HiperLife Tutorial: Thermal Conduction

Arash Imani

LaCàN

November 2, 2024

1 Problem Definition

Thermal conduction is the diffusion of thermal energy within one material or between materials in contact. The higher temperature object has molecules with more kinetic energy; collisions between molecules distributions this kinetic energy until an object has the same thermal energy throughout. Conduction is the main mode of heat transfer between or inside solid materials. Here we consider an example of time-dependent linear problem. Figure 1 shows a schematic representation of the problem statement.

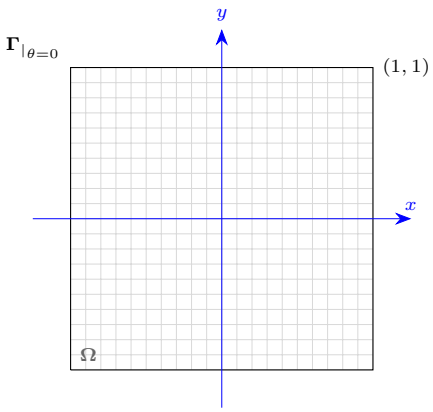

Figure 1: Geometry, BC and computational domain used for the analysis of transient heat transfer.

2 Governing Equations

In this section we present the governing equations, consider the transient heat conduction equation [1]

$$\frac{\partial \theta}{\partial t} - (\frac{\partial^2 \theta}{\partial x^2} + \frac{\partial^2 \theta}{\partial y^2}) = 1 \tag{1}$$

where θ is the non-dimensional temperature in the doamin of $\Omega = [-1, -1] \times [1, 1]$, which its boundary condition is $\theta = 0$ on Γ for $t > 0$. The initial condition is that $\theta(x, y, 0) = 0$. We wish to find the temperature field inside the domain for $t > 0$. The paramteres of problem are listed as

$$\begin{aligned} &\text{space discretiziation step} = \Delta x = \Delta y = 0.1, \\ &\text{time discretiziation step} = \Delta T = 0.05, \\ &\text{initial distribution} = f(x, y) = 0.0 \end{aligned} \tag{2}$$

Note that the most commonly used method for solving parabolic equation like ($[C_e]\{\dot{u}\} + [K_e]\{u_e\} = \{F_e\}$), is the α-family of approximation, in which a weighted average of the time derivative of a dependent variable is approximated at two consecutive time steps by linear interpolation of the values of the variable at the two steps:

$$(1-\alpha)\{\dot{u}\}^n + \alpha\{\dot{u}\}^{n+1} \approx \frac{\{u\}^{n+1} - \{u\}^n}{t^{n+1} - t^n} \quad \text{for} \quad 0 \leq \alpha \leq 1 \tag{3}$$

3 Weak Form

The starting point for the development of the finite element models of Eq. (1) is their weak forms. By considering $\Delta t = t^{n+1} - t^n$, and substituting u with our temperature field, Eq. (3) takes the following form

$$\{\theta\}^{n+1} = \{\theta\}^n + \Delta t(1-\alpha)\{\theta_t\}^n + \Delta t\alpha\{\theta_t\}^{n+1} \tag{4}$$

The time derivatives of θ on the right hand side of this equation can be calculated from Eq. (1) for each step, like this

$$\begin{aligned} \{\theta_t\}^{n+1} &= 1 + \nabla^2\{\theta\}^{n+1}\,, \\ \{\theta_t\}^{n} &= 1 + \nabla^2\{\theta\}^{n}\,. \end{aligned} \tag{5}$$

combining these two equations gives us

$$\{\theta\}^{n+1} - \Delta t\alpha[1 + \nabla^2\{\theta\}^{n+1}] - \{\theta\}^n - \Delta t(1-\alpha)[1 + \nabla^2\{\theta\}^n] = 0\,. \tag{6}$$

The variation formulation of our model problem can be introduced as find $\theta \in V$ such that

$$\mathcal{F}(\theta; v) = 0 \quad \forall v \in \hat{V}\,. \tag{7}$$

where

$$\mathcal{F}(\theta; v) = \int_\Omega v\{\theta\}^{n+1} - \Delta t\alpha v[1 + \nabla^2\{\theta\}^{n+1}] - v\{\theta\}^n - \Delta t(1-\alpha)v[1 + \nabla^2\{\theta\}^n]\ \mathrm{d}\Omega\,. \tag{8}$$

and

$$\begin{aligned} \hat{V} &= \{v \in H^1(\Omega) : v = 0 \text{ on } \Gamma\}, \\ V &= \{v \in H^1(\Omega) : v = 0 \text{ on } \Gamma\}\,. \end{aligned} \tag{9}$$

where v is a test function, which will be equated, in the our FE model to the interpolation function used for θ.

