有限元大作业报告
暨 STAP++ 程序说明文档

组长：黄云帆
组员：陈一彤 邓博元 管唯宇 杨正宇 卢晟昊

2018 年 1 月 7 日

目录

前言

有限元法基础是清华大学首批四门“挑战性示范课程”之一，基于有限元程序 STAP++ 的功能扩展的课程大作业对同学们的知识迁移与应用、程序设计与实现、时间的分配与管理以及团队分工与合作等能力都提出了比较高的要求，无疑很好地诠释了挑战性的意义。这篇报告正是对我们团队近三个月以来工作成果的集中总结，同时也作为扩展后的 STAP++ 程序的说明文档以飨读者。

第 1 章对桥梁算例作了概括性的描述，并给出了大规模算例的评测结果以及后处理得到的变形图象与应力云图。值得一提的是，在内存优化的评测中，本组以领先第二名近一倍的成绩拔得了所有使用稀疏直接求解器的小组中的头筹。与此同时，本组在保证程序的稳健性与计算结果的精度的基础上，在程序效率的评测方面也名列前茅。除此之外，在充分考虑了约束方式的合理性以及约束输入格式的多样性的基础上，本组的位移图像与应力云图均取得了与 Abaqus 最为接近的结果。

第 2 章介绍了扩展后 STAP++ 程序的框架，具体包括前处理模块、组装刚度阵、求解器模块、后处理模块以及测试模块。有关具体的输入输出文件格式，可以参看附录 A 和附录 B。

- 前处理使用 Python 完成。我们的前处理不仅严格遵守了 STAP++ 的约定输入格式，输出的 dat 输入文件完全兼容已有标准，而且对输入参数也具有一定的鲁棒性。
- 后处理使用 Tecplot 完成。我们的后处理较为充分考虑了工程实际的需要，输出了包括变形后节点位置、应力不变量、von Mise's 应力、各应力分量在内的位移与应力数据，并且将构造单元以 block 六面体实体格式进行输出，方便强度准则的应用。
- 测试模块是本组工作的一大亮点。通过使用持续集成服务，在每次与远端相关的推送与合并之前都可以进行一次编译检查和分片实验测试，防止错误和冲突影响到代码。这使得我们大幅度减少了 code review 的时间，同时保证了 master 的稳定性。并且，我们的测试脚本具有非常好的跨平台能力，能够分别在 Windows 和 Linux 平台上自动链接 MKL 静态链接库，无需再手动进行任何设置。

第 3 ~ 4 章集中介绍了扩展后的 STAP++ 程序拥有的基本单元与补充单元，共包括 Bar, 4Q, 3T, 9Q, 8H, 欧拉-伯努利梁、铁摩辛柯梁、矩形薄板、平板壳、截锥壳等 10 种常规单元，以及无限单元和过渡单元 2 种扩展单元。利用上述单元可以解决包括桥梁算例在内的各种类型的弹性力学问题，如平面问题（包括常规平面问题、无界问题、多尺度网格问题）、空间问题（包括空间杆系问题、圆柱壳问题、一般轴对称壳问题）等。一方面，我们充分考虑了包括梁板壳在内的构造单元的旋转自由度约束、个别方向刚度缺失等问题，通过改进输入文件格式、增加旋转自由度手动设置标签 RotationDOFManuallyInputFlag 等途径实现了在兼容已有输入格式基础上的完善。另一方面，在梁单元的构造中考虑了包括扭

转在内的所有变形模态，并增加了适用于短粗梁的铁摩辛柯梁单元，可以更为有效地再现工程中可能出现的各种梁结构问题。

第 5 章重点介绍了 STAP++ 程序的扩展功能，包括稀疏直接求解器、模态分析以及分片应力恢复等。稀疏求解器的难点在于非零元位置的标记工作，我们在避免内存访问冲突的基础上，通过在求解器阶段开启多线程选项以及内外存自动切换，有效实现了效率的大幅提升。由于时间所限，模态分析与分片应力恢复的工作均是以典型单元作为示范进行的，前者实现了基于 Bar 单元的广义特征值问题求解，后者则基于 8H 单元实现了分片应力恢复功能。上述扩展功能极大地提高了 STAP++ 的性能，增强了后处理的功能，拓展了处理复杂问题的能力，可以说是本组工作的又一大亮点所在。

STAP++ 程序的这一最终版本是团队所有成员两个多月以来的不懈努力与紧密合作的结晶。我们的 FEM-3 团队从第 5 周周五晚第 1 次组会开始，到第 15 周周三下午的最终答辩结束，经历了共 10 次常设组会以及不计其数地线上线下讨论。从最初讨论程序各部分工作量的调研分工，到根据工作量与个人能力特点进行组内的任务分配，再到基本单元的集中处理冲突到深夜、专题调研的积极讨论到闭馆，最后到课程最终答辩时的胸有成竹……团队合作为更加高效地完成工作奠定了良好的基础，让我们在面临抉择时做出的每一个决定都更加踏实可靠，更是在紧密的合作中促成了团队每一个人之间的深厚革命情谊。在程序架构中，我们从多个方面尝试了对程序进行优化，详细的优化尝试请参看附录 C。这里要特别提到的是，Git 作为一个敏捷高效的分布式版本控制系统，对于初学者其实并不是特别友好，这里要特别感谢管唯宇同学在有关 Git 软件的使用上对其余团队成员给予的热情帮助。详细的小组合作清单可以参看附录 D。

最后，感谢张雄老师和宋言学长在整个项目完成工作中对本组全组同学的耐心指导！感谢在部分任务中，其余组的部分同学与本组同学的积极讨论与互相扶持！最要感谢的，是我们组里的每一位同学，感谢大家一直以来的不懈努力以及团队所有成员的通力合作！

FEM-3 全体成员

钱学森力学班 2015 级

2018.1.7

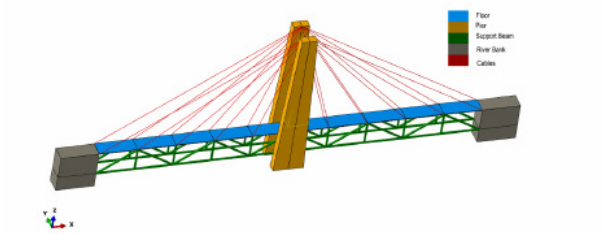
Chapter 1

问题描述与成果展示

这一章简述了大作业的题目要求并列举了评测结果 (见 1.1 节)，并就 STAP++ 的结果与 ABAQUS 的结果进行了对比分析 (见 1.2 节)。

1.1 问题简述与评测结果

- 扩展后的STAP++程序必须能够求解给定的桥梁问题（同时包括杆单元、梁单元、六面体实体单元和板单元），并用ABAQUS进行验证



- 除以上基本要求外，各组可以自行选择增加其他功能，包括但不限于分片应力恢复(SPR)、稀疏求解器、半带宽优化、无限单元、超级单元、过渡单元、模态分析、动力学响应分析和弹塑性杆单元等。

图 1.1: Problem

算例具体描述如下：

Job	*NumNode*	*NumEle*	*S4R*	*C3D8R*	*B31*	*T3D2*
1	4.16E3	2.88E3	4E2	1.76E3	7.04E2	20
2	3.72E4	3.04E4	2.5E3	2.65E4	1.35E3	20
3	\	2.33E5	1E4	2.2E5	2.7E3	20
4	1.91E6	1.81E6	4E4	1.76E6	5.42E3	20

经过大规模评测，测试结果如下：

Job	t_A/s	t_B/s
1	0.27	0.1016
2	2.35	1.18585
3	$25.0_{\ 6244M}$	13.6638
4	\	3090.13

其中，环境 A [1] 和环境 B [2]的说明见脚注。值得说明的是，本组在环境 A 中对 Job—3 进行内存优化测试的结果为 $1385M_{\ 132.63s}$，为相同求解思路（即直接求解刚度方程）、相同测试环境下所有小组中的最好结果，同时保证了较高的求解精度。

1.2 变形图像与应力云图

这里我们仅以 Job—1 为例进行展示。其中，图 1.2 展示了利用 STAP++ 程序算得的系统变形后图像，由对称性可以初步判断计算结果的合理性。图 1.3 ∼ 1.5 分别展示了利用 STAP++ 程序或 ABAQUS 软件计算得到的系统应力云图。为了便于对照，这里二者的位移放大倍数均取成了 107.7 ，并且尽可能地选取了一致的应力标尺。

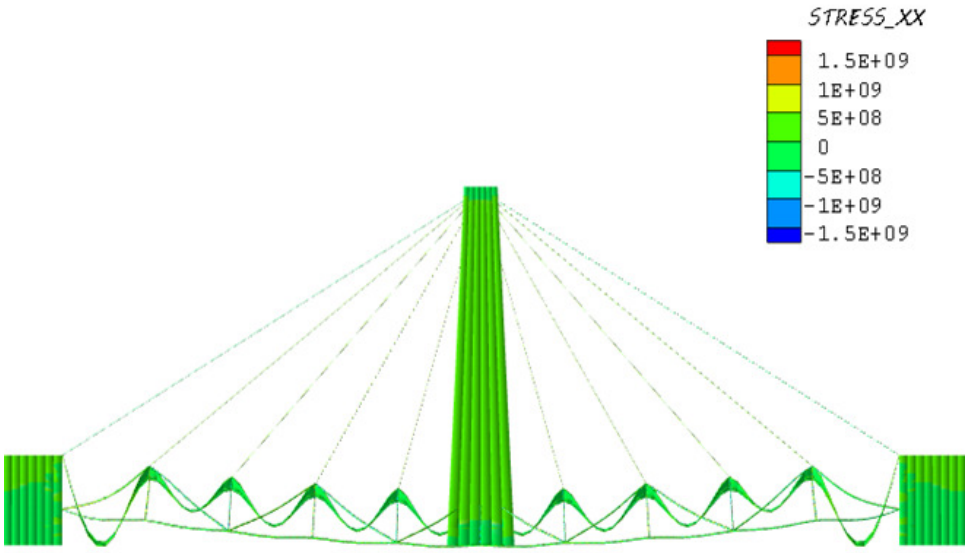


图 1.2: STAP++ 算得的系统变形后图像

对比图 1.3 与图 1.4 可以发现至少三方面的差异：

1. 梁的位移差异很大，STAP++ 的结果中甚至出现了明显的梁外翻的现象。这是由于在 STAP++ 程序中，将桥面附近相连的平板壳单元与梁单元的旋转自由度处理为固接约束，因此梁会随着桥面的平板壳单元发生较大的旋转。相比之下，由于固接的方式更接近工程实际，因此我们建议在下排两侧梁之间增加一些梁单元进行加固，以提高子系统的刚度。

2. 梁的图示应力有一定差异，STAP++ 的应力云图比 ABAQUS 的结果大了很多。实际上，这一差异是由于 STAP++ 输出的是梁表面的应力，而 ABAQUS 输出的是中性面上的应力，因此这一差异的出现是不足为奇的，并且 STAP++ 的输出更有利于材料力学中强度准则的应用。

3. 桥中心的桥柱变形有一定差异，STAP++ 的结果中变形稍小。注意到图 1.4 中桥柱顶端出现了明显的网格畸变，可以初步判断出现了零能模态。通过查阅 ABAQUS 文档

[1]环境 A：CPU: Intel Xeon CPU e5-2620 v4，2.10Hz，16核；内存: 64G；操作系统: win10；MSVC 编译, O2 优化.

[2]环境 B：CPU: Intel Core i9-7920X，4.40Hz，12核；内存: 32G；操作系统: ubuntu16.04；clang 3.8.0 编译, O3 优化, Intel Parallel Studio XE 2018.1.038 静态链接, OOC 使用 HDD.

我们了解到，建模时对三维实体单元使用的 *C3D8R* 实际上利用了减缩积分，而我们的 STAP++ 使用的是精确积分。因此，后者会出现一定程度的剪切闭锁，而前者尽管避免了剪切闭锁，却很有可能出现零能模态。我们还利用 ABAQUS 选取精确积分的实体单元进行建模，由此得到的图 1.5 中网格畸变现象消失，并且位移结果相比之下明显变小，这也在一定程度上验证了我们的上述判断。

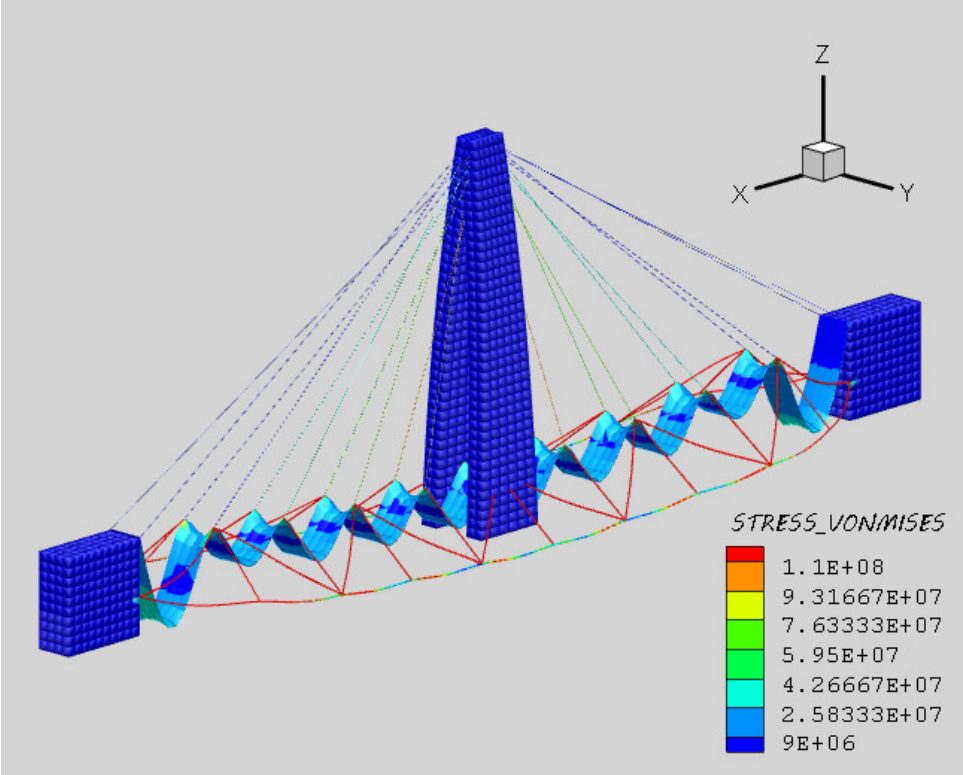


图 1.3: STAP++ 算得的 von Mise's 应力云图

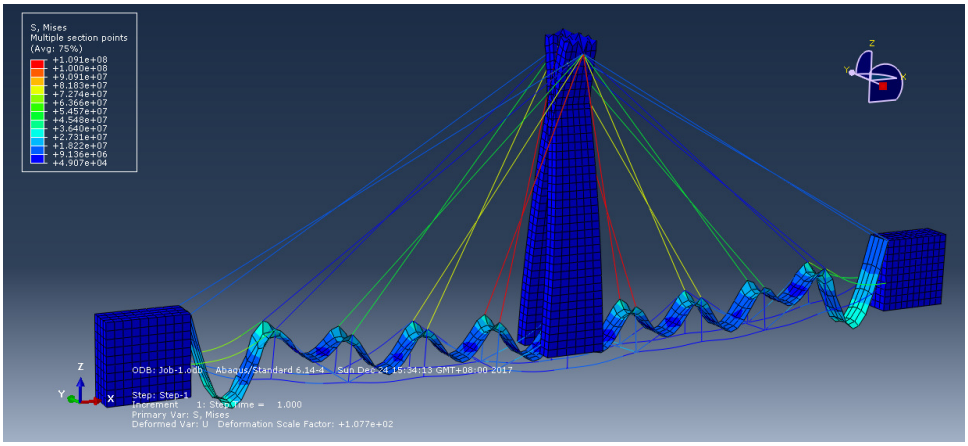


图 1.4: ABAQUS 算得的 von Mise's 应力云图 (实体单元利用了减缩积分)

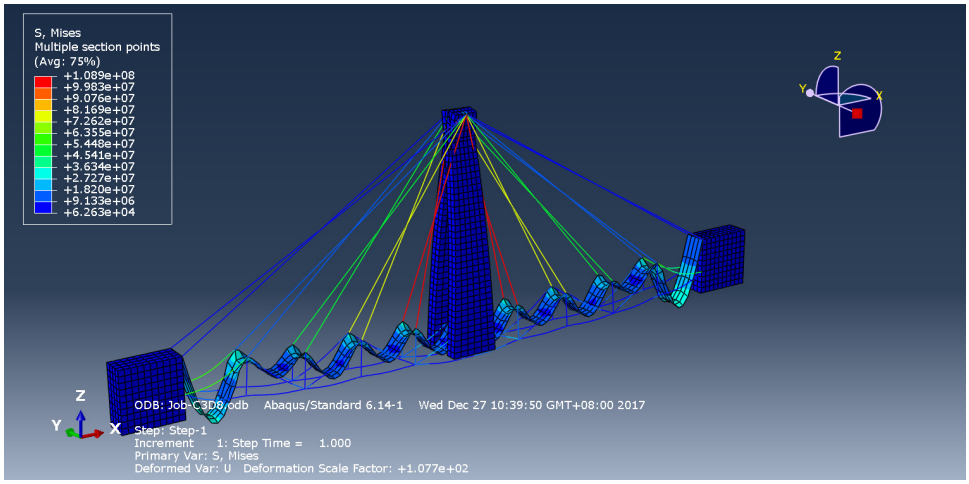


图 1.5: ABAQUS 算得的 von Mise's 应力云图 (实体单元未利用减缩积分)

Chapter 2

STAP++ 程序框架

这一章介绍了 STAP++ 有限元程序求解弹性力学问题的整体框架，包括前处理模块 (见 2.1 节)、组装刚度阵 (见 2.2 节)、求解器模块 (见 2.3 节)、后处理模块 (见 2.4 节)和测试模块（见 2.5 节）。其中，组装刚度阵模块包括单元刚度阵的生成以及总体刚度阵的组装两部分，前者将在第 3 章以及第 4 章详细说明，后者的过程是平凡的，这里不再赘述。另外，本章只涉及求解器模块的概述，有关稀疏求解器的详细内容见扩展功能一章的第 5.1 节。

2.1 前处理模块

2.1.1 前处理概述

前处理模块使用 Python 完成，支持 Python 3.6+ 环境，环境依赖为 numpy。

前处理模块的使用方法为

```
Usage: main.py [options] inputFile

Options:
-h, --help              show this help message and exit
-o DEST, --output=DEST
                        output dat file to dest
```

对输入的参数，前处理有一定的鲁棒性。

2.1.2 前处理结构

前处理结构分成三个大部分：解析部分、输出部分和计算部分。另外，还有 ABAQUS 模块（定义了 ABAQUS 中的数据结构）、STAPpp 模块（定义了 STAP++ 中的数据结构）和一个简单的进度条库，在标准等宽字体下有很好的效果。

