Parameter Estimation of Heat Distribution in Heterogeneous Media

Authors:
Muhammad Sajid Ali
Shubhaditya Burela
Pourya Pilva
Aneesh Futane

Supervisors:
Klaus Leppkes
Univ.-Prof. Dr.rer. nat. Uwe Naumann

A report submitted as the work done in
Simulation Science Laboratory

Software and Tools for Computational Engineering

RWTH AACHEN UNIVERSITY

Contents

1 Abstract

The main aim of this project is to solve a 2D heat diffusion equation by using finite difference method on a rectangular domain with given heat diffusivity, heat source and initial temperature with isothermal boundary conditions on either ends of the rectangular domain. The project also involves parameter optimization of the heat diffusivity for the obtained temperature values. The parameter is optimized by minimizing an objective function by gradient descent method. The gradients are obtained by algorithmic differentiation using `dco/c++` by either adjoint mode or tangent mode. The two modes are compared for their performance. Finally the heat diffusivity values obtained by parameter optimization are compared with the initial values. The temperature and the heat diffusivity values so obtained are written to a `vtk` file that can be visualized in `Paraview`.

2 Introduction

In this project we are going to solve a 2D heat equation model. We can divide our introduction into two main parts, with the fist part being Heat Distribution, as shown in Equation 1, and in the second part we are looking at the Parameter Estimation and optimize the heat diffusivity over the domain with respect to the measurement data. In Figure 1, we can see a schematic of a 2D plate with lengths of L_x and L_y that we use for our model problem. As we can see in the Figure 1 our heat source and heat diffusivity in the domain are space-dependent.

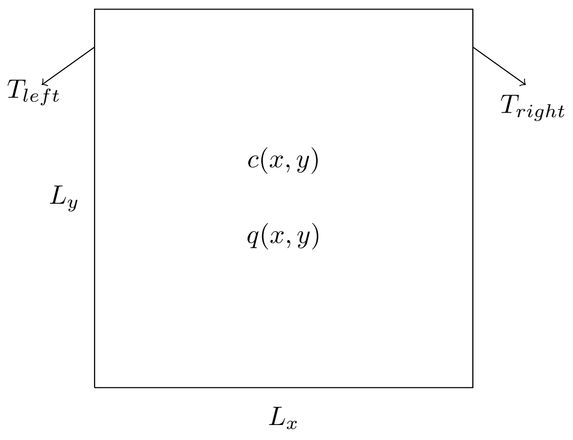


Figure 1: A schematic of 2D plane.

2.1 Heat Distribution

The general form of our 2 dimensional heat equation is as follows:

$$\frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(c(x,y)\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(c(x,y)\frac{\partial T}{\partial y}\right) + q(x,y) \quad \begin{array}{r} x \in [0, L_x] \\ y \in [0, L_y] \\ t \in [0, t_f] \end{array} \tag{1}$$

As we have already showed in the Figure 1 The boundary condition that we are considering for this problem are of the form:

$$T(x, y, t = 0) = T_{init} \quad x \in [0, L_x], y \in [0, L_y]$$

$$T(x = 0, y, t) = T_{left} \quad y \in [0, L_y], t \in [0, t_f]$$

$$T(x = L_x, y, t) = T_{right} \quad y \in [0, L_y], t \in [0, t_f]$$

In this equation we have $T = T(x, y, t, c)$ where T describes the temperature, x, y are the spatial coordinates, t represents time, $c(x, y)$ and $q(x, y)$ are the space-dependent heat diffusivity and heat flux of our source respectively.

2.2 Parameter Estimation

As we know in our model problem the temperature is dependent on the diffusivity array (c array) and here we are trying to find the optimized c values. Thus,we are using the following cost function equation (2) which measures the error between the observed temperature values and the measurement data. Afterwards we should try to minimize our cost function which can be achieved through different methods and among them we chose to use the steepest descent algorithm with a fixed step size of α. Thus, as we can see in equation (3) we update c in each optimization iteration and the iterations are performed till a particular optimization error is reached.

$$J(c) = \frac{1}{n}\sum_{j=0}^{n-1}(T_j^o(c) - T_j^m)^2 \tag{2}$$

$$c^{i+1} = c^i - \nabla_i T(c) = c^i - \alpha \cdot \frac{dJ}{dc}. \tag{3}$$

where, T_j^m is the measured temperature value for each node in the mesh. T_j^o is the experimental observation value for each node in the mesh and $c(x, y)$ is the parameter being optimized in each each iteration step.

3 Theory

In order to start solving the temperature over the 2D domain, the first step would be to discretize the equation shown in equation (1) which is described in the next subsection more in details.

3.1 Discretization in space

We consider the dimensions of our plate as L_x and L_y with n_x number of discretization nodes in x-direction and and n_y number of discretization nodes in y-direction. The step sizes in each direction can be represented as:

$$\begin{aligned}\Delta x &= \frac{L_x}{n_x - 1}\\ \Delta y &= \frac{L_y}{n_y - 1}\end{aligned} \tag{4}$$

In order to discretize Equation (1), central difference scheme is chosen. The second order derivatives in x and y-directions discretized are shown in equations

(5) and (6) respectively.

