LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR STATISTIK
SONDERFORSCHUNGSBEREICH 386

Heumann, Fieger, Kastner:

C++ Utilities zur Implementierung statistischer Verfahren unter Berücksichtigung fehlender Werte

Sonderforschungsbereich 386, Paper 109 (1998)

Online unter: http://epub.ub.uni-muenchen.de/

Projektpartner

gsf

MAX-PLANCK-GESELLSCHAFT

C++ Utilities zur Implementierung statistischer Verfahren unter Berücksichtigung fehlender Werte

C. Heumann, A. Fieger, C. Kastner

Institut für Statistik
Akademiestr. 1
80799 München

April 1998

Abstract

Die hier vorgestellten Erweiterungen der bereits bestehenden generischen Bibliothek zur linearen Algebra (Fieger, Heumann, Kastner und Watzka, 1997) stellen Funktionen bereit, die bei der Implementierung statistischer Verfahren benötigt wird. Besondere Beachtung findet der Umgang mit fehlenden Daten.

Inhaltsverzeichnis

1 Einleitung . . . 2
2 Statistische Funktionen für Matrizen . . . 3
3 Listen . . . 7
4 Zufallszahlen . . . 8
5 Verteilungen etc. . . . 9
5.1 Verteilungsfunktionen . . . 9
5.2 Funktionen für p-values . . . 10
6 Weitere Utilities . . . 11
6.1 Binomialkoeffizient . . . 11
6.2 Kombinatorik . . . 12
6.3 Sortieren . . . 12
7 Beispiele . . . 13

1 Einleitung

In diesem Bericht werden die Datentypen und Templatefunktionen beschrieben, die auf den in Fieger, Heumann, Kastner und Watzka (1997) vorgestellten Datentypen basieren und diese erweitern. Diese Erweiterungen sollen zur einfachen Implementierung statistischer Verfahren beitragen, wobei dem Aspekt fehlender Werte besondere Aufmerksamkeit geschenkt wird.

arraytyp.h definiert einige oft benötigte Datentypen für Matrizen, Arrays und Listen.

```
typedef DataMatrix              matrix;
typedef PreMatrix<int>          intMatrix;
typedef PreMatrix<unsigned>     unsignedMatrix;

typedef Array<real>             realArray;
typedef Array<int>              intArray;
typedef Array<unsigned>         unsignedArray;
typedef Array<matrix>           matrixArray;
typedef Array<intMatrix>        intMatArray;
typedef Array<unsignedArray>    ArrayOfUnsignedArray;

typedef Array2D<real>           realArray2D;
typedef Array2D<int>            intArray2D;
typedef Array2D<unsigned>       unsignedArray2D;
typedef Array2D<matrix>         matrixArray2D;
typedef Array2D<intMatrix>      intMatrixArray2D;
```

```
typedef Array<unsignedArray2D>  unsignedArrayOfArray2D;

typedef LinkedList<real>        realList;
typedef LinkedList<int>         intList;
typedef LinkedList<unsigned>    unsignedList;
typedef LinkedList<matrix>      matrixList;
typedef LinkedList<intMatrix>   intMatrixList;
typedef LongList<real>          realLongList;
typedef LongList<int>           intLongList;
typedef LongList<unsigned>      unsignedLongList;
typedef LongList<matrix>        matrixLongList;
typedef LongList<intMatrix>     intMatrixLongList;
```

utils.h liest alle in diesem Bericht beschriebenen Header-Dateien ein. Dadurch steht mittels Einbinden von `utils.h` auf einfache Weise die gesamte hier beschriebene Funktionalität zur Verfügung. Das Programmprojekt wird dadurch natürlich nicht gerade kleiner, so daß es sich für kleine Projekte anbietet, nur die benötigten Definitionen explizit einzubinden (siehe Quelltext von `utils.h` für Beispiele und Dokumentation).

bool.h.hide Die Headerdatei bool.h wird z. B. vom GNU C++ Compiler nicht benötigt (darum `.hide`), für andere Compiler, die den Typ `bool` nicht kennen wird diese benötigt, d. h. sie ist in `bool.h` umzubenennen.

