Guida all'uso di gretl

Gnu Regression, Econometrics and Time-series library

Allin Cottrell
Department of Economics
Wake Forest university

Riccardo "Jack" Lucchetti
Dipartimento di Economia
Università Politecnica delle Marche

Cristian Rigamonti
(traduzione italiana)

Febbraio 2008

ATTENZIONE: questa traduzione è aggiornata alla versione 1.7.1 di gretl. Per versioni successive del programma si rimanda alla documentazione in inglese.

Indice

1 Introduzione **1**

1.1 Caratteristiche principali 1
1.2 Ringraziamenti 1
1.3 Installazione del programma 2

I Uso del programma **4**

2 Iniziare **5**

2.1 Eseguire una regressione 5
2.2 Risultati della stima 6
2.3 I menù della finestra principale 8
2.4 Scorciatoie da tastiera 11
2.5 La barra degli strumenti di gretl 12

3 Modalità di lavoro **13**

3.1 Script di comandi 13
3.2 Salvare oggetti da uno script 15
3.3 Il terminale di gretl 15
3.4 Il concetto di sessione 16

4 File di dati **19**

4.1 Formato interno 19
4.2 Altri formati dei file di dati 19
4.3 Database binari 19
4.4 Creazione di un file di dati 20
4.5 Struttura di un dataset 22
4.6 Valori mancanti nei dati 26
4.7 Dimensione massima dei dataset 26
4.8 Raccolte di file di dati 27

5 Modifica del campione **30**

5.1 Introduzione 30
5.2 Impostazione del campione 30
5.3 Restrizione del campione 31
5.4 Campionamento casuale 32
5.5 I comandi del menù Campione 32

6 Grafici e diagrammi **33**

6.1 Grafici gnuplot . . . 33
6.2 Boxplot . . . 34

7 Funzioni speciali in genr 36
7.1 Introduzione . . . 36
7.2 Varianza di lungo periodo . . . 36
7.3 Filtri per serie storiche . . . 36
7.4 Dati panel . . . 38
7.5 Ricampionamento e bootstrapping . . . 39
7.6 Densità cumulate e p-value . . . 40
7.7 Gestione dei valori mancanti . . . 41
7.8 Recupero di variabili interne . . . 42
7.9 Procedure numeriche . . . 43
7.10 La trasformata discreta di Fourier . . . 44

8 Variabili discrete 48
8.1 Dichiarazione delle variabili discrete . . . 48
8.2 Comandi per le variabili discrete . . . 49

9 Costrutti loop 53
9.1 Introduzione . . . 53
9.2 Varianti di controllo del loop . . . 53
9.3 La modalità progressiva . . . 55
9.4 Esempi di loop . . . 56

10 Funzioni definite dall'utente 60
10.1 Definizione di una funzione . . . 60
10.2 Chiamata di una funzione . . . 61
10.3 Cancellazione di una funzione . . . 62
10.4 Programmazione delle funzioni . . . 62
10.5 Pacchetti di funzioni . . . 67

11 Liste e stringhe definite dall'utente 71
11.1 Liste definite dall'utente . . . 71
11.2 Stringhe definite dall'utente . . . 73

12 Operazioni con le matrici 77
12.1 Creazione di matrici . . . 77
12.2 Selezione di sotto-matrici . . . 78
12.3 Operatori matriciali . . . 79
12.4 Operatori matrice-scalare . . . 81
12.5 Funzioni matriciali . . . 81
12.6 Matrici vuote . . . 90
12.7 Matrici accessorie . . . 90

12.8 Conflitti tra nomi . . . 91
12.9 Creazione di una serie di dati da una matrice . . . 91
12.10 Matrici e liste . . . 92
12.11 Eliminazione di matrici . . . 92
12.12 Stampa di matrici . . . 92
12.13 Esempio: OLS usando le matrici . . . 92

13 Esempi svolti **95**
13.1 Gestione dei dataset . . . 95
13.2 Creazione e modifica di variabili . . . 96
13.3 Trucchi utili . . . 97

II Metodi econometrici **99**

14 Stima robusta della matrice di covarianza **100**
14.1 Introduzione . . . 100
14.2 Dati cross-section e HCCME . . . 101
14.3 Serie storiche e matrici di covarianza HAC . . . 102
14.4 Problemi speciali con dati panel . . . 106

15 Dati panel **108**
15.1 Stima di modelli panel . . . 108
15.2 Modelli panel dinamici . . . 112
15.3 Esempio: la Penn World Table . . . 113

16 Minimi quadrati non lineari **115**
16.1 Introduzione ed esempi . . . 115
16.2 Inizializzazione dei parametri . . . 115
16.3 Finestra di dialogo NLS . . . 116
16.4 Derivate analitiche e numeriche . . . 116
16.5 Arresto della procedura . . . 117
16.6 Dettagli sul codice . . . 117
16.7 Accuratezza numerica . . . 118

17 Stima di massima verosimiglianza **120**
17.1 Stima di massima verosimiglianza con gretl . . . 120
17.2 Stima di una Gamma . . . 122
17.3 Funzioni di costo con frontiera stocastica . . . 123
17.4 Modelli GARCH . . . 124
17.5 Derivate analitiche . . . 126
17.6 Debug del codice mle . . . 128

18 Stima GMM **129**
18.1 Introduzione e terminologia . . . 129

18.2 Minimi quadrati ordinari (OLS) come GMM . . . 130
18.3 Minimi quadrati a due stadi (TSLS) come GMM . . . 132
18.4 Opzioni per la matrice di covarianza . . . 132
18.5 Un esempio reale: il Consumption Based Asset Pricing Model . . . 134
18.6 Avvertenze . . . 137

19 Criteri di selezione dei modelli 138
19.1 Introduzione . . . 138
19.2 Criteri di informazione . . . 138

20 Modelli per serie storiche 140
20.1 Introduzione . . . 140
20.2 Modelli ARIMA . . . 140
20.3 Test per radici unitarie . . . 145
20.4 ARCH e GARCH . . . 148

21 Cointegrazione e modelli vettoriali a correzione d'errore 151
21.1 Introduzione . . . 151
21.2 Modelli vettoriali a correzione di errore (VECM) come rappresentazione di un sistema cointegrato . . . 152
21.3 Interpretazione delle componenti deterministiche . . . 153
21.4 I test di cointegrazione di Johansen . . . 155
21.5 Identificazione dei vettori di cointegrazione . . . 156
21.6 Restrizioni sovra-identificanti . . . 158
21.7 Metodi di soluzione numerica . . . 164

22 Variabili dipendenti discrete e censurate 167
22.1 Modelli logit e probit . . . 167
22.2 Il modello Tobit . . . 170

III Dettagli tecnici 174

23 Gretl e TEX 175
23.1 Introduzione . . . 175
23.2 I comandi TEXnei menù . . . 176
23.3 Personalizzare l'output di TEX . . . 177
23.4 Codifica dei caratteri . . . 179
23.5 Installazione e utilizzo di TEX . . . 180

24 Risoluzione dei problemi 181
24.1 Segnalazione dei bug . . . 181
24.2 Programmi ausiliari . . . 181

25 L'interfaccia a riga di comando 182

25.1 Gretl sul terminale 182
25.2 La modalità non interattiva 182

IV Appendici **183**

A Dettagli sui file di dati **184**
A.1 Formato interno di gretl 184
A.2 Formato tradizionale di ESL 184
A.3 Dettagli sui database binari 185

B Compilare gretl **187**
B.1 Requisiti 187
B.2 Istruzioni per la compilazione 188

C Accuratezza numerica **191**

D Altro software libero utile **192**

E Elenco degli URL **193**

Capitolo 1

Introduzione

1.1 Caratteristiche principali

Gretl è un pacchetto econometrico che comprende una libreria condivisa, un programma client a riga di comando e un'interfaccia grafica.

Amichevole Gretl offre un'interfaccia utente intuitiva, che permette di entrare subito nel vivo dell'analisi econometrica. Grazie all'integrazione con i libri di testo di Ramu Ramanathan, di Jeffrey Wooldridge, di James Stock e Mark Watson, il pacchetto offre molti file di dati e script di comandi, commentati e pronti all'uso. Gli utenti di gretl hanno inoltre a disposizione la documentazione sul programma e la mailing list gretl-users.

Flessibile È possibile scegliere il proprio metodo di lavoro preferito: dal punta-e-clicca interattivo alla modalità batch, oppure una combinazione dei due approcci.

Multi-piattaforma La piattaforma di sviluppo di Gretl è Linux, ma il programma è disponibile anche per MS Windows e Mac OS X, e dovrebbe funzionare su qualsiasi sistema operativo simile a UNIX che comprenda le librerie di base richieste (si veda l'appendice B).

Open source L'intero codice sorgente di Gretl è disponibile per chiunque voglia criticarlo, correggerlo o estenderlo. Si veda l'appendice B.

Sofisticato Gretl offre un'ampia varietà di stimatori basati sui minimi quadrati, sia per singole equazioni che per sistemi di equazioni, compresa l'autoregressione vettoriale e i modelli a correzione di errore. Comprende anche vari stimatori di massima verosimiglianza (ad es. probit, ARIMA, GARCH), mentre altri metodi avanzati sono implementabili dall'utente attraverso la stima generica di massima verosimiglianza o la stima GMM non lineare.

Estensibile Gli utenti possono estendere gretl scrivendo le proprie funzioni e procedure in un linguaggio di scripting che include una gamma abbastanza ampia di funzioni matriciali.

Accurato Gretl è stato testato a fondo con vari dataset di riferimento, tra cui quelli del NIST. Si veda l'appendice C.

Pronto per internet Gretl può scaricare i database da un server alla Wake Forest University; inoltre, comprende una funzionalità di aggiornamento che controlla se è disponibile una nuova versione del programma e, nella versione per MS Windows, permette di aggiornarlo automaticamente.

Internazionale Gretl supporta le lingue inglese, francese, italiana, spagnola, polacca, portoghese, tedesca, o basca, a seconda della lingua impostata sul computer.

1.2 Ringraziamenti

La base di codice di Gretl deriva da quella del programma ESL ("Econometrics Software Library"), scritto dal Professor Ramu Ramanathan della University of California, San Diego. Siamo molto grati al Professor Ramanathan per aver reso disponibile questo codice con licenza GNU General Public License e per aver aiutato nello sviluppo iniziale di Gretl.

Siamo anche grati agli autori di molti testi di econometria che hanno permesso la distribuzione delle versioni Gretl dei dataset contenuti nei loro libri. Questa lista al momento comprende William Greene, autore di *Econometric Analysis*, Jeffrey Wooldridge (*Introductory Econometrics: A Modern Approach*); James Stock e Mark Watson (*Introduction to Econometrics*); Damodar

Gujarati (*Basic Econometrics*); Russell Davidson e James MacKinnon (*Econometric Theory and Methods*); Marno Verbeek (*A Guide to Modern Econometrics*).

La stima GARCH in Gretl si basa sul codice pubblicato sul *Journal of Applied Econometrics* dai Prof. Fiorentini, Calzolari e Panattoni, mentre il codice per generare i *p*-value per i test Dickey Fuller è di James MacKinnon. In ognuno dei casi sono grato agli autori per avermi permesso di usare il loro lavoro.

Per quanto riguarda l'internazionalizzazione di Gretl, vorrei ringraziare Ignacio Díaz-Emparanza (spagnolo), Michel Robitaille e Florent Bresson (francese), Cristian Rigamonti (italiano), Tadeusz Kufel e Pawel Kufel (polacco), Markus Hahn e Sven Schreiber (tedesco), Hélio Guilherme (Portoghese) e Susan Orbe (Basco).

Gretl ha beneficiato largamente del lavoro di molti sviluppatori di software libero e open-source: per i dettagli si veda l'appendice B. Devo ringraziare Richard Stallman della Free Software Foundation per il suo supporto al software libero in generale, ma in particolare per aver accettato di "adottare" Gretl come programma GNU.

Molti utenti di Gretl hanno fornito utili suggerimenti e segnalazioni di errori. Un ringraziamento particolare a Ignacio Díaz-Emparanza, Tadeusz Kufel, Pawel Kufel, Alan Isaac, Cristian Rigamonti, Sven Schreiber, Talha Yalta, Andreas Rosenblad e Dirk Eddelbuettel, che cura il pacchetto Gretl per Debian GNU/Linux.

