باسمه تعالی

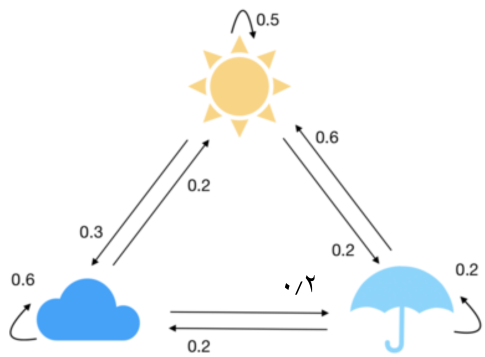
گزارش پروژه ی آمار و احتمال

دکتر کرباسی

سید علیرضا موسوی ۴۰۰۱۰۲۰۸۵

طاها معماری ۴۰۰۱۰۱۹۸۹

پرسش تئوری ۱) یکی از معروف ترین مثال ها و کاربرد های این دنباله، مدل آب و هواست. به این صورت که فرض کنید سه حالت آفتابی، ابری و بارانی وجود داشته باشد. احتمال این که روز بعد وضع هوا چگونه باشد تنها به وضع فعلی هوا بستگی دارد و ماتریس انتقال آن با گراف زیر قابل توصیف است:



که ماتریس انتقال متناظر آن برابر است با: $\begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0.2 & 0.6 & 0.2 \\ 0.6 & 0.2 & 0.2 \end{bmatrix}$

مثال دیگر، مثال مشهور راه رفتن مرد مست است. فرض کنین او در هر مرحله ۱+ خانه به جلو یا ۱- خانه به عقب برود. عدد خانه بعدی ای که روی آن قدم میگذارد تنها وابسته به شماره خانه فعلی اش است و نه به خانه هایی که قبل از آن آنجا بوده است.

پرسش تئوری ۲) می دانیم به طور کلی برای توزیع X_i داریم: $\lambda_i = \lambda_{i-1}P$

پس طبق تعریف: $\lambda_1 = \lambda P\ ,\ \lambda_2 = \lambda_1 P = \lambda P^2\ ,\ \lambda_3 = \lambda_2 P = \lambda P^3$

پس در نهایت نتیجه می شود: $\lambda_n = \lambda P^n$

پرسش تئوری ۳) طبق روابط به دست آمده در پرسش قبل:

$$P(X_n = j\ |X_{n-1} = i\ ) = p_{ij}\ , \lambda_n = \lambda P^n$$

$$p_{ij}^{(n)} = [P^n]_{ij} = P(X_n = j\ |X_0 = i)$$

می دانیم که p_{ij} احتمال رفتن از حالت i به j است و اندیس ها نشان دهنده حالت مبدا و مقصد هستند.

حال طبق رابطه λ_n برای n=2 $p_{ij}^{(2)}$ نشان دهنده این است که حالت مبدا i است و مقصد j است اما بعد از دوگام و مرحله این اتفاق افتاده است. برای اعداد بالاتر و به طور کلی n نیز پس می توان ثابت کرد که $p_{ij}^{(n)}$ احتمال رفتن از حالت i به j در n گام است.

پس از n مرحله مقدار X_n باید مقداری در محدود $1 \le j \le M$ (بازه تمام حالات ممکن) باشد. در نتیجه مجموع تمام حالات ممکن یک خواهد بود. همچنین این موضوع را با گراف های نشان دهنده تابع انتقال

هم میتوان نشان داد. از هر حالت یا میتوان دوباره در حالت فعلی ماند یا به حالات دیگر رفت و حالت دیگری وجود ندارد پس مجموع احتمالات هر سطر برابر یک است.

$$\sum_j p_{ij}^{(n)} = 1$$

می‌خواهیم بردار ویژه متناظر با $\lambda_1 = 1$ را محاسبه کنیم. معادله *(A−I)q=0*، به صورت زیر است

$$\begin{pmatrix} p_{11}-1 & \cdots & p_{1M} \\ \vdots & \ddots & \vdots \\ p_{M1} & \cdots & p_{MM}-1 \end{pmatrix} \begin{pmatrix} q_1 \\ \vdots \\ q_M \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$q_i(p_{ii}-1) + \sum_{r \neq i} q_r p_{ir} = 0 \rightarrow q_i = \sum_r q_r p_{ir} = \frac{\sum_{r \neq i} q_r p_{ir}}{p_{ii}}$$

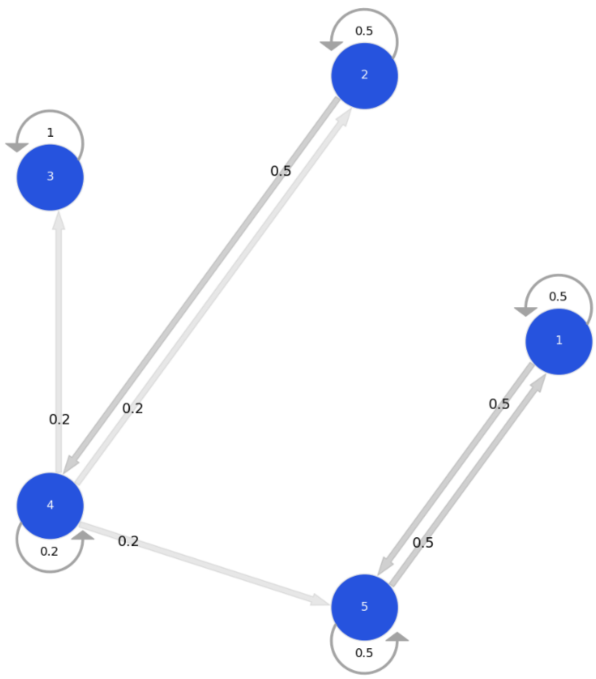
پس در نهایت به *M* معادله با *M* مجهول خواهیم رسید که حداقل یک جواب خواهد داشت.

به علاوه آن، شکل دیگری از جواب که حتما وجود دارد ، حالت ثابت $q_i = q =$ است که به معادله $\sum_r p_{ir} = 1$ می رسد.

پرسش تئوری ۴) طبق فرض که دارای حافظه به طول *k* است، پس داده های قبل از طول حافظه(از *n-k* تا صفر) تاثیری روی X_{n+1} ندارند و میتوان از آنها صرف نظر کرد و با *X* حالت k تا n ام، احتمال X_{n+1} را به صورت یک دنباله مارکوف مرتبه *k* نوشت که در آن هر پدیده به جای یک پدیده قبل به *k* پدیده قبل وابسته باشند.

پرسش تئوری ۵) یک طرف قضیه به سادگی قابل اثبات است. اگر حاصل ضرب همه احتمال ها در هم مثبت باشد، بدین معناست که تک تک آن احتمالات مثبت بوده و در گراف توصیف کننده تابع انتقال، بین هر دو s_i و s_{i+1} یک مسیر وجود دارد و درنتیجه بین مبدا و مقصد مسیر یعنی **i** و**j** نیز یک مسیر وجود دارد پس **j** برای **i** دسترس پذیر است. برای اثبات عکس قضیه از برهان خلف استفاده می کنیم و فرض میکنیم **j** برای **i** دسترس پذیر است ولی حاصل ضرب احتمال ها صفر می شود. این بدین معناست که حداقل یکی احتمال ها برابر صفر است و این بدین معناست که حداقل بین دو جفت از s_i و s_{i+1} ها مسیری وجود ندارد. پس در واقع بین **i** و **j** نیز مسیری وجود ندارد که این با فرض در تناقض است. پس دو طرف قضیه اثبات شد.

پرسش شبیه سازی ۱) در این سوال ابتدا کتابخانه های مورد نیاز را import می کنیم. کتابخانه Markowchain را از طریق فایل گیت هاب داده شده در سوال import می کنیم. سپس ماتریس انتقال را به صورت یک آرایه دو بعدی تعریف کرده و گراف متناظرش را کشیده و در فایل Q1.png ذخیره میکنیم. تصویر گراف حاصل به شکل زیر است:



با توجه به گراف بالا کلاس های مخابراتی آن بدین شرح اند:

- کلاس ۱ تشکیل شده از حالت ۱ و ۵ که یک کلاس بسته نیز هست
- کلاس ۲ شامل حالات ۲ و ۴ که یک کلاس بسته نیست(حالت ۴ به ۳ و ۵ نیز راه دارد)
- کلاس ۳ که تنها متشکل از حالت ۳ است و یک کلاس بسته است.

پرسش تئوری ۷) اگر یک حالت *i* وجود داشته باشد که در آن $P_{ii}>0$ باشد، آنگاه مجموعه تمام حالت‌های قابل دسترس از *i*، یک کلاس ***recurrent*** است.

این قضیه نشان می‌دهد که اگر حداقل یک حالت وجود داشته باشد که با احتمال مثبت به خودش برمی‌گردد، آنگاه مجموعه تمام حالت‌های قابل دسترس از این حالت، یک کلاس ***recurrent*** است.