2 Statistische Funktionen für Matrizen

In `datamat.h` werden einfache (real) Matrizen auf Datenmatrizen erweitert. Die Erweiterungen bestehen im wesentlichen aus zwei Teilen, der Möglichkeit mit fehlenden Daten umgehen zu können und der Einführung elementarer statistischer Funktionen wie Mittelwerte oder Standardabweichungen. Mit den Definitionen in `arraytyp.h` kann dann anstelle von `DataMatrix` kurz `matrix` verwendet werden.

Bei den Rückgabewerten der statistischen Funktionen ist zu beachten, daß der Ergebnisvektor stets ein Spaltenvektor ist!

NAME `colmeans()` – Spaltenweise Mittelwerte

SYNOPSIS

```
DataMatrix DataMatrix::colmeans();
```

BESCHREIBUNG `colmeans()` berechnet spaltenweise Mittelwerte bei Datenmatrizen

---

NAME `colmeans()` – Spaltenweise Mittelwerte

SYNOPSIS

```
DataMatrix DataMatrix::colmeans(const real miss);
```

BESCHREIBUNG `colmeans()` berechnet spaltenweise Mittelwerte bei Datenmatrizen mit fehlenden Werten

`miss` beliebiger Wert, der als Indikator für fehlende Werte dient

BEMERKUNGEN Falls alle Werte in einer Spalte fehlen, ist das jeweilige Ergebnis NaN

NAME `colmeans()` – Spaltenweise Mittelwerte

SYNOPSIS

```
DataMatrix DataMatrix::colmeans(const realMatrix& R);
```

BESCHREIBUNG `colmeans()` berechnet spaltenweise Mittelwerte bei Datenmatrizen mit fehlenden Werten

`R` Indikatormatrix für fehlende Werte

BEMERKUNGEN Falls alle Werte in einer Spalte fehlen, ist das jeweilige Ergebnis NaN

NAME `colstddevs()` – Spaltenweise Standardabweichungen

SYNOPSIS

```
DataMatrix DataMatrix::colstddevs();
```

BESCHREIBUNG `colstddevs()` berechnet spaltenweise Standardabweichungen bei Datenmatrizen

BEMERKUNGEN Teilung durch $n - 1$.

NAME `colstddevs()` – Spaltenweise Standardabweichungen

SYNOPSIS

```
DataMatrix DataMatrix::colstddevs(const real miss);
```

BESCHREIBUNG `colstdddevs()` berechnet spaltenweise Standardabweichungen bei Datenmatrizen mit fehlenden Werten

`miss` beliebiger Wert, der als Indikator für fehlende Werte dient

BEMERKUNGEN Teilung durch $n-1$. Falls alle Werte in einer Spalte fehlen, ist das jeweilige Ergebnis NaN

NAME `colstddevs()` – Spaltenweise Standardabweichungen

SYNOPSIS

```
DataMatrix DataMatrix::colmeans(const realMatrix& R);
```

BESCHREIBUNG `colstdddevs()` berechnet spaltenweise Standardabweichungen bei Datenmatrizen mit fehlenden Werten

`R` Indikatormatrix für fehlende Werte

BEMERKUNGEN Teilung durch $n-1$. Falls alle Werte in einer Spalte fehlen, ist das jeweilige Ergebnis NaN

NAME `colsums()` – Spaltensummen

SYNOPSIS

```
DataMatrix DataMatrix::colsums();
```

BESCHREIBUNG `colsums()` berechnet Spaltensummen bei Datenmatrizen

NAME `colsums()` – Spaltensummen

SYNOPSIS

```
DataMatrix DataMatrix::colsums(const real miss);
```

BESCHREIBUNG `colsums()` berechnet Spaltensummen bei Datenmatrizen mit fehlenden Werten

`miss` beliebiger Wert, der als Indikator für fehlende Werte dient

BEMERKUNGEN Falls alle Werte in einer Spalte fehlen, ist das jeweilige Ergebnis 0

NAME `colsums()` – Spaltensummen

SYNOPSIS

```
DataMatrix DataMatrix::colsums(const realMatrix& R);
```

BESCHREIBUNG `colsums()` berechnet Spaltensummen bei Datenmatrizen

`R` Indikatormatrix für fehlende Werte

BEMERKUNGEN Falls alle Werte in einer Spalte fehlen, ist das jeweilige Ergebnis 0

