HiperLife Tutorial: PoissonSurfaceNeumann

LaCàN

August 16, 2024

1 Introduction

1.1 Problem Definition

The Poisson equation is the canonical elliptic PDE. For a domain $\Omega \subset \mathbb{R}^n$ with boundary $\partial\Omega$, the Poisson equation is like:

$$\begin{aligned} -\Delta U &= f \quad \text{in } \Omega \\ \nabla U \cdot n &= g \quad \text{on } \partial\Omega \end{aligned} \tag{1}$$

Here, f and g are input data which in this case we assume that $f = 1$ and $g = x_2 \cos(3\pi x_1)$, and n denotes the normal vector of boundary.

1.2 Boundary Condition

We have used Neumann Condition along the lines $x_2 = 0$, and $x_1 = 1$, as depicted in Figure 1.

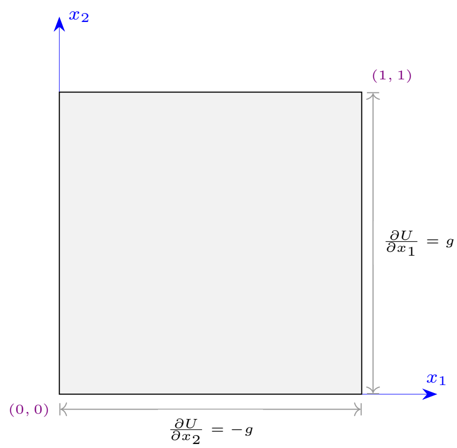


Figure 1: Illustration of Geometry in $\Omega = [0,1] \times [0,1]$ and Boundary conditions on $\partial\Omega$.

2 Formulation

2.1 Weak Form

To establish the weak form of Eq. (1), it is multiplied with a weight-function, $w(x_1, x_2)$ to obtain

$$\begin{aligned} -w\nabla^2 U &= wf \quad \text{in } \Omega \\ w\nabla U \cdot n &= wg \quad \text{on } \partial\Omega \end{aligned} \tag{2}$$

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

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

according to Gauss's theorem

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

by applying Eq. (5) to Eq. (4)

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

the right-hand side of the Eq. (5) is the second part of Eq. (2), So now we can rewrite the Eq. (4) like this

$$\int_{\Omega} \nabla w \cdot \nabla U dA = \int_{\Omega} wf dA + \int_{\partial\Omega} wg dS \tag{7}$$

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 Galerkin 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

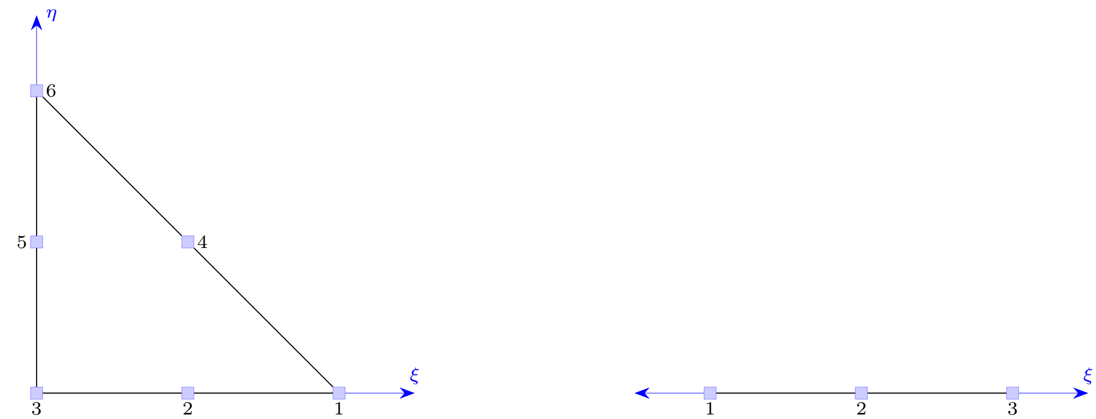
Triangular Element is used for discretization of bulk in this example, as shown in Fig. 2a,

The quadratic element consists of 9 nodes would have interpolation functions for geometry and field variables like this. Let $\phi_I = N_I$ at element T.

$$\begin{array}{ll} N_1(\xi,\eta) = \xi(2\xi - 1) & N_2(\xi,\eta) = 4\xi(1-\xi-\eta) \\ N_3(\xi,\eta) = (1-\xi-\eta)(1-2\xi-2\eta) & N_4(\xi,\eta) = 4\xi\eta \\ N_5(\xi,\eta) = 4\eta(1-\xi-\eta) & N_6(\xi,\eta) = \eta(2\eta-1) \end{array} \tag{11}$$

