

**SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM  
MAROSVÁSÁRHELYI KAR,  
INFORMATIKA SZAK**



**SAPIENTIA**  
ERDÉLYI MAGYAR  
TUDOMÁNYEGYETEM

A dolgozat címe

**DIPLOMADOLGOZAT**

Témavezető:  
Témavezető neve,  
Egyetemi tanár

Végzős hallgató:  
Szerző neve

**2021**

UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA  
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,  
SPECIALIZAREA INFORMATICĂ



UNIVERSITATEA  
SAPIENTIA

Titlul lucrării

LUCRARE DE DIPLOMĂ

Coordonator științific:  
Témavezető neve,  
Profesor universitar

Absolvent:  
Szerző neve

2021

**SAPIENTIA HUNGARIAN UNIVERSITY OF  
TRANSYLVANIA  
FACULTY OF TECHNICAL AND HUMAN SCIENCES  
COMPUTER SCIENCE SPECIALIZATION**



**SAPIENTIA**  
HUNGARIAN UNIVERSITY  
OF TRANSYLVANIA

Title of the Bachelor thesis

**BACHELOR THESIS**

Scientific advisor:  
Témavezető neve,  
Full Professor

Student:  
Szerző neve

**2021**

## Declarație

Subsemnata/ul ....., absolvent(ă) al/a specializării ....., promoția..... cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapiientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licență/proiect de diplomă/disertație se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea,

Data:

Absolvent

Semnătura.....

# Kivonat

Dolgozatom témája a differenciálegyenletek megoldása különböző technológiák segítségével. Mint tudjuk a körülöttünk lévő világban szinte minden eseményt, jelenséget, problémát le tudunk írni differenciálegyenletek segítségével. Majd a kapott egyenletet vagy egyenleteket megoldjuk számítógép segítségével, valamilyen technológiát felhasználva.

Vannak olyan helyzetek az életben, amikor fontos, hogy ne csak pontosan, hanem a lehető leggyorsabban is meg tudjuk oldani ezeket az egyenleteket. Tehát számít az időtényező is, mert ezen akár emberi életek is múlhatnak. Ilyen jellegű probléma lehet például valamilyen természeti katasztrófa előrejelzése vagy egy betegséggel kapcsolatban felmerülő jövőbeli kérdés megválaszolása. Ezekben az esetekben nagyon fontos lehet az, hogy a rendelkezésünkre álló szoftverek és programok közül, melyiket választjuk a probléma megoldására.

Jelen dolgozatomban arra a kérdésre keresem a választ, hogy milyen típusú szoftvert érdemes használni, ahhoz hogy az adott feladat egyenleteit a lehető leggyorsabban tudjuk megoldani. A szoftvereket két kategóriára osztom fel és így fogom megvizsgálni: olyan szoftverek melyek már léteznek és használhatóak differenciálegyenletek megoldására, valamint saját megvalósítású szoftverek.

A választott és megvalósított programokkal megoldatjuk ugyanazokat a feladatokat és minden esetben mérjük a futási időt. A teszteket megismétljük és átlagokat számítunk, hogy az eredmények még hitelesebbek legyenek. A végső eredményeket összehasonlítjuk és kiértékeljük.

Végül az eredmények alapján levonjuk a következtetéseket, hogy melyik módszer vagy technológiát érdemes használni vagy esetleg továbbfejleszteni a jövőben. Továbbá megállapítjuk azt is, hogyha egy adott technológiával nem érdemes tovább próbálkozni.

# Rezumat

Tema tezei este rezolvarea ecuațiilor diferențiale cu ajutorul diferitelor tehnologii. Cum știim în jurul nostru aproape toate acțiunile și fenomenele ale naturii sau probleme date pot fi descrise cu ajutorul ecuației diferențiale. Ecuațiile primite vom rezolva cu ajutorul calculatorului, folosind oricare tehnologie.

Sunt situații în viață când este important nu numai exactitatea ci și rapiditatea rezolvării ecuațiilor. Deci contează și factorul timpului, fiindcă de aceasta pot depinde vieți omenești. Asemenea probleme pot fi de exemplu pronosticul catastrofelor naturale sau rezolvarea problemelor apărute în unele boli. În aceste cazuri este foarte importantă alegerea corectă a software-ului pentru rezolvarea problemei.

În teză caut răspunsul la folosirea software-ului potrivit pentru rezolvarea în mod cât mai rapid a problemei. Software-ele am împărțit în două grupuri: programuri care deja există și pot fi folosite pentru rezolvarea ecuațiilor și programe construite de mine.

Cu ajutorul programelor alese și rezolvate vom rezolva problema dată și măsurăm timpul necesar. Vom repeta testele și calculăm media pentru autentificarea rezultatelor. Facem compararea și evaluarea rezultatelor.

În final tragem concluzia ca care dintre tehnologii merită folosire sau dezvoltare. În plus constatăm în ce tehnologie nu merită să investigăm.

# Abstract

The aim of my thesis is to solve differential equations using different technologies. We know that in the world around us, we can describe almost every event, phenomenon using differential equations. We can then solve the given equation or system of equations with the help of a computer, using some technology.

There are situations in real life when it is necessary to solve the equations not only as accurately but as quickly as possible. The time factor is also very important because human lives may also depend on it. Such a problem might be a natural disaster forecast or the need to urgently answer a disease related question. In these cases our choice of available software is critical in solving the problem.

In this thesis I aim to identify the best software for solving the problem as quickly as possible. I split the software into two groups: programs that already exist and which can be used to solve the equations, and programs that I have built.

With the help of the chosen and solved programs, we will solve the problem and measure the time taken to achieve this. We will repeat the tests and calculate the median for the authentication of the results. We compare and evaluate the results.

Finally, we reveal which technology is worth using or developing. In addition, we identify which technology is not worth investigating.

# Tartalomjegyzék

<b>1. Bevezető</b>	<b>10</b>
<b>2. Programok, technológiák bemutatása</b>	<b>12</b>
2.1. Beépített szoftverek, könyvtárak . . . . .	12
2.1.1. Matlab - ode45 . . . . .	12
2.1.2. Boost - Odeint . . . . .	13
2.2. Általam megvalósított szoftverek . . . . .	15
2.3. Szoftverek összehasonlítása . . . . .	17
2.4. Differenciálegyenletek megoldása GPU-n . . . . .	18
<b>3. Eredmények, következtetések</b>	<b>20</b>
3.1. Beépített szoftverek esetén . . . . .	20
3.2. Saját szoftverek esetén . . . . .	22
3.3. Összességében . . . . .	22
<b>4. Számítógépes elemzés</b>	<b>24</b>
<b>5. Szoftver</b>	<b>26</b>
5.1. A szoftver bemutatása . . . . .	28
5.2. A szoftver megírásához használt könyvtárak . . . . .	28
5.3. Diagramok . . . . .	30
5.3.1. Use Case diagram . . . . .	30
5.3.2. Osztálydiagram . . . . .	30
5.3.3. Szekvencia diagram . . . . .	32
<b>Összefoglaló</b>	<b>34</b>
<b>Köszönetnyilvánítás</b>	<b>35</b>
<b>Ábrák jegyzéke</b>	<b>36</b>
<b>Táblázatok jegyzéke</b>	<b>37</b>
<b>Irodalomjegyzék</b>	<b>38</b>
<b>Függelék</b>	<b>39</b>
F.1. A TeXstudio felülete . . . . .	39
F.2. Válasz az „Élet, a világmindenség, meg minden” kérdésére . . . . .	40



# 1. fejezet

## Bevezető

Felmerülhet bennünk a kérdés, hogy mi is az a differenciálegyenlet és hogy egyáltalán mire jó, vagy hol használhatjuk fel? Röviden összefoglalva a differenciálegyenlet egy olyan egyenlet, amelyben az ismeretlen egy függvény és az egyenlet tartalmazza a függvény valamilyen rendű deriváltját is. Tehát nem ismerjük a konkrét függvényt, csak annak változását különböző időpillanatokban.

