Iterativni izračun Nashevega ravnotežja v matričnih igrah

Eva Babnik

Maj, 2022

1 Nashevo ravnotežje in vrednost igre v matričnih igrah

Spodobi se, da najprej v nekaj stavkih opišem Nashevo ravnotežje ter vrednost igre v matričnih igrah z ničelno vsoto za dva igralca. Matrično igro z dvema igralcema lahko predstavimo z matriko izplačil $A = (a_{ij})$, kjer prvi igralec izbere eno izmed m vrstic, drugi pa hkrati izbere enega izmed n stolpcev. $A_{i\cdot}$ naj označuje i-to vrstico, $A_{\cdot j}$ pa j - ti stolpec. Če igralca izbereta i-to vrstico in j-ti stolpec, potem drugi igralec plača prvemu igralcu a_{ij}.

Če prvi igralec izbere i-to vrstico z verjetnostjo x_i in drugi izbere j-ti stolpec z verjetostjo y_j, pri čemer velja:

$$x_i \geq 0, \tag{1}$$

$$\sum x_i = 1, \tag{2}$$

$$y_i \geq 0, \tag{3}$$

$$\sum y_i = 1. \tag{4}$$

Potem je pričakovano izplačilo prvemu igralcu $\sum\sum a_{ij}x_iy_j$. Poleg tega velja tudi:

$$\min_j \sum_i a_{ij}x_i \leq \max_i \sum_j a_{ij}y_j.$$

Trditev o minimaksu nam pove, da za neka vektorja verjetnosti $x = (x_1, \cdots, x_m)$ in $y = (y_1, \cdots, y_n)$ v zgornji enačbi velja enakost. Tak par (x^*, y^*) predstavlja optimalno strategijo igre. Vrednost igre v pa je definirana kot:

$$v = \min_j \sum_i a_{ij}x_i = \max_i \sum_j a_{ij}y_j.$$

Problem matrične igre za dva igralca lahko enostavno rešimo z linearnim programiranjem:

$$\begin{aligned} &\min pJ_m^T \\ &\text{p. p. } pA_j \geq 1, \quad j = 1, \cdots, n \\ &p \geq 0 \end{aligned}$$

in

$$\begin{aligned} &\max qJ_n \\ &\text{p. p. } A_iq^T \leq 1 \quad i = 1, \cdots, m \\ &q \geq 0, \end{aligned}$$

pri čemer sta J_m in J_n vektorja samih enic primernih velikosti. x^*, y^* in v^* dobimo kot $v^* = \frac{1}{\sum_1^n q_j^*} = \frac{1}{\sum_1^m p_i^*}$, $y_j^* = q_j^* v^*$ in $x_i = p_i^* v^*$. Algoritem, ki nam z linearnim programiranjem vrne vrednost matrične igre in optimalno strategijo, sem implemenitrala v funkciji *resitevLP.py*, ki kot vhodni argument sprejme matriko.

2 Iterativno računanje rešitve igre

V projektu sem implementirala dva iterativna algoritma, ki nam vrneta optimalno strategijo in vrednost igre ter analizirala kako hitra je konvergenca posameznih metod. Za implementacijo sem uporabila programski jezik *Python*, pri analizi metod pa sem si pomagala tudi s programskim jezikom *R*. Celotna koda, ki vsebuje vse implementacije iterativnih algoritmov, se nahaja v datoteki *projekt.py*, analiza metod pa se nahaja v datoteki *analiza.R*

2.1 Metoda I

Najprej uvedimo še nekaj nove notacije. Naj velja:

$$A_i = [a_{1i}, \cdots, a_{mi}, \underbrace{0, \cdots, 0}_{n-\text{komponent}}, -1],$$
$$A_{0i} = [\underbrace{1, \cdots, 1}_{m-\text{komponent}}, \underbrace{0, \cdots, 0}_{n-\text{komponent}}, 0]$$
$$\text{za } i = 1, \cdots, n$$

in

$$A_i = [\underbrace{0, \cdots, 0}_{m-\text{komponent}}, -a_{i-n,1}, \cdots, -a_{i-n,n}, 1],$$
$$A_{0i} = [\underbrace{0, \cdots, 0}_{m-\text{komponent}}, \underbrace{1, \cdots, 1}_{n-\text{komponent}}, 0]$$
$$\text{za } i = n+1, \cdots, m+n.$$