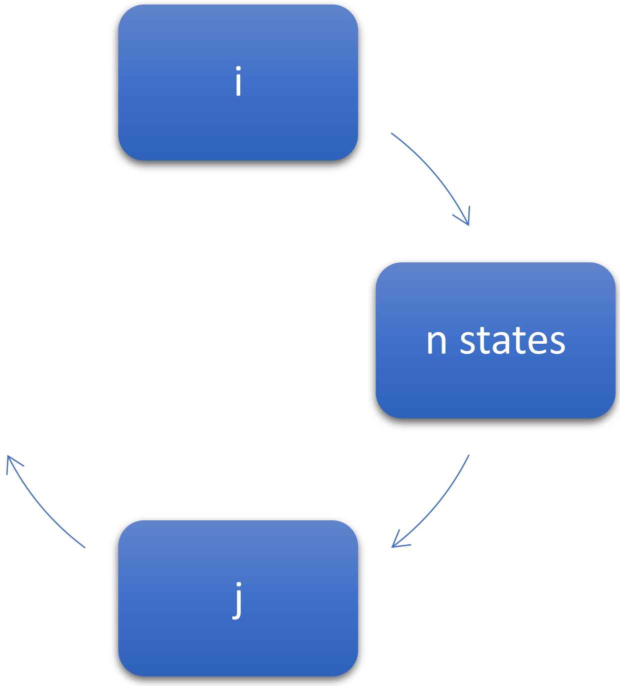
پرسش تئوری ۸) می دانیم f_{ii} برابر با احتمال آن است که از حالت i به خودش باز گردیم ، در نتیجه متمم آن احتمال آن خواهد بود که از این حالت به حالت دیگری برویم. بنابراین احتمال آن که پس از V_i دفعه از این حالت گذر کنیم برابر خواهد بود با :

$$P = f_{ii}^{V_i - 1}(1 - f_{ii})$$

که این رابطه مانند رابطه PMF توزیع هندسی است. در نتیجه خواهیم داشت:

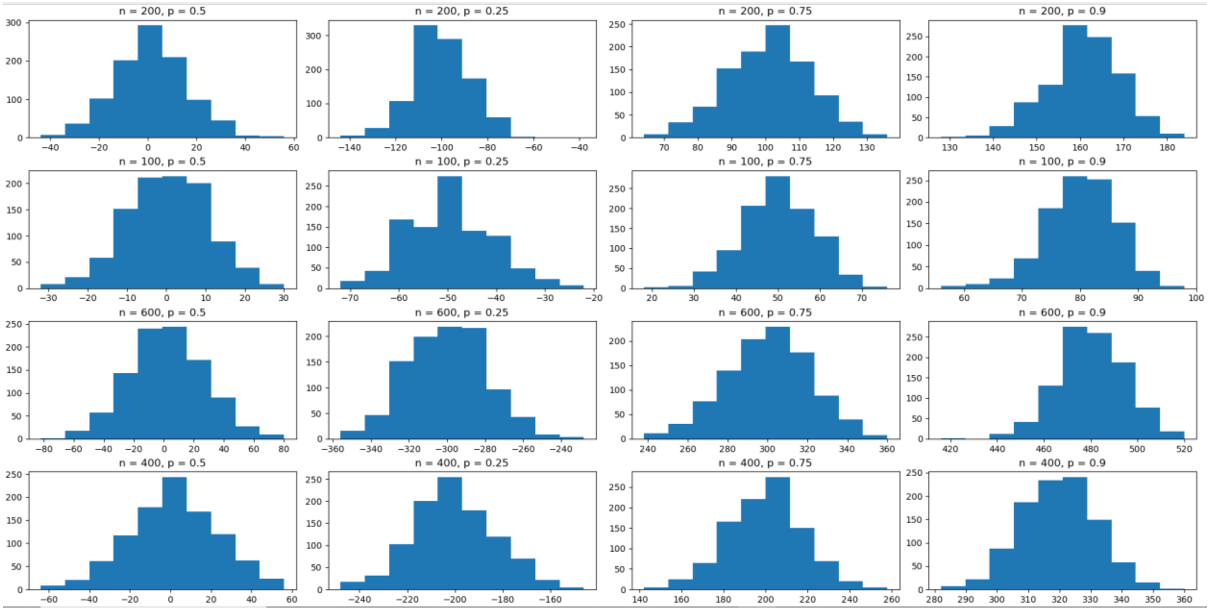
$$V_i | X_۰ = i \sim Geometric(۱ - f_{ii})$$

پرسش تئوری ۹) در ارتباط بودن i و j را با نمودار زیر نمایش میدهیم:



پس طبق این نمودار: $di = dj = m+n$

پرسش شبیه سازی ۲) در کد این سوال در هر مرحله یک عدد تصادفی تولید می شود که تعیین میکند به حالت قبلی برویم یا بعدی و این کار هر بار هزار بار تکرار می شود. همچنین علاوه بر $n=200,p=0.5$ سه تعداد و سه احتمال دیگر را هم بررسی و نمودار هیستوگرام آنها را پلات کردیم تا تاثیر تغییر متغیر ها را ببینیم:



شکل توزیع ها نزدیک به نمودار *PDF* توزیع نرمال یا *PMF* پواسون شبیه است.

مشاهده می شود که افزایش احتمال p و افزایش n هردو باعث می شوند شکل هیستوگرام به شکل دو توزیع معروف نامبرده نزدیک تر شوند. همچنین افزایش p باعث شیفت دادن نمودار به سمت راست و کاهش آن باعث شیفت به سمت چپ شده است. و افزایش p پراکندگی داده ها را افزایش داده است(این موضوع به طور واضح تری در چهار تصویر ستون اول قابل مشاهده است)

" زمان برخورد "

(پرسش تئوری ۱۵)

⊛ Law of total probability :

$$h_i^A = P[\inf\{n \geq 0 : X_n \in A\} < \infty \mid X_0 = i] =$$

$$\sum_{j \in X} P(X_1 = j \mid X_0 = i)\, h_j^A = \boxed{\sum_{j \in X} P_{ij} h_j^A \quad \forall i \in A}$$

$$\forall i \in A \Rightarrow \boxed{h_i^A = 1}$$

(پرسش تئوری ۱۶)

$$\begin{cases} K_i^A = 0 & \forall i \in A \\ K_i^A = 1 + \sum_{j \in X} P_{ij} K_j^A & \forall i \notin A \end{cases}$$

$K_i^A = 0 \quad \forall i \in A \Rightarrow$ زمان برخورد صفر ← در حالت شروع عضو A

$K_i^A = 1 + \sum_{j \in X} P_{ij} K_j^A \quad \forall i \notin A \Rightarrow$ ابتدا خارج از A ← حالت ناپویا $\xrightarrow[\text{total Probability}]{\text{Law of}}$

جمع وزن‌دار زمان برخورد حالت j روی همه‌ی حالات

(پرسش تئوری ۱۷)

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$h^{\{4\}} \overset{\text{I}}{=} \{h_1^{\{4\}}, h_2^{\{4\}}, h_3^{\{4\}}, h_4^{\{4\}}\} = \{0, h_2^{\{4\}}, h_3^{\{4\}}, 1\} \Rightarrow$$

$$h_2^{\{4\}} = \sum_{j \in X} P_{2j} h_j^{\{4\}} = h_3^{\{4\}} \times \frac{1}{2} \ \circledast$$

$$h_3^{\{4\}} = (h_2^{\{4\}} \times \tfrac{1}{2}) + (1 \times \tfrac{1}{2}) \overset{\circledast}{=} \frac{h_3}{4} + \frac{1}{2} \Rightarrow \begin{cases} h_2^{\{4\}} = \frac{1}{3} \\ h_3^{\{4\}} = \frac{2}{3} \end{cases} \overset{\text{I}}{\Rightarrow}$$

$$\boxed{h^{\{4\}} = \{0, \tfrac{1}{3}, \tfrac{2}{3}, 1\}}$$

$$\left.\begin{cases} K^{\{۱,۴\}} = \left\{K_۱^{\{۱,۴\}}, K_۲^{\{۱,۴\}}, K_۳^{\{۱,۴\}}, K_۴^{\{۱,۴\}}\right\} \\ K_۱^{\{۱,۴\}} = ۱ \\ K_۴^{\{۱,۴\}} = ۱ \end{cases}\right\} \Rightarrow$$

$$K_۲^{\{۱,۴\}} = ۱ + \sum_{j \in \mathcal{X}} P_{۲j} K_j^{\{۱,۴\}} = ۱ + \frac{۱}{۲} + \frac{K_۳^{\{۱,۴\}}}{۲} \quad (*)$$

$$K_۳^{\{۱,۴\}} = ۱ + \sum_{j \in \mathcal{X}} P_{۳j} K_j^{\{۱,۴\}} = ۱ + \frac{K_۲^{\{۱,۴\}}}{۲} + \frac{۱}{۲} \overset{(*)}{=} ۲{,}۲۵ + \frac{K_۳^{\{۱,۴\}}}{۴}$$

$$\Rightarrow \frac{۳}{۴} K_۳^{\{۱,۴\}} = ۲{,}۲۵ \Rightarrow K_۳^{\{۱,۴\}} = ۳ \,,\; K_۲^{\{۱,۴\}} = ۳ \Rightarrow$$

$$\boxed{K^{\{۱,۴\}} = \{۱, ۳, ۳, ۱\}}$$

" نظریه ی بازی ها "

No.

Date.

پرسش تئوری ۱۹)

		Mat: Confess	Mat: Deny
Pat	Confess	۵ , ۵	۰ , ۲۰
	Deny	۲۰ , ۰	۱ , ۱

Pat's POV: → Mat has Confessed { I confess → I get ۵ y ; I deny → I get ۲۰ y } Confess (۵<۲۰)

→ Mat has denied { I confess → I get ۰ y ; I deny → I get ۱ y } Confess (۰<۱)

درنتیجه بدون توجه به اینکه Mat چه کرده است، به نفع Pat است اعتراف کند ⟸ (اگر جای او بودیم اعتراف می‌کردیم)

چون استراتژی ها قرینه اند به طریق مشابه اثبات می‌شود که Mat نیز باید فارغ از هر کاری که Pat کرده است اعتراف کند.

درنتیجه Nash Equilibrium در اعتراف کردن هردوی آن ها است.

