Methoden-Workshop

Leading House „Economics of Education“/SKBF

Ben Jann und Rudi Farys

Universität Zürich, Rämistrasse 71, Raum KOL-F-123, 1.–3. Februar 2016

Makros, Loops, Programme und Workflow

Übersicht

- Makros und Loops
- Programme
- Workflow
- Reproducibility

Macros, scalars, matrices

- In Stata-Jargon ist ein Makro ein Platzhalter, denn man mit beliebigen Zeichen füllen kann. Siehe `help macro`.
- Unterschieden wird zwischen globalen on lokalen Makros.
  - `global`: Makros, die überall in Stata verfügbar sind.
  - `local`: Makros, die nur auf in dem do-file oder Program, in dem sie definiert wurden, verfügbar sind.
- Lokales Makro definieren und abrufen:

```
. local a "was auch immer"
. display "`a'"
was auch immer
```

- Globales Makro definieren und abrufen:

```
. global a "etwas anderes"
. display "${a}"
etwas anderes
```

Macros, scalars, matrices

- Makros können auch problemlos verwendet werden, um Zahlen zu speichern. Die Zahlen werden jedoch als Text repräsentiert und nicht als Gleitkommazahlen.

```
. local a 1
. display "`a'"
1
. local a = `a' + 1
. display "`a'"
2
. local a = sqrt(2)
. display "`a'"
1.414213562373095
. display `a'-sqrt(2)
-2.220e-16
. local a sqrt(2)
. display "`a'"
sqrt(2)
```

Macros, scalars, matrices

- Wenn die Präzision eine Rolle spielt, sollte man einen numerischen `scalar` verwenden (siehe `help scalar`):

```
. local a = sqrt(2)
. display `a'-sqrt(2)
-2.220e-16
. scalar a = sqrt(2)
. display scalar(a)-sqrt(2)
0
```

- Ein `scalar` ist ein globales Objekt. Es empfiehlt sich deshalb die Verwendung eines temporären Namens (siehe `help tempname`). Dadurch wird der `scalar` zu einem lokalen Objekt (Objekte mit temporären Namen werden bei Ende des do-files oder Programms automatisch zerstört).

```
. tempname a
. scalar `a' = sqrt(2)
. display scalar(`a')
1.4142136
```

Macros, scalars, matrices

- Um mehrere Zahlen zu speichern und später wieder abzurufen, ist ferner der `matrix`-Befehl nützlich (siehe `help matrix`) (Matrizen sind ebenfalls globale Objekte, auch hier empfiehlt sich deshalb u.U. die Verwendung von temporären Namen):

```
. matrix M = (1, 0, 0) \ (2, sqrt(2), 4)
. matrix list M
M[2,3]
           c1          c2          c3
r1          1           0           0
r2          2   1.4142136           4
. display M[2,2]
1.4142136
. display M[2,2]*M[2,3]
5.6568542
```

Macros, scalars, matrices

- Nochmals zurück zu den Makros. Es gibt verschiedene Wege, ein Makro zu definieren.
  - Kopie des getippten Textes (ohne äussere Anführungszeichen)

```
. local a "was auch immer"
. display "`a'"
was auch immer
. local a was auch immer
. display "`a'"
was auch immer
. local a `"Die Antwort lautet: "irgend etwas""'
. display `"`a'"'
Die Antwort lautet: "irgend etwas"
. local a Die Antwort lautet: "irgend etwas"
. display `"`a'"'
Die Antwort lautet: "irgend etwas"
. local a "= sqrt(2)"
. display "`a'"
= sqrt(2)
```

Macros, scalars, matrices

- Ergebnis einer Gleichung:

```
. local a = 2 + 2
. display "`a'"
4
. local a = invttail(100, 0.025)
. display "`a'"
1.983971518523553
```

- Erhöhung/Verringerung um eins:

```
. local a 1
. local ++a
. display "`a'"
2
. local --a
. display "`a'"
1
```

Macros, scalars, matrices

- Extended macro functions (siehe `help extended_fcn`):

```
. local a "eins zwei drei"
. local b: word 2 of `a'
. display "`b'"
zwei
. local b: subinstr local a "zwei" "2"
. display "`b'"
eins 2 drei
```

