三维并行自适应有限元软件平台 PHG 简易教程

张林波
中科院数学与系统科学研究院
zlb@lsec.cc.ac.cn

2010 年 4 月 15 日

目　录

第一章　PHG 的编译及安装 **4**
1.1 PHG 简介 . . . 4
1.2 主要功能 . . . 4
1.3 PHG 的下载与配置、编译、安装 . . . 4
1.4 可选第三方软件 . . . 5
1.5 BLAS/LAPACK . . . 6
1.6 线性解法器 . . . 6
1.7 特征值解法器 . . . 7
1.8 可视化软件 . . . 7
1.9 网格文件格式 . . . 8
1.10 编译、运行测试程序 . . . 8

第二章　基础知识 **9**
2.1 PHG 程序基本结构 . . . 9
2.2 三角形、四面体网格的二分自适应细化与粗化 . . . 9
2.3 层次网格结构 . . . 9
2.4 网格剖分与分布式层次网格结构 . . . 9

第三章　基本数据类型与网格管理 **10**
3.1 基本数据类型及常量 . . . 10
3.2 单元对象 . . . 10
3.3 网格对象 . . . 11
3.4 网格导入与销毁 . . . 12
3.5 网格导出 . . . 12
3.6 网格单元遍历 . . . 12
3.7 网格剖分及负载平衡 . . . 12
3.8 网格的加密与放粗 . . . 13

第四章　自由度管理与有限元基函数 **14**
4.1 自由度类型与有限元基函数 . . . 14
4.2 自由度对象 . . . 15
4.3 创建、导出和销毁自由度对象 . . . 16
4.4 自由度编号 . . . 16
4.5 直接访问自由度数据 . . . 16
4.6 自由度对象的运算 . . . 17
4.7 有限元函数求值 . . . 17
4.8 自由度对象的模 . . . 18
4.9 自由度对象的微分 . . . 18
4.10 特殊自由度对象 . . . 18
4.11 访问邻居单元的自由度数据 . . . 19
4.12 几何量自由度对象 . . . 19

第五章　数值积分 **21**
5.1 有限元计算中的数值积分 . . . 21
5.2 数值积分公式 . . . 21
5.3 对称型数值积分公式 . . . 21
5.4 PHG 的数值积分公式 . . . 22
5.5 双、三线性型的计算 . . . 23
5.6 计算面跳量 . . . 24
5.7 基函数、函数值的 cache . . . 24

第六章　命令行选项 **26**
6.1 自定义命令行选项 . . . 27
6.2 获取命令行参数的当前值 . . . 27
6.3 动态设置命令行选项的参数值 . . . 27
6.4 显示命令行及命令行选项 . . . 28

第七章　线性解法器 **29**
7.1 线性解法器对象 . . . 29
7.2 创建和销毁解法器对象 . . . 29
7.3 组装线性系统的系数矩阵及右端项 . . . 29
7.4 Dirichlet 边界条件 . . . 30
7.5 用同一个系数矩阵反复求解不同右端项 . . . 30
7.6 PHG 支持的解法器 . . . 30

第八章　矩阵及向量操作 **31**
8.1 MAP . . . 31
8.2 向量 (VEC) . . . 33
8.3 矩阵 (MAT) . . . 34
8.4 MAT、VEC 与解法器 . . . 35

第九章　特征值、特征向量计算 **36**
9.1 特征值问题求解器 . . . 36

附录一　程序实例 **37**
A.1 Poisson 方程 . . . 37
A.2 时谐 Maxwell 方程 . . . 38
A.3 特征值问题 . . . 39

第一章　PHG 的编译及安装

§1.1　PHG 简介

PHG: Parallel Hierarchical Grid

软件著作权证书：2006SRBJ1700

主页：`http://lsec.cc.ac.cn/phg`

PHG 是“科学与工程计算国家重点实验室”发展的一个三维并行自适应有限元软件开发平台，并且得到了下述项目的支持：

- 国家 973 项目“大规模科学计算研究”第二课题组
  “大规模并行计算研究”(2005CB321702)
- 国家自然科学基金重点项目
  “若干非线性问题的数值解法”(10531080)
- 国家自然科学基金面上项目
  “并行自适应电磁场有限元计算算法及软件研究”(60873177)

PHG 采用 C 语言开发，基于 MPI 消息传递通信并行

§1.2　主要功能

- 面向二维三角形单元或三维四面体单元协调网格
- 基于单元二分的网格局部加密与放粗
- 基于网格剖分的并行实现、动态负载平衡
- 自由度管理、有限元基函数、数值积分
- 隐藏网格及并行细节
- 线性解法器、特征值解法器接口
- Tcl/Tk 脚本，VTK 接口
- 可视化文件输出

§1.3　PHG 的下载与配置、编译、安装

源码下载地址：`http://lsec.cc.ac.cn/phg/download.htm`

编译、安装：

```
tar xjpvf phg-x.x.x-xxxxxxxx.tar.bz2
cd phg-x.x.x
./configure
gmake
gmake install
```

必要时需使用一些选项来指定某些软件的位置。运行 `./configure --help` 将显示出所有选项。

如果发生问题，可在运行 `./configure` 后查看文件 `config.log` 来找出某些模块配置不成功的原因。

`configure` 选项的例子：

```
env FC="f77 -64" F77="f77 -64" CC="gcc -mabi=64" \
   CXX="g++ -mabi=64 -DMPI_NO_CPPBIND" \
   LDFLAGS="-L/usr/freeware/lib64 -L/usr/local/lib64" \
   ./configure --with-fc-libs="-lfortran -lftn" \
     --libdir=/usr/local/lib64 \
     --with-mumps-lib="-ldmumps -lpord -L/usr/local/SCALAPACK \
                       -lscalapack-64 -L/usr/local/BLACS/LIB64 \
                       -lblacs -lblacsF77init -lblacs -lpthread" \
     --with-superlu-incdir=/usr/local/superlu_dist/include \
     --with-superlu-lib="-L/usr/local/superlu_dist/lib64 \
                         -lsuperlu_dist_2.0"\
     --with-hypre-dir=/usr/local/hypre64 \
     --with-tcl-incdir=/usr/freeware/include \
     --with-tcl-libdir=/usr/freeware/lib64 \
     --with-metis-libdir=/usr/local/parmetis/lib64 \
     --with-metis-incdir=/usr/local/parmetis/include \
     --with-vtk-libdir=/usr/local/lib64/vtk \
     --with-vtk-incdir=/usr/local/include/vtk \
     --with-vtk-vtktcl=/usr/local/lib/vtk/tcl/vtktcl.c \
```

§1.4 可选第三方软件

§1.4.1 MPI

`configure` 缺省使用 `mpicc`、`mpicxx` (或 `mpiCC`) 编译。可用环境变量 `CC` 和 `CXX` 指定 C 和 C++ 编译器，用选项 `--with-mpi-libdir` 或 `--with-mpi-lib` 以及 `--with-mpi-incdir` 分别指定 MPI 库和头文件。

由于许多第三方软件需要 MPI 支持，因此即使编译 PHG 的串行版本时最好也使用支持 MPI 的编译系统。

§1.4.2 网格剖分：METIS/ParMETIS

从 `ftp://159.226.92.111/pub/RPMS/` 中下载、安装 `parmetis` 包。

注：这些 RPM 包修改了 `metis.h` 和 `parmetis.h`，将 METIS/ParMETIS 的其余头文件放在名为 `metis` 的下一级子目录中，以避免与其它软件包的头文件名发生冲突。

§1.4.3 脚本接口：Tcl/Tk

包含在 Linux 发布版中 (Fedora: `tcl-devel` 和 `tk-devel` 包)

§1.4.4 可视化接口：VTK

直接编译、安装网上下载的 cmake 和 VTK 源码或 RPM 包:

`http://www.vtk.org/`

只要程序 `cmake` 位于可执行文件的搜索路径中，`configure` 便可通过它找到 VTK 的配置。

注：最新 PHG 版本已经不需要 cmake。

§1.4.5 性能统计：PAPI

http://icl.cs.utk.edu/papi/

PAPI 的主要作用是程序运行时可以实时获得许多性能参数。Linux 系统需要对内核打补丁 (perfctr 补丁)，然后重新编译内核。

§1.5 BLAS/LAPACK

- 使用系统中的 BLAS 和 LAPACK 包：

```
./configure --with-blas=-lblas --with-lapack=-llapack
./configure --with-lapack="-lblas -llapack"
```

- 使用 Intel MKL：

```
./configure --with-lapack="-L/opt/intel/mkl/lib/32 \
             -lmkl_lapack -lmkl_def -lguide -lpthread"
```

- 其它高性能 BLAS 库：Goto BLAS、ATLAS。

如果使用 PETSc, `configure` 可自动通过 PETSc 的配置找到它所使用的 BLAS/LAPACK 库。

PHG 用到的所有软件包应该使用同样的 BLAS/LAPACK 库以免发生冲突。

§1.6 线性解法器

§1.6.1 LASPack

从 `ftp://159.226.92.111/pub/RPMS/` 中下载、安装 RPM 包即可。

§1.6.2 PETSc

`configure` 通过环境变量 `PETSC_DIR` 得到 PETSc 的安装目录及配置。

§1.6.3 Hypre

下载、编译、安装 Hypre 源码。缺省情况下 `configure` 会自动找到 Hypre 的头文件和库，必要时用 `--with-hypre-dir` 指定 Hypre 的安装目录 (PHG 假定头文件和库分别位于该目录中的 `lib` 和 `include` 子目录)

网址 `http://lsec.cc.ac.cn/phg/download.htm` 处提供了一个 Hypre-2.0.0 的 AMS 解法器补丁，它修正了 AMS 解法器的几个小 bug。

§1.6.4 MUMPS

PHG 只用到头文件 `dmumps*.h` 和库文件 `libdmumps.a` 和 `libpord.a`。必要时可用 `configure` 的选项 `--with-mumps-incdir` 和 `--with-mumps-lib` 来指定这些头文件和库文件的位置。

§1.6.5 SuperLU

从 `ftp://159.226.92.111/pub/RPMS/` 中下载、安装 `superlu-dist` 包

§1.6.6 SPOOLES

从 `ftp://159.226.92.111/pub/RPMS/` 中下载、安装 RPM 包

§1.6.7 Trilinos

直接从网上下载 Trilinos 源码编译、安装即可。

PHG 缺省情况下禁用 Trilinos, 必须用选项 `--enable-trilinos` (线性解法器) 和 `--enable-trilinos-anasazi` (特征值) 来启用

§1.6.8 其它解法器

PaSTiX, PARDISO (`mkl\_solver`), SuiteSparse (UMFPACK), etc.

