Implementation der naiven Simplexmethode

Horst Hollatz

1. September 2005

Zusammenfassung

Die C++-Klasse `lo_Simplex` ist eine einfache Implementation der Simplexmethode zum Lösen linearer Optimierungsaufgaben. Dabei darf die zu lösende Aufgabe in Normalform oder in Standardform vorliegen.

1. Problemstellung

Unter der Normalform einer linearen Optimierungsaufgabe soll eine Formulierung des Problems in der Form

$$\text{opt}\left\{ \sum_{j=1}^{n} c_j x_j \;\middle|\; \sum_{j=1}^{n} a_{ij} x_j \leqq b_i, i = 1,\ldots,m,\ x_j \geqq 0,\ j = 1,\ldots,n \right\}$$

verstanden sein; entsprechend ist

$$\text{opt}\left\{ \sum_{j=1}^{n} c_j x_j \;\middle|\; \sum_{j=1}^{n} a_{ij} x_j = b_i, i = 1,\ldots,m,\ x_j \geqq 0,\ j = 1,\ldots,n \right\}$$

die Standardform (opt $\in \{\max, \min\}$).

2. Implementation

In der Implementation wird ein reduziertes Rechenschema verwendet. Die implementierten Simplexmethoden erfordern keine zulässige Basislösung. Falls für die Normalform der Nullpunkt nicht zulässig, also mindestens eine rechte Seite negativ ist, wird in allen Restriktionen mit einer negativen rechten Seite eine Hilfsvariable eingeführt (in allen die gleiche) und die so erweiterte Aufgabe gelöst:

$$\text{opt}\left\{ \sum_{j=1}^{n} c_j x_j - N x_{n+1} \;\middle|\; \sum_{j=1}^{n} a_{ij} x_j - s x_{n+1} \leqq b_i, i = 1,\ldots,m,\ x_j \geqq 0,\ j = 1,\ldots,n \right\}.$$

Hierin ist $s = 1$, falls $b_i < 0$; sonst gilt $s = 0$. Die Zahl N stelle man sich als hinreichend groß vor; sie wird explizit nicht benötigt. Hat in der gefundenen optimalen Lösung die Hilfsvariable einen positiven Wert, ist der zulässige Bereich der ursprünglichen Aufgabe leer. Von diesem Trick merkt der Nutzer jedoch nichts.
Während die Aufgabe in Normalform mit einer primalen Simplexmethode gelöst wird, ist für die Aufgabe in Gleichungsform ein Primal-Dual-Algorithmus implementiert, bei dem die Gleichungsauflösung mit der Optimierung verknüpft ist. Während des Lösungsprozesses werden die den Gleichungen entsprechenden Spalten der Nichtbasisvariablen aus dem Rechenschema gestrichen.

Die Klasse lo_Simplex benötigt das System LS vom gleichen Autor. Die daraus verwendeten Klassen sind in der gz-Datei enthalten. Mit ihnen ist der Modellaufbau wesentlich beschleunigt.
Ein Beispielprogramm:

```
//: $CC -o beispiel3 beispiel3.cpp $OPTIONS -lm
#define ls_REAL float
#include "lo_Simplex.h"
int main()
{ ls_UINT m=4, n=3; ls_REAL zfwert;

// Aufbau der Zielfunktion c, der Restriktionsmatrix A
// und der rechten Seite b mit den LS-Klassen.
// Es geht um die Aufgabe:
// max{2x-3y+z | 2x+ y- z <= 5,  x- y- z <= -2,
//               -x+2y+3z <= 10, 2x-2y+3z <= 8, x,y,z >= 0 }

  ls_Matrix A(m,n); ls_Vector c(n), b(m), x(n);
  c[0]=2.,     c[1]=-3., c[2]=1.;
  A[0][0]=2.,  A[0][1]=1.,  A[0][2]=-1.,
  A[1][0]=1.,  A[1][1]=-1., A[1][2]=-1.,
  A[2][0]=-1., A[2][1]=2.,  A[2][2]=3.,
  A[3][0]=2.,  A[3][1]=-2., A[3][2]=3.;
  b[0]= 5., b[1]=-2., b[2]=10., b[3]=8.;
  lo_Simplex B(c,A,b);        // Modelldeklaration
  B.max_with_inequalities(zfwert, x)>>"";
                          // Optimierung:
                          // Rückkehrwert ist ein m-Vektor,
                          // der die Erfüllung der Restriktionen
                          // zeigt.
  x>>"";                  // Anzeigen der Lösung
  cout<<"optimal value: "<<zfwert<<endl;
  return 0;
}
```

