VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
MATEMATINĖS INFORMATIKOS KATEDRA

Henryk Česnolovič *parašas*
Bioinformatikos studijų programa

Paprastas karkasas biosistemų tyrimams ir jo taikymai
Simple Framework for Biosystems Researches and its Applications

Bakalauro baigiamasis darbas

Vadovas: Lektorius **Irus Grinis** *parašas*

Vilnius 2016

Turinys

Įvadas

Biosistemas galima tirti įvairiais metodais. Vienas svarbiausių yra matematinis modeliavimas. Naudojant matematinius modelius galima labai lengvai keisti modelio parametrus ir matyti kaip sistema elgiasi. Dažniausiai skaičiavimai vyksta žymiai greičiau negu realūs eksperimentai. Be to, jie būna žymiai pigesni.

Savo ruožtu matematiniai modeliai gali būti įvairūs. Šiame darbe mes nagrinėsime biosistemas, kurias galima tirti naudojant paprastų diferencialinių lygčių sistemas. Šiems modelių tyrimams galima naudoti tokius įrankius, kaip matlab, maple. Kurse *Principles of Synthetic Biology*[1] buvo pateikti matlab kalba parašyti paketai, kurių pagalba galima atlikti biosistemų tyrimus. Kadangi matlab licensija nėra pigi, todėl iškėliau tikslą padaryti šitą karkasą atviro kodo pasinaudojus python galimybėmis.

Darbo tikslas

- Sukurti įrankį, kuris sugebės analizuoti tam tikros klasės biologinius modelius, bei pateikti kelis pavyzdžius įrankio panaudojimui.

Rezultatai, kurių tikimasi

- Sukurti paprastą karkasą, kuris spręs diferencialines lygčių sistemas.
- Įrodyti karkaso korektiškumą palyginant su jo pagalba gaunamus rezultatus su kaikuriose moksliniuose straipsniuose gautais rezultatais.

1 Biosistemų tyrimai

Biosistemų tyrimai tapo populiarūs biologijoje maždaug 2000-jų metų pradžioje.

Pirmuosius sisteminės biologijos tyrimus galima matyti tokiose srityse:

- fermentų kinetikoje;
- populiacijos dinamikos matematinėse modeliuose;

Pirmasis matematinis modelis ląstelių biologijoje buvo sukurtas 1952 metais Britiško fiziologo ir Nobelio laureato Alan'o Lloyd Hodgkin'o ir Andrew Fielding Huxley. 1960 metais Denis Nobel sukūrė pirmą elektrokardiostimuliatoriaus kompiuterinį modelį. Sistemų biologijos sąvoka buvo įvesta 1966 metais Mihailo Meserovic. Disciplinos tikslas yra projektavimas ir analizė realių ir sintetinių biologinių, biocheminių procesų, pasitelkus matematiniais modeliais.

1.1 Klasikinis fermentų kinetinis modelis

Išnagrinėsime fermentų reakcijos modelį:

$S + F \underset{k_2}{\overset{k_1}{\rightleftharpoons}} P$

Šitą modelį galima suskaidyti į dvi dalis:

1. $S + F \xrightarrow{k_1} P$
2. $S + F \xleftarrow[k_2]{} P$

Iš pirmo modelio gauname:

$$\begin{cases} \frac{dP}{dt} = S \cdot F \cdot k_1 \\ \frac{dS}{dt} = -S \cdot F \cdot k_1 \\ \frac{dF}{dt} = -S \cdot F \cdot k_1 \end{cases}$$

Antroje dalyje iš produkto P gauname substratą S ir fermentą F su greičiu k_2. Reikia pastebėti, kad šiuo atveju $k_2 << k_1$. Antros dalies diferencialinių lygčių sistema atrodys taip:

$$\begin{cases} \frac{dP}{dt} = -P \cdot k_2 \\ \frac{dS}{dt} = P \cdot k_2 \\ \frac{dF}{dt} = P \cdot k_2 \end{cases}$$

O sujungę dvi dalis gausime:

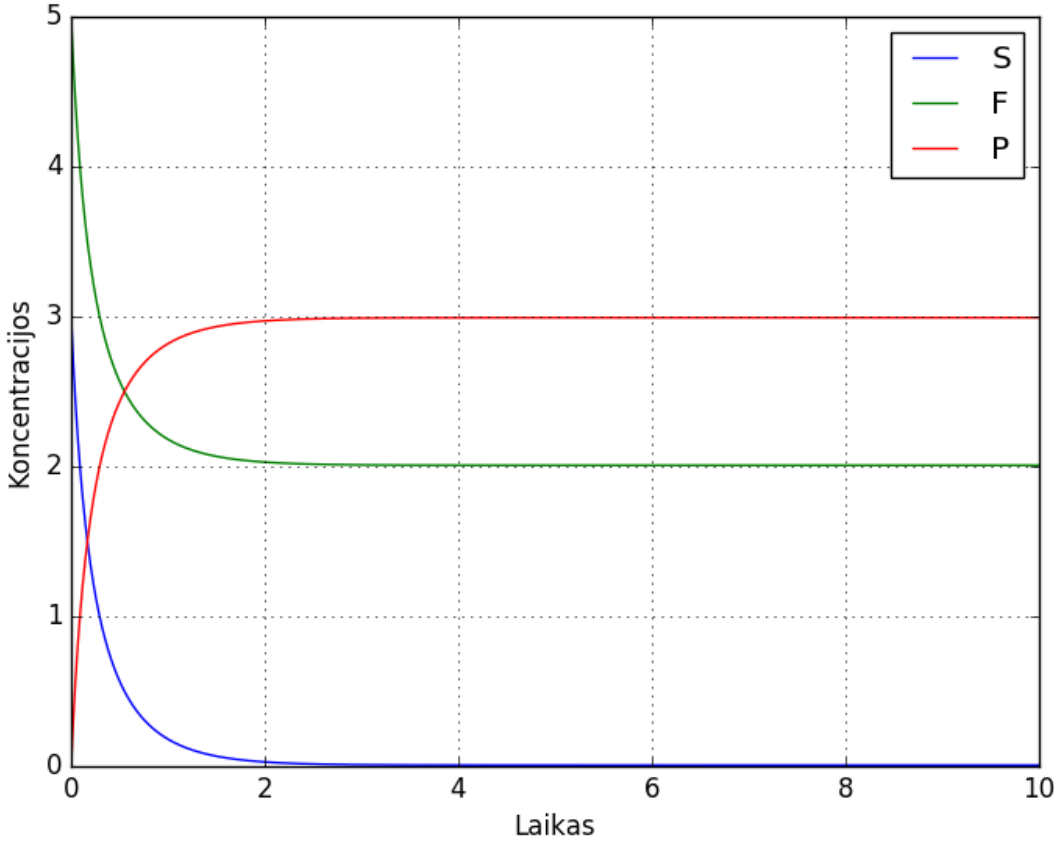
$$\begin{cases} \frac{dP}{dt} = S \cdot F \cdot k_1 - P \cdot k_2 \\ \frac{dS}{dt} = -S \cdot F \cdot k_1 + P \cdot k_2 \\ \frac{dF}{dt} = -S \cdot F \cdot k_1 + P \cdot k_2 \end{cases}$$

Tarkime, kad pradiniu laiko momentu $t = 0$:

- $S_0 = 3$
- $F_0 = 5$
- $P_0 = 0$

Modelio greičio konstantas parinksime tokias: $k_1 = 1$, $k_2 = 0.005$. Sprendžiame sistemą kai $0 \leq t \leq 10$.

Pav. 1: Koncentracijų kitimas fermentinėje kinetikoje



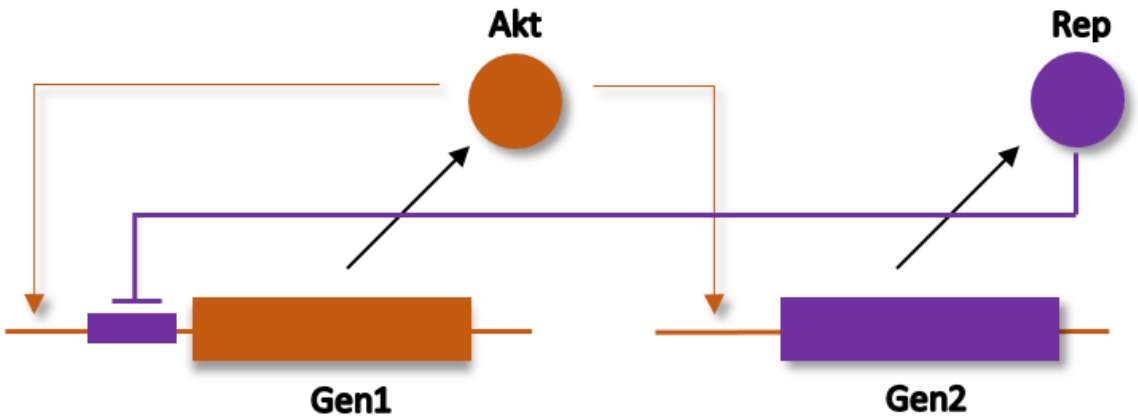
1.2 Nagrinėjamų modelių struktūra

Apibendrinant ką tik tai nagrinėtą pavyzdį, galima pamatyti, kad dideliai biosistemų, kurių virsmas aprašomas paprastų diferencialinių lygčių pagalba, klasei galima suskirtyti į penkias dalis:

1. Reakcijų greičiai;
2. Greičių konstantos;
3. Kompozitoriai;
4. Sistemos dalys;
5. Sistema.

Kaip pavyzdį, iš straipsnio[3], kuriame yra išanalizuoti du modeliai, paimsime pirmąjį modelį ir suskirtysime į ankščiau paminėtas dalis. Šitame modelyje (žiurėti Pav. 2) yra nagrinėjami du genai, aktyvatorius ir represorius. Aktivatorius veikia abu genus, o represorius tik pirmą.

