在写计算模拟程序的时候可能需要的框架

李睿

August 2, 2018

1 导言

计算模拟源远流长，上至古希腊时期模拟天体运行（以下省略一万句）

2 如何构思程序

从我所写的所有程序给我的经验来讲，一个完整的程序需要经历：

1. 读取文件
2. 进行运算
3. 输出结果

在《C 程序设计》一课中最简单的例子是

1. #include <stdio.h>
2. 建立 main 函数以及可能的子函数
3. printf

但应该要说明的是，计算模拟程序往往非常庞大，而且需要非常高的灵活性来进行对不同情况的计算，因此往往需要分成不同的文件来干这些工作。

2.1 输入文件

你可能首先要处理输入的部分。你要明确在整个程序中可能会需要用到的所有的参数，并使用合理的文件结构及读取机制来实现一个正常的输入文件读取过程。

1. 首先，你要合理地设想你可能会使用的输入文件的格式。最开始未必追求过度的整齐划一的界面，相同类型的数据互相之间只用空格隔开，不同类型之间用换行隔开。
2. 你必须要对你所使用的语言如何读取文件要有所了解，如何建立一个足够好的判断机制来将这些数据分门别类进行存储。判断机制在一开始可能会显得不太有必要，但要明白你的程序是循序渐进的，有可能在编写的过程你发现可能需要通过输入文件手动调某些参数，也有可能事实上这个体系需要更多数据来维持某些运算。而如果像流水账一般把输入文件转换为语言里面的数据，那么会给后续添加带来极大困难，而好的判断机制能够让你更好的将更多的元素添加进来。

3. 必须用一个好的方式将输入文件里面的数据存储起来，这对数组的理解至关重要。为了能够对数组有更好的掌控能力，往往会涉及多维数组，这样才能方便地传递数据至不同的模块。这意味着你在存储的时候必需明确你在哪一级数据存储这些东西，并应一并写出子函数来调用里面的数据。

   为更好地进行演示，我们利用电化学模拟来说明一个复杂的数据结构可能会是什么样。

   我所使用的程序的数据结构如下：

   (a) 不同时刻下的体系——第一维

   (b) 该体系下所具有的空间自由度——+3 维

   (c) 每个坐标下所具有的属性，包括浓度、扩散系数——+1 维

   也就是说，如果从一个最直接的方法来考虑，我需要五维数组来完成这个事情。Mathematica 能够通过各种函数完成对每一级数据进行操作，但这个意思并不太大。我们尝试用 C 来描述这个问题，并应不断训练自己用指针来思考这些问题。

   我们使用反推的方式，一层一层往上赋予含义。比如

```
double *****p;

*****p, *****(p+1) //代表这个是浓度还是扩散系数
****(*p+0),****(*p+1),.... //代表z方向
***(**p+1),***(**p+2),.... //代表y方向
.... //代表x方向
*(****p),*(****p+1),..... //代表不同时刻
```

   但你会发现，这个实在是让人头疼。

   数学告诉我们，不同维数组都是可数集，是等势的，所以原则上只需要一维数组就可以描述多维数组。但**多维数组/指针的优势在于可以直接通过 +1 访问只改变一个维度的数据**。这使得一个可变的数据存储模式成为可能，并且能够减少不必要的访问方式。

   但毕竟这会让你的头很疼。因此就需要进行折衷。某种程度上 struct 可以在很大程度上解决该问题，比如将所具有的属性打包为结构，而时刻和空间自由度保持指针变化，做成四维结构指针。

   如果知道自己肯定要进行各种程度上的遍历，就可以再缩小所使用的指针。例如对空间的遍历，我们可以建立二维结构指针，在结构里面存储三维坐标和该坐标下对应的属性，并实现空间自由度下的链表来实现。也就是说，通过一维来实现不同时刻下的访问，而另一维存储这个时刻下第一个空间坐标的数据，这个数据里面包含了下一个空间坐标的数据的指针，从而实现全空间访问。具体想法参照链表模块。

   因此并不是说任何时候都需要按照所有的维度构建数组。

   但话又说回来，

   你存储所有数据也是有好处的。

4. 保证数据的封装性良好。程序编写过程中你可能需要添加不同的功能，而这需要在编写每一个模块的过程中时刻让自己做好相关的封装工作。在这方面 struct 会更有优势，因为只需要在 struct 里添加一行即可实现新的一个维度的数据储存。

2.2 进行运算

进行运算是程序的关键，无论从难度上还是从运行时间上这个部分仍旧具有着核心地位。然而事实上一个好的运算逻辑/最优算法可遇而不可求，在刚开始进行程序构思时不应花太多时间在如何获得最快的运算过程，而是将整个需要解决的问题拆分成为几个小部分。同时应该指出，每个程序之间的联系是飘忽不定的，或许其逻辑可以借鉴，或许其实相关性为零，很难针对广泛的程序有成熟的套路。不过，在编写过程中提醒自己需要注意的点还是很有帮助的。