1.3 Installazione del programma

Linux

Sulla piattaforma Linux[1], è possibile compilare da sé il codice di Gretl, oppure usare un pacchetto pre-compilato.

Se si vuole utilizzare la versione di sviluppo di gretl, o modificare il programma per le proprie esigenze, occorre compilare dai sorgenti; visto che questa operazione richiede una certa abilità, la maggior parte degli utenti preferirà usare un pacchetto pre-compilato.

Alcune distribuzioni Linux contengono già gretl, ad esempio Debian o Ubuntu (nell'archivio *universe*). Chi usa queste distribuzioni deve solo installare gretl usando il proprio gestore di pacchetti preferito (ad es. synaptic).

Pacchetti binari in formato rpm (utilizzabili su sistemi Red Hat Linux e simili) sono disponibili sul sito di gretl: `http://gretl.sourceforge.net`.

In ogni caso, speriamo che gli utenti che possiedono conoscenze di programmazione trovino gretl abbastanza interessante da migliorare ed estendere. La documentazione dell'API di libgretl non è ancora completa, ma è possibile trovare alcuni dettagli seguendo il link "Libgretl API docs" sul sito web di gretl. Chi è interessato allo sviluppo del programma è invitato a iscriversi alla mailing list gretl-devel.

Per chi vuole compilare gretl da sé, le istruzioni si trovano nell'Appendice B.

MS Windows

La versione MS Windows è disponibile sotto forma di file eseguibile auto-estraente. Per installarlo, occorre scaricare `gretl_install.exe` ed eseguire questo programma. Verrà chiesta una posizione in cui installare il pacchetto.

Aggiornamento

Se si ha un computer connesso a internet, all'avvio Gretl può collegarsi al proprio sito web alla Wake Forest University per vedere se sono disponibili aggiornamenti al programma. In caso positivo, comparirà una finestra informativa. Per attivare questa funzionalità, occorre abilitare

[1] In questo manuale verrà usata l'abbreviazione "Linux" per riferirsi al sistema operativo GNU/Linux. Ciò che viene detto a proposito di Linux vale anche per altri sistemi simili a UNIX, anche se potrebbero essere necessari alcuni adattamenti.

la casella “Avvisa in caso di aggiornamenti di gretl” nel menù “Strumenti, Preferenze, Generali...” di Gretl.

La versione MS Windows di Gretl fa un passo in più: dà anche la possibilità di aggiornare automaticamente il programma. È sufficiente seguire le indicazioni nella finestra pop-up: chiudere Gretl ed eseguire il programma di aggiornamento “gretl updater” (che di solito si trova vicino alla voce Gretl nel gruppo Programmi del menù Avvio di Windows). Quando il programma di aggiornamento ha concluso il suo funzionamento, è possibile avviare di nuovo Gretl.

Parte I

Uso del programma

Capitolo 2

Iniziare

2.1 Eseguire una regressione

Questa introduzione è dedicata prevalentemente alla versione grafica del programma; si veda il capitolo 25 e la *Guida ai comandi di gretl* per i dettagli sulla versione a riga di comando del programma, gretlcli.

È possibile fornire il nome di un file di dati da aprire come argomento a gretl, ma per il momento non facciamolo: avviamo semplicemente il programma[1]. Apparirà la finestra principale (che di solito mostra le informazioni sul dataset, ma che ora è vuota) e vari menù, alcuni dei quali disabilitati.

Cosa si può fare a questo punto? Si possono sfogliare i file di dati (o i database) forniti, aprire un file di dati, crearne uno nuovo, leggere l'aiuto in linea, o aprire un file di comandi. Per ora, sfogliamo i file di dati forniti: dal menù "File", scegliamo "Apri dati, File di esempio, Ramanathan. . . ". Si dovrebbe aprire una seconda finestra, che presenta un elenco dei file di dati forniti con il pacchetto (si veda la Figura 2.1). La numerazione dei file corrisponde all'organizzazione dei capitoli di Ramanathan (2002), che descrive l'analisi di questi dati, ma i dati sono utilizzabili per fare pratica anche senza avere il testo.

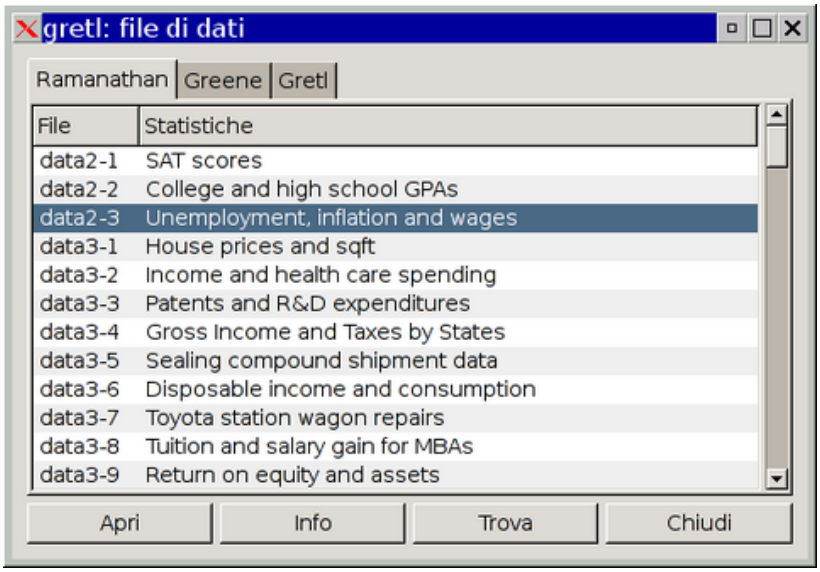


Figura 2.1: Finestra dei file di esempio

Selezionando una riga in questa finestra e facendo clic su "Info", si aprirà una finestra di descrizione del dataset in questione (che può contenere informazioni a proposito della fonte dei dati e della definizione delle variabili). Se si trova un file interessante, è possibile aprirlo facendo clic su "Apri", o semplicemente facendo doppio clic sul nome del file. Per il momento, apriamo `data3-6`.

[1] Per comodità, in questo manuale chiamerò semplicemente gretl il client grafico del programma; si noti comunque che il nome specifico del programma è differente a seconda della piattaforma: su Linux si chiama `gretl_x11`, mentre su MS Windows è `gretlw32.exe`. Sui sistemi Linux viene installato anche uno script chiamato `gretl`, si veda anche la *Guida ai comandi di gretl*.

☞ Nelle finestre di gretl che contengono liste, facendo doppio clic su una riga viene eseguita l'azione predefinita per la relativa voce nella lista: ad esempio mostrare i valori di una serie, o aprire un file.

Questo file contiene dati relativi a un oggetto classico dell'econometria, la funzione di consumo. La finestra dei dati dovrebbe ora contenere il nome del file di dati in uso, l'intervallo completo dei dati e quello del campione, i nomi delle variabili, insieme a delle loro brevi descrizioni (si veda la Figura 2.2).

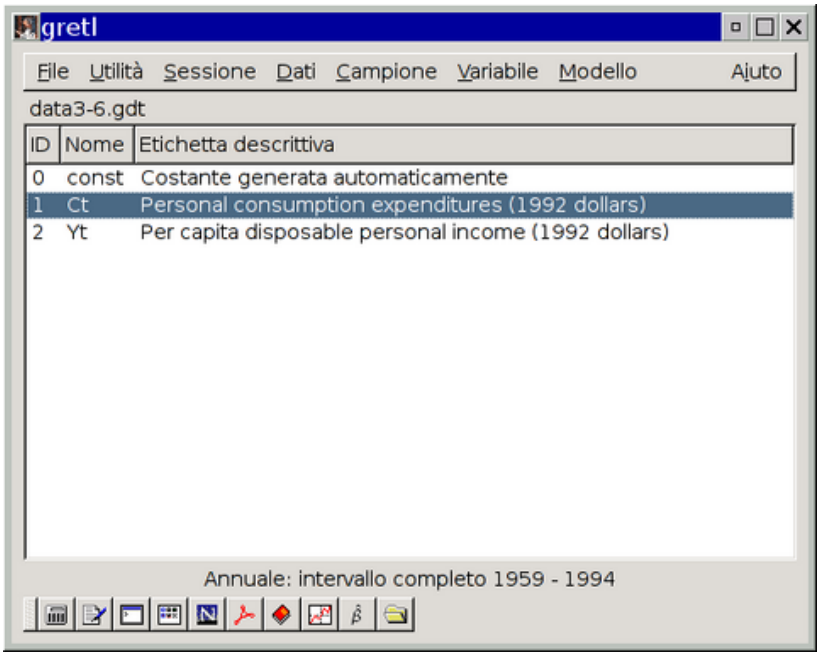


Figura 2.2: Finestra principale, con un file di esempio aperto

OK, cosa possiamo fare ora? Le varie opzioni dei menù dovrebbero essere abbastanza chiare: per ora ci concentreremo sul menù Modello, ma una panoramica di tutti i menù della finestra principale è fornita più avanti (si veda la sezione 2.3).

Il menù Modello di gretl offre varie routine di stima econometrica: quella più semplice e nota è rappresentata dai minimi quadrati ordinari (Ordinary Least Squares - OLS). Scegliendo OLS, si apre una finestra di dialogo che richiede una *specificazione del modello*; si veda la Figura 2.3.

Per selezionare la variabile dipendente, fare clic su una variabile nella lista di sinistra e premere il pulsante "Scegli" con la freccia che punta verso il riquadro della variabile dipendente. Selezionando la casella "Imposta come predefinito", la variabile scelta verrà sempre pre-selezionata come variabile dipendente durante le prossime aperture della finestra di dialogo. Trucco: facendo doppio clic su una variabile sulla sinistra, viene selezionata come variabile dipendente e impostata come scelta predefinita. Per selezionare le variabili indipendenti, fare clic su di esse nella lista di sinistra e premere il pulsante "Aggiungi" (o fare clic col pulsante destro del mouse). È possibile selezionare più variabili contigue trascinando il mouse; se le variabili da selezionare non sono contigue, occorre fare clic tenendo premuto il tasto `Ctrl`. Per eseguire una regressione con il consumo come variabile dipendente e il reddito come variabile indipendente, fare clic su `Ct` nel riquadro della variabile dipendente e aggiungere `Yt` alla lista delle variabili indipendenti.

2.2 Risultati della stima

Una volta specificato un modello, apparirà una finestra che mostra i risultati della regressione, in un formato sufficientemente chiaro e standard (Figura 2.4).

La finestra dei risultati contiene dei menù che consentono di ispezionare o mostrare graficamente i residui e i valori stimati, e di eseguire vari test diagnostici sul modello.

