Kutatómunka információs eszközei projekt dokumentáció

Tamás Krisztián - HVZK7Y

2018.05.27.

1. Bevezetés

A készített program két faj együttélésének populációdidinamikai szimulációja különféle modellek alapján. A programot hárman írtuk C++ nyelven. A program írásához használt github repository ide kattintva érhető el. Itt, illetve függelékben megtalálható a program kódja. A következőkben a program segítségével létrehozott adatokból készült ábrákról, a program mögötti elméleti háttérről és a kivitelezésről lesz szó.

2. Elméleti leírás és kivitelezés

Egy populáció létszámának időbeli változásának leírásához figyelembe vesszük a születési/halálozási rátát, illetve azt, hogy az erőforrások limitáltak. Igy a következő differenciálegyenletet írhatjuk fel a létszám változására:

$$\frac{dn}{dt} = rn(1 - \frac{n}{k})$$

ahol n a létszám, r a születést/halálozást jellemző szám, k a maximum létszám amit az erőforrások el tudnak tartani. Ha x = n/k-val átskálázzuk, a logisztikus egyenlethez jutunk.

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

aminek analitikus megoldása:

$$x(t) = \frac{1}{1 + (\frac{1}{x_0} - 1)e^{-rt}}$$

2.1. Csatolt logisztikus modell

Itt két faj kűzd egyazon táplálékért, a fajok változását leíró egyeneletek:

$$\frac{dn_1}{dt} = r_1 n_1(1 - \frac{n_1 - \alpha n_2}{k_1})$$

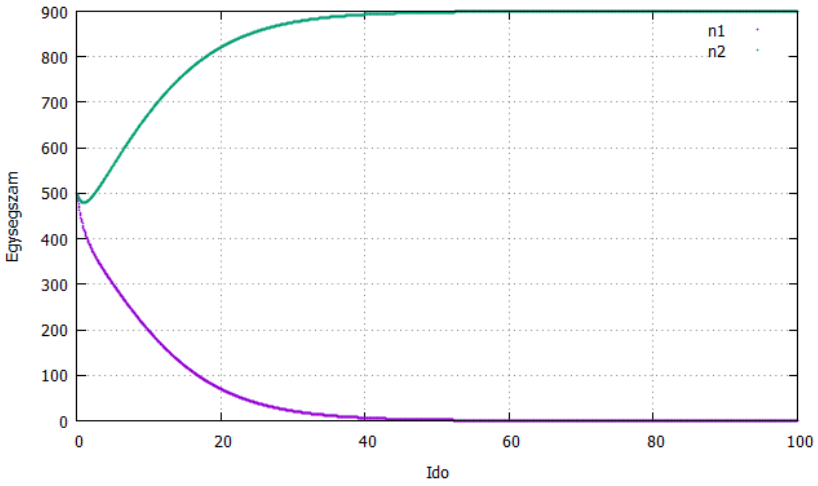
$$\frac{dn_2}{dt} = r_2 n_2(1 - \frac{n_2 - \alpha n_1}{k_2})$$

Kezdeti feltételnek a nyulak és a rókák számát egyaránt 500-nak vettem. A használt kódrészlet:

```
double* competitive(double x, double y)
{
    double np[2];
    np[0] = r1*x*(1-(x+alpha*y)/k1);
    np[1] = r2*y*(1-(y+beta*x)/k2);
    return np;
}
```

A kódban megadott változók értéke a következő ábrához:

$$\alpha = \beta = 1, r_1 = r_2 = 1, k_1 = 800, k_2 = 900$$



1. ábra. Az egységszámok az idő függvényében

Az ábrán látszik, hogy a nagyobb k értékű kiszorítja a másikat.

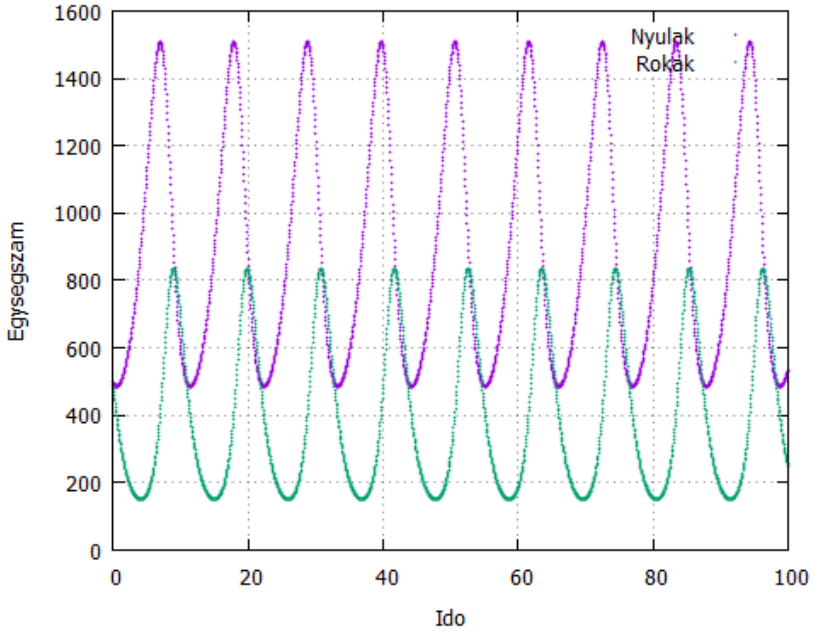
3. A Lotka-Volterra-modell (LV-modell)

