Prof. R. Hiptmair
G. Alberti,
F. Leonardi

AS 2015

Numerical Methods for CSE

ETH Zürich
D-MATH

Problem Sheet 0

These problems are meant as an introduction to EIGEN in the first tutorial classes of the new semester.

Problem 1. Gram-Schmidt orthogonalization with EIGEN

[1, Code 1.5.3] presents a MATLAB code that effects the Gram-Schmidt orthogonalization of the columns of an argument matrix.

(1a) ⚁ Based on the C++ linear algebra library EIGEN implement a function

```
template <class Matrix>
Matrix gramschmidt(const Matrix &A);
```

that performs the same computations as [1, Code 1.5.3].

Solution: See `gramschmidt.cpp`.

(1b) ⚁ Test your implementation by applying it to a small random matrix and checking the orthonormality of the columns of the output matrix.

Solution: See `gramschmidt.cpp`.

Problem 2. Fast matrix multiplication

[1, Rem. 1.4.9] presents Strassen's algorithm that can achieve the multiplication of two dense square matrices of size $n = 2^k$, $k \in \mathbb{N}$, with an asymptotic complexity better than $O(n^3)$.

(2a) ⚁ Using EIGEN implement a function

```
MatrixXd strassenMatMult(const MatrixXd & A, const
    MatrixXd & B)
```

that uses Strassen's algorithm to multiply the two matrices $\mathbf{A}$ and $\mathbf{B}$ and return the result as output.

Solution: See Listing 1.

(2b) Validate the correctness of your code by comparing the result with EIGEN's built-in matrix multiplication.

Solution: See Listing 1.

(2c) Measure the runtime of your function `strassenMatMult` for random matrices of sizes 2^k, $k = 4, \ldots, 10$, and compare with the matrix multiplication offered by the $*$-operator of EIGEN.

Solution: See Listing 1.

Listing 1: EIGEN Implementation of the Strassen's algorithm and runtime comparisons.

```
#include <Eigen/Dense>
#include <iostream>
#include <vector>

 #include "timer.h"

using namespace Eigen;
using namespace std;

//! \brief Compute the Matrix product A×B using
    Strassen's algorithm.
//! \param[in] A Matrix 2^k×2^k
//! \param[in] B Matrix 2^k×2^k
//! \param[out] Matrix product of A and B of dim 2^k×2^k

MatrixXd strassenMatMult(const MatrixXd & A, const
    MatrixXd & B)
{
    int n=A.rows();
    MatrixXd C(n,n);
```

```
    if (n==2)
    {
        C<< A(0,0)*B(0,0) + A(0,1)*B(1,0),
            A(0,0)*B(0,1) + A(0,1)*B(1,1),
            A(1,0)*B(0,0) + A(1,1)*B(1,0),
            A(1,0)*B(0,1) + A(1,1)*B(1,1);
        return C;
    }

    else
    {   MatrixXd
      Q0(n/2,n/2),Q1(n/2,n/2),Q2(n/2,n/2),Q3(n/2,n/2),
       Q4(n/2,n/2),Q5(n/2,n/2),Q6(n/2,n/2);

        MatrixXd A11=A.topLeftCorner(n/2,n/2);
        MatrixXd A12=A.topRightCorner(n/2,n/2);
        MatrixXd A21=A.bottomLeftCorner(n/2,n/2);
        MatrixXd A22=A.bottomRightCorner(n/2,n/2);

        MatrixXd B11=B.topLeftCorner(n/2,n/2);
        MatrixXd B12=B.topRightCorner(n/2,n/2);
        MatrixXd B21=B.bottomLeftCorner(n/2,n/2);
        MatrixXd B22=B.bottomRightCorner(n/2,n/2);

        Q0=strassenMatMult(A11+A22,B11+B22);
        Q1=strassenMatMult(A21+A22,B11);
        Q2=strassenMatMult(A11,B12-B22);
        Q3=strassenMatMult(A22,B21-B11);
        Q4=strassenMatMult(A11+A12,B22);
        Q5=strassenMatMult(A21-A11,B11+B12);
        Q6=strassenMatMult(A12-A22,B21+B22);

        C<< Q0+Q3-Q4+Q6 ,
        Q2+Q4,
        Q1+Q3,
        Q0+Q2-Q1+Q5;
        return C;
```

```
    }
}

int main(void)
{
    srand((unsigned int) time(0));

    //check if strassenMatMult works
    int k=2;
    int n=pow(2,k);
    MatrixXd A=MatrixXd::Random(n,n);
    MatrixXd B=MatrixXd::Random(n,n);
    MatrixXd AB(n,n), AxB(n,n);
    AB=strassenMatMult(A,B);
    AxB=A*B;
    cout<<"Using Strassen's method, A*B="<<AB<<endl;
    cout<<"Using standard method, A*B="<<AxB<<endl;
    cout<<"The norm of the error is "<<(AB-AxB).norm()<<endl;

    //compare runtimes of strassenMatMult and of direct multiplication

    unsigned int repeats = 10;
    timer<> tm_x, tm_strassen;
    std::vector<int> times_x, times_strassen;

    for(unsigned int k = 4; k <= 10; k++) {
        tm_x.reset();
        tm_strassen.reset();
        for(unsigned int r = 0; r < repeats; ++r) {
            unsigned int n = pow(2,k);
            A = MatrixXd::Random(n,n);
            B = MatrixXd::Random(n,n);
            MatrixXd AB(n,n);

            tm_x.start();
            AB=A*B;
```

```
            tm_x.stop();

            tm_strassen.start();
            AB=strassenMatMult(A,B);
            tm_strassen.stop();
        }
        std::cout << "The standard matrix multiplication took:       " << tm_x.avg().count() / 1000000. << " ms" << std::endl;
        std::cout << "The Strassen's algorithm took:       " << tm_strassen.avg().count() / 1000000. << " ms" << std::endl;

        times_x.push_back( tm_x.avg().count() );
        times_strassen.push_back( tm_strassen.avg().count() );
    }

    for(auto it = times_x.begin(); it != times_x.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;
    for(auto it = times_strassen.begin(); it != times_strassen.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;

}
```

