Reinforcement Learning Assignment № 2

Ruslan Burakov, s1569105 17/03/2015

Note

All full code fragments are located in the end of the report in the appendix section. Naming convention. By one episode I mean period when agent is initialised randomly within maze and it tries to reach goal. So learning process consists of many episodes. Once the learning is over and the final policy is evaluated I call it the end of experiment. I repeat several experiments to determine mean convergence time and confidence intervals.

1 Function Approximation Discussion

According to the lectures Q function can be approximated by k basis function in the following expression:

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_k f_k(s,a) = w^T f(s,a) \tag{1}$$

$$\text{s - state, a - action} \tag{2}$$

For this question and following ones our basis functions can be partitioned into this product:

$$f_i(s,a) = f_i(s) f_i(a) \tag{3}$$

In the expression above each i'th basis function which depends only on action is

$$f_i(a) = I(a = a')$$

where $I(a = a')$ is identity/indicator function of particular action. In the assignment we use 25 basis functions which depends only on the state $f_i(s)$ and also we have 5 different actions which means we need 5 action basis functions $I(a = a')$ and therefore by multiplying these domains we will get the following number of general basis functions:

$$|f_i(s,a)| = |f_i(s)| * |f_i(a)| = 25 * 5 = 125 \tag{4}$$

and due to this we get that in 1 we have 125 weights overall or 25 weight per 5 actions. This can be simplified even further during the learning step (here I omit indexes which means that these are vectors):

$$\delta \leftarrow r + \gamma max_{a' \in A} w^T f(s', a') - w^T f(s,a) \tag{5}$$

$$w \leftarrow w + \eta \delta f(s,a) \tag{6}$$

$$\tag{7}$$

by using the fact that $f_i(a) = I(a = a')$ and noting that all actions which differs from action identity will make no contribution in the product $w^T f(s,a)$:

$$Q(s,a) = w1_a f_1(s) + w2_a f_2(s) + \ldots + w25_a f_{25}(s) = w_a^T f(s) \tag{8}$$

$$\delta \leftarrow r + \gamma max_{a' \in A} w_{a'}^T f(s') - w_a^T f(s) \tag{9}$$

$$w_a \leftarrow w_a + \eta \delta f(s) \tag{10}$$

$$\text{where each } w_a \text{ is 25 vector of weights which corresponds to particular action } a \tag{11}$$

Relying on the fact that our task is discrete one I precompute values of each 25 basis functions for each cell of maze in order to save computational time. Here how it looks approximately in my code (particular cases for identity functions $f_i(s) = I(s = s_i)$ or RBF are in the appendix) like this:

```
%...
%precompute basis functions in advance
% number of states
Sx = 10;  % width of grid world
Sy = 10;  % length of grid world
M = 2; % cover neighbouring area
%basis functions
F = zeros(Sx, Sy, 25);

%iterate through maze cells
for ix = 1:Sx
  for iy = 1:Sy
    %identity functions
    %iterate through 25 basis functions
    for kx = 1:5
      for ky = 1:5
        %basis function index (can be from 1 to 25)
        fId = 5 * (ky - 1) + kx;
        %Here it is identity functions
        %It can be RBF as well
        F(ix, iy, fId) = M*(kx-1) < ix && ix <= M*kx &&...
                         M*(ky-1) < iy && iy <= M*ky;
      end
    end
  end
end

%EPISODES/LEARNING LOOP GOES BELOW
%...
```

These give sufficient speed up, especially when using RBF (almost 20 times). Also, in terms of computation speed there is basically no difference whether identity functions or RBF are used as their precomputed before hand.

Additionally by analysing equations 8 it can be seen that it is possible to rewrite them as dot products between vectors and matrices. Code snippet which demonstrates dot products speed ups:

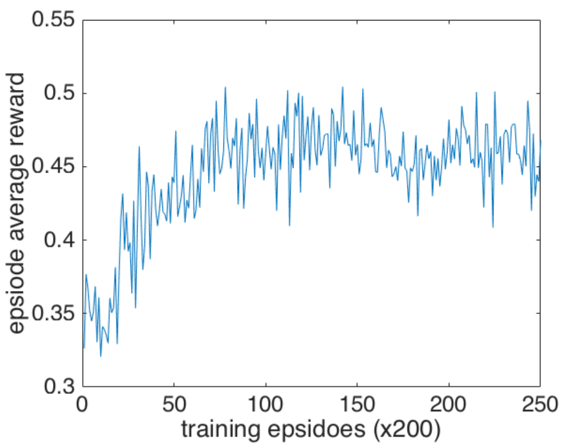
```
%...

%INITIALISATION
%number of actions
A = 5;
%weights matrix. For each actions keeps values of 25 basis functions
W = zeros(A,25);
%basis functions
F = zeros(Sx, Sy, 25);

%...

%INSIDE PARTICULAR EPISODE

%...

% choosing best action is the dot product
[V_s0, a0]=max(W * reshape(F(x0, y0, :), 25, 1));

%...

%learning
V_s1 = max(W * reshape(F(x1, y1, :), 25, 1));
%approximate Q(s0, a0) by dot product
Q_s0_a0 = W(a0, :) * reshape(F(x0, y0, :), 25, 1);
delta = r+gamma*V_s1 - Q_s0_a0;
%basic functions in state x0 y0
bfs = reshape(F(x0, y0, :), 1, 25);
%updating weights matrix
W(a0, :) = W(a0, :) + eta * delta * bfs;

%...
```

Dot products work much faster in contrast to usual for loops in MATLAB. That is why due to such implementation I am able to go trough 150000 episodes with RBF functions on 25 by 25 maze in 2 minutes.

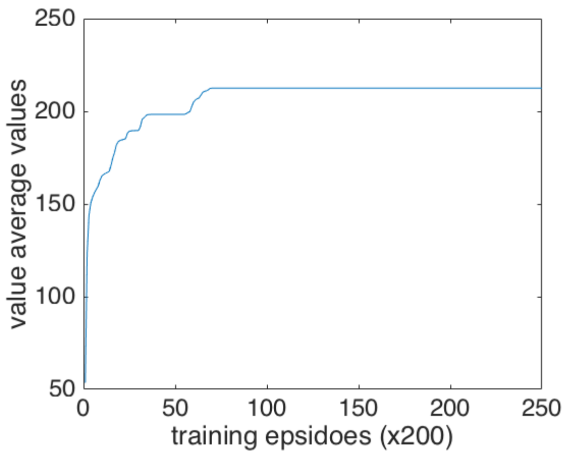
2 Identity functions representation in 5x5 maze

In this task we are using $f_i(s) = I(s = s_i)$ as state basis functions. First of all, let's see results achieved by true Q function (table representation) for this task. I have used the following learning parameters:

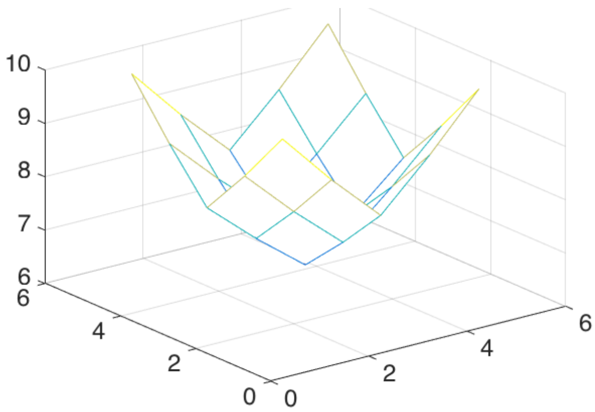
Learning rate η	0.2
Exploration rate ϵ	0.1
Discount rate γ	0.9;



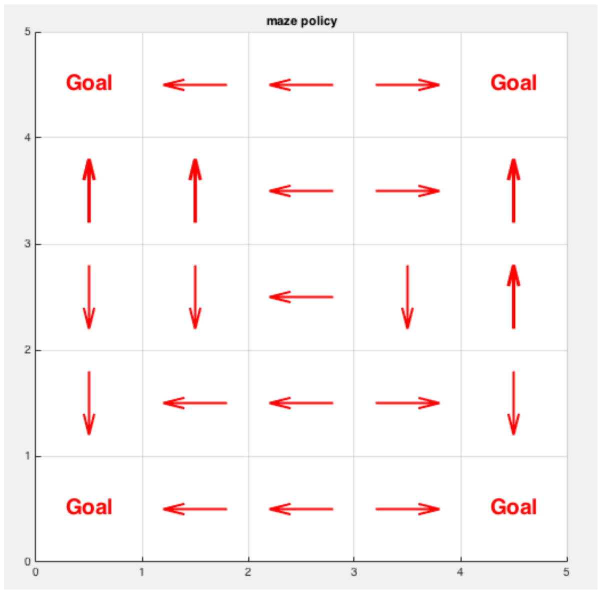
(a) Average reward



(b) Value function convergence



(c) Final value function on maze states



(d) Final policy

Figure 1: 5x5 maze. Table representation

In this problem our reward doesn't depend on how many steps agent did within one episode in order to reach reward (it is just 1 if agent has reached its goal). That is why reward in the end of one episode is not very representative value in terms of how well the learning process goes as it won't show how fast we reached our goal. Due to that I use average reward per episode, as the more time agent spent in the maze the smaller his average reward will be.

```
%...
%start episode
for u = 1:Sx*Sy
    %learning and acting in maze
    %...

    % goto next episode once the goal is reached
    if ismember([x0, y0], Goals, 'rows')
      avg_r = r/u;
      %display(avg_r, 'reached the goal!');
      break;
    end
end
%end of episode

%store average reward
avg_rs(t) = avg_r;
%...
```

All values below are averaged over 200 training episodes () to ease fluctuations due to random starting position and ϵ-greedy learning.

The average reward is presented on Figure 1a. It reaches approximately 0.45 which corresponds to the $1/0.45 = 2.222$ steps in the one episode on average. This is logical as in 5x5 maze with goals in the corners of the maze the average number of steps to reach goal should be between 2 and 3.

To determine dynamic of learning process more precisely I use value function $V(sx, sy)$. I sum up all entries of value function which corresponds to particular positions in the maze (we shouldn't have negatives ones because there are no rewards -> only 1 and 0) and rely on the fact that if entries of value function doesn't change then their sum doesn't change, as well. The situations when some entries declined by exactly the same amount other entries increased are extremely unlikely. Code snippet which is responsible for this:

```
%...
% after each episode:

%compute value function
VV = max(Q,[],3);%Size: (Sx,Sy)
%storing for future plotting
samples(t) = sum(reshape(VV, 1,numel(VV)));

%...

%after end of training
%averaging to remove most fluctuations
```

```
average_over = 200;
avg_samples = mean(reshape(samples, average_over, []));

%...
```

On Figure 1b we can see that values are almost not changed after approximately $160*200 = 32000$ episodes. I determine convergence by computing standard deviation of the averaged value function (over 200 episodes) with window 50 and after that I find the first spike in its values from the end. Such procedure allows to avoid determining convergence too early when value function reaches local maximum (on Figure 1b the local maximum is reached near $80 * 200 = 8000$). The disadvantage of this method that it doesn't allow to determine convergence online during learning that is why it is impossible to abort learning earlier. Additionally, especially for approximation with basis functions, value function reaches certain level and then starts to fluctuate around some particular value. That is why converges criteria must be adjusted in order to accommodate that (probably convergence criteria is not the best name for it) and its initial value is estimated through value function plots over episodes. Due to that plots must be shown together with these values. Ideally, convergence must be measured by comparing generated policy with optimal (which can handle several best actions) one for each certain amount of episodes. Code snippet responsible for this:

```
%...

%computing standard deviations over window 50
smp_stds = zeros(1,(size(avg_samples, 2) - 50));
for t=1:(size(avg_samples, 2) - 50)
  smp_std = std(avg_samples(t:(t+50)));
  smp_stds(t) = smp_std;%abs(avg_samples(t) - avg_samples(t-1))/↩
      avg_samples(t);
end

%find first spike bigger than 0.2 from the end
convg = find(smp_stds > 0.2, 1, 'last')
%convergence time
convg_t = convg * average_over
%if it is not empty then store in experiment array
%to compute confidence intervals
if ~isempty(convg_t)
  convg_times = [convg_times, convg_t];
end

%...
```

Here, I set criteria for standard deviation to be bigger than 0.2 starting from the end in order to determine convergence.

After repeating experiment for 10 time I got the following results for convergence with 95% confidence interval (from here on it is always 95% confidence interval):

Convergence criteria (STD)	0.2
Number of experiments	10
Mean convergence time	15480 ± 2462 episodes

Finally, on Figure 1c we can see value function $V(sx, sy)$ in the end of learning (in the end of one of the experiments) for each of maze cells. It has symmetric convex shape with 4 peaks located exactly in the goals positions in the corners of the maze as it would be expected. On Figure 1d we can see the final generated policy (after the end of learning process for each maze state I choose actions absolutely greedily). It is optimal one because in this task there are many equally optimal policies given that sometimes distances to goals are the same and there is several best action. Overall, this indicates that learning has been successful.

