Korelirana ravnovesja

Melisa Brulić

5. junij 2024

Motivacija

Igra Strahopetec (*Game of Chicken*):

	0	1
0	4,4	1,5
1	5,1	0,0

Nasheva ravnovesja:

- $(0,1)$
- $(1,0)$
- $\left(\begin{pmatrix} 0 & 1 \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}\right)$

Korelirano ravnovesje:

	0	1
0	0	$\frac{1}{2}$
1	$\frac{1}{2}$	0

Bimatrične igre

Obravnavamo $n \times m$ bimatrične igre med dvema igralcema.
Bimatrična igra je par matrik $(A, B) \in (\mathbb{R}^{n \times m})^2$, kjer je $A = [a_{ij}]$ izplačilna matrika igralca 1 in $B = [b_{ij}]$ izplačilna matrika igralca 2.

Matriki združimo v bimatriko izplačil:

$$\begin{bmatrix} a_{00}, b_{00} & a_{01}, b_{01} & \cdots & a_{0,m-1}, b_{0,m-1} \\ a_{10}, b_{10} & a_{11}, b_{11} & \cdots & a_{1,m-1}, b_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0}, b_{n-1,0} & a_{n-1,1}, b_{n-1,1} & \cdots & a_{n-1,m-1}, b_{n-1,m-1} \end{bmatrix}$$

Izid igre je vektor potez obeh igralcev, torej (i, j), kjer $0 \leq i \leq n-1, 0 \leq j \leq m-1$.
Množica vseh možnih izidov je $\{0, \ldots, n-1\} \times \{0, \ldots, m-1\}$.

Korelirano ravnovesje

Korelirano ravnovesje je slučajna porazdelitev x na množici vseh izidov, kjer velja:

$$\sum_{s=0}^{m-1} (a_{is} - a_{js}) x_{is} \geq 0 \qquad \forall i, j = 0, \ldots, n-1$$

$$\sum_{s=0}^{n-1} (b_{si} - b_{sj}) x_{si} \geq 0 \qquad \forall i, j = 0, \ldots, m-1$$

Implementacija bimatrične igre v Pythonu

```python
class Igra:
    def __init__(self, izplacilna1=None, izplacilna2=None, n=None, m=None):
        # Konstruktor razreda za ustvarjanje n*m bimatrične igre

        if izplacilna1 is None and izplacilna2 is None:

            if n is None or m is None:
            # nimamo parametrov, n in m sta naključni števili med 2 in 5
                self.n = np.random.randint(2, 6)
                self.m = np.random.randint(2, 6)
            else:
                self.n = n
                self.m = m

            # izplačilni matriki sta naključni matriki velikosti n*m z elementi med 0 in 10
            self.izplacilna1 = np.random.randint(0, 11, size=(self.n, self.m))
            self.izplacilna2 = np.random.randint(0, 11, size=(self.n, self.m))

        else:
            self.n, self.m = len(izplacilna1), len(izplacilna1[0])
            self.izplacilna1 = np.array(izplacilna1)
            self.izplacilna2 = np.array(izplacilna2)

        self.igra = nash.Game(self.izplacilna1, self.izplacilna2)
```

Računanje Nashevega ravnovesja

```
def izračunaj_nashevo_ravnovesje(self):
    # izračunamo vsa Nasheva ravnovesja dane igre s pomočjo vgrajenega algoritma Vertex enumeration
    nasheva_ravnovesja = list(self.igra.vertex_enumeration())

    seznam_NR = []
    n, m = self.n, self.m

    # zapišemo Nasheva ravnovesja v obliki kot bodo zapisana korelirana
    for strategija1, strategija2 in nasheva_ravnovesja:
        NR = np.array([strategija1[i] * strategija2[j] for i in range(n) for j in range(m)])
        seznam_NR.append(NR)

    return tuple(seznam_NR)
```