§1.7 特征值解法器

§1.7.1 PARPACK

从 `ftp://159.226.92.111/pub/RPMS/` 中下载、安装 RPM 包，该 RPM 包修正了标准发布版中的一个小 bug。

§1.7.2 JDBSYM

从 `ftp://159.226.92.111/pub/RPMS/` 中下载 `jdbsym-parallel` 包，它是一个 (部分) 并行化的 JDBSYM 版本。

§1.7.3 BLOPEX (LOBPCG)

从 `ftp://159.226.92.111/pub/RPMS/` 中下载 `blopex-abstract` 包。

§1.7.4 SLEPc

直接编译、安装官方发布版，并设定环境变量 `SLEPC_DIR` 即可。

§1.7.5 PRIMME

从 `ftp://159.226.92.111/pub/RPMS/` 中下载 RPM 包。

§1.8 可视化软件

§1.8.1 ParaView

`http://www.paraview.org/`

§1.8.2 VisIt

`http://www.llnl.gov/visit/`

§1.8.3 MayaVi

`http://mayavi.sourceforge.net/`

§1.8.4 OpenDX

`http://www.opendx.org/`

§1.9 网格文件格式

参看 PHG 手册

§1.10 编译、运行测试程序

参看 PHG 手册

第二章　基础知识

§2.1　PHG 程序基本结构

```
#include "phg.h"

int
main(int argc, char **argv)
{
    ... ...
    phgInit(&argc, &argv);
    ... ...
    phgFinalize();
    exit(0);
}
```

§2.2　三角形、四面体网格的二分自适应细化与粗化

参看 PHG 手册

§2.3　层次网格结构

参看 PHG 手册

§2.4　网格剖分与分布式层次网格结构

参看 PHG 手册

第三章　基本数据类型与网格管理

§3.1　基本数据类型及常量

§3.1.1　数据类型

为方便使用不同精度进行计算，PHG 定义了一组浮点型、整型、字符型和布尔型变量类型。

```
FLOAT, INT, UINT, SHORT, USHORT, CHAR, BYTE, BOOLEAN (TRUE/FALSE)
```

§3.1.2　数学函数

对应于 FLOAT 类型的数学函数名通过将标准 C 库中的函数名的首字母改为大写得到，如：Sin、Sqrt、Pow、Fabs 等。

§3.1.3　常量

```
#define Dim     3       /* 空间维数 */
#define NVert   4       /* 每个单元的顶点数 */
#define NFace   4       /* 每个单元的面数 */
#define NEdge   6       /* 每个单元的边数 */
```

§3.1.4　其它类型

GTYPE 用于描述几何对象的几何类型，也用于定义单元的加密类型，可取值为 VERTEX、EDGE、FACE、DIAGONAL (ELEMENT)、OPPOSITE、MIXED 和 UNKNOWN。

BTYPE 用于描述边界类型，可应用于顶点、边、面或单元，它由一组标志位构成，这些标志位有：UNREFERENCED (未被叶子单元引用), OWNER (本进程所拥有), INTERIOR (区域内部), REMOTE (与其它进程共享), DIRICHLET (Dirichlet), NEUMANN (Neumann), BDRY_USER0 (用户类型 0), ..., BDRY_USER9 (用户类型 9), UNDEFINED (未指定类型的边界)

§3.2　单元对象

```
typedef struct {
    SIMPLEX     *children[2];           /* 指向两个子单元的指针 */
    void        *neighbours[NFace];     /* 指向邻居单元的指针 */
    INT         verts[NVert];           /* 4个顶点的本地编号 */
    INT         edges[NEdge];           /* 6条边的本地编号 */
    INT         faces[NFace];           /* 4个面的本地编号 */
    INT         index;                  /* 单元编号 */
    SHORT       mark;                   /* 用于标注加密、粗化 */
    BTYPE       bound_type[NFace];      /* 单元面的边界类型 */
    ... ...
} SIMPLEX;
```

其中 mark 成员用于标注希望加密或放粗的单元。mark > 0 表示要求将单元加密 mark 次，mark < 0 表示允许将单元放粗 $-$mark 次。

单元的顶点、边、面的编号分为全局编号、本地 (或局部) 编号和单元内编号三种。一个顶点和其相对的面的单元内编号相同。6 条边的单元内编号顺序为 0 (0–1), 1 (0–2), 2 (0–3), 3 (1–2), 4 (1–3), 5 (2–3)。

下述宏计算各种单元内编号：

```
GetEdgeVertex(edge_no, v)   /* 边的顶点编号, v 取 0 或 1 */
GetFaceVertex(edge_no, v)   /* 面的顶点编号, v 取 0、1 或 2 */
GetEdgeNo(v0, v1)           /* 两个顶点对应的边编号 */
```

例如：

GetEdgeVertex(3,0) $= 1$

GetEdgeVertex(3,1) $= 2$

GetFaceVertex(2,1) $= 1$

GetEdgeNo(1,3) $= 4$

§3.3 网格对象

PHG 的网格以子网格的形式分布在部分或全部进程中。

```
typedef struct {
    MPI_Comm    comm;           /* 包含网格的 MPI 通信器 */
    int         nprocs, rank;   /* 进程数、进程编号 */
    FLOAT       lif;            /* 负载不平衡因子 */
    COORD       *verts;         /* 子网格中所有顶点的坐标 */
    BTYPE       *types_vert;    /* 子网格中所有顶点的边界类型 */
    BTYPE       *types_edge;    /* 子网格中所有边的边界类型 */
    BTYPE       *types_face;    /* 子网格中所有面的边界类型 */
    BTYPE       *types_elem;    /* 子网格中所有单元的边界类型 */
    INT         nleaf;          /* 子网格中的叶子单元数 */
    INT         nvert, nvert_global;    /* 本地、全局顶点数 */
    INT         nedge, nedge_global;    /* 本地、全局边数 */
    INT         nface, nface_global;    /* 本地、全局面数 */
    INT         nelem, nelem_global;    /* 本地、全局单元数 */
    ... ...
} GRID;
```

- comm、nprocs、rank 分别为构成网格的通信器、进程数和当前进程在该通信器中的编号
- GRID 对象中的 verts 成员包含 (子) 网格中所有顶点的坐标。假设 e 是一个单元，则 e 的顶点 k $(0 \le k < 4)$ 的坐标为：

$$x = \texttt{g->verts[e->verts[k]][0]}$$

$$y = \texttt{g->verts[e->verts[k]][1]}$$

$$z = \texttt{g->verts[e->verts[k]][2]}$$

- lif 表示网格划分的负载不平衡因子 (load imbalance factor)：

$$\texttt{lif} = \frac{\text{子网格最大单元数}}{\text{子网格平均单元数}}$$

它会随网格的变化 (加密、放粗、重分布) 自动更新。

§3.4 网格导入与销毁

```
GRID *g;
g = phgNewGrid(-1);
phgImport(g, "cube.dat", FALSE);
... ...
phgFreeGrid(&g);
```

phgNewGrid 的参数是一些标志位，包括：VERT_FLAG (顶点类型与编号)、EDGE_FLAG (边的类型与编号)、FACE_FLAG (面的类型与编号)、ELEM_FLAG (单元的类型与编号)、GEOM_FLAG (可通过 phgGeom* 函数访问的几何量)

phgImport 最后一个参数取 TRUE 表示导入网格时强制对网格进行剖分并分布到 MPI_COMM_WORLD 的所有进程中 (此时 comm 等于 MPI_COMM_WORLD)；FALSE 表示导入网格时不对网格进行剖分，网格只存在于进程 0 中 (此时 comm 等于 MPI_COMM_SELF)

§3.5 网格导出

```
phgExportALBERT(GRID *g, const char *filename);
phgExportDX(GRID *g, const char *fn, DOF *dof, ...);
phgExportDXn(GRID *g, const char *fn, int ndof, DOF **dofs);
phgExportVTK(GRID *g, const char *fn, DOF *dof, ...);
phgExportVTKn(GRID *g, const char *fn, int ndof, DOF **dofs);
```

