Gretl und Hansl

Statistik und Ökonometrie anwenden mit freier Software

Artur Tarassow[1]

[1] Fachbereich Wirtschaft
Technische Hochschule

Brandenburg an der Havel, 17.01.2024

Outline

Zweck und allgemeiner Charakter

Hauptzwecke der Ökonometrie und Statistik:[1]

- Testen von Theorien
- Prognostik
- Politikevaluation

Anwendung ökonomischer und statistischer Theorien auf Basis sozio-ökonomischer Daten.

Aber auch *Entwicklung* statistischer Theorien.

[1] Der folgende Inhalt basiert auf früheren Folien von von Allin Cottrell (Wake Forest University) and Riccardo ”Jack”Lucchetti (Università Politecnica delle Marche).

Doppelter Status der Ökonometrie

Die Ökonometrie ist nicht ein ”Teilgebiet”der Wirtschaftswissenschaften im gleichen Sinne wie z. B. Arbeitsökonomie oder Gesundheitsökonomie.

Vielmehr ein Set von *Werkzeugen*, die in fast allen Teilgebieten eingesetzt werden (außer in der reinen Theorie) – plus ein Fachgebiet, das ebenso viel mathematische Statistik wie Wirtschaftswissenschaften ist.

Jedoch ist ein Ökonometriker auch kein Statistiker: ein Ökonometriker ist ein Wirtschaftswissenschaftler mit einem überdurchschnittlichen Verständnis von Statistik.

Ökonometrie-Codierung

- Pioniere der 1960er Jahre verwendeten hauptsächlich Fortran (einige große Namen verwenden dies noch)
- In der frühen PC-Ära gab es Kommandozeilenprogramme, die vorgefertigte Routinen anboten
- Gauss (1984, MS-DOS), Matlab (ebenfalls 1984), Ox (gegen 1997)
- 1990er bis heute: Entwicklung von Kommandozeilenprogrammen: Hinzufügen von GUIs und auch Elemente matrixorientierter Sprachen (Stata, 1985; Eviews, 1994)
- Jüngste Tendenz: Pakete, die auf matrixorientierten Sprachen aufbauen (Dynare)
- Auch: Weg von domänen-spezifischen Sprachen hin zu allgemeinen Sprachen (Python, Julia)

Numerische Verfahren in der Ökonometrie

- Weit verbreitete Verwendung von Matrizen (hauptsächlich reell, nur selten komplex)
- Klassische Optimierungstechniken für stetige Funktionen
- Zufallszahlengenerator (RNG) (zunehmend beliebt, insbesondere für bayesianische Verfahren)
- Einige Konzepte aus der Ingenieurliteratur: Spektren, Filterung, Signalextraktion.

Traditionell ist die Dimensionalität von Problemen relativ klein: Ein paar KB RAM reichen oft aus, um Daten zu speichern. Dies ändert sich heutzutage mit 'Big Data'-Problemen rapid.

Der 'Markt' bzw. Anwendungen für ökonometrische Software

1. Bachelor- und Masterstudiengänge (eine große Branche)
2. Professionelle angewandte Arbeit
   1. akademische Nutzung (hauptsächlich Universitäten)
   2. geschäftliche Nutzung (traditionell große Finanzinstitute, aber heutzutage auch große Einzelhändler wie Amazon/ Google)
   3. Politik (Zentralbanken, andere Regierungs-/supranationale Institutionen)
3. Entwicklung neuer Schätzer/ Werkzeuge

Derzeit von proprietärer Software dominiert – wie z.B. Stata und Eviews (für 1 und 2), plus Matlab und Python (für 2.3 und 3).

Aber auch Verbreitung von R und Gretl, und geringe Verwendung von lower-level Programmiersprachen für Verwendung 3.

Sehr wenige Menschen benutzen kompilierte Sprachen (C, Fortran usw.); selbst nicht für rechenintensive Aufgaben.

Einige Hintergrundinformationen zu Gretl I

- Akronym für: **G**nu **R**egression, **E**conometrics und **T**ime-series **L**ibrary
- URL: `http://gretl.sourceforge.net/`
- Besteht aus
  1. einer großen Shared Library (lib-gretl)
  2. einem gemeinsamen Kommandozeilenprogramm (gretlcli)
  3. und einem GUI-Client (gretl)
- Verwendung zuverlässiger open-source Pakete, z.B. (multithreaded) LAPACK/BLAS, fftw, GTK, gnuplot, etc.

Einige Hintergrundinformationen zu Gretl II

- Erste Version wurde im Januar 2000 veröffentlicht
- Wird seitdem aktiv weiterentwickelt → *Open-Source* und *kostenlos*.
- In C geschrieben und für Windows, OS X und Linux verfügbar.
- Benutzeroberfläche ist in 16 Sprachen verfügbar.
- Gretl ist in einem *Benutzerhandbuch* von über 467 Seiten und einem *Kommandoreferenz* von über 267 Seiten dokumentiert

Einige Hintergrundinformationen zu Gretl III

- Gretl verfügt über eine voll ausgestattete grafische Benutzeroberfläche (GUI).
- Ausführung von Befehlen und Funktionen via Hansl-Skripting oder durch die GUI gesteuert.

Alleinstellungsmerkmal

- Gretl bietet eine **hoch entwickelte matrix-orientierte Sprache** ähnlich zu Matlab und Gauss
- **UND** eine hoch entwickelte Sprache, die auf **Ökonometrie/ Statistik** abgestimmt ist.

Datensätze

Ein Datensatz ist im Wesentlichen die Vereinigung der Matrizen **y** (T x k) und **X** (T x m) mit zusätzlichen Metadaten.

- Drei Datentypen sind relevant: (i) Querschnitt, (ii) Zeitreihe und (iii) Panel.
- Betrachten Sie einen Datensatz als eine große Matrix. Um eine Regression zu verstehen, muss man wissen:
  - Worauf beziehen sich die Spalten und Zeilen?
  - Repräsentation der Reihen als Zeitperioden: (i) Beginn und Ende der Stichprobe, (ii) Frequenz, mit der Daten erfasst wurden?
- In ökonometrischer Software ist der Datensatz typischerweise nicht als solche eine Matrix, sondern eine reichhaltigere Struktur und umfasst Metadaten.

Datensätze

1. Dualität in Hansl: die Verfügbarkeit des *Datensatzes* als einer spezifischen Datenstruktur neben Computerdarstellungen des Standardmathematischen Typs.

- Datensatz (plus Serien, Listen von Serien)
- *Bundle* (als Träger-Objekt für weitere Datentypen; eine Art *dictionary*)

2. Dualität

- Matrizen, Skalare, Zeichenfolgen, Arrays, Bündel, Funktionen (schwieriger aber flexibler) *versus*
- Datensätze, Serien, Befehle (einfach)

Abschwächung der Dualität durch Zugriffselemente, die nach Befehlen verwendet werden können.

