MI - H14

February 16, 2017

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors
from functools import reduce
import itertools
import sklearn.datasets
import os
import re
%matplotlib inline
```

In [415]:
```
def plot(data, ax=None, enum=False, title='', labels=None, legend=False,
    axes_defined = ax != None
    if not axes_defined:
        fig, ax = plt.subplots(1, 1, figsize=(13, 4))
    plotted = None
    if enum:
        plotted = ax.plot(data, **kwargs)
    else:
        mapping = np.array(data).T
        plotted = ax.plot(mapping[0], mapping[1], **kwargs)
    if labels:
        ax.set_xlabel(labels[0])
        if (len(labels) > 1):
            ax.set_ylabel(labels[1])
    if legend:
        ax.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
    ax.set_title(title)
    ax.grid(True)
    if not axes_defined:
        fig.tight_layout()
    return ax

def scatter(data, ax=None, enum=False, title='', labels=None, legend=Fals
    axes_defined = ax != None
    if not axes_defined:
        fig, ax = plt.subplots(1, 1, figsize=(13, 4))
    scattered = None
    if enum:
```

```
            scattered = ax.scatter(range(len(data)), data, **kwargs)
        else:
            mapping = np.array(data).T
            scattered = ax.scatter(mapping[0], mapping[1], **kwargs)
        if labels:
            ax.set_xlabel(labels[0])
            if (len(labels) > 1):
                ax.set_ylabel(labels[1])
        if legend:
            ax.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0)
        if xlim:
            ax.set_xlim(xlim)
        ax.set_title(title)
        ax.grid(True)
        if colorbar:
            cax = plt_ax.make_axes_locatable(ax).append_axes("right", size="5
            cbar = plt.colorbar(scattered, cax=cax)
            cbar.set_ticks([-1, 0, 1])
        if not axes_defined:
            fig.tight_layout()
        return ax

    def heatmap(ax, data, title):
        cax = ax.imshow(data, extent=[-np.pi/4, np.pi/4, -3, 3], aspect="auto
        ax.set_xlabel('angle')
        ax.set_ylabel('velocity')
        ax.set_title(title)
        plt.colorbar(cax, ax=ax)
```

0.0.1 Global variables

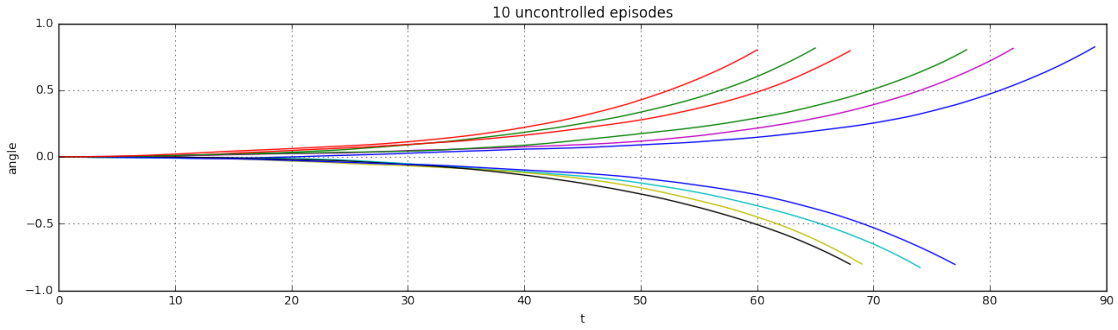
In [19]:
```
l = 1  # m
m = 2  # kg
g = 9.81  # m / s^2
# Shape of state variable: (angle, velocity)

ACTION_SUB = -4  # N
ACTION_WAIT = 0  # N
ACTION_ADD = 4  # N
actions = (ACTION_SUB, ACTION_WAIT, ACTION_ADD)
```

In [53]:
```
def reward(state):
    angle, _ = state
    if abs(angle) > np.pi/4:
        return -1
    else:
        return 0
```

```
        def is_failed(state):
            return reward(state) == -1

        # Calculations for dynamics in a small timestamp dt = 0.02 s
        def move(state, action, dt=0.02, sigma=3):
            angle, velocity = state
            epsilon = np.random.normal(0, sigma)
            next_velocity = velocity + (g / l) * np.sin(angle) * dt + (action / m)
            next_angle = angle + next_velocity * dt
            # TODO: Add reward, etc.
            return (next_angle, next_velocity)

        def simulate_uncontrolled():
            state = (0, 0)
            states = [state]
            while not is_failed(state) and len(states) < 1000:
                state = move(state, ACTION_WAIT)
                states.append(state)
            return state, states
```

0.0.2 Exercise 14.1 (a) Uncontrolled Episodes

