Solution - Task 06

October 9, 2018

1. Import numpy as np. Try np.arange(n), .shape, .reshape(r,s), .ndim, np.zeros(r,s), np.ones(r,s), .linspace(a, b, n). np.eye(n), .min(), .max().

```
In [3]: import numpy as np
        A = np.arange(16)
        print "np.arange(16) :", A

np.arange(16) : [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
```

```
In [4]: print "Shape is", A.shape

Shape is (16,)
```

```
In [6]: print "We reshape it as 4 x 4 matrix", A.reshape(4,4)

We reshape it as 4 x 4 matrix [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
```

```
In [7]: print "Dimension is", A.ndim

Dimension is 1
```

```
In [8]: print "A 2 x 3 zero matrix is", np.zeros((2,3))

A 2 x 3 zero matrix is [[ 0.  0.  0.]
 [ 0.  0.  0.]]
```

```
In [9]: print "A 2 x 3 matrix with all entries equal to 1 is", np.ones((2,3))

A 2 x 3 matrix with all entries equal to 1 is [[ 1.  1.  1.]
 [ 1.  1.  1.]]
```

```
In [10]: print "np.linspace(0,45,10) :", np.linspace(0,45,10)
```

```
np.linspace(0,45,10) : [  0.   5.  10.  15.  20.  25.  30.  35.  40.  45.]
```

```
In [11]: print "np.eye(3)", np.eye(3)
```

```
np.eye(3) [[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

```
In [12]: print "A.min :", A.min()
```

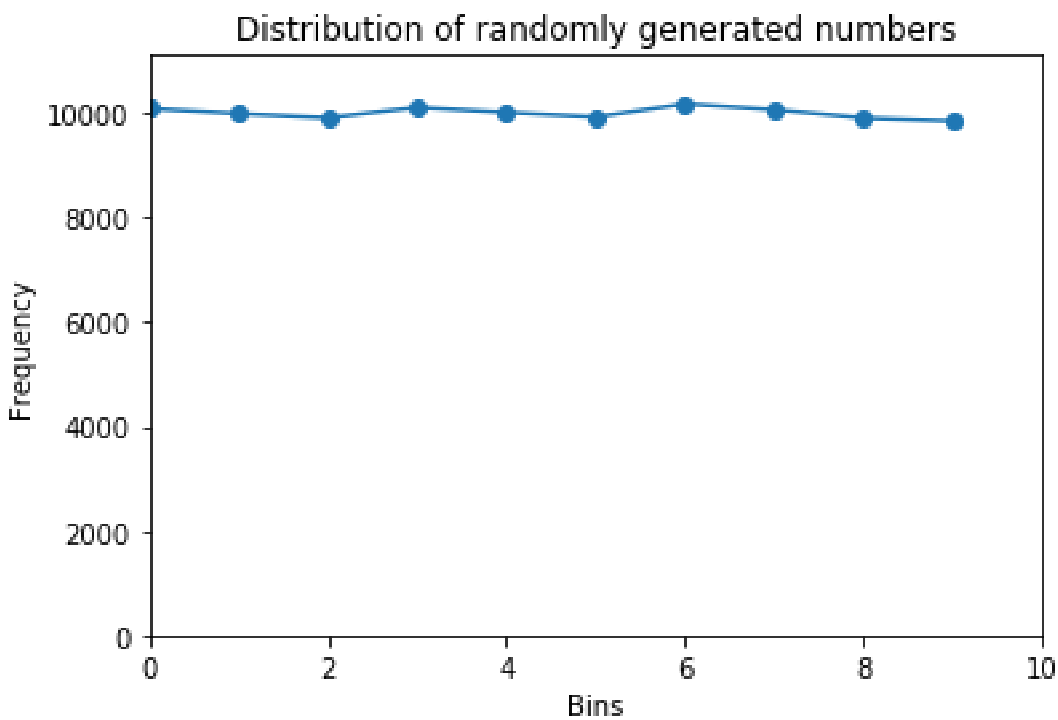
```
A.min : 0
```

```
In [13]: print "A.max :", A.max()
```

```
A.max : 15
```

2. Write a function randvect(n) that returns a one dimensional array of size n, whose entries are random numbers between 0 and 1. Use your plotting skills to guess if these entries follow a uniform distribution, as n increases.

