21*x+y*y*d2*c22+y*
d2*c23*z+z*d3*c21*x+z*d3*c22*y+z*z*d3*c23+b2;
        dx[3]=a31*x+a32*y+a33*z+x*x*d1*c31+x*d1*c32*y+x*d1*c33*z+y*d2*c31*x+y*y*d2*c32+y*
d2*c33*z+z*d3*c31*x+z*d3*c32*y+z*z*d3*c33+b3;
        //Obecna Jacobbiho matice
        //Deleni po sloupcich
        A=a11+2*x*d1*c11+d1*c12*y+d1*c13*z+y*d2*c11+z*d3*c11;
        B=a21+2*x*d1*c21+d1*c22*y+d1*c23*z+y*d2*c21+z*d3*c21;
        C=a31+2*x*d1*c31+d1*c32*y+d1*c33*z+y*d2*c31+z*d3*c31;
        //-----
        D=a12+x*d1*c12+d2*c11*x+2*y*d2*c12+d2*c13*z+z*d3*c12;
        E=a22+x*d1*c22+d2*c21*x+2*y*d2*c22+d2*c23*z+z*d3*c22;
        F=a32+x*d1*c32+d2*c31*x+2*y*d2*c32+d2*c33*z+z*d3*c32;
        //-----
        G=a13+x*d1*c13+y*d2*c13+d3*c11*x+d3*c12*y+2*z*d3*c13;
        H=a23+x*d1*c23+y*d2*c23+d3*c21*x+d3*c22*y+2*z*d3*c23;
        I=a33+x*d1*c33+y*d2*c33+d3*c31*x+d3*c32*y+2*z*d3*c33;
    }
    if (Form1->pocitatpodle==2){

dx[1]= a11*x + a12*y + a13*z +(0.5)*b1*(abs2(d1*x+d2*y+d3*z+1)-abs2(d1*x+d2*y+d3*z-1));
dx[2]= a21*x + a22*y + a23*z +(0.5)*b2*(abs2(d1*x+d2*y+d3*z+1)-abs2(d1*x+d2*y+d3*z-1));
dx[3]= a31*x + a32*y + a33*z +(0.5)*b3*(abs2(d1*x+d2*y+d3*z+1)-abs2(d1*x+d2*y+d3*z-1));

        //Obecna Jacobbiho matice
        //Deleni po sloupcich
        A=a11+.5*b1*(abs2(d1*x+d2*y+d3*z+1)*d1-abs2(d1*x+d2*y+d3*z-1)*d1);
        B=a21+.5*b2*(abs2(d1*x+d2*y+d3*z+1)*d1-abs2(d1*x+d2*y+d3*z-1)*d1);
        C=a31+.5*b3*(abs2(d1*x+d2*y+d3*z+1)*d1-abs2(d1*x+d2*y+d3*z-1)*d1);
        //-----
        D=a12+.5*b1*(abs2(d1*x+d2*y+d3*z+1)*d2-abs2(d1*x+d2*y+d3*z-1)*d2);
        E=a22+.5*b2*(abs2(d1*x+d2*y+d3*z+1)*d2-abs2(d1*x+d2*y+d3*z-1)*d2);
        F=a32+.5*b3*(abs2(d1*x+d2*y+d3*z+1)*d2-abs2(d1*x+d2*y+d3*z-1)*d2);
        //-----
        G=a13+.5*b1*(abs2(d1*x+d2*y+d3*z+1)*d3-abs2(d1*x+d2*y+d3*z-1)*d3);
        H=a23+.5*b2*(abs2(d1*x+d2*y+d3*z+1)*d3-abs2(d1*x+d2*y+d3*z-1)*d3);
        I=a33+.5*b3*(abs2(d1*x+d2*y+d3*z+1)*d3-abs2(d1*x+d2*y+d3*z-1)*d3);
    }
    //-----/

```

```

//NASOBENI MATIC dx=J*Y

for (i=0;i<=2;i++){
dx[4+i]= A*Y[4+i]+D*Y[7+i]+G*Y[10+i];
//-----/
dx[7+i]= B*Y[4+i]+E*Y[7+i]+H*Y[10+i] ;
//-----/
dx[10+i]= C*Y[4+i]+F*Y[7+i]+I*Y[10+i];
}