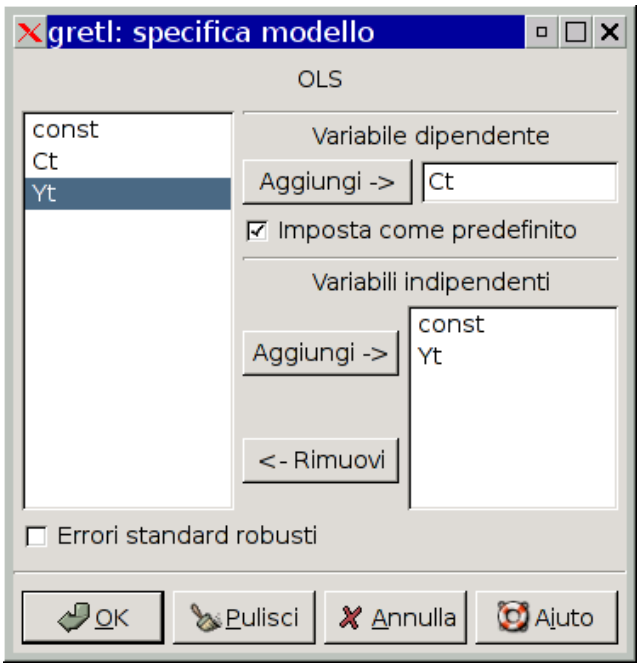


Figura 2.3: Specificazione del modello

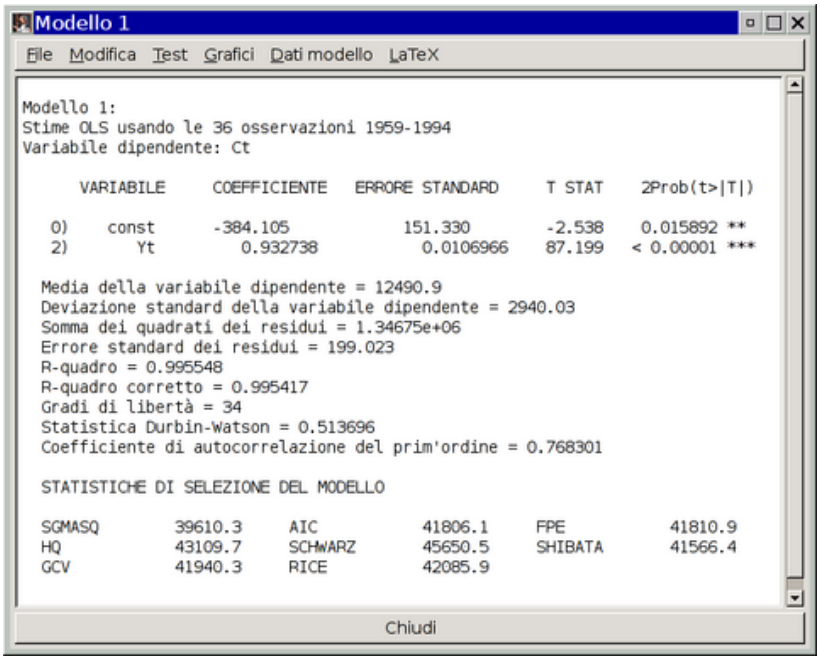


Figura 2.4: Finestra dei risultati del modello

Per la maggior parte dei modelli c'è anche un'opzione per stampare il risultato della regressione in formato LaTeX. Si veda il capitolo 23 per i dettagli.

Per importare i risultati di gretl in un word processor, è possibile fare copia e incolla da una finestra dei risultati usando il menù Modifica (o il pulsante Copia, in alcuni contesti) nel programma di arrivo. Molte finestre di gretl (non tutte) offrono l'opzione di copiare in formato RTF (il "Rich Text Format" di Microsoft) o come LaTeX. Se si deve incollare in un word processor, RTF può essere una buona opzione, visto che il formato tabulare dei risultati viene preservato[2]. In alternativa, è possibile salvare i risultati come file di testo semplice e importare successivamente il file nel programma di elaborazione testi: quando si conclude una sessione di gretl si ha l'opportunità di salvare tutti i risultati della sessione in un unico file.

Si noti che nel desktop gnome e in MS Windows, il menù File contiene un comando per inviare i risultati direttamente a una stampante.

☞ Quando si incollano o si importano dei risultati di gretl sotto forma di testo semplice in un word processor, conviene selezionare un carattere a spaziatura fissa, in stile macchina da scrivere (ad es. il Courier), per preservare il formato tabulare dei risultati. Selezionare un carattere non troppo grande (Courier da 10 punti dovrebbe andare bene) eviterà che le righe dei risultati vengano spezzate nei punti sbagliati.

2.3 I menù della finestra principale

Sulla barra dei menù della finestra principale si trovano, nell'ordine da sinistra a destra, i menù File, Strumenti, Dati, Visualizza, Aggiungi, Campione, Variabile, Modello e Aiuto.

- Menù file
  - Apri dati: apre un file di dati in formato interno di gretl o lo importa da altri formati. Si veda il capitolo 4.
  - Aggiungi dati: aggiunge dati al dataset in uso, da un file di dati di gretl, un file con dati separati da virgole, o un foglio elettronico.
  - Salva dati: salva il file di dati gretl in uso.
  - Salva dati come: salva il dataset in uso in formato interno, con la possibilità di usare la compressione gzip. Si veda il capitolo 4.
  - Esporta dati: salva il dataset in uso in formato CSV (valori separati da virgole), o nei formati di GNU R o GNU Octave. Si veda il capitolo 4 e anche l'appendice D.
  - Invia a: invia il dataset come allegato in un'e-mail.
  - Nuovo dataset: permette di creare un dataset vuoto, in cui è possibile immettere dati manualmente o importando delle serie da un database. Si veda oltre per i dettagli sui database.
  - Abbandona dataset: cancella dalla memoria il dataset in uso. Di solito questa operazione non è necessaria (visto che aprendo un nuovo file di dati, quello in uso viene sostituito), ma ci sono casi in cui è utile.
  - Comandi: uno "script" è un file che contiene una sequenza di comandi di gretl. Questo menù contiene voci che permettono di aprire uno script di comandi creato in precedenza ("File utente"), uno script di esempio tra quelli forniti, o una finestra di editor per creare un nuovo script.

[2] Si noti che quando si copia come RTF in MS Windows, Windows permetterà di incollare il materiale solo in applicazioni che "comprendono" l'RTF. Quindi, sarà possibile incollare in MS Word, ma non nel Blocco Note. Inoltre sembra esserci un bug in alcune versioni di Windows, per cui l'operazione di copia non funziona se l'applicazione "di arrivo" (ad es. MS Word) non è stata avviata prima di copiare il materiale in questione.

  - Sessioni: un file di sessione contiene una fotografia di una precedente sessione di lavoro con gretl, che comprende il dataset usato e tutti i modelli e i grafici che sono stati salvati. Questo menù permette di aprire una sessione salvata in precedenza o di salvare la sessione in corso.
  - Database: permette di consultare alcuni ampi database di serie storiche, disponibili sul proprio computer o, se si è connessi a internet, sul server dei database di gretl. Per i dettagli, si veda la sezione 4.3.
  - Funzioni: gestisce i "pacchetti di funzioni" (si veda la sezione 10.5), che permettono di accedere a funzioni scritte da altri utenti e di distribuire le proprie.
  - Esci: abbandona il programma. Verrà proposto di salvare il lavoro svolto.

- Menù strumenti

  - Tavole statistiche: cerca i valori critici per alcune distribuzioni di uso comune (normale o Gaussiana, *t*, chi-quadro, *F* e Durbin–Watson).
  - Calcola p-value: calcola i p-value per le distribuzioni Gaussiana, *t*, chi-quadro, *F*, gamma, binomiale o Poisson. Si veda anche il comando `pvalue` nella *Guida ai comandi di gretl*.
  - Grafici distribuzione: produce grafici di varie distribuzioni di probabilità. Il menù pop-up della finestra grafica che viene creata contiene il comando "Aggiungi un'altra curva", che permette di mostrare più curve sullo stesso grafico (ad esempio è possibile disegnare la distribuzione *t* con vari gradi di libertà).
  - Calcola test: calcola le statistiche test e i p-value per una serie di test di ipotesi di uso comune (media della popolazione, varianza e proporzione, differenza delle medie o delle varianze e proporzioni).
  - Test non parametrici: calcola statistiche per vari test non parametrici (test dei segni, test di Wilcoxon, test delle successioni).
  - Seme per numeri casuali: imposta il seme per il generatore di numeri casuali (l'impostazione predefinita si basa sull'ora di sistema in cui il programma è stato avviato).
  - Visualizza log comandi: apre una finestra che contiene il registro dei comandi eseguiti fino a questo momento.
  - Terminale di Gretl: apre una finestra di "terminale" in cui è possibile digitare dei comandi, come se si stesse usando la versione a riga di comando gretlcli, invece di quella con interfaccia grafica.
  - Avvia Gnu R: avvia R (se è presente sul sistema) e vi carica una copia del dataset in uso in gretl. Si veda l'appendice D.
  - Ordina variabili: riordina l'elenco delle variabili nella finestra principale, secondo il numero identificativo o alfabeticamente.
  - Test NIST: controlla l'accuratezza numerica di gretl usando i test di riferimento per la regressione lineare adottati dal National Institute of Standards and Technology statunitense.
  - Preferenze: imposta i percorsi per vari file a cui gretl ha bisogno di accedere, sceglie il carattere usato per mostrare i risultati, attiva o disattiva l'avviso in caso di nuove versioni disponibili del programma, configura la barra degli strumenti, e molte altre opzioni. Per ulteriori dettagli, si veda la *Guida ai comandi di gretl*.

- Menù dati

  - Seleziona tutto: seleziona tutte le variabili; molti comandi dei menù hanno effetto sulle variabili selezionate nella finestra principale.
  - Mostra valori: apre una finestra con un elenco (non modificabile) dei valori delle variabili (tutte o un sottoinsieme di esse).
  - Modifica valori: apre una finestra di foglio elettronico, con cui è possibile modificare valori, aggiungere nuove variabili, o estendere il numero delle osservazioni.

  - Aggiungi osservazioni: mostra una finestra di dialogo in cui è possibile scegliere un numero di osservazioni da aggiungere alla fine del dataset attuale; da usare per le previsioni.
  - Rimuovi osservazioni in più: attivo solo se sono state aggiunte automaticamente delle osservazioni durante la procedura di previsione; cancella queste osservazioni aggiuntive.
  - Visualizza descrizione, Modifica descrizione: "Visualizza descrizione" mostra le informazioni disponibili per il file di dati in uso; "Modifica descrizione" permette di modificarle (se si ha il permesso di farlo).
  - Visualizza informazioni complete: apre una finestra con una descrizione completa del dataset in uso, che include le informazioni di riepilogo e quelle specifiche di ogni variabile.
  - Aggiungi marcatori: richiede di specificare un file di testo che contiene "marcatori per le osservazioni" (brevi stringhe che identificano singole osservazioni) e aggiunge queste informazioni al dataset. Si veda il capitolo 4.
  - Rimuovi marcatori: attivo solo se il dataset contiene marcatori per le osservazioni; rimuove questi marcatori.
  - Struttura dataset: permette di modificare l'interpretazione strutturale del dataset in uso. Ad esempio, se i dati sono stati importati come cross section, è possibile fare in modo che il programma li interpreti come serie storiche o come panel. Si veda anche la sezione 4.5.
  - Compatta dati: per serie storiche con frequenza superiore a quella annuale, permette di diminuire la frequenza dei dati, usando uno dei quattro metodi di compattamento disponibili (media, somma, inizio del periodo, fine del periodo).
  - Espandi dati: per serie storiche, permette di aumentare la frequenza dei dati.
  - Trasponi dati: trasforma ogni osservazione in una variabile e viceversa (o, in altre parole, ogni riga della matrice dei dati diventa una colonna della nuova matrice dei dati); può essere utile per "raddrizzare" dati importati in modo errato.

- Menù visualizza
  - Finestra icone: apre una finestra che mostra la sessione corrente di gretl sotto forma di un insieme di icone. Per i dettagli si veda la sezione 3.4.
  - Grafico: apre una finestra di dialogo che permette di scegliere tra un grafico temporale, un grafico a dispersione X–Y, un grafico X–Y a impulsi (barre verticali), un grafico X–Y "con fattore" (ossia, con i punti colorati in modo diverso a seconda del valore di una data variabile dummy), un boxplot e un grafico 3D. Si veda il capitolo 6 per i dettagli.
  - Grafici multipli: permette di comporre un insieme di grafici (al massimo sei) in un'unica finestra. I grafici possono essere a dispersione o di serie storiche.
  - Statistiche descrittive: mostra un insieme abbastanza ricco di statistiche descrittive per tutte le variabili del dataset, o per le variabili selezionate.
  - Matrice di correlazione: mostra i coefficienti di correlazione fra le variabili selezionate.
  - Tabulazione incrociata: mostra una tabulazione incrociata fra le variabili selezionate. Occorre che almeno due variabili del dataset siano state marcate come discrete (si veda il Capitolo 8).
  - Componenti principali: produce un'analisi delle componenti principali delle variabili selezionate.
  - Distanze di Mahalonobis: calcola la distanza di Mahalonobis per ogni osservazione dal centroide dell'insieme di variabili selezionate.
  - Correlogramma incrociato: calcola e mostra il correlogramma incrociato per due variabili selezionate.

