

BABEȘ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION [Secție]

DIPLOMA THESIS

[Titlu lucrare]

Supervisor
[Grad, Lorand Gabriel Parajdi]

Author
Stan Ioan-Victor

ABSTRACT

Abstract: un abstract în engleză sau în română un rezumat în limba engleză
cu prezentarea, pe scurt, a conținutului pe capitole, punând accent pe contribuțiile proprii și
originalitate

Contents

1	Introduction	1
2	Introduction to Chemical Reaction Networks	2
3	Dynamical Systems	3
3.1	Defining ODEs	3
3.2	Forming ODE systems	4
3.3	Applications of dynamical systems and classifications	4
4	Existence and Absence of Hopf Bifurcation in Phosphorylation–Dephosphorylation CRN	5
4.1	What is a Phosphorylation–Dephosphorylation CRN?	5
4.2	What is a Hopf Bifurcation?	5
4.3	Putting them together	5
5	CoNtRoL Simulations Web Application	6
5.0.1	Overview of the technologies	6
5.0.2	Running it	6
5.0.3	The bottom line	11
	Bibliography	12

Chapter 1

Introduction

Introducere: obiectivele lucrării și descrierea succintă a capitolelor, prezentarea temei, prezentarea contribuției proprii, respectiv a rezultatelor originale și menționarea (dacă este cazul) a sesiunii de comunicări unde a fost prezentată sau a revistei unde a fost publicată.

Chapter 2

Introduction to Chemical Reaction Networks

Chapter 3

Dynamical Systems

3.1 Defining ODEs

Definition 1. An *ordinary differential equation* is an equation of an unknown function of one variable. This can be expressed as a function of this unknown function and its various derivatives.

Its general form looks something like:

$$F(t, y(t), y'(t), \dots, y^{(n)}(t)) = 0, \quad (3.1)$$

Where $y(t)$ is the unknown function of independent variable t and $F : \Omega \rightarrow \mathbb{R}$, $\Omega \subseteq \mathbb{R}^{n+1}$. This would be what's called the implicit form of the differential equation.

Definition 2. The *order* of an ODE is the highest order of the derivative present in the equation.

In our case, the order is n . If, however F satisfies the regularity condition of the **implicit function theorem** then the equation can be written in a much more digestible form.

Restricting the domain Ω to one that allows the implicit form to be represented as a function of the form

$$y^{(n)}(t) = f(t, y(t), y'(t), \dots, y^{(n-1)}(t)) \quad (3.2)$$

would yield what's called the **explicit** form of the ODE.

The **implicit function theorem** allows one to convert a relation (implicit form) to functions of some variables. These functions may not be unique, but together, their graph may locally satisfy the relation.

As an example: The relation for a cardioid cannot be expressed as a sole function, we'd instead need the union of the graph of two separate functions for expressing it. (si aici gen faci tu niste graphuri cu maple sau alte d-astea vezi cum faci virtual machineu ala sau mergi la faculta la un pc cu maple si iti faci lista cu ce vrei sa graphuiesti)

Reminder: A function $f(x)$ is continuously differentiable if $\exists f'(x)$ and $f'(x) \in C^n$ where $n \geq 0$

Theorem 1. Let $F : D \subseteq \mathbb{R}^{n+1} \rightarrow \mathbb{R}$ be a continuously differentiable function with the relation that $F(t, y(t), y'(t), \dots, y^{(n)}(t)) = 0$. Let us express a point in the set $\mathbb{R}^{n+1} = \mathbb{R} \times \mathbb{R}^n$ as $(t, \mathbf{Y}) = (t, y, y', \dots, y^{(n)})$ and fix one such point s.t. $F(t, \mathbf{Y}) = 0$. So $\exists \square_{(t, \mathbf{Y})} \ni U \times V \subseteq D$ s.t. $F \in C^1(U \times V)$ and $\frac{\partial F}{\partial y}(t, \mathbf{Y}) \neq 0$. Then this means $\exists \square_t \ni U_0 \subseteq U, \square_{\mathbf{Y}} \ni V_0 \subseteq V$ and a function $f : U_0 \rightarrow V_0$ s.t. $f(t) = \mathbf{Y}$ and $F(t, f(t)) = 0, \forall t \in U_0; f \in C^1(U_0)$ and $\frac{\partial f}{\partial t_i} = -\frac{\frac{\partial F}{\partial t_i}(t, f(t))}{\frac{\partial F}{\partial y}(t, f(t))}, \forall i = \overline{1, n}, \forall t \in U_0$ and $F \in C^1(U \times V), K \in \mathbb{N}^* \Rightarrow f \in C^K(U_0)$.

