Tensor Algebra
University Federal do Ceará
PPGETI - 2020.2
Saulo Mendes de Melo

Homework 6

Unfolding, folding and n-mode product

1 Problem 1

For a third-order tensor $\mathcal{X} \in \mathbb{C}^{I \times J \times K}$, using the concept of n-mode fibers, implement the function *unfold* according to the following prototype

$$[\mathcal{X}]_{(n)} = unfold(\mathcal{X}, n)$$

Solution

Listing 1 contains the declaration of a tensor $\mathcal{A} \in \mathbb{C}^{3\times4\times2}$ which is used for testing in Problems 1 and 2. The shape definition looks wrong, but it is done like that due to the way *numpy* defines 3-dimensional tensors. They are defined as arrays of matrices, so the tube fibers dimension comes first.

```python
np.random.seed(0)
shape = (2, 3, 4)

A = randn(shape[0], shape[1], shape[2])

np.random.seed(1)
A = A + randn(shape[0], shape[1], shape[2])*1j
```

Listing 1: Creating $\mathcal{A} \in \mathbb{C}^{3\times4\times2}$

The unfolding function implementation is listed in Listing 2. As means of testing, this work uses the *Tensorly* library functions for *python* to compare the results of the custom made functions. In *Python* array indexing starts with zero. As to maintain a cohesiveness, both the custom made *unfold* and *Tensorly*´s unfold consider the n-modes starting at zero.

```python
def unfold(A, n):
    C = np.concatenate([np.arange(0, n), np.arange(n+1, A.ndim)])

    M = A.shape[n]
    N = np.prod([A.shape[i] for i in C])

    X = np.reshape(np.transpose(A, np.concatenate([[n], C])), (M, N))

    return X
```

Listing 2: custom made unfold function

An important consideration to be made is the impact of the dimensional ordering previously described. Due to the fact the that tube fibers dimension comes first, the tube fibers mode unfolding, usually considered to be the 3-mode unfolding in the case of 3-dimensional tensors, is equals to the first mode unfolding (0-mode) in the computational methods discussed. So throughout this work, the 0-mode unfolding of a tensor should always be considered as the tube fibers dimension unfolding.

The code in Listing 2 does the unfolding by moving the n-mode axis to the first dimension and reshaping the remaining axes as a product of them, resulting in a matrix with dimensions $I_n \times I_1 I_2 \dots I_{n-1} I_{n+1} \dots I_N$.

Tensor $\mathcal{A}$ is made of two frontal slices:

$$\mathbf{A}_1 = \begin{bmatrix} 1.8+1.6j & 0.4-0.6j & 1-0.5j & 2.2-1.1j \\ 1.9+0.9j & -1-2.3j & 1+1.7j & -0.2-0.8j \\ -0.1+0.3j & 0.4-0.2j & 0.1+1.5j & 1.5-2.1j \end{bmatrix}$$

$$\mathbf{A}_2 = \begin{bmatrix} 0.8-0.3j & 0.1-0.4j & 0.4+1.1j & 0.3-1.1j \\ 1.5-0.2j & -0.2-0.9j & 0.3+0j & -0.9+0.6j \\ -2.6-1.1j & 0.7+1.1j & 0.9+0.9j & -0.7+0.5j \end{bmatrix}$$

And it's 2-mode unfolding is given by:

$$[\mathcal{A}]_{(2)} = \begin{bmatrix} 1.8+1.6j & 1.9+0.9j & -0.1+0.3j & 0.8-0.3j & 1.5-0.2j & -2.6-1.1j \\ 0.4-0.6j & -1-2.3j & 0.4-0.2j & 0.1-0.4j & -0.2-0.9j & 0.7+1.1j \\ 1-0.5j & 1+1.7j & 0.1+1.5j & 0.4+1.1j & 0.3+0j & 0.9+0.9j \\ 2.2-1.1j & -0.2-0.8j & 1.5-2.1j & 0.3-1.1j & -0.9+0.6j & -0.7+0.5j \end{bmatrix}$$

With the simple loop in Listing 3, we can assure that the implementation is correct that the implementation is correct for every n-mode by comparing with *Tensorly*'s functions results.

```python
for n in range(A.ndim):
    assert tl.unfold(tl.tensor(A), n).all() == unfold(A, n).all()
```

