Molekuladinamika

Márton Tamás

Eötvös Lóránd Tudományegyetem, Informatikus Fizikus
Számítógépes szimulációk laboratórium.
VI. jegyzőkönyv.

Tartalomjegyzék

Ábrák jegyzéke

1. Fizikai probléma ismertetése

A szimulációs probléma során differenciálegyenlet megoldással modellezünk populációdinamikai jelenségeket, ahol a populáció n létszámának időbeli változását vizsgáljuk. Első közelítésben feltehetjük, hogy a populáció gyarapodása arányos magával a populáció létszámával, így bevezethetünk egy a szaporodási rátát, ami megmondja, hogy Δt idő alatt mennyivel változik a populáció nagysága:

$$n(t + \Delta t) = n(t) + an(t). \tag{1.1}$$

Ha Δt kicsi, akkor az 1 egyenlet folytonossá tételével a következő differenciálegyenlet formára alakítható:

$$\frac{dn}{dt} = an, \tag{1.2}$$

aminek megoldása:

$$n = e^{at} \tag{1.3}$$

Ez a modell realisztikusabbá tehető ha bevezetünk egy d halálozási rátát is, s azzal írjuk át a 1 egyenletet:

$$\frac{dn}{dt} = an - dn. \tag{1.4}$$

Ennek, a 1 egyenletnek a megoldása az $r = a - d$ mennyiség előjelétől függ:
lehet exponenciálisan növekvő, vagy exponenciálisan csökkenő görbe:

$$n = e^{rt} \tag{1.5}$$

viszont a konstans megoldás láthatóan nem stabil. Ha figyelembe vesszük viszont, hogy akár az élelem, akár a nyersanyag korlátos, akkor a 1 egyenletet módosíthatjuk:

$$\frac{dn}{dt} = rnF(n) \tag{1.6}$$

Éljünk azzal a feltevéssel, hogy az erőforrások egy maximum k létszámú populációt tarthatnak el. F (n) legegyszerűbben n-ben lineárisan, kis nagyságú populáció esetén nem módosítja az eredeti 1 egyenletet, viszont a k kapacitás elérésekor megállítja a szaporodást, és ezt a következő formában kielégíti:

$$F(n) = 1 - \frac{n}{k}. \tag{1.7}$$

Ennek segítségével a megoldandó differenciálegyenlet a következő alakot ölti:

$$\frac{dn}{dt} = rn\left(1 - \frac{n}{k}\right). \tag{1.8}$$

Ha bevezetünk a következő átskálázást:

$$x = \frac{n}{k}, \tag{1.9}$$

akkor a 1 egyenlet, amit átskálázás után már logisztikus egyenletnek hívunk, a következő formában írható fel:

$$\frac{dx}{dt} = rx(1 - x), \tag{1.10}$$

amelynek megoldása r-től, és a kezdeti x 0 paramétertől függően növekedő vagy csökkenő szigmoid jellegű görbe:

$$x(t) = \frac{1}{1 + (1/x_0 - 1)\, e^{-rt}}. \tag{1.11}$$

Fontos viszont észrevenni, hogy a 1. a logisztikus egyenlet egy nemlineáris differenciálegyenlet. A nemlineáris differenciálegyenletek analitikusan nem mindig oldhatók meg. Az egyenlet fixpontjai, ahol:

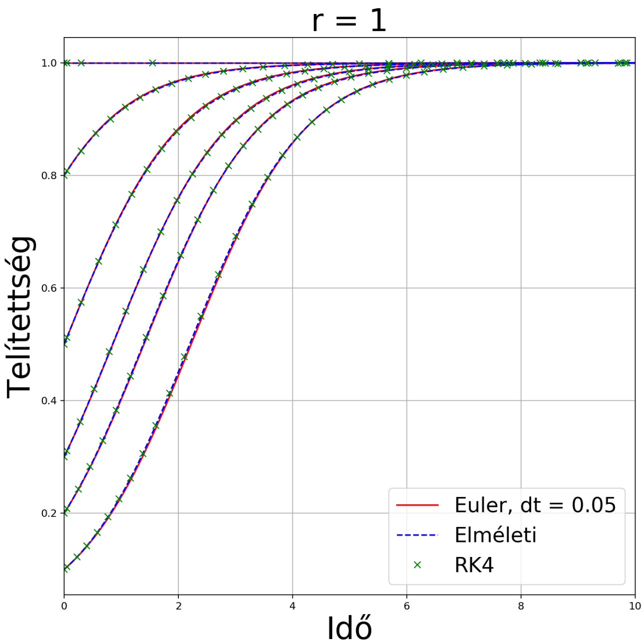
$$\frac{dx}{dt} = 0, \tag{1.12}$$

jelen esetben az $x = 0$ és az $x = 1$.

