HiperLife Tutorial: Nonlinear Poisson

Arash Imani

LaCàN

November 3, 2024

1 Problem Definition

The nonlinear Poisson equations are significantly more challenging to solve than its linear counterpart because the relationship between the potential and source term is complex. Understanding and solving the nonlinear Poisson equation is essential for designing and optimizing devices, interpreting material behaviors, and simulating natural processes in many areas of physics and engineering.

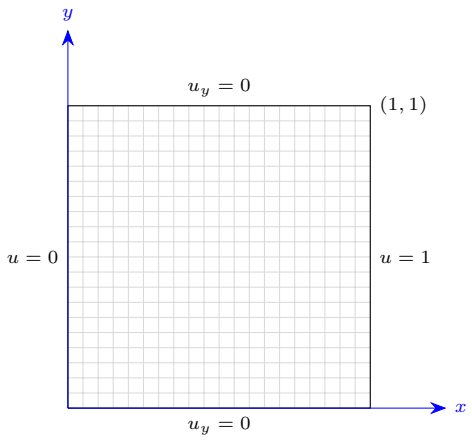

Figure 1: Geometry, BC and computational domain used for the analysis of nonlinear Poisson.

The exact solution for this particular problem is given by

$$u(x, y) = (3x + 1)^{1/2} - 1\,. \tag{1}$$

Here we want to use finite element method to solve it. Because in general unlike linear equations, solutions cannot be directly superimposed, and standard analytical methods like separation of variables are usually not applicable for nonlinear type. Instead, specialized techniques such as variational approaches, and numerical methods are employed to tackle these problems.

2 Governing Equations

In this section we present the governing equations, consider this nonlinear PDE:[1]

$$-\nabla \cdot [(1 + u)\nabla u] = f \quad \text{in } \Omega \tag{2}$$

As it is demonstrated in Figure 1, The domain Ω is a square of dimensions $[0, 1] \times [0, 1]$ along the x and y coordinates, where $f = 0$ and the boundary conditions to be

$$\begin{aligned} u(0, y) = 0,\ u(1, y) = 1 \quad &\text{on } \Gamma_D \\ u_y(x, 0) = u_y(x, 1) = 0 \quad &\text{on } \Gamma_N \end{aligned} \tag{3}$$

3 Weak Form

The starting point for the development of the finite element models of Eq. (1) is their weak forms. The variational formulation of our model problem reads: Find $u \in V$ such that

$$\mathcal{F}(u;v) = 0 \quad \forall v \in \hat{V} \tag{4}$$

where

$$\mathcal{F}(u;v) = -\int_\Omega v\nabla\cdot[(1+u)\nabla u] - vf\mathrm{d}\Omega\,. \tag{5}$$

and

$$\begin{aligned} \hat{V} &= \{v \in H^1(\Omega) : v = 0 \text{ on } x = 0 \text{ and } x = 1\}, \\ V &= \{v \in H^1(\Omega) : v = 0 \text{ on } x = 0 \text{ and } v = 1 \text{ on } x = 1\}\,. \end{aligned} \tag{6}$$

where v is a test function, which will be equated, in the our FE model to the interpolation function used for u.The discrete problem arises as usual by restricting V and $\hat{V}$ to a pair of discrete spaces. Since $\mathcal{F}$ is a nonlinear function of u, the variational statement gives rise to a system of nonlinear algebraic equations. using integrating by part this expression over Ω, we have

$$\mathcal{F}(u;v) = -\int_\Omega \nabla\cdot[v(1+u)\nabla u] + \int_\Omega (1+u)\nabla v\nabla u - \int_\Omega vf\mathrm{d}\Omega\,. \tag{7}$$

Using Gauss's theorem we get

$$\mathcal{F}(u;v) = -\int_\Gamma [v(1+u^2)\nabla u]\cdot n\ \mathrm{d}\Gamma + \int_\Omega (1+u)\nabla u\nabla v\mathrm{d}\Omega - \int_\Omega vf\mathrm{d}\Omega\,. \tag{8}$$

