

HiperLife Tutorial: Thermal Conduction

Arash Imani¹

¹LaCàN

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 distribute 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.

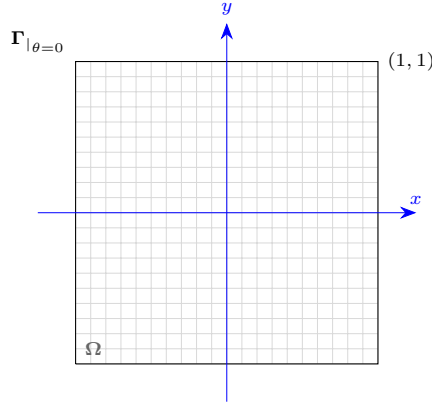


Figure 1: Geometry, BC and computational domain used for the analysis of transient heat transfer.

6

2 Governing Equations

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

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

where θ is the non-dimensional temperature in the domain 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 parameters of problem are listed as

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

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

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

15 3 Weak Form

16 The starting point for the development of the finite element models of Eq. (1) is their weak forms. By considering
 17 $\Delta t = t^{n+1} - t^n$, and substituting u with our temperature field, Eq. (3) takes the following form

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

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

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

20 combining these two equations gives us

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

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

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

22 where

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

23 and

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

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

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

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

27 4 Finite Element Model

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

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

31 Lets rewrite this equation for known and unknown variables.

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

32 The above equations can be written symbolically in matrix form as

$$\mathbf{K}\theta = \mathbf{F}. \quad (13)$$

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

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

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

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

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

36 5 Choice of Elements

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

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

41 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.

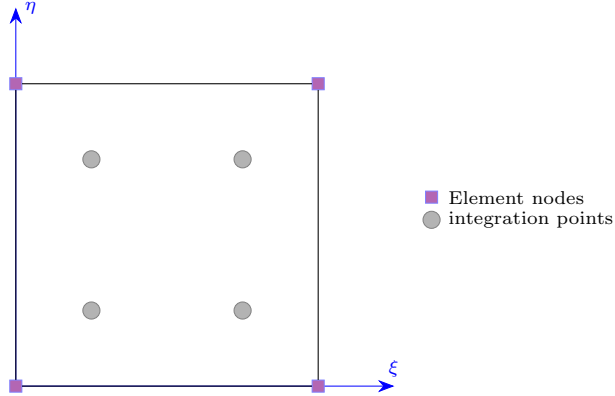


Figure 2: Linear quadrilateral element used for finite element model.

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

46 6 Implementation

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

```

1  /* Heat Transfer conduction*/
2  // cpp headers
3  #include <iostream>
4  #include <fstream>
5  #include <cmath>
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"
19 #include "hl_LinearSolver_Iterative_AztecOO.h"
20
21 // Header to auxiliary functions
22 #include "AuxHeatTransfer.h"
23
24 // =====//
25 //          MAIN FUNCTION          //
26 // =====//
27
28 int main(int argc, char** argv)
29 {
30     using namespace std;
31     using namespace hiperlife;
32     using namespace hiperlife::Tensor;
33
34     // =====//
35     // *****          INITIALIZATION          ***** //
36     // =====//
37
38     // Initialize MPI
39     hiperlife::Init(argc, argv);
40
41     // =====//
42     /// *****          DATA INPUT          ***** //
43     // =====//
44
45     // Put parameters in the user structure
46     SmartPtr<ParamStructure> paramStr = CreateParamStructure<HeatParams>();
47
48     // Data
49     paramStr->setRealParameter(HeatParams::delta_t, 0.05);
50     paramStr->setRealParameter(HeatParams::alpha, 0.5);
51     double delta_t = paramStr->getRealParameter(HeatParams::delta_t);
52     double alpha = paramStr->getRealParameter(HeatParams::alpha);
53
54     // =====//
55     /// *****          MESH CREATION          ***** //
56     // =====//
57
58     // Create a line structured mesh
59     SmartPtr<StructMeshGenerator> structMesh = Create<StructMeshGenerator>();
60     structMesh->setNDim(3);
61     structMesh->setBasisFuncType(BasisFuncType::Lagrangian);
62     structMesh->setBasisFuncOrder(1);
63     structMesh->setElemType(ElemType::Square);
64     structMesh->genRectangle(16, 16, 2.0, 2.0);
65     structMesh->translateX(-1.0);
66     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 Processing -----//
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"

