# Robot Learning Assignment 01

Team members:

Roberto Cai Wu

Ramesh Kumar

In [155]:
```
import numpy as np
import matplotlib.pyplot as plt
import random
from random import randint
```

# Exercise 1.1

In [107]:
```
# To compute individual expected value of each arm
arms = np.array([[2,3],[-1,5],[1,5],[-2,4],[0,3],[2,6]])
expected_value_each_arm = np.mean(arms,axis=1)
for i in range(6):
    print('expected_value arm ' + str(i+1) + " = ",expected_value_each_arm[i
])
cumulative_expected_value = np.sum(expected_value_each_arm)
print('cumulative expected value',cumulative_expected_value)
```

```
expected_value arm 1 =  2.5
expected_value arm 2 =  2.0
expected_value arm 3 =  3.0
expected_value arm 4 =  1.0
expected_value arm 5 =  1.5
expected_value arm 6 =  4.0
cumulative expected value 14.0
```

The expected reward for sampling the 6 arms is:

$$E[R] = \sum_{k=0}^{N=5} r_k$$

, where

$$r_k$$

is the expected value for chosing arm k.

In this case we have:

$$R = r_0 + r_1 + r_2 + r_3 + r_4 + r_5 = 14.0$$

In [150]:
```
class Armed_Bandits(object):
    def __init__(self, bandits):
        self.bandits = bandits #Receives the intervals for each arm
        self.arms = len(bandits[:,0]) #Number of arms

    # Exercise 1.2

    # Return a random uniform number from the selected arm
    def sample_bandit(self, bandit_index):
        expected_value = np.random.uniform(self.bandits[bandit_index,0],self
.bandits[bandit_index,1])
        return expected_value

    #Average expected reward from taking uniform samples
    def average_reward(self,iterations):
        self.Q = np.zeros(self.arms)
        self.C_actions = np.zeros(self.arms)
        for i in range(iterations):
            arm_index = np.random.randint(0,self.arms)
            self.C_actions[arm_index] += 1
            sample = self.sample_bandit(arm_index)
            self.Q[arm_index] += (sample - self.Q[arm_index])/(self.C_action
s[arm_index]+1)
        best_arm = np.argmax(self.Q)
        best_reward = self.Q[best_arm]
        cumulative_reward = np.sum(self.Q)
        print("Arm with highest reward = ",best_arm," with reward = ",best_r
eward)
        print("Cumulative rewards = ",cumulative_reward)
        print(self.Q)
        print(self.C_actions)

    # Exercise 1.3

    def print_Q(self):
        percentage = 100*(self.C_actions/np.sum(self.C_actions[:]))

        print("\nRewards: ",self.Q)
        print("Number of actions: ",self.C_actions)
        print("Percentages: ",percentage)
        print(np.sum(self.C_actions))

    #Expected reward from taking an e-greedy rate
    def e_greedy(self, iterations, frequency, e):
        self.Q = np.zeros(self.arms)
        self.C_actions = np.zeros(self.arms)
        for i in range(iterations):
            # 90% exploitation rate
            if np.random.random() > e:
                arm_index = np.argmax(self.Q)
            # 10% exploration rate
            else:
                arm_index = np.random.randint(0,self.arms)
            self.C_actions[arm_index] += 1
            sample = self.sample_bandit(arm_index)
            self.Q[arm_index] += (sample - self.Q[arm_index])/(self.C_action
s[arm_index]+1)

            if ((i + 1) % frequency) == 0:
                self.print_Q()

    # Exercise 1.4

    #Expected reward adding a learning rate
    def third_arm_uniformly(self,iterations,frequency,e,alpha):
        self.Q = np.zeros(self.arms)
        self.C_actions = np.zeros(self.arms)
        for i in range(iterations):
```

# Exercise 1.2

In [151]:
```
reward_distribution = np.array([[2,3],
                                [-1,5],
                                [1,5],
                                [-2,4],
                                [0,3],
                                [2,6]])
bandits = Armed_Bandits(reward_distribution)
print('\n With 100 iterations:')
samples = 100
bandits.average_reward(samples)
print('\n With 1000 iterations:')
samples = 1000
bandits.average_reward(samples)
```

```
 With 100 iterations:
Arm with highest reward =  5  with reward =  3.64987684529
Cumulative rewards =  13.7677137739
[ 2.39178293  1.75197068  3.16878242  1.18274535  1.62255555  3.64987685]
[ 19.  13.  15.  18.  12.  23.]

 With 1000 iterations:
Arm with highest reward =  5  with reward =  3.98849971899
Cumulative rewards =  13.8512897643
[ 2.47302757  2.06342472  2.89355753  0.96597462  1.4668056   3.98849972]
[ 173.  183.  147.  170.  165.  162.]
```

We can see that our cumulative reward is different from our expected reward from Ex. 1.1 because 100 iterations are not enough to converge to expected reward. If we increse the number of iterations, it will come closer to the expected rewards.