For the purpose of discretization of the boder in this example, 3-node line element is used, as shown in Fig. 2b. which its interpolation functions would be like this.

$$N_1(\xi,\eta) = \frac{1}{2}\xi(\xi-1) \quad N_2(\xi,\eta) = 4\xi(1-\xi^2) \quad N_3(\xi,\eta) = \frac{1}{2}\xi(\xi+1) \tag{12}$$



(a) (b)

Figure 2: Schematic of the (a) bulk element (b) border element

2.4 Elemental Integral

We want to compute the integral at Eq. (7) over the element T

$$\int_{\Omega_T} \left(\frac{\partial N_I}{\partial x_1}\frac{\partial N_J}{\partial x_1} + \frac{\partial N_I}{\partial x_2}\frac{\partial N_J}{\partial x_2}\right) U dA = \int_{\Omega_T} N_J f dA + \int_{\partial\Omega} N_J^{border} g dS \tag{13}$$

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

$$\begin{aligned} \frac{\partial N}{\partial \xi} &= \frac{\partial N}{\partial x_1}\frac{\partial x_1}{\partial \xi} + \frac{\partial N}{\partial x_2}\frac{\partial x_2}{\partial \eta} \\ \frac{\partial N}{\partial \eta} &= \frac{\partial N}{\partial x_1}\frac{\partial x_1}{\partial \xi} + \frac{\partial N}{\partial x_2}\frac{\partial x_2}{\partial \eta} \end{aligned} \tag{14}$$

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

$$\begin{bmatrix} dx_1 \\ dx_2 \end{bmatrix} = \begin{bmatrix} \frac{\partial x_1}{\partial \xi} & \frac{\partial x_2}{\partial \xi} \\ \frac{\partial x_1}{\partial \eta} & \frac{\partial x_2}{\partial \eta} \end{bmatrix} \begin{bmatrix} d\xi \\ d\eta \end{bmatrix} \tag{15}$$

by defining Jacobian as

$$J = \begin{bmatrix} \frac{\partial x_1}{\partial \xi} & \frac{\partial x_2}{\partial \xi} \\ \frac{\partial x_1}{\partial \eta} & \frac{\partial x_2}{\partial \eta} \end{bmatrix} \tag{16}$$

so we can rewrite the Eq. (13)

$$\begin{bmatrix} \frac{\partial N}{\partial x_1} \\ \frac{\partial N}{\partial x_2} \end{bmatrix} = J^{-1} \begin{bmatrix} \frac{\partial N}{\partial \xi} \\ \frac{\partial N}{\partial \eta} \end{bmatrix} \tag{17}$$

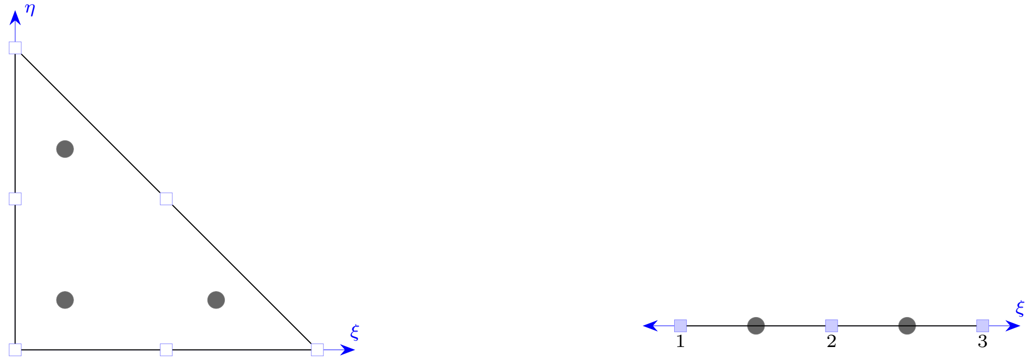
by obtaining the terms Eq. (12) now we can calculate the integral. So the elements of tangent matrix K could be defined as

$$K(i,j) = \int_{\Omega_T} \left(\frac{\partial N_i}{\partial x_1}\frac{\partial N_j}{\partial x_1} + \frac{\partial N_i}{\partial x_2}\frac{\partial N_j}{\partial x_2}\right) dA \tag{18}$$

and for right-hand side

$$F(i) = \int_{\Omega_T} N_i^{bulk} f dA + \int_{\partial\Omega} N_i^{border} g dS \tag{19}$$

Note that as it discussed earlier, different types of elements is used for bulk and border. At last we define j as $j = det(J)$ because $dA = dx_1 \times dx_2 = j d\xi d\eta$.