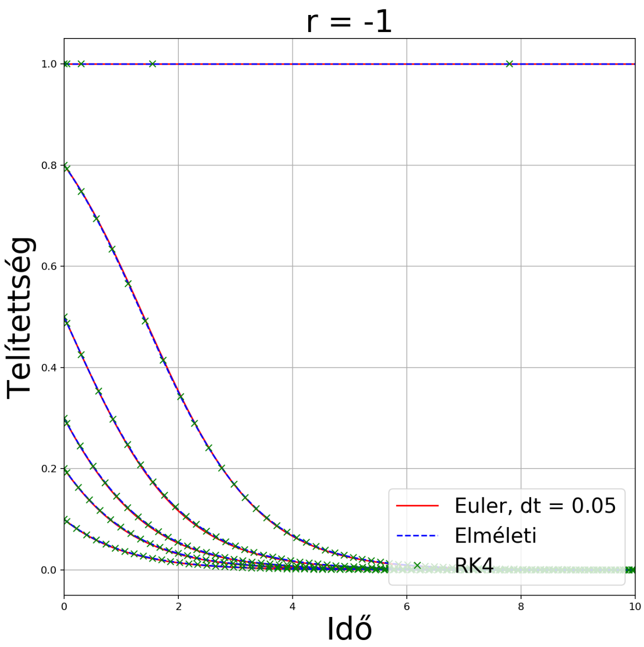
2. Szimuláció

2.a. Első feladat, logisztikus egyenlet numerikus vizsgálata.

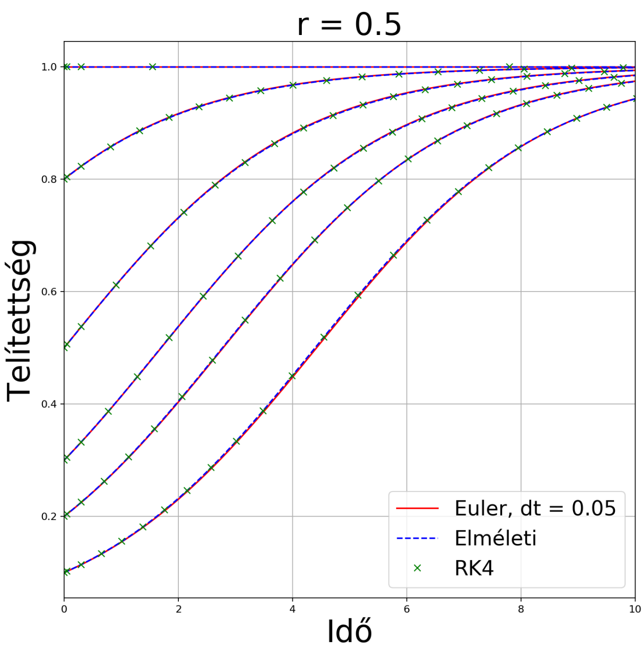
A szimuláció első feladata az volt, hogy oldjuk meg numerikusan a logisztikus egyenletet az Euler-módszer, és az adaptív Runge-kutta-módszer segítségével, és reprodukáljuk a diasoron látható ábrákat. A szimulációmat C++ nyelven írtam, a már korábbról ismert odeint csomag segítségével (5.a. kódrészlet). A szimulációt lefuttattam 5 különböző r értékre és 6 különböző x_0 értékre Euler és Runge-Kutta-módszerrel is.
A kimeneti .data fájlokból Python segítségével ábrázoltam. Mindegyik ábrán rajta vannak mind az Euler, mind a Runge-Kutta-módszer eredményei, illetve az analitikus megoldás is (1. egyenlet). Az Euler-módszer esetében ahogy a programkódból is látszik, dt = 0.05 lépésközt használtam. Tökéletesen látszik az ábrákról (2.a.1, 2.a.2., 2.a.3., 2.a.4. és 2.a.5. ábrák), hogy mind az adaptív Runge-Kutta, és mind az Euler–módszer a megfelelően választott dt = 0.05 lépésközzel tökéletesen leírják az analitikus (1. egyenlet) megoldást.