Ha jobban belegondolunk a mindennapi életben is rengeteg ilyen esemény, jelenség, folyamat, stb. vesz körül, amit csak megfigyelni tudunk, de nem tudjuk konkrétan leírni matematikai vagy fizikai képletekkel. Itt jönnek képbe a differenciálegyenletek, az előbbiekben elmondottak alapján tökéletesen alkalmasak az ilyen jellegű problémák felírására, modellezésére. Egy egyszerű kis példa az az eset, amikor a sütőből kiveszünk egy forró süteményt és ennek a kihűlését szeretnénk valamilyen módon megvizsgálni. Ez úgy lehetséges, hogy különböző időpillanatokban megmérjük a sütemény hőmérsékletét és lejegyezzük, majd ezekből az előzetes ismeretekből felállítjuk a differenciálegyenletünket. Miután megvan az egyenletünk (modellünk) már csak meg kell oldani. Mai fejlett világunkban ezt már nem papíron analitikus módszerekkel végezzük, azért sem, mert sok esetben nem is lehetséges kézzel megoldani az egyenleteket, ezért segítségül hívjuk a számítógépet és numerikus számításokkal próbáljuk megoldani az adott problémát.

A valós életben vannak sokkal bonyolultabb esetek is, amikor nem ilyen egyszerű felírni, megalkotni vagy megoldani a probléma modelljét (figyelembe kell venni számos más környezeti behatást, tényezőt). Emellett nagyon fontos az is, hogy egy adott problémát milyen gyorsan és hatékonyan tudunk megoldani a mai számítógépek és technológiák segítségével. Dolgozatom témájának megválasztásánál is ez a feladat keltette fel leginkább az érdeklődésem, hogy hogyan lehet azokat a bizonyos egyenleteket a leggyorsabban megoldani. Vannak olyan esetek vagy problémák, ahol a gyorsaság és hatékonyság elengedhetetlen. Például egy súlyos betegség vagy természeti katasztrófa előrejelzésénél fontos, hogy a modellünket a lehető leggyorsabban megoldjuk és még időben tudjuk jelezni ha baj van. Ezekben az esetekben nagyon fontos, hogy az adott modellt megoldó szoftverünk milyen technológiákkal és módszerekkel van megvalósítva.

A dolgozat további részében ismertetem a differenciálegyenletek numerikus megoldásának elméleti alapjait, majd olyan konkrét modelleket vizsgálunk meg, amelyek esetében fontos az időtényező. Továbbá bemutatom a különböző technológiákkal megvalósított szoftvereket, megvizsgáljuk ezeknek a hatékonyságát külön-külön és egymáshoz képest is.

Végül megpróbáljuk megtalálni a megvalósított szoftverek közül azt, amelyik a legjobban teljesít gyorsaság szempontjából a különböző tesztek során. [Knu11]

„Maxwell’s equations” are named for James Clark Maxwell and are as follow:

$$\vec{\nabla} \cdot \vec{E} = \frac{\rho}{\epsilon_0} \quad \text{Gauss's Law} \quad (1.1)$$

$$\vec{\nabla} \cdot \vec{B} = 0 \quad \text{Gauss's Law for Magnetism} \quad (1.2)$$

$$\vec{\nabla} \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \quad \text{Faraday's Law of Induction} \quad (1.3)$$

$$\vec{\nabla} \times \vec{B} = \mu_0 \left( \epsilon_0 \frac{\partial \vec{E}}{\partial t} + \vec{J} \right) \quad \text{Ampere's Circuital Law} \quad (1.4)$$

Equations (1.1), (1.2), (1.3), and (1.4) are some of the most important in Physics.

## 2. fejezet

# Programok, technológiák bemutatása

### 2.1. Beépített szoftverek, könyvtárak

Dolgozatom ezen részében először vizsgáljunk meg olyan beépített differenciálegyenlet megoldó szoftvereket, melyeket már megemlítettem. Ezek közül én a Matlab és a Boost - Odeint beépített programját használtam és vizsgáltam meg. Mindkét esetben közösleges differenciálegyenletek (ODE) megoldására, az előre megírt algoritmus segítségével, melynek háttérében természetesen a Dormand-Prince módszer áll.

$$x = 1 + 1 + 1 + 1 \tag{2.1}$$

$$= 4. \tag{2.2}$$

$$\begin{aligned} y &= mx + b \\ z &= nw + c. \end{aligned} \tag{2.3}$$

$$\int_0^1 \sum_a^b \prod_{\alpha}^{\beta}. \tag{2.4}$$

$$\frac{12}{34}.$$

#### 2.1.1. Matlab - [ode45](#)

A Matlab egy programcsomag és egyben egy technikai nyelv is, mely magas szinten lehetőséget biztosít számítások elvégzésére, modellezésre, szimulációra, megjelenítésre, vizualizációra és számos más hasznos mérnöki munka elvégzésére. Esetünkben a legfontosabb, hogy könnyedén tudunk közösleges differenciálegyenleteket megoldani az ode45 program segítségével. Emellett az eredményeket egy jól megtervezett felületen ki is tudjuk ábrázolni. Az alábbi példában jól látható, hogy milyen egyszerű és kényelmes a használata:

```
f = @(t,y) y;
t = [0 10];
y0 = 1.0;
ode45(f, t, y0);
plot(t, y(:));
```

### 2.1. kódrészlet. Matlab példakód diff. egyenlet megoldására.

A fenti bemenetre  $n = 100$  - szor lefuttattuk az algoritmust és a következő időeredményeket kaptuk:

- Futási idők **átlaga**: 0.0043 sec
- Futási idők **minimuma**: 0.0026 sec
- Futási idők **maximuma**: 0.1504 sec



### 2.1.2. Boost - Odeint

Egy másik előre megírt közönséges differenciálegyenlet megoldó a Boost könyvtár-csomag Odeint nevezetű könyvtára. Ez egy modern C++ nyelven írt csomag, lényeges jellemzői, hogy **nagy teljesítményre** képes és **nagyon magas szinten** (absztraktn - Template Metaprogramming) van megírva, így **rugalmas** és könnyen integrálható különböző rendszerekbe. Emellett rugalmas a bementi adatok típusát illetően is. Ugyanakkor jó tudni, hogy ez a nagyon absztrakt megvalósítás hátrány is lehet, mert bizonyos esetekben nagyon nehéz megérteni vagy megoldani egy felmerülő problémát. Most lássunk egy egyszerű példát a használatára:

```
#include <iostream>
#include <boost/numeric/odeint.hpp>

using namespace std;
using namespace boost::numeric::odeint;

typedef runge_kutta_dopri5<double> stepper_type;

void rhs( const double x , double &dxdt , const double t ) {
    dxdt = 3.0/(2.0*t*t) + x/(2.0*t);
}

void write_cout( const double &x , const double t ) {
    cout << t << '\t' << x << endl;
}

int main() {
```

```
double x = 0.0;  
integrate_adaptive( make_controlled( 1E-12 , 1E-12 , stepper_type() ) ,  
rhs , x , 1.0 , 10.0 , 0.1 , write_cout );  
}
```