Applying integration by part, and also considering boundary condition, the weak form takes the following form

$$\mathcal{F}(\theta; v) = \int_\Omega v\{\theta\}^{n+1} - \Delta t\alpha[v - \nabla v(\nabla\{\theta\}^{n+1})]\mathrm{d}\Omega - \int_\Omega v\theta^n + \Delta t(1-\alpha)[v - \nabla v\nabla\theta^n]\mathrm{d}\Omega\,. \tag{10}$$

4 Finite Element Model

Since we are developing the Ritz-Galerkin finite element model, the choice of the weight functions is restricted to the spaces of approximation functions used for the solution field. Suppose that the dependent variable θ approximated by expansions of the form

$$\theta(\mathbf{x}, t) = \sum_{m=1}^{M} \phi_m(\mathrm{x})\theta^m(t) = \mathbf{\Phi}^T\boldsymbol{\theta}\,, \tag{11}$$

Lets rewrite this equation for known and unkown variables.

$$\mathcal{F}(\theta; v) = \int [\mathbf{\Phi}\mathbf{\Phi}^T + \Delta t\alpha\nabla\mathbf{\Phi}\nabla\mathbf{\Phi}^T]\{\theta\}^{n+1}\mathrm{d}\Omega - \int [\mathbf{\Phi}\theta^n + \Delta t\alpha\mathbf{\Phi} + \Delta t(1-\alpha)(\mathbf{\Phi} - \nabla\mathbf{\Phi}\nabla\theta^n)]\mathrm{d}\Omega\,. \tag{12}$$

The above equations can be written symbolically in matrix form as

$$\mathbf{K}\theta = \mathbf{F}\,. \tag{13}$$

The coefficient matrices shown in Eq. (9) are defined by

$$\mathbf{K}_{ij} = \int_{\Omega^e} [\phi_i \phi_j + \Delta t \alpha \nabla \phi_i \nabla \phi_j] \mathrm{d}A\,, \quad \mathbf{F}_i = \int_{\Omega^e} [\phi_i \theta^n + \Delta t \alpha \phi_i + \Delta t (1-\alpha)(\phi_i - \nabla \phi_i \nabla \theta^n)] \mathrm{d}A\,. \tag{14}$$

We also can define the matrices as the way they are implemented in the Hiperlife:

$$\begin{aligned} Bk(i) &= jac \times [\phi_i \theta^n + \Delta t \phi_i - \Delta t (1-\alpha)(\nabla \phi_i \cdot \nabla \theta^n)] \\ Ak(i,j) &= jac \times [\phi_i \phi_j + \Delta t \alpha (\frac{\partial \phi_i}{\partial x}\frac{\partial \phi_j}{\partial x} + \frac{\partial \phi_j}{\partial y}\frac{\partial \phi_j}{\partial y})] \end{aligned} \tag{15}$$

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.

$$\mathbf{\Phi}_I(\xi, \eta) = \frac{1}{4}(1 + \xi_I \xi)(1 + \eta_I \eta) \quad (I \text{ from 1 to 4}) \tag{16}$$

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.

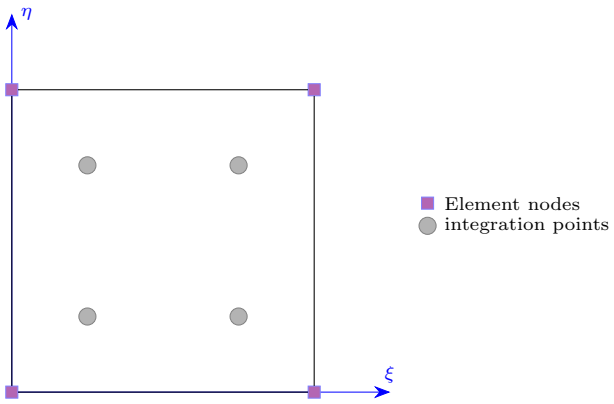

Figure 2: Linear quadrilateral element used for finite element model.

