班　　级　力6

有限元法基础
桥梁设计竞赛报告

指 导 教 师：　张雄教授

院 (系) 名 称：　航天航空学院

专 业 名 称：　工程力学

学 生 姓 名：　毕恺峰　胡昌平　黄轩宇　易泽吉　张逸葑

二〇一八年三月

Contents

1　摘要

2　桥梁的基本选择与设计思路

2.1　悬索斜拉桥

2.2　拱形桥

2.3　xx 桥

2.4　xx 桥

3　桥梁的 Abaqus 实现

3.1　零件的设计

3.2　有限元单元类型的选择

3.3　有限元单元尺寸的选择

3.4　施加约束条件

4　桥梁的 cost 计算

4.1　Abaqus-Python 脚本读取过程

在桥梁的大体结构设计确定之后，在进行几何参数改良时，如果在 CAE 界面改参数将变得非常繁琐，而将模型进行 python 录制后直接读入相关参数进行计算将非常方便；与此同时，由于我们最终需要集中优化的参数为桥梁的最大位移，桥梁的总造价和承载应力情况，而 CAE 界面只能查看应力的大体分布情况，无法对所有单元的应力分布情况做一个具体的分析，因此我们还需要利用 python 脚本对整个 Abaqus 数据结构进行有选择性的数据读取，其中主要包括节点位移，单元应力，以及单元对应的高斯点坐标，单元几何属性等；与此同时我们可以利用 matlab 等软件作为 python 的外部接口，即为 python 脚本提供输入数据（inp 文件），并对输出数据进行相应的分析。整个脚本的程序逻辑结构如图4.1所示：

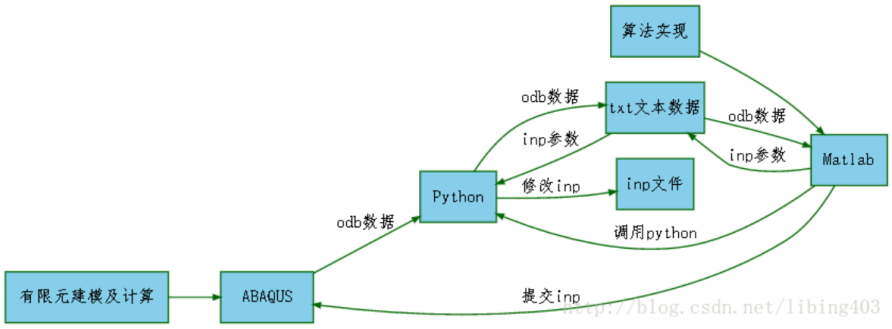

Figure 4.1　python 后处理设计逻辑

4.1.1　ODB 对象数据结构：

如图4.2所示：ODB 对象主要对包含计算模型对象数据（Model Data）和计算结果数据 (Result Data)，计算模型对象数据包含装配体 (rootAssenbly) 、零件 (parts) 、界面分类 (sectionCategories)、材料 (materials) 等子对象，计算结果数据 steps 包含分析步 (step) 、帧 (frame) 、历史变量输出 (historyoutputs) 和场变量输出 (field outputs) 等。场变量的读取路径:odb.Setps[].frames[].fieldOutputs[];

场变量包括: 应力分量-'S'; 应-'E'; 位移-'U';

历史变量的读取路径:odb.Setps[].historyRegions[].historyOutputs[];

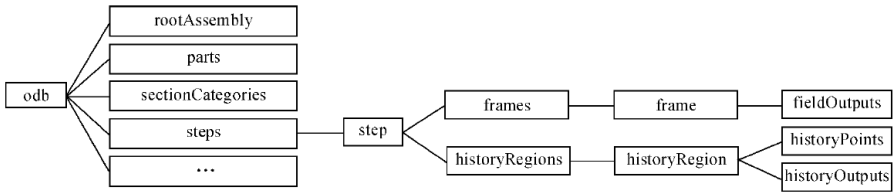

Figure 4.2　Abaque odb 数据结构示意图

4.1.2　MDB 对象数据结构

如图4.3所示：MDB 对象主要对包含计算模型对象数据 (Model Data) 和工作对象数据 (Job Data)，其中 model 对象，包含零件 (parts) 、材料 (materials) 等子对象。零件 (parts) 子对象包含单元 (elements) 、集合 (sets) 和节点 (nodes) 子对象，其下又包含编号 (label) 、坐标 (coordinate) 、单元所属节点组 (connectivity) 等属性。

