Übung 1 *LR Zerlegung im Besonderen*

Sei $A \in \mathbb{R}^{n\times n}$ gegeben durch

$$a_{ij} = \begin{cases} +1 & \text{wenn } i = j \text{ oder } j = n \\ -1 & \text{wenn } i > j \\ 0 & \text{sonst.} \end{cases}$$

a) Begründen Sie, dass die LR Zerlegung ohne Pivotisierung von A die Eigenschaften

$$|l_{ij}| \leq 1 \qquad \text{und} \qquad r_{nn} = 2^{n-1}$$

erfüllt.

b) Zeigen Sie, dass für eine LR Zerlegung mit totaler (Zeilen und Spalten) Pivotisierung gilt:

$$|r_{nn}| = 2 = \max_{i,j=1..n} \{r_{ij}\}$$

Tipp: Erstmal für $n = 4$ ausprobieren und dann passenden Induktionsbeginn wählen.

(5 Punkte)

Übung 2 *LR-Zerlegung tridiagonaler Matrizen*

Gegeben sei eine Tridiagonalmatrix

$$A = \begin{pmatrix} a_1 & b_1 & & & 0 \\ c_2 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & b_{n-1} \\ 0 & & & c_n & a_n \end{pmatrix}$$

mit

$$\begin{aligned} &|a_1| > |b_1| > 0 \\ &|a_j| \geq |b_j| + |c_j| > 0, \quad b_j, c_j \neq 0, \quad j \in \{2, ..., n-1\} \\ &|a_n| \geq |c_n| > 0\,. \end{aligned}$$

Zeigen Sie: Der Algorithmus

$$\begin{aligned} r_1 &:= a_1 \\ l_j &:= c_j / r_{j-1} & j \in \{2, ..., n\} \\ r_j &:= a_j - l_j b_{j-1} & j \in \{2, ..., n\} \end{aligned}$$

ist durchführbar (d.h. $r_1, \cdots, r_n \neq 0$) und liefert die LR-Zerlegung

$$A = \begin{pmatrix} 1 & & & 0 \\ l_2 & 1 & & \\ & \ddots & \ddots & \\ 0 & & l_n & 1 \end{pmatrix} \begin{pmatrix} r_1 & b_1 & & 0 \\ & r_2 & \ddots & \\ & & \ddots & b_{n-1} \\ 0 & & & r_n \end{pmatrix}$$

Es gilt somit $det(A) = \prod_{i=1}^{n} r_i \neq 0$, woraus die Invertierbarkeit von A folgt.

(4 Punkte)

Übung 3 *LR-Zerlegung mit Pivotsuche (Praktische Übung)*

a) Füge der Headerdatei `LR.hh` aus der letzten praktischen Übung drei neue Funktionen hinzu:

```
template<typename T>
void row_equilibrate (hdnum::DenseMatrix<T>& A, hdnum::Vector<T>& s)
{...}
```

soll die Zeilenäquilibrierung der Matrix A durchführen. In den Vektor `s` sollen die einzelnen Zeilenbetragssummen abgespeichert werden. Diese werden benötigt, auch die rechte Seite b richtig zu skalieren. Dafür ist die Funktion

```
template<typename T>
void apply_equilibrate (const hdnum::Vector<T>& s, hdnum::Vector<T>& b)
{...}
```

gedacht.

Die Funktion

```
template<typename T>
void lr_partialpivot (hdnum::DenseMatrix<T>& A,
                      hdnum::Vector<std::size_t>& perm)
{...}
```

soll daraufhin die LR-Zerlegung von A *mit Spaltenpivotsuche* berechnen und das Ergebnis L und R wiederum in die Matrix `A` speichern. Der Indexvektor `perm` soll sich die Permutation der Zeilen merken. Diese Funktion soll die Funktion `void lr(...)` aus der letzten Übung ersetzen. Der Rest der Schritte zur Lösung eines linearen Gleichungssystems analog wie in der letzten praktischen Übung. (Siehe `testLR.cc` aus dem Übungsblatt 7.)

b) Schreiben Sie ein neues C++-Programm, welches unter Benutzung der LR-Zerlegung

- ohne Spaltenpivotsuche
- mit Spaltenpivotsuche
- mit Spaltenpivotsuche und zusätzlich noch vorheriger Zeilenäquilibrierung

das Gleichungssystem

$$\begin{pmatrix} 10^{-16} & 0 & 10^{-16} & 3\cdot 10^{-16} \\ 1 & 10 & 0 & 23 \\ 4 & 12 & 12 & 0 \\ 5\cdot 10^{12} & 0 & 10^{12} & 10^{11} \end{pmatrix} \cdot x = \begin{pmatrix} 1.6\cdot 10^{-15} \\ 113 \\ 64 \\ 8.4\cdot 10^{+12} \end{pmatrix}$$

löst und vergleichen Sie das numerische Ergebnis mit der exakten Lösung $x = (1,2,3,4)^T$.

c) *Bonusaufgabe:* Schreiben Sie eine neue Funktion

```
template<typename T>
void lr_fullpivot( hdnum::DenseMatrix<T>& A,
                   hdnum::Vector<std::size_t>& p,
                   hdnum::Vector<std::size_t>& q )
{...}
```

zur LR-Zerlegung von *A mit totaler Pivotisierung*. Dabei merkt man sich nicht nur die Zeilenvertauschungen in einem Indexvektor `p`, sondern auch die Spaltenvertauschungen in einem Indexvektor `q`. Die Zeilenvertauschungen in `p` müssen mittels der Funktion `void permute_forward()` nach `b` transportiert werden. Entsprechend müssen die Spaltenvertauschungen in `q` mittels einer neuen Funktion

```
template<typename T>
void permute_backward (const hdnum::Vector<std::size_t>& q,
                       hdnum::Vector<T>& x )
{...}
```

nach `x` transportiert werden!

(6+3 Punkte)