Nashpy

فصل یک: یافتن تعادل برای یک بازی

۱-۱- مقدمه ای بر نظریه بازی ها

نظریه بازی مطالعه تعاملات استراتژیک بین عوامل منطقی است، زمانی که طرفین درگیر تلاش می کنند آنچه را که از دیدگاه آنها بهترین است انجام دهند. به عنوان مثال سنگ، کاغذ و قیچی را در نظر بگیرید. این یک بازی رایج است که در آن دو بازیکن یکی از ۳ گزینه را انتخاب می کنند. در تئوری بازی به این گزینه ها، استراتژی می گویند.

برنده بر اساس موارد زیر تعیین می شود:

- سنگ قیچی را خرد می کند.
- کاغذ به سنگ برتری دارد.
- قیچی کاغذ را برش می دهد.

ما می توانیم این را به صورت ریاضی با استفاده از یک ماتریس ۳ در ۳ نشان دهیم:

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

در ماتریس Aij در نظر بگیرید که بازیکن یک سطرها (ردیف i) و بازیکن دو ستون ها (ستون j) را کنترل می کند. اگر بازیک یک استراتژی سوم (قیچی) و بازیکن دو استراتژی دوم (کاغذ) را انتخاب کند:

$A32 = 1$ و قیچی کاغذ را برش می‌دهد.

یک کتاب متنی توصیه شده در مورد نظریه بازی ها [Maschler2013] است.

Maschler, M., Eilon Solan, and Shmuel Zamir. “Game theory. Translated from the Hebrew by Ziv Hellman and edited by Mike Borns.” (2013).

۱-۲- نصب نش پای

پیش نیازهای نصب نش پای:

Python 3.5 به بالا

Scipy 0.19.0 به بالا

Numpy 1.12.1 به بالا

با فرض اینکه آن ها را نصب کرده اید، برای نصب Nashpy در Mac OSX یا linux یک ترمینال و در ویندوز Command prompt یا موارد مشابه را باز کنید و و تایپ کنید:

$ python -m pip install nashpy

اگر این کار نمی کند، ممکن است پایتون یا یکی از وابستگی های دیگر را نداشته باشید. همچنین ممکن است به دلیل شناسایی نشدن پیپ با مشکل مواجه شوید. برای غلبه بر این موارد، استفاده از Anaconda توصیه می‌شود، زیرا به‌راحتی روی همه سیستم‌عامل‌ها نصب شده و همچنین شامل کتابخانه‌های مورد نیاز برای اجرای Nashpy است.

۱-۳- ایجاد یک بازی

ما می توانیم بازی فوق را با استفاده از Nashpy ایجاد کنیم:

```
>>> import nashpy as nash
>>> import numpy as np
>>> A = np.array([[0, -1, 1], [1, 0, -1], [-1, 1, 0]])
>>> rps = nash.Game(A)
>>> rps
Zero sum game with payoff matrices:

Row player:
[[ 0 -1  1]
 [ 1  0 -1]
 [-1  1  0]]

Column player:
[[ 0  1 -1]
 [-1  0  1]
 [ 1 -1  0]]
```

همچنین، ما می توانیم یک جفت ماتریس را به کلاس گیم ارسال کنیم تا همان بازی را ایجاد کنیم:

```
>>> B = - A
>>> rps = nash.Game(A, B)
>>> rps
Zero sum game with payoff matrices:

Row player:
[[ 0 -1  1]
 [ 1  0 -1]
 [-1  1  0]]

Column player:
[[ 0  1 -1]
 [-1  0  1]
 [ 1 -1  0]]
```

ما دقیقاً همان بازی را دریافت می کنیم، اگر یک بازی را پشت سر بگذاریم، نشپای فرض می کند که بازی یک **بازی با مجموع صفر** است: به عبارت دیگر ماتریس های هر دو بازیکن برعکس است. .