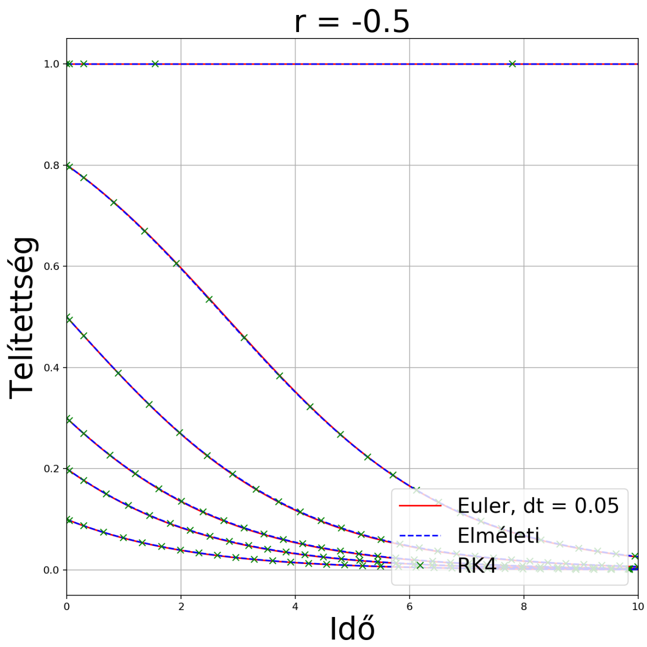
2.a.1. ábra. r = 1 eset.



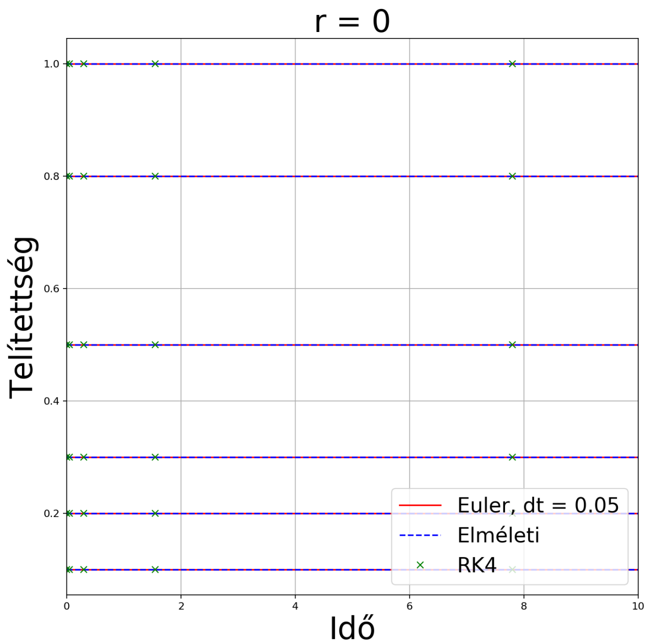
2.a.2. ábra. r = -1 eset.



2.a.3. ábra. r = 0.5 eset.



2.a.4. ábra. r = -0.5 eset.



2.a.5. ábra. r = 0 eset.

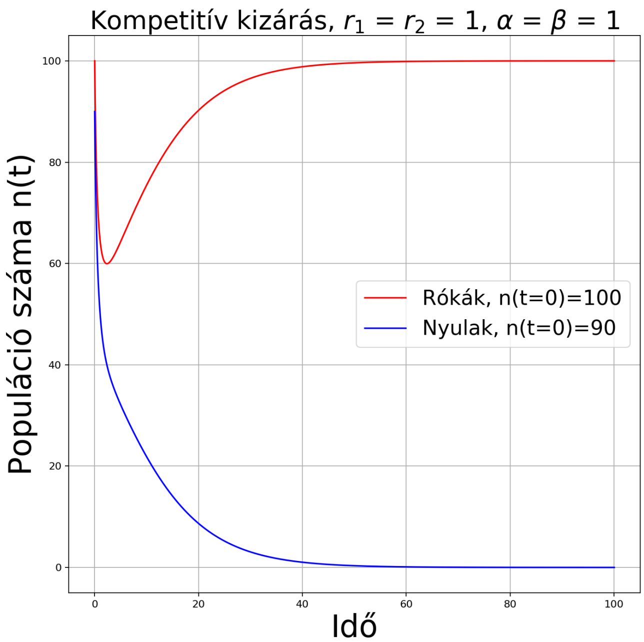
3. Második feladat, versengés.

