Véletlen fizikai folyamatok, nyolcadik házi feladat

Horváth Bendegúz

2018. április 23.

Szimulációk

A szimulációkat `python` nyelven készítettem el.

Véletlenszerű hálózat

A szimulációban egy N csúcsból álló hálózatot egy N elemű vektorban tároltam, a vektor N_i-edik elemének az értéke az i-edik csúcspont kapcsolatainak száma.

```python
network = [0]

def simulationStep(n, network):
    toConnect = int(random.uniform(0, n))
    network[toConnect] += 1
    network += [1]

for i in range(0,100):
    simulationStep(len(network), network)
```

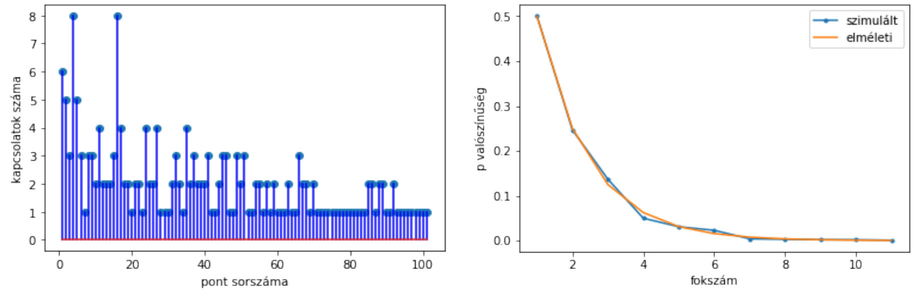
0. lépésben a hálózat egy csúcsból állt, értéke 0 volt. A szimulációs lépés ad nekünk egy egyenletes eloszlású vélelenszámgenerátorral egy 0 és a csúcspontok száma közötti egész típusú véletlenszámot, majd hozzákapcsolja az új csúcsot, és megnöveli az értékét eggyel. A szimuláció végén a hálózatot a következő módon elemezhetjük:

```python
Nk = np.zeros(max(network))
for i in range(len(network)):
    Nk[network[i]-1]+=1
```

Ebben az Nk_i értéke az $i+1$ fokszámú csúcsok száma. Az így kapott eredményeket a következő táblázatbn foglalom össze:

szimulációs lépés	fokszámok átlaga	legnagyobb fokszám
1000	1.998	12
1000	1.998	12
1000	1.998	11

A fokok számának átlagában nincs változás.



(a) (b)

1. ábra. (a) az egyes csúcsokhoz tartozó kapcsolatok száma 100 lépés után(b) a fokszámeloszlás, 1000 lépésből számolva.

Az órán számolt végeredmény a fokszámeloszlásra a következő volt:

$$P_k = e^{-k \ln 2}$$

ezt az elméleti görbét jól követi a szimulációból számolt görbe. Ahhoz, hogy megnézzük, mikor lesz 5%-os hibán belül fokszámeloszlás a következő módon futtattam a szimulációt:

```
def evalCondition(Nk):
    x = linspace(1, len(Nk), len(Nk))
    state = 0
    tf = True
    a = (exp(-x*log(2))-Nk)/exp(-x*log(2))
    for i in range(min(10, len(Nk))):
        if a[i] < 0.05:
            state += 1
        else:
            state += 0
    if state ==10:
        tf = False
    else:
        tf = True
    return tf

network = [0]
a = True
while a:
    simulationStep(len(network), network)
    Nk = np.zeros(max(network))
    for i in range(len(network)):
        Nk[network[i]-1]+=1
    a = evalCondition(Nk)
```

Az ebből kapott eredmények:

szükséges szimulációs lépés
1102
785
1211
727
870
699
868
1620
1714

A kapott eredmények átlaga: 1066.222, szórása : 358.287, így a szükséges lépések száma 1066.22 ± 358.287.
Az átlagos fokszámot 10^7 lépés során vizsgáltam, és a 5 tizedes pontossággal 1.99999-nek adott, így adódik a feltételezés, hogy $lim_{N\to\infty}\langle k\rangle = 2$. Az elméleti számolással a következő módon kaphatjuk meg, használhatjuk $P_k = \frac{1}{2^k}$ alakot:

$$\langle k\rangle = \sum_k^N kP_k = \sum_k^N k\frac{1}{2^k} = 2,$$

ha $N \to \infty$.

Antipreferenciális hálózat

A szimulációt úgy valósítottam meg (a feladat szövege szerint), hogyha a kapcsolódás rátája (w_k) kisebb mint egy véletlen P valószínűség, a kapcsolat nem jön létre, nem kerül új csúcs a rendszerbe.