---

**2.2. kódrészlet.** Odeint példakód.

Az előző kódrészlet eredménye a következő:

- Futási idők **átlaga**: 0.0989 sec
- Futási idők **minimuma**: 0.09 sec
- Futási idők **maximuma**: 0.099 sec

## 2.2. Általam megvalósított szoftverek

```
function [yy, tt, timeSpent] = fun_dopri45(f, y0, t0, tf, tolerance)
...
while t < tf
    if (t+h >= tf)
        h = tf-t;
    end

    % Calculate k1, k2, ... , k7
    k1 = h*feval(f, t+h*C(1), y);
    k2 = h*feval(f, t+h*C(2), y + A2(1)*k1);
    ...
    k7 = h*feval(f, t+h*C(7), y + A7(1)*k1 + A7(3)*k3 + ... + A7(6)*k6);

    % Calculate the next point
    yt = y + A7(1)*k1 + A7(3)*k3 + A7(4)*k4 + A7(5)*k5 + A7(6)*k6;
    % Calculate the error
    err = abs(E(1)*k1 + E(3)*k3 + ... + E(7)*k7);

    if max(err) < tolerance
        t = t + h;
        y = yt;
        ...
    end

    % Calculate optimal step size
    scale = 1.25*(maxErr/tolerance)^(1/5);
    if scale > 0.2
        h = h / scale;
    else
        h = 5.0*h;
    end
end
end
```

### 2.3. kódrészlet. Matlab kód ode45 használata nélkül.

- Futási idők **átlaga**: 0.0043 sec
- Futási idők **minimuma**: 0.0058 sec

- Futási idők **maximuma**: 0.0287 sec

A következő szoftvert, amit bemutatok **Java** nyelven írtam objektum orientáltan, a fejlesztés során pedig Eclipse fejlesztői környezetet használtam. Ez a szoftver három egységből áll: fő -, differenciálegyenlet megoldó - és kifejezés kiértékelő egység (ezt a legnehezebb megvalósítani vagy megtalálni a megfelelő könyvtárat). Az alkalmazás nem rendelkezik grafikus felhasználói felülettel, a bemeneti adatokat egy szöveges állományból olvassa be (amelynek jól meghatározott szerkezete van) és a kívánt eredményeket a standard kimenetre írja ki. A kifejezés kiértékelő egység megírásánál felhasználtam egy előre elkészített Java könyvtárat, melyet *JEP*-nek neveznek. A szoftverben arra használtam fel, hogy egy sztringként megadott kifejezést kiértékeltem és elvégeztem a segítségével. Mind ezt úgy csinálja, hogy a háttérben felépít egy kifejezésfát, aminek a leveleiben lesznek az értékek, csúcsaiban a műveletek, zárójelek, stb. (lásd az alábbi ábrát):

- Futási idők **átlaga**: 0.1422 sec
- Futási idők **minimuma**: 0.123 sec
- Futási idők **maximuma**: 0.341 sec

Harmadiként lássuk a **C++ szoftvert**, amelyet szintén objektum orientáltam valósítottam meg. Ennek a szoftvernek a szerkezete hasonló a Java szoftver szerkezetéhez. Ebben az esetben is szükség volt egy kifejezés kiértékelő könyvtárra, hogy ne kelljen egy sajátot írni. Először kipróbáltam egy *muparser* nevű könyvtárat, aztán egy másikat is, aminek *ExprTk* (Expression Toolkit Library) a neve. Fontos elmondani, hogy mindkét könyvtár ingyenesen elérhető és használható. Az első könyvtárat nehezebb volt hozzáadni a szoftverhez és mivel több fájlból állt több időbe került a fordítása is. A legfőbb ok, amiért mégis a másodikat használtam az volt, hogy jelentősen gyorsabb és hatékonyabb volt az én szoftveremben. Tehát végül az *ExprTk* könyvtárat használtam a jobb teljesítménye és könnyebb integrálhatósága miatt.

- Futási idők **átlaga**: 0.1086 sec
- Futási idők **minimuma**: 0.096 sec
- Futási idők **maximuma**: 0.214 sec

Végül nézzük meg az utolsó, Dormand-Prince módszeren alapuló szoftvert, amely egy **Android alkalmazás**. Mint tudjuk napjainkban a mobil eszközök nagyon elterjedtek és teljesítményük is jelentősen megnőtt, lassan felveszik a versenyt a személyi számítógépekkel. Ezért mindenképp szerettem volna az algoritmust megvalósítani mobil eszközökre is és megnézni itt is a teljesítményt. Mivel az Android alkalmazások fejlesztésénél Java nyelvet használunk, így könnyű dolgom volt, hiszen a Java szoftverből át tudtam venni a már jól megírt és elkülönített osztályokat. Ugyanazt a *JEP* könyvtárat használtam itt is a kifejezések kiértékelésére, így majd az eredmények összehasonlítását is könnyebbé tettem. A Java és C++ szoftverrel ellentétben rendelkezik egy kis grafikus felhasználói felülettel, de a bemeneti adatokat itt is egy szöveges állományból olvassuk be, mert nem túl kényelmes azt a sok számadatot, meg egyenletet beviteli mezőkön keresztül beírni.

- Futási idők **átlaga**: 37.9015 sec
- Futási idők **minimuma**: 31.116 sec
- Futási idők **maximuma**: 70.876 sec

## 2.3. Szoftverek összehasonlítása

Az előző alfejezetekben ismertettem a már létező Dormand-Prince módszeren alapú differenciálegyenlet megoldók közül kettőt és négy saját megvalósítást is. Mindegyik esetében láthattunk futási időket és számadatokat, azonban nem láttuk ezeket egymás mellett. Ebben az alfejezetben összegezzük és összehasonlítjuk a kapott eredményeket.

Fontos, hogy minden algoritmust ugyanazon a hardveren teszteljünk, mert csak így reálisak és összehasonlíthatóak a mérési adatok. Esetünkben használt hardver konfigurációja:

- Intel Core i5-7200U, 2.50 GHz processzor, 8.00 GB RAM memória

Természetesen az Android alkalmazást csak mobileszközökön lehetett vizsgálni, itt két készüléket használtam a tesztelésre:

- Motorola Moto E2, Quad-core 1.2 GHz processzor, 1 GB RAM memória
- Samsung Galaxy Core Prime, Quad-core 1.2 GHz processzor, 1 GB RAM memória

Technológia	Átlagidő (s)	Min. idő (s)	Max. idő (s)	Hardver
Matlab - ode45	0.0032	0.0029	0.0038	Intel Core i5
Boost - Odeint	0.0989	0.0900	0.1290	Intel Core i5
Matlab	0.0062	0.0060	0.0066	Intel Core i5
Java	0.2224	0.1970	0.3020	Intel Core i5
C++	0.1047	0.1010	0.1240	Intel Core i5
Android	37.9015	31.1160	70.8760	Moto E2
Android	35.9987	29.4200	87.6120	Core Prime