Problem 3.   Householder reflections

This problem is a supplement to [1, Section 1.5.1] and related to Gram-Schmidt orthogonalization, see [1, Code 1.5.3]. Before you tackle this problem, please make sure that you remember and understand the notion of a QR-decomposition of a matrix, see [1, Thm. 1.5.7]. This problem will put to the test your advanced linear algebra skills.

(3a) ⚂ Listing 2 implements a particular MATLAB function.

Listing 2: MATLAB implementation for Problem 3. in file `houserefl.m`

```
function Z = houserefl(v)
% Porting of houserefl.cpp to Matlab code
% v is a column vector
    % Size of v
    n = size(v,1);

    w = v/norm(v);
    u = w + [1;zeros(n-1,1)];
    q = u/norm(u);
    X = eye(n) - 2*q*q';

    % Remove first column X(:,1) \in span(v)
    Z = X(:,2:end);
end
```

Write a C++ function with declaration:

```
void houserefl(const VectorXd &v, MatrixXd &Z);
```

that is equivalent to the MATLAB function `houserefl()`. Use data types from EIGEN.

Solution:

Listing 3: C++implementation for Problem 3. in file `houserefl.cpp`

```
void houserefl(const Eigen::VectorXd & v,
   Eigen::MatrixXd & Z)
{
    unsigned int n = v.size();
    Eigen::VectorXd w = v.normalized();
    Eigen::VectorXd u=w;
    u(0) += 1;
    Eigen::VectorXd  q=u.normalized();
    Eigen::MatrixXd X = Eigen::MatrixXd::Identity(n, n)
       - 2*q*q.transpose();
```

```
9       Z = X.rightCols(n-1);
10  }
```

(3b) Show that the matrix $\mathbf{X}$, defined at line 10 in Listing 2, satisfies:

$$\mathbf{X}^\top \mathbf{X} = \mathbf{I}_n$$

HINT: $\|\mathbf{q}\|^2 = 1$.

Solution:

$$\begin{aligned}
\mathbf{X}^\top \mathbf{X} &= (\mathbf{I}_n - 2\mathbf{q}\mathbf{q}^\top)(\mathbf{I}_n - 2\mathbf{q}\mathbf{q}^\top) \\
&= \mathbf{I}_n - 4\mathbf{q}\mathbf{q}^\top + 4\mathbf{q}\underbrace{\mathbf{q}^\top\mathbf{q}}_{=\|\mathbf{q}\|=1}\mathbf{q}^\top \\
&= \mathbf{I}_n - 4\mathbf{q}\mathbf{q}^\top + 4\mathbf{q}\mathbf{q}^\top \\
&= \mathbf{I}_n
\end{aligned}$$

(3c) Show that the first column of $\mathbf{X}$, after line 9 of the function `houserefl`, is a multiple of the vector $\mathbf{v}$.

