$$\frac{T_{i,j}^{n+1}-T_{i,j}^{n}}{\Delta t} = \frac{1}{\Delta y^2}\left(\frac{c^{i,j}+c^{i,j-1}}{2}\right)T_{i,j-1}^{n+1} + \frac{1}{\Delta x^2}\left(\frac{c^{i,j}+c^{i-1,j}}{2}\right)T_{i-1,j}^{n+1}$$
$$-\left\{\frac{1}{\Delta x^2}\left(\frac{c^{i-1,j}+2c^{i,j}+c^{i+1,j}}{2}\right) + \frac{1}{\Delta y^2}\left(\frac{c^{i,j-1}+2c^{i,j}+c^{i,j+1}}{2}\right)\right\}T_{i,j}^{n+1}$$
$$+\frac{1}{\Delta x^2}\left(\frac{c^{i+1,j}+c^{i,j}}{2}\right)T_{i+1,j}^{n+1} + \frac{1}{\Delta y^2}\left(\frac{c^{i,j+1}+c^{i,j}}{2}\right)T_{i,j+1}^{n+1} + q_{i,j}$$

$$\frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(c(x,y)\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(c(x,y)\frac{\partial T}{\partial y}\right) + q(x,y)$$

$$\Delta x = \frac{L_x}{n_x - 1}, \quad \Delta y = \frac{L_y}{n_y - 1}$$

$$J(c) = \frac{1}{n}\sum_{j=0}^{n-1}\left(T_j^O(c) - T_j^M(c)\right)^2$$

$$c^{i+1} = c^i - \nabla_c J(c) = c^i - \alpha\frac{\partial J}{\partial c}$$

$$T^n = \begin{bmatrix} T_{0,0}^n \\ T_{1,0}^n \\ T_{2,0}^n \\ \vdots \\ T_{n_x-1,0}^n \\ T_{0,1}^n \\ \vdots \\ T_{i,j-1}^n \\ \vdots \\ T_{i-1,j}^n \\ T_{i,j}^n \\ T_{i+1,j}^n \\ \vdots \\ T_{i,j+1}^n \\ \vdots \\ T_{n_x-1,n_y-1}^n \end{bmatrix} \quad T^{n+1} = \begin{bmatrix} T_{0,0}^{n+1} \\ T_{1,0}^{n+1} \\ T_{2,0}^{n+1} \\ \vdots \\ T_{n_x-1,0}^{n+1} \\ T_{0,1}^{n+1} \\ \vdots \\ T_{i,j-1}^{n+1} \\ \vdots \\ T_{i-1,j}^{n+1} \\ T_{i,j}^{n+1} \\ T_{i+1,j}^{n+1} \\ \vdots \\ T_{i,j+1}^{n+1} \\ \vdots \\ T_{n_x-1,n_y-1}^{n+1} \end{bmatrix} \quad Q = \begin{bmatrix} Q_{0,0} \\ Q_{1,0} \\ Q_{2,0} \\ \vdots \\ Q_{n_x-1,0} \\ Q_{0,1} \\ \vdots \\ Q_{i,j-1} \\ \vdots \\ Q_{i-1,j} \\ Q_{i,j} \\ Q_{i+1,j} \\ \vdots \\ Q_{i,j+1} \\ \vdots \\ Q_{n_x-1,n_y-1} \end{bmatrix}$$

Parameter Estimation Of 2D Heat Distribution In Heterogeneous Media

SiSc Laboratory
Project group 2

Muhammad Sajid Ali
Shubhaditya Burela
Aneesh Futane
Pourya Pilva

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

IT Center
RWTH AACHEN UNIVERSITY

Overview

Introduction

Introduction: Heat Distribution

2D Heat Equation with heat source :

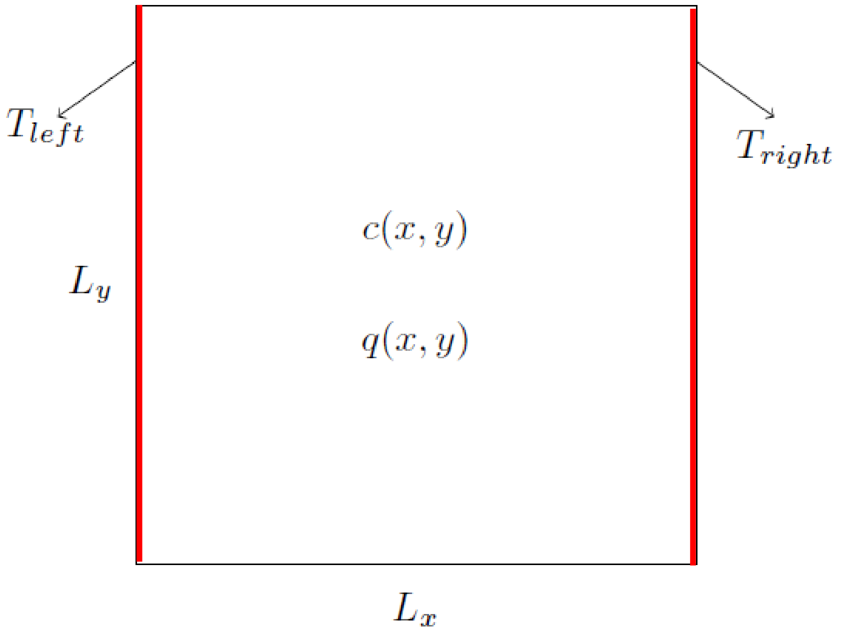
$$\frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(c(x,y)\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(c(x,y)\frac{\partial T}{\partial y}\right) + q(x,y)$$

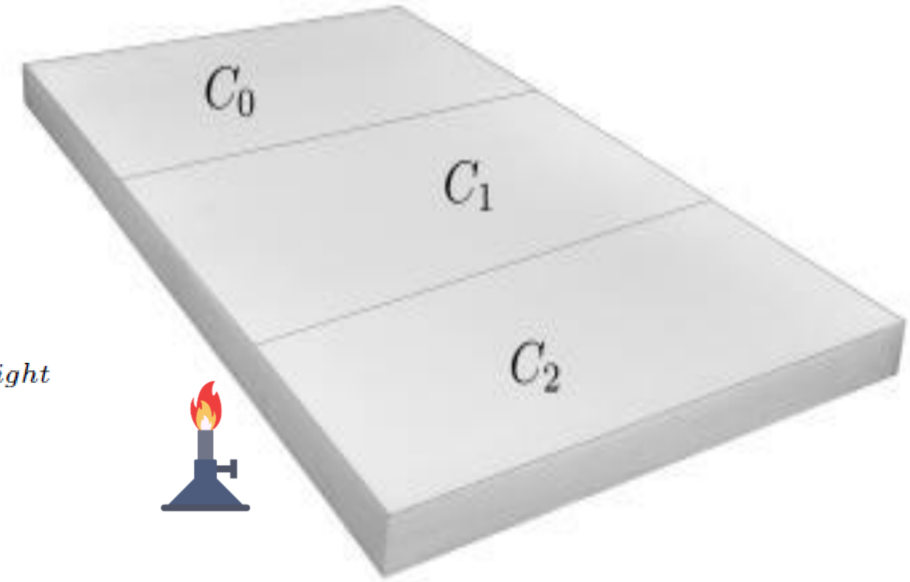
where $T = T\,(x, y, t, c)$

x : Space
y : Space
t : Time
c : Heat diffusivity
T : Temperature
q : Heat flux

Boundary Conditions:

$T\,(x, y, t = 0) = T_{\text{init}}$
$T\,(x = 0, y, t) = T_{\text{left}}$
$T\,(x = L_x, y, t) = T_{\text{right}}$





Introduction: Parameter Estimation

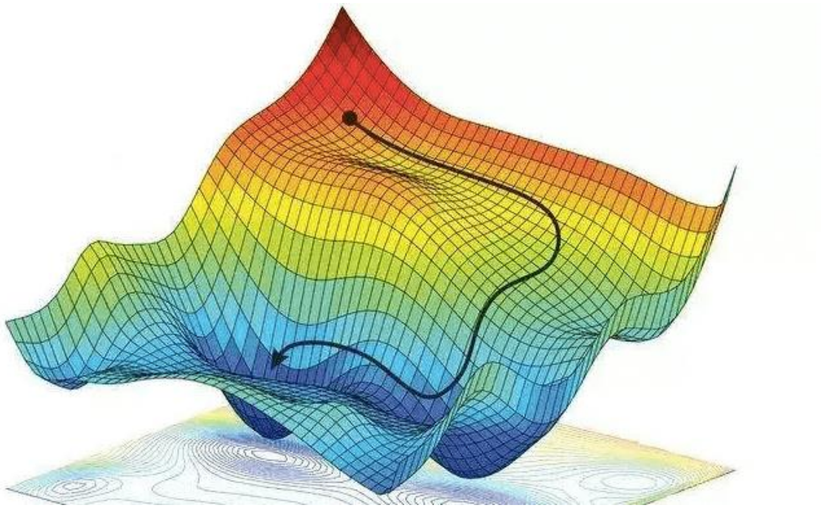
Minimizing the objective function:

$$\boxed{J(c) = \frac{1}{n}\sum_{j=0}^{n-1}\left(T_j^O(c) - T_j^M(c)\right)^2}$$

.......with the help of gradient descent

$$c^{i+1} = c^i - \nabla_C J(c) = c^i - \alpha\frac{\partial J}{\partial c}$$

- $c(x, y)$ is the optimal parameter
- $J(c)$ is the objective function
- T_j^O is the observed temperature
- T_j^M is the measured temperature



Source: easyai.tech

FDM

Discretization in Space

- Central difference scheme

$$\Delta x = \frac{L_x}{n_x - 1}\ ,\quad \Delta y = \frac{L_y}{n_y - 1}$$

$$\frac{\partial}{\partial x}\left(c(x,y)\frac{\partial T_{i,j}}{\partial x}\right) \approx \frac{1}{\Delta x}\left\{\left(\frac{C_{i+1,j}+C_{i,j}}{2}\right)\left(\frac{T_{i+1,j}-T_{i,j}}{\Delta x}\right) - \left(\frac{C_{i-1,j}+C_{i,j}}{2}\right)\left(\frac{T_{i,j}-T_{i-1,j}}{\Delta x}\right)\right\} + O(\Delta x)^2$$

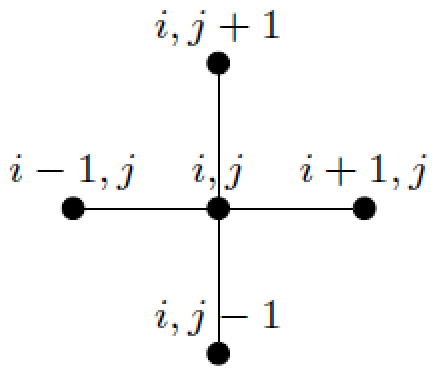
$$\frac{\partial}{\partial y}\left(c(x,y)\frac{\partial T_{i,j}}{\partial y}\right) \approx \frac{1}{\Delta y}\left\{\left(\frac{C_{i,j+1}+C_{i,j}}{2}\right)\left(\frac{T_{i,j+1}-T_{i,j}}{\Delta y}\right) - \left(\frac{C_{i,j-1}+C_{i,j}}{2}\right)\left(\frac{T_{i,j}-T_{i,j-1}}{\Delta y}\right)\right\} + O(\Delta y)^2$$

$$\mathrm{i} = 0,\ldots\ldots,n_x - 1$$

$$\mathrm{j} = 0,\ldots\ldots,n_y - 1$$

- Combining

$$\frac{\partial}{\partial x}\left(c(x,y)\frac{\partial T_{i,j}}{\partial x}\right) + \frac{\partial}{\partial y}\left(c(x,y)\frac{\partial T_{i,j}}{\partial y}\right) = \frac{1}{\Delta y^2}\left(\frac{C_{i,j}+C_{i,j-1}}{2}\right)T_{i,j-1} +$$
$$\frac{1}{\Delta x^2}\left(\frac{C_{i-1,j}+C_{i,j}}{2}\right)T_{i-1,j} - \left\{\frac{1}{\Delta x^2}\left(\frac{C_{i-1,j}+2C_{i,j}+C_{i+1,j}}{2}\right) + \frac{1}{\Delta y^2}\left(\frac{C_{i,j-1}+2C_{i,j}+C_{i,j+1}}{2}\right)\right\}T_{i,j} +$$
$$\frac{1}{\Delta x^2}\left(\frac{C_{i+1,j}+C_{i,j}}{2}\right)T_{i+1,j} + \frac{1}{\Delta y^2}\left(\frac{C_{i,j+1}+C_{i,j}}{2}\right)T_{i,j+1}$$



Discretization in Time

- Implicit Euler

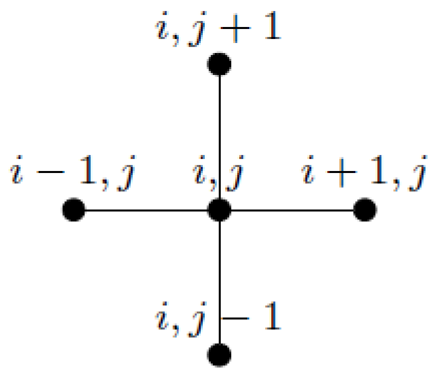
$\Delta t = \frac{t_f}{m}$ where t_f is the final time and m is the no. of steps

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

Combining

- On combining..

