2challenge

October 2, 2017

1 Challenge 2

An important aspect of pragmatic vector space methods is the ability to handle vectors and matrices. A large collection of linear algebra functions is available in SciPy.linalg. These functions can be employed in conjunction with the tools available in NumPy. We note that the main object in NumPy is the homogeneous multidimensional array.

1.1 Matrix

We begin by creating a simple matrix. One possible approach to complete this task is to use `scipy.linalg.circulant(c)`.

```
In [1]: from scipy.linalg import circulant
        my_circ_matrix = circulant([1, 2,3])
        print(my_circ_matrix)
```

```
[[1 3 2]
 [2 1 3]
 [3 2 1]]
```

Alternatively, you can construct the familiar discrete Fourier transform matrix with `scipy.linalg.dft(n)`.

```
In [2]: from scipy.linalg import dft
        my_dft_matrix = dft(3)
        print(my_dft_matrix)
```

```
[[ 1.0+0.j          1.0+0.j          1.0+0.j        ]
 [ 1.0+0.j         -0.5-0.8660254j -0.5+0.8660254j]
 [ 1.0+0.j         -0.5+0.8660254j -0.5-0.8660254j]]
```

The inverse of a matrix can be computed using `scipy.linalg.inv(a)`.

```
In [3]: from scipy.linalg import inv
        my_idft_matrix = inv(my_dft_matrix)
        print(my_idft_matrix)
```

```
[[ 0.33333333 +2.77555756e-17j  0.33333333 +2.77555756e-17j
   0.33333333 -5.55111512e-17j]
 [ 0.33333333 +5.55111512e-17j -0.16666667 +2.88675135e-01j
  -0.16666667 -2.88675135e-01j]
 [ 0.33333333 -1.11022302e-16j -0.16666667 -2.88675135e-01j
  -0.16666667 +2.88675135e-01j]]
```

The operation `numpy.dot(a, b)` computes the dot product of two arrays. For 2-D arrays it is equivalent to matrix multiplication, and for 1-D arrays to inner product of vectors (without complex conjugation).

```
In [4]: import numpy as np
        matrix_prod1 = np.dot(my_dft_matrix, my_circ_matrix)
        matrix_prod2 = np.dot(matrix_prod1, my_idft_matrix)

        np.set_printoptions(suppress=True)
        print(matrix_prod2)
```

```
[[ 6.0-0.j         0.0+0.j        -0.0+0.j       ]
 [-0.0-0.j        -1.5+0.8660254j -0.0-0.j       ]
 [ 0.0-0.j         0.0-0.j        -1.5-0.8660254j]]
```

1.1.1 Questions

These steps and their solutions immediately bring up three questions. * Are circulant matrices always diagonalized by the discrete Fourier transform matrix and its inverse? * Are product of circulant matrices (of a same size) always circulant matrices? * Do all pairs of circulant matrices commute under matrix multiplication?

1.1.2 Solution

- Yes, Suppose the circulant matrix is

$$S = \begin{pmatrix} a & c & b \\ b & a & c \\ c & b & a \end{pmatrix} \tag{1}$$

and $\omega = e^{\frac{2\pi j}{3}}$, so the fourier transform matrix is

$$F = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \omega & \omega^2 \\ 1 & \omega^2 & \omega^4 \end{pmatrix} \tag{2}$$

So we can get

$$FS = \begin{pmatrix} a+b+c & a+b+c & a+b+c \\ a+b\omega+c\omega^2 & c+a\omega+b\omega^2 & b+c\omega+a\omega^2 \\ a+b\omega^2+c\omega^4 & c+a\omega^2+b\omega^4 & b+c\omega^2+a\omega^4 \end{pmatrix} \tag{3}$$

Suppose $A = a + b + c, B = a + b\omega + c\omega^2, C = a + b\omega^2 + c\omega^4$, we can get

$$FS = \begin{pmatrix} A & A\omega^0 & A\omega^0 \\ B & B\omega^1 & B\omega^2 \\ C & C\omega^2 & C\omega^4 \end{pmatrix} = \begin{pmatrix} A & B & C \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 1 \\ 1 & \omega^1 & \omega^2 \\ 1 & \omega^2 & \omega^4 \end{pmatrix} = \begin{pmatrix} A & B & C \end{pmatrix} \times F \quad (4)$$

So

$$FSF^{-1} = SFF^{-1} = SI = \begin{pmatrix} A & 0 & 0 \\ 0 & B & 0 \\ 0 & 0 & C \end{pmatrix} \quad (5)$$

Actually, this can be applied to any circulant matrices, because the time shift property of fourier transform.

