ETH Lectures 401-0663-00L Numerical Methods for Computer Science
401-2673-00L Numerical Methods for CSE

Homework Problems and Projects

Prof. R. Hiptmair, SAM, ETH Zurich

Autumn Term 2021, Version of October 2, 2022
(C) Seminar für Angewandte Mathematik, ETH Zürich

Link to the current version of this homework collection

0.1 General Information

0.1.1 Weekly Homework Assignments

All problems will be published in this single ".pdf" file. Every week, we publish a list of problem numbers on the lecture Moodle page. These are the current assignments for that week. Details are given on the lecture Moodle page.

0.1.2 Importance of Homework

Homework assignments are **not** mandatory. However, it is **very important** that you constantly exercise with the material you learn. "Solving" the homework assignments one week before the main exam by looking at the solutions will likely result in failure.

We provide hints and solutions from the beginning. It is your responsibility to look at the solutions only if you are stuck with a difficult problem and you tried to find a solution for a sufficient amount of time.

Most homework problems come with an estimate for the time required for its solution. These times are what we expect a bright and well-prepared student to require for that problem during an exam.

0.1.3 Corrections and Grading of Assignments

Homework assignments will not be graded.

However, you may submit your handwritten solutions by handing them to your tutor directly or uploading them via the Moodle upload interface, see the lecture Moodle page for instructions. The assistants will

Problem 3-16: Compact Storage Format for QR-Factorization

It is essential for the efficiency of orthogonalization techniques that orthogonal matrices, which commonly occur as factors in matrix products. are stored as products of elementary orthogonal transformations like Householder reflections or Givens rotations, see [Lecture → Rem. 3.3.3.21]. This approach is adopted in all numerical libraries including EIGEN. This problem addresses some related algorithmic issues.

This problem assumes familiarity with orthogonal matrices [Lecture → Section 3.3.2] and Householder reflections [Lecture → § 3.3.3.11].

A legacy FORTRAN-77 numerical library routine returns a raw array (type **double** $*$) of n^2 **double** floating-point numbers, $n \in \mathbb{N}$, which represents an $n \times n$-matrix $\mathbf{A} \in \mathbb{R}^{n,n}$ according to the following specification:

(I) The array encodes an $n \times n$ matrix $\mathbf{M} \in \mathbb{R}^{n,n}$ in column-major format [Lecture → Section 1.2.3].

(II) The column vectors $\mathbf{u}_j \in \mathbb{R}^n$, $j = 1, \ldots, n-1$, defined as[1]

$$\left(\mathbf{u}_j\right)_k := \begin{cases} 0 & \text{for } k < j\,, \\ \sqrt{1 - \sum\limits_{\ell=k+1}^{n} (\mathbf{M})_{\ell,j}^2} & \text{for } k = j\,, \\ (\mathbf{M})_{k,j} & \text{for } k > j\,, \end{cases} \qquad j = 1, \ldots, n-1\,, \tag{3.16.1}$$

are the building blocks for the product matrix

$$\mathbf{Q} = (\mathbf{I}_n - 2\mathbf{u}_1\mathbf{u}_1^\top)(\mathbf{I}_n - 2\mathbf{u}_2\mathbf{u}_2^\top) \cdot \cdots \cdot (\mathbf{I}_n - 2\mathbf{u}_{n-1}\mathbf{u}_{n-1}^\top)\,. \tag{3.16.2}$$

(III) The matrix $\mathbf{A}$ is given as $\mathbf{A} = \mathbf{QR}$, where $\mathbf{R} \in \mathbb{R}^{n,n}$ is the *upper triangular* part of $\mathbf{M}$:

$$(\mathbf{R})_{k,j} := \begin{cases} (\mathbf{M})_{k,j} & \text{for } k \le j\,, \\ 0 & \text{else,} \end{cases} \qquad k, j \in \{1, \ldots, n\}\,. \tag{3.16.3}$$

