Rheinisch-Westfälische Technische Hochschule
Institut für Geometrie und Praktische Mathematik

Mathematisches Praktikum (MaPra) — Sommersemester 2011

Prof. Dr. Wolfgang Dahmen — M.Sc. Mathieu Bachmann, Dipl.-Math. Jens Berger, M.Sc. Liang Zhang

Aufgabe 4

Bearbeitungszeit: zwei Wochen (bis Montag, 23. Mai 2011)

Testattermin: Donnerstag, der 26. Mai 2011

Mathematischer Hintergrund: Berechnung von Eigenwerten/-vektoren mit der Potenzmethode

Elemente von C++: Klassen, Überladen von Operatoren, dynamische Speicherverwaltung, Kommandozeilenargumente, Makefiles

20 Punkte

Aufgabenstellung

Schreiben Sie ein Programm, das zu einer gegebenen Matrix A den betragsgrößten Eigenwert samt Eigenvektor mit der Potenzmethode bestimmt. Vervollständigen Sie zu diesem Zweck die Vektorklasse, die Ihnen zur Verfügung gestellt wird, und verfassen Sie analog eine Matrixklasse.

Potenzmethode

Gegeben sei eine Matrix $A \in \mathbb{R}^{n\times n}$. Der betragsgrößte Eigenwert λ^* sei eindeutig bestimmt (er ist damit insbesondere reell) und habe die algebraische Vielfachheit 1. Der zugehörige Eigenvektor sei x^*, und es sei k ein beliebig gewählter Index, so dass $x^*_k \neq 0$ gilt. Da der Eigenvektor nur bis auf Vielfache bestimmt ist, kann man o. B. d. A. $x^*_k = 1$ annehmen. Man startet die Potenzmethode mit einem beliebigen Vektor $x^0 \in \mathbb{R}^n \setminus \{0\}$ und iteriert wie folgt:

$$\begin{aligned} \tilde{x}^i &\leftarrow Ax^{i-1}, \\ x^i &\leftarrow \tilde{x}^i/\tilde{x}^i_k. \end{aligned}$$

Die Vektoren x^i konvergieren nun für fast alle Startwerte x^0 (Beweisidee siehe unten) gegen den Eigenvektor x^*, und die Einträge $\tilde{x}^i_k$ konvergieren gegen λ^*. Weil man zu Beginn jedoch nicht weiß, für welche Werte von k die Komponente x^*_k nicht verschwindet, und um zu vermeiden, dass $\tilde{x}^i_k$ zu klein oder gar 0 wird und das Verfahren damit instabil wird oder abbricht, modifiziert man den Algorithmus noch leicht: Man startet mit $k = 1$ und wählt k jeweils neu, wenn $c \cdot |\tilde{x}^i_k| \leq \|\tilde{x}^i\|_\infty$ gilt, und zwar so, dass $|\tilde{x}^i_k| = \|\tilde{x}^i\|_\infty$ ist. Hierbei ist $c > 1$ eine (nicht zu große) Konstante, z. B. $c = 2$. Nach hinreichend vielen Iterationen ändert sich k schließlich nicht mehr, und man erhält Konvergenz wie oben.

Beweisidee

$\mathbb{R}^n$ lässt sich (eindeutig) in die direkte Summe $\mathbb{R}^n = \langle x^* \rangle \oplus V$ zerlegen, wobei V unter A invariant ist, d. h. $Av \in V$ für alle $v \in V$. Damit kann man x^0 eindeutig darstellen als $x^0 = \alpha x^* + v$ mit $\alpha \in \mathbb{R}$ und $v \in V$. Dann gilt $A^i x^0 = (\lambda^*)^i \alpha x^* + A^i v$ bzw.

$$\frac{A^i x^0}{(\lambda^*)^i} = \alpha x^* + \frac{A^i v}{(\lambda^*)^i}.$$

Weil die Eigenwerte von $A|_V$ vom Betrag kleiner als $|\lambda^*|$ sind, konvergiert $\frac{A^i v}{(\lambda^*)^i}$ für $i \to \infty$ gegen 0, der gesamte Ausdruck folglich gegen αx^*. Wenn $\alpha \neq 0$ ist (daher funktioniert der Algorithmus auch nur für fast alle x^0), sind die x^i lediglich beschränkte Vielfache dieses Ausdrucks. Damit konvergieren sie (weil die k–te Komponente auf 1 normiert ist) gegen ein Vielfaches von αx^*, was wieder ein Eigenvektor von A zum Eigenwert λ^* ist. Die Konvergenz der $\tilde{x}_k^i$ gegen λ^* ergibt sich dann sofort. [2, 3]