```
In [32]: import numpy as np

         def randvect(n):
             return np.array(np.random.random(n))

         import matplotlib.pyplot as pl
         n = 100000
         Y = randvect(n)
         freqbin = np.zeros(10)
         for i in range(0,n):
             j = int(np.floor(Y[i]*10))
             freqbin[j] = freqbin[j] + 1
         pl.title("Distribution of randomly generated numbers")
         pl.axis([0,10,0,n/9])
         pl.xlabel('Bins')
         pl.ylabel('Frequency')
         pl.plot(freqbin, 'o-')
         pl.show()
         print "The distribution of 100000 random numbers in 10 equal sized bins is
```




```
The distribution of 100000 random numbers in 10 equal sized bins is
[ 10084.   9987.   9906.  10107.  10009.   9922.  10175.  10062.   9904.
   9844.]
```

3. Write a function toss(p) that simulates a coin toss. That is, it returns H with probability p and T with probability $1 - p$. Now, n such coins are tossed and for $r \leq n$ the integer P(r) denotes the number of coins where H appears. Estimate P(r) by conducting this experiment 10000 times.

```
In [37]: import numpy as np

         def toss(p):
             a = np.random.random()
             if a < p:
                 return 'H'
             else:
                 return 'T'

         def P(n, p):
             headcount = 0
             for i in range(n):
                 if toss(p) == 'H':
                     headcount = headcount+1
```

```
        return headcount

    n = 1000
    p = 0.5
    overall = 0
    for i in range(10000):
        overall = overall + P(n, p)
    print "Estmated P(r), for n = ", n ,"and p = ", p, "is", float(overall)/10
```