//EULEROVA INTEGRACNI METODA
for (i=1;i<=Na;i++){

    X[i]=X[i]+dx[i]*dt;

}
t=t+dt;
x=X[1];
y=X[2];
z=X[3];
for (i=1;i<=Na;i++) Y[i]=X[i]; //Vytvoreni pomocne matice

```

Vidíme, že samotný proces se děje ve for-cyklu. Ten se opakuje, dokud nedosáhneme požadované hodnoty iterací. Výsledné hodnoty numerické integrace jsou uloženy v poli X (viz. zdrojový kód). Výsledky násobení matic se ukládají do proměnné dx[i]. Přičemž první tři prvky pole obsahují výsledky výpočtu pro integraci základních rovnic. Všechny prvky vektoru d[i] je nyní třeba integrovat jednou z popsanych numerických metod. Zbytek je produktem vynásobení Jacobiho maticí.

Následně se musí první vektor normalizovat.

```

/* Normalizace prvního vektoru */
Znorm[1] = 0.0;
for (j = 1; j <= N; j++)
    Znorm[1] = Znorm[1] + X[N*j+1] * X[N*j+1];
Znorm[1] = sqrt(Znorm[1]);

for (j = 1; j <= N; j++)
    X[N*j+1] = X[N*j+1] / Znorm[1];

```

V dalším kroku je potřeba vygenerovat nový ortonormální set. Následně spočítat normu vektoru a daný vektor normalizovat. Ze zdrojového kódu je evidentní, že se celý proces děje ve for-cyklech.

```

/* Generovani noveho ortonormálního setu */
for (j = 2; j <= N; j++)
{
    /* Generovani J-1 GOS=GSC */
}

```

```

for (k = 1; k <= j-1; k++)
{
    GSC[k] = 0.0;
    for (l = 1; l <= N; l++)
        GSC[k] = GSC[k] + X[N*l+j] * X[N*l+k];
}

/* Novy vektor */
for (k = 1; k <= N; k++)
    for (l = 1; l <= j-1; l++)
        X[N*k+j] = X[N*k+j] - GSC[l] * X[N*k+l];

/* Pocitani normy vektoru */
Znorm[j] = 0.0;
for (k = 1; k <= N; k++)
    Znorm[j] = Znorm[j] + X[N*k+j] * X[N*k+j];
Znorm[j] = sqrt(Znorm[j]);

/* Normalizace noveho vektoru */
for (k = 1; k <= N; k++)
    X[N*k+j] = X[N*k+j] / Znorm[j];
}

```

Nyní stačí jen provést přepočítání amplitud složek pomocí vypočítané hodnoty logaritmu normy vektoru Znorm. V dalším kroku následuje normalizace exponentů časovou vzdáleností.

```

/* Update amplitud */
for (k = 1; k <= N; k++)
    CUM[k] = CUM[k] + log(Znorm[k]);

//normalizace exponentu

for (j = 1; j <= N; j++)
    lyp_exp[j] = CUM[j] / t;

```

Máme-li vytvořenou ortonormální soustavu, aplikace vzorce pro výpočet dimenze je triviální.

$$D_{KY} \equiv j - \frac{I_1 + \dots + I_j}{I_{j+1}}. \quad (12.5)$$

Ve zdrojovém kódu se výpočet dimenze provede následovně. Výsledná dimenze se ukládá do proměnné Dky.

```

/* Lyapunov dimension Dky */
double sum=0, lsum, Dky;
int kmax=0;

