Übung 1 *Singulärwertzerlegung rechteckiger Matrizen*
Sei $A \in \mathbb{R}^{m\times n}$ und $U\Sigma V^T = A$ die zugehörige Singulärwertzerlegung. Hierbei sind $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ orthogonal und $\Sigma \in \mathbb{R}^{m\times n}$ eine Diagonalmatrix, welche die Singulärwerte σ_k beinhaltet.

(a) Zeigen Sie: Sei $x \in \mathbb{R}^n$ eine Linearkombination der Basis $v_1, \dots, v_n \in \mathbb{R}^n$ mit der Anordnung $V = [v_1, \dots, v_n]$, also $x = \sum_{k=1}^n x_k v_k$. Dann gilt:

$$Ax = \sum_{k=1}^{\min(m,n)} x_k \sigma_k u_k \ .$$

Hier sind $u_1, \dots, u_m$ Basis des $\mathbb{R}^m$ mit der Anordnung $U = [u_1, \dots, u_m]$.

(b) Berechnen Sie die Frobenius Norm $\|A\|_2$ und die Konditionszahl $\text{cond}_2(A)$ mit Hilfe der Singulärwerte von A.

(5 Punkte)

Übung 2 *Singulärwertzerlegung konkret*
Berechnen Sie die Singulärwertzerlegung der Matrix

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} .$$

Hinweis: Zeigen Sie, dass die Singulärvektoren U, V Eigenvektoren von AA^T, A^TA sind. Folgern Sie daraus die Singulärwerte stehend in Σ und errechnen Sie aus einer Singulärvektorbasis die andere.
(5 Punkte)

Übung 3 *QR-Zerlegung mittels Householder-Spiegelung (Praktische Übung)*
(Für diese praktische Übung gibt es zwei Wochen Bearbeitungszeit. Abgabe 05.Juli 2018)
In der vorherigen praktischen Übung haben Sie das lineare Ausgleichsproblem

$$\min_{x\in\mathbb{R}^n} \|Ax - b\|_2$$

($b \in \mathbb{R}^m, A \in \mathbb{R}^{m\times n}$ mit $m \geq n$ und Rang(A) = n) mit der LR-Zerlegung gelöst. Hier soll die QR-Zerlegung mit Householder-Spiegelungen eingesetzt werden.

Das auf der Vorlesungshomepage bereit gestellte Hauptprogramm `curvefitting_qr.cc` ist ähnlich zu `curvefitting.cc` aus dem vorherigen Übungsblatt. Die von Ihnen während der letzten praktischen Übung angefertigten Textdateien `b6.dat`, `A6_linear.dat` und `A6_quadratic.dat` werden hier wieder verwendet.

Im Hauptprogramm wird die Funktion

```
template<typename NUMBER>
void solveLeastSquares(hdnum::DenseMatrix<NUMBER>& A,
                       hdnum::Vector<NUMBER>& b,
                       hdnum::Vector<NUMBER>& x)
{
  ...
}
```

aufgerufen, welche die Lösung $x \in \mathbb{R}^n$ unseres linearen Ausgleichsproblems liefern soll.

(a) Implementieren Sie diese Funktion in der Headerdatei `solveLeastSquaresQR.hh` mit dem

Algorithmus 1

1. Berechne die QR-Zerlegung von A. Verwenden Sie dafür die Funktion aus Teil b).
2. Für $j = 1, \ldots, n$:

$$\begin{aligned} v_j &= 1, v_l = A_{lj} & l &= j+1, \ldots, m \\ \beta &= \frac{2}{v^T v} & v &= (v_j, v_{j+1}, \ldots, v_m)^T \\ b &= (I - \beta v v^T) b & b &= (b_j, b_{j+1}, \ldots, b_m)^T \end{aligned}$$

3. Löse das obere Dreieckssytem $Ax = b$ mit Hilfe der Headerdatei `solveR.hh` aus `hdnum`.

(b) Die Funktion

```
template<typename NUMBER>
void householder_QR(hdnum::DenseMatrix<NUMBER>& A)
```

liefert die QR-Zerlegung von A. Dabei wird A in dieser Funktion verändert und enthält am Ende die obere Dreiecksmatrix R und die normierten Householder-Vektoren. Normiert bedeutet hier, dass der Householder-Vektor in der ersten Komponente die 1 enthält und diese deshalb nicht extra abgespeichert werden muss.

Implementieren Sie diese Funktion in der Headerdatei `householderQR.hh` mit dem

Algorithmus 2

Für $j = 1, \ldots, n$:

1. Definiere den Vektor y mit den Einträgen $y_l = A_{lj}, l = j, \ldots, m$, und berechne dafür den normierten Householder-Vektor v mit zugehörigem β. Benutze dafür die Funktion `householder_vector(...)` (siehe weiter unten).
2. Wende die durch $P := I - \beta v v^T$ definierte Householdertransformation auf die Teilmatrix $(A)_{pq}$ mit $p \in \{j, \ldots, m\}$ und $q \in \{j, \ldots, n\}$ an. Benutze dafür die Funktion `householderP_times_matrix(...)` (siehe weiter unten).
3. Solange $j < m$ ist, setze $A_{lj} = v_{l-j+1}$ für $l \in \{j+1, \ldots, m\}$.

Beachten Sie, dass in diesem Algorithmus die Dimensionen der Vektoren und Matrizen mit j variieren.

Die Funktion

```
template<typename NUMBER>
void householder_vector(hdnum::Vector<NUMBER>& x,
                        hdnum::Vector<NUMBER>& v,
                        NUMBER& beta)
```

liefert für einen Vektor x beliebiger Dimension einen normierten Householder-Vektor v gleicher Dimension und eine Zahl β zurück, mit denen man die Householdertransformation $P = I - \beta v v^T$ aufstellen kann. Hier wird P gar nicht explizit berechnet, denn es wird nur eine Wirkung auf eine andere Matrix benötigt.
Die Funktion

```
template<typename NUMBER>
void householderP_times_matrix(const NUMBER& beta,
                               const hdnum::Vector<NUMBER>& v,
                               hdnum::DenseMatrix<NUMBER>& A)
```

bekommt als Input die Zahl β, den Vektor v, eine beliebige Matrix A und gibt das Matrizenprodukt $PA = (I - \beta vv^T)A$ zurück.

Testen Sie Ihre Implementierung für die beiden Fälle `A6_linear.dat` und `A6_quadratic.dat` und vergleichen Sie die dabei gewonnenen Lösungen und deren Defektnorm mit den Resultaten aus der vorherigen praktischen Aufgabe.

Für weitere Einzelheiten bzgl. der Algorithmen, siehe *Golub, van Loan: Matrix Computations*. **(5+5 Punkte)**