Definirajmo še vektor, ki predstavlja rešitev igre: $z^* = [x^*, y^*, v]$. Z^* mora poleg 1, 2, 3, 4, ustrezati še pogoju:

$$A_i \cdot Z^* \geq 0 \text{ za } i = 1, \cdots, m+n. \tag{5}$$

Metoda se začne s poljubnim vektorjem $z^{(1)}$, ki zadošča 1, 2, 3 in 4. Sedaj predpostavimo, da smo prišli na k-ti korak iteracije, in dobili vektor $z^{(k)}$,

ki ustreza 1, 2, 3 in 4. Če velja tudi 5, je $z^{(k)}$ rešitev igre in smo zaključili. Sicer pa naj bo j_k tak indeks, da bo veljalo $A_{j_k} \cdot z^{(k)} \leq A_i \cdot z^{(k)}$ za vse $i = 1, \cdots, m+n$. Če obstaja več takih indeksov, lahko poljubno izberemo. Če torej poznamo indeks j_k, lahko dobimo nov vektor $\bar{z}^{(k+1)} = [\bar{x}^{(k+1)}, \bar{y}^{(k+1)}, \bar{v}^{(k+1)}]$ na sledeči način:

$$\bar{z}^{(k+1)} = z^{(k)} + \alpha B_{j_k} + \beta B_{0j_k},$$

kjer je

$$\alpha = -z^{(k)} \cdot B_{j_k}[1 - \cos^2 \theta_{j_k}]^{-1},$$

$$\beta = b_{0j_k} - [z^{(k)} + \alpha B_{j_k}] \cdot B_{0j_k},$$

$$b_{0j_k} = \frac{1}{(A_{0j_k} \cdot A_{0j_k})^{1/2}},$$

$$B_{j_k} = \frac{A_{j_k}}{(A_{j_k} \cdot A_{j_k})^{1/2}},$$

$$B_{0j_k} = \frac{A_{0j_k}}{(A_{0j_k} \cdot A_{0j_k})^{1/2}}$$

in

$$\cos \theta_{j_k} = \frac{A_{0j_k} \cdot A_{j_k}}{(A_{0j_k} \cdot A_{0j_k})^{1/2}(A_{j_k} \cdot A_{j_k})^{1/2}}.$$

Sedaj predpostavimo, da velja $j_k < (n+1)$. (V primeru, da bi bil $j_k \geq n+1$, bi komponente x ostale nespremenjene, postopek, opisan v nadaljevanju, pa bi veljal za y komponente.) Če $\bar{z}^{k+1}$ ustreza 5, potem nastavimo $z^{k+1} = \bar{z}^{k+1}$, v nasprotnem primeru pa moramo, da dobimo Z^{k+1}, narediti še nekaj korakov. Najprej vse negativne x-komponente vektorja $\bar{z}^{k+1}$ nastavimo na 0. Predpostavimo, da so $\bar{x}_1^{(k+1)}, \cdots, \bar{x}_r^{(k+1)}$, $r < m$ negativne komponente vektorja $\bar{Z}^{(k+1)}$. Nato izračunamo vse vsote $\bar{x}_i^{(k+1)} + \frac{\sum_{i=1}^{r} \bar{x}_i^{(k+1)}}{m-r}$ za $i = r+1, \cdots, m$. Za vsak tak i, za katerega je vsota negativna, nastavimo $x_i^{(k+1)} = 0$. Če nobena vsota ni negativna, lahko tvorimo preostanek vektorja $z^{(k+1)}$. Spet predpostavimo, da so nekatere vsote za $i = r+1, \cdots, r+s$ negativne. Ponovno izračunamo vsote $\bar{x}_i^{(k+1)} + \frac{\sum_{i=1}^{r+s} \bar{x}_i^{(k+1)}}{m-(r+s)}$ za $i = r+s, \cdots, m$. Če nobena vsota ni negativna, tvorimo preostanek vektorja $Z^{(k+1)}$, sicer pa ponavljamo zgornji postopek, dokler nobena od vsot ni nenegativna.