§3.6 网格单元遍历

```
GRID *g;
SIMPLEX *e;

ForAllElements(g, e) {
    ... ...
}
```

§3.7 网格剖分及负载平衡

```
 phgBalanceGrid(GRID *g, FLOAT lif_threshold,
    INT submesh_threshold, DOF *weights, FLOAT power);
```

- lif_threshold: 当负载不平衡因子大于该值时重新划分网格
- submesh_threshold: 子网格最小单元数，用于确定子网格数目
- weights: 单元权重
- power: 单元权重指数 (单元权重为 weights 的 power 次方)

§3.8 网格的加密与放粗

```
phgRefineAllElements(GRID *g, int level);
phgRefineMarkedElements(GRID *g);
phgCoarsenMarkedElements(GRID *g);
```

后两个函数根据单元中的 mark 成员加密、放粗网格。单元每加密一次 mark 值会被减去 1，每放粗一次 mark 值会被加上 1。由于加密是强制性的，因此从 phgRefineMarkedElements 返回时所有单元的 mark 成员一定是非正值。

第四章　自由度管理与有限元基函数

自由度对象用于管理有限元函数及分布在网格上的其它数据。自由度可以定义在顶点上、边上、面上和单元内部。

§4.1　自由度类型与有限元基函数

自由度类型由结构 DOF_TYPE 定义，它描述自由度在单元上的分布、有限元基函数及性质等。

```
typedef struct DOF_TYPE_ {
    const char          *name;          /* 自由度类型的名称 */
    FLOAT               *points;        /* 自由度位置 (插值点) */
    DOF_TYPE            *grad_type;     /* 梯度的自由度类型 */
    DOF_INTERP_FUNC     InterpC2F;      /* 父单元到子单元插值 */
    DOF_INTERP_FUNC     InterpF2C;      /* 子单元到父单元插值 */
    DOF_INIT_FUNC       InitFunc;       /* 初始化 (投影) 函数 */
    DOF_BASIS_FUNC      BasFuncs;       /* 计算基函数值 */
    DOF_BASIS_GRAD      BasGrads;       /* 计算基函数梯度值 */
    BOOLEAN             invariant;      /* 基函数与单元形状无关 */
    SHORT               nbas;           /* 一个单元中基函数个数 */
    BYTE                order;          /* 基函数多项式次数 */
    CHAR                continuity;     /* 有限元函数的连续性 */
    SHORT               dim;            /* 基函数维数 (1或Dim) */
    SHORT               np_vert;        /* 顶点自由度个数 */
    SHORT               np_edge;        /* 边自由度个数 */
    SHORT               np_face;        /* 面自由度个数 */
    SHORT               np_elem;        /* 单元自由度个数 */
    ... ...
} DOF_TYPE;
```

其中：

- np_vert: 每个顶点上的自由度数
- np_edge: 每条边上的自由度数
- np_face: 每个面上的自由度数
- np_elem: 每个单元中的自由度数

用于描述有限元基函数的自由度类型还有下列成员：

- order: 基函数的多项式次数
- BasFuncs: 计算基函数值
- BasGrads: 计算基函数相对于重心坐标的梯度值
- dim: 基函数的维数
- grad_type: 梯度的自由度类型

§4.1.1 已定义的有限元类型

- DOF_P0 – DOF_P4
  0 至 4 阶 Lagrange 元 (除 DOF_P0 外均为 H^1 conforming)
- DOF_HB1 – DOF_HB15
  1 至 15 阶 Hierarchical H^1 conforming 元
- DOF_HC0 – DOF_HC15
  1 至 15 阶 Hierarchical $\boldsymbol{H}_{\text{curl}}$ conforming 元
- DOF_DG0 – DOF_DG15: 0 至 15 阶 Discontinuous Galerkin 元 (L_2)

其中 DOF_HCn 为向量基 (三维)，其余为标量基。

DOF_DGn 基于 DOF_Pn ($n \leq 4$) 或 DOF_HBn ($n > 4$) 构造。

另外，DOF_ND1 (线性 Nédélec 元) 等价于 DOF_HC0，DOF_HFEB1 等价于 DOF_HC1，DOF_HFEB2 等价于 DOF_HC2

§4.2 自由度对象

自由度对象包含自由度类型、自由度维数以及存贮自由度数据的缓冲区

```
typedef struct DOF_ {
    char        *name;          /* name */
    GRID        *g;             /* the mesh */
    DOF_TYPE    *type;          /* type */
    SHORT       dim;            /* # variables per location */
    BTYPE       DB_mask;        /* Dirichlet boundary mask */
    FLOAT       *data;          /* data array */
    FLOAT       *data_vert;     /* pointer to vertex data */
    FLOAT       *data_edge;     /* pointer to edge data */
    FLOAT       *data_face;     /* pointer to face data */
    FLOAT       *data_elem;     /* pointer to element data */
    ... ...
} DOF;
```

当自由度对象中的自由度类型是有限元基函数时，该自由度对象可以看作一个定义在当前网格上的有限元函数。

自由度对象的主要成员如下：

- name: 自由度对象名称
- g: 网格
- type: 自由度类型
- dim: 维数
- data: (本地) 自由度数据缓冲区
- data_vert: 指向 data 中顶点自由度数据的起始地址
- data_edge: 指向 data 中边自由度数据的起始地址
- data_face: 指向 data 中面自由度数据的起始地址
- data_elem: 指向 data 中单元自由度数据的起始地址

§4.3 创建、导出和销毁自由度对象

```
DOF *u;
u = phgDofNew(GRID *g, DOF_TYPE *type, SHORT dim,
              const char *name, DOF_USER_FUNC userfunc);
... ...
phgDofFree(&u);
```

其中 userfunc 可以取下述形式：

- DofNoData: 不保存自由度数据
- DofNoAction: 网格改变 (加密或放粗) 时不对自由度数据进行插值
- DofInterpolation: 网格改变时自动对自由度数据进行插值
- 函数指针: 指向形为

  ```
  void userfunc(FLOAT x, FLOAT y, FLOAT z, FLOAT *values)
  ```

  的函数，PHG 根据该函数的函数值来计算自由度值 (通过插值或 L_2 投影)

  函数 phgExportVTK 和 phgExportDX 允许将任意数量的自由度对象写入到可视化文件中。

§4.4 自由度编号

PHG 对自由度的编号采用*局部编号*，它基于子网格中的顶点、边、面和单元的局部编号。具体编号方式是：首先对子网格中所有顶点自由度按顶点的局部编号顺序进行编号，然后对子网格中所有边自由度按边的局部编号顺序进行编号，接着对子网格中所有面自由度按面的局部编号顺序进行编号，最后对子网格中所有单元自由度按单元的局部编号顺序进行编号。

对于同时属于多个子网格的自由度，它们在不同子网格中有着不同的编号，其中唯一一个子网格为它的“属主”，该子网格便是它所在的顶点、边、面或单元属主。

PHG 不提供自由度的全局编号，需要自由度的全局编号时可以通过 MAP 和 VEC 对象来进行创建和管理。

§4.5 直接访问自由度数据

对自由度 u，DofData(u) 指向数据缓冲区，(等价于 u->data)，DofVertexData(u,vertex_no) 指向指定顶点的自由度数据的起始地址，DofEdgeData(u,edge_no) 指向指定边的自由度数据的起始地址，DofFaceData(u,face_no) 指向指定面的自由度数据的起始地址，DofElementData(u,elem_no) 指向指定单元的自由度数据的起始地址。

对自由度数据的直接访问通常结合单元、顶点、边或面的遍历进行，例如：

```
int i;
DOF *u;
SIMPLEX *e;
FLOAT *data;
... ...
ForAllElements(u->g, e) {
    for (i = 0; i < NFace; i++) {
        data = DofFaceData(u, e->faces[i]);
        ... ...
    }
```

```
}
```

§4.6 自由度对象的运算

```
phgDofCopy(DOF *src, DOF **dest, DOF_TYPE *newtype, char *name)
```

创建自由度对象 dest，它是自由度对象 src 的拷贝。newtype 取 NULL 表示 dest 的类型等于 src->type。如果 newtype 指定的类型不同于 src->type，则 dest 等于 src 到新的有限元空间上的投影。

```
phgDofAXPY(FLOAT a, DOF *x, DOF **y)
```

计算 $y := ax + y$

```
phgDofAXPBY(FLOAT a, DOF *x, FLOAT b, DOF **y)
```

计算 $y := ax + by$

```
phgDofMatVec(FLOAT alpha, DOF *A, DOF *x, FLOAT beta, DOF **y)
```

计算 $y := \alpha Ax + \beta y$。这里 A 为方阵，根据其维数与 x 维数的关系分别做为标量、对角阵和满阵与 x 进行运算，A = NULL 表示单位阵。

```
phgDofMM(MAT_OP transa, MAT_OP transb, int M, int N, int K,
       FLOAT alpha, DOF *A, int blka, DOF *B, FLOAT beta, DOF **C)
```

该函数相当于 (按点) 计算

$$C := \alpha\,\mathtt{transa}(A)_{M\times K}\mathtt{transb}(B)_{K\times N} + \beta\, C_{M\times N}$$

其中 transa 和 transb 可分别取 MAT_OP_N 或 MAT_OP_T。它涵盖了 phgDofAXPY、phgDofAXPBY 和 phgDofMatVec 的功能。

当 blka ≤ 0 时，A 的每组自由度值被看做一个 $M \times K$ 或 $K \times M$ 的矩阵，因此 A 的维数 (DofDim(A) = A->dim × A->type->dim) 应该等于 MK。

