Programmierbeispiele zur Aufbereitung von FDZ Personendaten in STATA

Drews, Nils; Groll, Dominik; Jacobebbinghaus, Peter

Veröffentlichungsversion / Published Version
Arbeitspapier / working paper

Zur Verfügung gestellt in Kooperation mit / provided in cooperation with:
SSG Sozialwissenschaften, USB Köln

FDZ Methodenreport
Methodische Aspekte zu Arbeitsmarktdaten

Nr. 6/2007

Programmierbeispiele zur Aufbereitung von FDZ Personendaten in STATA

Nils Drews, Dominik Groll, Peter Jacobebbinghaus

Bundesagentur für Arbeit

Inhaltsverzeichnis

1 Einleitung

Der vorliegende FDZ-Methodenreport soll das Einarbeiten in die Personendaten des FDZ der BA im IAB vereinfachen und so zu Zeitersparnissen führen. Die vorgestellten Programme stammen aus unserem "Workshop zur Einführung in das Arbeiten mit den Personendaten der BA und des IAB", den wir in den letzten Jahren wiederholt für neue Nutzer unserer Daten angeboten haben. Wir haben die Programme in diesem FDZ-Methodenreport zusammengestellt und dokumentiert, damit sie zum einen allen Nutzern zur Verfügung stehen und damit zum anderen das Vor- und Nachbereiten zukünftiger Workshops erleichtert wird.

Beispielhaft für die FDZ Personendaten wird hier die Stichprobe der Integrierten Erwerbsbiografien (IEBS) verwendet. Die Programme laufen also mit den IEBS-Testdaten, die an gleicher Stelle wie dieser Methodenreport zum Download bereit stehen. Sie sind aber auch für andere Personendaten des FDZ relevant, insbesondere Daten in Spellform wie die IABS und das LIAB-Längsschnittmodell. Um die Programme zu verstehen, empfiehlt es sich die ersten beiden Kapitel des Datenreports 6/2005 zur IEBS zu lesen. Weiterhin sind Grundkenntnisse in der Verwendung von Stata erforderlich.

Mit den FDZ Personendaten weniger vertrauten Forschern empfehlen wir den Methodenreport zunächst von vorne nach hinten durchzuschauen, da manche Abschnitte auf vorhergehenden aufbauen. Dabei sollte man die Programme simultan laufen lassen und zum besseren Verständnis auch verändern. Wer sich mit Daten und Stata bereits auskennt, kann auch direkt in die ihn interessierenden Abschnitte springen. Wir möchten noch betonen, dass die hier vorgesellten Programme nur zum Teil als Musterlösungen zu interpretieren sind. So zeigen wir in Kapitel 3.4 zur Bereinigung von Inkonsistenzen Lösungswege auf, die konkrete Umsetzung ist aber vom Forscher selbst auf das jeweilige Forschungsziel abzustimmen.

2 Tipps zum Sparen von Hardware und Zeit

Die Personendaten des FDZ sind in der Regel mehrere GB groß, daher sollte man sich immer um einen sparsamen Umgang insbesondere von Arbeitsspeicher bemühen.

2.1 Nur die benötigten Variablen und Beobachtungen einlesen

```
use persnr quelle sex if quelle == 1 in 1/1000 using iebs_1_0_test, clear
```

In diesem Beispiel werden nur die Variablen *persnr* uns *quelle* eingelesen, wenn *quelle* == *1* und nur für die ersten 1000 Fälle. Auf diese Weise kann man selbst mit dem PC Teile aus Dateien öffnen, deren Größe den lokalen Arbeitsspeicher weit übertrifft.

2.2 Probeläufe mit Zufallsstichproben durchführen

Beim Schreiben von do-Dateien ist es wegen der langen Laufzeiten bei den großen Originaldaten immer ratsam zunächst mit Stichproben zu arbeiten.

```
use if uniform()<0.05 using iebs_1_0_test, clear
```

Hier wird direkt beim Einlesen der Daten eine einfache 5% Stichprobe der Spells gezogen.

```
use iebs_1_0_test, clear
by persnr, sort: egen long neuepersnr = group(persnr)
keep if neuepersnr <=100
```

Hier wird zunächst die ganze Datei eingelesen, dann werden aber nur die Spells der ersten 100 Personen behalten.

2.3 Neue Variablen mit möglichst wenig Speicherbedarf generieren

```
help data types                      // gibt Überblick über alle existierenden Speichertypen.

generate byte male = 1 if sex == 1   // bildet den Dummy male direkt als byte und nicht also float,
                                     // wodurch 4-mal so viel Speicher belegt würde

compress                             // weist im Nachhinein für jede neu gebildete Variabel den optimalen data type
                                     // =storage type zu und sollte vor jedem Speichern der Daten mit save eingestzt
                                     // werden.

memory                               // zeigt die aktuelle Speichernutzung an
```

2.4 Langsame Befehle durch schnelle ersetzen

Aufgrund der Datenmenge sind die Rechenzeiten selbst für einfache Transformationen oft lang. Durch die Wahl der Stata-Befehle lassen sie sich verkürzen:

Langsam	Schneller	Anmerkung
`recode`	`replace`	
`table`	`tabulate`	immer tabulate verwenden, wenn lediglich Häufigkeiten ausgezählt werden
`duplicates drop varlist`	`by varlist, sort: keep if _n == 1`	siehe Abschnitt 3.2

`duplicates report varlist`	`by varlist, sort: gen neuvar = _n`	

3 Hilfreiche Transformationen

3.1 Bilden von Identifikatoren mit laufenden Nummern

In der IEBS identifiziert eine 10-stellige Personennummer in der Variablen „persnr“ alle Beobachtungen, die zur gleichen Person gehören. Werden Personen gelöscht, ist die Variable nicht mehr fortlaufend, was u.U. nicht gewünscht ist. Im Folgenden wird ein neuer fortlaufender Personenidentifikator gebildet. Das Vorgehen ist auf jegliche Identifikatoren für Gruppen von Beobachtungen übertragbar, z.B. auch Betriebsnummern.

Der *egen*-Befehl zusammen mit der Funktion *group* bildet eine neue Variable mit den Ausprägungen 1 bis N. Die Funktion *group* ist wichtig, da eine Person in der Regel viele verschiedene Spells und somit Zeilen besitzt. Alle Beobachtungen der ersten Person im Datensatz erhalten die Ausprägung 1, alle Beobachtungen der zweiten Person die 2 etc. Die Beobachtungen werden also nach dem Merkmal „persnr“ gruppiert und neu nummeriert.

```
clear
set mem 50m
use iebs_1_0_test
sort persnr spell
browse persnr spell begepi endepi quelle erwstat grund

rename persnr persnr_alt
egen persnr = group(persnr_alt)         // eine neue Personennummer mit Werten von 1 bis N wird gebildet
label variable persnr "Personennummer"  // zur Vereinfachung und verbesserten Übersicht

sort persnr spell
```

```
browse persnr persnr_alt spell begepi endepi quelle erwstat grund
drop persnr_alt
```

3.2 Zeilenübergreifendes Arbeiten mit egen und _n

Daten im Spellformat erfordern in den meisten Fällen ein zeilenübergreifendes Arbeiten. Da sich die Daten zum Erwerbsverlauf einer Person über zahlreiche Zeilen erstrecken können, und jede Zeile eine andere Information enthält, ist es oft nötig, Werte von einer auf andere Zeilen einer Person zu übertragen. Hierzu eignet sich besonders der Befehl *egen*. Ein Beispiel: Die Information ob eine Person zu einem beliebigen Zeitpunkt in Ihrem Erwerbsverlauf bei der Bundesagentur für Arbeit arbeitslos gemeldet (*erwstat* = *31*) war, soll auf alle Spells dieser Person übertragen werden. Zuerst werden dafür alle Spells mit *erwstat* = *31* mit einer 1 markiert, alle anderen Spells erhalten die Null (Variable *mark*). Dann wird diese gerade gesetzte Markierung auf alle Spells der Person übertragen, indem *egen* für jede Person getrennt (siehe *by*) den Wert 1 (siehe *max*) in eine neue Variable schreibt (*markpers*). Alternativ kann man beispielsweise auch nur die Spells markieren, die parallel zum Arbeitslosigkeitsspell in derselben Episode liegen (zweiter *egen*-Befehl).

```
gen     mark = 0
replace mark = 1 if erwstat==31           // Markierung aller Spells mit Arbeitsuche bei Arbeitslosigkeit
egen markpers = max(mark==1), by(persnr)  // Uebertragung der Markierung auf alle Spells der Person
egen markepi  = max(mark==1), by(persnr begepi)
                                          // Uebertragung auf alle Spells der Person innerhalb der Episode
sort persnr spell
browse persnr spell begepi endepi mark markpers markepi quelle erwstat
```

Ein weiteres wichtiges Hilfsmittel ist der Zeilenoperator _n. Anhand des Zeilenoperators kann eine bestimmte Datenzeile angesprochen werden. Dabei ist _n die aktuelle Zeilennummer.

```
sort persnr spell
```

```
gen zeile = _n                          // neue Variable enthaelt die aktuelle Zeilennummer
*browse persnr spell zeile
```

Alternativ kann für _n auch ein absoluter Wert angegeben werden, womit eine bestimmte Zeile des Datensatzes angesprochen wird. Praktische Verwendung findet aber häufiger die Verwendung von _n plus oder minus einen bestimmten Wert, womit bestimmte Zeilen im Abstand zur aktuellen Zeile angesprochen werden.

```
by persnr, sort: gen zeilepers = _n         // Zeilennummer pro Person
                                            // (entspricht der Variablen spell)
by persnr begepi, sort: gen zeileepi = _n   // Zeilennummer pro Person und Episode
                                            // (aehnlich wie die Variable level2)
*browse persnr spell begepi zeile zeilepers zeileepi
sort persnr spell
gen summe1 = 0
replace summe1 = zeilepers                  if persnr~=persnr[_n-1]
replace summe1 = zeilepers + summe1[_n-1] if persnr==persnr[_n-1]
                                            // Aufsummieren der Zeilennummern

by persnr: gen summe2 = sum(zeilepers)      // einfacher mit der Funktion -sum-
```

3.3 Kombinationen von Zuständen mit Bitmuster-Variablen

Bitmuster-Variablen sind dann hilfreich, wenn verschiedene Kombinationen von Zuständen betrachtet werden sollen. Es ist z.B. möglich, dass eine Person zu einem Zeitpunkt Spells nicht nur aus einer, sondern aus mehreren Quellen besitzt, z.B. einen zu einer geringfügigen Beschäftigung einen anderen zu parallelem Leistungsbezug.

```
gen quelle4 = quelle                    // Zunaechst werden die Auspraegungen von quelle zusammen gefasst
recode quelle4  4 8 16 32 = 3 64 = 4    // Die neue Variable quelle4 unterschiedet nur noch
                                        // 1=BeH, 2=LeH, 3=MTG, 4=BewA
```

Eine Frage wäre, für wie viele Personen in der IEBS Spells aus allen vier Quellen BeH, LeH, MTG und BewA vorliegen. Hierzu müssen aus den vier Kategorien der Variable „quelle4“ zunächst vier Dummy-Variablen gebildet werden. Eine Möglichkeit besteht darin, mittels einer Hilfsvariable („hilf“) die Spells der betroffenen Kategorie zu markieren und diese Markierung im nächsten Schritt mit dem *egen*-Befehl auf alle weiteren Spells der Person zu übertragen.

```
gen     hilf = 1 if quelle4==1              // markiert BeH-Spells mit Wert 1
replace hilf = 0 if quelle4~=1              // alle anderen Spells erhalten den Wert 0
by persnr: egen q1 = max( hilf )            // uebertraegt die Markierung auf alle Spells der Person
```

