Python 有限元分析软件详细设计文档

魏华祎

2019 年 10 月 14 日

摘要

Python 有限元分析软件 (简称 FEALPy) 是由湘潭大学计算数学研究团队开发, 本文档首先介绍了 FEALPy 的总体设计思想和系统架构, 接着介绍了 FEALPy 的主要功能模块, 最后给出 FEALPy 的安装说明, 并给出几个偏微分方程数值求解的应用例子。

1 背景与总体设计

计算机辅助工程 (CAE) 仿真在工业设计和智能制造当中作用日益突显, 已经成为当今工业设计和智能制造的必备手段。但由于种种原因, 我国主流商用 CAE 软件大多数来自欧美发达国家, 其在中国的市场占有率高达 97 %, 而且垄断了大部分的核心技术。我国为应对国际自由贸易环境面临的重大挑战, 解决制造业大而不强, 核心技术被人卡脖子的严峻现状, 提出了中国制造 2025 计划, 对工业软件的安全性和自主性日益重视, 对国产自主 CAE 软件的发展的产业政策支持力度也日益加强, 未来数年将是国产工业软件蓬勃发展的黄金时段。

有限元算法作为 CAE 仿真中的核心算法, 最初起源于土木工程和航空工程中的弹性和结构分析问题的研究。它的发展可以追溯到 Alexander Hrennikoff(1941) 和 Richard Courant (1942) 的工作。1965 年, 中国计算数学专家冯康先生, 基于五十年代至六十年代大型水坝计算研究的实践经验, 发表了《基于变分原理的差分格式》一文, 奠定了有限元计算方法的严格数学理论, 为后世有限元计算方法的实际应用提供了理论保证。有限元算法的基本思想是把复杂的连续物理区域离散为很多简单形状单元的集合, 然后利用变分原理把连续的偏微分方程的求解问题转化为代数方程的求解。由于有限元算法对复杂区域良好的适应性, 促使其成为 CAE 仿真中应用最广泛的算法。

湘潭大学计算数学研究团队, 经过数十年的研究, 在有限元、快速算法等方面积累了丰富的研究成果, 但一直没有一个统一的数值计算软件平台来集成这些研究成果, 不但限制了团队内部合作科研和教学的效率, 也限制了这些成果的实际应用和推广。无论多么先进的算法, 如果仅停留在纸面上, 不能变成高效易用的软件, 是无法转变为真正生产力的。同时, 一个算法如果得不到实际应用的检验, 也就没有向前演化的动力。

为解决以上问题, 并对国产 CAE 软件发展和人才培养贡献一份微薄的力量, 湘潭大学计算数学研究团队在学习借鉴已有成熟有限元软件的基础之上, 开发了数值计算软件 “Python 有限元分析软件”, 简称 FEALPy, 并开源托管在 Github 上, https://github.com/weihuayi/fealpy。

检验一个数值软件优劣的标准一般有三个: 易用性、可靠性和速度。很多情况下, 易用性比另外两个标准更重要。比如做为一个科学计算算法研究者, 如果一个数值软件非常方便使用, 写很少的代码就可以快速直接验证迭代自己的想法, 这样就可以大大提高

科研的效率。因此，FEALPy 设计之初的首要目标就是方便易用，所以编程语言采用了 Python 。Python 是一门非常优秀的高级解释性通用语言，它有如下特点：

- 支持面向对象编程的解释性语言，无需编译，易学易用。
- 代码可读性非常强。
- 有庞大的用户社群，包括 NASA, ANL, Google 等国际知名的科研机构和公司都选择 Python 做为高性能计算的开发语言。Python 的初学者和开发者很容易从社群中获得帮助和开发文档。
- 有丰富的科学计算基础软件包，如：
  - NumPy: http://numpy.scipy.org - Numerical Python，主要提供多维数组及相关运算功能。
  - SciPy: http://www.scipy.org - Scientific Python，提供高效的优化、FFT、稀疏矩阵等科学计算模块。
  - Matplotlib: http://www.matplotlib.org - Graphics library，提供成熟 2D 和 3D 画图软件功能。
  - SymPy: http://www.sympy.org - 数学符号计算。
- 开源免费

一个好的数值软件，不但要用户方便使用，而且要方便开发者进行扩展和升级。因此，基于 Python 对面向对象设计的灵活支持，FEALPy 的设计完全采用面向对象的模块化设计，软件架构设计见图 1。

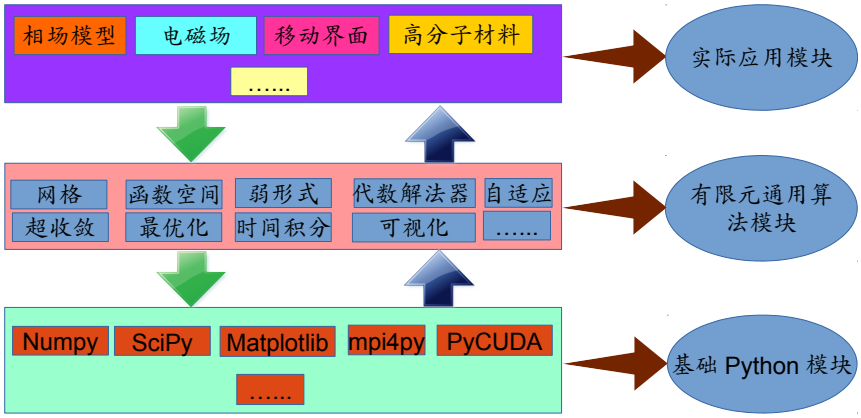

图 1： FEALPy 软件架构设计图。

其中的网格、有限元空间和弱形式等模块也是有限元算法中涉及到的标准数学对象，也就是说 FEALPy 中实现的对象是和数学对象是一一对应的。各个模块之间设计了标准的接口，因此每个模块都不依赖其它模块的具体实现，这样就可以方便地对各个模块进行替换和重构，极大地方便了 FEALPy 的扩展。

除了易用性，软件执行速度也非常重要。FEALPy 采用 Python 语言实现，可以交互使用，非常方便。但方便是要付出代价的，Python 本身也是以面向对象的理念设计的。相对于其它面向对象的语言来说，Python 走的更远，认为一切皆为对象，就连最基本的数值类型，如整型、浮点型变量都是对象。Python 在这些基本数值类型上加上了很多额外的东西，因此这些数值类型的变量参与运算时，要比编译型语言慢。如果是在很大循环

中，Python 执行数值运算的效率非常低。因此，为了避免这种效率低下的问题，FEALPy 基于 Python 的 Numpy 模块，尽量采用面向数组的方式设计核心算法，尽量避免使用循环次数很多的循环。FEALPy 可以这样做的根本原因是，有限元算法本身就是建立在矩阵向量运算的基础之上的。面向数组编程的第一个好处是，用一行代码即可实现 C 中多行代码的功能，因此可大大压缩代码缩写量，提高编程的效率。第二个好处是大大提高执行软件的速度，因为 Numpy 本身是用 C 语言实现，支持多线程计算，调用 Numpy 的功能时，其内部自动进行了多线程计算，从而充分利用了现代多核计算机的计算资源，大大提高了程序执行速度。

2 主要功能模块简介

FEALPy V1.0 目前主要包含如下功能模块：

1. 常用的网格数据结构，如二维的三角形、四边形和多边形网格，三维的四面体、六面体和多面体网格。
2. 任意次任意维的拉格朗日有限元空间。
3. 多边形网格上的二维任意次的 Virtual Element 空间。
4. 三角形和四面体网格上的胡张混合有限元空间。
5. 三角形和四面体网格上的二分自适应加密算法。
6. 四叉树和八叉树自适应加密算法。
7. 常用偏微分方程模型及有限元求解模块。

下面就分别介绍各个模块的接口和功能。

2.1 网格

在偏微分方程数值计算程序设计中，网格是最核心的数据结构，是下一步实现数值离散方法的基础。用节点数组 node 和单元数组 cell，就可以表示一个网格，其它的如边数组 edge 、面数组 face 及拓扑关系数组都可以由 cell 生成。在这里我们把 node、edge、face 和 cell 通称为网格的体 entity。下面将着重介绍 FEALPy 中的 mesh 模块，其中实现了偏微分方程数值计算中常见网格类型，见表格 2。

表 1: FEALPy 中网格类对象数据成员的命名约定

IntervalMesh	一维网格
TriangleMesh	三角形网格
QuadrangleMesh	四边形网格
TetrahedronMesh	四面体网格
HexahedronMesh	六面体网格
PolygonMesh	多边形网格
PolyhedronMesh	多面体网格
StructureQuadMesh	结构四边形网格
StructureHexMesh	结构六面体体网格
Tritree	三角形树结构网格
Quadtree	四边形树结构网格，即四叉树
Octree	六面体树结构网格，即八叉树

上面的网格类都可以通过类似于下面的语法导入：

Python code 1 FEALPy 中的网格导入语法。

```python
from fealpy.mesh import TriangleMesh
from fealpy.mesh import QuadrangleMesh
from fealpy.mesh import Tritree
```

注意上面的 StructureQuadMesh 和 StructureHexMesh 的实现与其它非结构网格类的具体实现是完全不同的，它们的实现充分考虑了结构网格的特点，并没有显式存储单元或者其它的拓扑关系数组，提高了内存使用和代码运行的效率。但它们的接口和非结构网格基本是一样的，用户在使用过程中可以用非常一致的方式调用，而不用关心底层的实现。

FEALPy 中的所有网格类的成员变量和成员函数都遵循同样的命名约定，尽量做到与维数和网格类型无关。基本成员变量的命名约定见表 2。

表 2: FEALPy 中网格类对象数据成员的命名约定。

变量名	含义
NN	节点的个数
NC	单元的个数
NE	边的个数
NF	面的个数
NCV	单元顶点的个数
NFV	面顶点的个数
GD	空间维数
TD	拓扑维数
node	节点数组，形状为 (NN, GD)
cell	单元数组，形状为 (NC, NCV)
edge	边数组，形状为 (NE, 2)
face	面数组，形状为 (NF, NFV)
ds	网格的拓扑数据结构对象，所有的拓扑关系数据都由其管理和获取
nodedata	字典对象，存储定义在节点上的数据，默认为空
celldata	字典对象，存储定义在单元上的数据，默认为空
edgedata	字典对象，存储定义在边上的数据，默认为空
facedata	字典对象，存储定义在面上的数据，默认为空
itype	网格所用的整型数据类型
ftype	网格所有的浮点型数据类型

因为多边形网格的每个单元顶点个数不一样，所以 PolygonMesh 的 cell 数组是一个一维数组，另外还增加了一个长度为 NC+1 一维数组 cellLocation，来标记每个单元在 cell 数组中的起始位置，注意这里的顶点编号是按逆时针排序。

假设有网格对象 mesh，它的常用成员函数见表格 3：

表 3: 网格对象的成员函数列表。

成员函数名	功能
mesh.geo_dimension()	获得网格的几何维数
mesh.top_dimension()	获得网格的拓扑维数
mesh.number_of_nodes()	获得网格的节点个数
mesh.number_of_cells()	获得网格的单元个数
mesh.number_of_edges()	获得网格的边个数
mesh.number_of_faces()	获得网格的面的个数
mesh.number_of_entities(etype)	获得 etype 类型实体的个数
mesh.entity(etype)	获得 etype 类型的实体
mesh.entity_measure(etype)	获得 etype 类型的实体的测度
mesh.entity_barycenter(etype)	获得 etype 类型的实体的重心
mesh.integrator(i)	获得该网格上的第 i 个积分公式
mesh.bc_to_points(bc)	转换重心坐标 bc 到实际单元上的点
mesh.add_plot(axes)	在坐标系 axes 中画出网格
mesh.find_node(axes)	在坐标系 axes 中标记出网格节点
mesh.find_edge(axes)	在坐标系 axes 中标记出网格边
mesh.find_face(axes)	在坐标系 axes 中标记出网格面
mesh.find_cell(axes)	在坐标系 axes 中标记出网格单元

上面的 etype 可以取 0, 1, 2, 3 或者字符串 cell, node, edge, face , 当然对于二维网格 etype 就不能取 3 或者 face。

网格对象 mesh 的拓扑数据结构成员变量 ds 函数接口见表 4:

表 4: 网格拓扑数据成员 ds 的接口。

成员函数名	功能
cell2cell = mesh.ds.cell_to_cell(...)	单元与单元的邻接关系
cell2face = mesh.ds.cell_to_face(...)	单元与面的邻接关系
cell2edge = mesh.ds.cell_to_edge(...)	单元与边的邻接关系
cell2node = mesh.ds.cell_to_node(...)	单元与节点的邻接关系
face2cell = mesh.ds.face_to_cell(...)	面与单元的邻接关系
face2face = mesh.ds.face_to_face(...)	面与面的邻接关系
face2edge = mesh.ds.face_to_edge(...)	面与边的邻接关系
face2node = mesh.ds.face_to_node(...)	面与节点的邻接关系
edge2cell = mesh.ds.edge_to_cell(...)	边与单元的邻接关系
edge2face = mesh.ds.edge_to_face(...)	边与面的邻接关系
edge2edge = mesh.ds.edge_to_edge(...)	边与边的邻接关系
edge2node = mesh.ds.edge_to_node(...)	边与节点的邻接关系
node2cell = mesh.ds.node_to_cell(...)	节点与单元的邻接关
node2face = mesh.ds.node_to_face(...)	节点与面的邻接关系
node2edge = mesh.ds.node_to_edge(...)	节点与边的邻接关系
node2node = mesh.ds.node_to_node(...)	节点与节点的邻接关
isBdNode = mesh.ds.boundary_node_flag()	一维逻辑数组, 标记边界节点
isBdEdge = mesh.ds.boundary_edge_flag()	一维逻辑数组, 标记边界边
isBdFace = mesh.ds.boundary_face_flag()	一维逻辑数组, 标记边界面
isBdCell = mesh.ds.boundary_cell_flag()	一维逻辑数组, 标记边界单元
bdNodeIdx = mesh.ds.boundary_node_index()	一维整数数组, 边界节点全局编号
bdEdgeIdx = mesh.ds.boundary_edge_index()	一维整数数组, 边界边全局编号
bdFaceIdx = mesh.ds.boundary_face_index()	一维整数数组, 边界面全局编号
bdCellIdx = mesh.ds.boundary_cell_index()	一维整数数组, 边界单元全局编号

注意，上面接口中的省略号表示有默认参数，用户可以根据需要来设定。也要注意返回的实体间邻接关系，默认可能是二维数组，也可能是稀疏矩阵。在使用过程中，用户要根据网格的类型或者实际需要，来控制返回邻接关系的数据类型。

上面介绍的接口只是一部分常用接口，更多接口用户可以自己去代码中探索发现。最后，给出一段代码来展示 FEALPy 中网格对象的基本用法，网格的显示图形见图 2。

Python code 2 FEALPy 中的网格对象用法示例。

```python
import numpy as np
import matplotlib.pyplot as plt
from fealpy.mesh import TriangleMesh

# [0, 1]^2 区域上的网格
node = np.array([
    (0, 0),
    (1, 0),
    (1, 1),
    (0, 1)], dtype=np.float)
cell = np.array([
    (1, 2, 0),
    (3, 0, 2)], dtype=np.int)
# 建立三角形网格对象
tmesh = TriangleMesh(node, cell)
# 一致加密一次
tmesh.uniform_refine()
# 获得节点数组
node = tmesh.entity('node')
# 获得边数组
edge = tmesh.entity('edge')
# 获得单元数组
cell = tmesh.entity('cell')
# 获得单元的面积
cellArea = tmesh.entity_measure('cell')
# 获得边的长度
edgeLength = tmesh.entity_measure('edge')
# 获得单元的重心
cellBC = tmesh.entity_barycenter('cell')
# 获得边的重心
edgeBC = tmesh.entity_barycenter('edge')
# 获得每条边的单位法向和切向
n, t = tmesh.edge_frame()
# 获得单元的邻接关系数组, 形状为 (NE, 3)
cell2cell = tmesh.ds.cell_to_cell()
# 获得边与单元的邻接关系数组,形状为 (NE, 4)
edge2cell = tmesh.ds.edge_to_cell()
# 打印网格的信息
tmesh.print()
# 建立画图对象
fig = plt.figure()
# 获得坐标系
axes = fig.gca()
# 在坐标系中画网格
tmesh.add_plot(axes)
# 显示所有节点编号
tmesh.find_node(axes, showindex=True)
# 显示所有边的编号
tmesh.find_edge(axes, showindex=True)
# 显示所有单元的编号
tmesh.find_cell(axes, showindex=True)
plt.show()
```

2.2 函数空间模块

在网格模块的基础之上，就可以建立有限维的函数空间模块。FEALPy 中最常用的有限维函数空间类为 LagrangeFiniteElementSapce 类，它是有限元中任意 p 次拉

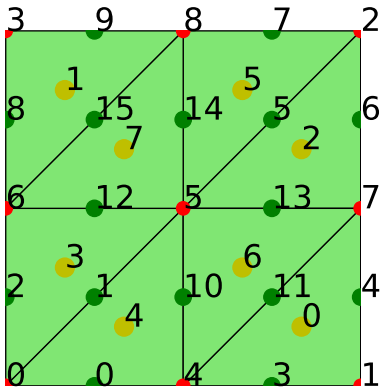

图 2: 三角形网格示意图, 其中红色标记 node, 绿色标记 edge, 黄色标记 cell。

格朗日空间的实现, 该类的数据成员见表 5。

表 5: LagrangeFiniteElementSapce 的数据成员

mesh	网格对象, 可以为任意维的单纯型网格, 如一维的区间、二维的三角形和三维的四面体网格
p	空间的次数
GD	几何空间的维数
TD	几何空间的拓扑维数
dof	函数空间的自由度管理对象, 用于管理函数空间的自由度
spacetype	函数空间的类型, 取值'C' 表示空间是分片连续的, 取值'D' 表示空间是间断的
ftype	函数空间所用的浮点数类型
itype	函数空间所用的整数类型

LagrangeFiniteElementSapce 的函数成员见表格 6。

表 6: LagrangeFiniteElementSapce 的函数成员

space = LagrangeFiniteElementSapce(mesh, p)	创建一个 p 次的拉格朗日有限元空间对象
ldof = space.number_of_local_dofs()	获得每个单元上的自由度个数
gdof = space.number_of_global_dofs()	获得所有单元上的总自由度个数
cell2dof = sapce.cell_to_dof()	获得单元与自由度的对应关系数组
bdof = space.boundary_dof()	获得边界的自由度
GD = space.geo_dimension()	获取函数空间的几何维数
TD = space.top_dimension()	获取函数空间的拓扑维数
phi = space.edge_basis(bc, cellidx, lidx)	计算边上积分点的基函数值
phi = space.basis(bc)	计算单元上积分点处的基函数值
gphi = space.grad_basis(bc)	计算单元上积分点处的基函数梯度值
val = space.value(uh, bc)	计算自由度值为 uh 的函数在积分点 bc 处的值
val = space.grad_value(uh, bc)	计算自由度值 uh 的函数在积分点 bc 处的梯度值
uI = space.interpolation(u)	计算函数 u 在有限元空间中的插值
uh = space.function()	创建一个有限元函数对象
A = space.stiff_matrix(...)	构造刚度矩阵
M = space.mass_matrix(...)	构造质量矩阵
F = space.source_vector(.. .)	构造源项向量

下面给出一个有限元空间的创建例子:

Python code 3 FEALPy 中的拉格朗日有限元函数空间。

```python
import numpy as np
import matplotlib.pyplot as plt
from fealpy.mesh import TriangleMesh
from fealpy.functionspace import LagrangeFiniteElementSapce

# [0, 1]^2 区域上的网格
node = np.array([
    (0, 0),
    (1, 0),
    (1, 1),
    (0, 1)], dtype=np.float)
cell = np.array([
    (1, 2, 0),
    (3, 0, 2)], dtype=np.int)
# 建立三角形网格对象
tmesh = TriangleMesh(node, cell)
# 一致加密一次
tmesh.uniform_refine()

p = 2
space = LagrangeFiniteElementSapce(tmesh, p)
```

2.3 数值积分模块

数值积分是进行有限元计算的必备工具, 所以 FEALPy 中对也加入了比较丰富的数值积分公式, 见表格 7.

表 7: FEALPy 中的数值积分公式。

GaussLobattoQuadrature	高斯 Labatto 积分公式
GaussLegendreQuadrature	高斯 Legendre 积分公式
IntervalQuadrature	区间单元上的积分公式
TriangleQuadrature	三角形单元上的积分公式
TetrahedronQuadrature	四面体单元上的积分公式
QuadrangleQuadrature	四边形单元上的积分公式
HexahedronMeshQuadrature	六面体网格上的积分公式
FEMeshIntegralAlg	有限元网格上的积分算法
PolygonMeshIntegralAlg	多边形网格上的积分算法

FEALPy 中的积分公式对象都包含一个二维数组 quadpts 和一维数组 weights, 其中 quadpts 是积分点数组, 每一行存储一个积分点对应的重心坐标。注意对于区间、三角形和四面体来说, 重心坐标分别有 2、3 和 4 个分量。对于四边形单元来说, 重心坐标就有 4 个分量, 六面体单元有 8 个分量。weights 的长度和 quadpts 行数一样, 且所有元素求和为 1。

这些积分公式对象, 用户可通过 mesh.integrator(index) 创建, 也就是说 mesh 对象来负责选择积分公式。

另外，FEALPy 还提供了两个积分算法类 FEMeshIntegralAlg 和 PolygonMeshIntegralAlg 专门用来计算各种类型函数积分。

下面给出一个积分使用例子

Python code 4 FEALPy 中的积分公式使用示例。

```python
import numpy as np
import matplotlib.pyplot as plt
from fealpy.mesh import TriangleMesh

def u(p):
    x = p[..., 0]
    y = p[..., 1]
    val = np.sin(np.pi*x)*np.cos(np.pi*y)
    return val

p = 2 # 有限元空间的维数
q = 5 # 积分公式的编号,编号越大积分精度越高
# [0, 1]^2 区域上的网格
node = np.array([
    (0, 0),
    (1, 0),
    (1, 1),
    (0, 1)], dtype=np.float)
cell = np.array([
    (1, 2, 0),
    (3, 0, 2)], dtype=np.int)

# 建立三角形网格对象
tmesh = TriangleMesh(node, cell)
# 计算三角形单元面积
area = tmesh.entity_measure('cell')
# 获取积分公式
integrator = tmesh.integrator(q)

# 创建 p 次分片连续的拉格朗日有限元空间
space = LagrangeFiniteElementSapce(tmesh, p)

# 计算函数 u 在 space 中的插值函数
uI = space.interpolation(u)

# 创建积分算法对象
alg = FEMeshIntegralAlg(integrator, mesh, area)

# 计算 L2 插值误差
L2 = alg.L2_error(u, uI)
```

2.4 偏微分方程模型模块

在偏微分方程数值方法的研究中，经常用到一些标准的模型来检测数值方法的优劣。为了便于研究者快速验证算法，给出数值实验结果，FEALPy 中集成了很多常用的典型偏微分方程模型，比如

- 1 维到 3 维的 Poisson 方程模型
- 2 维和 3 维的线弹性方程模型
- 2 维的四阶 Biharmonic 的方程模型

用户可以方便地通过下面的方式导入这些模型：

Python code 5 FEALPy 中的偏微分方程模型示例。

```
from fealpy.pde.poisson_1d import CosData as PDE
from fealpy.pde.poisson_2d import CosCosData as PDE
from fealpy.pde.poisson_3d import CosCosCosData as PDE
```

3 安装与使用

因为 FEALPy 完全基于 Python 语言开发, 天然具有跨平台的特点。下面分别介绍不同系统下的安装方式。

在 Ubuntu Linux 系统下, 安装过程如下:

```
$ sudo apt install git              # 版本控制软件
$ sudo apt install python3          # Python 3 解释器
$ sudo apt install python3-pip      # Python 3 软件包管理器
$ sudo apt install python3-tk       # 画图模块的前端
$ git clone https://github.com/weihuayi/fealpy.git # 克隆最新的 FEALPy
$ cd fealpy
$ sudo -H pip3 install -e ./ # 安装 fealpy 到系统中
```

注意上面的安装是软安装, 即只是建立一个符号链接, 这样每次只要更新了 fealpy, 在其它的文件路径下就可以使用最新版本的 FEALPy, 更新的方式如下:

```
$ cd fealpy
$ git pull # 拉取最新的 FEALPy 更新
```

在 Windows 或者 MacOS 系统下, 最好先安装 Python 的 Anaconda 集成开发环境, 下载地址为:https://www.anaconda.com/distribution/。

下载安装 Anaconda 后, 再下载最新的 FEALPy 并解压到目标位置, 下载地址为 https://github.com/weihuayi/fealpy/archive/master.zip。最后打开 Anaconda 的 Prompt 命令行终端, 输入以下命令

```
$ cd fealpy-master
$ pip install -e ./
```

即可安装 FEALpy 到系统中。

下面给出几个利用 FEALPy 求解偏微分方程的例子, 并给出运行结果。

Python code 6 求解 Poisson 问题的演示代码，文件 fealpy/example/PoissonFEMRate.py。

```python
import sys
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as sio

# 导入 Poisson 有限元模型
from fealpy.fem.PoissonFEMModel import PoissonFEMModel

from fealpy.tools.show import showmultirate, show_error_table

# 问题维数
d = int(sys.argv[1])

if d == 1:
    from fealpy.pde.poisson_1d import CosData as PDE
elif d==2:
    from fealpy.pde.poisson_2d import CosCosData as PDE
elif d==3:
    from fealpy.pde.poisson_3d import CosCosCosData as PDE


p = int(sys.argv[2]) # 有限元空间的次数
n = int(sys.argv[3]) # 初始网格的加密次数
maxit = int(sys.argv[4]) # 迭代加密的次数

pde = PDE() # 创建 pde 模型

# 误差类型与误差存储数组
errorType = ['$|| u - u_h||_0$', '$||\\nabla u - \\nabla u_h||_0$']
errorMatrix = np.zeros((len(errorType), maxit), dtype=np.float)

# 自由度数组
Ndof = np.zeros(maxit, dtype=np.int)

# 创建初始网格对象
mesh = pde.init_mesh(n)

for i in range(maxit):
    fem = PoissonFEMModel(pde, mesh, p, p+2) # 创建 Poisson 有限元模型
    ls = fem.solve() # 求解
    Ndof[i] = fem.space.number_of_global_dofs() # 获得空间自由度个数
    errorMatrix[0, i] = fem.get_L2_error() # 计算 L2 误差
    errorMatrix[1, i] = fem.get_H1_error() # 计算 H1 误差
    if i < maxit - 1:
        mesh.uniform_refine() # 一致加密网格

# 显示误差
show_error_table(Ndof, errorType, errorMatrix)
# 可视化误差收敛阶
showmultirate(plt, 0, Ndof, errorMatrix, errorType)
plt.show()
```

在 FEALPy 的目录下, 进入 example 目录, 运行 PoissonFEMRate.py 文件可测试不同维数和次数下 Poisssn 模型的有限元计算结果。

下面是针对一个 1 维问题, 用 1 次有限元求解, 初始网格加密 3 次后, 再迭代 4 次, 在迭代过程中依次加密的网格上求解有限元解, 并给出真解和有限元解的 L_2 和 H_1 收敛率, 结果与理论结果一致。

```
$ cd fealpy-master/example
$ python3 PoissonFEMRate.py 1 1 3 4
Construct linear system time: 0.0013978090000819066
Solve time: 0.00020607599981303792
Construct linear system time: 0.0011339430002408335
Solve time: 0.00015868700029386673
Construct linear system time: 0.0011290240004200314
Solve time: 0.000165621000178362
Construct linear system time: 0.007911985000191635
Solve time: 0.00023100899989003665
\begin{table}[!htdp]
\begin{tabular}[c]{|c|c|c|c|c|}\\\hline
Dof &  9 & 17 & 33 & 65
$|| u - u_h||_0$ & 9.9214e-03 & 2.4865e-03 & 6.2202e-04 & 1.5553e-04
Order & — & 2. & 2. & 2.
$||\nabla u - \nabla u_h||_0$ & 2.5118e-01 & 1.2583e-01 & 6.2947e-02 & 3.1477e-02
Order & — & 1. & 1. & 1.
\end{tabular}
\end{table}
```

下面是针对一个 1 维问题, 用 4 次有限元求解, 初始网格加密 3 次后, 再迭代 4 次, 在迭代过程中依次加密的网格上求解有限元解, 并给出真解和有限元解的 L_2 和 H_1 收敛率, 结果与理论结果一致。

```
$ python3 PoissonFEMRate.py 1 4 3 4
Construct linear system time: 0.001463798999793653
Solve time: 0.0002590839999356831
Construct linear system time: 0.0012664870000662631
Solve time: 0.00024567799982833094
Construct linear system time: 0.0012917519998154603
Solve time: 0.00042058600001837476
Construct linear system time: 0.0014157900000100199
Solve time: 0.0004810600003111176
\begin{table}[!htdp]
\begin{tabular}[c]{|c|c|c|c|c|}\\\hline
Dof &  33 &  65 & 129 & 257
$|| u - u_h||_0$ & 1.0543e-07 & 3.2982e-09 & 1.0310e-10 & 3.2679e-12
Order & — & 5.    & 5.    & 4.98
$||\nabla u - \nabla u_h||_0$ & 1.0466e-05 & 6.5487e-07 & 4.0941e-08 & 2.5590e-09
Order & — & 4. & 4. & 4.
\end{tabular}
\end{table}
```

下面是针对一个 2 维问题, 用 1 次有限元求解, 初始网格加密 3 次后, 再迭代 4 次, 在迭代过程中依次加密的网格上求解有限元解, 并给出真解和有限元解的 L_2 和 H_1 收敛率, 结果与理论结果一致。

```
$ ./PoissonFEMRate.py 2 1 3 4
Construct linear system time: 0.003660057000161032
Solve time: 0.00032057899989013094
Construct linear system time: 0.015442209999946499
Solve time: 0.0009926980001182528
Construct linear system time: 0.03358073199979117
Solve time: 0.004240456999923481
Construct linear system time: 0.07656500299981417
Solve time: 0.015677019000122527
\begin{table}[!htdp]
\begin{tabular}[c]{|c|c|c|c|c|}\\\hline
Dof &   81 &  289 & 1089 & 4225
$|| u - u_h||_0$ & 1.9408e-02 & 4.9543e-03 & 1.2452e-03 & 3.1173e-04
```

```
Order & — & 1.97 & 1.99 & 2.
$||\nabla u - \nabla u_h||_0$ & 4.3180e-01 & 2.1754e-01 & 1.0898e-01 & 5.4514e-02
Order & — & 0.99 & 1.    & 1.
\end{tabular}
\end{table}
```

下面是针对一个 2 维问题, 用 3 次有限元求解, 初始网格加密 3 次后, 再迭代 4 次, 在迭代过程中依次加密的网格上求解有限元解, 并给出真解和有限元解的 L_2 和 H_1 收敛率, 结果与理论结果一致。

```
$ ./PoissonFEMRate.py 2 3 3 4
Construct linear system time: 0.018175761000293278
Solve time: 0.002668634000201564
Construct linear system time: 0.06449292999968748
Solve time: 0.011512577999837958
Construct linear system time: 0.15427415300018765
Solve time: 0.1347163990003537
Construct linear system time: 0.6181561340004009
Solve time: 1.29351221800016
\begin{table}[!htdp]
\begin{tabular}[c]{|c|c|c|c|c|}\\\hline
Dof &   625 &  2401 &  9409 & 37249
$|| u - u_h||_0$ & 2.0314e-05 & 1.2297e-06 & 7.5597e-08 & 4.6864e-09
Order & — & 4.05 & 4.02 & 4.01
$||\nabla u - \nabla u_h||_0$ & 1.6634e-03 & 2.0664e-04 & 2.5722e-05 & 3.2079e-06
Order & — & 3.01 & 3.01 & 3.
\end{tabular}
\end{table}
```

下面是针对一个 3 维问题, 用 1 次有限元求解, 初始网格加密 2 次后, 再迭代 4 次, 在迭代过程中依次加密的网格上求解有限元解, 并给出真解和有限元解的 L_2 和 H_1 收敛率, 结果与理论结果一致。

```
$ ./PoissonFEMRate.py 3 1 2 4
Construct linear system time: 0.016805515000669402
Solve time: 0.0010303139988536714
Construct linear system time: 0.06578156800060242
Solve time: 0.0018806569987646071
Construct linear system time: 0.5245011309998517
Solve time: 0.14901864800049225
Construct linear system time: 4.276179997999861
Solve time: 17.404089397999996
\begin{table}[!htdp]
\begin{tabular}[c]{|c|c|c|c|c|}\\\hline
Dof &   125 &   729 &  4913 & 35937
$|| u - u_h||_0$ & 5.7403e-01 & 2.1725e-01 & 6.4566e-02 & 1.7064e-02
Order & — & 1.4  & 1.75 & 1.92
$||\nabla u - \nabla u_h||_0$ & 4.4783e+00 & 2.6304e+00 & 1.4046e+00 & 7.2094e-01
Order & — & 0.77 & 0.91 & 0.96
\end{tabular}
\end{table}
```

下面是针对一个 3 维问题, 用 2 次有限元求解, 初始网格加密 1 次后, 再迭代 4 次, 在迭代过程中依次加密的网格上求解有限元解, 并给出真解和有限元解的 L_2 和 H_1 收敛率, 结果与理论结果一致。

```
$ ./PoissonFEMRate.py 3 2 1 4
Construct linear system time: 0.010831390000021202
Solve time: 0.00026890499975706916
Construct linear system time: 0.07217314900117344
Solve time: 0.002281274999404559
Construct linear system time: 0.33482521299993095
Solve time: 0.25496843200016883
Construct linear system time: 2.665320297001017
Solve time: 30.768967505000546
\begin{table}[!htdp]
\begin{tabular}[c]{|c|c|c|c|c|}\\\hline
```

```
Dof &    125 &    729 &   4913 & 35937
$|| u - u_h||_0$ & 5.3174e-01 & 1.2649e-01 & 1.6787e-02 & 2.0515e-03
Order & -- & 2.07 & 2.91 & 3.03
$||\nabla u - \nabla u_h||_0$ & 3.4937e+00 & 1.6013e+00 & 4.8751e-01 & 1.3138e-01
Order & -- & 1.13 & 1.72 & 1.89
\end{tabular}
\end{table}
```

4 总结

本文重点介绍了 FEALPy 的开发动机、基本设计原则、软件架构及关键的功能模块，最后还给出了不同空间维数下用不同次数有限元求解 Poisson 方程的数值算例。另外，FEALPy 已经深入应用到湘潭大学计算数学科研团队的教学科研中，大大提高了团队的科研和教学效率。但 FEALPy 在快速算法和文档方面还需要进一步的完善，这是下一步的目标。