source/slae.hpp

```cpp
#pragma once

#include "utils.hpp"
#include <cassert>



// Backwards gaussian elimination. O(N^2) complexity.
//
// Assumes 'R' to be upper-triangular matrix.
//
inline Vector backwards_gaussian_elimination(const Matrix& R, Vector rhs) {
    assert(R.rows() == R.cols());
    assert(R.rows() == rhs.rows());

    for (Idx i = R.rows() - 1; i >= 0; --i) {
        for (Idx j = i + 1; j < R.cols(); ++j) rhs(i) -= R(i, j) * rhs(j);
        rhs(i) /= R(i, i);
    }

    return rhs;
}


// Forward gaussian elimination. O(N^3) complexity.
//
// Partial pivoting.
//
inline void partial_piv_forward_gaussian_elimination(Matrix& A, Vector& rhs) {
    assert(A.rows() == A.cols());
    assert(A.rows() == rhs.rows());

    for (Idx i = 0; i < A.rows(); ++i) {
        // Partial pivot
        Idx i_max = i;
        for (Idx ii = i; ii < A.rows(); ++ii)
            if (std::abs(A(ii, i)) > std::abs(A(i_max, i))) i_max = ii;

        if (i != i_max) {
            // Swap rows (matrix)
            const Vector tmp_A   = A.row(i);
            A.row(i)             = A.row(i_max);
            A.row(i_max)         = tmp_A;
            // Swap rows (rhs)
            const double tmp_rhs = rhs(i);
            rhs(i)               = rhs(i_max);
            rhs(i_max)           = tmp_rhs;
        }

        // Elimination (normalize current row)
        const double factor = 1. / A(i, i);
        for (Idx j = i; j < A.cols(); ++j) A(i, j) *= factor;
        rhs(i) *= factor;

        // Elimination (substract current row from all the rows below)
        for (Idx k = i + 1; k < A.rows(); ++k) {
```

```
57              const double first = A(k, i);
58              for (Idx j = i; j < A.cols(); ++j) A(k, j) -= first * A(i, j);
59              rhs(k) -= first * rhs(i);
60          }
61      }
62  }
63
64  // Forward + backwards gaussian elimination. O(N^3) complexity.
65  //
66  // Partial pivoting. Jacobi preconditioner.
67  //
68  inline Vector partial_piv_gaussian_elimination(Matrix A, Vector rhs) {
69      // Jacobi preconditioner
70      //
71      // When testing N = 4 one of the (A - lambda I) matrices with epsilon = 0.1
    was almost degenerate (det = 1e-7),
72      // without preconditioning it went into 'nan's, Jacobi was just good enough to
    prevent this.
73      //
74      // N = 10 case is luckier and doesn't require this, but why not do it
    regardless.
75      //
76      Matrix preconditioner     = Matrix::Zero(A.rows(), A.cols());
77      preconditioner.diagonal() = A.diagonal();
78      A                         = preconditioner * A;
79      rhs                       = preconditioner * rhs;
80
81      // Gaussian elimination
82      partial_piv_forward_gaussian_elimination(A, rhs);
83      return backwards_gaussian_elimination(A, rhs);
84  }
```