Your task is to write an EIGEN-based wrapper class for the raw data that offers certain operations with the matrix $\mathbf{A}$. The class definition is a follows:

C++ code 3.16.4: Class definition of CompactStorageQR

```
2  class CompactStorageQR {
3  public:
4   // Constructor taking raw data array, which has to be persistent
5   CompactStorageQR(const double *data, unsigned int n) : n_(n), M_(data, n, n) {
6     for (unsigned int k = 0; k < n_ - 1; ++k) {
7       double sn{0};
8       for (unsigned int j = k + 1; j < n_; ++j) {
9         const double t{M_(j, k)};
10        sn += t * t;
11      }
12      if (sn > 1.0) {
13        throw std::runtime_error(
14            "CompactStorageQR: Illegal subdiagonal column norm!");
15      }
16    }
```

[1] A sum whose lower summation bound is larger than the upper summation bound is understood to evaluate to zero.

```
  }
  // Determinant of the matrix stored in this object
  double det() const;
  // Right multiplication with another matrix
  Eigen::MatrixXd matmult(const Eigen::MatrixXd &X) const;
  // Solution of linear systems of equations
  Eigen::MatrixXd solve(const Eigen::MatrixXd &B) const;
 private:
  unsigned int n_;                        // Matrix dimensions
  Eigen::Map<const Eigen::MatrixXd> M_;   // Raw data wrapped into a matrix
};
```

(3-16.a) ⚄ (45 min.) In the file `compactstorageqr.hpp` implement the method

```
Eigen::MatrixXd CompactStorageQR::matmult(
  const Eigen::MatrixXd &X) const;
```

which returns the matrix product $\mathbf{A}\cdot\mathbf{X}$ of the matrix $\mathbf{A}\in\mathbb{R}^{n,n}$ stored in the **CompactStorageQR** object and of a matrix $\mathbf{X}\in\mathbb{R}^{n,k}$, $k\in\mathbb{N}$, passed in `X`. Your implementation must not incur asymptotic computational cost worse than $O(n^2k)$ for $n,k\to\infty$.

SOLUTION of (3-16.a):

By the associativity of matrix multiplication

$$\mathbf{AX}=\mathbf{Q}(\mathbf{RX})=\mathbf{Q}_1\big(\mathbf{Q}_2\big(\dots\mathbf{Q}_{n-1}(\mathbf{RX})\dots\big)\big)\,,\quad \mathbf{Q}_j:=\mathbf{I}_n-2\mathbf{u}_j\mathbf{u}_j^\top\,.$$

As discussed in [Lecture → Ex. 1.4.3.1], for the sake of efficiency the multiplication of a matrix $\mathbf{Y}\in\mathbb{R}^{n,k}$ with $\mathbf{Q}_j$ has to be implemented as follows

$$\mathbf{Q}_j\mathbf{Y}=\mathbf{Y}-2\mathbf{u}_j\Big(\mathbf{u}_j^\top\mathbf{Y}\Big)\,,$$

which requires an asymptotic computational effort of $O(nk)$ for $n,k\to\infty$. These matrix multiplications have to be embedded into an outer loop over $j=n-1,n-2,\dots,1$.

C++ code 3.16.5: Method `matmult()` of CompactStorageQR

```
Eigen::MatrixXd CompactStorageQR::matmult(const Eigen::MatrixXd &X) const {
  assert((X.rows() == n_) && "Wrong size of matrix factor");
  Eigen::MatrixXd Y(n_, X.rows());
  // START Student code
  // Multiplication with R-factor
  Y = M_.template triangularView<Eigen::Upper>() * X;
  // Multiplication with Q-factors
  Eigen::VectorXd u{Eigen::VectorXd::Zero(n_)};
  for (int k = n_ - 2; k >= 0; --k) {
    u.tail(n_ - k - 1) = M_.block(k + 1, k, n_ - k - 1, 1);
    u[k] = std::sqrt(1.0 - u.squaredNorm());
    Y -= 2 * u * (u.transpose() * Y);
  }
  // END Student code
  return Y;
}
```

An even more efficient implementation is possible by taking into account that the first $j-1$ entries of the vector $\mathbf{u}_j$ are zero.