**2.1. táblázat.** Mérési eredmények  $n = 10$  tesztesetre.

Technológia	Átlagidő (s)	Min. idő (s)	Max. idő (s)	Hardver
Matlab - ode45	0.0043	0.0026	0.1504	Intel Core i5
Boost - Odeint	0.0912	0.0900	0.0990	Intel Core i5
Matlab	0.0067	0.0058	0.0287	Intel Core i5
Java	0.1422	0.1230	0.3410	Intel Core i5
C++	0.1086	0.0960	0.2140	Intel Core i5
Android	—	—	—	Moto E2
Android	—	—	—	Core Prime

**2.2. táblázat.** Mérési eredmények  $n = 100$  tesztesetre.



## 2.4. Differenciálegyenletek megoldása GPU-n

A továbbiakban nézzük meg a két algoritmus magjának szekvenciális és párhuzamosított változatait:

```
for (int i = 0; i < numberOfVariables; ++i) {
    mResult.at(i)[j + 1] = mResult.at(i)[j] + mInputs->getStepSize() *
        (mFunctionsParsers.at(i)->computeFunctionValue(values));
}
```

**2.4. kódrészlet.** Euler módszer szekvenciális kód.

```
__global__ void computeFunctionsValuesKernel(double* resultValues,
double* previousValues, double* functionValues, double stepSize, int N) {
    int i = threadIdx.x;

    if (i < N) {
        resultValues[i] = previousValues[i] + stepSize * functionValues[i];
    }
}
```

**2.5. kódrészlet.** Euler módszer párhuzamosított kód.

```
for (int k = 0; k < numberOfVariables; ++k) {
    mResult.at(k)[i + 1] = mResult.at(k)[i] + (mInputs->getStepSize() / 6)*
        (K[0][k] + 2 * K[1][k] + 2 * K[2][k] + K[3][k]);
}
```

**2.6. kódrészlet.** Runge-Kutta módszer szekvenciális kód.

```
__global__ void computeValuesKernel(double* resultValues, double* K,
double* previousValues, double stepSize, int numberOfVariables) {
    int i = threadIdx.x;

    if (i < numberOfVariables) {
        resultValues[i] = previousValues[i] + ((stepSize / 6) *
            (K[i] + 2 * K[i + 1 * numberOfVariables] +
            2 * K[i + 2 * numberOfVariables] + K[i + 3 * numberOfVariables]));
    }
}
```

**2.7. kódrészlet.** Runge-Kutta módszer párhuzamosított kód.

Nézzünk meg pár mérési eredményt a következő differenciálegyenlet rendszerre:

$$\begin{cases} y_1'(t, y) = y_1 \\ \vdots \\ y_n'(t, y) = y_n \\ y_1'(t_0) = 1.0 \\ \vdots \\ y_n'(t_0) = 1.0 \end{cases}, t \in [0, 100], n = 5 \quad (2.5)$$

	CPU sec (Intel Core i5)		GPU sec (GeForce 940MX)	
	Euler	Runge-Kutta	Euler	Runge-Kutta
$h = 10.0$	0.006	0.036	0.587	0.036
$h = 1.0$	0.074	0.523	0.841	0.490
$h = 0.1$	0.689	4.987	1.565	3.399
$h = 0.01$	7.202	52.800	14.321	50.110
$h = 0.001$	71.071	500.999	98.445	321.965

**2.3. táblázat.** Mérési eredmények  $n = 5$  egyenlet esetén.

$$\begin{cases} y_1'(t, y) = y_1 \\ \vdots \\ y_n'(t, y) = y_n \\ y_1'(t_0) = 1.0 \\ \vdots \\ y_n'(t_0) = 1.0 \end{cases}, t \in [0, 100], n = 10 \quad (2.6)$$

	CPU sec (Intel Core i7)		GPU sec (GeForce 950M)	
	Euler	Runge-Kutta	Euler	Runge-Kutta
$h = 10.0$	0.039	0.256	1.070	0.222
$h = 1.0$	0.320	2.482	0.994	2.087
$h = 0.1$	3.042	23.962	4.143	20.978
$h = 0.01$	30.812	241.977	35.009	206.842
$h = 0.001$	305.092	2394.930	344.485	2067.870

**2.4. táblázat.** Mérési eredmények  $n = 10$  egyenlet esetén.

## 3. fejezet

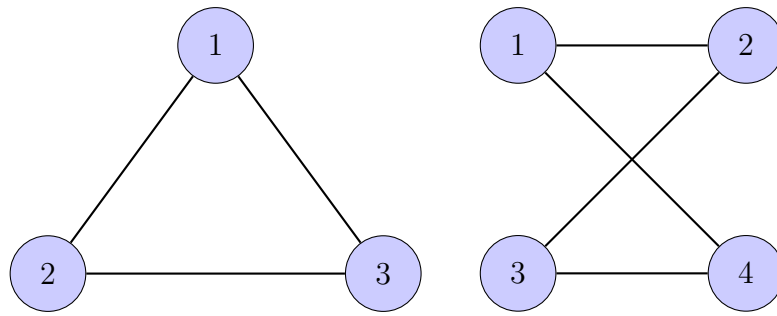
# Eredmények, következtetések

### 3.1. Beépített szoftverek esetén

A 2. fejezetben bemutatam két beépített, előre megvalósított szoftvert, melyek a Matlab ode45 programja és a Boost - Odeint könyvtár. Az elért eredmények és tesztek azt mutatják, hogy a Matlab ode45 differenciálegyenlet megoldója sokkal gyorsabb és hatékonyabb, mint az Odeint. Ha a tesztesetekben mért átlagidőket összehasonlítjuk láthatjuk (2.3. alfejezet), hogy az ode45 20 – 30 -szor gyorsabb a Odeintnél. Ez talán annak is köszönhető, hogy a Matlab egy nagyon komoly szoftver, amelynek programcsomagjain mérnökök és programozók százai (vagy akár ezrei) dolgoznak, így természetes, hogy az algoritmusok jobban optimalizáltak és hatékonyabbak az ingyenes szoftvereknél. A továbbiakban összegezzük, hogy a két technológiának milyen előnyei és hátrányai vannak vagy éppen miért érdemes/nem érdemes használni őket, lásd [LS15]

#### Matlab ode45 előnyei:

- nagyon egyszerű a használata, nem igényel komoly programozási ismereteket
- könnyű beépíteni és összekötni más Matlab programokkal
- a kapott eredményeket mátrix vagy vektor típusokban téríti vissza, ami könnyűvé teszi az eredmények további kezelését
- az eredményeket grafikus felületen azonnal meg tudjuk jeleníteni
- jobb eredményeket produkált, mint az Odeint
- jól dokumentált, sok példa van a használatára



3.1. ábra. Egyszerű gráf TIKZ segítségével

#### Matlab ode45 hátrányai:

- komoly hátránya az Odeinttel szemben, hogy **fizetni kell a használatáért**
- nagyon sok memóriát használ, komolyan igénybeveszi a számítógép erőforrásait

#### Odeint előnyei:

- ingyenes és nyílt forráskódú, használható személyi és kereskedelmi célokra egyaránt
- nagyon rugalmas, absztrak, így könnyedén változtatható a bemeneti adatok típusa vagy struktúrája
- C++ nyelven íródott, támogatva a modern programozási technológiákat (Generikus programozás, Template Metaprogramming)

#### Odeint hátrányai:

- használata nehezebb, mint a Matlab ode45 programé, szükséges a C++ programozási nyelv ismerete
- a kapott eredményekkel nem olyan könnyű bánni, mint a Matlab esetében
- absztraktsága miatt nehéz a felmerülő problémákat megoldani
- dokumentáltsága jóval szegényesebb, mint a Matlabé

## 3.2. Saját szoftverek esetén

Saját szoftverek esetében sikerült négy különböző technológia segítségével megvalósítani a Dorman-Prince differenciálegyenlet megoldó algoritmust (lásd 2.2. alfejezet). A felhasznált technológiák: Matlab, Java, C++ és Android voltak.