其中，解析部分代码位于 ABAQUSparser.py 文件中，负责解析 inp 文件，返回一个 dict 对象，将结果传给输出部分。这一部分通过一个 ABAQUSparser.Parser 类实现，通过调用 Parser.parse 函数实现解析输入文件后返回约定的数据。

因为本次大作业中，输入文件的格式比较死，因此我们没有花费时间去实现一个递归下降语法分析器，而是基于本次的输入格式简单实现了一个顺序解析器。这个解析器会以依次查找关键词的方式来实现解析。通过 Parser.parseHeading、Parser.parseParts、

Parser.parseAssembly 等函数来依次解析，然后调用 Parser.analyse 函数，将 ABAQUS 中的类转换为 STAPpp 中的类，进行单元的编号、节点的去重、编号，材料截面的转换，并返回约定格式的 dict 对象。

输出部分代码位于 outputter.py 文件中，负责将解析部分返回的数据按照 STAP++ 的约定格式输出到对应的输出文件中。

计算部分独立为一个模块，是因为考虑到在解析部分解析完输入文件后，其实已经可以释放对应 ABAQUS 部分的储存空间，因此在解析部分不计算体力，而是将节点数据、单元数据输出到缓存文件中，再用计算部分读取计算，最后进行文件拼接。这样在不明显降低速度的前提下，我们可以大幅减少内存占用（峰值减少了接近一半）。

计算部分读取缓存文件中点和单元数据以及解析部分缓存的材料信息，用高斯积分计算体力，根据四种单元类型（四边形、杆、口字梁、六面体）计算等效节点力。

我们的前处理严格遵守了 STAP++ 的约定输入格式，输出的 dat 输入文件完全兼容已有标准。

2.2 组装刚度阵 (略)

2.3 求解器模块

这一部分，我们在原有的 LDLT 求解器的基础上增加了 Intel MKL Pardiso 稀疏直接求解器。 Pardiso 稀疏直接求解器输入格式要求是稀疏矩阵储存方式，支持的比较好的是行压缩储存格式（CSR），我们也选用了 CSR 矩阵作为我们储存刚度阵的方法。有关稀疏求解器的详细内容请见第 5.1 节。

2.4 后处理模块

2.4.1 后处理环境的选择

课上老师所推荐的主要后处理软件有二，即 Paraview 和 Tecplot。两者都是功能强大的可视化工具，并且在文献 [1] 附录中都提供了详细的教程。我们可以认为难度是近似的。其中，Paraview 对于数据的读取由于采用二进制的 VTK 等格式而更快，更加擅长动画读取。而 Tecplot 可以直接读取 dat、txt 等多种文本格式文件，使用更加方便。

由于组内同学的课外研究所使用的商用软件 (Coventor) 配有 Tecplot 的后处理接口，我组同学觉得可以趁此机会进一步深入掌握该软件的使用，于是选择了Tecplot 360 作为后处理工具。除此之外，该公司还有 Tecplot focus 等其他可用于后处理的子产品。

2.4.2 Tecplot 程序使用探究

虽然文献 [1] 中对于基本输入格式给出了范例，但仍有我们首先对 Tecplot 所能接受的输入文件进行一些基本尝试和研究。

1. 杆单元的绘制。在文献 [1] 的附录中略去了线段的单元类型，但事实上依照计算和单元特性，bar 单元、梁单元等使用线段类型是自然的想法。经过研究，成功地在后处理中加入了线段单元。但在最终的效果图中因为线段类型不能绘制云图而最终将梁、杆单元都使用六面体绘制。

2. 单元结点的输入顺序。为了提高后处理输出文件的简便性，并提高后处理的可靠度、鲁棒性，测试不同的结点输入顺序对后处理绘图的影响十分重要。经过大量测试我们发现，按照下图中的编号，六面体使用如下顺序：1-2-3-4-5-6-7-8, 1-4-6-7-2-3-5-8 均可以顺利读出。在最终的程序编写中，我们对 8H、杆、梁、板、壳均使用的是六面体单元，3T、4Q、9Q 使用的是四边形单元。

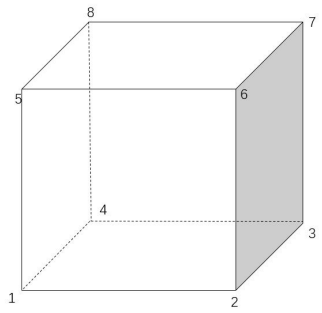


图 2.1: 六面体结点排序

3. 使用 ZONETYPE 和使用 PEPOINT 的区别。文献 [1] 中未提及的另一个是输出点格式问题，在有限元格式下，仍可选择 `DATAPACKING = POINT, ZONETYPE = FEBRICK` 或 `F = FEPOINT` 两种模式。经过测试，这两种模式都可以在 Tecplot 中画出理想图像，但是后者的效果更好。因此最终实现时我们选取了后者。

4. Tecplot 输出效果调试调整和切片。Tecplot 提供了丰富的设置调整，如背景颜色设置、单元颜色设置、三维切片等。对这些功能的尝试帮助我们学习了该软件的高阶使用，并画出了效果更好的后处理图像（如图 1.3 ）。

2.4.3 后处理程序实现

主要的实现算法分为如下几步，主体程序请参见 `PostOutputter.cpp`。

1. 读入输入文件的文件名 filename，控制信息。写入新文件 `filename_post.dat`。
2. 读入单元组（单元类型）信息，该单元组所包含的单元数目、对应的 Tecplot 画图类型等。
   (a) 在每个单元组开始前加入信息行，写入 ELEMENTTYPE, N, E, F, C 等信息。不采用从输出文件中读取数据，是为了节约时间，避免重复劳动。
   (b) 调用写在每个单元 cpp 中的 ElementPostInfo 函数，将之前计算的位移调用，分别输出结点坐标、位移、各个应力分量。
   (c) 加入放大因子 coeff，按序输出变量：
      i. 形变后坐标：$x_{post}, y_{post}, z_{post}$
      ii. 三个应力不变量：$\sigma_1, \sigma_2, \sigma_3$
      iii. von Mise's 应力：$\sigma_{vonMises}$
      iv. 六个应力分量：$\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \tau_{xy}, \tau_{yz}, \tau_{xz}$
3. 根据上方位移的输出顺序，按编号顺序输出每个单元的结点编号。

绘制的图形示例如图 1.2。

2.4.4 后处理附加功能及优化

Tecplot 作为一款功能强大的可视化软件，其中的一些功能可以帮助我们更好地实现有限元的后处理：

1. 3D 切片功能。Tecplot 360 画出的单元并不是透明的，所以仅从外观难以看到内部各单元的应力分布情况。可以使用它的三维切片功能，分析结构体内部的应力情况。
2. 背景设置与分量切换。可以通过 Frame 菜单下的 edit current frame 功能切换背景颜色，获得更好的视图效果（如 Paraview 的默认背景效果是灰黑色），同时可以方便的切换绘制各个应力分量的云图。

结合 Tecplot 360 的丰富功能，我们也对后处理进行了进一步优化：

1. 结合后处理输出的三个应力分量，可以方便的使用 von Mise's 强度准则、Tsai-Wu 断裂判据等，进行强度核算。具有更高的工程实用性。
2. 由于单元划分较密，应力平滑和平均的效果不甚明显。但是为了获得更好的后处理效果，我们也在8H单元上实现了分片超收敛应力恢复功能 (SPR)，具体请参见 SPR 章节 (第 5.3 节)。

2.5 测试模块

因为我们组使用 GitHub 的私有库，其相对于 GitLab 服务器最大的好处之一就在于对公网开放，因此可以引入第三方服务。在注册 GitHub 教育优惠后，我们可以免费使用 travis-ci.com 提供的持续集成（Continuous integration, CI）服务，这使得我们在每次 push 和提交 Pull Request 的时候都可以进行一次编译检查和分片实验测试，防止错误和冲突影响到代码。在实际使用过程中，CI 多次起到了作用，代码编译错误或是分片试验没通过就不能通过检查，也不能合并到主分支，大幅度减少了 code review 的时间。我们每一个 master 上的提交都能完成编译且通过分片测试，保证了 master 的稳定性。

2.5.1 测试模块的实现

首先，我们实现了一个跨平台的判断分片实验通过与否的脚本，见 `data/verify.py`。该脚本可以找到输出文件中所有点的位移和转角，并通过读取 `data/accurate.json` 获得解析的精确解表达式，自动计算精确解，与输出文件中的计算值比较，在设定的阈值（10^{-4}）精度下通过分片实验。

该脚本非常易于添加新单元，只需要在 `data/accurate.json` 添加新的精确解析解，并在测试脚本 `data/run-patch.py` 中添加测试即可。

添加精确解的过程也很简单，在 `data/accurate.json` 中如下书写线性表达式即可。

```
"3T": {
    "args": {
        "b": 10,
        "E": 1.0e6,
        "v": 0.3
    },
```

```
    "dx": "b/E*node.x",
    "dy": "-v*b/E*node.y"
}
```

通过运行 data/run-patch.py，将编译好的二进制 stap++ 地址作为命令行参数传入，即可一次性进行全部单元的分片试验。命令行如下：

```
$ py ./data/run-patch.py ./build/stap++
```

另外，在 test/test.py 中，我们还实现了一个自动编译、自动测试的跨平台测试脚本。这个脚本通过调用 cmake，分别在使用 LDLT 和 Pardiso 求解器两个选项的情况下进行编译和测试。它跨平台，在 Windows 下通过调用 MSBuild.exe 进行编译，在 Linux 下通过调用 Makefile 进行编译。调用命令行如下：

```
$ py ./test/test.py
```

2.5.2 持续集成的实现

在绑定 travis CI 后，在项目根目录下添加 .travis.yml 文件，在其中实现了自动安装依赖 MKL 和 eigen-3 的功能。安装好依赖之后，自动运行 test/test.py，进行编译和分片实验的测试。只有全部分片实验通过，测试才算通过。测试过程会重复两遍，分别使用 gcc 和 clang 作为编译器。

为了提高编译效率，我们设置了每次编译后 travis 自动缓存 MKL 的 include 文件夹和链接库文件夹。在没有找到缓存的情况下，为在 travis CI 上自动安装 MKL，我们还实现了一个 test/installMKL.py 脚本，它可以在 Linux 平台下自动安装 MKL。

另外，为了在 Linux 下正常测试，我们对 src/CMakeLists.txt 进行了调整，在其中设置了 include 路径和链接路径，能够分别在 Windows 和 Linux 平台上自动链接 MKL 静态链接库，无需再手动进行任何设置。因此，我们的程序有极好的跨平台能力。

Chapter 3

基本单元

这一章着重介绍在桥梁算例中使用的各种单元类型。

3.1 Bar 杆单元 (略)

3.2 8H 实体单元

3.2.1 单元构造

8H 单元是线性单元，可以看出是 4Q 单元的一个自然扩展。我们首先考虑位移函数。由于 8H 单元有 24 个自由度，但是三维二阶完备多项式，每个方向有 10 项，所以每个位移方向的两项应当被去掉。考虑到方向性，我们去掉三项二阶式而增加一项三阶式，得到如下的位移函数：

$$
\begin{aligned}
u &= \alpha_1 + \alpha_2\psi + \alpha_3\eta + \alpha_4\zeta + \alpha_5\psi\eta + \alpha_6\eta\zeta + \alpha_7\psi\zeta + \alpha_8\psi\eta\zeta, \\
v &= \alpha_9 + \alpha_{10}\psi + \alpha_{11}\eta + \alpha_{12}\zeta + \alpha_{13}\psi\eta + \alpha_{14}\eta\zeta + \alpha_{15}\psi\zeta + \alpha_{16}\psi\eta\zeta, \\
w &= \alpha_{17} + \alpha_{18}\psi + \alpha_{19}\eta + \alpha_{20}\zeta + \alpha_{21}\psi\eta + \alpha_{22}\eta\zeta + \alpha_{23}\psi\zeta + \alpha_{24}\psi\eta\zeta.
\end{aligned}
$$

随后，由以下形式的形函数进行插值：

$$
N_i = \frac{1}{8}(1+\psi_i\psi)(1+\eta_i\eta)(1+\zeta_i\zeta), \quad i = 1, 2, ..., 8.
$$

则有应变矩阵:

$$
B_i = \begin{pmatrix}
\frac{\partial N_i}{\partial x} & 0 & 0 \\
0 & \frac{\partial N_i}{\partial y} & 0 \\
0 & 0 & \frac{\partial N_i}{\partial z} \\
\frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} & 0 \\
0 & \frac{\partial N_i}{\partial z} & \frac{\partial N_i}{\partial y} \\
\frac{\partial N_i}{\partial z} & 0 & \frac{\partial N_i}{\partial x}
\end{pmatrix}.
$$

其中，$B = [B_1\ B_2\ B_3\ B_4\ B_5\ B_6\ B_7\ B_8]$。

为了将形函数映射到物理坐标系中，我们使用和 4Q 方法类似的 Jacobi 矩阵。由此构造出单元刚度阵：

$$
K^e = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} B^T D B |J| d\psi d\eta d\zeta.
$$

由此可以构造出总体刚度阵，从而求解系统刚度方程。

3.2.2 单元验证

为了验证所构造的 8H 单元，我们采用一个线性场的分片试验进行 test C。由于线性位移场实质上只包括常正应变与常剪应变两种情形，因此我们下面分别构造了单轴拉伸与纯剪载荷的算例进行验证。分片试验由 7 个单元组成，单元划分方式如图 3.1 所示。

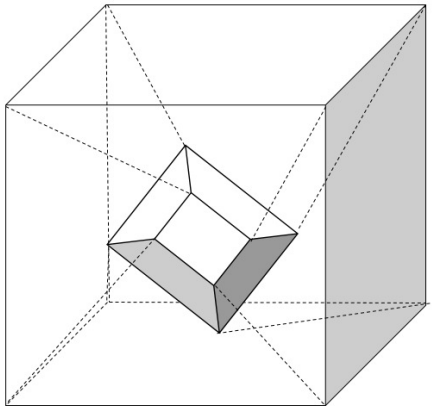
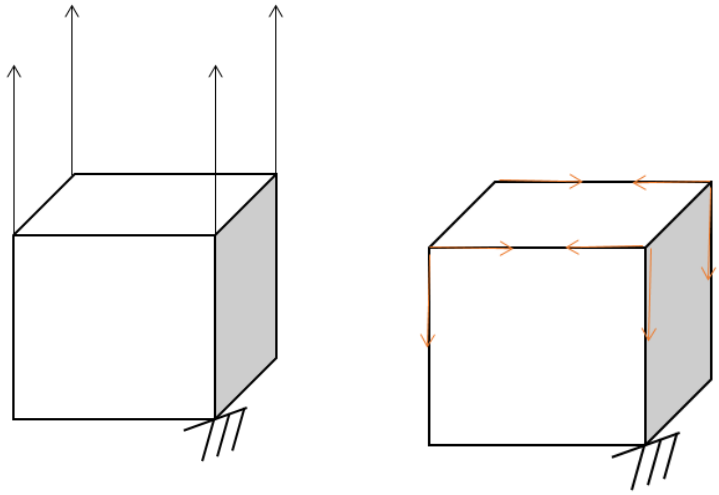


图 3.1: 分片试验单元划分示意图



(a) 单轴拉伸　　(b) 纯剪载荷

图 3.2: 分片试验载荷与约束示意图

单轴拉伸

首先是单轴拉伸的分片试验，施加如图 3.2(a) 所示的载荷。相应的位移场精确解应为：

$$
\begin{aligned}
u_x &= -\nu \frac{4P}{El_z l_x} x, \\
u_y &= -\nu \frac{4P}{El_z l_y} y, \\
u_z &= \frac{4P}{El_z^2} z.
\end{aligned}
$$

与图 3.3 所示的计算值对比，可以看到符合得很好，这说明所构造的 8H 单元具有至少二阶精度。实际上，由理论分析易知，8H 单元确实只具有二阶精度。

D I S P L A C E M E N T S

NODE	X-DISPLACEMENT	Y-DISPLACEMENT	Z-DISPLACEMENT
1	-4.00000e-002	0.00000e+000	0.00000e+000
2	-4.00000e-002	-4.00000e-002	0.00000e+000
3	2.08167e-017	-4.00000e-002	0.00000e+000
4	0.00000e+000	0.00000e+000	0.00000e+000
5	-4.00000e-002	-9.02056e-017	2.00000e-001
6	-4.00000e-002	-4.00000e-002	2.00000e-001
7	-1.45717e-016	-4.00000e-002	2.00000e-001
8	-2.05630e-016	-4.85723e-017	2.00000e-001
9	-2.40000e-002	-8.00000e-003	4.00000e-002
10	-2.40000e-002	-2.40000e-002	8.00000e-002
11	-8.00000e-003	-2.40000e-002	4.00000e-002
12	-8.00000e-003	-8.00000e-003	4.00000e-002
13	-2.40000e-002	-8.00000e-003	1.20000e-001
14	-2.40000e-002	-2.40000e-002	1.20000e-001
15	-8.00000e-003	-2.40000e-002	1.20000e-001
16	-8.00000e-003	-8.00000e-003	1.20000e-001

图 3.3: 单轴拉伸位移场计算结果

纯剪载荷

考虑到分片试验是单轴拉伸，再构造一个纯剪的算例，加载方式如图 3.2(b) 所示。进而得到了如图 3.4 所示的计算结果，这一结果与理论解符合得很好。这里的 8H 单元使用的是应力平滑 (即 Gauss 点的应力外推) 后的应力。关于应力的进一步处理，将在 SPR 章节 (第 5.3 节) 中继续讨论。

D I S P L A C E M E N T S

NODE	X-DISPLACEMENT	Y-DISPLACEMENT	Z-DISPLACEMENT
1	0.00000e+000	0.00000e+000	0.00000e+000
2	0.00000e+000	0.00000e+000	0.00000e+000
3	0.00000e+000	0.00000e+000	0.00000e+000
4	0.00000e+000	0.00000e+000	0.00000e+000
5	0.00000e+000	2.40000e+000	4.99600e-016
6	0.00000e+000	2.40000e+000	-5.37672e-016
7	-3.33067e-016	2.40000e+000	-2.70884e-016
8	-1.11022e-016	2.40000e+000	-1.01581e-016