A LV-modellben két fajt veszünk amelyek egymással kölcsönhatnak. A példában a nyulaknak korlátlan táplálékuk van, a rókák megeszik a nyulakat, és korlátlan a kapacitásuk. Az egyedszámok változását leíró egyenlet:

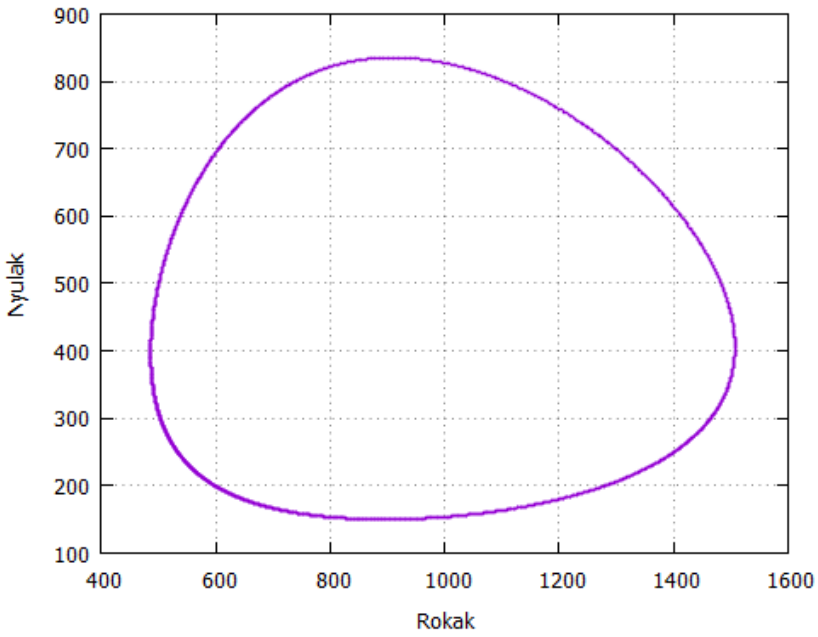
$$\frac{dn_R}{dt} = an_R - bn_F n_R$$
$$\frac{dn_F}{dt} = cn_R n_F - dn_F$$

A használt kódrészlet:

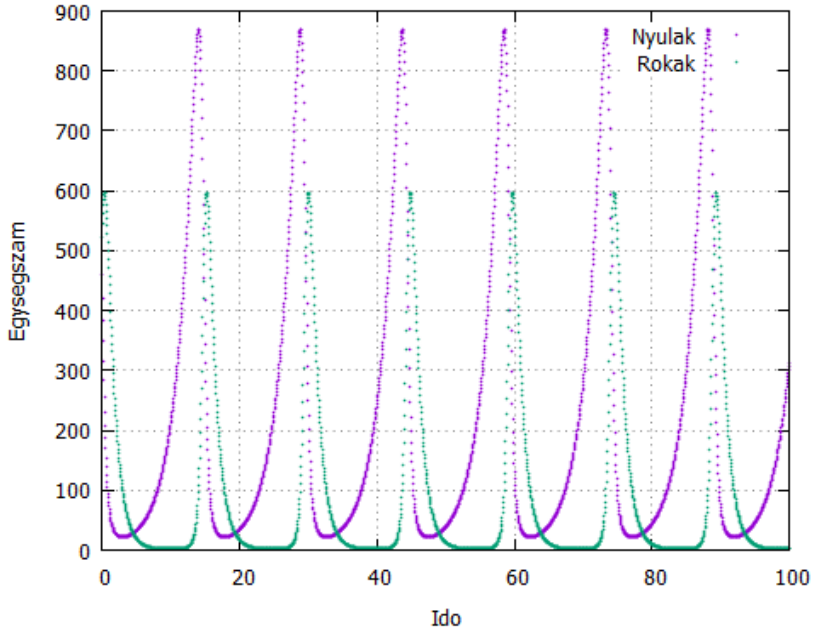
```
double* LotkaVolterra(double nr, double nf){
    double np[2];
    np[0] = a*nr-b*nr*nf;
    np[1] = c*nr*nf-d*nf;
    return np;
}
```



