گزارش مینی پروژه اول درس یادگیری تقویتی

مریم واقعی

دانشگاه فردوسی مشهد، دانشکده مهندسی

ma.vaghei78@gmail.com

اطلاعات گزارش

تاریخ:

۲۷ فروردین ۱۴۰۲

واژگان کلیدی:

Policy iteration

Value iteration

Gym

سیاست بهینه

ارزش بهینه

gamma

چکیده

در این پروژه، ابتدا با تعریف محیط بازی FrozenLake در کتابخانه Gym، یک محیط بازی را ایجاد می‌کنیم. سپس با استفاده از دو الگوریتم Policy Iteration و Value Iteration، برای پیدا کردن بهترین سیاست و برای بهترین عملکرد محیط، تغییرات مختلفی را در ضریب تنزیل و جریمه اعمال می‌کنیم. در انتها، با تحلیل و بررسی نتایج به دست آمده، می‌توانیم نتیجه‌گیری‌هایی در مورد عملکرد هر الگوریتم و تاثیر تغییرات مختلف بر آن‌ها بیان کنیم.

فهرست مطالب

۱-مقدمه

در این پروژه، قصد داریم برای محیط بازی FrozenLake از کتابخانه Gym یک پیاده‌سازی از دو الگوریتم Policy Iteration و Value Iteration را انجام دهیم. در این پروژه، ابتدا با تعریف محیط FrozenLake، سپس با استفاده از دو الگوریتم Policy Iteration و Value Iteration، برای پیدا کردن بهترین سیاست و برای بهترین عملکرد محیط، تغییرات مختلفی را در ضریب تنزیل و جریمه اعمال می‌کنیم و تاثیرات آن را روی محیط و عامل گزارش میکنیم.

۲- الگوریتم Policy Iteration

این الگوریتم را با استفاده از الگوریتمی که در کتاب آورده شده است پیاده میکنیم:

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

برای گام اول این الگوریتم، مقادیر اولیه ارزش به صورت پیش فرض صفر قرار داده شده است.

برای گام دوم ما متد evaluate_policy را پیاده کردیم که برای یک policy مشخص مقادیر ارزش را بدست می‌آورد و به این صورت policy را مورد ارزیابی قرار میدهد و در این تابع مقادیر ارزش بدست آمده برای هر state با استفاده از policy مشخص return میشود.

برای گام دوم ما متد improve_policy را پیاده کردیم که در آن با استفاده از مقادیر ارزش جدید که در متد قبلی محاسبه شده، مقادیر policy را آپدیت میکند.

این دو گام یعنی گام دوم و سوم تا زمانی که مقادیر policy تغییر نکنند ادامه میابد.

در نهایت پس از ثبات مقادیر policy مقادیر ارزش و سیاست در متد policy_iteration بازگردانده میشوند.

۳- الگوریتم Value Iteration

این الگوریتم را نیز با استفاده از الگوریتمی که در کتاب آورده شده است پیاده میکنیم:

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$

در این الگوریتم نیز مشابه الگوریتم قبلی، مقادیر اولیه ارزش برابر با صفر قرار داده شده است.

در گام دوم الگوریتم، مشابه الگوریتم قبلی ما مقدار ارزش را به ازای هر حالت بدست می آوریم با این تفاوت که در اینجا مقدار ارزش برابر با حداکثر مقدار ارزش به ازای هر action قرار میگیرد و ما به جای اینکه سیاست را در نظر بگیریم برای تمام اکشن ها مقدار ارزش را بدست می آوریم و بیشترین مقدار در V(s) قرار میگیرد.

پس از این مرحله در گام سوم ما مقدار سیاست را برای هر حالت بدست می‌آوریم که این کار را در متد improve_policy انجام میدهیم که مقدار policy مربوط به هر حالت برابر با اکشن با بیشترین مقدار ارزش قرار میگیرد.

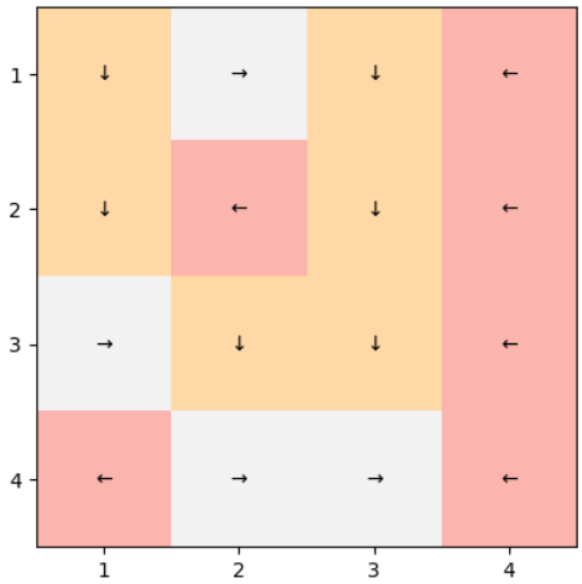
در نهایت نیز مقادیر ارزش و سیاست در متد value_iteration بازگردانده میشوند.

