Übungen zur Vorlesung

"Objektorientiertes Programmieren im Wissenschaftlichen Rechnen"

Dr. Olaf Ippisch, Ole Klein

Abgabe am 25. 06. 2013 in der Vorlesung

Die Übung am 18. 06. 2013 entfällt. Der Zettel ist erst nach zwei Wochen abzugeben und erhält die doppelte übliche Punktzahl.

Übung 1 STL-Containertypen

Sie haben in der Vorlesung die verschiedenen Containertypen der STL besprochen. Jeder Typ bietet gewisse Garantien und ist für unterschiedliche Verwendungszwecke prädestiniert.

Sie haben eine Liste von Szenarien, welchen Containertyp würden Sie verwenden und warum?

1. Sie schreiben eine Finite-Differenzen-Diskretisierung der Laplace-Gleichung. Sie haben hierzu ein strukturiertes Gitter. Zu jedem Gitterknoten wird die Lösung an diesem Punkt gespeichert. Alle Knoten des Gitters sind konsekutiv durchnummeriert und die Anzahl ist a-priori bekannt.

   Welcher Containertyp ist für die Speicherung der Knotenwerte gut geeignet?

2. Sie schreiben einen Quicksort. Um die rekursiven Funktionsaufrufe zu sparen möchten sie die Pivotelemente in einer Stack-Struktur speichern.

   Welcher Containertyp ist gut geeignet um die Pivotelemente zu speichern und welcher um die Daten selbst zu speichern?

3. Bei der Implementierung direkter Löser für dünnbesetzte Matrizen ist es üblich, zunächst die Bandbreite der Matrix zu minimieren, um den Speicherbedarf der resultierenden LU-Zerlegung zu reduzieren. Eine Methode hierzu ist der Cuthill-McKee Algorithmus. Im Laufe dieses Algorithmus müssen Elemente in einem FIFO (first-in, first-out) zwischengespeichert werden. Der FIFO behält hierbei ursprüngliche Nummerierung der Elemente bei.

   Welchen Containertyp würden Sie als FIFO einsetzen?

4. In Ihrem Finite-Differenzen-Programm wird die Lösung auf einem Teil des Gebietsrandes a-priori festgeschrieben. Knoten auf diesem Teil des Gebietsrandes (Dirichlet-Knoten) sind keine echten Freiheitsgrade und müssen beim Aufstellen der Matrix gesondert behandelt werden. Die Anzahl dieser Knoten ist klein im Vergleich zur Gesamtanzahl an Knoten.

   Sie möchten die Liste der Dirichlet-Knoten, sowie deren Lösungwerte, dynamisch aus einer Konfigurationsdatei lesen und knotenweise zugreifbar machen. Bei der Lösung linearer PDEs ist meistens der Speicher der limitierende Faktor, da man sehr große Probleme lösen möchte.

   In welchem Container würden Sie die Indizes der Dirichlet-Knoten und die Werte an diesen Knoten speichern?

12 Punkte

Übung 2 Matrixiteratoren

Bisher wurden ausschließlich dicht besetzte Matrizen behandelt. Im wissenschaftlichen Rechnen ist jedoch oft die Struktur der zugrundeliegenden Matrizen genauer bekannt. In der Regel hat man es mit dünnbesetzten Matrizen tun, manchmal sind es Bandmatrizen, oder obere Dreiecksmatrizen.

In solchen spezielleren Fällen möchte man diese Strukturinformationen ausnutzen und optimierte Datenstrukturen verwenden. Insbesondere heißt das, dass nicht mehr alle Matrixeinträge gespeichert werden, sondern nur jene, dir nicht a priori bekannt sind.

Um Algorithmen unabhängig von der verwendeten Matrixdatenstruktur schreiben zu können, und auch die spezielle Matrixstruktur auszunutzen, muss man zunächst die Zugriffsmethoden auf die Matrixdaten abstrahieren. Eine geeignete Abstraktion ist das Konzept der Iteratoren.