Predpostavimo, da za $i = 1, \cdots, t$ velja, da je $\bar{x}_i^{(k+1)} \leq 0$ ali pa, da je $\bar{x}_i^{(k+1)}$ tak, da je zanj katera od zgoraj definiranih vsot negativna. Potem lahko vektor $z^{(k+1)}$ tvorimo na sledeči način:

$$x_1^{(k+1)} = \cdots = x_t^{(k+1)} = 0,$$

$$x_i^{(k+1)} = \bar{x}^{(k+1)} + \frac{\sum_{i=1}^{t} \bar{x}_i^{(k+1)}}{m-t} \text{ za } i = t+1, \cdots, m,$$
$$y_j^{(k+1)} = \bar{y}_j^{(k+1)} \text{ za } j = 1, \cdots, n,$$
$$v^{(k+1)} = \bar{v}^{(k+1)}.$$

Izkaže se, da $z(k)$ konvergira proti pravi rešitvi matrične igre. Opisana metoda je implementirana s funkcijo *iteracija2* in sprejme dva argumenta - *steviloIteracij* in *matrika*. Za začetne približke sem vzela $x_0 = (1/m, \cdots, 1/m)$, $y_0 = (1/n, \cdots, 1/n)$, v_0 pa sem generirala iz enakomerne porazdelitve $U(a, b)$, kjer sta a in b najmanjši in največji element matrike po absolutni vrednosti. Funckija *iteracija2* po k-iteracijah vrne vektor $z^k = (x^k, y^k, v^k)$, ki predstavlja rešitev igre.

2.2 Metoda II

Naslednja metoda, ki sem jo implemenitrala se imenuje statistična Brownova metoda. Temelji na ideji, da so sedanje odločitve odvisne od zgodovine. 1. igralec najprej igra poljubno čisto strategijo x_{i_1}, pri čemer je i_1 komponenta iz množice $(1, \cdots, m)$ enaka 1, ostale komponente pa so enake 0. 2. igralec nato začne z vektorjem akumulativnih vsot $A^{(1)} = [a_1^{(1)}, \cdots, a_n^{(1)}]$, kjer $A^{(1)}$ predstavlja i_1-to vrstico matrike. 2. igralec nato igra čisto strategijo y_{j_1}, pri čemer je j_1 tak indeks iz množice $(1, \cdots, n)$, da velja, da je $a_{j_1}^{(1)}$ najmanjša komponenta vektorja $A^{(1)}$. Če je minimumov več načeloma vzamemo poljubnega, pri implementaciji pa sem vzela prvi minimum. 1. igralec nato začne z vektorjem akumulativnih vsot $B^{(1)} = [b_1^{(1)}, \cdots, b_m^{(1)}]$, pri čemer je $B^{(1)}$ j-ti stolpec matrike. 1. igralec tako uporabi čisto strategijo x_{i_2}, pri čemer je i_2 tak indeks iz množice $(1, \cdots, m)$, da velja, da je $b_{j_2}^{(1)}$ največja komponenta vektorja $B^{(1)}$ Če je maksimumov več načeloma vzamemo poljubnega, pri implementaciji pa sem vzela prvi maksimum. 1. igralec nato vektorju $A^{(1)}$ prišteje i_2 - to vrstico matrike, zatem pa 2. igralec igra na enak način kot prej. Na k-tem koraku imamo vektorja akumulativnih vsot $A^{(k)} = [a_1^{(k)}, \cdots, a_n^{(k)}]$ in $B^{(k)} = [b_1^{(k)}, \cdots, b_m^{(k)}]$. 1. igralec nato igra čisto strategijo x_{i_k+1}, pri čemer je i_{k+1} tak indeks, da je $b_{j_k+1}^k$ maksimalna komponenta vektorja $B^{(k)}$. $A^{(k+1)}$ dobimo tako, da vektorju $A^{(k)}$ prištejemo $j_{(k+1)}$-to vrstico matrike izplačil. 2. igralec tako igra čisto strategijo $y_{j_(k+1)}$, pri čemer je $j_{(k+1)}$ tak indeks, da je $a_{j_(k+1)}^{(k+1)}$ najmanjša komponenta vektorja $A^{(k+1)}$. Postopek ponavljamo. Dokazati se da, da zaporedje $\frac{\sum_{n=1}^{k} X_{i_n}}{k}$ ali konvergira k optimalni strategiji x^* ali pa ima tako podzaporedje. Prav tako tudi zaporedje $\frac{\sum_{n=1}^{k} y_{i_n}}{k}$