for (j=1; j<=N; j++)
{
    sum = sum + lyp_exp[j];
}

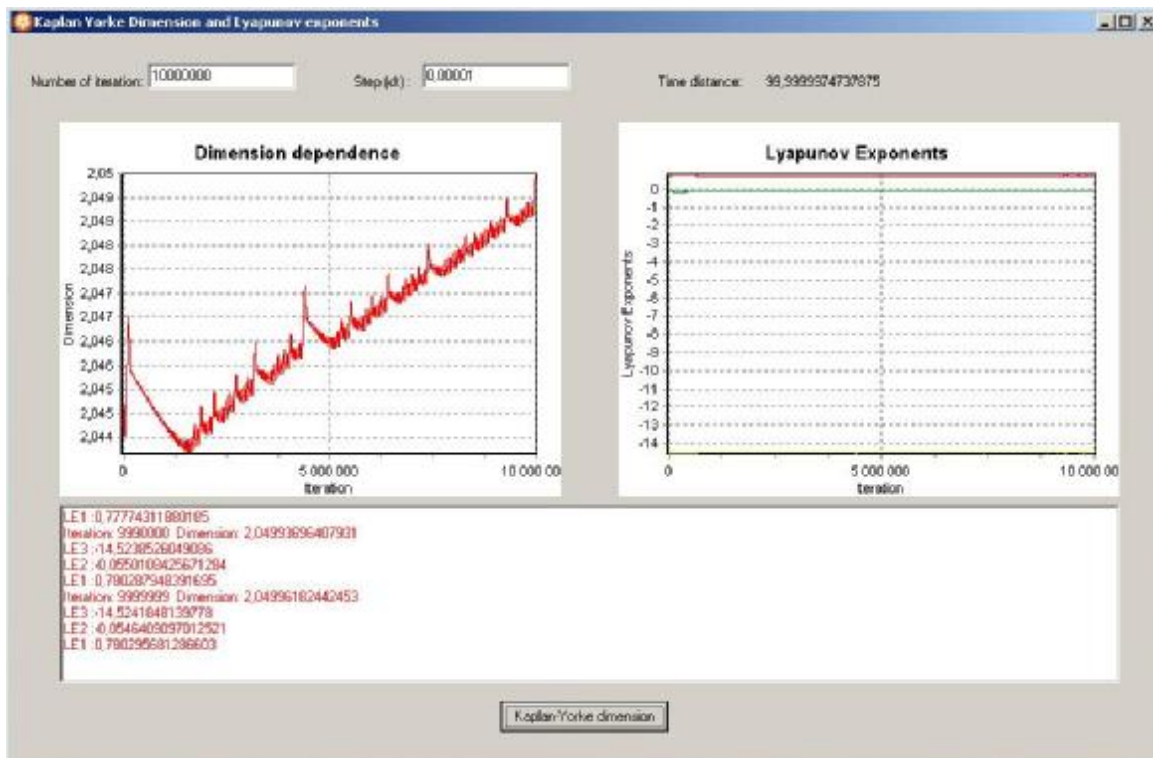
```

```

if( sum>=0 )
{
    kmax=j;
    lsum=sum;
}
}
if( sum>=0 )
    Dky = kmax - lsum/lyp_exp[kmax+1];

```

Celý program je vytvořený do nového formuláře. Ten si ze základního přebírá nastavení parametrů matic, počáteční podmínky, počet iterací a časovou konstantu  $d(t)$  (obě se však dají ještě změnit). Před spuštěním modulu se však nemusí provádět integrace. Ta se kvůli své složitosti počítá za pomoci vlastních algoritmů popsaných výše.



Obr. 27 Základní vzhled okna s výpočtem pro Lorenzův atraktor

Metoda výpočtu je velice rychlá v porovnání s mřížkovou metodou (box-counting method) výpočtu. Dosahované výsledky metody jsou velice blízké teoretickým předpokladům. Celá simulace byla provedena pro Lorenzův atraktor u kterého víme, že teoretická hodnota Lyapunových exponentů je  $\lambda_1 = 0,78, \lambda_2 = 0, \lambda_3 = -14,5$ . Hodnoty dosažené programem jsou  $\lambda_1 = 0,7802, \lambda_2 = -0,005, \lambda_3 = -14,524$ . Z toho hodnota teoretické dimenze je  $D=2,06$  a hodnota stanovená pomocí výpočetní metody je  $D=2,06$ . Ve výpočtech však nastávají numerické chyby dané numerickou integrací, tedy správnou volbou integrační konstanty. To znamená volbou matematické metody principiálně dochází k chybám. Dále výpočetní chyby vznikají při převodu datových typů. Například při převodu datového typu string na double float. Chyba tak vzniká již na pátém desetinném místě. Metodu můžeme ale považovat za velice efektivní vzhledem k době potřebné pro stanovení přesných výsledků. Výsledné výpočty dimenze a Lyapunových exponentů jsou programem vyneseny do grafu v závislosti na počtu iterací. Z daných charakteristik se tak dá určit následný průběh exponentů a hodnoty, ke kterým se limitně blíží.

Počítačové řešení je však velice závislé na velikosti operační paměti systému. Aby se celý výpočet urychlil, není možné využívat pevný disk pro ukládání mezivýsledků. Proto nastavíme výpis výsledků po 10000 iteracích. Znamená to tedy, že každá iterace provedená po 10000 iteracích se uloží do paměti a vypočítá se hodnota dimenze. Tento proces velice urychlí výpočty.

Následně srovnáme stejné řešení pomocí programu Matlab 2007b s vytvořeným programem v jazyce C++. Celý výpočet provedeme pro známý Lorenzův atraktor. Nastavení parametrů bude:

Čas, do kterého chceme integrovat:  $t=100$  jednotek.

Program v jazyce Matlab využívá pro integraci implementovanou funkci ode45. Jedná se o integraci pomocí Runge Kuttovy metody 4. řádu.

Abychom měli stejnou řesnost výpočtu, nastavíme ve vytvořeném programu v jazyce C++:

$d(t)=0,0001$  jednotek