几何集的读取路径:mdb.models[].parts[].sets[].odb.rootAssembly.instance[].elementSets[]；

几何集所属单元，节点信息读取路径:

单元编号:mdb.models[].parts[].sets[].elements[].elementLabel.odb.rootAssembly.Instances[].elementLabel；

单元所属节点组:mdb.models[].parts[].sets[].elements[].Connectivity.odb. rootAssembly.instances[].elementSets[]．elements[]. connectivity；

节点坐标:mdb.models[].parts[].sets[].nodes[].coordinates；

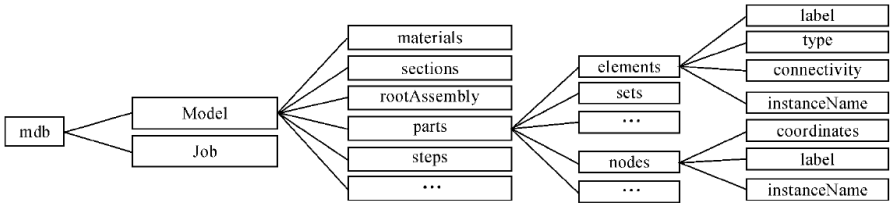

Figure 4.3　Abaque mdb 数据结构示意图

4.1.3　最大位移读取

我们在 odb 对象中把位移场’U’的数据读取出来，然后对节点进行循环得到最大的位移数值，如图4.4所示：

```
def Getmaxdisplacement():
    import math
    # Get the current working directory
    cwd=os.getcwd()
    # Open the .odb
    o=session.openOdb(name='{0}//Job-{1}.odb'.format(cwd,1));
    RS=o.steps['Step-1'].frames[-1].fieldOutputs['U'].values;
    maxdisp=0
    for i in RS :
        l=math.sqrt(i.data[0]*i.data[0]+i.data[1]*i.data[1]+i.data[2]*i.data[2])
        if l>maxdisp:
            maxdisp=l
    return maxdisp
```

Figure 4.4　最大位移读取程序示意图

4.1.4　单元应力和最大许用应力

首先先在 odb 对象中把应力场’S’中的数据读取出来，然后再将其写入 txt 文件中，同时输入相应的单元类型和单元号，以便之后对应相应的单元属性，如图4.5所示：

```
def Getmaxdisplacement():
    import math
    # Get the current working directory
    cwd=os.getcwd()
    # Open the .odb
    o=session.openOdb(name='{0}//Job-{1}.odb'.format(cwd,1));
    RS=o.steps['Step-1'].frames[-1].fieldOutputs['U'].values;
    maxdisp=0
    for i in RS :
        l=math.sqrt(i.data[0]*i.data[0]+i.data[1]*i.data[1]+i.data[2]*i.data[2])
        if l>maxdisp:
            maxdisp=l
    return maxdisp
```

Figure 4.5　应力读取程序示意图

4.1.5　单元属性读取

最后我们在 mdb 对象中得到所有单元的材料属性和坐标等参数，写入到另一个 txt 文件中，如图4.6所示：

```
o=session.openOdb(name='{0}//Job-{1}.odb'.format(cwd,1))
RF=o.steps['Step-1'].frames[-1].fieldOutputs['S'].values
with open( 'D:\Abacus\FEM Course\Bridge\Scomponent.txt','w') as f1:
 for v in RF:
    f1.writelines( str( v.baseElementType) + ' ')
    f1.writelines( str( v.elementLabel) + ' ')
    f1.writelines( str( v.mises) + ' \n')
```

Figure 4.6　单元属性读取程序示意图

之后再将相关的单元应力和单元信息参数一同导入到 matlab 中进行相关的循环检索，利用超过许用应力的单元对应到其单元类型，利用其单元数据计算出其

体积；最后检索所有的单元信息得到所有单元的体积总和，一方面计算相应的参数

5 桥梁的优化过程

对于桥梁优化的整体指标为竞赛说明文件中所述的 cost function 为

$$\Pi_i = \sqrt{0.4\left(\frac{d_i}{a_{1,i}}\right)^2 + 0.4\left(\frac{p_i}{a_{2,i}}\right)^2 + 0.2\left(\frac{v}{a_{3,i}}\right)^2} \tag{5.1}$$

其中 d_i 为桥梁平面的最大位移。同时记单元中有 $\frac{3}{4}$ 高斯点在测试工况下达到许用应力的 0.8 倍的单元为危险单元。p_i 反应了危险单元体积占整体单元体积的比例。