ali konvergira k optimalni strategiji y^* ali pa ima podzaporedje, ki konvergira. Vrednost igre pri tem algoritmu dobimo kot $\max_{k=1,2,\cdots} \frac{\min_{i\in(1,\cdots,n)} a_i^{(k)}}{k}$ in kot $\min_{k=1,2,\cdots} \frac{\max_{i\in(1,\cdots,m)} b_i^{(k)}}{k}$.

Opisana metoda je implementirana s funkcijo *iteracijaBrown* in sprejme 2 argumenta - *matrika* in *steviloIteracij*. Funkcija vrne optimalni strategiji po k-tih korakih iteracije $x^{(k)}$ in $y^{(k)}$, ter vrednost igre $v^{(k)}$. Za vrednost igre sem vzela povprečje $\max_{k=1,2,\cdots} \frac{\min_{i\in(1,\cdots,n)} a_i^{(k)}}{k}$ in $\min_{k=1,2,\cdots} \frac{\max_{i\in(1,\cdots,m)} b_i^{(k)}}{k}$

3 Analiza konvergence, časovna in rezultati

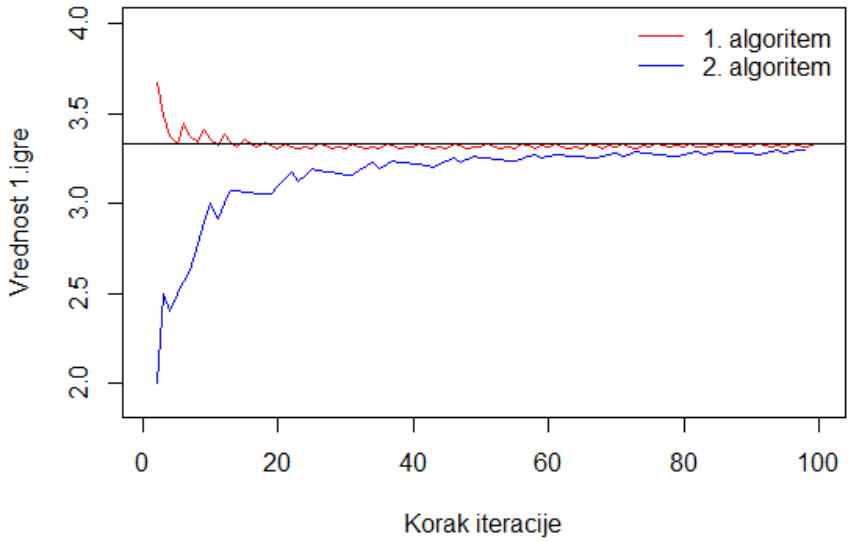
Časovna zahtevnost prvega algoritma je $\mathcal{O}(k(m^2 \vee n^2))$, časovna zahtevnost drugega algoritma pa znaša $\mathcal{O}(k)$.
Vse meritve sem opravila s prenosnim računalnikom s procesorjem 2.40GHz Intel(R) Core(TM) i5-6300U CPU in 8GB spomina. Naj najprej povem, da sem vsak poskus ponovila 10-krat in nato vzela povprečje rezultatov. Najprej si bomo pogledali, kako na treh matrikah različnih dimenzij delujejo zgoraj opisani algoritmi. Začnimo z matriko dimenzije 3×2:

$$\begin{bmatrix} 4 & 3 \\ 2 & 4 \\ 5 & 2 \end{bmatrix}$$

Vrednost prve matrične igre je $v^* = \frac{10}{3}$, medtem, ko je mešano Nahevo ravnovesje $x^* = [\frac{2}{3}, \frac{1}{3}, 0]$ in $y^* = [\frac{1}{3}, \frac{2}{3}]$.