```
Estmated P(r), for n =  1000 and p =  0.5 is 4998.143
```

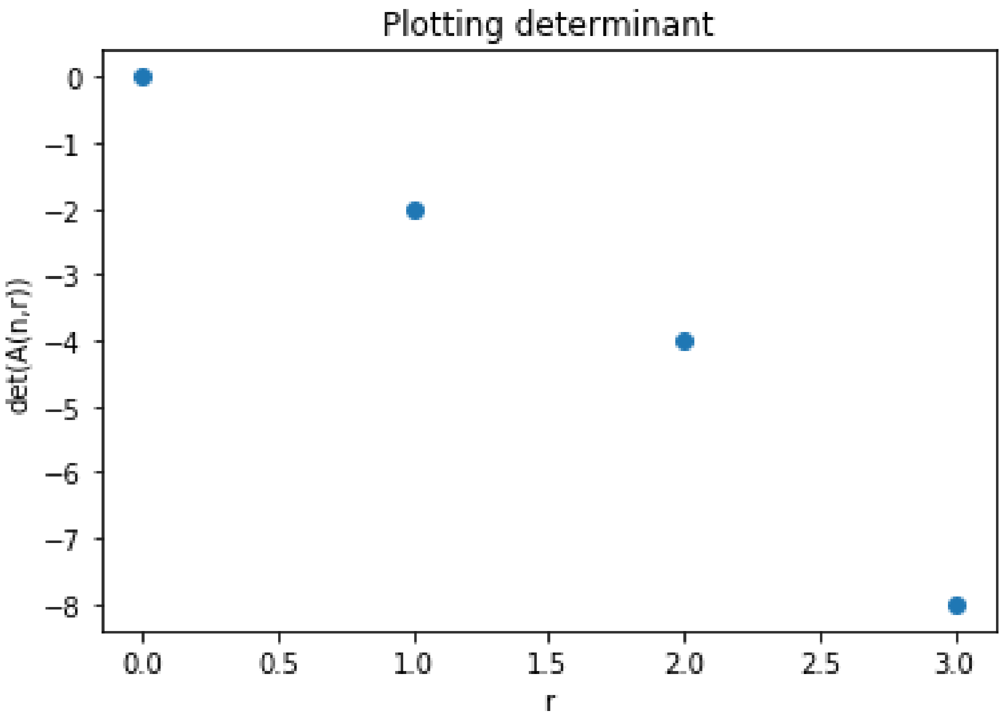
4. A magic square with row/column sum n is a square matrix with integer entries whose each row and column adds up to n. Write a program to check if a given matrix is a magic square.

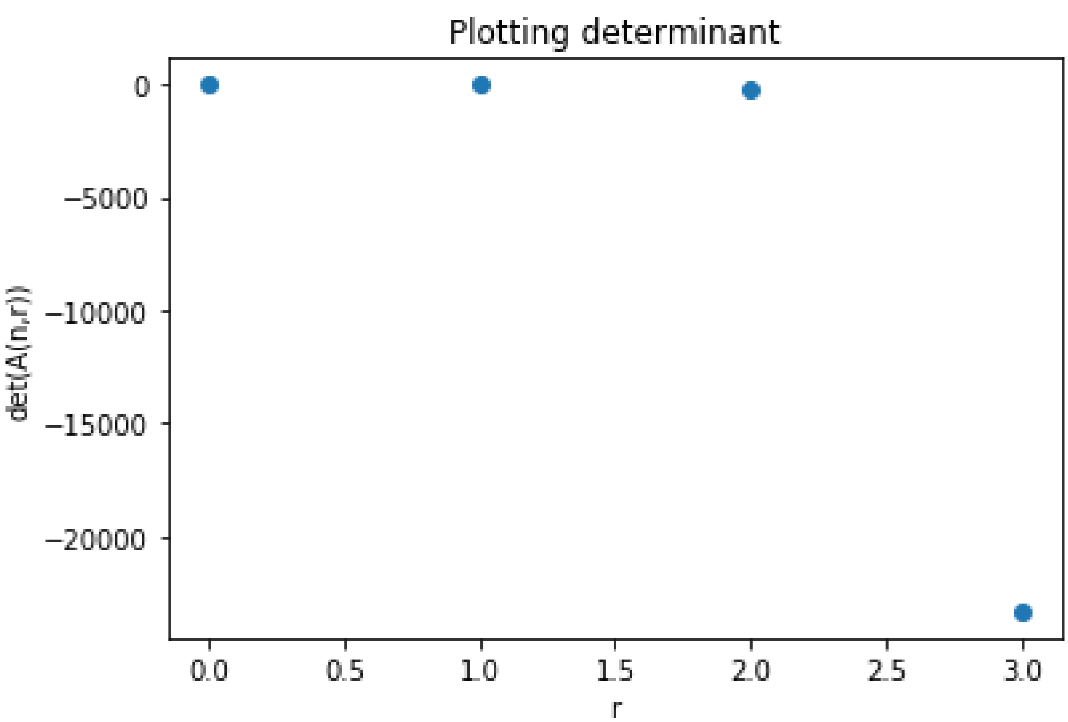
```
In [21]: import numpy as np

         def sumrow(i,A):
             B = A[i,:]
             r = len(B)
             a = 0
             for j in range(r):
                 a = a + A[i][j]
             return a

         def sumcol(j,A):
             B = A[:,j]
             r = len(B)
             a = 0
             for i in range(r):
                 a = a + A[i][j]
             return a

         def ismagic(A):
             rowcount = A.shape[0]
             colcount = A.shape[1]
             commsum = sumrow(0,A)
             for i in range(1, rowcount):
                 if sumrow(i,A) != commsum:
                     return False
             for j in range(0, colcount):
                 if sumcol(j,A) != commsum:
                     return False
             return True