▲

(3-16.b) ⚂ (30 min.) In the file `compactstorageqr.hpp` supply the missing parts of the implementation of

```
Eigen::MatrixXd CompactStorageQR::solve(const Eigen::MatrixXd &B)
    const;
```

which is to compute $\mathbf{A}^{-1}\mathbf{B}$ for $\mathbf{A} \in \mathbb{R}^{n,n}$ stored in the current **CompactStorageQR** object and $\mathbf{B} \in \mathbb{R}^{n,k}$ contained in the argument `B`. Again, the asymptotic complexity of your implementation must be $O(n^2k)$ for $n, k \to \infty$.

HINT 1 for (3-16.b): You must exploit that the factor matrices in (3.16.2) are Householder matrices [Lecture → § 3.3.3.11], in particular, *orthogonal*. ⌟

SOLUTION of (3-16.b):

Based on the factorization of $\mathbf{A}$,

$$\mathbf{A} = \mathbf{Q}\mathbf{R} = \mathbf{Q}_1\mathbf{Q}_2 \cdot \cdots \cdot \mathbf{Q}_{n-1}\mathbf{R}\,, \quad \mathbf{Q}_j := \mathbf{I}_n - 2\mathbf{u}_j\mathbf{u}_j^\top\,,$$

we have

$$\mathbf{A}^{-1}\mathbf{B} = \mathbf{R}^{-1}\mathbf{Q}_{n-1}^{-1}\mathbf{Q}_{n-2}^{-1} \cdot \cdots \cdot \mathbf{Q}_1^{-1}\mathbf{B}\,.$$

Note that, since the vectors $\mathbf{u}_j$ from (3.16.1) are normalized, $\left\|\mathbf{u}_j\right\|_2 = 1$, the matrices $\mathbf{Q}_j$ are *orthogonal* and symmetric **Householder matrices** [Lecture → § 3.3.3.11] satisfying

$$\mathbf{Q}_j^{-1} = \mathbf{Q}_j^\top = \mathbf{Q}_j\,, \quad j = 1, \ldots, n-1\,. \tag{3.16.6}$$

We infer

$$\mathbf{A}^{-1}\mathbf{B} = \mathbf{R}^{-1}\mathbf{Q}_{n-1}\mathbf{Q}_{n-2} \cdot \cdots \cdot \mathbf{Q}_1\mathbf{B}\,,$$

which suggests an implementation very similar to `CompactStorageQR::matmult()` in Code 3.16.5. Of course $\mathbf{R}^{-1}\mathbf{Y}$ for $\mathbf{Y} \in \mathbb{R}^{n,k}$ is obtained by solving the k linear systems of equations $\mathbf{R}\mathbf{x} = (\mathbf{Y})_\ell$, $\ell = 1, \ldots, k$, which can be done with an asymptotic computational effort of $O(n^2)$ for $n \to \infty$, because $\mathbf{R}$ is *upper triangular*.