- Manipulation von Listen (`help macrolists`):

```
. local a "A B B A"
. local b: list uniq a
. display "`b'"
A B
. local c "A C D C"
. local b: list a & c
. display "`b'"
A
```

`if`-Verzweigungen und Loops

- Loops können verwendet werden, um ähnliche Operationen zu wiederholen (z.B. Wiederholung analoger Analysen für verschiedene Gruppen). Innerhalb von Loops, erweisen sich `if`-Verzweigungen häufig als nützlich (siehe `help ifcmd`).
- Palma-Ratio (top 10% share divided by bottom 40% share):

```
sysuse nlsw88, clear
sort union wage
by union: generate cwage = sum(wage)
forvalues i = 0/1 {
    if `i'==1 display as txt _n "==> union"
    else      display as txt _n "==> nonunion"
    summarize wage if union==`i', meanonly
    local total = r(sum)
    _pctile cwage if union==`i', p(40, 90)
    local palma = (`total'-r(r2)) / r(r1)
    display as txt "Palma ratio = " as res `palma'
}
```

if-Verzweigungen und Loops

- Ergebnis:

```
. sysuse nlsw88, clear
(NLSW, 1988 extract)

. sort union wage

. by union: generate cwage = sum(wage)

. forvalues i = 0/1 {
  2.     if `i'==1 display as txt _n "==> union"
  3.     else      display as txt _n "==> nonunion"
  4.     summarize wage if union==`i', meanonly
  5.     local total = r(sum)
  6.     _pctile cwage if union==`i', p(40, 90)
  7.     local palma = (`total'-r(r2)) / r(r1)
  8.     display as txt "Palma ratio = " as res `palma'
  9. }

==> nonunion
Palma ratio = 1.0421662

==> union
Palma ratio = .83048847
```

`if`-Verzweigungen und Loops

- Gleiches Resultat mit `levelsof` und „inline expansion“ einer „extended macro function“ anstatt `if`:

```
. sysuse nlsw88, clear
(NLSW, 1988 extract)

. sort union wage

. by union: generate cwage = sum(wage)

. quietly levelsof union

. foreach i in `r(levels)' {
  2.     display as txt _n "==> `: label (union) `i''"
  3.     summarize wage if union==`i', meanonly
  4.     local total = r(sum)
  5.     _pctile cwage if union==`i', p(40, 90)
  6.     local palma = (`total'-r(r2)) / r(r1)
  7.     display as txt "Palma ratio = " as res `palma'
  8. }

==> nonunion
Palma ratio = 1.0421662

==> union
Palma ratio = .83048847
```

Verschiedene Arten von Loops

- Loop über Nummern (siehe `help forvalues`):

```
. forvalues i = 1/10 {
  2.     display "`i' " _c
  3. }
1 2 3 4 5 6 7 8 9 10
.
. forvalues i = 1(2)10 {
  2.     display "`i' " _c
  3. }
1 3 5 7 9
.
. forvalues i = 10(-2)1 {
  2.     display "`i' " _c
  3. }
10 8 6 4 2
.
. forvalues i = 10 15 to 30 {
  2.     display "`i' " _c
  3. }
10 15 20 25 30
```

Verschiedene Arten von Loops

- Loop über Elemente (siehe `help foreach`):

```
. local i 0
. foreach s in a b c {
  2.     local ++i
  3.     display "`i':`s' " _c
  4. }
1:a 2:b 3:c
.
. local mylist one two three
. local i 0
. foreach s of local mylist {
  2.     local ++i
  3.     display "`i':`s' " _c
  4. }
1:one 2:two 3:three
```

Verschiedene Arten von Loops

- While-Schlaufen (siehe `help while`):

```
. local i 0
. while (`++i'<=10) {
  2.     display "`i' " _c
  3. }
1 2 3 4 5 6 7 8 9 10
```

- Elemente überspringen oder einen Loop abbrechen (siehe `help continue`):

```
. forvalues i = 1/10 {
  2.     if `i'==5 {
  3.         di "(don't like five) " _c
  4.         continue
  5.     }
  6.     display "`i' " _c
  7. }
1 2 3 4 (don't like five) 6 7 8 9 10
.
. forvalues i = 1/10 {
  2.     if `i'==5 continue, break
  3.     display "`i' " _c
  4. }
1 2 3 4
```

