# 一、获资助以来已开展的工作及取得的主要进展

# 简化版本:

软件开发方面, 项目组一直在进行高效易用有限元软件包的设计开发工作, 目前已经完成初步的框架设计, 开发工作也在持续进行中. 我们主要采用 Python 和 C++ 两种语言进行开发. 编程主要采用面向数组(Array-oriented) 和 面向对象(Object-oriented)的方式. 面向数组的方式更适合更适合计算数学科研人员用简单的代码写出更高效的数值实验程序. 而面向对象的编程方式, 可以使软件更加模块化, 易于扩展的维护.

现在我们的软件包已经实现了如下的功能:

* 简单二维区域上的结构和非结构网格生成与一致加密, 其中区域可用隐函数的形式表示.
* 三角形网格上任意次 Lagrangian 有限元空间
* Laplace 算子的和方程右端源项的弱形式
* 四阶问题的恢复型弱形式
* Dirichlet 边界条件的处理
* 与代数多重网格解法器 pyamg 的接口

软件开发方面, 项目组一直在进行高效易用有限元软件包的设计开发工作, 目前已经完成初步的框架设计, 开发工作也在持续进行中.

这里选择 Python 主要有以下几方面原因. 首先, Python 是一门非常简洁的面向对象的解释性语言, 做为软件的主要界面语言, 简单易用, 无需编译, 可快速开发出面向对象的算法模块, 且易于维护, 优化, 扩展和重用. 其次, Python 有很大的用户社群, 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. 另外, Python 已经 有大量适合科学计算的软件模块, 如

* NumPy: http://numpy.scipy.org - Numerical Python
* SciPy: http://www.scipy.org - Scientific Python
* Matplotlib: http://www.matplotlib.org - Graphics library

上面这三个 Python 软件模块会在我们设计的软件包中大量使用. 其中, NumPy 提供了

1. 任意维数的数组对象 ndarray
2. 快速高效的数组运算函数
3. 线性代数, Fourier 变换和随机数生成.

SciPy 在 NumPy 的基础之上提供了高效的优化, FFT, 稀疏矩阵等科学计算模块. 而 Matplotlib 是 Python 中的成熟 2D 和 3D 画图软件包.

我们将主要在上述三个软件包的基础上开发我们的有限元软件包. 编程方式采用面向 数组(Array-oriented)和面向对象(Object-edoriented)相结合的方式. 面向数组的编程方式会最大化利用 NumPy 提供的向量化操作函数对同类数据进行并行运算操作, 程序中可避免大量循环的使用, 充分发挥现代计算机的多线程并发执行功能, 大大提高程序的运算效率. 这种面向数组的编程方式与数学中的矩阵向量运算相一致, 更适合计算数学科研人员用简单的代码写出更高效的数值实验程序. 而面向对象的编程方式, 可以使软件更加模块化, 易于扩展的维护.

对于难于向量化的操作, Python 执行起来效率低下, 但 Python 又 很容易和其它的编译性语言结合起来, 如 Fortran, C 和 C++, 这些语言也是高性能科学计算的主要语言. 对于难于向量化的操作我们就用 C++ 实现, 再用 Python 接口调用.

最后, Python 对进程和线程的并行处理, 进程间通信(MPI), 和 GPU (如 OpenCL 和 CUDA) 都有很好的支持, 因此也适合在高性能并行计算集群上使用. 我们也会开相应的并行有限元程序.

相比于 Matlab 来说, 使用 Python 不需要购买许可, 可节省大量科研资金.

在代码设计中, 我们用面向有限元数学对象的方式组织程序模块, 如网格, 有限元空间, 弱形式, 边界条件等, 这很适合有限元方向的学生和研究者快速编写有限元的数值计算程序, 而不用考虑编译安装的问题, 同时数学对象背后的算法都以面向数组的方式实现, 这又保证了程序的高效性.

我们的设计借鉴了 Fenics [3] 和 IFEM [4] 成熟的编程编程思想和接口, 如接口模仿了 Fenics 的面向有限元数学对象接口方式和 IFEM中的面向数组的算法实现方式. 我们的设计与 Fenics 的不同在于, 我们实现主要基于 Numpy 和 SciPy 来实现有限元的算法, 而不是像Fenics 实现是完全基于 C++ 的, Python 只是用来做接口, 所以我们的设计更简单轻量化, 更易于安装, 使用和扩展, 同时还能保证足够的效率. 另外, Fenics 是一款强调有限元代码自动生成的软件, 封装了有限元实现过程的很多关键细节, 这对学生深刻理解有限元的程序实现过程造成了很大的障碍. 而 ifem 虽然是面向数组的编程方式, 但它整个软件的设计是面向过程的, 代码组织不够清晰明了, 不易于学习的使用.

现在我们的软件包已经实现了如下的功能:

* 简单二维区域上的结构和非结构网格生成与一致加密, 其中区域可用隐函数的形式表示.
* 三角形网格上任意次 Lagrangian 有限元空间
* Laplace 算子的和方程右端源项的弱形式
* 四阶问题的恢复型弱形式
* Dirichlet 边界条件的处理
* 与代数多重网格解法器 pyamg 的接口

下面是求解 Possion 方程示例代码:

mesh = squaremesh(0, 1, 0, 1, refine=3) # 获得区域 [0,1]^2 的初始三角形网格  
maxit = 6 # 网格加密 6 次,   
error = np.zeros((maxit,), dtype=np.float) # 存储每层网格上的误差  
Ndof = np.zeros((maxit,), dtype=np.int) # 存储每层网格上的自由度个数  
  
for i in range(maxit):  
 V = FunctionSpace(mesh, 'Lagrange', 3) # 获得网格上的 3 次 Lagrange 有限元空间   
 Ndof[i] = V.number\_of\_global\_dofs() # 获得空间的自由度个数   
 a = LaplaceSymetricForm(V, 6) # 获得 Lapalce 算子的弱形式  
 L = SourceForm(V, f, 6) # 获得方程右端源项弱形式  
  
 BC = DirichletBC(V, gd, isBoundaryDof) # 生成 Dirichlet 边界处理对象   
 uh = solve(a, L, BC, 'amg') # 用 amg 方法求解   
 eu = V.interpolation(u) # 把真解插值到有限元空间   
 error[i] = np.sqrt(np.sum((eu - uh)\*\*2)/Ndof[i]) # 计算 l\_2 误差  
  
 if i < maxit-1:  
 mesh.uniform\_refine() # 加密网格为进行下一步计算准备网格   
  
print(error)

[1] https://www.python.org

[2] <https://en.wikipedia.org/wiki/Array_programming>

[3] https://fenicsproject.org/

[4] <http://www.math.uci.edu/~chenlong/programming.html>

# 二、下一步工作设想

目前, 我们的软件包的设计和实现还很初步, 还有很多优化和改进的空间, 下一步我们会为增加更多的功能, 如:

* 二维网格的二分法自适应加密算法
* 支持更多的网格类型, 如四边形, 多边形, 四面体和多面体网格的网格类型
* 网格优化功能
* 实现更多的有限元空间, 如混合有限元空间, 虚单元法有限元空间等

并在此基础上, 针对相场模型, 设计相应的可重用的程序模块, 更好支持本项目的研究工作的进行.

# 三、存在的问题和建议

无

**四、已发表论文目录**

**无**