We also choose a uniform mesh of 16×16 to model the domain, and the in the result section we would investigate the stability and accuracy of forward method ($\alpha = 0.0$), Crank-Nicolson ($\alpha = 0.5$) and backward difference ($\alpha = 1.0$) for the solution.

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 Jacobian and Hessian.

6.1 HeatTransfer.cpp

```cpp
/* Heat Transfer conduction*/
// cpp headers
#include <iostream>
#include <fstream>
#include <cmath>

// hiperlife headers
#include "hl_Core.h"
#include "hl_Parser.h"
#include "hl_TypeDefs.h"
#include "hl_DOFsHandler.h"
#include "hl_HiPerProblem.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"

// Header to auxiliary functions
#include "AuxHeatTransfer.h"

// ------------------------------------------------------------------------//
// ---------------------     MAIN FUNCTION     ----------------------------//
// ------------------------------------------------------------------------//

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

        // ------------------------------------------------------------------//
        // *****                  INITIALIZATION                    *****  //
        // ------------------------------------------------------------------//

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

        // ------------------------------------------------------------------//
        /// *****                    DATA INPUT                      *****///
        // ------------------------------------------------------------------//

        // Put parameters in the user structure
        SmartPtr<ParamStructure> paramStr = CreateParamStructure<HeatParams>();

        // Data
        paramStr->setRealParameter(HeatParams::delta_t, 0.05);
        paramStr->setRealParameter(HeatParams::alpha, 0.5);
        double delta_t = paramStr->getRealParameter(HeatParams::delta_t);
        double alpha = paramStr->getRealParameter(HeatParams::alpha);

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

        // Create a line structured mesh
        SmartPtr<StructMeshGenerator> structMesh = Create<StructMeshGenerator>();
        structMesh->setNDim(3);
        structMesh->setBasisFuncType(BasisFuncType::Lagrangian);
        structMesh->setBasisFuncOrder(1);
        structMesh->setElemType(ElemType::Square);
        structMesh->genRectangle(16, 16, 2.0, 2.0);
        structMesh->translateX(-1.0);
        structMesh->translateY(-1.0);
```

```
67
68          // Distributed mesh
69          SmartPtr<DistributedMesh> disMesh = Create<DistributedMesh>();
70          disMesh->setMesh(structMesh);
71          disMesh->setBalanceMesh(true);
72          disMesh->setElementLocatorEngine(ElementLocatorEngine::BoundingVolumeHierarchy);
73          disMesh->Update();
74          disMesh->printFileLegacyVtk("mesh");
75
76          //------------------------------------------------------------------//
77          /// *****                DOFsHANDLER CREATION                *****///
78          //------------------------------------------------------------------//
79
80          // DOFHandler
81          SmartPtr<DOFsHandler> dofHand = Create<DOFsHandler>(disMesh);
82          dofHand->setNameTag("dofHand");
83          dofHand->setNumDOFs(1);
84          dofHand->setDOFs({"theta"});
85          dofHand->Update();
86          // ---------------------- Initial conditions ---------------------- //
87          //------------------------------------------------------------------//
88          double f;
89          for (int i = 0; i < disMesh->loc_nPts(); i++)
90          {
91                  // Coordinate
92                  double x = dofHand->mesh->nodeCoord(i, 0, hiperlife::IndexType::Local);
93                  f = 0.0 * x;
94                  // Initial condition
95                  dofHand->nodeDOFs->setValue("theta", i, IndexType::Local, f);
96          }
97
98          // Update
99          dofHand->UpdateGhosts();
100         // ---------------------- Boundary conditions ---------------------- //
101         //------------------------------------------------------------------//
102         dofHand->setBoundaryCondition(0, MAxis::Xmin, 0.0);
103         dofHand->setBoundaryCondition(0, MAxis::Xmax, 0.0);
104
105         dofHand->setBoundaryCondition(0, MAxis::Ymin, 0.0);
106         dofHand->setBoundaryCondition(0, MAxis::Ymax, 0.0);
107         // Update
108         dofHand->UpdateGhosts();
109         //------------------------------------------------------------------//
110         /// *****                HIPERPROBLEM CREATION                *****///
111         //------------------------------------------------------------------//
112
113         SmartPtr<HiPerProblem> hiperProbl = Create<HiPerProblem>();
114         hiperProbl->setParameterStructure(paramStr);
115         hiperProbl->setDOFsHandlers({dofHand});
116         hiperProbl->setIntegration("Integ", {"dofHand"});
117         hiperProbl->setCubatureGauss("Integ", 4);
118         hiperProbl->setElementFillings("Integ", LS);
119         hiperProbl->Update();
120
121         //------------------------------------------------------------------//
122         /// *****                  SOLVER CREATION                   *****///
123         //------------------------------------------------------------------//
124         SmartPtr<AztecOOIterativeLinearSolver> solver=Create<AztecOOIterativeLinearSolver>();
125         solver->setHiPerProblem(hiperProbl);
126         solver->setTolerance(1.E-8);
127         solver->setMaxNumIterations(500);
128         solver->setSolver(AztecOOIterativeLinearSolver::Solver::Gmres);
129         solver->setPreconditioner(AztecOOIterativeLinearSolver::Preconditioner::None);
130         solver->setDefaultParameters();
131         solver->setVerbosity(AztecOOIterativeLinearSolver::Verbosity::High);
132         solver->Update();
133         //------------------------------------------------------------------//
134         /// *****                SOLVE HIPERPROBLEM                  *****///
```

```
135         // ------------------------------------------------------------------//
136
137         // Time params
138         double maxTime  = 1.05;
139         double maxTimeSteps = 1000;
140
141         // Initialize
142         double time{delta_t};
143         int timeStep{1};
144
145         // Time loops
146         while ((time <= maxTime) && (timeStep < maxTimeSteps))
147         {
148                 // initial value for theta
149                 dofHand->nodeDOFs0->setValue(dofHand->nodeDOFs);
150                 // Update
151                 dofHand->UpdateGhosts();
152                 // time step info
153                 if (hiperProbl->myRank() == 0)
154                 {
155                         cout<<endl<<"TS:"<<timeStep<<"dt:"<<delta_t<<"|"
156                         <<"Time"<<time<<"of time"<<maxTime<<endl;
157                 }
158                 // ---------------------- Linear solver ----------------------------- //
159                 // ------------------------------------------------------------------ //
160                 bool converged = solver->solve();
161                 if (!converged)
162                 {
163                         throw runtime_error("Error: Linear solver does not converge.");
164                         break;
165                 }
166                 // Update solution
167                 solver->UpdateSolution();
168                 // Update time variables
169                 timeStep ++;
170                 time += delta_t;
171                 // ---------------------- Post Proccessing  ----------------------//
172                 // ------------------------------------------------------------------//
173                 string solName = "Conduct." + to_string(timeStep);
174                 dofHand->printFileLegacyVtk(solName,true);
175         }
176
177         hiperlife::Finalize();
178         return 0;
179 }
```