Ha egy adott élőhelyen már több különböző faj küzd ugyanazon táplálékért, akkor a véges erőforrások miatt kölcsönhatásba kerülnek a fajok. Egymáshoz viszonyított szaporodási rátájuk, és az élőhely eltartóképessége miatt a "rátermettebb" faj el is foglalhatja teljes mértékben az élőhelyet, ezzel elpusztítva a "gyengébb" populációt. A második feladatban két fajt modelleztünk. Ha bevezetünk egy α és egy β dimenziótlan paramétert, amik kifejezik, hogy milyen arányban fogyasztja az egyik faj a másik erőforrásait, a populációk differenciálegyenlete a kölcsönhatásuk miatt a következő formában írható fel:

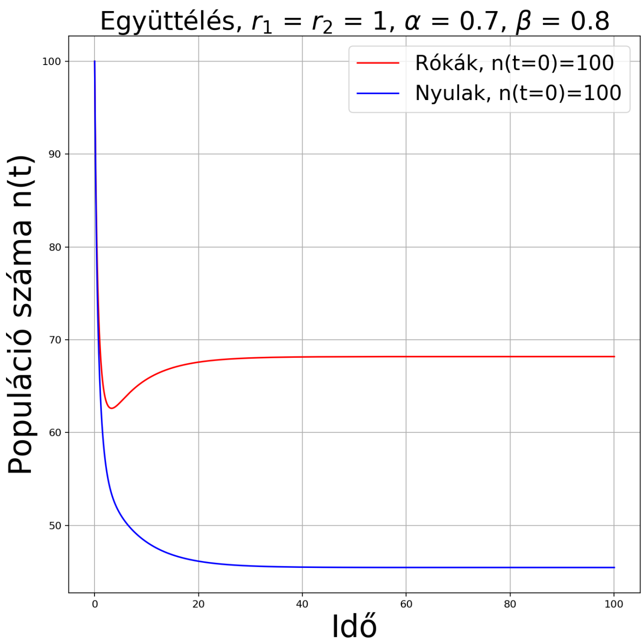
$$\frac{dn_1}{dt} = r_1 n_1 \left(1 - \frac{n_1 + \alpha n_2}{k_1}\right). \tag{3.1}$$

$$\frac{dn_2}{dt} = r_2 n_2 \left(1 - \frac{n_2 + \alpha n_2}{k_1}\right) \tag{3.2}$$

A szimulált rendszer fixpontjai természetesen az $\alpha, \beta, r_1, r_2, k_1$ és a k_2 paraméterektől függenek. A szimulációt ismét C++ nyelven írtam, viszont ebben az esetben már csak az Euler-módszerrel dolgoztam (5.b. kódrészlet).
Elsőnek az $\alpha = \beta = 1$ esetet kellett szimulálni, vagyis demonstrálni azt az esetet, amikor a két faj nem létezhet stabilan együtt (kompetitív kizárás): a nagyobb k értékű faj kiszorítja a másikat, ezt láthatjuk a 3..1. ábrán. Következő teendő az volt, hogy mutassuk meg, hogy a két faj együttélése csak akkor stabil, ha teljesül, hogy $\alpha k_2 < k_1$ és $\beta k_1 < k_2$. Erre, a stabil együttélésre láthatunk példát a 3..2. ábrán, ahol a paraméterek teljesítik az imént kirótt feltételt.



3..1. ábra. Kompetitív kizárás esete.



3..2. ábra. Két faj együttélésének esete.