图 3.4: 纯剪载荷位移场计算结果

3.3 Euler-Bernoulli 梁单元

3.3.1 单元构造

梁单元的单元构造很简单，由四个模态组成：轴向拉伸，沿轴扭转以及两个方向的弯曲。由于数值简单，不需要进行数值积分。具体构成如图 3.5 所示。

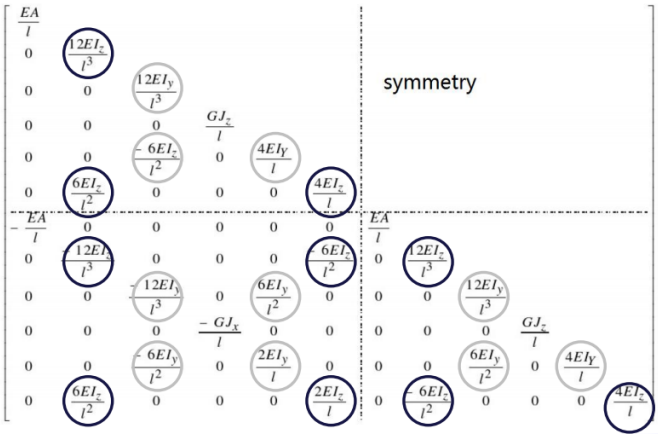


图 3.5: 梁单元刚度阵，不同模态用不同颜色圈出来

现规定沿轴方向为 x 轴，则对应的位移向量和载荷向量表示为:

$$d^e = \begin{pmatrix} u_{x1} & u_{y1} & u_{z1} & u_{x1} & u_{y1} & u_{z1} & u_{x2} & u_{y2} & u_{z2} & \theta_{x2} & \theta_{y2} & \theta_{y3} \end{pmatrix}^T$$

$$f^e = \begin{pmatrix} F_{x1} & F_{y1} & F_{z1} & M_{x1} & M_{y1} & M_{z1} & F_{x2} & F_{y2} & F_{z2} & M_{x2} & M_{y2} & M_{y3} \end{pmatrix}^T$$

其中 u 表示位移， θ 表示转角， F 表示剪力， M 表示弯矩，下标的 1、2 分别表示左右两端点。于是便得到了基本的单元方程:

$$K^e d^e = f^e$$

可以以这个形式加入总体刚度方程。另外需要注意的是，当局部的坐标系与整体坐标系不一致时，需要对单元刚度阵做一个对应的正交变换。

3.3.2 梁单元截面的输入格式

除了基本的杨氏模量 E ，泊松比 ν 之外，由于本工程项目中使用的是空心矩形截面梁，因此依次输入的参数是：矩形的长和宽（a, b），四个面上的壁厚（t_1, t_2, t_3, t_4），以及一组单位向量（n_1, n_2, n_3）来表征局部坐标系的 y' 轴在整体坐标系中的指向。

3.3.3 分片试验

对于梁进行的不规则划分如图 3.6 。

由于梁有四个解耦的模态，为了将其完全约束，在进行分片试验时需要对四个模态分别都加以约束。

载荷条件为：两端简支，两边施加等大反向的弯矩，整个梁为纯弯状态，且各处弯矩相等。位移场的精确解为:

$$w = 0.05x^2 - 0.05x, \theta = 0.1x - 0.05$$

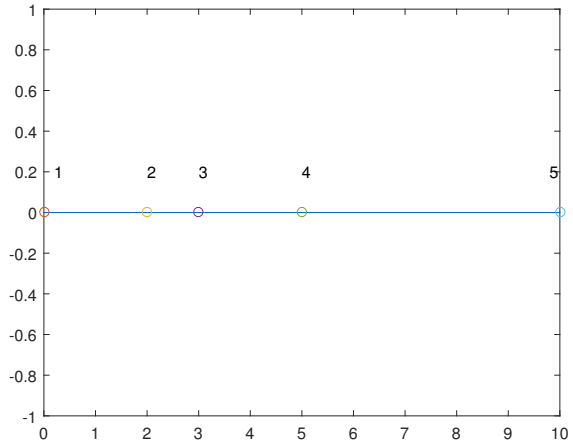


图 3.6: 分片试验的划分

分片试验结果如图 3.7 。

D I S P L A C E M E N T S

NODE	X-DISPLACEMENT	Y-DISPLACEMENT	Z-DISPLACEMENT			
1	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000	-5.00000e-001
2	0.00000e+000	-8.00000e-001	0.00000e+000	0.00000e+000	0.00000e+000	-3.00000e-001
3	0.00000e+000	-1.05000e+000	0.00000e+000	0.00000e+000	0.00000e+000	-2.00000e-001
4	0.00000e+000	-1.25000e+000	0.00000e+000	0.00000e+000	0.00000e+000	3.95517e-016
5	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000	5.00000e-001

图 3.7: 分片试验的结果

节点位移和转角精确，通过分片试验。

3.3.4 收敛率分析

对于长度为 8 的梁，依次将其等分为 2,4,8 个单元，如图 3.8 。

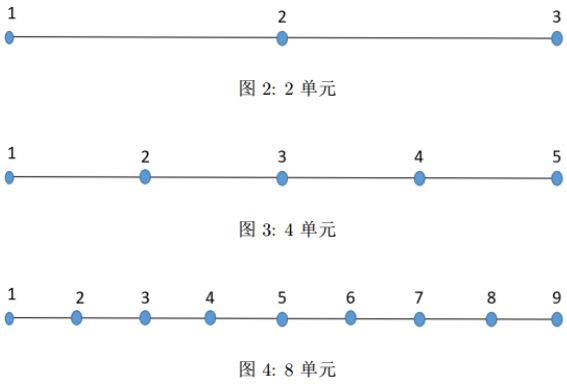


图 3.8: 收敛率分析的细分

梁单元能精确重构三次位移场，为了计算误差，通过一端固支，并合理加载均布载荷、另一端的集中力和弯矩，构造出精确解为:

$$w = 0.0001x^4$$

考虑到单元的刚度为常数，故直接取误差范数的表达式为:

$$\|e\| = \int_0^8 \left(\frac{du}{dx} - \frac{du^h}{dx} \right)^2 dx$$

计算得到的结果如图 3.9 。

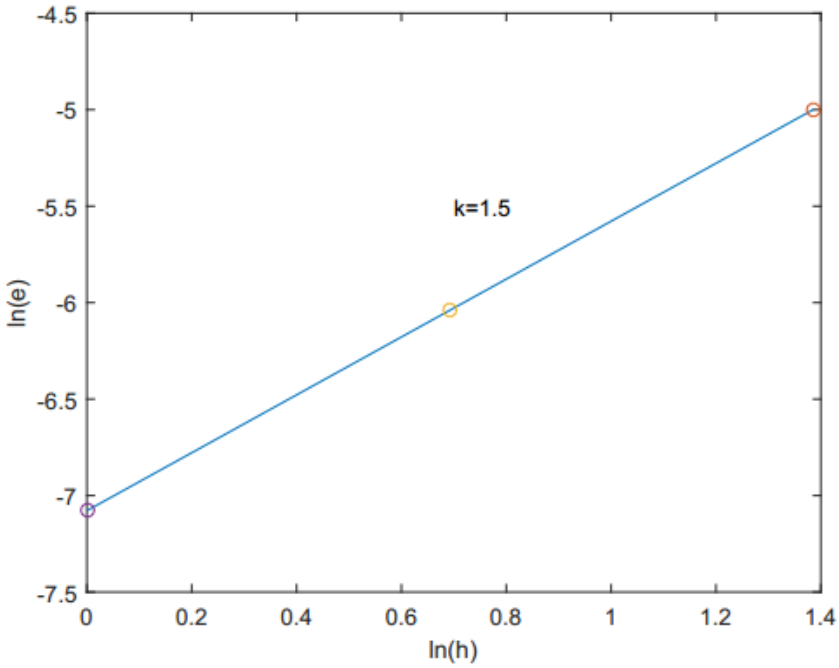


图 3.9: 收敛率分析的结果

得到收敛率为 1.5 阶。

3.4 平板壳单元

我们使用的是平板壳单元,即 4Q 单元与 plate 单元的叠加,详细的说明可见 plate 单元部分.

我们对壳单元构造的分片实验算例采取板单元与 4Q 单元算例的叠加,二者应该互不影响且在各自的自由度上得到正确的解.弯曲变形部分仍然采用板单元的分片算例(见图4.17:我们在 $[-1,1]^2$ 的正方形板构造这样的位移场 $w = x^2 - \nu y^2$, 得到一个纯弯场 $M_{11} = -2D(1-\nu^2)$. 利用 $x = 0; x = 0.3; y = -0.2; y = 0$ 四条直线将正方形板分割成九个分片,对应的将法线沿 x 轴方向边界上的 y 向弯矩分配到每个点上.对于面内载荷我们加一个沿 y 方向面密度为 2 的载荷得到一个单向拉伸场,这样对应的位移场为 $u = -(x+1)/60, v = (y+1)/12$. 具体输入文件见 shell_patch_test, 图3.10显示了壳单元的位移,其中最后三行的输出是 x 方向旋转, y 方向旋转以及 z 方向旋转;可见位移基本准确,误差在浮点数范围内,认为壳单元通过测试.

NODE	X-DISPLACEMENT	Y-DISPLACEMENT	Z-DISPLACEMENT			
1	0.00000e+000	0.00000e+000	8.00000e-001	4.00000e-001	2.00000e+000	0.00000e+000
2	-1.66667e-002	3.25261e-016	-2.00000e-001	4.00000e-001	5.33532e-014	0.00000e+000
3	-2.16667e-002	2.86229e-016	-1.10000e-001	4.00000e-001	-6.00000e-001	0.00000e+000
4	-3.33333e-002	0.00000e+000	8.00000e-001	4.00000e-001	-2.00000e+000	0.00000e+000
5	-1.90820e-016	6.66667e-002	9.92000e-001	8.00000e-002	2.00000e+000	0.00000e+000
6	-1.66667e-002	6.66667e-002	-8.00000e-003	8.00000e-002	3.06699e-014	0.00000e+000
7	-2.16667e-002	6.66667e-002	8.20000e-002	8.00000e-002	-6.00000e-001	0.00000e+000
8	-3.33333e-002	6.66667e-002	9.92000e-001	8.00000e-002	-2.00000e+000	0.00000e+000
9	-1.87350e-016	8.33333e-002	1.00000e+000	-3.41949e-014	2.00000e+000	0.00000e+000
10	-1.66667e-002	8.33333e-002	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000
11	-2.16667e-002	8.33333e-002	9.00000e-002	-2.52836e-015	-6.00000e-001	0.00000e+000
12	-3.33333e-002	8.33333e-002	1.00000e+000	-8.27116e-015	-2.00000e+000	0.00000e+000
13	-1.80411e-016	1.66667e-001	8.00000e-001	-4.00000e-001	2.00000e+000	0.00000e+000
14	-1.66667e-002	1.66667e-001	-2.00000e-001	-4.00000e-001	4.92991e-014	0.00000e+000
15	-2.16667e-002	1.66667e-001	-1.10000e-001	-4.00000e-001	-6.00000e-001	0.00000e+000
16	-3.33333e-002	1.66667e-001	8.00000e-001	-4.00000e-001	-2.00000e+000	0.00000e+000

图 3.10: 壳单元输出位移

Chapter 4

其他单元

这一章将在上一章的基础上介绍新增的其他单元，包括可用于 3 维背景空间的平面单元（包括 4Q, 3T, 9Q，详见 4.1, 4.2, 4.3 节）、为适应结构的尺度效应而发展出的构造单元（包括 Timoshenko 梁单元、薄板单元、截锥壳单元，详见 4.4, 4.5, 4.6 节）以及可用于解决更广泛的弹性力学问题的扩展单元（包括无限单元和过渡单元，详见 4.7, 4.8 节）。

4.1 4Q 平面单元

4.1.1 单元构造

对于 4Q 单元，首先要将空间降维，在 4Q 单元所在平面上建立局部坐标系。计算法向量 $\hat{n}$：

$$\hat{n} = \hat{p}_{21} \times \hat{p}_{31}$$

设 $\hat{i}$ 向量平行于 p_{21}：

$$\hat{i} = \hat{p_{21}}$$

根据 $\hat{i}$ 和 $\hat{n}$ 在右手系中生成 j： $\hat{j} = \hat{n} \times \hat{i}$.

于是根据公式

$$\begin{aligned} K^{e\prime} &= \int B^T D B d\Omega \\ &= \sum W_i W_j \left(B^T D B\right) |J| \end{aligned}$$

$$J^e = GN^{4Q} \begin{bmatrix} x^e & y^e \end{bmatrix}$$

$$[B^e] = \left(J^e\right)^{-1} GN^{4Q}$$

可以先生成 GN^{4Q} 再生成 J^e 和 $[B^e]$. 从 $[B^e]$ 生成 B：

$$B = \begin{pmatrix} [B]_{11} & 0 & [B]_{12} & 0 & [B]_{13} & 0 & [B]_{14} & 0 \\ 0 & [B]_{21} & 0 & [B]_{22} & 0 & [B]_{23} & 0 & [B]_{24} \\ [B]_{21} & [B]_{11} & [B]_{22} & [B]_{12} & [B]_{23} & [B]_{32} & [B]_{24} & [B]_{14} \end{pmatrix}$$

计算 $K_{ij}^{e\prime}$:

$$\begin{aligned} K_{ij}^{e\prime} &= B_{ik}^T D_{kl} B_{lj} \\ &= B_{ki} B_{lj} D_{kl} \end{aligned}$$

为减少计算量，展开 D_{kl} 并只考虑 $(k,l)=(1,1),(1,2),(2,1),(2,2),(3,3)$. 进一步合并同类项可以进一步减少计算量。

计算出局部坐标系下的 K^e 后，将其转回全局坐标系。

$$K^e = R^T K^{e\prime} R$$

其中

$$R = \begin{pmatrix} i_0 & i_1 & i_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ j_0 & j_1 & j_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & i_0 & i_1 & i_2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & j_0 & j_1 & j_2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & i_0 & i_1 & i_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & j_0 & j_1 & j_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & i_0 & i_1 & i_2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & j_0 & j_1 & j_2 \end{pmatrix}$$

即

$$\begin{aligned} K^e_{ij} &= R^T_{ik} K^e_{kl}{}' R_{lj} \\ &= R_{ki} R_{lj} K^e_{kl}{}' \end{aligned}$$

4.1.2 后处理部分

计算应力公式：

$$\begin{aligned} \sigma^e &= D^e \varepsilon^e \\ &= D^e B^e d^e \\ \sigma^e_i &= (DB)^e_{ij} d^e_j \quad (j = 1:6) \\ &= D^e_{ik} B^e_{kj} d^e_j \end{aligned}$$

其中

$$d^e = \begin{pmatrix} u^e_{1x} & u^e_{1y} & u^e_{2x} & u^e_{2y} & u^e_{3x} & u^e_{3y} & u^e_{4x} & u^e_{4y} \end{pmatrix}^T = Rd$$

4.1.3 收敛率分析

宏 `__TEST__` 被定义时，会在计算应力时额外输出高斯点的位移和积分的系数。这个宏可以在 cmake 中通过 `STAP++_TEST` 选项开启。

根据

$$e^2 = \int_\Omega (u-u^e)^2 \mathrm{d}\Omega = \sum W_i W_j (u-u^e)^2 |J|$$

就可以计算误差的积分。上式中，$W_i W_j |Je|$ 合并为 weight，即

$$e^2 = \sum_{i=1}^{NOE} w_i \sum_{j=1}^{3} \left(u_{ij} - u^e_{ij}\right)^2$$

通过一个简单的 python 文件，根据高斯点的坐标算出准确位移，我们就可以计算出总的误差 e。

考虑如图 4.1 这样一个简单的情况。体力随 X 线性变化，Y 方向限制自由度。体力 $\boldsymbol{f} = b\boldsymbol{x}$（均匀梯度重力场）

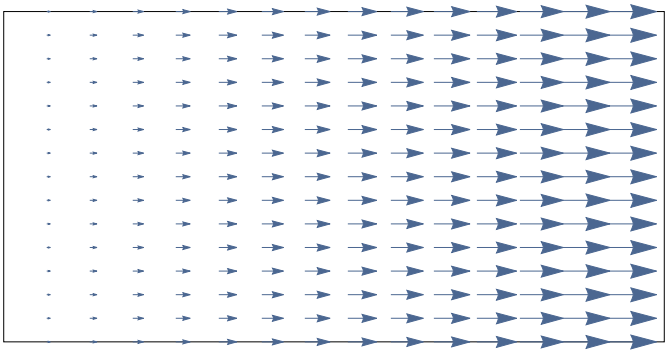


图 4.1: 4Q 施加载荷

容易计算得到应变满足

$$\begin{cases} \varepsilon_x = \frac{1-\nu^2}{E} b \left(2 - \frac{x^2}{2}\right) \\ \varepsilon_y = 0 \\ \gamma_{xy} = 0 \end{cases}$$

位移满足

$$\begin{cases} u_x = \frac{(1-\nu^2)b}{6E} x \left(12 - x^2\right) \\ u_y = 0 \end{cases}$$