Diese drei Zeilen können zu einer zusammengefasst werden, indem die if-Bedingung direkt in die *max*-Option eingefügt wird (siehe „q2“ bis „q4“). Mit diesen vier Dummys kann schon die Frage nach den Personen mit allen vier Quellen beantwortet werden (siehe *count*-Befehl).

```
by persnr: egen q2 = max( quelle4==2 )     // markiert Personen mit LeH-Spells  mit Wert 1, sonst 0
by persnr: egen q3 = max( quelle4==3 )     // markiert Personen mit MTG-Spells  mit Wert 1, sonst 0
by persnr: egen q4 = max( quelle4==4 )     // markiert Personen mit BewA-Spells mit Wert 1, sonst 0
browse persnr spell begepi endepi quelle4 q1 q2 q3 q4 in 1/100

* Wie viele Personen haben Spells aus allen 4 Quellen?
count if q1==1 & q2==1 & q3==1 & q4==1 & spell==1
```

Die Frage ist also mithilfe der Dummys zu beantworten. Eine Bitmuster-Variable kann aber mehr, denn sie zeigt alle in den Daten enthaltenen Kombinationen an. Die Variable besteht aus vier Stellen, eine Stelle für jede Quelle. Steht an einer bestimmten Stelle eine 1, so besitzt die Person mindestens einen Spell aus der entsprechenden Quelle. Wird an der Stelle eine Null angezeigt, so gibt es im ganzen Erwerbsverlauf dieser Person keinen Spell aus dieser Kategorie. Die Bitmuster-Variable „kom_q“ wird gebildet, indem die Dummys mit einer der Stelle entsprechenden Zehnerpotenz multipliziert und anschließend addiert werden. Steht die erste Quelle von rechts gelesen an erster Stelle, wird ihr Dummy mit 1 multipliziert. Soll die zweite Quelle von rechts gelesen an

zweiter Stelle stehen, wird ihr Dummy mit 10 multipliziert. Der Dummy der dritten Quelle wird mit 100 und der der vierten wird mit 1000 multipliziert. Die anschließende Auszählung zeigt, wie oft welche Kombination auf Spell- bzw. Personenebene vorkommt.

```
gen kom_q = q1 + q2*10 + q3*100 +  q4*1000      // Bitmuster zeigen alle Kombinationsmoeglichkeiten an
tab kom_q                                       // Spellebene
tab kom_q if spell==1                           // Personenebene
```

Die Zahl der Personen, die Spells aus allen vier Quellen besitzen, entspricht somit der Häufigkeit der Kombination „1111“ aus dem zweiten *tab*-Befehl (der erste zählt Spells, keine Personen).

3.4 Rechnen mit Datumsangaben

Datumsangaben liegen in den FDZ-Personendaten als ganzzahlige, numerische Variablen vor, die die Anzahl der Tage seit dem 1.1.1960 enthält. Durch das Zuweisen des display formates %d werden diese Variablen als Datum angezeigt. Display formates ändern Werte nicht, sondern nur, wie sie angezeigt werden.

```
list persnr begepi in 1/10    // Anzeigen von Beginndaten, die bereits das display format %d haben

format begepi %6.0f           // ein numerisches display format zuweisen
list persnr begepi in 1/10    // Anzeigen der Werte, die hinter den Daten stehen

gen dauer = endepi - begepi + 1   // Dauern in Tagen sind einfach Differenzen von Datumsangaben + 1 Tag

help date functions           // es gib eine Reihe von Funktionen, die das Rechnen mit Datumsangaben erleichtern
gen begjahr = year(begepi)    // Beispiel; year() extrahier das Jahr aus einem Datum
tab begjahr

display d(31dec1991)          // die d()-Funktion zeigt an, welche Zahl hinter einem Datum steht
keep if endepi>=d(1jan2000)   // wir behalten nur Spells, die nach dem 1.1.2000 enden
```

4 Änderungen der Datenstruktur

4.1 Verdichtung der relevanten Informationen auf eine Datenzeile pro Zeitpunkt

Die Reduktion der Daten auf einen Spell pro Zeitpunkt bzw. pro Episode ist für viele Anwendungen sinnvoll, zum Beispiel für Auszählungen zu einem Zeitpunkt, Berechnung von Übergängen oder Verweildaueranalysen. Die Variable „level2“ der IEBS nummeriert zeitlich parallele Spells, der erste erhält den Wert 0, der zweite 1, der dritte 2 u.s.w. Die Reihenfolge der Spells ist dabei nicht zufällig, so stehen BeH-Spells immer vor LeH-Spells, höheres Tagesentgelt vor geringem u.s.w. (siehe Datenreport). Der nach dieser Priorisierung wichtigste Spell steht also ganz oben. Daher wird im Folgenden *level2 == 0* als oberstes Level bezeichnet. Die Anzahl der parallelen Spells einer Episode steht in der Variablen nlevel2.

```
clear
set mem 50m
use iebs_1_0_test
tab level2                                          // in den meisten Episoden gibt es keine parallelen Spells
sort persnr begepi level2
gen quelle4 = quelle
recode quelle4  4 8 16 32 = 3 64 = 4
browse persnr begepi level2 quelle4 if nlevel2>1    // Anzeigen von Episoden mit parallelen Spells
```

Ziel der nächsten Schritte ist das Übertragen aller benötigten Informationen auf das oberste Level (*level2 == 0*). Anschließend können die darunter liegenden Level gelöscht werden. Welche Informationen zu übertragen sind, hängt vom Analyseziel ab. Bei paralleler Beschäftigung stellt sich zum Beispiel die Frage, ob nur das Entgelt der Hauptbeschäftigung betrachtet oder das Entgelt weiterer Beschäftigungen hinzu addiert werden soll. Bei der Analyse der Betriebszugehörigkeit ist zu überlegen, ob die Untersuchungseinheit das Beschäftigungsverhältnis ist und nicht die Person. Ziel wäre dann die Verdichtung der Information auf einen Spell pro Beschäftigungsverhältnis und Episode und nicht pro Person und Episode. Die Verdichtung eines Querschnitts auf eine Zeile pro Person und Zeitpunkt erfolgt analog.

Im Folgenden wird eine eindeutige Zustandsvariable pro Episode generiert, wobei Informationen aus allen Levels eingehen. Nur die Information dieser Zustandsvariablen wird auf das Level 0 übertragen.

Es werden folgende Erwerbstypen definiert:

1 = Maßnahmen der Subvention von Beschäftigungsverhältnissen

2 = Maßnahmen zur Förderung der Selbständigkeit

3 = Maßnahmen der Fort- und Weiterbildung

4 = Maßnahmen der Freien Förderung

5 = Vollzeiterwerbstätigkeit ohne Maßnahme

6 = Teilzeiterwerbstätigkeit ohne Maßnahme

7 = Arbeitslos gemeldet einschließlich arbeitsunfähig wegen Krankheit ohne Maßnahme

Jede Episode soll genau einem dieser Erwerbstypen zugeordnet werden. Das heißt, entweder schließen sich die Erwerbstypen gegenseitig aus, oder man priorisiert. Beispiel: Erwerbstypen 5 schlägt Erwerbstypen 6, das heißt, geht jemand sowohl einer Voll- als auch einer Teilzeitbeschäftigung nach, wird er der Vollzeitbeschäftigung zugeordnet. Gebildet werden die Variablen mit dem *egen*-Befehl.

```
*** Zusammenfassung der Variablen Erwerbsstatus zur Vorbereitung

tab erwstat quelle4, miss              // Kreuztabelle Erwerbsstatus - Datenquelle

* Zusammenfassung der Merkmalsauspraegungen
gen     erwstat2 = erwstat
replace erwstat2 = 104 if quelle4==1                    // sonstige Beschaeftigte
replace erwstat2 = 101 if quelle4==1 &  erwstat==101    // Beschaeftigte ohne bes. Merkmale
replace erwstat2 = 102 if quelle4==1 &  erwstat==102    // Auszubildende
```

```
replace erwstat2 = 103 if quelle4==1 & (erwstat==109 | erwstat==110                    ///
                                     | erwstat==202 | erwstat==209 | erwstat==210)
                                                    // geringfuegig Beschaeftigte
replace erwstat2 = 10002 if erwstat >=11100 & erwstat <=11101 & quelle4==3  // UEG
replace erwstat2 = 10002 if erwstat ==11801                   & quelle4==3  // ExGZ (Ich-AG)
replace erwstat2 = 10003 if erwstat >=14000 & erwstat <=14280 & quelle4==3  // FBW, TM, DSL
replace erwstat2 = 10004 if erwstat >=13000 & erwstat <=13099 & quelle4==3  // FF
replace erwstat2 = 10005 if erwstat >=15000 & erwstat <=15008 & quelle4==3  // ESF-BA
replace erwstat2 = 10001 if erwstat2>=10100 & erwstat2 ~=.    & quelle4==3  // ABM SAM LKZ EGZ EZN EV BHI
                                                              // EZV BSI AEZ BTE

cap lab drop Lerwstat2
#d ;                                                // Vergabe neuer value labels
label define Lerwstat2
-7      "-7    keine Angabe                                                  "
-9      "-9    Ohne Angabe/Keine Zuordnung möglich                           "
101     "101   Soz.-pfl. Beschaeftigte ohne besondere Merkmale               "
102     "102   Auszubildende                                                 "
103     "103   Geringfuegig Beschaeftigte                                    "
104     "104   sonstige Beschaeftigte                                        "
1       "1     ALG Arbeitslosengeld                                          "
2       "2     ALHI Arbeitslosenhilfe                                        "
3       "3     UHG Unterhaltsgeld                                            "
10001   "10001 ABM SAM LKZ EGZ EZN EV BHI EZV BSI AEZ BTE                    "
10002   "10002 UEG ExGZ (Ich-AG)                                             "
10003   "10003 FBW, TM, DSL                                                  "
10004   "10004 FF                                                            "
10005   "10005 ESF-BA                                                        "
31      "31    arbeitslos (impliziert: bei der BA arbeitsuchend gemeldet)    "
32      "32    arbeitsunfaehig waehrend der Arbeitslosigkeit (generiert, s.o.)"
33      "33    nicht arbeitslos, aber bei der BA arbeitsuchend gemeldet      "
;
#d cr
label values erwstat2 Lerwstat2, nofix


*** "Hochziehen" von Informationen auf den obersten Level einer Episode (level2==0):

sort persnr begepi
by persnr begepi: egen vollz = max( quelle4==1 & stib>=-9 & stib<=7 | quelle4==4 & erwstat2==33)
by persnr begepi: egen teilz = max( quelle4==1 & (stib==8 | stib==9))
```

```
by persnr begepi: egen alos = max( quelle4==4 & (erwstat2==31 | erwstat2==32)      ///
                | quelle4==2 & (erwstat2==1 | erwstat2==2)                    )
by persnr begepi: egen byte  ABMua = max( quelle4==3 & erwstat2==10001)
by persnr begepi: egen byte  UEGua = max( quelle4==3 & erwstat2==10002)
by persnr begepi: egen byte  FBWua = max( quelle4==3 & erwstat2==10003)
by persnr begepi: egen byte  FF    = max( quelle4==3 & erwstat2==10004)
```

An dieser Stelle wurde noch nicht priorisiert, so dass häufig mehrere der 7 Erwerbstypen gleichzeitig auftreten dürften. Eine Auszählung der Kombinationen lässt sich leicht anhand einer Bitmuster-Variable erstellen (siehe hierzu Seite 8).

```
gen long kom_zu = vollz*1000000 + teilz*100000 + alos*10000 + ABMua*1000 + UEGua*100 + FBWua*10 + FF
format kom_zu %10.0f
tab kom_zu, miss
tab kom_zu erwstat2                                                                  // zur Kontrolle
```

Es gibt Spells mit *kom_zu == 0*, die folglich noch keinem der 7 Erwerbstypen zugeordnet wurden. Anhand der letzten Auszählung erkennt man, dass dies Episoden mit ausschließlich Unterhaltsgelt (UHG) oder mit Zuschüssen aus dem BA-Programm mit Mitteln aus dem Europäischen Sozialfonds (ESF-BA) sind. Diese zwei Fälle sollen im Folgenden nicht weiter betrachtet. In der Auszählung gibt es anschließend keine Nullen mehr, d.h. alle Spells sind mindestens einem Erwerbstyp zugeordnet.

```
drop if kom_zu==0      // Episoden mit ausschliesslich UHG oder ESF-BA werden gelöscht.
tab kom_zu, miss       // jetzt gibt es keine Nullen mehr
```