当 blka > 0 时，A 的每组自由度值被看做一个对角块大小为 blka × blka 的块对角矩阵，此时要求 DofDim(A) = M/blka。

B 的每组自由度值被看做一个 $K \times N$ 或 $N \times K$ 的矩阵，DofDim(B) = NK。

C 的每组自由度值被看做一个 $M \times N$ 的矩阵，DofDim(C) = MN。

详见用户手册附录中的说明。

§4.7 有限元函数求值

```
FLOAT *phgDofEval(DOF *u, SIMPLEX *e, const FLOAT lambda[],
                  FLOAT *values);
FLOAT *phgDofEvalGradient(DOF *u, SIMPLEX *e,
            const FLOAT lambda[], FLOAT *values);
```

```
FLOAT *phgDofEvalDivergence(DOF *u, SIMPLEX *e,
            const FLOAT lambda[], FLOAT *values);
FLOAT *phgDofEvalCurl(DOF *u, SIMPLEX *e,
            const FLOAT lambda[], FLOAT *values);
```

这些函数计算有限元函数在指定重心坐标位置的函数值或导数值

§4.8 自由度对象的模

```
FLOAT phgDofNormL1(DOF *u);
FLOAT phgDofNormL1Vec(DOF *u);
FLOAT phgDofNormL2(DOF *u);
FLOAT phgDofNormL2Vec(DOF *u);
FLOAT phgDofNormH1(DOF *u);
FLOAT phgDofNormInftyVec(DOF *u);
```

§4.9 自由度对象的微分

```
DOF *phgDofGradient(DOF *src, DOF **newdof,
            DOF_TYPE *newtype, const char *name);
DOF *phgDofDivergence(DOF *src, DOF **newdof,
            DOF_TYPE *newtype, const char *name);
DOF *phgDofCurl(DOF *src, DOF **newdof,
            DOF_TYPE *newtype, const char *name);
```

§4.10 特殊自由度对象

§4.10.1 常量型自由度对象

常量型自由度对象的类型为 DOF_CONSTANT，它相当于一个取值为常数的 dim 维函数。

例如：

```
DOF *A;
A = phgDofNew(g, DOF_CONSTANT, 3, "A", DofNoAction);
phgDofSetDataByValuesV(A, 3, "A", 1., 2., 3.);
```

定义了一个取值为常向量 $(1,2,3)$ 的函数。

常量型自由度对象可以和其它自由度对象一样参与代数运行、数值积分、进行求值等。

§4.10.2 解析型自由度对象

解析型自由度对象的类型为 DOF_ANALYTIC，它调用一个用户提供的函数计算自由度对象的值。

```
static void
f(FLOAT x, FLOAT y, FLOAT z, FLOAT *value)
{
    *value = (x < 0.5 ? 1.0 : -1.0);
}
    ... ...
```

```
DOF *c;
c = phgDofNew(g, DOF_ANALYTIC, 1, "coefficient", f);
```

一个分片常数函数可以用一个解析型自由度对象来表示，当网格面与间断面一致时也可以存储在一个类型为 DOF_P0 的自由度对象中。

解析型自由度对象可以和其它自由度对象一样参与代数运行、数值积分、进行求值等，但不允许对其进行微分运算。

§4.11 访问邻居单元的自由度数据

串行计算时，可以通过 e->neighbours[i] 直接访问单元 e 的第 i 个邻居单元。而并行计算时，邻居单元可能位于其它进程，对邻居单元的访问不便于直接用 MPI–1 实现。

为避免对邻居单元的直接访问，一个方法是将计算顺序按照单元遍历的形式进行组织，将每个单元的贡献叠加到计算结果上。可以利用 PHG 的向量组装功能来完成计算结果的叠加，关于这种方法参看 33 页§8.2.1 。

另外一个方法是调用 phgNeighbourData 来获取 (远程) 邻居单元的数据 (PHG 会自动完成必要的通信)。

例如，假设自由度对象 u 的类型为 DOF_P0 (分片常数)，下述代码计算 u 在所有面上的平均值。

```
NEIGHBOUR_DATA *nd;
SIMPLEX *e;
int i;
double a;

nd = phgDofInitNeighbourData(u, NULL);
ForAllElements(u->g, e) {
    for (i = 0; i < NFace; i++) {
        if (边界面) {
           a = *DofElementData(u, e->index);
        }
        else {
           a = (*DofElementData(u, e->index) +
                *phgDofNeighbourData(nd, e, i, 0, NULL)) * .5;
        }
        ... ...
    }
}
phgDofReleaseNeighbourData(&nd);
```

§4.12 几何量自由度对象

PHG 内部自动维护一个特殊的自由度对象，称为几何量自由度对象，用于计算和存储与边、面和单元相关的一些常用几何量，并提供一组函数供用户访问这些量。

```
FLOAT phgGeomGetVolume(GRID *g, SIMPLEX *e);
FLOAT phgGeomGetDiameter(GRID *g, SIMPLEX *e);
FLOAT *phgGeomGetJacobian(GRID *g, SIMPLEX *e);
FLOAT phgGeomGetFaceArea(GRID *g, SIMPLEX *e, int face);
```

```
FLOAT phgGeomGetFaceAreaByIndex(GRID *g, INT face_no);
FLOAT phgGeomGetFaceDiameter(GRID *g, SIMPLEX *e, int face);
FLOAT *phgGeomGetFaceNormal(GRID *g, SIMPLEX *e, int face);
FLOAT *phgGeomGetFaceOutNormal(GRID *g, SIMPLEX *e, int face);
FLOAT *phgGeomXYZ2Lambda(GRID *g, SIMPLEX *e,
                    FLOAT x, FLOAT y, FLOAT z);
phgGeomLambda2XYZ(GRID *g ,SIMPLEX *e, const FLOAT *lambda,
                    FLOAT *x, FLOAT *y, FLOAT *z);
```

第五章　数值积分

§5.1　有限元计算中的数值积分

有限元离散通常归结到计算自由度对象、基函数间的二次型、三次型。PHG 中这些计算通常利用数值积分来完成。

§5.2　数值积分公式

Gauss 型数值积分公式用被积函数在给定区域上一些特定点处的值的线性组合来近似函数在该区域上的积分。以单元上的积分为例，假定 e 是一个单元，$f(x)$ 是定义在 e 上的一个函数，则：

$$\int_e f(x)\,dx \approx \sum_{i=0}^{n-1} w_i f(x_i)$$

其中 x_i 称为积分点 (abscissas)，w_i 称为权重 (weights)。适当选取积分点和权重可以使得上述公式对不超过某个次数的任意多项式精确成立。一个积分公式的阶指积分公式对所有次数不超过该阶的多项式精确成立。

PHG 的数值积分公式中使用归一化的权重，即将积分写成：

$$\int_e f(x)\,dx \approx V(e) \sum_{i=0}^{n-1} w_i f(x_i)$$

其中 $V(e)$ 表示单元 e 的体积。显然，如果积分公式至少对 0 次多项式是精确的，则积分权重必须满足：

$$\sum_{i=0}^{n-1} w_i = 1$$

PHG 中用单元中的重心坐标表示积分点，易知这种表示下积分点和权重相对于仿射变换是不变的，换言之，一个积分公式的积分点和权重与单元的形状、位置无关。

PHG 提供任意阶数值积分公式，部分公式通过解析或特定的数值方法得到，其它公式则是利用一维 Gauss-Jacobi 积分公式通过张量积的方式构造。

§5.3　对称型数值积分公式

PHG 中提供的所有非张量积数值积分公式都是对称的，即对一个积分点的重心坐标分量进行任意置换得到的点都是积分点，并且这些积分点对应的权重是一样的。

以三维积分点为例，假设 $(\frac{1}{2}, \frac{1}{2}, 0, 0)$ 是一个积分点，则 $(\frac{1}{2}, 0, \frac{1}{2}, 0)$, $(\frac{1}{2}, 0, 0, \frac{1}{2})$, $(0, \frac{1}{2}, \frac{1}{2}, 0)$, $(0, \frac{1}{2}, 0, \frac{1}{2})$ 和 $(0, 0, \frac{1}{2}, \frac{1}{2})$ 必然也是积分点。

为了方便地表示对称型数值积分公式，文件 `include/quad.h` 中定义了一组宏 `PermXXX` 和 `DupXXX`，例如，上述 6 个积分点可简单地写成 `Perm22(0.5)`，假设它们的权重为 0.25，相应的权重可写成 `Dup22(0.25)`。

一维积分公式		
宏	积分点	权重
Perm2(1/2) Perm11(a)	(1/2,1/2) permute{(a,1-a)}	Dup2(w) = w Dup11(w) = w,w
二维积分公式		
宏	积分点	权重
Perm3(1/3) Perm21(a) Perm111(a,b)	(1/3,1/3,1/3) permute{(a,a,1-2a)} permute{(a,b,1-a-b)}	Dup3(w) = w Dup21(w) = w,w,w Dup111(w) = $\underbrace{\text{w},\ldots,\text{w}}_{6}$
三维积分公式		
宏	积分点	权重
Perm4(1/4) Perm31(a) Perm22(a) Perm211(a,b) Perm1111(a,b,c)	(1/4,1/4,1/4,1/4) permute{(a,a,a,1-3a)} permute{(a,a,0.5-a,0.5-a)} permute{(a,a,b,1-2a-b)} permute{(a,b,c,1-a-b-c)}	Dup4(w) = w Dup31(w) = w,w,w,w Dup111(w) = $\underbrace{\text{w},\ldots,\text{w}}_{6}$ Dup211(w) = $\underbrace{\text{w},\ldots,\text{w}}_{12}$ Dup1111(w) = $\underbrace{\text{w},\ldots,\text{w}}_{24}$