生成输入文件的脚本见 `data/4Q/genDat.py`。

处理输出文件的脚本见 `data/4Q/calcErr.py`。

网格划分为每单位长度 $1 \sim 32, 64, 128, 256$ 个单元，计算结果如图 4.2 所示。

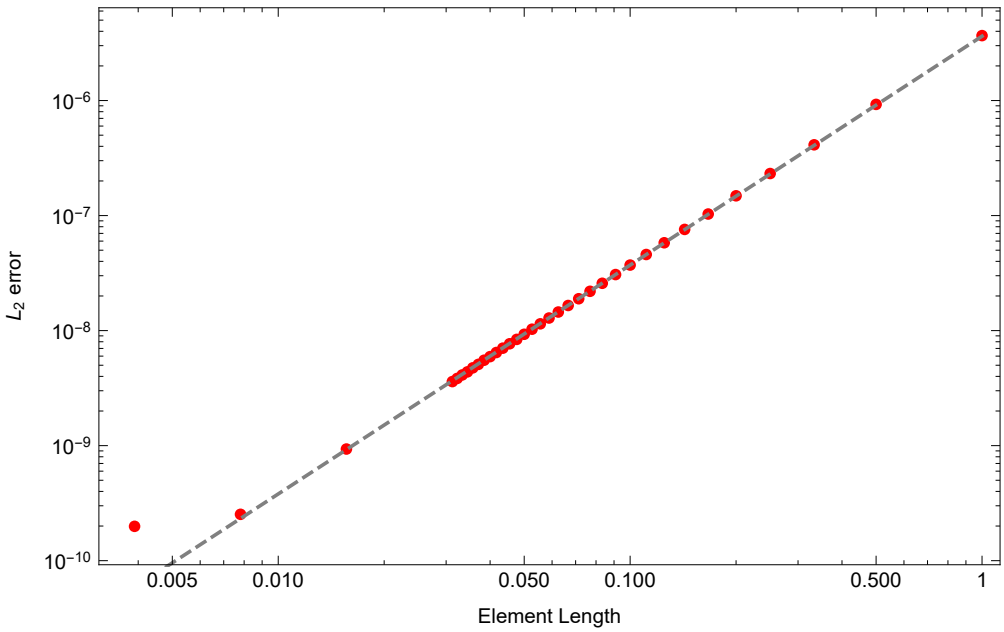


图 4.2: 4Q 收敛率分析

从图中可以看到，$n = 256$ 的点处偏离了曲线。原因可能是受限于输出文件的精度限制（输出文件仅有 6 位有效数字），有浪费现象。

去除明显偏差的点（$n = 256$），图像拟合得到表达式

$$\log e = -12.5277 + 1.98972 \log l$$

即，4Q 单元位移二阶收敛。符合理论预期。

4.2 3T 平面单元

4.2.1 组装刚度阵

类似 4Q 单元，进行一次 3d-2d-3d 的转换。
n, i, j 同 4Q 一样生成，但对于 3T 单元，设置 $i = \bar{p}_{21}$ 可以简化计算量。
计算刚度阵有

$$K^e = A^e B^{eT} D B^e$$

计算应力有

$$\sigma = D\varepsilon = DB^e d^e$$

由于 D, B^e, d^e 都是常矩阵，故 3T 单元中为常应力场。

4.2.2 分片试验

分片实验输入文件见 `data/3T/patch.dat` ，计算结果如图 4.3 所示。
图中，节点为精确解，实体为计算解。

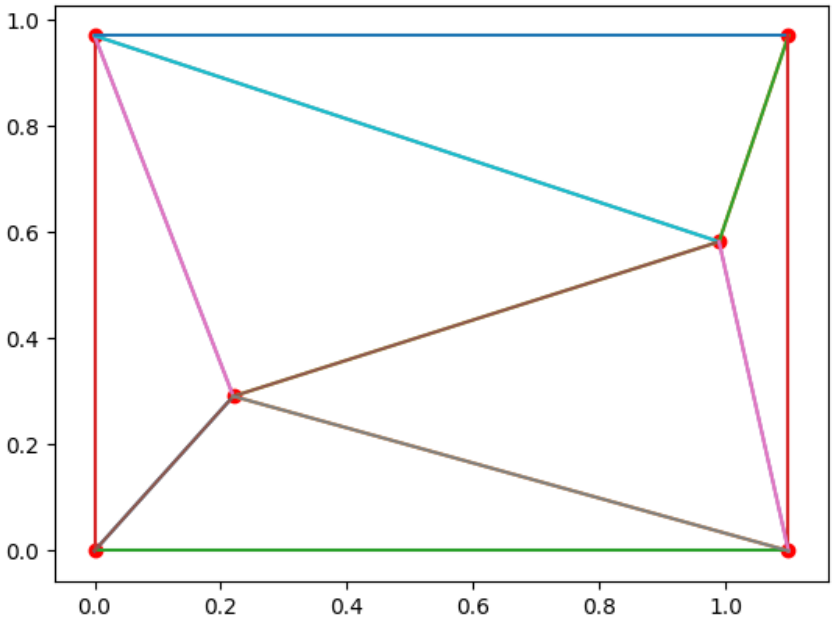


图 4.3: 分片试验

3T 单元在 10^{-4} 误差下通过分片实验。

4.2.3 收敛率计算

宏 `__TEST__` 被定义时，会在计算应力时额外输出高斯点的位移和积分的系数。(采用三点高斯积分)。

这个宏可以在 cmake 中通过 STAP++_TEST 选项开启。

根据

$$e^2 = \int_\Omega (u-u^e)^2 \mathrm{d}\Omega = \sum W_i (u-u^e)^2 |J|$$

就可以计算误差的积分。上式中，$W_i|Je|$ 合并为 weight，即

$$e^2 = \sum_{i=1}^{NOE} w_i \sum_{j=1}^{3} \left(u_{ij} - u_{ij}^e\right)^2$$

通过一个简单的 python 文件，根据高斯点的坐标算出准确位移，我们就可以计算出总的误差 e。

考虑如图 4.4 这样一个简单的情况。体力随X线性变化，Y 方向限制自由度。体力 $\boldsymbol{f} = b\boldsymbol{x}$（均匀梯度重力场）

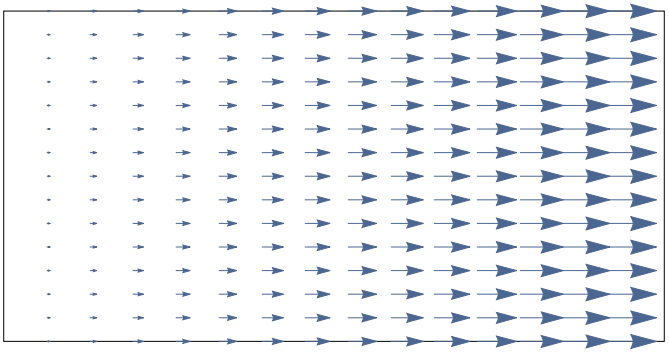


图 4.4: 施加载荷

容易计算得到应变满足

$$\begin{cases} \varepsilon_x = \frac{1-\nu^2}{E} b \left(2 - \frac{x^2}{2}\right) \\ \varepsilon_y = 0 \\ \gamma_{xy} = 0 \end{cases}$$

位移满足

$$\begin{cases} u_x = \frac{\left(1-\nu^2\right)b}{6E} x \left(12 - x^2\right) \\ u_y = 0 \end{cases}$$

生成任务文件的脚本见 data/3T/genDat.py。

处理输出文件的脚本见 data/3T/calcErr.py。

测试文件见 data/3T/run-rate.py。

网格划分为每单位长度 $1 \sim 32, 64, 128$ 个单元，计算结果如图 4.5 所示。

即，3T 单元位移二阶收敛。

4.3 9Q 平面单元

4.3.1 程序设计

9Q 单元的实现中利用了一个我们自己编写的简单矩阵库（src/h/matrix.h）。

计算刚度阵使用

$$K = \int B^T D B |Je| \mathrm{d}\xi \mathrm{d}\eta$$

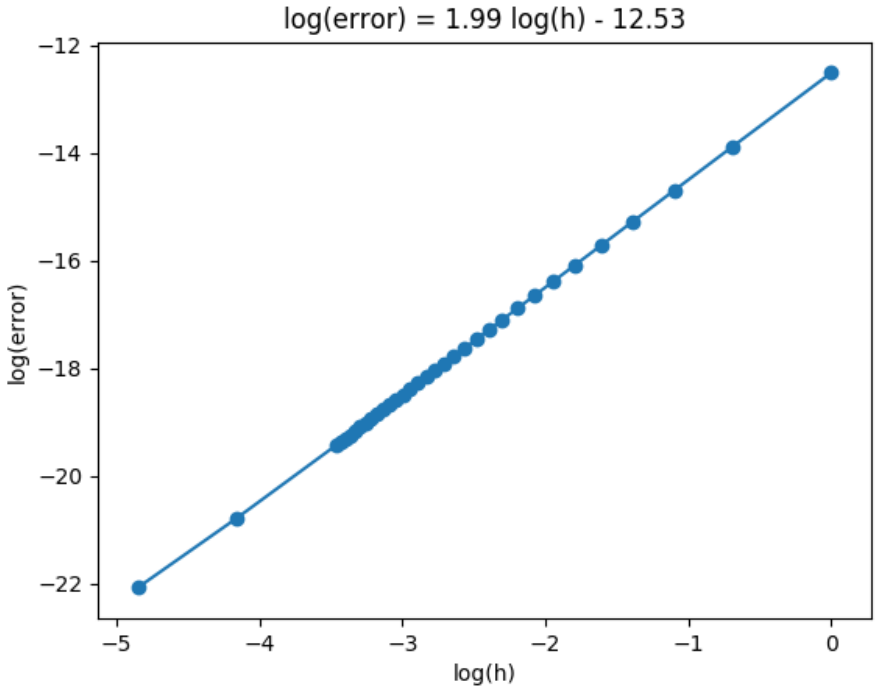


图 4.5: 3T收敛率分析

因为两点积分会出现零能模态，因此采用三点高斯积分，

$$x_i = \pm\sqrt{\frac{5}{6}}, 0$$

$$w_i = \frac{5}{9}, \frac{8}{9}$$

计算荷载使用

$$f_i = \int N_i b \mathrm{d}\eta$$

可知，对于边界上三点受力，比例为 $1:4:1$。

4.3.2 patch test

基于教案上的测试算例（图 4.6）进行了改动，将左下角设为固支。

实际计算在 10^{-4} 误差下通过分片测试。

4.4 Timoshenko 梁单元

Timoshenko 梁与 Euler-Bernoulli 梁是两种经典的梁模型，二者的共同之处在于均采用了平截面假设，即变形前位于中性轴法线上的点在变形后仍然共线。二者的关键性区别在于，后者的模型中采用了直法线假设，这直接导致了剪应变为零，因此只适用于剪切变形可以忽略的细长梁；而前者对此作了修正，采用了剪应变沿截面均匀分布的假设，从而可以解决包括短粗梁在内的更广泛的工程问题。需要指出的是，事实上梁的剪应变沿截面并非均匀分布的，因此在实际应用中还需进行系数修正。

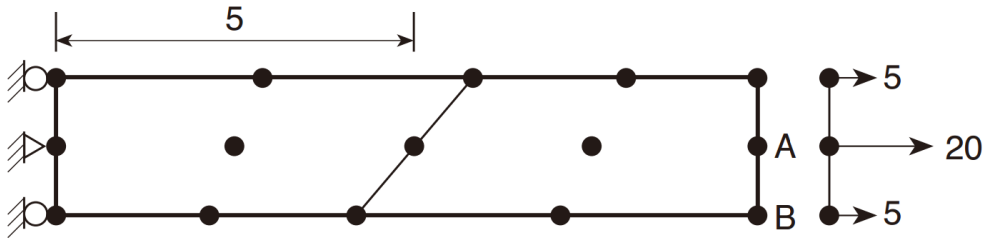


图 4.6: 9Q patch test

下面，我们先从变分原理出发导出 Timoshenko 梁的控制方程，进而以此为基础构造分片试验所需要的精确解；接着，给出两种 Timoshenko 梁单元的构造思路及过程；最后，通过分片试验与收敛率分析两种途径对各种 Timoshenko 梁单元进行收敛性分析。

4.4.1 变分原理与精确解构造

变分原理与强形式

为简便起见，仅以挠度在 y 方向的情形进行推导。写出系统的势能泛函表达式

$$\Pi_p = \int_0^l \frac{1}{2} EI_z \kappa_z^2 dx + \int_0^l \frac{1}{2} \frac{GA}{k} \gamma_{xy}^2 dx - \int_0^l \bar{q} v dx - \overline{N}_{yj} v_j - \overline{M}_{zk} \theta_{zk}.$$

其中，

$$\kappa_z \equiv \frac{d\theta_z}{dx},$$
$$\gamma_{xy} \equiv \frac{dv}{dx} - \theta_z.$$

利用最小势能原理 $\delta\Pi_p = 0$ 可得到如下控制方程

$$EI_z \frac{d\kappa_z}{dx} + \frac{GA}{k}\gamma_{xy} + \bar{M}_{zk}\delta(x - x_k) = 0, \tag{4.4.1}$$
$$\frac{GA}{k}\frac{d\gamma_{xy}}{dx} + \bar{q} + \bar{N}_{yj}\delta(x - x_j) = 0. \tag{4.4.2}$$

这里已经假定梁的截面性质沿梁均匀分布。这样就得到了 Timoshenko 梁控制方程的强形式。

与上面等价的另一种推导强形式的方法是，利用截面局部坐标系中内力分量之间的关系

$$\frac{dM_z}{dx} + F_y = 0, \tag{4.4.3}$$
$$\frac{dF_y}{dx} + \overline{q} = 0. \tag{4.4.4}$$

这里没有考虑集中力载荷与集中力矩载荷的作用。其中，

$$M_z \equiv EI_z \kappa_z = EI_z \frac{d\theta_z}{dx},$$
$$F_y \equiv \frac{GA}{k}\gamma_{xy} = \frac{GA}{k}\left(\frac{dv}{dx} - \theta_z\right).$$

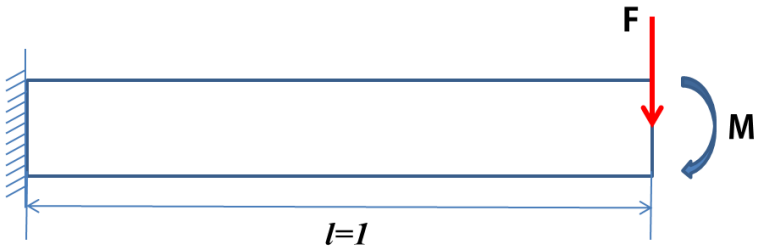


图 4.7: Example

悬臂梁：用于分片试验的精确解

我们考虑一端固定、另一端受集中载荷的矩形截面悬臂梁的情形，如图 4.7 所示。材料/几何尺寸以及相应的材料/截面性质依次为

- Young's Modulus $E = 1000$, Poisson's ratio $\nu = 0.25$, $k = \frac{6}{5}$;
- $a := Height_y = 0.3$, $b := Height_z = 0.6 \rightarrow Area = 0.18$, $I_{yy} = 0.0054$, $I_{zz} = 0.00135$;
- $Theta_{y1} = 0$, $Theta_{y2} = 0$, $Theta_{y3} = -1$, $l = 1$.

由于 Timoshenko 梁模型的控制方程为线性方程，因此只需分别求出两种工况下的位移精确解，再进行线性叠加即可。下面给出了两种工况下的精确解表达式，其中的坐标均在单元局部坐标系中。

1. 集中弯矩载荷 $M_z = 1$：

$$
\begin{aligned}
v &= \frac{Mx^2}{2EI_z} = \frac{10}{27}x^2 = 0.\dot{3}7\dot{0}x^2; \\
\theta_z &= \frac{Mx}{EI_z} = \frac{20}{27}x = 0.\dot{7}4\dot{0}x.
\end{aligned}
$$

2. 集中力载荷 $F_y = 1$：

$$
\begin{aligned}
v &= -\frac{F}{6EI_z}x^3 + \frac{Fl}{2EI_z}x^2 + \frac{k}{GA}x \\
&= -\frac{10}{81}x^3 + \frac{10}{27}x^2 + \frac{1}{60}x \\
&= -0.\dot{1}23456790\dot{x}^3 + 0.\dot{3}7\dot{0}x^2 + 0.01\dot{6}x; \\
\theta_z &= -\frac{F}{2EI_z}x^2 + \frac{Fl}{EI_z}x \\
&= -\frac{10}{27}x^2 + \frac{20}{27}x \\
&= -0.\dot{3}7\dot{0}x^2 + 0.\dot{7}4\dot{0}x.
\end{aligned}
$$

4.4.2 有限元离散：两种不同的格式构造方案

坐标变换

我们首先约定，带“∼”为单元局部坐标系中的量，否则为单元背景坐标系中的量。上述坐标系的定义如下

单元局部坐标系 以节点 1 为原点 $\tilde{O}$，以 $1 \to 2$ 为单元局部坐标系的 $\tilde{x}$ 轴，$\tilde{y}, \tilde{z}$ 轴由惯性主矩的方向确定，且要求 $\tilde{O}\tilde{x}\tilde{y}\tilde{z}$ 成右手系。

单元背景坐标系 以节点 1 为原点 $\tilde{O}$，以背景空间的 x, y, z 轴方向建立单元背景坐标系 $\tilde{O}xyz$。

对于无限小转动，角位移可以近似利用向量进行描述，因此可直接利用坐标转换矩阵

$$\boldsymbol{L} = diag\{\boldsymbol{Q}, \boldsymbol{Q}, \boldsymbol{Q}, \boldsymbol{Q}\}. \tag{4.4.5}$$

其中，Q 即为坐标变换矩阵，其元素定义为 $Q_{ij} = \boldsymbol{e}_i \cdot \tilde{\boldsymbol{e}}_j$。由此可得向量与二阶张量的变换公式分别为

$$\boldsymbol{a} = \boldsymbol{Q} \cdot \tilde{\boldsymbol{a}}; \tag{4.4.6}$$

$$\boldsymbol{K} = \boldsymbol{Q} \cdot \tilde{\boldsymbol{K}} \cdot \boldsymbol{Q}^T. \tag{4.4.7}$$

本节以下所有讨论均在单元局部坐标系中进行。

位移转角一致插值 (Euler-Bernoulli 梁的直接修正)

将总位移分解为弯曲位移项与纯剪位移项

$$\begin{aligned}
\boldsymbol{a} &= \boldsymbol{a}_b + \boldsymbol{a}_s. \\
\boldsymbol{a}_i &= (u_i,\ v_i,\ w_i,\ \varphi_i,\ \theta_{yi},\ \theta_{zi})^T; \\
\boldsymbol{a}_b &= (\boldsymbol{a}_{b1},\ \boldsymbol{a}_{b2})^T; \\
\boldsymbol{a}_s &= (\boldsymbol{a}_{s1},\ \boldsymbol{a}_{s2})^T; \\
\boldsymbol{a}_{bi} &= (u_i,\ v_{bi},\ w_{bi},\ \varphi_i,\ \theta_{yi},\ \theta_{zi})^T; \\
\boldsymbol{a}_{si} &= (0,\ v_{si},\ w_{si},\ 0,\ 0,\ 0)^T.
\end{aligned}$$

其中，

$$\begin{aligned}
\frac{dv_{bi}}{dx} &= \theta_{zi}; \\
\frac{dw_{bi}}{dx} &= -\theta_{yi}.
\end{aligned}$$

下面以 y 方向的刚度阵为例进行推导。对弯曲位移项进行两点 Hermite 插值，对剪切位移项进行两点线性插值

$$\begin{aligned}
v_b &= N_1 v_{b1} + N_2 \theta_1 + N_3 v_{b2} + N_4 \theta_4, \\
v_s &= N_5 v_{s1} + N_6 v_{s2}.
\end{aligned}$$

其中，（$0 \le \xi \le 1$）

$$\begin{aligned}
N_1 &= 1 - 3\xi^2 + 2\xi^3 & N_2 &= (\xi - 2\xi^2 + \xi^3) \\
N_3 &= 3\xi^2 - 2\xi^3 & N_4 &= (\xi^3 - \xi^2)l \\
N_5 &= 1 - \xi & N_6 &= \xi.
\end{aligned}$$

将上述用形函数表达的位移项代入能量泛函中进行变分，可以得到

$$\boldsymbol{K}_b \boldsymbol{a}_b = \boldsymbol{P}_b, \boldsymbol{K}_s \boldsymbol{a}_s = \boldsymbol{P}_s. \tag{4.4.8}$$

由于单元内部成立平衡方程 $Q = \frac{dM}{dx}$，因此弯曲位移项与剪切位移项均可以表达为总位移与转角的线性组合

$$\begin{pmatrix} v_{b2} - v_{b1} \\ v_{s2} - v_{s1} \end{pmatrix} = \begin{pmatrix} \dfrac{1}{1+b_z} & \dfrac{lb_z}{2(1+b_z)} \\ \dfrac{b_z}{1+b_z} & -\dfrac{lb_z}{2(1+b_z)} \end{pmatrix} \begin{pmatrix} v_2 - v_1 \\ \theta_{z1} + \theta_{z2} \end{pmatrix}. \tag{4.4.9}$$