۴- بررسی نتایج

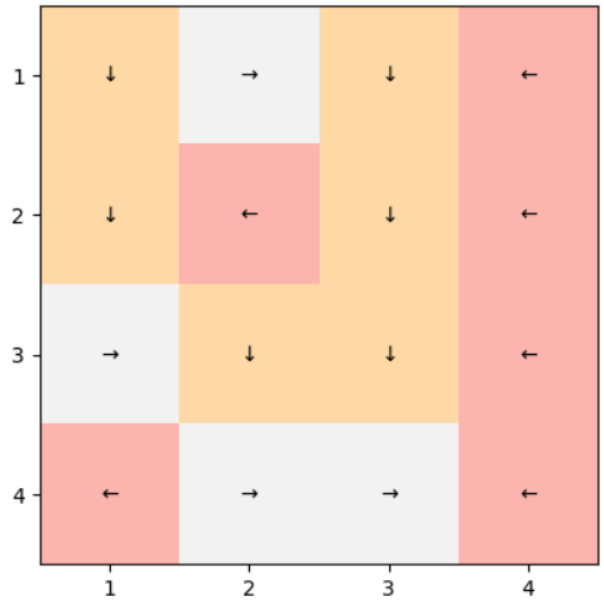
۴-۱- بررسی خروجی کد

در این بخش فانکشن show_policy را برای نمایش سیاست با استفاده از کتابخانه matplotlib نوشته ایم.

با محیط پیش فرض frozenLake ، مقادیر سیاست برای policy iteration و value iteration به صورت زیر می باشد:



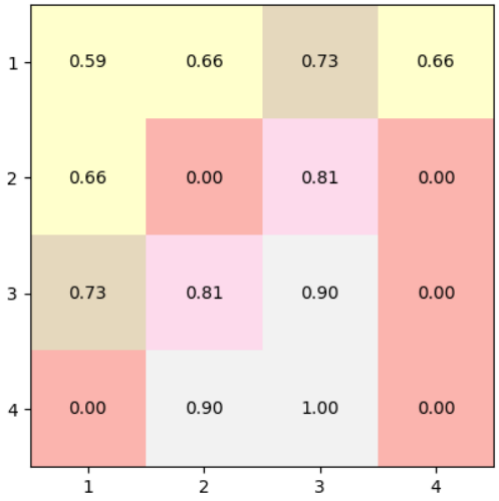
تصویر ۱- خروجی سیاست بهینه برای الگوریتم policy iteration با gamma = 0.9



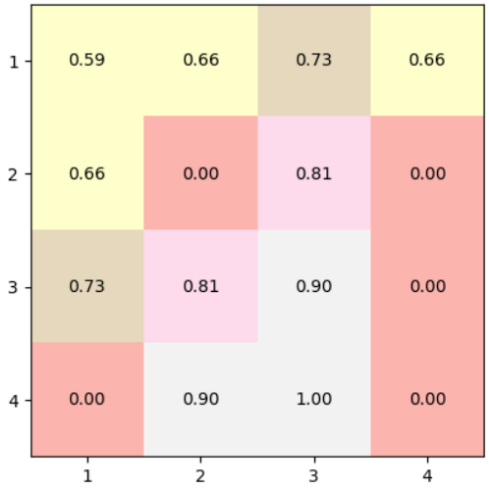
تصویر ۲- خروجی سیاست بهینه برای الگوریتم value iteration با gamma = 0.9

همانطور که در هر دو سیاست مشاهده میکنیم، اگر از خانه $(0,0)$ شروع کنیم، با دنبال کردن سیاست بهینه به خانه $(4,4)$ که مقصد است می رسیم بدون اینکه در چاله ها بیفتیم.

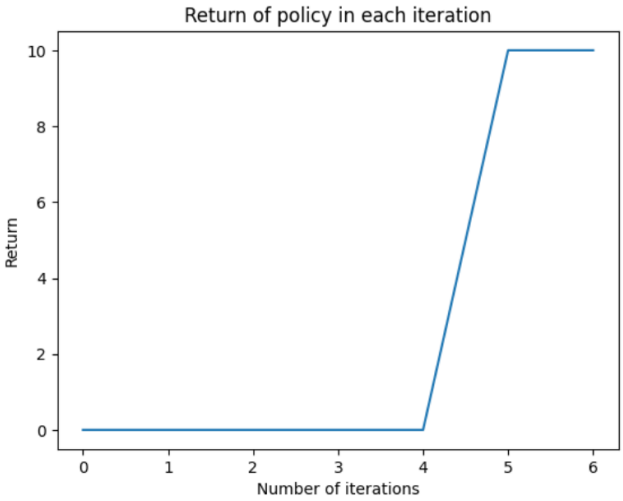
همچنین مقادیر ارزش بهینه برای این دو الگوریتم به صورت زیر میباشد:



تصویر ۳- خروجی ارزش بهینه برای الگوریتم policy iteration با gamma = 0.9



تصویر ۴- خروجی ارزش بهینه برای الگوریتم value iteration با gamma = 0.9



تصویر ۵- مقدار Return به ازای هر بار آپدیت policy برای gamma = 0.9

همانطور که میبینیم برای استیت های غیر ترمینال هرچه به هدف یعنی خانه (4,4) نزدیک میشویم، مقدار ارزش بیشتر میشود به طوری که در حالت (4,3) که state یکی مانده به هدف است، مقدار ارزش ۱ است زیرا با اکشن راست به هدف میرسیم.

برای خانه های چاله نیز مقدار ارزش صفر هست، چون ترمینال هستند و رسیدن به آنها پاداش ندارد.