Skalare oder Matrizen

Kommando

```
scalar a = 1.5
scalar b = -2

print a b

matrix m = { 1, 2, 3;\
             4, 5, 6 }

matrix n = seq(1, 10, 2)

print m n
```

Output

```
a =  1.5000000

b = -2.0000000

m (2 x 3)
1   2   3
4   5   6

n (1 x 5)
1   3   5   7   9
```

Serien und Listen

Kommando

```
# Erstelle Datensatz n=3
nulldata 3

# Zufalls-Variablen
series y = normal()
series x = normal()

print y x -o

list L = y x

print L -o --range=1:2
```

Output

```
                y             x
1        0.373209    -1.846103
2       -0.338461     1.076625
3        1.053826    -1.874034

                y             x

1        0.373209    -1.846103
2       -0.338461     1.076625
```

Strings

Kommando

```
set verbose off
string s = "Hello world!"
print s

string a = "Hier ist"
string b = " Gretl!"
string c = a ~ b

print c
```

Output

```
Hello world!
Hier ist Gretl!
```

Arrays – Strings

Kommando

```
set verbose off
strings S = array(2)
S[1] = "Hello"
S[2] = "world!"
print s

S += "Hier
print S
```

Output

```
Hello world!
Array of strings, length 3
[1] "Hello"
[2] "world!"
[3] "Hier ist Hier ist "

Array of strings, length 2
[1] "world!"
[2] "Hier ist Hier ist "
```

Arrays – Matrizen

Kommando

```
set verbose off
matrices M = array(2)
M[1] = { 1, 2, 3;\
         4, 5, 6 }

M[2] = seq(1, 10, 2)

print M

print M[1]
print M[2]
```

Output

```
Array of matrices, length 2
[1] 2 x 3
[2] 1 x 5

1   2   3
4   5   6

1   3   5   7   9
```

Bundle

Kommando

```
set verbose off

bundle B = _(a = "Hello, world!",
 b = -1.5,
 c = { 1, 2, 3; 4, 5, 6 },
 d = defarray("A", "B", "C"))

print B

print B.a
print B["d"]

B.b = 33
print B.b
```

Output

```
bundle B:
b = -1.5
c (matrix: 2 x 3)
a = "Hello, world!"
d = array of strings, length 3

Hello, world!
Array of strings, length 3
[1] "A"
[2] "B"
[3] "C"

33
```

Hansl – Hansl's A Neat Scripting Language

- Der Werdegang der Gretl-Entwicklung ist der proprietärer Software ähnlich (DOS-Kommandozeilenprogramm → Cross-Plattform → GUI hinzufügen → Matlab-ähnliche Matrixfunktionalität hinzufügen → Erweitertes Scripting hinzufügen → Parallelisierung)
- 'Gefühl' der Skriptsprache hat Ähnlichkeiten zur Bash Shell (UNIX).
- C-Back-End (natürlich mit ein wenig Hilfe von Freunden: netlib, BLAS, lapack, FFTW und anderen)
- Übergang zur Entwicklung von Gretl über Hansl (Funktionspakete mit optionaler GUI-Integration)
- Einige 'Legacy'-Formulierungen und Inkonsistenzen, aber Hansl ist sauber und einfach zu lernen

Hansl – Befehle

Hansl umfasst mehr 204 Befehle:[2]

- Tests (Hypothese)
- Statistik und Wahrscheinlichkeitsrechnung
- Datensatz (Manipulation, Sortierung, usw.)
- Schätzung (OLS, MLE, GMM, Einzelgleichung und Systeme usw.)
- Graphen (Streudiagramme, Boxplots, Zeitreihen, usw.)
- Programmierung (Steuerungsfluss und Fehlersuche)
- Transformationen
- String-Operationen
- Prognostik

[2] `https://gretl.sourceforge.net/gretl-help/cmdref.html`

Hansl – Funktionen

Hansl umfasst etwa 250 Funktionen:[3]

- mathematische
- statistische
- Strings
- Daten-Tools
- Programmierung
- numerische Methoden
- Matrix-Manipulation
- Zeitreihen
- Transformationen
- Komplexe Zahlen
- Wahrscheinlichkeitsrechnung
- Lineare Algebra
- Kalenderfunktionen
- nicht-parametrische Modelle

[3] `https://gretl.sourceforge.net/gretl-help/funcref.html`

Modelle

Implementiert sind eine Vielzahl von Modellen und Methoden

- Zeitreihenmethoden
  - ARIMA, univariate GARCH-Typ, (S)VARs und VECMs, Einheitswurzel- und Gleichgewichtstests, Kalman-Filter, MIDAS, Echtzeit-Datensätze
- Begrenzt abhängige Variablen
  - logit, probit, tobit, Stichprobenselektion, Intervallregulierung, Modelle für Zähler- und Dauerdaten usw.
- Panelldatenschätzer, einschließlich Instrumentvariablen, Probit- und GMM-basierter dynamischer Panelmodelle
- Maschinelles Lernen: Ridge, LASSO, Elastic-Net, SVM, Random Forests (via R)

3rd-party Pakete von Nutzern

Derzeit werden über 100 nutzergeschriebene Pakete bereitgestellt.

Siehe hier:

`https://gretl.sourceforge.net/cgi-bin/gretldata.cgi?opt=SHOW_FUNCS`

Installieren und Laden von 3rd-party Paketen

```
set verbose off
# Installiere Paket vom Server
pkg install PairPlot

# Lade Paket in den Speicher
include PairPlot.gfn

# Zeige die Hilfe
help PairPlot
```

Beispiel-Skripte sind in jedem Gretl-Paket enthalten.

Unterstützte Datenformate

Unterstützte Formate, um Daten zu laden umfassen:

- Eigene XML-Datendatei (`*.gdt` und `*.gdtb`)
- Komma-separierte Textdatei (txt, csv)
- Excel-Arbeitsblätter
- Gnumeric und OpenDocument-Arbeitsblätter
- Stata-Dateien (.dta)
- SPSS-Dateien (.sav)
- Eviews-Arbeitsdateien
- JMulTi-Datendateien
- Eigene Binärdatenbanken im eigenen Format (ermöglicht gemischte Datenfrequenzen und Serienlängen)
- RATS 4-Datenbanken und PC-Give-Datenbanken
- Beinhaltet eine Beispieldatenbank für die US-Wirtschaft. Weitere Informationen finden Sie auf der Gretl-Datenseite.

Kommunikation mit anderen Programmen

Gretl kann mit anderen Softwarepaketen interagieren.

Datensätze und Matrizen einfach senden und empfangen.

Andere Programme über Gretls `foreign-language`-Block aufrufen.