- 再次明确从输入文件得到的数据结构及访问方式。

  虽然在上一章节中反复强调了这一点，但它仍旧可能成为你的程序的致命 Bug 的一部分。一个不太合适的数据结构只会让你的处理难度加大，同时限制了你做封装[1]处理，在扩展性和维护性上大大降低。明确自己在引用这些数据时动用了什么，将减少在中间数据传递过程中可能出现的问题。

- 对每一个运算步骤做到心中有数。

  在上一步骤做好的前提下，下一步应该关注的是每一个运算对象所经历的所有过程。它有可能经历循环，经历条件分支，也可能只是简单的数学运算。每一个复杂的运算逻辑某种程度上也不外乎这些内容的拼接组合，但在**开始编写**过程中应该有意识地将这些步骤分开。在着手于运算逻辑时，可以设置多个 Checkpoints，比如设置一个 flag 及条件函数来检查某些关键变量的状态，设置一个简单处理函数及对应存储变量来观察每一次运算过后是否是合乎物理常识的。在刚开始编写的过程中就能够有意识地在这一方面做好的话会非常有利于后续的 debug 过程，因为它能够帮助你了解每一个数据所经过的过程，从而让你迅速寻找到关键问题所在。

- 适当地开始封装处理。

  在程序编写完之前可以大致保留这些中间变量，但应带着明确的结构化的意识对所处理的内容进行一定程度上的分割，防止自己在写程序时写得过于流水账而增大后续工作量。当你对数据所经历的流程有清晰的把握时结构化意识会非常自然地产生。最简单的封装操作包括对矩阵的所有元素进行某种数学变换，对多个维度的数据进行统一的数据处理等。当完成相关部分的 debug 工作后，应开始考虑将其打包处理，利用函数或声明全局变量来自动完成某些工作，并让自己不用再担心这个地方出现问题。为此，大可以先另设一个小程序来尝试性地做出其中的一个 routine，输入预设的数据并核对结果来检验自己的运算逻辑是否合理。

- 坚持下去直至程序写完。

  当程序能够正常输出结果且符合直觉的时候才能松一口气，并享受其带来的成就感。但其间保留的中间变量仍应留存到该步骤，因为“能够输出结果”离一个“功能完善”的程序还有相当长的距离，而这些中间变量能够起到非常关键的作用。它可以让你得知每一个运算步骤所大致消耗的时间，提醒你应该如何无缝地添加一些新功能。

- 对程序进行优化。

  优化的重要性不亚于开始构建的过程。一个良好的优化能够使本来消耗过长时间的程序变得可以接受，能够有更多的尝试。从中间变量给出的信息来判断可能存

[1]特别是在面对一个大程序时，封装是一个非常关键的行为。”封”这个词恰恰说明了你将一些“东西”整合成为一个新的函数，具有明确的输入及输出的方式，或者成为对象，具有明确的输出内容，但无论如何你不需要再关注里面所包含的任何逻辑，如同你不需要知道 printf 是怎么实现的一样。

在的逻辑冗余，考虑是否存在更好的算法，是否能够有更好的数据访问形式等等，而这些应具体情况具体分析。同时可以开始着手处理内存堆放的问题，力求减少不必要的数据存储及访问。

- 对中间变量进行取舍。

  当程序优化到一定程度时，数据的存储及读取带来的时间延迟开始逐渐显现。因此需要对中间变量开始进行取舍，哪些适合作为过程文件中的一部分进行输出，哪些可能在后续的处理过程中利用价值很小而可以放弃占用的内存。它应建立在程序框架基本不会有大的改变的前提之上。

- 进行大规模的试验。

  当这些都完成时，就可以用大规模试验来考核程序的运行能力。它会帮助你寻找可能仍旧存在的漏洞及优化方法。它应建立在你的程序已经有足够的优化的基础上。

- 上传 Github。

  向世界装个逼。

我们再利用电化学模拟来看看大体上是如何完成程序编写工作的。

1. 确定数据框架，利用所学经验及语言特性来制成可以构建任意维，任意宽度的电化学模型的相关数据结构及对应访问函数。
2. 构建函数，输入一个点的坐标，读取该坐标对应浓度及在任意一个维度上 +1,-1 得到的新的坐标及这些坐标对应的浓度，按照一定的法则进行加减乘除，从而算出该坐标对应的近似二阶导。
3. 构建函数，让该坐标的二阶导直接对该坐标的浓度按照 Randles-Sevcik 方程进行修改。
4. 建立循环，使浓度变化函数能够遍及全空间。
5. 建立循环，引入电压变化，完成电压变化与浓度变化过程不断循环。
6. 引入输出函数，将中间的过程以合适的形式输出。