پرسش تئوری ۲۰) الف)

		Z: Dive left	Z: Dive Right
P	Shoot Right	۰/۸ , ۰/۲	۰/۱۵ , ۰/۸۵
	Shoot left	۰/۲ , ۰/۸	۰/۹۵ , ۰/۰۵

وقتی گفته می‌شود که احتمال گل کردن ۰/۸ است به تبع یعنی احتمال گل نشدن و گرفتن آن توسط گلر ۰/۲ است...

با بررسی مسئله با دید مسئله قبلی متوجه می‌شویم که در اینجا به یک خانه ی واحد نمی‌رسیم و با یک مسئله Mixed Strategy طرف هستیم.

No.

Date.

خانه‌های جدول را جابجا می‌کنیم تا کمی مرتب‌تر شوند (تغییری در دنیا ایجاد نمی‌کنیم) :

		q Dive left	$1-q$ Dive Right
p	Shoot left	0.2 , 0.8	0.95 , 0.05
$1-p$	Shoot right	0.8 , 0.2	0.15 , 0.85

Putin's Payoff : $pq(0.2) + p(1-q)(0.95) + (1-p)(q)(0.8) + (1-p)(1-q)(0.15) =$

$$0.2pq + 0.95p - 0.95pq + 0.8q - 0.8pq + 0.15 - 0.15p - 0.15q + 0.15pq =$$

$$-1.4pq + 0.8p - 0.65q + 0.15 \xrightarrow{\frac{\partial}{\partial p} = 0}$$

$$-1.4q + 0.8 = 0 \Rightarrow \boxed{q = \frac{8}{14} = \frac{4}{7} \simeq 0.571}$$

Zelensky's Payoff : $pq(0.8) + p(1-q)(0.05) + (1-p)(q)(0.2) + (1-p)(1-q)(0.85) =$

$$0.8pq + 0.05p - 0.05pq + 0.2q - 0.2pq + 0.85 - 0.85p - 0.85q + 0.85pq =$$

$$1.4pq - 0.65q - 0.8p + 0.85 \xrightarrow{\frac{\partial}{\partial q} = 0}$$

$$1.4p - 0.65 = 0 \Rightarrow \boxed{p = \frac{65}{140} = \frac{13}{28} \simeq 0.464}$$

یعنی اگر Putin بداند Zelensky با احتمال $q \simeq 0.571$ به سمت چپ می‌پرد فرقی ندارد که چپ یا راست
شوت کند و همچنین اگر Zelensky بداند Putin با احتمال $p \simeq 0.464$ به سمت چپ shoot می‌کند
برایش فرقی ندارد که به سمت چپ بپرد یا راست. ← Nash Equilibrium
in mixed strategies

No.

Date.

	DL	DR
SR	۰٫۸ \ ۰٫۲	۰٫۱۵ \ ۰٫۸۵
SM	۰٫۵ \ ۰٫۵	۰٫۵ \ ۰٫۵
SL	۰٫۲ \ ۰٫۸	۰٫۹۵ \ ۰٫۰۵

ب) در این مسئله با یک Table ۳×۲ مواجه هستیم که می‌توانیم برای ساده‌کردن آن چک کنیم که آیا weak/strict dominance داریم یا نه

اگر putin با احتمال $\frac{۱}{۲}$ به سمت راست و با احتمال $\frac{۱}{۲}$ به سمت پایین شوت کند:

اگر Zelensky به راست پریده باشد: Putin's Payoff $= \frac{۱}{۲}(۰٫۱۵) + \frac{۱}{۲}(۰٫۹۵) = ۰٫۵۵ > \overset{\text{Middle Payoff}}{۰٫۵}$

اگر Zelensky به چپ پریده باشد: // $= \frac{۱}{۲}(۰٫۸) + \frac{۱}{۲}(۰٫۲) = ۰٫۵ = \overset{\text{Middle Payoff}}{۰٫۵}$

مشاهده می‌کنیم در یک حالت Payoff بیشتر خواهیم داشت و در یک حالت Payoff برابر در نتیجه با اینکه انتخاب صرفاً چپ و راست strictly dominant نیست ولی می‌توان به قرینه‌ی weakly dominant بودن از ردیف وسط صرف نظر کرد:

	DL	DR
SR	۰٫۸	۰٫۱۵
SL	۰٫۲	۰٫۹۵

حال مسئله همان مسئله‌ی قبل است در نتیجه:

با احتمال $۰٫۵۷۱ \approx \frac{۴}{۷} = q$ پرش به سمت چپ جهت شوت کردن برای پوتین فرقی ندارد

به احتمال $۰٫۴۶۴ \approx \frac{۱۳}{۲۸} = p$ شوت به سمت چپ جهت پرش برای زلنسکی فرقی ندارد

و شوت کردن به مرکز کار درستی نخواهد بود و Putin با بررسی این جدول می‌داند که نباید به مرکز شوت کند.

No.

Date.

ج)

		Putin: p Left	Putin: $1-p$ Right
Zelensky	q left	x , 2	0 , 0
	$1-q$ right	0 , 0	2 , 2

Putin's Payoff : $pq(2)+(1-p)q(0)+(p)(1-q)(0)+(1-p)(1-q)(2)=$

$$2pq+2+2pq-2p-2q \xrightarrow{\frac{\partial}{\partial p}=0}$$

$$4q-2=0 \Rightarrow \boxed{q=\frac{1}{2}}$$

Zelensky's Payoff : $pq(x)+(1-p)q(0)+p(1-q)(0)+(1-p)(1-q)(2)=$

$$pqx+2-2p-2q+2pq \xrightarrow{\frac{\partial}{\partial q}=0}$$

$$px-2+2p=0 \Rightarrow \boxed{p=\frac{2}{2+x}}$$

Nash Equilibrium :

اگر Zelensky با احتمال $q=\frac{1}{2}$ به سمت چپ شوت کند برای Putin فرقی نمی‌کند کدام سمت شیرجه کند

اگر Putin با احتمال $p=\frac{2}{2+x}$ به سمت چپ شوت کند برای Zelensky فرقی نمی‌کند کدام سمت بپرد

* q نسبت به تغییر x بی‌تفاوت است اما p با کم شدن x به 1 نزدیک شده و با افزایش x به صفر میل می‌کند

پرسش تشریحی ۲۱)

الف) Game Matrix که Normal form یا Strategic form نیز نامیده می‌شود به ماتریس Payoff های player ها براساس انتخاب‌های مختلف آن‌ها گفته می‌شود. در اینجا تنها Payoff های player ردیف را داریم؛ می‌توان ماتریس مشابهی برای Player دیگر تدارک دید یا اعداد آن ماتریس دیگر را کنار اعداد این ماتریس اضافه کرد. در اینجا ماتریس A به نحوی است که به این Game ها Zero-sum game می‌گویند که total payoff بازیکن‌ها در انتخاب‌های مختلف صفر است. این گیم‌ها خواصی دارند که می‌تواند محاسبات تکراری را کاهش دهند.

No.

Date.

ب) $\sigma_r = (1 \quad 0 \quad 0)^T \longrightarrow$ Row player's strategy $\rightarrow$ plays only rock

$\sigma_C = (0 \quad 1 \quad 0) \longrightarrow$ Column player's strategy $\rightarrow$ plays only paper

$$A = \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix} \xrightarrow[\text{Column player's payoff}]{\text{zero sum game}} B = \begin{pmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix} \Rightarrow$$

$$\left.\begin{aligned} U_r(\sigma_r, \sigma_C) &= \sum_{i=1}^{3}\sum_{j=1}^{3} A_{ij}\,\sigma_{r_i}\,\sigma_{c_j} \\ U_C(\sigma_r, \sigma_C) &= \sum_{i=1}^{3}\sum_{j=1}^{3} B_{ij}\,\sigma_{r_i}\,\sigma_{c_j} \end{aligned}\right\} \Rightarrow$$

$U_r(\sigma_r, \sigma_c)$ (فرهان)

$$= 1\big((0\times0) + (1\times(-1)) + (0\times1)\big) + 0\big((0\times1) + (1\times0) + (0\times(-1))\big) + 0\big((0\times(-1)) + (1\times1) + (0\times0)\big) = \boxed{-1}$$

$U_c(\sigma_r, \sigma_c)$ (پارسا)

$$= 1\big((0\times0) + (1\times1) + (0\times(-1))\big) + 0\big((0\times(-1)) + (1\times0) + (0\times1)\big) + 0\big((0\times1) + (1\times(-1)) + (0\times0)\big) = \boxed{+1}$$

نتایج utility ها معقول نیز هستند، فرهان با همواره سنگ بازی کردن در حالی که پارسا همواره کاغذ بازی می‌کند، نتیجه‌ای نامطلوب دریافت می‌کند (-1) و برعکس آن برای پارسا است که همواره می‌برد و نتیجه مطلوبش است. (+1)

ج)

$$\sigma_r = (0.25 \;\; 0.35 \;\; 0.4)^T$$

$$\sigma_c = (0.25 \;\; 0.35 \;\; 0.4)$$

$$A = \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix}$$

$$\Rightarrow U_r(\sigma_r, \sigma_c) = 0.25\big((0.25\times 0) + (0.35\times(-1)) + (0.4\times 1)\big) + 0.35\big((0.25\times 1) + (0.35\times 0) + (0.4\times(-1))\big) + 0.4\big((0.25\times(-1)) + (0.35\times 1) + (0.4\times 0)\big) =$$

$$0.25(0.05) + 0.35(-0.15) + 0.4(0.1) = \boxed{0}$$

$$U_c(\sigma_r, \sigma_c) = 0.25\big((0.25\times 0) + (0.35\times 1) + (0.4\times(-1))\big) + 0.35\big((0.25\times(-1)) + (0.35\times 0) + (0.4\times 1)\big) + 0.4\big((0.25\times 1) + (0.35\times(-1)) + (0.4\times 0)\big) =$$

$$0.25(-0.05) + 0.35(0.15) + 0.4(-0.1) = \boxed{0}$$

(از آنجایی که هر دو استراتژی مشابهی را پیش گرفتند، انتظار صفر شدن utility هایشان را داشتیم.)

