Numerical Software I

Linear Regression with Eigen

Viktor Mosenkis

Informatik 12:
Software and Tools for Computational Engineering (STCE)

RWTH Aachen University
Contents

Objectives and Learning Outcomes

Numerical Software

Linear Regression

Normal Equation
  Linear Regression with Normal Equation
  Implementation with Eigen

QR Decomposition
  Linear Regression with QR decomposition
  Implementation with Eigen

Singular Value Decomposition
  Linear Regression with SVD
  Implementation with Eigen

Summary and Next Steps
Outline

Objectives and Learning Outcomes

Numerical Software

Linear Regression

Normal Equation
  Linear Regression with Normal Equation
  Implementation with Eigen

QR Decomposition
  Linear Regression with QR decomposition
  Implementation with Eigen

Singular Value Decomposition
  Linear Regression with SVD
  Implementation with Eigen

Summary and Next Steps
Objective

▶ Introduction to numerical software,
▶ Use Eigen to solve the linear regression problem for nonscalar case.

Learning Outcomes

▶ You will understand how to solve linear regression problem with
  ▶ normal equation
  ▶ QR decomposition
  ▶ Singular Value Decomposition (SVD).

▶ You will be able to
  ▶ implement linear regression methods with Eigen.
Outline

Objectives and Learning Outcomes

Numerical Software

Linear Regression

Normal Equation
  Linear Regression with Normal Equation
  Implementation with Eigen

QR Decomposition
  Linear Regression with QR decomposition
  Implementation with Eigen

Singular Value Decomposition
  Linear Regression with SVD
  Implementation with Eigen

Summary and Next Steps
Numerical software is software that implements numerical algorithms. E.g. QR decomposition, Newton’s method, etc.

Numerical Software that will be used in this course

- Eigen
- GSL - GNU Scientific Library
- NAG Library
Why should I use Numerical Software?

**Pro:**
- Concentrate on your problem. Typically your task is to solve some problem, e.g. solve linear equation system rather than implement the corresponding algorithm.
- Implement correct and stable numerical algorithm is not a trivial task.
- Faster development
- Try alternative algorithms at small costs
- Better performance
- Stay in your area of expertise

**Contra:**
- Rely on the software vendor to keep supporting the product. E.g. port it to different systems, fix bugs etc.
- Licensing issues
Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.

- Eigen is Free Software and starting from 3.1.1 version is licensed under the MPL2
- Eigen is a header only library so there is no library to link
- You can download Eigen and access documentation under https://eigen.tuxfamily.org
Outline

Objectives and Learning Outcomes

Numerical Software

Linear Regression

Normal Equation
  Linear Regression with Normal Equation
  Implementation with Eigen

QR Decomposition
  Linear Regression with QR decomposition
  Implementation with Eigen

Singular Value Decomposition
  Linear Regression with SVD
  Implementation with Eigen

Summary and Next Steps
Linear Regression

Exploitation of Special Structure

A linear (in $p$) multivariate scalar model

$$y = f(p, x) = g(x)^T \cdot p$$, for $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$

yields the linear regression problem

$$A \cdot p \approx y$$

for given data $X \in \mathbb{R}^{m \times n}, y \in \mathbb{R}^m, m \geq n$ and $A = (g(X_i^T)^T) \in \mathbb{R}^{m \times n}$.

Minimization of the error

$$E(p) = \sum_{i=0}^{m-1} (f(p, X_i^T) - y_i)^2 = \sum_{i=0}^{m-1} (g(X_i^T)^T \cdot p - y_i)^2 = \|A \cdot p - y\|^2_2$$

can be regarded as a convex minimization problem (see module Newton_II). Exploitation of special problem structure yields potentially more efficient solution methods.
Outline

Objectives and Learning Outcomes

Numerical Software

Linear Regression

Normal Equation
  Linear Regression with Normal Equation
  Implementation with Eigen

QR Decomposition
  Linear Regression with QR decomposition
  Implementation with Eigen

Singular Value Decomposition
  Linear Regression with SVD
  Implementation with Eigen

Summary and Next Steps
Normal Equation

Nonscalar Case

The optimality conditions for minimization of the error function

\[ E = \| A \cdot p - y \|_2^2 = \sum_{i=0}^{m-1} (a_i \cdot p - y_i)^2 \quad (\in \mathbb{R}) , \]

where \( a_i \in \mathbb{R}^{1 \times n} \) denotes the \( i \)-th row of \( A \), require

\[ \sum_{i=0}^{m-1} a_i^T \cdot a_i \cdot p - a_i^T \cdot y_i = 0 \]

yielding the normal equation

\[ A^T \cdot A \cdot p = A^T \cdot y \]

which can be solved by \( LL^T \) (\( LDL^T \)) factorization.