Ezen technológiák közül az Androidos szoftverrel kapcsolatban már előzetes félelmünk voltak, mivel azt feltételeztük, hogy bármennyire is fejlettek napjainkban a mobil-eszközök, mégis hardveresen nem lesznek elegendőek ahhoz, hogy versenybe tudjanak szállni a számítógépekkel. Néhány teszt után feltételezéseink beigazolódtak, láthatjuk a 2.3. alfejezet teszteseteiben is, hogy az Android szoftver mennyire gyengén teljesített (mind a Motorola Moto E2, mind a Samsung Galaxy Core Prime esetében). Összehasonlítva a többi algoritmussal az átlagidőket nézve 150 – 160 - szor lassabb a Javanál és 300 – 350 - szor a C++ - nál! Tehát azt a következtetést vonhatjuk le, hogy nem érdemes mobil-eszközön differenciálegyenleteket oldani, mivel túlságosan nagy a hardver igénye és egyelőre ezen a téren nem képesek tartani a lépést a számítógépekkel.

A további három technológia közül (Matlab, Java, C++) meglepő módon itt is a Matlab teljesített a legjobban, igaz, hogy ebben az esetben már nem használtuk az ode45 programot, hanem megírtam én a saját függvényemet. Ez a teljesítményen is meglátszott, mert az általam írt függvény nem tudott jobban teljesíteni a tesztek alatt, mint az ode45. Mindezek ellenére 35 – 40 - szor gyorsabb volt a Javanál és megközelítőleg 15 – 20 - szor gyorsabb a C++ - nál.

A Java és C++ szoftvereket összehasonlítva elmondhatom, hogy az esetek többségében a C++ körülbelül 2 - szor volt gyorsabb a Javanál, ami megfelel az előzetes elvárásoknak.

Amit nagyon fontosnak tartok kihangsúlyozni, hogy az általam megvalósított C++ szoftver a tesztek során nagyon jól teljesített, felvette a versenyt az Odeint könyvtárral és az elért átlagok is csak nagyon kicsivel maradnak el az Odeint által produkált eredményektől (2.3. alfejezet).

Továbbá megvalósítottam az Euler és Runge-Kutta módszerek párhuzamosított változatait is CUDA technológia segítségével. Ebben az esetben a tesztek azt mutatták, hogy az Euler módszer esetében többbe kerül a sok CPU és GPU memória közötti másolás művelete, mint amennyit nyerünk a számítások elvégzése során. Tehát ebben az esetben ez a fajta párhuzamosítási megközelítés nem éri meg. Ezzel ellentétben a Runge-Kutta módszer esetében a megközelítés eredményesnek bizonyult abban az esetben, ha az egyenletek száma nagy és a lépésköz kicsi. A 2.4 alfejezetben láthattuk, hogy abban az esetben ha az egyenletek száma  $n = 10$  és a lépésköz  $h = 0.001$ , a GPU-n megközelítőleg 5 és fél perccel hamarabb lefutott az algoritmus, mint a CPU-n. Ezzel a párhuzamosítási módszerrel nem tudtuk kihasználni a videokártya által nyújtott maximális számítási kapacitást, de így is jelentős különbséget sikerült elérni a futási időket nézve, lásd [K08].

## 3.3. Összességében

A fentieket összegezve elmondhatom, hogy megéri előre megírt szoftvereket vagy könyvtárakat használni differenciálegyenletek megoldására. Nagyon megkönnyíthetik az eltűnkét egyszerűségükkel és nagy teljesítményükkel. Viszont fontos elmondani, hogy használatuk problémákkal is járhat, például fizetni kell értük vagy nem lehet belenyúlni az

algoritmusokba kedvünk szerint, esetleges fellépő hibák esetén nagyon nehéz lenyomozni a hiba forrását (vagy szinten lehetetlen).

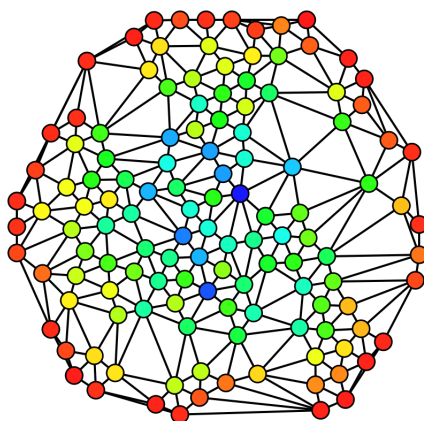
Saját algoritmusok terén bátran elmondhatom, hogy megéri a C++ technológiát választani és ezen a vonalon továbbhaladni egy esetleges saját könyvtár megírása, megvalósítása felé. Láthattuk, hogy az általam megírt C++ szoftver is felvette a versenyt az Odeint könyvtárral, ami szintén C++ technológiát alkalmaz, lásd [\[Ant07\]](#).

Egy másik vonal, amit érdemes sokkal jobban felderíteni az a CUDA technológiával és grafikus kártyával történő differenciálegyenlet megoldása. Láthattuk, hogy egy kis párhuzamosítás is jelentős időbeli különbséget jelenthet bizonyos algoritmusok esetében. Annak tudatában is, hogy a differenciálegyenletek megoldása nem a legjobban adatpárhuzamosítható feladatok közé sorolható azt mondom, hogy megéri ezzel a technológiával foglalkozni. Ennek kapcsán a legnagyobb motiváció számomra a jövőre nézve a parciális differenciálegyenletek párhuzamosításának megvalósítása és tanulmányozása, mivel ezeknél az egyenleteknél jobban ki lehet használni a videokártya rácsos szerkezetét.

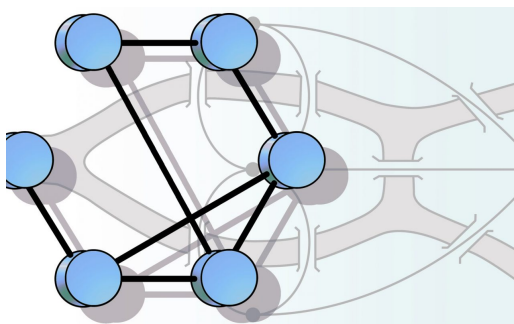
## 4. fejezet

# Számítógépes elemzés

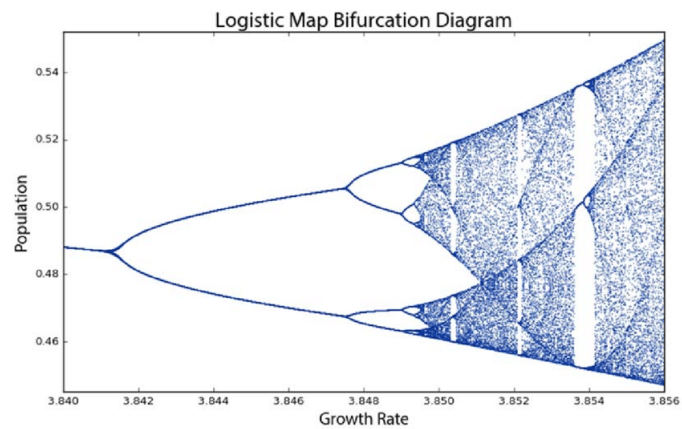
Ebben a fejezetben bemutatom az előző fejezetben elviekben ismertetett modelleket példákön és valós adatokon keresztül. A szoftver segítségünkre lesz abban, hogy megoldjuk a felmerülő differenciálegyenleteket és egyenletrendszereket, végül az eredményeket grafikus felületen is megtekinthetjük.