نکته جالب توجه در هر دو بخش ب و ج این است که از آنجایی که با یک Zero Sum Game طرف هستیم می‌توانستیم تنها یکی از utility ها را محاسبه کنیم و آن را قرینه کنیم تا utility فرد دیگر بدست آید، که الان با محاسبه هر دو نیز این قضیه را مشاهده کردیم.

Column Player

د)

$$\text{Row Player}\quad \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix} \begin{array}{l} \xrightarrow{\min} -1 \\ \xrightarrow{\min} -1 \\ \xrightarrow{\min} -1 \end{array} \Bigg\} \overset{\text{Maxmin}}{\Longrightarrow} -1$$

$$\begin{array}{ccc} \max\downarrow & \max\downarrow & \max\downarrow \\ 1 & 1 & 1 \end{array}$$

$$\underbrace{1 \quad 1 \quad 1}_{} \Big\downarrow \text{minmax} \quad 1$$

با برابر نبودن Maxmin و minMax در می‌یابیم که در این گیم Pure Strategy وجود ندارد و با یک مسئله‌ی Mixed Strategy روبرو هستیم. پس به احتمال انتخاب هر کدام از option ها یک عدد نسبت می‌دهیم:

$$\begin{array}{cc} & \begin{array}{ccc} \quad y_1 & \quad y_2 & \quad y_3 \end{array} \\ & CP \\ RP \begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array} & \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix} \end{array}$$

اگر CP سنگ را انتخاب کند: RP expected Payoff $= x_2 - x_3$

اگر CP کاغذ را انتخاب کند: // $= x_3 - x_1$

اگر CP قیچی را انتخاب کند: // $= x_1 - x_2$

CP باید استراتژی پیدا کند که با $\min(x_2-x_3, x_3-x_1, x_1-x_2)$ سر و کار داشته باشد.

در نتیجه RP باید استراتژی‌ای پیدا کند که min مقدار expected Payoff های او تا حد ممکن بزرگ شود و کمترین ضرر را کند.

$$v = \min(x_2-x_3, x_3-x_1, x_1-x_2) \Rightarrow$$

$$\begin{cases} v \le x_2 - x_3 \\ v \le x_3 - x_1 \\ v \le x_1 - x_2 \end{cases} \overset{\oplus}{\Longrightarrow} 3v \le 0 \Rightarrow \boxed{v \le 0 \xrightarrow{\mathrm{I}} \text{بزرگترین مقدار } v \text{، } v=0 \text{ است.}}$$

No.

Date.

حال از زاویه دیگر به مسئله نگاه می‌کنیم:

اگر RP سنگ را انتخاب کند: $RP\ expected\ payoff = y_3 - y_2$

اگر RP کاغذ را انتخاب کند: $\quad " \quad = y_1 - y_3$

اگر RP قیچی را انتخاب کند: $\quad " \quad = y_2 - y_1$

RP باید استراتژی را انتخاب کند که از $\max(y_3 - y_2, y_1 - y_3, y_2 - y_1)$ بهره ببرد.

همچنین CP باید استراتژی انتخاب کند که احتمال رخداد Max مقدارهای فوق را حداقل کند:

$W = \max(y_3 - y_2, y_1 - y_3, y_2 - y_1) \Rightarrow$

$$\left.\begin{array}{l} W \geq y_3 - y_2 \\ W \geq y_1 - y_3 \\ W \geq y_2 - y_1 \end{array}\right\} \xrightarrow{\oplus} 3W \geq 0 \Rightarrow W \geq 0 \Rightarrow$$

II کوچکترین مقدار W، $W = 0$ است

پس براساس I و II بهترین استراتژی با توجه به زاویه دید هر دو فرد $\begin{cases} W = 0 \\ V = 0 \end{cases}$ است.

می‌دانیم $\rightarrow x_1 + x_2 + x_3 = 1$

$\qquad\quad \rightarrow y_1 + y_2 + y_3 = 1$

همچنین براساس I و II $\Rightarrow \begin{cases} 0 = y_3 - y_2 \\ 0 = y_1 - y_3 \\ 0 = y_2 - y_1 \end{cases} \Rightarrow y_1 = y_2 = y_3$

$\begin{cases} 0 = x_2 - x_3 \\ 0 = x_3 - x_1 \\ 0 = x_1 - x_2 \end{cases} \Rightarrow x_1 = x_2 = x_3$

$\Rightarrow$ "Nash Equilibrium"

$$\boxed{\begin{array}{l} x_1 = x_2 = x_3 = \frac{1}{3} \\ y_1 = y_2 = y_3 = \frac{1}{3} \end{array}}$$

No.

Date.

(پرسش تئوری ۲۲)

$U_1(y_1, y_2, y_3) = y_1 + y_1 y_2 - y_1^2$

$U_2(y_1, y_2, y_3) = y_2 + y_1 y_2 - y_2^2$

$U_3(y_1, y_2, y_3) = (10 - y_1 - y_2 - y_3) \cdot y_3$

با توجه به اینکه هر فرد برای بهبود utility function خودش تنها می‌تواند استراتژی خودش را بهینه کند و کنترلی روی استراتژی بقیه ندارد، با مشتق گیری نسبت به استراتژی هر شخص و برابر صفر قرار دادن آن می‌توان نقاط max را پیدا کرد (پیدا کردن نقاط بحرانی)

$$\left.\begin{aligned} \frac{\partial U_1}{\partial y_1} &= 1 + y_2 - 2y_1 = 0 \Rightarrow y_2 = 2y_1 - 1 \\ \frac{\partial U_2}{\partial y_2} &= 1 + y_1 - 2y_2 = 0 \Rightarrow y_1 = 2y_2 - 1 \end{aligned}\right\} \Rightarrow y_2 = 2(2y_2 - 1) - 1 = 4y_2 - 3 \Rightarrow \boxed{y_2 = 1,\ y_1 = 1} \ (*)$$

$$\frac{\partial U_3}{\partial y_3} = 10 - y_1 - y_2 - 2y_3 = 0 \overset{(*)}{\Rightarrow} 8 = 2y_3 \Rightarrow \boxed{y_3 = 4}$$

$$\Rightarrow \boxed{(y_1^*, y_2^*, y_3^*) = (1, 1, 4) \rightarrow \text{Nash Equilibrium}}$$

“Nashpy”

حال میخواهیم با استفاده از این کتابخانه نقطه(نقاط) تعادل نش بازی سنگ کاغذ قیچی را که در قسمت اول دیدید را پیدا کنیم. یک بار با استفاده از

game.support_enumeration

و بارهای دیگر با استفاده از

game.vertex_enumeration , Lemke Howson

کد را پیاده سازی کنید. و در پایان با جستجو در اینترنت بگویید که تفاوت عملکرد این سه در چیست و خروجی هر سه را با هم مقایسه کنید. آیا خروجی ها

pure strategy

است یا

mixed strategy?

In [13]:
```
#==========================================================================================================#
```

بررسی Nash Equilibrium در سنگ کاغذ قیچی

In [54]:
```
RowPlayer =np.array([[0, -1, 1], [1, 0, -1], [-1, 0, 1]])
ColumnPlayer =np.array([[0, 1, -1], [-1, 0, 1], [1, -1, 0]])


#support_enumeration:
game1 = nash.Game(RowPlayer,ColumnPlayer)
eq1 = game1.support_enumeration()
print("game.support_enumeration Method :")
print(list(eq1))
print("")


#vertex_enumeration:
game = nash.Game(RowPlayer, ColumnPlayer)
eq2 = game.vertex_enumeration()
print("game.vertex_enumeration Mehtod :")
print(list(eq2))
print("")


#lemke_howson:
game3 = nash.Game(RowPlayer,ColumnPlayer)
eq3 = game3.lemke_howson(initial_dropped_label=0)
print("game.lemke_howson Mehtod :")
print(list(eq3))
```

```
game.support_enumeration Method :
[(array([0.33333333, 0.33333333, 0.33333333]), array([0.33333333, 0.33333333, 0.33333333]))]

game.vertex_enumeration Mehtod :
[(array([0.33333333, 0.33333333, 0.33333333]), array([0.33333333, 0.33333333, 0.33333333]))]

game.lemke_howson Mehtod :
[array([0.33333333, 0.33333333, 0.33333333]), array([0.33333333, 0.33333333, 0.33333333])]
```

In [14]:
```
#==========================================================================================================#
```

همانطور که در محسبات دستی به استراتژی 1/3 , 1/3 , 1/3 رسیدیم ، محاسبات سه متود بالا نیز پاسخمان را تایید کرد. با تشکیل گیم ها و دادن ماتریس payoff ها کافی است از اتریبیوت مربوطه ی هر متود استفاده کنیم تا Nash Equilibria ی مربوطه به دیتا را پیدا کند.

در روش Support Enumeration با exhausting search تمامی کامبینیشن های مختلف از استراتژی ها بررسی می شوند تا موردی پیدا شود که شروط nash equilibrium را دارا باشد. این روش به دلیل اینکه همه ی حالات را بررسی می کند از دو روش دیگر بهینگی کمتری دارد.

در روش Vertex Enumeration با بررسی ساختار گیم ،کاندیدا هایی را پیدا می کند که می توانندNash Equilibrium را برقرار کنند و سپس بررسی نهایی را بین آن ها انجام می دهد به همین جهت از روش قبلی بهینه تر است.