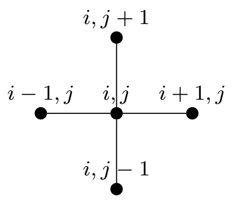
$$\begin{aligned}\frac{\partial}{\partial x}\left(c(x,y)\frac{\partial T_{i,j}}{\partial x}\right) \approx \frac{1}{\Delta x}\Bigg(&\left(\frac{c_{i+1,j}+c_{i,j}}{2}\right)\left(\frac{T_{i+1,j}+T_{i,j}}{\Delta x}\right) \\ &-\left(\frac{c_{i,j}+c_{i-1,j}}{2}\right)\left(\frac{T_{i,j}+T_{i-1,j}}{\Delta x}\right)\Bigg) + O(\Delta x)^2\end{aligned} \tag{5}$$

$$\begin{aligned}\frac{\partial}{\partial y}\left(c(x,y)\frac{\partial T_{i,j}}{\partial y}\right) \approx \frac{1}{\Delta y}\Bigg(&\left(\frac{c_{i,j+1}+c_{i,j}}{2}\right)\left(\frac{T_{i,j+1}+T_{i,j}}{\Delta y}\right) \\ &-\left(\frac{c_{i,j}+c_{i,j-1}}{2}\right)\left(\frac{T_{i,j}+T_{i,j-1}}{\Delta y}\right)\Bigg) + O(\Delta y)^2\end{aligned} \tag{6}$$

For all of the equation the i, j are:

$$i = 0, 1, ..., n_x - 1$$
$$j = 0, 1, ..., n_y - 1$$

The stencil for the above discretization looks as follows.



We can combine the equations (5) and (6) to reach the general two dimensional equation:

$$\begin{aligned}&\frac{\partial}{\partial x}\left(c(x,y)\frac{\partial T_{i,j}}{\partial x}\right) + \frac{\partial}{\partial y}\left(c(x,y)\frac{\partial T_{i,j}}{\partial y}\right) = \\ &\frac{1}{\Delta y^2}\left(\frac{c_{i,j}+c_{i,j-1}}{2}\right)T_{i,j-1} + \frac{1}{\Delta x^2}\left(\frac{c_{i,j}+c_{i-1,j}}{2}\right)T_{i-1,j} - \\ &\left(\frac{1}{\Delta x^2}\left(\frac{c_{i-1,j}+2c_{i,j}+c_{i+1,j}}{2}\right) + \frac{1}{\Delta y^2}\left(\frac{c_{i,j-1}+2c_{i,j}+c_{i,j+1}}{2}\right)\right)T_{i,j} + \\ &\frac{1}{\Delta x^2}\left(\frac{c_{i+1,j}+c_{i,j}}{2}\right)T_{i+1,j} + \frac{1}{\Delta y^2}\left(\frac{c_{i,j+1}+c_{i,j}}{2}\right)T_{i,j+1}\end{aligned} \tag{7}$$

3.2 Discretization in Time

Now that we have the space discretization we move on to the time discretization, time is discretized equidistantly with m as total number of time steps and the fixed time step size can be represented as:

$$\Delta t = \frac{t_f}{m} \tag{8}$$

Where $t_f$is the final time and m is the number of time steps.
In order to discretize our equation in time,we use the backward Euler method. The results of spatial and temporal discretization are shown in equation (9).

$$\frac{\partial T_{i,j}}{\partial t} = \frac{T_{i,j}^{n+1} - T_{i,j}^{n}}{\Delta t} + O(\Delta t) \tag{9}$$

3.3 Linear system

Now we will apply the discretization to our initial problem in equation (1) to get the complete discretized version of our 2D heat equation:

$$\begin{aligned}\frac{T_{i,j}^{n+1} - T_{i,j}^{n}}{\Delta t} &= \frac{1}{\Delta y^2}\left(\frac{c_{i,j}+c_{i,j-1}}{2}\right)T_{i,j-1}^{n+1} + \frac{1}{\Delta x^2}\left(\frac{c_{i,j}+c_{i-1,j}}{2}\right)T_{i-1,j}^{n+1} - \\ &\left[\frac{1}{\Delta x^2}\left(\frac{c_{i-1,j}+2c_{i,j}+c_{i+1,j}}{2}\right) + \frac{1}{\Delta y^2}\left(\frac{c_{i,j-1}+2c_{i,j}+c_{i,j+1}}{2}\right)\right]T_{i,j}^{n+1} \\ &+\frac{1}{\Delta x^2}\left(\frac{c_{i+1,j}+c_{i,j}}{2}\right)T_{i+1,j}^{n+1} + \frac{1}{\Delta y^2}\left(\frac{c_{i,j+1}+c_{i,j}}{2}\right)T_{i,j+1}^{n+1} + q_{i,j}\end{aligned} \tag{10}$$

Then next step is to rearrange equation (10) so that we can write in the form of a linear system.