Pav. 2: Genetinis dviejų elementų osciliatorius



$$\begin{cases} \frac{dx}{dt} = \Delta(\xi_x \frac{1+\rho x^2}{1+x^2+\sigma y^2} - x) \\ \frac{dy}{dt} = \Delta\xi_y \frac{1+\rho x^2}{1+x^2} - y \end{cases}$$

1.2.1 Reakcijų greičiai

Biologiniame modelyje reakcijos greitis – tai koncentracijos kitimo pagal laiką aprašymas. Nagrinėjamas atvejis turi dvi reakcijas. x atveju tai
$\frac{dx}{dt} = \Delta(\xi_x \frac{1+\rho x^2}{1+x^2+\sigma y^2} - x)$, o y - $\frac{dy}{dt} = \Delta\xi_y \frac{1+\rho x^2}{1+x^2} - y$.

1.2.2 Greičių konstantos

Konstantos yra sistemos dalys, kurios nekinta. Mūsų atveju tai bus reakcijų greičio konstantos. Šitame modelyje turime penkias konstantas:

1. Δ – santykis tarp aktyvatoriaus ir represoriaus;
2. σ – represoriaus stiprumas;
3. ρ – baltymo produkcija;
4. ξ_x – aktyvatoriaus efektyvumas;
5. ξ_y – represoriaus efektyvumas.

1.2.3 Kompozitoriai

Kompozitoriai – tai medžiagų koncentracijos. Šiuo atveju tai x ir y.

1.2.4 Dalys

Dalys apima kompozitorių ir reakcijų greičių išraiškų asociaciją. Šiuo atveju tai bus kompozitorių x ir y sujungimas su jiems atitinkančioms reakcijų greičių išraiškoms.

1.2.5 Sistema

Biologinė sistema – tai visuma, kuri apjungia visas išvardintas modelio dalis. Pradžioje sistema gavus visas dalis sujungia kompozitorius su koncentracijų kitimo greičiais. Taip susidaro diferencialinių lygčių sistema. Tada greičio išraiškuose yra pakeičiamos greičio konstantos į jų reikšmes. Sistemai reikia nurodyti sutartinį pradinį ir galutinį laiką.

2 Paprastas programinis karkasas skirtas tam tikros klasės biosistemų tyrimams

Iš *EDx* kurso[1] buvo paimtos matlab klasės[2], kurių pagalba galima buvo tirti biologinius modelius. Šį karkasą sudaro tokios dalys: BioSystem, Compositor, Const, Rate, Part.

2.1 Python moduliai

Kadangi pagrindinis darbo tikslas buvo perrašyti karkasą iš matlab į python, tam prireikė tokių modulių:

- *SymPy*;
- *SciPy*;
- *NumPy*;
- *matplotlib*.

2.1.1 SymPy

Pilnai python kalba parašytas modulis skirtas simbolinei matematikai. SymPy paskirtis yra gebėti atlikti tam tikrus kompiuterinės algebros skaičiavimus.

Symbols – metodas pakeičia tekstinės eilutės dalis į simbolius, pavyzdžiui, $x = symbols('x')$. Jeigu toliau parašysime, tarkime, $5 + x$, tai programa matys tai kaip išraišką.

parse_expr – metodas, kuris gauna tekstinę eilutę ir apdoroja ją, kaip matematinę išraišką. Tai yra, visas matematines išraiškas paskaičiuoja, o kintamieji palieka. Pavyzdžiui, $parse\_expr('3**2')$ yra lygu 9, o $parse\_expr('x**3') = x^3$.

Subs – išraiškoje pakeičia simbolinius kintamuosius reikšmėmis arba kitais simboliniais kintamaisiais. Pavyzdžiui, parašius $parse\_exp('3**x').subs('x', 3)$ gausime rezultatą 27.

Lamdify – pertvarko funkciją taip, kad būtų lengva ja naudotis. Tarkime sukuriame f: $f = lamdify(['x','y'], parse\_expr('x**y'))$. Vėliau kviečiant $f(x, y)$, galima nurodyti skaičius ir gauti rezultatą. Pavyzdžiui, $f(2, 3)$ bus lygu 8.

2.1.2 NumPy

NumPy – vienas iš pagrindinių modulių moksliniams skaičiavimams. Karkase panaudotas NumPy tik dviem atvejais. Tai NumPy tipo masyvai gaunami po integravimo ir metodas *linspace*.

$linspace(x0, x1, num = n)$ – metodas, kuriuo pagalba galima lengvai padaryti skaičių masyvą. Pavyzdžiui, įvykdžius komandą $vekt = numpy.linspace(1, 10, num = 10)$ *vekt* kintamajame yra masyvas $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$.

2.1.3 SciPy

Paketas skirtas matematiniams, inžinieriniams ir moksliniams tikslams. Šiuo paketo pagalba buvo sprendžiamos diferencialinės lygtys ir sistemos. Panagrinėkime du plačiai naudojamus diferencialinių lygčių integravimų metodus.

scipy.integrate.odeint - metodas skirtas diferencialinių lygčių sistemų integravimui pasinaudojant *ISODA* biblioteką, kuri buvo parašyta *FORTRAN* programavimo kalba. Pagrindiniai parametrai, kuriuos reikia paduoti šiam metodui – tai funkcija, pagal kurią integratorius skaičiuos atitinkamų išraiškų reiškmes, pradines reikšmes, bei laiko taškų sąrašą.

scipy.integrate.ode - metodas gali skaičiuoti diferencialines lygčių sistemas įvairiais metodais:

- vode;
- zvode;
- Isoda;
- dopri5;
- dop853.

Programoje, po tam tikro testavimo, buvo pasirinktas metodas *dopri5* - tai yra Runge-Kutta eilės 4(5) metodas. Pirmiausiai reikia nustatyti integratoriaus metodą.

```
solver = ode(f).set_integrator('dopri5')
```

f – tai funkcija pagal kurią integratorius skaičiuos reikšmes. Tada reikia nustatyti pradines koncentracijas ir pradinį laiką.

```
solver.set_initial_value(y0, t0)
```

Vėliau nurodome iki kokio laiko mes norime skaičiuoti $t1$ ir kokie yra laiko žingsniai Δt. Paleidžiame ciklą.

```
while solver.successful() and solver.t < t1:
        print(solver.t, solver.integrate(r.t+dt))
```

2.1.4 matplotlitb

Modulis naudojamas 2D grafikų vizualizacijai.

plot – x ir y tai skaičių masyvai, kurie aprašo taškų padėtį grafike. *Label* yra to grafiko dalies vardas.

title – grafiko vardas.

grid – užmeta tinklą ant grafiko.

xlabel – nustato vardą x ašiai.

ylabel – nustato vardą y ašiai.

show – nupiešia grafiką.

2.2 Sukurto karkaso struktūra

Karkasas susideda iš kelių klasių:

1. Constant.py
2. Rate.py
3. Compositor.py
4. Part.py
5. BioSystem.py

2.2.1 Constant.py

Šioje klasėje yra saugomos reikšmės, kurios nekinta laike. Dažniausiai tai būna reakcijų greičių konstantos. Sukuriant naują šios klasės objektą paduodame du parametrus: konstantos vardą ir reikšmę. Klasės objektas turi tokius laukus:

1. name – konstantos vardas naudojamas BioSystem.py klasėje sujungimui su jo įkėlimo indeksu;
2. sym – konstantos pavadinimas simbolinio tipo naudojamas išraiškoje keičiant simbolį į reikšmę;
3. value – konstantos reikšmė.

```
1 from sympy import *
2
3 class Const(object):
4
5 def __init__(self, name, value):
6         self.name = name
7         self.sym = symbols(name)
8         self.value = value
```

Objekto sukūrimas:

```
new_compositor = Compositor('k', 0.5)
```

2.2.2 Rate.py

Klasė **Rate** skirta tam, kad saugotų diferencialinių lygčių sistemos atskiros lygties dešinės pusės dalies tekstinės eilutės pavidalu kintamajame *rate_string*.

```
1 class Rate(object):
2
3 def __init__(self, rate):
4         self.rate_string = str(rate)
```

