

# HiperLife Tutorial: PoissonSurfaceNeumann

LaCàN

August 12, 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  $x_2 = 1$ , as depicted in Figure 1.

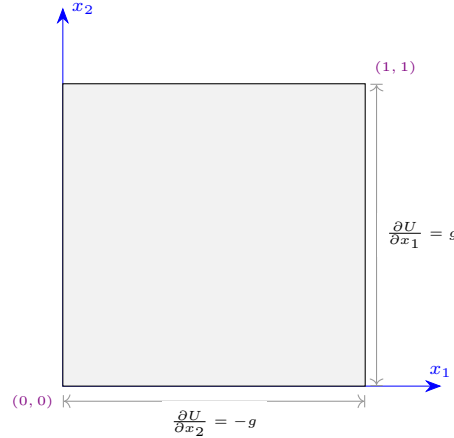


Figure 1: Illustration of Geometry in  $\Omega = [0, 1] \times [0, 1]$  and Boundary conditions on  $\partial\Omega$ .

## 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

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

By integrating this expression over  $\Omega$ , we have

$$-\int_{\Omega} w \nabla^2 U = \int_{\Omega} w f \quad (3)$$

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

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

14 according to Gauss's theorem

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

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

$$-\int_{\Omega} w \nabla^2 U = \int_{\Omega} \nabla w \cdot \nabla U - \int_{\partial \Omega} w \nabla U \cdot n dS \quad (6)$$

16 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

$$\int_{\Omega} \nabla w \cdot \nabla U dA = \int_{\Omega} w f dA + \int_{\partial \Omega} w g dS \quad (7)$$

## 17 2.2 Basis Function

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

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

19 by applying Galerkin method the weight function is the same as basis function.

$$w_i = \phi_i \quad (9)$$

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

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

## 21 2.3 Element

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

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

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

25 For the purpose of discretization of the border in this example, 3-node line element is used, as shown in Fig.  
26 2b. which its interpolation functions would be like this.

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

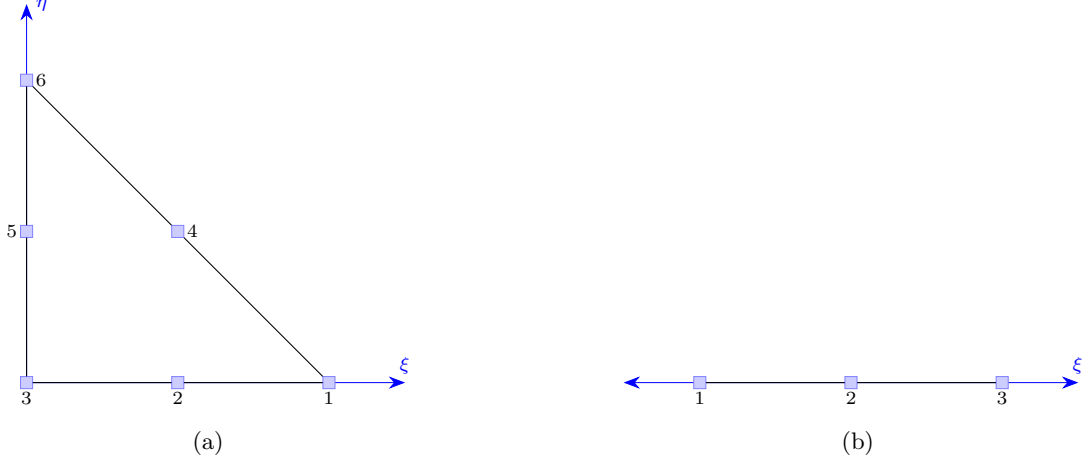


Figure 2: Schematic of the (a) bulk element (b) border element

## 2.4 Elemental Integral

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

$$\int_{\Omega_T} \left( \frac{\partial N}{\partial x_1} \frac{\partial N}{\partial x_1} + \frac{\partial N}{\partial x_2} \frac{\partial N}{\partial x_2} \right) U dA = \int_{\Omega_T} N f dA + \int_{\partial\Omega} N g dS \quad (13)$$

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

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

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

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

by defining Jacobian as

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

so we can rewrite the Eq. (13)

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

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

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

and for right-hand side

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

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  $dA = dx_1 \times dx_2 = j d\xi d\eta$ .