Liste der unterstützten Software:

- R (noch mehr Unterstützung)
- Ox
- Octave
- Stata
- Python
- Julia

Gretl und R Beispiel

```
function list RStructTS(series myseries)
  smpl ok(myseries) --restrict
  sx = argname(myseries)
  foreign language=R --send-data --quiet
    @sx <- gretldata[, "myseries"]
    strmod <- StructTS(@sx)
    compon <- as.ts(tsSmooth(strmod))
    gretl.export(compon)
  end foreign
  append @dotdir/compon.csv
  rename level @sx_level
  rename slope @sx_slope
  rename sea @sx_seas
  list ret = @sx_level @sx_slope @sx_seas
  return ret
end function
# ------------ main -------------------------
open bjg.gdt
list X = RStructTS(lg)
```

Materialien zu Gretl I

- *Material-on-Gretl*
  `https://github.com/gretl-project/material-on-gretl`
- *Wiki 1*
  `https://github.com/gretl-project/material-on-gretl/wiki`
- *Wiki 2*
  `https://gretlwiki.econ.univpm.it/index.php/Main_Page`
- *Gretl Command Reference*
  `https://gretl.sourceforge.net/gretl-help/cmdref.html`
- *Gretl User's Guide*:
  `http://sourceforge.net/projects/gretl/files/manual/`

Materialien zu Gretl II

- *gretl-users E-Mail-Liste*: Die meisten gut überlegten Fragen werden relativ schnell beantwortet und ausführlich beantwortet. `https://gretlml.univpm.it/postorius/lists/gretl-users.gretlml.univpm.it/`
- Lehrbuch *Using gretl for Principles of Econometrics* (5. Auflage) `http://www.learneconometrics.com/gretl/`
- Gretl Cheat-sheet `https://github.com/gretl-project/gretl_cheatsheet`
- *Beispielskripts*: Das Gretl-Paket enthält eine Vielzahl von Beispiel- oder Übungsskripts (unter dem Menüpunkt /Datei/Skriptdateien/Übungsdatei).
- *Funktionspakete*: Ambitionierte Beispiele für Hansl-Codierung (über den Gretl-Menüpunkt /Werkzeuge/Funktionspakete/Auf Server).)

Mit dem Datensatz arbeiten

Serien erzeugen

```
set verbose off

# Erstelle Datensatz n=3
nulldata 3

# Normalverteilte Zufallsvariable
# mean = 4, std-dev = 0.5
series y = normal(4, 0.5)

series y_sq = y^2
series log_y = log(y)
series exp_y = exp(y)
series x = {1, 2, 3}'

series z = y - x

print y y_sq log_y exp_y x z --byobs
```

```
             y          y_sq         log_y         exp_y          x
1     4.353309      18.95130      1.470936       77.7353          1
2     4.267813      18.21423      1.451102       71.3654          2
3     4.985425      24.85446      1.606519      146.2658          3

             z

1     3.353309
2     2.267813
3     1.985425
```

Dummyvariablen erzeugen

```
open "./data/abdata.csv" --quiet --preserve

# Dummyvariablen erzeugen
series DUM = (YEAR == 1977 || YEAR == 1980)

print YEAR DUM -o --range=1:10
```

```
gretl version 2024a-git
Current session: 2024-01-16 13:35

? print YEAR DUM -o --range=1:10

             YEAR          DUM

 1         1976            0
 2         1977            1
 3         1978            0
 4         1979            0
 5         1980            1
 6         1981            0
 7         1982            0
 8         1983            0
 9         1984            0
10         1976            0
```

Metadaten für Serien hinzufügen

```
nulldata 3
series y = normal()

# Beschreibung für Serie hinzufügen
setinfo y --description="Some random number"

# Anstelle von 'y' soll 'Cool variable'
# bei Plots erscheinen
setinfo y --graph-name="Cool variable"

boxplot y --output=display
```

Werte ersetzen

```
set verbose off

# Neuen Datensatz mit 4 Beobachtungen
# erstellen
nulldata 5 --preserve
# Generiere Serie mit komischen Werten
series weird_values = {5, 6, 10, 20, NA}'
print weird_values --byobs

# Let's replace values
help replace
matrix find = {5, 6, 10, 20, NA}
matrix replace_by = {0, 1, 2, 3, -1}

series y = replace(weird_values, find, replace_by)
print weird_values y --byobs
```

```
weird_values

1             5
2             6
3            10
4            20
5

weird_values            y

1             5         0
2             6         1
3            10         2
4            20         3
5                      -1
```

String-Werte in Series einfügen

```
set verbose off
nulldata 20

series y = randgen(i, 1, 3)
setinfo y --description="3 different categories"

print y --byobs --range=1:5

# Create strings for categories 1, 2 and 3
strings series_labels = defarray("Low income",\
  "Medium income", "High income")
# Attache strings to categorical series
help stringify
stringify(y, series_labels)

print y --byobs --range=1:5
```

```
                y

1               2
2               3
3               2
4               1
5               3
```

```
                y

1 Medium income
2   High income
3 Medium income
4    Low income
5   High income
```

Einlesen eines Datensatzes und Werte zeigen

```
clear
set verbose off

# Arbeitsordner definieren
string DIR = "<PATH>"
set workdir "@DIR"

# Relative Pfadangabe zum
# Ordner "data"
open "./data/abdata.csv"

# Zeige die ersten 10 Zeilen
# einiger Serien
list Y = IND YEAR n w k
print Y --byobs --range=1:10
```

	IND	YEAR	n	w	k
1		1976			
2	7	1977	1.617604	2.576543	-0.5286502
3	7	1978	1.722767	2.509746	-0.4591824
4	7	1979	1.612433	2.552526	-0.3899363
5	7	1980	1.550749	2.624951	-0.4827242
6	7	1981	1.409278	2.659539	-0.6780615
7	7	1982	1.152469	2.699218	-0.8606196
8	7	1983	1.077048	2.623102	-0.9364935
9		1984			
10		1976			

Datensatz um 'markers' (Beobachtungslabels) erweitern

```
nulldata 4
series y = {1, 2, 3, 4}
series x = normal()

strings S = defarray("Artur", "Fritzi",
  "Katharina", "Olga")
markers --from-array=S

print y x -o

# Berühre die Datenunkte
# und die Labels erscheinen
gnuplot y x --output=display
```

```
                y             x

Artur           1    -0.6772452
Fritzi          2     0.4454971
Katharina       3     0.4316438
 Olga           4     0.1343885
```

