1a)

Write the system

$$-f_{i,j} = -\frac{\sigma_{i+1/2,j}}{h^2} u_{i+1,j} - \frac{\sigma_{i-1/2,j}}{h^2} u_{i-1,j} - \frac{\sigma_{i,j+1/2}}{h^2} u_{i,j+1} - \frac{\sigma_{i,j-1/2}}{h^2} u_{i,j-1} + \frac{\sigma_{i+1/2,j}+\sigma_{i-1/2,j} + \sigma_{i,j+1/2}+\sigma_{i,j-1/2}}{h^2} u_{i,j}$$

$$A = \frac{1}{h^2}\begin{bmatrix} B_1 & C_{1+1/2} & & \\ C_{1+1/2} & B_2 & C_{2+1/2} & \\ & \ddots & \ddots & \ddots \\ & C_{(N-1)-1/2} & & C_{(N-1)+1/2} \\ & & C_{N-1/2} & B_N \end{bmatrix} \qquad B_j = \begin{bmatrix} -K(1,j) & \sigma_{1+1/2,j} & & \\ \sigma_{2-1/2,j} & -K(2,j) & & \\ & & \ddots & \sigma_{(N-1)+1/2,j} \\ & & \sigma_{N-1/2,j} & -K(N,j) \end{bmatrix}$$

$$C_k = \begin{bmatrix} \sigma_{1,k} & & \\ & \ddots & \\ & & \sigma_{N,k} \end{bmatrix}$$

1b)

(Core problem) In the template file `finite_difference.cpp`, implement the function

```
void createPorousMediaMatrix2D(SparseMatrix& A, FunctionPointer sigma, int N, double dx),
```

Set matrix a from one vector containing the matrices B and C

double K → sum of σ

Special conditions to .push_back in the right places around the diagonal

1c)

(Core problem) In the template file `finite_difference.cpp`, implement the function

```
void createRHS(Vector& rhs, FunctionPointer f, int N, double dx),
```

$F = dx^2 f(x,y)$ → Serie 3

1d)

$\frac{\ldots_{i-1/2,j} + \sigma_{i,j+1/2}+\sigma_{i,j-1/2}}{h^2} u_{i,j}$

(Core problem) In the template file `finite_difference.cpp`, implement the function

```
Vector porousMediaSolve(FunctionPointer f, FunctionPointer f, int N),
```

Normal LU solve like in the series

Sparse matrix requires Eigen/SparseLU

1f)

(Core problem) In the template file `finite_difference.cpp`, implement the function

```
void convergeStudy(FunctionPointer F, FunctionPointer sigma),
```

$e = \text{abs}(u - \text{exactS})$

2a)

Write the variational formulation for (4)-(5).

Write an expression for the entries of **A** and **F** in (7).

$A\mu = F$

$A_{ij} = \int_\Omega (\alpha(\underline{x}) \nabla\varphi_j^N(\underline{x}) \nabla\varphi_i^N(\underline{x}) + \gamma\varphi_i^N(\underline{x})\varphi_j^N(\underline{x}))\, d\underline{x}$

$F_i = \int_\Omega f(\underline{x})\varphi_i^N(\underline{x})\, dx$

2c)

(Core problem) Complete the template file `shape.hpp` implementing the function

```
inline double lambda(int i, double x, double y)
```

Implemented the 3 conditions given

2d)

(Core problem) Complete the template file `grad_shape.hpp` implementing the function

```
inline Eigen::Vector2d gradientLambda(const int i, double x, double y)
```

Switch to decide the shape functions starting from i

2e)

(Core problem) Complete the template file `stiffness_matrix.hpp` implementing the routine

```
template<class MatrixType, class Point>
void computeStiffnessMatrix(MatrixType& stiffnessMatrix,
```

Given: J_n, $\det J_n$, $(J_n^{-1})^T$, function

coeffT, vol F, elem Map

Fix every cell of the matrix by the

integration:

$A_{ij} = \int (\alpha(\underline{x}) \nabla\varphi_j^N(\underline{x}) \nabla\varphi_i^N(\underline{x}) + \gamma\varphi_i^N(\underline{x})\varphi_j^N(\underline{x}))$

