在我上研之后，主要负责导师项目中MATLAB转C++语言的工作。到目前为止已经给导师做过三个相关的项目。在转换过程中遇到了一些困难，但是结合本科时期的编程经验，问题都在一定程度上得到了解决。下面我分别从多维线性插值程序设计方法、龙格库塔程序设计方法、矩阵计算设计方法三个方面阐述我在做项目的时候的程序设计思路。

一、多维线性插值程序设计方法

多维线性插值是由一维线性插值的推广，因此两种算法的设计思想是相通的，这也意味着多维线性插值可以分解为一维线性插值问题。将上述思想直接通过C语言翻译出来，便可以通过目标点在多维空间中的投影，将一个多维线性插值问题转换为一“树状结构”的多个一维插值问题，该“树状结构”的计算可以用一个递归直接实现，下面介绍具体实现。

整个过程使用一个递归函数进行实现，每次递归传入的是待求解插值的坐标数组，在递归过程中需要从一维到n维进行遍历。在遍历过程中，首先判断该维度下的坐标是否为数据集节点的坐标，如果是数据集中节点维度坐标则进行下一个维度的判断，否则将求取该维度下该节点坐标的上下两个边界以及其对应的数据值。关键的操作在于：分别用上下边界代替待求解插值的数组在该维度位置的坐标，并将替换后的两个n维数组作为待求解的插值数组重新调用该递归函数进行求解，求解后的插值结果记作Xup与Xdown，刚才求得的边界记作Dup与Ddown，待求解数组在该维度下的坐标记作M\_，则直接返回（(M\_-Ddown)/(Dup-Ddown)\*(Xup-Xdown)），如果在维度遍历过程中发现待求解数组的各个维度坐标均为数据集中的标定坐标，则在循环外直接返回数据集该坐标组对应的数值。

下面给出具体实现的代码，其中：if\_a()函数用于判断该数串是否属于已知数据集；up\_a()函数用于获得某一维度下该点的数据上界坐标；r\_up()函数用于获取某一维度下该数据上界的值；down\_a()、r\_down()则与up\_a()、r\_up()相反。

1. **double** interPolation(**double** \*a){
2. **double** r\_up, r\_down;
3. **int** a\_up, a\_down;
4. **double** \*aa = (**double**\*)malloc(**sizeof**(**double**)\*dimension);
5. **int** \*aaa=(**int**\*)malloc(**sizeof**(**int**)\*dimension);
6. memcpy\_s(aa, **sizeof**(**double**) \* dimension,a, **sizeof**(**double**) \* dimension);
7. **for** (**int** i = 0; i < dimension; i++){
8. **if** ((\*(aaa + i) = if\_a(\*(a + i), i)) == -1){
9. a\_up = up\_a(\*(a + i), i);
10. \*(aa + i) = window[i][a\_up];
11. r\_up = interPolation(aa);
12. a\_down = down\_a(\*(a + i), i);
13. \*(aa + i) = window[i][a\_down];
14. r\_down = interPolation(aa);
15. free(aa);
16. free(aaa);
17. **return** (\*(a + i) - window[i][a\_down]) / (window[i][a\_up] - window[i][a\_down])\*(r\_up - r\_down)+r\_down;
18. }
19. }
20. **double** result = GET\_IT(aaa);
21. free(aa);
22. free(aaa);
23. **return** result;
24. }

在上述代码的撰写过程中，我一直以代码简洁和能够使用为目的，在一定程度上忽视了算法的效率问题。针对上述代码在效率上的弊端可以从下面几个方面出发进行改进和完善：

①递归函数其实是一种低效的算法实现方式，作为一个思维方式的直接程序翻译可以用递归函数来实现，但如果从效率入手可以将算法进行数学上的化简。通过进一步的思考和MATLAB实验，我得到以下新的思路：首先求取待求解点的相邻数据点（有2^n个），其次对每个数据点的数据值进行加权相加及为该待求解点的插值结果，通过化简可以知道权重即为数据点的权重为数据点相对于待求点的对角点与待求点在各个维度上的坐标距离的乘积与各维度上下界距离乘积的商。

②在上次与师兄交流的过程中受到启发，可以使用快速检索与插值检索方法代替二分法，用来寻找数据点在某一个维度下的上下界。当数据集是规则排列时（即各维度若像[2 4 6 8 10...]类的等差数列的情况），则无需使用二分等算法，可以直接求得，原则仍然是具体问题可以用来具体分析，以求得最优方法。

二、4阶龙格库塔程序设计方法

4阶龙格库塔程设计难点不在于算法，而在于对于模型的架构上，尤其是用C++做导弹的过程，需要将导弹本身与其制导控制律进行分离，通过各个类之间的调用完成4阶龙格库塔中的迭代。因此我在具体实现的过程中，把导弹的C++类与制导类、控制类分开进行了撰写。

三、矩阵计算设计方法

矩阵计算设计，难点不是矩阵函数的撰写，而在于如果能够实现如matlab一样可以使用返回量直接进行长式的计算。为了解决这个问题，我引入了三阶指针，通过三阶指针的传入与传出实现了长式的计算。并在计算函数中使用malloc()函数进行空间的申请，同时建立了内存池与内存垃圾回收机制在杜绝内存无限增长、减少空间申请释放操作的同时实现了矩阵的计算功能。