DCCM计算过程优化?

我们想要获知成对残基之间的运动模式关联，可以计算成对残基之间的协方差。

公式如下：

$c(i,j) =< \Delta R_i \cdot \Delta R_j >$

其中: $\Delta R_i = R_i - < R_i >$

i和j表示的是蛋白质氨基酸残基的序号，也即这里的$c(i,j)$是残基i和残基j的协方差。尖括号表示的是系综平均，也即计算的是一系列模型的平均，包括模拟轨迹里面的多帧构象、NMR得到的多个结构等等。ΔR_i表示的是位置偏移量，且是基于系综平均位置的偏移量。

所谓的动态互相关矩阵，也就是把残基对之间的协方差转换为互相关指数，并按残基编号组织起来，成为一个矩阵而已。

$C(i,j) = \frac{c(i,j)}{[c(i,i) \cdot c(j,j)]^{1/2}}$

例如我们的蛋白质有N个残基，要计算残基alpha-C原子的DCCM。我们首先要做的是对这段儿轨迹进行周期性校正，去除平动转动。之后需要将每一帧的原子坐标**对齐**到参考结构，得到对齐之后的每一帧的原子坐标。

然后对于每一个C原子，计算其在这段轨迹中的平均位置。有了这个平均坐标，就可以计算每一帧中这个C原子相对于其平均位置的偏移向量(x, y, z)了。

之后对于每一帧轨迹，计算两两C原子偏移量之间的点积，然后再将多帧的协方差对帧数取平均。如此我们就得到两两残基之间在这段模拟轨迹中的协方差矩阵了。最后，对协方差矩阵上的元素计算相关性系数就可以得到动态互相关矩阵

方法一：按照公式编写的代码，因为有三层循环，执行非常慢，10001帧130原子需要十几分钟

```
#### method origin !!!
## calculate the position offset to averaged positions
time_number = allxyz.shape[0]
atom_number = allxyz.shape[1]
atomcoord_mean=[]
for i in range(atom_number):
    x, y, z = [], [], []
    for j in range(time_number):
        x.append(allxyz[j][i][0])
        y.append(allxyz[j][i][1])
        z.append(allxyz[j][i][2])
    atomcoord_mean.append([np.mean(x),np.mean(y),np.mean(z)])
delta0 = np.array(atomcoord_mean)
offset = allxyz - delta0

## calculate the covariance and correlation matrix
## 这里的offset就是多帧的C原子的位置偏移量，shape是(帧数、原子数、坐标)
print(offset.shape) # (10001, 130, 3) numpy.array

## calculate the Covariance by np.dot of atom i and j
## the covariance calculation is fucking slow
covariance = np.zeros((atom_number, atom_number))
for i in range(atom_number):
    for j in range(atom_number):
        ## 对每一帧，计算任意两个原子的位置偏移量之间的点积，然后对时间求平均
        covariance[i, j] = np.mean([np.dot(offset[t][i], offset[t][j]) for t in range(time_number)])

## 从协方差矩阵计算互相关矩阵
corr=np.zeros((atom_number,atom_number))
for i in range(0,atom_number):
    for j in range(0,atom_number):
        corr[i,j] = covariance[i,j]/np.sqrt(covariance[i,i]*covariance[j,j])
```

方法二：重新编写的代码，非常快，10001帧130原子需要几秒钟

```python
## new method to calculate the covariance and correlation matrix
## Two methods produce numbers with deviation less than 0.00001
offset = allxyz
print(offset.shape) # (10001, 130, 3)
# np.cov for x, y, and z, then sum
## 这里对数据重新处理了一下，把坐标的三维拆开，然后重新处理成(原子，时间帧)的形式
## 例如offset_X是包含了130个列表的变量，每一个列表里面保存了一个原子的x坐标随时间变化的10001个数据
offset_X, offset_Y, offset_Z = offset[:, :, 0].T, offset[:, :, 1].T, offset[:, :, 2].T
print(offset_X.shape)  # (130, 10001)

## 对每一个维度的数据做协方差矩阵，得到的是一个 (130, 130) 的矩阵
## 也即每一个维度上，原子之间的协方差矩阵
# np.cov for x, y, and z, then sum
covariance_X = np.cov(offset_X, ddof=0) # /n not /(n-1)
covariance_Y = np.cov(offset_Y, ddof=0)
covariance_Z = np.cov(offset_Z, ddof=0)
## 将三个维度相加，得到最终的协方差矩阵
covariance = covariance_X + covariance_Y + covariance_Z
print(covariance_Z.shape) #  (130,  130)
print(covariance.shape) #  (130,  130)

## 将协方差矩阵转换为互相关矩阵
corr=np.zeros((atom_number,atom_number))
for i in range(0,atom_number):
    for j in range(0,atom_number):
        corr[i,j] = covariance[i,j]/np.sqrt(covariance[i,i]*covariance[j,j])
```

比较方法一和方法二产生的DCCM的数值，矩阵上每一个对应的元素的**差值小于0.0001。**

同时将方法二与correlationplus的结果比较，由于correlationplus的结果只保存到小数点后六位，所以差值在0.00001以内。

总结：方法二在数学上和方法一是等价的

点积的方法：

$$C(i,j) =< \Delta R_i \cdot \Delta R_j >= \frac{1}{T}\sum_{t=0}^{T}[(X_{it}-\bar{X}_i)(X_{jt}-\bar{X}_j)+(Y_{it}-\bar{Y}_i)(Y_{jt}-\bar{Y}_j)+(Z_{it}-\bar{Z}_i)(Z_{jt}-\bar{Z}_j)]$$

对于其中某一个维度：

$$C(i,j)_X = \frac{1}{T}\sum_{t=0}^{T}[(X_{it}-\bar{X}_i)(X_{jt}-\bar{X}_j)]$$

此即为协方差的公式，也即 `np.cov(, ddof=0)` ，也即为优化之后的方法。

如此，只需要对点云fit之后的坐标进行拆分，分别对X、Y、Z三个维度上的原子坐标随时间的变化求协方差，最后加起来，就得到了最终的协方差矩阵。