4. Harmadik feladat, Lotka-Voltera-modell.

A Lotka-Volterra–modell (LV) esetén a nyulak és rókák populációjának időfejlődését leíró differenciál egyenletek:

$$\frac{dn_R}{dt} = an_R - bn_F n_R \tag{4.1}$$

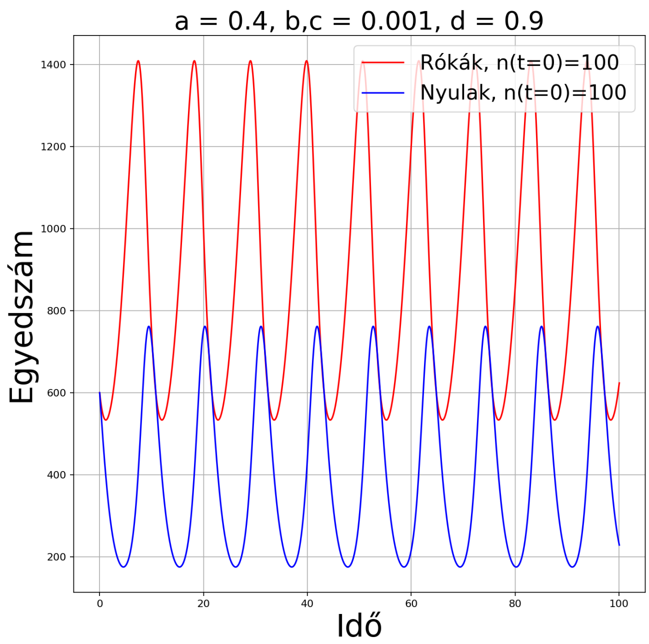
$$\frac{dn_F}{dt} = cn_R n_F - dn_F \tag{4.2}$$

ahol n_R a nyulak és n_F a rókák száma, a a nyulak szaporodási rátája, d a rókák pusztulási rátája, bn_F a nyulak pusztulási rátája és cn_R a rókák szaporodási rátája. A modell realisztikusabbá tehető ha korlátozzuk a nyulak táplálékforrását, valamit a rókák nyúlfogyasztási képességét:

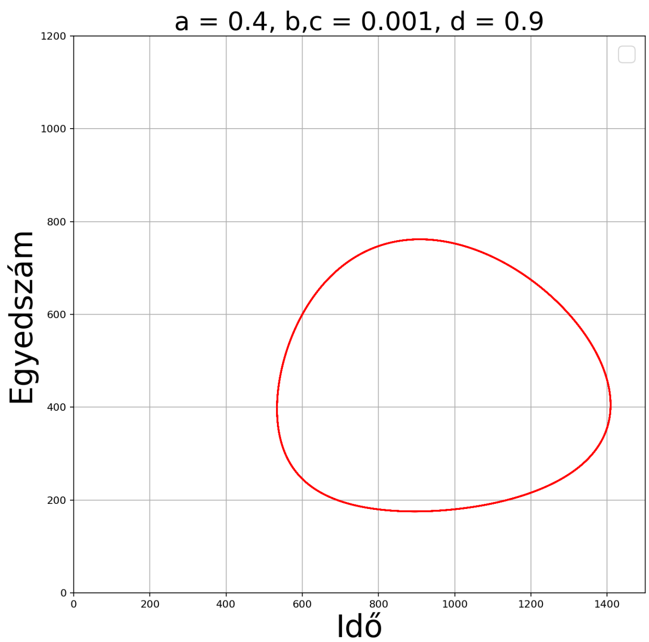
$$a \rightarrowtail a\left(1 - \frac{n_R}{k}\right) \tag{4.3}$$