Applying boundary conditions ($\Gamma = \Gamma_N \cup \Gamma_D$):

$$\begin{aligned} v &= 0 \quad \text{on } \Gamma_D\,, \\ \nabla u\cdot n &= 0 \quad \text{on } \Gamma_N\,. \end{aligned} \tag{9}$$

and by assuming $f = 0$ we get the final form for $\mathcal{F}$

$$\mathcal{F}(u;v) = \int_\Omega (1+u)\nabla u\cdot\nabla v\,\mathrm{d}\Omega, \tag{10}$$

After having discretized our nonlinear PDE problem, we now need to linearize it, we may use Newton's method to solve the system of nonlinear algebraic equations. Newton's method for the system $\mathcal{F}_i(U_1,\ldots,U_j)$ can be formulated by the first terms of a Taylor series approximation for the value of the variational as

$$\begin{aligned} \sum_{j=1}^{N} \frac{\partial}{\partial U_j}\mathcal{F}_i(U_1^k,\ldots,U_N^k)\delta U_j = -\mathcal{F}_i(U_1^k,\ldots,U_N^k), \quad i = 1,\ldots,N, \\ U_j^{k+1} = U_j^k + \delta U_j, \quad j = 1,\ldots,N, \end{aligned} \tag{11}$$

where k is an iteration index. An initial guess u^0 must be provided to start the algorithm. We need to compute the $\partial\mathcal{F}_i/\partial U_j$ and the right-hand side vector $-\mathcal{F}_i$. Our present problem has $\mathcal{F}_i$ given by above. so, Hessian is given by

$$\mathcal{J}(u;\phi_j,\phi_i) = \frac{\partial F_i}{\partial U_j} = \int_\Omega \left[\frac{\partial u}{\partial U_j}\nabla u^k\cdot\nabla v + (1+u^k)\nabla[\frac{\partial u}{\partial U_j}]\cdot\nabla v\right]\mathrm{d}\Omega\,. \tag{12}$$

4 Finite Element Model

For Ritz-Galerkin FE model, the choice of the test functions is restricted to the spaces of approximation functions used for the solution field. Suppose that the variable u approximated by expansions of the form

$$u(\mathbf{x}) = \sum_{j=1}^{M} \phi_j(\mathrm{x})u_j = \mathbf{\Phi}^T\mathbf{u}\,. \tag{13}$$

By considering Eq. (13), the Hessian ($\mathcal{J} = \partial \mathcal{F}_i / \partial U_j$) can be introduced in this way

$$\mathcal{J}(u; \phi_j, \phi_i) = \frac{\partial F_i}{\partial U_j} = \int_\Omega \left[\phi_j \nabla u^k \cdot \nabla \phi_i + (1 + u^k) \nabla \phi_j \cdot \nabla \phi_i \right] \, \mathrm{d}\Omega \,. \tag{14}$$

The above equations can be written symbolically in matrix form as

$$\mathbf{K}\theta = \mathbf{F} \,. \tag{15}$$

In the Hiperlife legacy codes the coefficient matrices $\mathbf{K}$ and $\mathbf{F}$ shown in Eq. (15) are defined as Hessain and Jacobian, respectively. In this problem they are given by

$$\begin{aligned} \mathbf{K}_{ij} &= \int_{\Omega^e} \left[\phi_j \nabla u^k \cdot \nabla \phi_i + (1 + u^k) \nabla \phi_j \cdot \nabla \phi_i \right] \mathrm{d}A \,, \\ \mathbf{F}_i &= - \int_{\Omega^e} [(1 + u^k) \nabla u^k \cdot \nabla \phi_i] \mathrm{d}A \,. \end{aligned} \tag{16}$$

The elemental representation of the vector and matrix required for implementation in the Hiperlife would be like[1]

$$\begin{aligned} Bk(i) &= -\mathcal{F}_i = -jac \times [(1 + u^k) \nabla u^k \cdot \nabla \phi_i], \\ Ak(i,j) &= \mathcal{J}_{ij} = jac \times [\phi_j \nabla u^k \cdot \nabla \phi_i + (1 + u^k) \nabla \phi_j \cdot \nabla \phi_i] \,. \end{aligned} \tag{17}$$

Note that $dA = dx_1 \times dx_2 = jac \; d\xi d\eta$, which $jac = \det(Jacobian)$.

