Kutatómunka információs eszközei gyakorlat: Dokumentáció

Horváth Bendegúz, ZNL3LK

2018. május 26.

Feladat ismertetése, bevezetés

A program, amit megvalósítottunk egy egyszerű, két fajt kezelő populációdinamika szimuláció, `c++` nyelven. A fejlesztést a *github* oldalon hangoltuk össze, mindenki írt egy kis részt.

Populációdinamika az élőlényféleségek egyedszám- és népességviszonyainak térbeli és időbeli változásával foglalkozik, valamint ezek szerepével az emberek életkörülményeire, életlehetőségeire. Ezen kívűl a popluációdinamika általános keretrendszernek tekinthető, az egyenleteit meghatározó csatolt differenciálegyenletekbe behelyettesíthetjük vegyületek koncentrációit, vagy a gazdaság modellezésekor termelőket és fogyasztókat[1].

Elméleti bevezető

A populáció létszámának (n) változását vizsgáljuk, az idő (t) függvényében. A legegyszerűbb modellben nincs más tényező, csak a szaporodása az egyedeknek, ami a létszámmal arányos:

$$n(t + \Delta t) = n(t) + an(t),$$

ahol a a szaporodási ráta. Δt-t infinitezimálisan kicsinek választva átírható differenciálegyneletre:

$$\frac{dn}{dt} = an(t).$$

Ennek megoldása az exponenciális növekedés, ami nagyon gyorsan elszál. A modell realisztikuabbát tételének érdekében bevezethető a halálozási ráta:

$$\frac{dn}{dt} = an - dn,$$

$r = a - d$ helyettesítéssel a megoldás $n(t) = e^{rt}$, axponenciálisan növekvő vagy csökkenő görbe, $r = 0$ helyen nem stabil. Ha bevezetjük a korlátosságot az egyedszában, például az élelem véges kapacitású:

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

ahol k a kapacitás, ha ezt eléri n, a további növekedés nem lehetséges. Ez az egyenlet a logisztikus egyenlet, átskálázva $x = n/k$-val:

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

aminek megoldása r-től és x_0-tól függően növekvő vagy csökkenő szigmoid görbe:

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

Ha egy élőhelyen (niche) több faj küzd egyazon táplálékért, a véges erőforrásokon keresztül kölcsönhatásba kerülnek. Egymáshoz viszonyított szaporodási rátájuk és a környezet eltartóképessége függvényében a rátermettebb faj akár teljesen el is foglalhatja a nichet. Ezt a kompeticiót a következő differenciálegyenlettel lehet leírni:

$$\frac{dn_1}{dt} = r_1 n_1 (1 - \frac{n_1 + \alpha n_2}{k_1})$$
$$\frac{dn_2}{dt} = r_2 n_2 (1 - \frac{n_1 + \beta n_1}{k_2}),$$

az egyneletekebn k_i az i-edik faj erőforrásának kapacitása, α és β pedig azt jelenti, hogy mennyit fogyaszt az egyik a másikéból. Két faj versengést nem csak közös erőforrásért folytathat, hanem ragadozó-préda viszpn is fennállhat. Ezt az esetet a Lotka-Volterra modell írja le:

$$\frac{dn_r}{dt} = an_r - bn_f nr$$
$$\frac{dn_f}{dt} = cn_r n_f - dn_f$$

n_r a nyulak száma, a a nyulak szaporodásirátája, bn_f a nyulak pusztulásirátája, n_f a rókák száma, cn_r a rókák szaporodásirátája, és d a rókák pusztulásirátája. Ez a modell realiztikusabbá tehető, ha ha korlátozzuk a rókák nyúlfogyasztási képességét és ha a nyulak táplálékforrását korlátozzuk. A paramétereken a következő módosítást kell végrehajtani[2] :