```
In [54]: ax = None
         for _ in range(10):
             final_state, all_states = simulate_uncontrolled()
             angles = [angle for angle, _ in all_states]
             ax = plot(angles, ax, enum=True, labels=['t', 'angle'], title='10 unco
```



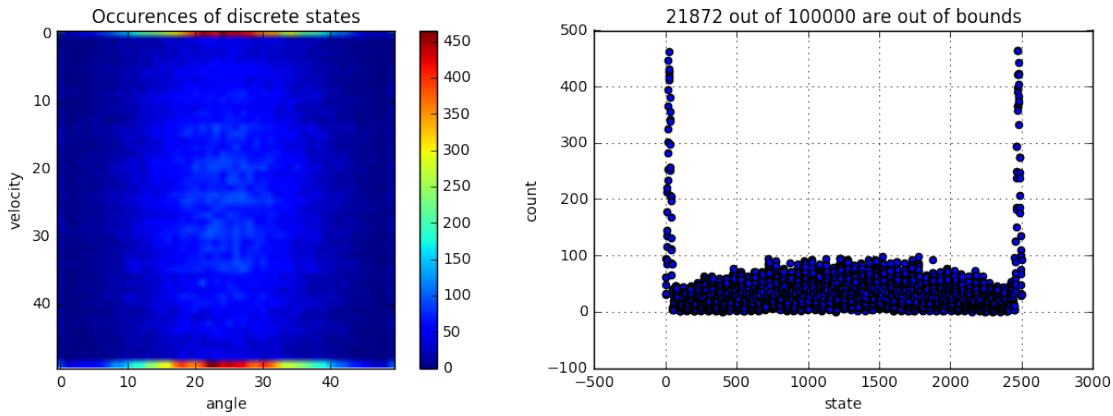
0.0.3 Exercise 14.1 (b) Discrete Spaces

```
In [426]: def discretize(value, value_space):
              min_i, min_ = -1, 10000
              for i, disc_value in enumerate(value_space):
                  # print(i, disc_value, np.abs(value - disc_value))
                  if np.abs(value - disc_value) < min_:
```

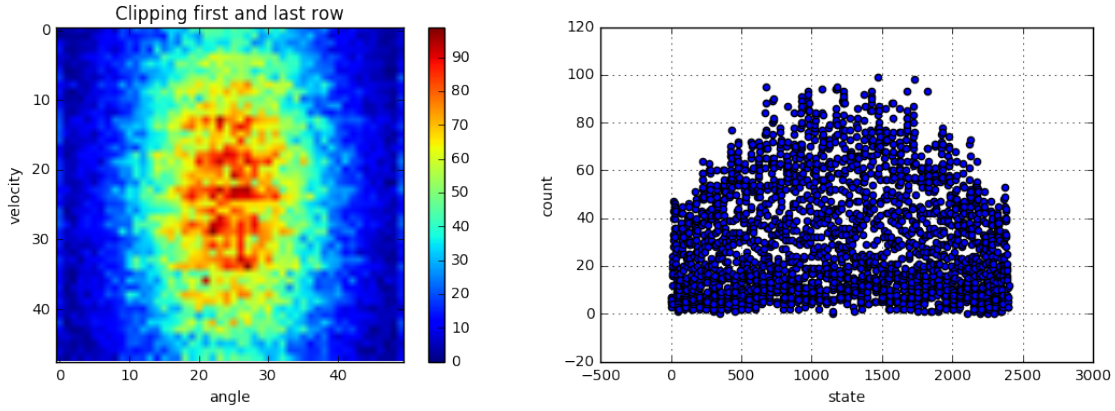
```
                min_i = i
                min_ = np.abs(value - disc_value)
            else:
                # Increasing distance -> Done
                break
        return value_space[min_i], min_i

    def discrete_state(x, flatten=False, angle_range=angle_values, velocity_r
        angle, velocity = x
        angle, angle_index = discretize(angle, angle_range)
        velocity, velocity_index = discretize(velocity, velocity_range)
        if flatten:
            return (angle, velocity), angle_index * len(velocity_range) + vel
        return (angle, velocity), (angle_index, velocity_index)
```

In [414]:
```
D1 = 50
D2 = 50
angle_values = np.linspace(-np.pi/4, np.pi/4, D1).round(2)
velocity_values = np.linspace(-3, 3, D2).round(2)
discrete_space = list(itertools.product(angle_values, velocity_values))
# Draw 100.000 continuous states from the normal dist (described in exerc
states = np.random.multivariate_normal([0, 0], [[np.pi/8, 0], [0, 3/2]],
# 50 x 50 image plot for the colorcoded amount of each discrete state
states = [discrete_state(x)[0] for x in states]
# Count occurences for each discrete state
counts = [states.count(disc_state) for disc_state in discrete_space]
```

In [416]:
```
fig, axes = plt.subplots(1, 2, figsize=(13, 4))
im = axes[0].imshow(np.reshape(counts, (50, 50)))
axes[0].set_title('Occurences of discrete states')
axes[0].set_xlabel('angle')
axes[0].set_ylabel('velocity')
plt.colorbar(im, ax=axes[0])
ax = scatter(counts, ax=axes[1], enum=True, labels=['state', 'count'], ti
    len(states) - len([x for x in states if -np.pi/4 <= x[0] <= np.pi/4 a
    len(states)))
```



