HiperLife Tutorial: Poisson Equation

LaCàN

August 19, 2024

1 Introduction

1.1 Problem Definition

Poisson Equation is a simple elliptic model, given by

$$-\Delta U = -\nabla^2 U = -\frac{\partial^2 U}{\partial x_1^2} - \frac{\partial^2 U}{\partial x_2^2} = f \tag{1}$$

We will use this equation in this example for introducing the implemention of finite element method in the HiperLife. Notice that here we have used $f = 0$.

1.2 Boundary Condition

As shown in Figure 1, We have used homogeneous Dirichlet ($U = 0, \quad U = 1$) along the lines ($x_1 = 0, \quad x_1 = 1$), and along the lines ($x_2 = 0.5, \quad x_2 = 1$) we applied Inhomogenous Dirichelet ($U = x_1, \quad U = x_1^2$), respectively.

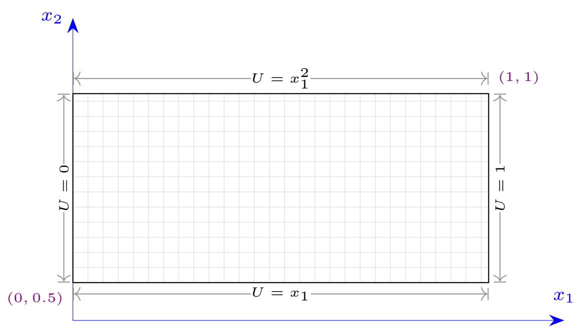

Figure 1: Illustration of the model in $\Omega = [0, 1] \times [0.5, 1]$ and Boundary conditions on $\partial\Omega$.

2 Formulation

2.1 Weak Form

To establish the weak form of Eq. (1), it is multiplied with a weight-function, $w(x, y)$ to obtain

$$-w\nabla^2 U = wf \tag{2}$$

By integrating this expression over Ω, we have

$$-\int_\Omega w\nabla^2 U = \int_\Omega wf \tag{3}$$

We know from calculus that $\nabla(w\nabla U) = \nabla w \cdot \nabla U + w\nabla^2 U$. So we can write

$$-\int_\Omega w\nabla^2 U = \int_\Omega \nabla(w\nabla U) - \int_\Omega \nabla w \cdot \nabla U = \int_\Omega wf dA \tag{4}$$

Using Gauss's theorem we get

$$\int_\Omega \nabla(w\nabla U) = \int_{\partial\Omega} w\nabla U \cdot n dS = 0 \tag{5}$$

Eq. (4) now reduces to

$$\int_\Omega w\nabla^2 U = -\int_\Omega \nabla w \cdot \nabla U \tag{6}$$

so we get

$$\int_\Omega \nabla w \cdot \nabla U dA = \int_\Omega wf dA \tag{7}$$

2.2 Basis Function

We need to define basis functions for our 2D-domain and by it we can give an approximation of U.

$$U(x, y) = \sum_{i=1}^{n} u_i \phi_i(x_1, x_2) \tag{8}$$

by applying Gelerkin method the weight function is the same as basis function.

$$w_i = \phi_i \tag{9}$$

in isoparametric concept even geometry is interpolated by same function. so

$$X(x_1, x_2) = \sum_{i=1}^{n} x_i \phi_i(x_1, x_2) \tag{10}$$

2.3 Element

Quadrilateral Elements is the simplest quadrilateral element consists of four nodes. The associated interpolation functions for geometry and field variables are bilinear. Let $\phi_I = N_I$ at element T.

$$N_I(\xi, \eta) = \frac{1}{4}(1 + \xi_I \xi)(1 + \eta_I \eta) \quad (I \text{ from 1 to 4}) \tag{11}$$

where ξ_I and η_I are the corner coordinates at element T in domain of $\Omega_T \in (-1, 1)^2$.

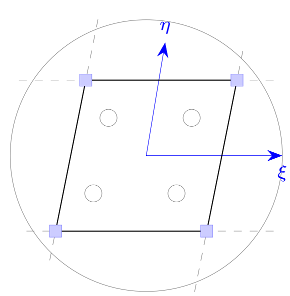

Figure 2: Schematic of an Element.

In this example each node only have one degree of freedom and for the purpose of discretization we use 500×500 uniform mesh.

