A1_Intro_and_Random_Number_Generation

February 27, 2019

1 Assignment - Random Number Generation

1.0.1 Year 2018-2019 - Semester II

1.0.2 CCE3502

Developed by - Adrian Muscat, 2019

1.1 #### Lab Assistant - Brandon Birmingham

1.2 # Matthew Vella, 428698M, BSc CS, Yr II

1.3 First few cells are non-graded exercises and practice in lab. These are followed by 5 graded questions

```
In [35]: # import useful libraries
         import numpy as np
         import matplotlib.pyplot as plt
         import scipy.stats as stats
         import time as time
         #this line plots graphs in line
         %matplotlib inline
```

```
In [15]: ### examples in numpy and vector notation
         a=np.array([0,1,2,3,4], dtype=float)
         b=np.array([5,6,7,8,9], dtype=float)
         print(a)
         print(b)
         print(a*b)
         c=np.arange(4,20,3.)
         print(c)
```

```
[0. 1. 2. 3. 4.]
[5. 6. 7. 8. 9.]
[ 0.  6. 14. 24. 36.]
[ 4.  7. 10. 13. 16. 19.]
```

```
In [16]: #slicing and broadcasting examples
         c2 = c[:-1]
```

```
c3 = np.array([2])
c2.shape, c3.shape

c2+c3
```

Out[16]: array([6., 9., 12., 15., 18.])

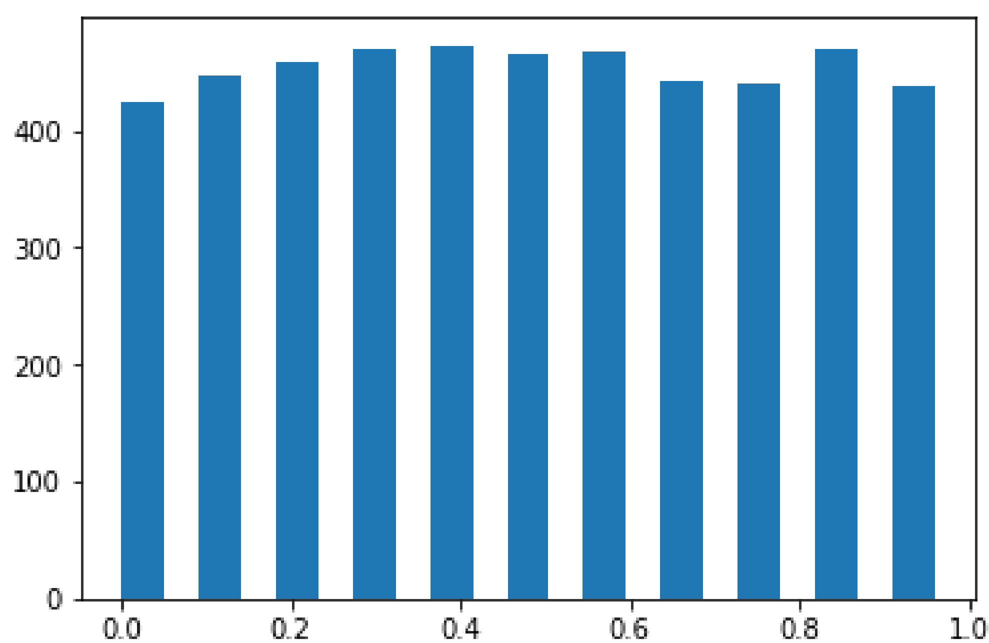
In [17]:
```
### generating RNs using numpy functions
max(np.random.randint(1,9, 10000000))
```