$$\begin{aligned}&-\frac{\Delta t}{\Delta y^2}\left(\frac{c_{i,j}+c_{i,j-1}}{2}\right)T_{i,j-1}^{n+1} - \frac{\Delta t}{\Delta x^2}\left(\frac{c_{i,j}+c_{i-1,j}}{2}\right)T_{i-1,j}^{n+1} + \\ &\left[1 + \frac{\Delta t}{\Delta x^2}\left(\frac{c_{i-1,j}+2c_{i,j}+c_{i+1,j}}{2}\right) + \frac{\Delta t}{\Delta y^2}\left(\frac{c_{i,j-1}+2c_{i,j}+c_{i,j+1}}{2}\right)\right]T_{i,j}^{n+1} - \\ &\frac{\Delta t}{\Delta x^2}\left(\frac{c_{i+1,j}+c_{i,j}}{2}\right)T_{i+1,j}^{n+1} - \frac{\Delta t}{\Delta y^2}\left(\frac{c_{i,j+1}+c_{i,j}}{2}\right)T_{i,j+1}^{n+1} = T_{i,j}^{n} + \Delta t\, q_{i,j}\end{aligned} \tag{11}$$

Now that our equation is rearranged so that it has the form of the linear system as we can write the matrix form as:

$$(I - \Delta t\, R)T^{n+1} = T^{n} + \Delta t\, Q \tag{12}$$

Where $T^n \in \mathbb{R}^{n_x \times n_y}$ and $T^{n+1} \in \mathbb{R}^{n_x \times n_y}$ are the temperature vectors in the current time step and the next time step respectively. $R \in \mathbb{R}^{n_x \times n_y} \times \mathbb{R}^{n_x \times n_y}$ is the stiffness matrix and $Q \in \mathbb{R}^{n_x \times n_y}$ is the heat source vector. Furthermore it is worth mentioning that we are using lexicographical slicing here as follows.

$$T^n = \begin{bmatrix} T^n_{0,0} \\ T^n_{1,0} \\ \vdots \\ T^n_{n_x-1,0} \\ T^n_{0,1} \\ \vdots \\ T^n_{i,j-1} \\ \vdots \\ T^n_{i-1,j} \\ T^n_{i,j} \\ T^n_{i+1,j} \\ \vdots \\ T^n_{i,j+1} \\ \vdots \\ T^n_{n_{x-1},n_{y-1}} \end{bmatrix} \qquad Q = \begin{bmatrix} Q_{0,0} \\ Q_{1,0} \\ \vdots \\ Q_{n_{x-1},0} \\ Q_{0,1} \\ \vdots \\ Q_{i,j-1} \\ \vdots \\ Q_{i-1,j} \\ Q_{i,j} \\ Q_{i+1,j} \\ \vdots \\ Q_{i,j+1} \\ \vdots \\ Q_{n_{x-1},n_{y-1}} \end{bmatrix}$$

Our R matrix in the linear equation is:

$$\text{R=}\begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & \cdots & \cdots & 0 & \cdots & \cdots & \cdots & 0 & 0 \\ \vdots & 0 & \ddots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & \vdots & 0 & 1 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \vdots & & & & \ddots & & & & & \cdots \\ & \cdots & -S^y_{i,j+1} & \cdots & -S^x_{i-1,j} & (S^x_{i-1,j} + S^x_{i+1,j} + S^y_{i,j-1} + S^x_{i,j+1}) & -S^x_{i+1,j} & \cdots & -S^x_{i,j+1} & \cdots \\ \vdots & & & & & & & & & \cdots \\ \vdots & & & & & & & \vdots & \cdots & \cdots \\ \vdots & & & & & \vdots & 1 & & \cdots & \cdots \\ \vdots & & & & & \vdots & & \ddots & \cdots & \cdots \\ \vdots & \vdots & & & & & \vdots & & 1 & \cdots \\ \vdots & \vdots & \vdots & & & & & \vdots & & 1 \end{bmatrix}$$

Where $S^x_{i-1,j}, S^x_{i+1,j}, S^y_{i,j-1}, S^x_{i,j+1}$ are defined by the following equations:

$$\begin{aligned}
S^x_{i-1,j} &= \frac{1}{\Delta x^2}\left(\frac{c_{i-1,j}+c_{i,j}}{2}\right)\\
S^x_{i+1,j} &= \frac{1}{\Delta x^2}\left(\frac{c_{i+1,j}+c_{i,j}}{2}\right)\\
S^y_{i,j-1} &= \frac{1}{\Delta y^2}\left(\frac{c_{i,j-1}+c_{i,j}}{2}\right)\\
S^y_{i,j+1} &= \frac{1}{\Delta y^2}\left(\frac{c_{i,j+1}+c_{i,j}}{2}\right)
\end{aligned} \tag{13}$$

4 The code

The code for the project is written in `c++`. The main code reads the inputs for the simulation from a input `json` file and initializes the temperature, heat source and heat diffusivity vectors for the simulation. The code then discritizes the problem as explained in the earlier section and solves the arising linear equation. The temperature thus obtained is stored as the observed temperature values and is then used for parameter optimization of the heat diffusivity. Parameter optimization is done either by the adjoint mode or the tangent mode of algorithmic differentiation using `dco/c++`. The 2D heat diffusion problem is once again solved for the optimized heat diffusivity values. The temperature and the heat diffusivity values are written to a `vtk` file for visualization in `Paraview`. The code consists of the following `c++` files.

1. `main.cpp`: This is the main executable file which executes the entire program. It uses the following header files for its execution.
2. `input.h`: Contains the `input` class and functions used to read the input `json` file and initialize the vectors.
3. `diffusion.h`: Contains all the helper functions that are required for the main code.
4. `solver.h`: Contains the solver for the problem. This includes creating the LHS of the equation (12) and a time loop that iteratively creates the RHS of equation (12) and solves the arising linear systems till the end time.
5. `optimizer.h`: Contains the optimizer function required to optimize the heat diffusivity values by gradient descent method. It consists of the cost function and functions to compute gradients by either the adjoint mode or the tangent mode of algorithmic differentiation.
6. `visualize.h`: Contains the function required to write the `vtk` file.
7. `test_solve.cpp`: This is a separate executable file that consists of all the google tests. This file is executed for testing the code.