Indexing und `by`-Prefix

- Bei der Datenaufbereitung und -analyse sind Loops vor allem nützlich, um Arbeitsschritte zu wiederholen, ohne den Code vervielfältigen zu müssen.
- Vermeiden sollte man hingegen die Verwendung von Loops, um Operationen für einzelne Beobachtungen zu wiederholen. Dafür gibt es eigentlich immer eine viel effizientere Lösung.
- Der `egen`-Befehl bietet viele vorgefertigte Funktionen, die sich in diesem Zusammenhang häufig als nützlich erweisen (siehe `help egen`).
- Auf elementarerer Ebene bekommt man eigentlich alles hin durch Indizierung mit Hilfe von `_n` und `_N`, ggf. in Kombination mit `by`.

Indexing und by-Prefix

- Beispiel: Gruppenmittelwerte berechnen und als neue Variable speichern

```
. sysuse auto, clear
(1978 Automobile Data)

. generate mprice = price

. sort foreign

. by foreign: replace mprice = mprice + mprice[_n-1] if _n>1
(72 real changes made)

. by foreign: replace mprice = mprice[_N] / _N
(74 real changes made)

. fre mprice
```

mprice

		Freq.	Percent	Valid	Cum.
Valid	6072.4231	52	70.27	70.27	70.27
	6384.6818	22	29.73	29.73	100.00
	Total	74	100.00	100.00	

Indexing und by-Prefix

- Alternative 1:

```
. sysuse auto, clear
(1978 Automobile Data)

. sort foreign

. by foreign: generate mprice = sum(price)

. by foreign: replace mprice = mprice[_N] / _N
(74 real changes made)

. fre mprice
```

mprice

		Freq.	Percent	Valid	Cum.
Valid	6072.4231	52	70.27	70.27	70.27
	6384.6818	22	29.73	29.73	100.00
	Total	74	100.00	100.00	

Indexing und by-Prefix

- Alternative 2:

```
. sysuse auto, clear
(1978 Automobile Data)
. sort foreign
. by foreign: egen mprice = mean(price)
. fre mprice
```

mprice

		Freq.	Percent	Valid	Cum.
Valid	6072.4231	52	70.27	70.27	70.27
	6384.6818	22	29.73	29.73	100.00
	Total	74	100.00	100.00	

Programme

- Als Alternative zu Loops bietet es sich häufig auch an, ein kleines Programm zu schreiben (siehe `help program`).
- Programme zu schreiben, ist keine Hexerei und kann sehr effizient sein. Führt zudem i.d.R. dazu, dass man weniger Fehler macht (oder zumindest immer den gleichen Fehler).
- Eiserne Gesetze für Programme:
  - Ein Programm soll die Daten genau so hinterlassen, wie es sie vorgefunden hat (ausser eine Änderung der Daten ist der Zweck des Programms).
  - Ein Programm soll keine Objekte hinterlassen (abgesehen von Objekten, deren Erstellung der Zweck des Programms ist).
  - Dies soll auch gelten, wenn das Programm aufgrund einer Fehlers oder aufgrund der Break-Taste abbricht.

Programme

- Einige hilfreiche Dinge, um diese Ziele zu erreichen:
  - `program ..., sortpreserve`
    - ★ Sortierung der Daten am Ende automatisch wiederherstellen (siehe `help sortpreserve`)
  - `tempvar`, `tempname`, `tempfile`
    - ★ temporäre Objekte anlegen, die am Ende automatisch vernichtet werden (siehe `help macro`)
  - `preserve`
    - ★ Daten zwischenspeichern und am Ende automatisch wiederherstellen (tendenziell nur verwenden, wenn's anders nicht geht/zu kompliziert ist/zu lange dauert)

Programme

- Beispiel: Palma-Ratio

```
program palma, sortpreserve
    syntax varname [if] [in]
    marksample touse
    sort `touse' `varlist'
    tempvar cum
    quietly by `touse': generate double `cum' = sum(`varlist') if `touse'
    quietly by `touse': replace `cum' = `cum' / `cum'[_N] if `touse'
    _pctile `cum' if `touse', p(40, 90)
    display as txt "Palma ratio = " as res (1-r(r2)) / r(r1)
end
```