Vektor- und Matrixklasse

Um den Algorithmus möglichst übersichtlich gestalten zu können, sollen die Vektor- und Matrixoperationen in eigenen Klassen versteckt werden. Neben der Header–Datei `vektor.h` zu der Vektorklasse werden Ihnen einige Beispielfunktionen in `vektor.cpp` zur Verfügung gestellt. Ihre Aufgabe ist es, die Funktionen der Vektorklasse zu vervollständigen und analog eine Matrixklasse `Matrix` mit den entsprechenden Operationen zu entwickeln. Dazu sollten Sie zwei Dateien `matrix.h` und `matrix.cpp` anlegen. Einen Ausdruck der Header–Datei `vektor.h` finden Sie am Ende des Aufgabenblatts.

Anforderungen an die Matrixklasse

An Ihre Matrixklasse werden einige Anforderungen gestellt. Durch

```
Matrix A(m,n);
```

wird eine Matrix `A` mit `m` Zeilen und `n` Spalten angelegt und mit Nullen gefüllt. Um auch Felder von Matrizen anlegen zu können (dabei kann der Konstruktor nur ohne Parameter aufgerufen werden), sollten `m` und `n` den Default–Wert `1` bekommen. Mittels

```
A(i,j)
```

erfolgt der lesende und schreibende Zugriff auf das Matrixelement $\texttt{A}_{\texttt{ij}}$. Falls die Matrix das gewünschte Element nicht enthält, sollte mit einer Fehlermeldung abgebrochen werden. Die Dimension der Matrix sollte sich analog zur Elementfunktion `Laenge()` der Vektorklasse über die beiden Elementfunktionen `Zeilen()` und `Spalten()` auslesen lassen. Mit

```
A.ReDim(m,n);
```

wird die Matrix neu dimensioniert und wieder mit Nullen gefüllt. Der (private) Daten-Teil Ihrer Klasse könnte beispielsweise die folgende Form haben:

```
int    Zeil, Spalt;
double *Mat;
```

Der Konstruktor könnte dann mittels `new` ein entsprechend großes (eindimensionales) Feld reservieren, dessen Adresse Sie sich in `Mat` merken können. Ihre Funktionen müssen dafür sorgen, dass die Matrixelemente darin korrekt abgelegt werden. Achten Sie darauf, dass der Destruktor den Speicherplatz mit `delete[]` wieder freigibt. In `Zeil` und `Spalt` können Sie die Matrixdimension speichern. Neben den üblichen Operatoren für Matrizen sollte Ihre Matrixklasse auch entsprechende Operatoren für das Matrix-Vektor-Produkt zur Verfügung stellen.

Sicherheit der Vektor- und Matrixklassen

Ein sehr häufig auftretender Fehler besteht darin, dass sich der Programmierer bei den Zeilen- und Spaltenindizes irrt - besonders oft ist der angegebene Index um Eins zu gross oder zu klein. Besonders schwer zu finden ist ein solcher Fehler, wenn der Zugriff dann über die Grenzen eines Arrays hinaus erfolgt und fremde

Speicherinhalte falsch interpretiert oder sogar überschrieben werden. Wird dabei der dem Programm zugeteilte Speicherplatz überschritten, so bricht das Betriebssystem die Anwendung mit einer Schutzverletzung[1] ab. Wird der zugewiesene Speicher nicht überschritten, so kann es sein, dass das Programm weiterläuft, aber fehlerhafte Ergebnisse produziert.

Deshalb sollten Sie solchen Fehlern vorbeugen. Alle Zugriffe auf die Elemente des Arrays, das der Vektor- und Matrixklasse zur Datenspeicherung zugrude liegt, sollen ausschließlich in den Zugriffsoperatoren geschehen. Alle anderen Funktionen müssen die Zugriffsoperatoren benutzen, wenn sie Einträge lesen oder schreiben.