NAME `RIndMat()` – Indikatormatrix für fehlende Werte

SYNOPSIS

```
DataMatrix DataMatrix::RIndMat(const real miss);
```

BESCHREIBUNG `RIndMat()` liefert eine Indikatormatrix für fehlende Werte

`miss` beliebiger Wert, der als Indikator für fehlende Werte dient

BEMERKUNGEN $R_{ij} = 1$ falls X_{ij} beobachtet, d. h. nicht gleich miss ist, 0 sonst

---

NAME `rowmeans()` – Zeilenweise Mittelwerte

SYNOPSIS

```
DataMatrix DataMatrix::rowmeans();
```

BESCHREIBUNG `rowmeans()` berechnet zeilenweise Mittelwerte bei Datenmatrizen

BEMERKUNGEN Falls alle Werte in einer Zeile fehlen, ist das jeweilige Ergebnis NaN (not a number)

NAME `rowmeans()` – Zeilenweise Mittelwerte

SYNOPSIS

```
DataMatrix DataMatrix::rowmeans(const real miss);
```

BESCHREIBUNG `rowmeans()` berechnet zeilenweise Mittelwerte bei Datenmatrizen mit fehlenden Werten

`miss` beliebiger Wert, der als Indikator für fehlende Werte dient

BEMERKUNGEN Falls alle Werte in einer Zeile fehlen, ist das jeweilige Ergebnis NaN

NAME `rowmeans()` – Zeilenweise Mittelwerte

SYNOPSIS

```
DataMatrix DataMatrix::rowmeans(const realMatrix& R);
```

BESCHREIBUNG `rowmeans()` berechnet zeilenweise Mittelwerte bei Datenmatrizen mit fehlenden Werten

`R` Indikatormatrix für fehlende Werte

BEMERKUNGEN Falls alle Werte in einer Zeile fehlen, ist das jeweilige Ergebnis NaN

NAME `rowstddevs()` – Zeilenwesie Standardabweichungen

SYNOPSIS

```
DataMatrix DataMatrix::rowstddevs();
```

BESCHREIBUNG `rowstddevs()` berechnet zeilenweise Standardabweichungen bei Datenmatrizen

BEMERKUNGEN Teilung durch $n - 1$

NAME `rowstddevs()` – Zeilenweise Standardabweichungen

SYNOPSIS

```
DataMatrix DataMatrix::rowstddevs(const real miss);
```

BESCHREIBUNG `rowstddevs()` berechnet zeilenweise Standardabweichungen bei Datenmatrizen mit fehlenden Werten

`miss` beliebiger Wert, der als Indikator für fehlende Werte dient

BEMERKUNGEN Teilung durch $n - 1$. Falls alle Werte in einer Zeile fehlen, ist das jeweilige Ergebnis NaN

NAME `rowstddevs()` – Zeilenweise Standardabweichungen

SYNOPSIS

```
DataMatrix DataMatrix::rowstddevs(const realMatrix& R);
```

BESCHREIBUNG `rowstddevs()` berechnet zeilenweise Standardabweichungen bei Datenmatrizen mit fehlenden Werten

`R` Indikatormatrix für fehlende Werte

BEMERKUNGEN Teilung durch $n - 1$. Falls alle Werte in einer Zeile fehlen, ist das jeweilige Ergebnis NaN

NAME `rowsums()` – Zeilensummen

SYNOPSIS

```
DataMatrix DataMatrix::rowsums();
```

BESCHREIBUNG `rowsums()` berechnet Zeilensummen bei Datenmatrizen

NAME `rowsums()` – Zeilensummen

SYNOPSIS

```
DataMatrix DataMatrix::rowsums(const real miss);
```

BESCHREIBUNG `rowsums()` berechnet Zeilensummen bei Datenmatrizen

`miss` beliebiger Wert, der als Indikator für fehlende Werte dient

BEMERKUNGEN Falls alle Werte in einer Zeile fehlen, ist das jeweilige Ergebnis 0

NAME `rowsums()` – Zeilensummen

SYNOPSIS

```
DataMatrix DataMatrix::rowsums(const realMatrix& R);
```

BESCHREIBUNG `rowsums()` berechnet Zeilensummen bei Datenmatrizen

`R` Indikatormatrix für fehlende Werte

BEMERKUNGEN Falls alle Werte in einer Zeile fehlen, ist das jeweilige Ergebnis 0

NAME `numCompleteCases()` – Vollständige Fälle zählen

SYNOPSIS

```
unsigned DataMatrix::rowsums(const real miss);
```

BESCHREIBUNG `numCompleteCases()` bestimmt die Anzahl der vollständig beobachteten Fälle, d. h. die Anzahl der Zeilen ohne Werte gleich miss