Note: \( \text{cond}(A^T A) = [\text{cond}(A)]^2 \). More details see module LinearRegression_II
template<typename T, int M, int N>
void Regression_NormalEquation(
    const Eigen::Matrix<T,M,N> &A,
    Eigen::Matrix<T,N,1> &p,
    const Eigen::Matrix<T,M,1> &y
) {
    p=(A.transpose() ∗ A).llt().solve(A.transpose() ∗ y);
}

int main(int argc, char* argv[]) {
    assert(argc==3); int m=std::stoi(argv[1]), n=std::stoi(argv[2]);
    using T=double;
    using MT=Eigen::Matrix<T,Eigen::Dynamic,Eigen::Dynamic>;
    using VT=Eigen::Matrix<T,Eigen::Dynamic,1>;
    MT A=MT::Random(m,n); VT p(n), y=VT::Random(m);
    Regression_NormalEquation(A,p,y);
    std::cout << "p=" << p.transpose() << std::endl;
    return 0;
}
Outline

Objectives and Learning Outcomes

Numerical Software

Linear Regression

Normal Equation
  Linear Regression with Normal Equation
  Implementation with Eigen

QR Decomposition
  Linear Regression with QR decomposition
  Implementation with Eigen

Singular Value Decomposition
  Linear Regression with SVD
  Implementation with Eigen

Summary and Next Steps
**Linear Regression with QR decomposition**

**Nonscalar Case**

*QR decomposition* of the matrix $A \in \mathbb{R}^{m \times n}$ yields

$$A = Q \cdot \begin{bmatrix} R \\ 0 \end{bmatrix}$$

with *orthogonal* $Q \in \mathbb{R}^{m \times m}$ and *upper triangular* $R \in \mathbb{R}^{n \times n}$. Hence the linear regression problem $A \cdot p \approx y$ can be reformulated as follows

$$\| r \|_2^2 = \| y - Ap \|_2^2 = \| y - Q \begin{bmatrix} R \\ 0 \end{bmatrix} p \|_2^2 = \| Q^T y - Q^T Q \begin{bmatrix} R \\ 0 \end{bmatrix} p \|_2^2$$

$$= \| Q^T y - \begin{bmatrix} R \\ 0 \end{bmatrix} p \|_2^2 = \| \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} - \begin{bmatrix} R \\ 0 \end{bmatrix} p \|_2^2 = \| c_1 - R p \|_2^2 + \| c_2 \|_2^2$$

Hence to solve the linear regression problem we must choose $p$ such that $R p = c_1$. 

Numerical Software I, info@stce.rwth-aachen.de
Linear Regression with QR decomposition

Implementation with Eigen

In **LinearRegression_II** you will learn how to implement *QR* decomposition algorithms with Householder Reflections and Givens Rotations. In this module we simply use the corresponding implementation from Eigen.

```cpp
#include <Eigen/Dense>

template<typename T, int M, int N>
void Regression_QR(
    const Eigen::Matrix<T,M,N> &A,
    Eigen::Matrix<T,N,1> &p,
    const Eigen::Matrix<T,M,1> &y) {
    p = A.colPivHouseholderQr().solve(y);
}

int main(int argc, char* argv[]) {
    assert(argc==3); int m=std::stoi(argv[1]), n=std::stoi(argv[2]);
    using T=double;
    using MT=Eigen::Matrix<T,Eigen::Dynamic,Eigen::Dynamic>;
    using VT=Eigen::Matrix<T,Eigen::Dynamic,1>;
    MT A=MT::Random(m,n); VT p(n), y=VT::Random(m);
    Regression_QR(A,p,y);
    std::cout << "p=" << p.transpose() << std::endl;
    return 0;
}
```

Numerical Software I, info@stce.rwth-aachen.de
Outline

Objectives and Learning Outcomes

Numerical Software

Linear Regression

Normal Equation
  Linear Regression with Normal Equation
  Implementation with Eigen

QR Decomposition
  Linear Regression with QR decomposition
  Implementation with Eigen

Singular Value Decomposition
  Linear Regression with SVD
  Implementation with Eigen

