Ludwig-Maximilians-Universität München
Mathematisches Institut
Gregor Kleen

Programmieren II für Studierende der Mathematik

Blatt 7 – Lösungsvorschlag

Aufgabe 8 Eine symmetrische, positiv definite Matrix $A = (a_{ij})_{i,j=0,\dots,n-1} \in \mathbb{R}^{n\times n}$ ist darstellbar als Produkt $A = LL^T$ mit $L = (l_{ij})_{i,j=0,\dots,n-1}$ einer unteren linken Dreiecksmatrix mit positiven Hauptdiagonalelementen (Cholesky-Zerlegung). L kann berechnet werden wie folgt:

$$\left.\begin{aligned} l_{jj} &= \sqrt{a_{jj} - \textstyle\sum_{k=0}^{j-1} l_{jk}^2} \\ l_{ij} &= \tfrac{1}{l_{jj}} \left(a_{ij} - \textstyle\sum_{k=0}^{j-1} l_{ik} l_{jk}\right) \quad i = j+1, \dots, n-1 \end{aligned}\right\} \quad j = 0, \dots, n-1$$

Erstellen Sie eine Klasse `Cholesky` mit einem Konstruktor, der die Cholesky-Zerlegung für eine als Referenzarameter übergegebene Matrix durchführt und nur L als Attribut des erzeugten Objektes speichert. Der Konstruktor soll ein weiteres Argument ε akzeptieren (optional mit Voreinstellung $1 \cdot 10^{-10}$). Wenn bei der Berechnung von l_{jj} der Radikant betragsmäßig kleiner oder gleich $\varepsilon \cdot \mathrm{Spur}(A)$ mit $\mathrm{Spur}(A) = \sum_{i=0}^{n-1} a_{ii}$ ist, soll das Programm unter Ausgabe einer aussagekräftigen Fehlermeldung geeignet abgebrochen werden. Zudem soll der Konstruktor überprüfen ob die gegebene Matrix quadratisch ist und geeignet abbrechen, falls nicht.

Typaliase

```
using DVector = vector<double>;
using DMatrix = vector<vector<double>>;
```

Cholesky

```
class Cholesky {
  private:
    DMatrix L;

  public:
    Cholesky (const DMatrix& matrix, double eps = 1e-10)
      : L(matrix.size(), vector<double>(matrix.size())) {
      unsigned int n = L.size();
      for (unsigned int i = 0; i < n; i++)
        if (matrix[i].size() != n) {
          ostringstream sstr;
          sstr << "Matrix nicht quadratisch an Zeile " << i;
          throw invalid_argument{sstr.str()};
        }

      double trace = 0;
      for (unsigned int i = 0; i < n; i++)
        trace += matrix[i][i];

      for (unsigned int j = 0; j < n; j++) {
        L[j][j] = matrix[j][j];
        for (unsigned int k = 0; k < j; k++)
          L[j][j] -= L[j][k] * L[j][k];
```

```
        if (abs(L[j][j]) <= abs(eps * trace)) {
          ostringstream sstr;
          sstr << "Diagonalelement zu klein bei " << j;
          throw runtime_error{sstr.str()};
        }
        L[j][j] = sqrt(L[j][j]);

        for (unsigned int i = j + 1; i < n; i++) {
          L[i][j] = matrix[i][j];
          for (unsigned int k = 0; k < j; k++)
            L[i][j] -= L[j][k]*L[i][k];
          L[i][j] /= L[j][j];
        }
      }
    }

    getEntry

    Cholesky Ausgabe

    solve
};
```

Implementieren Sie eine Methode `getEntry` von `Cholesky` mit zwei Parametern i und j. Rückgabewert der Methode soll $a_{ij} = \sum_{k=0}^{\min(i,j)} l_{jk} l_{ik}$ als Wert vom Typ `double` sein.

getEntry

```
double getEntry(unsigned int i, unsigned int j) const {
  double res = 0;
  for (unsigned int k = 0; k <= min(i, j); k++) {
    res += L[j][k] * L[i][k];
  }
  return res;
}
```

Überladen Sie den Ausgabeoperator für `Cholesky` geeignet. Es soll die ursprüngliche Matrix A unter Verwendung der Methode `getEntry` ausgegeben werden.

Cholesky Ausgabe

```
friend ostream& operator<<(ostream& stream, const Cholesky& c) {
  for (unsigned int i = 0; i < c.L.size(); i++) {
    stream << "(";
    for (unsigned int j = 0; j < c.L.size(); j++)
      stream << setw(9) << c.getEntry(i, j);
    stream << ")";
    if (i + 1 < c.L.size())
      stream << endl;
  }
  return stream;
}
```

Implementieren Sie ein Hauptprogramm in dem zunächst die Dimension n gefolgt von den Einträgen einer $n \times n$-Matrix A von der Standardeingabe eingelesen wird. Es soll dann ein Objekt der Klasse `Cholesky` aus A

erzeugt werden (also die Cholesky-Zerlegung durchgeführt werden). Geben Sie das erstellte Objekt dann zur Kontrolle auf der Standardausgabe aus.

