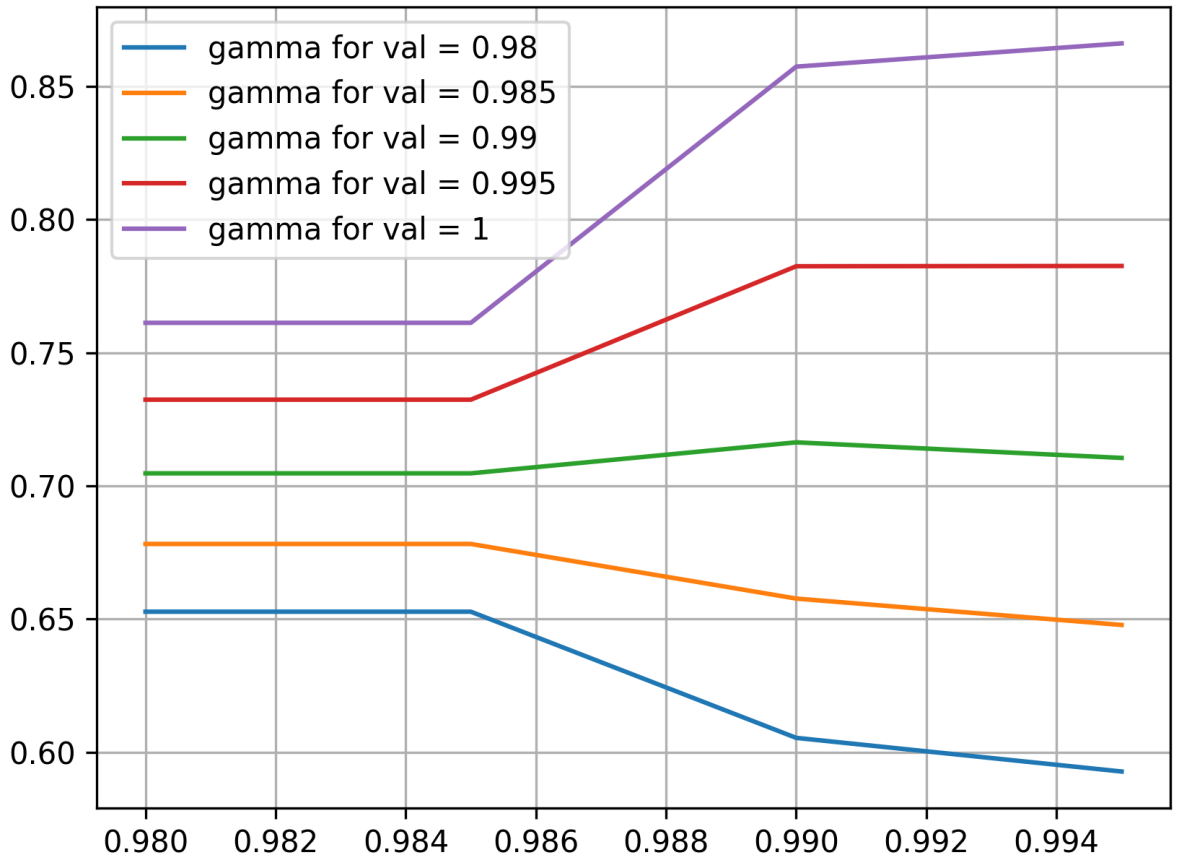
1.

Для исследования параметра гамма, который в большей степени влияет на качество решения первой задачи, я посчитал модель для следующего набора параметров:

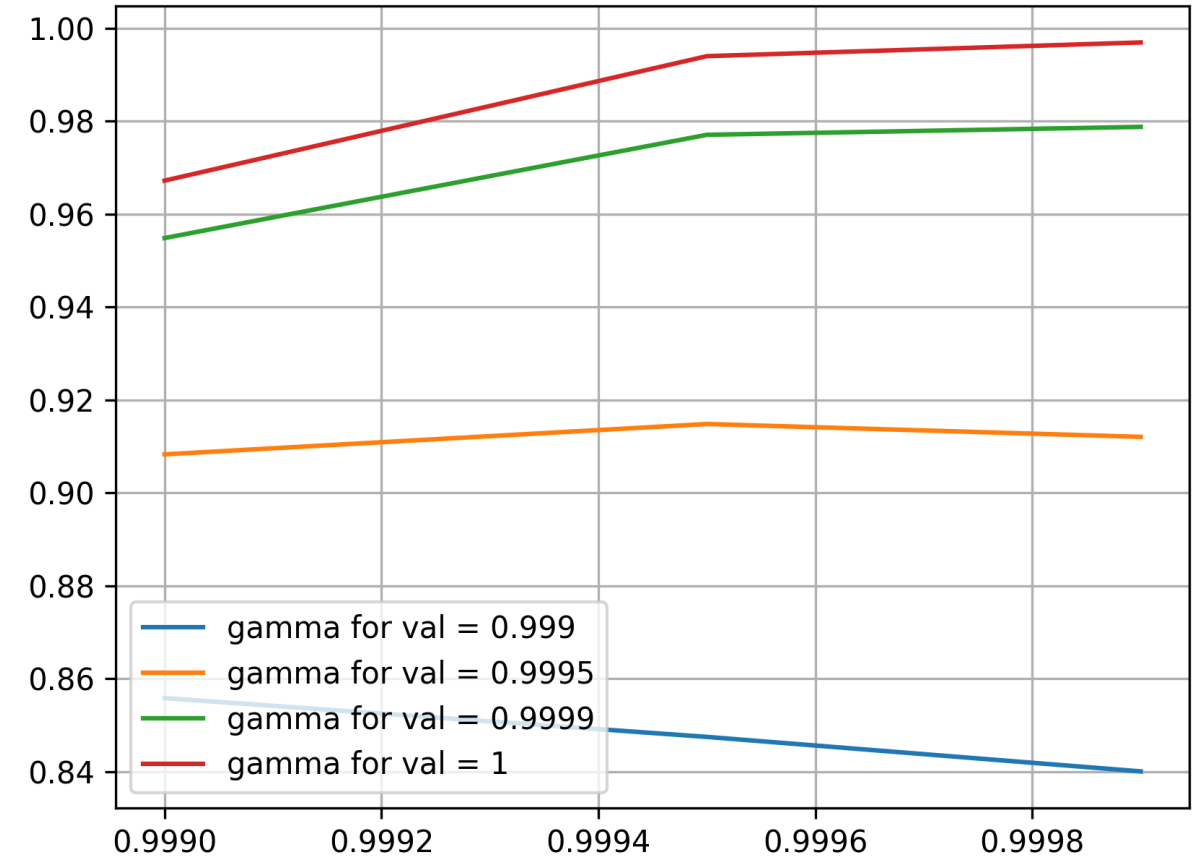
q = [ 0.980, 0.985, 0.990, 0.995, 0.999, 0.9995, 0.9999]

Изначально для валидации я использовал среднюю награду на 1000 запусках, как это было предложено на семинаре. Однако даже 100.000 прогонов оказалось недостаточно, чтобы подтвердить наглядно мои предположения, о том, что обучая с меньшим гамма мы достигаем оптимума, но не для изначальной задачи, а задачи со штрафом. Тогда для валидации я стал использовать значение value function в стартовой точке. Но для этого я немного изменил код, чтобы можно было валидировать политику полученную с помощью одного значения гамма на другом значении данного параметра. Ниже представлены графики средней награды в зависимости от параметра гамма при обучении для разных гамма на валидации.