در روش Lemke-Howson با یک الگوریتم iterative و تکرار شونده در ارتباط هستیم که ابتدا یک سولوشن ساده را در نظر می گیرد و مدام چک می کند که آیا به Nash Equilibrium رسیده است یا خیر و آنقدر پاسخ را با توجه به خروجی قبلی اصلاح می کند تا به پاسخ برسد ولی فقط می تواند در لحظه یک equilibrium را بدست آورد.

در خصوص pure یا mixed بودن استرتژی ، مسلما چون با احتمالات مختلف از آپشن های مختلف بهره می بریم mixed strategy داریم و پاسخ یکتایی وجود ندارد که بخواهد pure strategy را تشکیل دهد.

ابتدا درباره مساله

matching pennies

تحقیق کنید. اگر بخواهیم چند بار انتخاب استراتژی انجام بگیرد، کدی بنویسید که تکرار عملیات صورت بگیرد تعداد تکرار را برابر 2 در نظر بگیرید. ماتریس بازی را نمایش دهید.

In [58]:
```
import nashpy.repeated_games
MP_GameMatrix = np.array([[1, -1], [-1, 1]])
MP_Game = nash.Game(MP_GameMatrix)
eq_MP = nash.repeated_games.obtain_repeated_game(game=MP_Game, repetitions=2)
eq_MP
```

Out[58]:
```
Zero sum game with payoff matrices:

Row player:
[[ 2.  2.  2. ...  0. -2. -2.]
 [ 2.  2.  2. ...  0. -2. -2.]
 [ 2.  2.  2. ...  0. -2. -2.]
 ...
 [ 0.  0.  0. ...  2.  0.  2.]
 [-2. -2. -2. ...  0.  2.  0.]
 [-2. -2. -2. ...  2.  0.  2.]]

Column player:
[[-2. -2. -2. ...  0.  2.  2.]
 [-2. -2. -2. ...  0.  2.  2.]
 [-2. -2. -2. ...  0.  2.  2.]
 ...
 [ 0.  0.  0. ... -2.  0. -2.]
 [ 2.  2.  2. ...  0. -2.  0.]
 [ 2.  2.  2. ... -2.  0. -2.]]
```

مسئله ی matching pennies یک مسئله ی معروف از Zero-Sum game ها است که payoff مثبت یا برد یک نفر به منزله ی باخت و payoff منفی فرد دیگر است. در این مسئله یک سکه دست ما و یک سکه دست حریف است. اگر هر دو سکه را به یک رو، بر روی میز قرار دهیم، 1$ از حریف می گیریم ولی اگر رو ها متفاوت باشند 1$ به حریف می دهیم.

در قسمت سوال این مسئله تعبیر های متفاوتی نقل می شود. من این نقل را در نظر می گیرم که می گویند فرض می کنیم طرف مقابل ذهن خوان است و می داند ما قرار است چه رویی از سکه را روی میز بگذاریم، بنابراین ما همواره محکوم به باخت هستیم چرا که او روی دیگر را می آورد و 1$ می گیرد. استراتژی ای که می توان جلوی او بازی کرد این است که سکه را به هوا پرت کنیم تا نه خودمان بدانیم چه رویی می آید نه حریف بتواند متوجه شود که چه رویی بازی کرده ایم. پس احتمال شیر یا خط ما 0.5 می شود و همچنین حریف نیز مجبور است با احتمال 0.5 شیر یا خط بیاورد چون از انتخاب ما باخبر نیست.

برای بررسی این امر، از اتریبیوت repeated_games. Obtain_repeated_game استفاده می کنیم و تعداد تکرار را روی ۲ ست کرده و بازی را به آن می دهیم و نتیجه را مشاهده می کنیم.

```
Row player:
[[ 2.  2.  2. ...  0. -2. -2.]
 [ 2.  2.  2. ...  0. -2. -2.]
 [ 2.  2.  2. ...  0. -2. -2.]
 ...
 [ 0.  0.  0. ...  2.  0.  2.]
 [-2. -2. -2. ...  0.  2.  0.]
 [-2. -2. -2. ...  2.  0.  2.]]

Column player:
[[-2. -2. -2. ...  0.  2.  2.]
 [-2. -2. -2. ...  0.  2.  2.]
 [-2. -2. -2. ...  0.  2.  2.]
 ...
 [ 0.  0.  0. ... -2.  0. -2.]
 [ 2.  2.  2. ...  0. -2.  0.]
 [ 2.  2.  2. ... -2.  0. -2.]]
```

در پاسخ بدست آمده هم Zero-Sum بودن بازی مشهود است و هم استراتژی 50% 50% بازی کردن.

حال اگر بخواهیم تعداد تکرار ها را بیشتر کنیم چه؟ برای تکرار های بیشتر این روش محاسبات زیادی میخواهد بجایش از روش های لرنینگ استفاده میکنیم.

در زیر الگوریتم فیکتس

game.fictitious_play

یکی از روش های لرن کردن میباشد. البته همیشه این الگوریتم همگرا نمیشود

```
In [59]: A = np.array([[0, 1, 0], [0, 0, 1], [1, 0, 0]])
         B = np.array([[0, 0, 1], [1, 0, 0], [0, 1, 0]])
         game = nash.Game(A, B)
         iterations = 10000
         np.random.seed(0)
         play_counts = tuple(game.fictitious_play(iterations=iterations))
         print(play_counts[-1])

[array([5464., 1436., 3100.]), array([2111., 4550., 3339.])]
```

در مساله فوق اعداد آرایه

play_counts

باید بر تعداد

iteration

را نسبت به row player تقسیم کنید تا احتمال هر استراتژی بدست بیاید و خروجی را نشان دهید. و در نهایت احتمال هر استراتژی را برای بازیکن اول

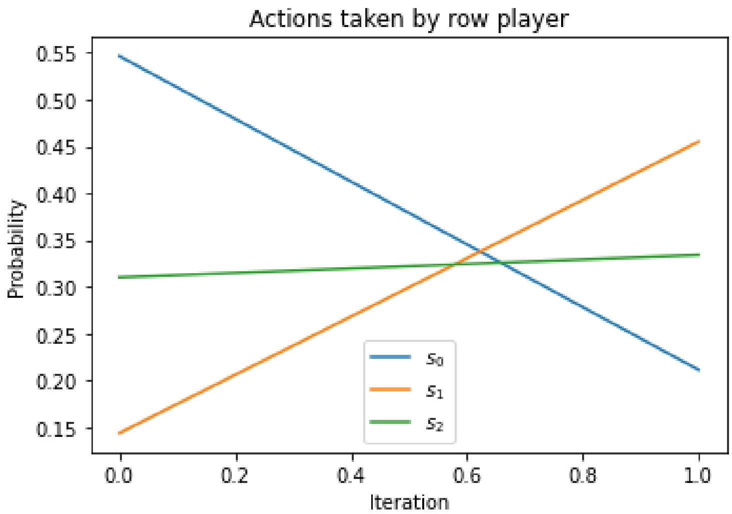
.iterations

پلات کنید آیا الگوریتمها همگرا میشوند؟ توضیح دهید...

... را بصورت مناسب تعریف کنید ptobabilities شما صرفا باید به روش گفته شده

```python
In [60]: import matplotlib.pyplot as plt
         plt.figure()

         probabilities = [n/iterations for n in play_counts[-1]]

         for number, strategy in enumerate(zip(*probabilities)):
             plt.plot(strategy, label=f"$s_{number}$")



         plt.xlabel("Iteration")
         plt.ylabel("Probability")
         plt.title("Actions taken by row player")
         plt.legend()
```

```
Out[60]: <matplotlib.legend.Legend at 0x2381f5ea520>
```



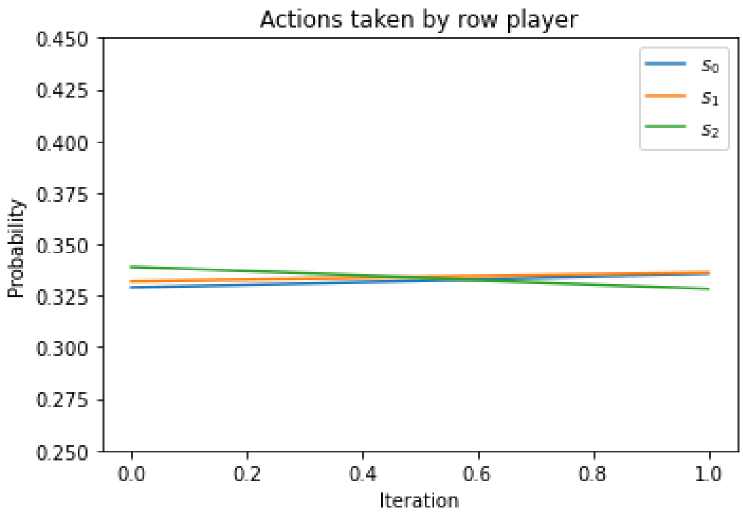
برای تقسیم کرده پاسخ به تعداد بار iteration کافی است یک آرایه ی دیگر درست کرده و داخل آن یک لوپ بزنیم که هر المان را بر تعداد بار iteration تقسیم کند و احتمالات بدست آیند. مشاهده می کنیم که هر سه، تغییراتی شدیدی داشته اند و به اصطلاح همگرایی رخ نداده است.