4.1 Input file

A class by name `input` is used to parse the values in the input file which is written in the `json` file format. `Rapidjson` library is used to parse the values from the input `json` file with the help of the method `read_input()`. The values are then stored in the defined `input` class. The input file in the `json` file format contains the following information.

Name - The name of the test case
L_x *- Length in x-direction*
L_y *- Length in y-direction*
n_x *- Number of points in x-direction*
n_y *- Number of points in y-direction*
t_f *- Final time*
m *- Number of time steps*
T_{init} *- Initial temperature*
T_{left} *- Left wall boundary temperature*
T_{right} *- Right wall boundary temperature*
n_c *- Number of different heat diffuitivity cases*
c_1 *- Heat diffitivity values in the form of* $[c, x_{start}, x_{end}, y_{start}, y_{end}]$
q *- Heat source values in the form of* $[q, x_{start}, x_{end}, y_{start}, y_{end}]$
c_{init} *- Heat diffusivity starting value for parameter optimization*
α *- Descent step size*
opt_steps - Number of maximum optimization steps
opt_err - RMS error of gradient to stop the optimization loop

The required test cases are loaded as input to run the simulation for the different test cases. The input file for the first test case `case_1.json` looks as follows.

case_1.json
```
{
    "name": "Case 1: Single diffusivity",
    "Lx": 1.0,
    "Ly": 1.0,
    "nx": 10,
    "ny": 10,
    "tf": 40,
    "m": 400,
    "T_init": 300,
    "T_left": 300,
    "T_right": 330,
    "q": [1.0,0.4,0.6,0.4,0.6],
    "nc": 1,
    "c1": [0.0001,0.0,1.0,0.0,1.0],
    "c_init": 0.0002,
    "alpha": 1e-8,
    "opt_steps": 1000,
    "opt_err": 1e-8
}
```

4.2 Linear Solver

Here we describe the methods we used for solving the linear system of equations generated by the formulation of the heat diffusion problem. We have implemented *Gauss Elimination algorithm* for our linear system. To compare the performance of our algorithm we have also used *Dense* and *Sparse* solution algorithms for solving the linear system of equations from the external library *Eigen*. All the three methods are defined as `solve_ls_gauss()`, `solve_ls_dense()` and `solve_ls_sparse()` respectively and can be setup as required in the `solver.h` header file. All the algorithms used are described below:

4.2.1 Gauss elimination algorithm

For the linear system $Ax = b$ we formulate an augmented matrix $(A|b)$. This augmented matrix system is given below:

$$(A|b) = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} & b_1 \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} & b_n \end{pmatrix}$$

We then use use elementary row operations to transform the augmented matrix into the form:

$$\begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} & b_1 \\ 0 & u_{2,2} & \cdots & u_{2,n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & u_{m,n} & b_n \end{pmatrix}$$

This form makes it very easy to solve. We immediately see that $x_n = b_n$. We can then take this and substitute this into the equation $x_{n-1} + a_{(n-1,n)}x_n = b_{n-1}$ and solve for x_{n-1}. Repeating this process and we obtain the entire solution to the original system by successive Back Substitution. This entire technique is known as Gaussian Elimination algorithm.

4.2.2 Dense Linear solver

As stated previously, we have used pre-built libraries to solve the formulated linear systems to compare the performance of Gauss elimination algorithm with the algorithm used. Here we use *LU decomposition with partial pivoting* to solve our problem. The minimum requirement for this algorithm is the invertibility of the coefficient matrix.LU decomposition refers to the decomposition of A(coefficient matrix), with proper row and/or column orderings or permutations, into two factors – a lower triangular matrix L and an upper triangular matrix U:

$$A = LU \tag{14}$$

LU decomposition with partial pivoting *(LUP)* refers often to LU decomposition with row permutations only:

$$PA = LU \tag{15}$$

P is a permutation matrix, which, when left-multiplied to A, reorders the rows of A.
The condition of invertibility can be given away with if we use LU decomposition with full pivoting. But because of its slower execution compared to partial pivoting and also our previous knowledge about the invertibility of coefficient matrix we tend to use partial pivoting approach.

4.2.3 Sparse Linear solver

The coefficient matrix which is generated in our problem is of sparse type. It has only a few non-zero elements. We therefore have used *Sparse LU* decomposition from *Eigen* library for our problem. This solver by default uses column major storage format for the non zero entries of the coefficient matrix. The sparse LU is suitable for any square matrix.

