Fieger:

C++ Klassen zur Linearen Regression bei fehlenden Kovariablen

Sonderforschungsbereich 386, Paper 61 (1997)

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

Projektpartner

C++ Klassen zur Linearen Regression bei fehlenden Kovariablen

A. Fieger
Institut für Statistik
Akademiestr. 1
80799 München
andreas@stat.uni-muenchen.de

29. Juli 1997

Abstract. In diesem Bericht werden C++ Klassen zu linearen Modellen mit fehlenden Werten in der Kovariablenmatrix X vorgestellt. Diese Klassen implementieren erste verwendbare Modelle wie Zero Order Regression, First Order Regression oder modified First Order Regression und dienen als Ausgangsbasis für weitere Modellklassen. Die hier vorgestellten Klassen können in Simulationsstudien oder für konkrete Datensätze verwendet werden.

Keywords. C++ Klassen, fehlende Werte, Imputationsmechanismen, lineare Regression, Simulationsstudien

1 Einleitung

1.1 Ziel

Die im folgenden vorgestellten Klassen dienen zur Erleichterung der Implementierung von Methoden im Bereich des linearen Regressionsmodells bei fehlenden Werten. Durch die Bereitstellung von Basisklassen für die Organisation der Daten und die Berechnung der Schätzer ist die Umsetzung neuer Verfahren in relativ kurzer Zeit möglich.

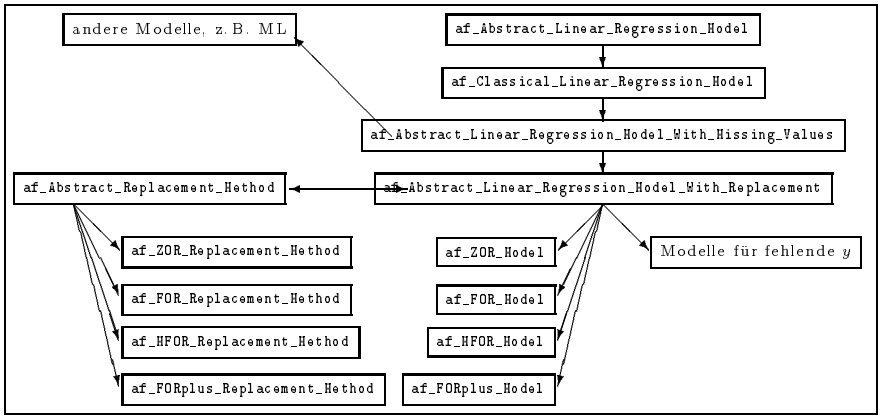

Abbildung 1: Schematischer Aufbau der C++ Klassen zur linearen Regression bei fehlenden Kovariablen

1.2 Aufbau

Wesentlicher Bestandteil aller hier implementierten Modelle sind die Matrizen Template-Libraries die von Kurt Watzka und Christian Heumann zur Verfügung gestellt wurden. Für eine Beschreibung der zugrundeliegenden Matrizenbibliotheken wird an dieser Stelle auf [1] verwiesen. Es soll hier lediglich in Erinnerung gerufen werden, daß Vektoren stets als Matrizen (meist $n \times 1$, aber auch $1 \times n$) aufgefaßt werden. Der Aufbau der hier vorgestellten Klassen zur linearen Regression läßt sich wie in Abbildung 1 schematisch darstellen.

1.3 Verfügbarkeit

Die jeweils aktuellste Version der hier beschriebenen Klassen können über anonymous ftp unter der Adresse `ftp.stat.uni-muenchen.de` bezogen werden. Sie befinden sich im Verzeichnis `/pub/andreas/cc/linreg`. Die `*.cc` Dateien sind ausführlich kommentiert. Durch Bearbeitung mit `lgrind`[1] und anschließendem LaTeX-Lauf kann eine gedruckte Dokumentation der Quelltexte erstellt weden.

[1] CTAN: `/tex-archive/support/lgrind/`

2 af_linrg.h

In dieser Einheit werden im wesentlichen abstrakte Klassen eingeführt, die als Grundlage für lineare Regressionsmodelle bei teilweise fehlenden Daten dienen. Es entsteht hierbei auch bereits eine direkt einsetzbare Klasse für ein klassisches lineares Regressionsmodell ohne fehlende Werte. Die bei fehlenden Werten verwendbaren Klassen werden in Abschnitt 3 vorgestellt.

Die grundlegende Struktur zur Datenorganisation besteht in allen hier vorgestellten Klassen in der Unterteilung des Modells

$$y = X\beta + \epsilon$$

mit teilweise unbeobachteten Kovariablenwerten in X in das Modell

$$\begin{pmatrix} y_c \\ y_* \end{pmatrix} = \begin{pmatrix} X_c \\ X_* \end{pmatrix} \beta + \begin{pmatrix} \epsilon_c \\ \epsilon_* \end{pmatrix}$$

Die fehlenden Werte in X_* werden im folgenden durch verschiedene Imputationsmethoden aufgefüllt und die so vervollständigte Datematrix im mixed Schätzer

$$\hat{\beta} = (X_c'X_c + X_R'X_R)^{-1}(X_c'y_c + X_R'y_*)$$

verwendet.