این بار برای ماتریس های

A , B

همان الگوریتم قبلی را تکرار کنید و نشان دهید که این بار برای بازیکن دوم الگوریتم ها همگرا میشوند

```
In [76]: A = np.array([[1 / 2, 1, 0], [0, 1 / 2, 1], [1, 0, 1 / 2]])
         B = np.array([[1 / 2, 0, 1], [1, 1 / 2, 0], [0, 1, 1 / 2]])
         game = nash.Game(A, B)
         np.random.seed(0)

         iterations = 10000
         play_counts2 = tuple(game.fictitious_play(iterations=iterations))

         plt.figure()
         probabilities2 = [n/iterations for n in play_counts2[-1]]
         print(probabilities2)

         for number, strategy in enumerate(zip(*probabilities2)):
             plt.plot(strategy, label=f"$s_{number}$")



         plt.xlabel("Iteration")
         plt.ylabel("Probability")
         plt.title("Actions taken by row player")
         plt.ylim(0.25,0.45)
         plt.legend()

         [array([0.329, 0.332, 0.339]), array([0.3356, 0.3361, 0.3283])]

Out[76]: <matplotlib.legend.Legend at 0x23820785820>
```



مشاهده می کنیم که در این گیم پاسخ ها تا حد خوبی تغییرات نداشته اند (نسبت به اینکه ۱۰۰۰۰ بار ایتریشن رخ داده تغییرات واقعا کم است) و می توان گفت که در حدود 0.333 همگرا شده اند.

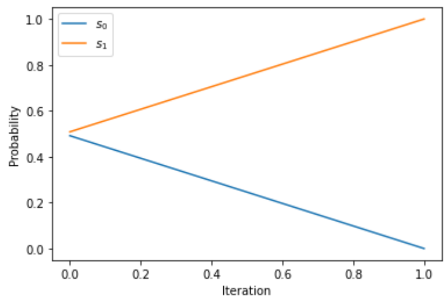
حال میخواهیم ببینیم یک بازیکن در گذر زمان چه احتمالی از استراتژی ها را بازی میکند. با استفاده از الگوریتم

replicator dynamics

این کار را انجام دهید. (البته پیش از این کار همانند قسمت قبل احتمال هر استراتژی را برای بازیکن اول و دوم پلات بگیرید) ,A برای ماتریس های هزینه

```
import nashpy as nash
import numpy as np

A = np.array([[3, 2], [4, 2]])
B = np.array([[1, 3], [2, 4]])
game = nash.Game(A,B)

np.random.seed(0)

iterations = 10000
play_counts3 = tuple(game.fictitious_play(iterations=iterations))

plt.figure()
probabilities3 = [n/iterations for n in play_counts3[-1]]

for number, strategy in enumerate(zip(*probabilities3)):
    plt.plot(strategy, label=f"$s_{number}$")



plt.xlabel("Iteration")
plt.ylabel("Probability")
plt.legend()



RD_result = game.replicator_dynamics()
print(RD_result)
```

```
[[0.5        0.5       ]
 [0.49875032 0.50124968]
 [0.49750377 0.50249623]
 ...
 [0.10199196 0.89800804]
 [0.10189853 0.89810147]
 [0.10180527 0.89819473]]
```

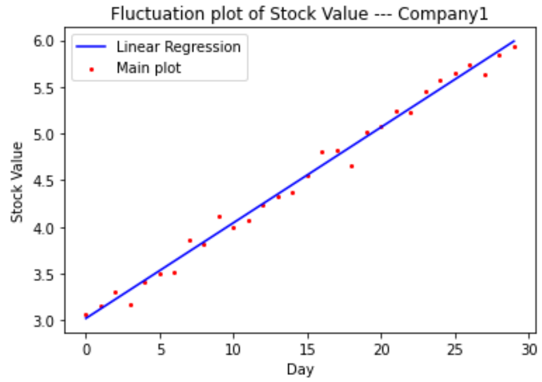


قسمت اول که مشابه کد های قبلی است. برای استفاده از replicator dynamics کافی است از اتریبیوت آن بر روی گیمی که تعریف کردیم استفاده کنیم و خروجی آن را پرینت کنیم. مشاهده می کنیم که دیتای خروجی آن که احتمال انتخاب استراتژی های مختلف است همچون خطوط رسم شده در حال تغییر با مقادیر مشابهی است.

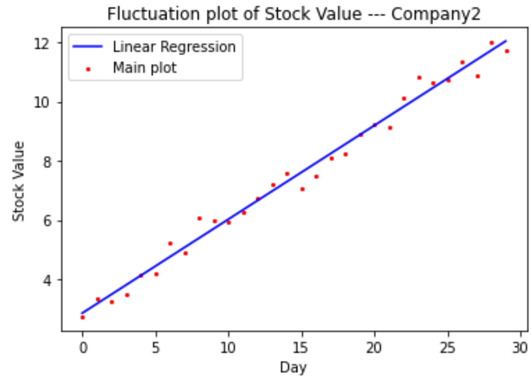
شبیه سازی 5

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd



# Linear Regression Coefficients Finder

def LRCF(PandasDataFrame,MemoryMatrix):
    for i in range(0,6):

        index = np.arange(0,30)
        CompanyData = np.array(PandasDataFrame[i])

        index_mean = np.mean(index)
        CompanyData_mean = np.mean(CompanyData)


        CompanyData = CompanyData - CompanyData_mean
        index = index - index_mean

        squaredX = index**2
        XY = index * CompanyData

        beta_1 = np.sum(XY) / np.sum(squaredX)
        beta_0 = CompanyData_mean - beta_1*index_mean

        MemoryMatrix[i,0] = beta_0
        MemoryMatrix[i,1] = beta_1

        print(f'Company{i+1} :')
        print(f"β0 = {beta_0}  ---->")
        print(f"β1 = {beta_1}  ---->")
        print(f"y = {round(beta_0,3)} + {round(beta_1,3)}x")

        y_linear = []
        for x in range (0,30):
            y_linear.append(beta_0 + beta_1*x)

        plt.scatter(np.arange(0,30),PandasDataFrame[i],s=4,color='r')
        plt.plot(y_linear,color='b')
        plt.ylabel("Stock Value")
        plt.xlabel("Day")
        plt.title(f"Fluctuation plot of Stock Value --- Company{i+1}")
        plt.legend(["Linear Regression","Main plot"])
        plt.show()



#change it to whatever path the data file exists
dataFile = pd.read_csv("I:\\SUT\\Term4\\amar\\PROJECT\\Data.csv", usecols=[0,1,2,3,4,5,6], header=None)

shape = (6, 2)
beta_log = np.empty(shape)

LRCF(dataFile,beta_log)


print('beta_log array :')
print(beta_log)
```

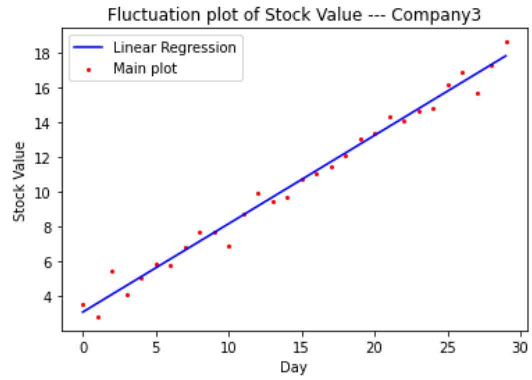
```
Company1 :
β0 = 3.0195452805441025  ---->
β1 = 0.10241457418458706  ---->
y = 3.02 + 0.102x
```



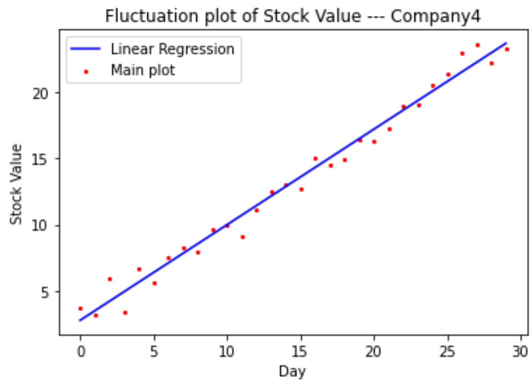

```
Company2 :
β0 = 2.8511800131711977  ---->
β1 = 0.31698482043161647  ---->
y = 2.851 + 0.317x
```



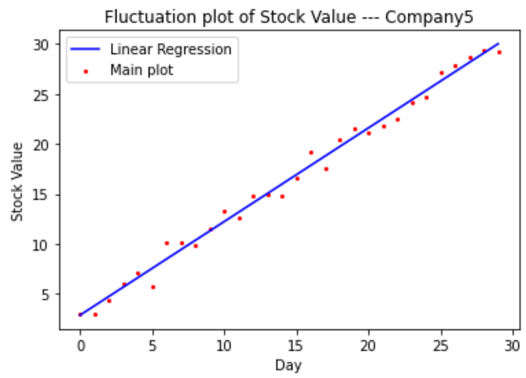

```
Company3 :
β0 = 3.0742853767088834  ---->
β1 = 0.5095406765807879  ---->
y = 3.074 + 0.51x
```



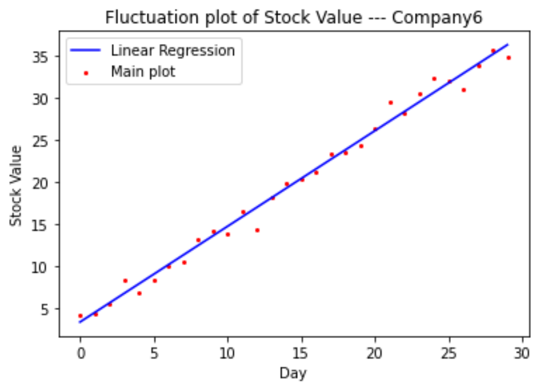

```
Company4 :
β0 = 2.7826912745549865  ---->
β1 = 0.7193582859238877  ---->
y = 2.783 + 0.719x
```




```
Company5 :
β0 = 2.8428764079413735  ---->
β1 = 0.9374915774304288  ---->
y = 2.843 + 0.937x
```