```
In [417]: fig, axes = plt.subplots(1, 2, figsize=(13, 4))
          im = axes[0].imshow(np.reshape(counts[50:-50], (48, 50)))
          axes[0].set_title('Clipping first and last row')
          axes[0].set_xlabel('angle')
          axes[0].set_ylabel('velocity')
          plt.colorbar(im, ax=axes[0])
          ax = scatter(counts[50:-50], ax=axes[1], enum=True, labels=['state', 'cou
```

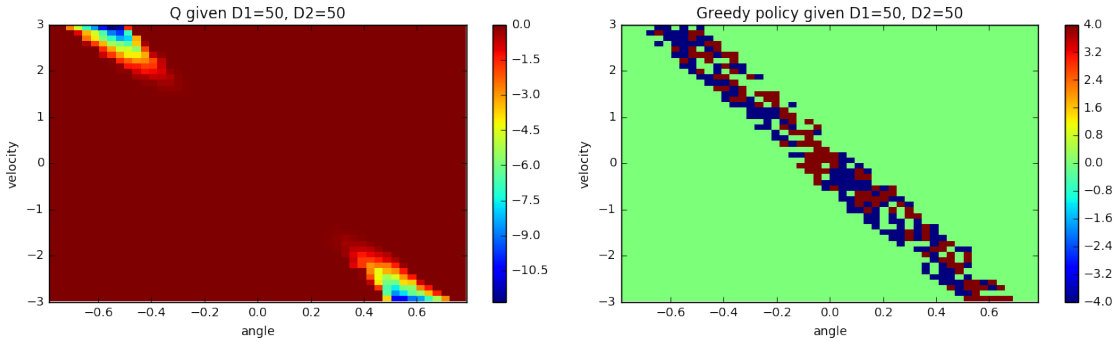


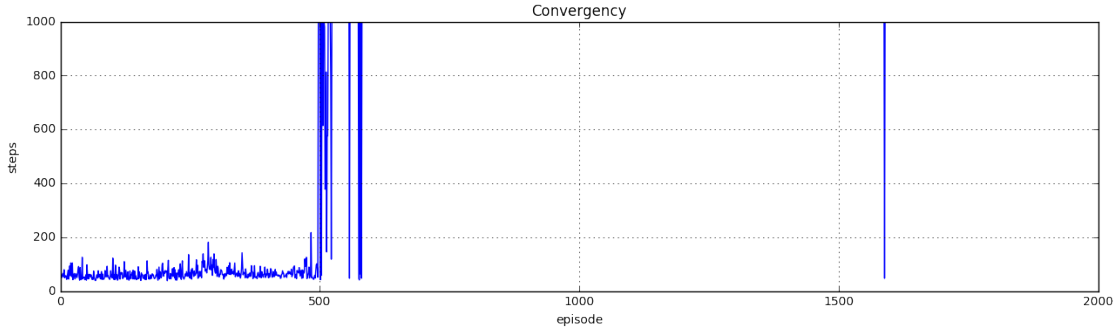
0.0.4 Exercise 14.2 (a) Q-Learning with e-greedy policy

```
In [451]: def egreedy_policy(Q, state_pos, action=None, e=0, seed=0):
              choice = -1
              randomize = np.random.RandomState(seed)
              if randomize.rand() < e or all(Q[state_pos] == 0):
                  choice = randomize.randint(len(actions))
              else:
                  # Find index of the best action optimizing Q[state, action]
                  choice = Q[state_pos].argmax()
              if not action:
                  return actions[choice], choice
              return int(action == choice)

          def qlearning(Q, angle_values, velocity_values, eta=0.5, gamma=0.9, e=0):
              # SARSA algorithm for Q approximation (on policy)
              state, state_pos = discrete_state((0, 0), flatten=True, angle_range=a
              exact_state = state
              states = [state]
              seed = np.random.randint(100)
              while not is_failed(state) and len(states) < 1000:
                  r = reward(state)
```