2f)

(Core problem) Complete the template file `load_vector.hpp` implementing the routine

```
template<class Vector, class Point>
void computeLoadVector(Vector& loadVector, const Point& a, const Point& b,
```

Given: J_n, $\det J_n = dV$, function to integrate

Same as matrix but here we integrate

$F_i = \int_\Omega f(\underline{x})\varphi_i^N(\underline{x})\, dx$ to get the load vector

2g)

(Core problem) Complete the template file `stiffness_matrix_assembly.hpp` implementing the routine

```
template<class Matrix>
void assembleStiffnessMatrix(Matrix& A, const Eigen::MatrixXd& vertices,
```

→ Get indices of vertices

→ Build the 3x3 matrix

→ Push vertices and matrix as a triplet

Build the matrix

2h)

(Core problem) Complete the template file `load_vector_assembly.hpp` implementing the routine

```
void assembleLoadVector(Eigen::VectorXd& F, const Eigen::MatrixXd& vertices,
```

→ get triangle vertices

→ get load vector for that triangle

→ push everything in the bigger load vector

2i)

(**Core problem**) Complete the template file `fem_solve.hpp` with the implementation of the function

```
int solveFiniteElement(Vector& u, const Eigen::MatrixXd& vertices,
```

- → Build A and F
- → Set Boundaries
- → Solve "interior" system, where there are no boundaries
- → Combine boundary with internal solution

3a)

Denote by h the mesh width, that is $h = \frac{1}{N+1}$. Write down the matrix $\mathbf{A}$ and the vector $\boldsymbol{G}(t)$ explicitly.

$$\frac{1}{h^2}\begin{bmatrix} 2a(x_1) & -a(x_1) & \cdots & & \\ -a(x_2) & 2a(x_2) & -a(x_2) & & \\ & \ddots & \ddots & \ddots & \\ & & & & -a(x_{N-1}) \\ & & & -a(x_N) & 2a(x_N) \end{bmatrix} = A \qquad \frac{1}{h^2}\begin{bmatrix} a(h)g_L(t) \\ 0 \\ \vdots \\ 0 \\ a(1-h)g_R(t) \end{bmatrix} = G(t)$$

3b)

Apply the *forward Euler* scheme to (11), denoting by $\boldsymbol{u}^k = \{u_i^k\}_{i=1}^N$ the approximate value of the vector $\boldsymbol{u}$ at time k, for $k = 0, \ldots, K$, and by $\Delta t = \frac{T}{K}$ the time step. How does the update formula at each time step look like?

$$\frac{\underline{u}^{k+1} - \underline{u}^k}{\Delta t} + \underline{\underline{A}}\,\underline{u}^k = \underline{G}(t_k)$$

3c)

(**Core problem**) In the template file create_poisson_matrix.cpp, implement the function

```
SparseMatrix createPoissonMatrix(int N, const std::functional<double(double)>& a),
```

- → prepare matrices at the right side
- → Assign value of $a(x)$ on the "diagonals"
  - ↳ push right / left
- → Build A
- → Divide by h^2

3d)

(**Core problem**) In the template file forward_euler.cpp, implement the function

```
std::pair<Eigen::MatrixXd, Eigen::VectorXd> forwardEuler(
```

$$u^{k+1} = (I - \underline{\underline{A}}\Delta t)\,\underline{u}^k + \Delta t\,\underline{G}(t_k)$$

u.col(k+1) B u.col(k) G

$$\frac{1}{h^2}\begin{bmatrix} a(h)g_L(t) \\ 0 \\ \vdots \\ 0 \\ a(1-h)g_R(t) \end{bmatrix} = G(t) = G$$

3i)

(Core problem) In the template file crank_nicolson.cpp, implement the function

```
std::pair<Eigen::MatrixXd, Eigen::VectorXd> crankNicolson(
```

$$\left(I + \frac{\lambda}{2} A\right) u^{n+1} = A \cdot u^{n+1} = B u^{n} = \left(I - \frac{\lambda}{2} A\right) u^{n}$$