Now let's compare above results with the exactly same task where Q functions was approximated by 25 basis identity state functions $f_i(s) = I(s = s_i)$ I have used the same learning parameters:

Learning rate η	0.2
Exploration rate ϵ	0.1
Discount rate γ	0.9;

I have received the following convergence results:

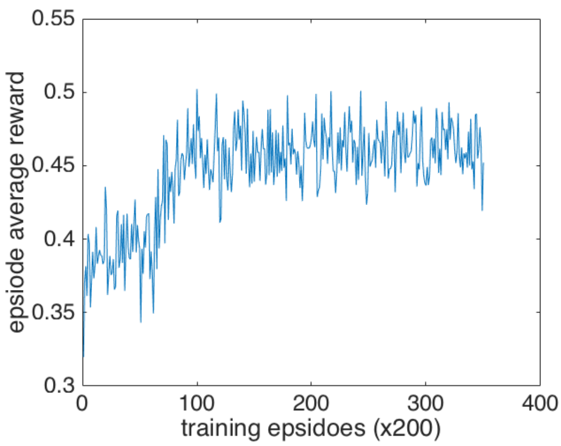
Convergence criteria (STD)	0.2
Number of experiments	10
Mean convergence time	16220 ± 1475 episodes

Within its confidence intervals this results matches the convergence time obtained for tabular representation.

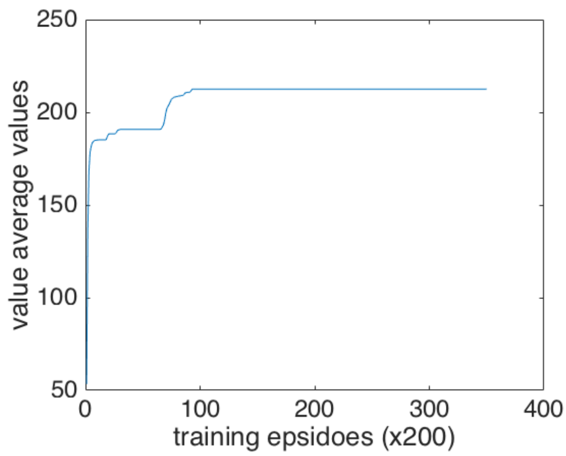
When basic function approximation is applied the approximated value functions can be computed from weights $W(A,\ number\ of\ basis\ functions)$ and basis functions $F(Sx, Sy,\ number\ of\ basis\ functions)$ using the following code snippet:

```
% reshape(F, Sx * Sy, 25) % Sx*Sy x 25
% W' %  25 x A
QQ = reshape(F, Sx * Sy, 25) * W'; % (Sx*Sy x 25) * (25 x A) = Sx*Sy x A
VV = max(QQ,[],2); %Sx*Sy
%approximated value function
VV = reshape(VV, Sx, Sy);
```

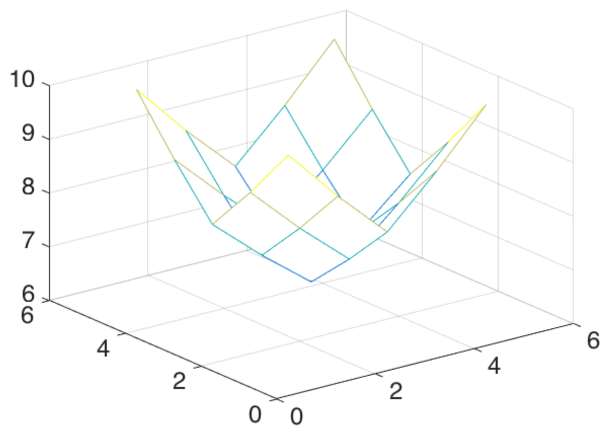
Next, Figures 2a, 2b, 2c are in good correspondence with its analogues from tabular representation on Figure 1. This is expected as when we are using 5x5 Identity functions for 5x5 maze



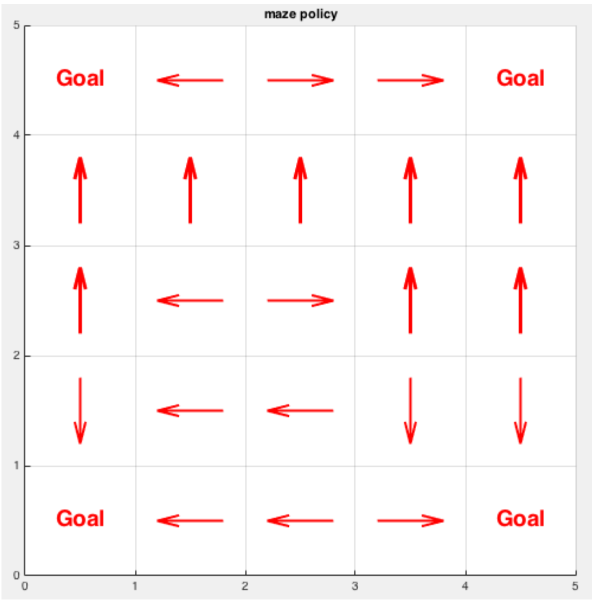
(a) Average reward



(b) Value function convergence



(c) Final value function on maze states

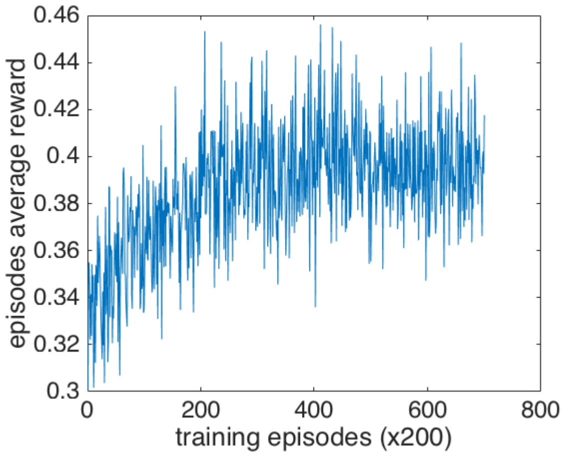


(d) Final policy

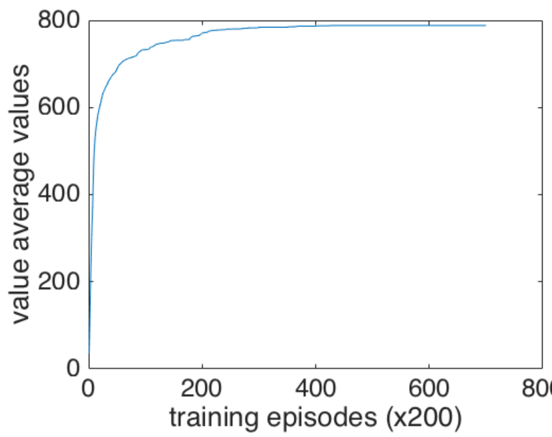
Figure 2: 5x5 maze. Identity Functions $f_i(s) = I(s = s_i)$ representation

the tabular representation and basis functions representation should be essentially the same. This can be seen by comparing the number of parameters to be learn. In tabular representation we learn the Q values $|Q(Sx, Sy, A)| = 5 * 5 * 5 = 125$. In basis functions representation we learn weight matrix from Equation 8 -> $|W(A, number\ of\ basis\ functions)| = 5 * 25 = 125$. The final policy depicted on Figure 2d differs from the one on Figure 1d but it is still optimal policy and the difference between them can be described by the fact that in this maze problem there are many equally optimal policies.

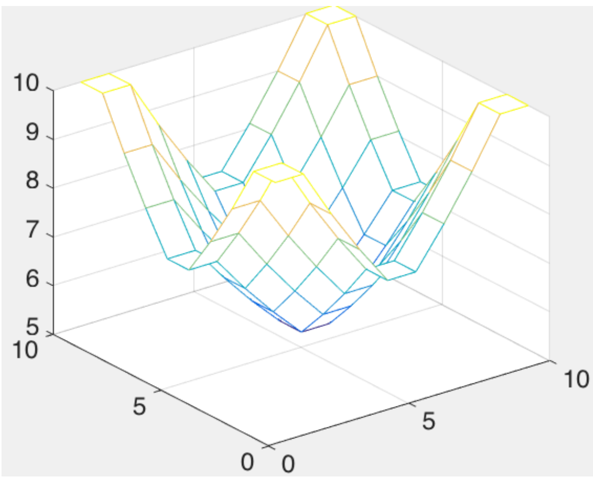
3 Identity functions representation in 10x10 maze



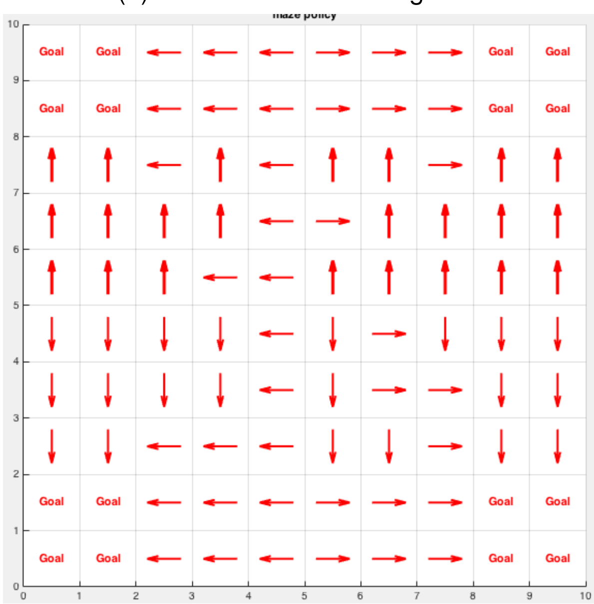
(a) Average reward



(b) Value function convergence



(c) Final value function on maze states



(d) Final policy

Figure 3: 10x10 maze. Table representation

In this question identity basis functions evenly cover patches 2x2 of maze cells. It has been already shown how state identity functions are computed in that case in the first code listing of question 1.

Here, I redefine goals as patches 2x2 placed in the corners of the maze because in this representation agent doesn't have enough information to reach final goal reliably once it is within goal patch 2x2. It happens because for all closest neighbours around goal states the same identity functions will be activated.

First, we compute true results in tabular representation of Q function for maze 10x10. For that I have used the same learning parameters as in previous question:

Learning rate η	0.2
Exploration rate ϵ	0.1
Discount rate γ	0.9;

The converges results are:

Convergence criteria (STD)	0.2
Number of experiments	10
Mean convergence time	86240 $\pm$ 4727 episodes

The convergence time is 5 times bigger than for the case of 5x5 maze which roughly matches the fact that there are 4 times more states to explore ($10 * 10 = 100$ in contrast to $5 * 5 = 25$ for the 5x5 maze).

On Figure 3a the average reward per episode has almost the same mean as on Figure 1a but it has significantly bigger variance in results. This can be described by the fact that now the maze has become bigger and average path to nearest the goal has become longer. However, the chances of appearing on the goal cell from the beginning has remained the same. It was $4/25 = 0.16$ for 5x5 maze and now for 10x10 maze $4 * 4/100 = 0.16$. That is why starting point plays bigger role which leads to bigger variance in average reward per episode.

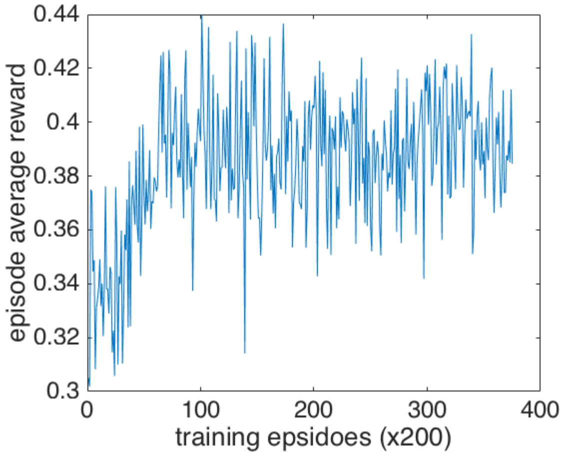
On Figure 3c we can see Value Function shape for 10x10 where goals are patches 2x2 in the corners. Once again it has 4 peaks exactly in corners locations. The policy shown on Figure 3d is also one of optimal ones.

Now, let's run experiments for identity functions approximation. It uses the same learning parameters as table representation above.