6.2 AuxHeatTransfer.h

```
1  #ifndef AUXHeat_H
2  #define AUXHeat_H
3
4  // C headers
5  #include <iostream>
6
7  // hiperlife headers
8  #include "hl_Core.h"
9  #include "hl_Parser.h"
10 #include "hl_TypeDefs.h"
11 #include "hl_DOFsHandler.h"
12 #include "hl_HiPerProblem.h"
13 #include "hl_FillStructure.h"
14 #include "hl_ParamStructure.h"
15 #include "hl_DistributedMesh.h"
16 #include "hl_StructMeshGenerator.h"
17 #include "hl_GlobalBasisFunctions.h"
18 #include "hl_NonlinearSolver_NewtonRaphson.h"
```

```
#include "hl_LinearSolver_Iterative_AztecOO.h"

// time parameters
// alpha = 0.0: forward;         delta_t = 0.001
// alpha = 0.5: crank-nicolson;  delta_t = 0.05
// alpha = 1.0: backward;        delta_t = 0.05

struct HeatParams
{
        enum RealParameters
        {
                delta_t,
                alpha,
        };
        HL_PARAMETER_LIST DefaultValues
        {
                {"delta_t,", 0.05},
                {"alpha,", 0.5},
        };
};

void LS(hiperlife::FillStructure& fillStr);

#endif
```