# Exercise 1.3

In [153]:
```
samples = 1000
print_at = 100
e = 0.1
bandits.e_greedy(samples,print_at,e)
```

```
Rewards:  [ 2.49308515  1.11773078  2.47082835 -0.37679751  1.64625208  3.96
322189]
Number of actions:  [ 31.   2.   3.   1.   3.  60.]
Percentages:  [ 31.   2.   3.   1.   3.  60.]
100.0

Rewards:  [ 2.49308515  1.21364114  2.64649659  0.97825969  1.3974162   4.06
109198]
Number of actions:  [  31.    3.    6.    2.    7.  151.]
Percentages:  [ 15.5   1.5   3.    1.    3.5  75.5]
200.0

Rewards:  [ 2.48977484  1.21364114  2.99293676  0.97825969  1.3974162   4.03
000485]
Number of actions:  [  34.    3.    9.    2.    7.  245.]
Percentages:  [ 11.33333333   1.           3.           0.66666667   2.33333
333
  81.66666667]
300.0

Rewards:  [ 2.49453413  1.62334728  2.94956422  1.1127994   1.3974162   4.00
396897]
Number of actions:  [  36.    4.   10.    5.    7.  338.]
Percentages:  [  9.     1.     2.5    1.25   1.75  84.5 ]
400.0

Rewards:  [ 2.49481667  0.85149908  2.94956422  1.18812529  1.35581282  3.98
568296]
Number of actions:  [  38.    7.   10.    6.   10.  429.]
Percentages:  [  7.6   1.4   2.    1.2   2.   85.8]
500.0

Rewards:  [ 2.49274703  1.4877718   3.23334982  1.62004036  1.36266219  3.95
45445 ]
Number of actions:  [  41.   10.   14.    9.   11.  515.]
Percentages:  [  6.83333333   1.66666667   2.33333333   1.5          1.83333
333
  85.83333333]
600.0

Rewards:  [ 2.48540142  1.89892538  3.17591304  1.94480829  1.20167126  3.92
558516]
Number of actions:  [  44.   12.   16.   11.   14.  603.]
Percentages:  [  6.28571429   1.71428571   2.28571429   1.57142857   2.
     86.14285714]
700.0

Rewards:  [ 2.49541177  1.81344415  3.17591304  1.87291406  1.34142261  3.91
730631]
Number of actions:  [  47.   13.   16.   12.   18.  694.]
Percentages:  [  5.875   1.625   2.      1.5     2.25   86.75 ]
800.0

Rewards:  [ 2.49227146  1.81344415  3.281386    1.60476038  1.40334237  3.90
823125]
Number of actions:  [  50.   13.   19.   14.   19.  785.]
Percentages:  [  5.55555556   1.44444444   2.11111111   1.55555556   2.11111
111
  87.22222222]
900.0

Rewards:  [ 2.50366081  1.89894946  3.21856378  1.60476038  1.44771943  3.92
464467]
Number of actions:  [  52.   16.   24.   14.   20.  874.]
Percentages:  [  5.2   1.6   2.4   1.4   2.   87.4]
1000.0
```

From the exercise we can observe that taking an e-greedy approach quickly recognizes arm 6 as the optimal reward.

## Exercise 1.4

In [154]:
```
samples = 1000
print_at = 100
e = 0.1
alpha = 0.01
bandits.third_arm_uniformly(samples,print_at,e,alpha)
```

```
Rewards:  [ 1.54491342  0.06676239  0.          0.03814428  0.05344189  0.07
578076]
Number of actions:  [ 90.   2.   0.   1.   5.   2.]
Percentages:  [ 90.   2.   0.   1.   5.   2.]
100.0

Rewards:  [ 2.11924825  0.20518657  0.08489823  0.05439157  0.05434393  0.10
604299]
Number of actions:  [ 181.    6.    2.    2.    6.    3.]
Percentages:  [ 90.5   3.    1.    1.    3.    1.5]
200.0

Rewards:  [ 2.34780504  0.22262436  0.08489823  0.05439157  0.08173925  0.12
873107]
Number of actions:  [ 276.    7.    2.    2.    9.    4.]
Percentages:  [ 92.          2.33333333   0.66666667   0.66666667   3.
      1.33333333]
300.0

Rewards:  [ 2.43247169  0.27905352  0.10909016  0.05439157  0.11454337  0.21
514756]
Number of actions:  [ 367.   10.    3.    2.   12.    6.]
Percentages:  [ 91.75   2.5    0.75   0.5    3.     1.5 ]
400.0

Rewards:  [ 2.47664069  0.30835028  0.13819775  0.0284417   0.13252977  0.26
957362]
Number of actions:  [ 458.   12.    4.    4.   14.    8.]
Percentages:  [ 91.6   2.4   0.8   0.8   2.8   1.6]
500.0

Rewards:  [ 2.48531833  0.30257881  0.43780551  0.0284417   0.13426864  0.32
413685]
Number of actions:  [ 551.   13.    8.    4.   15.    9.]
Percentages:  [ 91.83333333   2.16666667   1.33333333   0.66666667   2.5
      1.5       ]
600.0

Rewards:  [ 2.48092838  0.29030607  0.57087771  0.06261835  0.14654593  0.41
205994]
Number of actions:  [ 642.   14.   10.    6.   16.   12.]
Percentages:  [ 91.71428571   2.          1.42857143   0.85714286   2.28571
429
   1.71428571]
700.0

Rewards:  [ 2.4658027   0.34684495  0.57087771  0.06261835  0.14654593  0.46
979247]
Number of actions:  [ 738.   16.   10.    6.   16.   14.]
Percentages:  [ 92.25   2.     1.25   0.75   2.     1.75]
800.0

Rewards:  [ 2.45433359  0.37328475  0.64441865  0.05832265  0.16402196  0.46
979247]
Number of actions:  [ 832.   18.   11.    8.   17.   14.]
Percentages:  [ 92.44444444   2.          1.22222222   0.88888889   1.88888
889
   1.55555556]
900.0

Rewards:  [ 2.44684267  0.37328475  0.70727947  0.09710139  0.16402196  0.59
530486]
Number of actions:  [ 924.   18.   12.   11.   17.   18.]
Percentages:  [ 92.4   1.8   1.2   1.1   1.7   1.8]
1000.0
```