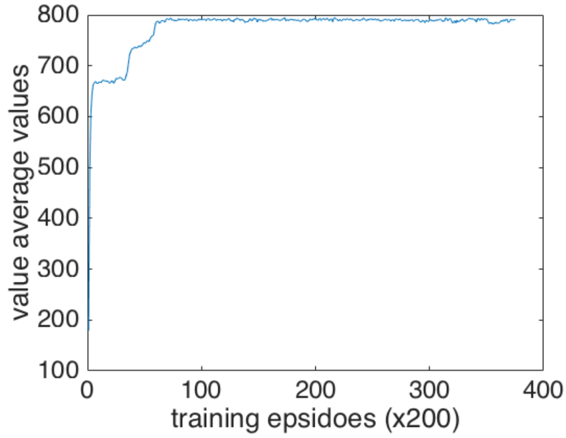
On Figure 4b we can see that values fluctuate stronger than in case of table representation on Figure 3b or identity basis functions representation for 5x5 maze on Figure 2b. This can be explained by the fact that now it is only approximation of real value function on Figure 3c as number of learning parameters is 4 times lower in that case ($|Q(Sx, Sy, A)| = 10 * 10 * 5 = 500$ for tabular representation in comparison to the same $|W(A,\ number\ of\ basis\ functions)| = 5 * 25 = 125$ for function approximation). Due to that I relaxed convergence criteria and the convergence results are:

Convergence criteria (STD)	6.0
Number of experiments	10
Mean convergence time	13200 $\pm$ 1331 episodes

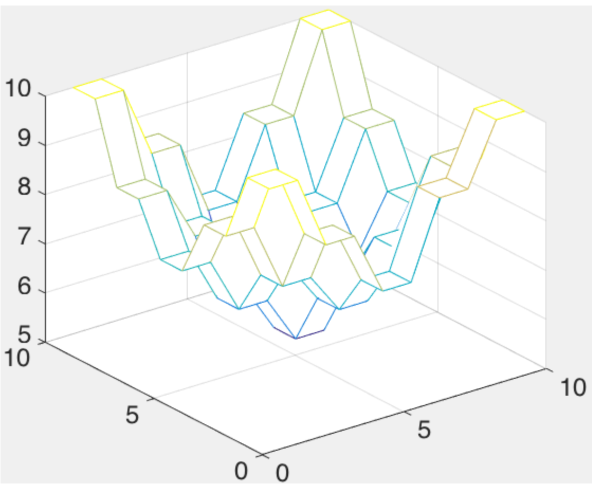
This time is essentially the same as for the case of identity basis functions approximation for 5x5 maze (16220 $\pm$ 1475 episodes). Again, it can be explained by the fact that in 5x5 maze



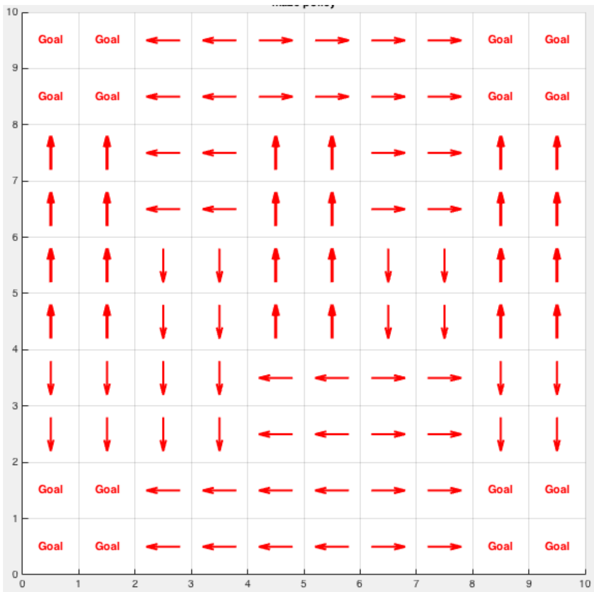
(a) Average reward



(b) Value function convergence



(c) Final value function on maze states



(d) Final policy

Figure 4: 10x10 maze. Identity basis functions representation

and 10x10 for identity basis functions approximation we have the same number of parameters to learn ($|W(A, number\ of\ basis\ functions)| = 5 * 25 = 125$). Here, we start to see the advantage of using basis functions approximation as convergence time is approximately 6 times faster than in plain tabular representation.

One of the most notable difference in comparison to table representation is on Figure 4c where the value functions is essentially 2 dimensional step function. This is logical as each 2x2 patch in the maze corresponds to the same 5 weights (for each action) in basis functions approximation. That is why once the maximum by action is taken then the 2x2 patches have the same value. Moreover, the policy generated on Figure 4d is optimal one and if you look carefully you will notice that each 2x2 patch shares the same actions. This is again due to the same reason that each 2x2 patch in the maze corresponds to the same 5 weights (for each action) in basis functions approximation.

4 Radial Basis Functions

For a Gaussian shape like basis functions I have used smoothed RBF kernel functions from the lectures as they proved to be more stable in contrast to usual Gaussians (it is easier to set their width parameter). Because I tried it myself and I saw in other people simulations when plain Gaussians are used then one goal becomes prevalent in comparison to all other. That is why many people just run their simulation for one goal in one of the corners at this stage. For smoothed RBF basis functions values must sum up to on for each state in the maze. They are precomputed in the following code snippet:

```
%...
%precompute basis functions in advance
% number of states
Sx = 10;  % width of grid world
Sy = 10;  % length of grid world
M = 2; % cover neighbouring area
%basis functions
F = zeros(Sx, Sy, 25);

%iterate through maze cells
for ix = 1:Sx
  for iy = 1:Sy
    %identity functions
    %iterate through 25 basis functions
    for kx = 1:5
      for ky = 1:5
        %basis function index (can be from 1 to 25)
        fId = 5 * (ky - 1) + kx;
        %position of rbf in maze coordinates
        cx = kx*M - (M - 1)/2;
        cy = ky*M - (M - 1)/2;
```

```
        E = exp(-(1/(2*SIGMA^2))*((ix-cx)^2+(iy-cy)^2));
        F(ix, iy, fId) = E/Z;
      end
    end
    Z = sum(F(ix, iy, :));
    %smoothing
    F(ix, iy, :) = F(ix, iy, :) / Z;
  end
end

%EPISODES/LEARNING LOOP GOES BELOW
%...
```

Note here that basis functions are centered in the middle of MxM patches (by subtracting $(M-1)/2$). Also the Gaussian centers are computed in maze space that is why parameter SIGMA which corresponds to the width of RBF is also measured in maze space.

For this representation I still use wider patches for goals in the corners of the maze because as it was said in the previous question we may need to redefine goal and here we are still having approximation although much better one than simple identity functions. These wider goal patches allows algorithm to start convergence much faster especially on few initial steps in 25x25 maze as it finds goal with much higher probability (as there are more goals overall).

I use the following learning parameters unless it is said otherwise

Learning rate η	0.2
Exploration rate ϵ	0.1
Discount rate γ	0.9;

For the 5x5 maze using RBF with width set to 0.5 I get the following convergence time result:

Convergence criteria (STD)	0.2
Number of experiments	10
Mean convergence time	11900 $\pm$ 1077 episodes

The convergence time is slightly faster in comparison to results obtained in question 2 (we can compare them because convergence criteria is same as in question 2). This potentially can be described by the fact RBF gives additional positional information on early stages so inference can be done better. For example, let say we are located in position (1,1) in the 5x5 then values of my RBF functions will be:

0	1	2	3	4	5
1	0.7753	0.1049	0.0003	0.0000	0.0000
2	0.1049	0.0142	0.0000	0.0000	0.0000
3	0.0003	0.0000	0.0000	0.0000	0.0000
4	0.0000	0.0000	0.0000	0.0000	0.0000
5	0.0000	0.0000	0.0000	0.0000	0.0000

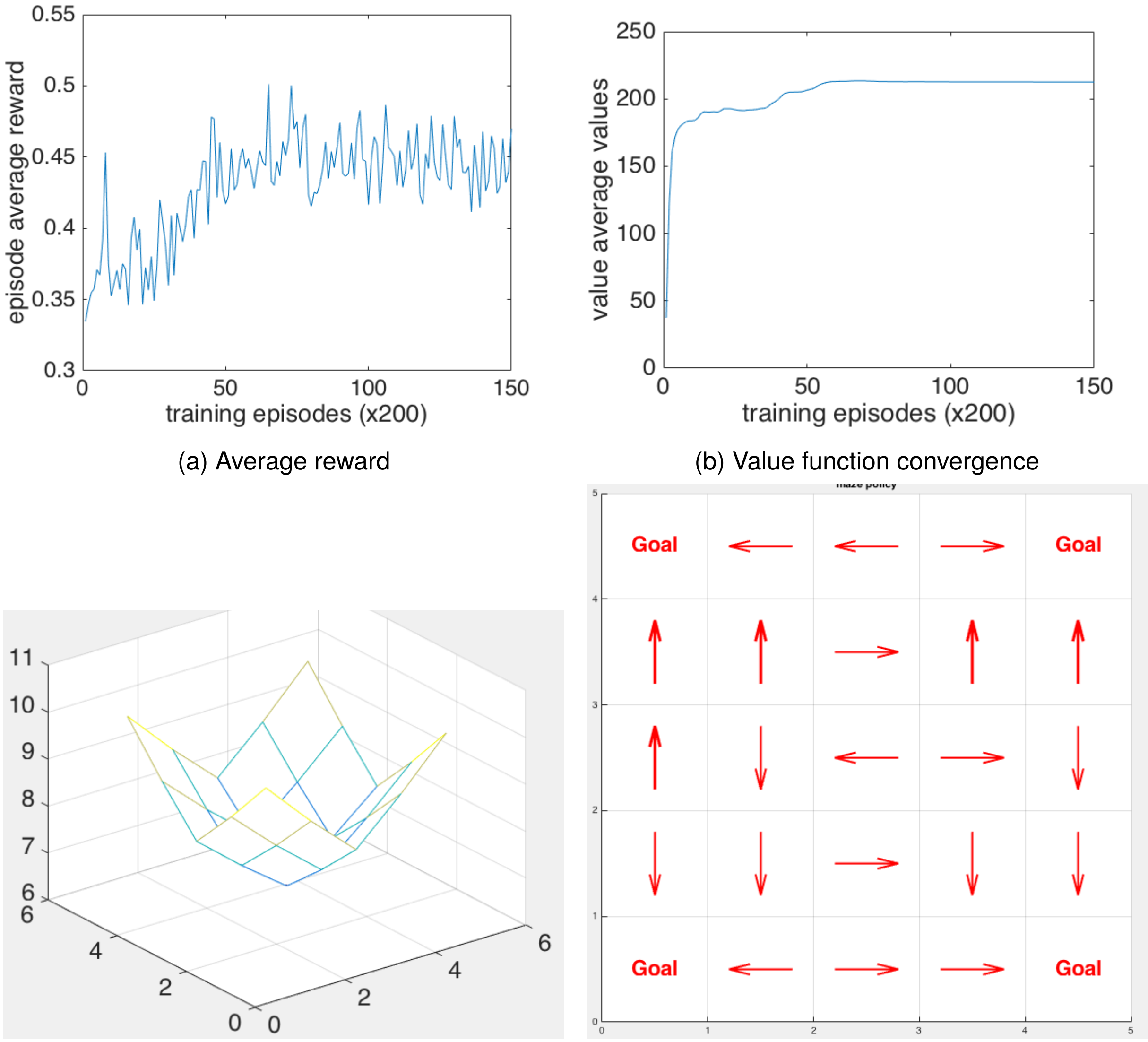
As we can see the highest values gives RBF functions located in (1,1). Next, much smaller ones are located in (1,2) and (2,1). After that we have a few tiny values and rest is zero.

Another example, if we are located in the middle of the maze (3,3):

0	1	2	3	4	5
1	0.0000	0.0000	0.0002	0.0000	0.0000
2	0.0000	0.0113	0.0837	0.0113	0.0000
3	0.0002	0.0837	0.6187	0.0837	0.0002
4	0.0000	0.0113	0.0837	0.0113	0.0000
5	0.0000	0.0000	0.0002	0.0000	0.0000

We have high value 0.6187 in the (3,3) and after that all other values goes down symmetrically from it.

All the other results for 5x5 maze on Figure 5 are identical to Figures 1 and 2 in question 2.



(a) Average reward

(b) Value function convergence

(c) Final value function on maze states

(d) Final policy

Figure 5: 5x5 maze. RBF representation with width 0.5

Essentially for RBF when the width (SIGMA) shrinks it becomes more and more peaked. In that case for 10x10 and 25x25 mazes value functions should become a step function over 2x2 and 5x5 patches once width shrinks to certain value.

For the 10x10 maze experiment with width (SIGMA) equals to 1.0 the value function is never converged under convergence criteria of 6.0 for 100000 episodes due to relatively high fluctuation as it can be seen on Figure 6b. That is why I increased that criteria to 25.0 but in that case it is not quite right to compare convergence results with previous questions. However, as it can be seen on Figure 6b that after approximately $200 * 200 = 40000$ episodes value function reach a certain level and it keeps fluctuating around it.

Here is the convergence results for width (SIGMA) 1.0 10x10 maze:

Convergence criteria (STD)	25
Number of experiments	10
Mean convergence time	45840 $\pm$ 8467 episodes

As you can see here the confidence interval is really large which is in agreement with Figure 6b as fluctuations are quite high in comparison to attained level of value function. But this convergence time is almost 2 times faster than convergence time for table representation in 10x10 maze (It was 86240 $\pm$ 4727 episodes)

As it can be seen on Figure 6c the resulting shape value function approximation is quite similar to value function in the table representation on Figure 3. The produced policy on Figure 6d is an optimal one as well.

After that I have changed width (SIGMA) to value of 0.5 which makes RBF functions really peaked and very similar to identity functions. That is why here I used 6.0 as convergence criteria from question 3. Here is the convergence results for width (SIGMA) 0.5 on 10x10 maze:

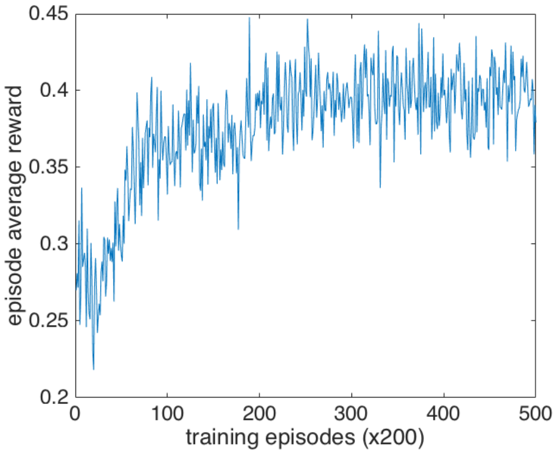
Convergence criteria (STD)	6.0
Number of experiments	10
Mean convergence time	15240 $\pm$ 1172 episodes

This result within confidence interval is identical to convergence time achieved in 10x10 maze with identity functions in question 3.