Za začetek si poglejmo graf 1, ki prikazuje, kako hitro konvergira vrednost igre, dobljena z obema metodama proti pravi vrednosti, ki je označena s črno črto. Ta in vsi naslednji grafi zaradi boljše preglednosti prikazujejo le prvih 100 korakov iteracije.
Na grafu opazimo, da se vrednost igre, dobljene s prvim algoritmom, že po nekaj korakih iteracije popolnoma približa pravi vrednosti, medtem ko je konvergenca pri drugi metodi počasnejša. Zanimiv je tudi pojav, da se vrednost pri prvem algoritmu v nekaterih zgodnejših korakih prej približa pravi vrednosti igre, kot v poznejših. To bi lahko pomenilo, da prva metoda ne izbere nujno najbližje rešitve.

Naslednji graf 2 prikazuje kako se obnaša največja maksimalna napaka, dobljena pri obeh algoritmih v odvisnosti od korakov iteracije. Napaka je definirana kot $napaka = ||z^* - z^k||_\infty$. Vidimo, da je napaka najprej največja pri drugem algoritmu, a zelo strmo pada in pri 20.tem koraku prehiti napako, dobljeno s prvim algoritmom. Glede na to, da na prejšnjem grafu 1 hitreje



Slika 1: Vrednosti prve matrične igre

konvergirajo vrednosti, dobljene s prvim algoritmom, je najbrž konvergenca za optimalni strategiji x^* in y^* počasnejša pri prvem algoritmu.

Tabela 1 prikazuje konkretne vrednosti, dobljene z obema algoritmoma pri različnemu številu korakov. Vidimo, da je časovna zahtevnost prve metode precej večja od druge; pri 15000 iteracijah prva metoda za izračun rešitve igre porabi $5,5$-krat več časa. Pri $k = 1000$ imamo pri obeh metodah natančnost pri dveh decimalnih mestih za vse komponente. Pri $k = 15000$ prva metoda že popolnoma skonvergira k rešitvi igre, druga metoda pa je natačna za štiri decimalna mesta.

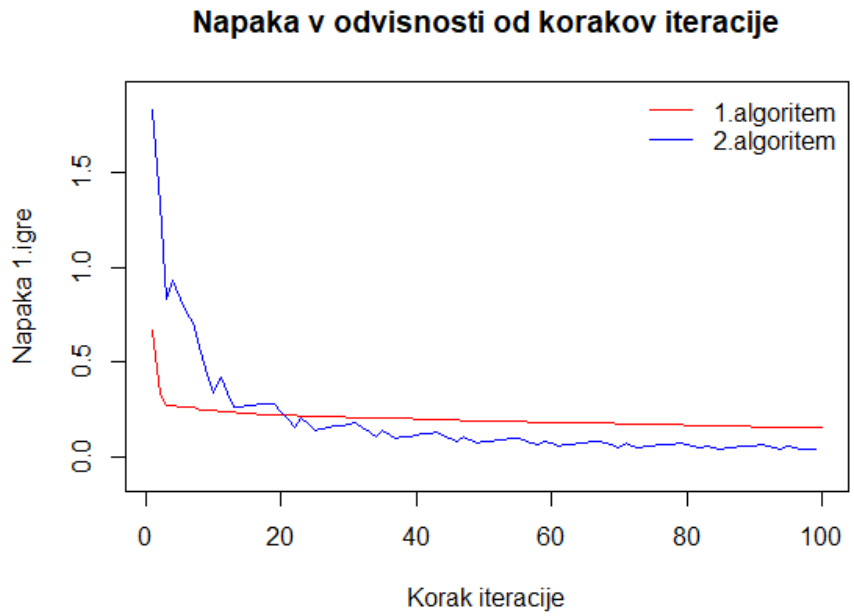
Nato si oglejmo matriko velikosti 7×7:

$$\begin{bmatrix} 0 & -1 & -2 & 0 & 0 & 2 & 1 \\ 1 & 0 & 0 & 4 & 0 & 0 & -1 \\ 2 & 0 & 0 & -1 & 1 & 0 & -1 \\ 0 & -4 & 1 & 0 & 0 & -1 & 1 \\ 0 & 0 & -1 & 0 & 0 & -3 & 1 \\ -2 & 0 & 0 & 1 & 3 & 0 & -1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 0 \end{bmatrix}$$

Ker je 2. matrična igra simetrična, je njena vrednost $v^* = 0$. Optimalni strategiji sta $x^* = [1/9, 1/9, 15/90, 5/90, 15/90, 5/90, 1/3]$ in $y^* =$

	1. metoda				2. metoda			
	čas	v	x	y	čas	v	x	y
k = 1	0.0020	3	0.33333 0.33333 0.33333	0 1	0.0009	1.5	0.5 0.5 0	0 0.5
k = 2	0.0020	3.49020	0.33333 0.33333 0.33333	0.5 0.5	0.0014	2	0.66667 0.33333 0	0.33333 0.33333
k = 10	0.0155	3.31150	0.34199 0.45804 0.19997	0.34425 0.65575	0.0020	3	0.45454 0.45454 0	0.27273 0.63636
k = 50	0.0156	3.33100	0.47646 0.40436 0.11918	0.33100 0.66900	0.0030	3.25490	0.64706 0.33333 0.01961	0.35294 0.62746
k = 100	0.0369	3.33147	0.51447 0.39017 0.09536	0.33146 0.66854	0.0049	3.28713	0.66337 0.32673 0.00990	0.33663 0.65347
k = 500	0.0953	3.33301	0.64108 0.34289 0.01603	0.33302 0.66670	0.0156	3.32335	0.66467 0.33333 0.00200	0.33134 0.66667
k = 1000	0.1207	3.33330	0.66391 0.33436 0.00173	0.33330 0.66670	0.0312	3.33290	0.66633 0.33257 0.00010	0.33367 0.66533
k = 15000	2.4360	3.33333	0.66667 0.33333 0	0.33333 0.66667	0.4397	3.33304	0.66656 0.33338 0.00007	0.33338 0.66656