```
            action, action_pos = egreedy_policy(Q, state_pos, e=e, seed=seed+
            exact_state = move(exact_state, action)
            next_state, next_state_pos = discrete_state(exact_state, flatten=
            # print(state_pos, action_pos, next_state_pos)
            # print(state, action, exact_state, next_state)
            Q[state_pos, action_pos] += eta * (r + gamma * Q[next_state_pos].
            state, state_pos = next_state, next_state_pos
            states.append(state)
        # Add the reward value (-1 if failed)
        r = reward(state)
        action, action_pos = egreedy_policy(Q, state_pos, e=e, seed=seed+len(
        exact_state = move(exact_state, action)
        next_state, next_state_pos = discrete_state(exact_state, flatten=True
        Q[state_pos, action_pos] += eta * (r + gamma * Q[next_state_pos].max(
        return Q, states

    def train_qlearning(episodes=2000, D1=50, D2=50, eta=0.5, gamma=0.9, e=0,
        Q = np.zeros((D1 * D2, len(actions)))
        Qsteps = []
        angle_values = np.linspace(-np.pi/4, np.pi/4, D1).round(2)
        velocity_values = np.linspace(-3, 3, D2).round(2)
        D_space = np.array(list(itertools.product(angle_values, velocity_valu
        for i in range(2000):
            Q, Qstates = qlearning(Q, angle_values, velocity_values, eta, gam
            Qsteps.append(len(Qstates))
            if i % 200 == 0 and not quiet:
                print('Episode {} (D1={}, D2={})'.format(i, D1, D2))
        return Q, D_space, Qsteps