در اینجا نکته ای وجود دارد، اگر به مقدار ارزش استیت هدف دقت کنیم مشاهده میکنیم که مقدار صفر است. بعد از بررسی هایی که روی مقادیر reward ها در state transition probability انجام شد متوجه میشویم که تنها برای استیت ۱۴ یعنی خانه (4,3) و برای اکشن سمت راست، مقدار پاداش ۱ در نظر گرفته شده و برای تمام حالت ها و اکشن های دیگر مقدار پاداش صفر است. از طرف دیگر s_prime یعنی استیتی که از خانه ۱۵ یعنی به آن منتقل میشویم همان خانه ۱۵ است و تمام اکشن ها در استیت ۱۵ مقدار پاداش صفر دارند، در نتیجه در هنگام آپدیت مقدار ارزش خانه ۱۵ که با فرمول زیر بدست می آید، مقدار این استیت صفر باقی می ماند.

```python
# policy evaluation
def evaluate_policy(P, nS, nA, value_function, policy, gamma=0.9, tol=1e-4):
  while True:
    delta = 0
    for s in range(nS):
      value_s = value_function[s]
      policy_s = policy[s]
      q = 0
      value_function[s] = 0
      for prob, s_prime, reward, done in P[s][policy_s]:
        v_s_prime = value_function[s_prime]
        value_function[s] += prob * (reward + gamma * v_s_prime)
      delta = max(delta, abs(value_s - value_function[s]))
    if delta < tol:
      break
  return value_function
```

تصویر ۶- آپدیت مقادیر ارزش در الگوریتم policy iteration

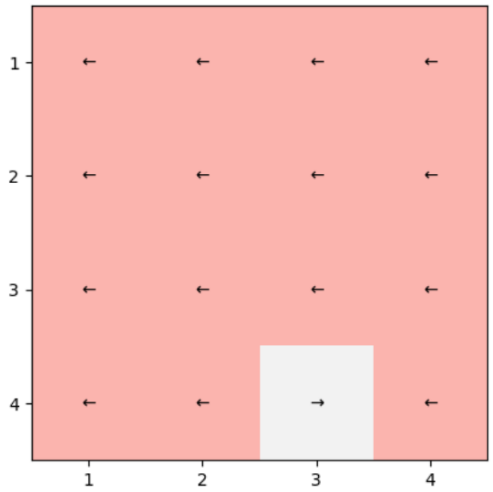
```python
# policy evaluation
def evaluate_policy(P, nS, nA, value_function, policy, gamma=0.9, tol=1e-4):
  while True:
    delta = 0
    for s in range(nS):
      value_s = value_function[s]
      q = np.zeros(nA)
      value_function[s] = 0
      for a in range(nA):
        for prob, s_prime, reward, done in P[s][a]:
          v_s_prime = value_function[s_prime]
          q[a] += prob * (reward + gamma * v_s_prime)
      value_function[s] = np.max(q)
      delta = max(delta, abs(value_s - value_function[s]))
    if delta < tol:
      break
  return value_function
```

تصویر ۷- آپدیت مقادیر ارزش در الگوریتم value iteration

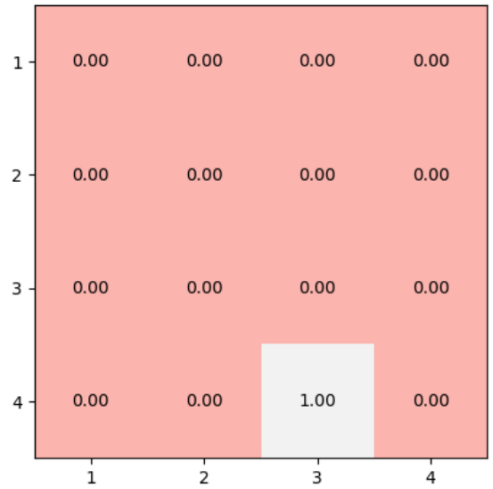
۴-۲- تغییر ضریب تنزیل و تاثیر آن در خروجی

در این مرحله برای مقادیر مختلف گاما مقدار ارزش بهینه و سیاست بهینه را بدست آوردیم و انها را باهم مقایسه میکنیم:

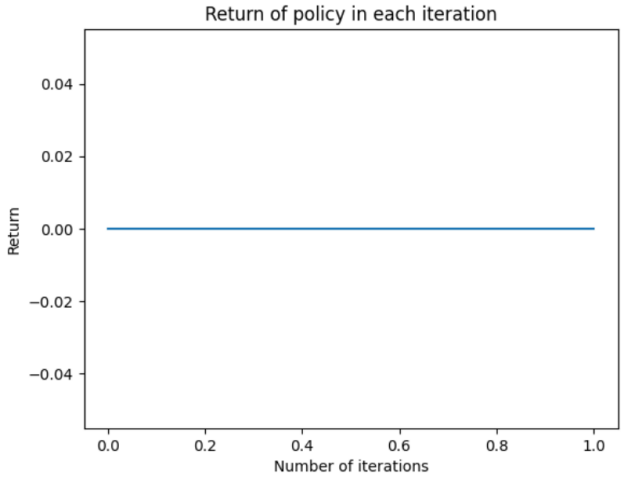
۱. حالتی که gamma = 0 می باشد، مقدار ارزش بهینه و سیاست بهینه به صورت زیر می باشد:



تصویر ۸- خروجی سیاست بهینه با gamma = 0



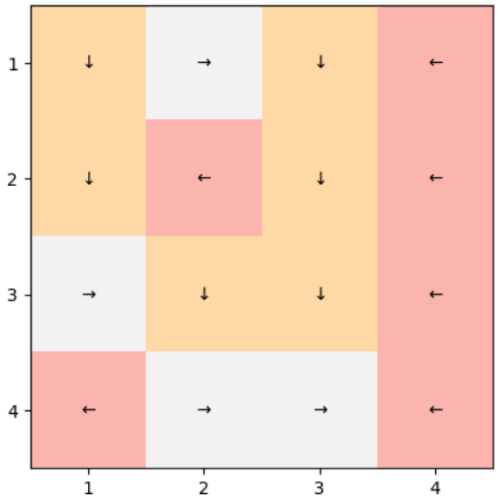
تصویر ۹- خروجی ارزش بهینه با gamma = 0



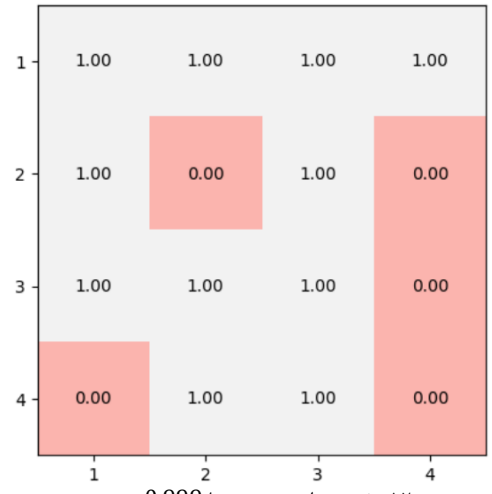
تصویر ۱۰- مقدار Return به ازای هر بار آپدیت policy برای gamma = 0