这仍旧是一个较为简单的运算逻辑，只是对数据的运用及存储有较高的要求。而在更为复杂的计算中，会牵涉到数值积分、高斯积分等需要外界帮助的情况。

3 输出文件

当你做好上述步骤后，事实上需要费脑子的地方已经不多了。在编写主程序的过程中应该早就完成了部分过程变量的保留及痕迹记录，它们可能直接以过程文件的形式输出进入硬盘空间，以释放硬盘空间；也可能记录在一个数组，最后进行整体输出，方便后续的数据处理。这两种方法的选择需要以具体所使用的空间大小来衡量，如果真的十分庞大（如一个数量比较庞大的双精度矩阵）不妨考虑直接输出成为文件，即便在后续程序需要也可以考虑反过来读取文件来对内存空间进行必要的管理。

文件的格式并不需要过于美观，毕竟我们也不至于做成商业软件。但一些必要的处理还是必要的，而这个必要的处理仍旧应建立在设计程序的时候铺设的桥梁上。如果在

进行时间模拟应充分注意将时间节点写入每一时刻下得到的各种数据，coordinates 也应做好一个大致的输入，从而方便后续利用关键字进行提取。从这一方面来讲，熟悉自己所用的数据处理软件是非常重要的。一般输出文件都是要以 txt 格式来进行输出，因为这个可以直接由主程序进行生成并写入，但 txt 转换为常用的数据处理软件所青睐的格式是有一定要求的。在庞大数据下过度依赖图形界面显得非常不切实际，而利用关键词搜索过滤来获取相关信息显得尤为重要。因此在每一个数据点赋予足够的信息（包括空间坐标、时间节点、特征性质等）会在后续处理中轻松很多。

bug 总会显得出其不意，因此一切过程变量都值得探讨是否写入输出文件。如果程序尚未完善，每一个循环后的变量值都值得进行输出，以判断是否运行正常。

应该来说，在并不以商业化为前提的情况下编写软件，在主体程序不应一并做出数据处理给出统计结果，而另设程序进行输出文件读取。

4 数据处理

数据处理并没有那么简单。事实上数据处理的最大的目标在于可视化，而可视化又是一个牵涉到人体感观的东西。美很难去用系统化语言进行定义，也因此一个漂亮的可视化结果可遇而不可求。不同的体系下对线条粗细、立体图角度都有着独到的要求，而一味地模板化只会失去主动性。

同时应该注意在设计程序的时候引入的条件。周期性边界条件是一个最为显著的例子，它的存在可能会使一个描述分子在空间中分布的图变得杂乱无章，因为有可能一部分原子在盒子的一侧，而另一部分的原子在另一侧，从而引入了贯穿整个盒子的线条。

同时，关键词的过滤、按照条件进行筛选仍旧是一个非常重要的技术，面对 txt 文件也有可能需要再将其还原为原来的数组模式进行处理，而这些都基于自己选择的数据处理软件进行充分实践。[2]

5 常见 bug

~~可是我遇到的 bug 都是自己 sb 造成的~~

1. ~~其实没有在进行循环~~
2. 数据层数读错，用 C 语言的角度来讲指针遍历的位置出错，就像

```
for(i=0;i<length;i++)
    for(j=0;j<length;j++){
        printf("%lf",*(*(matrix+j)+i));
        printf("\n");
    };
```

   这里面的 i 和 j 对调意味着输出方式完全不同。熟知自己的矩阵应该是按照行向量的方式对待还是列向量的方式对待。

3. ~~函数打错~~
4. 由于精度不够导致的神奇现象，甚至数据溢出
5. 内存管理不善导致的数组被冲散

[2] ~~所以为什么还不投入 *Mathematica* 的怀抱呢~~

6. ~~数学函数输错~~
7. 没有充分理解内置函数的行为导致的奇怪的现象
8. 没有及时关闭写入数据流，使得文件一直被占用
9. ~~其实没有 bug 硬是写进去了 bug~~
10. etc.

~~*Mathematica* 推荐原因~~

6 可能可以给自己提供的练习

1. 尝试着自己编写一个矩阵相乘对应的函数，并设计好最后输出的数据结构。
2. 在上一题的基础上尝试着做出对任意行数和列数的矩阵都能够通用的函数。
3. 尝试按照文中所述的流程完成一维的电化学模拟。
4. 了解他们一般是怎么进行数值积分的，自己寻找是否存在相对应的运算库，并完成 Hartree 程序的编写。
5. 下载同一个目录下的 npt.gro 文件，画出里面的分子分布，并自制一个径向分布函数对其进行统计，观察数据处理结果。