Schreiben Sie in den Zugriffsoperatoren eine Überprüfung, ob die Indizes im gültigen Bereich liegen und brechen Sie das Programm ab, falls dieser überschritten ist[2]. Um keine grossen Geschwindigkeitseinbußen zu erleiden, sollten Sie diese Überprüfung per Präprozessoranweisung `#ifdef` abschalten können, wenn Sie sicher sind, dass ihr Programm fehlerfrei läuft.

Test der Vektor- und Matrixklassen

Die Datei `test4.cpp` soll Ihnen bei der Fehlersuche in den Klassen helfen. Bevor Sie also zur Programmierung der Potenzmethode schreiten, sollten Ihre Klassen sämtliche darin enthaltenen Tests bestehen. Dazu müssen Sie lediglich aus `test4.cpp` und Ihren beiden Dateien `vektor.cpp` und `matrix.cpp` ein ausführbares Programm erzeugen und starten.

Schnittstellen

Durch die Header–Datei `unit4.h` werden Ihnen einige Variablen und Funktionen zur Verfügung gestellt. Wie üblich, müssen Sie zu Beginn die Funktion

```
void Start ( int Bsp, Matrix &A, Vektor &x0, double &eps );
```

aufrufen, wobei `Bsp` wieder im Bereich von `1` bis `AnzahlBeispiele` liegen darf. Zur Matrix `A` soll dann mit Hilfe der Potenzmethode der betragsgrößte Eigenwert samt Eigenvektor bestimmt werden. Als Startvektor verwenden Sie bitte den Vektor `x0`. Wenn $\|x^i - x^{i-1}\|_\infty \leq$ `eps` und $|\tilde{x}_k^i - \tilde{x}_k^{i-1}| \leq$ `eps` erfüllt sind, können Sie die Iteration abbrechen. Ihr Resultat übergeben Sie zusammen mit der Anzahl der benötigten Iterationen an die Funktion

```
void Ergebnis ( const Vektor &EigVek, double EigWert, long int
   Iterationen );
```

wo es bewertet und ausgegeben wird.

Kommandozeilenparameter

Die Nummer des zu rechnenden Beispiels soll diesmal nicht zur Laufzeit des Programms abgefragt, sondern als Kommandozeilenparameter übergeben werden. Hat das ausführbare Programm etwa den Namen `aufgabe4`, so soll z. B. mit dem Aufruf `aufgabe4 3` das dritte Beispiel gerechnet werden. Verwenden Sie dazu den Mechanismus von C++ zur Übergabe von Kommandozeilenparametern: Deklarieren Sie das Hauptprogramm als Funktion

```
int main ( int ArgCount, char *ArgValues[] );
```

[1] Fehlermeldung: `segmentation fault`

[2] *engl. range checking*

Die Zahl `ArgCount` gibt Ihnen dann die Zahl der Argumente an, wobei der Programmname als erstes Argument zählt. Im obigen Beispiel hat `ArgCount` daher den Wert `2`. Die Variable `ArgValues` ist ein Feld von Zeigern, so dass `ArgValues[i]` auf die Zeichenkette[3] im C-Format zeigt, die dem `i`-ten Kommandozeilenparameter entspricht. Also zeigt `ArgValues[0]` auf den String `"aufgabe4"` und `ArgValues[1]` auf den String `"3"`.

Zur Umwandlung von Zeichenketten in Integerwerte können Sie sogenannte *String-Streams* verwenden. Dazu müssen Sie die Standard-Header-Datei `sstream`[4] in Ihr Programm einbinden. Ist die Zeichenkette `s` beispielsweise definiert als `char s[10]`, dann können Sie mit der Deklaration

```
istringstream IStr(s);
```

einen Input-String-Stream `IStr` definieren, dessen Inhalt aus der Zeichenkette `s` besteht. Aus diesem können Sie dann Objekte verschiedenen Typs wie aus dem Eingabe-Stream `cin` auslesen. Achten Sie darauf, dass Ihr Programm auch auf fehlerhafte Eingaben sinnvoll reagiert.