A **solution** is an expression of the unknown function $y(t)$ which satisfies the relation $y^{(n)}(t) = f(t, y(t), y'(t), \dots, y^{(n-1)}(t))$. A **general solution** includes all of these functions and usually has some constants of integration in the expression, while a **particular solution** doesn't have them. A particular solution is usually found if we also have some initial conditions given for the unknown function, like $y(a) = b, y'(a) = c$ for some $a, b, c \in \mathbb{R}$. Or, more formally:

Definition 3. A function $y : I \rightarrow \mathbb{R}$ is a solution of the equation 3.1 if the following conditions are met:

1. $I \subseteq \mathbb{R}$ is nondegenerate interval. ($|I| > 1$)
2. $y \in C^n(I)$ and $(t, y(t), y'(t), \dots, y^{(n)}(t)) \in D_f, \forall t \in I$.
3. $y^{(n)}(t) = f(t, y(t), y'(t), \dots, y^{(n-1)}(t)), \forall t \in I$.

3.2 Forming ODE systems

3.3 Applications of dynamical systems and classifications

Chapter 4

Existence and Absence of Hopf Bifurcation in Phosphorylation–Dephosphorylation CRN

- 4.1 What is a Phosphorylation–Dephosphorylation CRN?
- 4.2 What is a Hopf Bifurcation?
- 4.3 Putting them together

Chapter 5

CoNtRoL Simulations Web Application

Here we will present the open-source Web application developed for easing work involving chemical reaction networks, hopefully for students and researchers one day. It can be used for obtaining numerical analysis of chemical reaction networks, as well as plotting species against time, one another and so on. We'll present the technologies used throughout the project, how to run it locally and a couple of usage examples involving what we've presented in the previous chapters.

5.0.1 Overview of the technologies

The website is a back-end application built in **Python**, a programming language known for its use in basically every single science, including natural sciences so it's a no-brainer when it comes to plotting. The web server is built using **Flask**, a lightweight web app framework and the webpages served are server-side rendered by Flask's template engine dependency - **Jinja**.

The crux of the functionality is aided by the Python library **Tellurium**; which is, as their docs say; "A Python Environment for Reproducible Dynamical Modeling of Biological Networks". It uses a subset of the Systems Biology Markup Language (**SBML**) called **Antimony** which can be used in this app to create a Chemical Reaction Network, as well as the friendlier selects form. So the bits doing the magic are the calls to `road_runner.loada()` which are used to *load* antimony code into the model. the `road_runner..simulate()` function is then used for running and obtaining simulation data, followed by `road_runner.plot()`, which in turn calls a `matplotlib` headless backend for writing the plotted results to a file.

5.0.2 Running it

As explained in the `README` of the project, running locally is done automatically by the `run_script.sh`, which figures out your machine's local IP, sets the required environment variables and runs `flask --debug run --host="$ip"`. Output regarding traffic to the server as well as internal workings of the app will now be redirected to `stdout` of the terminal.

All the user has to do beforehand is:

- clone the project

```
1 git clone https://github.com/viktorashi/Open-CoNtRol.git
```

- change directory into it.

```
1 cd Open-CoNtRol
```

- install the requirements

```
1 pip install -r requirements.txt
```

- give the `run_script.sh` execute permissions

```
1 chmod +x ./run_script.sh
```

- and finally run it

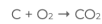
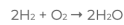
```
1 ./run_script.sh
```

Among the wall of output will also be the line showing the address your server is located at, for example: * Running on `http://192.168.0.94:5000`, address at which you'll be greeted with this screen

Chemical Reaction Network simulation tool (CoNtRol-Sim)

Chemical reaction network theory is an area of applied mathematics that attempts to model the behavior of real-world chemical systems. Since its foundation in the 1960s, it has attracted a growing research community, mainly due to its applications in biochemistry and theoretical chemistry

A chemical reaction network (CRN) comprises a set of reactants, a set of products and a set of reactions. For example, the pair of combustion reactions:



[Here](#) you can write [Antimony](#) code if you're into that

+

-

--

⇌

Get Numerical Analysis

You can either use this as a starting point or the page located at the `/antimony` path:

Chemical Reaction Network simulation tool (CoNtRol-Sim)

You can either use the following textbox to write [Antimony](#) [↗] code, or the +/- dropdowns below to fill out the CRN.