由于无从知晓其他组的各个指标，我们假设各组的指标间的比例关系与我们类似，所以我们的优化过程中简单的将代价函数替换为

$$\Pi_i = \sqrt{0.4\left(\frac{d_i}{b_{1,i}}\right)^2 + 0.4\left(\frac{p_i}{b_{2,i}}\right)^2 + 0.2\left(\frac{v}{b_{3,i}}\right)^2} \tag{5.2}$$

将原有的分母部分改成遗传算法中出现的分项指标的最大值。将此作为我们的优化目标。

整体的优化思路为，首先选取不同样式的桥，分析桥的构型，确定基本设计方案，以及设计方案中的可变参数，其次通过 Abaqus 的 Python 脚本接口，参数化的生成桥，之后同样通过 Python 脚本导出相关的各项指标，如桥面最大位移，以及危险单元占比，之后将桥的相关参数及各项指标导入优化算法，计算出最终的 cost，同时分析计算出下一步的各项参数，在 Abaqus 中画出桥，并计算 cost，之后不断迭代，达到指定步长后提取出全局最优解及相关参数。

5.1 算法的选择

5.1.1 离散与连续

算法的选择上有两个层面的选择问题，一是离散和连续算法的选择，二是确定不同种类的离散或连续算法。首先对各项指标分析，工程造价这一项指标可以比较显示的表达成不同单元体积与材料成本的乘积，但桥面最大位移，以及危险单元占比可能很难显示的表达，因此整体的代价函数中有一部分为有解析描述另一部分无解析描述。单纯采用连续的算法并不可行，此时可以采取的思路包括，采取分裂算子的思路，将连续指标与离散指标分裂开来，分部的优化迭代，但分裂算子方法较为复杂，同时对于纯离散问题，算子分裂也并没有成熟稳定的解决方案，因此我们选择了最后一种方案，即将整题的问题考虑为一个离散问题，直接用离散方法对其求解。

5.1.2 离散算法的选择

模拟退火算法具有摆脱局部最优解的能力，能够以随机搜索技术从概率的意义上找出目标函数的全局最小点。但是，由于模拟退火算法对整个搜索空间的状况了解不多，不便于使搜索过程进入最有希望的搜索区域，使得模拟退火算法的运算效率不高。模拟退火算法对参数（如初始温度）的依赖性较强，且进化速度慢。

遗传算法具有良好的全局搜索能力，可以快速地将解空间中的全体解搜索出，而不会陷入局部最优解的快速下降陷阱；并且利用它的内在并行性，可以方便地进行分布式计算，加快求解速度。但是遗传算法的局部搜索能力较差，导致单纯的遗传算法比较费时，在进化后期搜索效率较低。

综上所述，结合了此处桥梁设计问题的特点，以及算法的效率和可行性分析，因为结果随参数的波动可能较为剧烈，为较快的搜索到全局最优解的邻域，我们选取了遗传算法。

5.2 遗传算法的简介

生物在漫长的进化过程中，从低等生物一直发展到高等生物。这是自然环境选择的结果。人们研究生物进化现象，总结出进化过程包括复制、杂交、变异、竞争和选择。一些学者从生物遗传、进化的过程得到启发，提出了遗传算法（GA）。算法中称遗传的生物体为个体，个体对环境的适应程度用适应值表示。适应值取决于个体的染色体，在算法中染色体常用一串数字表示，数字串中的一位对应一个基因。一定数量的个体组成一个群体。对所有个体进行选择、交叉和变异等操作，生成新的群体，称为新一代。

遗传算法属于进化算法 Evolutionary Algorithms 的一种, 它通过模仿自然界的选择与遗传的机理来寻找最优解。遗传算法有三个基本算子: 选择、交叉和变异。数值方法求解这一问题的主要手段是迭代运算。一般的迭代方法容易陷入局部极小的陷阱而出现” 死循环” 现象，使迭代无法进行。遗传算法很好地克服了这个缺点，是一种全局优化算法。

5.3 程序设计思路

5.3.1 编码

（1）选择合适的编码方案，将各参量拼接转换为染色体。联系实际问题，为每个待优化的变量规定上界与下界，如桥墩的间隔这一参数就小于河床的宽度，大于要求的至少 150m。之后确定每个参量的基因组长度，即其离散的密度和优化的

精度，对于二进制编码，优化的精度为 $\frac{t_{upper-limit}-t_{lower-limit}}{2^n}$。将上下限中的点等分，对应到二进制编码中。