Out[17]: 8

1.4 histogram and plots

In [19]:
```
### histogram and plots
r = np.random.random(size=5000)
h = np.histogram(r,bins=11)
print(h[0])
print(h[1])
print
plt.bar(h[1][:-1], h[0], width = .05, align='edge')
plt.show()
```

```
[425 447 458 471 473 465 468 443 441 471 438]
[1.41061033e-05 9.09205697e-02 1.81827033e-01 2.72733497e-01
 3.63639961e-01 4.54546424e-01 5.45452888e-01 6.36359352e-01
 7.27265815e-01 8.18172279e-01 9.09078742e-01 9.99985206e-01]
```



```
In [20]: #Timing code
         start = time.clock()
         for i in range(10000):
             for j in range(10000):
                 c = 8.89*7.76
         end = time.clock()
         print (end-start), 'sec'
```

```
C:\Users\matth\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: DeprecationWarning: time.cl
```

```
10.253124507999928
```

```
C:\Users\matth\Anaconda3\lib\site-packages\ipykernel_launcher.py:6: DeprecationWarning: time.cl
```

```
Out[20]: (None, 'sec')
```

1.5 comparison of variables - ANOVA

```
In [21]: ### ANOVA comparison
         # A study on Perceived difficulty [from Heiman-92]
         # We told three samples of five subjects each that some
         # math problems were easy (A1), of medum difficulty (A2), or difficult (A3)
         # we measured the number of problems they correctly solved
         # we want to check for significant difference between these g# measurements(scores ob
         # are given below for the three samples
         A1 = np.array([9,12,4,8,7],dtype=float)
         A2 = np.array([4,6,8,2,10],dtype=float)
         A3 = np.array([1,3,4,5,2],dtype=float)
         print A1.mean(), A1.var()
         print A2.mean(), A2.var()
         print A3.mean(), A3.var()
         print stats.f_oneway(A1,A2)
         print stats.f_oneway(A1,A3)
         print stats.f_oneway(A2,A3)
         print stats.f_oneway(A1,A1)
         A = [A1,A2,A3]
         _ = plt.boxplot(A)
```

```
  File "<ipython-input-21-d0dc1ac42267>", line 11
print A1.mean(), A1.var()
```

```
         ^
SyntaxError: invalid syntax
```

```
In [ ]: A = [A1,A2,A3]
        _ = plt.boxplot(A)
```

```
In [ ]: # extra- anova calculations
        #genenrate A1, A2, normally distributed numbers
        #plot box plots
        # compute anova F
```

1.6 Chi square goodness of fit

```
In [ ]: # generating Table for chi-square
        print("Critical value")

        for k in range(1,11):
            crit = stats.chi2.ppf(q = 1.0-np.array([0.01, .05]), # Find the critical value for
                                  df = k)   # Df = number of variable categories - 1

            print(k, crit)
```

```
In [ ]: #f_observed=np.array([30,20,35,36,17,14,29,20,18,31],dtype=float)  # reject
        f_observed=np.array([30,25,25,30,17,14,29,20,18,31],dtype=float)  # assume similar

        N=len(f_observed)
        f_expected = np.full_like(f_observed,np.sum(f_observed)/N)
        print f_observed
        print f_expected
        #
        chi_2_obt = np.sum(((f_expected-f_observed)**2.0)/f_expected)
        print 'chi_2_obt=',chi_2_obt
        #
        chi_2=stats.chi2.ppf(q=1.0-.05,df=N-1)
        print 'chi_2 from table = ', chi_2
        #
        if chi_2_obt>chi_2:
            print "We reject the hypothesis that the two distributions are similar with a conf
        else:
            print "We assume that the distributions are similar with a confidence of 95%"
```

1.7 RNG - analytical inversion method

```
In [ ]: # analytical inversion method
        # generate exponentially distributed numbers using the analytical method developed in
        # generate same using numpy.random exponential generator
        # test which one is faster to compute
```

```
In [ ]: # generating using analytical inversion technique.
        # use formula from class for exp distribution
        # use chi-square to compare to numpy's exp generator
```