- Menù aggiungi: offre alcune trasformazioni standard per le variabili (logaritmi, ritardi, quadrati, ecc) che è possibile aggiungere al dataset. Dà anche l'opzione di aggiungere variabili casuali e (per i dataset di serie storiche) variabili dummy stagionali (ad es. variabili dummy trimestrali per dati trimestrali).
- Menù campione
  - Imposta intervallo: seleziona punti di partenza e arrivo diversi per il campione in uso, all'interno dell'intervallo di dati disponibili.
  - Ripristina campione completo: si spiega da sé.
  - Imposta in base a dummy: data una variabile dummy (indicatore) con valori 0 o 1, vengono scartate dal campione tutte le osservazioni per cui la variabile dummy vale 0.
  - Imposta in base a condizione: simile al precedente, tranne per il fatto che non si ha bisogno di una variabile predefinita: basta fornire una condizione Booleana (ad es. `sqft > 1400`) e il campione sarà ristretto alle osservazioni che soddisfano la condizione. Si veda la voce `genr` nella *Guida ai comandi di gretl* per maggiori dettagli sugli operatori Booleani che possono essere usati.
  - Sotto-campione casuale: estrae un campione casuale dal dataset.
  - Scarta valori mancanti: scarta dal campione corrente tutte le osservazioni per cui almeno una variabile ha un valore mancante (si veda la sezione 4.6).
  - Conta valori mancanti: produce un rapporto sulle osservazioni per cui mancano dei valori. Può essere utile durante l'esame di un dataset panel, dove è abbastanza comune incontrare valori mancanti.
  - Imposta codice valori mancanti: imposta il valore numerico che sarà interpretato come "mancante" o "non disponibile". Questo comando è utile se si stanno utilizzando dati importati, per cui gretl non ha riconosciuto il codice per i valori mancanti.
- Menù variabile: la maggior parte di questi comandi opera su una sola variabile alla volta. La variabile "attiva" viene impostata facendo clic sulla riga che la contiene nella finestra principale. La maggior parte delle opzioni si spiegano da sole. Si noti che è possibile rinominare una variabile e modificare la sua etichetta descrittiva usando "Modifica attributi". È anche possibile definire una nuova variabile attraverso una formula (che può essere una funzione di una o più variabili esistenti). Per la sintassi di queste formule, si veda la sezione "Definisci nuova variabile" della guida in linea o la voce `genr`. Un semplice esempio:

  ```
  pippo = x1 * x2
  ```

  creerà la nuova variabile `pippo` come prodotto delle variabili esistenti `x1` e `x2`. In queste formule, le variabili devono essere indicate per nome, non per numero identificativo.
- Menù modello: per i dettagli sui vari stimatori offerti da questo menù, si consulti la *Guida ai comandi di gretl*. Si veda anche il capitolo 16 a proposito della stima di modelli non lineari.
- Menù aiuto: usatelo! Fornisce dettagli sulla sintassi dei comandi e delle finestre di dialogo.

2.4 Scorciatoie da tastiera

Mentre si lavora nella finestra principale di gretl, è possibile compiere alcune operazioni comuni utilizzando la tastiera, come mostrato nella tabella seguente:

`Invio`	Apre una finestra contenente i valori delle variabili selezionate, ossia, esegue il comando “Dati, Mostra valori”.
`Canc`	Cancella le variabili selezionate. Per evitare cancellazioni accidentali è richiesta una conferma.
`e`	Ha lo stesso effetto del comando “Modifica attributi” del menù “Variabile”.
`F2`	Ha lo stesso significato di “e”, per compatibilità con altri programmi.
`g`	Ha lo stesso effetto del comando “Definisci nuova variabile” dal menù “Variabile” (che richiama il comando `genr`).
`h`	Apre la finestra di aiuto per i comandi di gretl.
`F1`	Ha lo stesso significato di “h”, per compatibilità con altri programmi.
`r`	Aggiorna l’elenco delle variabili nella finestra principale: ha lo stesso effetto del comando “Aggiorna finestra” nel menu “Dati”.
`t`	Mostra in un grafico la variabile selezionata; per i dataset di tipo serie storiche viene mostrato un grafico temporale, mentre per i dati di tipo cross section si ottiene un grafico di distribuzione di frequenza.

2.5 La barra degli strumenti di gretl

In basso a sinistra nella finestra principale si trova la barra degli strumenti.

Le icone sulla barra hanno il seguente significato, nell’ordine:

1. Avvia una calcolatrice. Una funzione comoda quando si ha bisogno di usare velocemente una calcolatrice mentre si lavora in gretl. Il programma avviato in modo predefinito è `calc.exe` in MS Windows, o `xcalc` nel sistema X window. È possibile cambiare il programma nel menù “Strumenti, Preferenze, Generali”, sezione “Programmi”.
2. Inizia un nuovo script. Apre una finestra in cui è possibile digitare una serie di comandi da eseguire in modalità batch.
3. Apre il terminale di gretl. Una scorciatoia per il comando del menù “Terminale di Gretl” (si veda la sezione 2.3).
4. Apre la finestra delle icone di gretl.
5. Apre il sito web di gretl nel proprio browser (funziona solo se si è connessi a internet e si dispone di un browser).
6. Apre l’ultima versione del manuale di gretl in formato PDF. Richiede una connessione a internet e un browser configurato.
7. Apre questa guida in formato PDF.
8. Apre la guida in linea per la sintassi dei comandi (che mostra i dettagli di tutti i comandi disponibili).
9. Apre la finestra di dialogo per costruire un grafico.
10. Apre la finestra di dialogo per stimare un modello con i minimi quadrati ordinari.
11. Apre una finestra che elenca i dataset distribuiti insieme a gretl e altri dataset eventualmente installati.

Se non si desidera visualizzare la barra degli strumenti, è possibile disabilitarla nel menù “Strumenti, Preferenze, Generali”, de-selezionando la casella “Mostra la barra degli strumenti di gretl”.

Capitolo 3

Modalità di lavoro

3.1 Script di comandi

I comandi gretl eseguiti utilizzando le finestre di dialogo dell'interfaccia grafica vengono registrati sotto forma di file "script" o "log". Questi file possono essere modificati e ri-eseguiti, usando gretl o l'applicazione a riga di comando gretlcli.

Per visualizzare lo stato del log dei comandi durante una sessione di gretl, basta scegliere "Visualizza log comandi" dal menù "Strumenti". Questo file di log è chiamato `session.inp` e viene sovrascritto ogni volta che si inizia una nuova sessione: per conservarlo, basta salvarlo con un nome diverso. I file di comandi vengono visualizzati più facilmente nella finestra di selezione dei file se vengono salvati con l'estensione "`.inp`".

Per aprire uno script di diversa provenienza, occorre usare il comando del menù "File, Comandi, File utente"; per creare uno script da zero, occorre usare "File, Comandi, Nuovo" dal menù, oppure il pulsante "Nuovo file comandi" dalla barra degli strumenti. In entrambi i casi si aprirà una finestra comandi (si veda la figura 3.1).

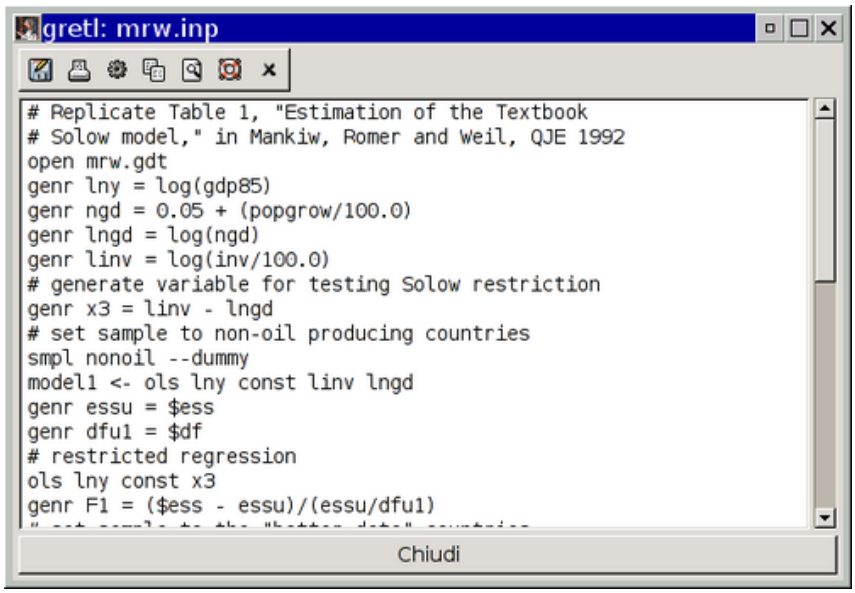


Figura 3.1: Finestra comandi, modifica di un file di comandi

La barra degli strumenti in cima alla finestra comandi offre le seguenti funzioni (da sinistra a destra): (1) Salva il file; (2) Salva il file con un nome specifico; (3) Stampa il file (questo comando non è disponibile su tutte le piattaforme); (4) Esegui i comandi nel file; (5) Copia il testo selezionato; (6) Incolla il testo selezionato; (7) Cerca e sostituisci testo; (8) Annulla l'ultimo comando Incolla o Sostituisci; (9) Aiuto (spostando il cursore sulla parola di un comando e premendo il punto di domanda si ottiene aiuto su quel comando); (10) Chiudi la finestra.

Quando si esegue lo script, facendo clic sull'icona "Esegui" o premendo Ctrl-r, i risultati dei comandi compariranno in un'unica finestra, dove possono essere modificati, salvati, o copiati negli appunti. Per conoscere meglio le possibilità fornite dagli script, è possibile usare il comando del menù "Aiuto, Guida comandi", oppure eseguire la versione a riga di comando del programma gretlcli ed eseguire il comando "help", oppure ancora consultare la *Guida ai comandi di gretl*.

Se si esegue lo script dopo averne selezionato una parte, gretl eseguirà solo quella parte. In più, se si vuole eseguire solo la riga corrente basta premere Ctrl-Enter[1].

Facendo clic col tasto destro del mouse nella finestra di modifica dello script si apre un menù pop-up, che permette di eseguire la riga su cui si trova il cursore, oppure una regione dello script selezionata in precedenza. Se lo script è modificabile, questo menù offre anche la possibilità di aggiungere o rimuovere i marcatori di commento dall'inizio della riga, o delle righe.

Il pacchetto gretl contiene più di 70 script "di esempio": la maggior parte di essi è relativa a Ramanathan (2002), ma essi possono essere utili anche per studiare le possibilità di scripting offerte da gretl, oltre ad alcuni aspetti della teoria econometrica. È possibile esplorare i file di esempio dal menù "File, Comandi, File di esempio": si troverà un elenco dei file, insieme a una breve descrizione dei problemi illustrati nello script e dei dati utilizzati. Basta aprire un file ed eseguirlo per vederne i risultati. Si noti che in uno script i comandi lunghi possono essere suddivisi in due o più righe, usando una barra inversa come carattere di continuazione della riga.

È anche possibile, se si vuole, usare insieme l'interfaccia grafica e le funzionalità di script, sfruttando le comodità offerte da ognuno dei due approcci. Ecco alcuni suggerimenti:

- Aprire un file di dati dall'interfaccia grafica, esplorare i dati, generare grafici, stimare regressioni, eseguire test. Quindi aprire il log dei comandi, rimuovere eventuali comandi ridondanti, salvarlo con un nome specifico ed eseguirlo, in modo da generare un singolo file che contiene i risultati della propria sessione di lavoro.
- Partire da un nuovo file script e scrivere i comandi necessari per eseguire le trasformazioni desiderate su un dataset (si veda il comando genr nella *Guida ai comandi di gretl*). Tipicamente è possibile svolgere questo tipo di operazioni in modo più efficace scrivendo una sequenza ben ragionata di comandi, piuttosto che puntando e cliccando nell'interfaccia grafica. Salvare ed eseguire lo script: la finestra dei dati verrà aggiornata e sarà possibile continuare l'esplorazione dei dati attraverso l'interfaccia grafica. Per ripristinare lo stato iniziale dei dati in un secondo momento, è sufficiente aprire ed eseguire di nuovo lo script "preparatorio".