4.3 Optimizer

The main goal of this project is to optimize the values of heat diffusivity over the 2D rectangular domain given by minimizing the cost function(J):

$$J(c) = \frac{1}{n-1}\sum_{j=0}^{n-1}(T_j^m(c) - O_j)^2 \tag{16}$$

where $T_j^m(c)$ is the temperature depending on c and O_j is the experimental observation value of temperature at each and every node. To minimize this cost function various algorithms are available, among which we choose steepest descent algorithm to minimize our cost function J. The update step for this algorithm is given as:

$$c^{i+1} = c^i - s \cdot \frac{dJ}{dc} \tag{17}$$

where s is the descent step size and $\frac{dJ}{dc}$ is the gradient of our cost function with respect to heat diffusivity values. The major challenge in this algorithm is to find out the gradient term. Analytical approach is these cases is generally the first point of consideration, but because of the complicated implicit dependency amongst the variables, it is difficult to formulate on computer. In similar fashion *finite differences* can be used to approximate the gradient, but because of the round-off errors generated with dicretization process, is not suitable for our problem. Finally, both these methods are slow at computing partial derivatives of a function with respect to many inputs, as is needed in our problem. We therefore use *Algorithmic differentiation* to calculate the gradient.
Algorithmic differentiation is the tool which we use to calculate the derivatives

by computer. There are two different modes to calculate the derivatives : *Tangent Linear mode* and *Adjoint mode*. Let us consider a function:

$$y = F(x)\ : R^n \rightarrow R^m \tag{18}$$

The jacobian matrix of the function given above is $\nabla F(x) \in R^{m\times n}$. Now the tangent mode starts from input variable x and calculates the jacobian ∇F column by column, and the adjoint mode starts from the output variable y and calculates the jacobian ∇F row by row. In our case where we only a scalar output of the cost function $J(m = 1)$ and an array as input, the adjoint mode only needs to go through the row of the jacobian once, but the tangent mode will have to compute the jacobian column by column n times. Therefore in our cases we prefer to use adjoint mode over tangent mode.

4.4 Visualization

The code writes `vtk` files in the rectilinear grid format as output. The file `visualize_final_0.vtk` contains the temperature T and heat diffusivity c values obtained by solving the heat diffusion using the input c values.The file `visualize_final_1.vtk` contains the temperature T and heat diffusivity c values obtained by solving the heat diffusion using the optimized c values. The files are visualized in `Paraview`. The files with group name `visualize.vtk` shows the temperature T and heat diffusivity c values obtained from the optimized c values at every iteration of optimization.

5 Results

In order to validate our code after implementing the solver and optimization procedure, we discuss here 3 different test cases along with the paraview plots of the results of these cases. The result discussion is provided at the end of this section. We will also look at the different timings obtained with different linear solvers and optimization method for various mesh sizes.

5.1 Test Case 1

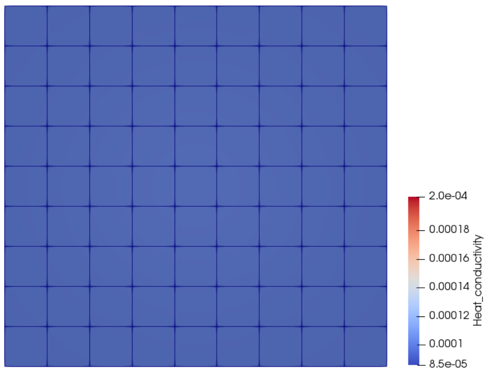
The input conditions for this test case are given below. This is a homogeneous heat diffusivity case and we consider only a single heat source at the centre of the rectangular domain. Our domain size is standard $L_x \times L_y = 1.0 \times 1.0$. The final time in temperature solver is 40 s. We solve our problem with first order adjoint mode. The number of time steps taken are 400. As already described above we use gradient descent algorithm for solving the optimization problem. The descent step which we consider is taken to be $1e - 8$ to prevent any overshooting and diverging solution of the gradient descent algorithm.

The temperature was specified at the boundaries as 330K at the right boundary and 300K at the left boundary with heat source of 1.0 at the centre of

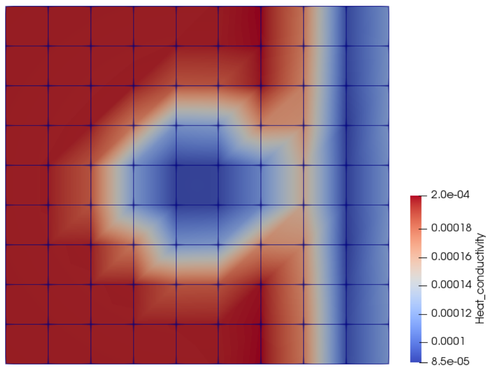
Parameter	**value**
L_x	1.0
L_y	1.0
n_x	10
n_y	10
T_{init}	300
T_{left}	300
T_{right}	330
Heat diffusivity c	0.0001
Heat Source q	1.0 at $(0.45, 0.55) \times (0.45, 0.55)$
Time step size dt	0.1
Number of time steps m	400
c_{init} for optimization	0.0002
Number of optimization steps	1000
Size of descent step α	1e-8

Table 1: Parameters for Test case 1

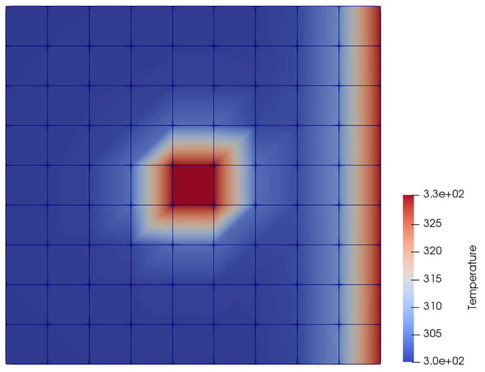
the domain. The initial heat diffusivity values given were 0.0001 (Figure 1.(a)) which were then perturbed to 0.0002 and the optimization problem was run with gradient descent algorithm.