C++ code 3.16.7: Method `solve()` of class CompactStorageQR

```
2  Eigen::MatrixXd CompactStorageQR::solve(const Eigen::MatrixXd &B) const {
3    assert((B.rows() == n_) && "Wrong size of right-hand side");
4    Eigen::MatrixXd X{B};
5    // START student code
6    // Check invertibility by inspecting diagonal elements of R-factor
7    const double atol = 1.0E-16;
8    const double rtol = 1.0E-8;
9    double maxrii = std::abs(M_(0, 0));
10   for (unsigned int k = 1; k < n_; ++k) {
11     const double tmp = std::abs(M_(k, k));
12     if (maxrii < tmp) {
13       maxrii = tmp;
14     }
15   }
```

```
  for (unsigned int k = 1; k < n_; ++k) {
    const double rii = std::abs(M_(k, k));
    if ((rii < rtol * maxrii) || (rii < atol)) {
      throw std::runtime_error("CompactStorageQR::solve: matrix not regular");
    }
  }
  // Multiplication with transposed Q-factors
  Eigen::VectorXd u(n_);
  for (unsigned int k = 0; k < n_ - 1; ++k) {
    double sn{0};
    for (unsigned int j = k + 1; j < n_; ++j) {
      u[j] = M_(j, k);
      sn += u[j] * u[j];
    }
    u[k] = std::sqrt(1.0 - sn);
    std::cout<<"output here"<< u << std::endl;
    // Multiplication with current Q-factor
    X -= 2 * u * (u.transpose() * X);
    u[k] = 0.0;
  }
  // Solve upper triangular linear system RX = Q^T B
  X = M_.template triangularView<Eigen::Upper>().solve(X);
  // END student code
  return X;
}
```

This code also checks for the invertibility of the matrix $\mathbf{A}$, which was not requested in this sub-problem.

Inspecting the loop we see that the total asymptotic computational cost of `solve()` for $\mathbf{B} \in \mathbb{R}^{n,k}$ is $O(n^2k)$, which is optimal.

▲

(3-16.c) ⚂ (30 min.) In the file `compactstorageqr.hpp` write the code for the method

```
double CompactStorageQR::det() const;
```

which gives the **determinant** $\det \mathbf{A}$ of the matrix $\mathbf{A} \in \mathbb{R}^{n,n}$ stored in the current **CompactStorageQR** object.

HINT 1 for (3-16.c): From linear algebra remember that $\det(\mathbf{AX}) = \det \mathbf{A} \cdot \det \mathbf{X}$ for all $\mathbf{AX} \in \mathbb{R}^{n,n}$.

SOLUTION of (3-16.c):

Thanks for multiplication theorem for determinants the factorization

$$\mathbf{A} = \mathbf{QR} = \mathbf{Q}_1\mathbf{Q}_2 \cdot \cdots \cdot \mathbf{Q}_{n-1}\mathbf{R}\,, \quad \mathbf{Q}_j := \mathbf{I}_n - 2\mathbf{u}_j\mathbf{u}_j^\top\,,$$

immediately implies

$$\det \mathbf{A} = \prod_{j=1}^{n-1} \det \mathbf{Q}_j \cdot \det \mathbf{R}\,.$$

Note that

- the matrices $\mathbf{Q}_j$ are *orthogonal* and symmetric **Householder matrices** [Lecture → § 3.3.3.11] satisfying $\det \mathbf{Q}_j = -1$,
- the determinant of the *upper triangular* matrix $\mathbf{R}$ is equal to the product of its diagonal elements.

Remark. To see that $\det \mathbf{Q}_j = -1$, let $\mathbf{U}_j \in \mathbb{R}^{n,n}$ an *orthogonal* matrix with first column $\mathbf{u}_j$:

$$\mathbf{U}_j \mathbf{Q}_j \mathbf{U}_j = \mathbf{U}_j^\top (\mathbf{I}_n - 2\mathbf{u}_j \mathbf{u}_j^\top)\mathbf{U}_j = \operatorname{diag}(-1,1,\ldots,1)\ .$$

C++ code 3.16.8: Method `det` of class CompactStorageQR

```
2  double CompactStorageQR::det() const {
3    double d;
4    // START Student code
5    // The determinant is just the product of the diagonal entries of
       the
6    // R-factor, because the determinant of the Q-factor is 1.
7    d = 1.0;
8    for (unsigned int j = 0; j < n_; ++j) {
9      d *= M_(j, j);
10   }
11   // Sign correction, because Household transformations
12   // have determinant -1 and n-1 of them are inviolved.
13   d *= ((n_%2 == 0)?-1.0:1.0);
14   // END Student code
15   return d;
16 }
```

▲

End Problem 3-16, 105 min.