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



- On rearranging

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

The linear system

- In matrix form :

$$\boxed{(I - \Delta t.R)T^{n+1} = T^n + \Delta t.Q}$$

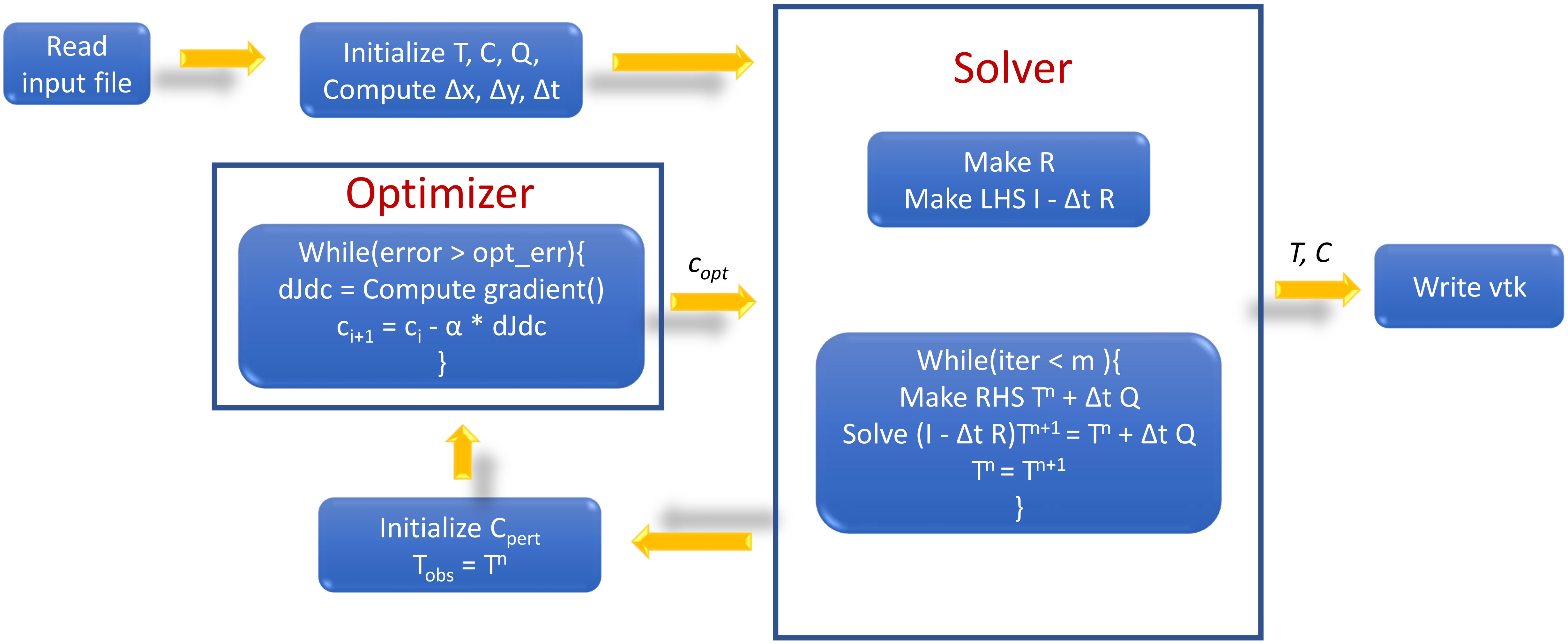
$$T^n = \begin{bmatrix} T^n_{0,0} \\ T^n_{1,0} \\ T^n_{2,0} \\ \vdots \\ T^n_{n_x-1,0} \\ T^n_{0,1} \\ \vdots \\ T^n_{i,j-1} \\ \vdots \\ T^n_{i-1,j} \\ T^n_{i,j} \\ T^n_{i+1,j} \\ \vdots \\ T^n_{i,j+1} \\ \vdots \\ T^n_{n_x-1,n_y-1} \end{bmatrix} \quad T^{n+1} = \begin{bmatrix} T^{n+1}_{0,0} \\ T^{n+1}_{1,0} \\ T^{n+1}_{2,0} \\ \vdots \\ T^{n+1}_{n_x-1,0} \\ T^{n+1}_{0,1} \\ \vdots \\ T^{n+1}_{i,j-1} \\ \vdots \\ T^{n+1}_{i-1,j} \\ T^{n+1}_{i,j} \\ T^{n+1}_{i+1,j} \\ \vdots \\ T^{n+1}_{i,j+1} \\ \vdots \\ T^{n+1}_{n_x-1,n_y-1} \end{bmatrix} \quad Q = \begin{bmatrix} Q_{0,0} \\ Q_{1,0} \\ Q_{2,0} \\ \vdots \\ Q_{n_x-1,0} \\ Q_{0,1} \\ \vdots \\ Q_{i,j-1} \\ \vdots \\ Q_{i-1,j} \\ Q_{i,j} \\ Q_{i+1,j} \\ \vdots \\ Q_{i,j+1} \\ \vdots \\ Q_{n_x-1,n_y-1} \end{bmatrix}$$