Tabela 1: Primerjava rezultatov za 1. igro

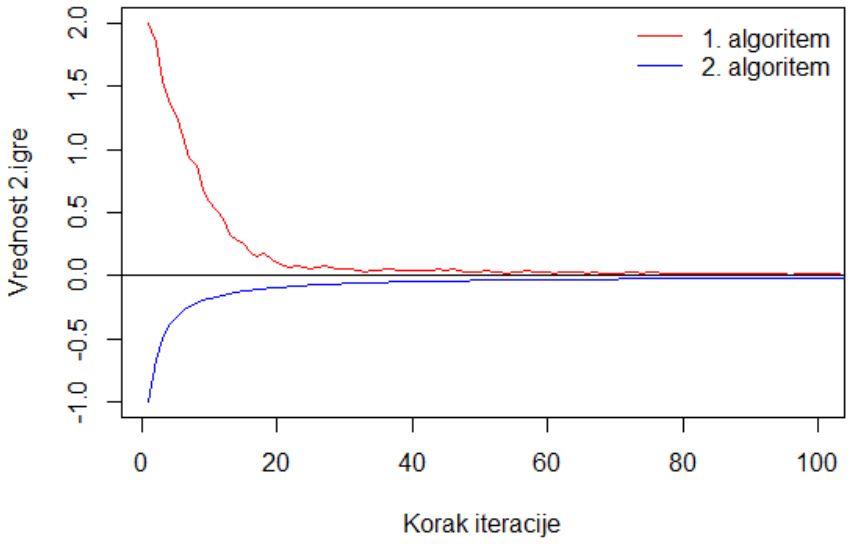


Slika 2: Napaka algoritmov za prvo matrično igro

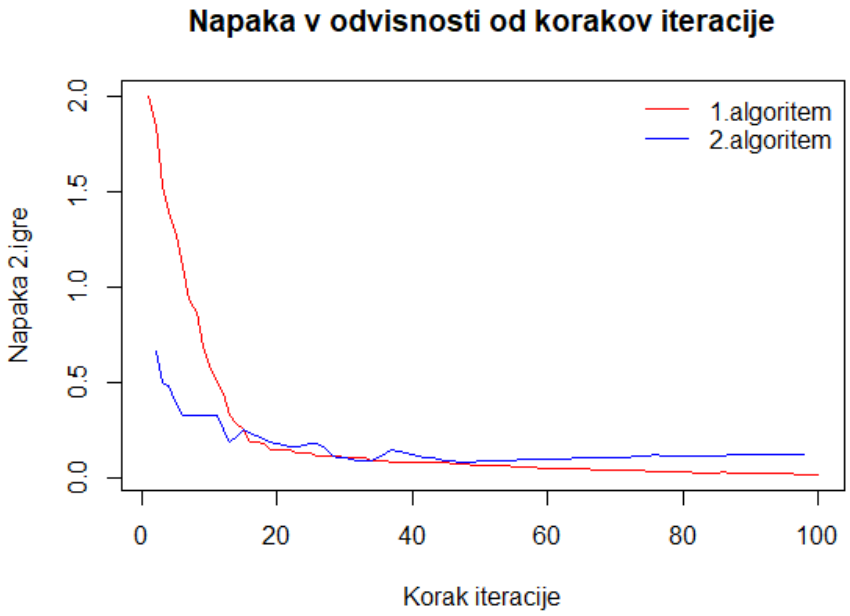
$[1/9, 1/9, 15/90, 5/90, 15/90, 5/90, 1/3]$. Podobno kot prej, si poglejmo graf 3, ki prikazuje, kako hitro konvergira vrednost 2. igre, dobljena z obema metodama proti pravi vrednosti ($v = 0$). Ponovno sem zaradi boljše preglednosti naredila le 100 korakov iteracije. Opazimo, da se vrednost igre, dobljene z drugim algoritmom, hitreje približa pravi vrednosti igre, kar je najverjetneje povezano z izbiro začetnega približka. Zdi se, da je konvergenca hitra pri obeh algoritmih.

Naslednji graf 4 prikazuje, kako se spreminja napaka, dobljena pri obeh algoritmih v odvisnosti od števila iteracij. Zanimivo je, da je situacija obratna, kot pri prvi matrični igri. Tu je konvergenca napake, dobljene s prvim algoritmom nekoliko hitrejša kot konvergenca napake, dobljene z drugim algoritmom.

Podobno kot prej, tabela 2 prikazuje konkretne vrednosti, dobljene z obema algoritmoma pri različnemu številu korakov. Pri $k = 500$ se prva metoda približa pravi rešitvi v dveh decimalnih mestih natačno pri vseh komponentah. Druga metoda konvergira nekoliko počasneje in se pravi rešitvi ne približa v dveh decimalnih mestih niti pri $k = 15000$, medtem ko se zdi, da pri takem številu korakov, prva metoda že popolnoma skonvergira. Zopet je čas izvajanja algoritma daljši pri prvi metodi.