It's optional to also write the initial values for each species / reaction constant, if not filled out the equations and stoichiometric matrix will be shown.

If filled out inside the textbox, you'll be prompted to choose what type of graph do you want represented. **WARNING! THIS MEANS THE OTHER INPUT FORM WILL BE LOST ALONG WITH EVERYTHING YOU WROTE INSIDE OF IT**

```
2H2 + O2 -> 2H2O; k1*H2*H2*O2
C + O2 -> CO2; k2*C*O2
```

Get Numerical Analysis

Both of these redirect to `/numerical_analysis` analysing the system ... well, numerically. As an example, this Antimony code:

```
1 S0 -> KS1; k1*S0
2 KS1 -> S2; k2*KS1
3 S2 + F -> FS2; k3*S2*F
4 FS2 -> F; k4*FS2
5 F -> S0 + F; k5*F
```

and its longer to write alternative:

Chemical Reaction Network (CRN)

In each txt file, we have represented a CRN with 1, 2 or 3 species and below the time of representation of graph

crn.txt

Start Time

End Time

Title of the Graph
Time

x_title
time

y_title
concentration

A, B, C represent the species of the chemical reaction network together with the initial values of each species

F

FS2

KS1

S0

S2

k1, k2, k3 represent the constants of the each reaction in CRN and their values

k1

k2

k3

k4

k5

Generate Graph

from which we fill out the initial values of the concentrations for each species as, well as the reaction rates. So given, for example the values:

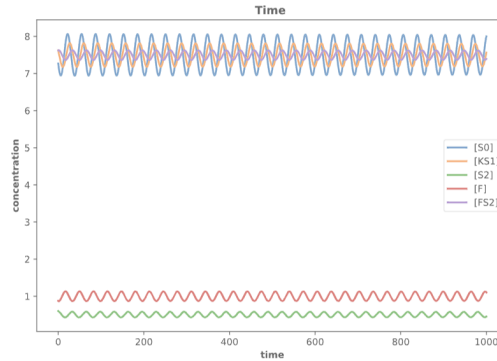
```

1  F = 0.874108
2  FS2 = 7.620157734
3  KS1 = 7.620157734
4  S0 = 7.270157734
5  S2 = 0.6000000000
6
7  k1 = 0.1329759342
8  k2 = 0.1329759342
9  k3 = 2
10 k4 = 0.1329759342
11 k5 = 1

```

outcomes the graph:

Chemical Reaction Network (CRN) - 2D



Generate new Graph

Chemical Reaction Network

$S0 \rightarrow KS1; k1*S0$
 $KS1 \rightarrow S2; k2*KS1$
 $S2 + F \rightarrow FS2; k3*S2 * F$
 $FS2 \rightarrow F; k4*FS2$
 $F \rightarrow S0 + F; k5*F$

$k1 = 0.1329759342;$
 $k2 = 0.1329759342;$
 $k3 = 2;$
 $k4 = 0.1329759342;$
 $k5 = 1;$

$F = 0.874108;$
 $FS2 = 7.620157734;$
 $KS1 = 7.620157734;$
 $S0 = 7.270157734;$
 $-----$

Stoichiometric Matrix

$S0$	-1	0	0	0	1
$KS1$	1	-1	0	0	0
$S2$	0	1	-1	0	0
F	0	0	-1	1	0
$FS2$	0	0	1	-1	0

5.0.3 The bottom line

So this is how the workflow of the app typically goes.

write out your system \rightarrow get numerical analysis \rightarrow choose your desired graph
 Voilà \rightarrow fill out data values

Bibliography