(a) (b)

Figure 3: (a) 2-D integration on Triangle element (b) 1-D integration on line element

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

$$\int_{-1}^{1}\int_{-1}^{1} f(\xi_i, \eta_i) d\xi d\eta = \sum_{k=1}^{n}\sum_{l=1}^{m} \omega_k \omega_l f(\hat{\xi_i}, \hat{\eta_i}) \tag{20}$$

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 3 points integration as it shown in Fig. 3a. $[W_p = \{\frac{1}{6}, \frac{1}{6}, \frac{1}{6}\}, (\xi_p, \eta_p) = \{(\frac{1}{6}, \frac{1}{6}), (\frac{1}{6}, \frac{2}{3}), (\frac{2}{3}, \frac{1}{6})\}]$. At the border we use 2 points integration as it shown in Fig. 3b. $[W_p = 1, \xi_p = \mp\frac{1}{\sqrt{3}}]$

3 Implementation

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

3.1 Headers

```
#include <iostream>
#include <fstream>
#include "hl_Core.h"
#include "hl_ParamStructure.h"
#include "hl_Parser.h"
#include "hl_TypeDefs.h"
#include "hl_MeshLoader.h"
#include "hl_StructMeshGenerator.h"
#include "hl_DistributedMesh.h"
#include "hl_FillStructure.h"
#include "hl_DOFsHandler.h"
#include "hl_HiPerProblem.h"
#include "hl_SurfLagrParam.h"
#include "hl_LinearSolver_Iterative_AztecOO.h"
```

3.2 Parameters

```
struct PoissonParams
{
        enum RealParameters
        {
                strength
```

```
        };

        enum StringParameters
        {
                filemesh
        };

        HL_PARAMETER_LIST DefaultValues{
                {"filemesh", ""},
                {"strength", 1.0},
        };
};
```

3.3 Initializing

```
void LS(hiperlife::FillStructure& fillStr);
void RHS_Border(hiperlife::FillStructure& fillStr);

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

3.4 Defining parameters of the model

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

3.5 Mesh Generation

```
        SmartPtr<StructMeshGenerator> mesh = Create<StructMeshGenerator>();
        mesh->setElemType(ElemType::Triang);
        mesh->setBasisFuncType(BasisFuncType::Lagrangian);
        mesh->setBasisFuncOrder(2);
        mesh->genSquare(4,1.0);
```

3.6 Mesh Distribution

```
        SmartPtr<DistributedMesh> disMesh = Create<DistributedMesh>();
        disMesh->setMesh(mesh);
        disMesh->setBalanceMesh(true);
        disMesh->Update();
```

3.7 DOFs Handler

```
        SmartPtr<DOFsHandler> dofHand = Create<DOFsHandler>(disMesh);
        dofHand->setNameTag("dofHand");
        dofHand->setNumDOFs(1);
        dofHand->setNumElemAuxF(1);
        dofHand->Update();
```

3.8 Boundary conditions and Constraints

```
        if (!disMesh->hasBorder())
        {
                if (disMesh->myRank() == 0)
                dofHand->setConstraint(0, 0, IndexType::Local, 0.0);
        }
        else
        {
                for (int i = 0; i < disMesh->loc_nPts(); i++)
                {
                        if (disMesh->nodeCrease(i, IndexType::Local) > 0 and
                        disMesh->nodeCoord(i, 1, IndexType::Local) > 0)
                        dofHand->setConstraint(0,i,IndexType::Local, 0.0);
                }
        }
        dofHand->UpdateGhosts();
```

3.9 HiperProblem

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

        hiperProbl->setParameterStructure(paramStr);
        hiperProbl->setDOFsHandlers({dofHand});

        hiperProbl->setIntegration("Integ", {"dofHand"});
        hiperProbl->setCubatureGauss("Integ",3);
        hiperProbl->setElementFillings("Integ", LS);

        hiperProbl->setIntegration("BorderInteg", {"dofHand"});
        hiperProbl->setCubatureBorderGauss("BorderInteg",2,{MAxis::Xmax,MAxis::Ymin});
        hiperProbl->setElementFillings("BorderInteg", nullptr, nullptr, RHS_Border);

        hiperProbl->setGlobalIntegrals({"BorderLength"});

        hiperProbl->Update();
```

3.10 Solver Settings

```
        SmartPtr<AztecOOIterativeLinearSolver> solver=Create<AztecOOIterativeLinearSolver>();
        solver->setHiPerProblem(hiperProbl);
        solver->setTolerance(1.E-8);
        solver->setMaxNumIterations(500);
        solver->setSolver(AztecOOIterativeLinearSolver::Solver::Gmres);
        solver->setPreconditioner(AztecOOIterativeLinearSolver::Preconditioner::Neumann);
        solver->setDefaultParameters();
        solver->setVerbosity(AztecOOIterativeLinearSolver::Verbosity::High);
        solver->Update();

        solver->solve();
        solver->UpdateSolution();
```