- Anmerkungen:
  - `syntax`
    - ★ Befehl, der dem Programm eine standardmässige Stata-Syntax gibt
  - `marksample`
    - ★ legt eine temporäre 0/1-Variable an, die die relevanten Beobachtungen markiert
  - `quietly`
    - ★ unterdrückt den Output

Programme

```
. capture program drop palma
. program palma, sortpreserve
  1.     syntax varname [if] [in]
  2.     marksample touse
  3.     sort `touse' `varlist'
  4.     tempvar cum
  5.     quietly by `touse': generate double `cum' = sum(`varlist') if `touse'
  6.     quietly by `touse': replace `cum' = `cum' / `cum'[_N] if `touse'
  7.     _pctile `cum' if `touse', p(40, 90)
  8.     display as txt "Palma ratio = " as res (1-r(r2)) / r(r1)
  9. end
. sysuse nlsw88, clear
(NLSW, 1988 extract)
. palma wage
Palma ratio = 1.3197332
. quietly levelsof union
. foreach i in `r(levels)' {
  2.     display as txt _n "==> `: label (union) `i''"
  3.     palma wage if union==`i'
  4. }
==> nonunion
Palma ratio = 1.0421662
==> union
Palma ratio = .83048847
```

Programme

- Resultate in `r()` speichern, so dass sie weiterverarbeitet werden können (siehe `help return`)

```
program palma, rclass sortpreserve
    syntax varname [if] [in]
    marksample touse
    sort `touse' `varlist'
    tempvar cum
    quietly by `touse': generate double `cum' = sum(`varlist') if `touse'
    quietly by `touse': replace `cum' = `cum' / `cum'[_N] if `touse'
    _pctile `cum' if `touse', p(40, 90)
    tempname palma
    scalar `palma' = (1-r(r2)) / r(r1)
    display as txt "Palma ratio = " as res `palma'
    return scalar palma = scalar(`palma')
end
```

- Anmerkungen:
  - `program ..., rclass` deklariert, dass Resultate in `r()` gespeichert werden
  - `return scalar` speichert einen numerischen Skalar in `r()`

```
. capture program drop palma

. program palma, rclass sortpreserve
  1.     syntax varname [if] [in]
  2.     marksample touse
  3.     sort `touse' `varlist'
  4.     tempvar cum
  5.     quietly by `touse': generate double `cum' = sum(`varlist') if `touse'
  6.     quietly by `touse': replace `cum' = `cum' / `cum'[_N] if `touse'
  7.     _pctile `cum' if `touse', p(40, 90)
  8.     tempname palma
  9.     scalar `palma' = (1-r(r2)) / r(r1)
 10.     display as txt "Palma ratio = " as res `palma'
 11.     return scalar palma = scalar(`palma')
 12. end

. sysuse nlsw88, clear
(NLSW, 1988 extract)

. palma wage
Palma ratio = 1.3197332

. return list

scalars:
              r(palma) =  1.319733197388888

. bootstrap r(palma), nodots nowarn: palma wage

Bootstrap results                                 Number of obs     =      2,246
                                                  Replications      =         50

      command:  palma wage
        _bs_1:  r(palma)
```

	Observed Coef.	Bootstrap Std. Err.	z	P>|z|	Normal-based [95% Conf.	Interval]
_bs_1	1.319733	.0551567	23.93	0.000	1.211628	1.427838