`miss` beliebiger Wert, der als Indikator für fehlende Werte dient

---

NAME `numCompleteCases()` – Vollständige Fälle zählen

SYNOPSIS

```
unsigned DataMatrix::rowsums(const realMatrix& R);
```

BESCHREIBUNG `numCompleteCases()` bestimmt die Anzahl der vollständig beobachteten Fälle
`R` Indikatormatrix für fehlende Werte

---

3 Listen

In der Headerdatei `tlonglst.h` wird eine einfache verkettete Liste definiert. Eine komplexere Implementierung einer Liste, die hier aber nicht besprochen werden soll, kann in `tlinklst.h` gefunden werden. Das Arbeiten mit einer `LongList` besteht immer aus zwei Teilen. Der Liste selbst und einem 'zur Liste gehörigen' `LongListIterator` (die Verknüpfung ist allerdings genau in umgekehrter Richtung). Elemente werden in die Liste eingefügt und über den Iterator angesprochen.

NAME `insert()` – Einfügen in die Liste

SYNOPSIS

```
int LongList::insert( const T& v );
```

BESCHREIBUNG `insert()` fügt ein neues Element in die Liste ein. Der Rückgabewert ist 1, falls das Element erfolgreich eingefügt werden konnte. Tritt ein Fehler auf ist der Rückgabewert 0.

WARNUNGEN Das Element muß einen Copy-Konstruktor besitzen

NAME `size()` – Länge einer Liste

SYNOPSIS

```
unsigned LongList::size();
```

BESCHREIBUNG `size()` liefert die Länge einer Liste zurück

BEMERKUNGEN `len()` liefert das gleiche Ergebnis

NAME `LongListIterator()` – Konstruktor

SYNOPSIS

```
LongListIterator::LongListIterator( LongList<T> &list );
```

BESCHREIBUNG `()` Konstruktor, der die Verbindung des Iterators zu einer Liste herstellt
`list` Liste mit der der Iterator verknüpft werden soll

NAME `()` – Zugriffsoperator

SYNOPSIS

```
T LongListIterator::operator()( void );
```

BESCHREIBUNG `()` getattet den lesenden und schreibenden Zugriff auf das aktuelle Element der mit dem Iterator verknüpften Liste

---

NAME `++` – Inkrementoperator

SYNOPSIS

```
void LongListIterator::operator++( void );
```

BESCHREIBUNG `++` Setzt den Iterator auf das nächste Element der Liste, mit der er verknüpft ist

---

NAME `reset()` – Zugriffsoperator

SYNOPSIS

```
void LongListIterator::reset();
```

BESCHREIBUNG `reset()` Setzt den Iterator auf das erste Element der Liste mit der er verknüpft ist

---

4 Zufallszahlen

In der Headerdatei `agrandom.h` werden zwei Zufallszahlengenerator definiert. Die Initialisierung von `ag_Time_RNG` basiert auf der Systemzeit, die automatisch ermittelt wird. Der Generator `ag_Seed_RNG` muß vom Benutzer selbst mit einem Startwert (Seed) initialisiert werden. Durch die Explizite Angabe des Startwerts besteht die Möglichkeit mehrere Wiederholungen der selben Folge von Zufallszahlen zu erhalten.

Die einzelnen Zufallszahlentypen sind in Fieger, Heumann, Kastner und Watzka (1997) beschrieben (siehe Abschnitt 4, Erzeugen von Zufallszahlen).

5 Verteilungen etc.

In diesem Abschnitt werden einige Funktionen zur Berechnung von Verteilungen und 'p-values' vorgestellt. Hier ist bei weitem keine volle Funktionalität vorhanden, die kommentierten Quelltexte (`distribs.cc`) sollten jedoch als Basis für einfache eigene Erweiterungen dienen können. Die zugrundeliegende Basis sind die Funktionen in `cdflib.h` (cdflib von Barry W. Brown und James Lovato, siehe StatLib Server `http://www.stat.cmu.edu/gereral/cdflib`).

5.1 Verteilungsfunktionen

In der Headerdatei `normvert.h` werden die Verteilungsfunktion und die inverse Verteilungsfunktion der Standardnormalverteilung bereitgestellt. Andere Verteilungsfunktionen werden hier nicht bereitgestellt. Sie können, wie oben erwähnt, auf `cdflib.h` basierend selbst leicht implementiert werden.