2.1 Die Basisklasse

Als Basiskalasse für alle linearen Regressionsmodelle führen wir die abstrakte Klasse `af_Abstract_Linear_Regression_Model` ein (Abbildung 2). Mit dieser Klasse wird (datentechnisch) alles brereitgestellt, was von einem linearen Regressionsmodell benötigt wird. Dies sind Daten-Felder für X und y, Daten-Felder für die Schätzer wie $\hat{\beta}$, $\hat{\epsilon}$, $\hat{y}$ und $\hat{\sigma}^2$, R^2, etc. Zusätzlich werden bereits abstrakte Versionen der Schätzfunktionen bereitgestellt (`get_hatxxxx()`). Die dafür benötigeten Funktionen `compute_hatxxxx()` müßen in den von dieser Klasse abgeleiteten Klassen jedoch erst implementiert werden, sie sind in dieser Klasse nur deklariert.

Hier wie auch in allen abgeleiteten Klassen gilt, daß die Funktionen, die die einzelnen Schätzer zurückliefern unabhängig voneinander aufgerufen werden können. So setzt z. B. ein Aufruf von `get_haty()` nicht einen vorherigen Aufruf von `get_hatbeta()` durch den Benutzer voraus. Der für die Berechnung

```
class af_Abstract_Linear_Regression_Model
{
public:
  /* default constructor. */
  af_Abstract_Linear_Regression_Model( void );

  /* initialize repsonse y and datamatrix X assuming
     that all variables are continuous */
  af_Abstract_Linear_Regression_Model( const matrix& y, const matrix& X );

  /* initialize repsonse y and datamatrix X and variabletypes */
  af_Abstract_Linear_Regression_Model(
        const matrix& y, const matrix& X, const intMatrix& variabletypes );

  /* default destructor */
  virtual ~af_Abstract_Linear_Regression_Model( void ) {}

  /* element functions */
  const matrix& get_y( void );
  const matrix& get_X( void );
  const intMatrix& get_variabletypes( void );

  virtual const matrix& get_hatbeta( void );
  virtual const matrix& get_haty( void );
  virtual const matrix& get_hatepsilon( void );
  virtual const double get_hatsigmaepsilon( void );
  virtual const double get_Rsquared( void );
  virtual const double get_adjustedRsquared( void );
  virtual matrix get_MSEI( const matrix& truebeta );
  virtual double get_MSEII( const matrix& truebeta );
  virtual double get_MSEIII( const matrix& truebeta );
  virtual matrix get_Bias( const matrix& truebeta );
};
```

Abbildung 2: `af_Abstract_Linear_Regression_Model`

von $\hat{y}$ benötigte Schätzer $\hat{\beta}$ wird – falls er nicht bereits zu einem vorherigen Zeitpunkt schon berechnet wurde – mit dem Aufruf von `get_haty()` automatisch berechnet und abgespeichert.

```
if ( !m_hashatbeta ){ get_hatbeta(); }
/* auf %$\hat{\beta}$% kann nun zugegriffen werden.
   Berechnung des von %$\hat{\beta}$% abhängigen Schätzers  */
get_hatxxx();
```

Schätzfunktionen in abgeleiteten Klassen sollten sich an dieses oben schematisch dargestellte Konzept halten um fehlerhafte Ergebnisse zu vermeiden, die entstehen könnten, wenn eine gewisse Aufrufreihenfolge der Funktionen durch den Benutzer nötig wäre.

Kovariablen Datentyp. In allen Modellen kann optional ein Vektor übergeben werden, der die Art der Kovariablen X_j (stetig, binär, kategoriell) spezifiziert. Wird dieser nicht angegeben, so wird von stetigen Variablen ausgegangen. Zu beachten ist hierbei jedoch, daß die Methoden für mehrkategorielle Va-

riablen im momentanen Stand der Klassenbibliothek noch **nicht** vollständig implementiert sind; es wird gegebenenfalls eine Fehlermeldung ausgegeben. Liegen die Daten jedoch bereits in kodierter Form vor, so spielen diese Einschränkungen bei der Parameterschätzung $\hat{\beta}$ keine Rolle, da die Schätzer für β dann durchweg implementiert sind. Bei den Imputationsmethoden ist dieser Umweg jedoch nicht möglich, da z. B. eine fehlende $(0, 0, 1)$ kodierte '3' nicht durch die Spaltenmittel der einzelnen Dummys ersetzt werden kann, die für einen einzelnen Dummy durchaus sinnvoll sein mögen.

2.2 Elementfunktionen

Die im folgenden beschriebenen Elementfunktionen werden mit `af_Abstract_Linear_Regression_Model` eingeführt und stehen damit in allen abgeleiteten Klassen zur Verfügung.

NAME `get_hatbeta()` – Schätzer $\hat{\beta}$
SYNOPSIS

```
matrix get\_hatbeta();
```

BESCHREIBUNG `get_hatbeta()` liefert den Schätzer $\hat{\beta} = (X'X)^{-1}X'y$ zurück.

NAME `get_haty()` – Schätzer $\hat{y}$
SYNOPSIS

```
matrix get\_haty();
```

BESCHREIBUNG `get_haty()` liefert den Schätzer $\hat{y} = X\hat{\beta}$ zurück.

NAME `get_hatepsilon()` – Schätzer $\hat{\epsilon}$
SYNOPSIS

```
matrix get\_hatepsilon();
```

BESCHREIBUNG `get_hatepsilon()` liefert den Schätzer $\hat{\epsilon} = y - \hat{y}$ zurück.

NAME `get_hatsigmaepsilon()` – Schätzer $\hat{\sigma}^2$

SYNOPSIS

```
matrix get\_hatsigmaepsilon();
```

BESCHREIBUNG `get_hatsigmaepsilon()` liefert den Schätzer $\hat{\sigma}^2 = \frac{1}{n-p}\hat{\epsilon}'\hat{\epsilon}$ zurück.

BEMERKUNGEN n entspricht der Anzahl der Zeilen der Datenmatrix X, p der Anzahl ihrer Spalten.