Programme

- Mehrere Variablen zulassen und Resultate in `e()` speichern (siehe `help ereturn`).
  - `program ..., eclass`
  - `version`, um sicher zu stellen, dass das Programm auch in späteren Stata-Versionen immer noch das gleiche tut (ein „Muss“ für Programme, die man als Ado-File zur Verfügung stellt)
  - `replay()`, so dass `palma` ohne Angabe von Variablen die zuvor berechneten Ergebnisse erneut anzeigt
  - `matrix`, um die Ergebnisse in einer Matrize sammeln
  - `ereturn post`, um die Ergebnisse in `e()` zu speichern
  - `_coef_table_header` und `ereturn display`, um die Ergebnisse anzuzeigen

```
program palma, eclass sortpreserve
    version 13.1
    if replay() {
        if e(cmd)!="palma" {
            display as err "last palma results not found"
            exit 301
        }
        _coef_table_header
        ereturn display `0'
        exit
    }
    syntax varlist [if] [in] [, * ]
    marksample touse
    quietly count if `touse'
    local N = r(N)
    tempname b
    matrix `b' = J(1, `: list sizeof varlist', .)
    matrix colnames `b' = `varlist'
    tempvar cum
    local i 0
    foreach v of local varlist {
        local ++i
        sort `touse' `v'
        quietly by `touse': generate double `cum' = sum(`v') if `touse'
        quietly by `touse': replace `cum' = `cum' / `cum'[_N] if `touse'
        _pctile `cum' if `touse', p(40, 90)
        matrix `b'[1, `i'] = (1-r(r2)) / r(r1)
        drop `cum'
    }
    ereturn post `b', obs(`N') esample(`touse')
    ereturn local title "Palma Ratio"
    ereturn local depvar "`varlist'"
    ereturn local cmd "palma"
    _coef_table_header
    ereturn display, `options'
end
```

Programme

```
. sysuse nlsw88, clear
(NLSW, 1988 extract)

. palma wage hours
```

Palma Ratio — Number of obs = 2,242

	Coef.
wage	1.320945
hours	.4715703

```
. bootstrap: palma wage hours
(running palma on estimation sample)

Bootstrap replications (50)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
..................................................    50
```

Palma Ratio — Number of obs = 2,242; Replications = 50

	Observed Coef.	Bootstrap Std. Err.	z	P>|z|	Normal-based [95% Conf.	Interval]
wage	1.320945	.0506506	26.08	0.000	1.221672	1.420219
hours	.4715703	.0075096	62.80	0.000	.4568518	.4862889

Programme

- Eine sinnvolle zusätzliche Erweiterung wäre z.B. eine `over()` option, um Resultate nach Subgruppen zu berechnen.
- Weitere Hinweise:
  - Programme debuggen (siehe `help trace`):

```
set trace on
palma wage hours
set trace off
```

    oder:

```
ssc install tr
tr: palma wage hours
tr 1: palma wage hours
```

  - Viele nützliche Befehle zum Programmieren findet man in `help program` (bzw. im [P]-Manual). Weitere nützliche Tools, die einem Programmierarbeit abnehmen können, findet man auch unter `help undocumented` (z.B. Befehle, um Resultate anzuzeigen).

Workflow

- Um Produktiv arbeiten zu können, ist eine gute Arbeitsumgebung zentral.
  - Ein wichtiges Element ist der Texteditor. Essenziell ist für mich ...
    - ★ ... dass ich den Code in einem Do-File blockweise bearbeiten kann (i.e. gleichzeitige Bearbeitung mehrerer Zeilen),
    - ★ ... dass der Editor flexible Find-Replace-Funktionen zur Verfügung stellt,
    - ★ ... dass ich nach Herzenslust Tastenkombinationen u.ä. definieren kann, um gewisse Dinge zu tun (z.B. um ein Code-Fragment zur Ausführung an Stata zu schicken),
    - ★ ... dass ich unterschiedliche Dateitypen definieren kann (z.B. mit spezifischem Syntax-Highlighting und spezifischen Tastenkombinationen),
    - ★ ... dass eine übersichtliche Verwaltung von komplexen Projekten mit vielen Dateien möglich ist.
  - Persönlich verwende ich TextMate 2, aber es gibt eine Vielzahl guter Editoren; siehe etwa http://fmwww.bc.edu/repec/bocode/t/textEditors.html.

Workflow

- Meiner Erfahrung nach lohnt es sich, eine Datenanalyse in zwei konzeptionelle Schritte zu unterteilen:
  1. Die eigentliche Analyse. Die ganzen Ergebnisse von Modellschätzungen etc. werden gesammelt und archiviert.
  2. Reporting, i.e. Produktion von Grafiken und Tabellen, ggf. einige einfache Zusatzberechungen anhand der Resultate.
- Dies hat den Vorteil, dass man in Zukunft auf die Resultate zurückgreifen kann, ohne die ganze Analyse wiederholen zu müssen.
- `estwrite`: User-Ado, mit dem man `e()`-Resultate speichern und später wieder einlesen kann (`ssc install estwrite`; ähnlich wie der offizielle Befehl `estimates save`, hat aber den Vorteil, dass man mehrere Modelle in eine Datei schreiben kann).