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

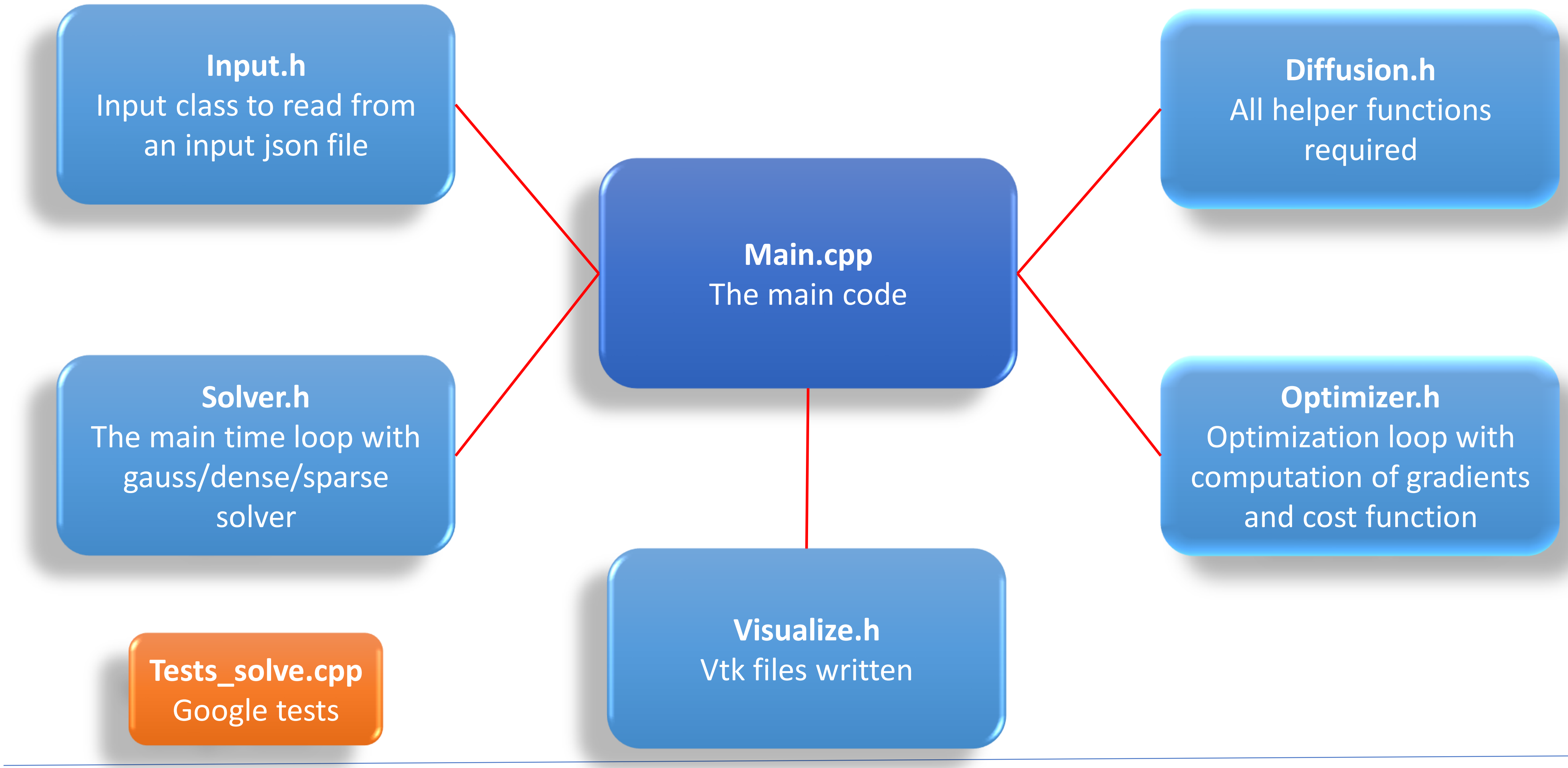
where $S^x_{i-1,j}$ is defined as $\frac{1}{\Delta x^2}\left(\frac{C_{i-1,j}+C_{i,j}}{2}\right)$. Similarly $S^x_{i+1,j} = \frac{1}{\Delta x^2}\left(\frac{C_{i+1,j}+C_{i,j}}{2}\right)$, $S^y_{i,j-1} = \frac{1}{\Delta y^2}\left(\frac{C_{i,j-1}+C_{i,j}}{2}\right)$ and $S^y_{i,j+1} = \frac{1}{\Delta y^2}\left(\frac{C_{i,j+1}+C_{i,j}}{2}\right)$

Implementation

The code: Flowchart



The code: Structure



Input

- Json file to read input
- *Rapidjson* library used

Name - The name of the test case
Lx - Length in x-direction
Ly - Length in y-direction
nx - Number of points in x-direction
ny - Number of points in y-direction
tf - Final time
m - Number of time steps
T_init - Initial temperature
T_left - Left wall boundary temperature
T_right - Right wall boundary temperature
nc - Number of different heat diffusivity cases
c1 - Heat diffusivity 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

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
}
```

Linear solvers

- Gauss elimination

$$Ax = b \Rightarrow (A|b) = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} & b_1 \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} & b_n \end{pmatrix} \Rightarrow \begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} & b_1 \\ 0 & u_{2,2} & \cdots & u_{2,n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & u_{m,n} & b_n \end{pmatrix} \Rightarrow x_{n-1} + a_{(n-1,n)}x_n = b_{n-1}$$

- Eigen/Dense

$$Ax = b \Rightarrow \begin{aligned} L\mathbf{y} &= P\mathbf{b} \\ U\mathbf{x} &= \mathbf{y} \end{aligned}$$

P - permutation matrix
L - Lower triangular matrix
U - upper triangular matrix

- Eigen/Sparse
  - LU decomposition (same as Eigen/Dense)
  - Sparse column major storage of matrix A

Optimizer

Adjoint mode

```
typedef active DCO_BASE_TYPE;
typedef ga1s<DCO_BASE_TYPE> DCO_MODE;
typedef DCO_MODE::type DCO_TYPE;
typedef DCO_MODE::tape_t DCO_TAPE_TYPE;
DCO_MODE::global_tape=DCO_TAPE_TYPE::create();
std::vector<DCO_TYPE> CC(nx*ny), TT_obs(nx*ny);
typename dco::ga1s<active>::type J_adjoint;
std::vector<DCO_TYPE> Tnew(my_input.nx * my_input.ny);

// Initialization of the variables
for(int i = 0; i < my_input.nx * my_input.ny; i++){
    CC[i] = C[i];
    TT_obs[i] = T_obs[i];
    dJdc[i] = 0.0;
    DCO_MODE::global_tape->register_variable(CC[i]);
}

// Solve for Tnew
Tnew = solve(T,CC,Q,my_input);

// Seeding
DCO_MODE::global_tape->register_output_variable(J_adjoint);
J_adjoint = cost_function(TT_obs, Tnew, my_input.nx, my_input.ny);
derivative(J_adjoint) = 1.0;

// Harvest the derivatives
DCO_MODE::global_tape->interpret_adjoint();

