将多个矩阵方程的求解并行化

2021 年 1 月 30 日

1 问题叙述

假设我们有分块矩阵

$$\begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{bmatrix}, \quad \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \boldsymbol{b}_3 \\ \boldsymbol{b}_4 \end{bmatrix} \tag{1.1}$$

我们希望找到 $\boldsymbol{x}_1$,$\boldsymbol{x}_2$,$\boldsymbol{x}_3$,$\boldsymbol{x}_4$ 满足

$$\begin{cases} A_1\boldsymbol{x}_1 = \boldsymbol{b}_1 \\ A_2\boldsymbol{x}_2 = \boldsymbol{b}_2 \\ A_3\boldsymbol{x}_3 = \boldsymbol{b}_3 \\ A_4\boldsymbol{x}_4 = \boldsymbol{b}_4 \end{cases} \tag{1.2}$$

如何将这样的计算并行化呢?

2 解法

线性代数的知识告诉我们:

$$\begin{bmatrix} A_1 & & & \\ & A_2 & & \\ & & A_3 & \\ & & & A_4 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_4 \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \boldsymbol{b}_3 \\ \boldsymbol{b}_4 \end{bmatrix} \tag{2.1}$$

所以只需凑出这个分块对角矩阵 $\mathrm{diag}\{A_1, A_2, A_3, A_4\}$ 然后再求它的乘法逆就可以了，可是在实践中，有很多个这样的矩阵 $A_1, A_2, \cdots, A_n$，这里 n 是一个不小的数，如果说每一个这样的 A_i 是 $d_0 \times d_1$ 的形状的话，那么这个分块对角矩阵就有 $n \times n \times d_0 \times d_1$ 个元素，并且这个矩阵是高度稀疏的，所以如果直接存储的话，非常浪费空间.

3 实现

借助于 SciPy 库为稀疏矩阵实现的一些方法，我们可以解决原问题. 作为演示，我们就随机生成 4 个 3×3 的方阵:

```
import numpy as np

```

```
mat_A1 = np.random.rand(3, 3)
mat_A2 = np.random.rand(3, 3)
mat_A3 = np.random.rand(3, 3)
mat_A4 = np.random.rand(3, 3)
```

然后我们再来检验它们是否都可逆:

```
print(np.linalg.det(mat_A1))
print(np.linalg.det(mat_A2))
print(np.linalg.det(mat_A3))
print(np.linalg.det(mat_A4))
```

如果行列式的绝对值值太小，比如说绝对值小于 1×10^{-6} 的话，可以考虑再重新生成随机矩阵，直到可逆为止．确保它们都可逆后，拼成一个大数组，并且一维化:

```
data = np.concatenate((mat_A1, mat_A2, mat_A3, mat_A4,), axis=0)
data = data.reshape(data.shape[0]*data.shape[1])
```

下面构造 `indptr`, `indices` 这两个量，与 `data` 一起，用来初始化一个 `csr_matrix` 实例，这就是原来这 4 个矩阵的分块对角矩阵的稀疏矩阵表示:

```
from scipy.sparse import csr_matrix

indptr = np.arange(0, 4*3*3+1, 3)
indices = np.arange(0, 4*3, 1).reshape(4, 3)
indices = np.concatenate((indices, indices, indices,), axis=1)
indices = indices.reshape(indices.shape[0]*indices.shape[1])

sp_mat = csr_matrix((data, indices, indptr))
```

这个 `sp_mat` 就是 $\mathrm{diag}\{A_1, A_2, A_3, A_4\}$ 的稀疏表示，从而大大地节省了存储空间，下面我们随机生成四组系数代表 $\boldsymbol{b}_1$,$\boldsymbol{b}_2$,$\boldsymbol{b}_3$,$\boldsymbol{b}_4$，然后开始求解:

```
from scipy.sparse.linalg import splu

b1 = np.random.rand(3)
b2 = np.random.rand(3)
b3 = np.random.rand(3)
b4 = np.random.rand(3)
b = np.concatenate((b1,b2,b3,b4))

answer = splu(sp_mat, b)
```

得到的这个 `answer` 是一个 10 元组，第一个元素是解，其余的如果读者感兴趣可去翻看文档，为了方便验证，我们把每个解都提取出来，然后验证看解是否求对了:

```
sol1 = answer[0][0:3]
sol2 = answer[0][3:6]
sol3 = answer[0][6:9]
sol4 = answer[0][9:12]

print(np.max(np.abs((mat_A1 @ sol1) - b1)))
print(np.max(np.abs((mat_A2 @ sol2) - b2)))
print(np.max(np.abs((mat_A3 @ sol3) - b3)))
print(np.max(np.abs((mat_A4 @ sol4) - b4)))
```

在我的实验中，最大的误差绝对值不超过1×10^{-6}，还算是比较理想的．借助于 SuperLU[1]，计算效率会非常高．

4 总结和补充

在我们给的演示案例中，看不出这样做的好处，但实际上，通过将多个矩阵的逆归结为一个矩阵的逆，可以使得求多个矩阵的逆这个过程得以并行化，所以我们才说，是「同时」求出若干组矩阵方程的解，而不是一个一个地求．如果是一个一个地求，计算设备的很多时间会花在等待上，相比起来，并行化使得硬件资源被充分利用，所以理论上应该会更加快一些．

参考文献

[1] Xiaoye S. Li. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Transactions on Mathematical Software*, 31(3):302–325, September 2005.