1. Schritt: Analyse

```
. sysuse nlsw88, clear
(NLSW, 1988 extract)

. bootstrap, nodots: palma wage hours if union==0
Palma Ratio                                     Number of obs     =      1,416
                                                Replications      =         50
```

	Observed Coef.	Bootstrap Std. Err.	z	P>|z|	Normal-based [95% Conf.	Interval]
wage	1.041652	.0329989	31.57	0.000	.9769753	1.106328
hours	.4571916	.0107745	42.43	0.000	.436074	.4783091

```
. estimates store nonunion

. bootstrap, nodots: palma wage hours if union==1
Palma Ratio                                     Number of obs     =        461
                                                Replications      =         50
```

	Observed Coef.	Bootstrap Std. Err.	z	P>|z|	Normal-based [95% Conf.	Interval]
wage	.8304885	.0533129	15.58	0.000	.7259972	.9349797
hours	.4210792	.0114161	36.88	0.000	.3987041	.4434542

```
. estimates store union

. estwrite * using myresults, replace
(saving nonunion)
(saving union)
(file myresults.sters saved)
```

2. Schritt: Reporting

```
. estread myresults
```

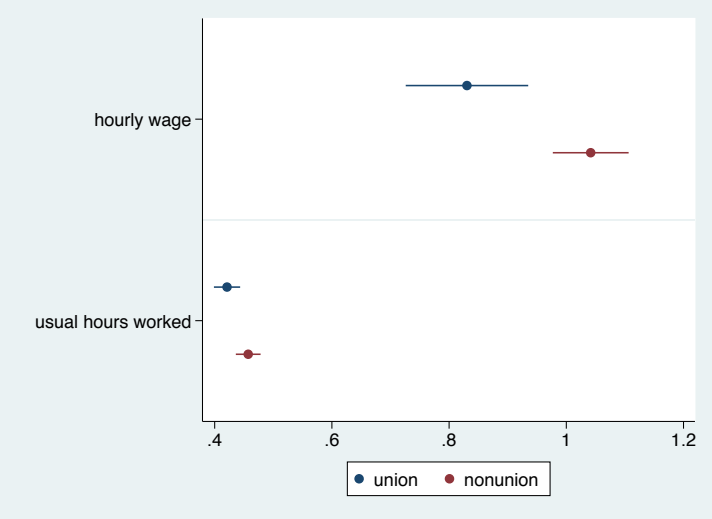
name	command	depvar	npar	title
nonunion	palma	mult. depvar	2	
union	palma	mult. depvar	2	

```
. esttab union nonunion, se wide mtitles nonumber nostar
```

	union		nonunion	
wage	0.830	(0.0533)	1.042	(0.0330)
hours	0.421	(0.0114)	0.457	(0.0108)
N	461		1416	