NAME `get_Rsquared()` – Schätzer für R^2

SYNOPSIS

```
matrix get\_Rsquared();
```

BESCHREIBUNG `get_Rsquared()` liefert einen Schätzer für das Bestimmtheitsmaß R^2 zurück.

BEMERKUNGEN $R^2 = \frac{SS_{\text{Reg}}}{SYY} = 1 - \frac{RSS}{SYY}$ mit $SYY = \sum_{i=1} n(y_i - \bar{y})^2$ und $RSS = \sum_{i=1} n(y_i - \hat{y}_i)^2$.

NAME `get_adjustedRsquared()` – Schätzer für adjustiertes R^2

SYNOPSIS

```
matrix get\_adjustedRsquared();
```

BESCHREIBUNG `get_adjustedRsquared()` liefert das adjustierte Bestimmtheitsmaß zurück, bei dem die Anzahl der Regressionsparameter berücksichtigt wird.

NAME `get_MSEI()` – MSE I Matrix

SYNOPSIS

```
matrix get_MSEI( const matrix& truebeta );
```

BESCHREIBUNG `get_MSEI()` liefert die MSE I Matrix $\mathrm{V}(\hat{\beta}) + \mathrm{Bias}(\hat{\beta}, \beta)\,\mathrm{Bias}(\hat{\beta}, \beta)'$ zurück.

`truebeta` In Simulationen bekannter wahrer Wert des Parameters β.

SIEHE AUCH `get_MSEII()`, `get_MSEIII()`, `get_Bias()`

NAME `get_MSEII()` – MSE II

SYNOPSIS

```
double get_MSEII( const matrix& truebeta );
```

BESCHREIBUNG `get_MSEII()` liefert den skalaren MSE II $\mathrm{E}(\hat{\beta}-\beta)'(\hat{\beta}-\beta)$ zurück.

`truebeta` In Simulationen bekannter wahrer Wert des Parameters β.

SIEHE AUCH `get_MSEI()`, `get_MSEIII()`, `get_Bias()`

NAME `get_MSEIII()` – MSE III

SYNOPSIS

```
double get_MSEIII( const matrix& truebeta );
```

BESCHREIBUNG `get_MSEIII()` liefert den skalaren MSE III $\mathrm{E}(X\hat{\beta}-X\beta)'(X\hat{\beta}-X\beta)$ zurück.

`truebeta` In Simulationen bekannter wahrer Wert des Parameters β.

SIEHE AUCH `get_MSEI()`, `get_MSEII()`, `get_Bias()`

NAME `get_Bias()` – Bias

SYNOPSIS

```
matrix get_Bias( const matrix& truebeta );
```

BESCHREIBUNG `get_Bias()` liefert den Bias Vektor $\mathrm{E}(\hat{\beta}-\beta)$ zurück.

`truebeta` In Simulationen bekannter wahrer Wert des Parameters β.

SIEHE AUCH `get_MSEI()`, `get_MSEII()`, `get_MSEIII()`

2.3 *Abgeleitete Klassen*

Hier kommt das erste praktisch verwendbare Modell. Die erste von der Basisklasse abgeleitete Klasse, `af_Classical_Linear_Regression_Model` (Abbildung 3), implementiert das klassische lineare Regressionsmodell ohne fehlende Werte. Hier werden keine weiteren Felder oder Funktionen bereitgestellt, es werden lediglich die Schätzprozeduren (`compute_xxxx()`) implementiert.

```
class af_Classical_Linear_Regression_Model :
public af_Abstract_Linear_Regression_Model
{
public:
  /* default constructor */
  af_Classical_Linear_Regression_Model( void );

  /* initialize repsonse y and datamatrix X,
     all %$X$% variables are assumed CONTINUOUS */
  af_Classical_Linear_Regression_Model( const matrix& y, const matrix& X );

  /* initialize repsonse y and datamatrix X and variabletypes */
  af_Classical_Linear_Regression_Model( const matrix& y, const matrix& X,
                                        const intMatrix& variabletypes );

  /* default destructor */
  virtual ~af_Classical_Linear_Regression_Model( void );
};
```

Abbildung 3: `af_Classical_Linear_Regression_Model`

2.4 Ein abstrakter Zwischenschritt

Es folgt wieder ein abstrakter Zwischenschritt. Die im folgenden vereinbarte Klasse `af_Abstract_Linear_Regression_Model_With_Missing_Values` (Abbildung 4), wird von der Klasse `af_Classical_Linear_Regression_Model` abgeleitet. Es werden zusätzlich einige weitere Datenfelder bereitgestellt, um mit fehlenden Daten umgehen zu können. Dies sind eine 'missing data' Indikatormatrix R, die Kovariablenmatrix X_* und der zugehörige Responsevektor y_* für das Submodell mit unvollständigen Beobachtungen. Die von `af_Classical_Linear_Regression_Model` geerbten Felder enthalten nun die Daten des 'complete case' Modells, also die Matrizen X_c und y_c:

$$\begin{pmatrix} y_c \\ y_* \end{pmatrix} = \begin{pmatrix} X_c \\ X_* \end{pmatrix} \beta + \begin{pmatrix} \epsilon_c \\ \epsilon_* \end{pmatrix}$$

Schätzfunktionen analog zu `get_hatxxxx()`, die alle Daten des Modells verwenden werden noch nicht deklariert. Dies erfolgt im nächsten Schritt (Abschnitt 2.5) zusammen mit der Einführung eines Imputationsmechanismus.