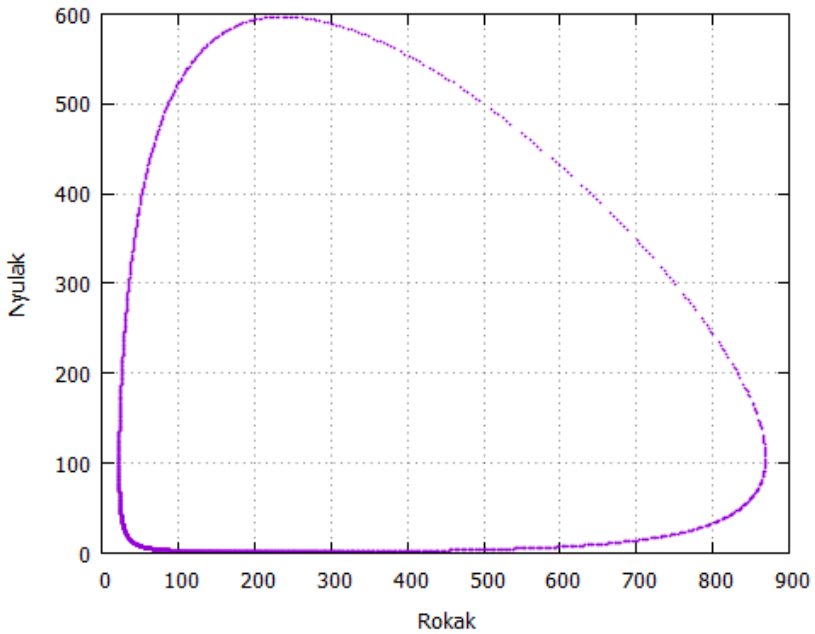
2. ábra. Egységszám az idő függvényében a következő paraméterrekkel: a = 0.4, b = 0.004, c = 0.001, d = 0.9



3. ábra. A rókák és nyulak száma a következő paraméterrekkel: a = 0.4, b = 0.001, c = 0.001, d = 0.9



4. ábra. Egységszám az idő függvényében a következő paraméterrekkel: a = 0.4, b = 0.004, c = 0.004, d = 0.9



5. ábra. A rókák és nyulak száma a következő paraméterrekkel: a = 0.4, b = 0.004, c = 0.004, d = 0.9

3.1. A realisztikusabb LV-modell

A modellben korlátozzuk a nyulak táplálékforrását, és a rókák nyúlfogyasztási képességét. Bevezetjük a kapacitást:

$$a \Rightarrow a(1 - \frac{n_R}{k})$$

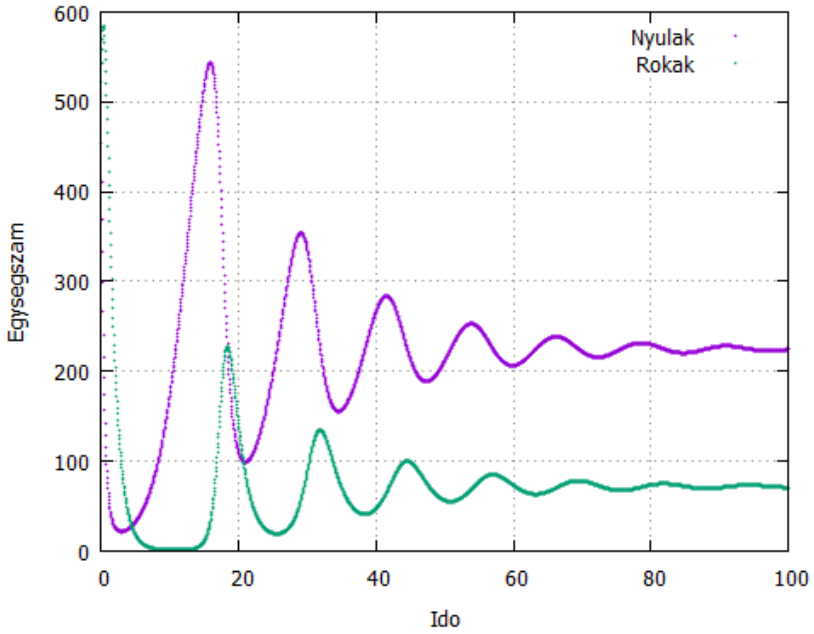
és a telítődést:

$$n_R n_F \Rightarrow \frac{n_R n_F}{1 + n_R s}$$

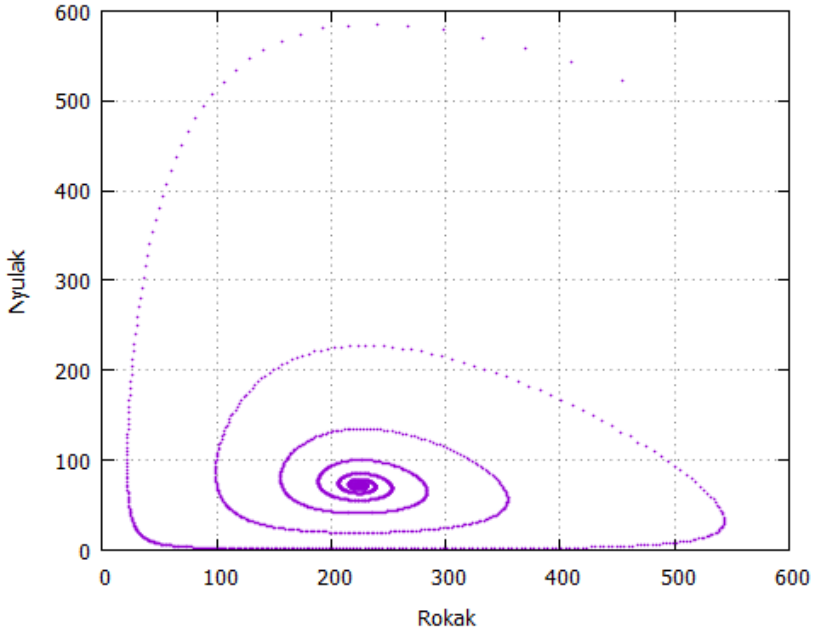
A használt kódrészlet:

```
double* RealisticLV(double nr, double nf)
{
    double np[2];
    np[0] = a*nr*(1-nr/K)-b*nr*nf/(1+nr/s);
    np[1] = c*nr*nf/(1+nr/s)-d*nf;
    return np;
}
```

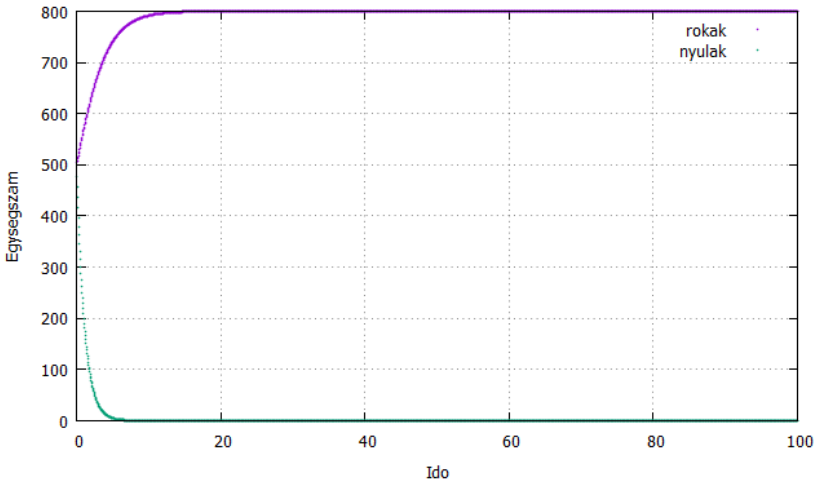
Kezdeti feltételnek a nyulak és a rókák számát egyaránt 500-nak vettem.



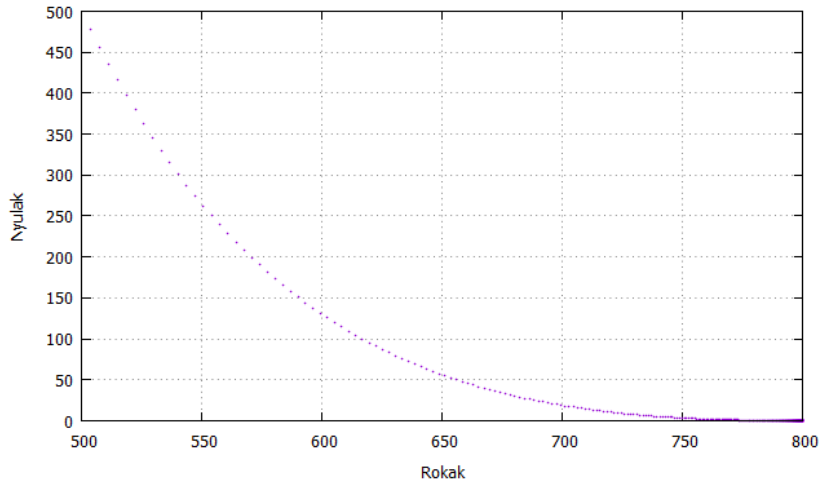
6. ábra. Egységszám az idő függvényében a következő paraméterrekkel: a = 0.4, b = 0.004, c = 0.004, d = 0.9, K = 800