// Store the derivatives
for(int i = 0; i < my_input.nx * my_input.ny; i++)
    dJdc[i] = derivative(CC[i]);
DCO_TAPE_TYPE::remove(DCO_MODE::global_tape);
```

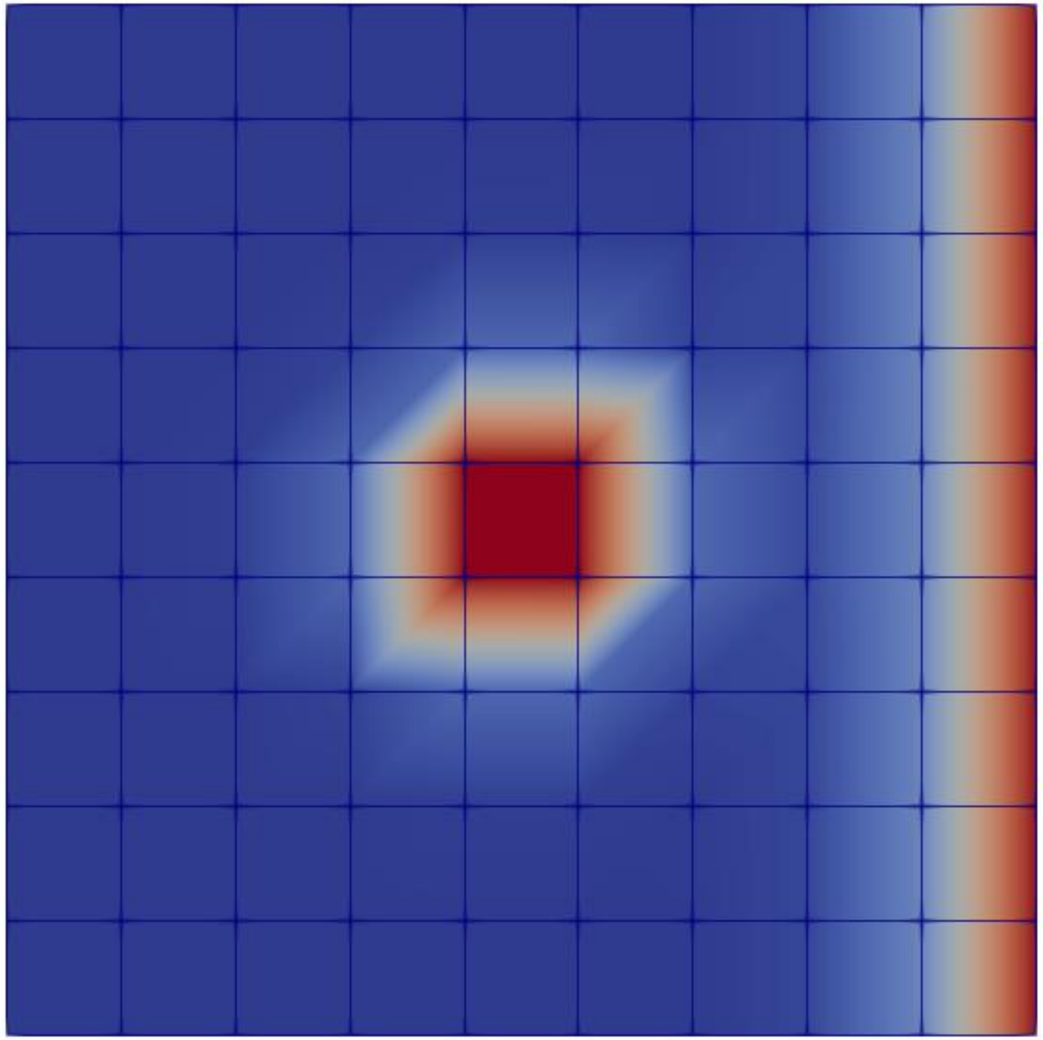
Tangent mode

```
typedef active DCO_BASE_TYPE;
typedef gt1s<DCO_BASE_TYPE> DCO_MODE;
typedef DCO_MODE::type DCO_TYPE;
std::vector<DCO_TYPE> CC(nx*ny), TT_obs(nx*ny);
typename dco::gt1s<active>::type J;
std::vector<DCO_TYPE> Tnew(my_input.nx * my_input.ny);

// Initialization of the variables
for(int i = 0; i < my_input.nx * my_input.ny; i++){
    CC[i] = C[i];
    TT_obs[i] = T_obs[i];
    dJdc[i] = 0.0;
}