$$n_R n_F \rightarrowtail \frac{n_R n_F}{1 + n_R S} \tag{4.4}$$

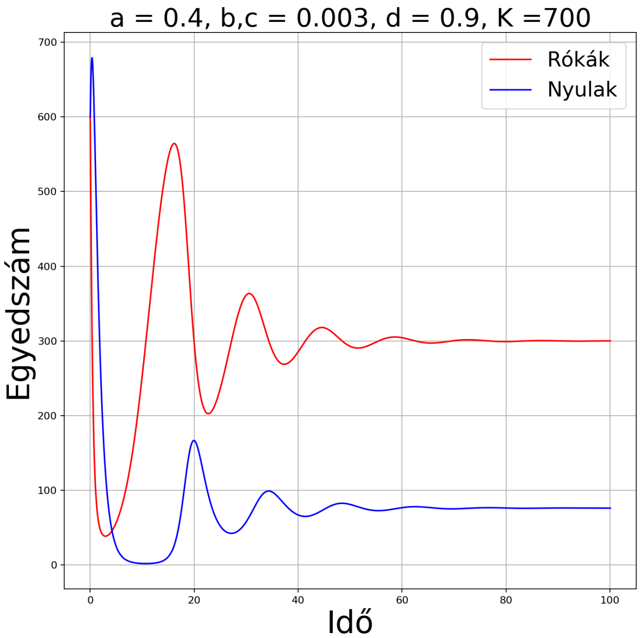
A szimulációmban implementáltam mind az alap LV-modellt, és a 4. és 4. kiegészítő lehetőséget (5.c. kódrészlet).
A sima LV–modell esetében és amikor bevezetjük a kapacitást sikerült reprodukálnom a diasor ábráit, ezek láthatók a 4..1.4..2.4..3.4..4. ábrákon. Viszont a harmadik esetben, amikor az S paraméter kerül bevezetésre, hosszas próbálkozás után sem tudtam egy stabil megoldást produkálni. Mindig azt tapasztaltam, hogy a nyulak vagy a rókák száma elszáll, míg a másik kinullázódik. Először arra gyanakodtam, hogy az Euler–módszer miatt nem stimmel a megoldás, ezért próbálkoztam adaptív és sima RK4 algoritmussal is, de úgy sem kaptam stabil rendszert.



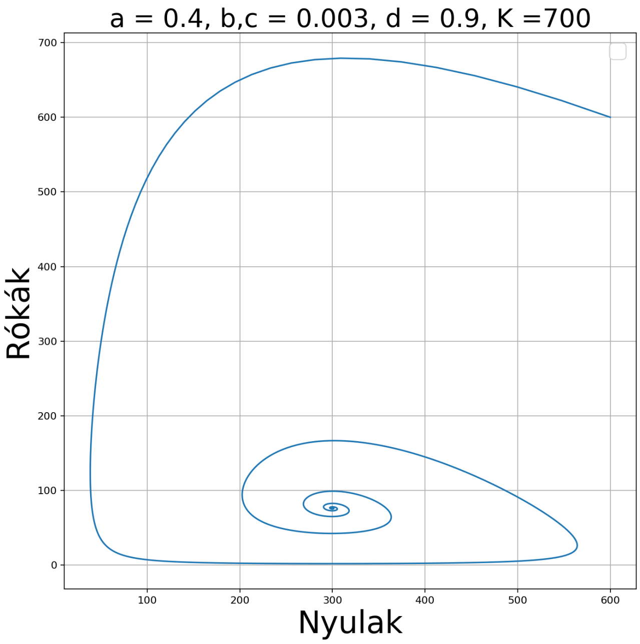
4..1. ábra. LV–modell.



4..2. ábra. LV–modell.



4..3. ábra. LV–modell a kapacitás bevezetésével.



4..4. ábra. LV–modell a kapacitás bevezetésével

5. Kódrészletek

5.a. Kódrészlet 1

```
#include <cmath>
#include <cstdlib>
#include <iostream>
#include <fstream>
#include <cstdlib>
#include "vector.hpp"
#include "odeint.hpp"
#include <math.h>

using namespace cpl;
using namespace std;


double r;
double x0;
double x_aktualis;
int euler_vagy_rk4;
char* fileName;


Vector f(const Vector& x);
void Euler( cpl::Vector& x, double& tau,
        cpl::Vector derivs(const cpl::Vector&)){
        x += tau * derivs(x);
        }



int main(int argc, char **argv)
{

//Futtatas: ./popdin.bin e/r x0 valami.data r

        if(*argv[1] == 'e') euler_vagy_rk4 = 1;
        if(*argv[1] == 'r') euler_vagy_rk4 = 2;
        x0 = atof(argv[2]);
        fileName = argv[3];
        r = atof(argv[4]);

        ofstream file(fileName);

        double dt = 0.05;
```

```
        double accuracy = 1e-6;
        double t = 0;
        double tMax = 10;
        Vector x(2);

        x[0] = t;
        x[1] = x0;
        file << t << '\t' << x0 << '\n';

        if(euler_vagy_rk4 == 1)
        {
                while(t < tMax)
                {
                        Euler(x,dt,f);
                        t += dt;
                        x_aktualis = x[1];
                        file << t << '\t' << x_aktualis << '\n';
                }
        }
        if(euler_vagy_rk4 == 2)
        {
                while(t < tMax)
                {
                adaptiveRK4Step(x,dt,accuracy,f);
                t = x[0];
                x_aktualis = x[1];
                file << t << '\t' << x_aktualis << '\n';
                }
        }

        cout << "Adatok: "<< fileName << endl;
        file.close();
        return 0;
}


        Vector f(const Vector& x)
        {
        double x_aktualis = x[1];
        Vector f(2);
        f[0] = 1;
        f[1] = r * x_aktualis * ( 1 - x_aktualis );
        return f;
        }
```