2.5 Noch ein abstrakter Zwischenschritt

Es folgt noch ein weiterer abstrakter Zwischenschritt. „Wieso denn nicht gleich in einem Schritt?“ Motivation dafür, einen weiteren Zwischenschritt einzufügen, ist die damit offengehaltene Möglichkeit, später an dieser Stelle in eine Klasse von Modellen zu verzweigen, die nicht 'with replacement' sind, wie z. B. ML-Methoden. Die Klasse `af_Abstract_Linear_Regression_Model_`

```
class af_Abstract_Linear_Regression_Model_With_Missing_Values :
public af_Classical_Linear_Regression_Model
{
public:
  /* default constructor */
  af_Abstract_Linear_Regression_Model_With_Missing_Values( void );

  /* initialize complete-repsonse y, complete-model datamatrix X
     incomplete-model repsonse yast and incomplete-model datamatrix Xast
     and missing indicator matrix R (all variables CONTINUOUS) */
  af_Abstract_Linear_Regression_Model_With_Missing_Values(
        const matrix& yc, const matrix& Xc,
        const matrix& yast, const matrix& Xast,
        const intMatrix& R );

  /* same as above plus initializing m_variabletypes */
  af_Abstract_Linear_Regression_Model_With_Missing_Values(
        const matrix& yc, const matrix& Xc,
        const matrix& yast, const matrix& Xast,
        const intMatrix& R, const intMatrix& variabletypes );

  /* default destructor */
  virtual ~af_Abstract_Linear_Regression_Model_With_Missing_Values( void );

  /* element functions, first two are virtual because we later
     override them as the replacement procedures are implemented.
     %$R$% and the complete case data are never changed */
  virtual const matrix& get_yast( void );
  virtual const matrix& get_Xast( void );
  const intMatrix& get_R( void );

  /* compute estimated missing probabilities from the info in %$R$% */
  virtual const matrix& get_missprobs( void );
};
```

Abbildung 4: `af_Abstract_Linear_Regression_Model_With_Missing_Values`

`With_Replacement` (Abbildung 5) bietet zusätzlich zu den Feldern, die von der im letzten Abschnitt vorgestellten Klasse `af_Abstract_Linear_Regression_Model_With_Missing_Values` geerbt werden noch eine Ersetzungsmethode für die fehlenden Werte in X_*, genauer einen Zeiger auf ein 'Ersetzungsobjekt' (siehe Abschnitt 4).

Zusätzlich zu den bereits vorhandenen Schätzprozeduren `get_hatxxxx()`, die nur die vollständig beobachteten Daten verwenden (complete case), werden wie oben bereits angesprochen nun auch analog arbeitende Prozeduren `get_hatxxxxall()` bereitgestellt, die alle Daten verwenden. Diese Funktionen werden vollkommen analog zu den `get_hatxxx()` Funktionen verwendet. Der einzige Unterschied besteht in der zusätzlichen Verwendung der aufgefüllten Daten.

Bei Verwendung einer solchen Funktion wird zunächts geprüft, ob die feh-

lenden Werte bereits ersetzt wurden. Falls dies nicht der Fall ist, so wird die Ersetzung an dieser Stelle durchgeführt und dann die gewünschte Schätzfunktion unter Verwendung der nun vervollständigten Daten aufgerufen.

```
if ( !m_hasbeenreplaced ){ do_replacement(); }
/* fehlende Daten sind nun ersetzt */

/* Berechnung des jeweiligen Schätzers  */
get_hatxxx();
```

Die Ersetzung der Fehlenden Werte geschieht mittels der Methode `replace_missingvalues()`, die durch das zugehörige Ersetzungsobjekt bereitgestellt werden muß. Für die Parameterschätzung kann damit an dieser Stelle stets der mixed Schätzer

$$\hat{\beta} = (X_c' X_c + X_R' X_R)^{-1}(x_c' y_c + X_R' y_*)$$

beziehungsweise eine Version mit Gewichtung der unvollständigen Fälle (vgl. [2]) verwendet werden. Die verschiedenen Modelle (ZOR, FOR, MFOR, ...) ergeben sich durch die verschiedenen Imputationsmechanismen, die dann jeweils ein anderes X_R liefern.

Desweiteren werden hier eine Reihe von Diagnosemaßen eingeführt, die aus dem Bereich Ausreißerentdeckung stammen und hier zur Diagnose bezüglich des Fehlendmechanismus verwendet werden. Es sei dazu auf die Diplomarbeit von Walbrunn [5] verwiesen. Diese Diagnosemaße sind