for(int i = 0; i<my_input.nx*my_input.ny;i++){
    derivative(CC[i]) = 1.0;
    Tnew = solve(T,CC,Q,my_input);
    J = cost_function(TT_obs, Tnew, my_input.nx, my_input.ny);
    dJdc[i] = derivative(J);
    derivative(CC[i]) = 0.0;
}
```

Visualize

- Vtk files written
  - Rectilinear grid
  - Field data 2
    - Temperature
    - Heat diffusivity

- Visualized with Paraview

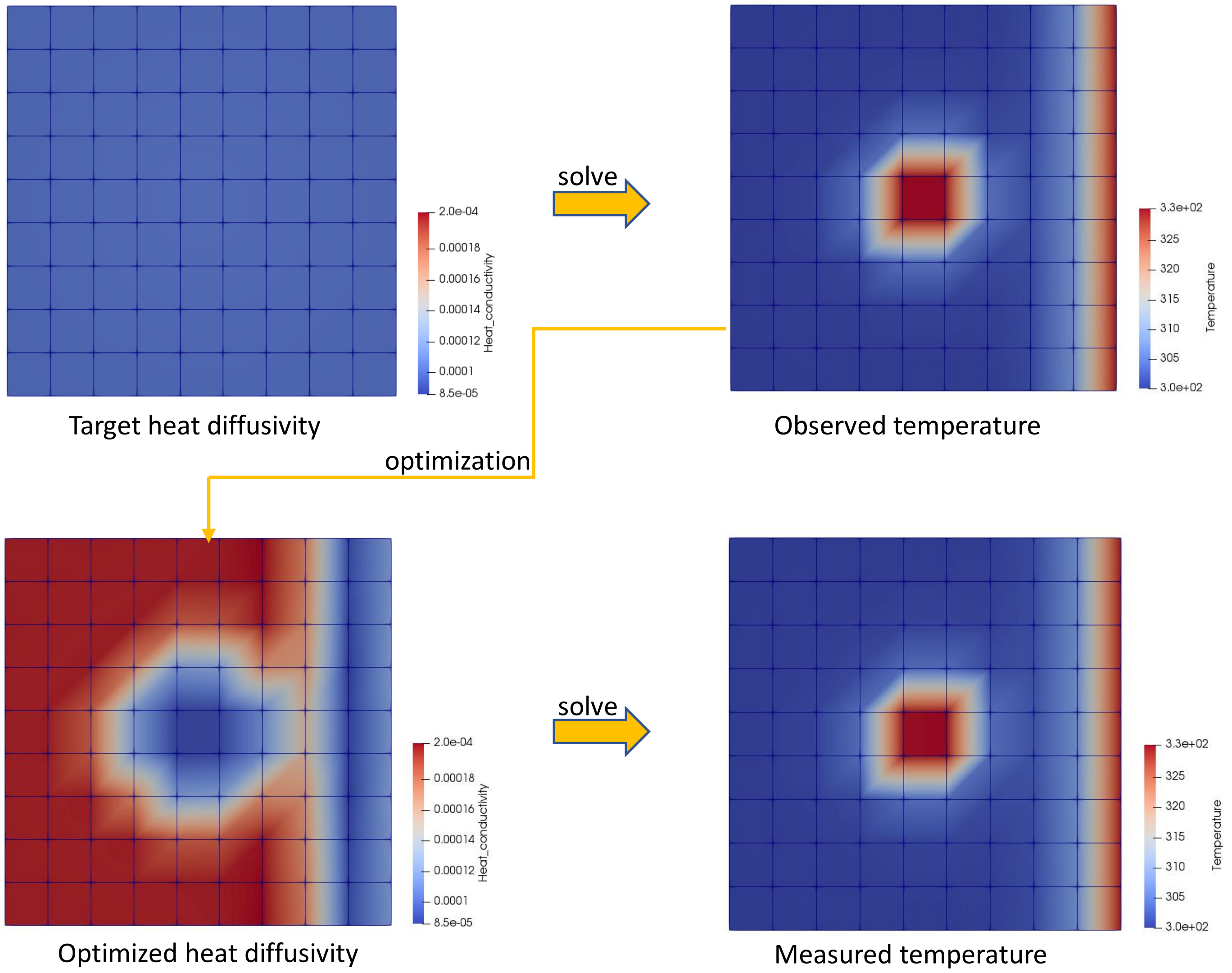


Results

Test case 1

Case_1.json

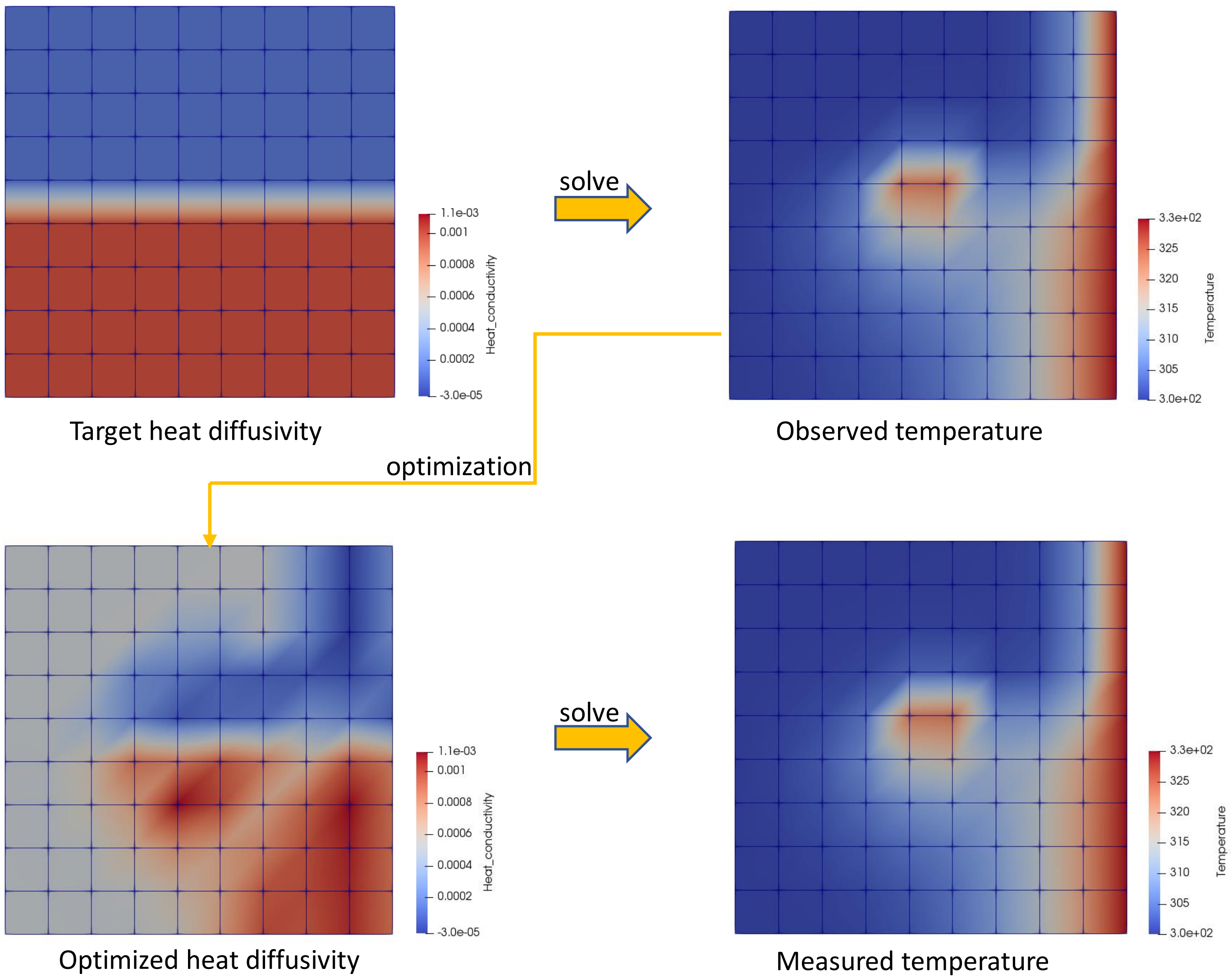
```
{
    "name": "Case 1: Single diffusivity",
    "Lx": 1.0,
    "Ly": 1.0,
    "nx": 10,
    "ny": 10,
    "tf": 40,
    "m": 400,
    "T_init": 300,
    "T_left": 300,
    "T_right": 330,
    "q": [1.0,0.4,0.6,0.4,0.6],
    "nc": 1,
    "c1": [0.0001,0.0,1.0,0.0,1.0],
    "c_init": 0.0002,
    "alpha":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