Čas, do kterého chceme integrovat je také  $t=100$  jednotek. Vidíme, že bude zapotřebí provést  $10^6$  iterací. Dosažené výsledky jsou uvedeny v tabulce:

Matlab 2007b	C++
Trvání = 120 sekund	Trvání = 22 sekund
L1=0,7375	L1=0,7625
L2=0,0047	L2=-0,005
L3=-14,409	L3=-14,565
D=2,06	D=2,06

Tab. 12.6.1 Srovnání výsledků

## 13 ZÁVĚR

V bakalářské práci jsem se seznámil s metodami výpočtů metrických dimenzí stavových atraktorů autonomních deterministických systémů. Konkrétně s Hausdorffovou metodou a mřížkovou metodou stanovování dimenzí. První z metod jde použít jen na soběpodobné útvary jako je Kochova křivka nebo Mandelbrotova množina. Druhou metodu jsem vybral pro realizaci pomocí počítače. Její přesnost je závislá na volbě hrany mřížky, respektive krychle a na volbě integračního kroku.

Dále jsem vytvořil algoritmy pro numerickou integraci soustav diferenciálních rovnic Eulerovou a Runge-Kuttovou metodou. Tyto metody jsem porovnal z hlediska přesnosti a výpočetní náročnosti. Pro realizaci jsem zvolil Eulerovu metodu.

Dále jsem vytvořil program pro výpočet metrických dimenzí stavových atraktorů včetně vizualizace dat ve 2 a 3 rozměrném prostoru. Vytvořený program umožňuje data a simulační profily ukládat a načítat do souboru.

Správnost navržených programů jsem ověřoval na vybraných dynamických systémech. Přesnost výpočtu dimenze Lorenzova atraktoru byla stanovena s odchylkou 0,19. Vyšší přesnosti je možné dosáhnout volbou menších mřížek (krychlí) a volbou menší integrační konstanty. Lze konstatovat, že výpočet je použitelný jen pro informativní odhad.

V další části jsem se věnoval souvislostem mezi Lyapunovovými exponenty a velikostí dimenze. Vytvořil jsem algoritmy pro výpočet jednorozměrného spektra Lyapunovových exponentů. Následně jsem vypočetl velikost Kaplan Yorkeho dimenze. K dosažení výborných výsledků je však potřeba správně volit integrační konstanty a počet iterací. Metoda je početně náročná jen na numerickou integraci a na Gram-Schmidtovu ortonormalizaci soustavy. Následný výpočet velikosti dimenze je triviální. Pro Lorenzův atraktor mi velikost dimenze vycházela  $D=2,06$ , což se přesně shoduje s teoretickými předpoklady. Program vynáší Lyapunovovy exponenty a dimenzi v závislosti na počtu iterací do přehledného grafu, ze kterého je možno stanovit následný vývoj výsledků. Všechny programy jsem vytvořil v programovacím jazyce C++ a Matlab 2007b. Dosažené numerické výsledky se nelišily. Program vytvořený v jazyce Matlab ale vyžadoval pro výpočet asi o 80% více času.

Dále jsem nastínil fyzikální hranice, které předem předurčují (Heisenbergův princip neurčitosti) nepřesnost metody. To tedy znamená, že nikdy nebude možné vypočítat průběh dopředu s absolutní přesností.