Listing 3: unfold assertion

2 Problem 2

Implement the function *fold* the converts the unfolding $[\mathcal{X}]_{(n)}$ obtained with $unfold(\mathcal{X}, n)$ back to the tensor $\mathcal{X} \in \mathbb{C}^{I \times J \times K}$ (i.e., a 3-d array in Matlab/Octave), according to the following prototype:

$$\mathcal{X} = fold([\mathcal{X}]_{(n)}, [I \ J \ K], n)$$

Solution

The custom implementation for the *fold* method can be seen in Listing 4. First it reshapes unfolded $[\mathcal{A}]_{(n)}$ with dimensions $I_n \times I_1 I_2 \dots I_{n-1} I_{n+1} \dots I_N$ into a matrix with shape $I_n \times I_1 \times I_2 \times \dots \times I_{n-1} \times I_{n+1} \times \dots \times I_N$. Then, it moves the axis I_n in the first position back to the n-th position, giving back a tensor with shape $I_1 \times I_2 \times \dots I_n \times \dots \times I_N$.

```python
def fold(unfolded_A, n, shape):
    C = np.concatenate([np.arange(0, n), np.arange(n+1, len(shape))])

    new_shape = []
    new_shape.append(shape[n])
    for item in [shape[i] for i in C]:
        new_shape.append(item)


    A = unfolded_A.reshape(tuple(new_shape))

    X = np.moveaxis(A, 0, n)

    return X
```

Listing 4: custom made fold function

Plugging into the function the unfolded $[\mathcal{A}]_{(2)}$ we get the tensor $\mathcal{A}$ with the two frontal slices $\mathbf{A}_1$ and $\mathbf{A}_2$. A simple verification of correctness can be made with the loop in Listing 5

```python
for n in range(A.ndim):
    unfolded_A = unfold(A, n)
    assert tl.fold(unfolded_A, n, shape).all() == fold(unfolded_A, n, shape).all()
```

Listing 5: custom made fold assertion

3 Problem 3

For given matrices $\mathbf{A} \in \mathbb{C}^{P\times I}$, $\mathbf{B} \in \mathbb{C}^{Q\times J}$, and $\mathbf{C} \in \mathbb{C}^{R\times K}$ and tensor $\mathcal{X} \in \mathbb{C}^{I\times J\times K}$, calculate the tensor $\mathcal{Y} \in \mathbb{C}^{P\times Q\times R}$ via the following multilinear transformation:

$$\mathcal{Y} = \mathcal{X} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}$$

Solution

Making use of the previously implemented *unfold* and *fold* methods the n-mode product was implemented as seen in Listing 6. The resulting tensor of the n-mode product has the same dimensions as the input $\mathcal{A}$, but with n-th dimension exchanged with the first dimension of the multiplying matrix U. This is encapsulated in the variable y_shape.

```python
def n_mode_product(A, U, n):
    y_shape = list(A.shape)
    y_shape[n] = U.shape[0]

    unfolded_Y = np.matmul(U, unfold(A, n))

    Y = fold(unfolded_Y, n, tuple(y_shape))

    return Y
```

Listing 6: n-mode product

The following step makes use of the property $[\mathcal{Y}]_{(n)} = \mathbf{U}[\mathcal{X}]_{(n)}$ to calculate $[\mathcal{Y}]_{(n)}]$, which is then folded into the resulting tensor with dimensions of y_shape previously calculated. The Listing 7 defines matrices and tensors required for the problem.

```python
P, Q, R = 2, 3, 4
I, J, K = 5, 6, 7

np.random.seed(0)
A = randn(P, I)
B = randn(Q, J)
C = randn(R, K)
X = randn(I, J, K)

np.random.seed(1)
A = A + randn(P, I)*1j
B = B + randn(Q, J)*1j
C = C + randn(R, K)*1j
X = X + randn(I, J, K)*1j
```

Listing 7: n-mode product

$\mathbf{A} \in \mathbb{C}^{2\times 5}$, $\mathbf{B} \in \mathbb{C}^{3\times 6}$, and $\mathbf{C} \in \mathbb{C}^{4\times 7}$ and $\mathcal{X} \in \mathbb{C}^{5\times 6\times 7}$ are used by subsequently applying the n-mode product function, and we have the result as a $\mathcal{Y} \in \mathbb{C}^{2\times 3\times 4}$.

```python
Y = n_mode_product(n_mode_product(n_mode_product(X, A, 0), B, 1), C, 2)
Y_tl = mode_dot(mode_dot(mode_dot(X, A, 0), B, 1), C, 2)

assert Y.all() == Y_tl.all()
```

Listing 8: n-mode product verification