NAME `invphi()` – Inverse Normalverteilungsfunktion

SYNOPSIS

```
real invphi(real x);
```

BESCHREIBUNG `invphi()` Inverse der Verteilungsfunktion der Standardnormalverteilung.

`x` Stelle, an der ausgewertet werden soll

BEMERKUNGEN `invphi(x)` $= \Phi^{-1}(x)$

NAME `phi()` – Normalverteilungsfunktion

SYNOPSIS

```
real phi(real x);
```

BESCHREIBUNG `phi()` Verteilungsfunktion der Standardnormalverteilung.

`x` Stelle, an der ausgewertet werden soll

BEMERKUNGEN `phi(x)` $= \Phi(x)$

5.2 Funktionen für p-values

In der Headerdatei `distribs.h` werden Funktionen für einseitige p-values bestimmter Verteilungen definiert. Diese Verteilungen sind die Beta-Verteilung, die Binomial-Verteilung, die χ^2-Verteilung, die F-Verteilung, die Standardnormal-Verteilung, die Poisson-Verteilung und die t-Verteilung.

Die Funktion `p_value_xx(`⟨*Parameter für xx*⟩`, TG)` liefert den Wert α gemäß

$$P(X \geq \texttt{TG}) = \alpha\,,$$

Mit `xx` ist die jeweilige Funktion (`beta`, `binomial`, ...) zu bezeichnen, deren Parameter an der Stelle ⟨*Parameter für xx*⟩ angegeben werden.

NAME `p_value_beta()` – Beta-Verteilung

SYNOPSIS

```
real p_value_beta(const real a, const real b, const real TG );
```

BESCHREIBUNG `p_value_beta()` liefert den p-value einer betaverteilten Zufallsgröße

`a` erster Parameter der Beta(a, b)-Verteilung

`b` zweiter Parameter der Beta(a, b)-Verteilung

`TG` Testgröße

NAME `p_value_binomial()` – Binomialverteilung

SYNOPSIS

```
real p_value_binomial( const unsigned N,
       const real Prob, const unsigned TG );
```

BESCHREIBUNG `p_value_binomial()` liefert den p-value einer binomialverteilten Zufallsgröße

`N` Stichprobenumfang
`Prob` Wahrscheinlichkeit für Erfolg bei jedem Bernoulli-Experiment
`TG` Testgröße

NAME `p_value_chisquare()` – χ^2-Verteilung

SYNOPSIS

```
real p_value_chisquare( const unsigned df, const real TG );
```

BESCHREIBUNG `real p_value_chisquare()` liefert den p-value einer χ^2-verteilten Zufallsgröße
`df` Freiheitsgrade
`TG` Testgröße

NAME `p_value_f()` – F-Verteilung

SYNOPSIS

```
real p_value_f( const unsigned dfn,
                const unsigned dfd, const real TG );
```

BESCHREIBUNG `p_value_f()` liefert den p-value einer F-verteilten Zufallsgröße
`dfn` Freiheitsgrade Zähler (nominator)
`dfd` Freiheitsgrade Nenner (denominator)
`TG` Testgröße

NAME `p_value_standardnormal()` – Standardnormalverteilung

SYNOPSIS

```
real p_value_standardnormal( const real TG );
```

BESCHREIBUNG `p_value_standardnormal()` liefert den p-value einer standardnormalverteilten Zufallsgröße
`TG` Testgröße

BEMERKUNGEN Die Funktion `invphi(TG)` liefert das gleiche Resultat

NAME `p_value_poisson()` – Poisson-Verteilung

SYNOPSIS

```
real p_value_poisson( const real lambda, const unsigned TG );
```

BESCHREIBUNG `p_value_poisson()` liefert den p-value einer poissonverteilten Zufallsgröße
`lambda` Parameter der Poissonverteilung
`TG` Testgröße

NAME `p_value_t()` – Students t-Verteilung

SYNOPSIS

```
real p_value_t( const unsigned df, const real TG );
```

BESCHREIBUNG `p_value_t()` liefert den p-value einer t-verteilten Zufallsgröße
`df` Freiheitsgrade
`TG` Testgröße

---

6 Weitere Utilities

6.1 Binomialkoeffizient

In der Headerdatei `bincoeff.h` wird eine Funktion für den Binomialkoeffizienten $\binom{n}{k}$ definiert.