其中，$b_z := \frac{12EI_z k}{GAl^2}$ 为剪切附加项，对于矩形截面梁有 $k = \frac{6}{5}$。注意到公式 (4.4.9) 的特点，将方程 (4.4.8) 的式 1 与式 5 求和，式 3 与式 6 求和，将公式 (4.4.9) 代入方程 (4.4.8) 中，可以得到最终的刚度方程

$$\boldsymbol{K}\boldsymbol{a} = \boldsymbol{P}.$$

其中，

$$\boldsymbol{K} = \frac{EI_z}{(1+b_z)l^3} \begin{pmatrix} 12 & -12 & 6l & 6l \\ & 12 & -6l & -6l \\ & & (4+b_z)l^2 & (2-b_z)l^2 \\ [symmetry] & & & (4+b_z)l^2 \end{pmatrix} \tag{4.4.10}$$

$$\boldsymbol{a} = (v_1,\ v_2,\ \theta_{z1},\ \theta_{z2})^T. \tag{4.4.11}$$

一般地，考虑所有的位移分量

$$\begin{aligned} \boldsymbol{a} &= \boldsymbol{a}_b + \boldsymbol{a}_s. \\ \boldsymbol{a}_i &= (u_i,\ v_i,\ w_i,\ \varphi_i,\ \theta_{yi},\ \theta_{zi})^T. \\ \boldsymbol{P} &= (N_{xi},\ N_{yi},\ N_{zi},\ M_{xi},\ M_{yi},\ M_{zi})^T. \end{aligned}$$

则在单元局部坐标系内的刚度阵可写为

$$\boldsymbol{K} = \begin{pmatrix} \text{Tens} & 0 & 0 & 0 & 0 & 0 & -\text{Tens} & 0 & 0 & 0 & 0 & 0 \\ & \text{Bendz1} & 0 & 0 & 0 & \text{Bendz2} & 0 & -\text{Bendz1} & 0 & 0 & 0 & \text{Bendz2} \\ & & \text{Bendy1} & 0 & -\text{Bendy2} & 0 & 0 & 0 & -\text{Bendy1} & 0 & -\text{Bendy2} & 0 \\ & & & \text{Tors} & 0 & 0 & 0 & 0 & 0 & -\text{Tors} & 0 & 0 \\ & & & & \text{Bendy3} & 0 & 0 & 0 & \text{Bendy2} & 0 & \text{Bendy4} & 0 \\ & & & & & \text{Bendz3} & 0 & -\text{Bendz2} & 0 & 0 & 0 & \text{Bendz4} \\ & & & & & & \text{Tens} & 0 & 0 & 0 & 0 & 0 \\ & & & & & & & \text{Bendz1} & 0 & 0 & 0 & -\text{Bendz2} \\ & & & & & & & & \text{Bendy1} & 0 & \text{Bendy2} & 0 \\ & & [symmetry] & & & & & & & \text{Tors} & 0 & 0 \\ & & & & & & & & & & \text{Bendy3} & 0 \\ & & & & & & & & & & & \text{Bendz3} \end{pmatrix}. \tag{4.4.12}$$

其中，

- $\text{Tens} = \frac{EA}{l}$, $\text{Tors} = \frac{GJ}{l}$;
- $\text{Bendy1} = \frac{12EI_y}{(1+b_y)l^3}$, $\text{Bendy2} = \frac{6EI_y}{(1+b_y)l^2}$, $\text{Bendy3} = \frac{(4+b_y)EI_y}{(1+b_y)l}$, $\text{Bendy4} = \frac{(2-b_y)EI_y}{(1+b_y)l}$;
- $\text{Bendz1} = \frac{12EI_z}{(1+b_z)l^3}$, $\text{Bendz2} = \frac{6EI_z}{(1+b_z)l^2}$, $\text{Bendz3} = \frac{(4+b_z)EI_z}{(1+b_z)l}$, $\text{Bendz4} = \frac{(2-b_z)EI_z}{(1+b_z)l}$.

输出的单元内力分量包括节点的轴力、剪力和扭矩值。具体计算公式如下

$$
\begin{aligned}
F_{x2} &= -F_{x1} = EA\varepsilon_{xx} = \frac{EA(u_2-u_1)}{l},\\
F_{y2} &= -F_{y1} = \frac{GA}{k}\gamma_{xy} = \frac{GA}{kl}\frac{b_z}{1+b_z}\left((v_2-v_1)-\frac{1}{2}(\theta_{z1}+\theta_{z2})l\right),\\
F_{z2} &= -F_{z1} = \frac{GA}{k}\gamma_{xz} = \frac{GA}{kl}\frac{b_y}{1+b_y}\left((w_2-w_1)+\frac{1}{2}(\theta_{y1}+\theta_{y2})l\right),\\
M_{x2} &= -M_{x1} = GJ\alpha = \frac{GJ(\varphi_2-\varphi_1)}{l},\\
M_{y1} &= EI_y\frac{d\theta_y}{dx} = -\frac{EI_y}{l^2}\left(6\frac{w_2-w_1}{b_y+1}-3\frac{b_y}{b_y+1}(\theta_{y1}+\theta_{y2})l+(4\theta_{y1}+2\theta_{y2})l\right),\\
M_{y2} &= EI_y\frac{d\theta_y}{dx} = -\frac{EI_y}{l^2}\left(-6\frac{w_2-w_1}{b_y+1}+3\frac{b_y}{b_y+1}(\theta_{y1}+\theta_{y2})l-(2\theta_{y1}+4\theta_{y2})l\right),\\
M_{z1} &= EI_z\frac{d\theta_z}{dx} = \frac{EI_z}{l^2}\left(6\frac{v_2-v_1}{b_z+1}+3\frac{b_z}{b_z+1}(\theta_{z1}+\theta_{z2})l-(4\theta_{z1}+2\theta_{z2})l\right),\\
M_{z2} &= EI_z\frac{d\theta_z}{dx} = \frac{EI_z}{l^2}\left(-6\frac{v_2-v_1}{b_z+1}-3\frac{b_z}{b_z+1}(\theta_{z1}+\theta_{z2})l+(2\theta_{z1}+4\theta_{z2})l\right).
\end{aligned}
$$

输出单元中性面上的 Gauss 点应力，用于恢复并输出节点应力。具体包括单元的轴向正应力 σ_{xx}、弯曲剪应力 σ_{xy} 与 σ_{xz}，其计算公式为

$$
\begin{aligned}
\sigma_{xx} &= E\varepsilon_{xx} = \frac{E(u_2-u_1)}{l},\\
\sigma_{xy} &= \frac{G}{k}\gamma_{xy}\\
&= \frac{G}{kl}\frac{b_z}{1+b_z}\left((v_2-v_1)-\frac{1}{2}(\theta_{z1}+\theta_{z2})l\right),\\
\sigma_{xz} &= \frac{G}{k}\gamma_{xz}\\
&= \frac{G}{kl}\frac{b_y}{1+b_y}\left((w_2-w_1)+\frac{1}{2}(\theta_{y1}+\theta_{y2})l\right).
\end{aligned}
$$

位移转角独立同阶插值

位移转角独立进行两点线性插值，以 v 方向的刚度阵为例，其单元刚度阵为

$$
\boldsymbol{K} = \frac{EI_z}{l}\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&1&-1\\0&0&-1&1\end{pmatrix} + \frac{GA}{4kl}\begin{pmatrix}4&-4&2l&2l\\&4&-2l&-2l\\&&l^2&l^2\\[sym]&&&l^2\end{pmatrix}. \tag{4.4.13}
$$

类似地，考虑所有的位移分量时，单元局部坐标系内的刚度阵可写为与式 (4.4.12) 相同的形式，只是其中的变量有所不同。其中，

- Tens $=\frac{EA}{l}$, Tors $=\frac{GJ}{l}$;
- Bendy1 $=\frac{GA}{kl}$, Bendy2 $=\frac{GA}{2k}$, Bendy3 $=\frac{GAl}{4k}\,(or\ \frac{GAl}{3k})+\frac{EI_y}{l}$, Bendy4 $=\frac{GAl}{4k}\,(or\ \frac{GAl}{6k})-\frac{EI_y}{l}$;
- Bendz1 $=\frac{GA}{kl}$, Bendz2 $=\frac{GA}{2k}$, Bendz3 $=\frac{GAl}{4k}\,(or\ \frac{GAl}{3k})+\frac{EI_z}{l}$, Bendz4 $=\frac{GAl}{4k}\,(or\ \frac{GAl}{6k})-\frac{EI_z}{l}$.

上述式子中，括号外的项是对转角进行选择性减缩积分（ Selective Reduced Integration，简称 SRINT）的结果，括号内为精确积分（ Accurate Integration，简称 ACINT）的结果。

输出的单元内力分量包括节点的轴力、剪力和扭矩值。具体计算公式如下

$$
\begin{aligned}
F_{x2} &= -F_{x1} = EA\varepsilon_{xx} = \frac{EA(u_2-u_1)}{l},\\
F_{y2} &= -F_{y1} = \frac{GA}{k}\gamma_{xy} = \frac{GA}{kl}\left((v_2-v_1)-\frac{(\theta_{z1}+\theta_{z2})l}{2}\right),\\
F_{z2} &= -F_{z1} = \frac{GA}{k}\gamma_{xz} = \frac{GA}{kl}\left((w_2-w_1)-\frac{(\theta_{y1}+\theta_{y2})l}{2}\right),\\
M_{x2} &= -M_{x1} = GJ\alpha = \frac{GJ(\varphi_2-\varphi_1)}{l},\\
M_{y2} &= EI_y\frac{d\tilde{\theta}_y}{dx} = \frac{EI_y}{l}(\theta_{y2}-\theta_{y1})+\frac{1}{2}F_{z2}l,\\
M_{y1} &= EI_y\frac{d\tilde{\theta}_y}{dx} = \frac{EI_y}{l}(\theta_{y2}-\theta_{y1})-\frac{1}{2}F_{z2}l,\\
M_{z2} &= EI_z\frac{d\tilde{\theta}_z}{dx} = \frac{EI_z}{l}(\theta_{z2}-\theta_{z1})-\frac{1}{2}F_{y2}l,\\
M_{z1} &= EI_z\frac{d\tilde{\theta}_z}{dx} = \frac{EI_z}{l}(\theta_{z2}-\theta_{z1})+\frac{1}{2}F_{y2}l.
\end{aligned}
$$

需要说明的是，由于转角插值函数的阶数限制，有可能出现单元主矢或单元主矩不为零的情况，因此我们用单元中点上的剪力代替节点处的剪力，并依此来修正节点处的弯矩。细心的读者可能会注意到，这种处理方式与选择性减缩积分的思想是相通的。

输出单元中性面上的 Gauss 点应力，用于恢复并输出节点应力。具体包括单元的轴向正应力 σ_{xx}、弯曲剪应力 σ_{xy} 与 σ_{xz}，其计算公式为

$$
\begin{aligned}
\sigma_{xx} &= E\varepsilon_{xx} = \frac{E(u_2-u_1)}{l},\\
\sigma_{xy} &= \frac{G}{k}\gamma_{xy}\\
&= \frac{G}{kl}\left((v_2-v_1)-\frac{1}{2}(\theta_{z1}+\theta_{z2})l\right),\\
\sigma_{xz} &= \frac{G}{k}\gamma_{xz}\\
&= \frac{G}{kl}\left((w_2-w_1)+\frac{1}{2}(\theta_{y1}+\theta_{y2})l\right).
\end{aligned}
$$

剪应变直接插值

对于 Timoshenko 梁单元的情况，直接用剪应变插值的方法与挠度转角独立插值并减缩积分的方法是等价的，这里不再赘述。

4.4.3 程序实现与收敛性分析

程序实现

我们在 `STAPpp` 程序中新增了两种单元，即 `TimoshenkoSRINT` 和 `TimoshenkoEBMOD`，分别对应于两种不同的 Timoshenko 梁单元格式构造思路，即位移转角独立插值方法以及欧拉伯努利梁剪切修正方法。对于前者，我们还设置了预处理器标签 `TIMOSHENKOACCURATEINTEGRATION_` 用于单元刚度阵数值积分方法的转换，开启时采用精确积分、关闭时采用减缩积分。另外，由于两种单元涉及的材料/截面性质是类似的，我们在基类 `Material` 的基础上定义了派生类 `CTimoshenkoMaterial` 用于储存上述两种单元的材料/截面性质。

单元刚度阵组装以及单元应力计算的算法上一大节已经给予了充分的说明，下面主要叙述材料/截面性质相关的实现细节。在阅读了 ABAQUS 文档后，笔者最终决定在 Timoshenko 梁单元部分采用杨氏模量、泊松比、截面面积、局部坐标轴惯性矩以及局部 y 轴方向的输入参数设置方式。在输入格式的具体设置上，材料/截面性质控制行与纯弯梁单元完全相同，依次为杨氏模量 `E`、泊松比 `nu`、截面面积 `Area`、单元局部 y 轴惯性矩 `Iyy`、单元局部 z 轴惯性矩 `Izz`、单元局部 y 轴方向 `Thetay1, Thetay2, Thetay3`。

分片试验结果

我们首先探讨 Timoshenko 梁单元的分片试验的含义。一般认为，分片试验应该包括单元的刚体模态与常应变模态。前者的概念是清楚的，而后者的含义则随单元类型的不同而有所差异，但总的来说是指使得单元应变能表达式中的被积函数项为常数的位移模态。对于 Euler-Bernoulli 梁单元来说，所谓常应变模态实际上是常曲率模态，这与该单元应变能表达式中只含有弯曲应变能项是一致的。对于 Timoshenko 梁来说，所谓常应变模态实际上应该指常曲率且常剪应变模态，具体到位移与转角的分布上，可写为

$$
\begin{aligned}
v &= \frac{1}{2}ax^2 + cx + d\,; \\
\theta_z &= ax + b.
\end{aligned}
$$

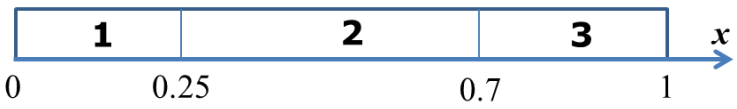


图 4.8: Patch test

容易看到，这实际上正对应于一端受集中剪力载荷、集中弯矩载荷共同作用的悬臂梁情形，其精确解我们在第一大节已经给出。作为分片试验，我们采用非等距单元（如图 4.8 所示），并分别在端部施加集中剪力载荷以及集中弯矩载荷。

图 4.9, 4.10 分别给出了采用选择性减缩积分的 TimoshenkoSRINT 格式在集中弯矩和集中剪力作用下计算得到的有限元解与精确解的对比图；图 4.11, 4.12 分别给出了精确积分时对应的有限元解与精确解的对比图。

对比图 4.9, 4.10 和图 4.11, 4.12 可以看到，选择性缩减积分提高了节点位移和转角的计算精度；并且，除了端部剪力载荷工况下的位移曲线外，经过选择性缩减积分的 TimoshenkoSRINT 格式均通过了分片试验。为了确保收敛性，下一小节我们将对 TimoshenkoSRINT 格式进行收敛率分析。

图 4.13, 4.14 分别给出了 TimoshenkoEBMOD 格式在集中弯矩和集中剪力作用下计算得到的有限元解与精确解的对比图。由此可见，该单元可以通过分片试验，且由形函数的特点可知该单元具有四阶收敛性。

收敛率分析

为了得到 TimoshenkoSRINT 梁单元的收敛率，我们仍沿用分片试验时的模型来考察误差应变能随等距单元数加倍时的性态。图 4.15 给出了 TimoshenkoSRINT 格式两种积分方法在端部集中弯矩载荷工况下的收敛率曲线，图 4.16 则给出了集中剪力作用下的收敛率曲线。这里，误差采用能量范数的平方度量，其中误差应变能定义为 $U(u-u^h)$。