6.3 AuxHeatTransfer.cpp

```
// Header to auxiliary functions
#include "AuxHeatTransfer.h"

// Hiperlife headers
#include "hl_Core.h"
#include "hl_ParamStructure.h"
#include "hl_Parser.h"
#include "hl_TypeDefs.h"
#include "hl_GlobalBasisFunctions.h"
#include "hl_StructMeshGenerator.h"
#include "hl_DistributedMesh.h"
#include "hl_FillStructure.h"
#include "hl_DOFsHandler.h"
#include "hl_HiPerProblem.h"
#include "hl_LinearSolver_Iterative_AztecOO.h"
#include "hl_NonlinearSolver_NewtonRaphson.h"

using namespace std;
using namespace hiperlife;
using namespace hiperlife::Tensor;


// Conduction

void LS(hiperlife::FillStructure& fillStr)
{
        double alpha  = fillStr.getRealParameter(HeatParams::alpha);
        double delta_t = fillStr.getRealParameter(HeatParams::delta_t);

        //-------------------------------------------------------------------------------//
        // ------------------------------ INPUT DATA ------------------------------------//
        //-------------------------------------------------------------------------------//

        // Dimensions
        SubFillStructure& subFill = fillStr["dofHand"];
        int nDOFs = subFill.numDOFs;
        int eNN   = subFill.eNN;
        int nDim  = subFill.nDim;
        int pDim  = subFill.pDim;
```

```
40          // Nodal values at Gauss points
41          wrapper<double,1> nborDOFs(subFill.nborDOFs.data(),eNN);
42          wrapper<double,1> nborDOFs0(subFill.nborDOFs0.data(),eNN);
43
44          // Shape functions and derivatives at Gauss points
45          double jac;
46          wrapper<double,1>  bf(subFill.nborBFs(), eNN);
47          tensor<double,2> Dbf(eNN,pDim);
48          GlobalBasisFunctions::gradients(Dbf, jac, subFill);
49
50          //------------------------------------------------------------------------------//
51          // ------------------------------ OUTPUT DATA ------------------------------------//
52          //------------------------------------------------------------------------------//
53          wrapper<double,2> Ak(fillStr.Ak(0, 0).data(), eNN, eNN);
54          wrapper<double,1> Bk(fillStr.Bk(0).data(),eNN);
55
56          //------------------------------------------------------------------------------//
57          // ---------------------------- KNOWN VARIABLES ----------------------------------//
58          //------------------------------------------------------------------------------//
59
60          // Temperature
61          double theta = bf*nborDOFs0;
62          // Temperature Gradient
63          tensor<double,1> grad_theta(pDim);
64          for (int i = 0; i < eNN; i++)
65          {
66                  for (int d = 0; d < pDim; d++)
67                  grad_theta(d) += Dbf(i,d)*nborDOFs0(i);
68          }
69
70          //------------------------------------------------------------------------------//
71          //-------------------------------- EQUATIONS ------------------------------------//
72          //------------------------------------------------------------------------------//
73          for (int i = 0; i < eNN; i++)            //i for basis functions
74          {
75                  // (gradient of the basis function) *  (gradient of u)
76                  double dotdg{};
77                  for (int d = 0; d < pDim; d++)
78                  dotdg += Dbf(i,d)*grad_theta(d);
79                  // Fill RHS
80                  Bk(i) += jac * (bf(i)*theta + delta_t*bf(i) - delta_t*(1.0-alpha)*dotdg);
81                  for (int j = 0; j < eNN; j++)       //j for variable.
82                  {
83                          // (gradient of the basis function)*(gradient of the basis function)
84                          double dotdd{};
85                          for (int d = 0; d < pDim; d++)
86                          dotdd += Dbf(i,d)*Dbf(j,d);
87
88                          // Fill matrix
89                          Ak(i,j) += jac * (bf(i)*bf(j) + delta_t*alpha*dotdd);
90                  }
91          }
92  }
```