- `get_CooksD()` Cooks Distance, berechnet aus Complete Case gegen Mixed Modell mit imputierten Werten,
- `get_DXX()` Verhältnis der Determinanten, $\frac{|X_c' X_c|}{|(X_c', X_R')(X_c', X_R')'|}$
- `get_DRSS()` Veränderung der Residuenquadratsumme, Complete Case gegen Mixed Modell.

```
class af_Abstract_Linear_Regression_Model_With_Replacement :
public af_Abstract_Linear_Regression_Model_With_Missing_Values
{
public:
  /* default contructor */
  af_Abstract_Linear_Regression_Model_With_Replacement( void );

  /* all %$X$% variables CONTINUOUS */
  af_Abstract_Linear_Regression_Model_With_Replacement(
    const matrix& yc, const matrix& Xc,
    const matrix& yast, const matrix& Xast,
    const intMatrix& R );

  /* same as above plus initializing m_variabletypes */
  af_Abstract_Linear_Regression_Model_With_Replacement(
    const matrix& yc, const matrix& Xc,
    const matrix& yast, const matrix& Xast,
    const intMatrix& R, const intMatrix& variabletypes );

  /* default destructor */
  virtual ~af_Abstract_Linear_Regression_Model_With_Replacement( void ) {}

  /* element functions */
  virtual const matrix& get_hatbetaall( void );
  virtual const matrix& get_hatyall( void );
  virtual const matrix& get_hatepsilonall( void );
  virtual const double get_hatsigmaepsilonall( void );

  /* return incomlete matrices after imputation */
  virtual const matrix& get_yast( void );
  virtual const matrix& get_Xast( void );

  /* statistics */
  virtual const double get_Rsquaredall( void );
  virtual const double get_adjustedRsquaredall( void );
  virtual matrix get_MSEIall( const matrix& truebeta );
  virtual double get_MSEIIall( const matrix& truebeta );
  virtual double get_MSEIIIall( const matrix& truebeta );
  virtual matrix get_Biasall( const matrix& truebeta );
  virtual const double get_CooksD( void );
  virtual const double get_DXX( void );
  virtual const double get_DRSS( void );

  /* switch the usage of weights for the imputed data on or off */
  virtual void set_useweights( const unsigned onoff );
  virtual void set_whichweight( const unsigned weightnumber ); /* see Little, 1992 */
  virtual void reset_estimators ( void );
};
```

Abbildung 5: `af_Abstract_Linear_Regression_Model_With_Replacement`

3 af_imprg.h

Die Klassen in der Bibliothek `af_imprg.h` basieren auf den oben vorgestellten Klassen. Sie stellen Klassen für Modelle mit fehlenden Daten und konkreten Ersetzungs- und Schätzfunktionen bereit. Sie besitzen alle den gleichen Aufbau, der im folgenden beispielhaft an der Klasse `af_ZOR_Model` (Mixed-Schätzer nach ZOR Ersetzung) demonstriert wird. Über die Bereitstellung einer neuen Ersetungsmethode kann so schnell ein zugehöriges 'Schätzobjekt' abgeleitet werden.

- Sie vereinbaren drei Konstruktoren und einen Destruktor.
- Sie sind alle von der Klasse `af_Abstract_Linear_Regression_Model_With_Replacement` abgeleitet. Damit besitzen sie bereits (einen Zeiger auf) eine 'replacement' Methode `do_replacement()`, sowie die complete case Schätzfunktionen und Versionen dieser Funktionen für die Verwendung mit imputierten Daten (`get_hatxxxxall()`).
- Sie besitzen ein Feld `m_replacementmethod`, welches ein Zeiger auf ein Objekt einer 'Ersetzungsmethoden Klasse' ist (siehe Abschnitt 4). Über die ererbte Methode `do_replacement()` wird hier die jeweilige Ersetzungsmethode ausgeführt (Abbildung 6).

```
void af_Abstract_Linear_Regression_Model_With_Replacement::do_replacement( void )
{
  /* call the replacement procedure, that will be implemented in
     a class derived from af_abstract_replacement_procedure.
     The field is a pointer! */
  m_replacementmethod -> replace_missingvalues();

  /* set the new %$X\sb{*}$% which is now %$X\sb{R}$% */
  m_Xast =  m_replacementmethod -> get_XR();

  /* set flag, indicating replacement has been performed */
  m_hasbeenreplaced = 1;
}
```

Abbildung 6: `do_replacement()` Methode

3.1 af_ZOR_Model

Die Klasse `af_ZOR_Model` (Abbildung 7) implementiert ein lineares Regressionsmodell mit fehlenden Werten, die durch die Zero-Order-Regression Ersetzungsmethode (unconditional mean imputation) aufgefüllt wurden. Fehlende Werte in einer Spalte werden durch den Spaltenmittelwert der vollständig beobachteten Fälle $\bar{x}_j^c$ ersetzt.

```
class af_ZOR_Model :
public af_Abstract_Linear_Regression_Model_With_Replacement
{
public:
  /* default constructor */
  af_ZOR_Model(void)
  :
  af_Abstract_Linear_Regression_Model_With_Replacement()
  {}

  /* initialize complete-repsonse m_y, complete-model datamatrix m_X
   incomplete-model repsonse m_yast and incomplete-model datamatrix m_Xast
   and missing indicator matrix m_R (all variables assumed CONTINUOUS) */
  af_ZOR_Model( const matrix& yc, const matrix& Xc,
                const matrix& yast, const matrix& Xast,
                const intMatrix& R );

  /* same as above plus initializing m_variabletypes */
  af_ZOR_Model( const matrix& yc, const matrix& Xc,
                const matrix& yast, const matrix& Xast,
                const intMatrix& R,
                const intMatrix& variabletypes  );

  /* destructor */
  virtual ~af_ZOR_Model(void); {
};
```

Abbildung 7: `af_ZOR_Model`

3.2 Die anderen Modelle

Weitere implementierte Modelle sind

- `af_FOR_Model`: First-Order-Regression Ersetzung (conditional mean imputation). Die Ersetzung fehlender Werte in X_* geschieht mittels 'Hilfsregressionen' der Gestalt

$$X_*^j = X_*^{(-j)} \hat{\gamma}^j$$

  wobei X_*^j die j-te Spalte von X_*, $X_*^{(-j)}$ die Matrix X_* ohne die j-te Spalte und $\hat{\gamma}^j$ der geschätzte Regressionskoeffizient der Regression der j-ten Spalte von X_c auf die verbleibenden Spalten von X_c ist, also aus den vollständigen Daten geschätzt wird.

- `af_MFOR_Model`: Modified First-Order-Regression Ersetzung. Wie `af_FOR_Model` jedoch mit zusätzlicher Verwendung der Responsevariablen y bei Hilfsregressionen zur Ersetzung.