1. Erweitern sie die Template-Matrix-Klasse aus der letzten Übung um das Konzept der Iteratoren.
2. Wir führen zwei Arten von Iteratoren ein:

   Zeileniteratoren:

   ```
   class RowIterator {
       Row operator * ();
       const Row operator * () const;
       int row() const;
   };
   ```

   Spalteniteratoren:

   ```
   class ColIterator {
       T & operator * ();
       const T & operator * () const;
       unsigned int col() const;
   };
   ```

3. Von beiden Iteratortypen gibt es noch jeweils eine zweite Variante, den `Const{Row,Col}Iterator`.
4. Von der Matrix erhält man Zugriff auf die Zeileniteratoren, welche Zugriff auf eine einzelne Zeile bieten.
5. Von der Zeile erhält man Zugriff auf die Spalteniteratoren, welche wiederum Zugriff auf einzelne Einträge gewähren.
6. Die Matrix muss um die `begin()` und `end()` Methoden erweitert werden, welche einen `RowIterator` liefern. Denken sie auch an die `const` Varianten.
7. Die Klasse `Row` hat `begin()` und `end()` Methoden, die jeweils einen `ColIterator` liefern. Denken sie auch hier an die `const` Variante.
8. Implementieren sie alle hier skizzierten Klassen als *nested Class* der Matrix Klasse.

10 Punkte

ÜBUNG 3 ABSTRAKTE MATRIXALGORITHMEN: MATRIXNORM

Die in Aufgabe 2 implementierte abstrakte Matrixschnittstelle soll dazu genutzt werden eine abstrakten Algorithmus zu programmieren, welcher auch auf Matrizen mit anderen Datenstrukturen laufen würde.

Programmieren Sie eine Funktion

```
template <class M>
double frobeniusnorm(const M & matrix);
```

welche die Frobeniusnorm der Matrix berechnet.

Die Frobeniusnorm einer Matrix A wird berechnet als

$$\|A\|_{\mathrm{F}} = \sqrt{\sum_{i,j} |a_{ij}|^2},$$

wobei a_{ij} die Einträge der Matrix bezeichnet.

8 Punkte

Basierend auf der oben eingeführten Matrixschnittstelle wollen wir nun einen Algorithmus zur iterativen Lösung von Gleichungssystemen programmieren.

Der Gauß-Seidel-Algorithmus liefert für Matrizen, deren deren Spektraradius kleiner als 1 ist, eine näherungsweise Lösung des linearen Gleichungssystems.

Gegeben eine $n \times n$ Matrix A und eine rechte Seite b, suchen wir eine näherungsweise Lösung des Vektors x, so dass

$$Ax = b$$

erfüllt ist. Dies entspricht n Gleichungen eines linearen Gleichungssystems.

Um das System zu lösen, wird die k-te Gleichung nach x_k aufgelöst. Im $m+1$-ten Iterationsschritt ergibt somit folgende Gleichung:

$$x_k^{(m+1)} = \frac{1}{a_{k;k}} \left(b_k - \sum_{i=1}^{k-1} a_{k;i} \cdot x_i^{(m+1)} - \sum_{i=k+1}^{n} a_{k;i} \cdot x_i^{(m)} \right)$$

Implementieren Sie den Gauss-Seidel-Algorithmus basierend auf der abstrakten Iterator-Schnittstelle. Schreiben Sie Ihr Programm zunächst so, dass eine feste Anzahl an Iterationen durchgeführt wird. Das Interface soll folgendermaßen aussehen:

```
template<class M, class T>
void gaussseidel (const M& A, const std::vector<T> & b, std::vector<T> & x, int maxIter);
```

Hinweis: Testen sie ihr Programm an folgendem Beispiel:

$$\begin{pmatrix} 2 & 1 & 0 \\ 1 & 3 & 2 \\ 0 & 1 & -4 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ -0.5 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

10 Punkte

Wie immer gilt: Kommentieren Sie Ihr Programm. Erklären Sie was Sie tuen.