Deskriptive Statistiken

```
list Y = IND YEAR n w k

summary Y --simple
```

```
          Mean      Median      S.D.       Min       Max
IND      5.123       5.000     2.678     1.000     9.000
YEAR      1980        1980     2.583      1976      1984
n        1.056      0.8272     1.342    -2.263     4.687
w        3.143       3.178    0.2630     2.082     3.812
k      -0.4416    -0.6578     1.514    -4.431     3.852
```

```
# Statistiken je Jahr
# wobei die Jahre
# eingeschränkt werden
list Y = n w k

smpl YEAR >= 1976 \
  && YEAR <= 1977 --restrict

summary Y --by=YEAR --simple
```

```
YEAR = 1976 (n = 80):

             Mean      Median      S.D.       Min       Max
n           1.239       1.033     1.382    -1.444     4.588
w           3.237       3.266    0.2683     2.178     3.812
k         -0.2912    -0.5390     1.518    -2.966     3.531

YEAR = 1977 (n = 138):

             Mean      Median      S.D.       Min       Max
n           1.160      0.9683     1.340    -1.952     4.597
w           3.129       3.169    0.2714     2.123     3.730
k         -0.3876    -0.6218     1.472    -3.393     3.477
```

Aggregation

```
# Aggregiere die folgenden Variablen
list Y = n w

# Gruppiere nach der Variable IND
list groupby = IND

# Berechne den Mittelwert für 'Y' für jede
# Ausprägung von 'groupby'
matrix mean_values = aggregate(Y, groupby, "mean")

# Optionale Formatierung des Outputs
printf "\n%12.2f\n", mean_values
```

IND	count	n	w
1.00	41.00	1.27	3.11
2.00	32.00	1.23	3.50
3.00	34.00	0.72	3.25
4.00	76.00	1.44	3.32

Wegschreiben der Textausgabe

```
# Berechne den Mittelwert für 'Y' für jede
# Ausprägung von 'groupby'
matrix mean_values = aggregate(Y, groupby, "mean")

# Textausgabe als txt-Datei speichern
string filename = "./output/aggregation_mean_output.txt"

outfile "@filename
printf "\n%12.2f\n", mean_values
end outfile
```

Matrix als csv speichern und öffnen der csv

```
# Berechne den Mittelwert für 'Y' für jede
# Ausprägung von 'groupby'
matrix mean_values = aggregate(Y, groupby, "mean")

# Speichern der Matrix als csv
string filename = "./output/aggregation_als_matrix.csv"
store "@filename" --matrix=mean_values

# Matrix-csv als Datensatz einlesen für
# weitere Bearbeitung
open "@filename" --preserve
```

Sample restringieren

```
open abdata --quiet

list Y = IND YEAR n w

# Statistiken für alle Beobachtungen
summary Y --simple

# Statistiken für Jahre zw. 1976 - 1978
smpl YEAR >= 1976 && YEAR <= 1978 --restrict
summary Y --simple

# Wiederherstellen des vollen Datensatzes
summary Y --simple
```

IND	count	n	w
1.00	41.00	1.27	3.11
2.00	32.00	1.23	3.50
3.00	34.00	0.72	3.25
4.00	76.00	1.44	3.32

KQ Regression

```
# Conduct OLS regression (optionally with robust standard errors)
ols ys const n w  #--robust

# Store coefficients
matrix coeff = $coeff

# Store standard errors
matrix stderr = $stderr

# Store residuals
series uhat = $uhat

# Store fitted values
series yhat = $yhat

# Store R^2
scalar r2 = $rsq
```

```
Model 9: OLS, using observations 1-1260 (n = 1031)
Missing or incomplete observations dropped: 229
Dependent variable: ys

 coefficient   std. error   t-ratio    p-value
  ---------------------------------------------------------
  const       4.60388       0.0351033    131.2      0.0000  ***
  n           0.00626942    0.00217539     2.882    0.0040  ***
  w           0.00875566    0.0110959      0.7891   0.4302

Mean dependent var   4.638015   S.D. dependent var   0.093961
Sum squared resid    9.015800   S.E. of regression   0.093650
R-squared            0.008551   Adjusted R-squared   0.006622
F(2, 1028)           4.433125   P-value(F)           0.012105
Log-likelihood       980.1866   Akaike criterion    -1954.373
Schwarz criterion   -1939.558   Hannan-Quinn        -1948.751
```

Spezifikationstests

```
# See the help on tests
help modtest

# Conduct OLS regression
ols ys const n w  #--robust

# Tests are conducted for the
# last model estimated

# Normality test
modtest --normality --quiet

# White's test on homoscedasticity
modtest --white
scalar teststat_white = $test
scalar pvalue_white = $pvalue

# RESET test on functional form
reset --squares-only
```

```
Test for null hypothesis of normal distribution:
Chi-square(2) = 82.810 with p-value 0.00000
```

```
White's test for heteroskedasticity
Test statistic: TR^2 = 29.920521,
with p-value = P(Chi-square(5) > 29.920521) = 0.000015
```

```
RESET test for specification (squares only)
Null hypothesis: specification is adequate
Test statistic: F = 12.054076,
with p-value = P(F(1,1027) > 12.0541) = 0.000538
```

Testen von Restriktionen

```
# Restriktionen
ols ys const n w

# t-Test: H0: beta(n) = 0,
# H1: beta(n) != 0
help omit
omit n --test-only

# F-Test: H0: beta(n) = beta_(w) 0,
# H1: beta(n) != 0 und/ oder
# beta_(w) != 0
list drop = n w
omit drop --test-only

# More flexible: the restrict-block
help restrict

restrict # --bootstrap
      b[const] = 4
  b[w] = 0.1
end restrict
```

```
Test on Model 13:
Null hypothesis: the regression parameter is zero for n
Test statistic: F(1, 1028) = 8.30579, p-value 0.00403412

Test on Model 13:
Null hypothesis: the regression parameters are
zero for the variables
n, w
Test statistic: F(2, 1028) = 4.43312, p-value 0.0121052


Restriction set
1: b[const] = 4
2: b[w] = 0.1

Test statistic: F(2, 1028) = 3675.38, with p-value = 0
```

Zusammenfassung Regressionsergebnisse eines Modells

Speicher Regressionsergebnisse als RTF- (Word) oder tex-Datei.

```
open "./data/abdata.csv" --quiet --preserve

ols ys const n w --simple

tabprint --output="./output/regression_output.rtf"
```

Alternativ für tex-Dateien auch der Befehl `eqnprint`.