NAME `NoverK()` – Binomialkoeffizient

SYNOPSIS

```
unsigned NoverK(unsigned n, unsigned k);
```

BESCHREIBUNG `NoverK()` Binomialkoeffizient $\binom{n}{k}$.
`n` natürliche Zahl
`k` natürliche Zahl

6.2 Kombinatorik

In der Headerdatei `combinat.h` werden Funktionen zur Kombinatorik bereitgestellt.

NAME `Enumerate_NoverK_Set()` – Auflistung von Teilmengen

SYNOPSIS

```
unsignedArray2D Enumerate_NoverK_Set
   ( const unsigned N, const unsigned K, const bool AddOne );
```

BESCHREIBUNG `Enumerate_NoverK_Set()` Auflistung von Teilmengen. N-faches Ziehen ohne Zurücklegen aus einer Urne mit K verschiedenen Kugeln.

`N` Anzahl der Ziehngen

`K` Anzahl der Kategorien, bezeichnet mit $0, \ldots, K-1$

`AddOne` Falls true, Bezeichnug der Kategorien mit $1, \ldots, K$

BEMERKUNGEN Beachte $N < K$

NAME `Enumerate_Set_Combinations()` – Auflistung von Teilmengen

SYNOPSIS

```
unsignedArray2D Enumerate_Set_Combinations
   ( const unsignedArray &SetSizes, const bool AddOne );
```

BESCHREIBUNG `Enumerate_Set_Combinations()` Auflistung von Teilmengen. Ziehen aus verschiedenen Urnen verschiedener Umfänge n_i (jeweils ein Mal je Urne i)

`SetSizes` Array mit den Umfängen n_i der einzelnen Urnen

`AddOne` Falls true, Bezeichnug der Kategorien mit $1, \ldots, K_i$

6.3 Sortieren

In der Headerdatei `qsort.h` wird ein Quick-Sort Algorithmus für Arrays (`Array<T>`) definiert.

NAME `quicksort()` – Quicksort

SYNOPSIS

```
void quicksort(  Array<T>& vector );
```

BESCHREIBUNG `quicksort()` sortiert einen Array in aufsteigender Ordnung

---

7 Beispiele

Anwendung von statistischen Funktionen; die Rückgabe erfolgt in einem Spaltenvektor:

```
/* spaltenweise Mittelwerte einer bereits
   bestehenden Matrix X berechnen */
matrix bar_X = X.colmeans();

/* Mittelwerte ausgeben */
for (int i=0; i<X.rows(); i++) {
  cout << bar_X.get(i,0) << endl;
}
```

Erzeugen eines Störvektors $\epsilon \sim N(0,1)$ mit Zufallszahlen:

```
const unsigned int n = 100;
matrix epsilon(n,1);
ag_Time_RNG<real> epstime;

for (int i=0; i<n; i++ ) {
  epsilon.put(i,0, epstime.normal(0,1));
}

// selbes mit definiertem Startwert,
// d.h. wiederholbar
epsseed = ag_seed_RNG<real>(12345);

for (int i=0; i<n; i++ ) {
  epsilon.put(i,0, epsseed.normal(0,1));
```

```
}
// noch einmal ...
epsseed = ag_seed_RNG<real>(12345);
// usw.
```

Testen von Hypothesen; Verwenden von p-values:

```
const real Niveau = 0.05;
const unsigned int Freiheitsgrade = 99;

/* vergleichen einer bereits berechneten
   Testgröße TG mit zugehörigem p-value */

if ( real p_value_t( Freiheitsgrade, TG ) < Niveau ) {
  cout << "H_0 ablehnen" << endl;
} else {
  cout << "H_0 nicht ablehnen" << endl;
}
```

Literaturverzeichnis

Brown, B. W. und Lovato, J. (1993). CDFLIB, library of fortran routines for cumulative distribution functions, inverses, and other parameters, `http://www.stat.cmu.edu/general/cdflib`. (StatLib).

Fieger, A., Heumann, C., Kastner, C. und Watzka, K. (1997). Generische Bibliothek zur Linearen Algebra und zur Simulation in C++, *SFB386 – Discussion Paper 63*, Ludwig-Maximilians-Universität München.