Objekto sukūrimas:

```
Rate('-k * A * M')
```

2.2.3 Compositor.py

Šita klasė yra skirta tų modelio dalių saugojimui, kurių reikšmės kinta laike. Biologinėje sistemoje tai gali būti pavyzdžiui baltymas, mRNA. Sukuriant objektą reikia paduoti kompozitoriaus vardą ir reikšmę. Reikšmė tai pradinė koncentracija. Taip pat šita klasė turi du metodus. Vienas metodas yra skirtas apdoroti tekstinės eilutės tipo išraišką į symbolic tipą naudojant *sympy.parsing.sympy_parser* galimybes ir išsaugoti kintamajame *rate*. Antras metodas *set_initial_value* skirtas yra pradinės koncentracijos reikšmės pakeitimui. Dar vienas kintamasis tai *ratef*. Jame yra saugoma *lamdify* tipo funkcija.

```
from sympy import *
from sympy.parsing.sympy_parser import *

class Compositor(object):
        rate = symbols("0")
        ratef = None

        def __init__(self, name, init_value):
                self.name = name
                self.sym = symbols(name)
                self.value = init_value
                self.init_value = init_value

        def AddRate(self, new_rate):
                self.rate = self.rate + parse_expr(new_rate.rate_string)

        def SetInitialValue(self, init_value):
                self.init_value = init_value
```

Objekto sukūrimas:

```
dMdt = sys.add_compositor('M', 1)
```

2.2.4 Part.py

Klasė **Part** yra atsakinga už visą diferencialinių lygčių sistemos informacijos saugojimą. Kuriant šios klasės objektą reikia paduoti tris parametrus:

1. *name* – dalies vardas;
2. *compositors* – kompozitoriai;
3. *rates* – atitinkamai kiekvienam kompozitoriui diferencialinės lygties reakcijų greičius naudojant **Rate** objektus.

```
class Part(object):
        def __init__(self, name, compositors, rates):
                self.name = name
                self.compositors = compositors
                self.rates = rates
```

Objekto sukūrimas:

```
Part('A + M -k> B + M',
          [dAdt, dBdt, dMdt],
          [Rate('-k * A * M'), Rate('k * A * M'), Rate('0')])
```

2.2.5 BioSystem.py

BioSystem yra pagrindinė klasė, kuri apdoroja visą informaciją, saugoma kitose klasėse bei vykdo integravimą. Klasės konstruktoriui nereikia paduoti jokių argumentų. Kuriant klasės objektą sukuriami dict ir list tipo kintamieji, kurie saugo visų kitų klasės objektus, bei jos suindeksoja eilės tvarka. *add_compositor*, *add_constant*, *add_part*, metodai yra atsakingi už greičių kontantų, kompozitorių ir dalių sukūrimą bei išsaugojimą į sąrašą. *compositor_idex* metodas, paduodant kompozitoriaus vardą kaip parametrą, grąžina jo indeksą iš *map_compositors* kintamojo.

change_constant_value – keičia greičių konstantos reikšmę.

change_init_value – keičia kompozitoriaus pradinį koncentracijos kiekį.

sys_ode ir *sys_ode_int* metoduose įteruojami kompozitoriai, kurių *ratef* funkcijoms perduodami koncentracijos kiekiai.

run_ode metode yra iškviečiama *determine_rates* metodas ir vėliau vykdomas integravimas. Šiuo atveju yra naudojamos *SciPy* modulio *ode* galimybės. Šiam metodui reikia paduoti laiko pradžios, pabaigos ir laiko žingsnio parametrus. Metodas gražina du *NumPy* masyvus:

1. laiko masyvą;
2. laiko atitinkantį koncentracijų masyvą.

run_ode_int atveju naudojamas *odeint* metodas. Kaip parametrą reikia paduoti laiko masyvą, o gražinamos reikšmės bus tik koncentracijų kiekių masyvas.

```
from sympy import *
import numpy as np
from scipy.integrate import *
from scipy import *

from Compositor import Compositor
from Const import Const
from Rate import Rate
from Part import Part
```

```
10
11 class BioSystem(object):
12
13         def __init__(self):
14                 ...
15
16         def add_compositor(self, *args):
17                 ...
18
19         def compositor_index(self, name):
20                 ...
21
22         def add_part(self, new_part):
23                 ...
24
25         def add_constant(self, *args):
26                 ...
27
28         def determine_rates(self):
29                 ...
30
31         def reset_rates(self):
32                 ...
33
34         def change_constant_value(self, name, value):
35                 ...
36
37         def change_init_value(self, name, value):
38                 ...
39
40         def reset_state_variables(self):
41                 ...
42
43         def sys_ode_int(self, y, t):
44                 ...
45
46         def run_ode_int(self, tspan):
47                 ...
48
49         def sys_ode (self, t, y):
50                 ...
51
52         def run_ode (self, t0, dt, t1):
53                 ...
```

2.3 Naudojimas

Kaip pavyzdį paimsime paprastą reakciją. Tarkime yra kažkokių A ir B koncentracijos. M yra fermentas. Reakcijoje yra paverčiama A į B. Reakcija atrodo taip:

$$\frac{dA}{dt} = -k \cdot A \cdot M$$
$$\frac{dB}{dt} = k \cdot A \cdot M$$
$$\frac{dM}{dt} = 0$$

Pirmiausia reikia sukūrti BioSystem klasės objektą (kintamasis *sys*).

```
sys = BioSystem()
```

Vėliau galima pridėti konstantą k. Šiuo atveju k yra greičio konstanta.

```
sys.add_constant('k', 0.1)
```

Taip pat sukuriami trys kompozitoriai A, B ir M. Jiems reikia priskirti pradines koncentracijas.

```
1 dAdt = sys.add_compositor('A', 10)
2 dBdt = sys.add_compositor('B', 0)
3 dMdt = sys.add_compositor('M', 1)
```

Toliau aprašomos **Part**, t.y. pati sistema.

```
1   reaction = Part('A + M -k> B + M',
2                   [dAdt, dBdt, dMdt],
3                [Rate('-k * A * M'), Rate('k * A * M'), Rate('0')])
4   sys.add_part(reaction)
```

Metodui *run_ode_int* reikia nurodyti laiko sąrašą. Laiko masyvui sukurti naudosime *linspace* metodą. Tam, kad pasinauduoti *odeint* integravimu reikia iškviesti *run_ode_int* metodą. **BioSystem**'os bibliotekos *run_ode_int* gražinamos reikšmės yra visų kompozitorių koncentracijų kiekiai nurodytame laike.

```
1   time = np.linspace(0, 100, num = 1000)
2   y = sys.run_odeint(time)
```

Naudojant paprastą *ode* metodą nurodoma laiko pradžia, laiko žingsnis ir laiko pabaiga. Gražinamos reikšmės šiuo atveju yra du masyvai. Vienas masyvas yra laiko, kitas koncentracijų kiekiai.

```
1   solution = sys.run_ode(0, 1, 100)
2   y = solution[0]
3   time = solution[1]
```

Rezultatą atvaizduoti pasinaudosime python *matplotlib* moduliu. Abiem atvejais naudojamas tas pats kodas.

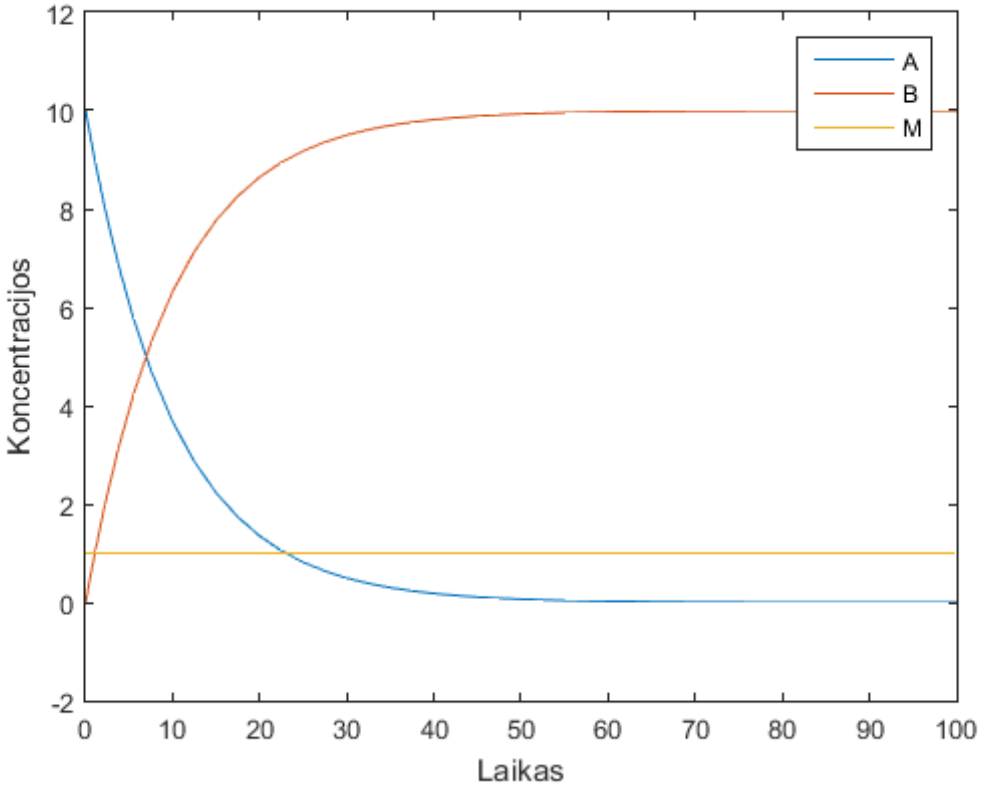
```
1   A = list()
2   B = list()
3   M = list()
```

```
4
5   for i in range(0, len(y)):
6       A.append(y[i][sys.compositor_idex('A')]);
7       B.append(y[i][sys.compositor_idex('B')]);
8       M.append(y[i][sys.compositor_idex('M')]);
9
10  plt.plot(time, A, 'b', label='A')
11  plt.plot(time, B, 'g', label='B')
12  plt.plot(time, M, 'r', label='M')
13
14  plt.legend()
15  plt.xlabel('Laikas')
16  plt.ylabel('Koncentracijos')
17  plt.axis((0,100,-2,12))
18  plt.show()
```