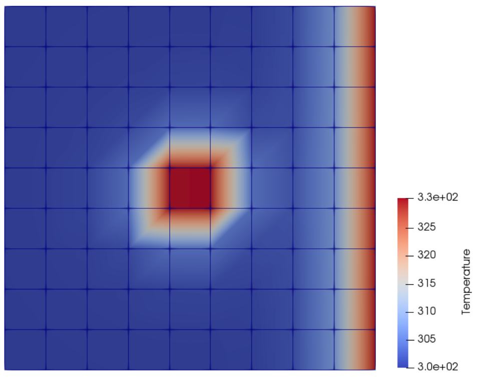
(a) Target heat diffusivity



(b) Optimized heat diffusivity



(c) Observed temperature



(d) Measured temperature

Figure 2: Temperature and heat diffusivity plots for *test case 1*

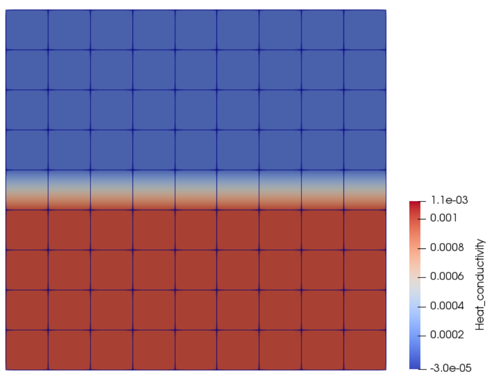
5.2 Test Case 2

The input conditions for this test case are given below. This is a heterogeneous heat diffusivity case with two different heat diffusivities and we consider only a single heat source at the centre of the rectangular domain. Our domain size is standard $L_x \times L_y = 1.0$. The final time in temperature solver is 40 s.

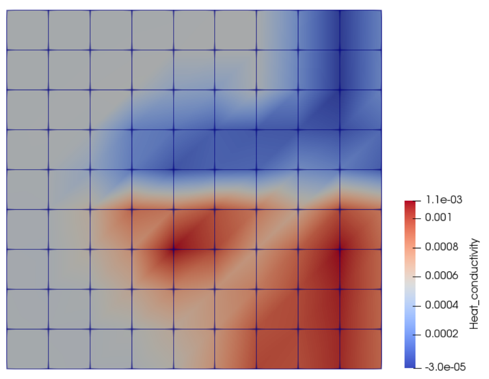
Parameter	**value**
L_x	1.0
L_y	1.0
n_x	10
n_y	10
T_{init}	300
T_{left}	300
T_{right}	330
$c1$	0.001 at $(0, 1.0) \times (0, 0.5)$
$c2$	0.0001 at $(0, 1.0) \times (0.5, 1.0)$
Heat Source q	1.0 at $(0.45, 0.55) \times (0.45, 0.55)$
Time step size dt	0.1
Number of time steps m	400
c_{init} for optimization	0.0005
Number of optimization steps	1000
Size of descent step α	1e-8

Table 2: Parameters for Test case 2

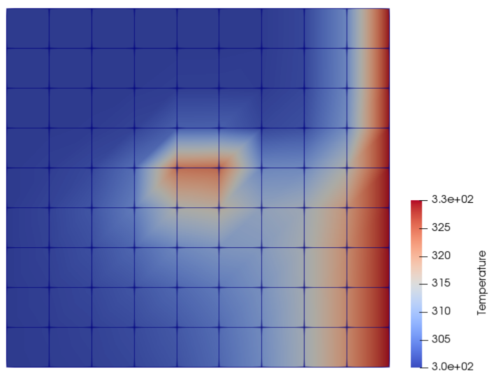
The temperature was specified at the boundaries as 330K at the right boundary and 300K at the left boundary with heat source of 1.0 at the centre of the domain. The initial heat diffusivity values given were 0.001 and 0.0001 (Figure 3.(a)) which were then perturbed to 0.0005 and the optimization problem was run with gradient descent algorithm.



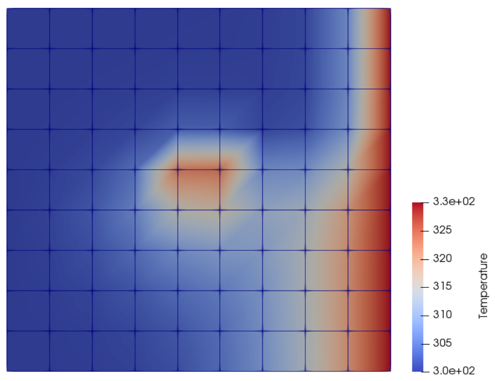
(a) Target heat diffusivity



(b) Optimized heat diffusivity



(c) Observed temperature



(d) Measured temperature

Figure 3: Temperature and heat diffusivity plots for *test case 2*

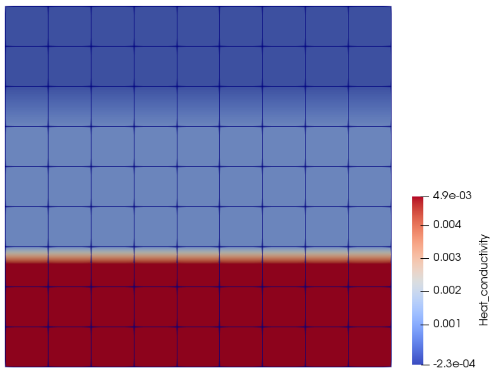
5.3 Test Case 3

The input conditions for this test case are given below. This is a heterogeneous heat diffusivity case with three different heat conductivities and we consider only a single heat source at the centre of the rectangular domain. Our domain size is standard $L_x \times L_y = 1.0$. The final time in temperature solver is 40 s.