Durch Priorisierung soll schließlich die eigentliche Zustandsvariable gebildet werden, die pro Episode eindeutig ist. Die Priorisierung erfolgt durch das Festlegen der Reihenfolge, in der die einzelnen Kategorien der Zustandsvariablen generiert werden. Durch die zweite if-Bedingung (*zustand* == .) wird gewährleistet, dass die im vorherigen Schritt gebildete Ausprägung nicht überschrieben wird.

```
*** Bildung eines eindeutigen Erwerbstyps pro Episode per Dominanzprinzip:
```

```
gen     zustand = 1 if ABMua==1                    // durch Reihenfolge ergibt sich die Prioritaet
replace zustand = 2 if UEGua==1 & zustand==.
replace zustand = 3 if FBWua==1 & zustand==.
replace zustand = 4 if FF   ==1 & zustand==.
replace zustand = 5 if vollz==1 & zustand==.
replace zustand = 6 if teilz==1 & zustand==.
replace zustand = 7 if alos ==1 & zustand==.

tab zustand, miss     // Zustand ist fuer Episoden missing, die nicht zugeordnet werden konnten.

label define Lzustand 1 "1 ABMua" 2 "2 UEGua" 3 "3 FBWua" 4 "4 FF" 5 "5 vollz" 6 "6 teilz" 7 "7 alos"
label values zustand Lzustand

* vollz teilz alos ABMua UEGua FBWua FF
tab kom_zu zustand, miss                   // Vergleich mit kom_zu zur Kontrolle: passt
                                           // aber: Mehrfachzaehlung paralleler Spells

keep if level2==0                          // nur das oberste Level wird behalten
tab kom_zu zustand, miss                   // Auszaehlung einmal pro Episode
```

4.2 Episodensplitting rückgängig machen

Episodensplitting bedeutet, dass der Datensatz so aufbereitet ist, dass keine sich überschneidenden Intervalle mehr existieren, sondern nur noch vollständig parallele Zeiträume (siehe Datenreport für Details). Wenn wir die Spells markieren, die durch das Episodensplitting zusätzlich erzeugt wurden, können wir auszählen, wie häufig das Episodensplitting in einzelnen Jahren vorkommt.

```
use iebs_1_0_test, clear
gen splitspell = begorig~=begepi           // markieren der zusaetzlich generierten Spells
gen begjahr = year(begepi)
tab begjahr quelle if splitspell==1        // ab 2000 werden deutlich mehr kuenstliche Spells eingefuegt
```

Generell gilt, dass mehr Datenquellen zu mehr Überschneidungen und somit zu mehr künstlich erzeugten Spells führen. Dies spiegelt sich in der obigen Auszählung durch den starken Anstieg der Zahl der gesplitteten Spells wider, da nicht alle Quellen von Anfang an vollständig gefüllt sind. Das Episodensplitting führt auch zu Besonderheiten bei der Betrachtung im Querschnitt. So ist bei der Auszählung von Querschnitten unbedingt auf die Wahl der Datumsangabe zu achten (Originalspell oder Episode). Im Folgenden soll ausgezählt werden, wie viele Frauen und wie viele Männer zum Stichtag 31. Dezember 2000 im Datensatz vorhanden sind. Während die erste und zweite Auszählung noch zum richtigen Ergebnis führen, zählt die dritte neben den Originalspells auch die durch das Splitting künstlich generierten Spells. Die vierte Auszählung gibt dazu die erklärende Aufschlüsselung: Die Differenz zwischen richtiger und falscher Auszählung entsteht durch die zusätzlich gebildeten Spells.

```
tab sex if begepi <=d(31dec2000) & endepi >=d(31dec2000)                    // richtig
tab sex if begorig<=d(31dec2000) & endorig>=d(31dec2000) & begorig==begepi  // richtig
tab sex if begorig<=d(31dec2000) & endorig>=d(31dec2000)                    // falsch: Doppeltzaehlungen!
tab sex splitspell if begorig<=d(31dec2000) & endorig>=d(31dec2000)         // Erklärung für Doppeltzählung
```

Für bestimmte Forschungsfragen ist das Episodensplitting eher hinderlich. Die dadurch zusätzlich generierten Spells können jedoch ohne Probleme wieder gelöscht werden. Da alle aus einem Originalspell entstandenen Episoden dieselbe Satznummer haben, kann man den Datensatz zum einen auf einen Spell pro Satznummer beschränken:

```
duplicates drop satznr, force
```

Nachteil der Vorgehensweise ist, dass die Auswahl der gelöschten und bewahrten Spells zufällig ist. Manche Merkmale wie die Dauer der Arbeitslosigkeit werden jedoch nach dem Episodensplitting also für jede Episode getrennt berechnet. Die Werte unterscheiden sich also innerhalb einer Satznummer. Möchte man diese Merkmale nach dem Löschen der zusätzlich generierten Spells auswerten, muss man sich genauer überlegen, welche Spells man löscht, damit die Information in diesen Merkmalen auf

das Datumsangaben des Originalspells zutrifft. I.d.R. behält man dazu entweder den zeitlich ersten oder den letzten Spell innerhalb einer Satznummer:

```
keep if begorig==begepi   // behält den ersten Spell
```

oder

```
keep if endorig==endepi   // behält den letzten Spell
```

4.3 Erstellung von Querschnitten oder Paneldaten

Ein Querschnitt lässt sich aus der IEBS mit einem Befehl erstellen. Es werden alle Spells behalten, die einen Stichtag überlappen.

```
keep if begepi<=d(31dec1991) & endepi>=d(31dec1991)        // im Datensatz bleiben nur noch nur Episoden,
                                                           // die den 31.12.1991 einschliessen
```

Im Folgenden sollen die Querschnitte der Jahre 1997 bis 2003 zu einem Panel zusammengefügt werden. Dafür wird zunächst völlig analog zu oben für jedes Jahr ein Querschnitt gebildet und abgespeichert. Die Variable „jahr“ zeigt an, aus welchem Jahr die Beobachtung kommt.

```
use iebs_1_0_test, clear
keep if begepi<=d(31dec1997) & endepi>=d(31dec1997)
gen jahr = 1997
save quer1997, replace

use if begepi<=d(31dec1998) & endepi>=d(31dec1998) using iebs_1_0_test, clear   // -use- und -keep- in
gen jahr = 1998                                                                 // einem Schritt
save quer1998, replace
```

```
use if begepi<=d(31dec1999) & endepi>=d(31dec1999) using iebs_1_0_test, clear
gen jahr = 1999
save quer1999, replace

use if begepi<=d(31dec2000) & endepi>=d(31dec2000) using iebs_1_0_test, clear
gen jahr = 2000
save quer2000, replace

use if begepi<=d(31dec2001) & endepi>=d(31dec2001) using iebs_1_0_test, clear
gen jahr = 2001
save quer2001, replace

use if begepi<=d(31dec2002) & endepi>=d(31dec2002) using iebs_1_0_test, clear
gen jahr = 2002
save quer2002, replace

use if begepi<=d(31dec2003) & endepi>=d(31dec2003) using iebs_1_0_test, clear
gen jahr = 2003
save quer2003, replace
```

Die Querschnitte werden untereinander gehängt. Der so entstandene Datensatz ist ein Panel.

```
use          quer1997, clear   // Untereinanderschreiben der zuvor
append using quer1998          // gespeicherten Querschnitte.
append using quer1999
append using quer2000
append using quer2001
append using quer2002
append using quer2003
```

Wiederholen sich die Befehle immer wieder und ändern sich nur einzelne Werte in den Befehlen, eignen sich Schleifen dazu, den Befehlsblock erheblich zu verkürzen. Im Folgenden wird das gleiche Panel generiert, diesmal mithilfe zweier Schleifen: Eine zur Bildung der Querschnitte, die andere zum Zusammenführen der Querschnitte zu einem Panel.

```
foreach jahr of numlist 1997/2003 {
```

```
      use if begepi<=d(31dec`jahr') & endepi>=d(31dec`jahr') using iebs_1_0_test, clear
      gen jahr = `jahr'
      save quer`jahr', replace
      }

use quer1997, clear
foreach jahr of numlist 1998/2003 {
      append using quer`jahr'
      }

save panel, replace
```

4.4 Wechsel zwischen long und wide-Format bei Paneldaten mit reshape

In diesem Abschnitt soll gezeigt werden, wie Veränderungen zwischen zwei Zeitpunkten mithilfe einer sogenannten Übergangsmatrix ausgezählt werden können. Es soll im Folgenden der Wohnort der Personen vom Jahr 1999 mit dem vom Jahr 2003 verglichen und eventuelle Wohnortwechsel ausgezählt werden. Hierzu wird analog zu oben ein Panel aus den Jahren 1999 und 2003 gebildet.

```
use if begepi<=d(30jun1999) & endepi>=d(30jun1999) using iebs_1_0_test, clear
gen jahr = 1999
save querauf1999, replace

use if begepi<=d(30jun2003) & endepi>=d(30jun2003) using iebs_1_0_test, clear
gen jahr = 2003
save querauf2003, replace

use querauf1999, clear            // untereinander schreiben der zuvor
append using querauf2003          // gespeicherten Querschnitte.
save panelauf, replace
```

Anhand der Variable „level2“ erkennt man, dass es noch parallele Spells in den einzelnen Querschnitten gibt. Zur Vereinfachung sollen parallele Spells gelöscht werden. Diese Vorgehensweise ist unproblematisch, wenn der Wohnort in den Daten konsistent über alle Quellen vorliegt, was bedeutet, es existiert pro Zeitpunkt (hier der 30. Juni des Jahres) nur ein gültiger Wohnort. Ist dies nicht der Fall, müssen vorher diese Inkonsistenzen bereinigt werden. Davon wird hier abgesehen.

```
tab level2                          // es gibt parallele Spells
keep if level2==0                   // Achtung: Vereinfachung!
tab level2
```

Für die Analyse relevant sind nur drei Variablen: Die Personennummer („persnr“), das Jahr („jahr“) und der Wohnort („wo_bula“). Alle anderen werden gelöscht (*keep*). Ein Trick soll die Auszählung der Übergänge nun erheblich vereinfachen. Der *reshape*-Befehl erlaubt es, den Datensatz vom langen Format zum breiten Format umzuformen. Der Datensatz ist momentan in der langen Form, das heißt, es gibt pro Person mehrere Spells, für jedes Jahr einen. Dementsprechend ist der Wohnort in einem bestimmten Jahr in einer *Zeile* abgespeichert. Wird der Datensatz nun ins breite Format umgewandelt, besitzt jede Person nur noch genau einen Spell. In der durch den *reshape*-Befehl generierten Variable „wo_bula1999“ ist der Wohnort von 1999, in der Variable „wo_bula2003“ ist der Wohnort von 2003 abgelegt. Das bedeutet, dass der Wohnort in einem bestimmten Jahr nun in einer *Spalte* abgespeichert ist.

```
keep persnr jahr wo_bula
sort persnr jahr
reshape wide wo_bula, i(persnr) j(jahr)    // -reshape- erlaubt den Wechsel zwischen lang und breit
```

Mt *reshape long* kann man wieder in die ursprüngliche Form zurück wechseln. Mithilfe einer einfachen Kreuztabelle zwischen den zwei Wohnortvariablen können im wide Format Wohnortwechsel ausgezählt werden. Diese Tabelle ist die Übergangsmatrix.

```
tab wo_bula1999 wo_bula2003, miss
```

5 Bereinigung von Inkonsistenzen

5.1 Staatsangehörigkeit

Zu den Variablen, die aus statistischen Gründen von den Arbeitgebern gemeldet werden und damit eine geringere Validität aufweisen als Merkmale die einen Anspruch gegen die Sozialversicherungen begründen, wie das Entgelt, zählt die Staatsangehörigkeit. Im Folgenden soll anhand dieser Variablen demonstriert werden, wie eine Bereinigung von Inkonsistenzen durchgeführt werden kann. Der folgende Stata-Code ist ausdrücklich nicht als endgültige Bereinigung der Staatsangehörigkeitsvariablen zu verstehen, sondern als Vorschlag, welche Schritte möglich sind. Darüber hinaus ist die Berücksichtigung von Betriebsinformationen denkbar oder eine unterschiedliche Behandlung von Information aus verschiedenen Quellen. Die hier vorgestellte Systematik ist mit entsprechenden Anpassungen ebenfalls auf andere Variablen übertragbar. Grundsätzlich besteht bei der Bereinigungen von Inkonsistenzen immer ein Trade-off zwischen weniger Inkonsistenzen und der Gefahr, eine falsche Angabe einzufügen. Letztlich bleibt es dem Nutzer selbst überlassen, welche Schritte er vornimmt.