    def plot_qlearning(Q, D_space, Qsteps, D1=50, D2=50, e=0):
        fig, axes = plt.subplots(1, 2, figsize=(13, 4))
        heatmap(axes[0], Q.sum(axis=1).reshape(D1, D2), 'Q given D1={}, D2={}
        policies = np.array([egreedy_policy(Q, i, e=e)[0] for i in range(len(
        heatmap(axes[1], policies.reshape(D1, D2), 'Greedy policy given D1={}
        fig.tight_layout()
        plot(Qsteps, enum=True, title='Convergency', labels=['episode', 'step
```

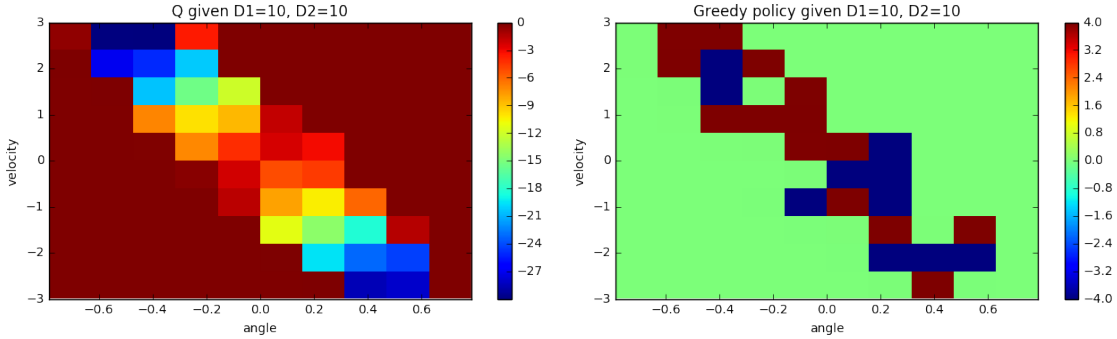
In [452]:
```
Q50, Q50_D_space, Q50_steps = train_qlearning()
plot_qlearning(Q50, Q50_D_space, Q50_steps)
```

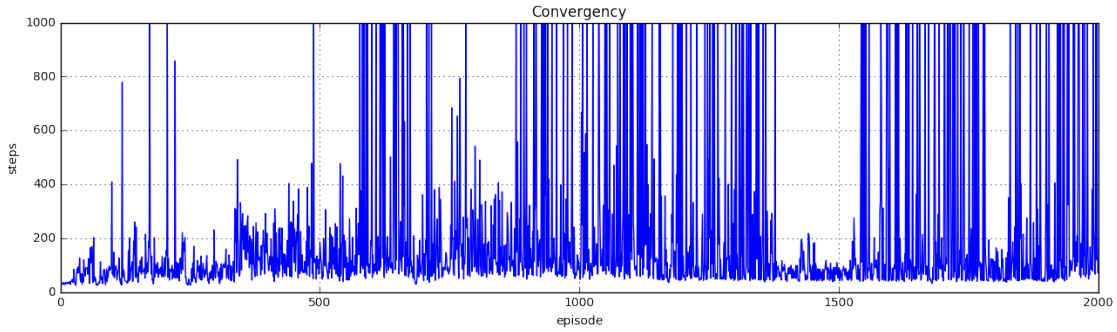


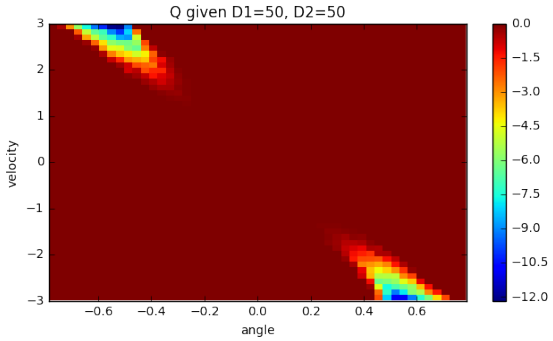


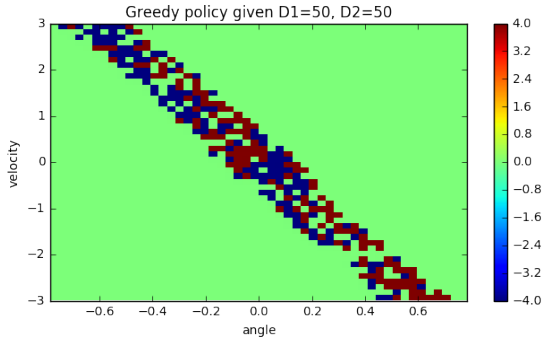
0.0.5 Exercise 14.2 (b) Changing D1 and D2

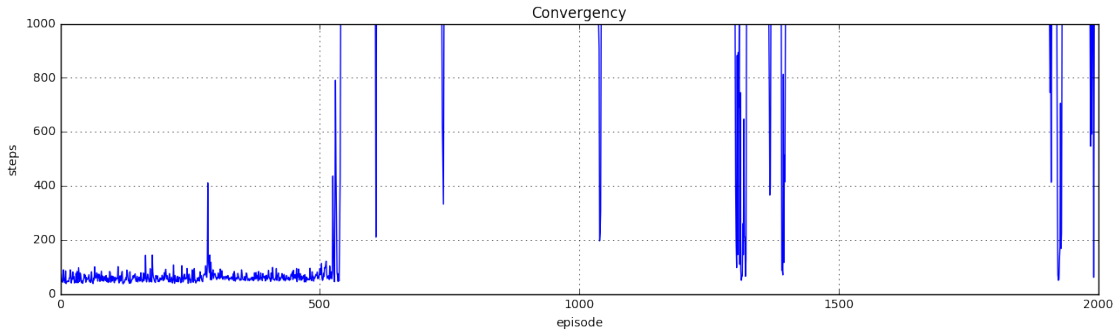
```
In [453]: for D in (10, 50, 200):
              Q, D_space, Q_steps = train_qlearning(D1=D, D2=D)