§5.4 PHG 的数值积分公式

PHG 用结构 `QUAD` 存储数值积分公式，这些公式存储在文件 `src/quad.c` 中。该结构体的主要成员如下：

```
char *name;
int dim;        /* 1: 边, 2: 三角形, 3: 四面体 */
int order;      /* 积分公式的阶数 */
int npoints;    /* 积分公式的点数 */
FLOAT *points;  /* 积分点重心坐标 */
FLOAT *weights; /* 积分点权重 */
```

函数 `phgQuadGetQuad`n`D(`p`)` ($n = 1, 2, 3$) 返回指向一个 n 维 p 阶 `QUAD` 结构的指针。如果指定阶数的积分公式不存在，该函数会自动构造一个张量积形式的数值积分公式。

例：假设 u 是一个 (维数为 1 的) 自由度对象，下述代码分别用 3 阶积分公式计算 u 在单元 e 的边、面和单元上的积分。

§5.4.1 第 k 条边 ($0 \le k < 6$) 上的积分

```
QUAD *q;
FLOAT lambda[] = {0., 0., 0., 0.}, sum = 0., value, *p, *w;
int i, v0, v1;
FLOAT sum = 0., value, *p, *w;

v0 = GetEdgeVertex(k, 0);
v1 = GetEdgeVertex(k, 1);
q = phgQuadGetQuad1D(3);
p = q->points; w = q->weights;
for (i = 0; i < q->npoints; i++) {
```

```
        lambda[v0] = *(p++);
        lambda[v1] = *(p++);
        phgDofEval(u, e, lambda, &value);
        sum += *(w++) * value;
    }
    sum *= 边长度;
```

(边长度可以通过 u->g->verts[e->verts[]] 计算)

§5.4.2 第 k 个面 $(0 \le k < 4)$ 上的积分

```
    QUAD *q;
    FLOAT lambda[] = {0., 0., 0., 0.}, sum = 0., value, *p, *w;
    int i, v0, v1, v2;

    v0 = GetFaceVertex(k, 0);
    v1 = GetFaceVertex(k, 1);
    v2 = GetFaceVertex(k, 2);
    q = phgQuadGetQuad2D(3);
    p = q->points; w = q->weights;
    for (i = 0; i < q->npoints; i++) {
        lambda[v0] = *(p++);
        lambda[v1] = *(p++);
        lambda[v2] = *(p++);
        phgDofEval(u, e, lambda, &value);
        sum += *(w++) * value;
    }
    sum *= phgGeomGetFaceArea(u->g, e, k);
```

§5.4.3 单元上的积分

```
    QUAD *q;
    FLOAT sum = 0., value, *p, *w;
    int i;

    q = phgQuadGetQuad3D(3);
    p = q->points;
    w = q->weights;
    for (i = 0; i < q->npoints; i++, p += 4) {
        phgDofEval(u, e, p, &value);
        sum += *(w++) * value;
    }
    sum *= phgGeomGetVolume(u->g, e);
```

§5.5 双、三线性型的计算

有限元中经常需要计算双或三线性型，例如：

$$\int_e \varphi_i\varphi_j, \quad \int_e \nabla\varphi_i\cdot\nabla\varphi_j, \quad \int_e A\nabla\varphi_i\cdot\nabla\varphi_j, \quad \int_e f\varphi_i, \quad \int_f g\varphi_i$$

其中 φ_i, φ_j 为基函数，f 为某个函数，A 是一个标量函数或 3×3 矩阵函数，e 为单元，f 为单元的一个面。

可以调用下述函数完成这些计算：

```
phgQuadBasDotBas(e, u, i, v, j, QUAD_DEFAULT);
phgQuadGradBasDotGradBas(e, u, n, v, m, QUAD_DEFAULT);
phgQuadGradBasAGradBas(e, u, n, A, v, m, QUAD_DEFAULT);
phgQuadDofTimesBas(e, f, u, n, QUAD_DEFAULT, result);
phgQuadFaceDofDotBas(e, k, g, DOF_PROJ_NONE, v, n,
          QUAD_DEFAULT);
```

§5.6 计算面跳量

有限元后验误差估计中经常需要计算面跳量，PHG 提供一个通用的面跳量计算函数：

```
DOF *phgQuadFaceJump(DOF *u, DOF_PROJ proj, const char *name,
            int quad_order);
```

该函数返回一个自由度对象，每个面上一个自由度值，该值等于 u 在该面上的跳量的模的平方的积分(面上 L_2 模的平方)。

proj 表示相对于面的外法向的投影。令 $\boldsymbol{n}$ 为面的外法向，$[\cdot]$ 表示函数跨过面的跳量，则：

proj = DOF_PROJ_NONE　　计算 $\int_f |[u]|^2$

proj = DOF_PROJ_DOT　　计算 $\int_f |\boldsymbol{n}\cdot[u]|^2$

proj = DOF_PROJ_CROSS　　计算 $\int_f |\boldsymbol{n}\times[u]|^2$

§5.7 基函数、函数值的 cache

许多情况下数值积分中基函数或函数的值会被重复用到。为避免重复计算，PHG 可以将计算过的基函数值或基函数梯度值保存在特定的 cache 中。

编写数值积分函数时，如果调用下述函数计算被积函数或基函数值，则 PHG 会根据函数或基函数的性质自动 cache 基函数或其梯度的值：

```
FLOAT *phgQuadGetFuncValues(GRID *g, SIMPLEX *e,
            int dim, DOF_USER_FUNC userfunc, QUAD *quad);
FLOAT *phgQuadGetDofValues(SIMPLEX *e, DOF *u, QUAD *quad);
FLOAT *phgQuadGetBasisValues(SIMPLEX *e, DOF *u, int n,
            QUAD *quad);
FLOAT *phgQuadGetBasisGradient(SIMPLEX *e, DOF *u, int n,
            QUAD *quad);
FLOAT *phgQuadGetBasisCurl(SIMPLEX *e, DOF *u, int n,
            QUAD *quad);
```

例如，下面的代码计算 $\int_e \nabla\varphi_n\cdot\nabla\varphi_n$，其中 φ_n 和 φ_n 分别表示自由度对象 u (这里假定其维数为 1) 的第 n 和第 m 个基函数。

```
QUAD *quad;
const FLOAT *g1, *g2, *w;
FLOAT d;
```

```
int i;

quad = phgQuadGetQuad3D(2 * u->type->order - 2);
g1 = phgQuadGetBasisGradient(e, u, n, quad);
g2 = phgQuadGetBasisGradient(e, u, m, quad);
w = quad->weights;
d = 0.;
for (i = 0; i < quad->npoints; i++, g1 += 3, g2 += 3, w++)
    d += (g1[0] * g2[0] + g1[1] * g2[1] + g1[2] * g2[2]) * *w;
d *= phgGeomGetVolume(u->g, e);
```

目前 PHG 暂时只提供三维积分的 cache 机制。

第六章　命令行选项

PHG 可以通过命令行选项的方式来设定、改变程序运行的参数。将一组命令行选项放在一个文件中可以起到参数文件的作用。

PHG 选项的形式为 '-选项名 [参数]' 或 '-选项名[=参数]'，选项之间用空格隔开，选项名与参数间用空格或等号隔开。

对于不带参数的选项，'+选项名' 表示与 '-选项名' 相反的含义。

当同一个命令行选项多次出现时，以最后一次的值为准。运行任何一个 PHG 程序时，都可以用'-help' 列出它所支持的命令行选项及参数。

函数 phgInit 负责对所有命令行选项进行处理。调用 phgInit 之后，命令行 (argc, argv) 中将只留下非选项形式的命令行参数及 oem_options 选项中的参数。

PHG 的每个选项都有一个与之相对应的变量，选项的作用就是根据选项的参数设定该变量的值。PHG 支持下述几种类型的命令行选项：

选项类型	参数类型	变量类型
无参数	无	BOOLEAN
整型	整数	INT
浮点型	浮点数	FLOAT
字符串	字符串	char *
文件名	文件名	char *
枚举型	关键字	int
函数型	字符串	函数指针 (option handler)

选项 '-options_file 文件名' 将指定文件 (称为选项文件) 中的所有选项插入到当前位置。一个选项文件中可以包含任意数目的选项，选项间用空格或换行隔开。

如果存在名为 "可执行文件名.options" 的选项文件，PHG 会自动插入该文件中的选项。

-oem_options 是一个特殊的选项，PHG 将其参数原封不动地传递给其它软件 (如 PETSc、Hypre 等)。当有多个 '-oem_options' 选项时，PHG 将它们的所有参数按顺序合并起来。例如，下例选用 CG 解法器和 block Jacobi 预条件子做为 PETSc 中的缺省解法器：

```
-oem_options "-ksp_type cg -pc_type bjacobi"
```

用户程序可以在调用 phgInit 前调用 phgOptionsPreset 来预设一些选项的值。可以多次调用 phgOptionsPreset，每次调用 phgOptionsPreset 可以给出多个选项。例如：

```
phgOptionsPreset("-dof_type P4");
phgOptionsPreset("-solver_maxit 1000 -solver_rtol 1e-5");
phgOptionsPreset("-solver hypre");
```