Ker algoritem vrne ravnovesja v obliki:

```
[(array([-6.9388939e-17,  1.0000000e+00]), array([ 1.00000000e+00, -2.77555756e-17]))]
```

Zapišemo na način kot bojo zapisana korelirana ravnovesja

```
(array([-6.93889390e-17,  1.92592994e-33,  1.00000000e+00, -2.77555756e-17]),)
```

Računanje koreliranih ravnovesij

Korelirana ravnovesja $x = (x_{00}, x_{01}, \ldots, x_{n-1,m-1})$ so rešitve LP:

$$Ux \geq 0$$

$$\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} x_{ij} = 1$$

$$x \geq 0,$$

kjer je U matrika, ki jo dobimo iz pogojev za korelirano ravnovesje. Prve $n(n-1)$ vrstice dobimo iz pogojev za igralca 1:

$$\sum_{s=0}^{m-1} (a_{is} - a_{js}) x_{is} \geq 0 \qquad \forall i, j = 0, \ldots, n-1$$

Preostalih $m(m-1)$ vrstic pa iz pogojev za igralca 2:

$$\sum_{s=0}^{n-1} (b_{si} - b_{sj}) x_{si} \geq 0 \qquad \forall i, j = 0, \ldots, m-1$$

Matrika U

```python
def U(self):
    # konstruiramo matriko U, s katero so opisani pogoji za korelirano ravnovesje

    A = self.izplacilna1
    B = self.izplacilna2
    n, m = self.n, self.m

    U = np.zeros((n*(n-1) + m*(m-1),n*m))

    vrstica = 0

    # neenakosti za igralca 1:
    for i in range(n):
        for j in [x for x in range(n) if x != i]:
            for s in range(m):
                U[vrstica, i*m + s] = A[i, s] - A[j, s]
            vrstica +=1

    # neenakosti za igralca 2:
    for i in range(m):
        for j in [x for x in range(m) if x != i]:
            for s in range(n):
                U[vrstica, s*m + i] = B[s, i] - B[s,j]
            vrstica += 1
    return U
```

Reševanje LP

V projektu poiščemo samo korelirana ravnovesja, ki rešijo linearen program, ki maksimizira:

- koristnost igralca 1
- koristnost igralca 2
- skupno koristnost

LP rešimo s funkcijo `linprog` iz Pythonove knjižnice `Scipy`, zato moramo LP prevesti v drugo obliko:

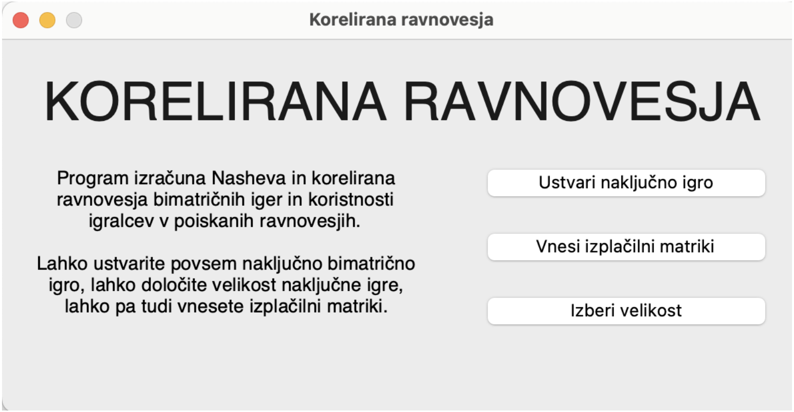
$$
\begin{aligned}
\max\ & c^T x \\
& Ux \geq 0 \\
& \mathbf{1}x = 1 \\
& x \geq 0
\end{aligned}
\qquad\qquad
\begin{aligned}
\min\ & -c^T x \\
& -Ux \leq 0 \\
& \mathbf{1}x = 1 \\
& x \geq 0
\end{aligned}
$$