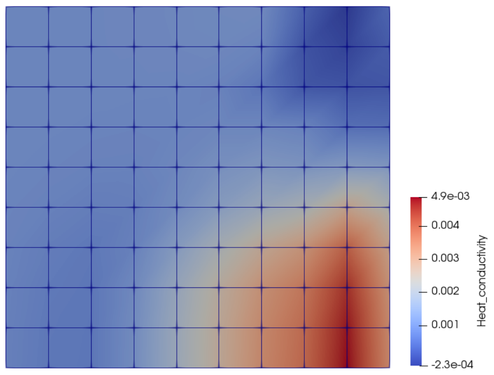
Parameter	**value**
L_x	1.0
L_y	1.0
n_x	10
n_y	10
T_{init}	300
T_{left}	300
T_{right}	330
$c1$	0.01 at $(0, 1.0) \times (0, 0.33)$
$c2$	0.001 at $(0, 1.0) \times (0.33, 0.67)$
$c3$	0.0001 at $(0, 1.0) \times (0.67, 1.0)$
Heat Source q	1.0 at $(0.45, 0.55) \times (0.45, 0.55)$
Time step size dt	0.1
Number of time steps m	400
c_{init} for optimization	0.001
Number of optimization steps	1000
Size of descent step α	1e-8

Table 3: Parameters for Test case 3

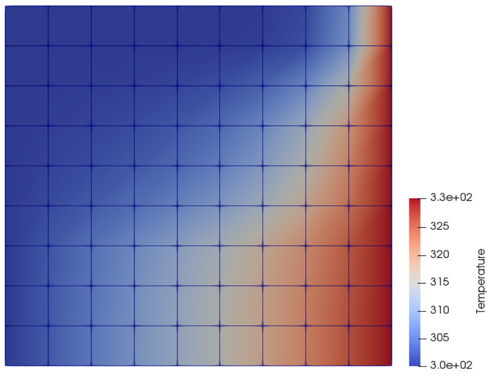
The temperature was specified at the boundaries as 330K at the right boundary and 300K at the left boundary with heat source of 1.0 at the centre of the domain. The initial heat diffusivity values given were 0.01, 0.001 and 0.0001 (Figure 4.(a)) which were then perturbed to 0.001 and the optimization problem was run with gradient descent algorithm.



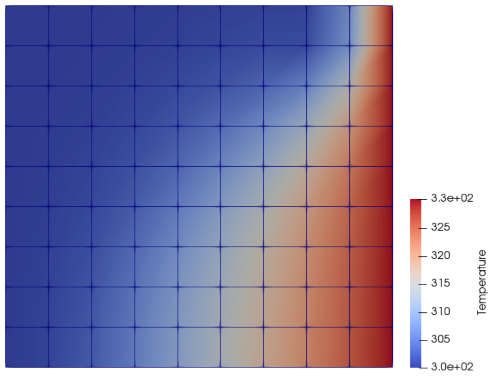
(a) Target heat diffusivity



(b) Optimized heat diffusivity



(c) Observed Temperature



(d) Measured Temperature

Figure 4: Temperature and heat diffusivity plots for *test case 3*

5.4 Result discussion

In this part we discuss the behaviour of the results and the inferences associated with them. The 2D heat diffusion problem which we solve by optimizing the heat diffusivity values is an inverse problem that is typically ill-posed. This means it does not have a unique solution.
Consequently we observe from (Figure 2.(b), 3.(b), 4.(b)) that the algorithm is not able to learn the heat diffusivity values properly. Moreover the application of gradient based methods to solve such ill-posed problems are not without their own drawbacks. The gradient based methods although are very fast and can reach the minimum in a short span of time, but are highly susceptible to getting stuck in the local minimum. The gradient based algorithms are also highly sensitive to the initial guess values and as in our case we have random initial guess values, it becomes difficult to find the correct initial guess value to land the optimization in the global minimum, which can now only be found out by try and error method.
We also observe that the gradients computed in the red region in (Figure 2.(b)) are very low due to which the algorithm is not able to learn the values properly. This region is referred to as *plateau*. This is a region where the cost function is flat, or nearly flat i.e. the derivative is zero or very close to zero. Gradient descent performs very badly on the plateaux because the parameters change very slowly. As we have the cost function depending upon multiple heat diffusivity values, the algorithm actually is multivariate gradient descent algorithm where the heat diffusivity values at all the grid points are treated as independent parameters to be optimized. We also observe (Figure 2.(d), 3.(d), 4.(d)) that the optimized temperature values are still in accordance with the observed temperature values. This can be attributed to the fact that the multivariate gradient descent algorithm performs the traversing individually for all the parameters involved:

$$c^i_{new} = c^i_{old} - \alpha_1 \frac{\partial J}{\partial c^i_{old}} \qquad \text{where } i \in (0, N) \tag{19}$$

But as stated earlier, some will get stuck in the plateau region and some updates get to the local minimum. And as previously stated that the problem being ill-posed can have multiple solutions, we get the same temperature profile even for an entirely different optimized heat diffusivity profile because the cost function is anyhow minimized.