Zusammenfassung Regressionsergebnisse mehrer Modelle

Speicher Regressionsergebnisse mehrer Modelle als RTF- (Word) oder tex-Datei.

```
open abdata.gdt --quiet --preserve
modeltab free
m1 <- ols ys const --quiet
modeltab add

m2 <- ols ys const n --quiet
modeltab add

m3 <- ols ys const n w --quiet
modeltab add

modeltab show
modeltab --output="./output/regression_table.rtf"
```

Daten plotten

Plotting

- Gnuplot als Plotting-Engine (sehr leistungsfähig) (`http://www.gnuplot.info/`)
- Es gibt für die grundlegenden Plots bereits native Unterstützung in Gretl
- Hier einige Beispiel

Scatterplot

```
set verbose off

open mroz87.gdt

gnuplot WE log(FAMINC) \
  --output=display
```

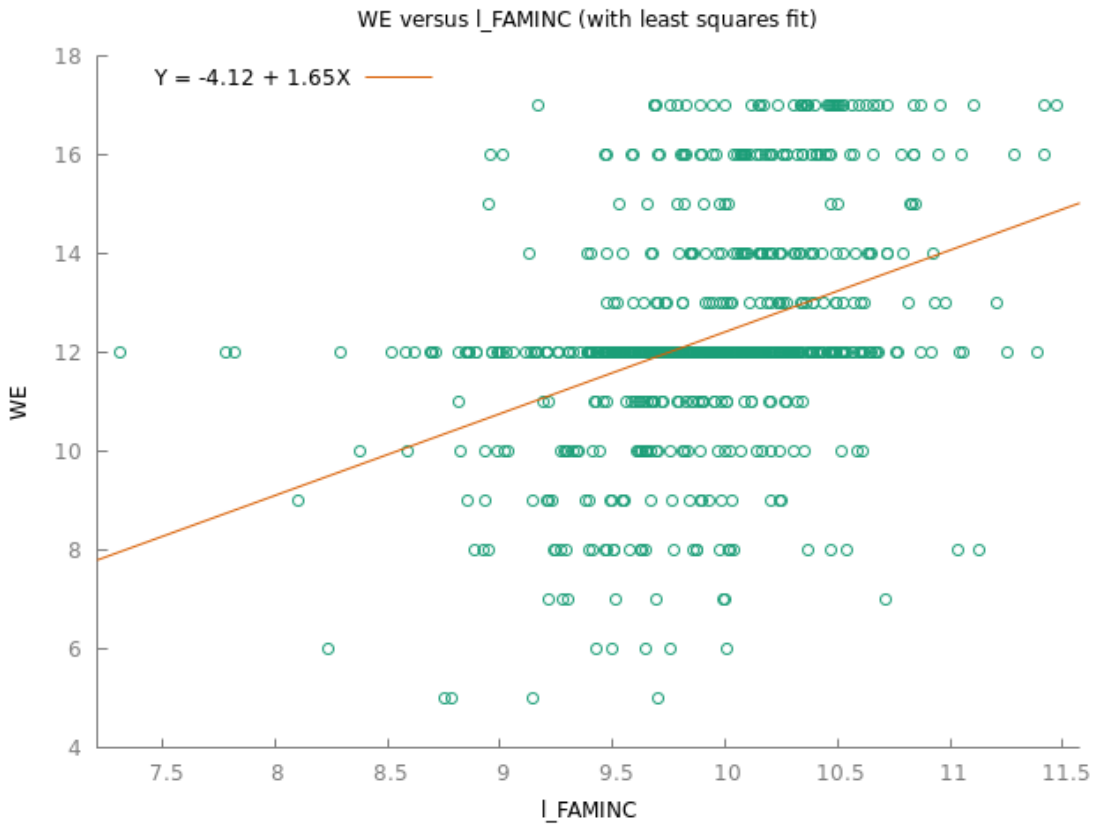

Abbildung: Scatterplot mit Gretl

Scatterplot mit markers

```
open mrw.gdt --quiet

gnuplot gdp60 school --output=display
```

Recht-klick in den Plot und dann Äll data labelsäuswählen.

	school
Algeria	4.5
Angola	1.8
Benin	1.8
Botswana	2.9
Burkina	0.4

Scatterplot mit Dummyausprägungen

```
set verbose off

open mroz87.gdt

gnuplot WE log(FAMINC) CIT --dummy \
  --output="scatterplot_factorized.png" \
  { set title "Some cool title" font ',15';\
  set linetype 1 lc rgb 'orange' ps 1;\
  set linetype 2 lc rgb 'blue' ps 0.5;\
  set xtics font ',15';\
  set grid;}
```

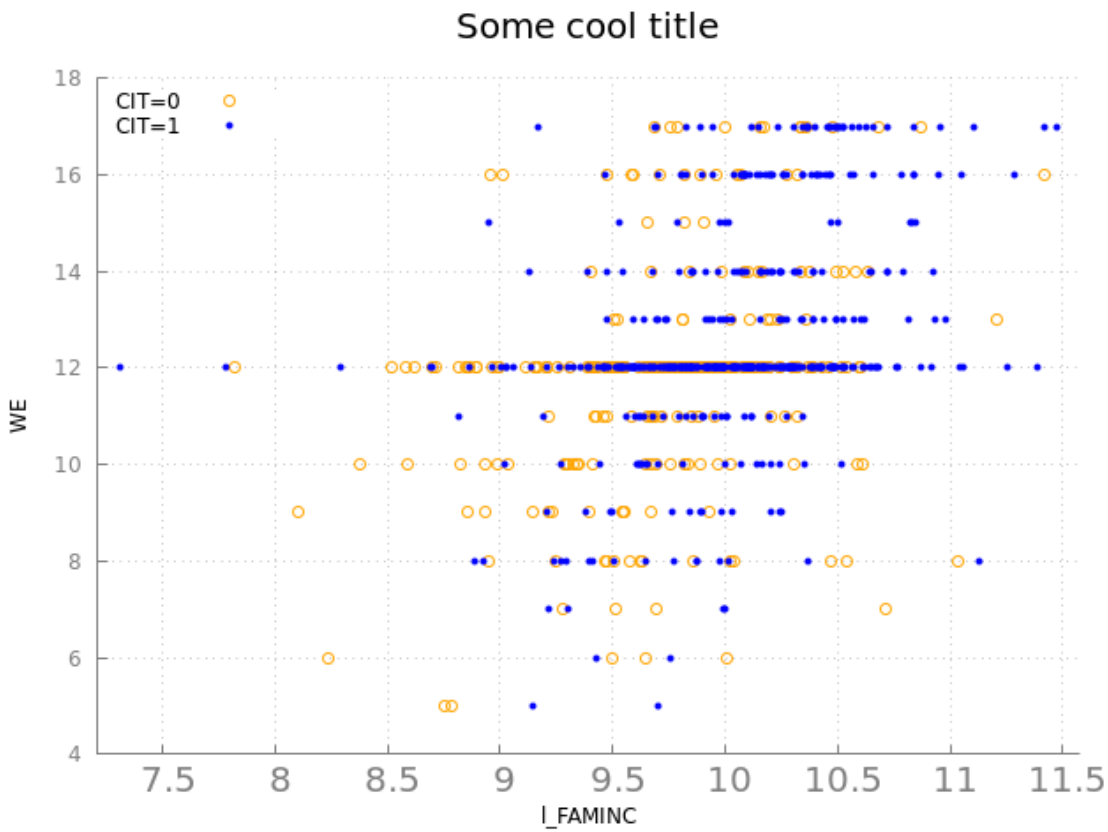

Abbildung: Scatterplot mit separaten Punkten je Ausprägung der Dummy-Variable `CIT`

Scatterplot-Matrix

```
set verbose off

open mroz87.gdt

list Y = WHRS WW HHRS

scatters WE ; Y \
  --output=display
```