              plot_qlearning(Q, D_space, Q_steps, D1=D, D2=D)
```

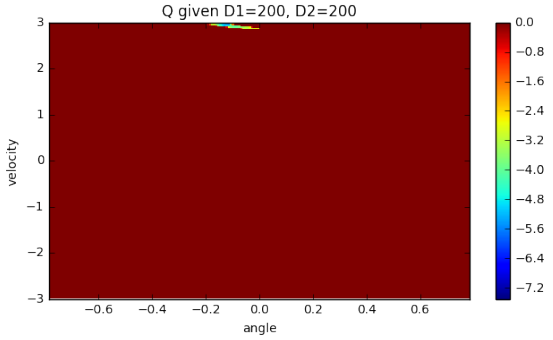


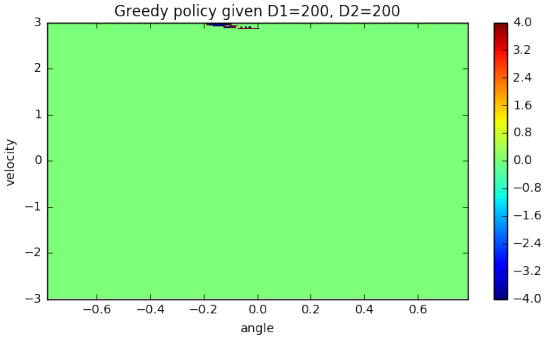

Convergency
steps
episode
0
200
400
600
800
1000
0
500
1000
1500
2000

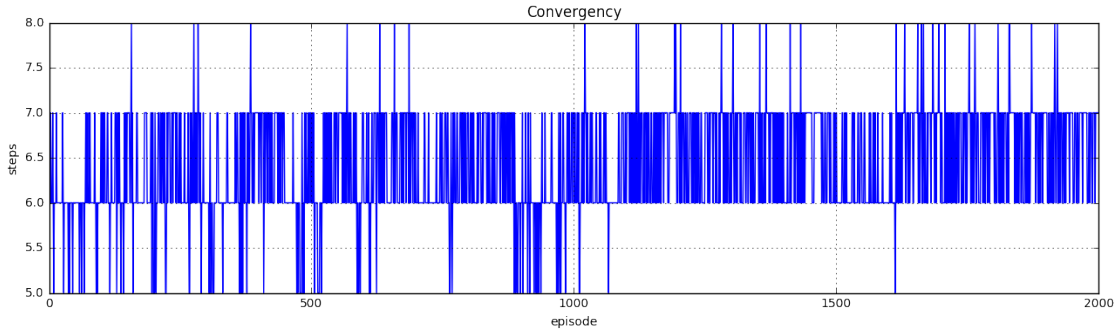

Q given D1=50, D2=50
velocity
angle
3
2
1
0
−1
−2
−3
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.0
−1.5
−3.0
−4.5
−6.0
−7.5
−9.0
−10.5
−12.0


Greedy policy given D1=50, D2=50
velocity
angle
3
2
1
0
−1
−2
−3
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
4.0
3.2
2.4
1.6
0.8
0.0
−0.8
−1.6
−2.4
−3.2
−4.0


Convergency
steps
episode
0
200
400
600
800
1000
0
500
1000
1500
2000


Q given D1=200, D2=200
velocity
angle
3
2
1
0
−1
−2
−3
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.0
−0.8
−1.6
−2.4
−3.2
−4.0
−4.8
−5.6
−6.4
−7.2


Greedy policy given D1=200, D2=200
velocity
angle
3
2
1
0
−1
−2
−3
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