Anhand ausgewählter Personen werden im Folgenden die einzelnen Schritte betrachtet. Diese werden in einer nicht beliebigen Reihenfolge durchgeführt; das Vorgehen ist sukzessive. Begonnen wird mit Schritten, die mit höherer Wahrscheinlichkeit nicht zum Einfügen fehlerhafter Informationen führen. Zum Schluss werden sehr unsichere Heuristiken vorgestellt. In einem ersten Schritt wird ein erster Eindruck von der Variable „nation“ und von der Häufigkeit der Missings gewonnen. Es wird eine weitere Variable („nation2“) gebildet, um Änderungen in der Staatsangehörigkeitsvariablen besser verfolgen zu können.

```
use nation, clear                                  // Die Datei nation.dta steht am selben Ort wie der Methodenreport
                                                   // zum Download bereit
order persnr spell begepi endepi level2 nation
sort persnr spell
tab nation in 1/500
tab nation if nation == -7                         // Anzahl der Missings
```

```
gen nation2 = nation                                  // Bildung einer neuen Variable fuer Staatsangehoerigkeit
```

Als nächstes sollen diejenigen Spells markiert werden, in denen es bei derselben Person zu einem Wechsel der Staatsangehörigkeit kommt („sta_wec“). Die Gesamtzahl der Wechsel einer Person wird daraufhin auf alle Spells der Person übertragen („sta_wec_per“).

```
gen sta_wec = 0                                       // Dummy-Variable, die jeden Wechsel anzeigt
replace sta_wec = 1 if nation != nation[_n-1] & persnr == persnr[_n-1]
br if persnr == 15
by persnr: egen sta_wec_per = sum(sta_wec)            // Anz.d.Wechsel wird auf die gesamte Person uebertragen
br if sta_wec_per != 0
tab sta_wec_per if spell                              // wie viele Personen haben wie haeufig die Nationalität gewechselt
```

Im ersten Fall, der bereinigt werden soll, besitzt eine Person innerhalb einer Episode mehrere parallele Spells, in denen mindestens ein Spell mit fehlendem Wert (Missing) vorkommt. Dieser Spell soll die Staatsangehörigkeit des oder der parallelen und gültigen Spells (Nicht-Missings) annehmen. Hierfür werden zunächst ungültige Werte nach hinten sortiert und durch den Wert des Vorgängerspells ersetzt. Es wurde nicht für Fälle kontrolliert, die mehr als eine gültige Staatsangehörigkeit und eine Meldung mit ungültigem Wert innerhalb einer Episode aufweisen (nicht in den Testdaten enthalten).

```
gsort persnr begepi -nation2
by persnr: replace nation2 = nation2[_n-1] if begepi == begepi[_n-1] & nation2 == -7
                                                      // Missings innerhalb einer Episode werden auf den Wert
                                                      // des oder der parallelen Spells umgesetzt
br if persnr == 15, nol
tab nation2 if nation2 == -7                          // 33 von 1730 Missings koennen ersetzt werden
```

Im zweiten Bereinigungsfall besitzt eine Person innerhalb einer Episode verschiedene Staatsangehörigkeiten. Diese Inkonsistenz soll behoben werden, indem diese Spells auf Missing gesetzt werden. Hierfür markiert man zunächst alle Spells (mithilfe der Variable „sta_ink“), die eine andere Staatsangehörigkeit angeben als der unmittelbar vorhergehende Spell derselben Person in

derselben Periode. Danach überträgt man diese Markierung auf alle Spells dergleichen Episode (*replace* in Verbindung mit *sum*). Mithilfe dieser Markierung lassen sich schließlich diese Spells auf Missing (-9) setzen. Als fehlender Wert wird nicht die -7 sondern die -9 gewählt, um später die Originalmissings (-7) von den durch die Bereinigung entstandenen Missings (-9) unterscheiden zu können.

```
gen sta_ink = 0
by persnr: replace sta_ink = 1 if begepi == begepi[_n-1] & nation2 != nation2[_n-1]
                                                  // Dummy, der anzeigt ob verschiedene
                                                  // Staatsangehoerigkeiten waehrend einer Episode vorliegen
br if persnr == 42, nol
gsort persnr begepi -sta_ink
by persnr begepi: replace sta_ink = sum(sta_ink) // Uebertragen der Dummy-Variable auf die Episode;
                                                  // Variable ist immer dann nicht null wenn zwei
                                                  // unterschiedliche Staatsangehoerigkeiten vorliegen
br if persnr == 42, nol
replace nation2 = -9 if sta_ink != 0              // Behebung der Inkonsistenz, indem auf Missing gesetzt wird

order persnr spell begepi endepi nation nation2
sort persnr spell
br if persnr == 42, nol

tab nation if nation < 0                          // Es liegen 1730 Missings in der Original-Variable vor.
tab nation2 if nation2 < 0                        // 1869 Missings in der neuen Variable
                                                  // davon sind 172 neu erzeugte Missings
```

Im nächsten Schritt erweitert sich der Bereinigungsvorgang auf eine episodenübergreifende Betrachtung. Hierzu müssen zunächst wieder die Wechsel in der Variable Staatsangehörigkeit völlig analog zu „sta_wec“ markiert werden (jetzt heißt die Variable „help“). Dieser Schritt ist erneut notwendig, da durch die bisher erfolgten Bereinigungen ehemalige Staatsangehörigkeitswechsel, die durch Missings ausgelöst wurden, verschwunden sind, so dass die Variable „sta_wec“ nicht mehr verwendet werden kann. Auch hier wird die Markierung anschließend wieder auf alle Spells der Person übertragen („sta_wec2“).

```
sort persnr nation2                                  // Innerhalb der Personen wird nach Nationen sortiert
gen help = 0
replace help = 1 if persnr == persnr[_n-1] & nation2 != nation2[_n-1]
                                                     // Hilfsvariable ist immer dann 1, wenn die Nationalität wechselt
br if persnr == 45, nol
by persnr: egen sta_wec2 = sum(help)                 // Hilfsvariable wird aufsummiert
br if persnr == 45, nol
tab sta_wec2 if spell == 1                           // Anzahl der Wechsel auf Personenebene
```

Ziel des nächsten Bereinigungsschrittes ist es, Spells von Personen zu verbessern, die neben fehlenden Werten in der Variablen „nation“ genau eine Staatsangehörigkeit *(sta_wec2 == 1)* besitzen. Hier ist es plausibel, die Missings einfach in die vorhandene Staatsangehörigkeit umzuwandeln. Dies wird umgesetzt, indem die Missings einer Person nach unten sortiert werden und mit *replace* der Wert vom vorangegangenen Spells übertragen wird.

```
gsort persnr -nation2                                // Missings werden nach hinten sortiert
br if persnr == 15, nol
replace nation2 = nation2[_n-1] if nation2 < 0 & sta_wec2 == 1 & persnr == persnr[_n-1]
                                                     // Missings werden ersetzt, wenn nur eine
                                                     // Staatsangehoerigkeit vorliegt
br if persnr == 15, nol
tab nation2 if nation2 < 0                           // Von den 1730 Original-Missings verbleiben noch 1497
```

Folgende Fälle sollen im Weiteren geklärt werden: Eine Person besitzt eine Staatsangehörigkeit, die von einer anderen Staatsangehörigkeit in der Episode davor und danach eingeschlossen wird. Genauer gesagt darf nur genau ein Spell eingeschlossen sein, denn es scheint unplausibel, dass jemand seine Staatsangehörigkeit wechselt, um in der darauffolgenden Periode wieder seine alte anzunehmen. Um den Zustand vor und nach der Bereinigung vergleichen zu können, wird wieder eine neue Nationenvariable gebildet („nation3“). An dieser Stelle sei erneut auf den angesprochenen Trade-off zwischen Bereinigung und der Gefahr, eine fehlerhafte Information einzufügen, hingewiesen. Es ist zwar unwahrscheinlich, aber nicht unmöglich, dass eine Person ihre Staatsangehörigkeit wechselt, um nach einer sehr kurzen Zeit zu Ihrer ursprünglichen Staatsangehörigkeit zurückzuwechseln.

```
sort persnr spell
br if persnr == 69
gen nation3 = nation2
replace nation3 = nation3[_n-1] if nation3[_n-1] >0 & nation3[_n-1] == nation3[_n+1]                    ///
                                                                    & persnr[_n-1] == persnr[_n+1]
order persnr spell begepi endepi nation nation2 nation3
br if persnr == 69
```

Es existieren nun immer noch eine ganze Reihe von Originalmissings (-7). Bei Personen, die neben Originalmissings gültige Staatsangehörigkeitsspells besitzen, sollen diese Missings überschrieben werden, indem sie die Nationalität des vorhergehenden Spells annehmen. Es ist klar, dass dieser Schritt ein gewisses Risiko beinhaltet, eine falsche Information zu übertragen. Die Nationalität des vorhergehenden Spells muss beispielsweise nicht unbedingt der bei dieser Person vorherrschenden Nationalität entsprechen. Dieser Bereinigungsschritt liegt begründet in der Tatsache, dass es einfach keine Information darüber gibt, warum der Spell einen fehlenden Wert beinhaltet, und dass man keine plausiblere Staatsangehörigkeit vermuten kann, als die in der unmittelbar vorangegangenen Episode. Nach diesem Bereinigungsschritt existieren Originalmissings (-7) nur noch bei denjenigen Personen, die ausschließlich Originalmissings besitzen.

```
replace nation3 = nation3[_n-1] if nation3 == -7 & persnr == persnr[_n-1]
                                          // Originale Missings werden durch Vorgaenger-ST ersetzt
br if persnr == 4674
```

Übrig bleiben jedoch weiterhin die durch den Bereinigungsprozess entstandenen Missings (-9). Der nächste Schritt ähnelt inhaltlich der Bereinigung, bei der eingeschlossene Staatsangehörigkeiten durch die Nationalität der einschließenden Spells ersetzt wurden. Der Unterschied besteht nun hauptsächlich darin, dass *mehrere* Spells eingeschlossen sind. Spells wurden immer dann auf -9 gesetzt, wenn eine Person innerhalb einer Episode verschiedene Staatsangehörigkeiten besaß (siehe oben). Ein weiterer Unterschied besteht in der technischen Vorgehensweise. Die Tatsache, dass mehr als ein Spell eingeschlossen ist, erschwert die Bereinigung etwas.

Es werden zunächst zwei Hilfsvariablen gebildet („help“ und „help2“). Die erste Variable nimmt die Nationalität des ersten gültigen Spells nach den Missings an. Die zweite Variable nimmt die Nationalität des Spells vor dem ersten Missing an. Entscheidend bei dieser Vorgehensweise ist die Sortierung vor dem *by persnr: replace*-Befehl. Dieser ist für beide Variablen gleich. Nur werden für die „help“-Variable die Zustände umgekehrt chronologisch sortiert (*gsort persnr –spell*), während für die „help2“-Variable chronologisch sortiert wird (*sort persnr spell*). Im Gegensatz zu *sort* erlaubt der *gsort*-Befehl auch absteigende Sortierungen (mithilfe des Minuszeichens vor der Variablen).