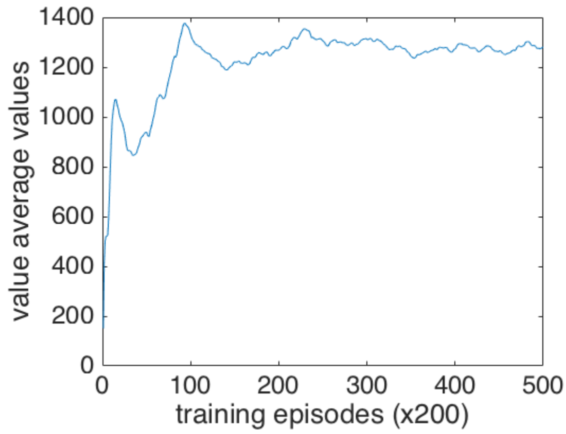
This is also confirmed by Figure 7. On Figure 7c we have value function which has almost step function shape similarly to Figure 4c. Also the policy shown on Figure 7d has actions only in groups by 2x2 patches as on Figure 4d.

For 25x25 case for table Q representation my simulation didn't converge even within 250000 learning episodes. The obtained result for that simulation are present on Figure 8. The resulting policy on Figure 8c almost optimal but there is few states for which action are chosen not optimally.

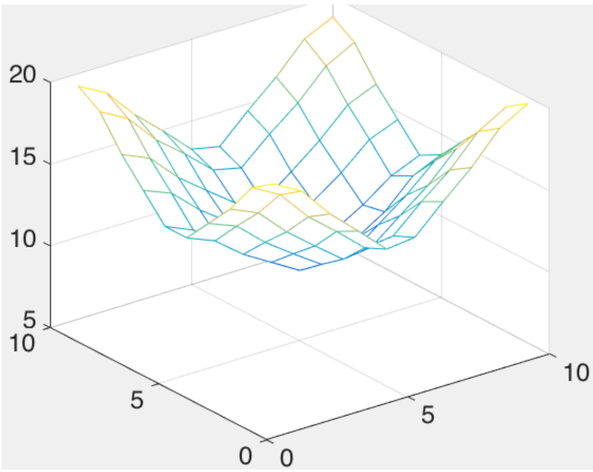
Now let's run simulation for RBF functions in 25x25 maze. It catches general pattern fairly quickly but it fluctuates a lot as it can be seen on 9b. The important distinction why it converges in contrast to Figure 8b is that on Figure 9b it stops growing at some point and resulting generated policy on Figure 8 is an optimal one. The received shape of value function on Figure 9a is similar to Figure 8a but it is more smooth one. By setting convergence criteria to the level of fluctuation on Figure 9b the following convergence time was computed:



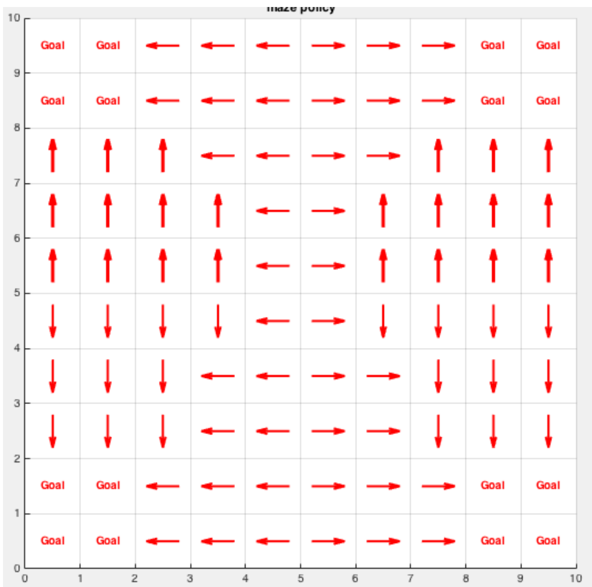
(a) Average reward



(b) Value function convergence

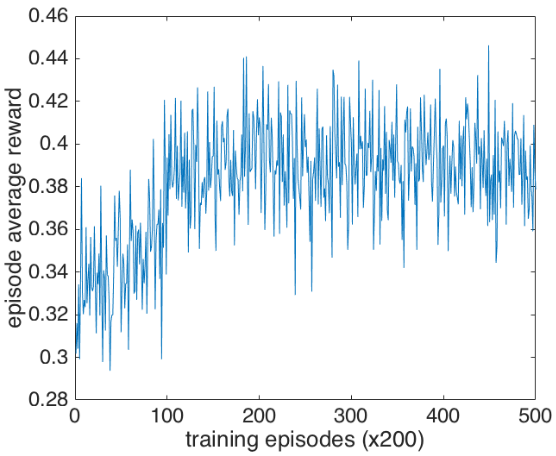


(c) Final value function on maze states

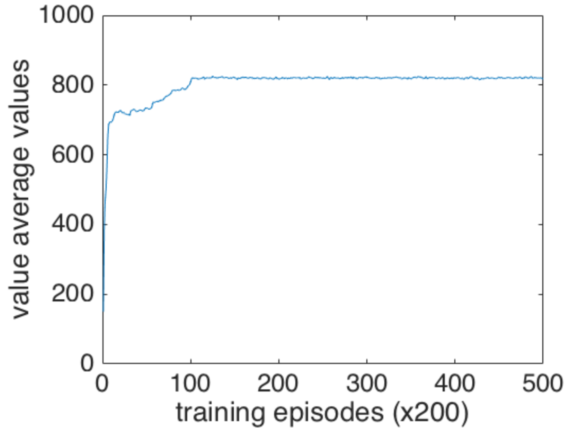


(d) Final policy

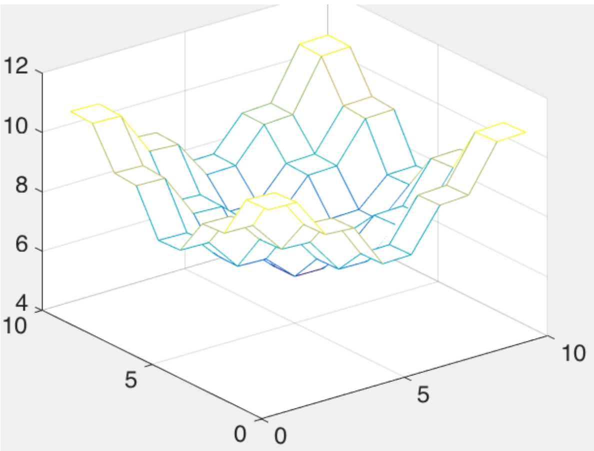
Figure 6: 10x10 maze. RBF representation with width 1.0



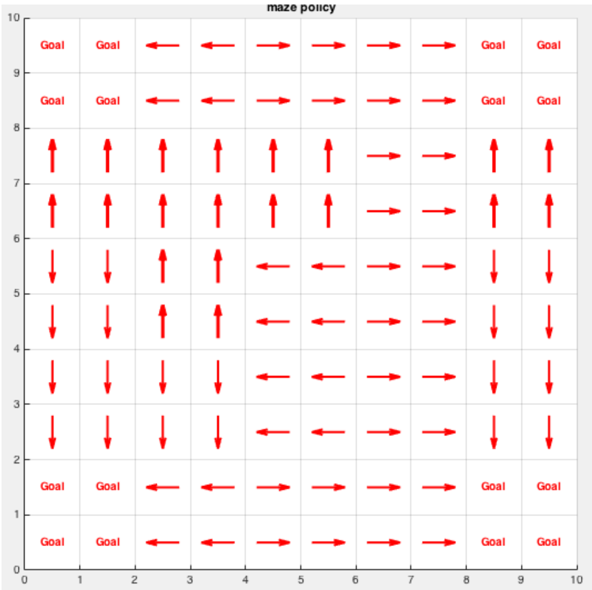
(a) Average reward



(b) Value function convergence

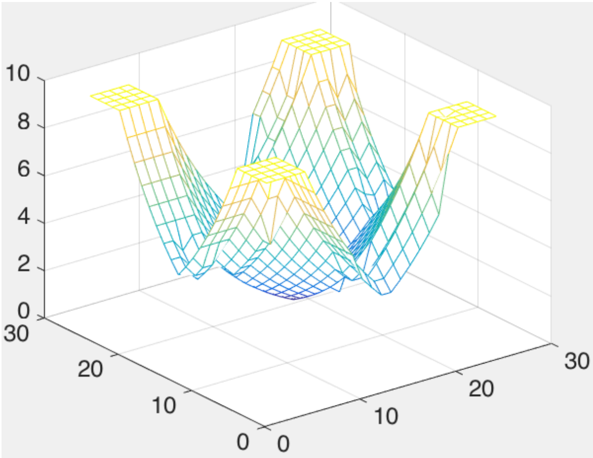


(c) Final value function on maze states

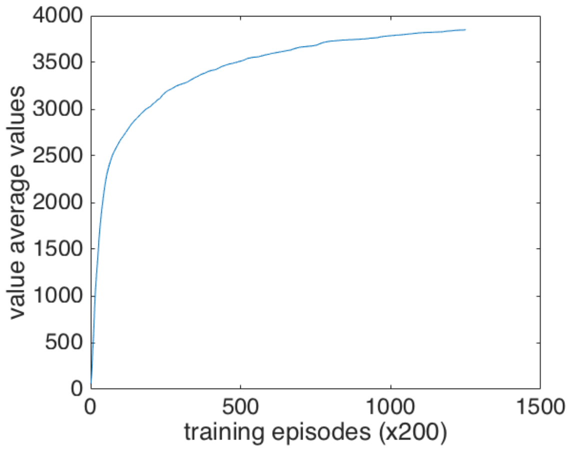


(d) Final policy

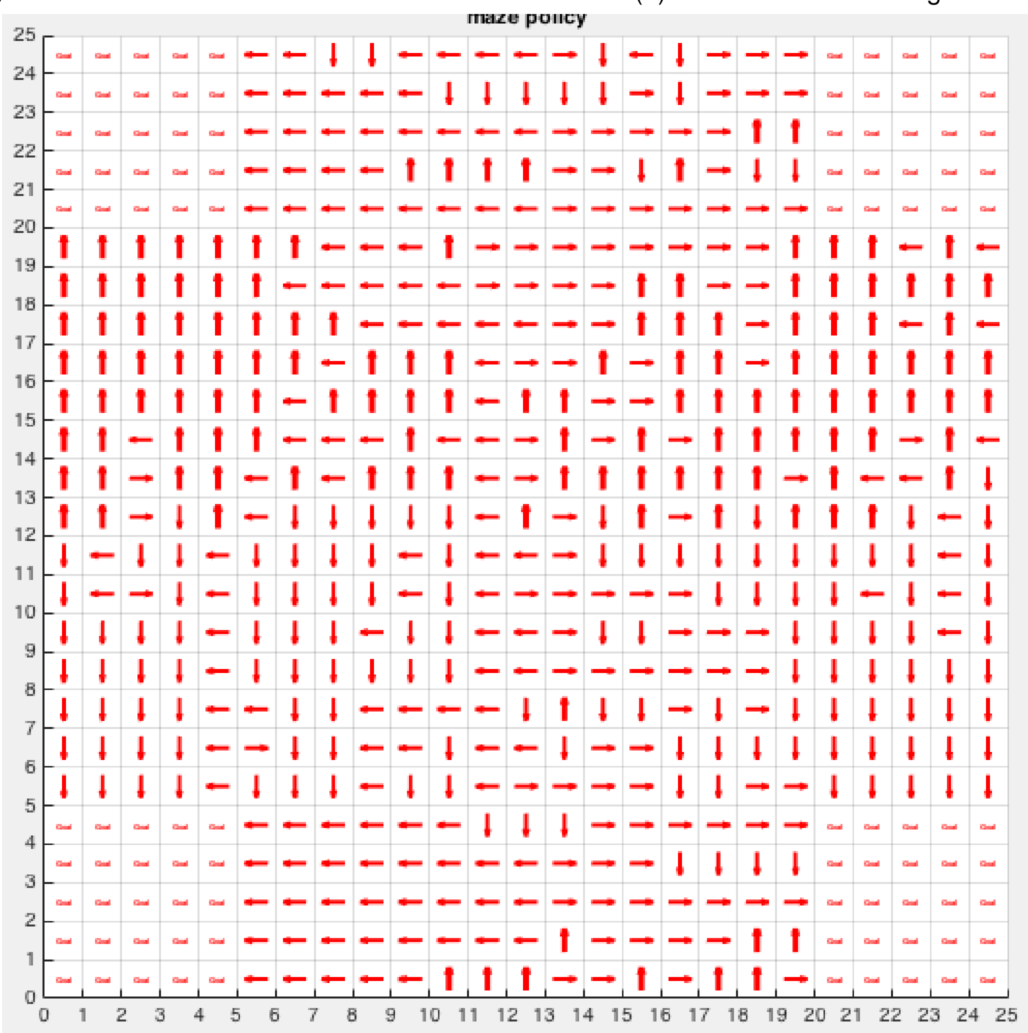
Figure 7: 10x10 maze. RBF representation with width (SIGMA) 0.5