V poslední části se zabývám možnostmi praktického využití. Nejvíce diskutovaná oblast je nyní oblast medicíny. Existují studie, které zkoumají možnosti využití teorie pro diagnostiku pacientů. Jde především o diagnostiku srdeční aktivity a o diagnostiku mozkové aktivity.

# 14 POUŽITÁ LITERATURA

- [1] WYK, M. A., STEEB, W. H. Chaos in electronics. Kluwer Academic Publishers 1997, ISBN 0-7923-4576-2.
- [2] Falconer K., Fractal Geometry Mathematical Foundations & Applications. John Wiley & Sons Ltd., 1990, ISBN 978-0471922872.
- [3] Williams G., Chaos Theory Tamed. Joseph Henry Press. 1997, ISBN 978-0-309-06351-7.
- [4] Kapitaniak T. a Bishop S., The Illustrated Dictionary of Nonlinear Dynamics and chaos. John Wiley & Sons Ltd., 1999, ISBN 0471983233.
- [5] Fajmon B. a Růžicková I., Matematika 3, Skripta FEKT VUT v Brně, 2005
- [6] *OpenGL*, (2002-2007), <http://www.opengl.org/>
- [7] Grafika Publishing, s.r.o. *Builder*, (1997-2006), <http://builder.cz>
- [8] Zelinka I., Včelař F., Čandlík M., Fraktální geometrie - principy a aplikace. BEN – technická literatura, Praha 2006, ISBN 80-7300-191-8.

# 15 SEZNAM OBRÁZKŮ:

- Obr. 1 Lorenzův atraktor popisuje pohyb systému ve stavovém prostoru. Zde pro počáteční hodnoty  $r = 28$ ,  $\sigma = 10$ ,  $b = 8/3$ .
- Obr. 2 Rozdílné výsledky při různých počátečních podmínkách
- Obr. 3 Euklidovské dimenze
- Obr. 4 Topologická dimenze (Vlevo-přímka, Vpravo- povrch)
- Obr. 5 Část Mandelbrotovy množiny
- Obr. 6 Rozdělení úsečky
- Obr. 7 Příklad určování mřížkové dimenze
- Obr. 8 Příklad výpočtu mřížkové dimenze ( $D=1.84$ )
- Obr. 9 Zobrazení v prostoru  $R^3(x,y,z)$
- Obr. 10 Vývoj proměnných v čase
- Obr. 11 Rozdíly v řešení pro  $h=0,1$
- Obr. 12 Vynesení odchylek
- Obr. 13 Rozdíly v řešení pro  $h=0,0001$
- Obr. 14 Vynesení odchylek
- Obr. 15 Porovnání vlivu volby kroku  $h$  pro Lorenzovy rovnice
- Obr. 13 Transformace kruhové trajektorie na eliptickou
- Obr. 14 Vizualizace vektorů (resp. Báze)
- Obr.15 Vizualizace vektorů po Gram- Schmidově ortonormalizace
- Obr. 16 Normální ECG
- Obr. 17 ECG při infarktu
- Obr. 19 Obrázek neuronu.
- Obr. 19 Různé průběhy akčního potenciálu
- Obr. 20 Magnetická resonance mozku
- Obr. 21 Stavová proměnná  $x$  v čase  $t$
- Obr. 22 Spektrum Lorenzova atraktoru pro stavovou proměnnou  $x$
- Obr. 23 Ukázka hlavního okna programu
- Obr. 24 Ukázka matic
- Obr. 25 Ukázka programu (3D vizualizace)
- Obr. 25 Ukázka 2D vizualizace
- Obr. 26 Výsledek pro počítání dimenze Lorenzova atraktoru
- Obr. 27 Základní vzhled okna s výpočtem pro Lorenzův atraktor



# 16 SEZNAM PŘÍLOH

Příloha číslo 1 – CD ROM s binárními a zdrojovými kódy celého programu včetně bakalářské práce v pdf.