Slika 3: Vrednosti druge matrične igre



Slika 4: Napaka algoritmov za prvo matrično igro

	1. metoda				2. metoda			
	čas	v	x	y	čas	v	x	y
k = 1	0.00365	1.84615	0.01282 0.16667 0.16667 0 0.16667 0.16667 0.32051	0.14286 0.14286 0.14286 0.14286 0.14286 0.14286 0.14286	0.00165	-1	0 0 0.5 0 0 0 0.5	0.5 0 0 0 0 0 0
k = 2	0.00489	0.73931	0.21520 0 0 0.20238 0.36904 0 0.21337	0.14286 0.14286 0.14286 0.14286 0.14286 0.14286 0.14286	0.00270	-0.66667	0 0.33333 0.33333 0 0 0 0.33333	0.33333 0 0 0.33333 0 0 0
k = 10	0.00319	0.53636	0.06687 0.22339 0.24321 0 0.21376 0 0.25274	0.14286 0.14286 0.14286 0.14286 0.14286 0.14286 0.14286	0.00575	-0.18182	0.18182 0.27273 0.09091 0.09091 0.09091 0.27273 0 0.09091	0.09091 0.27273 0.09091 0.09091 0 0.36364
k = 50	0.01561	0.02724	0.11101 0.10108 0.16020 0.06888 0.16563 0.04690 0.34627	0.10855 0.14050 0.14894 0.05297 0.16978 0.11520 0.26410	0.00418	-0.03922	0.11765 0.09804 0.27451 0.07843 0.07843 0.03922 0.31373	0.11765 0.13725 0.17647 0.05882 0.07843 0.07843 0.33333
k = 100	0.03798	0.00560	0.11286 0.11280 0.16732 0.05262 0.16899 0.05386 0.33153	0.11161 0.13054 0.14982 0.05716 0.16783 0.07269 0.31035	0.00557	-0.01980	0.10891 0.09901 0.15842 0.03960 0.17822 0.08911 0.32673	0.12871 0.11881 0.18812 0.04951 0.15842 0.03960 0.30693
k = 500	0.18201	0.00092	0.111015 0.111365 0.166570 0.055421 0.166630 0.055414 0.333585	0.11136 0.11839 0.16141 0.05547 0.16639 0.05588 0.33111	0.03899	-0.00399	0.09980 0.14371 0.16567 0.05589 0.16567 0.05190 0.31737	0.12575 0.15369 0.13573 0.04591 0.14970 0.06387 0.32335
k = 1000	0.30498	0.00021	0.111174 0.111040 0.166691 0.055660 0.166655 0.055518 0.333259	0.11109 0.11311 0.16518 0.05562 0.16665 0.05559 0.33277	0.04474	-0.00199	0.11289 0.11089 0.17183 0.04695 0.16484 0.05594 0.33666	0.11588 0.11189 0.17083 0.04595 0.17383 0.04895 0.33167
k = 15000	4.83584	3.1e-17	0.11111 0.11111 0.16667 0.05556 0.16667 0.05556 0.33333	0.11111 0.11111 0.16667 0.05556 0.16667 0.05556 0.33333	0.58264	-0.00013	0.11312 0.11112 0.16732 0.05466 0.16299 0.05520 0.33558	0.10752 0.11179 0.16739 0.05786 0.16545 0.05446 0.33544

Tabela 2: Primerjava rezultatov za 2. igro

4 Spletni vmesnik

V datoteki *spletnivmesnik.py* sem s pomočjo *Bottle*-a naredila aplikacijo za izračun rešitve matrične igre. Ta vas najprej vpraša po vaši matrični igri in nato izračuna vrednost in optimalno strategijo igre s pomočjo linearnega programiranja in obeh zgoraj obravnavnih iterativnih metod. (Aplikacija še ni popolnoma dokončana).

5 Zaključek

Oba implementirana iteracijska algoritma konvergirata proti rešitvi igre, a z različno hitrostjo in časovno zahtevnostjo.
Drugi algoritem ima manjšo časovno zahtevnost, a rezultati kažejo na to, da je konvergenca počasnejša kot pri prvem algoritmu. Menim, da je hitrost konvergence prve metode močno odvisna tudi od začetnega približka, kar je vsekakor vredno nadaljnjega raziskovanja. Prav tako, pa sem opazila tudi, da oba algoritma ne izbereta vedno najbližje rešitve.

Literatura

[1] J. Robinson, *An Iterative Method of Solving a Game*, Annals of Mathematics, **1951** strani od 296 do 301. Dostopno na: https://www.jstor.org/stable/1969530

[2] R. J. Jirka, *An iterative method for finding a solution to a zero-sum two person rectangular game*, **1959**.