5 Choice of Elements

Thus, for this simple problem every Lagrange and serendipity family of interpolation functions are admissible for the interpolation of the temperature field, our choice is would be linear quadrilateral element.Linear Quadrilateral Elements is the simplest quadrilateral element consists of four nodes. The associated interpolation functions for geometry and field variables are bilinear.

$$\mathbf{\Phi}_I(\xi, \eta) = \frac{1}{4}(1 + \xi_I \xi)(1 + \eta_I \eta) \quad (I \text{ from 1 to 4}) \tag{18}$$

where ξ_I and η_I are the corner coordinates at element T in domain of $\Omega_T \in (-1, 1)^2$. As it shown in Figure 2 we are using 2×2 Gauss–Legendre quadrature integration. We also chose a uniform mesh of size 10×10 to

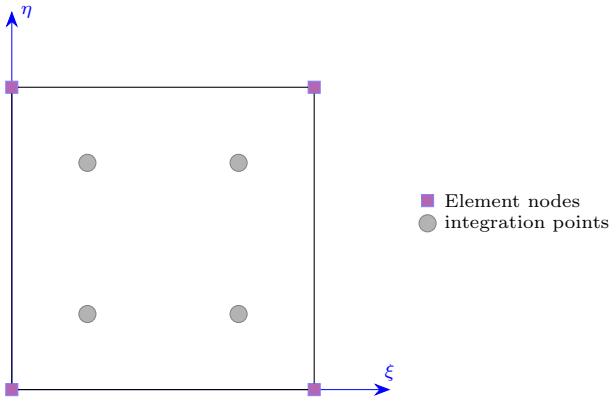

Figure 2: Linear quadrilateral element used for finite element model.

model the domain of our problem.

[1] Note that HiperLife by default applies the $-$ in $Bk(i)$ to Nonlinear problems, so it is not required to add it to your code!

6 Implementation

In this section, we present the implementation of our solution in the Hiperlife. The program is divided into three separate files, main part which we create our problem by the Hiperlife headers, auxiliary header where we introduce parameters and declare defined functions, and at last auxiliary file, where we define some functions which provide required matrices like the Jacobian and the Hessian.

6.1 PoissonNonL.cpp

```cpp
/*
* nonlinear Poisson equation (using nonlinear solver)
*/

// C++ headers
#include <iostream>
#include <fstream>
#include <time.h>

// hiperlife headers
#include "hl_Core.h"
#include "hl_Parser.h"
#include "hl_TypeDefs.h"
#include "hl_DOFsHandler.h"
#include "hl_HiPerProblem.h"
#include "hl_SurfLagrParam.h"
#include "hl_FillStructure.h"
#include "hl_ParamStructure.h"
#include "hl_DistributedMesh.h"
#include "hl_StructMeshGenerator.h"
#include "hl_GlobalBasisFunctions.h"
#include "hl_NonlinearSolver_NewtonRaphson.h"
#include "hl_LinearSolver_Iterative_AztecOO.h"
#include <hl_ConsistencyCheck.h>


// problem header
#include "AuxPoissonNonL.h"

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

        //////////////////////////////////////////////////////////////////
        /// *****                  INITIALIZATION                  **** ///
        //////////////////////////////////////////////////////////////////

        // Initialize the MPI execution environment
        hiperlife::Init(argc, argv);

        // Define parameters of the model
        SmartPtr<ParamStructure> paramStr = CreateParamStructure<PoissonParams>();
        double f  = paramStr->getRealParameter(PoissonParams::f);

        //////////////////////////////////////////////////////////////////
        // *****                  MESH CREATION                   *****//
        //////////////////////////////////////////////////////////////////

        // Create structured mesh
        SmartPtr<StructMeshGenerator> mesh = Create<StructMeshGenerator>();
        mesh->setNDim(3);
        mesh->setElemType(ElemType::Square);
        mesh->setBasisFuncType(BasisFuncType::Lagrangian);
        mesh->setBasisFuncOrder(1);
        mesh->genSquare(10,1.0);