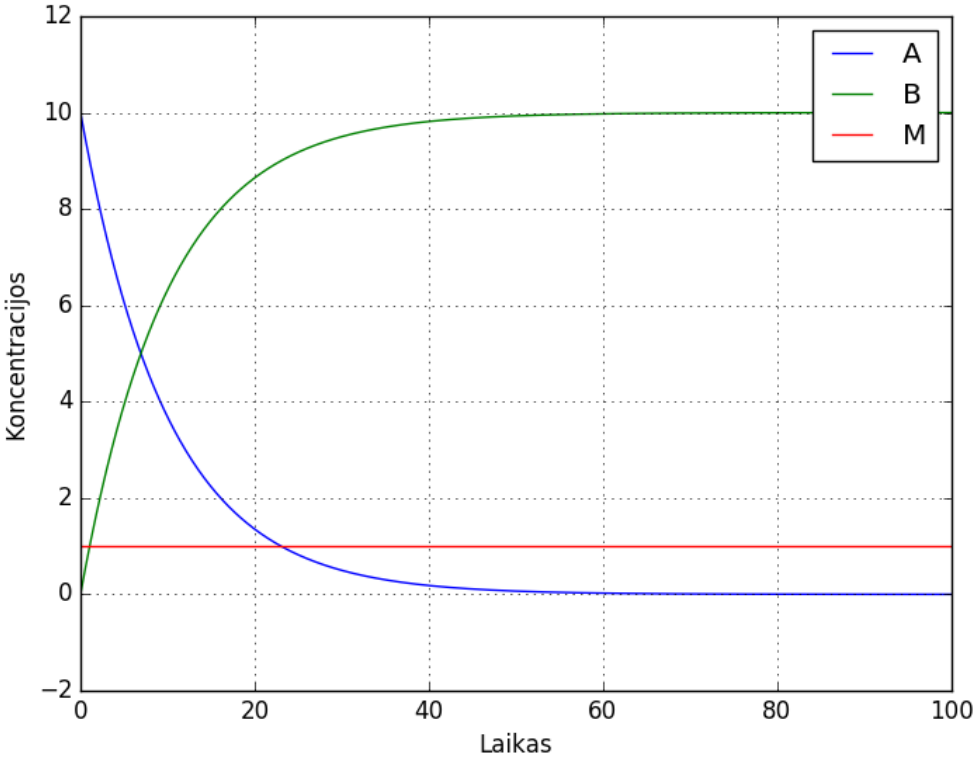
2.4 Grafikai

Lyginant matlab'o programą, *ode* ir *odeint* metodus galima teigti, kad rezultatai gaunasi panašūs.

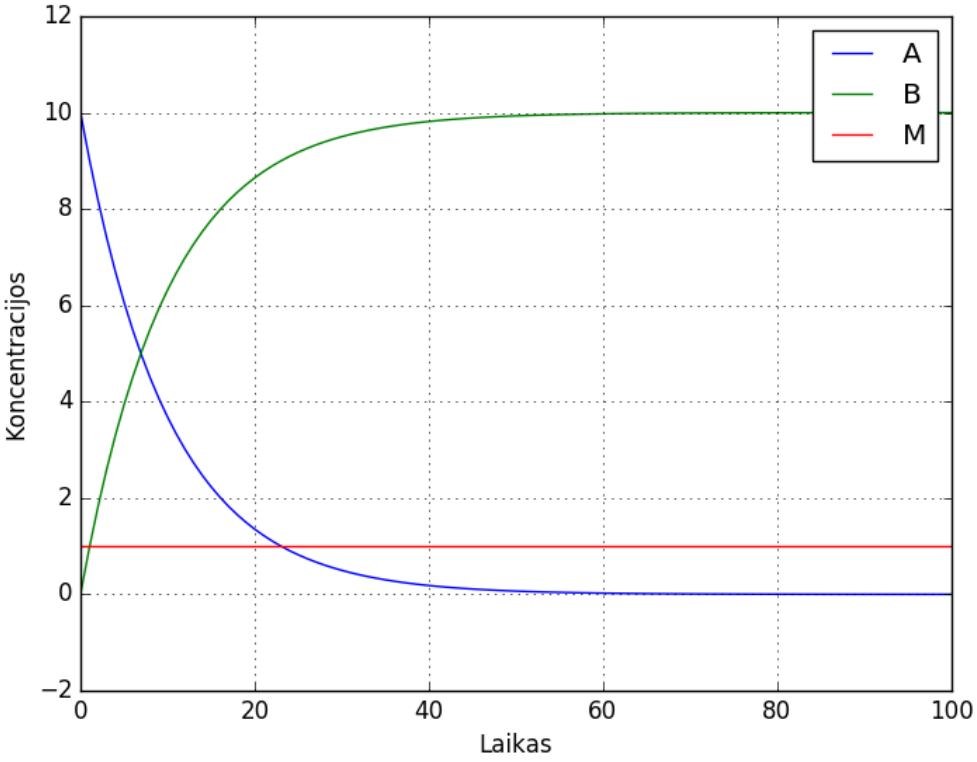
Pav. 3: Grafikas naudojant matlabo programą



Pav. 4: Grafikas naudojant *odeint* integravimą



Pav. 5: Grafikas naudojant *ode* integravimą



3 Van Der Pol osciliatorius

Van Der Pol osciliatorius – tai toks osciliatorius su netiesiniais slopinimais. Osciliatorius Van Der Pol buvo pasiūlytas Olandijos inžinieriaus ir fiziko Baltazaro van der Polo kai jis dirbo Philips įmonėje. Van der Pol osciliatorius turi du modelius.

Pirmas:

$$\begin{cases} \frac{dx}{dt} = u(x - \frac{1}{3} \cdot x^3 - y) \\ \frac{dy}{dt} = \frac{1}{u} \cdot x \end{cases}$$

Antras:

$$\begin{cases} \frac{dx}{dt} = y \\ \frac{dy}{dt} = u \cdot (1 - x^2) \cdot y - x \end{cases}$$

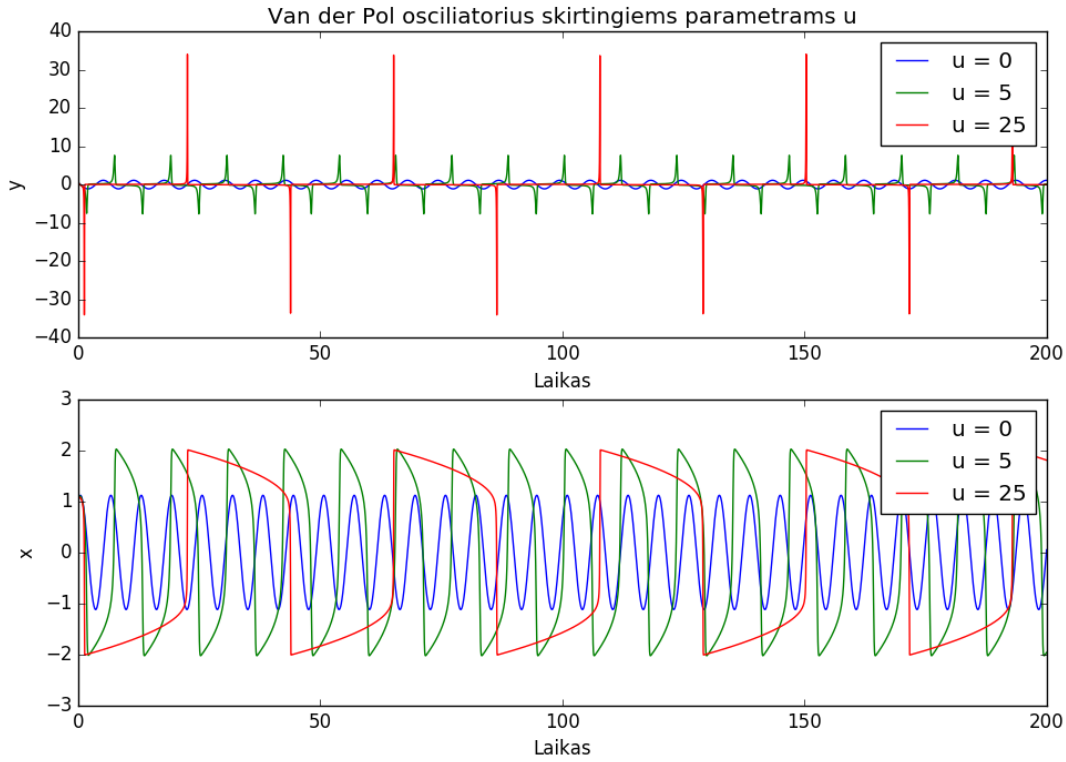
Naudojant BioSystem modulį nagrinėjame antrą diferencijalinių lygčių sistemą. Panagrinėkime du atvejus. Kai u lygus 0 tai osciliatorius veikia be slopinimų. Kai u yra didesnis nei 0 tai, sistema turi tam tikrus ciklus su slopinimais. Kuo u yra toliau nuo 0, tuo osciliatoriaus virpesiai atrodo mažiau harmoningi. Pradinės reikšmės:

- $x = 1$
- $y = 0.5$

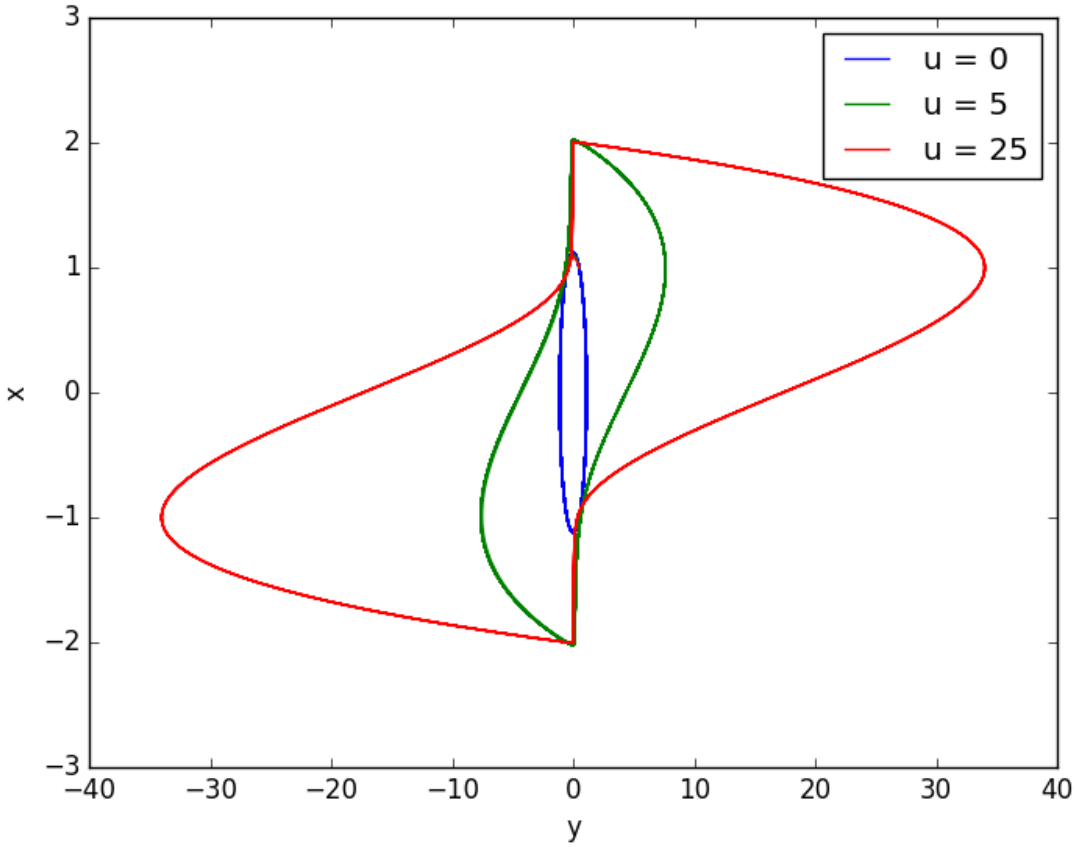
Grafikams sudaryti pasirinkau tris u reikšmes: 0, 5 ir 25.

Šiame grafike matome su kokiais y ir x vyksta slopinimas.

Pav. 6: Van Der Pol osciliatorius skirtingiems u reikšmėms



Pav. 7: Van Der Pol osciliatorius. Santykis tarp x ir y



4 Karkaso išbandymai

Nuodugniam karkaso išbandymui buvo pasirinkti du straipsniai[4][5], kuriuose nagrinėjamos tam tikros biosistemos ir sprendžiamos atitinkamos diferencialinių lygčių sistemos.

4.1 Ląstelės ciklo modeliavimas

Straipsnyje [4] autoriai aprašo *Xenopus* kiaušinių ląstelių ciklą. Nesigilindami į modelio detales galime pateikti atitinkamą diferecialinių lygčių sistemą:

$$
\begin{cases}
\frac{dCyclin}{dt} = k1 - (V2_i \cdot (TotAPC - APC) + V2_{ii} \cdot APC) \\
\qquad \cdot Cyclin - k3 \cdot Cyclin \cdot (TotCdk - MPF - preMPF) \\
\frac{dMPF}{dt} = k3 \cdot Cyclin \cdot (TotCdk - MPF - preMPF) - (V2_i \cdot (TotAPC - APC) \\
\qquad + V2_{ii} \cdot APC) \cdot MPF - (Vwee_i \cdot Wee1P + Vwee_{ii} \\
\qquad \cdot (TotWee1 - Wee1P)) \cdot MPF + (V25_i \cdot (TotCdc25 - Cdc25P) \\
\qquad + V25_{ii} \cdot Cdc25P)/cdotpreMPF \\
\frac{dpreMPF}{dt} = (Vwee_i \cdot Wee1P + Vwee_{ii} \cdot (TotWee1 - Wee1P)) \cdot MPF \\
\qquad - (V25_i \cdot (TotCdc25 - Cdc25P) + V25_{ii} \cdot Cdc25P) \cdot preMPF \\
\qquad - (V2_i \cdot (TotAPC - APC) + V2_{ii} \cdot APC) \cdot preMPF \\
\frac{dCdc25}{dt} = \frac{(TotCdc25 - Cdc25P) \cdot ka \cdot MPF}{TotCdc25 - Cdc25P + KKa} - \frac{Cdc25P \cdot kb \cdot PPase}{Cdc25P + KKb} \\
\frac{dWee1P}{dt} = \frac{(TotWee1 - Wee1P) \cdot ke \cdot MPF}{TotWee1 - Wee1P + KKe} - \frac{Wee1P \cdot kf \cdot PPase}{KKf + Wee1P} \\
\frac{dIEP}{dt} = \frac{(TotIE - IEP) \cdot kg \cdot MPF}{KKg + TotIE - IEP} - \frac{IEP \cdot kh \cdot PPase}{KKh + IEP} \\
\frac{dAPC}{dt} = \frac{(TotAPC - APC) \cdot kc \cdot IEP}{TotAPC - APC + KKc} - \frac{APC \cdot kd \cdot PPase}{KKd + APC}
\end{cases}
$$

Pateiksime aukščiau užrašytos sistemos kintamųjų fizikinę prasmę:

- Cyclin – ciklino koncentracija;
- MPF – brendimo skatinimo faktoriaus koncentracija;
- preMPF – neaktyvaus brendimo skatinimo faktorius;
- Cdc25P –Cdc25 forforilinta fosfatazė;
- Wee1P –Wee1 fosforilinta kinazė;
- IEP – Fosforilinta anafazės skatinimo komplekso dalis;