(a) Final value function on maze states



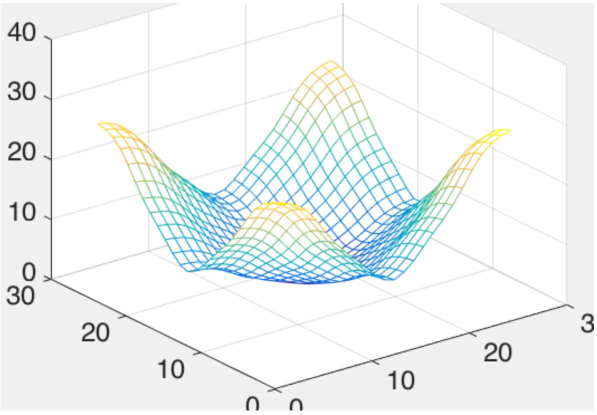
(b) Value function convergence



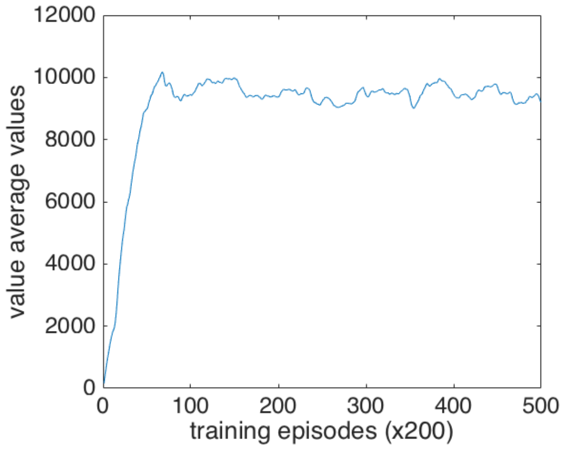
(c) Final policy. Not completely optimal one

Figure 8: 25x25 maze. Table representation

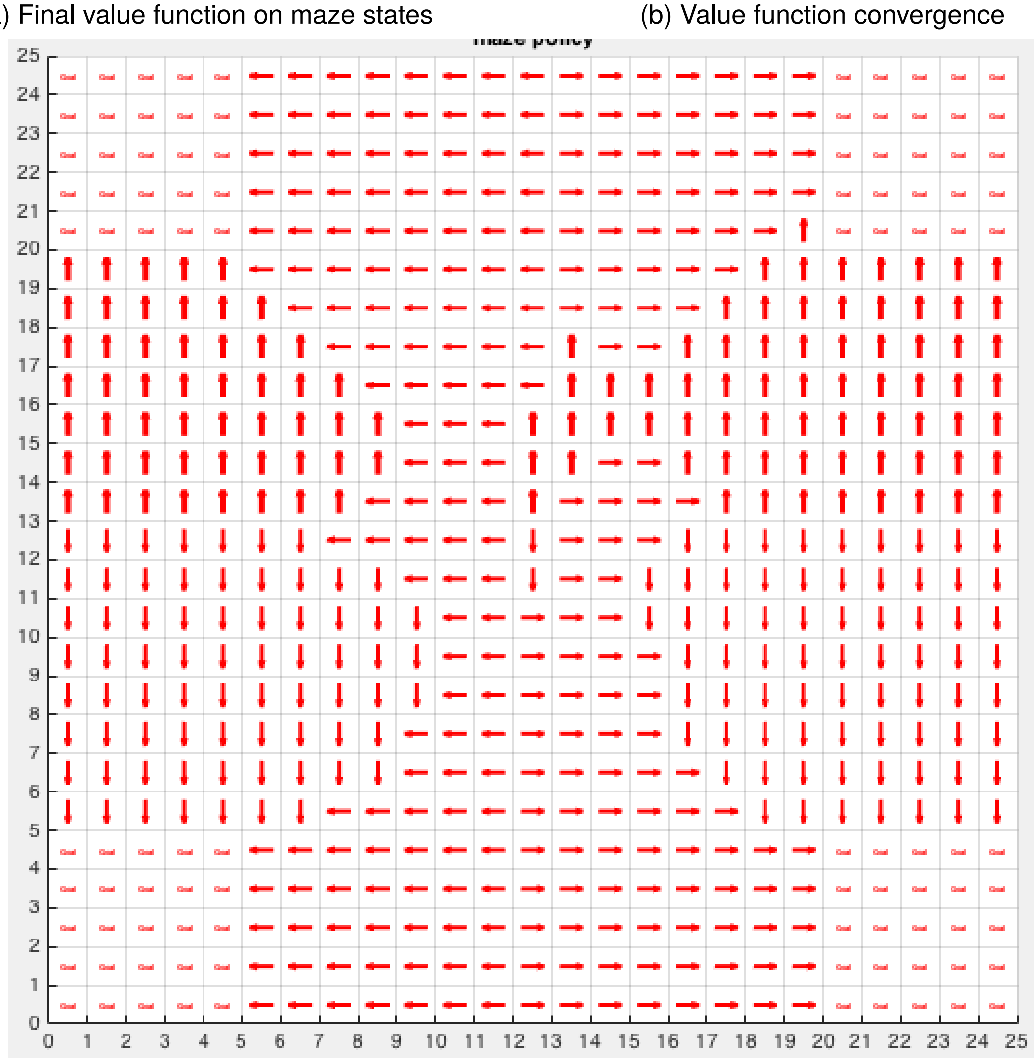
Convergence criteria (STD)	300
Number of experiments	10
Mean convergence time	15540 $\pm$ 2556 episodes



(a) Final value function on maze states



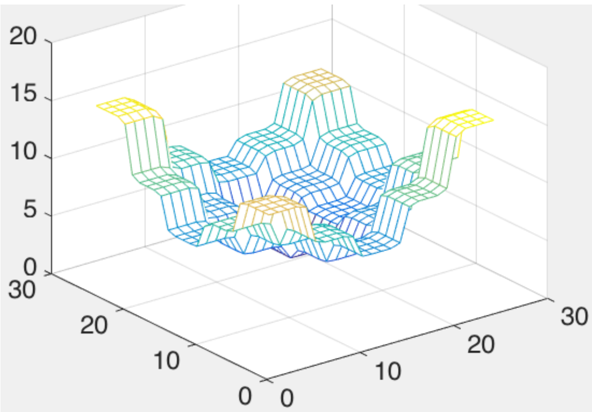
(b) Value function convergence



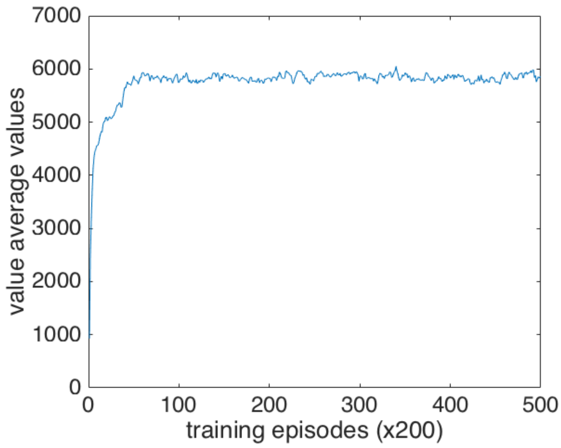
(c) Final policy

Figure 9: 25x25 maze. RBF representation with width (SIGMA) 2.5

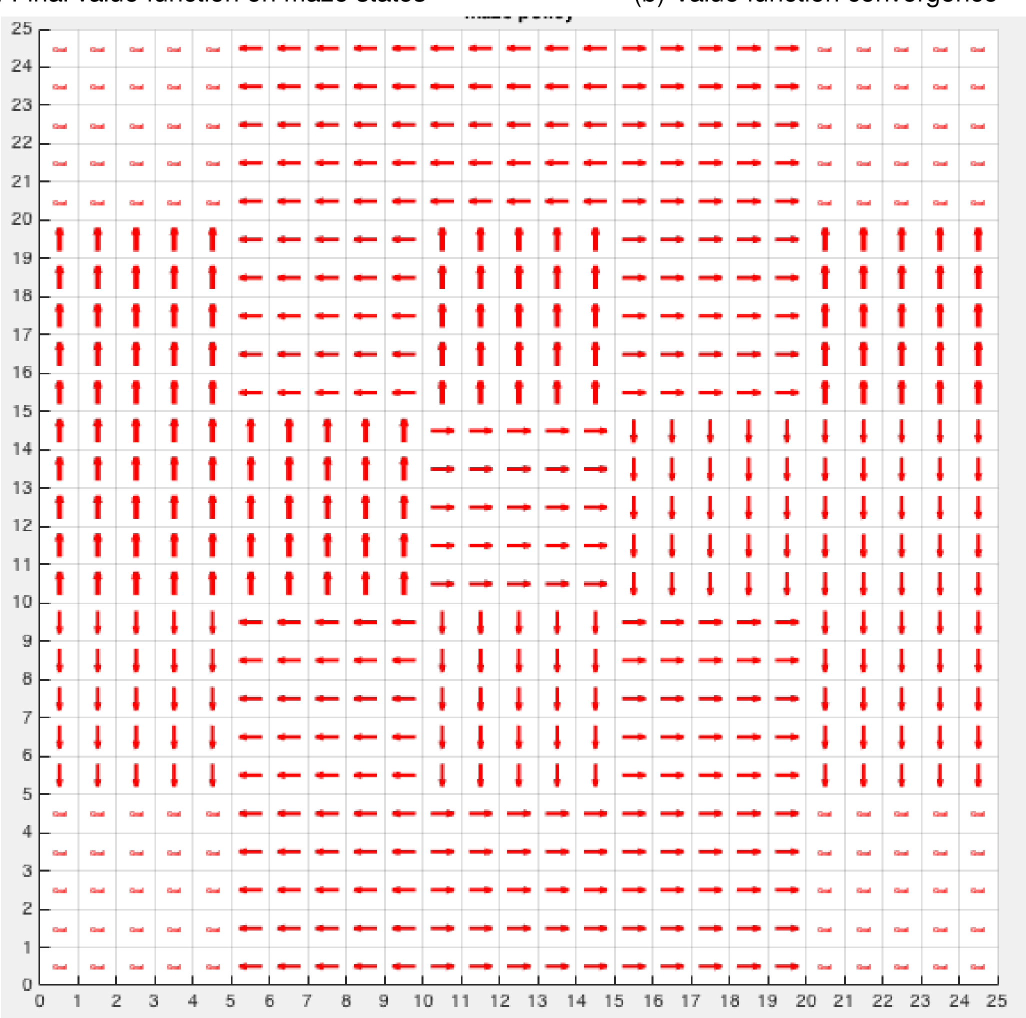
Now, let set smaller width in order to get more peaked distribution and we should get kind of step function. So for 25x25 maze I set width (SIGMA) and the results I got are on Figure 10. As we can see on Figure 10a it looks like a step function and it is also confirmed by policy on Figure 10c as actions are the same for patches 5x5. Overall, we can conclude that setting small width (SIGMA) relative to patch size results in highly peaked basis functions which leads to almost identity functions approximation.



(a) Final value function on maze states



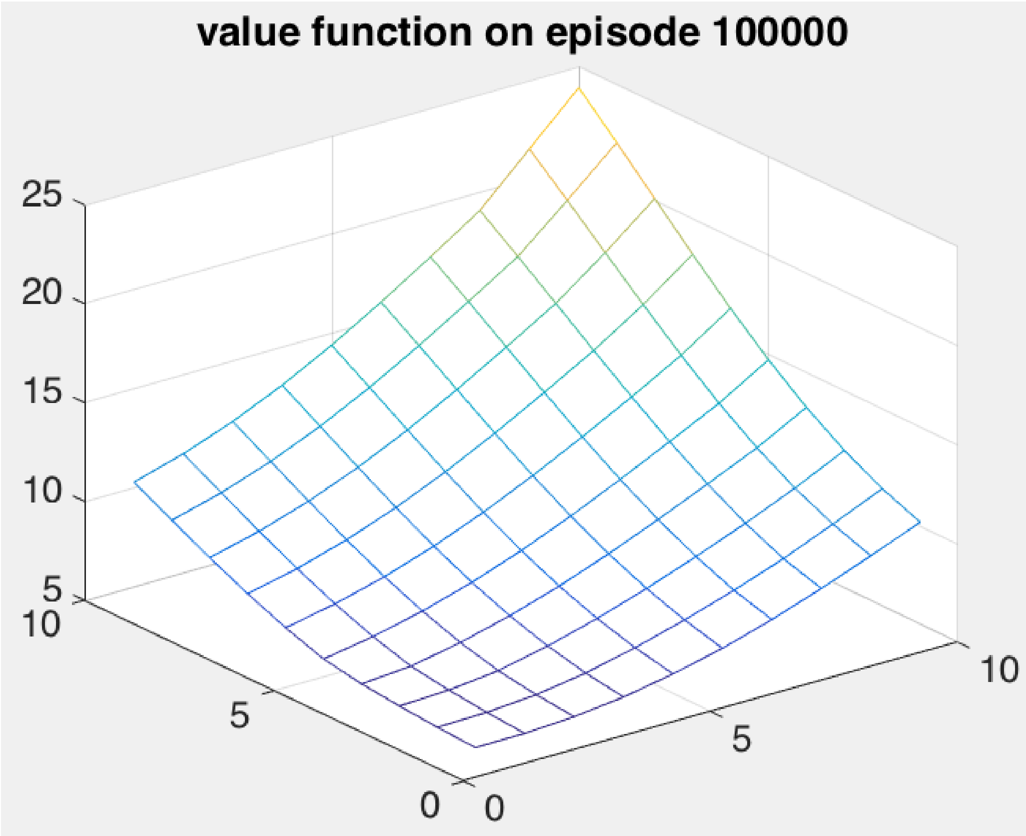
(b) Value function convergence



(c) Final policy

Figure 10: 25x25 maze. RBF representation with width (SIGMA) 1.0

Also for the case of very large width (SIGMA) 4.0 the produced value function has the following shape:



(a) Value function with RBF for width (SIGMA) 4.0

As width (SIGMA) 4.0 is very large in comparison to the size of the maze it means that basically RBF produce everywhere the same values as they are broad and therefore they are able to cover the whole maze. That is why very soon one solution dominates all other. Due to the same instability usual non-smoothed RBF are not suitable for this task as they produce result similar to Figure 11a for all almost any width (SIGMA).