Abbildung: Scatterplot mit `WE` auf der x-Achse aber verschiedenen Variablen auf der y-Achse

Boxplot

```
set verbose off
open mroz87.gdt

boxplot FAMINC --output=display
```

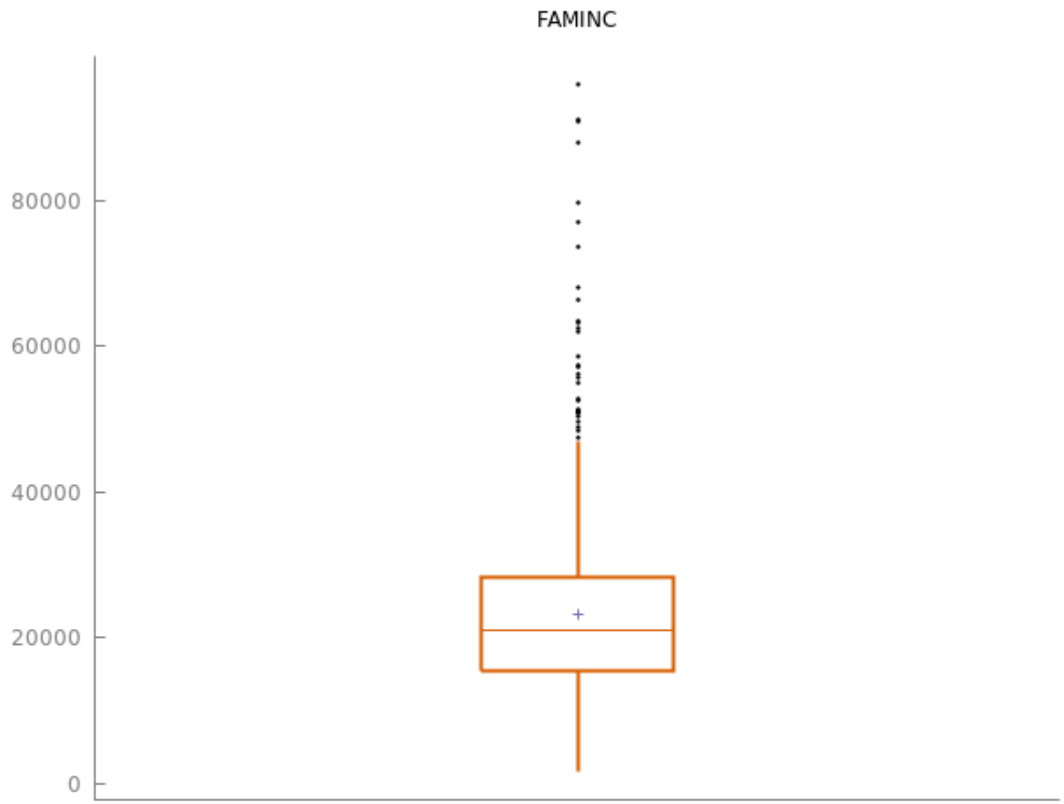

Abbildung: Boxplot

Boxplot je Dummyausprägungen

```
set verbose off

open mroz87.gdt

boxplot FAMINC WE --factorized \
  --output="boxplot_factorized.png" \
  { set grid; set title "Foo" font ',15';\
  set xlabel "Ausbildungsjahre" font ',14';}
```

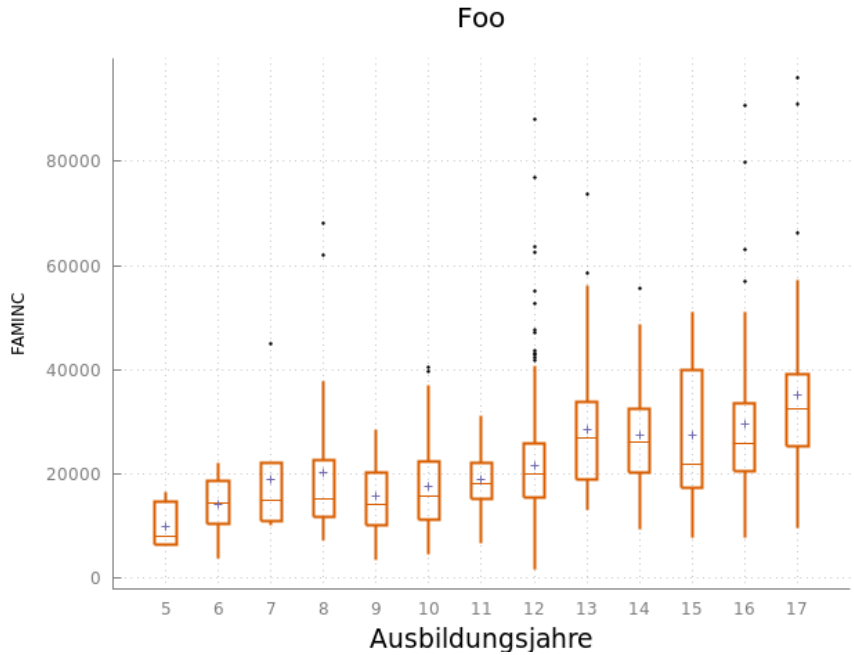

Abbildung: Boxplot je Ausbildungsjahr

Kern-Dichte Plot

```
set verbose off

open mroz87.gdt

kdplot FAMINC --output=display
```