2.4 Elemental Integral

We want to compute the integral at Eq. (7) over the element T. We assumed $U = N_I U_I$, and $w = N$, so $\nabla U = \frac{\partial N_I}{\partial x_i} U_I \otimes \mathbf{e}_i$, and $\nabla w = \frac{\partial N_J}{\partial x_j} \otimes \mathbf{e}_j$, so

$$\int_{\Omega_T} \frac{\partial N_I}{\partial x_i} \frac{\partial N_J}{\partial x_j} U_I \delta_{ij} \; dV = \int_{\Omega_T} \frac{\partial N_I}{\partial x_i} \frac{\partial N_J}{\partial x_i} U_I \; dV = \int_{\Omega_T} (\frac{\partial N_I}{\partial x_1} \frac{\partial N_J}{\partial x_1} + \frac{\partial N_I}{\partial x_2} \frac{\partial N_J}{\partial x_2}) U_I dA = \int_{\Omega_T} N_J f dA \tag{12}$$

by the linear mapping between (ξ, η) and (x_1, x_2), we can define $\frac{\partial N}{\partial x_1}$ and $\frac{\partial N}{\partial x_2}$

$$\begin{aligned} \frac{\partial N}{\partial \xi} &= \frac{\partial N}{\partial x_1} \frac{\partial x_1}{\partial \xi} + \frac{\partial N}{\partial x_2} \frac{\partial x_2}{\partial \eta} \\ \frac{\partial N}{\partial \eta} &= \frac{\partial N}{\partial x_1} \frac{\partial x_1}{\partial \xi} + \frac{\partial N}{\partial x_2} \frac{\partial x_2}{\partial \eta} \end{aligned} \tag{13}$$

Since $x = x(\xi, \eta)$, we get

$$\begin{bmatrix} dx_1 \\ \\ dx_2 \end{bmatrix} = \begin{bmatrix} \frac{\partial x_1}{\partial \xi} & \frac{\partial x_2}{\partial \xi} \\ \\ \frac{\partial x_1}{\partial \eta} & \frac{\partial x_2}{\partial \eta} \end{bmatrix} \begin{bmatrix} d\xi \\ \\ d\eta \end{bmatrix} \tag{14}$$

by defining Jacobian as

$$J = \begin{bmatrix} \frac{\partial x_1}{\partial \xi} & \frac{\partial x_2}{\partial \xi} \\ \\ \frac{\partial x_1}{\partial \eta} & \frac{\partial x_2}{\partial \eta} \end{bmatrix} \tag{15}$$

so we can rewrite the Eq. (13)

$$\begin{bmatrix} \frac{\partial N}{\partial x_1} \\ \\ \frac{\partial N}{\partial x_2} \end{bmatrix} = J^{-1} \begin{bmatrix} \frac{\partial N}{\partial \xi} \\ \\ \frac{\partial N}{\partial \eta} \end{bmatrix} \tag{16}$$

by obtaining the terms Eq (12) now we can calculate the integral. So the elements of tangent matrix K could be defined as

$$K(i,j) = \int_{\Omega_T} (\frac{\partial N_i}{\partial x_1} \frac{\partial N_j}{\partial x_1} + \frac{\partial N_i}{\partial x_2} \frac{\partial N_j}{\partial x_2}) dA \tag{17}$$

and for external source

$$F(i) = \int_{\Omega_T} N_i f dA \tag{18}$$

keep in mind that in our case $f = 0$. Note that $dA = dx_1 \times dx_2 = j d\xi d\eta$, which $j = det(J)$.

2.5 Integration method

Let $f(\xi_i, \eta_i) = j(\frac{\partial N_i}{\partial x_1} \frac{\partial N_j}{\partial x_1} + \frac{\partial N_i}{\partial x_2} \frac{\partial N_j}{\partial x_2})$, then by Gauss–Legendre quadrature we have

$$\int_{-1}^{1} \int_{-1}^{1} f(\xi_i, \eta_i) d\xi d\eta = \sum_{k=1}^{n} \sum_{l=1}^{m} \omega_k \omega_l f(\hat{\xi}_i, \hat{\eta}_i) \tag{19}$$

where n and m are the number of integration points used in each direction, ω is quadrature weights, $\hat{\xi}_i$ and $\hat{\eta}_i$ are the roots of the nth Legendre polynomial. In this example we use 2×2 integration as it shown in Figure 3. Which the value of quadrature weights are the same $\omega_p = 1$, and coordinate of guass points would be: $(\xi_p, \eta_p) = \{(-\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}), (\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}}), (-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}), (\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}})\}$

3 Implementation

In this section, we present the implementation of solution in the Hiperlife.

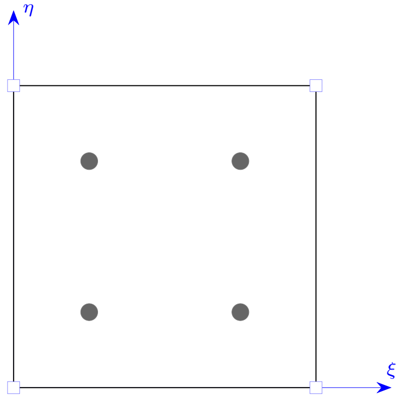

Figure 3: 2D integration on Quadrilateral Element.