$$n_r n_f \rightarrow \frac{n_r n_f}{1 + n_r/S},$$

és

$$a \rightarrow a(1 - \frac{n_r}{k}).$$

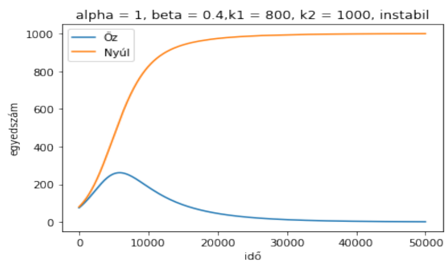
Megvalósítás

Kettő diferecniálegyenlet megoldó módszert készítettünk, kezdetnek egy Euler módszert, és egy Runge-Kuttát. A megvalósított differenciálegyenlet megoldó függvények 2 elemű tömböt adnak vissza a frissített rókák és nyulak számával, bemenetként pedig 4 értéket várnak, a nyulak számát, a rókák számát, a `dt` lépéshosszt, és egy függvényt, hogy melyik differenciálegynletrendszert oldják meg. Egy `for` ciklusban kell meghívni őket, és egy fájlba írja az egyedszámot. Fordítása a `g++ popdin.cpp -std=c++11` paranccsal lehetséges, valamint a mellékelt makefile-lal. A teljes kód megtalálható a függelékben valamint a *github repositoryban*[3].

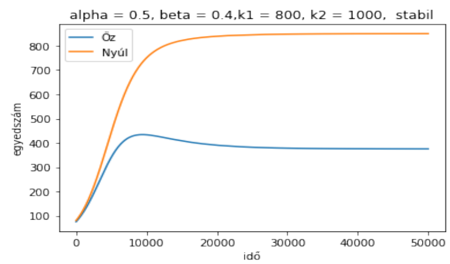
Közös erőforrásért való versengés

Az implementált függvény:

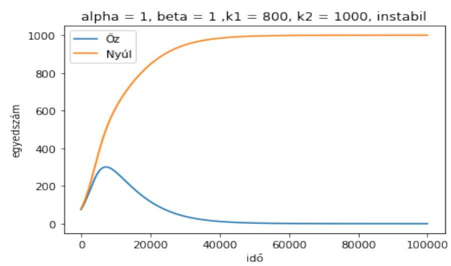
```
double* competitive(double x, double y){
    double np[2];
    np[0] = r1*x*(1-(x+alpha*y)/k1);
    np[1] = r2*y*(1-(y+beta*x)/k2);
    return np;
}
```



(a)

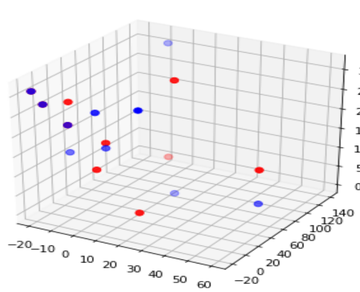


(b)

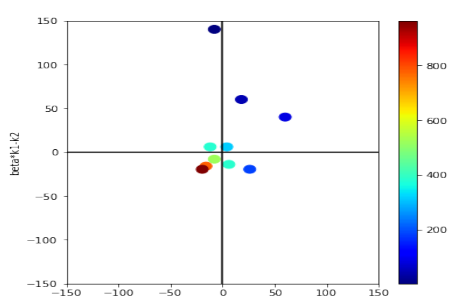


(c)

1. ábra. (a) a nyulak teljesen kiszorítják az őzeket (b) a két faj megél egymás mellett (c) betelítődik a nyulak száma, kiszorítják az őzeket.



(a)



(b)

2. ábra. (a) a nyulak és az őzek egyedszáma az $\alpha k_2 - k_1 : \beta k1 - k2$ sík felett(b) a két faj egyedszámának szorzata ábrázolva az $\alpha k_2 - k_1 : \beta k1 - k2$ síkon.

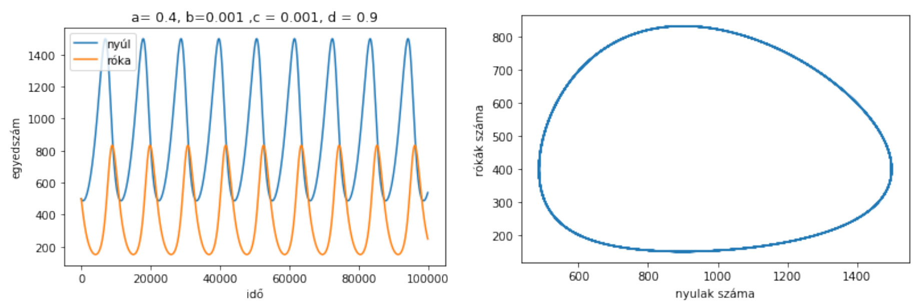
A 2 ábra b részén látható, hogy abban a síknegyedben a nagyobb az egyedszám, amikor $\alpha k_2 - k_1$ és $\beta k_1 - k_2$ is negatív. Ez azt jelenti, hogy a két faj megél egymás mellett, mert mindegyikből megközelítőleg egyforma van, nem szorítják ki egymást. A létszámuk összegének van egy felső határa a véges erőforrásaik miatt, ezért a szorzatuk akkor a legnagyobb, ha mind a kettőből egyforma mennyiségű van.