3.11 Finalization and Postprocessing

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

        if (dofHand->myRank() == 0)
        cout<<std::scientific<<std::setprecision(5)<<"Border Length = "<<endl;
        cout<<hiperProbl->globalIntegral("BorderLength")<<endl;

        ofstream rhsFile;
```

```
        rhsFile.open(paramStr->getStringParameter(PoissonParams::filemesh)+
        to_string(dofHand->myRank())+".txt");
        hiperProbl->sol->Print(rhsFile);
        rhsFile.close();

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

3.12 Filling left-hand side matrix

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

        double strength = fillStr.getRealParameter(PoissonParams::strength);

        SubFillStructure& subFill = fillStr["dofHand"];
        int DOF  = subFill.numDOFs;
        int eNN  = subFill.eNN;
        int nDim = subFill.nDim;
        int pDim = subFill.pDim;
        vector<double>& nborCoords  = subFill.nborCoords;

        double* bf  = subFill.nborBFs();
        double* dbf_l = subFill.nborBFsGrads();
        double x[3]={};
        for (int i = 0; i < eNN; i++) // i from 0 to 5
        {
                for (int n = 0; n < nDim; n++) // n from 0 to 2
                {
                        x[n] += nborCoords[i*nDim+n] * bf[i];
                }
        }

        double g_cc[4]={};
        double xu[3]={};
        double xv[3]={};
        SurfLagrParam::MetricTensor(g_cc, xu, xv, eNN, nborCoords.data(), dbf_l);

        double g_CC[4]={};
        Math::Invert2x2(g_CC, g_cc);

        double jac = sqrt(Math::DetMat2x2(g_cc));

        double source = strength * x[1]*cos(3.0*M_PI*x[0]);

        for (int i = 0; i < eNN; i++)
        {
                double* dbf_lI_c = &dbf_l[pDim*i];

                double dbf_lI_C[2]={};
                Math::MatProduct(dbf_lI_C, 2, 2, 1, g_CC, dbf_lI_c);

                for (int j = 0; j < eNN; j++)
                {
                        double* dbf_lJ_c = &dbf_l[pDim*j];

                        fillStr.Ak(0,0)[i*DOF*eNN+j*DOF] += jac*Math::Dot2D(dbf_lI_C,dbf_lJ_c);
                }

                fillStr.Bk(0)[i*DOF] += jac * bf[i] * source;
        }

```

```
        return;
}
```

3.13 Filling right-hand side vector

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

        SubFillStructure& subFill = fillStr["dofHand"];
        int DOF  = subFill.numDOFs;
        int eNN  = subFill.eNN;
        int nDim = subFill.nDim;
        vector<double>& nborCoords = subFill.nborCoords;
        double *bf   = subFill.nborBFs();
        double *dbf_l = subFill.nborBFsGrads();

        double Ip[4], xu[3], xv[3];
        SurfLagrParam::MetricTensor(Ip, xu, xv, eNN, nborCoords.data(), dbf_l);

        auto bTangentRef = subFill.tangentsBoundaryRef();
        double bTangent[3]={};
        for (int n = 0; n < nDim; n++)
        bTangent[n] = bTangentRef[0] * xu[n] + bTangentRef[1]*xv[n];
        double jac = Math::Norm3D(bTangent);

        for (int i = 0; i < eNN; i++)
        fillStr.Bk(0)[i*DOF] += jac*bf[i];

        fillStr.addToGlobalIntegral("BorderLength", jac);
}
```

4 Results

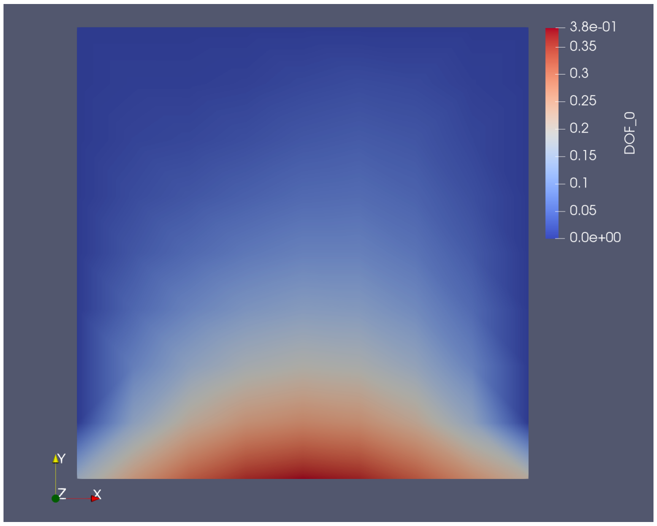


Figure 4: Solution.

Appendix