همانطور که میدانیم وقتی ضریب تنزیل مقدار صفر یا نزدیک به صفر میگیرد، در این صورت عامل فقط به reward های فوری توجه میکند و در نتیجه چون مقدار reward فوری تنها برای حالت ۱۴ یعنی (4,3) ۱ می باشد و برای بقیه حالت ها صفر است لذا مقدار ارزش تنها در خانه ۱۴ برابر با ۱ است و همچنین برای سیاست بهینه نیز تنها خانه ۱۴ سیاست درستی دارد و بقیه خانه ها به دلیل پاداش صفر، سیاست بهینه آنها همان مقدار صفر یعنی سمت چپ مانده است.
همچنین مقدار return به ازای هر بار آپدیت policy صفر است و نکته ای که وجود دارد این است که الگوریتم تنها یک iteration اجرا شده چون gamma صفر است لذا مقادیر ارزش برابر با مقدار پاداش لحظه ای هر استیت قرار می گیرد و سیاست بهینه نیز بر همین اساس بدست می آید و دیگر تغییر نمیکند.

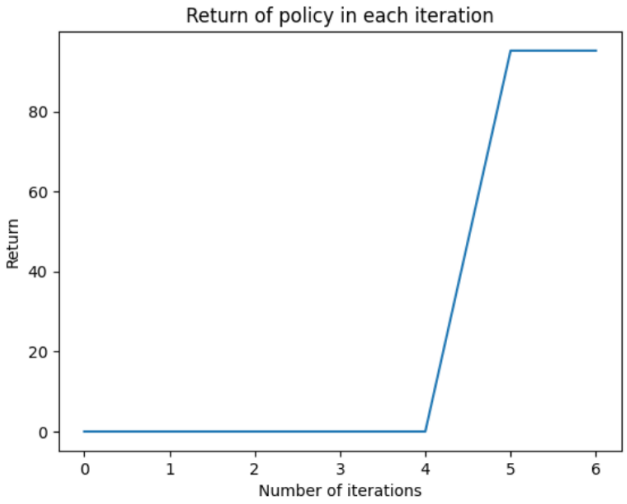
۲. حالتی که gamma = 0.999 می باشد، مقدار ارزش بهینه و سیاست بهینه به صورت زیر می باشد:



تصویر ۱۱- خروجی سیاست بهینه با gamma = 0.999



تصویر ۱۲- خروجی ارزش بهینه با gamma = 0.999

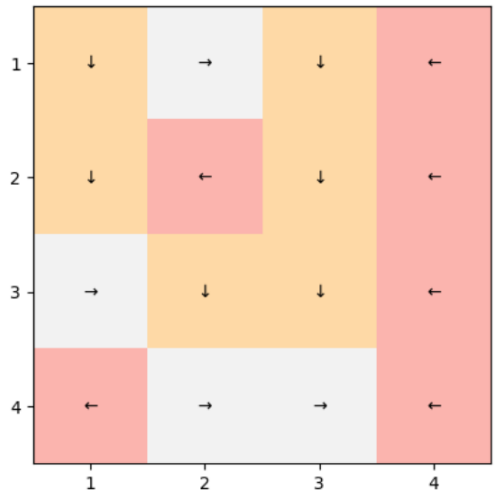


تصویر ۱۳- مقدار Return به ازای هر بار آپدیت policy برای gamma = 0.999

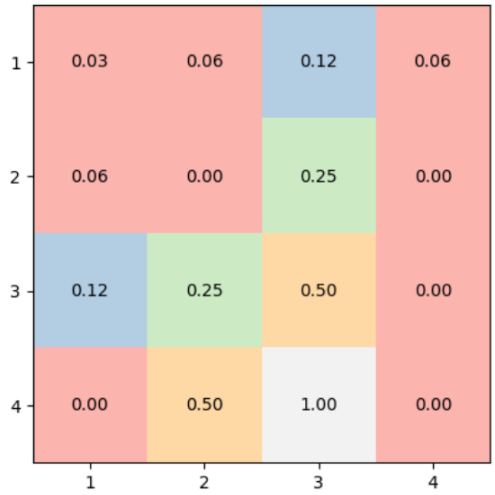
هنگامی که ضریب تنزیل روی یک عدد نزدیک به ۱ تنظیم می شود، سیاست بهینه و تابع ارزش بهینه تقریبا بدون هیچ گونه discounting در پاداش های آینده محاسبه می شوند. این به این معنی است که به همه پاداش‌های آینده همان وزن پاداش‌های فوری داده می‌شود و agent، سعی می‌کند مجموع پاداش‌هایی را که دریافت می‌کند در یک افق زمانی نامتناهی به حداکثر برساند، نه اینکه پاداش‌هایی را که از نظر زمانی نزدیک‌تر هستند را بر پاداش‌هایی که در آینده دورتر هستند، اولویت دهد.
بنابراین مشاهده میکنیم که مقدار ارزش بهینه برای همه استیت های غیرترمینال برابر با ۱ شده و تنها برای استیت های ترمینال مقدار پاداش صفر می باشد.
در مورد سیاست بهینه، چون agent افق زمانی طولانی دارد، در نتیجه سیاست بهینه با بیشترین دقت بدست آمده است و مشاهده میکنیم که جهت فلش ها به درستی در این سیاست نمایش داده شده. خروجی سیاست بهینه در این حالت مشابه حالتی است که gamma = 0.9 می باشد، تنها این دو در مقادیر ارزش با هم متفاوتند.