```

```
58          // Distributed mesh
59          SmartPtr<DistributedMesh> disMesh = Create<DistributedMesh>();
60          disMesh->setMesh(mesh);
61          disMesh->setBalanceMesh(true);
62          disMesh->setElementLocatorEngine(ElementLocatorEngine::BoundingVolumeHierarchy);
63                  disMesh->Update();
64
65          // checking mesh
66          disMesh->printFileLegacyVtk("mesh");
67
68          // ************************************************************//
69          // *****                  DOFsHANDLER CREATION                *****//
70          // ************************************************************//
71
72          // Create DOFsHandler
73          SmartPtr<DOFsHandler> dofHand = Create<DOFsHandler>(disMesh);
74          dofHand->setNameTag("dofHand");
75          dofHand->setNumDOFs(1);
76          dofHand->setDOFs({"u"});
77          dofHand->Update();
78
79          // ---------------- initial condition first guess ------------------- //
80          //-------------------------------------------------------------------//
81          for (int i = 0; i < disMesh->loc_nPts(); i++)
82          {
83                  // Coordinate
84                  std::vector<double> x = disMesh->nodeCoords(i, IndexType::Local);
85                  // Initial condition
86                  dofHand->nodeDOFs->setValue("u", i, IndexType::Local, x[0]);
87                  // ------------------------ Boundary condition ---------------- //
88                  //-------------------------------------------------------------//
89                  if (x[0] < 1e-5)
90                  {
91                          dofHand->nodeDOFs->setValue("u", i, IndexType::Local,0.0);
92                          dofHand->setConstraint("u",i, IndexType::Local,0.0);
93                  }
94                  if (x[0] > (1.-1e-5))
95                  {
96                          dofHand->nodeDOFs->setValue("u", i, IndexType::Local,1.0);
97                          dofHand->setConstraint("u",i, IndexType::Local,0.0);
98                  }
99          }
100
101         // Update
102         dofHand->UpdateGhosts();
103
104         // checking initial and boundary condition
105         dofHand->printFileLegacyVtk("PoissonNonL0");
106
107         // ********************************************************** //
108         /// *****                HIPERPROBLEM CREATION               **** ///
109         // ********************************************************** //
110
111         // Create hiperproblem
112         SmartPtr<HiPerProblem> hiperProbl = Create<HiPerProblem>();
113
114         // Set parameter structure and DOFsHandler
115         hiperProbl->setParameterStructure(paramStr);
116         hiperProbl->setDOFsHandlers({dofHand});
117
118         // Set integration in the bulk
119         hiperProbl->setIntegration("Integ", {"dofHand"});
120         hiperProbl->setCubatureGauss("Integ",4);
121         hiperProbl->setElementFillings("Integ", LS);
122
123         // Consistency Check
124         if (true)
125         {
```

```
126                     hiperProbl->setConsistencyDOFs("dofHand", {"u"});
127                     hiperProbl->setElementFillings("Integ", ConsistencyCheck<LS>);
128                     hiperProbl->setConsistencyCheckType(ConsistencyCheckType::Hessian);
129             }
130
131             // Update
132             hiperProbl->Update();
133
134
135             // ************************************************************ //
136             /// ****                  SOLVE HIPERPROBLEM                  **** ///
137             // ************************************************************ //
138
139             // Create linear solver
140             SmartPtr<AztecOOIterativeLinearSolver> linsolver = Create<AztecOOIterativeLinearSolver>();
141             linsolver->setHiPerProblem(hiperProbl);
142             linsolver->setTolerance(1.E-8);
143             linsolver->setMaxNumIterations(500);
144             linsolver->setSolver(AztecOOIterativeLinearSolver::Solver::Gmres);
145             linsolver->setPreconditioner(AztecOOIterativeLinearSolver::Preconditioner::None);
146             linsolver->setDefaultParameters();
147             linsolver->setVerbosity(AztecOOIterativeLinearSolver::Verbosity::None);
148             linsolver->Update();
149
150             // Create nonlinear solver
151             SmartPtr<NewtonRaphsonNonlinearSolver> nonLinSolver = Create<NewtonRaphsonNonlinearSolver>();
152             nonLinSolver->setLinearSolver(linsolver);
153             nonLinSolver->setConvRelTolerance(true);
154             nonLinSolver->setMaxNumIterations(15);
155             nonLinSolver->setResTolerance(1e-6);
156             nonLinSolver->setSolTolerance(1e-6);
157             nonLinSolver->setResMaximum(1e5);
158             nonLinSolver->setSolMaximum(1e5);
159             nonLinSolver->setExitRelMaximum(true);
160             nonLinSolver->setLineSearch(false);
161             nonLinSolver->setPrintSummary(false);
162             nonLinSolver->setPrintIntermInfo(true);
163             nonLinSolver->Update();
164
165             // Solve
166             bool converged = nonLinSolver->solve();
167
168             // Check convergence
169             if (converged)
170             {
171                     // Save solution
172                     dofHand->nodeDOFs0->setValue(dofHand->nodeDOFs);
173                     dofHand->nodeDOFs0->UpdateGhosts();
174
175                     // Print solution
176                     dofHand->printFileLegacyVtk("PoissonNonL");
177             }
178
179             // ************************************************************//
180             /// ****                    FINALIZE                      ****///
181             // ************************************************************//
182             hiperlife::Finalize();
183             return 0;
184     }
```