5.b. Kódrészlet 2

```
#include <cmath>
#include <cstdlib>
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

int main(int argc, char **argv)
{

//Futtatas: ./2popdin.bin r1 r2 k1 k2 alpha beta
valami.data

        double r1 = atof(argv[1]);
        double r2 = atof(argv[2]);
        int k1 = atoi(argv[3]);
        int k2 = atoi(argv[4]);
        double alpha = atof(argv[5]);
        double beta = atof(argv[6]);
        char* fileName = argv[7];

        ofstream file(fileName);

        double dt = 0.05;
        double t = 0;
        double tMax = 100;

        file << t << '\t' << k1 << '\t' << k2 << '\n';

        double n1 = k1;
        double n2 = k2;
        while(t < tMax)
        {
                n1 += r1 * n1 * (1 - ((n1 + alpha * n2) /
                 k1)) * dt;
                n2 += r2 * n2 * (1 - ((n2 + beta * n1) /
                 k2)) * dt;
                t += dt;
                file << t << '\t' << n1 << '\t' << n2 << '\n';
        }
        cout << "Adatok: "<< fileName << endl;
        file.close();
        return 0;
}
```

5.c. Kódrészlet 3

```
#include <cmath>
#include <cstdlib>
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

//Futtatas: ./3popdin.bin a b c d nyulakSzama rokakSzama
valami.data S k melyikModszer

int main(int argc, char **argv)
{
        double a = atof(argv[1]);
        double b = atof(argv[2]);
        double c = atof(argv[3]);
        double d = atof(argv[4]);
        double n_nyul = atof(argv[5]);
        double n_roka = atof(argv[6]);
        char* fileName = argv[7];
        int S = atoi(argv[8]);
        int k = atoi(argv[9]);
        int modszer = atoi(argv[10]);
        ofstream file(fileName);
        double dt = 0.05;
        double t = 0;
        double tMax = 100;
        file << t << '\t' << n_nyul << '\t'
                << n_roka << '\n';

        if(modszer==1)
                while(t<tMax)
                {
                        n_nyul += (a * n_nyul - b * n_roka
                        * n_nyul ) * dt;
                        n_roka += (c * n_nyul * n_roka - d
                        * n_roka) * dt;
                        t += dt;
                        file << t << '\t' << n_nyul << '\t'
                                << n_roka << '\n';
                }

        if(modszer==2)
                while(t<tMax)
                {
```

```
                        n_nyul += (a*(1 - n_nyul / k)
                        * n_nyul - b * n_roka * n_nyul ) * dt;
                        n_roka += (c * n_nyul * n_roka - d
                        * n_roka) * dt;
                        t += dt;
                        file << t << '\t' << n_nyul << '\t'
                                << n_roka << '\n';
                }

        if(modszer==3)
        {
                while(t<tMax)
                {
                        n_nyul += (a * n_nyul - b * ((n_roka *
                         n_nyul) / (1 + n_nyul * S)) ) * dt;
                        n_roka += (c * ((n_roka * n_nyul) /
                        (1 + n_nyul * S)) - d * n_roka) * dt;
                        t += dt;
                        file << t <<  \t  << n_nyul <<  \t
                        << n_roka <<  \n  ;
                }

        cout << "Adatok: "<< fileName << endl;
        file.close();
        return 0;
}
```