For the final part I have tried different exploration rate (greedy ϵ values) for RBF approximation on the 10x10 maze with the following parameters:

Learning rate η	0.2
Discount rate γ	0.9
Width (SIGMA)	1.0
Convergence criteria (STD)	25
Number of experiments	10

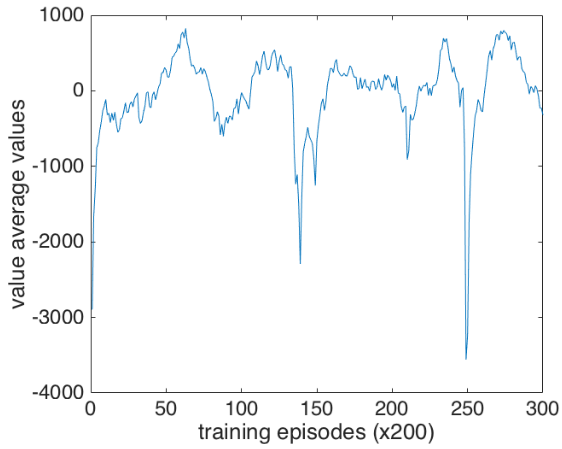
After running simulation for fixed ϵ values 0.1, 0.2 and 0.4 I got these results

Exploration rate ϵ	Convergence time
0.1	45840 ± 8467
0.2	23620 ± 4008
0.4	14740 ± 3409

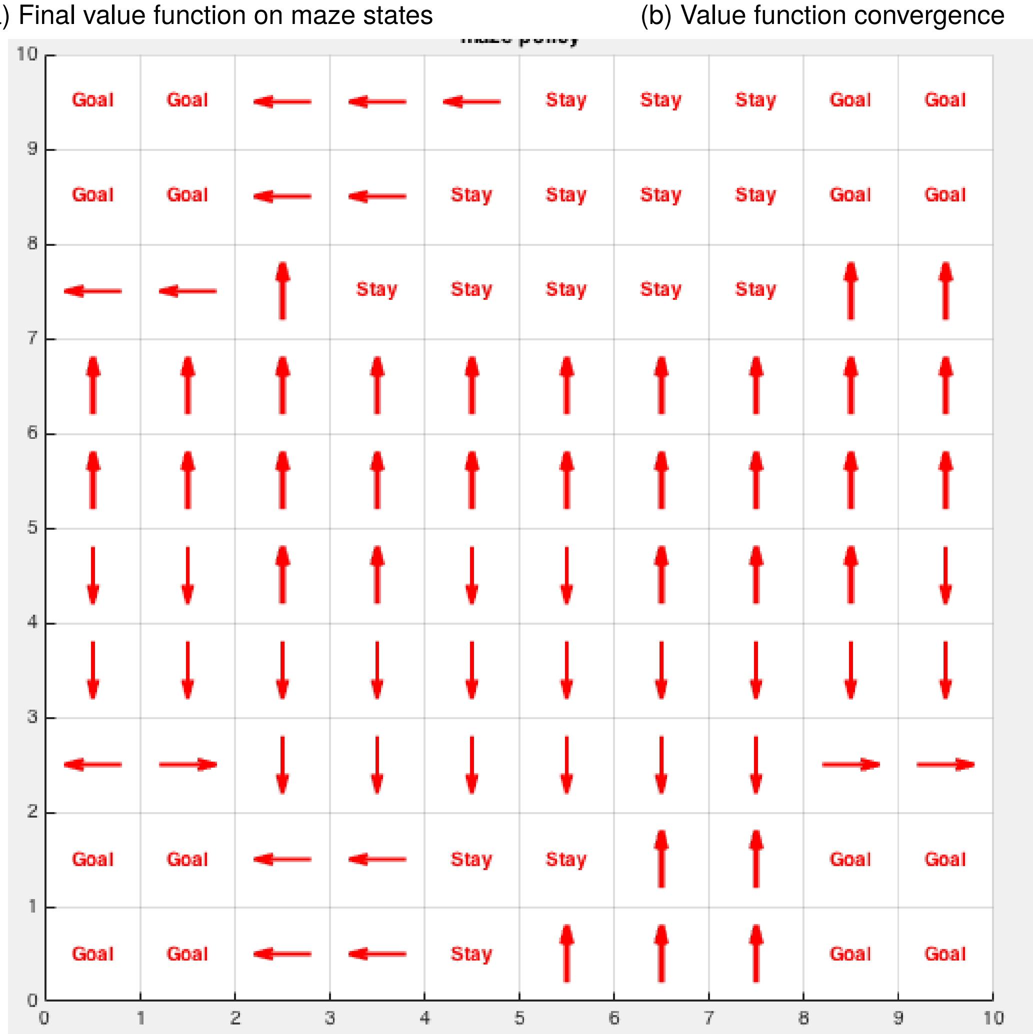
According to this table approximated value function convergence happens faster for bigger exploration rates.

5 Obstacles

(a) Final value function on maze states



(b) Value function convergence



(c) Final policy for maze with cliff

Figure 12: 10x10 maze with cliff. RBF representation with width (SIGMA) 1.0

The learning parameters for this task:

Learning rate η	0.2
Exploration rate ϵ	0.1
Discount rate γ	0.9;

As my obstacles I have created a cliff (reward -10) through the middle of the maze which splits world in two parts. Now the agent must avoid that cliff as punishment significantly out weights reward. However, the agent appears only on one side of the cliff that is why potentially it always has an opportunity to avoid cliff and reach the goal. But, it seems that RBF doesn't allow agent to avoid cliff reliably enough that is why especially on first several episodes it spends a lot of time in the maze. The results are presented on Figure 12. Figure 12c showing policy is especially interesting as agent learns to stay in some states. This can be explained by the fact that agent doesn't receive any discount if it stays on one place in contrast to stepping into cliff which is very dangerous. The only opportunity to escape states where agent is learnt to stay is ϵ greedy exploration. But still judging by Figure 12b task becomes very unstable and it doesn't seem to converge. However, if we run simulation for table representation of the task it will produce more symmetric value for the value functions although values will be much smaller than usual as discount for steeping into the cliff is too big. This is shown on Figure 13

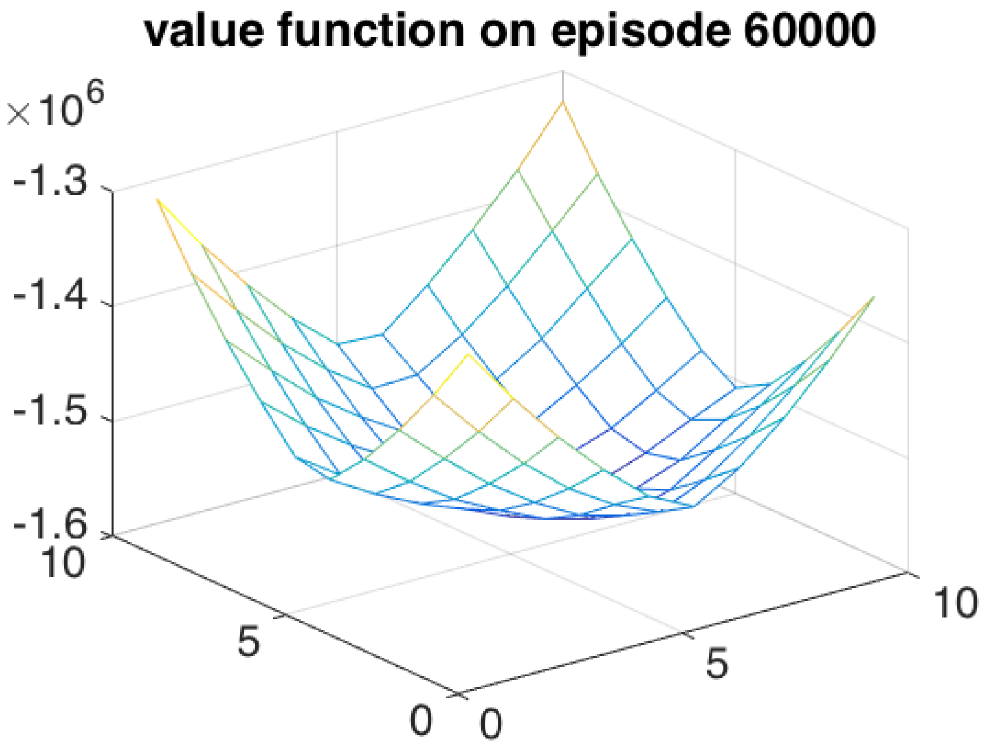


Figure 13: Final value function for 10x10 maze with obstacles using table representation of Q

Finally I tried to run simulation with the same conditions but in the case of 25x25 maze. Algorithm ran very slowly because agent basically learnt to stay on the same place as it is more safe as it doesn't have any discount for that. That is why I has changed the rules by prohibiting stay action and I also reduced the punishment for stepping into cliff to -1. After that my value functions converged to Figure 14 which essentially has the same shape as in prevous questions. It turned out that RBF representation is very sensible to the punishment reward. If punishment becomes lower than -1 then agent prefer to stay on the same place instead of acting and episodes take extremely long time to be completed.

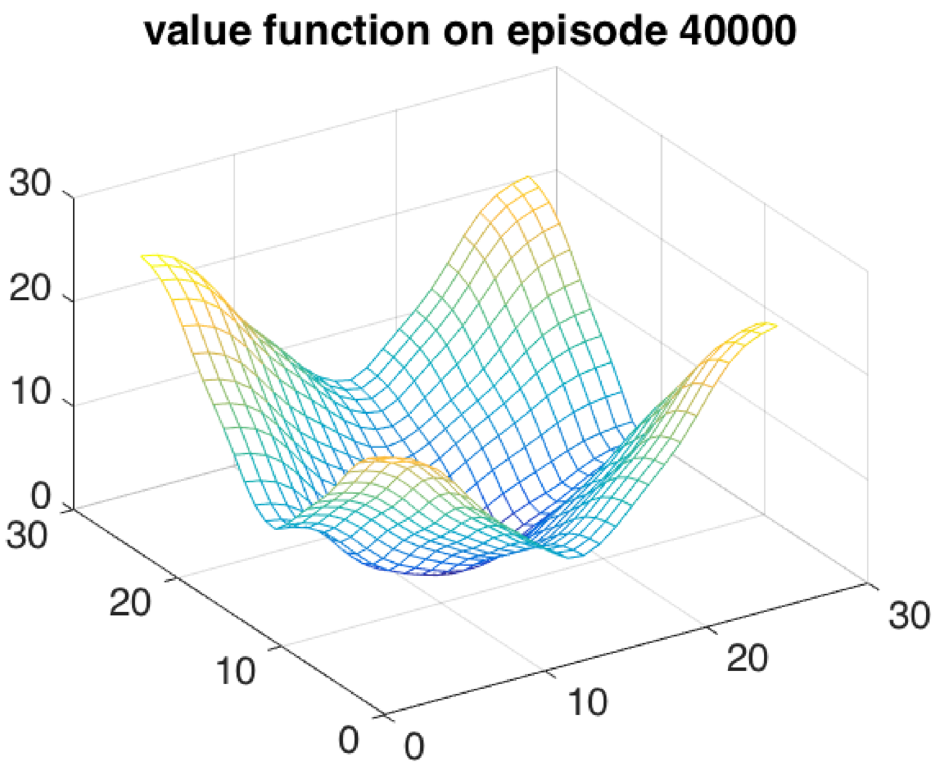


Figure 14: Value function for 25x25 maze with obstacles using RBF

6 Conclusion

Here we have seen examples how function approximation leads to faster convergence of the value function although sometime making it less stable. RBF basis functions predictably provided better idea about position of the agent in the world then identity functions. Also, and important observations that RBF with very small width (SIGMA) turns into essentially identity functions. On the other hand, very big value of width (SIGMA) leaves agent disoriented about its true position in the maze as for each state basis functions return almost the same value.