1.8 RNG - accept-reject method

```
In [ ]: # accept-reject method
        # code algorithm for accept-reject RNG
```

2 GRADED QUESTIONS BELOW: 5 questions, Total = 40, weight =10%

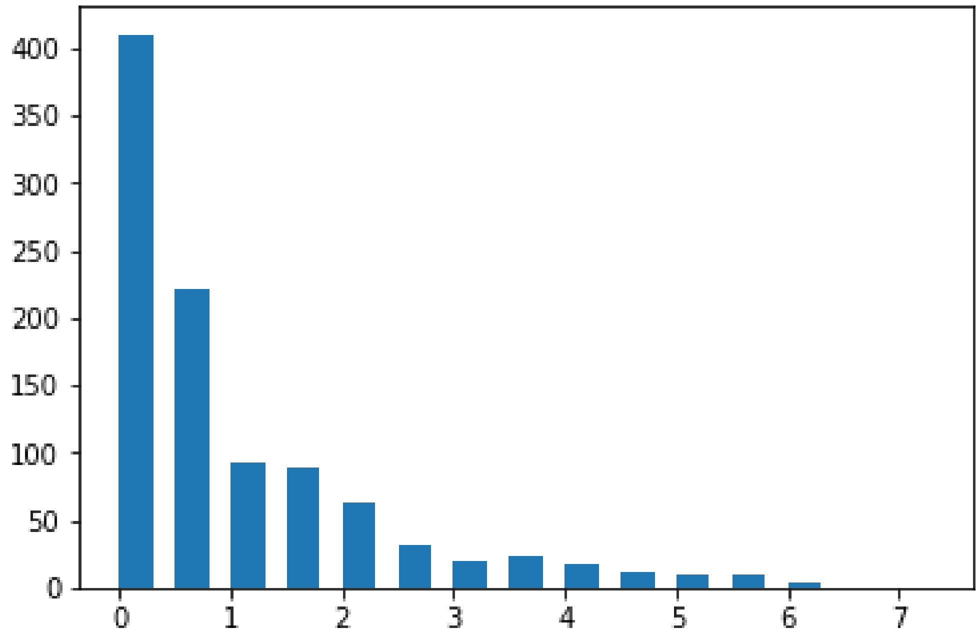
2.1 Load data as below [0 marks]

```
In [22]: in_file = open("rand_data.txt",'rb')
         f = in_file.readlines()
         in_file.close()
         data = np.zeros(len(f),dtype=float)
         for i in range(data.shape[0]):
             data[i]=float(f[i])
```

2.2 Compute a histogram for data = H_data [5 marks]

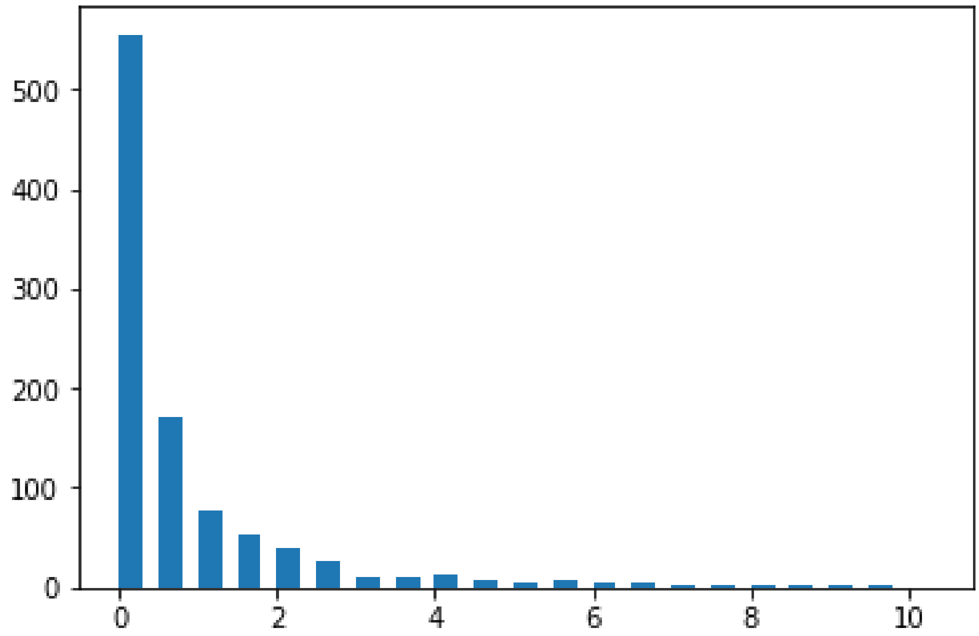
2.2.1 use bins of width = 0.5

```
In [23]: bins = np.arange(0,8,.5)
         h = np.histogram(data, bins)
         plt.bar(h[1][:-1],h[0],width=.3,align='edge')

Out[23]: <BarContainer object of 15 artists>
```