HINT: Use the previous hint, and the facts that $\mathbf{u} = \mathbf{w} + \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ and $\mathbf{w} = 1$.

Solution: Let $\mathbf{X} = [\mathbf{X}_1, \cdots, \mathbf{X}_n]$ be the matrix of line 9 in Listing 2, then:

$$
\begin{aligned}
\mathbf{X}_1 &= \mathbf{e}^{(1)} - 2q_1\mathbf{q} \\
&= \begin{bmatrix} 1 - 2q_1^2 \\ -2q_1q_2 \\ \vdots \\ -2q_1q_n \end{bmatrix} \\
&= \begin{bmatrix} 1 - 2\frac{u_1^2}{\sum_{i=1}^n u_i^2} \\ -2\frac{u_1u_2}{\sum_{i=1}^n u_i^2} \\ \vdots \\ -2\frac{u_1u_n}{\sum_{i=1}^n u_i^2} \end{bmatrix} \\
&\overset{Hint}{=} \begin{bmatrix} \frac{(w_1+1)^2+w_2^2+\cdots+w_n^2-2(w_1+1)^2}{(w_1+1)^2+w_2^2+\cdots+w_n^2} \\ -\frac{2(w_1+1)w_2}{(w_1+1)^2+w_2^2+\cdots+w_n^2} \\ \cdots \\ -\frac{2(w_1+1)w_n}{(w_1+1)^2+w_2^2+\cdots+w_n^2} \end{bmatrix} \\
&\overset{\|w\|=1}{=} \begin{bmatrix} \frac{2w_1(w_1+1)}{2(w_1+1)} \\ \frac{2(w_1+1)w_2}{2(w_1+1)} \\ \cdots \\ \frac{2(w_1+1)w_n}{2(w_1+1)} \end{bmatrix} \\
&= -\mathbf{w} = -\frac{\mathbf{v}}{\|\mathbf{v}\|},
\end{aligned}
$$

which is a multiple of $\mathbf{v}$.

(3d) ⚁ What property does the set of columns of the matrix $\mathbf{Z}$ have? What is the purpose of the function `houserefl`?

HINT: Use (3b) and (3c).

Solution: The columns of $\mathbf{X} = [\mathbf{X}_1, \cdots, \mathbf{X}_n]$ are an orthonormal basis (ONB) of $\mathbb{R}^n$ (cf. (3b)). Thus, the columns of $\mathbf{Z} = [\mathbf{X}_2, \cdots, \mathbf{X}_n]$ are an ONB of the complement of $\mathrm{Span}(\mathbf{X}_1) \overset{(3c)}{=} \mathrm{Span}(\mathbf{v})$. The function `houserefl` computes an ONB of the complement of $\mathbf{v}$.

(3e) ⚀ What is the asymptotic complexity of the function `houserefl` as the length n of the input vector $\mathbf{v}$ goes to ∞?

Solution: $O(n^2)$: this is the asymptotic complexity of the construction of the tensor product at line 9 of Listing 3.

(3f) Rewrite the function as MATLAB function and use a *standard function* of MATLAB to achieve the same result of lines 5-9 with a single call to this function.

HINT: It is worth reading [1, Rem. 1.5.10] before mulling over this problem.

Solution: Check the code in Listing 2 for the porting to MATLAB code. Using the QR-decomposition `qr` one can rewrite (cf. Listing 4) the C++ code in MATLAB with a few lines.

Listing 4: MATLAB implementation for Problem 3. in file `qr_houserefl.m` using QR decomposition.

```
function Z = qr_houserefl(v)
% Use qr decomposition to find ONB of complement of
   span(v)
    [X,R] = qr(v);

    % Remove first column X(:,1) \in span(v)
    Z = X(:,2:end);
end
```

Issue date: 21.09.2015

Hand-in: — (in the boxes in front of HG G 53/54).

Version compiled on: September 20, 2015 (v. 1.1).