7 Code Appendix

main_v1 runs simulation for tabular Q function:

```
function [samples, avg_rs, convg_times, actions_map] = main_v1()
  % a simple illustration of Q-learning: a walker in a 2-d maze

  % number of states
  Sx = 10;  % width of grid world
  Sy = 10;  % length of grid world
  M = 2; % cover neighbour area

  S = Sx*Sy;  % number of states in grid world
  %4 corners
  GoalsOrig =[[1,   1];
              [Sx,  1];
              [1,  Sy];
              [Sx, Sy]];

  %spread goals across patches MxM
  Goals = zeros(4 * M * M, 2);
  shift = M - 1;
  k = 1;
  for i = 1:4
    goal = GoalsOrig(i, :);
    for p = -shift:shift
      for q = -shift:shift
        cand = goal + [p, q];
        if 1 <= cand(1) && cand(1) <= Sx &&...
           1 <= cand(2) && cand(2) <= Sy
           Goals(k, :) = cand;
           k = k + 1;
        end
      end
    end
  end

  % number of actions
  A = 5;    % number of actions: E, S, W, N and 0
  % total number of learning trials
  T = 60000;

  eta = 0.2;
  gamma = 0.9;
  epsilon = 0.1;
```

```
NUM_EXPS = 1; %number of experiments
convg_times = [];
actions_map = zeros(Sx, Sy);

for exp_id = 1:NUM_EXPS
  samples = zeros(T, 1);
  avg_rs = zeros(T, 1);
  %initialisation
  Q = 0.1*rand(Sx, Sy, A);
  % run the algorithm for T trials
  r = 0;
  for t=1:T
    % set the starting state
    x0=randi(Sx);
    y0=randi(Sy);
    % start trial
    avg_r = 0;
    for u=1:S*S
      % exploration (epsilon-greedy)
      if (rand(1)<epsilon)
        a0=randi(A);
      else
        [V_s0,a0]=max(Q(x0, y0,:));  % we only need the a0 here
      end;

      % reward is for reaching the goal and staying there
      if ismember([x0, y0], Goals, 'rows') %&& (a0==5)
        r = r + 1;
      else
        r = r + 0;
      end

      if x0 == int32(Sx/2) % && y0 == int32(Sy/2)
        r = r - 10;
      end

      %%you can add obstacles simply by adding punishment
      %%e.g. somewhere in the middle
      % if (abs(rem(s0,Sx)-Sy/2)<2)&&(abs(idivide(cast(s0,'int8'),cast(Sy,'int8'))-Sx/2)<2)
      % r=-10;
      % end;
      %%The agent can still move through the obstacle, so it's more like a
      %%pothole.

```

```
      % now moving left, right, up, down, or not
      % and don't step outside the track
      % E, S, W, N and 0
      x1 = x0;
      y1 = y0;
      if     (a0==1)
        x1 = min(max(x0+1, 1), Sx); %east
      elseif (a0==2)
        y1 = min(max(y0-1, 1), Sy); %south
      elseif (a0==3)
        x1 = min(max(x0-1, 1), Sx); %west
      elseif (a0==4)
        y1 = min(max(y0+1, 1), Sy); %north
      end

      % now the learning step
      V_s1 = max(Q(x1, y1,:));
      Q(x0, y0, a0)=(1-eta)*Q(x0, y0, a0)+eta*(r+gamma*V_s1);

      % goto next trial once the goal is reached
      if ismember([x0, y0], Goals, 'rows')
        avg_r = r/u;
        %display(avg_r, 'reached the goal!');
        break;
      end

      x0=x1;
      y0=y1;
    end

    avg_rs(t) = avg_r;
    VV = max(Q,[],3);
    samples(t) = sum(reshape(VV, 1,numel(VV)));

    % % you may prefer using a mesh plot as it is a 2D example.
    if (rem(t, 1000) == 0)
      display(t, 'current trial');
    end

    if (rem(t,1000)==0 && NUM_EXPS == 1)
      figure(6);
      clf;
      zlim([0 1/(1-gamma)]);
      %samples(
      mesh(VV);
      %%you may also like to have a look at (a=2, choose also other ↩
         actions):
```

```
        %Q2 = reshape(Q(:,2),Sx,Sy);
        %mesh(Q2);
        title(['value function on episode ', num2str(t)]);
        fig=gcf;
        set(findall(fig,'-property','FontSize'),'FontSize',24)
        drawnow

      %t % don't know how to put current time in the figure
      end

    end
    %end of epoch
    display(exp_id, 'end of experiment');

    average_over = 200;
    avg_samples = mean(reshape(samples, average_over, []));

    if NUM_EXPS == 1
      figure(7);
      clf;
      title('value average values');
      plot(avg_samples);
      ylabel('value average values');
      xlabel(['training episodes (x', num2str(average_over), ')']);
      fig=gcf;
      set(findall(fig,'-property','FontSize'),'FontSize',24)


      figure(9);
      clf;
      title('epsiode average reward');
      plot(mean(reshape(avg_rs, average_over, [])));
      ylabel('episodes average reward');
      xlabel(['training episodes (x', num2str(average_over), ')']);
      fig=gcf;
      set(findall(fig,'-property','FontSize'),'FontSize',24)
    end

%     rel_diff = zeros(1, numel(avg_samples) - 1);
%     for t=2:size(avg_samples, 2)
%       rel_diff(t) = abs(avg_samples(t) - avg_samples(t-1))/avg_samples(t↩
  );
%     end
%
%     convg = find(rel_diff > 0.00001, 1, 'last')
%     convg_t = convg * average_over
%     if ~isempty(convg_t)
```

```
%       convg_times = [convg_times, convg_t];
%     end

    smp_stds = zeros(1,(size(avg_samples, 2) - 50));
    for t=1:(size(avg_samples, 2) - 50)
      smp_std = std(avg_samples(t:(t+50)));
      smp_stds(t) = smp_std;%abs(avg_samples(t) - avg_samples(t-1))/↩
         avg_samples(t);
    end

    convg = find(smp_stds > 0.2, 1, 'last')
    convg_t = convg * average_over
    if ~isempty(convg_t)
      convg_times = [convg_times, convg_t];
    end

    if NUM_EXPS == 1
      figure(10);
      clf;
      title('averaged value function standard deviation over 50');
      plot(smp_stds);
      ylabel('standard deviation');
      xlabel(['training episodes (x', num2str(average_over), ')']);
      fig=gcf;
      set(findall(fig,'-property','FontSize'),'FontSize',24)
    end

    if NUM_EXPS == 1
      %plot policy
      actions_map = zeros(Sx, Sy);
      %initial state in the left bottom corner with
      %pasenger and must go to G

      for xx=1:Sx
        for yy=1:Sy
          %perform greedy action
          [V_s,aa]=max(Q(xx, yy,:));
          actions_map(xx, yy) = aa;
        end
      end

      %lbls_actions = grid_to_lbls(actions_map);
      %lbls_actions = actions2str(lbls_actions);
      figure(31);
      clf;
      draw_maze(actions_map, 'maze policy', Sx, Sy, Goals);
    end
```

```
  end
  %end of experiments loop
  SE = std(convg_times)/sqrt(length(convg_times));
  display(mean(convg_times), 'mean convergence time');
  display(SE, 'standard error');
  display(SE * 1.96, 'confidence interval');
end
```

main_v2 runs simulation for identity basis functions:

```
function [samples, avg_rs, convg_times, actions_map] = main_v2()
  % a simple illustration of Q-learning: a walker in a 2-d maze

  % number of states
  Sx = 10;  % width of grid world
  Sy = 10;  % length of grid world
  M = 2; % cover neighbour area

  S = Sx*Sy;  % number of states in grid world
  %4 corners
  GoalsOrig =[[1,   1];
              [Sx,  1];
              [1,  Sy];
              [Sx, Sy]];

  %spread goals across patches MxM
  Goals = zeros(4 * M * M, 2);
  shift = M - 1;
  k = 1;
  for i = 1:4
    goal = GoalsOrig(i, :);
    for p = -shift:shift
      for q = -shift:shift
        cand = goal + [p, q];
        if 1 <= cand(1) && cand(1) <= Sx &&...
           1 <= cand(2) && cand(2) <= Sy
           Goals(k, :) = cand;
           k = k + 1;
        end
      end
    end
  end

  % number of actions
  A = 5;    % number of actions: E, S, W, N and 0
```

```
% total number of learning trials
T = 75000;

eta = 0.2;
gamma = 0.9;
epsilon = 0.1;

NUM_EXPS = 10; %number of experiments
convg_times = [];
actions_map = zeros(Sx, Sy);


F = zeros(Sx, Sy, 25);
%precompute basis functions in advance
for ix = 1:Sx
  for iy = 1:Sy
    %identity functions
    for kx = 1:5
      for ky = 1:5
        fId = 5 * (ky - 1) + kx;
        F(ix, iy, fId) = M*(kx-1) < ix && ix <= M*kx &&...
                         M*(ky-1) < iy && iy <= M*ky;
      end
    end
  end
end

for exp_id = 1:NUM_EXPS
  samples = zeros(T, 1);
  avg_rs = zeros(T, 1);
  %initialisation
  %weights
  W = zeros(A,25);%0.1*rand(A, 25);
  % run the algorithm for T trials
  for t=1:T
    % set the starting state
    x0=randi(Sx);
    y0=randi(Sy);
    % start trial
    avg_r = 0;
    for u=1:S*S
      % exploration (epsilon-greedy)
      if (rand(1)<epsilon)
        a0=randi(A);
      else
        [V_s0, a0]=max(W * reshape(F(x0, y0, :), 25, 1));  % we only need the a0 here
```

```
end;

% reward is for reaching the goal and staying there
if ismember([x0, y0], Goals, 'rows') %&& (a0==5)
  r=1;
else
  r=0;
end

%%you can add obstacles simply by adding punishment
%%e.g. somewhere in the middle
% if (abs(rem(s0,Sx)-Sy/2)<2)&&(abs(idivide(cast(s0,'int8'),cast(Sy,'int8'))-Sx/2)<2)
% r=-10;
% end;
%%The agent can still move through the obstacle, so it's more like a
%%pothole.

% now moving left, right, up, down, or not
% and don't step outside the track
% E, S, W, N and 0
x1 = x0;
y1 = y0;
if     (a0==1)
  x1 = min(max(x0+1, 1), Sx); %east
elseif (a0==2)
  y1 = min(max(y0-1, 1), Sy); %south
elseif (a0==3)
  x1 = min(max(x0-1, 1), Sx); %west
elseif (a0==4)
  y1 = min(max(y0+1, 1), Sy); %north
end

% now the learning step
%approximate value function V(s1)
V_s1 = max(W * reshape(F(x1, y1, :), 25, 1));
%approximate Q(s0, a0)
Q_s0_a0 = W(a0, :) * reshape(F(x0, y0, :), 25, 1);
delta = r+gamma*V_s1 - Q_s0_a0;
%basic functions in state x0 y0
bfs = reshape(F(x0, y0, :), 1, 25);
W(a0, :) = W(a0, :) + eta * delta * bfs;

% goto next trial once the goal is reached
if ismember([x0, y0], Goals, 'rows')
  avg_r = r/u;
```

```
      %display(avg_r, 'reached the goal!');
      break;
    end

    x0=x1;
    y0=y1;
  end

  avg_rs(t) = avg_r;
  % reshape(F, Sx * Sy, 25) % Sx*Sy x 25
  % W' %  25 x A
  QQ = reshape(F, Sx * Sy, 25) * W'; % Sx*Sy x A
  VV = max(QQ,[],2); %Sx*Sy
  VV = reshape(VV, Sx, Sy);


  %VV = max(Q,[],3);
  samples(t) = sum(reshape(VV, 1,numel(VV)));

  % % you may prefer using a mesh plot as it is a 2D example.
  if (rem(t, 1000) == 0)
    display(t, 'current trial');
  end

  if (rem(t,1000)==0 && NUM_EXPS == 1)
    figure(6);
    clf;
    zlim([0 1/(1-gamma)]);
    %samples(
    mesh(VV);
    %%you may also like to have a look at (a=2, choose also other actions):
    %Q2 = reshape(Q(:,2),Sx,Sy);
    %mesh(Q2);
    title(['value function on episode ', num2str(t)]);
    fig=gcf;
    set(findall(fig,'-property','FontSize'),'FontSize',24)
    drawnow

  %t % don't know how to put current time in the figure
  end

end
%end of epoch
display(exp_id, 'end of experiment');

average_over = 200;
```

```
    avg_samples = mean(reshape(samples, average_over, []));

    if NUM_EXPS == 1
      figure(7);
      clf;
      title('value average values');
      plot(avg_samples);
      ylabel('value average values');
      xlabel(['training epsidoes (x', num2str(average_over), ')']);
      fig=gcf;
      set(findall(fig,'-property','FontSize'),'FontSize',24)

      figure(9);
      clf;
      title('episode average reward');
      plot(mean(reshape(avg_rs, average_over, [])));
      ylabel('episode average reward');
      xlabel(['training epsidoes (x', num2str(average_over), ')']);
      fig=gcf;
      set(findall(fig,'-property','FontSize'),'FontSize',24)
    end

    smp_stds = zeros(1,(size(avg_samples, 2) - 50));
    for t=1:(size(avg_samples, 2) - 50)
      smp_std = std(avg_samples(t:(t+50)));
      smp_stds(t) = smp_std;%abs(avg_samples(t) - avg_samples(t-1))/↩
          avg_samples(t);
    end

    convg = find(smp_stds > 6, 1, 'last')
    convg_t = convg * average_over
    if ~isempty(convg_t)
      convg_times = [convg_times, convg_t];
    end

    if NUM_EXPS == 1
      figure(10);
      clf;
      title('averaged value function standard deviation over 50');
      plot(smp_stds);
      ylabel('standard deviation');
      xlabel(['training episodes (x', num2str(average_over), ')']);
      fig=gcf;
      set(findall(fig,'-property','FontSize'),'FontSize',24)
    end

    if NUM_EXPS == 1
```

```
    %plot policy
    actions_map = zeros(Sx, Sy);
    %initial state in the left bottom corner with
    %pasenger and must go to G
    QQ = reshape(F, Sx * Sy, 25) * W'; % Sx*Sy x A
    Q = reshape(QQ, Sx, Sy, A);
    for xx=1:Sx
      for yy=1:Sy
        %perform greedy action
        [V_s,aa]=max(Q(xx, yy,:));
        actions_map(xx, yy) = aa;
      end
    end

    %lbls_actions = grid_to_lbls(actions_map);
    %lbls_actions = actions2str(lbls_actions);
    figure(31);
    clf;
    draw_maze(actions_map, 'maze policy', Sx, Sy, Goals);
  end
 end
 %end of experiments loop
 SE = std(convg_times)/sqrt(length(convg_times));
 display(mean(convg_times), 'mean convergence time');
 display(SE, 'standard error');
 display(SE * 1.96, 'confidence interval');
end
```