如果在运行一个 PHG 程序时使用了 '-help' 选项，则会显示出它所支持的命令行选项及说明然后立即退出。-help 的参数是一个类别名，如 '-help hypre' 显示有关 Hypre 的选项，'-help all' 显示所有选项，而 '-help help' 则显示出所有类别名。

§6.1 自定义命令行选项

用户可以调用函数 phgOptionsRegisterXxxx 注册新的命令行选项。注册命令行选项必须在调用 phgInit 之前进行。

```
phgOptionsRegisterTitle(const char *str, const char *help,
                        const char *category);
phgOptionsRegisterNoArg(char *name, char *help, BOOLEAN *var);
phgOptionsRegisterInt(char *name, char *help, INT *var);
phgOptionsRegisterFloat(char *name, char *help, FLOAT *var);
phgOptionsRegisterString(char *name, char *help, char **var);
phgOptionsRegisterFilename(char *name, char *help, char **var);
phgOptionsRegisterKeyword(char *name, char *help,
                          char **keys, int *var);
phgOptionsRegisterHandler(char *name, char *help,
                          OPTION_HANDLER func, BOOLEAN append);
```

这些函数的用法可参看 poisson.c。

§6.2 获取命令行参数的当前值

下面的函数可以用来在程序运行过程中获取命令行选项的当前值：

```
BOOLEAN phgOptionsGetNoArg(char *op_name);
INT phgOptionsGetInt(char *op_name);
FLOAT phgOptionsGetFloat(char *op_name);
char *phgOptionsGetKeyword(char *op_name);
char *phgOptionsGetString(char *op_name);
char *phgOptionsGetFilename(char *op_name);
```

例如：

```
phgPrintf("Default solver: %s\n",
            phgOptionsGetKeyword("default_solver"));
```

§6.3 动态设置命令行选项的参数值

下列函数用于在程序运行过程中动态改变命令行选项对应的变量的值。需要注意的是，有些命令行选项是在程序启动之初发生作用，对于这些选项，程序运行中改变它们的设定不起作用，并且有可能导致程序异常。

```
void phgOptionsSetOptions(const char *str);
phgOptionsSetNoArg(char *op_name, BOOLEAN value);
phgOptionsSetInt(char *op_name, INT value);
phgOptionsSetFloat(char *op_name, FLOAT value);
phgOptionsSetKeyword(char *op_name, char *value);
phgOptionsSetString(char *op_name, char *value);
phgOptionsSetFilename(char *op_name, char *value);
phgOptionsSetHandler(char *op_name, char *value);
phgOptionsPush(void);
phgOptionsPop(void);
```

其中，phgOptionsPush 和 phgOptionsPop 用于保存、恢复当前所有命令行选项的设置。

§6.4 显示命令行及命令行选项

```
phgOptionsShowCmdline(void);
phgOptionsShowUsed(void);
```

`phgOptionsShowCmdline` 打印出运行程序的命令行。

`phgOptionsShowUsed` 打印出所有用户给出的命令行选项的最终值。

第七章　线性解法器

§7.1　线性解法器对象

```
typedef struct {
    OEM_SOLVER  *oem_solver;    /* 外部解法器 */
    MAT         *mat;
    VEC         *rhs;
    ... ...
} SOLVER;
```

其中，oem_solver 指所使用的“外部”解法器。当 oem_solver 为 NULL 时表示使用默认解法器，默认解法器可以在命令行上用选项 ‘-solver’ 来指定。

PHG 目前支持的“外部”解法器 (OEM solver) 包括:

```
SOLVER_DEFAULT,
SOLVER_PCG, SOLVER_GMRES, SOLVER_AMS,   /* 内部解法器 */
SOLVER_HYPRE, SOLVER_PETSC, SOLVER_TRILINOS,    /* 迭代法 */
SOLVER_MUMPS, SOLVER_SUPERLU, SOLVER_SPOOLES,   /* 直接法 */
SOLVER_SPC, SOLVER_LASPACK                      /* 其它解法器 */
```

§7.2　创建和销毁解法器对象

```
SOLVER *phgSolverCreate(OEM_SOLVER *oem_solver, DOF *u, ...);
int phgSolverDestroy(SOLVER **solver);
```

phgSolverCreate 中的可变参数给出一组构成未知量的自由度对象，可变参数表必须以空指针 NULL 结束。

§7.3　组装线性系统的系数矩阵及右端项

在 PHG 的线性解法器中，未知量通常对应一组自由度对象。组装系数矩阵或右端项时通常使用未知量的局部编号。如果未知量由单个自由度对象构成，则未知量的局部编号就是自由度的局部编号。

下述函数将自由度编号或单元基函数编号映射为未知量的局部编号：

```
phgSolverMapE2L(solver, dof_no, e, basis_no);
phgSolverMapD2L(solver, dof_no, dof_index);
```

下述函数用来将特定项叠加到系数矩阵或右端项中：

```
phgSolverAddMatrixEntry(SOLVER *solver, INT row, INT col,
                        FLOAT value);
phgSolverAddMatrixEntries(SOLVER solver, INT nrows, INT *rows,
                          INT ncols, INT *cols, FLOAT *values);
phgSolverAddRHSEntry(SOLVER *solver, INT index, FLOAT value);
```

```
phgSolverAddRHSEntries(SOLVER *solver, INT n, INT *indices,
                       FLOAT *values)
```

其中行、列编号使用未知量的局部编号。

§7.4 Dirichlet 边界条件

PHG 提供一个通用的处理 Dirichlet 边界条件的函数，它将一个指定位置的 Dirichlet 边界条件转化为一个关于某些自由度的线性方程。

```
BOOLEAN phgDofDirichletBC(DOF *u, SIMPLEX *e, int index,
                          DOF_USER_FUNC f, FLOAT mat[],
                          FLOAT rhs[], DOF_PROJ proj);
```

其中 `index` 代表单元基函数编号。返回值为 `FALSE` 表示该基函数对应的位置不是 Dirichlet 边界 (哪些边界属于 Dirichlet 边界可通过自由度对象中的 `DB_mask` 成员指定)，否则在 `mat` 中返回边界方程的系数，在 `rhs` 中返回边界方程右端项。数组 `mat` 的长度应该等于单元基函数的个数，而 `rhs` 的长度则应该等于 `u->dim`。

详见用户手册的附录中关于该函数的说明。

§7.5 用同一个系数矩阵反复求解不同右端项

```
创建解法器对象 solver;
生成系数矩阵和右端项;
组装系数矩阵和右端项;
phgSolverSolve(solver, FALSE, ... ...);
phgSolverResetRHS(solver);
生成、组装新的右端项;
phgSolverSolve(solver, FALSE, ... ...);
... ...
销毁解法器对象;
```

§7.6 PHG 支持的解法器

§7.6.1 PCG/GMRES

§7.6.2 LASPack

§7.6.3 PETSc

§7.6.4 HYPRE

§7.6.5 MUMPS

§7.6.6 SuperLU

§7.6.7 SPOOLES

§7.6.8 Trilinos/AztecOO

第八章　矩阵及向量操作

§8.1　MAP

MAP 用于管理连续分块存储的分布式向量，同时提供自由度到向量分量的映射关系。

§8.1.1　简单映射

```
MAP *phgMapCreateSimpleMap(GRID *g, INT m, INT M);
```

该函数定义一个全局长度为 M、本进程中局部长度为 m 的分布式向量。注意，调用该函数时，与网格 g 相关联的所有进程必须同时调用，并且使用相同的 M 参数。

假设 g 分布在 p 个进程中，进程 i 中的参数 m 等于 m_i，$i = 0, \ldots, p-1$，则 m_i 必须满足下述两个条件之一：

1) 所有 m 之和等于 M，即：

$$\sum_{i=0}^{p-1} m_i = \mathtt{M}$$

此时定义的是一个全局长度为 M、按进程序号分块顺序存储、进程 i 中的分块大小为 m_i 的分布式向量。

2) 所有进程中的 m 等于 M，即：

$$m_i = \mathtt{M}, \quad i = 0, \ldots, p-1$$

此时定义的是一个非分布的、同时存储在所有进程中、长度为 M 的向量。这种映射称为串行映射。

§8.1.2　基于自由度的映射

```
MAP *phgMapCreate(DOF *u, ...);
```

该函数所创建的向量分量对应于可变参数表中列出的所有自由度对象中的自由度，向量在进程中的分布由当前子网格剖分决定。参数表必须以空指针 NULL 结束。

例如：

```
DOF *u, *v;
MAP *map;
... ...
map = phgMapCreate(u, v, NULL);
```

创建一个 MAP，它描述 u 和 v 中的所有自由度所构成的一个向量。

§8.1.3　映射中的编号

局部编号

基于本地自由度的编号。对简单映射而言，它等价于本地向量编号。对基于自由度的映射而言，它等于将所有自由度对象中自由度的局部编号顺序联接起来得到的编号。

例如，假设上例中 u 的自由度的局部编号为 $0,\ldots,N-1$, v 的自由度的局部编号为 $0,\ldots,M-1$, 则映射中对应于 u 的自由度的局部编号为 $0,\ldots,N-1$, 对应于 v 的自由度的局部编号为 $N,\ldots,M+N-1$。