Script e file di dati

Un modo comune di condurre la ricerca econometrica con gretl è il seguente: comporre uno script, eseguirlo, ispezionare i risultati, modificare lo script, eseguirlo di nuovo, ripetendo gli ultimi tre passi per quanto è necessario. In questo contesto, si noti che quando viene aperto un file di dati la maggior parte delle informazioni di stato interne a gretl vengono cancellate. È quindi una buona idea cominciare il proprio script con un comando open: in questo modo il file di dati verrà riaperto ogni volta, e si sarà sicuri di ottenere risultati "freschi", non condizionati dalle esecuzioni precedenti.

Occorre notare un punto ulteriore: quando si apre un nuovo file di dati usando l'interfaccia grafica si viene avvertiti del fatto che aprire un nuovo file comporta la perdita del lavoro non salvato fino a quel momento. Quando invece si esegue uno script che apre un file di dati *non* si viene avvertiti. Questo comportamento assume che non ci sia alcun rischio di perdere lavoro fatto, visto che il lavoro è incorporato nello script stesso (e sarebbe scomodo ricevere un avvertimento ad ogni iterazione del ciclo di lavoro descritto sopra).

Ciò significa che occorre fare attenzione se si è eseguito del lavoro usando l'interfaccia grafica e si vuole poi eseguire uno script: il file di dati in uso può venir sostituito senza alcun avviso, ed è responsabilità dell'utente salvare i propri dati prima di eseguire lo script.

[1] Questa utile funzionalità, offerta anche da altri pacchetti econometrici, può indurre a scrivere degli script enormi che non sono mai eseguiti interamente, ma fungono solo da archivio per una serie di piccole porzioni di codice che vengono eseguite all'occorrenza. Visto che gretl permette di tenere aperte molte finestre di script nello stesso momento, è possibile mantenere i propri script ordinati in piccoli file separati.

3.2 Salvare oggetti da uno script

Se si stima un modello usando l'interfaccia grafica, i risultati vengono mostrati in una finestra separata, che comprende alcuni menù da cui è possibile effettuare test, disegnare grafici, salvare dati del modello, e così via. Se invece si stima un modello usando uno script, si ottiene un tabulato non interattivo dei risultati, ma è possibile "catturare" i modelli stimati in uno script, in modo da esaminarli interattivamente dopo l'esecuzione dello script. Ecco un esempio:

```
Modello1 <- ols Ct 0 Yt
```

Ossia: si indica un nome con cui verrà salvato il modello, seguito da una "freccia di assegnazione" rivolta all'indietro e dal comando di stima del modello. È possibile usare spazi nei nomi dei modelli, ma occorre racchiudere il nome tra virgolette doppie:

```
"Modello 1" <- ols Ct 0 Yt
```

I modelli salvati in questo modo appariranno come icone nella finestra delle icone di gretl (si veda la sezione 3.4) dopo l'esecuzione dello script. Inoltre, è possibile fare in modo che un modello venga mostrato in una finestra in modo automatico, usando:

```
Modello1.show
```

Ancora: se il nome contiene spazi, occorre metterlo tra virgolette:

```
"Modello 1".show
```

È possibile usare la stessa procedura anche per i grafici; ad esempio, il comando seguente crea un grafico di `Ct` rispetto a `Yt`, lo salva come "Grafico" (apparirà con questo nome nella finestra delle icone) e lo mostra automaticamente:

```
Grafico <- gnuplot Ct Yt
Grafico.show
```

È anche possibile salvare i risultati di un comando come oggetti testuali identificati da un nome (anche questi appariranno nella finestra della sessione, da cui sarà possibile aprirli in seguito). Ad esempio, questo comando invia i risultati di un test Dickey–Fuller aumentato a un "oggetto testuale" chiamato `ADF1` e li mostra in una finestra:

```
ADF1 <- adf 2 x1
ADF1.show
```

Gli oggetti salvati in questo modo (siano essi modelli, grafici o parti di testo) possono essere eliminati usando il comando `.free` aggiunto al nome dell'oggetto, ad esempio `ADF1.free`.

3.3 Il terminale di gretl

Un'altra funzionalità comoda è contenuta nel menù "Strumenti" di gretl: il "Terminale di gretl" (c'è anche un pulsante "Terminale di gretl" nella barra degli strumenti nella finestra principale). Si tratta di una finestra in cui è possibile scrivere comandi ed eseguirli interattivamente uno alla volta (premendo il tasto Invio), così come avviene nella versione a riga di comando gretlcli, con la differenza che l'interfaccia grafica viene aggiornata in base ai comandi eseguiti dal terminale, permettendo di lavorare con entrambi gli strumenti.

Nel terminale è disponibile la "storia dei comandi", ossia è possibile usare i tasti freccia su e freccia giù per scorrere la lista dei comandi già eseguiti. È possibile quindi recuperare un comando, modificarlo ed eseguirlo di nuovo.

In modalità terminale, è possibile creare, visualizzare e cancellare oggetti (modelli, grafici o testo) nel modo descritto sopra per la modalità script.

3.4 Il concetto di sessione

gretl offre il concetto di "sessione" per tenere traccia del proprio lavoro e richiamarlo in un secondo momento. L'idea di base è quella di fornire uno spazio che contiene, sotto forma di icone, vari oggetti relativi alla sessione di lavoro in corso (si veda la figura 3.2). È possibile aggiungere oggetti in questo spazio e salvarli assieme alla sessione, in modo che siano disponibili ad una successiva riapertura della sessione.

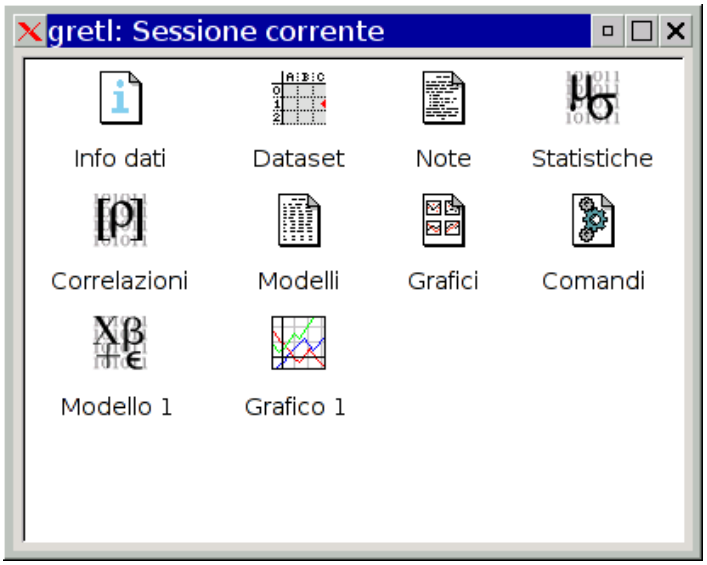


Figura 3.2: Finestra delle icone: oltre alle icone predefinite, sono stati aggiunti un modello e un grafico

Avviando gretl, aprendo un dataset e selezionando "Finestra icone" dal menù "Visualizza", è possibile visualizzare l'insieme predefinito di icone, che permettono di accedere rapidamente al log dei comandi ("Comandi"), alle informazioni sul dataset (se esistono), alla matrice di correlazione ("Correlazioni") e alle statistiche descrittive di riepilogo ("Statistiche"). Tutte queste funzioni sono attivate facendo doppio clic sull'icona relativa. L'icona "Dataset" è un po' più complessa: un doppio clic apre i dati nel foglio di lavoro integrato, mentre facendo clic col tasto destro del mouse si ottiene un menù con le altre azioni possibili.

Per aggiungere un modello alla finestra delle icone, occorre per prima cosa stimarlo usando il menù "Modello", quindi aprire il menù "File" nella finestra del modello e selezionare "Salva alla sessione come icona..." o "Salva come icona e chiudi". L'ultima operazione può essere eseguita semplicemente anche premendo il tasto S da dentro la finestra del modello.

Per aggiungere un grafico, occorre crearlo (dal menù "Visualizza, Grafico", o attraverso uno degli altri comandi gretl di generazione dei grafici). Facendo clic sulla finestra del grafico si ottiene un menù da cui si dovrà selezionare "Salva alla sessione come icona".

Una volta che un modello o un grafico è stato aggiunto, la sua icona dovrebbe comparire nella finestra delle icone. Facendo doppio clic sull'icona, l'oggetto viene visualizzato di nuovo, mentre facendo clic con il tasto destro del mouse si ottiene un menù che permette di visualizzare o cancellare l'oggetto, oppure di modificarlo, se si tratta di un grafico.

La tabella modelli

Nella ricerca econometrica è prassi comune stimare, per una stessa variabile dipendente, vari modelli che differiscono tra loro per le variabili indipendenti o per lo stimatore usato. In questa situazione è comodo poter rappresentare i risultati delle regressioni sotto forma di una tabella dove ogni colonna contiene i risultati (stime dei coefficienti e errori standard) per un dato modello e ogni riga contiene le stime per una certa variabile nei differenti modelli.

Nella finestra delle icone, gretl dà la possibilità di costruire una tabella simile (e di esportarla in testo semplice, LaTeX o RTF - Rich Text Format). Ecco come fare:[2]

1. Stimare un modello che si vuole includere nella tabella e selezionare, nel menù File della finestra di visualizzazione del modello, "Salva alla sessione come icona" o "Salva come icona e chiudi".
2. Ripetere il punto 1 per gli altri modelli da includere nella tabella (fino a un massimo di sei modelli).
3. Completata la stima dei modelli, aprire l'icona della sessione di gretl, selezionando "Finestra icone" nel menù "Visualizza" della finestra principale di gretl, o facendo clic sull'icona "Finestra icone" della barra degli strumenti di gretl.
4. La finestra delle icone contiene un'icona chiamata "Tabella Modelli". Per aggiungere alla tabella modelli il modello che deve apparire nella colonna più a sinistra della tabella, basta trascinare l'icona del modello sull'icona della Tabella Modelli, oppure fare clic col tasto destro sull'icona del modello e selezionare "Aggiungi alla tabella modelli" dal menù pop-up.
5. Ripetere il punto 4 per gli altri modelli da aggiungere alla tabella. Il secondo modello scelto apparirà nella seconda colonna da sinistra della tabella, e così via.
6. Ultimata la composizione della tabella, è possibile visualizzarla facendo doppio clic sulla sua icona. Per copiare la tabella negli appunti in uno dei formati supportati, basta fare clic sul menù Modifica della finestra in cui appare la tabella.
7. Se l'ordinamento dei modelli nella tabella non è quello voluto, fare clic col tasto destro sull'icona della tabella modelli e selezionare "Pulisci", quindi tornare al punto 4.

Un semplice esempio di tabella modelli di gretl è mostrato nella figura 3.3.

La pagina dei grafici

L'icona "Grafici" della finestra delle icone offre la possibilità di riunire vari grafici da stampare su una sola pagina, se si è installato il sistema di composizione LaTeX e si è in grado di generare e visualizzare file in formato PDF o postscript[3].

Nella finestra delle icone, è possibile trascinare fino a otto grafici sull'icona della pagina dei grafici. Facendo doppio clic sull'icona della pagina dei grafici (o facendo clic col tasto destro e selezionando "Mostra"), una pagina contenente i grafici selezionati (in formato EPS o PDF) verrà composta e aperta con il proprio visualizzatore, da cui sarà possibile stamparla.

Per pulire la pagina dei grafici, fare clic col tasto destro sull'icona e selezionare "Pulisci".

Su sistemi diversi da MS Windows, può essere necessario modificare l'impostazione del programma per visualizzare il postscript, attraverso la sezione "Programmi" della finestra di dialogo delle "Preferenze" di gretl (nel menù "Strumenti" della finestra principale). Su Windows può essere necessario dover impostare le regole di associazione dei file in modo che sia usato il visualizzatore adeguato per l'azione "Apri" sui file con estensione `.ps`.

Salvare e riaprire sessioni

Se si creano modelli o grafici che si pensa di poter riutilizzare in seguito, è utile selezionare "File, Sessione, Salva come..." prima di uscire da gretl. Per riaprire la sessione in seguito, è possibile:

[2] La tabella modelli può anche essere costruita in modo non interattivo in uno script. Per i dettagli si veda il comando `modeltab`.