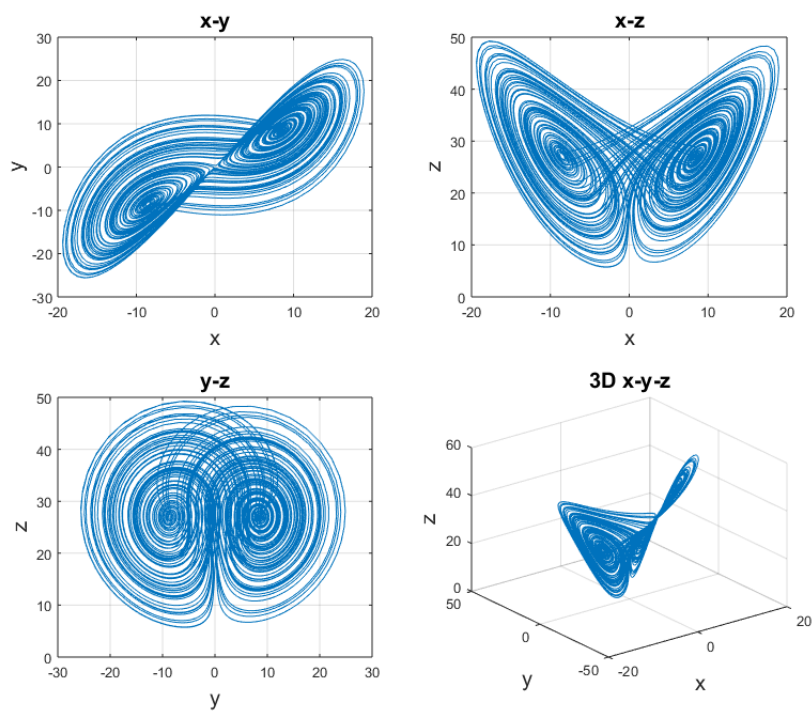
4.1. ábra. Gráf



4.2. ábra. Gráf 2



4.3. ábra. Bifurkációs diagram



4.4. ábra. Lorentz attraktor



## 5. fejezet

### Szoftver

---

#### 1. algoritmus: Euclidean Algorithm

---

**Input:** Integers  $a \geq 0$  and  $b \geq 0$

**Output:** GCD of  $a$  and  $b$

1 **while**  $b \neq 0$  **do**

2      $r \leftarrow a \bmod b$ ;

3      $a \leftarrow b$ ;

4      $b \leftarrow r$ ;

5 **end**

---

---

#### 2. algoritmus: How to write algorithms

---

**Data:** this text

**Result:** how to write algorithm with L<sup>A</sup>T<sub>E</sub>X2e

1 initialization;

2 **while** *not at end of this document* **do**

3     read current;

4     **if** *understand* **then**

5         go to next section;

6         current section becomes this one;

7     **else**

8         go back to the beginning of current section;

9     **end**

10 **end**

---

---

**3. algorithmus:** IntervalRestriction

---

**Data:**  $G = (X, U)$  such that  $G^{tc}$  is an order.

**Result:**  $G' = (X, V)$  with  $V \subseteq U$  such that  $G'^{tc}$  is an interval order.

```
1 begin
2    $V \leftarrow U$ 
3    $S \leftarrow \emptyset$ 
4   for  $x \in X$  do
5      $NbSuccInS(x) \leftarrow 0$ 
6      $NbPredInMin(x) \leftarrow 0$ 
7      $NbPredNotInMin(x) \leftarrow |ImPred(x)|$ 
8   end
9   for  $x \in X$  do
10    if  $NbPredInMin(x) = 0$  and  $NbPredNotInMin(x) = 0$  then
11      AppendToMin( $x$ )
12    end
13  end
14  while  $S \neq \emptyset$  do
15    REM remove  $x$  from the list of  $T$  of maximal index
16    while  $|S \cap ImSucc(x)| \neq |S|$  do
17      for  $y \in S - ImSucc(x)$  do
18        { remove from  $V$  all the arcs  $zy : \}$ 
19        for  $z \in ImPred(y) \cap Min$  do
20          remove the arc  $zy$  from  $V$ 
21           $NbSuccInS(z) \leftarrow NbSuccInS(z) - 1$ 
22          move  $z$  in  $T$  to the list preceding its present list
23          {i.e. If  $z \in T[k]$ , move  $z$  from  $T[k]$  to  $T[k - 1]$ }
24        end
25         $NbPredInMin(y) \leftarrow 0$ 
26         $NbPredNotInMin(y) \leftarrow 0$ 
27         $S \leftarrow S - \{y\}$ 
28        AppendToMin( $y$ )
29      end
30    end
31    end
32    RemoveFromMin( $x$ )
33  end
34 end
35 end
```

---

## 5.1. A szoftver bemutatása

Az általam írt szoftver egy **Java** nyelven, **NetBeans IDE 8.0.1** fejlesztői környezetben írt asztali alkalmazás, amelynek fő funkcionálitása a kezdetiérték-probléma típusú differenciálegyenletek numerikus megoldása és ezen megoldások grafikus felületen való ábrázolása. A fő funkcionálitás mellett a szoftver tartalmaz még két kisebb funkcionálitást is, ezek közül az egyik a kétdimenziós függvényábrázolási lehetőség, a másik pedig a háromdimenziós függvények megjelenítésének lehetősége.

Az szoftver a differenciálegyenletek megoldásához a 2. fejezetben leírt numerikus eljárásokat alkalmazza.

A grafikus felhasználói felület megalkotásához a **Swing** (Java) komponens készletet használtam. A Swing használatával célom az volt, hogy egy felhasználóbarát és könnyen kezelhető felületet hozzak létre, amelyen a felhasználó könnyedén eligazodhat. Továbbá e komponenskészlet használata mellett szól az is, hogy a későbbiekben bemutatásra kerülő könyvtárak, melyek az ábrázolás megvalósítására használtam szintén Swing komponensekkel vannak megvalósítva.

Felhasználói felület, valamint a három funkcionálitás bemutatása képekben:

---

### 4. algoritmus: Switch használata

---

```
1 switch order do
2   case bloody mary do
3     Add tomato juice;
4     Add vodka;
5     break;
6   case hot whiskey do
7     Add whiskey;
8     Add hot water;
9     Add lemon and cloves;
10    Add sugar or honey to taste;
11    break;
12  otherwise do
13    | Serve wine;
14  end
15 end
```

---

## 5.2. A szoftver megírásához használt könyvtárak

A szoftver elkészítésénél szükségem volt néhány előre megírt osztálykönyvtárra, amelyek megkönnyítették a munkámat. Ezekről tudni kell, hogy nyílt forráskódúak, tehát bárki számára elérhetőek az interneten, továbbá azt is, hogy ezek is Java nyelvben íródtak, hasonlóan, mint az általam írt alkalmazás. A továbbiakban szeretném bemutatni ezeket a könyvtárakat és azt, hogy mire- és hogyan használtam fel őket.