本地向量编号

属于本地的向量分量的编号，编号范围为 $0,\ldots,\mathtt{m}-1$

全局向量编号

向量分量的全局编号，编号范围为 $0,\ldots,\mathtt{M}-1$

§8.1.4 编号转换

```
INT phgMapD2L(MAP *map, int dof_no, INT index);
INT phgMapE2L(MAP *map, int dof_no, SIMPLEX *e, int index);
INT phgMapL2V(MAP *map, INT index);
INT phgMapL2G(MAP *map, INT index);
```

其中 dof_no 代表映射中自由度对象的序号，从 0 开始，如前例中 u 的序号为 0, v 的序号为 1。

- phgMapD2L 将自由度在自由度对象中的编号映射为 MAP 中的局部编号。
- phgMapE2L 将自由度在单元中的编号 (局部基函数编号) 映射为 MAP 中的局部编号。
- phgMapL2V 将局部编号映射为本地向量编号。
- phgMapL2G 将局部编号映射为全局向量编号。

§8.1.5 自由度与向量间的数据传递

```
int phgMapDofToLocalData(MAP *map, int ndof, DOF **dofs,
                         FLOAT *data);
int phgMapLocalDataToDof(MAP *map, int ndof, DOF **dofs,
                         FLOAT *data);

VEC *phgMapDofArraysToVec(MAP *map, int nvec, BOOLEAN bdry,
                          VEC **vecptr, DOF **u, ...);
VEC *phgMapDofArraysToVecN(MAP *map, int nvec, BOOLEAN bdry,
                           VEC **vecptr, int ndof, DOF ***dofs);

void phgMapVecToDofArrays(MAP *map, VEC *vec, BOOLEAN bdry,
                          DOF **u,...);
void phgMapVecToDofArraysN(MAP *map, VEC *vec, BOOLEAN bdry,
                           int ndof, DOF ***dofs);
```

§8.1.6 映射的消毁

```
phgMapDestroy(MAP **map_ptr);
```

为支持同一个映射被多个矩阵、向量引用，在映射内部保存有一个引用计数器。一个映射被创建时，它的引用计数器为 0，而每当一个新的对象 (矩阵、向量等) 引用该映射时，它的引用计数器会被加 1。调用 phgMapDestroy 时，如果引用计数器为正，则只将引用计数器减 1，只有当引用计数器为 0 时 phgMapDestroy 才实际消毁该映射。一个引用了映射的对象被消毁时，它会自动对所引用的所有映射调用 phgMapDestroy。因此，phgMapDestroy 只有当一个映射没被任何其它对象引用时才会消毁该映射。

例如，下面的代码是错误的：

```
MAT *mat = phgMatCreate(......);
MAP *map = mat->rmap;
... ...
phgMatDestroy(&mat);
... ...
phgMapDestroy(&map);
```

正确的代码应该是：

```
MAT *mat = phgMatCreate(......);
MAP *map = phgMatGetRowMap(mat);
... ...
phgMatDestroy(&mat);
... ...
phgMapDestroy(&map);
```

§8.2 向量 (VEC)

```
VEC *phgVecCreate(GRID *g, INT m, INT M, int nvec);
VEC *phgMapCreateVec(MAP *map, int nvec);
void phgVecDestroy(VEC **vec_ptr);

void phgVecAddEntry(VEC *vec, int which, INT index, FLOAT value);
void phgVecAddEntries(VEC *vec, int which, INT n, INT *indices,
                      FLOAT *values);
void phgVecAssemble(VEC *vec);
```

其它关于 VEC 的函数可用 'phgdoc phgVec' 列出。

phgVecAddEntry 和 phgVecAddEntries 中均使用向量分量的局部编号。

§8.2.1 利用向量组装功能完成跨单元的量的叠加

利用 PHG 的向量组装功能可以完成一些跨单元的量的叠加。以 19 页§4.11 中的计算为例，假设自由度对象 u 的类型为 DOF_P0 (分片常数)，为了计算 u 在面上的平均值，可以通过一个向量，对单元进行一次遍历完成。

```
static DOF_TYPE type = {... ..., 0, 0, 1, 0};
DOF *a;     /* 存储面上平均值的自由度对象 */
MAP *m; VEC *v; SIMPLEX *e; int i; FLOAT d;

a = phgDofNew(u->g, &type, 1, "average", DofNoAction);
m = phgMapCreate(a, NULL);
v = phgMapCreateVec(m, 1);
ForAllElements(g, e) {
    for (i = 0; i < NFace; i++) {
        d = *DofElementData(u, e->index);
        if (边界面) {
            if (!子网格拥有该面) continue;
            phgVecAddEntry(v, 0, e->faces[i], d);
        } else
```

```
                phgVecAddEntry(v, 0, e->faces[i], 0.5 * d);
        }
    }
    phgVecAssemble(v);
    phgMapVecToDofArrays(m, v, FALSE, a, NULL);
    phgVecDestroy(&v); phgMapDestroy(&m);
```

§8.3 矩阵 (MAT)

```
MAT *phgMapCreateMat(MAP *rmap, MAP *cmap);
void phgMatDestroy(MAT **Mat);

MAP *phgMatGetRowMap(MAT *mat);
MAP *phgMatGetColumnMap(MAT *mat);

phgMatAddEntry          phgMatAddEntries
phgMatAddGlobalEntry    phgMatAddGlobalEntries
phgMatAddGLEntry        phgMatAddGLEntries
phgMatAddLGEntry        phgMatAddLGEntries

phgMatSetEntry          phgMatSetEntries
phgMatSetGlobalEntry    phgMatSetGlobalEntries
phgMatSetGLEntry        phgMatSetGLEntries
phgMatSetLGEntry        phgMatSetLGEntries

void phgMatAssemble(MAT *mat);
MAT *phgMatRemoveBoundaryEntries(MAT *mat);
```

与映射类似，MAT 中保存一个引用计数 (主要供不同解法器引用同一矩阵)。只有当引用计数为 0 时 phgMatDestroy 才消毁矩阵对象。

§8.3.1 Dirichlet 边界条件

如果一个矩阵对象中的 BOOLEAN 成员 handle_bdry_eqns 的值为 TRUE，则 phgMatAssemble 函数会根据自由度对象中的 DB_mask 成员确定哪些元素对应于 Dirichlet 边界自由度，在组装前将这些元素对应的行、列保存在其它地方，然后对矩阵中的这些行列进行对角化处理，这样的处理有助于保持矩阵的对称性。

phgSolverAssembleRHS 函数在组装右端项之前利用保存的边界行列先将对应于 Dirichlet 边界自由度的未知量求解出来，并相应更新方程的右端项。

一个普通矩阵的 handle_bdry_eqns 成员的值通常为 FALSE，而通过函数 phgSolverCreate 所创建的矩阵的 handle_bdry_eqns 成员的值缺省为 TRUE。

§8.3.2 “无矩阵”矩阵

PHG 支持一类特殊的矩阵，称为 “matrix-free” matrix，其中用户不用提供矩阵元素，而只需提供一个负责完成矩阵与向量乘积的函数 (MV_FUNC)。

```
typedef int (*MV_FUNC)(MAT_OP op, MAT *A, VEC *x, VEC *y);
MAT *phgMapCreateMatrixFreeMat(MAP *rmap, MAP *cmap,
                MV_FUNC mv_func, void *mv_data, ...);
```

函数 MV_FUNC 应该处理 MAT_OP 等于 MAT_OP_N (矩阵乘以向量)、MAT_OP_T (矩阵转置乘以向量) 和 MAT_OP_D (矩阵的对角线部分乘以向量) 三种情况。

```
static int mat_vec(MAT_OP op, MAT *mat, VEC *x, VEC *y)
{
    switch (op) {
        case MAT_OP_N: ...; break;      /* y := A * x */
        case MAT_OP_T: ...; break;      /* y := trans(A) * x */
        case MAT_OP_D: ...; break;      /* y := diag(A) * x */
    }
    return 0;
}
```

§8.4 MAT、VEC 与解法器

下述函数负责矩阵与解法器之间的转换：

```
SOLVER *phgMat2Solver(OEM_SOLVER *oem_solver, MAT *mat);
SOLVER *phgSolverMat2Solver(OEM_SOLVER *oem_solver, MAT *mat);
MAT *phgSolverGetMat(SOLVER *solver);
```

函数 phgSolverGetMat 会使得矩阵的引用计数增加 1, 这是它与直接访问 mat 成员的主要区别。

例：下面的调用是错误的：

```
    SOLVER *solver = phgSolverCreate(......);
    MAT *mat = solver->mat;
    ... ...
    phgSolverDestroy(&solver);
    ... ...
    phgMatDestroy(&mat);
```

正确的调用应该是：

```
    SOLVER *solver;
    MAT *mat;
    ... ...
    SOLVER *solver = phgSolverCreate(......);
    MAT *mat = phgSolverGetMat(solver);
    ... ...
    phgSolverDestroy(&solver);
    ... ...
    phgMatDestroy(&mat);
```

第九章　特征值、特征向量计算

```
int phgEigenSolve(MAT *A, MAT *B, int n, int which,
                  FLOAT tau, FLOAT *evals, VEC **evecs, int *nit);
int phgDofEigenSolve(MAT *A, MAT *B, int n, int which,
                  FLOAT tau, int *nit, FLOAT *evals, MAP *map,
                  DOF **u, ...);
```