4.0
3.2
2.4
1.6
0.8
0.0
−0.8
−1.6
−2.4
−3.2
−4.0

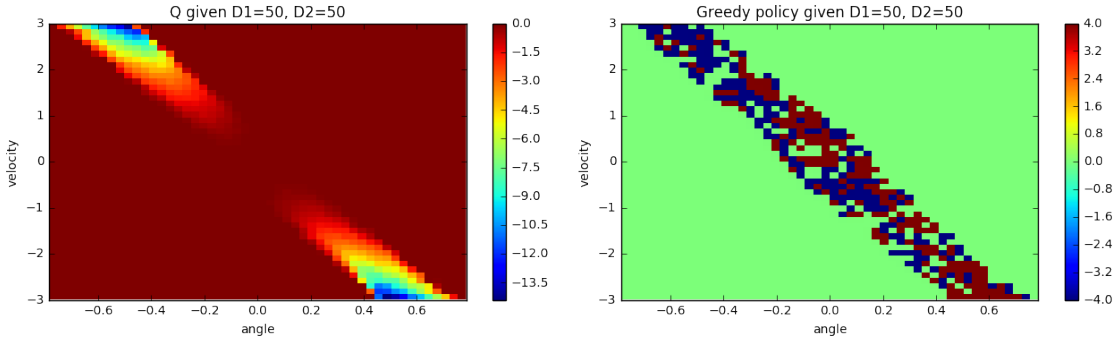


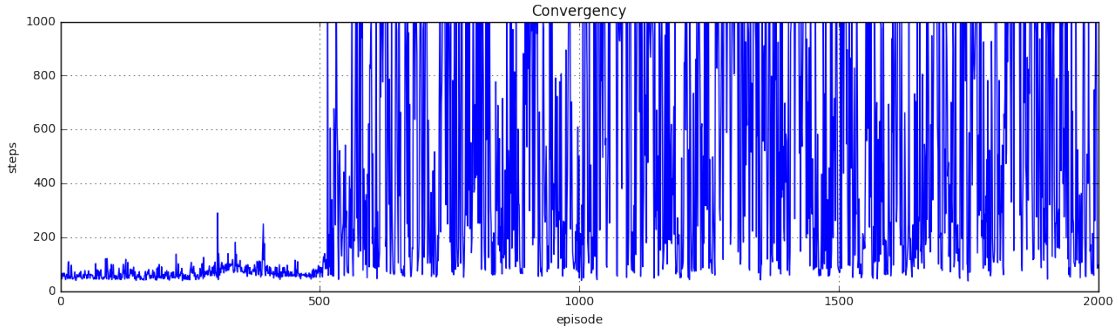
0.0.6 14.2 (c)

Smallest number that earns a good policy:

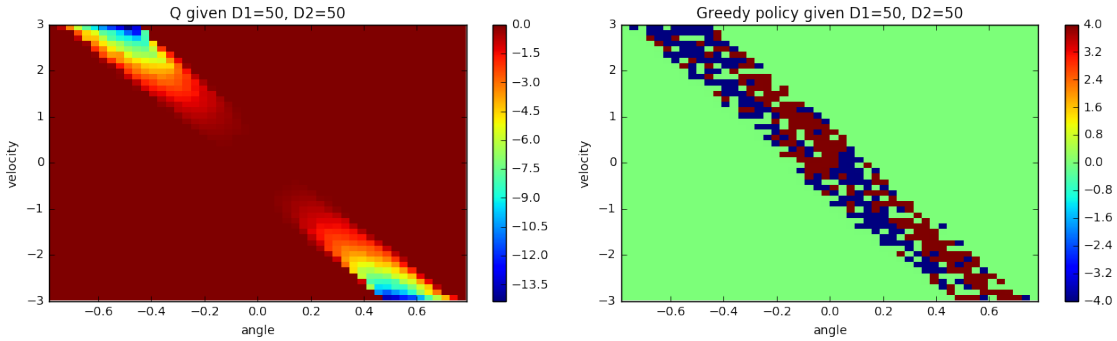
0.0.7 14.2 (d) e-greedy policy (e = 0.1)

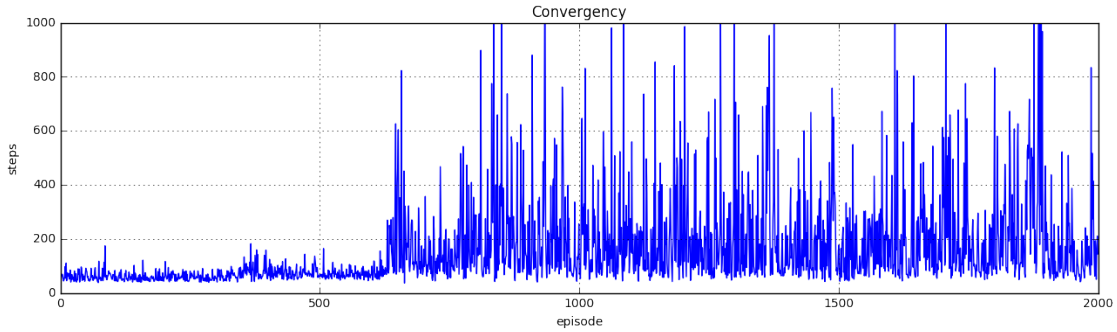
```
In [454]: Qe, Qe_D_space, Q_steps = train_qlearning(e=0.1)
          plot_qlearning(Qe, Qe_D_space, Q_steps, e=0.1)
```





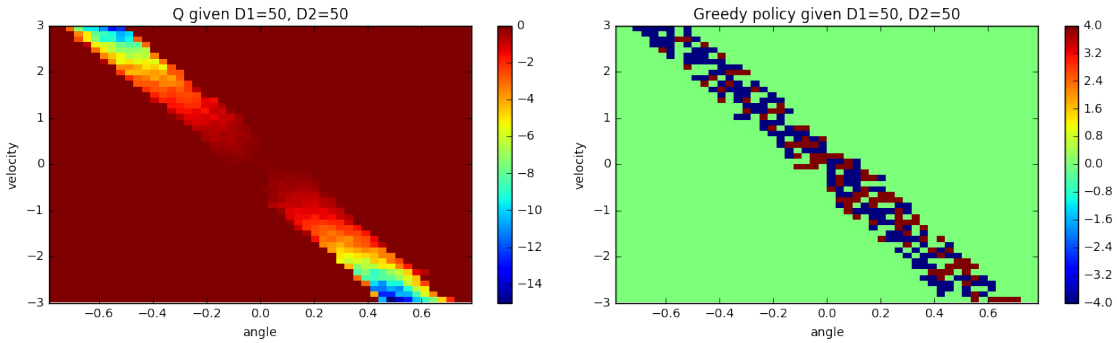
```
In [456]: Qe, Qe_D_space, Qe_steps = train_qlearning(e=0.3)
          plot_qlearning(Qe, Qe_D_space, Qe_steps, e=0.3)
```

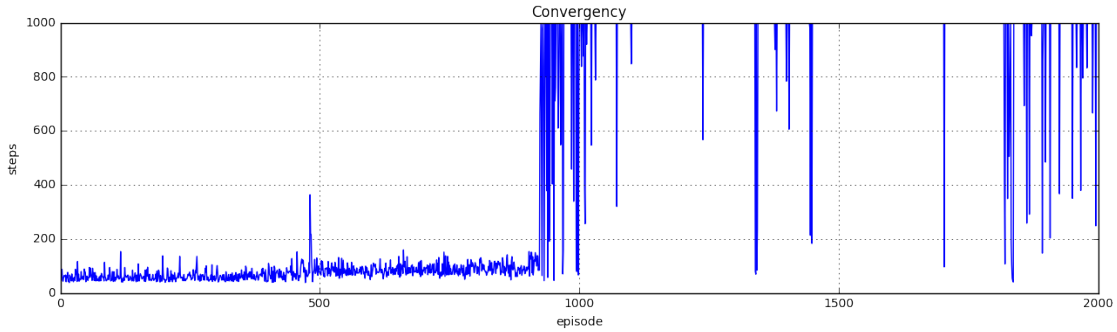




0.0.8 14.2 (e) fast learning rate (eta = 1)

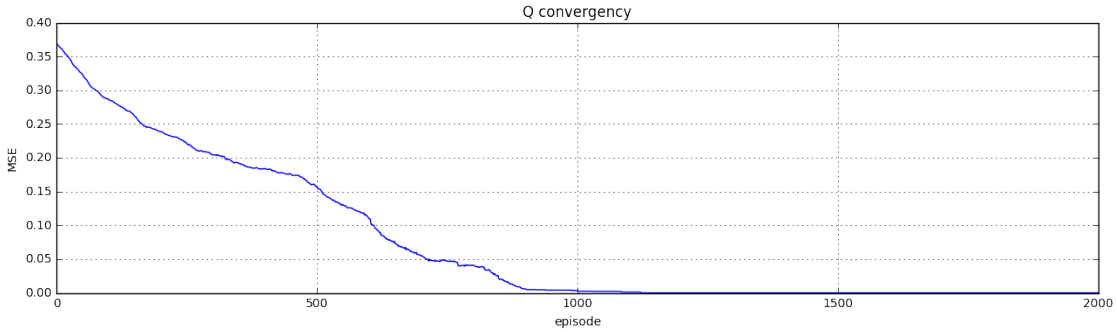
```
In [455]: Qeta, Qeta_D_space, Qeta_steps = train_qlearning(eta=1)
          plot_qlearning(Qeta, Qeta_D_space, Qeta_steps)
```

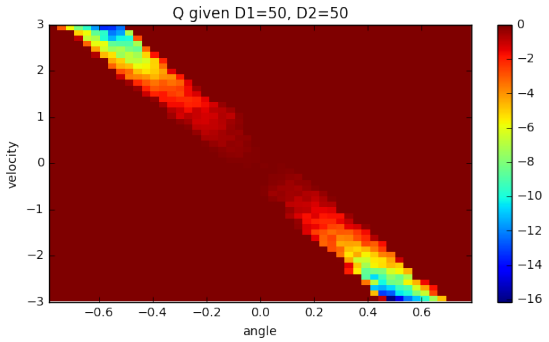