Lotka-Volterra-modell

Egyszerű Lotka-Volterra

Az implementált függvény:

```
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;
}
```



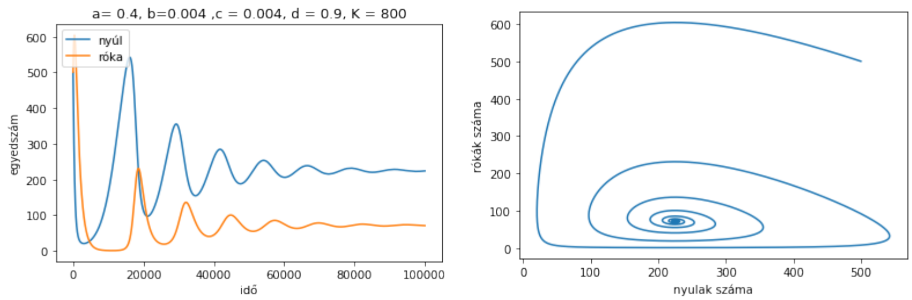
(a) (b)

3. ábra. Az egyszerű Lotka-Volterra-modellel a populáció változása és a fázistér.

Lotka-Volterra kapacitással

Az implementált függvény:

```
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;
}
```



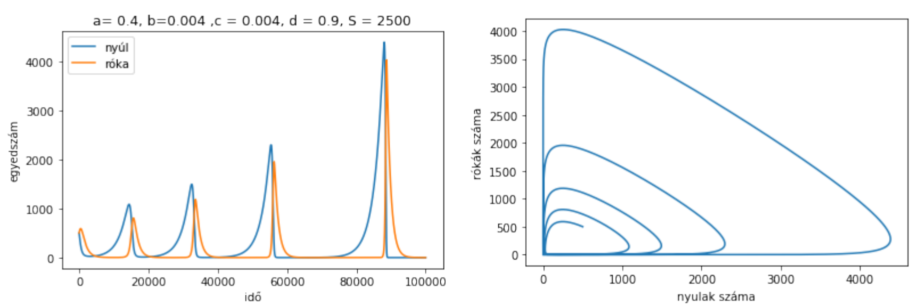
(a) (b)

4. ábra. Az kapacitással módosított Lotka-Volterra-modellel a populáció változása és a fázistér.

Lotka-Volterra telítődéssel

Az implementált függvény:

```
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;
}
```



(a) (b)

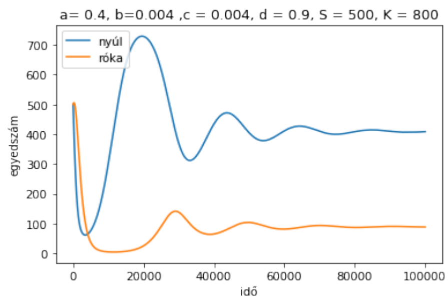
5. ábra. Az telítődéssel módosított Lotka-Volterra-modellel a populáció változása és a fázistér.

Lotka-Volterra telítődéssel és véges kapacitással

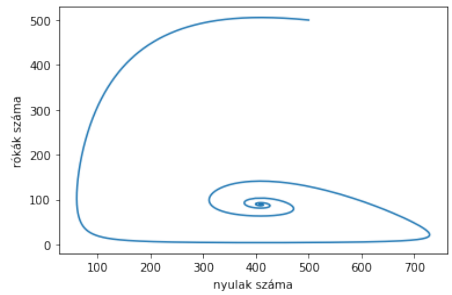
Az implementált függvény:

```
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;
```

```
}
```



(a)



(b)

6. ábra. A realisztikus Lotka-Volterra-modellel a populáció változása és a fázistér.

Diszkusszió

Az ábrák tanúsága szerint a szimuláció sikeresen működik, a kapott eredmények megegyeznek a vártakkal. A programot lehetne még "szebbé tenni", pédául a bemeneteket, a módokat valamint a differenciálegynelet megoldó módszer választását a parancssorban megadni, illetve 3 faj versengését kezelni.

Függelék

A. A teljes 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;
}
```

Hivatkozások

[1] https://hu.wikipedia.org/wiki/Populcidinamika

[2] http://csabai.web.elte.hu/http/szamszim/simLec6.pdf

[3] https://github.com/hbendeguz/Kutinfo_gyak