همچنین در نمودار return مشاهده میکنیم که تا iteration چهارم پاداش تجمعی صفر است و بعد از آن با بهبود policy این مقدار افزایش می‌یابد و در انتها به مقدار نزدیک به ۹۰ میرسد.

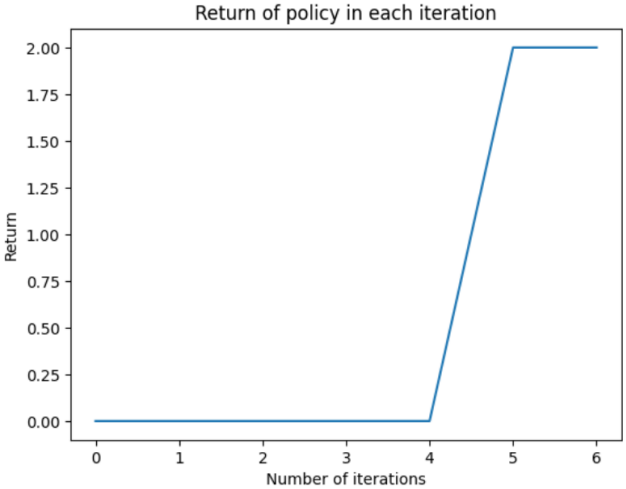
۳. حالتی که gamma = 0.5 می باشد، مقدار ارزش بهینه و سیاست بهینه به صورت زیر می باشد:



تصویر ۱۴- خروجی سیاست بهینه با gamma = 0.5



تصویر ۱۵- خروجی ارزش بهینه با gamma = 0.5



تصویر ۱۶- مقدار Return به ازای هر بار آپدیت policy برای gamma = 0.5

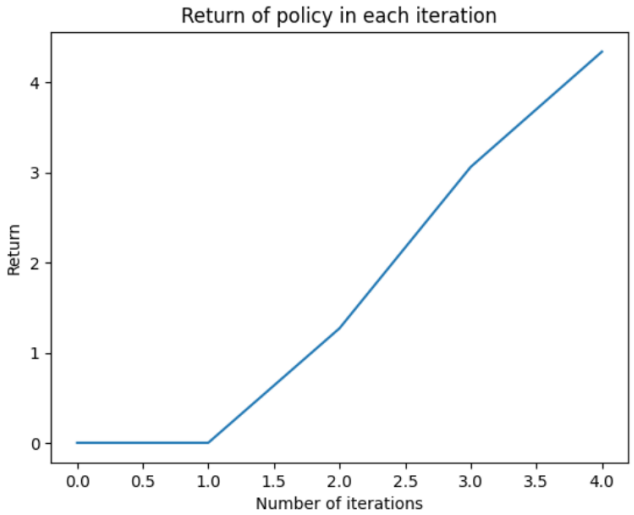
در اینجا نیز با gamma = 0.5 سیاست بهینه مناسب بدست آمده است و تنها مقادیر ارزش با حالت های قبل متفاوت است و در اینجا نیز مشاهده میکنیم که هر چه استیت به استیت هدف نزدیک میشود مقدار ارزش افزایش میابد.

بنابراین به طور کلی هرچه مقدار گاما کمتر باشد، agent به پاداش های فوری توجه بیشتری دارد و به پاداش های آینده توجه کمتری دارد و در نتیجه در حالتی که گاما صفر است، agent به صورت greedy عمل میکند.
از طرفی هرچه مقدار گاما افزایش یابد و به یک نزدیک شود، تاثیر و وزن پاداش های آینده به وزن پاداش لحظه ای نزدیک میشود و در نتیجه agent تلاش میکند، مجموع پاداش‌هایی را که دریافت می‌کند در یک افق زمانی نامتناهی به حداکثر برساند، نه اینکه پاداش‌هایی را که از نظر زمانی نزدیک‌تر هستند را بر پاداش‌هایی که در آینده دورتر هستند، اولویت دهد.

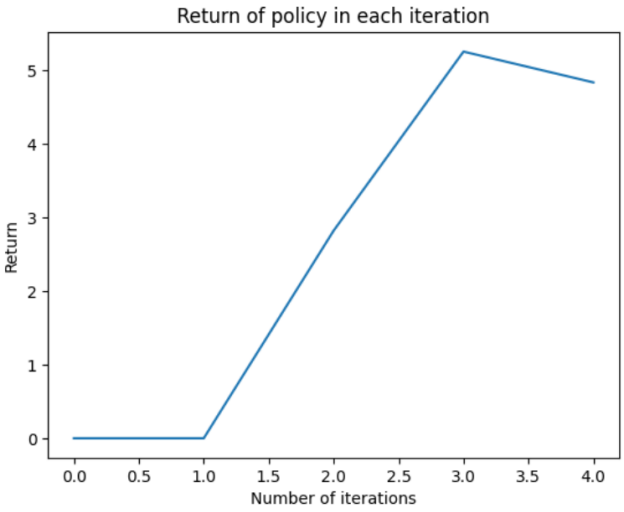
در مورد پاداش تجمعی همانطور که مشاهده میکنیم، نمودار return مشابه حالت gamma = 0.999 می باشد، با این تفاوت که مقدار پاداش تجمعی در اینجا حداکثر به ۲ میرسد.