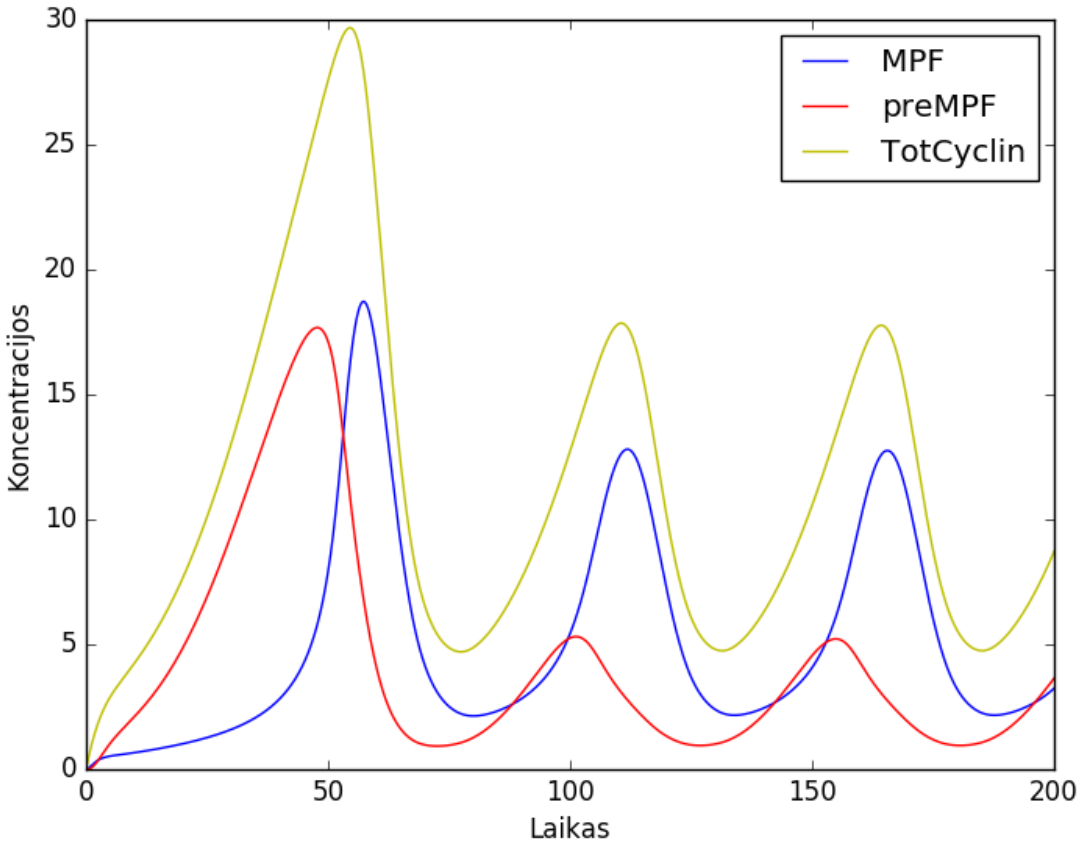
- APC – Anafazės skatinimo kompleksas.

Grafikams sukūrti buvo nauduotos tokios greičių konstantos:
$k1 = 1$, $V2_i = 0.005$,
$V2_{ii} = 0.25$, $k3 = 0.005$,
$V25_i = 0.017$, $V25_{ii} = 0.17$,
$Vwee_i = 0.01$, $Vwee_{ii} = 1$,
$ka = 0.02$, $KKa = 0.1$,
$kb = 0.1$, $KKb = 1$,
$kc = 0.13$, $KKc = 0.01$,
$k_d = 0.13$, $KKd = 1$,
$ke = 0.02$, $KKe = 0.1$,
$kf = 0.1$, $KKf = 1$,
$kg = 0.02$, $KKg = 0.01$,
$kh = 0.15$, $KKh = 0.01$,
$TotCdk = 100$, $TotCdc25 = 1$,
$TotWee1 = 1$, $TotIE = 1$,
$TotAPC = 1$, $PPase = 1$.

Pradiniai koncentracijų kiekiui $t = 0$:

- Cyclin = 0
- MPF = 0
- preMPF = 0
- Cdc25P = 0
- Wee1P = 0
- IEP = 1
- APC = 1

Pav. 8: MPF, preMPF ir ciklino koncentracijos kiekiai laike $0 \leq t \leq 200$.



Gautą grafiką galima palyginti su straipsnyje esančiu su žyme „*Figure 4*“[4] grafiku. Kaip matome rezultatai grafike yra identiški.

4.2 Trijų elementų genetinis osciliatorius

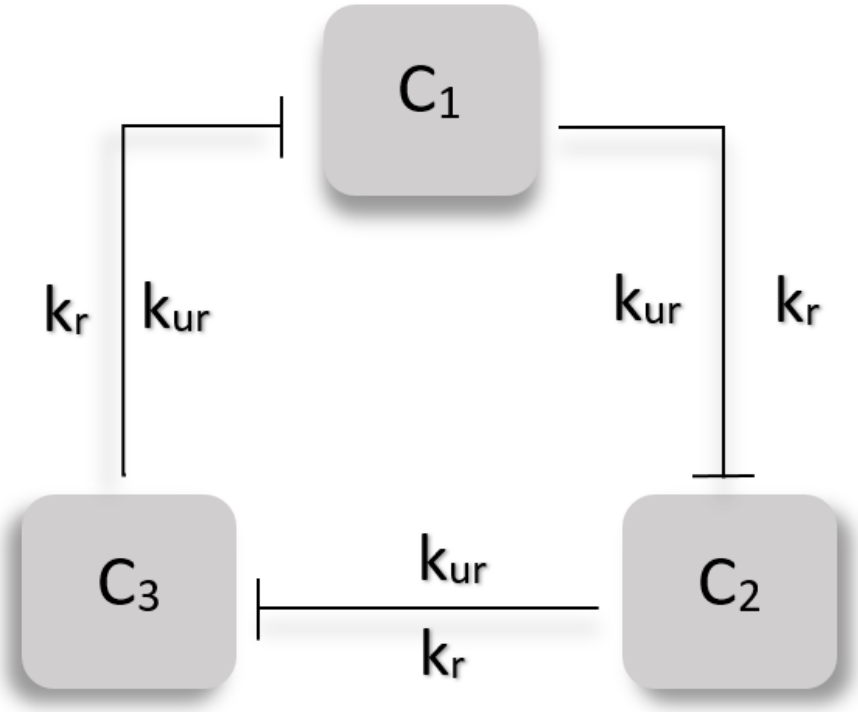
Straipnyje[5] yra aprašomas paprastas genetinis osciliatorius, žinomas kaip represiliatorius. Tai yra trijų elementų genetinis tinklas (žiurėti Pav. 9) sudarytas iš trijų genų ir atitinkamų represorių. Atitinkamas tinklas susideda iš:

- G – genų;
- M – mRNR;
- P – baltymų;
- D – dimerų.

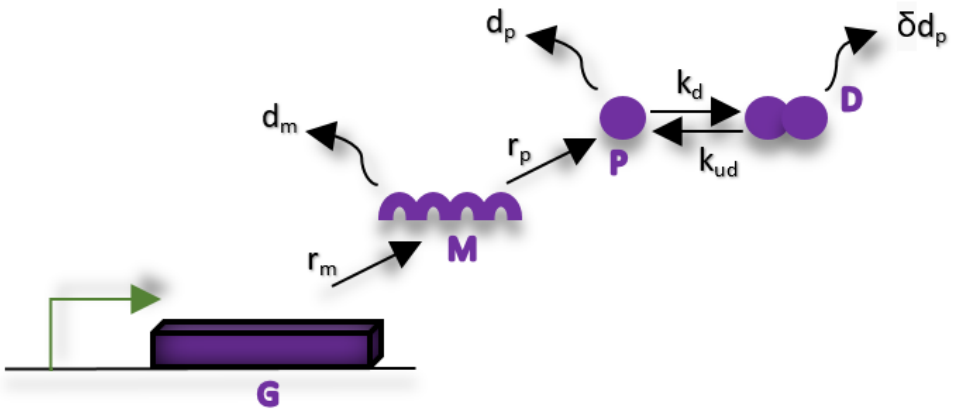
Genų transkripcija gamina mRNR, kuri po translacijos koduoja monomerą. Baltymai gali jungtis į dimerus. Dimeras represuoja trankripcijos veiksnius kitam pagal ciklą genui. Kiti du elementai turi tą patį principą.

Modelio idėja yra sena ir realizuota realiose bakterijose tipo ląstelių (trijų genų tinklas buvo įtrauktas į plazmides).

Pav. 9: Trijų elementų osceliatoriaus modelis



Pav. 10: Vieno elemento modelis



Kadangi kiekvienas elementas skirtosi į keturias dalis, tai diferencialinių lygčių sistemas sudaro dvylika lygčių. Matėmatinis modelis atrodo taip:

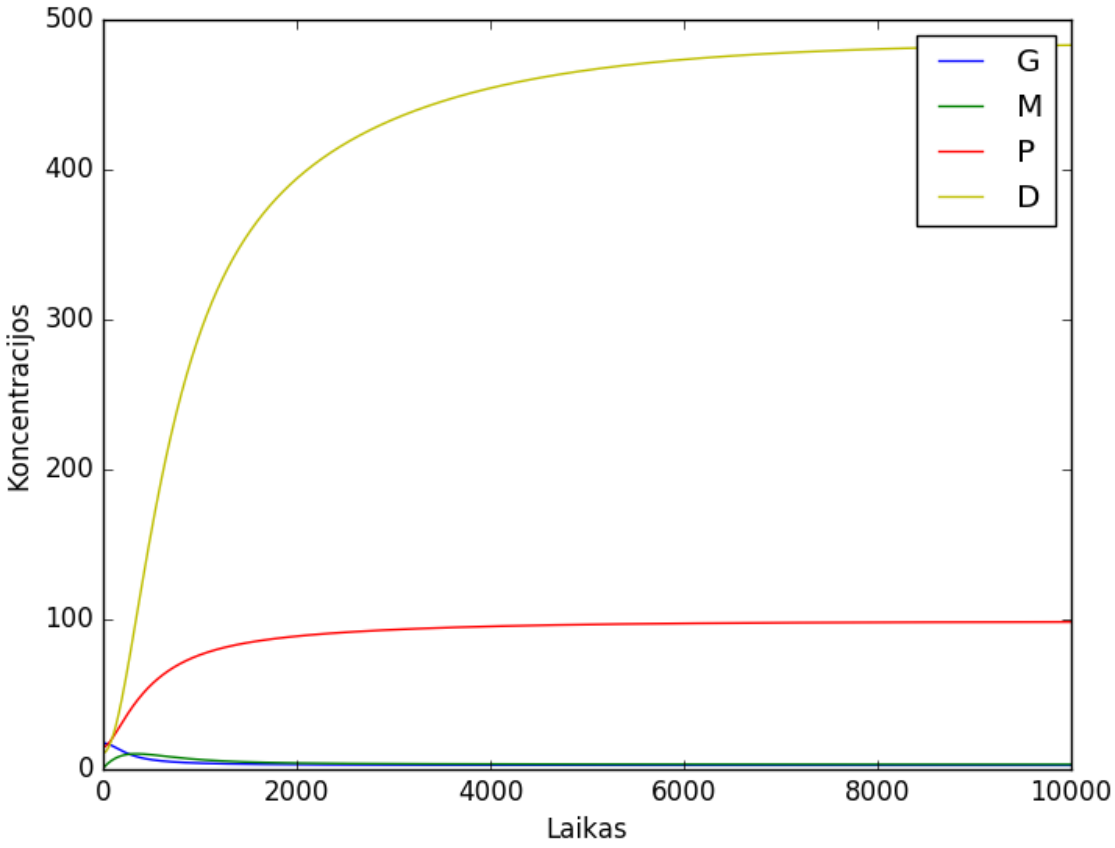
Matematinis modelis atrodo taip:

$$\begin{cases} \frac{dG_i}{dt} = k_{ur} \cdot (g_{tot} - G_i) - k_r \cdot D_k \cdot G_i \\ \frac{dM_i}{dt} = -d_m \cdot M_i + r_m \cdot G_i \\ \frac{dP_i}{dt} = r_p \cdot M_i - d_p \cdot P_i + 2 \cdot k_{ud} \cdot D_i - 2 \cdot k_d \cdot P_i^2 \\ \frac{dD_i}{dt} = -k_{ud} \cdot D_i + k_d \cdot P_i^2 + k_{ur} \cdot (g_{tot} - G_j) - k_r \cdot D_i \cdot G_j - \delta \cdot d_p \cdot D_i \end{cases}$$

Šios sistemos integravimui panaudojame *odeint* integravimu. Pradžiai paimsime, kad visos koncentracijos lygios vienetui. Pagrindinės konstantos yra tokios:

- $k_r = 0.012$ dimero prisijungimo greitis prie promotoriaus
- $k_u r = 0.9$ dimero atsijungimo greitis nuo promotoriaus
- $k_u d = 0.5$ dimero disociacijos greitis
- $k_d = 0.025$ dimero sukūrimo greitis
- $r_p = 0.1$ transliacijos greitis
- $d_m = 0.0033$ mRNR degradacijos greitis
- $d_p = 0.0033$ baltymo degradacijos greitis
- g_{tot} = plazmydžiu kopiju kiekis
- $r_m = 0.004$ mRNR trankripcijos greitis
- $d_g = 0$ dimerų degradacija

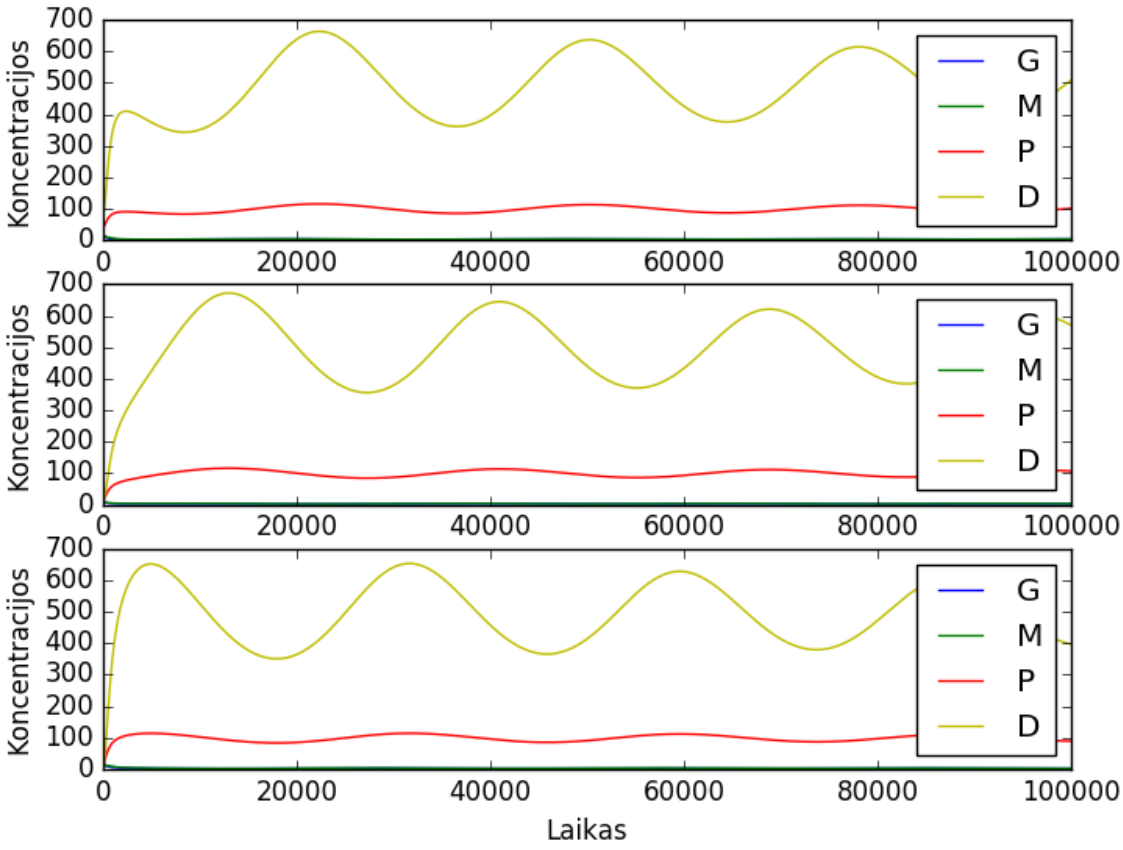
Pav. 11: Grafikas parodantys koncentracijų kitimą



Kadangi pradinės koncentracijos buvo vienodos, tai ir visų trijų elementų koncentracijos yra tokios pačios. Kaip matome, dimerų skaičius labai išaugo ir todėl tinklas nuėjo į stabilią būseną.

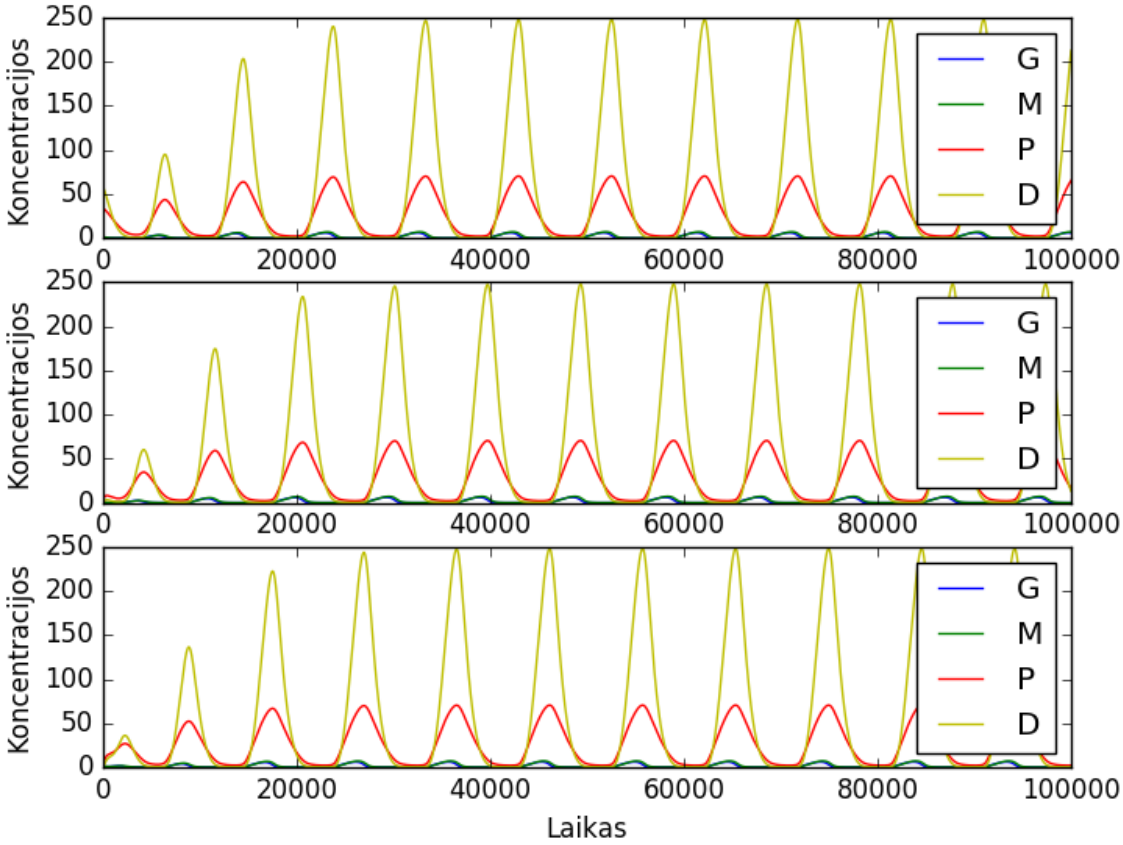
Galima padidinti baltymų kiekį viename iš tinklo elementų ir pažiūrėti kas įvyks. Turėsime tokią situaciją:

Pav. 12: Koncentracijos kitimas pakeičiant pirmo elemento baltymo kiekį iki 200



Taigi kaip rodo grafikas vyksta virpesiai. Labiausiai matosi tai ant dimerų ir truputi baltymų kiekiuose. Trečiame bandyme buvo padidinta kr konstantos reikšmė iki 1 ir kur konsntantos reikšmė sumažintą iki 0,1. Taip pat padidinau dg reikšmę iki 0,2.

Pav. 13: Koncentracijos kitimas padidinus dimero prisijungimo greitį prie promotoriaus ir sumažinos dimero atsijungimo greitį

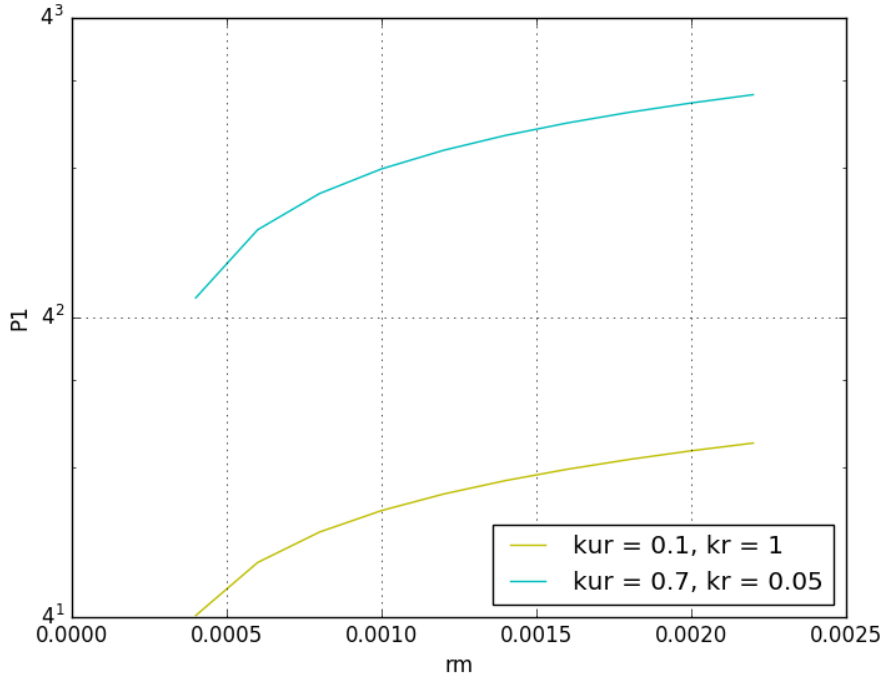


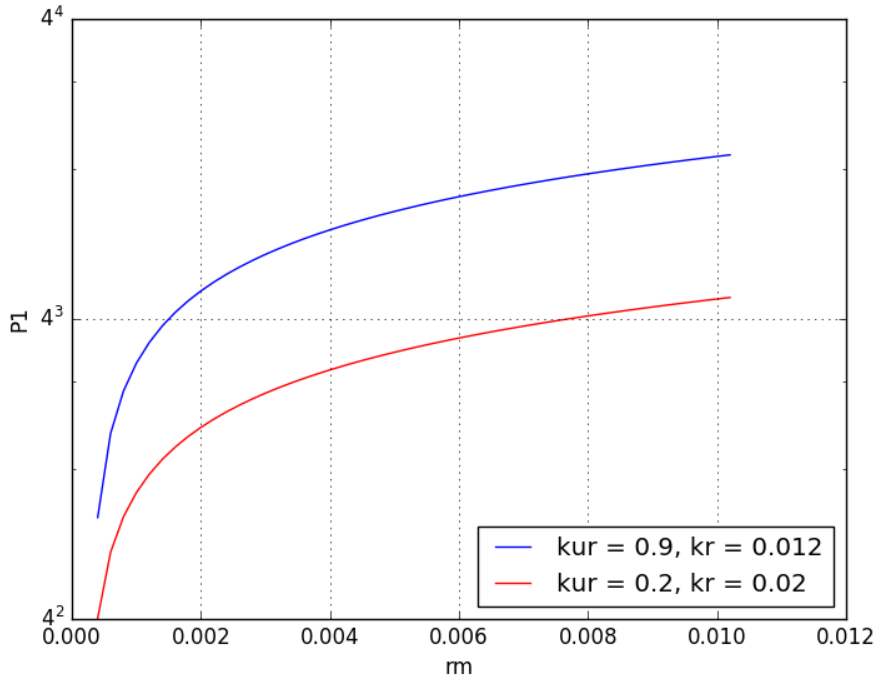
Kai nėra dimerų degradacijos iš praeitų grafikų galima buvo sakyti, kad jų yra daugiausia sistemoje, bet jeigu žymiai padidint d_g konstantos reikšmę, tai pamatysime kad dimero bus taip mažai, kad sistema labai greitai prieis prie stabilios būsenos.

4.2.1 Įrodymas, kad karkasas veikia taisingai

Grįžtame prie pirmų konstantų ir kai visos koncentracijos buvo 1. Tai dabar galėsime įrodyti straipsnyje esančio grafikų tikslumą. Pavyzdžiui keisime r_m konstantą ir paimsime labai didelį laiko tarpą. Taip galime būti įsitikinę, kad grafikas prie stabilios būsenos ir galėsiu palygint kokį santykį r_m ir baltymo iš sistemos pirmo elemento gavau aš su atitinkamu iš straipsnio[5].

Pav. 14: P_1 koncentracjos kiekis tam tikriems r_m skirtingiems k_{ur} ir k_r greičių konstantų reikšmėms





Jeigu lygintumėme straipsnio grafikus ir aprašyto karkaso gautą, tai galima teigti kad rezultatai yra panašūs.

5 Išvados ir rezultatai

Darbo pradžioje buvo aptarta kas yra biologinės sistemos. Taip pat panaudojame karkaso iš Edx kurso[1] ir sukūrėme nuosavą, kurį apiforminame kaip python modelius. Minėtą karkasą išbandėme su keliais mokslinėje literatūroje pateiktais modeliais ir įsitikinome, kad gaunami rezultatai nesiskiria.

Karkaso pliusai:

1. Karkaso pagalba galima nagrinėti didelę klasę biologinių modulių.
2. Karkasas turi du lankščius integravimo metodus.

Karkaso minusai:

1. Reakcijų greičiuose negalima nauduoti SymPy modulyje nepalaikomas funkcijas.
2. Kai kurias diferencialinių lygčių sistemas karkasas integruos lėčiau, negu matlab'as.

Santrauka

Darbe buvo sukurtas karkasas skirtas biologinių sistemų, kurias galima tirti paprastųjų diferencialinių lygčių sistemų pagalba, analizei. Pateikti jo panaudojimo pavyzdžiai įrodytas karkaso jo korektiškumas remiantis kai kurių mokslinių straipsnių rezultatais.

Raktiniai žodžiai: biologinių sitemų modeliavimas, paprastos diferencialinės lygtys ir sistemos, python

Summary

We have created simple framework for biology sytems analisys to calculate ordinary differential equations systems. This tool has been used to analyse a few examples. What's more, we have proven correctness of the tool by reproducing results, published in scientific articles.

Keywords: biology systems modeling, ordinary differential equations and systems, python

Literatūros sąrašas

[1] https://courses.edx.org/courses/course-v1:MITx+20.305x+3T2015/info

[2] http://web.mit.edu/20.305/www/part_composition_setup.m

[3] Raúl Guantes, Juan F Poyatos.
Dynamical Principles of Two-Component Genetic Oscillators.
Kovo 31, 2006. http://dx.doi.org/10.1371/journal.pcbi.0020030

[4] Jill C. Sible, John J. Tyson.
Mathematical Modeling as a Tool for Investigating Cell Cycle Control Networks.
Vasaris 2007. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1993813/

[5] Ilya Potapov, Boris Zhurov and Evgeny Volkov.
Multi-stable dynamics of the non-adiabatic repressilator.
2015 Kovo 6. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4345497/