Reševanje LP

```
def izračunaj_korelirana_ravnovesja(self):
    # z reševanjem treh linearnih programov izračuna 3 korelirana ravnovesja:
        # KR, ki maksimizira izplačilo igralca 1, izplačilo igralca 2, skupno izplačilo

    U = self.U() # konstruiramo matriko U iz definicije
    U_minus = -1*U # preoblikujemo U za LP

    A = self.izplacilna1
    B = self.izplacilna2

    n = self.n
    m = self.m

    ##### koeficienti ciljne funkcije
    cA = -1*A.reshape(-1) # če maksimizirano koristnost igralca 1
    cB = -1*B.reshape(-1) # če maksimiziramo koristnost igralca 2

    C = A + B
    cC = -1*C.reshape(-1) # če maksimiziramo skupno koristnost

    ##### pogoji
    # desna stran neenakosti -Ux<= 0
    b = np.zeros(n*(n-1) + m*(m-1))

    # 1x = 1
    A_eq = np.ones((1, n*m))
    b_eq = np.array([1])

    # x>= 0
    x_meje = (0, float("inf"))  # x_i >= 0 for all i, x <=1

    ##### LP
    # LP pri katerem maksimiziramo koristnost igralca 1
    rezultatA = linprog(c = cA, A_ub = U_minus, b_ub = b, A_eq = A_eq, b_eq = b_eq, bounds = [x_meje]*n*m)

    # LP pri katerem maksimiziramo koristnost igralca 2
    rezultatB = linprog(c = cB, A_ub = U_minus, b_ub = b, A_eq = A_eq, b_eq = b_eq, bounds = [x_meje]*n*m)

    # LP pri katerem maksimiziramo skupno koristnost
    rezultatC = linprog(c = cC, A_ub = U_minus, b_ub = b, A_eq = A_eq, b_eq = b_eq, bounds = [x_meje]*n*m)

    return rezultatA.x, rezultatB.x, rezultatC.x
```

Pomožne funkcije

```
def izračunaj_koristnost(self, x, igralec=None):
    # če ne podamo igralca izračuna skupno zadovoljstvo, sicer pa izračuna koristnost danega igralca
    A = self.izplacilna1
    B = self.izplacilna2
    C = A + B

    if igralec == 1:
        return np.matmul(A.reshape(-1), x)
    elif igralec == 2:
        return np.matmul(B.reshape(-1), x)
    else:
        return np.matmul(C.reshape(-1), x)

def zapisi_kot_matriko(self,ravnovesje):
    # funkcija, ki zapiše vektor (ravnovesje) kot n*m matriko - za lažje izpisovanje v GUI
    return np.array(ravnovesje).reshape((self.n,self.m))

def nova_ravnovesja(self):
    # preveri ali smo našli še kakšno korelirano ravnovesje

    CR = self.zapiši_samo_različna_KR()
    NR = self.izračunaj_nashevo_ravnovesje()

    return [ravnovesje for ravnovesje in CR if not any(np.allclose(ravnovesje, nashevo_ravnovesje, atol=1e-10) for nashevo_ravnovesje in NR)]

def zapiši_samo_različna_KR(self):
    # najdena različna KR

    CR = self.izračunaj_korelirana_ravnovesja()
    CR = [tuple(ravnovesje) for ravnovesje in CR]

    return tuple(set(CR))

def ali_je_nashevo(self, ravnovesje):
    # preveri ali je ravnovesje nashevo

    return any(np.allclose(ravnovesje, nr,1e-10) for nr in self.izračunaj_nashevo_ravnovesje())
```


Korelirana ravnovesja
KORELIRANA RAVNOVESJA
Program izračuna Nasheva in korelirana ravnovesja bimatričnih iger in koristnosti igralcev v poiskanih ravnovesjih.
Lahko ustvarite povsem naključno bimatrično igro, lahko določite velikost naključne igre, lahko pa tudi vnesete izplačilni matriki.
Ustvari naključno igro
Vnesi izplačilni matriki
Izberi velikost