2.3 Manually, fit an exponential function to the histogram [5 marks]

```
In [24]: r = np.random.exponential(size=len(data))
         bins = np.arange(0,11,.5)
         h = np.histogram(data*r, bins)
         plt.bar(h[1][:-1],h[0],width=.3,align='edge')

Out[24]: <BarContainer object of 21 artists>
```



2.4 Generate 1000 numbers under the exponential distribution and calculate goodness of fit statistic (compared to H-data) [10 marks]

In [27]:
```
random = np.random.exponential(size=1000)

f_observed = random

N=len(f_observed)
f_expected = data

chi_2_obt = np.sum(((f_expected - f_observed)**2.0)/f_expected)
print('chi_2_obt =',chi_2_obt)

chi_2=stats.chi2.ppf(q=1.0-.05,df=N-1)
print('chi_2 from table =', chi_2)

if chi_2_obt>chi_2:
    print("We reject the hypothesis that the two distributions are similar with a cond
else:
    print("We assume the two distributions are similar with a confidence of 95%")
```

```
chi_2_obt = 24793.10004235386
chi_2 from table = 1073.6426506574246
We reject the hypothesis that the two distributions are similar with a condifence of 95%
```

2.5 Generate another set of 1000 numbers using accept-reject under H_data graph, and calculate goodness of fit statistic (compared to H-data) [10 marks]

In [31]:
```
j = 0
k = 0
random_data = np.random.exponential(size=1000)
for i in range(len(data)):
    if random_data[i]<data[i]:
        j = j + 1
    else:
        k = k + 1

print(j,"numbers have been accepted")
print(k,"numbers have been rejected")
```

```
503 numbers have been accepted
497 numbers have been rejected
```

2.6 In your application, accuracy is more important than speed of computation. Which generator is the most suitable, and why? [10 marks]

In [34]:
```
# exponential distribution
start = time.clock()

random = np.random.exponential(size=1000)

f_observed = random

N=len(f_observed)
f_expected = data

chi_2_obt = np.sum(((f_expected - f_observed)**2.0)/f_expected)
print('chi_2_obt =',chi_2_obt)

chi_2=stats.chi2.ppf(q=1.0-.05,df=N-1)
print('chi_2 from table =', chi_2)

if chi_2_obt>chi_2:
    print("we reject the hypothesis that the two distributions are similar with a conc
else:
    print("We assume the two distributions are similar with a confidence of 95%")

end = time.clock()
print((end-start), 'sec')


# accept-reject method
```

```
start = time.clock()

j = 0
k = 0
random_data = np.random.exponential(size=1000)
for i in range(len(data)):
    if random_data[i]<data[i]:
        j = j + 1
    else:
        k = k + 1

print(j,"numbers have been accepted")
print(k,"numbers have been rejected")

end = time.clock()
print (end-start), 'sec'
```

```
chi_2_obt = 10250.715054267439
chi_2 from table = 1073.6426506574246
we reject the hypothesis that the two distributions are similar with a condifence of 95%
0.0017128639997281425 sec
497 numbers have been accepted
503 numbers have been rejected
0.0015101049998520466


C:\Users\matth\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: DeprecationWarning: time.cl

C:\Users\matth\Anaconda3\lib\site-packages\ipykernel_launcher.py:22: DeprecationWarning: time.c
C:\Users\matth\Anaconda3\lib\site-packages\ipykernel_launcher.py:28: DeprecationWarning: time.c
C:\Users\matth\Anaconda3\lib\site-packages\ipykernel_launcher.py:42: DeprecationWarning: time.c
```

```
Out[34]: (None, 'sec')
```

```
In [ ]: The faster method is the accept-reject method.
```