cholesky.cpp

```
#include <cmath>
#include <algorithm>
#include <iostream>
#include <iomanip>
#include <vector>
#include <stdexcept>

using namespace std;

Typaliase

Cholesky

Ausgabeoperator Vektor

int main() {
  unsigned int n;
  cout << "n: ";
  cin >> n;

  DMatrix A(n, vector<double>(n));
  for (unsigned i = 0; i < n; i++) {
    cout << "A[" << i << "][0.." << n - 1 << "]: ";
    for (double& x: A[i])
      cin >> x;
  }

  Einlesen Vektor

  cout << endl;

  Cholesky L{A};
  cout << L << endl
       << endl;
  Kontrollausgabe Vektor

  cout << endl;

  Aufruf solve
}
```

Ein lineares Gleichungssystem der Form $Ax = b$ kann bei gegebenem $b = (b_i)_{i=0,\dots,n-1}$ und gesuchtem $x = (x_i)_{i=0,\dots,n-1}$ mithilfe der Cholesky-Zerlegung von A effizient gelöst werden. Hierfür lösen wir die Gleichungssysteme $Ly = b$ mit $y = (y_i)_{i=0,\dots,n-1}$ und $L^T x = y$ wie folgt:

$$y_i = \frac{1}{l_{ii}} \left(b_i - \sum_{k=0}^{i-1} l_{ik} y_k \right) \qquad i = 0, \ldots, n-1$$
$$x_i = \frac{1}{l_{ii}} \left(y_i - \sum_{k=i+1}^{n-1} l_{ki} x_k \right) \qquad i = n-1, \ldots, 0$$

Implementieren Sie eine Methode `solve` von `Cholesky` die b als Referenzparameter akzeptiert und x liefert.

solve

```
DVector solve(const DVector& b) {
  DVector y(L.size()), x(L.size());
  for (unsigned int i = 0; i < L.size(); i++) {
    y[i] = b[i];
    for (unsigned int k = 0; k < i; k++)
      y[i] -= L[i][k] * y[k];
    y[i] /= L[i][i];
  }
  for (unsigned int m = 0; m < L.size(); m++) {
    const unsigned int i = L.size() - m - 1;
    x[i] = y[i];
    for (unsigned int k = i + 1; k < L.size(); k++)
      x[i] -= L[k][i] * x[k];
    x[i] /= L[i][i];
  }
  return x;
}
```

Erweitern Sie ihr Hauptprogramm darum, dass auch b eingelesen und zur Kontrolle wieder ausgegeben wird. Rufen Sie dann mit b die Methode `solve` auf dem Objekt der Klasse `Cholesky` auf und geben Sie das Ergebnis ebenfalls aus.

Einlesen Vektor

```
DVector b(n);
cout << "b[0.." << n - 1 << "]: ";
for (double& x: b)
  cin >> x;
```

Ausgabeoperator Vektor

```
ostream& operator<<(ostream& stream, const DVector& v) {
  stream << "(";
  for (unsigned int i = 0; i < v.size(); i++)
    stream << setw(9) << v[i];
  stream << ")";
  return stream;
}
```

Kontrollausgabe Vektor

```
cout << b << endl;
```

Aufruf solve

```
cout << L.solve(b) << endl;
```

Testen Sie Ihr Programm mit den folgenden Beispielen:

a) $A = \begin{pmatrix} 1 & 2 & 4 \\ 2 & 13 & 23 \\ 4 & 23 & 77 \end{pmatrix}$, $b = \begin{pmatrix} -3 \\ -21 \\ -73 \end{pmatrix}$

b) $A = \begin{pmatrix} 1 & 0 & 3 & 0 \\ 0 & 4 & 0 & 0 \\ 3 & 0 & 25 & 20 \\ 0 & 0 & 20 & 61 \end{pmatrix}$, $b = \begin{pmatrix} 10 \\ 8 \\ 158 \\ 304 \end{pmatrix}$

c) $A = \begin{pmatrix} 10 & 1 & 2 & 3 & 4 \\ 1 & 9 & -1 & 2 & -3 \\ 2 & -1 & 7 & 3 & -5 \\ 3 & 2 & 3 & 12 & 1 \\ 4 & -3 & -5 & 1 & 15 \end{pmatrix}$, $b = \begin{pmatrix} 12 \\ -8 \\ 1 \\ 3 \\ 4 \end{pmatrix}$

```
n: 3
A[0][0..2]: 1 2 4
A[1][0..2]: 2 13 23
A[2][0..2]: 4 23 77
b[0..2]: -3 -21 -73

(        1         2         4)
(        2        13        23)
(        4        23        77)

(       -3       -21       -73)

(        1         0        -1)
```

```
n: 4
A[0][0..3]: 1 0 3 0
A[1][0..3]: 0 4 0 0
A[2][0..3]: 3 0 25 20
A[3][0..3]: 0 0 20 61
b[0..3]: 10 8 158 304

(        1         0         3         0)
(        0         4         0         0)
(        3         0        25        20)
(        0         0        20        61)

(       10         8       158       304)

(        1         2         3         4)
```

```
n: 5
A[0][0..4]: 10 1 2 3 4
A[1][0..4]: 1 9 -1 2 -3
A[2][0..4]: 2 -1 7 3 -5
A[3][0..4]: 3 2 3 12 1
A[4][0..4]: 4 -3 -5 1 15
b[0..4]: 12 -8 1 3 4

(       10        1        2        3        4)
(        1        9       -1        2       -3)
(        2       -1        7        3       -5)
(        3        2        3       12        1)
(        4       -3       -5        1       15)

(       12       -8        1        3        4)

(  2.21842  -2.0232 -2.14001 0.691682   -1.489)
```