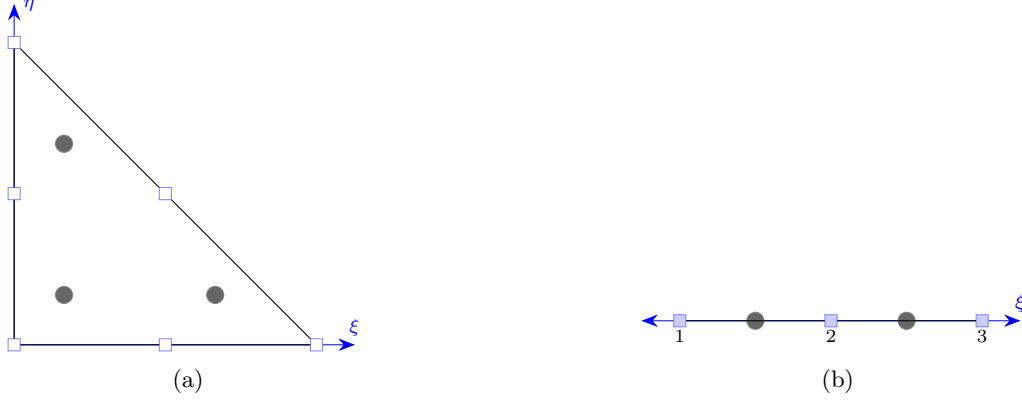


Figure 3: (a) 2-D integration on Triangle element (b) 1-D integration on line element

## 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

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

where  $n$  and  $m$  are the number of integration points used in each direction,  $\omega$  is quadrature weights,  $\hat{\xi}_i$  and  $\hat{\eta}_i$  are the roots of the  $n$ th 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
    };
};
```

```

68
69     enum StringParameters
70     {
71         filemesh
72     };
73
74     HLPARAMETERLIST DefaultValues{
75         {"filemesh", ""},
76         {"strength", 1.0},
77     };
78 };

```

### 3.3 Initializing

```

80 void LS(hiperlife::FillStructure& fillStr);
81 void RHS.Border(hiperlife::FillStructure& fillStr);
82
83 int main(int argc, char** argv)
84 {
85     using namespace std;
86     using namespace hiperlife;

```

### 3.4 Defining parameters of the model

```

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

```

### 3.5 Mesh Generation

```

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

```

### 3.6 Mesh Distribution

```

96 SmartPtr<DistributedMesh> disMesh = Create<DistributedMesh>();
97 disMesh->setMesh(mesh);
98 disMesh->setBalanceMesh(true);
99 disMesh->Update();

```

### 3.7 DOFs Handler

```

101 SmartPtr<DOFsHandler> dofHand = Create<DOFsHandler>(disMesh);
102 dofHand->setNameTag("dofHand");
103 dofHand->setNumDOFs(1);
104 dofHand->setNumElemAuxF(1);
105 dofHand->Update();

```

### 3.8 Boundary conditions and Constraints

```

107 if (!disMesh->hasBorder())
108 {
109     if (disMesh->myRank() == 0)
110         dofHand->setConstraint(0, 0, IndexType::Local, 0.0);
111 }
112 else
113 {

```

```

114         for (int i = 0; i < disMesh->loc.nPts(); i++)
115         {
116             if (disMesh->nodeCrease(i, IndexType::Local) > 0 and
117                 disMesh->nodeCoord(i, 1, IndexType::Local) > 0)
118                 dofHand->setConstraint(0,i,IndexType::Local, 0.0);
119         }
120     }
121     dofHand->UpdateGhosts();

```

### 122 3.9 HiperProblem

```

123     SmartPtr<HiPerProblem> hiperProbl = Create<HiPerProblem>();
124
125     hiperProbl->setParameterStructure(paramStr);
126     hiperProbl->setDOFsHandlers({dofHand});
127
128     hiperProbl->setIntegration("Integ", {dofHand});
129     hiperProbl->setCubatureGauss("Integ", 3);
130     hiperProbl->setElementFillings("Integ", LS);
131
132     hiperProbl->setIntegration("BorderInteg", {dofHand});
133     hiperProbl->setCubatureBorderGauss("BorderInteg", 2, {MAxis::Xmax, MAxis::Ymin});
134     hiperProbl->setElementFillings("BorderInteg", nullptr, nullptr, RHS_Border);
135
136     hiperProbl->setGlobalIntegrals({"BorderLength"});
137
138     hiperProbl->Update();

```

### 139 3.10 Solver Settings