From 1000 iterations is not possible to notice a change, but if we increase the learning rate or the number of iterations, we can see that the reward for the third arm will slowly increase. Since the learning rate is very slow, it initially fails to recognize arm 6 as the best arm.

## Exercise 1.5

In [146]:
```
samples = 1000
print_at = 100
e = 0.1
bandits.optimistic_intiallization(samples,print_at,e,alpha)
```

```
Rewards:  [ 9.1460399   9.05407557  9.13532991  9.1402683   9.14047616  9.10
418347]
Number of actions:  [ 12.  15.  35.  11.  11.  16.]
Percentages:  [ 12.  15.  35.  11.  11.  16.]
100.0

Rewards:  [ 8.37082428  8.37018609  8.40133363  8.32877368  8.35074615  8.36
017909]
Number of actions:  [ 24.  24.  77.  21.  22.  32.]
Percentages:  [ 12.   12.   38.5  10.5  11.   16. ]
200.0

Rewards:  [ 7.81718793  7.77105423  7.8641895   7.83495586  7.81781261  7.86
131371]
Number of actions:  [  34.   34.  132.   27.   30.   43.]
Percentages:  [ 11.33333333  11.33333333  44.          9.          10.
     14.33333333]
300.0

Rewards:  [ 7.40490364  7.40336803  7.45644529  7.45648522  7.40331951  7.44
878322]
Number of actions:  [  42.   40.  194.   32.   37.   55.]
Percentages:  [ 10.5   10.    48.5    8.     9.25  13.75]
400.0

Rewards:  [ 7.21670985  7.1532649   7.24265747  7.17958278  7.23068591  7.21
510783]
Number of actions:  [  46.   45.  269.   37.   40.   63.]
Percentages:  [  9.2   9.   53.8   7.4   8.   12.6]
500.0

Rewards:  [ 6.97478672  6.98274777  7.06245285  6.94312021  7.03490721  7.02
22882 ]
Number of actions:  [  51.   49.  348.   41.   43.   68.]
Percentages:  [  8.5          8.16666667  58.          6.83333333   7.16666
667
  11.33333333]
600.0

Rewards:  [ 6.83933706  6.87995323  6.99446477  6.85317733  6.84075258  6.97
283102]
Number of actions:  [  54.   51.  435.   43.   47.   70.]
Percentages:  [  7.71428571   7.28571429  62.14285714   6.14285714   6.71428
571  10.        ]
700.0

Rewards:  [ 6.7158872   6.67937351  6.97434241  6.61884168  6.74350814  6.88
559819]
Number of actions:  [  57.   55.  520.   46.   49.   73.]
Percentages:  [  7.125   6.875  65.      5.75    6.125   9.125]
800.0

Rewards:  [ 6.62978728  6.6173423   7.00384964  6.47342693  6.64626018  6.88
559819]
Number of actions:  [  59.   56.  613.   48.   51.   73.]
Percentages:  [  6.55555556   6.22222222  68.11111111   5.33333333   5.66666
667
   8.11111111]
900.0

Rewards:  [ 6.54503904  6.57929354  6.97805827  6.39914874  6.48062454  6.84
033098]
Number of actions:  [  61.   57.  705.   49.   54.   74.]
Percentages:  [  6.1   5.7  70.5   4.9   5.4   7.4]
1000.0
```

We can see that the selected arm is constantly changing because the higher initial reward is lowering after every selection. But overall, it will still converge slowly to the optimal arm.

In [ ]:

```
```

In [ ]:

```
```