۴-۳- حل مسئله به صورت غیرقطعی

تا اینجا نتایج براساس این بود که محیط قطعی باشد، حال با تغییر پارامتر is_slippery به True، محیط غیر قطعی میشود و احتمالات برابر به اکشن ها داده میشود.
در این حالت در هر بار اجرا، مقادیر return متفاوت است و نمودار پاداش تجمعی به ازای هر بار اجرای الگوریتم policy iteration به یک شکل می باشد.
مثلا زمانی که gamma = 0.9 است نمودار به شکل های مختلف مثل نمودار زیر می باشد:

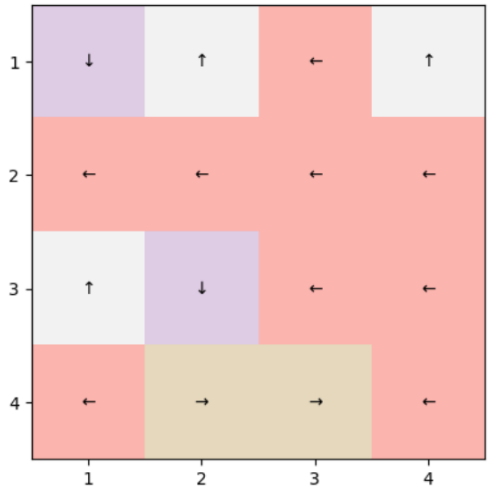


تصویر ۱۷- نمودار پاداش تجمعی برای حالت غیرقطعی با gamma = 0.9

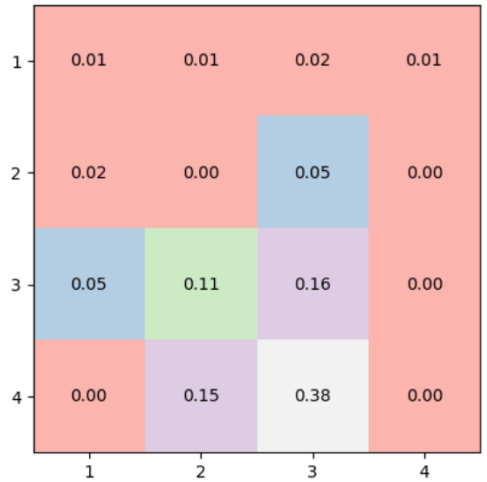


تصویر ۱۸- نمودار پاداش تجمعی برای حالت غیرقطعی با gamma = 0.9

برای نمودار بالا مقدار سیاست بهینه و ارزش بهینه به صورت زیر است:



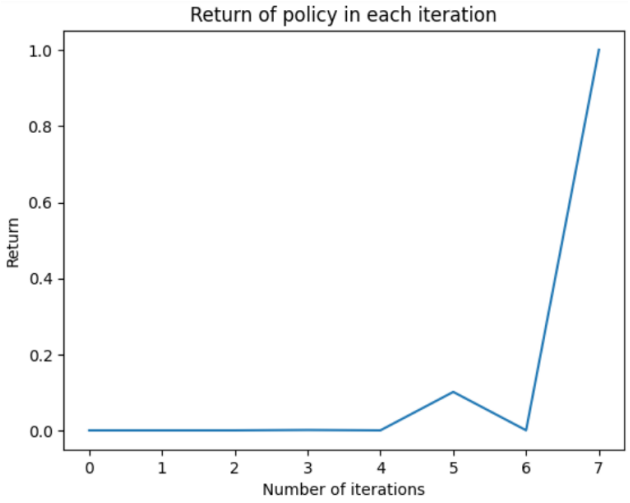
تصویر ۱۹- خروجی سیاست بهینه برای سیاست غیرقطعی با gamma = 0.9



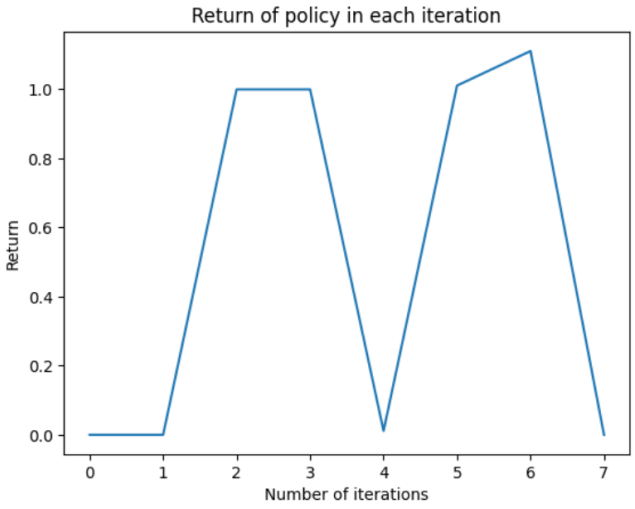
تصویر ۲۰- خروجی ارزش بهینه برای سیاست غیرقطعی با gamma = 0.9

با مقایسه نمودار پاداش تجمعی در حالت سیاست قطعی، تصویر ۵، با سیاست غیر قطعی، تصویر ۱۷ و ۱۸ متوجه میشویم اولا سیاست غیرقطعی در تعداد iteration های کمتری به مقادیر بهینه میرسد. همچنین مقدار پاداش تجمعی در حالتی که سیاست قطعی است حدودا ۱۰ می باشد در صورتی که در حالت غیرقطعی این مقدار تقریبا به نصف می رسد بنابراین agent در محیط با سیاست قطعی پاداش بیشتری دریافت میکند نسبت به حالتی که محیط و سیاست غیرقطعی است.

همچنین در سیاست غیرقطعی حالتی که gamma نزدیک به صفر میشود، میزان نوسان تابع تجمعی نسبت به حالتی که gamma نزدیک به ۱ است بیشتر میشود، در صورتی که در سیاست قطعی برای مقادیر $0 < gamma < 1$ شکل و شمایل نمودار پاداش تجمعی مشابه است فقط در مقادیر پاداش تجمعی متفاوتند.