Test case 2

Case_2.json

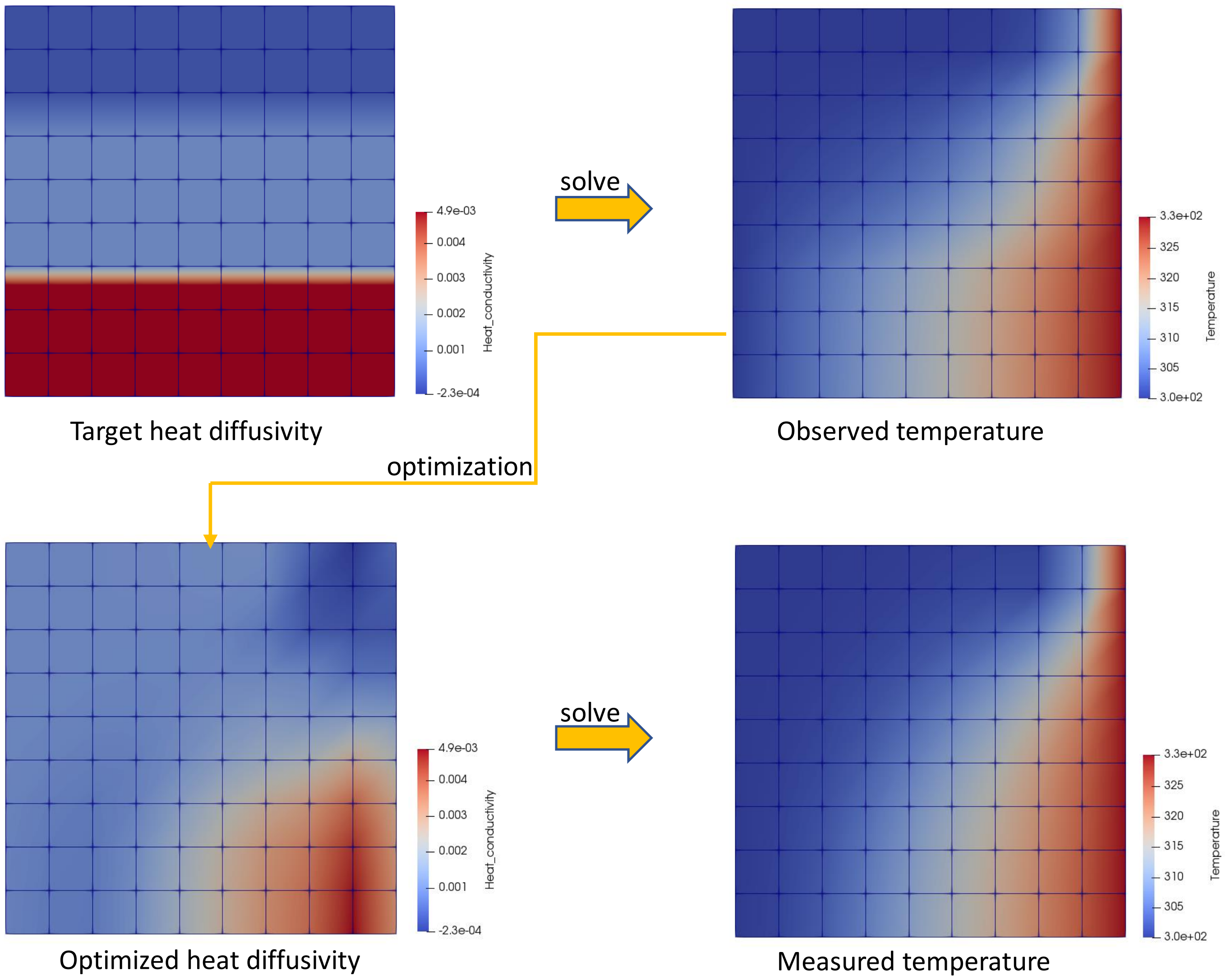
```
{
    "name": "Case 2: Dual diffusivity",
    "Lx": 1.0,
    "Ly": 1.0,
    "nx": 10,
    "ny": 10,
    "tf": 40,
    "m": 400,
    "T_init": 300,
    "T_left": 300,
    "T_right": 330,
    "q": [1.0,0.4,0.6,0.4,0.6],
    "nc": 2,
    "c1": [0.001,0.0,1.0,0.0,0.5],
    "c2": [0.0001,0.0,1.0,0.5,1.0],
    "c_init": 0.0005,
    "alpha": 1e-8,
    "opt_steps": 1000,
    "opt_err": 1e-8
}
```