Die eingeschlossenen Missings nehmen schließlich die Staatsangehörigkeit der einschließenden Spells an (siehe *replace*). Dort, wo die Nationalitäten vor und nach den Missings nicht übereinstimmen, bleiben die Werte unverändert bei -9.

```
                                                // Personen aber umgedreht
replace help = 0
gen help2 = 0
replace help = nation3
by persnr: replace help = help[_n-1] if help == -9
                                                // Bildung einer Variable, die ST aus den nachfolgenden
                                                // Spells enthaelt wenn ST Missing ist.

sort persnr spell
replace help2 = nation3
by persnr: replace help2 = help2[_n-1] if help2 ==-9
                                                // Bildung einer Variable, die ST aus den vorangegangenen
                                                // Spells enthaelt, wenn ST Missing ist.
* Der Sinn der beiden Dummy-Variablen besteht darin, dass man nun die ST vor und nach den neu gebildeten
* Missings fuer die betreffenden Beobachtungen kennt.
order help help2
br if persnr == 288
tab nation3 if nation3 < 0
replace nation3 = help if help == help2 & nation3 == -9
                                                // Wenn die ST vor und nach den Missings identisch ist,
                                                // dann werden alle Missings ersetzt.
tab nation3 if nation3 < 0
```

Besitzt der erste Spell nach den Missings dieselbe Staatsangehörigkeit wie der letzte Spell der Person, so sollen die eingeschlossenen Missings diese Nationalität übernehmen. Dieser Schritt kann wie folgt begründet werden: Es hat tatsächlich einen Wechsel der Staatsangehörigkeit gegeben (sonst wäre die Variable nicht auf -9 gesetzt worden); da Informationen von parallel liegenden Spells in fast allen Fällen entweder aus verschiedenen Meldeverfahren oder von verschiedenen Arbeitgebern kommen, ist es nicht unwahrscheinlich, dass ein Spell noch die alte, der parallele Spell jedoch schon die neue Nationalität anzeigt (es ist zum Beispiel denkbar, dass der alte Arbeitgeber bei einer erneuten Meldung die alte Staatsangehörigkeit übernimmt, während der neue Arbeitgeber auf Nachfrage beim Arbeitnehmer die neue Nationalität erhält).

Um diese Bereinigung umzusetzen, muss zunächst eine Variable gebildet werden, die in jeden Spell einer Person die letzte Staatsangehörigkeit dieser Person schreibt („sta_let“). Stimmt die Nationalität nach den Missings („help“-Variable aus vorherigem Schritt) mit der letzten Nationalität der Person überein, so erhalten die Missings diese Nationalität. Um die Änderungen besser nachvollziehen zu können, wird wieder eine neue Nationenvariable gebildet („nation4“).

```
gen sta_let = 0
by persnr: replace sta_let = nation3[_N]              // letzte ST der Person
gen nation4 = nation3
replace nation4 = help if nation4 == -9 & help == sta_let
                                                      // Immer wenn die ST Missing ist und die ST des
                                                      // darauffolgenden Spells der letzten ST enstpricht,
                                                      // dann wird ST des darauffolgenden Spells uebernommen.
order persnr spell begepi endepi nation nation2 nation3 nation4
br if persnr == 307
```

Da nun für die verbleibenden Missings (-9) keine Informationen mehr vorhanden sind, die eine Bereinigung anhand einigermaßen plausiblen Annahmen ermöglichen, werden im vorletzten Schritt alle verbleibenden Spells mit -9 pauschal mit der Nationalität des unmittelbar vorhergehenden Spells der Person überschrieben.

```
br if persnr == 926
```

```
replace nation4 = nation4[_n-1] if persnr == persnr[_n-1] & nation4 == -9
br if persnr == 926
```

Zum Schluss soll erneut ein Bereinigungsschritt wiederholt werden. Es werden direkt eingeschlossene Staatsangehörigkeiten durch die einschließende Staatsangehörigkeit ersetzt. Dieser Schritt ist notwendig, da durch die bisherigen Behebungen von Inkonsistenzen neue Einschlüsse entstanden sind, die vorher nicht existierten.

```
br if persnr == 2827
replace nation4 = nation4[_n-1] if persnr[_n+1] == persnr[_n-1] & nation4[_n-1] == nation4[_n+1]
br if persnr == 2827
```

Diese Sequenz von Bereinigungsschritten soll gezeigt haben, wie sukzessiv verschiedene Informationen genutzt werden können, um auf der einen Seite Inkonsistenzen zu beheben und auf der anderen Seite zu versuchen, mithilfe von Plausibiltätsüberlegungen fehlende durch gültige Werte zu ersetzen. Dabei ist von Schritt zu Schritt die Stichhaltigkeit der Annahmen gesunken und somit das Risiko gestiegen, Inkonsistenzen falsch zu beheben oder falsche Werte für Missings zu setzen. Wie weit man bei einer solchen Bereinigung gehen will, ist jedem Nutzer selbst überlassen und hängt von der Fragestellung der Analyse ab.

5.2 Lücken im Erwerbsverlauf: Einfügen von sehr wahrscheinlich fehlenden Meldungen

Betrachtet man die Erwerbsverläufe von Personen genauer, so stößt man bei vielen auf Lücken im Erwerbsleben. Einige dieser Lücken weisen besondere Kennzeichen auf: Sie erstrecken sich genau vom 1. Januar bis zum 31. Dezember eines Jahres, und sowohl davor als auch danach befindet sich ein Beschäftigungsspell (BeH). Handelt es sich vor und nach der Lücke um dasselbe Beschäftigungsverhältnis, so kann man davon ausgehen, dass eine reguläre Jahresmeldung fehlt. Es soll nun versucht werden, diese Lücken zu schließen. Dieses Füllen von Lücken wird bei Erzeugung der IAB-Beschäftigtenstichprobe (IABS) tatsächlich durchgeführt (sog. Ergänzungsverfahren), nicht jedoch für die IEBS.

Um eine Lücke feststellen zu können, wird zunächst eine Variable gebildet („gap1“), die die Anzahl der Tage zwischen zwei aufeinanderfolgenden Spells berechnet. Hierzu wird vorher chronologisch sortiert. Eine Hilfsvariable („help1“) soll anzeigen, bei welchen Spells diese Lücke am 1. Januar des jeweiligen Jahres beginnt und genau ein Jahr dauert. Dabei darf nicht vergessen werden, dass ein Schaltjahr nicht 365, sondern 366 Tage dauert. Die Variable „kont“, die im ersten Befehl gebildet wird, ist eine Kontrollvariable, die später zum Einsatz kommt.

```
use if level1 == 0 using iebs_1_0_test, clear              // Mehrfachbes. ausgeschlossen
gen kont = 0
sort persnr spell
gen gap1 = 0
replace gap1 = (begepi-endepi[_n-1]-1) if persnr == persnr[_n-1] & level2 == 0
                                                  // Variable enthaelt den zeitlichen Abstand zweier Spells
gen help1 = 0
replace help1 = 1 if gap1 == 365 &(year(begepi) != 1993 & year(begepi) != 1997 & year(begepi) != 2001)    ///
                  & day(begepi) == 1 & month(begepi) == 1
replace help1 = 1 if gap1 == 366 &(year(begepi) == 1993 | year(begepi) == 1997 | year(begepi) == 2001)    ///
                  & day(begepi) == 1 & month(begepi) == 1
                                                  // Hilfsvariable = 1 wenn der Abstand zwischen zwei
                                                  // Spells genau ein Jahr (Schaltjahre beachten) betraegt und
                                                  // das Beginndatum der Spells der 1.1. ist.
```

Die Variable help1 markiert genau die Lücken, die vom 1. Januar bis 31. Dezember eines Jahres dauern. Eine weitere Hilfsvariable (“help2”) soll nun die Fälle markieren, in denen vor und nach der Lücke dasselbe Beschäftigungsverhältnis besteht. Ein gleiches Beschäftigungsverhältnis liegt vor, wenn der Arbeitnehmer davor im selben Betrieb („betnr“), im selben Beruf („beruf“) und in derselben Branche („w93“) tätig ist (diese Kriterien stellen nur ein Beispiel dar; man kann natürlich noch weitere hinzuziehen).

```
gen help2 = 0
replace help2 = 1 if persnr == persnr[_n-1] & quelle == 1 & quelle[_n-1] == 1                          ///
                  & beruf == beruf[_n-1] & w93 == w93[_n-1]  & betnr == betnr[_n-1]
                                        // Hilfsvariable = 1 wenn Beruf, WZW und Betriebsnummer zweier
```

```
                                        // aufeinander folgender Beschaeftigungen uebereinstimmen
```

Nun werden die fehlenden Jahresmeldungen generiert, die die Lücken füllen sollen. Zuerst ist es nötig, den bisher modifizierten Datensatz zwischenzuspeichern. Dann wird für jede Lücke, die durch die oben gebildeten Hilfsvariablen („help1“ und „help2“) markiert ist, der unmittelbar darauffolgende Spell behalten. Die restlichen werden gelöscht (siehe *keep if*). Diese so herausgelesenen Spells bilden die Grundlage, aus denen die fehlenden Jahresmeldungen generiert werden. Dazu wird das Datum der Spells genau um ein Jahr zurückgesetzt („begepi“ und „endepi“). Dabei muss wieder darauf geachtet werden, dass die Jahre 1992, 1996 und 2000 Schaltjahre waren. Der so entstandene Datensatz, der nun alle fehlenden Jahresmeldungen enthält, wird abgespeichert. Die auf diese Weise künstlich generierten Spells werden durch die oben erwähnte Kontrollvariable („kont“) markiert, um sie als solche später identifizieren zu können.

```
save zwischen, replace                  // Daten zwischenspeichern
keep if help1 == 1                      // "Entwurf"-Spells für Spells, die eingefügt werden
replace kont = 1                        // Variable = 1 fuer Spells, die spaeter eingesetzt werden, zur Kontrolle

replace begepi = begepi-365             // Datum um ein Jahr zuruecksetzen
replace begepi = begepi-1 if year(begepi) == 1992 | year(begepi) == 1996 | year(begepi) == 2000
replace endepi = begepi+364             // Schaltjahre beachten
replace endepi = endepi+1 if year(endepi) == 1992 | year(endepi) == 1996 | year(endepi) == 2000
                                        // Enddatum enstpricht Beginndatum plus einem Jahr,
                                        // wiederum Schaltjahre beachten
br
save einsatz, replace                   // einzufuegende Spells zwischenspeichern
```

Die Jahresmeldungen müssen nun nur noch an den regulären Datensatz angehängt und durch chronologische Sortierung in die jeweilige Lücke gesetzt werden.

```
use zwischen, clear
append using einsatz                    // Zusaetzliche Spells werden hinzugespielt
```

```
sort persnr begepi
```

Es sei abschließend daraufhingewiesen, dass gegebenenfalls Informationen in den eingefügten Spells angepasst werden müssen. Sie enthalten beispielsweise das gleiche Originalbeginn- und –endedatum („begorig“ und „endorig“) und die gleiche Spellnummer („spell“) wie der nachfolgende Spell.

6 Anwendungen

6.1 Identifikation von an der Beitragsbemessungsgrenze zensierten Entgelten

Die Beitragsbemessungsgrenzen hängen von Jahr, Landesteil und Rentenversicherung ab. Man kann die 4 Werte pro Jahr per hand eingeben oder aus den Daten bestimmen. Es ist jedoch nicht das höchste Tagesentgelt im Datensatz, da es in Ausnahmefällen zu Entgelten oberhalb der Grenze kommen kann, sondern der häufigste Wert größer Null. Mit *egen* und der *mode*-Funktion lässt sich dieser ermitteln und auf alle Beobachtungen übertragen:

```
use if quelle==1 using iebs_1_0_test, clear              // Einlesen der BeH-Spells

gen west = inrange(ao_bula, 1,11)                        // Achtung: Missings werden hier dem Osten zugerechnet,
tab west ao_bula, miss                                   //          Berlin komplett dem Westen
gen jahr = year(begepi)

gen grosseEntgelte = tentgelt if tentgelt > 80           // BBMG ist auf jeden Fall > 80, daher alle anderen auf .

by jahr west, sort: egen bbmg = mode(grosseEntgelte)      // Beitragsbemessungsgrenze als haeufigstes Tagesentgelt,
                                                         // hier fehlt noch die Differenzierung nach Knappschaft
browse jahr west bbmg if jahr~=jahr[_n-1] | west~=west[_n-1]
```

Man sieht, dass dies mit den aktuellen Testdaten leider nicht funktioniert, da dort die wahren Werte mit Zufallszahlen überlagert wurden. Mit den originalen FDZ-Personendaten geht es aber.