تصویر ۲۱- نمودار پاداش تجمعی برای حالت غیرقطعی با gamma = 0.1

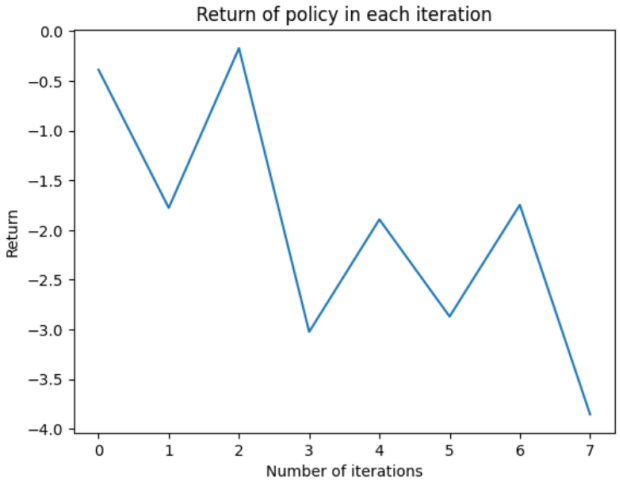


تصویر ۲۲- نمودار پاداش تجمعی برای حالت غیرقطعی با gamma = 0.1

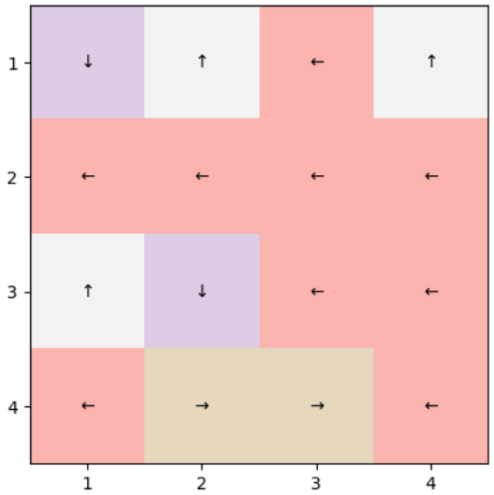
۴-۴- افزودن جریمه -۰.۰۵ به ازای هر تغییر موقعیت عامل

در این بخش به ازای هر تغییر موقعیت عامل جریمه -۰.۰۵ میدهیم.

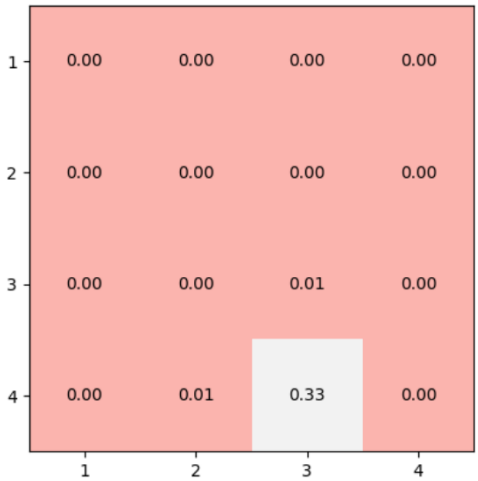
به همین دلیل نمودار پاداش تجمعی که در زیر مشاهده میکنیم، روند نزولی دارد.



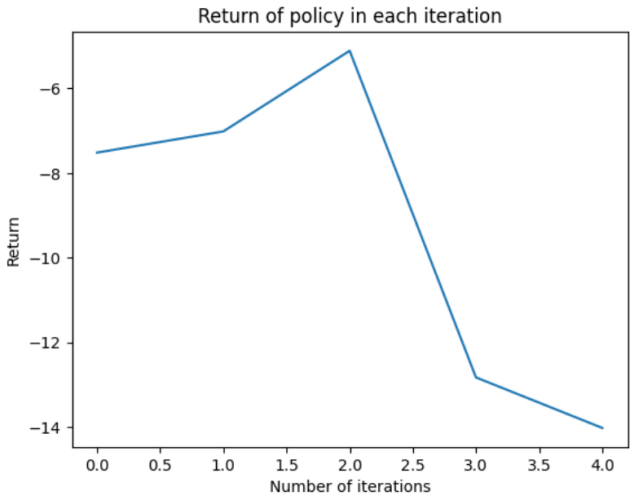
تصویر ۲۳- نمودار پاداش تجمعی برای حالت قطعی با gamma = 0.1 و جریمه -۰.۰۵ به ازای هر گام



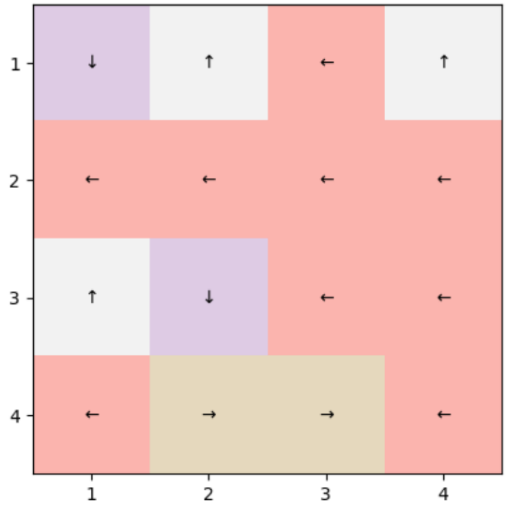
تصویر ۲۴- سیاست بهینه برای حالت قطعی با gamma = 0.1 و جریمه -۰.۰۵ به ازای هر گام