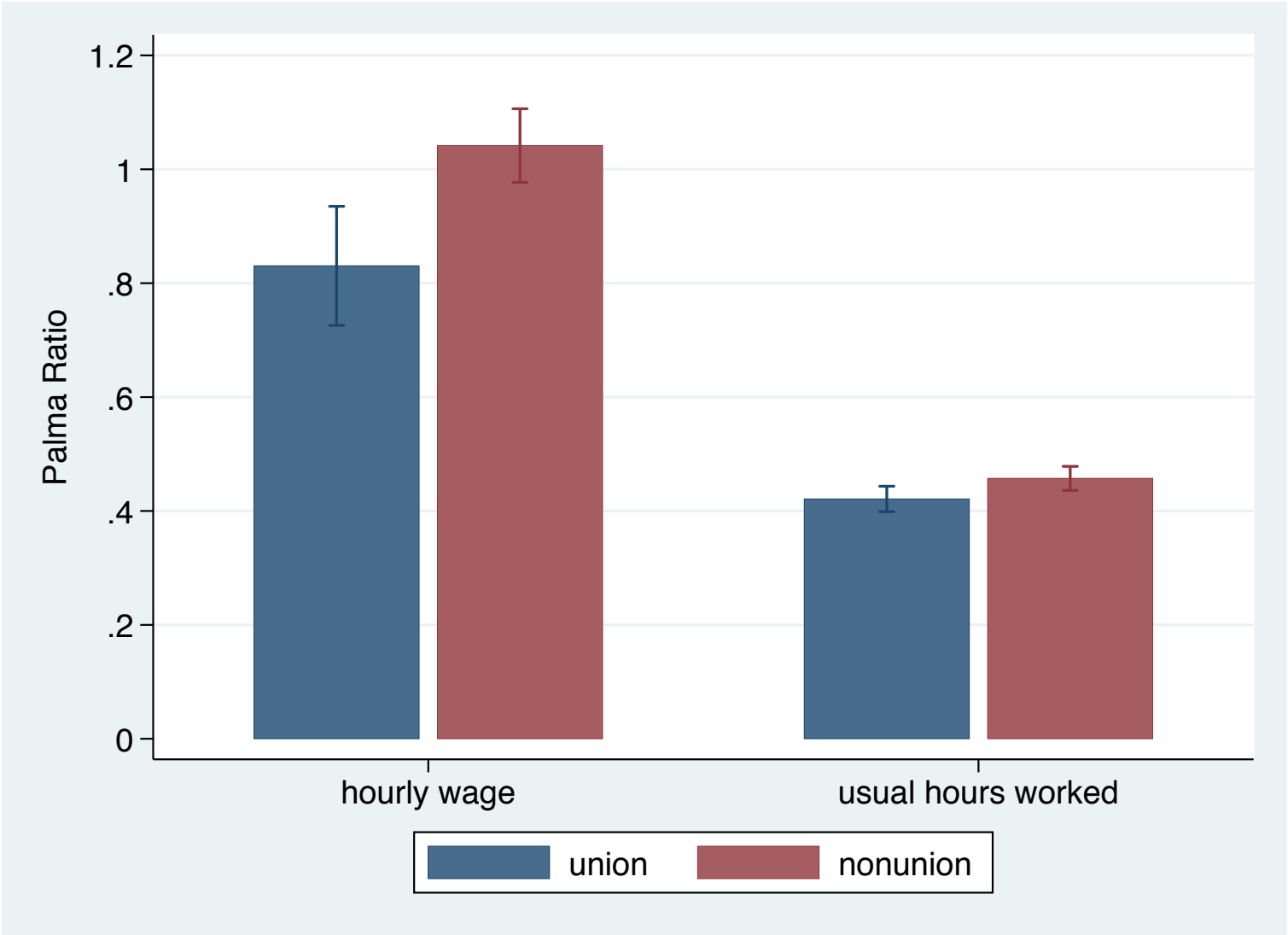
Standard errors in parentheses

```
. coefplot union nonunion
```



- 2. Schritt nochmals

```
. quietly estread myresults
. coefplot union nonunion, vertical recast(bar) base(0) barwidth(0.3) ///
>     finten(80) citop ciopts(recast(rcap)) ///
>     ylabel(0(.2)1.2, angle(horizontal)) ytitle(Palma Ratio)
```



Reproducibility

- Analysen sollten vollständig dokumentiert und reproduzierbar sein.
- Einige Prinzipien
  - Alles mit Skript, nichts manuell.
  - Um die Resultate zukünftig reproduzieren zu können, sollten Do-Files immer einen `version` Befehl enthalten. Falls Zufallszahlengeneratoren verwendet werden (Sampling, Bootstrap, Simulationen), sollte man zudem den Seed setzen.

```
version 13.1
set seed 34289024
...
```

  - Für komplexere Projekte lohnt sich die Aufteilung in verschiedene Do-Files, die von einem Master-Do-File ausgeführt werden.

```
// master do-file
do 1-datenaufbereitung.do
do 2-deskriptive-analyse.do
do 3-modelle.do
do 4-tabellen.do
```

Reproducibility

- Um die Analyse vollständig zu dokumentieren, eignet sich z.B. `texdoc` (`ssc install texdoc`).
- Ausgehend von einem Do-File, das Stata-Code und LaTeX-Code enthält, generiert `texdoc` ein LaTeX-File der Analyse inklusive Stata-Output, das dann zu einem PDF kompiliert werden kann.
- Ein Paper zu `texdoc` findet sich unter https://ideas.repec.org/p/bss/wpaper/14.html.
- Die vorliegenden Slides wurden übrigens auch mit `texdoc` generiert.