main_v3 runs simulation for RBF functions:

```
function [samples, avg_rs, convg_times, actions_map] = main_v3()
  % a simple illustration of Q-learning: a walker in a 2-d maze

  % number of states
  Sx = 10;  % width of grid world
  Sy = 10;  % length of grid world
  M = 2; % cover neighbouring area

  S = Sx*Sy;  % number of states in grid world
  %4 corners
  GoalsOrig =[[1,   1];
              [Sx,  1];
              [1,  Sy];
              [Sx, Sy]];
%   GoalsOrig =[[1,   1];
```

```
%               [Sx, Sy]];

  %spread goals across patches MxM
  Goals = zeros(size(GoalsOrig, 1) * M * M, 2);
  shift = M - 1;
  k = 1;
  for i = 1:size(GoalsOrig, 1)
    goal = GoalsOrig(i, :);
    for p = -shift:shift
      for q = -shift:shift
        cand = goal + [p, q];
        if 1 <= cand(1) && cand(1) <= Sx &&...
           1 <= cand(2) && cand(2) <= Sy
           Goals(k, :) = cand;
           k = k + 1;
        end
      end
    end
  end

  %Goals = GoalsOrig;

  % number of actions
  A = 5;    % number of actions: E, S, W, N and 0
  % total number of learning trials
  T = 60000;

  eta = 0.2;
  gamma = 0.9;
  epsilon = 0.1;%0.2;%0.4

  NUM_EXPS = 1; %number of experiments
  convg_times = [];
  actions_map = zeros(Sx, Sy);

  %1.0 for 10 is smooth like true function, 0.5 is almost like identity function
  %2.5 for 25 is smooth like true function (epsilon must be 0.1), 1.0 for 25 is almost like identity function
  %0.5
  SIGMA = 1.0;%1.0;

  F = zeros(Sx, Sy, 25);

  %precompute basis functions in advance


```

```
for ix = 1:Sx
  for iy = 1:Sy
    %identity functions
    for kx = 1:5
      for ky = 1:5
        fId = 5 * (ky - 1) + kx;
        %position of rbf in maze coordinates
        cx = kx*M - (M - 1)/2;
        cy = ky*M - (M - 1)/2;
        E = exp(-(1/(2*SIGMA^2))*((ix-cx)^2+(iy-cy)^2));
        Z = 1;%2*pi*SIGMA^2;
        F(ix, iy, fId) = E/Z;
      end
    end
    Z = sum(F(ix, iy, :));
    F(ix, iy, :) = F(ix, iy, :) / Z;
  end
end

for exp_id = 1:NUM_EXPS
  samples = zeros(T, 1);
  avg_rs = zeros(T, 1);
  %initialisation
  %weights
  W = zeros(A, 25);%0.1*rand(A, 25);
  % run the algorithm for T trials
  for t=1:T
    % set the starting state
    x0=randi(Sx);
    y0=randi(Sy);
    % start trial
    avg_r = 0;
    r = 0;
    for u=1:S*S
      % exploration (epsilon-greedy)
      if (rand(1)<epsilon)
        a0=randi(A);
      else
        [V_s0, a0]=max(W * reshape(F(x0, y0, :), 25, 1));  % we only ↩
           need the a0 here
      end;

      % reward is for reaching the goal and staying there
      if ismember([x0, y0], Goals, 'rows') %&& (a0==5)
        r = r + 1;
      else
        r = r + 0;
```

```
        end

        %%you can add obstacles simply by adding punishment
        %%e.g. somewhere in the middle
        % if (abs(rem(s0,Sx)-Sy/2)<2)&&(abs(idivide(cast(s0,'int8'),cast(Sy,'int8'))-Sx/2)<2)
        % r=-10;
        % end;
        %%The agent can still move through the obstacle, so it's more like a
        %%pothole.
        if x0 == int32(Sx/2) % && y0 == int32(Sy/2)
          r = r - 10;
        end
        % now moving left, right, up, down, or not
        % and don't step outside the track
        % E, S, W, N and 0
        x1 = x0;
        y1 = y0;
        if     (a0==1)
          x1 = min(max(x0+1, 1), Sx); %east
        elseif (a0==2)
          y1 = min(max(y0-1, 1), Sy); %south
        elseif (a0==3)
          x1 = min(max(x0-1, 1), Sx); %west
        elseif (a0==4)
          y1 = min(max(y0+1, 1), Sy); %north
        end

        % now the learning step
        %approximate value function V(s1)
        V_s1 = max(W * reshape(F(x1, y1, :), 25, 1));
        %approximate Q(s0, a0)
        Q_s0_a0 = W(a0, :) * reshape(F(x0, y0, :), 25, 1);
        delta = r+gamma*V_s1 - Q_s0_a0;
        %basic functions in state x0 y0
        bfs = reshape(F(x0, y0, :), 1, 25);
        W(a0, :) = W(a0, :) + eta * delta * bfs;

        % goto next episode once the goal is reached
        if ismember([x0, y0], Goals, 'rows')
          avg_r = r/u;
          %display(avg_r, 'reached the goal!');
          break;
        end

        x0=x1;
```

```
        y0=y1;
      end

      avg_rs(t) = avg_r;
      % reshape(F, Sx * Sy, 25) % Sx*Sy x 25
      % W' %  25 x A
      QQ = reshape(F, Sx * Sy, 25) * W'; % (Sx*Sy x 25) * (25 x A) = Sx*Sy x A
      VV = max(QQ,[],2); %Sx*Sy
      VV = reshape(VV, Sx, Sy);


      %VV = max(Q,[],3);
      samples(t) = sum(reshape(VV, 1,numel(VV)));

      % % you may prefer using a mesh plot as it is a 2D example.
      if (rem(t, 1000) == 0)
        display(t, 'current trial');
      end

      if (rem(t,1000)==0 && NUM_EXPS == 1)
        figure(6);
        clf;
        zlim([0 1/(1-gamma)]);
        %samples(
        mesh(VV);
        %%you may also like to have a look at (a=2, choose also other actions):
        %Q2 = reshape(Q(:,2),Sx,Sy);
        %mesh(Q2);
        title(['value function on episode ', num2str(t)]);
        fig=gcf;
        set(findall(fig,'-property','FontSize'),'FontSize',24)
        drawnow

      %t % don't know how to put current time in the figure
      end

    end
    %end of epoch
    display(exp_id, 'end of experiment');

    average_over = 200;
    avg_samples = mean(reshape(samples, average_over, []));

    if NUM_EXPS == 1
      figure(7);
```

```
    clf;
    title('value average values');
    plot(avg_samples);
    ylabel('value average values');
    xlabel(['training episodes (x', num2str(average_over), ')']);
    fig=gcf;
    set(findall(fig,'-property','FontSize'),'FontSize',24)

    figure(9);
    clf;
    title('episode average reward');
    plot(mean(reshape(avg_rs, average_over, [])));
    ylabel('episode average reward');
    xlabel(['training episodes (x', num2str(average_over), ')']);
    fig=gcf;
    set(findall(fig,'-property','FontSize'),'FontSize',24)
  end

  smp_stds = zeros(1,(size(avg_samples, 2) - 50));
  for t=1:(size(avg_samples, 2) - 50)
    smp_std = std(avg_samples(t:(t+50)));
    smp_stds(t) = smp_std;%abs(avg_samples(t) - avg_samples(t-1))/↩
       avg_samples(t);
  end

  convg = find(smp_stds > 25, 1, 'last')
  convg_t = convg * average_over
  if ~isempty(convg_t)
    convg_times = [convg_times, convg_t];
  end

  if NUM_EXPS == 1
    figure(10);
    clf;
    title('averaged value function standard deviation over 50');
    plot(smp_stds);
    ylabel('standard deviation');
    xlabel(['training episodes (x', num2str(average_over), ')']);
    fig=gcf;
    set(findall(fig,'-property','FontSize'),'FontSize',24)
  end

  if NUM_EXPS == 1
    %plot policy
    actions_map = zeros(Sx, Sy);
    %initial state in the left bottom corner with
    %pasenger and must go to G
```

```
      QQ = reshape(F, Sx * Sy, 25) * W'; % Sx*Sy x A
      Q = reshape(QQ, Sx, Sy, A);
      for xx=1:Sx
        for yy=1:Sy
          %perform greedy action
          [V_s,aa]=max(Q(xx, yy,:));
          actions_map(xx, yy) = aa;
        end
      end

      %lbls_actions = grid_to_lbls(actions_map);
      %lbls_actions = actions2str(lbls_actions);
      figure(31);
      clf;
      draw_maze(actions_map, 'maze policy', Sx, Sy, Goals);
    end
  end
  %end of experiments loop
  SE = std(convg_times)/sqrt(length(convg_times));
  display(mean(convg_times), 'mean convergence time');
  display(SE, 'standard error');
  display(SE * 1.96, 'confidence interval');
end
```

supporting utility to draw policy:

```
function [] = draw_maze(lbls, name, Sx, Sy, Goals)
    %lbls - labels - cell array of strings to display in the maze grid
    %size X*Y
    %name - plot title
    %draw location names with light green color

    hold on;
    grid on;
    h = findobj(gca,'Type','Text');
    delete(h);

    title(name);
    xlim([0 Sx]);
    ylim([0 Sy]);
    set(gca,'XTick',0:Sx);
    set(gca,'YTick',0:Sy);
    %to have nice grid cells as squares
    axis square;

```

```
  % grid domains
  xg = 1:Sx;
  yg = 1:Sy;
  %label coordinates, substract 0.5 for centering
  [xlbl, ylbl] = meshgrid(xg - 0.5, yg - 0.5);
  drawArrow = @(x,y,varargin) quiver( x(1),y(1),x(2)-x(1),y(2)-y(1),0, varargin{:} );
  for ix = 1:Sx
    for iy = 1:Sy
      xx = xlbl(ix, iy);
      yy = ylbl(ix, iy);
      a = lbls(ix, iy);
      if ismember([ix, iy], Goals, 'rows')
        text(xx, yy, 'Goal',...
          'Color', 'r',...
          'FontWeight', 'bold', 'FontUnits', 'normalized', 'FontSize', 1/( 5* Sy),...
          'HorizontalAlignment', 'center',...
          'VerticalAlignment', 'middle');
      elseif a == 1
        %right
        drawArrow([xx, xx], [yy - 0.3, yy + 0.3], 'MaxHeadSize',10,'Color','r','LineWidth',3);
      elseif a == 2
        %down
        drawArrow([xx + 0.3, xx - 0.3], [yy,yy], 'MaxHeadSize',12,'Color','r','LineWidth',2);

      elseif a == 3
        %left
        drawArrow([xx, xx], [yy + 0.3, yy - 0.3],'MaxHeadSize',12,'Color','r','LineWidth',2);
      elseif a == 4
        %up
        drawArrow([xx - 0.3, xx + 0.3], [yy,yy], 'MaxHeadSize',12,'Color','r','LineWidth',2);
      elseif a == 5
        text(xx, yy, 'Stay',...
          'Color', 'r',...
          'FontWeight', 'bold', 'FontUnits', 'normalized', 'FontSize', 1/( 5* Sy),...
          'HorizontalAlignment', 'center',...
          'VerticalAlignment', 'middle');
      end

    end
  end
```

```
%     text(xlbl(:), ylbl(:), lbls(:),'color','r',...
%         'HorizontalAlignment','center','VerticalAlignment','middle',...
%         'FontWeight', 'bold', 'FontUnits', 'normalized', 'FontSize', ↩
    1/(5*Y));

    hold off;
    drawnow;
end
```