7. ábra. A rókák és nyulak száma a következő paraméterrekkel: a = 0.4, b = 0.004, c = 0.004, d = 0.9, K = 800, s = 0



8. ábra. Egységszám az idő függvényében a következő paraméterrekkel: a = 0.4, b = 0.004, c = 0.004, d = 0.9, K=800, s = 2500



9. ábra. A rókák és nyulak száma a következő paraméterrekkel: a = 0.4, b = 0.004, c = 0.004, d = 0.9, K=800, s = 2500

4. Függelék

4.1. A használt kód

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

double nf = 500; // number of foxes
double nr= 500; //number of rabbits

double r = 1; //logistic rate
double a = 0.4; //rate of ...
double b = 0.004; // rate of ...
double c = 0.004;//rate of ...
double d = 0.9; // rate of ...
double K = 800; // capacity
double s = 2500;
double alpha = 2.3;

double beta =0.02;

double r1 = 0.5;
double r2 = 0.5;
double k1 = 20;
double k2 = 20;
```

```
// defining the functions:

double* LotkaVolterra(double nr, double nf){
    double np[2];
    np[0] = a*nr-b*nr*nf;
    np[1] = c*nr*nf-d*nf;
    return np;
}

double* competitive(double x, double y)
{
    double np[2];
    np[0] = r1*x*(1-(x+alpha*y)/k1);
    np[1] = r2*y*(1-(y+beta*x)/k2);
    return np;
}

double* CapacityLV(double nr, double nf)
{
    double np[2];
    np[0] = a*(1-nr/K)*nr-b*nr*nf;
    np[1] = c*nr*nf-d*nf;
    return np;
}

double* SaturationLV(double nr, double nf)
{
    double np[2];
    np[0] = a*nr-b*nr*nf/(1+nr/s);
    np[1] = c*nr*nf/(1+nr/s)-d*nf;
    return np;
}

double* RealisticLV(double nr, double nf)
{
    double np[2];
    np[0] = a*nr*(1-nr/K)-b*nr*nf/(1+nr/s);
    np[1] = c*nr*nf/(1+nr/s)-d*nf;
    return np;
}


double* eulerStep(double x, double tau, double y, function<double* (double,double)> f1)
{
    x = x+tau*f1(x, y)[0];
    y = y+tau*f1(x, y)[1];
    double ret[2];
    ret[0] = x, ret[1] = y;
    return ret;
}
```

```
double* RK4Step(double nr, double tau, double nf, function<double*
                (double,double)> f1)
{
    double k1 = tau * f1(nr, nf)[0];
    double k2 = tau * f1(nr+0.5 * k1, nf)[0];
    double k3 = tau * f1(nr+0.5 * k2, nf)[0];
    double k4 = tau * f1(nr+k3, nf)[0];
    double k5 = tau * f1(nr, nf)[1];
    double k6 = tau * f1(nr,0.5 * k5+nf)[1];
    double k7 = tau * f1(nr,0.5 * k6+nf)[1];
    double k8 = tau * f1(nr,k7+nf)[1];
    nr += (k1 + 2 * k2 + 2 * k3 + k4) / 6.0;
    nf += (k5 + 2 * k6 + 2 * k7 + k8) / 6.0;
    double ret[2];
    ret[0] = nr;
    ret[1] = nf;
    return ret;
}

int main(int argc, const char * argv[])
{
    // insert code here...

    cout << "Let the simulation begin!\n";

    ofstream file ("pop.data");

    for(int i = 0; i<100000; i++)
    {
        double x = nr;
        //  nr =eulerStep(nr, 0.00001, nf, Logistic)[0];
        //  nf =eulerStep(x, 0.00001, nf, Logistic)[1];

        nr =RK4Step(nr, 0.00001, nf, Logistic)[0];
        nf =RK4Step(x, 0.00001, nf, Logistic)[1];
        file << x/K <<'\t'<<nf << '\n';
    }
    file.close();
    cout << "Finished"<< endl;

    return 0;
}
```