source/eigenvalues.hpp

```cpp
#pragma once

#include "qr_factorization.hpp"
#include "thirdparty/Eigen/src/Core/util/Constants.h"
#include "utils.hpp"
#include <cassert>
#include <cstddef>
#include <cstdio>
#include <limits>

#include "thirdparty/Eigen/Core"

// QR-method for eigenvalues with NO shift and NO Hessenberg form optimization.
//
// Used as a reference. O(N^3) single iteration complexity.
//
Matrix eigenvalues_prototype(const Matrix& A) {
    assert(A.rows() == A.cols());

    Matrix T_shur = A;

    for (Idx i = 0; i < A.rows() * 100; ++i) {
        const auto [Q, R] = qr_factorize(T_shur); // O(N^3)
        T_shur            = R * Q;                // O(N^3)
        // no stop condition, just do a ton of iterations
    }

    return T_shur;
}

// QR-method for eigenvalues with shifts and Hessenberg form optimization.
//
// Requires 'A' to be in upper-Hessenber form (!).
// Using Hessenberg form brings complexity down to O(N^2) per iteration.
//
// Algorithm:
// ---------------------------------------------------------------------------
// -   while (N >= 2 && iteration++ < max_iterations) {                      -
// -      sigma = T_shur[N, N]                                  // O(1)     -
// -      [ Q, R, RQ ] = qr_factorize_hessenberg(T_shur[1:N, 1:N]) // O(N^2)  -
// -      T_shur[0:N, 0:N] = RQ + sigma I                       // O(N^2)  -
// -      if (|T_shur[N, N-1]| < eps) --N                       // O(1)     -
// -   }                                                                    -
// ---------------------------------------------------------------------------
//
// Note that matrix multiplication here is O(N^2) because 'R' is tridiagonal.
//
// As a stop-condition for deflating the block we use last row element under the diagonal,
// as soon as it becomes "small enough" the block can deflate.
//
// 'Q' and 'R' matrices aren't directly used anywhere, but still computed for debugging purposes.
//
Matrix eigenvalues(const Matrix& A) {
    assert(A.rows() == A.cols());

```

```
56      const std::size_t max_iterations = 500 * A.rows();
57      std::size_t       iteration      = 0;
58      Idx               N              = A.rows(); // mutable here since we shrink
    the working block (!)
59
60      Matrix T_schur = A;
61
62      while (N >= 2 && iteration++ < max_iterations) {
63          const double sigma = T_schur(N - 1, N - 1); // O(1)
64          [[maybe_unused]] const auto [Q, R, RQ] =
65              qr_factorize_hessenberg(T_schur.block(0, 0, N, N) - sigma *
    Matrix::Identity(N, N)); // (N^2)
66          T_schur.block(0, 0, N, N) = RQ + sigma * Matrix::Identity(N, N);
    // (N^2)
67          if (std::abs(T_schur(N - 1, N - 2)) < std::numeric_limits<double>
    ::epsilon()) --N; // O(1)
68      }
69
70      return T_schur;
71  }
72
```