[3] Per l'output in PDF occorre avere il lettore Acrobat Reader di Adobe, o xpdf, se si usa il sistema X11. Per il postscript, occorrono dvips e ghostscript, insieme a un visualizzatore come gv, ggv o kghostview. Il visualizzatore predefinito per sistemi diversi da MS Windows è gv.

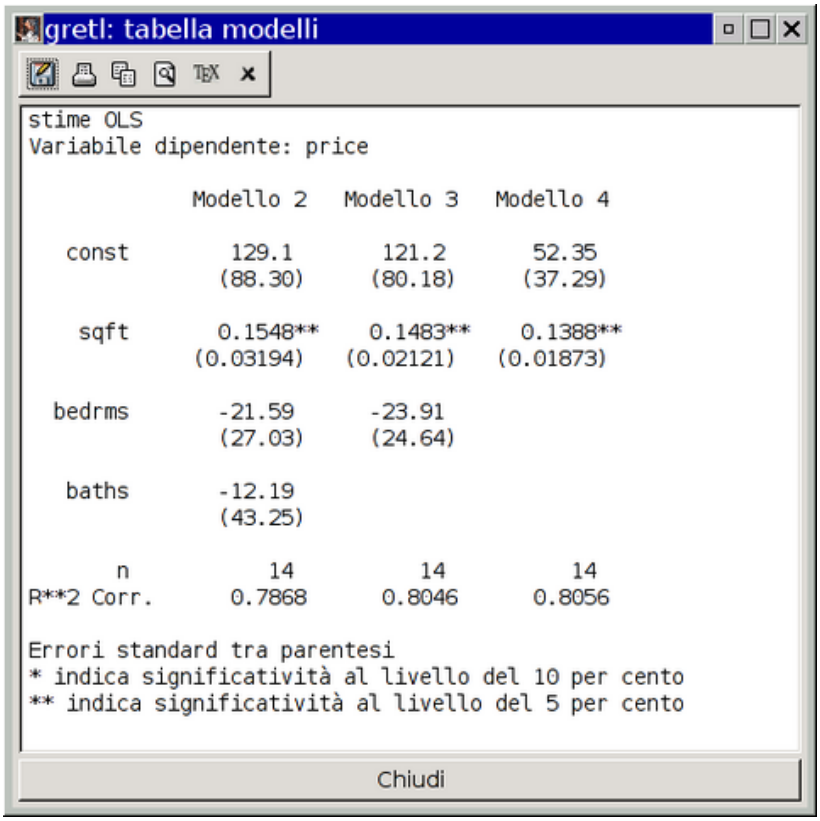


Figura 3.3: Esempio della tabella modelli

- Avviare gretl e riaprire il file della sessione usando il comando "File, Sessione, Apri", oppure
- Dalla riga di comando, scrivere `gretl -r` *file-sessione*, dove *file-sessione* è il nome del file in cui è stata salvata la sessione.

Capitolo 4

File di dati

4.1 Formato interno

gretl utilizza un suo formato interno per i file di dati. La maggior parte degli utenti probabilmente non è interessata a leggere o scrivere questi file con altri programmi, ma in alcune occasioni potrebbe essere utile farlo: per ulteriori dettagli si veda l'appendice A.

4.2 Altri formati dei file di dati

gretl legge anche file di dati in altri formati:

- File di testo semplice (ASCII). Possono essere importati in gretl usando il comando "File, Apri dati, Importa ASCII..." dell'interfaccia grafica o il comando `import` dell'interfaccia a riga di comando. Per ulteriori dettagli su questo tipo di file, si veda la sezione 4.4.
- File con valori separati da virgole (CSV). Possono essere importati in gretl usando il comando "File, Apri dati, Importa CSV..." dell'interfaccia grafica o il comando `import` dell'interfaccia a riga di comando. Si veda anche la sezione 4.4.
- Fogli di calcolo: MS Excel, Gnumeric e Open Document (ODS). Possono essere importati in gretl con il comando "File, Apri dati, Importa". La sezione 4.4 descrive i requisiti per questo tipo di file.
- File di dati di Stata (`.dta`).
- File di lavoro di Eviews (`.wf1`).[1]
- File di dati di JMulTi.

Quando vengono importati file in formato ASCII o CSV, gretl apre una finestra "diagnostica", che informa sullo stato della lettura dei dati. Se dovessero verificarsi dei problemi a causa di dati malformattati, questa finestra mostrerà dei suggerimenti per risolverli.

Per venire incontro a chi vuole eseguire analisi più sofisticate, gretl offre la possibilità di salvare i dati nei formati usati dai programmi GNU R, Octave, JMulTi e PcGive (si veda l'appendice D). Nell'interfaccia grafica, questa opzione si trova nel menù "File, Esporta dati", mentre nel client a riga di comando occorre usare il comando `store` con l'opzione appropriata.

4.3 Database binari

Per lavorare con grandi quantità di dati, gretl include una routine per operare su database binari. Un *database*, al contrario di un *file di dati*, non viene letto direttamente nello spazio di lavoro del programma, ma può contenere serie con frequenze e intervalli del campione diversi. È possibile aprire un database, selezionare delle serie e importarle nel dataset corrente; le serie potranno poi essere salvate in un file di dati. È possibile accedere ai database attraverso il comando "File, Database".

Per i dettagli sul formato dei database di gretl, si veda l'appendice A.

[1] Si veda http://www.ecn.wfu.edu/eviews_format/.

Accesso ai database online

Dalla versione 0.40, gretl è in grado di accedere ai database via internet. Alla Wake Forest University sono disponibili alcuni database, a cui è possibile accedere se il proprio computer è connesso a internet. Si veda la descrizione del comando "data" nel menù "Aiuto" di gretl.

☞ Per dettagli e aggiornamenti sui dati disponibili, basta visitare la pagina dei dati di gretl.

Formati di database esterni

Grazie a Thomas Doan di *Estima*, che ha reso disponibili le specifiche del formato di database usato da RATS 4 (Regression Analysis of Time Series), gretl è in grado di gestire anche alcuni tipi di database RATS 4: per la precisione quelli che contengono dati mensili o trimestrali.

Gretl può anche importare dati dai database PcGive. Questi sono costituiti da coppie di file, uno dei quali (con l'estensione `.bn7`) contiene i dati veri e propri, mentre quello con estensione (`.in7`) contiene informazioni supplementari.

4.4 Creazione di un file di dati

Ci sono vari modi per compiere questa operazione.

1. Acquisire, o creare con un editor di testo, un file di testo semplice ed aprirlo con il comando "Importa ASCII" di gretl.
2. Usare il proprio foglio di lavoro per inserire i dati, salvarlo in formato con valori separati da virgole (Comma Separated Values) se necessario (non dovrebbe essere necessario se il programma di foglio elettronico è MS Excel, Gnumeric o OpenOffice) e infine usare uno dei comandi "Importa" di gretl.
3. Usare il foglio elettronico contenuto in gretl.
4. Selezionare le serie di dati da un database.
5. Usare un editor di testo o altri programmi per creare un file di dati nel formato interno di gretl.

Seguono alcune note a proposito dei vari metodi presentati.

Note comuni sui dati importati

Le opzioni 1 e 2 richiedono di usare il comando "import" di gretl. Affinché i dati vengano letti correttamente, occorre che siano soddisfatte alcune condizioni:

- La prima riga deve contenere nomi di variabile validi, ossia lunghi al massimo 15 caratteri (i nomi di variabile più lunghi verranno troncati a 15 caratteri), che iniziano con una lettera e sono composti solo da caratteri alfanumerici e dal carattere trattino basso, _. Precisazioni per i file ASCII o CSV: se il file non contiene righe con i nomi delle variabili, il programma userà automaticamente i nomi `v1`, `v2` e così via. Inoltre, per "prima riga" si intende la prima riga *significativa*: nel caso dei file ASCII e CSV, le righe bianche e quelle che iniziano con un carattere cancelletto, #, vengono ignorate. Nel caso dell'importazione di file Excel e Gnumeric, viene presentata una finestra di dialogo in cui è possibile indicare il numero di righe e/o di colonne del foglio di lavoro da ignorare.
- I valori dei dati devono costituire un blocco rettangolare, con una variabile per colonna e un'osservazione per riga. Il numero delle varibili (colonne dei dati) deve corrispondere al numero dei nomi di variabile specificati. Si veda anche la sezione 4.6. Il programma si aspetta dati di tipo numerico, ma nel caso di importazione da file ASCII/CSV, c'è un supporto limitato per dati di tipo carattere (stringa): se una colonna contiene solo dati di tipo stringa, le stringhe sono sostituite da codici numerici progressivi, e quando l'importazione si conclude, viene mostrata una tabella di corrispondenza tra stringhe e codici.

- Date (o marcatori per le osservazioni): opzionalmente, la *prima* colonna può contenere stringhe, come date o etichette identificative per osservazioni su dati cross-section. Queste stringhe possono essere lunghe al massimo 8 caratteri (come avviene per le variabili, i nomi più lunghi verranno troncati), mentre la colonna che le ospita dovrebbe avere come nome `obs` o `date`, oppure nessun nome.

  Affinché una stringa sia riconosciuta come data, deve rispettare uno dei formati seguenti: per le serie *annuali*, l'anno deve essere indicato con quattro cifre; per le serie *trimestrali* occorre indicare l'anno con quattro cifre, seguito da un separatore (punto, due punti, o la lettera `Q`) e da una cifra che indica il trimestre, ad esempio: `1997.1`, `2002:3`, `1947Q1`; per le serie *mensili* occorre indicare l'anno con quattro cifre, seguito dal punto o dai due punti, e da due cifre che indicano il mese, ad esempio: `1997.01`, `2002:10`.

I file CSV possono usare virgole, spazi o tab come separatori fra le colonne: il separatore da usare può essere selezionato subito dopo aver eseguito il comando "Importa CSV". Se invece si usa "Importa ASCII" il programma cerca di riconoscere automaticamente il separatore usato nei dati.

Se si usa un foglio elettronico per preparare i dati, è possibile applicare varie trasformazioni ai dati "grezzi" (sommare variabili, calcolare percentuali, ecc.), ma queste elaborazioni possono essere compiute, forse più facilmente, anche in gretl, usando gli strumenti disponibili nel menù "Aggiungi".

Importare dati e aggiungerli

Può essere necessario costruire un dataset di gretl a poco a poco, importando successivamente i dati da varie fonti. Questa funzionalità è fornita dai comandi del menù "File, Aggiungi dati". gretl controllerà che i nuovi dati siano compatibili con il dataset esistente e in caso positivo aggiungerà i nuovi dati. In questo modo è possibile aggiungere nuove variabili, a patto che la frequenza dei dati corrisponda a quella del dataset esistente. È anche possibile aggiungere nuove osservazioni per le serie di dati presenti nel dataset; in questo caso i nomi delle variabili devono corrispondere esattamente. Attenzione: se invece di "Aggiungi dati" si sceglie "Apri dati", il dataset corrente verrà chiuso.

Usare il foglio elettronico interno

È possibile creare un dataset con il comando "File, Nuovo dataset", scegliendo il tipo di dati (ad es. serie storiche trimestrali, dati cross-section), la data iniziale e quella finale (o il numero di osservazioni), e il nome della prima variabile da creare nel dataset. Dopo aver effettuato queste scelte, viene presentato un semplice foglio elettronico in cui è possibile iniziare a inserire i valori. Facendo clic col tasto destro nella finestra del foglio elettronico, comparirà un menù che permette di aggiungere una nuova variabile (colonna), di aggiungere una nuova osservazione (aggiungere una riga in fondo al foglio), o di inserire un'osservazione nel punto indicato (i dati sottostanti saranno spostati in basso e verrà inserita una riga vuota).

Dopo aver inserito i dati nel foglio elettronico, è possibile importarli nel foglio di lavoro di gretl premendo il pulsante "Applica le modifiche" nella finestra del foglio elettronico.

Si noti che il foglio elettronico di gretl è molto semplice e non permette di inserire funzioni o formule: per trasformare i dati è possibile usare i comandi disponibili nei menù "Aggiungi" o "Variabile" nella finestra principale di gretl.