Bei der Berechnung des Tagensentgelts gibt es Rundungsfehler, so dass Entgelte an der Bemessungsgrenze oftmals etwas darüber oder darunter ausgewiesen werden. Möchte man alle zensierten Entgelte identifizieren – wie z.B. vor Tobit-Schätzungen mit dem Tagesentgelt als zensierter zu erklärender Variablen – empfiehlt es sich, die Bemessungsgrenze etwas geringer anzusetzen als den exakten Wert (zum Beispiel 3 Euro).

```
gen cens = tentgelt >= (bbmg - 3) if tentgelt~=. & bbmg ~=.    // Indikator für zensierte Entgelte
tab cens, miss
```

6.2 Grafische Darstellung von Zustandsdauern

Um sich mit der zeitlichen Struktur der Daten vertraut zu machen, wird im Folgenden gezeigt, für welchen Zeitraum der Datensatz Informationen enthält. Man wird sehen, dass nicht von Anfang an jede Quelle gefüllt ist. Nur für die Jahre 2000 bis 2003 liegen für jede der vier Quellen Informationen vor.

Da Zeitdaten immer mit dem tagesgenauen Datum abgespeichert werden, soll im Folgenden die Jahreszahl aus dem Beginndatum der Episode gezogen werden. Die Funktion *year* in Verbindung mit *generate* übernimmt von der Variable „begepi“ nur das Jahr und schreibt diese in die neugebildete Variable „begepiJ“. Eine Auszählung mit *tabulate* zeigt, über welchen Zeitraum sich der Datensatz erstreckt.

```
use iebs_1_0_test, clear
gen quelle4 = quelle                    // Zunaechst werden die Auspraegungen von quelle zusammen gefasst
recode quelle4  4 8 16 32 = 3 64 = 4    // Die neue Variable quelle4 unterschiedet nur noch
                                        // 1=BeH, 2=LeH, 3=MTG, 4=BewA
label define Lquelle4 1 "1 BeH" 2 "2 LeH" 3 "3 MTG" 4 "4 BewA"
```

```
label values quelle4 Lquelle4
tab quelle4

*** Spellanfang und -ende

sort persnr spell
browse persnr spell begepi endepi if persnr<= 1032065017
gen begepiJ = year(begepi)      // Extraktion des Jahres aus dem Beginndatum
tab begepiJ
tab begepiJ quelle4
* BeH: erst ab 1993 vollstaendig, ab 1999 Geringfuegige
* Nur fuer die Jahre 2000 bis 2003 sind alle Quellen gefuellt!
```

Mithilfe von grafischen Darstellungen kann die Zeit- und Spellstruktur der IEBS veranschaulicht werden. Die folgenden Grafiken werden mit *twoway rbar* gebildet. Der Befehl *rbar* eignet sich immer dann, wenn gewisse Spannbreiten gezeichnet werden sollen, deren Anfangs- und Endpunkte durch zwei verschiedenen Variablen beschrieben werden. Im vorliegenden Fall ist die Spannbreite die Dauer des Spells, der Anfang wird durch den Beginn der Episode „begepi“, und der Schluss durch das Ende „endepi“ markiert. Die y-Achse wird durch den Spellzähler gebildet. Durch Umdrehen dieser y-Achse mit *yscale(reverse)* entsteht so eine abwärtsgerichtete „Treppe“. Jeder Balken zeigt mit seiner Länge die Dauer des Spells an. Parallele Zustände werden demnach durch exakt übereinander stehende Balken dargestellt. Während die erste Grafik die Zustände von nur einer Person zeigt, sind in der zweiten Grafik die Erwerbsverläufe von drei Personen zu sehen (siehe if-Bedingung). Diese Art von grafischer Darstellung nennt sich „Sequence Index Plot“.[1]

```
* Exkurs: grafische Darstellung der Spells
egen pers = group(persnr)
replace persnr = pers
graph twoway (rbar begepi endepi spell, horizontal) if persnr==3, yscale(reverse) xtitle("Zeit")
graph twoway (rbar begepi endepi spell, horizontal) if persnr<=3, yscale(reverse) xtitle("Zeit")
```

[1] Das Vorgehen wurde übernommen aus: Kohler/Brzinsky-Fay (2005), Sequence index plots, Stata Journal 5(4), S. 601-602.

Einen etwas übersichtlicheren Plot, in dem erkennbar ist, welcher Spell aus welcher Quelle kommt, lässt sich wie folgt generieren. In einem ersten Schritt werden mit *separate* für die Anfangs- und Endzeitpunkte der Spells die Quellen getrennt, indem neue Variablen gebildet werden. Somit gibt es für jede Quelle eine Variable, die den Anfangs- bzw. den Endpunkt des Spells anzeigt. Mithilfe dieser neu generierten Merkmale wird nun analog zu oben eine neue Grafik erstellt, in der die einzelnen Quellen zu unterscheiden sind.

```
* Farbliche Unterscheidung der Spells nach Datenquelle:

cap drop begquelle
separate begepi, by(quelle4) gen(begquelle)   // zunaechst werden Beginn- und Endedaten separiert
separate endepi, by(quelle4) gen(endquelle)

browse begepi begquelle* endepi endquelle*    // d.h. die neuen Beginn- und Endedaten sind nur
                                              //      fuer die jeweilige Quelle definiert.
#d ;
graph twoway
      (rbar begquelle1 endquelle1 spell, horizontal)
      (rbar begquelle2 endquelle2 spell, horizontal)
      (rbar begquelle3 endquelle3 spell, horizontal)
      (rbar begquelle4 endquelle4 spell, horizontal)
      if  persnr<=3
      ,
      yscale(reverse)
      xtitle("Zeitachse")
      legend(order(1 "BeH" 2 "LeH" 3 "MTG" 4 "BewA") rows(1))
      ;
#d cr
```

Eine weitere Variante eines Sequence Index Plots wird so gezeichnet, dass auf der y-Achse nicht der Spellzähler abgetragen ist, sondern die Personennummer. Somit zeigt jede Reihe den Erwerbsverlauf einer Person in chronologischer Abfolge. Zu sehen sind in diesem Beispiel die Erwerbsverläufe der ersten 30 Personen im Datensatz (siehe if-Bedingung).

```
#d ;
graph twoway
      (rbar begquelle1 endquelle1 persnr, horizontal)
      (rbar begquelle2 endquelle2 persnr, horizontal)
      (rbar begquelle3 endquelle3 persnr, horizontal)
      (rbar begquelle4 endquelle4 persnr, horizontal)
      if  persnr<=30
      ,
      yscale(reverse)
      xtitle("Zeitachse")
      legend(order(1 "BeH" 2 "LeH" 3 "MTG" 4 "BewA") rows(1))
      saving(sequInd, replace)
      ;
#d cr
```

Trägt man auf der y-Achse die Personennummer ab, ändert aber die x-Achse von der Datumsangabe in das Alter der Personen um, werden die Erwerbsverläufe zusammen mit dem Alter der Personen veranschaulicht. Hierzu muss zuerst jedoch das Alter der Personen gebildet werden, und zwar zum Zeitpunkt des Anfangs des Spells („begalter“) und zum Zeitpunkt des Ende des Spells („endalter“). Es ist zu beachten, dass nur das Geburtsjahr der Personen bekannt ist, nicht das genaue Geburtsdatum. So wird mangels dieser Information hilfsweise mit dem 1. Januar als Geburtstag gerechnet. Da jedes vierte Jahr ein Schaltjahr ist, wird nicht durch 365, sondern durch 365,25 geteilt. Um wie bisher die Quelle der Spells erkennen zu können, wird schließlich analog zu oben der *separate*-Befehl eingesetzt.

```
* Plotten der Spells nach Lebensalter

gen begalter = (begepi -  mdy(1,1,gebjahr)) / 365.25   // Alter in Jahren
gen endalter = (endepi -  mdy(1,1,gebjahr)) / 365.25
separate begalter , by(quelle4) gen(begaltquelle)      // Beginn- und Ende nach Quelle separieren
separate endalter , by(quelle4) gen(endaltquelle)

#d ;
graph twoway
      (rbar begaltquelle1 endaltquelle1 persnr, horizontal)
```

```
      (rbar begaltquelle2 endaltquelle2 persnr, horizontal)
      (rbar begaltquelle3 endaltquelle3 persnr, horizontal)
      (rbar begaltquelle4 endaltquelle4 persnr, horizontal)
      if  persnr<=100
      ,
      yscale(reverse)
      xtitle("Alter")
      legend(order(1 "BeH" 2 "LeH" 3 "MTG" 4 "BewA") rows(1))
      ;
#d cr           // viele weisse Flaechen, da maximal 15 Jahre einer Person beobachtet werden
```

Zum Schluss wird eine Kohortenbetrachtung vorgestellt. Als Beispiel werden drei Kohorten ausgewählt: Frauen der Jahrgänge 1960, 1970 und 1980. Diese Personen werden durch die neugebildete Variable *kort* markiert. In einem zweiten Schritt nummeriert die Variable *persnrkort* die gerade markierten Personen neu (die Vorgehensweise mittels *egen* und *group* ist völlig analog zum neugebildeten Personenidentifikator auf Seite 6). Die so erstellte Grafik zeigt den drei Kohorten entsprechend drei große Blöcke. Da die IEBS ungefähr von 1990 bis 2005 gefüllt ist, sind diese Blöcke maximal 15 Jahre lang. Im Jahr 1990 sind die 1960 geborenen Frauen genau 30 Jahre alt, so dass der Block der ersten Kohorte beim Alter 30 beginnt. Die Kohorte der 1970 geborenen Frauen (zweiter Block) beginnt beim Alter 20 usw.

```
* Spells nach Lebensalter: Einschraenkung für 3 Kohorten

  gen     kort = 1 if gebjahr==1960 & sex==2   // 1960 geborene Frauen
  replace kort = 2 if gebjahr==1970 & sex==2
  replace kort = 3 if gebjahr==1980 & sex==2
  tab kort, miss
  egen persnrkort = group(kort persnr)         // die markierten Personen werden durchnummeriert

#d ;
graph twoway
      (rbar begaltquelle1 endaltquelle1 persnrkort, horizontal)
      (rbar begaltquelle2 endaltquelle2 persnrkort, horizontal)
      (rbar begaltquelle3 endaltquelle3 persnrkort, horizontal)
      (rbar begaltquelle4 endaltquelle4 persnrkort, horizontal)
      if  endalter<50
```

```
      ,
      yscale(reverse)
      xtitle("Alter")
      legend(order(1 "BeH" 2 "LeH" 3 "MTG" 4 "BewA") rows(1))
      ;
#d cr
```

Das Problem aller oben gezeigten Grafiken besteht darin, dass parallele Zustände nicht zu erkennen sind. Befindet sich eine Person beispielsweise in einer Trainingsmaßnahme (MTG) und bekommt gleichzeitig Unterhaltsgeld (LeH), so liegen ein MTG- und ein LeH-Spell deckungsgleich übereinander (siehe dazu Abschnitt 4.1).

6.3 Wechsel und Dauern

Wechseln zwischen Betrieben, Bundesländern, Erwerbsstatus etc. und Dauern innerhalb solcher Zustände sind in der Berechnung ähnlich. Daher wird hier beides in einem Abschnitt betrachtet.