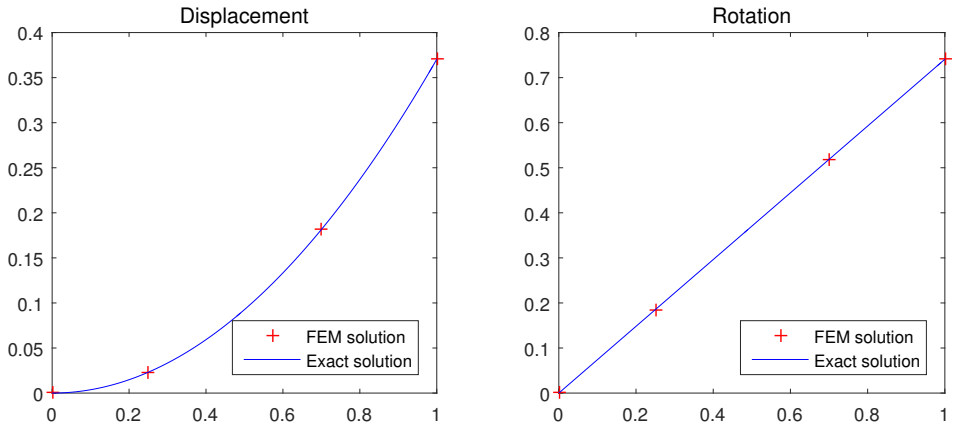


图 4.9: Concentrated M_z **WITH** SRINT

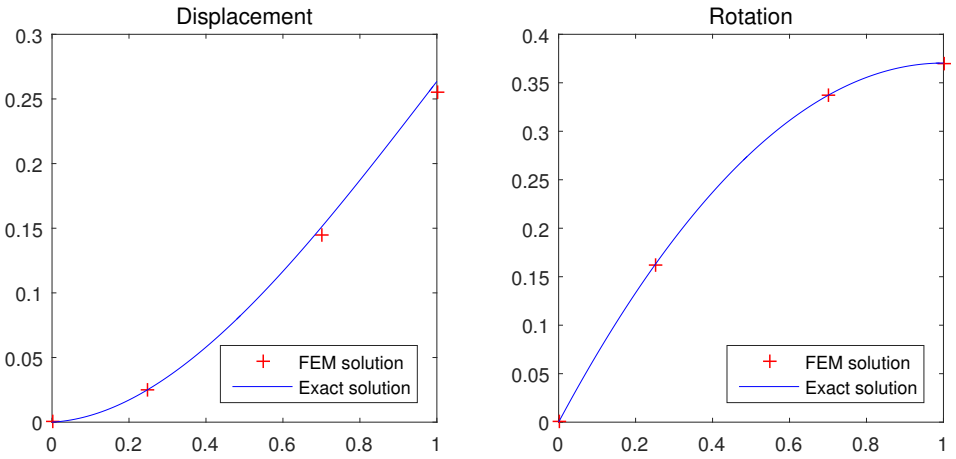


图 4.10: Concentrated F_y **WITH** SRINT

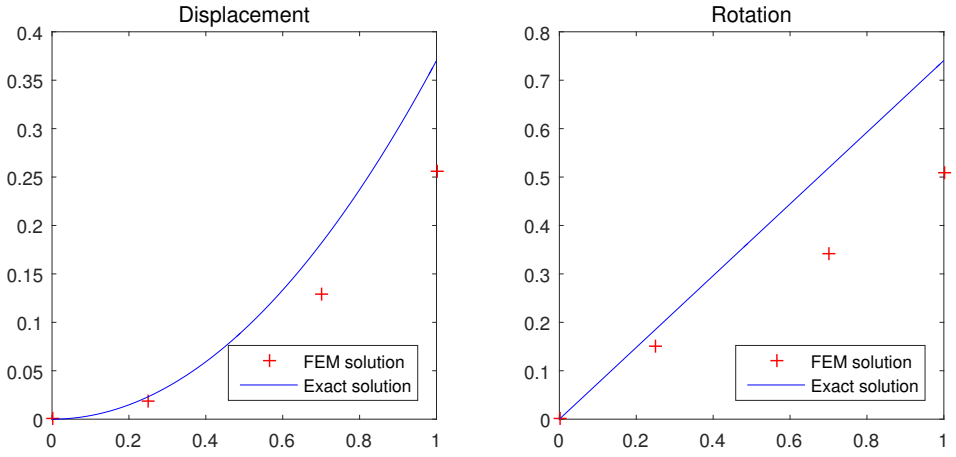


图 4.11: Concentrated M_z **WITHOUT** SRINT

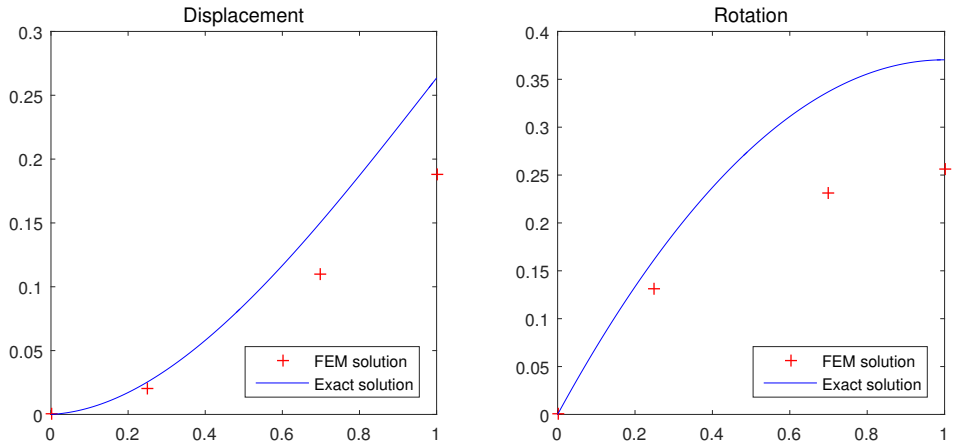


图 4.12: Concentrated F_y **WITHOUT** SRINT

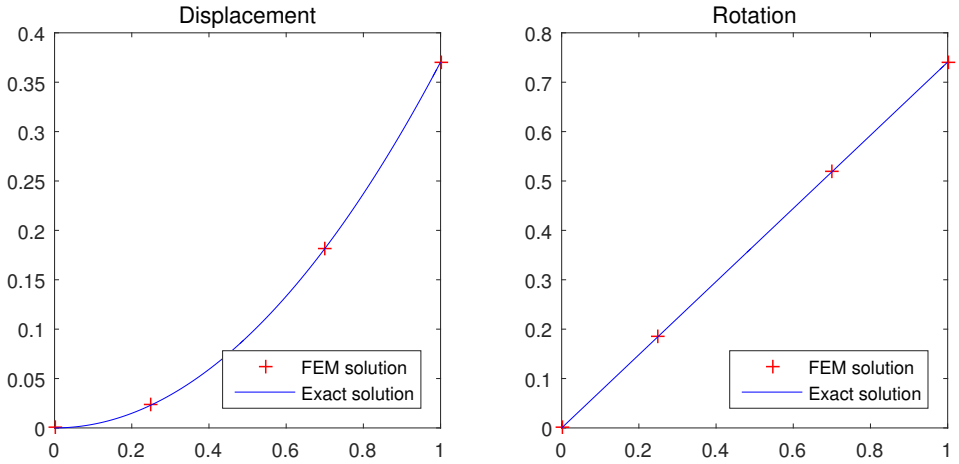


图 4.13: Concentrated M_z with EBMOD

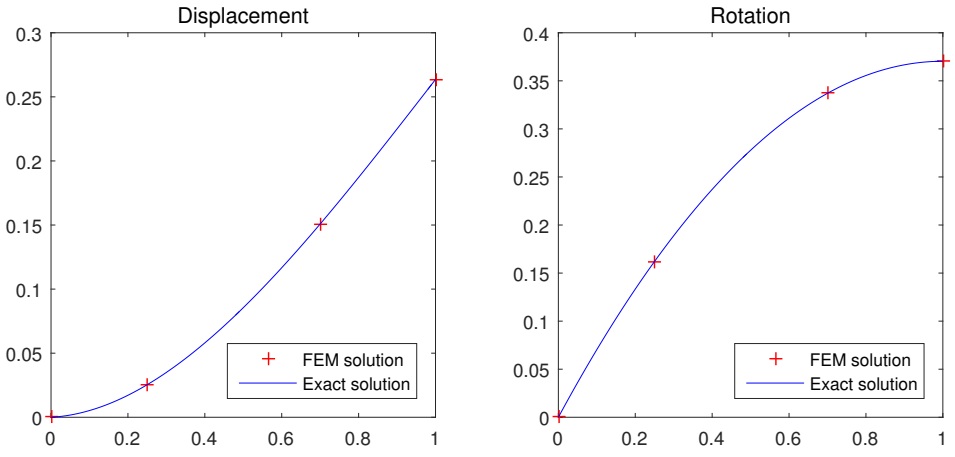


图 4.14: Concentrated F_y with EBMOD

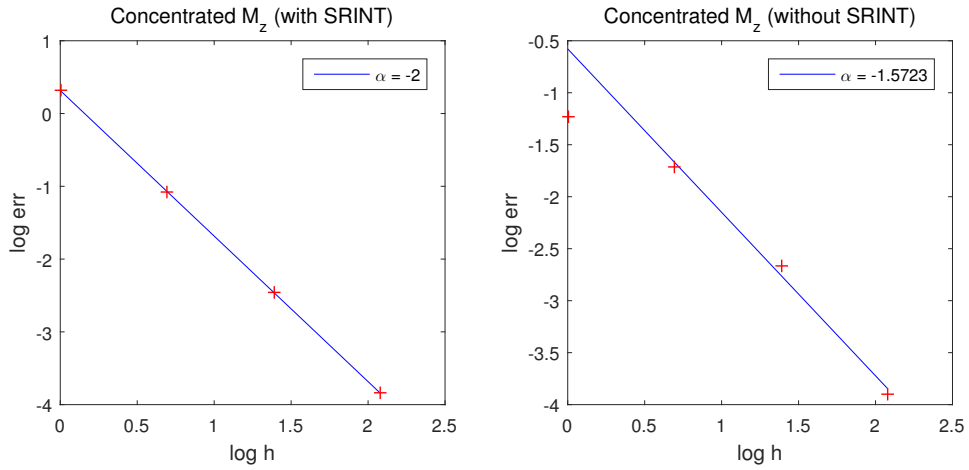


图 4.15: Convergence Curve (Concentrated M_z)

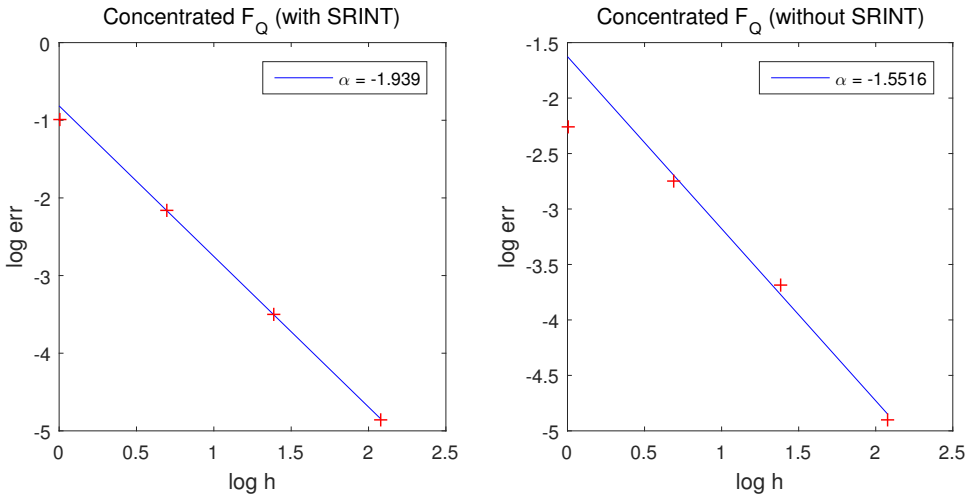


图 4.16: Convergence Curve (Concentrated F_y)

从上述收敛率分析的结果可以看出，在误差能量范数的意义下，选择性缩减积分的TimoshenkoSRINT 具有一阶收敛率，而精确积分情形的收敛率则很可能小于一阶。因此，尽管后者在单元数较少时可能会有较好的表现，但当单元数增多时，前者的精度会提高得更快并最终占据绝对优势。

4.5 薄板单元

4.5.1 板壳单元基本原理

我们所选用的板壳就是最简单的板壳,对于板选用的是 4 节点 12 自由度矩形 Kirchoff 板,壳在板的基础上加上前面的 4Q 单元部分. Kirchoff 板遵循平截面假设和直法线假设,因而有

$$\begin{cases} u = u_0 + z\theta_y \\ v = v_0 - z\theta_x \\ w = w_0(x,y) \end{cases}$$

通过假设 z 方向没有剪应变我们可以得到 $\theta_x = \partial x/\partial y, \theta_y = \partial w/\partial x$, 从而

$$\begin{cases} \varepsilon_x = -z\dfrac{\partial^2 w}{\partial x^2} \\ \varepsilon_y = -z\dfrac{\partial^2 w}{\partial y^2} \\ \gamma_{xy} = -2z\dfrac{\partial^2 w}{\partial x \partial y} \end{cases}$$

利用上式以及本构关系,只要我们给出了板单元的形函数,我们就能通过积分得到板单元的刚度矩阵.

4.5.2 板单元刚度阵的构造

这里我们采用了彭细荣书中给出的形函数,

$$\boldsymbol{N}^T = \begin{Bmatrix} -\frac{1}{8}(s-1)(t-1)(s^2+s+t^2+t-2) \\ -\frac{1}{8}b(s-1)(t-1)^2(t+1) \\ \frac{1}{8}a(s-1)^2(s+1)(t-1) \\ \frac{1}{8}(s+1)(t-1)(s^2-s+t^2+t-2) \\ \frac{1}{8}b(s+1)(t-1)^2(t+1) \\ \frac{1}{8}a(s+1)^2(s-1)(t-1) \\ -\frac{1}{8}(s+1)(t+1)(s^2-s+t^2-t-2) \\ \frac{1}{8}b(s+1)(t+1)^2(t-1) \\ -\frac{1}{8}a(s+1)^2(s-1)(t+1) \\ \frac{1}{8}(s-1)(t+1)(s^2+s+t^2-t-2) \\ -\frac{1}{8}b(s-1)(t+1)^2(t-1) \\ -\frac{1}{8}a(s-1)^2(s+1)(t+1) \end{Bmatrix}$$

根据 $\boldsymbol{B} = [-\frac{\partial^2}{\partial x^2}, -\frac{\partial^2}{\partial y^2}, -2\frac{\partial^2}{\partial x \partial y}]^T \boldsymbol{N}$ 以及 $\boldsymbol{K} = 4ab\boldsymbol{B}^T\boldsymbol{D}\boldsymbol{B}$ 得到总刚度阵.需要说明的是,这里的形函数事实上是一个沿板厚变化的值,但经过积分可以化为弯曲刚度中的一部

分.对于得到的刚度阵利用 Zienkiewicz 的原始论文进行初步的验证,二者得到的结果是一致的.

在输出刚度阵时,我们利用 MATLAB 进行完全的积分,并用 char 函数将符号运算输出为字符串,通过少量替换就可以直接在 C++ 中使用.

4.5.3 分片验证

我们在 $[-1,1]^2$ 的正方形板构造这样的位移场 $w=x^2-\nu y^2$, 对应地$\theta_x=-2\nu y$, $\theta_y=-2x$, 得到一个纯弯场 $M_{11}=-2D(1-\nu^2)$. 利用 $x=0; x=0.3; y=-0.2; y=0$ 四条直线将正方形板分割成九个分片,如图4.17,对应的将法线沿 x 轴方向边界上的 y 向弯矩分配到每个点上.具体输入文件见 plate_patch_test, 在程序中的输出见图4.18,最后三行的输出同壳单元;可见位移基本准确,误差在浮点数范围内.

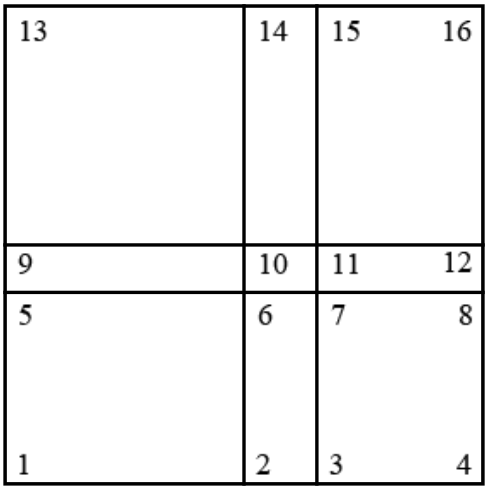


图 4.17: 分片示意

NODE	X-DISPLACEMENT	Y-DISPLACEMENT	Z-DISPLACEMENT			
1	0.00000e+000	0.00000e+000	8.00000e-001	4.00000e-001	2.00000e+000	0.00000e+000
2	0.00000e+000	0.00000e+000	-2.00000e-001	4.00000e-001	5.33532e-014	0.00000e+000
3	0.00000e+000	0.00000e+000	-1.10000e-001	4.00000e-001	-6.00000e-001	0.00000e+000
4	0.00000e+000	0.00000e+000	8.00000e-001	4.00000e-001	-2.00000e+000	0.00000e+000
5	0.00000e+000	0.00000e+000	9.92000e-001	8.00000e-002	2.00000e+000	0.00000e+000
6	0.00000e+000	0.00000e+000	-8.00000e-003	8.00000e-002	3.06699e-014	0.00000e+000
7	0.00000e+000	0.00000e+000	8.20000e-002	8.00000e-002	-6.00000e-001	0.00000e+000
8	0.00000e+000	0.00000e+000	9.92000e-001	8.00000e-002	-2.00000e+000	0.00000e+000
9	0.00000e+000	0.00000e+000	1.00000e+000	-3.41949e-014	2.00000e+000	0.00000e+000
10	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000
11	0.00000e+000	0.00000e+000	9.00000e-002	-2.52836e-015	-6.00000e-001	0.00000e+000
12	0.00000e+000	0.00000e+000	1.00000e+000	-8.27116e-015	-2.00000e+000	0.00000e+000
13	0.00000e+000	0.00000e+000	8.00000e-001	-4.00000e-001	2.00000e+000	0.00000e+000
14	0.00000e+000	0.00000e+000	-2.00000e-001	-4.00000e-001	4.92991e-014	0.00000e+000
15	0.00000e+000	0.00000e+000	-1.10000e-001	-4.00000e-001	-6.00000e-001	0.00000e+000
16	0.00000e+000	0.00000e+000	8.00000e-001	-4.00000e-001	-2.00000e+000	0.00000e+000

图 4.18: 输出位移

4.5.4 单点位移收敛率分析

取四边简支的正方形薄板,薄板大小同上,取 $E=230.4GPa, \nu=0.2$, 在中央加一集中载荷 P, 根据弹性力学可以得到其级数解 $w=\frac{16P}{D\pi^4}\sum\limits_{m=0}^{\infty}\sum\limits_{n=0}^{\infty}\frac{(-1)^(m+n)}{((2m+1)^2+(2n+1)^2)^2}\sin\frac{m\pi x_1}{2}\sin\frac{n\pi x_2}{2}$,

通过数值方法得到其中央的最大挠度为 $\frac{0.04640335P}{D} = 0.232017$. 对于 $2\times 2, 4\times 4, 8\times 8$ 依次进行求解,得到的结果的对数误差如图4.19.

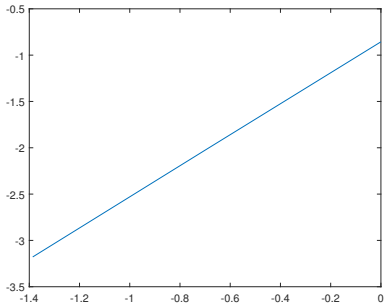


图 4.19: 中点收敛率分析

对于板,其形函数是三次以上的,而且构造的位移场是级数场,因而中点不是其高斯点.发现收敛率为 1.67 ,考虑到构造的不是多项式场,认为这样的收敛率是可以满意的,也与书上的结果符合,因此认为该板是收敛的.具体文件见 downpress1, downpress2, downpress3.

4.5.5 板壳单元自由度的约束

由于板单元与平板壳单元只能提供面外两个方向的旋转刚度,却不能提供面内的旋转刚度,因此我们需要约束相关方向的自由度,同时放开面外的旋转自由度.因此我们在梁解放相关自由度后还有对于板壳单元自由度的处理:当输入的自由度不含旋转自由度时,对板壳单元所在各点进行判断,如果各点的某一坐标(比如 z 坐标)相差不超过浮机器精度,认为该单元的法向是沿 z 方向的,从而释放对应几个点 x, y 方向的自由度而 z 方向自由度不变,这样也不会影响已经具有刚度而被放开的自由度.

对于倾斜方向的板需要约束变分条件才能解决,目前不在我们程序解决的范围之内.

4.5.6 板壳单元输入格式

板单元和平板壳单元的代号分别为 6 和 7 ,输入材料时传递三个参数,按顺序分别为单元的杨氏模量 E, 单元的泊松比 ν 和单元的厚度 h. 每个单元只需要输入四个顶点的单元编号.

4.6 截锥壳单元

壳单元通常可分为平板壳、扁壳、退化型壳等类型，除此之外还有针对轴对称壳直接构造出的截锥 (frustum) 单元。有关板壳单元的更全面的讨论可参考 `doc/litReview/On plate and shell elements` 文件夹。

4.6.1 单元构造

我们采用两点截锥单元，其单元构造方式与梁单元是完全类似的。二者的不同之处有二，其一是，由于截锥壳是初始曲率非零的壳体，因此其应变及曲率的表达式中增加了面内变形与弯曲变形的耦合项；其二是，截锥壳本质上属于二维背景空间中的单元，因此需

要约束掉其“面外变形”的自由度，如 y, θ_x, θ_z。有关单元刚度阵的具体构造方式这里不再赘述，具体可以参考 `memo/frustum` 文件夹中的 MATLAB 程序以及参考文献 [5]。

4.6.2 单元验证与讨论

由于一般的壳单元精确解不易求得，考虑到圆柱壳是轴对称壳中最简单的情形之一，我们选取“在一圆截面上有均匀分布载荷的长圆柱形壳的弯曲”作为单元构造的验证算例。这一问题的具体物理图像可以参考文献 [10] 第十五章第 115 节。

在以集中载荷的作用点为原点的坐标系中，可以写出圆柱壳的横向弯曲变形随轴向坐标的变化关系

$$w = \frac{PR^2\beta}{2Eh} e^{-\beta|x|}(\sin\beta|x| + \cos\beta|x|).$$

其中，$\beta^4 = \frac{3(1-\nu^2)}{a^2h^2}$。这里需要注意选取合适的参数，即要避免横向挠度变形大于截面半径情况的发生，这等价于要求圆柱壳的壳体足够薄；参数的具体选法可以参考相应的输入文件或者 `memo/frustum` 文件。作为验证，我们将圆柱壳沿轴向等分为 60 份截锥单元，于是可以得到如图 4.20 和图 4.21 所示的结果。

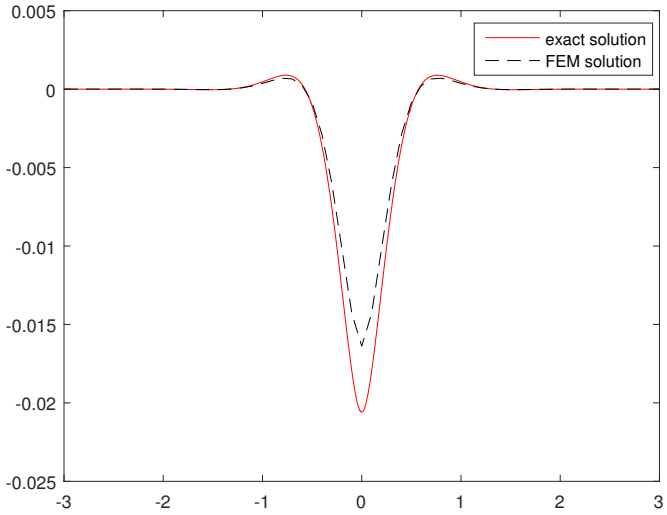


图 4.20: 有限元解与精确解横向挠度变形对比（全局）

这里有限元解放大至了原来的 10 倍。由图 4.20 和图 4.21 可见，尽管截锥壳单元的结果相较精确解小了一个数量级，但是截锥壳单元的有限元解与精确解之间的形状是十分相符的。需要注意的是，由于算例中的圆柱壳并非无限长，因此精确解实际上并不完全适合本算例；与此同时，实际上在集中载荷附近应该加密网格，因此上述结果中集中载荷附近的位移值也是不够精确的。

4.7 无限单元

4.7.1 单元构造

本质上，无限单元是 4Q 映射单元的一种，只是在一个方向上的映射比较特殊。在这个方向上需要将母空间的点投影到物理空间中的无穷远，而且要求对映射区域引入的试函数

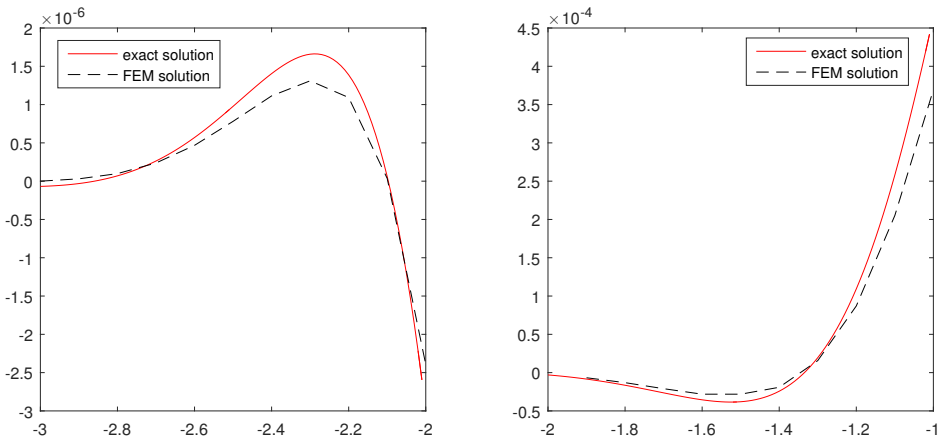


图 4.21: 有限元解与精确解横向挠度变形对比（细节）

序列在半径 r 增加的情况下仍能够完整描述模型的真实行为。一个理想的映射形函数为：

$$\frac{C_1}{r}+\frac{C_2}{r^2}+\frac{C_3}{r^3}+,$$

其中 C_i 为常数，r 是到“扰动”中心的距离。

但是很明显的上面这个式子没有实用性，现在引入一个符合要求的映射函数：

$$x=-\frac{\xi}{1-\xi}x_C+\frac{1}{1-\xi}x_Q$$

母空间和物理空间中的对应关系如图 4.22 。

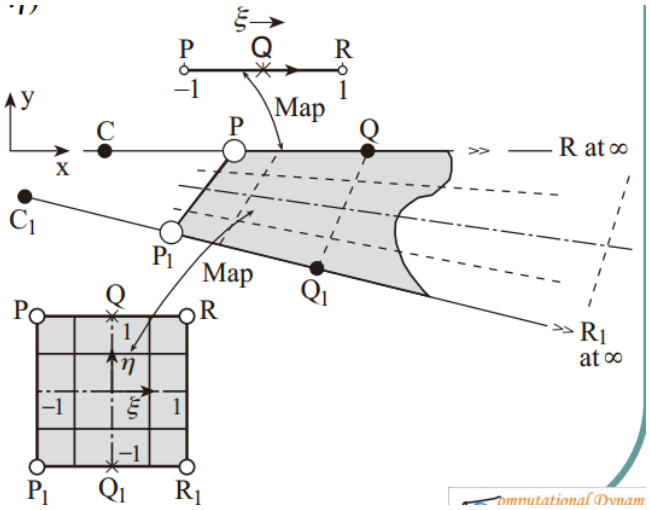


图 4.22: 无限单元的映射关系

可以看出：

$$\begin{cases}\xi=1, x=\frac{x_Q+x_C}{2}=x_P\\ \xi=0, x=x_Q\\ \xi=1, x=x_R\to\infty\end{cases}$$

其中 C 点在物理空间内是“扰动的原点”，这一点并不约束在无限单元内，而是一个自己定义的点。 P 点是和其他单元协调的节点，而这个形函数则约束了 P 为 C 与 Q 的中

ABAQUS 结果	$5 \times 10^{-}6$
STAP 结果	$4.51231 \times 10^{-}6$

表 4.1: 节点位移结果比较

点，从而确定了 C 点和 P 点就确定了 Q 点。考虑到 P 点的位移需要和其他单元协调，故形函数需要包含 P 点，改写为：

$$x = \frac{1+\xi}{1-\xi} x_Q + \frac{-2\xi}{1-\xi} x_P$$

加入另一个方向的形函数，得到整体的形函数表达为：

$$x = N_1(\eta) N_P(\xi) x_P + N_1(\eta) N_Q(\xi) x_Q + N_1(\eta) N_P(\xi) x_{P_1} + N_1(\eta) N_Q(\xi) x_{Q_1}$$

$$N_1 = \frac{1+\eta}{2}, N_2 = \frac{1-\eta}{2}$$

$$N_P = \frac{-2\xi}{1-\xi}, N_Q = \frac{1+\xi}{1-\xi}$$

得到形函数表达之后的操作和 4Q 单元完全一致。

4.7.2 算例验证

考虑到无限单元对应的算例都很难计算出理论解，故考虑采用和同一模型下 ABAQUS 的计算结果进行比较，算例如图，为无限大的平板中间挖一个正方形的小孔，四个箭头代表四个大小相等的力，如图 4.23 。

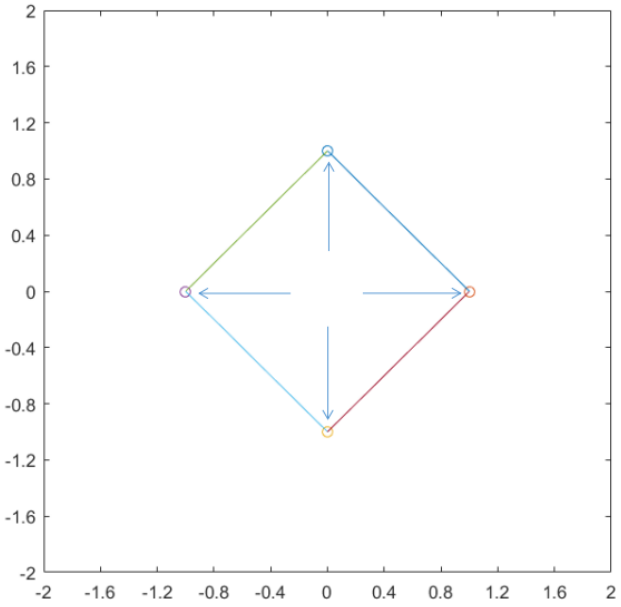


图 4.23: 无限单元算例

由于系统的高度对称性，只看一个节点的位移即可。结果见表 4.1 。

实际上，节点位移从理论上来看就不精确。另外，ABAQUS 用的也不是无限大平板——只是使用了一个非常大的平板而已。而且在实际使用中，正确的做法应该是方孔附近

使用 4Q 单元，距离中心有一段距离之后再使用无限单元与 4Q 协调。所以， 10% 的误差还是在可接受范围内。

4.8 过渡单元

4.8.1 单元构造

采用巧凑边点元的模型构造 9Q 与 4Q 之间过渡用的 5Q 单元，单元标号如图 4.24 。

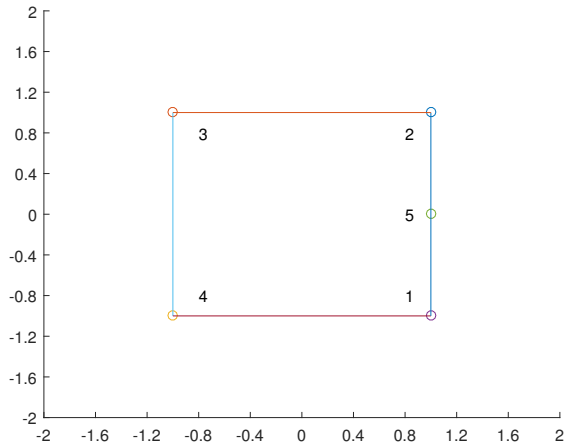


图 4.24: 5Q 单元的标号顺序

形函数构造如下：

$$\begin{cases} N_5^{5Q} = \frac{\left(1-\eta^2\right)(1+\xi)}{2} \\ N_1^{5Q} = N_1^{4Q} - \frac{N_5^{5Q}}{2} = \frac{(1+\xi)\left(\eta^2-\eta\right)}{4} \\ N_2^{5Q} = N_2^{4Q} - \frac{N_5^{5Q}}{2} = \frac{(1+\xi)\left(\eta^2+\eta\right)}{4} \\ N_3^{5Q} = N_3^{4Q} = \frac{(1-\xi)(1+\eta)}{4} \\ N_4^{5Q} = N_4^{4Q} = \frac{(1-\xi)(1-\eta)}{4} \end{cases}$$

之后的思路和 4Q 单元相同，由形函数求导得到 B 矩阵，再由公式得到单元刚度阵（注意此时刚度阵已经是 10 阶矩阵了）：

$$K^e = \int_\Omega B^T D B d\Omega$$

考虑到桥梁算例中没有应用 5Q 单元的必要，故没有做从局部坐标系到全局坐标系的变换。

4.8.2 算例验证

使用这样一个算例进行验证,如图 4.25 。

载荷情况：系统整体受到向右的均布载荷，为单向拉伸状态。13 号节点在 x，y 方向的位移被约束，12 号节点在 x 方向的位移被约束。位移场的精确解为 $u_x = 0.0003x, u_y = 0.00009x$，而算例输出的结果如图 4.26 。

节点位移精确，通过算例。

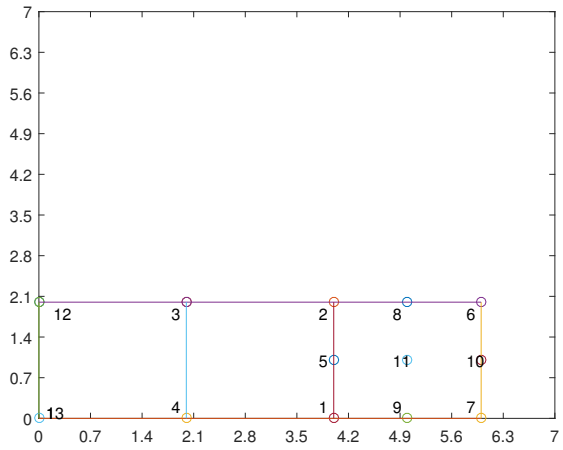


图 4.25: 算例验证的标号情况

D I S P L A C E M E N T S

NODE	X-DISPLACEMENT	Y-DISPLACEMENT	Z-DISPLACEMENT	X-ROTATION	Y-ROTATION	Z-ROTATION
1	1.20000e-004	-9.07306e-014	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000
2	1.20000e-004	-1.80000e-005	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000
3	6.00000e-005	-1.80000e-005	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000
4	6.00000e-005	-1.02530e-013	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000
5	1.20000e-004	-9.00000e-006	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000
6	1.80000e-004	-1.80000e-005	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000
7	1.80000e-004	-1.23326e-012	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000
8	1.50000e-004	-1.80000e-005	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000
9	1.50000e-004	4.36230e-013	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000
10	1.80000e-004	-9.00000e-006	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000
11	1.50000e-004	-9.00000e-006	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000
12	0.00000e+000	-1.80000e-005	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000
13	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000	0.00000e+000

图 4.26: 算例验证的结果

Chapter 5

扩展功能

这一章介绍了 STAP++ 程序的一些扩展功能，包括基于 MKL 的稀疏求解器 (见 5.1 节)、基于 Bar 单元的模态分析 (见 5.2 节)以及基于 8H 单元的分片应力恢复 (见 5.3 节)。

5.1 稀疏求解器

5.1.1 稀疏矩阵的实现

LDLT 求解器使用的是 skyline 矩阵（一维变带宽储存方式），因此我们实现了一个 CSR 矩阵 `CSRMatrix` 来适配 Pardiso 求解器。

类似 skyline 矩阵， CSR 矩阵也需要提前矩阵每行的长度来分配内存。但是每行有多少非零元是不可知的，因此，我们需要扫描一遍所有元素，标记每一行非零元的位置。为此，我们实现了 `CSRMatrix::beginPositionMark` 和 `CSRMatrix::markPostion` 方法来标记每一行的位置和列号。在标记的过程中，需要变长度地储存每一行地列号序列。实现这一功能有多种方法：基于链表结构、基于重新分配的数组结构、基于平衡树的有序结构。

考虑到开发过程的快速和效率，我们使用了上述数据结构所对应的 STL 标准容器，即 `std::list`、`std::vector`、`std::set`。STL 容器使用方便、在高优化下效率极高、接口设计科学、有独立的内存管理（Allocator）避免内存碎片化，有很大的优势。

几种结构中，`std::list` 是基于链表的结构，插入 $O(1)$，需要排序，排序 $O(n\log n)$，总体 $O(n\log n)$。 `std::vector` 是基于数组的结构，插入平均 $O(1)$，最坏 $O(n)$，需要排序，排序 $O(n\log n)$，总体 $O(n\log n)$。 `std::set` 是基于平衡树的结构，插入 $O(\log n)$，不需要排序，总体 $O(n\log n)$。

虽然这几种结构看起来复杂度是一样的，但是他们的时间常数并不一样。这几种结构中，我们首先排除的是 `std::list`，因为实际使用中，链表的插入和排序都远远慢于数组。在剩下的 `std::vector` 和 `std::set` 在实际测试中表现接近，在不同的数据下互有胜负。因此，我们在最后的程序中同时保留了两者，通过一个宏实现类型切换。

在标记位置结束后，调用 `CSRMatrix::allocate` ，对储存行的非零元位置的临时对象进行排序、去重（如果使用 `std::vector` ）、内存的分配和列的填入。

5.1.2 求解器的优化

我们在程序中打开了 pardiso 的多线程选项，链接了并行的静态库，可以会自动识别可用核心，开启多线程。多线程的线程数还可以使用环境变量 `MKL_NUM_THREADS` 控制：

```
$ export MKL_NUM_THREADS=4
```

pardiso 求解器会自动根据内存占用切换内核求解（In-Core, IC）和外核求解模式（Out-Of-Core, OOC）。通过修改运行目录下的 `pardiso_ooc.cfg`，我们可以设置 pardiso 所占用的最小内存和外核缓存路径。使用外核求解器时，如果将缓存路径设置到 SSD 上，可以将速度提升十数倍。

5.2 模态分析

5.2.1 模态分析简介

模态分析的主要问题是解广义特征值问题 $|\boldsymbol{K}-\lambda\boldsymbol{M}|=0$ 的前几阶特征值及其对应的特征向量,因此我们把模态分析分为两个部分来进行:组装对应的刚度矩阵和质量矩阵,以及求解广义特征值问题.当我们把自由度选在节点上时,刚度阵就是静态分析中的刚度矩阵,因此我们只需要对于每个单元以类似的方法组装质量矩阵.对于求解广义特征值问题,我们使用的是子空间迭代法.

5.2.2 程序实现

在组装质量矩阵时,程序仅给出了杆单元的质量矩阵为例.对单元增加私有变量密度 `rho`,并改变其读写函数,并增加纯虚函数组装质量矩阵 `ElementMass`. 我们使用的是一致质量矩阵,有 $\boldsymbol{M}=\int\rho\boldsymbol{N}^T\boldsymbol{N}\mathrm{d}\Omega$, 因此对于杆单元应当有

$$\boldsymbol{M}=\frac{W}{3}\begin{pmatrix}2&1\\1&2\end{pmatrix}$$

考虑到我们的杆单元是三维空间中的单元,因此对于左右还需进行坐标变换.

对广义特征值问题,首先给出初始向量 $\boldsymbol{X}_1$ 以及 $\boldsymbol{Y}_1=\boldsymbol{M}\boldsymbol{X}_1$, 之后进入循环.在求得的特征值尚未收敛时,利用逆迭代法求下一步的 $\boldsymbol{X}_{n+1}=\boldsymbol{K}^{-1}\boldsymbol{Y}_n$, $\boldsymbol{Y}_{n+1}=\boldsymbol{M}\boldsymbol{X}_{n+1}$, 并求缩并后的矩阵 $\bar{\boldsymbol{K}}=\boldsymbol{X}_{n+1}^T\boldsymbol{K}\boldsymbol{X}_{n+1}=\boldsymbol{X}_{n+1}^T\boldsymbol{Y}_n$, $\bar{\boldsymbol{M}}=\boldsymbol{X}_{n+1}^T\boldsymbol{M}\boldsymbol{X}_{n+1}=\boldsymbol{X}_{n+1}^T\boldsymbol{Y}_{n+1}$. 这一部分直接在 C++ 中编写代码运行,主要原因是考虑到将原矩阵扩充为对角阵或者带宽阵均会导致占用不必要的空间,我们小组有 LDLT 以及 Pardiso 两种求解器,目前振动模块只兼容 LDLT,但将这一部分单独写出也方便之后改用稀疏求解器.

缩并后的矩阵由于维数一般不高,可以取上三角阵存储,方便后续输入 LAPACK 的对应求解驱动进行求解,这里我们使用的是 `LAPACKE_dspgvd` 函数,这个函数的作用是对解三角存储的对称矩阵的广义特征值问题的所有特征值和特征向量.解得对应的特征值和特征向量之后,再利用选取的基向量把缩减自由度后的特征向量恢复至原坐标系中.如果特征值已经趋于收敛,那么跳出循环,得到的就是子空间迭代法的前 n 阶特征值与特征向量.

5.2.3 算例验证

对于该模态分析的验证我们选取了一个一维的质弹系统,如图5.1.

图 5.1: 验证算例

取线密度为 1, 杨氏模量为 4×10^{10}, 横截面积为 1×10^{-2}, 每个杆长度为 1. 这样得到三个特征值与对应的特征向量分别为

$$\lambda_1 = 1.219 \times 10^8, \lambda_2 = 1.2 \times 10^9, \lambda_3 = 3.9494 \times 10^9$$

$$\boldsymbol{d}_1 = \begin{pmatrix} 0.3536 \\ 0.6124 \\ 0.7071 \end{pmatrix}, \boldsymbol{d}_2 = \begin{pmatrix} -0.7071 \\ 0.0000 \\ 0.7071 \end{pmatrix}, \boldsymbol{d}_3 = \begin{pmatrix} -0.3536 \\ 0.6124 \\ -0.7071 \end{pmatrix}$$

当我们输入的要求特征值数为 3 时,得到的结果是准确的.当要求特征值数为 2 时,我们得到如下结果:

$$\lambda_1 = 1.2196 \times 10^8, \lambda_2 = 1.24072 \times 10^9$$

$$\boldsymbol{d}_1 = \begin{pmatrix} -0.4174 \\ -0.7248 \\ -0.8332 \end{pmatrix}, \boldsymbol{d}_2 = \begin{pmatrix} 1.0732 \\ -0.1402 \\ -0.8312 \end{pmatrix}$$

考虑到特征向量没有进行正则化,可以发现一阶特征值以及特征向量误差极小,特征值的误差不超过千分之一,特征向量的误差在千分之四以下.而二阶特征值的误差则相对较大,特征向量的问题则更加显著.这些问题都是因为里兹基向量导致的,在原程序中我们没有使用特别的处理,仅通过逆迭代法逐渐放大对应阶数,因此容易出现这样的问题.在实际情况下,一般要求取前 n 阶特征值时,我们选取 $\min(2n, n+8)$ 作为初始求取的特征值数.

5.2.4 程序使用以及部分增加函数说明

目前要通过程序计算振动模态,需要进行如下操作:在 Cmake 时增加勾选 _VIB_ 的选项,并同时勾选 _DEBUG_, 这是因为不明链接原因导致二者必须同时勾选;同时不能使用 USE_MKL, 因为虽然在程序中使用了 MKL 库, 但是没有使用 Pardiso 求解器,因此会出现错误.之后还要打开 Visual Studio 之后手动打开 MKL 库,程序就能正常运行.

对于输入文件也要进行改动.对杆单元连接的质弹系统,在输入杆的材料时,材料参数增加为三个,输入顺序改为密度 ρ, 杨氏模量 E, 横截面积 A. 此外在输入所有单元之后,要继续输入要求解的特征值数.

程序中主要增加了函数 `bool CDomain::VibSolver(unsigned int NVibModes)`, 输入参数为求解的特征值阶数.代码首先生成一个初始基向量组,这个初始基向量组可能不够好但是会随着逆迭代逐渐变好.在这里用到了我们定义的函数 `void CLDLTSolver::Multiple(double* acc, double* Force, unsigned int numeq, unsigned int vib_m)`, 用途是计算 LDLT 矩阵与列向量相乘的结果.下面定义循环中需要使用的动态数组并进入循环,循环中的内容如上程序实现介绍,最终释放动态数组.此外,程序中的 `assembly_mass` 与 `AssembleMassMatrix` 等函数与对应的刚度矩阵函数的意义是相似的,只不过是改为组装质量矩阵.

5.3 分片应力恢复 (SPR)

为了获得更好的应力恢复效果，在使用了应力平滑（如 Gauss 点的应力外推）的基础上，我们加入了 SPR 方法。超收敛分片应力恢复方法，亦称 SPR，是通过使用具有超收敛点的高斯积分点对结点进行应力恢复，可以获得比应力平滑方法更高的精度。这里我们以 8H 单元为例进行详细讨论。

5.3.1 程序原理

根据所采用的 $2 \times 2 \times 2$ 点高斯积分，二阶完备多项式就足以保证两点高斯积分精度。而考虑三维完备二阶多项式，共有 10 项 (即 $1, x, y, z, xy, xz, yz, x^2, y^2, z^2$)。显然，为确定这 10 个系数，只需要至少两个单元包含该结点（每个单元含有 8 个高斯点）。因此我们将内部点、边界上的点（边点）、仅在一个单元角上的点（角点）分开讨论。

对于内部点和边点，取它所属于的所有单元作为一个分片，根据公式

$$\Pi = \sum_{k=1}^{n} [\sigma_i(x_k, y_k) - p_k a]^2.$$

其中，$a = A^{-1}b, A = \sum_{k=1}^{n} P_k^T P_k, b = \sum_{k=1}^{n} P_k^T \sigma_i(x_k, y_k)$。只要通过最小二乘求出系数，代入结点的物理坐标，就可以求出结点的恢复应力。

对于角点，取它所在的各个分片在该点应力恢复值的算数平均值。这样既最大限度地利用了高斯点的信息和精度，又不至于增加太多计算量。每个内部点和边点都应当计算并只计算了一次。

5.3.2 算法实现

步骤如下，具体代码可以参见 SPR8H.cpp，这里使用了 Eigen 的矩阵运算库。

1. 按照单元组循环。
2. 对该单元组中的每个节点遍历，记录下每个节点所属于的单元数。
3. 对所属于单元数大于等于 2 的结点遍历，取它所属于的所有单元作为它的分片做最小二乘法，求出 10 个系数。代入该结点的坐标，计算出应力。
4. 对所属于单元小于 2 的结点，它总会至少出现在一个上一步所述分片中。采用这个分片计算出的拟合系数代入该点物理坐标，计算出恢复应力，并记录下这个点被计算的次数。
5. 将被计算次数多于一次的结点的应力去算数平均值。
6. 按照 Tecplot 格式输出结点坐标和对应应力值。

5.3.3 改进效果

通过在 Tecplot 中画图得到的效果和对比 ABAQUS 给出的解可以看出，SPR 算法进一步有效地改进了应力精度。并可以完全代替应力平滑，取得更好的效果。

Chapter 6

致谢

感谢张老师和宋言学长在整个项目完成工作中对本组全组同学的耐心指导！

感谢在部分任务中，其余组的部分同学与本组同学的积极讨论与互相扶持！

最要感谢的，是我们组里的每一位同学，感谢大家一直以来的不懈努力以及团队所有成员的通力合作！

附录 A

输入文件格式

输入格式完全兼容 STAP90 的输入格式。更新的输入格式如下。
为注释，忽略。 *args 代表多个参数。

```
# input.dat:
# Heading line
Heading line info

# Control line
NUMNP NUMEG NLCASE MODEX

# Nodal point data lines
N *ID X Y Z
#  here, len(ID) == 3 or len(ID) == 6.
#  when len(ID) == 6, ID[3:6] represents
# whether dof of rotation is fixed.
#  here, ID[i:j] := {ID[i],...,ID[j-1]}.

# Load data lines
LL NLOAD
NOD IDIRN FLOAD
# 1 <= IDIRN <= 6. when 4 <= IDIRN <= 6, it represents torque.

# Element data lines
NPAR(1) NPAR(2) NPAR(3)
N *ARGS
M *NODEINDEXS MTYP
#  here, len(ARGS) and len(NODEINDEXS) varies, depending on
# element type.
```

附录 B

输出文件格式 (后处理用)

这里我们采用的文件格式为 dat，具体数据格式如下

```
TITLE = " STAPpp FEM "

VARIABLES = "X_POST", "Y_POST", "Z_POST", "STRESS_I", "STRESS_II",
        "STRESS_III", "STRESS_VONMISES", "STRESS_XX", "STRESS_YY",
        "STRESS_ZZ", "STRESS_XY", "STRESS_YZ", "STRESS_ZX"

/* ---------------- 控制行 --------------- */
// NodeNum: 结点数，若有重复输出的结点，则重复计数;
// EleNum: 单元数
// FEPOINT: 有限元格式
// EleType: 单元类型，包括 BRICK,QUADRILATERAL 等
ZONE T = "Bridge", N = NodeNum, E = EleNum, F = FEPOINT, ET = EleType, C = RED

/* --------------- 结点信息 -------------- */
// 依次输入 "X_POST", "Y_POST", "Z_POST", "STRESS_I", "STRESS_II",
        "STRESS_III", "STRESS_VONMISES", "STRESS_XX", "STRESS_YY",
        "STRESS_ZZ", "STRESS_XY", "STRESS_YZ", "STRESS_ZX"

/* ---------- 单元-节点连接信息 ---------- */
// 依次输入每个单元对应的所有结点号，结点号排序方式按上一段输出的顺序
```

附录 C

优化思路简介

对 STAP++ 的优化我们尝试了如下几个方面：

- 组装刚度阵的并行优化尝试（见第八周组会报告）
- 组装刚度阵时全同单元的刚度阵重复利用（见第七周组会报告）
- 组装稀疏矩阵之后进行 trim 操作（见 Github opt/trim 分支）
- 标记稀疏矩阵 columns 时的算法、容器选取优化
- 组装单元刚度阵时的算法优化

C.1 组装刚度阵时的并行优化尝试

我们考虑过在组装刚度阵时进行并行优化（见第八周组会报告），虽然组装刚度阵时各单元互相解耦，对公共变量只有读的操作，但是在 scatter 这一步时，对稀疏矩阵存在写入操作，而且是 += 操作，因此可能会出现写入冲突的问题。

我们考虑过两个解决方法，一个是利用锁，但是这样多核心的优点发挥不出来，写入时会频繁访问锁，严重影响性能，失去了多线程组装刚度阵的意义。

另一个方法是将一个线程绑定为 scatter 线程，其余线程指定为组装单元刚度阵的线程。通过一个 FIFO 的队列，完成线程间的协作。这样可以通过牺牲一个组装线程来避免掉上一个方法中的访问冲突问题。但是这样做的问题是，每组装一个单元就需要 new 一个数组，这样一来如果线程较多，可能出现 scatter 线程中挤压任务，一核干活多核围观的场景。因此，这种方法我们最终也没有采用。

答辩时老师指出的分区域划分指派的方法，我们也考虑过，但是因为实现难度较大，我们没有进行深入考虑。最终我们没有将并行组装加入程序。

C.2 全同单元的刚度阵重复利用

我们还考虑了组装刚度阵时全同单元的刚度阵重复利用（见第七周组会报告）。

简单来说，因为组合体中可能出现大量尺寸相同、材料相同、朝向相同的单元（如桥面、两侧桥墩），因此对于全同的单元，我们可以重复利用他们的单元刚度阵。

为了实现这个功能，我们的想法是利用 hash 的方法。对每个元素进行 hash，将单元刚度阵挂载到 hash 表对应的地方，从而通过查询 hash 表实现重复利用刚度阵。

但是这个方法存在三个问题。

第一个问题是算法上难以对单元进行 hash。对单元来说，表征其特征的参数有 `nodes` 参数和 `elementMatrial` 参数，后者可以简单用内存地址表示（或是在单元组内用编号表示），但前者需要对每个点的 `xyz` 进行数值处理（考虑相对尺度，考虑差值），而 `xyz` 是浮点数，在读入文件时就存在一定的误差，因此难以进行针对整形的 hash 操作。如果将浮点强行转换为整形进行 hash，最坏的情况可能是全部 hash 都 miss 掉，反而成为负优化。

第二个问题是在进行 hash 时，效率存在很大的浪费。首先为了保证刚度阵组装的正确，我们必须考虑 hash 碰撞的问题，需要将 hash 表建立为数组+链表的形式，而每检测到 hash 相同，我们就必须对他们进行一次比对，确认单元全同才可以进行单刚重复利用，否则存在刚度阵组装错误的风险。而比较单元全同的时间并不是很短的，因此这存在一定的性能浪费。

第三个问题是存在很大的内存浪费。为了保存单刚，我们需要对不同的单元保存他们的单元刚度阵，因此考虑一个自由度的相关自由度是 20 的话，我们需要约 10 倍于总刚的空间来保存单刚，即不能及时 `delete` 掉 `Matrix` 数组。就算考虑不对所有单元保存单刚，维护一个计数器，对重复超过了某一个阈值的单元才保留单刚，我们也很难保证内存不炸掉。

因此，重复利用单刚在现有的输入文件的格式下很难实现。但是对于 ABAQUS 来说，他的输入格式有 part 和 instance 的概念，对这个方法能更好的适应。我们只需要保留每一个 part 的刚度阵，进行 part 级别的刚度阵变换和 scatter，就可以一定程度上避免重复刚度阵的计算。而且如果进一步考虑，在组装每一个 part 的刚度阵时，可以在 part 内部考虑单元刚度阵的重复利用。因为 part 多是规则的空间取向，点的坐标经常是整数或不存在浮点误差的浮点数（如 $0.5, 0.25, 0.75$），因此 hash 操作更好做，而且要求的储存空间更低。从这个意义上来说，我们的 dat 输入文件虽然简单，但是空间效率和计算效率都较 inp 格式的输入文件差。

C.3 稀疏矩阵的 trim 操作

我们还考虑了在组装稀疏矩阵之后进行 trim 操作（见 Github opt/trim 分支）。

具体来说，因为单刚中存在零元素，而我们在标记稀疏矩阵零元素时无法判断是否是零元素，因此在最终的总刚中这些零元素被标记为非零元，在稀疏求解器求解的部分占用了时间。因此，我们考虑在调用稀疏求解器之前进行一次 trim 操作，将这些被标记为非零元的零元素去掉。

我们虽然实现了这个算法，但是在测试的过程中发现，这个做法并不能起到减少时间的作用，反而降低了程序效率。

分析这些假非零元的数量，在 Job-1 时，假非零元的占比为 16.84%，Job-2 为 6.68%，Job-3 为 3.11%，因此单元划分越细，假非零元越少。这符合直觉：假非零元多由梁板壳梁单元等一维或二维单元产生，而随着单元划分加密， 8H 单元的数量占比三次方增大，占到绝对多数，而 8H 单刚多为满阵，因此假非零元减少。

因此，trim 操作起到的优化效果有限，而进行一次 trim 操作需要遍历一次数组，因此可能遍历操作的花销大于求解器优化时的开销。

另一个可能的原因是，在 trim 操作遍历数组时发生了 cache miss，这样会严重影响遍历的性能。由于时间和技术限制，我们没有对这方面进行进一步的优化。

C.4 其他优化

容器选择部分已经在 5.1 部分进行了阐述。

单元刚度阵组装算法的优化比较具体，包括考虑零元时的矩阵乘法简化、综合考虑到栈分配时间和乘法时间意义下的是否重复元素分配的最优算法等等。

附录 D

小组合作清单

D.1 主干工作

黄云帆(组长) 铁木辛柯梁 (减缩积分与一致插值两个版本)，截锥壳；统筹工作，组织组会。

管唯宇 3T, 4Q, 9Q 单元；前处理，稀疏求解器；效率/内存整体优化。

陈一彤 8H 单元；后处理 (基于 tecplot)；SPR (基于 8H)。

杨正宇 矩形板，平板壳；模态分析 (基于 Bar)。

邓博元 欧拉梁；无限单元，过渡单元 (基于 5Q)。

D.2 细节性工作

自动测试脚本 搭建基于 `travis.com` 的持续集成（自动测试流程）。可以利用已有的分片试验结果，在每一次 `Push` 和 `Pull Request` 时对代码进行全面的自动测试。这部分工作由管唯宇同学完成。

DOF 自动适配 在兼容已有输入格式的基础上，可以更灵活地适应不同的旋转自由度约束要求。这部分工作由邓博元、杨正宇、黄云帆、管唯宇四位同学共同完善而成。

若干 bug 修正 修正原计时模块在多线程下会出现的计时不正确 bug，修正 debug 模式下打印刚度阵会出现的访问越界 bug，修正多单元组时出现的基类数组转换 bug 等。这部分工作由管唯宇同学完成。

8H 刚度阵组装 由于此前调用 Eigen 库进行运算的效率过低，故改写为显式写法，大幅提高了效率。这部分工作由黄云帆同学在陈一彤同学原来代码的基础上稍作修改后完成。

STAP90 程序 为已有用 Fortran90 语言编写的 STAP90 程序的完善，增加了若干单元和后处理功能。这部分工作由卢晟昊同学完成。

参考文献

[1] 张雄等．计算动力学（第二版）．北京：清华大学出版社，2015．

[2] 彭细荣等．有限单元法及其应用．北京：清华大学出版社，北京交通大学出版社，2012．

[3] Zienkiewicz E C, Taylor R L．有限元方法（第一卷，基本原理）．曾攀译．北京：清华大学出版社，2008．

[4] 王勖成．有限单元法．北京：清华大学出版社，2003．

[5] 薛守义．有限单元法．北京：中国建材工业出版社，2005．

[6] 陆明万．弹性理论基础（第二版）．北京，德国：清华大学出版社，施普林格出版社，2001．

[7] Zienkiewicz O C, Cheung Y K. The Finite Element Method for Analysis of Elastic Isotropic and Orthotropic Slabs. Proc. Inst. Civ. Eng. 28:471-488, 1964.

[8] Wikipedia. Timoshenko beam theory. (2017.08.25) [2017.12.06]. https://en.wikipedia.org/wiki/Timoshenko_beam_theory.

[9] MKL reference. https://software.intel.com/en-us/mkl-developer-reference-c-spgvd#22ACC038-0667-4E23-8263-B4C0E67A769C.

[10] S.铁木辛柯，S.沃诺斯基．板壳理论.《板壳理论》翻译组译．北京：科学出版社，1977．