- `af_FORplus_Model`: Wie `af_MFOR_Model` plus überlagerte Störterme um die Varianz nicht durch 'zu glatte' Ersetzungen zu unterschätzen.

3.3 Ausblick

In die hier beschriebenen Bibliotheken noch nicht eingearbeitet, aber in Planung sind

- optionale Berechnung der Hilfsparameter aus den jeweils verfügbaren Fällen, in Anlehnung an die available case Schätzer.
- optionale Methoden zur Biaskorrektur mittels Bootstrap, sofern dies sich als praktikabel und sinnvoll herausstellt.

4 af_repl.h

Hier werden die in Abschnitt 3 angesprochenen Ersetzungsmethoden implementiert, die jeweils über die Methode `replace_missingvalues()` angesprochen werden können. Wie bei den oben vorgestellten Klassen werden wieder alle Gemeinsamkeiten in einer abstrakte Basisklasse vereint (Abbildung 8), die dann als Ausgangsbasis dient.

```
class af_Abstract_Replacement_Method
{
public:
  /* default constructor */
  af_Abstract_Replacement_Method(void);

  /* initialize repsonse m_y and datamatrix m_X */
  af_Abstract_Replacement_Method( const matrix& yc, const matrix& Xc,
                                  const matrix& yast, const matrix& Xast,
                                  const intMatrix& R, const intMatrix& typeinfo );

  virtual ~af_Abstract_Replacement_Method(void);

  /* replacement procedure */
  virtual void replace_missingvalues(void);

  /* return the filled-in matrix %$X_R$% */
  const matrix& get_XR(void);
  /* return the filled-in matrix %$y_R$% */
  const matrix& get_yR(void);
};
```

Abbildung 8: `af_Abstract_Replacement_Method`

Die grundsätzliche Struktur der daraus abgeleiteten Klassen sei hier wieder am Beispiel der Zero-Order-Regression (`af_ZOR_Replacement_Method`, Abbildung 9) verdeutlicht. Es existiert zu jeder Klasse die in Abschnitt 3 vorgestellt wurde, hier eine zugehörige Klasse `af_xxxx_Replacement_Method`, in

der die jeweilige Ersetzungsmethode implementiert wird. Diese Klassen bestehen immer aus einem oder mehreren Konstruktoren, einem Destruktor und der Methode `replace_missingvalues()`, in der die jeweilige Ersetzungsmethode realisiert ist.

```
class af_ZOR_Replacement_Method :
public af_Abstract_Replacement_Method
{
public:
  /* default constructor */
  af_ZOR_Replacement_Method(void);

  /* initialize matrices */
  af_ZOR_Replacement_Method( const matrix& yc, const matrix& Xc,
                             const matrix& yast, const matrix& Xast,
                             const intMatrix& R, const intMatrix& typeinfo );

  virtual ~af_ZOR_Replacement_Method(void);

  /* implementation of the ZOR replacement procedure */
  virtual void replace_missingvalues( void );
};
```

Abbildung 9: `af_ZOR_Replacement_Method`

5 Weitere Klassen für Simulationsstudien, Kurzbeschreibung

5.1 af_boot.h

Bootstrapmethoden für lineare Regressionsmodelle. Diese Klassen dienen zum Beispiel zur Schätzung des Bias von $\hat{\beta}^{\mathrm{MFOR}}$, um so eine biaskorrigierte Version dieses Schätzers zu erhalten. Zunächst die abstrakte Basisklasse (Abbildung 10). Ausgehend von der Basisklasse werden zwei weitere abstrakte Klassen eingeführt, die Vektor- und Residualsampling Methoden bereitstellen (siehe Abbildungen 11 und 12). Konkret verwendbare Modelle sind dann die hiervon abgeleiteten Klassen `af_MFOR_V_Bootstrap_Model` (modifizierte FOR, vector sampling) und `af_MFOR_R_Bootstrap_Model` (modifizierte FOR, residual sampling).

```
class af_Abstract_Bootstrap_Model
{
public:
  /* default constructor. */
  af_Abstract_Bootstrap_Model( void );

  /* initialize repsonse m_y and datamatrix m_X
     assuming that all variables are continuous */
  af_Abstract_Bootstrap_Model( const matrix& y, const matrix& X );

  /* initialize repsonse m_y and datamatrix m_X and m_variabletypes */
  af_Abstract_Bootstrap_Model( const matrix& y, const matrix& X,
                               const intMatrix& variabletypes );

  /* default destructor */
  virtual ~af_Abstract_Bootstrap_Model( void );

  /* element functions */
  const matrix& get_y( void ); { return m_y; }
  const matrix& get_X( void );
  const intMatrix& get_variabletypes( void );

  virtual const matrix& get_hatbeta( void );
};
```

Abbildung 10: `af_Abstract_Bootstrap_Model`

```
class af_Vector_Bootstrap_Model :
public af_Abstract_Bootstrap_Model
{
public:
  /* default constructor */
  af_Vector_Bootstrap_Model( void );

  /* initialize repsonse m_y and datamatrix m_X
     all %$X$% variables are assumed CONTINUOUS */
  af_Vector_Bootstrap_Model( const matrix& y, const matrix& X );

  /* initialize repsonse m_y and datamatrix m_X and m_variabletypes */
  af_Vector_Bootstrap_Model( const matrix& y, const matrix& X,
                             const intMatrix& variabletypes );

  /* default destructor */
  virtual ~af_Vector_Bootstrap_Model( void );
};
```