Summary and Next Steps
Singular Value Decomposition (SVD)

Overview

SVD decomposition of the matrix $A \in \mathbb{R}^{m \times n}$ yields

$$ A = U \cdot W \cdot V^T $$

with orthogonal $U \in \mathbb{R}^{m \times m}$, orthogonal $V^T \in \mathbb{R}^{n \times n}$ and diagonal, $W \in \mathbb{R}^{m \times n}$, i.e.,

$$ W = \begin{pmatrix} w_1 & \cdots & w_n \\ 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix} = \begin{pmatrix} W' \end{pmatrix}, \quad \text{for } m \geq n $$

with diagonal entries

$$ w_1 \geq w_2 \geq \cdots \geq w_n \geq 0. $$
Useful properties of SVD

- $w_i$ are called **singular values** of $A$.
- If $A$ is singular, some of the $w_i$ will be 0.
- $\text{rank}(A) = \text{number of nonzero } w_i$
- **Inverses**: $A^{-1} = (V^T)^{-1} \cdot W^{-1} \cdot U^{-1} = V \cdot W^{-1} \cdot U^T$
  - exploiting the fact that transpose=inverse for orthogonal matrices
  - $W$ is diagonal, hence $W^{-1}$ is also diagonal with reciprocals of entries of $W$
- **Pseudoinverses**: if $w_i = 0$ set $1/w_i$ to 0.
  - matrix ”closest” to inverse
  - defined for all (even non-square, singular, etc.) matrices
Using SVD the linear regression problem $A \cdot p \approx y$ can be reformulated as follows

$$\|r\|_2^2 = \|y - Ap\|_2^2 = \|y - U \begin{bmatrix} W' \\ 0 \end{bmatrix} V^T p\|_2^2 = \|U^T y - U^T U \begin{bmatrix} W' \\ 0 \end{bmatrix} V^T p\|_2^2 = \|U^T y - \begin{bmatrix} W' \\ 0 \end{bmatrix} V^T p\|_2^2 = \|c_1 - W' V^T p\|_2^2 + \|c_2\|_2^2$$

where $U^T y = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$

Hence to solve the linear regression problem we must choose $p$ such that $W' V^T p = c_1$. 
Linear Regression with SVD

Implementation

```cpp
template<typename T, int M, int N>
void Regression_SVD(
    const Eigen::Matrix<T, M, N> &A,
    Eigen::Matrix<T, N, 1> &p,
    const Eigen::Matrix<T, M, 1> &y) {
    Eigen::JacobiSVD<Eigen::Matrix<T, M, N>> svd(A, Eigen::ComputeThinU | Eigen::ComputeThinV);
    p = svd.solve(y);
}

int main(int argc, char* argv[]) {
    assert(argc==3);
    int m=std::stoi(argv[1]), n=std::stoi(argv[2]);
    using T=double;
    using MT=Eigen::Matrix<T, Eigen::Dynamic, Eigen::Dynamic>;
    using VT=Eigen::Matrix<T, Eigen::Dynamic, 1>;
    MT A=MT::Random(m, n); VT p(n), y=VT::Random(m);
    Regression_SVD(A,p,y);
    std::cout << "p=" << p.transpose() << std::endl;
    return 0;
}
```
Outline

Objectives and Learning Outcomes

Numerical Software

Linear Regression

Normal Equation
- Linear Regression with Normal Equation
- Implementation with Eigen

QR Decomposition
- Linear Regression with QR decomposition
- Implementation with Eigen

Singular Value Decomposition
- Linear Regression with SVD
- Implementation with Eigen

Summary and Next Steps
Linear Regression
What method should you choose?

- **Normal equations**
  - $O(mn^2)$ using $LL^T$
  - $\text{cond}(A^T A) = [\text{cond}(A)]^2$

- **QR decomposition**
  - $O(mn^2 - n^3/3)$ operation

- **SVD**
  - Expensive $O(mn^2 + n^3)$
  - Can handle rank-deficiency
  - Can handle near-singularity
Summary

- Implemented linear regression methods for multivariate scalar models in Eigen based on
  - normal equation
  - QR decomposition
  - SVD

Next Steps

- Play with sample code.
- Compare different implementations of QR and SVD algorithms included in Eigen. Can you confirm the recommendations given in Eigen documentation which algorithm QR or SVD algorithm to choose.
- Continue the course to find out more ...