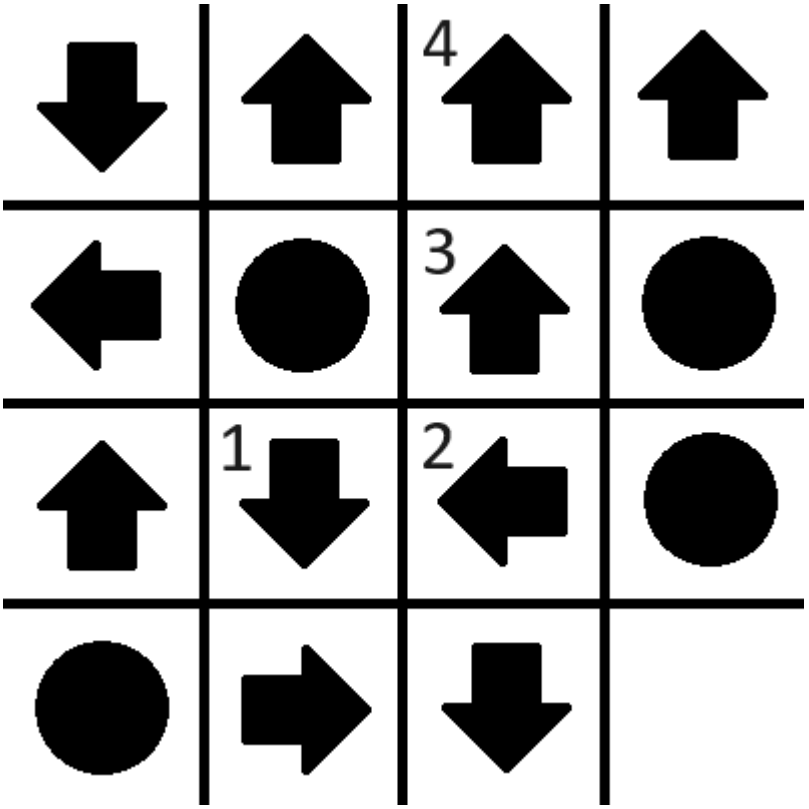



Из графика видно, что для каждого значения гамма для валидации, максимум достигается при совпадающим гамма для обучения. И на валидации без дисконта (гамма = 1) показывает себя модель обученная с помощью наибольшего гамма.
На графике может показаться, что красная линия имеет максимум в точке 0.99, однако это не так и при приближении видно, что максимум достигается именно в точке 0.995.

Аналогичный результат удается наблюдать для больших значений гамма



**Итог:**
Дальнейшее увеличении параметра гамма не дает результатов, но ухудшает сходимость. При данных гамма удается достигнуть средней награды ~0.9968. Данное значение исходя из приближенных вычислений является максимумом для данной задачи. И соответствующая политика имеет следующий вид.




Оптимальные траектории проходят через точку 1, и единственное действие которое гарантирует не попадание в яму на следующем шагу — идти вниз. Но есть шанс 0.1 попасть в точку 2, и выбраться из нее можно идя влево, но есть шанс 0.1 попасть в точку 3, откуда с вероятностью 0.2 мы попадаем в яму. Итого любая траектория проходящая через точку 1 в зависимости от политики с шансом как минимум 0.1*0.1*0.2 = 0.002 приведет нас в яму, поэто больше 0.998 нам не получить.
Также интересным фактом является выбор модели пойти в точке 3 на верх. Такое решение действительно не хуже чем пойти вниз, поскольку если мы попадаем в точку 4, то с вероятностью 100% мы попадаем в точку 1, а из точки 1 мы с вероятностью 100% либо попадаем в точку 2 либо побеждаем. А если пойти из точки три вниз, мы попадем в точку 2 с той же вероятностью, что и в точку 4 (0.8), но находясь в точке 2 мы имеем шанс упасть в яму (либо на следующем шаге, либо через точку 3). Так что точка 1 не хуже статистически чем точка 2 и следовательно идти вверх из положения 3 — не проигрышная стратегия.

2.

Для сравнения работы алгоритма, когда мы value function инициализируем нулями с алгоритмом при котором мы ее инициализируем с помощью предыдущих значений, я завел общийстечек для суммарного количества вызовов policy_evaluation_step. При это было добавлено следующее условие остановки итераци:

```
for state in v_values:
        max_dif = max(max_dif,np.abs(v_values_new[state]-v_values[state]))
v_values = v_values_new
if max_dif < 1e-5:
        break
```

**Итог:**
При таком условии и при обучении при трех разных гамма, я получил, что при нулевой инициализации функция policy_evaluation_step была вызвана 49860 раз против 21454 раз в случае ненулевой инициализации. Что в итоге привело к более чем двукратному приросту скорости обучения. Награда и политика при это осталась такой же для всех параметров гамма.

3.

В третьей задаче я так же считал количество обращений к функции policy_evaluation_step (внутренний цикл для policy iteration и внешний цикл для value iteration). Условие останова осталось тем же, и итоговая политика с наградой тоже не изменились. Единственное отличие — скорость обучения.

**Итог:**
При использовании value iteration алгоритма скорость обучения выросла еще в ~1.3 раза, и количество обращений к policy_evaluation_step  сократилось с 21454 до 14684