3.1 Flow chart

The Flowchart of a typical HiperLife program is shown in Figure 4.

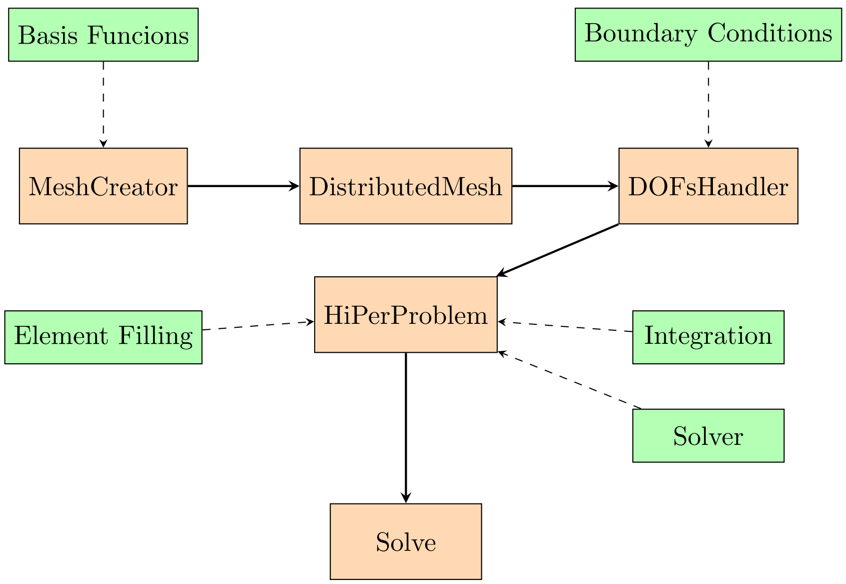

Figure 4: Flow chart.

3.2 Headers

```
// hiperlife(HL) headers
#include "hl_Core.h"
#include "hl_ParamStructure.h"
#include "hl_Parser.h"
#include "hl_TypeDefs.h"            // HL header that defines different data
                                    // types used throughout the library

#include "hl_GlobalBasisFunctions.h" // HL header that defines
                                    // auxiliary geometric-related functions

#include "hl_StructMeshGenerator.h"  // HL header that defines structured mesh objects
#include "hl_DistributedMesh.h"      // HL header that defines distributed mesh objects
#include "hl_FillStructure.h"        // HL header that defines objects to fill  the matrix
```

```
                                        // & vector of the problem

#include "hl_DOFsHandler.h"             // HL header that defines DOFHandler objects
#include "hl_HiPerProblem.h"            // HL header that defines HiPerProblem objects

#include "hl_LinearSolver_Iterative_AztecOO.h"
#include "hl_LinearSolver_Direct_MUMPS.h"
```

3.3 Parameters

```
struct PoissonParams
{
        enum RealParameters
        {
                force
        };
        HL_PARAMETER_LIST DefaultValues
        {
                {"force", 0.0},
        };
};
```

3.4 Initializing

Function that fills the elemental linear system of the problem. It is declared here but implemented at the bottom.

```
void ElementFilling(hiperlife::FillStructure& fillStr);
```

Main part

```
int main(int argc, char** argv)
{
        using namespace std;
        using namespace hiperlife;
        hiperlife::Init(argc, argv);
```

3.5 Defining parameters of the model

```
        SmartPtr<ParamStructure> paramStr = CreateParamStructure<PoissonParams>();
```

3.6 Mesh Generation

creating a model geometry, which is a rectangle and translate it in y-direction, and also introducing mesh. the header declaration of function can be find at Header 1.

```
        SmartPtr<StructMeshGenerator> structMesh = Create<StructMeshGenerator>();

        structMesh->setNDim(3);
        structMesh->setBasisFuncType(BasisFuncType::Lagrangian);
        structMesh->setBasisFuncOrder(1);
        structMesh->setElemType(ElemType::Square);
        structMesh->genRectangle(500, 500, 1.0, 0.5);
        structMesh->translateY(0.5);
```

3.7 Mesh Distribution

```
        SmartPtr<DistributedMesh> disMesh = Create<DistributedMesh>();

        disMesh->setMesh(structMesh);
        disMesh->setBalanceMesh(true);

        disMesh->Update();
```

3.8 DOFs Handler

```
        SmartPtr<DOFsHandler> dofHand = Create<DOFsHandler>(disMesh);

        dofHand->setNameTag("dofHand");
        dofHand->setNumDOFs(1);

        dofHand->Update();
```