- JMathPlot (<https://sites.google.com/site/mulabsltd/products/jmathplot>):
  - Java könyvtár, amelyet interaktív megjelenítésre, ábrázolásra fejlesztettek

- gyors és könnyű utat biztosít tudományos adatok megjelenítésére Swing komponensek segítségével (nem használ OpenGL-t)
  - az általa biztosított saját komponenseket úgy lehet használni, mint bármely más Swing komponenst
  - a számomra legfontosabb tulajdonsága az, hogy két- és háromdimenziós ábrázolási lehetőséget biztosít, ezt használtam fel az alkalmazásomban
- JMathArray (<https://sites.google.com/site/mulabsltd/products/jmatharray>):
    - olyan Java könyvtár, amely alapvető matematikai, lineáris algebrai műveleteket biztosít számunkra
    - a könyvtár által biztosított statikus metódusok tömbökre alkalmazhatóak
    - a szoftverben arra használtam, hogy egy megadott intervallum két végpontja között egy bizonyos lépésközzel haladva egy tömböt tudjak feltölteni (inkrementálás)
  - JEP (<http://www.cse.msu.edu/SENS/Software/jep-2.23/doc/website/>):
    - szintén egy Java könyvtár, amelyet különböző elemzésekre és kiértékelésekre fejlesztettek
    - segítségével egy szöveggént (sztring-ként) megadott kifejezést könnyedén kiértékelhetünk, elvégezhetünk
    - a szöveggént megadott kifejezésből a háttérben egy kifejezésfát épít fel, majd a későbbiekben ennek a fának a segítségével dolgozik
    - emellett sok általános matematikai függvény és konstans is bele van építve, amiket szintén könnyedén elérhetünk
    - az általam fejlesztett szoftverben a függvények sztringként adhatók meg egy beviteli mezőn keresztül, a JEP könyvtárat ezen függvények „parszolására” használtam fel

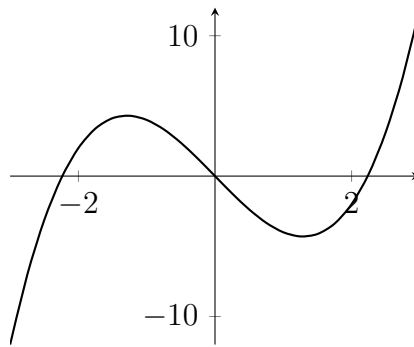
## 5.3. Diagramok

### 5.3.1. Use Case diagram

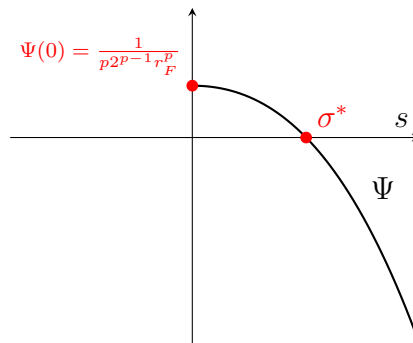
### 5.3.2. Osztálydiagram

A szoftver szerkezetileg két nagyobb részből (csomagból) áll, az egyik a felhasználói felület megalkotásához szükséges osztályokat tartalmazza, a másik pedig a differenciálegyenletek megoldására szolgáló osztályokat és a parszer osztályt, mely egy sztringként megadott függvény kiértékelésére szolgál.

Az implementációnál a felhasználói felület elemeit tartalmazó csomagot „View”-nak, a numerikus módszereket és a parszert tartalmazó csomagot „Model”-nek neveztem, emellett a 6.7-es ábrán megjelenik egy harmadik csomag is, amely tartalmazza a „MainClass”-t és egyben a `main()` metódust is. Az alábbi két diagramon láthatjuk a felsorolt csomagokat és a bennük lévő osztályokat, illetve a köztük lévő kapcsolatokat.

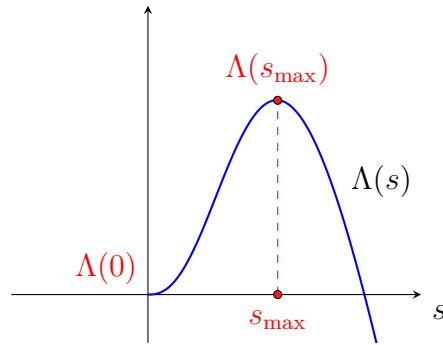


**5.1. ábra.** Az  $x^3 - 5x$  függvény grafikus képe PGFPLOT-al

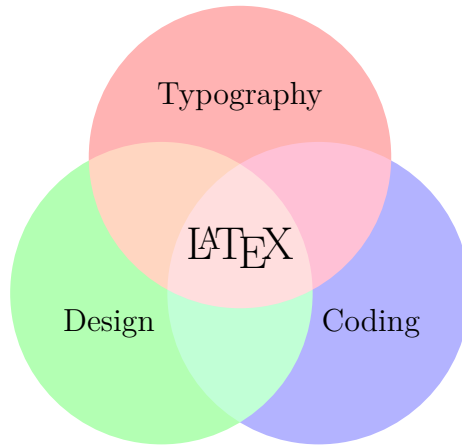


**5.2. ábra.** A  $\Psi$  grafikus képe

### 5.3.3. Szekvencia diagram



5.3. ábra. A  $\Lambda(s)$  grafikus képe



5.4. ábra. Venn diagram TIKZ segítségével

**5.3.1. Definíció.** Legyen  $(X, d)$  és  $(Y, \rho)$  két metrikus tér, legyen  $T : X \rightarrow Y$  egy leképezés. Azt mondjuk, hogy a  $T$  leképezés Lipschitz tulajdonságú, ha létezik egy olyan  $L > 0$  szám amelyre

$$\rho(Tx, Ty) \leq Ld(x, y) \quad \forall x, y \in X.$$

Az  $L$  számot Lipschitz állandónak nevezzük.