- Suppose A is a $n \times n$ circulant matrix. It obvious to know that A have three properties as follows.
  First, $A_{ij} = A_{(i+1)(j+1)}, (i < n, j < n)$
  Second, $A_{nj} = A_{1(j+1)}, (j < n)$
  Third, $A_{in} = A_{(i+1)1}, (i < n)$
  And any $n \times n$ matrix have these three properties is a circulant matrix.
  Suppose A and B is two $n \times n$ circulant matrices, so

$$(AB)_{ij} = \sum_{q=1}^{n} A_{iq} B_{qj}$$

$$(AB)_{(i+1)(j+1)} = \sum_{q=1}^{n} A_{(i+1)q} B_{q(j+1)} = \sum_{q=2}^{n} A_{i(q-1)} B_{(q-1)j} + A_{in} B_{nj} = \sum_{q=1}^{n} A_{iq} B_{qj} = (AB)_{ij}$$

  Follow similar sequence, we can prove other two properties of circulant matrix in AB. So AB is a circulant matrix and product of circulant matrices (of a same size) are always circulant matrices
- Suppose A and B is two $n \times n$ circulant matrices. We already get

$$(AB)_{ij} = \sum_{q=1}^{n} A_{iq} B_{qj}$$

$$(BA)_{ij} = \sum_{q=1}^{n} B_{iq} A_{qj}$$

  Because AB and BA are circulant matrices, so, if $(BA)_{1j} = (AB)_{1j}$, then $AB = BA$

$$
\begin{aligned}
(BA)_{1j} &= \sum_{q=1}^{n} B_{1q}A_{qj} \\
&= \sum_{q=2}^{j} A_{1(j-q+1)}B_{(n+2-q)1} + A_{1j}B_{11} + \sum_{q=j+1}^{n} A_{1(n+j-q+1)}B_{(n+2-q)1} \\
&= \sum_{q=1}^{j-1} A_{1q}B_{(n+1-j+q)1} + A_{1j}B_{11} + \sum_{q=j+1}^{n} A_{1q}B_{(1-j+q)1} \\
&= \sum_{q=1}^{j-1} A_{1q}B_{qj} + A_{1j}B_{jj} + \sum_{q=j+1}^{n} A_{1q}B_{qj} \\
&= \sum_{q=1}^{n} A_{1q}B_{qj} \\
&= (AB)_{1j}
\end{aligned} \tag{6}
$$

So we can conclude $AB = BA$ and all pairs of circulant matrices commute under matrix multiplication

1.2 Determinant

The determinant of a square matrix is a value derived arithmetically from the coefficients of the matrix, and it summarizes a multivariable phenomenon with a signle number. It can be computed with `scipy.linalg.det(a)`.

```
In [5]: from scipy.linalg import det
        from numpy import random
        my_circ_matrix = random.randint(1,10, size = (3,3))
        det(my_circ_matrix)
```

```
Out[5]: -93.99999999999999
```

The code below demonstrates how to create a function in Python, how to vectorize a function so that it can be applied to the elements of a matrix, and how to use `random`.

```
In [6]: import math

        def my_log(x):
            return math.log(x)

        my_vec_log = np.vectorize(my_log)

        A_step1 = my_vec_log(my_circ_matrix) # Numpy already offers a vectorized na

        print(A_step1)
        # A_step1 = np.log(matrix_prod2)

        max_index = 1000000
```

```
my_identity = np.identity(len(A_step1))
current_value = 0.0
for my_index in range(0, max_index):
    permutation_matrix = random.permutation(my_identity)
    sign_permuation = det(permutation_matrix)
    current_value += sign_permuation*(np.exp(np.trace(np.dot(A_step1, permu
a_step2 = math.factorial(len(A_step1)) * current_value / max_index
print(a_step2)
```

```
[[ 0.          1.09861229  0.        ]
 [ 2.19722458  0.          1.60943791]
 [ 1.94591015  1.38629436  2.07944154]]
-93.717738
```

```
In [7]: tryIdentity = np.identity(3)
        tryRandom = random.permutation(tryIdentity)
        print(tryIdentity)
        print(tryRandom)
        print(det(tryRandom))
```

```
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
[[ 0.  1.  0.]
 [ 0.  0.  1.]
 [ 1.  0.  0.]]
1.0
```

```
In [8]: dotTemp = np.dot(A_step1, tryRandom)
        print(dotTemp)
```

```
[[ 0.          0.          1.09861229]
 [ 1.60943791  2.19722458  0.        ]
 [ 2.07944154  1.94591015  1.38629436]]
```

Questions It appears that the output of the loop above is close to the determinant of the circulant matrix `my_circ_matrix`. * Go through the code and provide a compelling explain explanation of why these numbers are close. * Is this a property of circulant matrices, or would this finding extend to arbitrary matrices over the real numbers?

Solution

- Because the determinant of matrix is caculated by

$$\det(A) = \sum_{\sigma \in S_n} \mathrm{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)}$$

And the random process actually produce term of this equation and the produce probility of each term is equal. So it would end up with a value which is very close with $\det(A)$.

1.2.1 Tasks

- Build code to explore the fact that the determinant function is multiplicative: $\det(AB) = \det(A)\det(B)$.

```
In [9]: maxCheckTime = 10000
        trueTime = 0
        for myIndex in range(maxCheckTime):
            sizeMatrix = random.randint(1,3)  #size is ramdom within 3
            matrixA = random.randint(100, size = (sizeMatrix, sizeMatrix))
            matrixB = random.randint(100, size = (sizeMatrix, sizeMatrix)) #produce
        #     print(matrixA)
        #     print(matrixB)
        #     print(det(np.dot(matrixA,matrixB)))
        #     print(det(matrixA)*det(matrixB))
            if (np.absolute(det(np.dot(matrixA,matrixB))-det(matrixA)*det(matrixB))
                trueTime += 1
        print("accuracy rate is %f" % (trueTime/maxCheckTime))
```

```
accuracy rate is 1.000000
```

So according to the simulate result above, we can conclude $\det(AB) = \det(A)\det(B)$

```
In [ ]:
```