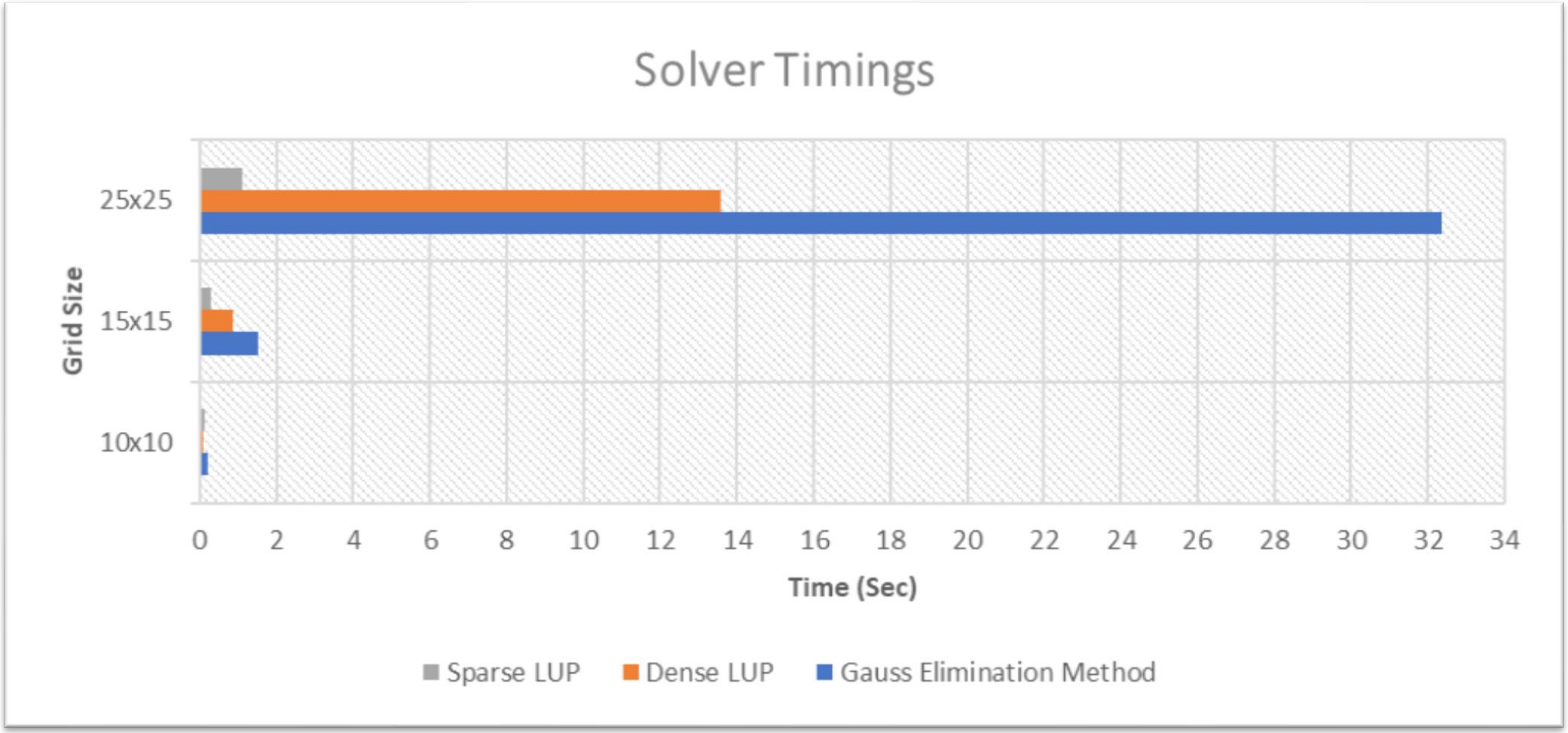
Test case 3

Case_3.json

```
{
    "name": "Case 3: Triple 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": [1.0,0.45,0.55,0.45,0.55],
    "nc": 3,
    "c1": [0.01,0.0,1.0,0.0,0.33],
    "c2": [0.001,0.0,1.0,0.33,0.67],
    "c3": [0.0001,0.0,1.0,0.67,1.0],
    "c_init": 0.001,
    "alpha": 1e-8,
    "opt_steps": 1000,
    "opt_err": 1e-8
}
```

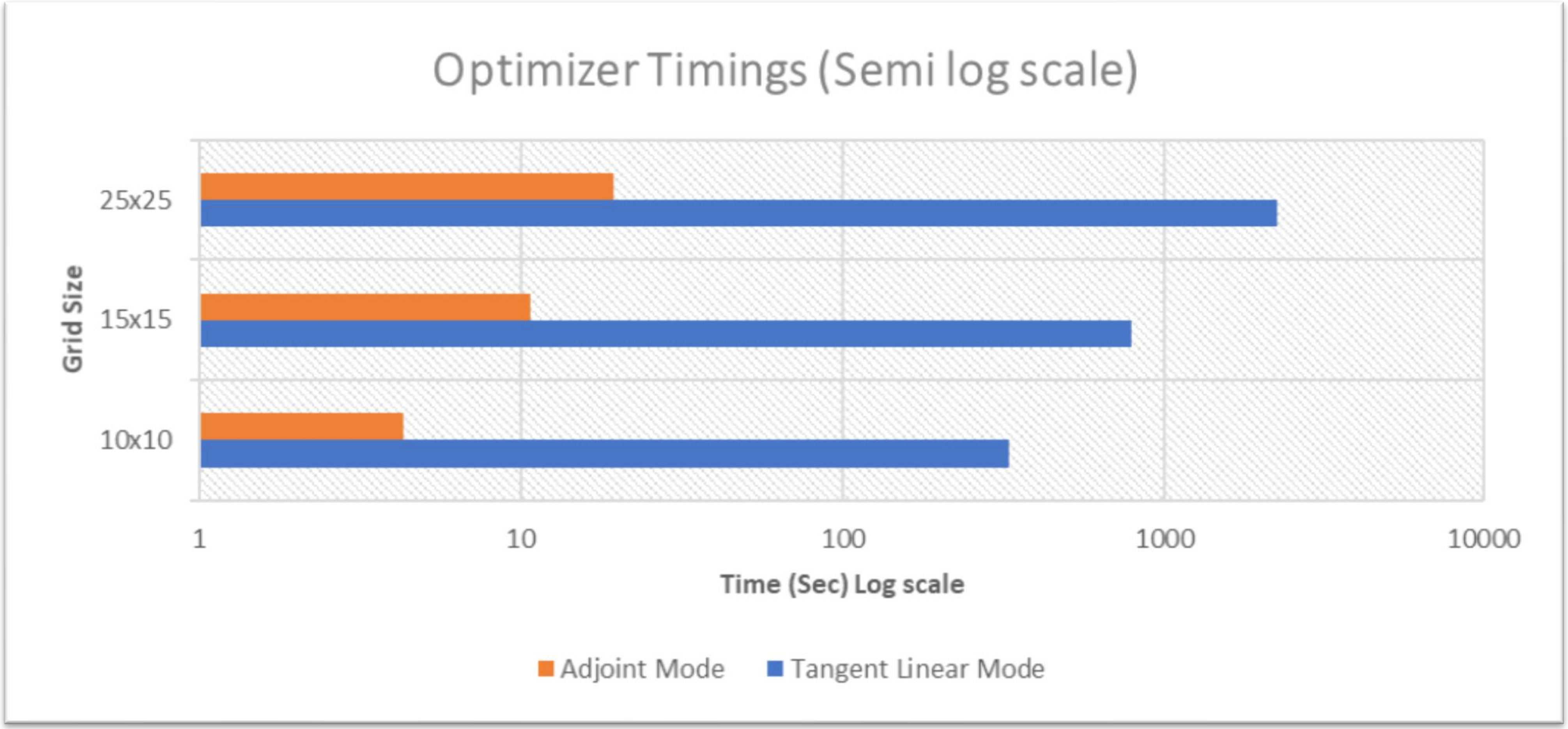


Solver Timings



Note: Timings for 1 solve

Tangent vs Adjoint Timings



Note: Timings for 20 optimization steps

Project Management

Project management

- Code written in C++
- RWTH GitLab used

Member	Main Responsibility
Muhammad Sajid Ali	Main code, input parser, solvers, code integration
Shubhaditya Burela	Optimizer (dco), cmake, RWTH cluster integration, running tests on cluster
Aneesh Futane	Google tests, presentation
Pourya Pilva	Visualization (vtk), report

Conclusion

Conclusion

- The problem successfully solved with FDM and parameters successfully optimized by Gradient Descent method
- Sparse solvers the fastest
- Adjoint mode better than Tangent mode for our problem
- All requirements met: config file (json), cmake, google testing, vtk, GitLab
- Parallelize?
  - Max time taken by linear solver and dco::interpreter()
  - Both are external libraries and hence hard to parallelize them

Scope for further work

  - Non-gradient optimization methods
  - Training Neural Networks using the inputs and/or gradients to predict temperature

Thank you ☺
Any questions???