```

```

19 #include "hl_LinearSolver_Iterative_AztecOO.h"
20
21 // time parameters
22 // alpha = 0.0: forward;          delta_t = 0.001
23 // alpha = 0.5: crank-nicolson;   delta_t = 0.05
24 // alpha = 1.0: backward;         delta_t = 0.05
25
26 struct HeatParams
27 {
28     enum RealParameters
29     {
30         delta_t ,
31         alpha ,
32     };
33     HLPARAMETERLIST DefaultValues
34     {
35         {"delta_t", 0.05},
36         {"alpha", 0.5},
37     };
38 };
39
40 void LS(hiperlife::FillStructure& fillStr);
41
42 #endif

```

6.3 AuxHeatTransfer.cpp

```

1 // Header to auxiliary functions
2 #include "AuxHeatTransfer.h"
3
4 // Hiperlife headers
5 #include "hl_Core.h"
6 #include "hl_ParamStructure.h"
7 #include "hl_Parser.h"
8 #include "hl_TypeDefs.h"
9 #include "hl_GlobalBasisFunctions.h"
10 #include "hl_StructMeshGenerator.h"
11 #include "hl_DistributedMesh.h"
12 #include "hl_FillStructure.h"
13 #include "hl_DOFsHandler.h"
14 #include "hl_HiPerProblem.h"
15 #include "hl_LinearSolver_Iterative_AztecOO.h"
16 #include "hl_NonlinearSolver_NewtonRaphson.h"
17
18 using namespace std;
19 using namespace hiperlife;
20 using namespace hiperlife::Tensor;
21
22
23 // Conduction
24
25 void LS(hiperlife::FillStructure& fillStr)
26 {
27     double alpha = fillStr.getRealParameter(HeatParams::alpha);
28     double delta_t = fillStr.getRealParameter(HeatParams::delta_t);
29
30     //-----//
31     // INPUT DATA -----//
32     //-----//
33
34     // Dimensions
35     SubFillStructure& subFill = fillStr["dofHand"];
36     int nDOFs = subFill.numDOFs;
37     int eNN = subFill.eNN;
38     int nDim = subFill.nDim;
39     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.

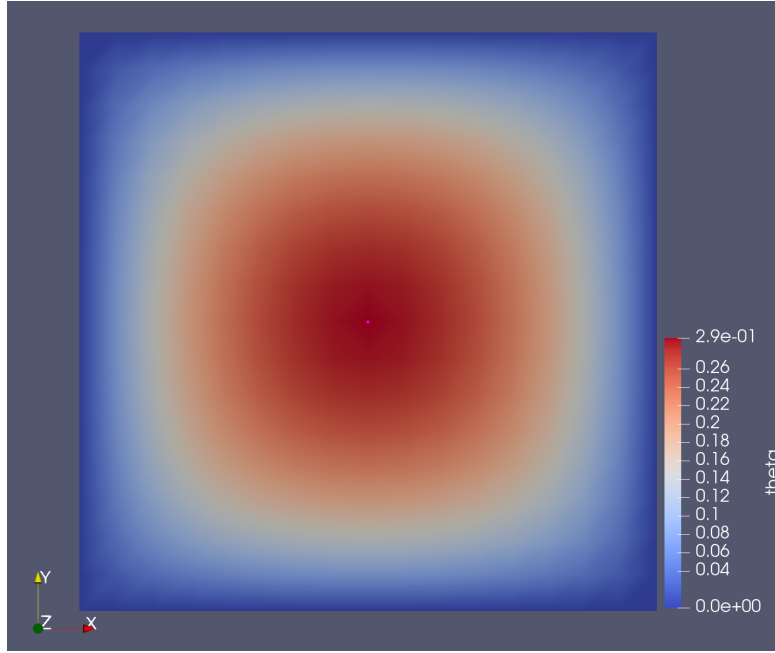


Figure 3: $\theta(x, y, 1.0)$

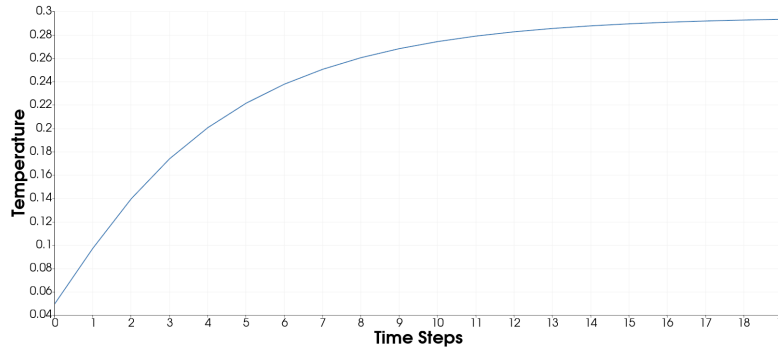


Figure 4: Evolution of $\theta(0, 0, t)$ for time steps.

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.