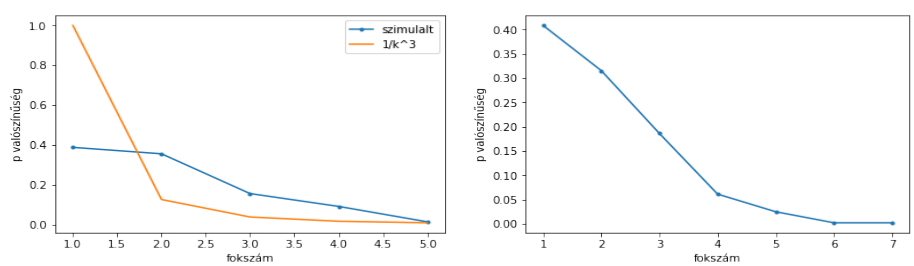
```
def calcNorm(network):
    Nk = np.zeros(max(network))
    for i in range(1, len(network)):
        Nk[network[i]-1]+=1
    a= 0
    for i in range(len(Nk)):
        a+= Nk[i]/(i+1)
    return a
def calcRates(k, network):
    A = calcNorm(network)
    return 1/A/network[k]
def simulationStep2(network):
    k = int(random.uniform(0,len(network)))

    w = calcRates(k, network)
    p = random.uniform(0, 1)
    if p < w:
        network[k] = network[k] + 1
        network += [1]
```

Ezáltal sokkal több iteráció kellett, a hálózat egyre lassabban nőtt.

iteráció	$\langle k \rangle$	hálózat nagyság
100	1.882352	17
1000	1.96153	52
10000	1.9854014	137
100000	1.9954648	441

A táblázatban látható, hogy $\langle k \rangle$ tart a 2 felé.



(a) (b)

2. ábra. (a) fokszámeloszlás és az $1/k^3$ függvény(b) a fokszámeloszlás, 10000 lépésből számolva.

Az ábrán látható, hogy nagyobb k értékekre közelíti az $1/k^3$ függvényt.

Eltolt lineáris preferenciális

Az előadáson megbeszéltek alapján a egy k fokszámú csúcshoz csatolódás valószínűsége $w_k = \frac{(k+\lambda)}{\sum_k (k+\lambda)N_k}$. Ezzel felírhatjuk a Master egyenletet:

$$N_k(N+1) = N_k(N) - w_k N_k + w_{k-1} N_{k-1}$$

$$N_1(N+1) = N_1(N) - w_1 N_1 + 1$$

Kihasználva, hogy $w_k = \frac{k+\lambda}{(2+\lambda)N}$, a következő egyenletet írhatjuk fel:

$$\frac{dN_k}{dN} = -\frac{k+\lambda}{2+\lambda} P_k + \frac{k+\lambda-1}{2+\lambda} P_{k-1},$$

ahol $P_k = N_k/N$. Kihasználva ezt:

$$\frac{dN_k}{dN} = \frac{dNP_k}{dN} = P_k + N\frac{dP_k}{dN} = P_k + \frac{dP_k}{d\ln N}$$

$$\frac{dP_k}{d\ln N} = -\frac{k+\lambda+(2+\lambda)}{2+\lambda} P_k + \frac{k+\lambda-1}{2+\lambda} P_{k-1}$$

$$\frac{dP_1}{d\ln N} = -\frac{2\lambda+3}{2+\lambda} P_1 + 1$$

Ebből $P_1 = \frac{2+\lambda}{2\lambda+3}$, ha egyensúlyi helyzetben nézzük, és az egyenlet bal oldalát 0-val tesszük egyenlővé. Most megnézve $k \neq 1$-re:

$$P_k = \frac{k+\lambda-1}{k+2\lambda+2} P_{k-1}.$$

Kihasználva a rekurziós összefüggést:

$$P_k = \frac{(k+\lambda-1)!}{(k+2\lambda+2)!} P_1$$

$\lambda = 1$ esetben:

$$P_k = \frac{4 \cdot 3 \cdot 2}{(k+4)\cdot(k+3)\cdot(k+2)\cdot(k+1)} P_1 = \frac{4 \cdot 3 \cdot 2}{(k+4)\cdot(k+3)\cdot(k+2)\cdot(k+1)} \frac{3}{5}$$

És így valóban, $P_k \sim k^{-4}$, ha $\lambda = 1$. Innen már belátható általános λ-ra is, mert amikor a faktoriálisokat egymással osztunk, a nevező mindig $(3+\lambda)$-nyival tartalmaz több k-s tagot.