```
Company6 :
β0 = 3.4196890116416476  ---->
β1 = 1.1318326183131526  ---->
y = 3.42 + 1.132x
```



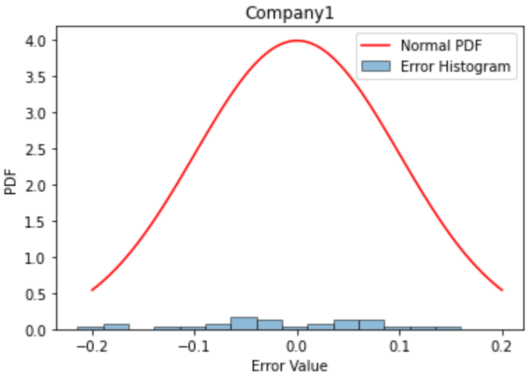

```
beta_log array :
[[3.01954528 0.10241457]
 [2.85118001 0.31698482]
 [3.07428538 0.50954068]
 [2.78269127 0.71935829]
 [2.84287641 0.93749158]
 [3.41968901 1.13183262]]
```

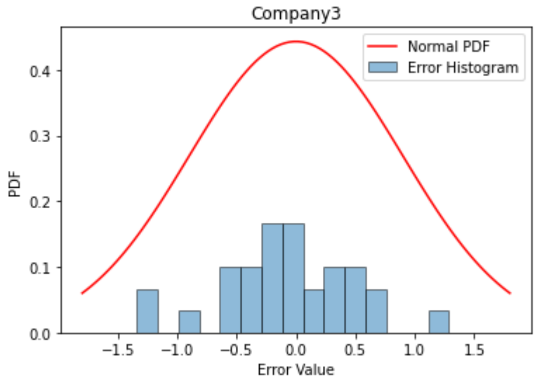
برای محاسبه ی تقریب خطی یا linear regression یک تابع می نویسیم. برای محاسبه ی ضرائب آن کتابخانه هایی نیز وجود دارد اما در اینجا به صورت دستی از روش least square آن ها را محاسبه می کنیم. سپس هم نمودار اصلی را scatter می کنیم و هم خط تقریب زده را رسم می کنیم و انطباق آن را مشاهده می کنیم. از آنجا که در قسمت های بعدی با این ضرائب سر و کار داریم آن ها را در یک آرایه ی 6x2 به نام beta_log ذخیره می کنیم.

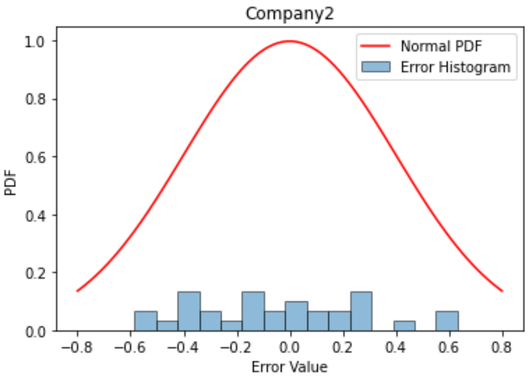
در خصوص شروع روز ها از روز صفر، در واقع به جای روز ۱ تا ۳۰ به صورت روز ۰ ام تا ۲۹ ام پلات شده اند، البته تاثیر خاصی در دیتا یا رسم ما نخواهد داشت.

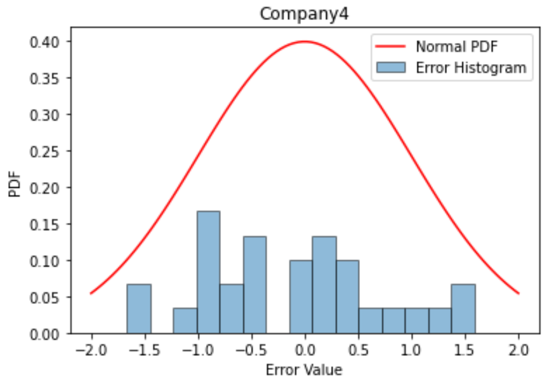
شبیه سازی 6

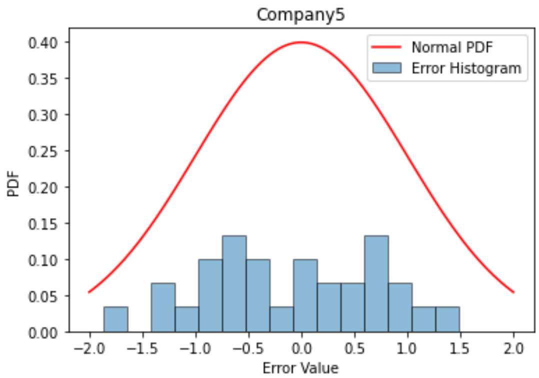
```
import math

def ErrorCalculator(PandasDataFrame,beta_log,Vars):

    for i in range(0,6):
        CompanyData = np.array(PandasDataFrame[i])

        beta_0 = beta_log[i,0]
        beta_1 = beta_log[i,1]

        ErrorLog = []

        for x in range (0,30):
            ErrorLog.append(CompanyData[x] - (beta_0 + beta_1*x))


        hist,bins = np.histogram(ErrorLog,bins=15)
        pdf = hist/30
        plt.bar(bins[:-1],pdf,width = np.diff(bins),edgecolor='k',alpha=0.5)


        mean = 0
        std = math.sqrt(suitable_vars[i])

        x = np.linspace(mean - 2*std, mean + 2*std, 100)
        plt.plot(x, norm.pdf(x, mean, std),color='r')
        plt.ylabel("PDF")
        plt.xlabel("Error Value")
        plt.title(f"Company{i+1}")
        plt.legend(["Normal PDF","Error Histogram"])
        plt.show()