Ha  $T : X \rightarrow Y$  leképezés Lipschitz tulajdonságú, és az  $L < 1$  akkor a  $T$  operátort **kontrakciónak** nevezzük. Azt mondjuk, hogy  $x^* \in X$  fixpontja a  $T$  operátornak ha

$$Tx^* = x^*.$$

**5.3.1. Tétel.** *Banach féle fixponttétel Legyen  $(X, d)$  teljes metrikus tér és  $T : X \rightarrow X$  leképezés egy kontrakció az  $L < 1$  állandóval. Ekkor igazak a következő állítások:*

1.  *$T$ -nek egy és csakis egy  $x^*$  fixpontja.*
2. *Bárhogy választunk meg egy  $x_0 \in X$  elemet, a  $x_{k+1} = Tx_k$  sorozat konvergens és  $Tx_k \rightarrow x^*$ , ahol  $k$  természetes szám.*
3. *Igaz, hogy*

$$d(x_k, x^*) \leq \frac{L^k}{1 - L} d(x_0, Tx_0).$$



# Összefoglaló

Dolgozatomban differenciálegyenletek megoldásával foglalkoztam, amelyet különböző programozási technológiák segítségével valósítottam meg. Először ismertettem a differenciálegyenletek numerikus megoldásának elméleti alapjait, majd megvizsgáltunk és levezettünk három numerikus módszert az Euler-, a Runge-Kutta és a Dormand-Prince módszereket. Ezek közül a mai technológiákban leginkább használatos Dormand-Prince algoritmus esetében megnéztük, hogy milyen szoftverekben találhatjuk meg, mint alapértelmezett differenciálegyenlet megoldó. A továbbiakban ismertettem két modellt, a leukémia betegség alap modelljét és a hullámmozgás modelljét, ezzel is kihangsúlyozva a téma fontosságát, hogy mennyire fontos az időtényező bizonyos problémák egyenleteinek megoldásánál. Ezek után részletesen is megnéztük, hogy milyen szoftvereket alkalmaztam és alkottam a differenciálegyenletek és rendszerek megoldására. A szoftvereket két kategóriába osztottuk fel, az első a már létező szoftverek kategóriája, a másik pedig az általam megvalósított szoftverek csoportja volt. Az első kategóriában ismertettem két technológiát, a Matlab által nyújtott ode45 beépített megoldót és a Boost könyvtárcsomagban található Odeint nevű könyvtárat. Az általam írt szoftverek csoportjában négy megvalósítást mutattam be ezek a Matlab, Java, C++ és Android technológiák segítségével készültek. Emellett megnéztük, hogy mennyire hatékonyan lehet párhuzamosítani a differenciálegyenletek megoldását CUDA technológia segítségével és a grafikus kártyát (GPU-t) felhasználva. Végül kiértékeltek a tesztelés során kapott eredményeket és összehasonlítottuk a különböző programokat, kiemelve azok erősségeit és gyengéit. Majd levontuk a következtetéseket, hogy melyik technológia irányában érdemes tovább haladni és melyik az, amellyel nem éri meg foglalkozni.

Jövőbeli terveimet illetően szeretnék jobban elmerülni a GPU-n történő differenciálegyenletek megoldásának módszereiben, valamint ezek alkalmazását kipróbálni és tanulmányozni a parciális differenciálegyenletek területén (PDE). Továbbá érdemesnek tartom a C++ szoftver továbbfejlesztését és egy differenciálegyenlet megoldó könyvtár megalkotását, amely ingyenesen használható és nyílt forráskódú lenne.

# Köszönetnyilvánítás

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

# Ábrák jegyzéke

3.1. Egyszerű gráf TIKZ segítségével . . . . .	21
4.1. Gráf . . . . .	24
4.2. Gráf 2 . . . . .	24
4.3. Bifurkációs diagram . . . . .	25
4.4. Lorentz attraktor . . . . .	25
5.1. Az $x^3 - 5x$ függvény grafikus képe PGFPLOT-al . . . . .	31
5.2. A $\Psi$ grafikus képe . . . . .	31
5.3. A $\Lambda(s)$ grafikus képe . . . . .	32
5.4. Venn diagram TIKZ segítségével . . . . .	32
F.1.1A TeXstudio $\text{\LaTeX}$ -szerkesztő. . . . .	39

# Táblázatok jegyzéke

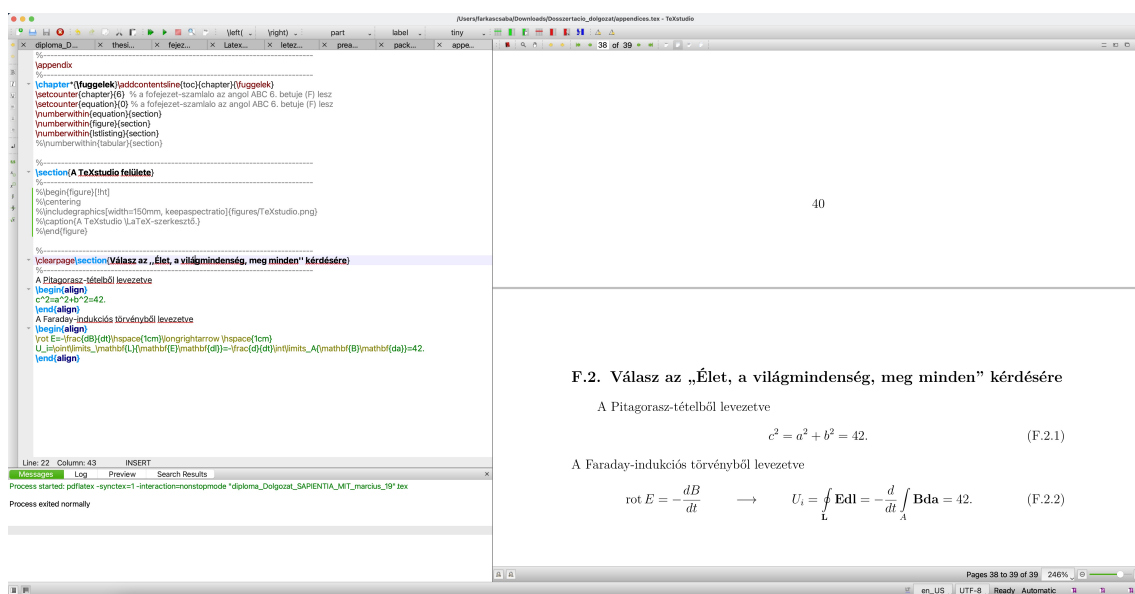
2.1.	Mérési eredmények $n = 10$ tesztesetre. . . . .	17
2.2.	Mérési eredmények $n = 100$ tesztesetre. . . . .	17
2.3.	Mérési eredmények $n = 5$ egyenlet esetén. . . . .	19
2.4.	Mérési eredmények $n = 10$ egyenlet esetén. . . . .	19

# Irodalomjegyzék

- [Ant07] Margit Antal. Toward a simple phoneme based speech recognition system. *Stud. Univ. Babeş-Bolyai Inform.*, 52(2):33–48, 2007.
- [K08] Zoltán Kátai. Dynamic programming as optimal path problem in weighted digraphs. *Acta Math. Acad. Paedagog. Nyházi. (N.S.)*, 24(2):201–208, 2008.
- [Knu11] Donald E. Knuth. *The Art of Computer Programming: Combinatorial Algorithms, Part 1*. Addison-Wesley Professional, 1st edition, 2011.
- [LS15] László Lovász and Balázs Szegedy. The automorphism group of a graphon. *J. Algebra*, 421:136–166, 2015.

# Függelék

## F.1. A TeXstudio felülete



F.1.1. ábra. A TeXstudio L<sup>A</sup>T<sub>E</sub>X-szerkesztő.

## F.2. Válasz az „Élet, a világmindenség, meg minden” kérdésére

A Pitagorasz-tételből levezetve

$$c^2 = a^2 + b^2 = 42. \quad (\text{F.2.1})$$

A Faraday-indukciós törvényből levezetve

$$\text{rot } E = -\frac{dB}{dt} \quad \longrightarrow \quad U_i = \oint_{\mathbf{L}} \mathbf{E} d\mathbf{l} = -\frac{d}{dt} \int_A \mathbf{B} d\mathbf{a} = 42. \quad (\text{F.2.2})$$