```

140     SmartPtr<AztecOOIterativeLinearSolver> solver=Create<AztecOOIterativeLinearSolver>();
141     solver->setHiPerProblem(hiperProbl);
142     solver->setTolerance(1.E-8);
143     solver->setMaxNumIterations(500);
144     solver->setSolver(AztecOOIterativeLinearSolver::Solver::Gmres);
145     solver->setPreconditioner(AztecOOIterativeLinearSolver::Preconditioner::Neumann);
146     solver->setDefaultParameters();
147     solver->setVerbosity(AztecOOIterativeLinearSolver::Verbosity::High);
148     solver->Update();
149
150     solver->solve();
151     solver->UpdateSolution();

```

### 152 3.11 Finalization and Postprocessing

```

153     dofHand->printFileLegacyVtk("PoissonSurface");
154
155     if (dofHand->myRank() == 0)
156         cout<<std::scientific<<std::setprecision(5)<<"Border - Length =->"<<endl;
157         cout<<hiperProbl->globalIntegral("BorderLength")<<endl;
158
159     ofstream rhsFile;
160     rhsFile.open(paramStr->getStringParameter(PoissonParams::filemesh)+
161         to_string(dofHand->myRank())+".txt");
162     hiperProbl->sol->Print(rhsFile);
163     rhsFile.close();
164
165     hiperlife::Finalize();
166     return 0;
167 }

```

### 168 3.12 Filling left-hand side matrix

```

169 void LS(hiperlife::FillStructure& fillStr)
170 {
171     using namespace std;
172     using namespace hiperlife;
173     using hiperlife::Tensor::tensor;
174
175     double strength = fillStr.getRealParameter(PoissonParams::strength);
176
177     SubFillStructure& subFill = fillStr["dofHand"];
178     int DOF = subFill.numDOFs;
179     int eNN = subFill.eNN;
180     int nDim = subFill.nDim;
181     int pDim = subFill.pDim;
182     vector<double>& nborCoords = subFill.nborCoords;
183
184     double* bf = subFill.nborBFs();
185     double* dbf_l = subFill.nborBFsGrads();
186     double x[3]={};
187     for (int i = 0; i < eNN; i++) // i from 0 to 5
188     {
189         for (int n = 0; n < nDim; n++) // n from 0 to 2
190         {
191             x[n] += nborCoords[i*nDim+n] * bf[i];
192         }
193     }
194
195     double g_cc[4]={};
196     double xu[3]={};
197     double xv[3]={};
198     SurfLagrParam::MetricTensor(g_cc, xu, xv, eNN, nborCoords.data(), dbf_l);
199
200     double g_CC[4]={};
201     Math::Invert2x2(g_CC, g_cc);
202
203     double jac = sqrt(Math::DetMat2x2(g_cc));
204
205     double source = strength * x[1]*cos(3.0*M_PI*x[0]);
206
207     for (int i = 0; i < eNN; i++)
208     {
209         double* dbf_lI_c = &dbf_l[pDim*i];
210
211         double dbf_lI_C[2]={};
212         Math::MatProduct(dbf_lI_C, 2, 2, 1, g_CC, dbf_lI_c);
213
214         for (int j = 0; j < eNN; j++)
215         {
216             double* dbf_lJ_c = &dbf_l[pDim*j];
217
218             fillStr.Ak(0,0)[i*DOF*eNN+j*DOF] += jac*Math::Dot2D(dbf_lI_C, dbf_lJ_c);
219         }
220
221         fillStr.Bk(0)[i*DOF] += jac * bf[i] * source;
222     }
223
224     return;
225 }

```

### 226 3.13 Filling right-hand side vector

```

227 void RHS.Border(hiperlife::FillStructure& fillStr)
228 {
229     using namespace std;
230     using namespace hiperlife;

```

```

231     using hiperlife::Tensor::tensor;
232
233     SubFillStructure& subFill = fillStr["dofHand"];
234     int DOF = subFill.numDOFs;
235     int eNN = subFill.eNN;
236     int nDim = subFill.nDim;
237     vector<double>& nborCoords = subFill.nborCoords;
238     double *bf = subFill.nborBFs();
239     double *dbf_l = subFill.nborBFsGrads();
240
241     double Ip[4], xu[3], xv[3];
242     SurfLagrParam::MetricTensor(Ip, xu, xv, eNN, nborCoords.data(), dbf_l);
243
244     auto bTangentRef = subFill.tangentsBoundaryRef();
245     double bTangent[3]={};
246     for (int n = 0; n < nDim; n++)
247         bTangent[n] = bTangentRef[0] * xu[n] + bTangentRef[1]*xv[n];
248     double jac = Math::Norm3D(bTangent);
249
250     for (int i = 0; i < eNN; i++)
251         fillStr.Bk(0)[i*DOF] += jac*bf[i];
252
253     fillStr.addToGlobalIntegral("BorderLength", jac);
254 }

```

## 255 4 Results