（2）选择合适的参数，包括每一步迭代中的群体大小、交叉概率 $P_{crossover}$ 和变异概率 $P_{mutation}$

（3）确定适应值函数 $f(x)$，考虑到这里的适应值为正值，所以对 cost 函数取相反数之后加上一个常数即可。

5.3.2 初始化

在最开始的桥的基础上，提取出参数，同时施加一些扰动。将这些作为第一代群体的一部分，另一部分由简单的随机染色体组成，这样既保证了合理解的邻域附近比较全面的搜索，同时完全随机的染色体也保证了群体的基因多样性。初始解的存在保证了优化算法的结果一定是优于最开始的计算结果。

5.3.3 繁衍

5.3.4 交叉

（1）对每串产生 [0，1] 间随机数，若 r>pc，则该串参加交叉操作，如此选出参加交叉的一组后，随机配对。

（2）对每一对，产生 [1，m] 间的随机数以确定交叉的位置。

(3) 将一对染色体 m 点后的位置互换

5.3.5 变异

如变异概率为 Pm，则可能变异的位数的期望值为 Pm ×m×M，每一位以等概率变异。具体为对每一串中的每一位产生 [0，1] 间的随机数 r，若 r<Pm，则该位发生反转，如对染色体二进制编码为数字 0 变为 1，1 变为 0。对群组不断进行变异和交叉互换操作，直到下一群体中个体数目达到所要求的 M。

5.3.6 选择

对群体中每一染色体（串）计算其适应值，同时计算群体的总适应值。计算每一串的选择概率 $Pi = \frac{fi}{F}$ 及累计概率。选择一般通过简单的古典概型描述，产生 [0，1] 间随机数 r，根据 r 所在区间选取对应的染色体个体。

5.3.7 算法总结

由于选择概率是由适应值决定的，即适应值大的染色体入选概率也较大，使选择起到” 择优汰劣” 的作用。交叉使染色体交换信息，结合选择规则，使优秀信

息得以保存，不良信息被遗弃。变异是基因中得某一位发生突变，以达到产生确实有实质性差异的新品种。遗传算法虽是一种随机算法，但它是有导向的，它所使用的” 按概率随机选择” 方法是在有方向的搜索方法中的一种工具。正是这种独特的搜索方法，使遗传算法自然地避开了其它最优化算法常遇到的局部最小陷阱。

遗传算法与传统的优化方法（枚举，启发式等）相比较，以生物进化为原型，具有很好的收敛性，在计算精度要求时，计算时间少，鲁棒性高等都是它的优点。

5.4 优化实例

对于这一优化算法我们选取了简单的示例进行验证，简单考虑一个悬索斜拉桥，具体的参数包括桥墩的跨度，每个支架所对应的悬索根数，桥墩的尺寸，悬索的面积。简单基于悬臂梁的推导，$d \propto x^4, M \propto x^2, M \propto \sigma_{max}^2, n \propto L^{-1}$。我们可以发现桥墩的跨度与桥面最大位移成四次方关系，假设单元的最大应力值呈线性分布，因此与危险单元占比个数呈二次方关系，与造价成反比关系。对于悬索根数，$d \propto L^4, L \propto n^{-1}$, 可以推出 $d \propto n^{-4}$, 与桥墩跨度类似，与危险单元个数呈负二次方关系，与造价成正比。桥墩的尺寸与悬索的截面积均与造价成正比，考虑其对危险单元个数与桥面最大位移影响不大。

