NumCSE

Autumn Semester 2017

Prof. Rima Alaifari

Exercise sheet 3
Linear Least Squares, QR decomposition

P. Bansal

Problem 3.1: Estimating a Tridiagonal Matrix

To determine the least squares solution of an overdetermined linear system of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$ we minimize the residual norm $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2$ w.r.t. $\mathbf{x}$. However, we also face a linear least squares problem when minimizing the residual norm w.r.t. the entries of $\mathbf{A}$.
Template: `tridiagleastsquares.cpp`

Let two vectors $\mathbf{z}, \mathbf{c} \in \mathbb{R}^n$, for $n > 2 \in \mathbb{N}$, be given. Define α^* and β^* as:

$$(\alpha^*, \beta^*) = \underset{\alpha,\beta\in\mathbb{R}}{\operatorname{argmin}} \left\| \mathbf{T}_{\alpha,\beta}\mathbf{z} - \mathbf{c} \right\|_2, \tag{3.1}$$

where $\mathbf{T}_{\alpha,\beta} \in \mathbb{R}^{n\times n}$ is the following tridiagonal matrix:

$$\mathbf{T}_{\alpha,\beta} = \begin{bmatrix} \alpha & \beta & 0 & \dots & 0 \\ \beta & \alpha & \beta & \ddots & \vdots \\ 0 & \beta & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \alpha & \beta \\ 0 & \dots & 0 & \beta & \alpha \end{bmatrix} \tag{3.2}$$

(a) Reformulate Eq. (3.1) as a linear least squares problem:

$$\mathbf{x}^* = \underset{\mathbf{x}\in\mathbb{R}^k}{\operatorname{argmin}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2, \tag{3.3}$$

where $\mathbf{A} \in \mathbb{R}^{m,k}$ and $\mathbf{b} \in \mathbb{R}^m$, for $m, k \in \mathbb{N}$.

Hint: For $\mathbf{x} = [\alpha, \beta]^\top$, find $\mathbf{A}$ such that $\mathbf{T}_{\alpha,\beta}\mathbf{z} = \mathbf{A}\mathbf{x}$.

(b) Implement a C++ function which solves the linear least squares problem of Eq. (3.1) using the normal equation method and returns the optimal parameters α^* and β^*:

```
VectorXd lsqEst(const VectorXd &z, const VectorXd &c);
```

Let $\mathbf{X} \in \mathbb{R}^{m,n}$, $\mathbf{a} \in \mathbb{R}^m$ and $\mathbf{b} \in \mathbb{R}^n$. Consider the scalar functions:

$$\Phi_1(\mathbf{X}) = \|\mathbf{X}\|_F^2 \tag{3.4a}$$

$$\Phi_2(\mathbf{X}) = \mathbf{a}^\top \mathbf{X}\mathbf{b}. \tag{3.4b}$$

Here $\|\cdot\|_F$ is the Frobenius norm of a matrix.

(c) Compute $\dfrac{\partial(\Phi_1(\mathbf{X}))}{\partial \mathbf{X}}$ and $\dfrac{\partial(\Phi_2(\mathbf{X}))}{\partial \mathbf{X}}$.

Problem 3.2: Sparse Approximate Inverse (SPAI)

The SPAI method is a technique used in the numerical solution of partial differential equations. From a least squares viewpoint, we encounter a non-standard least squares problems. SPAI techniques are applied to huge and extremely sparse matrices, say, of dimension $10^7 \times 10^7$ with only 10^8 non-zero entries. Therefore, sparse matrix techniques must be applied.

Let $\mathbf{A} \in \mathbb{R}^{N,N}$, $N \in \mathbb{N}$, be a regular sparse matrix with at most $n \ll N$ non-zero entries per row and column. We define the space of matrices with the same pattern as $\mathbf{A}$:

$$\mathcal{P}(\mathbf{A}) := \{\mathbf{X} \in \mathbb{R}^{N,N} : \ (\mathbf{A})_{ij} = 0 \ \Rightarrow \ (\mathbf{X})_{ij} = 0\} \ . \tag{3.5}$$

The "primitive" SPAI (sparse approximate inverse) $\mathbf{B}$ of $\mathbf{A}$ is defined as

$$\mathbf{B} := \underset{\mathbf{X} \in \mathcal{P}(\mathbf{A})}{\operatorname{argmin}} \|\mathbf{I} - \mathbf{A}\mathbf{X}\|_F \ , \tag{3.6}$$

where $\|\cdot\|_F$ stands for the Frobenius norm.

(a) Show that the columns of $\mathbf{B}$ can be computed independently of each other by solving linear least squares problems. Denote columns of $\mathbf{B}$ by $\mathbf{b}_i$.

(b) Implement an efficient C++ function

```
SparseMatrix<double> spai(SparseMatrix<double> & A);
```

for the computation of $\mathbf{B}$ according to (3.6). You may rely on the normal equations associated with the linear least squares problems for computing the columns of $\mathbf{B}$ or you may simply invoke the least squares solver of EIGEN.

Hint: Exploit the underlying CCS data structure of `SparseMatrix<double>` Moreover, build the output matrix $\mathbf{B}$ in EIGEN using the intermediate triplet format.

(c) What is the total asymptotic computational effort of `spai` in terms of the problem size parameters N and n?

Problem 3.3: QR decomposition

In this problem, we study the QR decomposition computed via Cholesky decomposition and Householder reflections. Refer section (2.8.13) in the lecture notes to read about Cholesky decomposition.
Template: `choleskyQR.cpp`

(a) Given a matrix $\mathbf{A} \in \mathbb{R}^{m,n}$, s.t. $\mathrm{rank}(\mathbf{A}) = n$, show that $\mathbf{A}^\top\mathbf{A}$ admits a Cholesky decomposition.

Hint: Cholesky decomposition of a matrix $\mathbf{B}$ exists only if $\mathbf{B}$ is symmetric and positive definite.

(b) Implement an EIGEN based C++ function:

```
void CholeskyQR(const MatrixXd & A, MatrixXd & Q, MatrixXd & R);
```

which computes an *economical* QR decomposition of a given full-rank matrix `A`. Give an analytical proof that your `CholeskyQR` works.

(c) Implement an EIGEN based C++ function:

```
void DirectQR(const MatrixXd & A, MatrixXd & Q, MatrixXd & R);
```

which computes an *economical* QR decomposition of `A` using `HouseholderQR` class from EIGEN. Compare the results from `CholeskyQR` and `DirectQR`.

(d) Let EPS denote the machine precision. Does your function `CholeskyQR` return the correct result, compare with `DirectQR`, for $\mathbf{A} = \begin{bmatrix} 1 & 1 \\ \frac{1}{2}\mathrm{EPS} & 0 \\ 0 & \frac{1}{2}\mathrm{EPS} \end{bmatrix}$? Explain.

Problem 3.4: Givens rotations

In this problem, we look at Givens rotations to compute a QR decomposition and also briefly compare it with Householder reflections.

Let $\mathbf{A} \in \mathbb{R}^{3,2}$,

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 1 & 1 \\ \sqrt{2} & 1 \end{bmatrix} \tag{3.7}$$

(a) Use Householder reflections to transform $\mathbf{A}$ to an upper triangular matrix $\tilde{\mathbf{A}}$. Perform the computations and show the steps on paper.

(b) Verify your computations for Sub-problem (a) using the `HouseholderQR` class provided by EIGEN.

(c) Implement an EIGEN C++ function:

```
void rotInPlane(const Vector2d& x, Matrix2d& G, Vector2d& y);
```

which applies Givens rotation on a 2d vector `x`. It should also avoid cancellation.

(d) Implement an EIGEN C++ function:

```
void givensQR(const Matrixxd& A, MatrixXd& Q, MatrixXd& R);
```

which uses the Givens rotation routine `rotInPlane` successively to compute the QR decomposition of a matrix `A`.

(e) Run basic sanity checks for your implementation of `givensQR` and compare your results with that of `HouseholderQR`. Is the QR decomposition unique?

(f) Compare the complexity of `givensQR` and `HouseholderQR` for a general input matrix $\mathbf{A} \in \mathbb{R}^{m,n}$.