7 Results

In this section, we present the results of our solution. Table 1 shows the evolution of $\theta(0,0,t)$ in different time approximation scheme. These values is presented for Crank-Nicholson scheme in Figure 3. The contour demonstration of temperature in the whole domain at $t = 1$ is also shown in Figure 4.

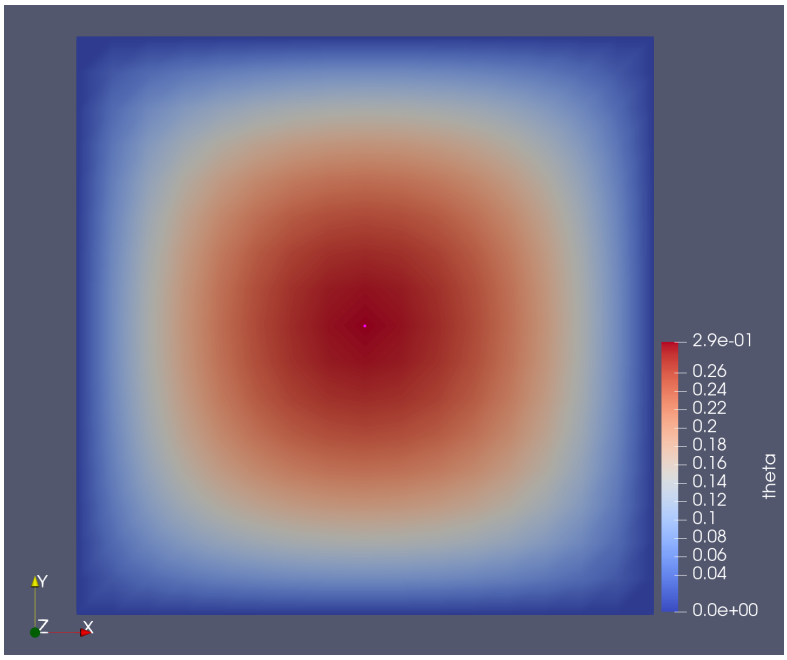

Figure 3: $\theta(x, y, 1.0)$

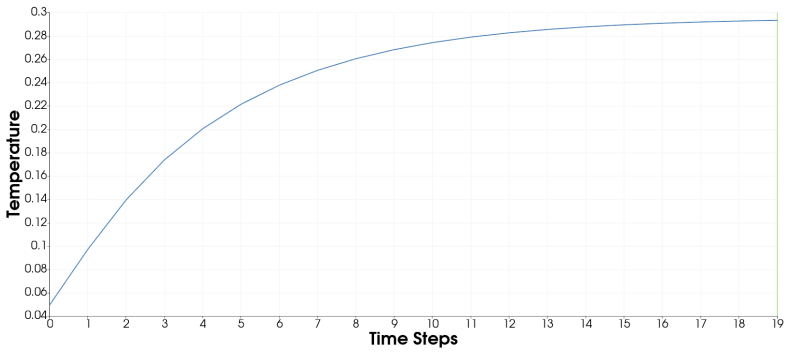

Figure 4: Evolution of $\theta(0, 0, t)$ for time steps.

Table 1: Evolution of $\theta(0, 0, t)$ with various time approximation schemes.

Time	Crank-Nocholson	Backward Difference	Forward Difference
0.05	0.0497	0.0480	0.0500
0.2	0.1740	0.1612	0.1737
0.4	0.2506	0.2395	0.2503
0.6	0.2790	0.2724	0.2788
0.8	0.2895	0.2860	0.2894
1.0	0.2933	0.2916	0.2933

References

[1] Junuthula Narasimha Reddy. *An Introduction to Nonlinear Finite Element Analysis Second Edition: with applications to heat transfer, fluid mechanics, and solid mechanics*. OUP Oxford, 2014.