Die zugehörige h-Datei liefert weitere Informationen.

```
#ifndef LO_SIMPLEX
#define LO_SIMPLEX
#include "lo_names.h"
#ifndef LS_MATRIX
#include LS_MATRIX_H // LS-System wird benötigt!
#endif

/*
Naive Simplexmethode (Ohne numerische Ansprüche!):
Man maximiere/minimiere (c,x) unter
Ungleichungsbedingungen Ax<=b, x>=0
oder unter Gleichungsbedingungen Ax=b,x>=0:
*/

class lo_Simplex
{ protected:
  ls_Matrix A, T;        // Problem-Matrix und Simplex-Tableau
  ls_Vector c, b;        // ZF- und rechte-Seiten-Vektor
```

```
  ls_array1<int> basis, nbasis;
                        // Indices der Basis-, Nichtbasisvar.
  char ONAME[ls_len];   // Objektname
  public:
  char *name, *A_name, *T_name, *c_name,
       *b_name, *basis_name, *nbasis_name;
                        // Zeiger auf Objektnamen
  ls_REAL eps;          // Genauigkeitsschranke
  int it;               // Anzahl der ausgeführten Iterationen
  lo_Simplex(char *nam="lo_Simplex");
                        // Standard-Konstruktor mit Objektnamen
  lo_Simplex(ls_Vector &cc, ls_Matrix &AA,
             ls_Vector &bb, char* ="lo_Simplex");
                        // Datenübernahme; danach sind
                        // die Objekte cc, AA, bb ohne Daten
  lo_Simplex(lo_Simplex &);
  const lo_Simplex& operator=(const lo_Simplex &);
  ls_Vector min_with_inequalities(ls_REAL &zfw, ls_Vector &x);
  ls_Vector max_with_inequalities(ls_REAL &zfw, ls_Vector &x);
                        // Modell in Ungleichungsform;
                        // auf x werden die optimale Lösung
                        // und auf zfw der optimale
                        // Zielfunktionswert abgeliefert;
                        // Rückkehrwert ist ein Vektor,
                        // der die Erfüllung der Restriktionen
                        // zeigt.
  lo_Simplex& min_with_equations(ls_REAL &, ls_Vector &x);
  lo_Simplex& max_with_equations(ls_REAL &, ls_Vector &x);
                        // Modell in Gleichungsform;
                        // es wird ein primal-dual-Algorithmus
                        // angewendet.
  ls_Vector balance(ls_REAL &zfw, const ls_Vector &x) const
  { ls_Vector r; r=b-A*x; zfw=c*x; return r;}
                        // Zu einem gegebenen Vektor werden die
                        // Erfüllung der Restriktionen und der
                        // Zielfunktionswert ausgegeben.
  const lo_Simplex& write(ostream &) const;
                        // Objektdaten-Ausgabe in eine Datei
  lo_Simplex& read(istream &);
                        // Objektdaten-Eingabe aus einer Datei
  const lo_Simplex& operator>>(char *) const;
                        // Ausgabe in eine namentliche Datei
  lo_Simplex& operator<<(char *);
                        // Eingabe aus einer namentl. Datei
};
#ifndef LO_LIB
#ifndef LO_SIMPLEXC
#include LO_SIMPLEX_C
#endif
#endif
#endif
```

Die hier verwendete Entwurfsstrategie entspricht der für die LS-Klassen. Es ist lediglich anstelle von ls_ bzw. LS_ hier lo_ bzw. LO_ zu ersetzen.