         A = np.eye(3)
         B = np.ones((3,4))
         C = np.array([[16,3,2,13],[5,10,11,8],[9,6,7,12],[4,15,14,1]])
         print ismagic(A), ismagic(B), ismagic(C)
```

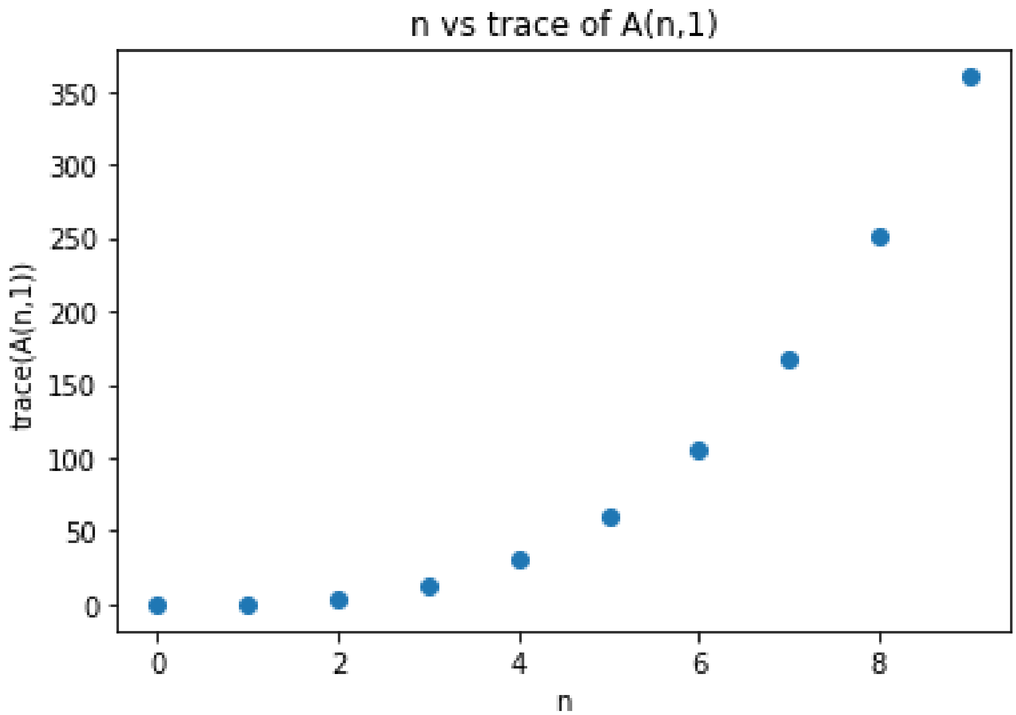
```
True False True
```

5. Let $A(n,r)$ denote an $n \times n$ matrix whose first row is $0, 1^r, 2^r, \cdots, (n-1)^r$, the second row is $n^r, (n+1)^r, (n+2)^r, (2n-1)^r$, and so on. Write a function that returns $A(n,r)$ for given n and r. For various values of n and r, plot n vs determinant of $A(n,r)$. Also plot n vs trace of $A(n,1)$.

```
In [64]: import matplotlib.pylab as pl
         import numpy as np

         def A(n,r):
             M = np.arange(n**2).reshape(n,n)
             return M**r

         def detplot(n):
             M = A(n,1)
             rvalues = range(0,4)
             X = [M**r for r in rvalues]
             Y = [np.linalg.det(P) for P in X]
             pl.xlabel('r')
             pl.ylabel('det(A(n,r))')
             pl.plot(rvalues,Y,'o')
             pl.title('Plotting determinant')
             pl.show()

         def traceplot():
             nvalues = range(0,10)
             X = [A(n,1) for n in nvalues]
             Y = [np.trace(P) for P in X]
             pl.xlabel('n')
             pl.ylabel('trace(A(n,1))')
             pl.plot(rvalues,Y,'o')
             pl.title('n vs trace of A(n,1)')
             pl.show()

         detplot(2)
         detplot(3)
         traceplot()
```


Plotting determinant
0
−1
−2
−3
−4
−5
−6
−7
−8
det(A(n,r))
0.0
0.5
1.0
1.5
2.0
2.5
3.0
r


Plotting determinant
0
−5000
−10000
−15000
−20000
det(A(n,r))
0.0
0.5
1.0
1.5
2.0
2.5
3.0
r



6. Write a program that solves a given linear system of equations for x, y and z,

$$\begin{aligned} a_{11}x + a_{12}y + a_{13}z &= b_1 \\ a_{21}x + a_{22}y + a_{23}z &= b_2 \\ a_{31}x + a_{32}y + a_{33}z &= b_3 \end{aligned}$$

where a_{ij} and b_k are real numbers.

```
In [80]: A = np.array([[1,2,3],[3,0,-2],[4,0,1]])
         b = np.array([6,5,3])
         Ainv = np.linalg.inv(A)
         print "Solution is [x, y, z] =", Ainv.dot(b)

Solution is [x, y, z] = [ 1.  4. -1.]
```