上述函数计算广义特征值问题 $Ax = \lambda Bx$ 的 n 个特征对。参数 `which` 可取为 `EIGEN_SMALLEST`、`EIGEN_LARGEST` 或 `EIGEN_CLOSEST`。

许多求解参数可以通过命令行选项或相应的接口设定。

§9.1　特征值问题求解器

§9.1.1　PARPACK

§9.1.2　JDBSYM

§9.1.3　BLOPEX (LOBPCG)

§9.1.4　Trilinos/Anasazi

§9.1.5　SLEPc

§9.1.6　PRIMME

附录一　程序实例

目录 examples 中提供了一些简单的自适应计算程序实例，包括：

simplest.c	Poisson 方程，2 阶 Lagrange 元
poisson.c	Poisson 方程，任意 H^1 元
eigen.c	Laplace 算子的特征值和特征向量
heat.c	热传导方程
non-smooth.c	二阶间断系数椭圆型问题
elastic.c	弹性力学方程
maxwell.c	时谐 Maxwell 方程，任意阶棱单元
maxwell-eigen.c	时谐 Maxwell 方程，特征值问题

另外，test 目录中有一些测试程序。阅读、运行它们有助于理解 PHG 的基本数据对象及操作。

§A.1　Poisson 方程

Dirichlet 边值问题：$\begin{cases} -\Delta u = f & x \in \Omega \\ u = g & x \in \partial\Omega \end{cases}$

§A.1.1　有限元离散

弱形式：给定协调四面体网格 Ω_h 及有限元空间 V_h，求 $u_h \in V_h$ 满足：

$$\int_{\Omega_h} \nabla u_h \cdot \nabla v_h \,\mathrm{d}x = \int_{\Omega_h} f v_h \,\mathrm{d}x, \quad \forall v_h \in V_h$$

这里，为简单起见，没有涉及到边界条件的处理。

设 $\{\varphi_i \mid i = 0, \ldots, N-1\}$ 为 V_h 的基函数，$u_h = \sum_{i=0}^{N-1} u_i \varphi_i$，则：

$$\sum_{i=0}^{N-1} \left(\int_{\Omega_h} \nabla\varphi_i \cdot \nabla\varphi_j \,\mathrm{d}x \right) u_i = \int_{\Omega_h} f\varphi_j \,\mathrm{d}x, \quad j = 0, \ldots, N-1$$

记

$$A = [a_{i,j}]_{N\times N}, \quad a_{i,j} = \int_{\Omega_h} \nabla\varphi_i \cdot \nabla\varphi_j \,\mathrm{d}x,$$
$$b = [b_0, \ldots, b_{N-1}]^T, \quad b_j = \int_{\Omega_h} f\varphi_j \,\mathrm{d}x,$$
$$u_h = [u_0, \ldots, u_{N-1}]^T,$$

则可得到有限元解 u_h 所满足的线性方程组：

$$Au_h = b$$

设 e 为 Ω_h 中的单元，记 e 中的局部基函数为 φ_i^e, $i = 0, \ldots, n-1$。令：

$$A^e = [a_{i,j}^e]_{n\times n}, \quad a_{i,j}^e = \int_e \nabla\varphi_i^e \cdot \nabla\varphi_j^e \,\mathrm{d}x,$$
$$b^e = [b_0^e, \ldots, b_{n-1}^e]^T, \quad b_j^e = \int_e f\varphi_j^e \,\mathrm{d}x$$

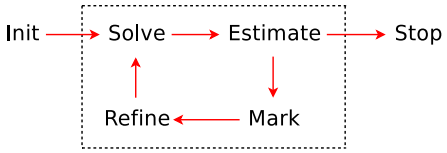
易知：$A = \sum_{e\in\Omega_h} P^e A^e$, $\quad b = \sum_{e\in\Omega_h} P^e b^e$, 其中 P^e 对应于单元基函数编号到全局基函数编号映射的变换矩阵。

§A.1.2 后验误差估计子

$$\eta^2 = h_e^2 \left|\Delta u_h + f\right|_e^2 + \frac{1}{2} \sum_{f\in F(e)\cap F(\Omega_h)} h_f \left|[\nabla u_h \cdot \boldsymbol{n}_f]_f\right|_f^2$$

其中 $F(e)$ 表示单元 e 的面的集合，$F(\Omega_h)$ 表示所有非边界面的集合，$\boldsymbol{n}_f$ 表示面 f 的法向量，h_e 代表单元 e 的直径，h_f 代表面 f 的直径，$[\cdot]_f$ 表示跨过面 f 的跳量，$|\cdot|_e$ 表示单元 e 上的 L_2 模，$|\cdot|_f$ 表示面 f 上的 L_2 模。

§A.1.3 自适应计算流程



§A.2 时谐 Maxwell 方程

$$\begin{cases} \nabla\times \mu^{-1}\nabla\times \boldsymbol{E} - k^2\boldsymbol{E} = \boldsymbol{J}, & \Omega \\ \boldsymbol{E}\times\boldsymbol{n} = 0, & \partial\Omega \end{cases}$$

这里 $k^2 = \omega^2\varepsilon - i\sigma\omega$。其中 $\boldsymbol{E}$ 为电场强度, $\boldsymbol{J}$ 为电流密度, ε 为介电常数, μ 为磁导率, σ 为电导率, ω 为角频率。

§A.2.1 有限元离散

$$a_{i,j}^e = \int_e \mu^{-1}\nabla\times\boldsymbol{\varphi}_i^e \cdot \nabla\times\boldsymbol{\varphi}_j^e \,\mathrm{d}x - k^2\int_e \boldsymbol{\varphi}_i^e\cdot\boldsymbol{\varphi}_j^e \,\mathrm{d}x$$
$$b_j^e = \int_e \boldsymbol{J}\cdot\boldsymbol{\varphi}_j^e \,\mathrm{d}x$$

注意基函数 $\boldsymbol{\varphi}_j^e$ 是矢量函数，可以用 Nédélec 或其它 $\boldsymbol{H}_{\mathrm{curl}}$ 元 (`DOF_HCn`)。

§A.2.2 后验误差估计子

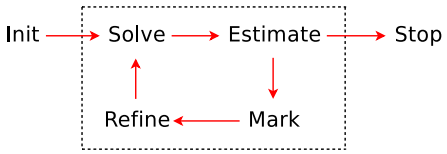
$$\eta^2 = \eta_e^2 + \sum_{f\in F(e)\cap F(\Omega_h)} \eta_f^2$$

其中：

$$\eta_e^2 = h_e^2\left(\left|\boldsymbol{J} + k^2\boldsymbol{E} - \nabla\times\mu^{-1}\nabla\times\boldsymbol{E}\right|_e^2 + \left|\operatorname{div}(\boldsymbol{J} + k^2\boldsymbol{E})\right|_e^2\right)$$

$$\eta_f^2 = h_f\left(\left|[(\mu^{-1}\nabla\times\boldsymbol{E})\times\boldsymbol{n}]\right|_f^2 + \left|[(\boldsymbol{J} + k^2\boldsymbol{E})\cdot\boldsymbol{n}]\right|_f^2\right)$$

§A.2.3 自适应计算流程



§A.3 特征值问题

$$\begin{cases} -\Delta u = \lambda u & x \in \Omega \\ u = 0 & x \in \partial\Omega \end{cases}$$

当 $\Omega = (0,1)^3$ 时，该特征值问题的解析解为：

$$\begin{cases} \lambda_{k_1,k_2,k_3} = (k_1^2 + k_2^2 + k_3^2)\pi^2, \\ u_{k_1,k_2,k_3} = \sin(k_1\pi x)\sin(k_2\pi y)\sin(k_3\pi z), \\ \qquad k_i \in \{1,2,\ldots\}, \quad i = 1,2,3 \end{cases}$$

§A.3.1 有限元离散

上述问题的有限元离散为下列广义代数特征值问题

$$Au_h = \lambda_h B u_h$$

其中 A 和 B 均为对称正定矩阵，$A = \sum_{e\in\Omega_h} P^e A^e$, $B = \sum_{e\in\Omega_h} P^e B^e$, A^e、B^e 分别为单元刚度矩阵和单元质量矩阵：

$$A^e = [a_{i,j}^e]_{n\times n}, \quad a_{i,j}^e = \int_e \nabla\varphi_i^e\cdot\nabla\varphi_j^e\,\mathrm{d}x,$$

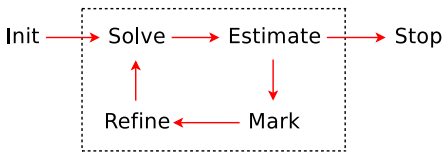
$$B^e = [b_{i,j}^e]_{n\times n}, \quad b_{i,j}^e = \int_e \varphi_i^e\varphi_j^e\,\mathrm{d}x$$

注意，矩阵 A 和 B 中不含边界上的自由度。

§A.3.2 后验误差估计子

$$\eta^2 = h_e^2\left|\Delta u_h + \lambda_h u_h\right|_e^2 + \frac{1}{2}\sum_{f\in F(e)\cap F(\Omega_h)} h_f\left|[\bigtriangledown u_h\cdot\boldsymbol{n}_f]_f\right|_f^2$$

§A.3.3 网格自适应



如果知道哪些需要计算的特征向量具有奇性，可以使用它们的后验误差估计子做为网格自适应的指标。也可以使用所有所计算的特征向量、特征值的后验误差估计子。