5.5 Timings

In this section we discuss the timings of the linear solvers used in our problem along with the timings of the tangent and adjoint mode for optimization. From (Figure.5) we observe that with increasing grid size the time taken by the Gauss elimination method to solve the problem increases very fastly and is very high in comparison to the time taken by the sparse solver. Thus due to the sparse structure of matrix formulated by the problem, it is an appropriate choice to go for a sparse solver for better computational efficiency and lesser computational

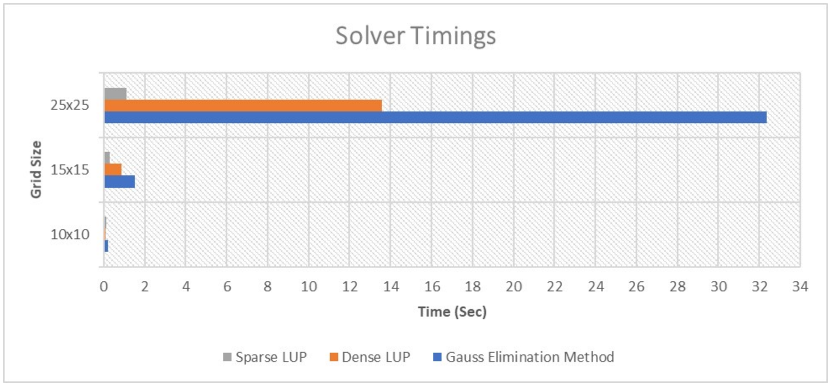


Figure 5: Solver Timings for 1 time solve

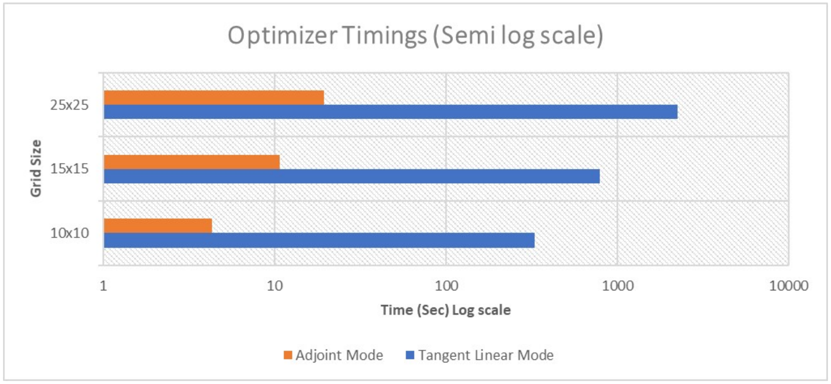


Figure 6: Optimizer Timings for 20 optimization iterations

cost.
Now regarding the time study of the optimizer, as expected we see (Figure.6) that the tangent mode takes much more time than adjoint mode for the concerned optimization problem. The graph has been plotted on a semi-log scale to scale the timings because of the large differences in the timings of both the methods. Thus for the present problem statement the adjoint mode is the appropriate choice for the optimization of the problem.

6 Conclusion

The required problem of 2d heat diffusion was solved successfully by finite difference approximations. The parameter estimation of heat diffusivity for the obtained temperatures by gradient descent method was successfully solved using the algorithmic differentiation tool dco/c++ for obtaining gradients. The optimized heat diffusivity values were comparable to the target heat diffusivity values wherever sufficient temperature gradients were observed. The temperature distribution obtained by the optimized heat diffusivity values is very similar to the initial temperature distribution despite the fact that the heat diffusivity values vary greatly. This shows the ill-posed nature of the parameter estimation part of our problem which is an inverse problem. It is also observed that the Eigen/Sparse Solver is the fastest when compared to Gaussian Elimination and Eigen/Dense for our linear equation system as the matrix we obtain using finite difference approximation for our problem is highly sparse. The Adjoint mode is much faster than the Tangent mode for our problem as our objective function is a multivariate scalar function and the gradient is harvested row-wise in adjoint mode but column by column in Tangent Mode.
As we saw that the ill-posedness of the problem is a major factor in getting non-unique solution for the problem, as a future prospect, various regularization methods(Tikhonov Regularization, TVD) can be implemented to regularize the ill-posed problem. This will result in a unique solution for the problem. Also as the gradient based methods are susceptible to problems described in the report, non-gradient based methods can be used for minimizing the cost function as these methods employ a global search algorithm instead of a local search algorithm (Gradient descent) for finding the minima. Moreover in order to make the gradient descent work properly is to formulate a cost function which is purely convex so that the function has a single minima which is the global minima.

7 Bibliography

[1]. Univ.-Prof.Dr.rer. nat. Uwe Naumann. Lecture Notes of Computational Differentiation at RWTH Aachen. WS2019/20.

[2]. dco/c++ user guide,www: http://www.nag.co.uk

[3]. Eigen library. http://eigen.tuxfamily.org/index.php?title=Main$_{Page}$