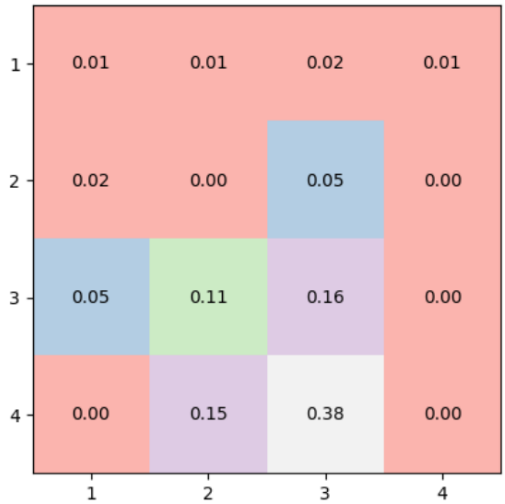
تصویر ۲۵- ارزش بهینه برای حالت قطعی با gamma = 0.1 و جریمه -۰.۰۵ به ازای هر گام



تصویر ۲۶- نمودار پاداش تجمعی برای حالت قطعی با gamma = 0.9 و جریمه -۰.۰۵ به ازای هر گام



تصویر ۲۷- سیاست بهینه برای حالت قطعی با gamma = 0.9 و جریمه -۰.۰۵ به ازای هر گام

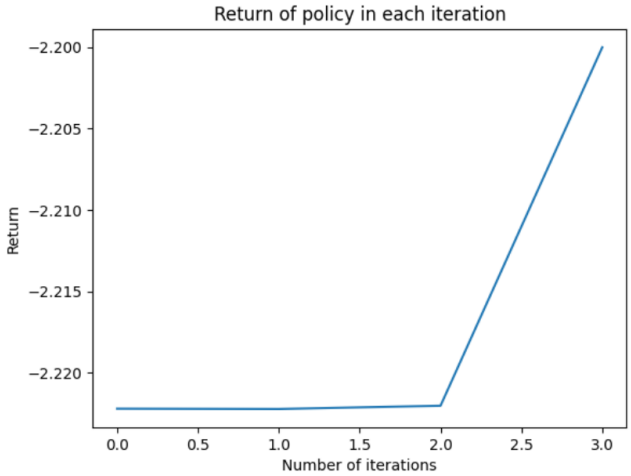


تصویر ۲۸- ارزش بهینه برای حالت قطعی با gamma = 0.9 و جریمه -۰.۰۵ به ازای هر گام

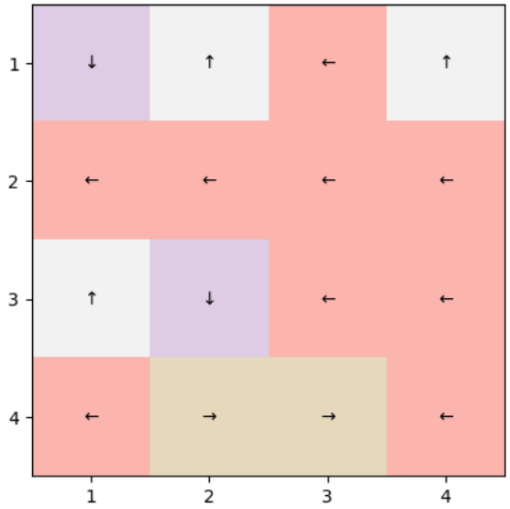
با توجه به مشاهداتی که انجام شد، میبینیم که عملکرد سیاست بهینه برای جریمه منفی مناسب نیست و هرچه مقدار گاما به صفر نزدیک میشود، عامل به صورت greedy تر عمل میکند و در نتیجه میبینیم که نمودار پاداش تجمعی برای این حالت نوسان بیشتری نسبت به گاما برابر ۰.۹ دارد.
دلیل این نزولی بودن به نظرم این هست که در ابتدا چون عامل سیاست تصادفی تری دارد در نتیجه ممکن است به جای رسیدن به مقصد، در چاله ها متوقف شود، اما زمانی که به مرور سیاست بهبود میابد و agent تلاش میکند به مقصد برسد در نتیجه گام های بیشتری طی میکند و ریوارد منفی بیشتری میگیرد.

۴-۵- افزودن جریمه -۲ برای تغییراتی که منجر به قرار گرفتن عامل در سوراخ ها شود

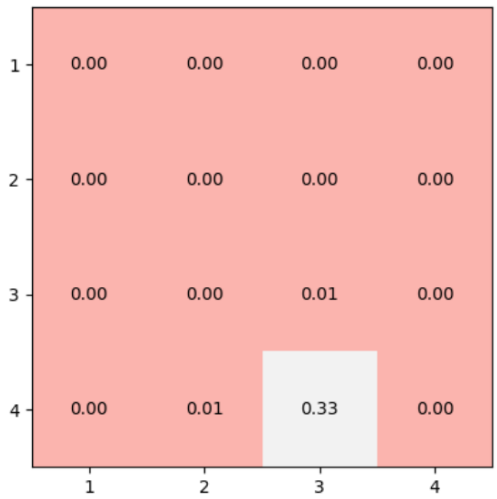
در اینجا ما برای استیت های سوراخ جریمه -۲ در نظر گرفتیم و نمودار پاداش تجمعی و سیاست بهینه و ارزش بهینه را رسم کردیم:



تصویر ۲۹- نمودار پاداش تجمعی برای حالت قطعی با gamma = 0.1 و جریمه -۲ برای سوراخ ها



تصویر ۳۰- سیاست بهینه برای حالت قطعی با gamma = 0.1 و جریمه -۲ برای سوراخ ها



تصویر ۳۱- ارزش بهینه برای حالت قطعی با gamma = 0.1 و جریمه -۲ برای سوراخ ها