Abbildung 11: `af_Vector_Bootstrap_Model`

5.2 af_ymisr.h

Diese Bibliothek (Abbildung 13) führt Klassen für fehlende Werte im Response ein. Der Aufbau ist analog zu den Klassen, die in Abschnitt 3 beschrieben werden. Die zugehörigen Klassen für die Ersetzungsmethoden befinden sich in der Bibliothek `af_repl.h` (vergleiche Abschnitt 4). Von der abstrakten Basisklasse werden die beiden Klassen `af_Yates_Model` und

```
class af_Residual_Bootstrap_Model :
public af_Abstract_Bootstrap_Model
{
public:
  /* default constructor */
  af_Residual_Bootstrap_Model( void );

  /* initialize repsonse m_y and datamatrix m_X,
     all %$X$% variables are assumed CONTINUOUS */
  af_Residual_Bootstrap_Model( const matrix& y, const matrix& X );

  /* initialize repsonse m_y and datamatrix m_X and m_variabletypes */
  af_Residual_Bootstrap_Model( const matrix& y, const matrix& X,
                               const intMatrix& variabletypes );

  /* default destructor */
  virtual ~af_Residual_Bootstrap_Model( void );
};
```

Abbildung 12: `af_Residual_Bootstrap_Model`

`af_Stein_Model` abgeleitet. Sie finden z. B. Verwendung für Shrinkage estimation, vgl. [4].

```
class af_Abstract_Linear_Regression_Model_With_Missing_Y :
public af_Abstract_Linear_Regression_Model_With_Replacement
{
public:
  /* default constructor */
  af_Abstract_Linear_Regression_Model_With_Missing_Y(void);

  /* initialize complete-repsonse m_y, complete-model datamatrix m_X
   incomplete-model repsonse m_yast and incomplete-model datamatrix m_Xast
   and missing indicator matrix m_R (all variables assumed CONTINUOUS) */
  af_Abstract_Linear_Regression_Model_With_Missing_Y(
               const matrix& yc, const matrix& Xc,
               const matrix& yast, const matrix& Xast,
               const intMatrix& R );

  /* same as above plus initializing m_variabletypes */
  af_Abstract_Linear_Regression_Model_With_Missing_Y(
               const matrix& yc, const matrix& Xc,
               const matrix& yast, const matrix& Xast,
               const intMatrix& R,
               const intMatrix& variabletypes  );

  /* destructor */
  virtual ~af_Abstract_Linear_Regression_Model_With_Missing_Y(void){}
};
```

Abbildung 13: `af_Abstract_Linear_Regression_Model_With_Missing_Y`

5.3 af_mdat.h

Diese Klassen dienen zur Erzeugung von Daten gemäß bestimmter Vorgaben, die in Simulationsstudien verwendet werden können. Wie immer: zuerst eine abstrakte Basisklasse für die Gemeinsamkeiten. Dies ist hier die Klasse `af_Abstarct_Make_Data` mit den Elementfunktionen

- `get_X()`, Zugriff auf die Kovariablenmatrix,
- `get_y()`, Zugriff auf den Responsevektor,
- `reset()`, Zurücksetzen des Erzeugungsmechanismus, d. h. bei erneutem Aufruf von `get_X()` oder `get_y()` werden andere Daten zurückgeliefert.

Davon werden dann die eigentlich verwendeten Klassen abgeleitet. Zum Beispiel die Klasse `af_Normal_Data` (Abbildung 14), die zur Erzeugung von multivariat normalverteilten Zufallszahlen dient (genauer: Matrizen, deren Zeilen iid. $N(\mu, \Sigma)$ verteit sind). `get_y()` liefert hier die erste Spalte der (`N`×`Sigma.cols()`) Datenmatrix, `get_X()` liefert die restlichen Spalten.

```
class af_Normal_Data :
public af_Abstarct_Make_Data
{
public:
  /* constructor if mean and covariance matrix are supplied */
  af_Normal_Data( const matrix& mu,
                  const matrix& Sigma,
                  const unsigned N );

  /* constructor if mean, variance vector and
     correlation matrix are supplied */
  af_Normal_Data( const matrix& mu,
                  const matrix& Var,
                  const matrix& Rho,
                  const unsigned N );

   /* default destructor */
   virtual ~af_Normal_Data(void);
};
```

Abbildung 14: `af_Normal_Data`

Weitere Klassen sind (momentaner Stand, Erweiterung je nach Bedarf)

- `af_LinearRegression_Data`. Die erzeugten Daten sind gemäß dem Modell $y = X\beta + \epsilon$ 'verteilt'. Die Fehlergröße wird als $\epsilon_i \overset{\text{iid.}}{\sim} N(0, \sigma^2)$ angenommen. Die Varianz σ^2 und die Kovariablenmatrix X werden im Konstruktor übergeben.

5.4 af_mmis.h

Diese Klassen dienen zur Erzeugung von 'fehlenden Werten' gemäß bestimmter Vorgaben, wie z.B. missing completely at random (MCAR) um Daten für Simulationsstudien zu erzeugen. Basis ist zunächst wieder eine abstrakte Klasse, die die nötigen Datenfelder und Zugriffsmechanismen bereitstellt (Abbildung 15).

```
class af_Abstract_Make_Missing
{
public:
  /* default constructor. */
  af_Abstract_Make_Missing( void );
  /* initialize repsonse m_y and datamatrix m_X */
  af_Abstract_Make_Missing( const matrix& y, const matrix& X );
  /* default destructor */
  virtual ~af_Abstract_Make_Missing( void );
  /* element functions */
  const matrix& get_yc( void );
  const matrix& get_yast( void );
  const matrix& get_Xc( void );
  const matrix& get_Xast( void );
  const intMatrix& get_R( void );
  /* reset's the data, i.e. sets m_hascreatedmissingvalues to false. */
  void reset( void );
};
```