6.3.1 Betriebseintritt und Betriebswechsel

Ziel der nächsten Analyse soll es sein, für jede Person das Eintrittsdatum in den Betrieb zu berechnen, die Betriebe, die eine Person durchläuft, zu nummerieren und Betriebswechsel zu markieren. Da für diese Analyse nur Beschäftigungsangaben (BeH-Spells) notwendig sind, werden die Spells der übrigen Quellen (LeH, MTG und BewA) gelöscht. Im Anschluss wird zuerst eine neue Betriebsnummer gebildet (analog zum Personenidentifikator, siehe oben); dies dient lediglich der besseren Übersicht.

```
use if quelle==1 using iebs_1_0_test, clear          // Einlesen der BeH-Spells

rename betnr betnr_alt                               // Vergabe einer einfachen Betriebsnummer
egen betnr = group(betnr_alt)                        // eine neue Betriebsnummer mit Werten von 1 bis N
replace betnr = . if betnr_alt==-7                   // nur zur Uebersichtlichkeit
```

Für die Analyse der Betriebswechsel soll zur Vereinfachung nur die Hauptbeschäftigung der Personen betrachtet werden. Folgende Definition soll für die hier beschriebene Untersuchung angewendet werden: Besitzt eine Person während einer Episode mehrere parallele Beschäftigungen, dann ist die Beschäftigung mit dem höchsten Tagesentgelt die Hauptbeschäftigung. Um alle anderen Nebenbeschäftigungen zu löschen, müssen die Spells behalten werden, die in der Variable „level1“ den Wert Null besitzen. Der Spellzähler „level1“ nummeriert parallele Sätze einer Episode innerhalb einer Quelle von Null bis N, basierend auf einer absteigenden Sortierung nach Geringfügigkeit und dem Tagesentgelt (siehe Datenreport). So besitzt der Spell mit der sozialversicherungspflichtigen Beschäftigung mit dem höchsten Tagesentgelt immer die Null.

```
keep if level1==0         // nur noch ein Spell pro Episode und zwar die Hauptbeschaeftigung
```

Zunächst soll das erste *Eintrittsdatum* (Wiedereintritte nach Unterbrechung werden nicht berücksichtigt) für jeden Betrieb, den eine Person durchlaufen hat, berechnet werden. Dieses Eintrittsdatum soll in jedem Beschäftigungsspell des betreffenden Betriebs stehen.

```
by persnr betnr, sort: egen eintritt = min(begepi)   // eintritt ist das kleinste begepi pro Person und Betrieb
format eintritt %dD_m_Y
sort persnr begepi level1
browse persnr begepi endepi level1 betnr eintritt
```

Sortiert man den Datensatz nach Personennummer („persnr“) und nach dem Eintrittsdatum, kann jeder Betrieb, bei dem eine Person gearbeitet hat, genau einmal markiert werden („dum0“). Die Variable „betPersNr“ *nummeriert* daraufhin die Betriebe, die eine Person im Laufe ihres Erwerbslebens durchläuft, wobei der chronologisch erste Betrieb die eins bekommt, der zweite die zwei usw. Schließlich soll die Gesamtzahl der Betriebe einer Person in allen Spells der Person angezeigt werden („anzbet0“). Man beachte, dass dies noch nicht die Anzahl der Betriebswechsel ist, sondern lediglich die Anzahl der Betriebe, in denen eine Person

jemals gearbeitet hat. Hat eine Person mehrmals mit Unterbrechung bei ein und demselben Betrieb gearbeitet, wird dieser Betrieb nur ein einziges Mal gezählt.

```
sort persnr eintr                                  // Sortierung nach Person und erstmaligem Betriebseintritt
gen dum0 = persnr~=persnr[_n-1] | (betnr~=betnr[_n-1] & betnr~=.)
                                                   // Dummy = 1 bei jedem weiteren Betrieb einer Person
by persnr: gen betPersNr = sum(dum0)               // Zaehler der Betriebe pro Person, den hoechsten Wert hat
                                                   // der letzte Betrieb (Nummerierung)
by persnr: egen anzbet0 = sum(dum0)                // die Anz. der Betriebe wird auf alle Spells uebertragen

replace betPersNr = . if betnr==.                  // wenn die Betriebsnummer fehlt, auch hier ein Missing
sort persnr spell
browse persnr begepi endepi level1 betnr betPersNr eintritt dum0 anzbet0

tab betPersNr, miss                                // Personen arbeiten in bis zu 13 unterschiedl. Betrieben
```

Um die Anzahl der Betriebswechsel zu erfassen, muss chronologisch sortiert werden (über „begepi“). Die Variable „dum1“ wird genauso gebildet wie „dum0“; durch die andere Sortierung wird aber auch erfasst, wenn eine Person zu einem Betrieb zurückgewechselt. Die Variable „anzwechs“ überträgt die Gesamtzahl der Betriebswechsel auf alle Spells der Person.

```
sort persnr begepi
gen dum1 = persnr~=persnr[_n-1] | (betnr~=betnr[_n-1] & betnr~=.)
                          // Markierung aller Betriebswechsel (inkl. Recalls) in der
                          // Hauptbeschaeftigung + ERSTER BETRIEB
egen anzwechs = sum(dum1), by(persnr)
```

Um auf Basis der oben gebildeten Variablen Auszählungen durchzuführen, reicht es, nur einen Spell pro Person zu behalten (*keep if*). Dabei spielt es keine Rolle, welcher Spell behalten wird, da alle interessanten Informationen auf alle Spells einer Person übertragen wurden (*egen*-Befehl). So kann zum Beispiel die Zahl der unterschiedlichen Betriebe, in denen eine Person gearbeitet hat („anzbet0“), mit der Zahl der Betriebswechsel dieser Person („anzwechs“) verglichen werden. Ist „anzwechs“ größer als „anz-

bet1“, dann ist die Person mindestens einmal in ihrem Erwerbsleben in einen Betrieb gewechselt, in dem sie schon einmal gearbeitet hat.

```
sort persnr
keep if persnr~=persnr[_n-1]   // Reduktion der Daten auf eine Zeile pro Person, jede im
                               // weiteren benoetigte Info wurde auf alle Spells jeder Person uebertragen.
compare anzbet0 anzwechs       // Vergleich Anz.d.Betr. in Hauptbesch. mit Anz.d.Wechsel in Hauptbesch.
```

6.3.2 Berechnung der Beschäftigungs- und der Betriebszugehörigkeitsdauer

Drei Fragen sollen beantwortet werden: Wie lange war eine Person während ihres Erwerbslebens insgesamt beschäftigt? Wie lange war eine Person in einem bestimmten Betrieb insgesamt beschäftigt? Und: Wie lange war eine Person in einem bestimmten Betrieb *durchgehend ohne Unterbrechung* beschäftigt?

Zunächst wird innerhalb einer Person nach Quelle, Betriebsnummer und Anfangsdatum der gesplitteteten Episode sortiert. Dies hat zur Folge, dass innerhalb einer Person die Beschäftigungsspells ganz oben stehen, und dass die Beschäftigungsmeldungen innerhalb desselben Betriebs chronologisch aufeinanderfolgen. Daraufhin wird für jeden einzelnen Spell die Dauer ausgerechnet („dauer_1“).

```
use if level1 == 0 using iebs_1_0_test, clear   // Mehrfachbeschaeftigung wird ausgeschlossen
sort persnr quelle betnr begepi
gen dauer_1 = (endepi-begepi) + 1               // Dauer eines einzelnen Spells
```

In Frage eins soll untersucht werden, wie lange eine Person insgesamt in ihrem Leben beschäftigt war. Beschäftigungsunterbrechungen und Betriebswechsel spielen keine Rolle. Es müssen demnach nur die Dauern aller Beschäftigungsspells einer Person

aufaddiert werden („d_bes_ges“). Die Gesamtdauer wird mit der Variable „d_bes_ges_max“ auf alle Spells der Person übertragen.

```
gen d_bes_ges = dauer_1 if quelle == 1
replace d_bes_ges = d_bes_ges + d_bes_ges[_n-1] if persnr == persnr[_n-1] & quelle == 1
                                                  // Beschaeftigungsdauer wird kummuliert
by persnr, sort: egen d_bes_ges_max = max(d_bes_ges)// Gesamtdauer wird auf alle Spells der Person übertragen
br persnr betnr quelle spell begepi endepi dauer_1 d_bes_ges d_bes_ges_max
```

Frage zwei untersucht, wie lange eine Person in einem bestimmten Betrieb insgesamt beschäftigt war. Dabei sind Unterbrechungen erlaubt (zum Beispiel durch Arbeitslosigkeit oder Beschäftigung in einem anderen Betrieb). Das Vorgehen ist völlig analog zu dem aus Frage eins, nur werden die Dauern der Spells nicht mehr über *alle* Beschäftigungsspells aufsummiert, sondern nur noch über die Beschäftigungsspells *desselben Betriebs*. Die Variable „d_bes_bet_max“ überträgt die Gesamtdauer auf alle Spells der Person mit dieser Betriebsnummer.

```
gen d_bes_bet = dauer_1 if quelle == 1
replace d_bes_bet = d_bes_bet + d_bes_bet[_n-1] if persnr == persnr[_n-1] & betnr == betnr[_n-1] & quelle == 1
by persnr betnr, sort: egen d_bes_bet_max = max(d_bes_bet)
br persnr betnr quelle spell begepi endepi dauer_1 d_bes_bet d_bes_bet_max
```

Im Unterschied zur Frage zwei soll in der dritten Frage untersucht werden, wie lange eine Person *durchgehend ohne Unterbrechung* in einem Betrieb gearbeitet hat (Betriebszugehörigkeitsdauer). Kleine Lücken sollen dabei dennoch erlaubt sein (hier als Beispiel maximal eine Woche.

Die Kennzeichnung von fortgestzten Beschäftigungsverhältnissen wird durch Generierung der Variable „ans_bet“ erzielt. Die Variable ist null für den ersten Spell eines neuen Beschäftigungsverhältnisses und eins, wenn der Spell zum gleichen Beschäftigungsverhältnis gehört wie der Spell zuvor. Die Variable „d_bet_max“, die mithilfe der gerade gebildeten Kennzeichnung generiert wird, zeigt die Betriebszugehörigkeitsdauer an, jedoch nur für den letzten Spell des Beschäftigungsverhältnisses.

Durch eine umgekehrt chronologische Sortierung (*gsort persnr betnr -begepi*) kann nun die Betriebszugehörigkeitsdauer auf alle Spells dieses Beschäftigungsverhältnisses übertragen werden.

```
gen d_bet = dauer_1 if quelle == 1
replace d_bet = d_bet + d_bet[_n-1] if persnr == persnr[_n-1] & betnr == betnr[_n-1] & ///
                                       begepi < endepi[_n-1] + 8 & quelle == 1
gen ans_bet = 0                                 // Dummy für fortgefuehrte Beschaeftigungsverhaeltnisse
                                                // im gleichen Betrieb
replace ans_bet = 1 if persnr == persnr[_n-1] & betnr == betnr[_n-1] & begepi < endepi[_n-1] + 8   ///
                     & quelle == 1
gen d_bet_max = d_bet if ans_bet[_n+1] == 0 & quelle == 1
gsort persnr betnr -begepi
replace d_bet_max = d_bet_max[_n-1] if persnr == persnr[_n-1] & ans_bet[_n-1] == 1
sort persnr quelle betnr begepi
br persnr betnr quelle spell begepi endepi dauer_1 ans_bet d_bet d_bet_max
```

6.3.3 Berechnung von Arbeitslosigkeitsdauern

Die Berechnung von Arbeitslosigkeitsdauern kann auf unterschiedliche Art und Weise durchgeführt werden mit zum Teil deutlichen Unterschieden. Drei Berechnungsvarianten sollen hier vorgestellt werden.