6.2 AuxPoissonNonL.h

```
1   #ifndef AUXPoisson_H
2   #define AUXPoisson_H
3
4   // C headers
```

```
5  #include <iostream>
6
7  // hiperlife headers
8  #include "hl_Core.h"
9  #include "hl_ParamStructure.h"
10 #include "hl_Parser.h"
11 #include "hl_TypeDefs.h"
12 #include "hl_MeshLoader.h"
13 #include "hl_StructMeshGenerator.h"
14 #include "hl_DistributedMesh.h"
15 #include "hl_FillStructure.h"
16 #include "hl_DOFsHandler.h"
17 #include "hl_HiPerProblem.h"
18 #include "hl_SurfLagrParam.h"
19 #include "hl_LinearSolver_Iterative_AztecOO.h"
20 #include "hl_GlobalBasisFunctions.h"
21 #include "hl_NonlinearSolver_NewtonRaphson.h"
22
23 struct PoissonParams
24 {
25         enum RealParameters
26         {
27                 f
28         };
29         enum StringParameters
30         {
31                 filemesh
32         };
33         HL_PARAMETER_LIST DefaultValues
34         {
35                 {"f", 0.0},
36                 {"filemesh", ""},
37         };
38 };
39
40
41 void LS(hiperlife::FillStructure& fillStr);
42
43 #endif
```

6.3 AuxPoissonNonL.cpp

```
1  // hiperlife headers
2  #include "hl_Core.h"
3  #include "hl_ParamStructure.h"
4  #include "hl_Parser.h"
5  #include "hl_TypeDefs.h"
6  #include "hl_MeshLoader.h"
7  #include "hl_StructMeshGenerator.h"
8  #include "hl_DistributedMesh.h"
9  #include "hl_FillStructure.h"
10 #include "hl_DOFsHandler.h"
11 #include "hl_HiPerProblem.h"
12 #include "hl_SurfLagrParam.h"
13 #include "hl_LinearSolver_Iterative_AztecOO.h"
14 #include "hl_GlobalBasisFunctions.h"
15 #include "hl_NonlinearSolver_NewtonRaphson.h"
16 // problem header
17 #include "AuxPoissonNonL.h"
18
19 void LS(hiperlife::FillStructure& fillStr)
20 {
21         using namespace std;
22         using namespace hiperlife;
23         using hiperlife::Tensor::tensor;
24
```

```
25          // Dimensions
26          SubFillStructure& subFill = fillStr["dofHand"];
27          int nDOFs = subFill.numDOFs;
28          int eNN   = subFill.eNN;
29          int nDim  = subFill.nDim;
30          int pDim  = subFill.pDim;
31
32          // Nodal values at Gauss points
33          vector<double>& nborCoords = subFill.nborCoords; // Vector of node coordinates
34          ttl::wrapper<double,1> nborDOFs(subFill.nborDOFs.data(),eNN);
35
36          // Shape functions and derivatives at Gauss points
37          double jac;
38          ttl::wrapper<double,1>  bf(subFill.nborBFs(), eNN);
39          tensor<double,2> Dbf(eNN,pDim);
40          GlobalBasisFunctions::gradients(Dbf, jac, subFill);
41
42          // source
43          double f = fillStr.getRealParameter(PoissonParams::f);
44
45          //-------------------------------------------------------------------//
46          //-------------------- OUTPUT DATA ----------------------------------//
47          //-------------------------------------------------------------------//
48          ttl::wrapper<double,2> Ak(fillStr.Ak(0, 0).data(), eNN, eNN);
49          ttl::wrapper<double,1> Bk(fillStr.Bk(0).data(),eNN);
50
51          //=====================  previous step values ======================//
52          // u
53          double u = bf*nborDOFs;
54          // grad u
55          tensor<double,1> gradu(pDim);
56          for (int i = 0; i < eNN; i++)
57          {
58                  for (int d = 0; d < pDim; d++)
59                  gradu(d) += Dbf(i,d)*nborDOFs(i);
60          }
61          // (gradient of the bf) *  (gradient of the bf)
62          tensor<double,2> DbfDbf = product(Dbf,Dbf,{{1,1}});
63          // (gradient of the bf) *  (gradient of u)
64          tensor<double,1> DuDbf = product(gradu,Dbf,{{0,1}});
65          //=====================  Fill nonlinear system ====================//
66          for (int i = 0; i < eNN; i++)
67          {
68                  // Fill jacobian
69                  Bk(i) +=  jac * (1.+u) * DuDbf(i);
70
71                  for (int j = 0; j < eNN; j++)
72                  {
73                          // Fill Hessian
74                          Ak(i,j) += jac * (bf(j)*DuDbf(i) + (1.+u)*DbfDbf(i,j));
75                  }
76          }
77
78          return;
79  }
```