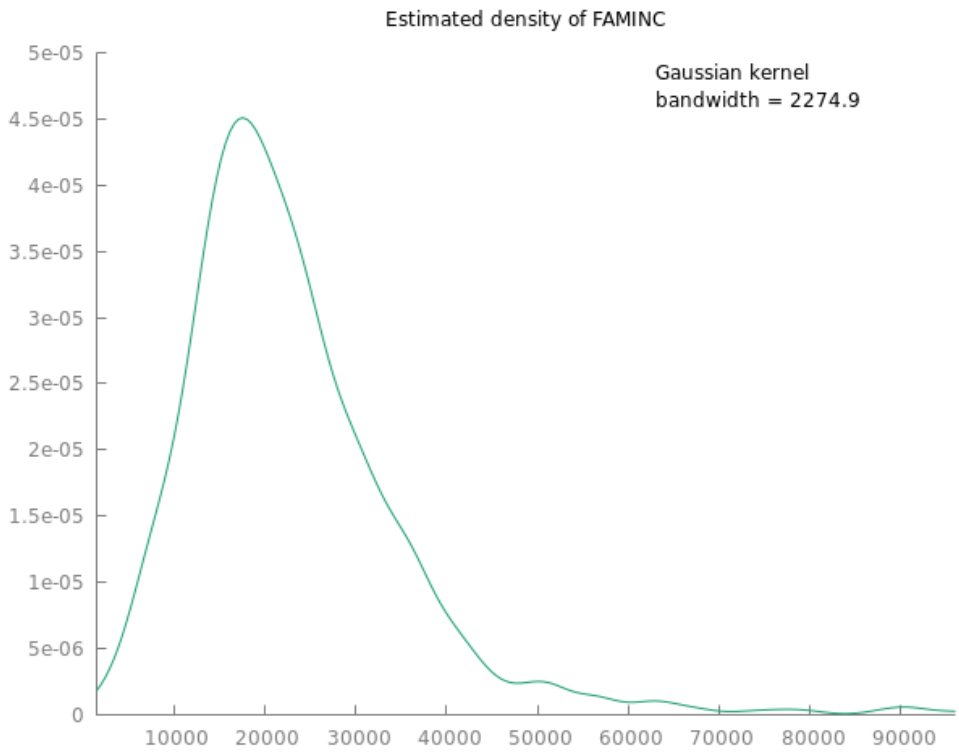

Abbildung: Kerndichteschätzung

Histogramm

```
set verbose off

open mroz87.gdt

freq FAMINC --gamma --plot=display
```

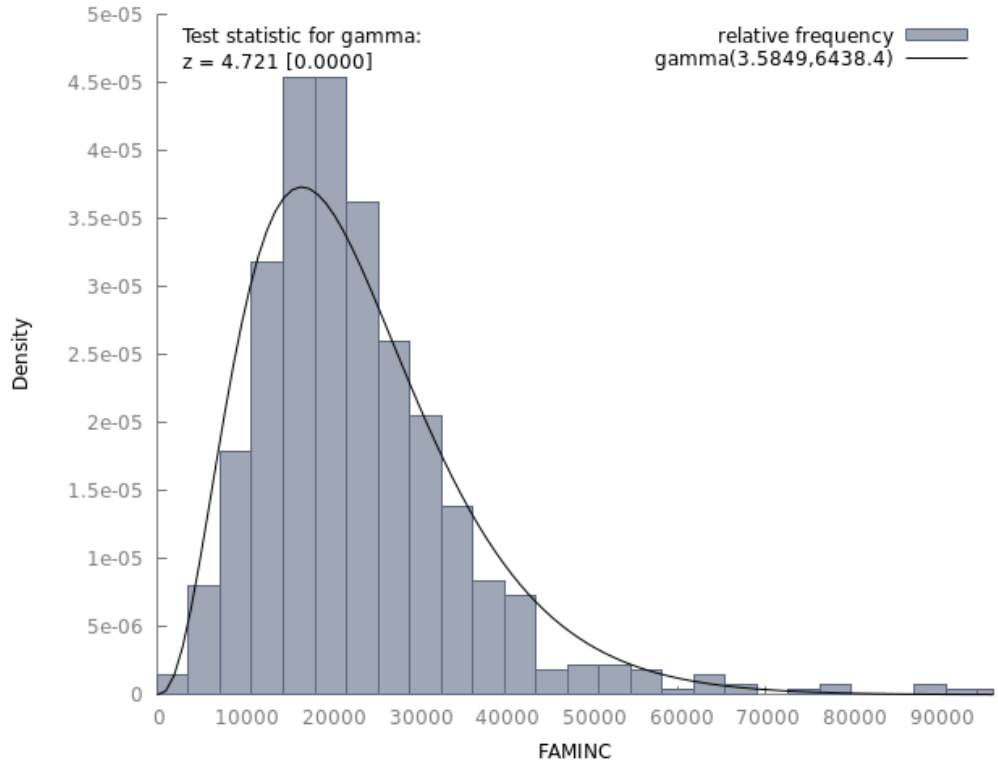

Abbildung: Histogramm mit Gamma-Verteilung

Korrelationsmatrix

```
set verbose off

open mroz87.gdt

list L = 1..5
corr L --triangle --plot=display
```

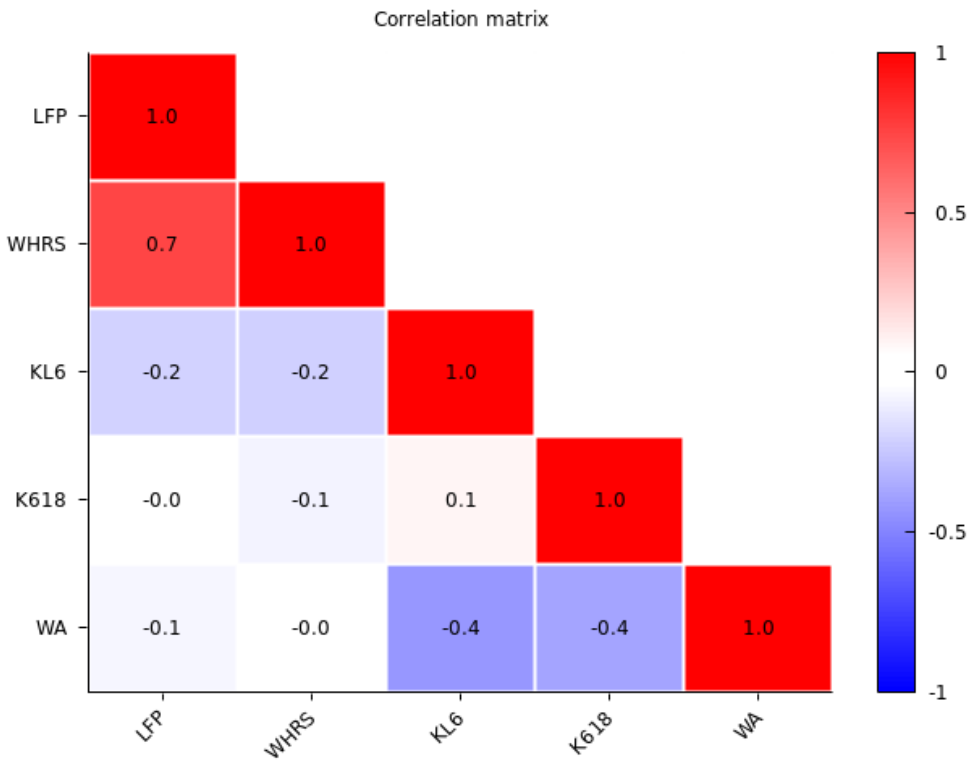

Abbildung: Korrelationsmatrix

Gridplot

```
set verbose off
open data4-10

strings MyPlots

gpbuild MyPlots
  gnuplot ENROLL CATHOL
  gnuplot ENROLL INCOME
  gnuplot ENROLL COLLEGE
  boxplot INCOME REGION --factorized
end gpbuild

gridplot MyPlots --output=display
```