In Variante eins wird die Dauer der Arbeitslosigkeit auf Basis der BewA-Spells berechnet, in denen der Arbeitsuchende arbeitslos ist. Hierzu werden nur die BewA-Spells benötigt, die in der „erwstat“-Variable den Wert 31 haben (Person ist arbeitslos gemeldet), alle übrigen Spells werden gelöscht. Zur Vereinfachung sollen darüber hinaus parallele Spells in einer Episode gelöscht werden.

```
use if level1 == 0 using iebs_1_0_test, clear        // Mehrfachbeschaeftigung wird ausgeschlossen
keep if erwstat == 3                                 // nur Arbeitsuche bei Arbeitslosigkeit
```

Das weitere Vorgehen ist analog zur Berechnung der Betriebszugehörigkeitsdauer. Zunächst werden die Spells markiert, die als Fortsetzung der Arbeitslosigkeit dieser Person betrachtet werden können („anschluss_alo“). Lücken von unter einer Woche sollen

die Dauer nicht unterbrechen. Da nur noch Arbeitslosigkeitsspells im Datensatz vorhanden sind, müssen lediglich die Lücken zwischen zwei aufeinanderfolgenden Spells einer Person untersucht werden.

```
sort persnr begepi
gen anschluss_alo = 0
by persnr: replace anschluss_alo = 1 if begepi-endepi[_n-1]-1 <= 7
```

Die Variable “spelldauer” berechnet die Dauer jedes Spells. Die Variable „alo_bewa“ summiert die Dauern schließlich immer dann auf, wenn die Arbeitslosigkeit laut „anschluss_alo“ fortgesetzt wird. Der Datensatz wird zum Schluss gespeichert, um später den Vergleich dieser mit den übrigen zwei Varianten zu ermöglichen.

```
gen spelldauer = endepi - begepi + 1
gen alo_bewa = spelldauer if anschluss_alo==0                                    // Beginn der Alo
by persnr: replace alo_bewa = alo_bewa[_n-1] +  spelldauer if anschluss_alo==1  // Kumulation bei
                                                                                 // Fortsetzung der Alo
keep persnr spell alo_bewa
sort persnr spell
save bewa.dta, replace
```

In Variante zwei wird die Arbeitslosigkeitsdauer auf Basis der BeH-Spells berechnet. Hierbei werden alle Lücken zwischen zwei Beschäftigungsspells als Arbeitslosigkeit betrachtet, diese Variante stellt somit die obere Grenze dar. Im ersten Schritt wird der Ausgangsdatensatz eingelesen, da wieder BeH-Spells benötigt werden. Alle übrigen Quellen können gelöscht werden. Parallele Beschäftigung kann ignoriert werden, da es für die Betrachtung der Lücken unerheblich ist, ob davor oder danach *mehrere* Beschäftigungen parallel in einer Episode liegen.

```
use if level1 == 0 & quelle == 1 using iebs_1_0_test, clear  // Mehrfachbeschaeftigung ausgeschlossen, nur BeH
```

Die Variable „alo_beh“ berechnet nun die Dauer der Lücke zwischen zwei aufeinanderfolgenden BeH-Spells einer Person. Hierbei sollen wiederum Lücken von einer Woche und weniger ignoriert und nicht als Arbeitslosigkeit gezählt werden. Zum Schluss wird für den späteren Vergleich der Varianten wieder zwischengespeichert.

```
sort persnr begepi
by persnr: gen alo_beh = begepi - endepi[_n-1] - 1      // Dauer der Luecke vor jedem BeH-Spell
replace alo_beh = 0 if alo_beh < 8                      // Luecken unter 8 Tagen werden auf Null gesetzt
keep persnr spell alo_beh
sort persnr spell
save beh, replace
```

Die dritte Variante ist eine Modifikation von Variante zwei. Es werden wieder Lücken zwischen zwei Beschäftigungen untersucht. Als Phasen der Arbeitslosigkeit werden sie aber nur dann angesehen, wenn innerhalb der Lücken auch ein Leistungsempfang stattgefunden hat. Zunächst wird wieder der Ausgangsdatensatz eingelesen. Neben den BeH- werden nun auch die LeH-Spells benötigt. Alle weiteren Quellen werden gelöscht. Parallele Beschäftigung und paralleler Leistungsempfang können wieder ignoriert werden.

```
use if level1 == 0 & quelle < 3 using iebs_1_0_test, clear  // Mehrfachbes. ausgeschlossen, nur BeH und LeH
```

Es kann sein, dass innerhalb einer Episode eine Beschäftigung und ein Leistungsempfang parallel liegen. Um diese Fälle später nicht fälschlicherweise als Arbeitslosigkeitsphase mitzuzählen, werden in einem solchen Fall die LeH-Spells gelöscht.

```
sort persnr begepi quelle
drop if quelle==2 & quelle[_n-1]==1 & begepi==begepi[_n-1]
```

Mithilfe der Variable „LvorB“ werden im Folgenden BeH-Spells markiert, denen ein LeH-Spell vorausgeht. Die LeH-Spells können danach gelöscht werden, da die Markierung und damit die interessierende Information im BeH-Spell steckt. Analog zu oben wird

daraufhin die Dauer der Lücke berechnet („alo_bleh“), wobei allerdings neben der „Toleranzwoche“ auch diejenigen Fälle nicht mitgezählt werden, in denen der Beschäftigung kein Leistungsempfang vorausgeht. Zum Schluss wird der Datensatz wieder abgespeichert.

```
gen     LvorB = 0
replace LvorB = 1 if persnr==persnr[_n-1] & quelle==1 & quelle[_n-1]==2
                             // BeH-Spells werden markiert, wenn im Spell davor ein Leistungsbezug erfolgte
keep if quelle==1            // Einschraenkung auf BeH-Spells
sort persnr begepi
by persnr: gen alo_bleh = begepi - endepi[_n-1] - 1   // Dauer der Luecke vor jedem BeH-Spell
replace alo_bleh = 0 if alo_bleh < 8 | LvorB~=1       // Luecken unter 8 Tagen werden auf Null gesetzt
                                                      // ebenso Luecken ohne LeH-Spell davor
replace alo_bleh = . if persnr~=persnr[_n-1]          // Missing beim ersten BeH-Spell
keep persnr spell alo_bleh
sort persnr spell
save bleh, replace
```

Um alle drei Varianten vergleichen zu können, werden die zuvor zwischengespeicherten Datensätze zusammengespielt.

```
use iebs_1_0_test, clear
sort persnr spell
merge persnr spell using bewa
drop _merge
sort persnr spell
merge persnr spell using beh
drop _merge
sort persnr spell
merge persnr spell using bleh
drop _merge
```

Es ist unproblematisch, Variante zwei und drei zu vergleichen, da in beiden Fällen die Arbeitslosigkeitsdauer in demselben Spell abgelegt ist. Eine Vergleichsmöglichkeit bietet der *compare*-Befehl.

```
compare alo_beh alo_bleh          // die letzte Variante ist erwartungsgemaess strikter
```

Problematischer ist der Vergleich dieser beiden Varianten mit Variante eins, da bei diesem Fall die Arbeitslosigkeitsdauer in einem BewA-Spell steckt. Verglichen werden können die Varianten darüber hinaus nur, wenn dem oder den BewA-Spells auch ein Beschäftigungsspell folgt. Ist der letzte beobachtete Spell einer Person zum Beispiel ein Arbeitslosigkeitsspell (*erwstat == 31*), dann wird diese Arbeitslosigkeit von den Varianten zwei und drei nicht erkannt – ein Nachteil dieser Varianten.

Die Fälle, in denen dem BewA-Spell ein BeH-Spell folgt, werden mit der Variable „vgl“ markiert. Dabei bekommt der BewA-Spell die eins, der BeH-Spell die zwei. Für den Vergleich werden nur die Quellen eins und vier benötigt.

```
keep if quelle==1 | quelle==64
gen     vgl = 1 if persnr==persnr[_n+1] & erwstat==31 & quelle[_n+1]==1   & endepi==begepi[_n+1] - 1
replace vgl = 2 if persnr==persnr[_n-1] & quelle==1   & erwstat[_n-1]==31 & begepi==endepi[_n-1] + 1
```

Die Arbeitslosigkeitsdauer der Variante eins, die im BewA-Spell steht, kann nun auf den BeH-Spell übertragen werden. Hierzu werden alle übrigen Spells vorher gelöscht, die nicht verglichen werden können. Nach der Übertragung der Dauer auf den BeH-Spell können schließlich auch die BewA-Spells gelöscht werden. Die Arbeitslosigkeitsdauern aller drei Varianten stehen daraufhin im selben Spell.

```
keep if vgl==1 | vgl==2 // nur die vergleichbaren Dauern werden behalten
replace alo_bewa = alo_bewa[_n-1] if  alo_bewa==.
keep if quelle==1
```

Zum Vergleich der Varianten eignet sich zum einen wieder der *compare*-Befehl, zum anderen können aber auch grafische Darstellungen Aufschluss über die Unterschiede geben (*tw scatter*). In der zweiten und dritten Grafik wird zusätzlich noch die Winkelhalbierende gezeichnet, da im Idealfall, wenn zwei Varianten zu hundert Prozent übereinstimmen, alle Punkte auf dieser Grade liegen.

```
compare alo_bewa alo_beh          // Die Abweichungen sind relativ deutlich. Vor allem die Berechnung
compare alo_bewa alo_bleh         // auf Basis der BeH ist aber auch sehr grob
tw scatter alo_bewa alo_beh
tw scatter alo_bewa alo_beh || function y=x, range(0 4000)
tw scatter alo_bewa alo_beh if alo_bewa < 1000 & alo_beh < 1000 || function y=x, range(0 1000)
```

6.3.4 Markierung von links- und rechtszensierten Dauern

Beschäftigungsverhältnisse, die laut Daten am 1.1.1990 beginnen, sind höchstwahrscheinlich linkszensiert, das heißt, der wahre Beginn der Beschäftigung liegt vor 1990. Beschäftigungen, die am 31.12.2003 enden, sind höchstwahrscheinlich rechtszensiert, da sie über das Jahr 2003 hinaus fortbestehen können. Als rechtszensiert soll darüber hinaus auch ein Spell gelten, der am 31.12.2002 endet und gleichzeitig der letzte beobachtete Beschäftigungsspell der Person ist. Die Begründung hierfür ist, dass in der IEBS 1.0 für das Jahr 2003 noch Meldungen fehlen.

```
sort persnr betnr begepi
gen links = 0
replace links = 1 if begepi == d(01, jan, 1990) & quelle == 1
                  // Spell ist linkszensiert, wenn er am 1.Januar 1990 beginnt
tab links
gen rechts = 0
replace rechts = 1 if endepi == d(31, dec, 2003) | (endepi == d(31, dec, 2002) & persnr != persnr[_n+1]) ///
                                                                                & quelle == 1
                                                  // Spell ist rechtszensiert, wenn Endedatum der 31.12.2003
                                                  // ist oder wenn Endedatum der 31.12.2002 ist und dies
                                                  // zugleich der letzte Spell der betreffenden Person ist.
tab links rechts
```

Nach Übertragung der Markierung auf alle weiteren Spells dieses Beschäftigungsverhältnisses kann anhand jedes Spells dieser Beschäftigung erkannt werden, ob die Beschäftigung an den rechten und/oder linken Rand des Beobachtungszeitraums stößt.

```
gen ans_bet = 0                                    // Dummy für fortgefuehrte Beschaeftigungsverhaeltnisse
                                                   // im gleichen Betrieb
replace ans_bet = 1 if persnr == persnr[_n-1] & betnr == betnr[_n-1] & begepi < endepi[_n-1] + 8   ///
                     & quelle == 1
replace links = links[_n-1] if ans_bet == 1     // Übertragung der Information linkszensiert auf das
                                                   // gesamte Beschaeftigungsverhaeltnis
gsort persnr betnr -begepi
replace rechts = rechts[_n-1] if ans_bet[_n-1] == 1   // Übertragen der Information rechtszensiert
sort persnr spell
order rechts links
br
```

Imprint

FDZ Methodenreport
No. 6/2007

Publisher
The Research Data Centre (FDZ)
of the Federal Employment Service
in the Institute for Employment Research
Regensburger Str. 104
D-90478 Nuremberg

Editorial staff
Stefan Bender, Dagmar Herrlinger

Technical production
Dagmar Herrlinger

Download
http://doku.iab.de/fdz/reporte/2007/MR_06-07.pdf

Internet
http://fdz.iab.de/

Corresponding author
Nils Drews, Tel.: +49-911-179-1770
E-Mail: nils.drews@iab.de