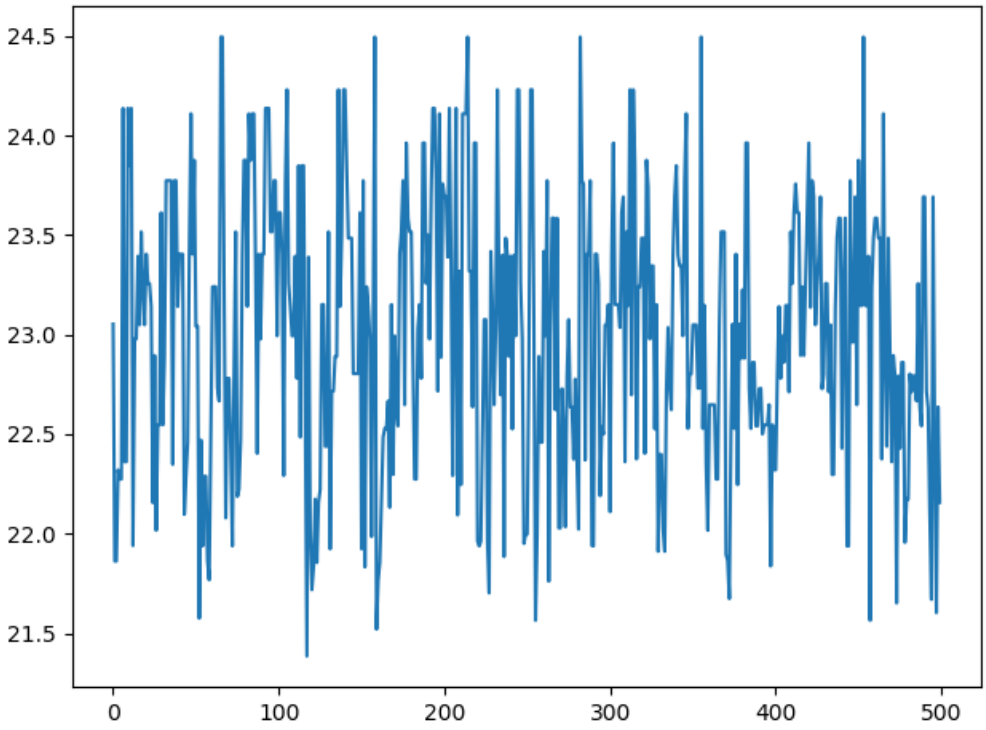

Figure 5.1 Symmetry and asymmetry lines and Boudary condition

优化结果如图所示，选取的桥梁跨度为 160m，选取的悬索根数（每段）52。

6 优化服务器

6.1 Abaqus 的 GPU 优化构架

6.2 服务器的 CPU-GPU 取舍

7 结论

最终我们选取的桥梁如下。。。Abaqus 相关文件见。。。。总体的 cost 为。。。。

Bibliography

[1] N. Bell and M. Garland. Efficient sparse matrix-vector multiplication on CUDA. NVIDIA Technical Report NVR-2008-004, NVIDIA Corporation, Dec. 2008.

[2] A. Bustamam, G. Ardaneswari, and D. Lestari. Implementation of cuda gpu-based parallel computing on smith-waterman algorithm to sequence database searches. In *International Conference on Advanced Computer Science Information Systems*, 2013.

[3] J. Choi, J. J. Dongarra, S. Ostrouchov, A. P. Petitet, D. W. Walker, and R. C. Whaley. Design and implementation of the scalapack lu, qr, and cholesky factorization routines. *Concurrency Computation Practice Experience*, 12(15):1481–1493, 2015.

[4] I. S. Duff and G. A. Meurant. The effect of ordering on preconditioned conjugate gradients. *Bit Numerical Mathematics*, 29(4):635–657, 1989.

[5] J. L. Greathouse and M. Daga. Efficient sparse matrix-vector multiplication on gpus using the csr storage format. In *International Conference for High Performance Computing, Networking, Storage Analysis*, 2014.

[6] D. Guo and W. Gropp. Adaptive thread distributions for spmv on a gpu. In *Extreme Scaling Workshop*, 2012.

[7] M. Heller and T. Oberhuber. Adaptive row-grouped csr format for storing of sparse matrices on gpu. *Computer Science*, 2015.

[8] E. J. Im, K. Yelick, and R. Vuduc. Sparsity: Optimization framework for sparse matrix kernels. *International Journal of High Performance Computing Applications*, 18(1):135–158, 2004.

[9] Z. Koza, M. Matyka, Łukasz Mirosław, and J. Poła. *Sparse Matrix-Vector Product*. 2014.

[10] M. Kreutzer, G. Hager, G. Wellein, H. Fehske, A. Basermann, and A. R. Bishop. Sparse matrix-vector multiplication on gpgpu clusters: A new storage format and a scalable implementation. In *Parallel Distributed Processing Symposium Workshops Phd Forum*, 2012.

[11] J. L. Greathouse and M. Daga. Efficient sparse matrix-vector multiplication on gpus using the csr storage format. *International Conference for High Performance Computing,Networking,Storage and Analysis, SC*, 2015:769–780, 01 2015.

[12] D. Langr and J. Trdlicka. Efficient converting of large sparse matrices to quadtree

format. In *International Symposium on Symbolic Numeric Algorithms for Scientific Computing*, 2015.

[13] S. Szkoda, Z. Koza, and M. Tykierko. Multi-gpgpu cellular automata simulations using openacc. 2014.

[14] J. Zhang, E. Liu, J. Wan, Y. Ren, M. Yue, and J. Wang. Implementing sparse matrix-vector multiplication with qcsr on gpu. *Applied Mathematics Information Sciences*, 7(2):473–482, 2013.