HW-4

Problem-1

1 i)

In [49]:

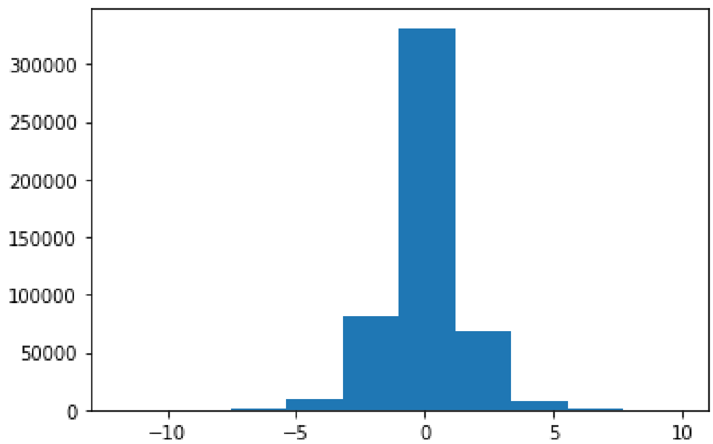
```
import math
import numpy as np
import random

def target_distribution(x):
    return 0.5 * math.exp(-abs(x))

def proposal_normal_distribution(mean, sd, num_of_samples):
    return np.random.normal(mean, sd, num_of_samples)[0]


x = [0] * 500000
x[0] = 0

for i in range(1,500000):
    current_x = x[i-1]
    proposed_x = proposal_normal_distribution(current_x, 1, 1)

    A = target_distribution(proposed_x)/target_distribution(current_x)

    if(random.uniform(0.0,1.0) < A):
        x[i] = proposed_x        # accept move with probabily min(1,A)
    else:
        x[i] = current_x         # otherwise "reject" move, and stay where we are
```

In [50]:

```
plt.hist(x)
```

Out[50]:

```
(array([5.00000e+01, 1.35000e+02, 1.05600e+03, 9.34100e+03, 8.08600e+04,
        3.31289e+05, 6.85180e+04, 7.69300e+03, 9.89000e+02, 6.90000e+01]),
 array([-11.8811912 ,  -9.70696661,  -7.53274203,  -5.35851744,
         -3.18429286,  -1.01006827,   1.16415631,   3.3383809 ,
          5.51260548,   7.68683007,   9.86105466]),
 <BarContainer object of 10 artists>)
```



The sampled values seem to converge to a distribution centered at 0 and resembles like an standard double exponential distribution.

1 ii)

In [81]:

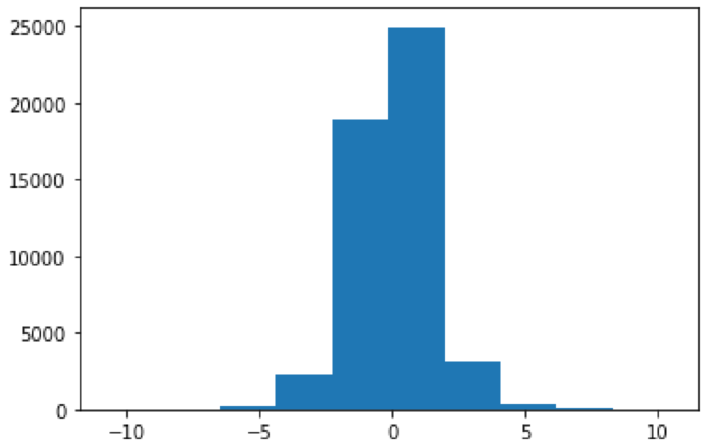
```
x2 = [0]*50000

for i in range(50000):
    u2 = random.uniform(0.0,1.0)
    if u2 < 0.5:
        x2[i] = math.log(2*u2)
    else:
        x2[i] = -1*math.log(2-2*u2)
```

In [101]:

```
plt.hist(x2)
```

Out[101]:

```
(array([3.0000e+00, 2.7000e+01, 2.6000e+02, 2.2290e+03, 1.8963e+04,
        2.4969e+04, 3.1110e+03, 3.8000e+02, 5.3000e+01, 5.0000e+00]),
 array([-10.69635939,  -8.58575746,  -6.47515554,  -4.36455361,
         -2.25395168,  -0.14334976,   1.96725217,   4.0778541 ,
          6.18845602,   8.29905795,  10.40965988]),
 <BarContainer object of 10 artists>)
```



The curve generated using inverse sampling is comparable to the one generated in part-a) and can be seen following the double exponential distribution

Problem 2

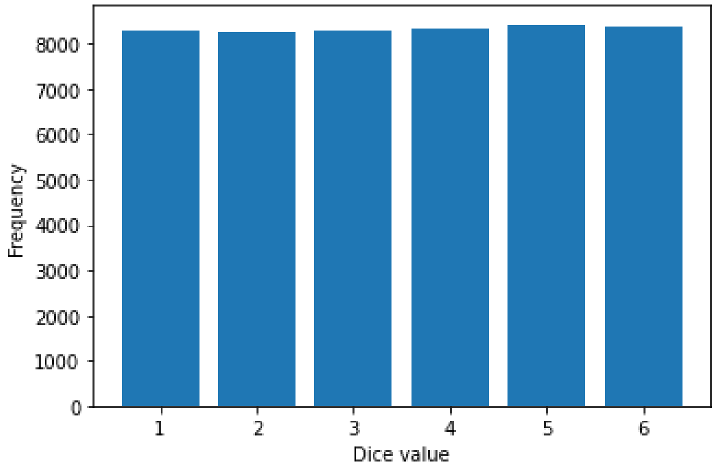
In [72]:

```
def target_distribution(x):
    return 1/6

def transition_probabilities(state, given_state):
    if given_state == 1:
        if state ==2: return 1
        else: return 0

    elif given_state == 2:
        if state == 1 or state == 3: return 0.5
        else: return 0

    elif given_state == 3:
        if state == 2 or state == 4: return 0.5
        else: return 0

    elif given_state == 4:
        if state == 3 or state == 5: return 0.5
```

```
            else: return 0

        elif given_state == 5:
            if state == 4 or state == 6: return 0.5
            else: return 0

        else:
            if state == 5: return 1
            else: return 0

def sampling_from_proposal_distr(current_state):
    if current_state == 1: return 2
    elif current_state == 2: return random.choice([1, 3])
    elif current_state == 3: return random.choice([2, 4])
    elif current_state == 4: return random.choice([3, 5])
    elif current_state == 5: return random.choice([4, 6])
    else: return 5

y = [0] * 50000
y[0] = 2

for i in range(1,50000):
    current_y = y[i-1]
    proposed_y = sampling_from_proposal_distr(current_y)

    A = transition_probabilities(current_y, proposed_y)/transition_probabilities(proposed
_y, current_y)

    if(random.uniform(0.0,1.0) < A):
        y[i] = proposed_y       # accept move with probably min(1,A)
    else:
        y[i] = current_y        # otherwise "reject" move, and stay where we are
```

In [73]:

```
import collections
c = collections.Counter(y)
c = sorted(c.items())

freq = [i[1] for i in c]
dice_value = [i[0] for i in c]

plt.bar(dice_value, freq)
plt.xlabel("Dice value")
plt.ylabel("Frequency")

plt.show()
```



The sample distribution of dice values shows all the values occuring almost the same number of times. Above is the frequency distribution of dice values. It resembles the target distribution wherein each dice value had equal probability.

In [ ]: