参考实现

说明：这**只是一个参考**，你也可以按自己的想法进行实现（yep，你可以把这份文档拖进回收站，自己写）， 但需满足实验说明文档的相关要求。

In [1]:

```
import numpy as np
from matplotlib import pyplot as plt

# %matplotlib inline
# %config InlineBackend.figure_formats = ['svg']
```

In [2]:

```
def calculate_target(alpha, x):
    return -(np.log2(alpha+x).sum())

def fill_water(alpha, total_water, precision, track=False):
    # implement your algorithm here
    # ...
    # return your solution
```

In [3]:

```
# you can enlarge these parameters to check how efficient
# your algorithm is(not required by this lab),
# I limit it to be very small to help you to begin with.
# And note that small value is good for visualization.
alpha_range = [0.0, 1.0]
total_water = 1.0
dimension = 10

precision = 1e-6

alpha = np.random.uniform(low=alpha_range[0],
                          high=alpha_range[1],
                          size=(dimension, 1))
print(alpha)
```

```
[[0.21889905]
 [0.13042259]
 [0.04632022]
 [0.89854737]
 [0.06499838]
 [0.08789615]
 [0.3975796 ]
 [0.71104049]
 [0.30180927]
 [0.52901178]]
```

In [4]:

```
x = fill_water(alpha=alpha,
          total_water=total_water,
          precision=precision)
print(x)
print(x.sum())

horizontal_line = (alpha + x).min()
print(horizontal_line)
```

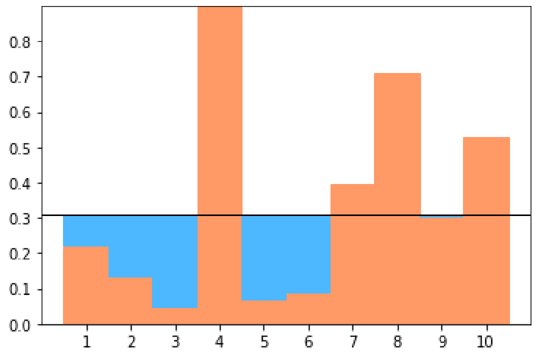
```
[[0.08949205]
 [0.17796851]
 [0.26207088]
 [0.        ]
 [0.24339272]
 [0.22049495]
 [0.        ]
 [0.        ]
 [0.00658184]
 [0.        ]]
1.0000009362768902
0.30839110058200936
```

In [5]:

```
def visualize_water(alpha, x, horizontal_line):
    alpha = alpha.squeeze()
    x = x.squeeze()

    x_range = range(1, x.shape[0]+1)
    plt.xticks(x_range)
    plt.bar(x_range, alpha, color='#ff9966',
            width=1.0, edgecolor='#ff9966')
    plt.bar(x_range, x, bottom=alpha, color='#4db8ff', width=1.0)

    plt.axhline(y=horizontal_line,linewidth=1, color='k')

    plt.show()
```

In [6]:

```
visualize_water(alpha, x, horizontal_line)
```



迭代过程查看

这部分**不是必须**的，但是当你的代码出现bug的时候，这些小技巧可能会有帮助。

In [7]:

```python
x, targets, errors = fill_water(alpha=alpha,
          total_water=total_water,
          precision=precision,
          track=True)
print(x)
print(targets)
print(errors)

# how much iterations your algorithm need to
# satisfy the precsion required?
print('iteration:', len(errors))
```

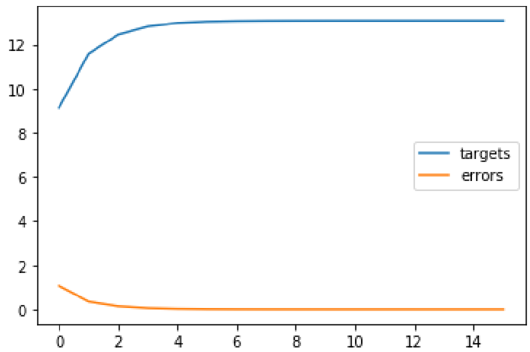
```
[[0.08949205]
 [0.17796851]
 [0.26207088]
 [0.        ]
 [0.24339272]
 [0.22049495]
 [0.        ]
 [0.        ]
 [0.00658184]
 [0.        ]]
[9.13766963174835, 11.583796599594791, 12.44940811760928, 12.821437975665376, 12.974845465178005, 13.036977890086906, 1
3.061956273275952, 13.07196784340105, 13.07597571587889, 13.07757938460351, 13.078220935290776, 13.078477568879832, 13.
078580224445885, 13.078621287013183, 13.078637712094647, 13.078644282135958]
[1.059111295205447, 0.3487903215987713, 0.13951612863950835, 0.05580645145580343, 0.022322580582321283, 0.0089290322329
28513, 0.0035716128931717606, 0.0014286451572687042, 0.0005714580629074817, 0.00022858322516294827, 9.143329006500167e-
05, 3.657331602635594e-05, 1.4629326410497967e-05, 5.8517305643324136e-06, 2.3406922255997387e-06, 9.362768902398955e-0
7]
iteration: 16
```

In [8]:

```python
def visualize_targets_and_errors(targets, errors):
    x = range(len(targets))
    plt.plot(x, targets, label='targets')
    plt.plot(x, errors, label='errors')

    plt.legend(loc='best')

    plt.show()
```

In [9]:

```python
visualize_targets_and_errors(targets, errors)
```



这个target在上升(或下降)是因为，迭代过程中，并不满足$1^T x = 1$， 目前的算法直接迭代到最优解附近，但是在迭代过程中不满足$1^T x = 1$。

注：这是我的实现，你的实现可能不会出现这样的情况。

查看你的最优解

一顿操作之后，你可能想确认你的结果是不是正确的，你可能可以参考一些“标准答案”， 但如果“标准答案”不好寻找的时候，也可以简单的测试一下。

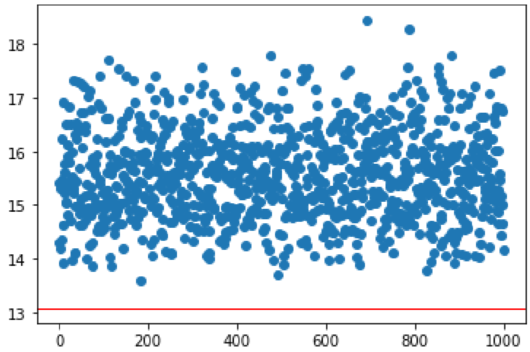
比如，我们的实现的算法应该比随机生成的解要好，我们来试试。

In [10]:

```
# let's play with monkey search, i.e., random search

def monkey_search(alpha):
    # random return x s.t. 1^T x = 1 and x > 0
    while True:
        monkey_solution = np.random.dirichlet(np.ones(10),size=1).reshape(-1,1)
        if np.less(monkey_solution, 0).any():
            continue
        return monkey_solution

def visualize_monkey_search(alpha, monkey_amount, optimal):
    monkey_solutions = [calculate_target(alpha, monkey_search(alpha)) \
                        for _ in range(monkey_amount)]

    plt.scatter(range(monkey_amount), monkey_solutions)
    plt.axhline(y=optimal,linewidth=1, color='r')

    plt.show()
```

In [11]:

```
# you can verify that a monkey can never cross the optimal line
optimal_line = targets[len(targets)-1]
visualize_monkey_search(alpha, 1000, optimal_line)
```



1000只猴子敲不出莎士比亚。