Dirk Praetorius
Michele Ruggeri
Bernhard Stiftner

Sommersemester 2016
14.06.2016

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 13

Aufgabe 13.1. Wir betrachten die Klasse `Matrix` und die davon abgeleitete Klasse `SquareMatrix` aus der Vorlesung. Implementieren Sie für die Klasse `SquareMatrix` die Methode `computeLU`, welche die LU-Zerlegung berechnet. Der Rückgabewert (Matrix $R \in \mathbb{R}^{n\times n}$) sei dabei wieder vom Type `SquareMatrix`, wobei die beiden Dreiecksmatrizen L und U in R gespeichert werden sollen. Die Diagonale von L muss hierbei nicht explizit gespeichert werden (Warum?). Nicht jede Matrix $A \in \mathbb{R}^{n\times n}$ hat eine normalisierte LU-Zerlegung $A = LU$, d.h.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ \ell_{21} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \ell_{n1} & \dots & \ell_{n,n-1} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & u_{n-1,n} \\ 0 & \dots & 0 & u_{nn} \end{pmatrix}.$$

Wenn aber A eine normalisierte LU-Zerlegung besitzt, so gilt

$$\begin{aligned} u_{ik} &= a_{ik} - \sum_{j=1}^{i-1} \ell_{ij} u_{jk} \quad \text{für } i = 1, \dots, n, \quad k = i, \dots, n, \\ \ell_{ki} &= \frac{1}{u_{ii}} \Big(a_{ki} - \sum_{j=1}^{i-1} \ell_{kj} u_{ji} \Big) \quad \text{für } i = 1, \dots, n, \quad k = i+1, \dots, n, \\ \ell_{ii} &= 1 \quad \text{für } i = 1, \dots, n, \end{aligned}$$

wie man leicht über die Formel für die Matrix-Matrix-Multiplikation zeigen kann. Alle übrigen Einträge von $L, U \in \mathbb{R}^{n\times n}$ sind Null. Testen Sie Ihren Code entsprechend!

Aufgabe 13.2. Die Determinante einer Matrix $A \in R^{n\times n}$ kann über die normalisierte LU-Zerlegung aus Aufgabe 13.1 berechnet werden. Es gilt namlich $\det(A) = \det(L)\det(U) = \det(U) = \prod_{j=1}^{n} u_{jj}$. Erweitern Sie die Klasse `SquareMatrix` um eine Methode `det`, die die Determinante über die LU-Zerlegung berechnet und zurückgibt. Die Matrix selbst soll hierbei nicht überschrieben werden. Testen Sie Ihren Code entsprechend!

Aufgabe 13.3. Erweitern Sie die Klasse `SquareMatrix` um eine Methode `solve`, welche die Lösung eines Gleichungsystem der Form $Ax = b$ mit Hilfe der LU-Zerlegung folgendermaßen berechnet. Für die Matrix $A = LU$ löst man zuerst $Ly = b$ und anschließend $Ux = y$. Gleichungssysteme mit Dreiecksmatrizen können analog zu Aufgabe 9.6 und Aufgabe 12.4 gelöst werden. Testen Sie Ihren Code an einem geeigneten Beispiel.

Aufgabe 13.4. Welchen Aufwand besitzt die LU-Zerlegung aus Aufgabe 13.1? Schreiben Sie das Ergebnis in der $\mathcal{O}$-Notation auf und erklären Sie wie sie auf das Ergebnis gekommen sind.

Aufgabe 13.5. Welchen Aufwand hat das Lösen eines linearen Gleichungssystems aus Aufgabe 13.3? Schreiben Sie das Ergebnis in der $\mathcal{O}$-Notation auf und erklären Sie wie sie auf das Ergebnis gekommen sind.

Aufgabe 13.6. Leiten Sie von der Klasse `SquareMatrix` die Klasse `DiagonalMatrix` ab. Speichern Sie nur die Einträge auf der Diagonale. Implementieren Sie Konstruktoren, Type-Cast und den Koeffizientenzugriff. Für jene Einträge A_{ij} mit $i \neq j$ gehen Sie vor wie in der Klasse `LowerTriangularMatrix` aus der Vorlesung: Speichern Sie `double zero` und `double const_zero` auf die sie beim Koeffizientenzugriff dann zugreifen. Testen Sie Ihren Code entsprechend!

Aufgabe 13.7. Redefinieren Sie die Methoden `det` und `solve` aus Aufgabe 13.2 bzw. Aufgabe 13.3 sodass Sie die Determinante von Diagonalmatrizen berechnen können bzw. lineare Gleichungssysteme $Ax = b$ lösen können. Nützen Sie dabei die diagonale Struktur der Matrix aus. Welchen Aufwand hat das Lösen dann? Schreiben Sie das Ergebnis in der $\mathcal{O}$-Notation. Erklären Sie, wie Sie auf das Ergebnis gekommen sind! Vergleichen Sie Ihr Ergebnis mit dem Ergebnis aus Aufgabe 13.5! Testen Sie Ihren Code entsprechend!

Aufgabe 13.8. Wie lautet die Ausgabe des folgenden Programms? Erklären Sie warum!

```
#include <iostream>
using std::cout ;
using std::endl ;

class Basisklasse {
protected :
  int N;
public :
  Basisklasse( int n = 0) {
   N = n;
   cout << " Konstr . Basisklasse , N = " << N << endl;
  }
  virtual ~Basisklasse() {
   cout << " Destr . Basisklasse , N = " << N << endl;
  }
  virtual void print() {
   klasse();
   cout << " N = " << N << endl;
  }
  void Add() {
  }
  virtual void klasse() const {
   cout << " In Basisklasse aber virtual ,";
  }
  void klasse() {
   cout << " In Basisklasse ,";
  }
};


class Abgeleitet : public Basisklasse {
public :
  Abgeleitet( int n = 0) {
   N = n;
   cout << " Konstr . Abgleitet , N = " << N << endl;
  }
  ~Abgeleitet() {
   cout << " Destr . Abgeleitet , N = " << N << endl;
  }
  void print() const {
   klasse();
   cout << " const N =" << N << endl;
  }
  void print() {
   klasse();
   cout << " N = " << N << endl;
  }
  void Add(){
```

```
   N = N + 100;
  }
  void klasse() const {
   cout << " In Abgeleitet fuer const , ";
  }
  void klasse() {
   cout << " In Abgeleitet ,";
  }
};


int main() {
  Basisklasse dp(1);
  Abgeleitet mr(10);
  Basisklasse * bs = & mr;
  {
   const Abgeleitet ah(200);
   dp.Add();
   mr.Add();
   bs->Add();
   ah.print();
  }
  dp.print();
  mr.print();
  bs->print();
  return 0;
}
```