代码2-6 利用NumPy工具包提升矩阵乘积计算效率

In [11]:

```
import numpy as np # 导入numpy工具包
import random # 导入random模块
from time import perf_counter # 从time模块导入perf_counter
class MyMatrix: # 定义MyMatrix类
    def __init__(self): # 构造方法
        self.mat = [] # mat属性用于保存矩阵中的元素值
    def inputElements(self): # 用于输入矩阵元素的方法
        print('请逐行输入矩阵元素（同一行元素之间以空格隔开，最后一行输入0表示矩阵结束）：')
        while True: # 永真循环，只能通过循环中的break语句退出循环
            linedata = input() # 输入以空格隔开的一行元素
            if linedata=='0': # 判断是否满足矩阵输入结束条件
                break # 满足结束条件则退出循环
            val_list = linedata.split() # 将字符串按空白符分割，返回分割各子串组成的列表
            self.mat.append([eval(x) for x in val_list]) # 通过列表生成表达式将列表中的字符串转为数值并存储到矩阵中
    def outputElements(self): # 用于输出矩阵元素的方法
        for rowindex in range(len(self.mat)): # 依次获取矩阵每一行的索引
            for colindex in range(len(self.mat[0])): # 依次获取矩阵每一列的索引
                print(self.mat[rowindex][colindex], end = ' ') # 输出一个元素后再输出一个空格
            print() # 输出一个换行
    def __mul__(self, mat2): # 实现矩阵乘法运算的内置方法
        mat_rlt = MyMatrix() # 保存矩阵乘积运算结果
        for row1 in range(len(self.mat)): # 从上至下依次得到第一个矩阵的每个行索引（从0开始）
            row_rlt = [] # 保存矩阵乘积的一行结果
            for col2 in range(len(mat2.mat[0])): # 从左至右依次得到第二个矩阵的每个列索引（从0开始）
                rlt = 0 # 保存矩阵乘积的一个结果元素
                for col1 in range(len(self.mat[0])): # col1同时表示第一个矩阵的列索引和第二个矩阵的行索引（二者具有一一对应关系）
                    rlt += self.mat[row1][col1]*mat2.mat[col1][col2] # 两个矩阵对应元素做乘法运算并加到rlt中
                row_rlt.append(rlt) # 将计算得到的一个结果元素添加到一行结果的尾部
            mat_rlt.mat.append(row_rlt) # 将一行结果添加到矩阵乘积运算结果的尾部
        return mat_rlt
    def randomElements(self, rows, cols): # 随机生成rows*cols个矩阵元素
        self.mat = []
        for r in range(rows): # 生成rows行元素
            linedata = [random.random()*10000 for _ in range(cols)] # 生成由cols个0~10000之间的随机数组成的列表
            self.mat.append(linedata) # 将生成的一行数据添加到矩阵中

if __name__=='__main__':
    mat1, mat2 = MyMatrix(), MyMatrix() # 创建两个矩阵类对象
    n_vals = [10, 50, 100, 150, 200] # 生成的都是方阵，n_vals中的元素表示矩阵的行/列数
    repeats = 10 # 实验重复次数（对于一种问题规模，重复多次实验取计算时间平均值以使结果更加稳定）
    for n in n_vals: # 依次取每一种问题规模
        array_total_time = 0 # 记录多次实验中ndarray对象实现矩阵乘积计算消耗的时间
        list_total_time = 0 # 记录多次实验中列表对象实现矩阵乘积计算消耗的总时间
        for i in range(repeats): # 重复repeats次实验
            mat1.randomElements(n, n) # 生成第一个矩阵的元素
            mat2.randomElements(n, n) # 生成第二个矩阵的元素
            array1 = np.array(mat1.mat) # 将第一个矩阵转为np.ndarray对象
```

```
            array2 = np.array(mat2.mat) # 将第二个矩阵转为np.ndarray对象
            start = perf_counter() # 矩阵乘积前记录一个时间点
            array_rlt = array1@array2 # np.ndarray对象实现矩阵乘积计算
            end = perf_counter() # 矩阵乘积后记录一个时间点
            array_total_time += end-start # 将本次矩阵乘积计算消耗时间加到array_total_time上
            start = perf_counter() # 矩阵乘积前记录一个时间点
            mat_rlt = mat1*mat2 # 自动调用__mul__内置方法
            end = perf_counter() # 矩阵乘积后记录一个时间点
            list_total_time += end-start # 将本次矩阵乘积计算消耗时间加到list_total_time上
        print('两个%d*%d矩阵（ndarray对象）乘积运算消耗时间平均值为：%.8f秒'%(n, n, array_total_time/repeats)) # 输出ndarray对象实现矩阵乘积计
        print('两个%d*%d矩阵（列表对象）乘积运算消耗时间平均值为：%.8f秒'%(n, n, list_total_time/repeats)) # 输出列表对象实现矩阵乘积计算消耗时
```

```
两个10*10矩阵（ndarray对象）乘积运算消耗时间平均值为：0.00001413秒
两个10*10矩阵（列表对象）乘积运算消耗时间平均值为：0.00056803秒
两个50*50矩阵（ndarray对象）乘积运算消耗时间平均值为：0.00014232秒
两个50*50矩阵（列表对象）乘积运算消耗时间平均值为：0.07277723秒
两个100*100矩阵（ndarray对象）乘积运算消耗时间平均值为：0.00020369秒
两个100*100矩阵（列表对象）乘积运算消耗时间平均值为：0.53704491秒
两个150*150矩阵（ndarray对象）乘积运算消耗时间平均值为：0.00050996秒
两个150*150矩阵（列表对象）乘积运算消耗时间平均值为：1.80630216秒
两个200*200矩阵（ndarray对象）乘积运算消耗时间平均值为：0.00092106秒
两个200*200矩阵（列表对象）乘积运算消耗时间平均值为：4.13539937秒
```