Estrarre dati da un database

Un modo alternativo di creare un dataset consiste nel selezionare le variabili da un database.

Selezionando il comando "File, Database", vengono presentate quattro alternative: "Gretl", "RATS 4", "PcGive" e "Sul server di gretl". Selezionando "Gretl", si troverà il file `fedstl.bin`, che contiene un'ampia raccolta di serie macroeconomiche USA ed è distribuito insieme al programma.

Non si troverà nulla sotto "RATS 4" a meno di non aver acquistato dei dati RATS[2]. Se si possiedono dati RATS, occorre usare il comando "Strumenti, Preferenze, Generali...", selezionare la finestra Database e inserire il percorso completo dei propri file RATS.

Se il proprio computer è connesso a internet è possibile accedere a vari database presenti alla Wake Forest University scegliendo "Sul server di gretl". È possibile consultare questi database da remoto, oppure installarli sul proprio computer. La finestra dei database ha una colonna che mostra, per ogni file, lo stato di installazione e lo stato di aggiornamento della copia locale rispetto alla versione disponibile alla Wake Forest.

Dopo aver aperto un database è anche possibile importare singole serie nello spazio di lavoro di gretl usando il comando "Serie, Importa" nella finestra del database, o nel menù che compare facendo clic col tasto destro, oppure trascinando la serie nella finestra principale del programma.

Creare un file di dati nei formati interni di gretl

Se si hanno già molti dati archiviati in formato elettronico, l'approccio migliore può essere quello di creare un file di dati in uno dei formati interni di gretl, usando un editor di testo o altri programmi come awk, sed o perl. Ovviamente occorrerà studiare i formati di dati di gretl (il formato XML o quello "tradizionale") descritti nell'appendice A.

4.5 Struttura di un dataset

Una volta che i dati sono stati letti da gretl, può essere necessario dover fornire alcune informazioni sulla natura dei dati stessi. Distinguiamo tre tipi di dataset:

1. Cross-section
2. Serie storiche
3. Dati panel

Lo strumento principale per eseguire questa operazione è il comando del menù "Dati, Struttura Dataset" nell'interfaccia grafica, o il comando `setobs` negli script e nell'interfaccia a riga di comando.

Dati cross-section

Per "dati cross-section" intendiamo osservazioni su una serie di "unità" (che possono essere imprese, paesi, individui, ecc.) realizzate nello stesso periodo temporale. Questa è l'interpretazione predefinita per un file di dati: se gretl non ha informazioni sufficienti per interpretare i dati come serie storiche o panel, essi sono automaticamente interpretati come cross-section. Nell'improbabile caso in cui dei dati cross-section siano interpretati come serie storiche, è possibile correggere l'errore usando il comando del menù "Dati, Struttura dataset", facendo clic sul pulsante "cross-section", quindi su "Avanti", e infine su "Ok".

Serie storiche

Quando si importano dati da un foglio elettronico o da un file di testo, gretl cerca di estrarre tutte le informazioni temporali dalla prima colonna dei dati. Se tuttavia la struttura di serie storiche dei dati non è riconosciuta, è possibile usare il comando "Dati, Struttura dataset", selezionare "Serie storiche", e successivamente selezionare la frequenza dei dati e l'osservazione iniziale. In ogni momento è possibile fare clic su "Indietro" per correggere le scelte fatte.

È opportuna qualche considerazione ulteriore a proposito della frequenza dei dati. In un dataset di serie storiche, tutte le serie devono avere la stessa frequenza; se occorre creare un dataset combinando serie di diversa frequenza, ad esempio mensili e trimestrali, occorre procedere nel modo seguente.

[2] Si veda www.estima.com

Per prima cosa occorre formulare una strategia: si desidera creare un dataset mensile o trimestrale? Un punto da tenere in considerazione consiste nel fatto che "compattare" i dati da una frequenza più alta (es. mensile) a una più bassa (es. trimestrale) di solito non presenta problemi. Ovviamente si perde informazione, ma in generale è accettabile, ad esempio, prendere la media di tre osservazioni mensili per creare un'osservazione trimestrale. D'altra parte, "espandere" i dati da una frequenza minore a una maggiore, in generale non è un'operazione valida.

Nella maggior parte dei casi, la strategia migliore consiste nel creare un dataset di frequenza *inferiore*, e di compattare i dati a frequenza maggiore. Quando si importano i dati a frequenza maggiore nel dataset, il programma propone la scelta del metodo di compattamento (media, somma, valore all'inizio del periodo, o alla fine del periodo). Nella maggior parte dei casi, prendere la media dei dati è una scelta appropriata.

È anche possibile importare dati di minore frequenza in un dataset a frequenza maggiore, ma non è una scelta raccomandabile in generale. In questi casi, gretl replica i valori della serie a frequenza minore per quante volte è richiesto dalla nuova frequenza. Ad esempio, ipotizzando di avere una serie trimestrale che vale 35.5 in 1990:1, il primo trimestre del 1990. Dopo l'espansione alla frequenza mensile, il valore 35.5 verrà assegnato alle osservazioni per i mesi di gennaio, febbraio e marzo del 1990. La variabile espansa non sarà quindi adatta per analisi temporali "fini", a meno che non si abbia buona ragione di ipotizzare che il suo valore rimanga costante nei sotto-periodi.

Una volta scelta la frequenza di un dataset, gretl offre comunque la possibilità di compattare o espandere tutte le serie del dataset, usando i comandi "Compatta dati" ed "Espandi dati" del menù "Dati", che ovviamente vanno eseguiti con cautela.

Dati panel

I dati panel possono essere visti sotto tre dimensioni, ossia le variabili, le unità cross-section e i periodi temporali. Ad esempio, un particolare valore in un dataset può essere identificato come l'osservazione della capitalizzazione di una certa azienda nel 1980. Una nota terminologica: useremo i termini "unità cross section", "unità" e "gruppo" in modo intercambiabile per riferirci alle entità che compongono la dimensione cross section del panel, che potrebbero essere, ad esempio, aziende, paesi o individui.

Per rappresentare i dati in un file testuale (e anche per poterli manipolare), queste tre dimensioni devono in qualche modo essere riportate a due. Questa procedura di "appiattimento" richiede di prendere degli "strati" di dati che apparterrebbero alla terza dimensione e di impilarli nella dimensione verticale del file.

Gretl si aspetta sempre di trovare dati organizzati "per osservazione", ossia in modo che ogni riga rappresenti un'osservazione (e che ogni variabile occupi esattamente una colonna). Alla luce di questo fatto, l'appiattimento dei dati panel può essere realizzato in due modi:

- Pila di dati cross section: ognuno dei blocchi di dati disposti verticalmente contiene i valori per tutte le unità cross-section in un determinato periodo.
- Pila di serie storiche: ognuno dei blocchi di dati disposti verticalmente contiene serie storiche per una determinata unità cross-section.

È possibile usare entrambi i metodi per inserire i dati. Internamente, gretl usa il formato "pila di serie storiche" per immagazzinare i dati.

Quando si importano dati panel in gretl da un foglio di calcolo o da un file con valori separati da virgole, la struttura panel non verrà riconosciuta automaticamente (molto probabilmente i dati verranno trattati come "non datati"). Per imporre un'interpretazione panel ai dati, è possibile usare l'interfaccia grafica o il comando `setobs`.

Nell'interfaccia grafica, occorre usare il comando dal menù "Campione, Struttura dataset". Nella prima finestra di dialogo occorre selezionare "Panel"; in quella successiva, si hanno tre scelte. Le prime due opzioni, "Pila di serie storiche" e "Pila di dati cross section" sono utilizzabili se il dataset è già organizzato in uno di questi due modi. Selezionando una di queste due opzioni, il passo successivo è quello di indicare il numero di unità cross section nel dataset. La terza

opzione "Usa variabili indice", è utilizzabile se il dataset contiene due variabili che indicizzano le unità e i periodi temporali; il passo successivo prevede di indicare queste due variabili. Ad esempio, un dataset potrebbe contenere una variabile con il codice dei paesi osservati e una variabile che rappresenta l'anno dell'osservazione. In questo caso, gretl riconoscerà la struttura panel dei dati a prescindere dal modo in cui le osservazioni sono organizzate.

Il comando testuale `setobs` supporta delle opzioni che corrispondono a quelle viste sopra nell'interfaccia grafica. Se sono disponibili delle variabili indice, è possibile procedere nel modo seguente:

```
setobs var-unita var-tempo --panel-vars
```

dove `var-unita` è una variabile che indicizza le unità e `var-tempo` è una variabile che indicizza i periodi. Altrimenti, è possibile usare la sintassi `setobs` *freq* `1:1` *struttura*, dove *freq* indica la "dimensione dei blocchi" di dati (ossia, il numero di periodi nel caso delle pile di serie storiche, o il numero di unità cross section nel caso di pila di dati cross section), mentre *struttura* può essere uguale a `--stacked-time-series` o `--stacked-cross-section`. Di seguito vengono mostrati due esempi: il primo per un dataset panel sotto forma di pila di serie storiche con osservazioni per 20 periodi, il secondo per un dataset panel sotto forma di pila di dati cross section, con 5 unità cross section.

```
setobs 20 1:1 --stacked-time-series
setobs 5 1:1 --stacked-cross-section
```

Dati panel organizzati per variabile

Talvolta i dati panel disponibili pubblicamente sono organizzati "per variabile". Si supponga di avere dati per due variabili, `x1` e `x2`, relativi a 50 stati per 5 anni (per un totale di 250 osservazioni per variabile). Una possibile rappresentazione testuale dei dati potrebbe iniziare con un blocco per `x1`, con 50 righe, corrispondenti agli stati e 5 colonne, corrispondenti agli anni. Seguirebbe, sotto, un blocco con una struttura simile, relativo alla variabile `x2`. Viene mostrato di seguito un frammento di questo file di dati, con osservazioni quinquennali per il periodo 1965–1985; occorre immaginare che la tabella continui per altri 48 stati, seguita da altre 50 righe per la variabile `x2`.

x1					
	1965	1970	1975	1980	1985
AR	100.0	110.5	118.7	131.2	160.4
AZ	100.0	104.3	113.8	120.9	140.6

Se un tale file di dati viene importato in gretl[3], il programma interpreterà le colonne come variabili diverse, rendendo inutilizzabili i dati. Esiste però un meccanismo per gestire queste situazioni, ossia la funzione `stack` del comando `genr`.

Si consideri la prima colonna di dati nel frammento visto sopra: le prime 50 righe di questa colonna costituiscono una cross-section per la variabile `x1` nell'anno 1965. Se potessimo creare una nuova variabile sistemando le prime 50 voci nella seconda colonna direttamente sotto le prime 50 voci della prima colonna, staremmo costruendo un dataset disposto "per osservazione" (nel primo dei due sensi definiti in precedenza: una pila di dati cross-section). Ossia, avremmo una colonna che contiene una cross-section per `x1` nel 1965, seguita da una cross-section per la stessa variabile nel 1970.

Il seguente script di gretl illustra come possiamo effettuare l'operazione, per `x1` e `x2`. Assumiamo che il file di dati originale si chiami `panel.txt` e che le colonne al suo interno siano precedute da intestazioni con i "nomi variabile" `p1`, `p2`, ..., `p5` (le colonne non sono vere variabili, ma per il momento "facciamo finta" che lo siano).

[3] Si noti che occorrerà fare alcune piccole modifiche al file affinché possa essere letto: bisognerà rimuovere la riga che contiene il nome della variabile (in questo esempio `x1`) e la riga iniziale che contiene gli anni, altrimenti essi verranno importati come valori numerici.

```
open panel.txt
genr x1 = stack(p1..p5) --length=50
genr x2 = stack(p1..p5) --offset=50 --length=50
setobs 50 1.01 --stacked-cross-section
store panel.gdt x1 x2
```

La seconda riga illustra la sintassi della funzione `stack`. Il doppio punto nella parentesi indica un intervallo di variabili da impilare: vogliamo impilare tutte le 5 colonne (per tutti i 5 anni). Il dataset completo contiene 100 righe: per sistemare la variabile `x1` vogliamo leggere solo le prime 50 righe di ogni colonna: facciamo questo aggiungendo l'opzione `--length=50`. Si noti che se occorre impilare un insieme di colonne non contigue, è possibile usare un elenco separato da virgole all'interno della parentesi, come in

```
genr x = stack(p1,p3,p5)
```

Nella riga 3 creiamo una pila di dati per la variabile `x2`. Ancora, vogliamo una lunghezza (`length`) di 50 per i componenti della serie impilata, ma questa volta vogliamo che gretl inizi a leggere dalla cinquantesima riga dei dati originali, quindi specifichiamo `--offset=50`. La riga 4 impone un'interpretazione panel sui dati; infine, salviamo i dati in formato gretl, con un'interpretazione panel, eliminando le "variabili" originali da `p1` a `p5`.