suitable_vars = [0.01 , 0.16 , 0.81 , 1 , 1 , 2]
ErrorCalculator(dataFile,beta_log,suitable_vars)
```

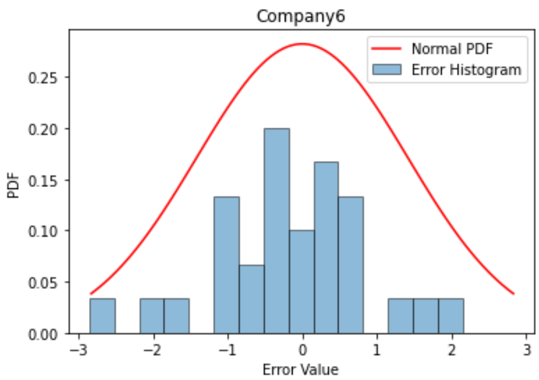

Company1
Normal PDF
Error Histogram
PDF
Error Value


Company3
Normal PDF
Error Histogram
PDF
Error Value


Company2
Normal PDF
Error Histogram
PDF
Error Value


Company4
Normal PDF
Error Histogram
PDF
Error Value


Company5
Normal PDF
Error Histogram
PDF
Error Value


Company6
Normal PDF
Error Histogram
PDF
Error Value

در این بخش تابعی می نویسیم که از اختلاف نمودار اصلی با تقریب خطی ما ، آرایه ای خطا ها تشکیل دهد و دیتای آن را به صورت یک هیستوگرام پلات کند. (برای اینکه در ادامه راحت تر باشیم هم نمودار خطا و هم نرمال را به صورت pdf اندازه گیری و ریسم می کنیم)

سپس باید هیستوگرام یک توزیع نرمال را رسم کنیم که برای اینکه شکل ها کمتر قاطی شوند آن را پلات می کنیم تا دو نمودار راحت تر قابل بررسی باشند. حال با بالا و پایین بردن مقادیر ماتریس suitable_vars به دنبال مقادیری می گردیم که به خوبی دیتای ما را با توزیع نرمال پوشش دهد. ابتدا به صورت کلی مقداری نسبت می دهیم و بعد نگاه می کنیم که در بازه های σ-+ و 2σ-+ نسبت به صفر یا همان میانگین چه تعدادی از دیتا ها پوشش داده شده اند و مجددا اگر نیاز به اصلاح واریانس بود آن را کم و زیاد می کنیم تا یک مقدار نسبی مناسب پیدا شود.

No.

Date.

پرسش تئوری ۲۳)

$$U_i = \mathcal{N}\left(m_i\left(1+\frac{Y}{H_i}\right) , \sigma_i^{۲}\right) \Rightarrow$$

Company $\frac{۱}{۶}$:

$$\left.\begin{aligned} m_۱ &= B_{۱_۱} \simeq ۰/۱۰۲ \\ \sigma_۱^۲ &= ۰/۰۱ \\ H_۱ &= ۲/۷(۷-۱) = ۱۶/۲ \end{aligned}\right\} \Rightarrow \boxed{U_۱ = N\left(۰/۱۰۲\left(1+\frac{Y}{۱۶/۲}\right) \text{ و } ۰/۰۱\right)}$$

Company $\frac{۲}{۶}$:

$$\left.\begin{aligned} m_۲ &= B_{۱_۲} \simeq ۰/۳۱۷ \\ \sigma_۲^۲ &= ۰/۱۶ \\ H_۲ &= ۲/۷(۷-۲) = ۱۳/۵ \end{aligned}\right\} \Rightarrow \boxed{U_۲ = N\left(۰/۳۱۷\left(1+\frac{Y}{۱۳/۵}\right) \text{ و } ۰/۱۶\right)}$$

Company $\frac{۳}{s}$: $m_{۳} = B_{1_{۳}} \simeq ۰/۵۰۹$

$\sigma_{۳}^{۲} = ۰/۸۱$

$H_{۳} = ۲,۷(۷-۳) = ۱۰/۸$

$$\Rightarrow \boxed{u_{۳} = N\left(۰/۵۰۹\left(1+\frac{Y}{۱۰/۸}\right) \text{ و } ۰,۸۱\right)}$$

Company $\frac{۴}{s}$: $m_{۴} = B_{1_{۴}} \simeq ۰/۷۱۹$

$\sigma_{۴}^{۲} = 1$

$H_{۴} = ۲,۷(۷-۴) = ۸,۱$

$$\Rightarrow \boxed{u_{۴} = N\left(۰/۷۱۹\left(1+\frac{Y}{۸,۱}\right) \text{ و } 1\right)}$$

Company $\frac{۵}{s}$: $m_{۵} = B_{1_{۵}} \simeq ۰/۹۳۷$

$\sigma_{۵}^{۲} = 1$

$H_{۵} = ۲,۷(۷-۵) = ۵/۴$

$$\Rightarrow \boxed{u_{۵} = N\left(۰/۹۳۷\left(1+\frac{Y}{۵/۴}\right) \text{ و } 1\right)}$$

Company $\frac{۶}{s}$: $m_{۶} = B_{1_{۶}} \simeq ۱,۱۳۱$

$\sigma_{۶}^{۲} = ۲$

$H_{۶} = ۲,۷(۷-۶) = ۲,۷$

$$\Rightarrow \boxed{u_{۶} = N\left(۱,۱۳۱\left(1+\frac{Y}{۲,۷}\right) \text{ و } ۲\right)}$$

پرسش تئوری ۲۴)

آرتا. $a_i = 7 - i$

سورنا $b_i = \sqrt{49 - i^2}$

۶	۵	۴	۳	۲	۱
$۴\sqrt{۳}$	$۳\sqrt{۵}$	$۲\sqrt{۱۰}$	$\sqrt{۳۳}$	$۲\sqrt{۶}$	$\sqrt{۱۳}$
Comp ۱	Comp ۲	Comp ۳	comp ۴	comp ۵	comp ۶

سورنا

آرتا	C_1	C_2	C_3	C_4	C_5	C_6
C_1	۰/۱۸۳	۰٫۱۳۹	۰/۱۳۹	۰/۱۳۹	۰/۱۳۹	۰/۱۳۹
C_2	۰٫۴۲	۰/۵۹۱	۰/۴۲	۰/۴۲	۰/۴۲	۰/۴۲
C_3	۰/۶۹۷	۰/۶۹۷	۰/۹۹۵	۰/۶۹۷	۰/۶۹۷	۰/۶۹۷
C_4	۰٫۹۸۵	۰/۹۸۵	۰٫۹۸۵	۱/۴۹۵	۰٫۹۸۵	۰/۹۸۵
C_5	۱٫۲۸۴	۱٫۲۸۴	۱٫۲۸۴	۱٫۲۸۴	۲/۱۳۴	۱٫۲۸۴
C_6	۱٫۵۵	۱٫۵۵	۱٫۵۵	۱٫۵۵	۱٫۵۵	۳٫۰۶

"Game_Matrix" with respect to Arta Bank's Payoff

سورنا

آرتا	C_1	C_2	C_3	C_4	C_5	C_6
C_1	۰/۱۸۳	۰/۴۷۴	۰/۸۰۷	۱٫۲۲۸	۱٫۷۸۷	۲٫۶۴
C_2	۰/۱۴۵	۰/۵۹۱	۰٫۸۰۷	۱٫۲۲۸	۱٫۷۸۷	۲٫۶۴
C_3	۰/۱۴۵	۰/۴۷۴	۰/۹۹۵	۱٫۲۲۸	۱٫۷۸۷	۲٫۹۴
C_4	۰/۱۴۵	۰/۴۷۴	۰٫۸۰۷	۱٫۴۹۵	۱٫۷۸۷	۲٫۶۴
C_5	۰/۱۴۵	۰/۴۷۴	۰/۸۰۷	۱٫۲۲۸	۲٫۱۳۴	۲٫۹۴
C_6	۰/۱۴۵	۰/۴۷۴	۰٫۸۰۷	۱٫۲۲۸	۱٫۷۸۷	۳/۰۶

"Game Matrix" with Respect to Sorena Bank's Payoff

با توجه به لیمت های بدست آمده در بالا و همچنین داشتن فرمول های توزیع های نرمال و دانستن اینکه expected توزیع نرمال همان میانگین آن است کافی است دو ماتریس payoff تشکیل دهیم و اعداد سرمایه گذاری را یا به صورت تک اگر فقط یک بانک سرمایه گذاری کرده و یا به صورت مجموع سرمایه گذاری اگر هر دو سرمایه گذاری کرده اند، قرار دهیم و محاسبات را انجام دهیم تا ماتریس گیم بدست آید. قطر ماتریس در هر دو ماتریس یکسان است. در ماتریس بانک آرتا ، از آنجا که در انتخاب ها کمپانی هایی که بانک دیگر سرمایه گذاری نکرده، بانک دیگر اهمیتی ندارد ، همواره اعداد سطر به جز در قطر ثابت و برعکس همین قضیه برای ماتریس بانک سورنا ، ستون ها به جز در المان های قطر اعداد ثابت اند. به جهت ازدیاد محاسبات از آوردن آن ها صرف نظر شد ولی محاسبه ی هر خانه به همین شیوه ی بیان شده صورت گرفته است. می توان ماتریس را به int ، cast کرد ولی خب چون دیگر محاسبات و بخش دردسردار را با اعداد دابل و اعشاری پیش رفته ایم ادامه می دهیم.

شبیه سازی 7

```
import nashpy as nash

Arta = np.array([[0.183 , 0.139 , 0.139 , 0.139 , 0.139 , 0.139],
                 [0.420 , 0.591 , 0.420 , 0.420 , 0.420 , 0.420],
                 [0.697 , 0.697 , 0.995 , 0.697 , 0.697 , 0.697],
                 [0.985 , 0.985 , 0.985 , 1.495 , 0.985 , 0.985],
                 [1.284 , 1.284 , 1.284 , 1.284 , 2.134 , 1.284],
                 [1.550 , 1.550 , 1.550 , 1.550 , 1.550 , 3.060]])

Sorena = np.array([[0.183 , 0.474 , 0.807 , 1.228 , 1.787 , 2.640],
                   [0.145 , 0.591 , 0.807 , 1.228 , 1.787 , 2.640],
                   [0.145 , 0.474 , 0.995 , 1.228 , 1.787 , 2.640],
                   [0.145 , 0.474 , 0.807 , 1.495 , 1.787 , 2.640],
                   [0.145 , 0.474 , 0.807 , 1.228 , 2.134 , 2.640],
                   [0.145 , 0.474 , 0.807 , 1.228 , 1.787 , 3.060]])


game = nash.Game(Arta, Sorena)
eq = game.vertex_enumeration()
print(list(eq))
```

```
[(array([ 0.00000000e+00, -2.12330153e-17,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  1.00000000e+00]), array([0., 0., 0., 0., 0., 1.]))]
```

ماتریس ها را وارد کرده و گیم را با یکی از متود هایی که در سوال ۷ با آن آشنا شدیم بررسی می کنیم. مشاهده می شود که یک pure strategy داریم که به صورت شهودی هم مشخص بود که سرمایه گذاری هر دو بانک در کمپانی ۶ برای هر دو سود آور خواهد بود. چرا که اعداد

همگی مثبت اند و خب شیب کمپانی ۶ در صورت سرمایه گذاری هر دو از هر خانه ای بیشتر است. که البته این قضیه به صورت کامپیوتری نیز با محاسبات کتابخانه ی Nashpy نیز مشاهده شد.

شبیه سازی 8

```
np.random.seed(400101989)
A = np.zeros((6,6))
B = np.zeros((6,6))


for i in range (6):
    sample = np.random.normal(0, math.sqrt(suitable_vars[i]), 1)
    A[i,:] = sample
    B[:,i] = sample

print(A)
print()
print(B)
print()


game2 = nash.Game(A, B)
eq2 = game2.vertex_enumeration()
print(list(eq2))
```

```
[[-0.15747374 -0.15747374 -0.15747374 -0.15747374 -0.15747374 -0.15747374]
 [ 0.54533115  0.54533115  0.54533115  0.54533115  0.54533115  0.54533115]
 [-0.39357292 -0.39357292 -0.39357292 -0.39357292 -0.39357292 -0.39357292]
 [-0.63468715 -0.63468715 -0.63468715 -0.63468715 -0.63468715 -0.63468715]
 [ 0.32205047  0.32205047  0.32205047  0.32205047  0.32205047  0.32205047]
 [-1.50979225 -1.50979225 -1.50979225 -1.50979225 -1.50979225 -1.50979225]]

[[-0.15747374  0.54533115 -0.39357292 -0.63468715  0.32205047 -1.50979225]
 [-0.15747374  0.54533115 -0.39357292 -0.63468715  0.32205047 -1.50979225]
 [-0.15747374  0.54533115 -0.39357292 -0.63468715  0.32205047 -1.50979225]
 [-0.15747374  0.54533115 -0.39357292 -0.63468715  0.32205047 -1.50979225]
 [-0.15747374  0.54533115 -0.39357292 -0.63468715  0.32205047 -1.50979225]
 [-0.15747374  0.54533115 -0.39357292 -0.63468715  0.32205047 -1.50979225]]

[(array([ 1.42602833e-17,  1.00000000e+00, -1.42602833e-17, -1.42602833e-17,
        0.00000000e+00,  0.00000000e+00]), array([ 1.42602833e-17,  1.00000000e+00, -1.42602833e-17, -1.42602833e-17,
        0.00000000e+00,  0.00000000e+00]))]
```

در این قسمت از کتابخانه ی نامپای کمک می گیریم و مقادیر واریانس ها را به آن می دهیم تا یک random sample از توزیع مرتبط با هر کدام درست کند و دو ماتریس گیم را تشکیل می دهیم و آن ها را بررسی می کنیم. مجددا یک pure strategy داریم که در شکل مشخص شده است.