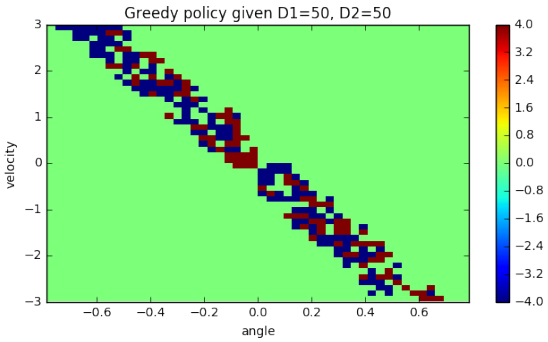



```
In [447]: Qeta2, Qeta_D_space2, Qeta_steps = train_qlearning(eta=1, check_converger
          diffs = [np.mean((x - Qeta2)**2) for x in Qetas2]
          plot(diffs, enum=True, title='Q convergency', labels=['episode', 'MSE'])
          plot_qlearning(Qeta2, Qeta_D_space2)
```

```
Episode 0 (D1=50, D2=50)
Episode 200 (D1=50, D2=50)
Episode 400 (D1=50, D2=50)
Episode 600 (D1=50, D2=50)
Episode 800 (D1=50, D2=50)
Episode 1000 (D1=50, D2=50)
Episode 1200 (D1=50, D2=50)
Episode 1400 (D1=50, D2=50)
Episode 1600 (D1=50, D2=50)
Episode 1800 (D1=50, D2=50)
```




Q given D1=50, D2=50
velocity
angle
3
2
1
0
−1
−2
−3
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0
−2
−4
−6
−8
−10
−12
−14
−16


Greedy policy given D1=50, D2=50
velocity
angle
3
2
1
0
−1
−2
−3
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
4.0
3.2
2.4
1.6
0.8
0.0
−0.8
−1.6
−2.4
−3.2
−4.0