7 Results

In this section, we present the results of our solution. Table 1 shows the comparison between numerical solution and exact value calculated by Eq. (3). The contour demonstration of result u is also shown in Figure 3.

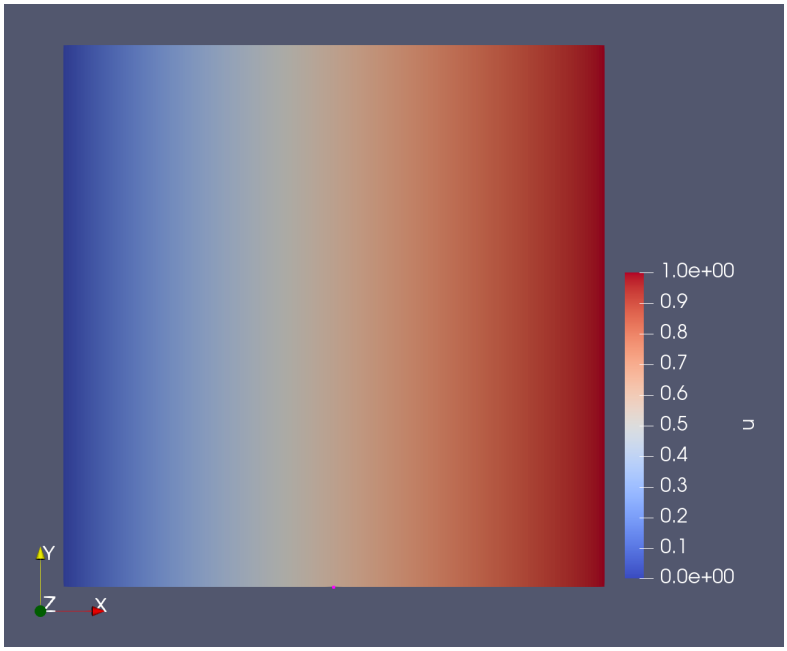

Figure 3: Illustration of the solution of Nonlinear Poisson problem.

Table 1: Numerical and exact solution for different x.

x	$\mathbf{u}_{numerical}$	$\mathbf{u}_{exact}$
0	0.0	0.0
0.2	0.26491	0.264911064
0.4	0.48324	0.483239697
0.6	0.67332	0.673320053
0.8	0.84391	0.843908891
1.0	1.0	1.0

References

[1] Igor A. Baratta, Joseph P. Dean, Jørgen S. Dokken, Michal Habera, Jack S. Hale, Chris N. Richardson, Marie E. Rognes, Matthew W. Scroggs, Nathan Sime, and Garth N. Wells. DOLFINx: the next generation FEniCS problem solving environment. preprint, 2023.