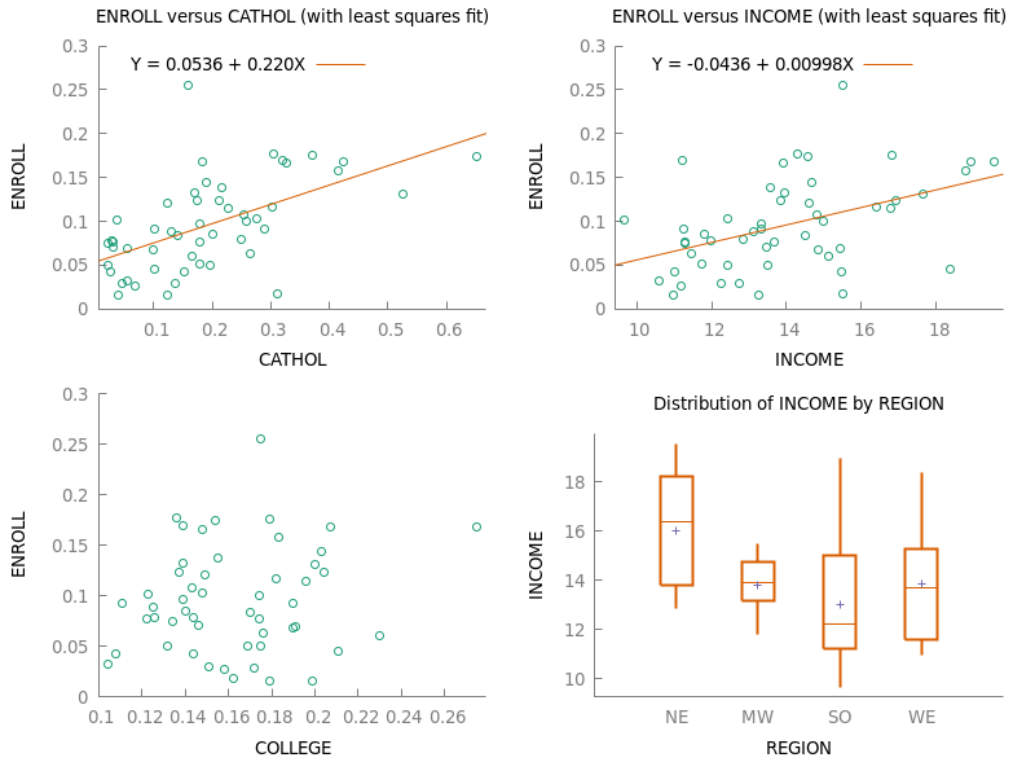

Abbildung: Korrelationsmatrix

Heatmap

```
open grunfeld.gdt --quiet
# Install and load the package
pkg install heatmap
include heatmap.gfn
# help heatmap

# Restict sample to the first 4
# panel units
smpl $unit < 5 --restrict

# Add information to the plot
bundle Options = \
  _(title = "Matrix A with contour lines",
  quiet = TRUE, xlabel = "Time dimension",
  ylabel = "Company")

scalar T = $pd
scalar N = $nobs / $pd
matrix m = value

matrix M = mshape(m, T, N)'
```

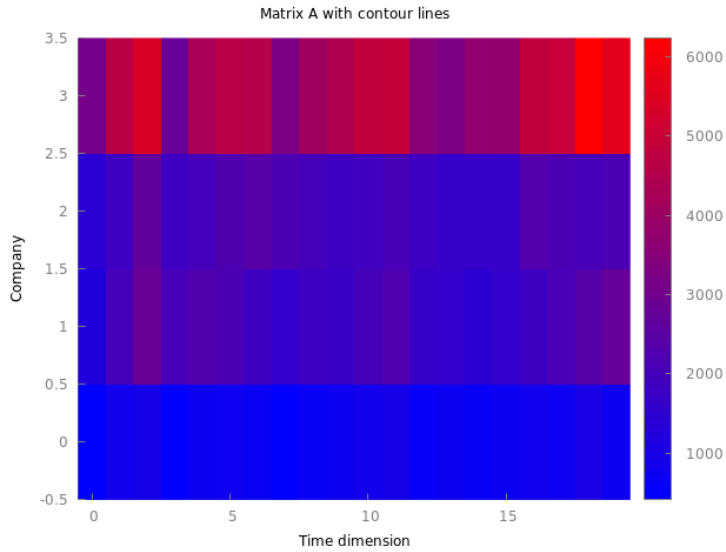

Abbildung: Heatmap

Pair-Plots

```
pkg install PairPlot
include PairPlot.gfn
#help PairPlot

open abdata --quiet
list y = n k
series factor = IND

bundle opts = _(transparency_level = 175,
  centroid = "median",
  tics = FALSE,
  pointsize = 1.5,
  centroid_pointsize = 3,
  centroid_linewidth = 3,
  height = 600,
  width = 600)

PairPlot(y, factor, opts)
```

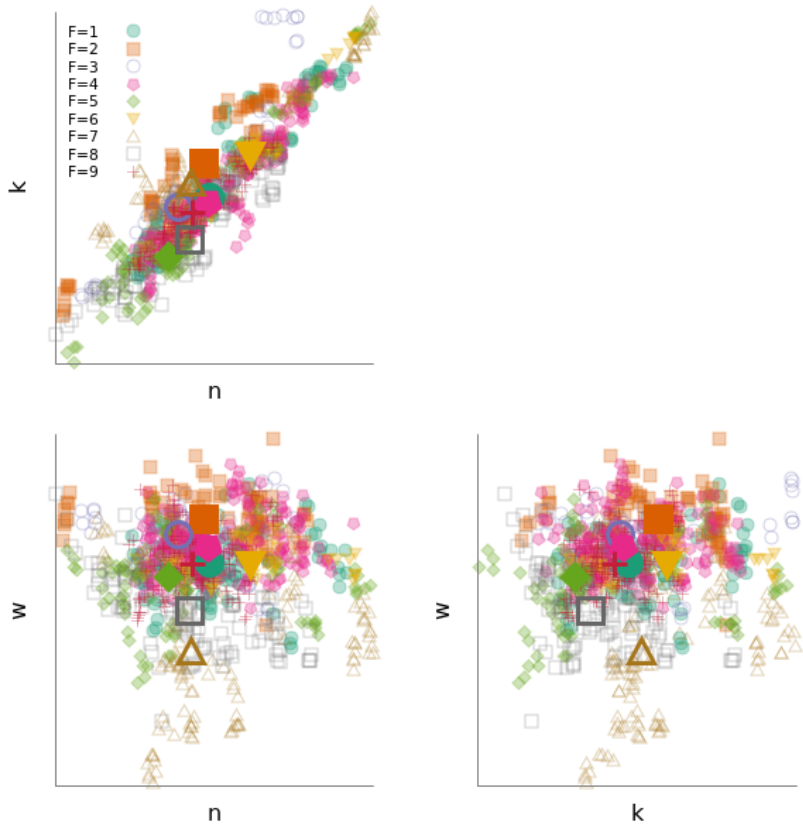

Abbildung: Kombinationen an Scatterplots mit Faktoren