Abbildung 15: `af_Abstract_Make_Missing`

Davon abgeleitet werden dann die jeweiligen 'Fehlendmechanismen', die die (private) Prozedur `make_missing()` implementieren und nach Bedarf weitere Felder bereitstellen. Als Beispiel sie die Klasse `af_MCAR_Mechanism` (Abbildung 16) angegeben.

```
class af_MCAR_Mechanism :
public af_Abstract_Make_Missing
{
public:
  af_MCAR_Mechanism( void );
  /* constructor using equal missing probability for all variables (columns) */
  af_MCAR_Mechanism( const matrix& y, const matrix& X, const double missprob );
  /* constructor using a row vector of missing probabilities for the variables */
  af_MCAR_Mechanism( const matrix& y, const matrix& X, const matrix& missprobvec );
  /* default destructor */
  virtual ~af_MCAR_Mechanism(void);
  /* set a new value for the missing probability */
  virtual void set_missprob( const double newvalue );
  /* same if a vector of probabilities was specified */
  virtual void set_missprobvec( const matrix& newvalues );
};
```

Abbildung 16: `af_MCAR_Mechanism`

5.5 af_matfu.h

Hier werden (Hilfs-) Funktionen für Matrizen wie z. B. die Berechnung von Spaltenmittelwerten, Blockweises Kopieren von Matrizen, etc. bereitgestellt (vergleiche Abbildung 17). Weitere Funktionen für Matrizen sind entweder bereits Bestandteil der Matrizenklasse[2], oder in der Bibliothek `matfun.h` (Ch. Heumann) implementiert.

```
double colmean( const unsigned colnum, const matrix& X );
/* computes the mean of a column of a given matrix. */

matrix get_block_rowwise ( const matrix& X,
                           const unsigned rowfirst, const unsigned rowlast );
/* return the 'block' (a matrix consisting of the rows of %$X$%,
 from 'rowfirst' to 'rowlast') as a new matrix */

matrix get_block_colwise ( const matrix& X,
                           const unsigned colfirst, const unsigned collast );
/* return the 'block' (a matrix consisting of the columns of %$X$%,
 from 'colfirst' to 'collast') as a new matrix */

matrix get_without_col ( const matrix& X, const unsigned colnum );
/* return the matrix consisting of the columns of %$X$%,
 except the column number colnum as a new matrix */

matrix get_without_row ( const matrix& X, const unsigned rownum );
/* return the matrix consisting of the rows of %$X$%,
 except the row number rownum as a new matrix */

void print_matrix_tabdelimited( const matrix& X, ostream& out );
/* like X.prettyPrint(), but we may set the width and precision of out
 and the elements of the matrix are delimited by tabs, which is usefull
 if we want to use this output as new input for statistical packages */

void set_row ( matrix& X, const unsigned rownum, const matrix& newrow );
/* let the values in newrow be the new values in %$X$%'s row number 'rownum' */

matrix get_covariancematrix ( matrix& rho, matrix& variances );
/* computes the variancecovariance matrix resulting from the correlations matrix rho
 and the variances vector variances */

intMatrix column_have_missings ( const intMatrix& R );
/* returns a row vector indicating if the respective column has missing values.
 %$R$% is the matrix containing the information about missingness of %$X_{ij}$% */

double det( const matrix& X );
/* returns %$\mathrm{det}(X)$% */
```

Abbildung 17: `af_matfu.h`

[2]Die hier bereitgestellten Funktionen werden aus der Bibliothek entfernt, sofern sie Bestandteil der in [1] beschriebenen Matrizenklassen werden. Dies ist z. B. für die blockweisen Zugriffsoperationen geplant.

5.6 af_mattex.h

Funktionen für Ausgabe von Matrizen als LaTeX-Quellcode für Tabellen oder als Matrizen im mathematischen Modus, etc. (Abbildung 18).

```
void print_TeX_matrix ( const matrix& X, ostream& out, const unsigned precision );
/* print the matrix as a %%\LaTeX%% matrix, assuming we're already im math mode */

void print_TeX_tabular ( const matrix& X, ostream& out, const unsigned precision );
/* print the matrix as a %%\LaTeX%% tabular, assuming we're using dcolumn.sty */

void print_TeX_header ( ostream& out );
/* print a default %%\LaTeX%% header */

void print_TeX_footer ( ostream& out );
/* print a default %%\LaTeX%% header */
```

Abbildung 18: af_mattex.h

6 Literatur

[1] A. Fieger, Ch. Heumann, Ch. Kastner und K. Watzka, (1997). Generische Bibliothek zur Linearen Algebra und zur Simulation in C++, *SFB386 Discussion Paper*, **63**.

[2] R. J. A. Little, (1992). Regression with missing X, *JASA*, **87**.

[3] Toutenburg, H., Srivastava, V.K. und Fieger, A., (1996). Estimation of parameters in multiple regression with missing X-observations using modified first order regression procedure. *SFB386 Discussion Paper* **38**.

[4] Toutenburg, H., Srivastava, V.K., Fieger, A., (1997). Shrinkage estimation of incomplete regression models by Yates procedure. *SFB386 Discussion Paper* **69**.

[5] Walbrunn, D., (1997). Regressionsdiagnostik zur Identifizierung von nicht-MCAR Prozessen. Dipomarbeit an der Ludwig Maximilians Universität, München.

[6] Watzka, K., (1996). Virtueller C++-Workshop. *http://www.stat.uni-muenchen.de/~kurt/Cwork/*.