Makefile

Das `Makefile` zur Erzeugung des Testprogramms für die Vektor- und Matrixklasse könnte zum Beispiel wie folgt beginnen (<*Tab*> steht für das Tab(ulator)-Zeichen):

```
CC      = g++
CFLAGS = -O2 -Wall

test4: vektor.o matrix.o test4.o
   <Tab> $(CC) $(CFLAGS) -o test4 vektor.o matrix.o test4.o

test4.o: vektor.h matrix.h test4.cpp
   <Tab> $(CC) $(CFLAGS) -c test4.cpp

vektor.o: vektor.h vektor.cpp
   <Tab> $(CC) $(CFLAGS) -c vektor.cpp

...
```

Zur Erzeugung von `test4` werden also die Dateien `vektor.o`, `matrix.o` und `test4.o` benötigt, und der Befehl dazu lautet

```
g++ -O2 -Wall -o test4 vektor.o matrix.o test4.o
```

Erstellen Sie ein vollständiges `Makefile`, damit es Ihre Programme erzeugt. Schreiben Sie insbesondere auch ein Ziel `clean`, das alle kompilierten Dateien löscht. Markieren Sie dieses Ziel als `.PHONY`. [1]

Literatur

[1] *Dokumentation zu GNU Make.* `http://www.gnu.org/software/make/`.

[2] Dahmen, W. und A. Reusken: *Numerische Mathematik für Ingenieure und Naturwissenschaftler.* Springer Verlag, Heidelberg, 2. Auflage, 2008.

[3] Golub, G. und C. van Loan: *Matrix Computations.* Johns Hopkins University Press, Baltimore, 2003.

[3] *engl. string*

[4] Vorsicht! Die Schreibweise `strstream` ist veraltet und sollte nicht mehr benutzt werden.

Header–Datei `vektor.h` zur Vektorklasse

```
#ifndef _VEKTOR_H                          // vektor.h nicht doppelt benutzen
#define _VEKTOR_H

#include <iostream>
using namespace std;

class Matrix;                              // fuer friend Matrix * Vektor etc.

class Vektor
{
  private:
    double* Vek;                           // Zeiger auf Feld fuer Vektorelemente
    int     Laeng;                         // Vektorlaenge

  public:
    Vektor  (const int i=1);               // Konstruktor mit Laengenangabe
    ~Vektor () { if(Vek) delete[] Vek; }   // Destruktor
    Vektor  (const Vektor&);               // Kopierkonstruktor

    double& operator () (const int);       // Zugriff auf Einträge des Vektors
    double  operator () (const int) const; // Zugriff falls Vektor const ist

    Vektor& operator =  (const Vektor&);   // Zuweisung
    Vektor& operator += (const Vektor&);   // Zuweisungen mit arithm. Operation
    Vektor& operator -= (const Vektor&);
    Vektor& operator *= (const double);
    Vektor& operator /= (const double);

    Vektor& ReDim   (const int);                // neue Laenge festlegen
    int     Laenge  () const { return Laeng; }  // Laenge
    double  Norm2   () const;                   // Euklidische Norm des Vektors
    double  NormMax () const;                   // Maximum-Norm des Vektors

    static void VekFehler (const char str[]);   // Fehlermeldung ausgeben

    friend Vektor   operator +  (const Vektor&, const Vektor&); // Addition
    friend Vektor   operator -  (const Vektor&, const Vektor&); // Subtraktion
    friend Vektor   operator -  (const Vektor&);                // Vorzeichen

    friend double   operator *  (const Vektor&, const Vektor&); // Skalarprodukt
    friend Vektor   operator *  (const double, const Vektor&);  // Vielfache
    friend Vektor   operator *  (const Vektor&, const double);
    friend Vektor   operator /  (const Vektor&, const double);

    friend bool     operator == (const Vektor&, const Vektor&); // Vergleich
    friend bool     operator != (const Vektor&, const Vektor&);

    friend istream& operator >> (istream&, Vektor&);            // Eingabe
    friend ostream& operator << (ostream&, const Vektor&);      // Ausgabe

    friend Vektor   operator *  (const Matrix&, const Vektor&); // Matrix-Vektor-
    friend Vektor   operator *  (const Vektor&, const Matrix&); // Multiplikation
};

#endif
```