3.9 Boundary conditions

applying boundary condition, as it declared in Header 2.

```
        dofHand->setBoundaryCondition(0, 0.0);
        dofHand->setBoundaryCondition(0, MAxis::Xmax, 1.0);
        dofHand->setBoundaryCondition(0, MAxis::Ymin, [](double x){return x;});
        dofHand->setBoundaryCondition(0, MAxis::Ymax, [](double x){return x*x;});

        dofHand->UpdateGhosts();
```

3.10 HiperProblem

```
        SmartPtr<HiPerProblem> hiperProbl = Create<HiPerProblem>();

        hiperProbl->setParameterStructure(paramStr);
        hiperProbl->setDOFsHandlers({dofHand});
        hiperProbl->setIntegration("Integ", {"dofHand"});
        hiperProbl->setCubatureGauss("Integ", 4);
        hiperProbl->setElementFillings("Integ", ElementFilling);

        hiperProbl->Update();
```

3.11 Solver Settings

```
        SmartPtr<AztecOOIterativeLinearSolver> solver=Create<AztecOOIterativeLinearSolver>();

        solver->setHiPerProblem(hiperProbl);
        solver->setSolver(AztecOOIterativeLinearSolver::Solver::Gmres);
        solver->setPrecond(AztecOOIterativeLinearSolver::Precond::DomainDecomp);
        solver->setSubdomainSolve(AztecOOIterativeLinearSolver::SubdomainSolve::Ilut);
        solver->setVerbosity(AztecOOIterativeLinearSolver::Verbosity::None);

        solver->setDefaultParameters();
        solver->Update();
        solver->solve();

        hiperProbl->UpdateSolution();
```

3.12 Finalization and Postprocessing

```
        dofHand->printFileLegacyVtk("Poisson");

        hiperlife::Finalize();
        return 0;
}
```

3.13 Element Filling function

```
void ElementFilling(hiperlife::FillStructure& fillStr)
{
        using namespace std;
        using namespace hiperlife;
        using namespace hiperlife::Tensor;

        const double force = fillStr.getRealParameter(PoissonParams::force);

        SubFillStructure& subFill = fillStr["dofHand"];

        int eNN  = subFill.eNN;
        int pDim = subFill.pDim;

        wrapper<double,1>  bf(subFill.nborBFs(), eNN);

        tensor<double,2> Dbf_g(eNN,pDim);
        double jac;
        GlobalBasisFunctions::gradients(Dbf_g, jac, subFill);

        wrapper<double,2> Ak(fillStr.Ak(0, 0).data(),eNN,eNN);
        wrapper<double,1> Bk(fillStr.Bk(0).data(),eNN);

        for (int i = 0; i < eNN; i++)
        {
                for (int j = 0; j < eNN; j++)
                {
                        double dotij{};
                        for (int d = 0; d < pDim; d++)
                        dotij += Dbf_g(i,d)*Dbf_g(j,d);

                        Ak(i,j) += jac * dotij;
                }

                Bk(i) += jac * bf(i) * force;
        }

}
```

4 Results

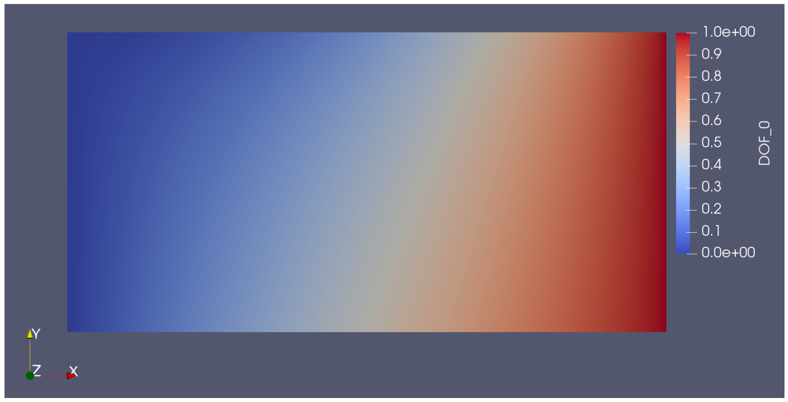

Figure 5: Solution.

Appendix

In this section we present the declaration of the used functions in headers or even their code for the purpose of clarification.

For creating a rectangle and Translation in y-direction:

```
// Generate a rectangle with nEx*nEy elements for given side lengths sidex and sidey.
void genRectangle(int nEx, int nEy, double sidex, double sidey);

//Translate mesh in y direction by an increment (incrY)
inline void translateY(double incrY){translate(0.0, incrY, 0.0);};
```

Header 1: geometry

For boundary condition:

```
// Set a constraint value at every axis for one dof
void setBoundaryCondition(int dof, double value)

// Set a constraint function at one axis for one dof in 2D
void setBoundaryCondition(int dof, MAxis ax, std::function<double(double)> f)
```

Header 2: B.C