Lo script di esempio visto sopra è appropriato quando il numero delle variabili da processare è piccolo. Quando ci sono molte variabili nel dataset, è più efficiente usare un comando loop per costruire le nuove variabili, come mostrato nell'esempio seguente, che ipotizza una situazione uguale a quella precedente (50 unità, 5 periodi) ma con 20 variabili invece che 2.

```
open panel.txt
loop for i=1..20
  genr k = ($i - 1) * 50
  genr x$i = stack(p1..p5) --offset=k --length=50
endloop
setobs 50 1.01 --stacked-cross-section
store panel.gdt x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 \
  x11 x12 x13 x14 x15 x16 x17 x18 x19 x20
```

Marcatori nei dati panel

Quando si lavora con dati panel, può essere utile usare dei marcatori di facile memorizzazione per identificare le osservazioni. Per questo scopo esiste una funzione speciale da usare con il comando `genr`.

Nell'esempio precedente, si supponga che tutti gli stati siano identificati con codici di due lettere, presenti nella colonna più a sinistra del file di dati originale. Quando si usa la funzione `stack`, questi codici verranno impilati assieme ai valori dei dati. Se la prima riga è marcata con `AR` per l'Arkansas, il marcatore `AR` verrà a trovarsi su ogni riga che contiene un'osservazione relativa all'Arkansas. Tutto bene, ma questi marcatori non danno alcuna informazione sulla data dell'osservazione. Per correggere la situazione potremmo eseguire:

```
genr time
genr year = 1960 + (5 * time)
genr markers = "%s:%d", marker, year
```

La prima riga genera un indice che parte da 1 e rappresenta il periodo di ogni osservazione, mentre la seconda riga usa la variabile `time` per generare una variabile che rappresenta l'anno dell'osservazione. La terza riga contiene questa funzionalità speciale: se (e solo se) il nome della nuova "variabile" da generare è `markers`, la parte del comando che segue il segno di uguaglianza viene interpretata come una stringa di formattazione nello stile del linguaggio C (andrà racchiusa tra virgolette doppie), seguita da una lista di argomenti separati da virgola. Gli argomenti verranno stampati seguendo la formattazione indicata e creeranno un nuovo insieme di marcatori per le osservazioni. È possibile indicare come argomento dei nomi di variabili

del dataset, o la stringa marker che rappresenta il marcatore preesistente. Gli specificatori di formato più utili in questo contesto sono %s per le stringhe e %d per i numeri interi. Le stringhe possono essere troncate: ad esempio %.3s indica di usare solo i primi tre caratteri della stringa. Per eliminare i caratteri iniziali da un marcatore esistente e costruirne un altro, si può usare la sintassi marker + n, dove n è un intero positivo: in questo caso, verranno omessi i primi n caratteri.

Dopo aver eseguito i comandi visti sopra, i marcatori delle osservazioni appariranno come, ad esempio, AR:1965, ossia, il codice a due lettere relativo allo stato, seguito dall'anno dell'osservazione, uniti da un carattere due punti.

4.6 Valori mancanti nei dati

I valori mancanti vengono rappresentati internamente come DBL_MAX, il più alto numero in virgola mobile rappresentabile sul sistema (che è probabile sia almeno 10 alla trecentesima potenza, e non va interpretato come un valore legittimo dei dati). Nei file di dati in formato interno vanno rappresentati come NA, mentre se si importano dati in formato CSV gretl riconosce alcuni modi comuni di rappresentare i valori mancanti: -999, la stringa NA (in maiuscolo o minuscolo), un singolo punto, o semplicemente una stringa vuota. Queste ultime, ovviamente, vanno delimitate in modo opportuno, ad es. 120.6,,5.38 indica che il valore di mezzo è mancante.

Per quanto riguarda il trattamento dei valori mancanti durante le analisi statistiche, gretl si comporta nel modo seguente:

- Nel calcolo delle statistiche descrittive (media, scarto quadratico medio, ecc.) con il comando summary, i valori mancanti sono semplicemente ignorati, e la dimensione del campione viene corretta adeguatamente.
- Nel calcolo delle regressioni, gretl per prima cosa corregge l'inizio e la fine del campione, troncandolo dove occorre. Ad esempio, possono esserci dei valori mancanti all'inizio del campione perché la regressione comprende serie differenziate, ritardate e così via. Oppure i valori mancanti possono trovarsi alla fine del campione, a causa della compresenza di serie con diverso livello di aggiornamento, o di serie anticipate.

Se gretl trova dei valori mancanti "all'interno" dell'intervallo del campione per una regressione (che può anche essere troncato), il risultato dipende dal tipo di dataset e dallo stimatore scelto. In molti casi, il programma eseguirà le stime saltando automaticamente le osservazioni che contengono valori mancanti, emettendo un messaggio che indica quante osservazioni sono state escluse. Tuttavia, ci sono procedure che non saltano automaticamente le osservazioni mancanti: tutti gli stimatori autoregressivi, gli stimatori di sistema (come il SUR) e i minimi quadrati non lineari. Nel caso di dati panel, l'esclusione automatica delle osservazioni mancanti avviene solo se il dataset risultante costituisce un panel "bilanciato". In tutti i casi in cui l'esclusione automatica delle osservazioni mancanti non è supportata, gretl emette un messaggio di errore e non produce stime.

In tutti i casi problematici dovuti a valori mancanti all'interno di un dataset, è possibile ricorrere alla funzione misszero (da usare con cautela!) del comando genr. Eseguendo genr pippo = misszero(pluto) è possibile produrre la serie pippo, che è identica a pluto, tranne per il fatto che tutti i valori mancanti sono stati trasformati in zeri. In seguito, costruendo opportunamente delle variabili dummy, sarà possibile eliminare dalla regressione le osservazioni che contengono valori mancanti, pur mantenendo lo stesso intervallo del campione.[4]

4.7 Dimensione massima dei dataset

La dimensione dei dataset (sia in termini di numero di variabili che di osservazioni) è sostanzialmente limitata solo dalle caratteristiche del computer. Gretl alloca la memoria dinamicamente

[4] genr offre anche la funzione inversa di misszero, ossia zeromiss, che sostituisce in una serie i valori zero con il codice per i valori mancanti.

e chiede al sistema operativo tutta la memoria richiesta dai dati. Quindi un limite insuperabile è dato dalla dimensione della memoria RAM.

Escludendo il comando OLS a precisione multipla, gretl di solito usa numeri in virgola mobile in precisione doppia. La dimensione in byte di questi numeri dipende dalla piattaforma, ma tipicamente è pari a otto. Per farsi un'idea delle grandezze in gioco, ipotizzando di avere un dataset con 10.000 osservazioni su 500 variabili, si avranno 5 milioni di numeri in virgola mobile, ossia 40 milioni di byte. Definendo il megabyte (MB) come 1024×1024 byte, come si è soliti fare parlando di memoria RAM, la memoria occupata sarà di circa 38 MB. Il programma richiede ulteriore memoria anche per lo spazio di lavoro, ma, anche tenendo conto di questo fatto, la gestione di un dataset di queste dimensioni è fattibile su un PC moderno, che ha tipicamente almeno 256 MB di RAM.

Se la RAM non pone problemi, c'è un'altra limitazione sulla dimensione dei dati, che però difficilmente diventa un vincolo stringente: le variabili e le osservazioni sono indicizzate usando numeri interi col segno, che un tipico PC memorizza come valori a 32 bit, avendo quindi un limite massimo pari a 2.147.483.647.

Questi limiti si applicano alle funzionalità "interne" di gretl. Ci sono limiti più stringenti per quanto riguarda due programmi esterni che sono disponibili come aggiunte a gretl per alcuni tipi di analisi delle serie storiche, ossia TRAMO/SEATS e X-12-ARIMA. Questi programmi utilizzano un meccanismo di allocazione della memoria a dimensione fissa e quindi non possono gestire serie più lunghe di 600 osservazioni.

4.8 Raccolte di file di dati

Se si usa gretl nell'attività didattica, può essere utile creare una raccolta di file di dati e/o di script di comandi, personalizzati per il proprio corso, ad uso degli studenti.

A partire dalla versione 1.2.1 di gretl, ci sono tre modi per accedere a una raccolta di file:

- Per i file di dati: selezionare dal menù "File, Apri dati, File di esempio", o fare clic sull'icona a forma di cartella sulla barra degli strumenti di gretl.
- Per i file di comandi: selezionare dal menù "File, Comandi, File di esempio".

Quando un utente seleziona uno dei comandi visti sopra:

- Vengono elencati automaticamente i file di dati o di comandi inclusi nella distribuzione di gretl (che comprendono i file relativi a *Introductory Econometrics* di Ramanathan e a *Econometric Analysis* di Greene).
- Il programma cerca alcune raccolte di dati opzionali, ad esempio i file relativi ad alcuni libri di testo (Davidson e MacKinnon, Gujarati, Stock e Watson, Verbeek, Wooldridge) e la Penn World Table (PWT 5.6). Si veda la pagina dei dati sul sito web di gretl per ulteriori informazioni su queste raccolte. Se queste raccolte vengono trovate, vengono aggiunte all'elenco dei file disponibili.
- Il programma infine cerca delle raccolte di dati (non necessariamente note) nei posti seguenti: la directory "di sistema" dei file di dati, la directory di sistema dei file di comandi, la directory utente e tutte le loro sotto-directory di primo livello. Valori tipici per i nomi di queste directory sono mostrati nella tabella 4.1. Si noti che PERSONAL è una parola chiave che viene espansa da Windows, ad esempio in "My Documents" se sul sistema è impostata la lingua inglese.

Le raccolte trovate verranno aggiunte all'elenco dei file disponibili. In che formato deve essere una raccolta per essere riconosciuta come tale? Una raccolta può essere costituita da un gruppo di file di dati di gretl in formato XML (con l'estensione `.gdt`) o da un gruppo di file di comandi (con l'estensione `.inp`), in entrambi i casi accompagnati da un "file principale" o catalogo. La distribuzione di gretl contiene vari esempi di file di catalogo, ad esempio il file

	Linux	*MS Windows*
Directory di sistema per i dati	`/usr/share/gretl/data`	`c:\Program Files\gretl\data`
Directory di sistema per i comandi	`/usr/share/gretl/scripts`	`c:\Program Files\gretl\scripts`
Directory utente	`$HOME/gretl`	`PERSONAL\gretl`

Tabella 4.1: Posizioni tipiche delle raccolte di file

`descriptions` nella sottodirectory `misc` della directory dati di gretl e il file `ps_descriptions` nella sottodirectory `misc` della directory dei comandi.

Se si intende aggiungere una propria raccolta, occorrerà creare dei file di catalogo, chiamati `descriptions` per i file di dati, e `ps_descriptions` per i file di comandi, nelle rispettive directory (ad es. `/usr/share/gretl/data/mydata` o `c:\userdata\gretl\data\mydata`).

La sintassi dei file di catalogo (che sono file di testo) è semplice; ecco ad esempio le prime righe del catalogo della raccolta di file di dati "misc" inclusa nella distribuzione di gretl:

```
# Gretl: various illustrative datafiles
"arma","artificial data for ARMA script example"
"ects_nls","Nonlinear least squares example"
"hamilton","Prices and exchange rate, U.S. and Italy"
```

La prima riga, che deve iniziare con un carattere cancelletto, contiene un nome breve, qui "Gretl", che comparirà come etichetta identificativa per questa raccolta nella finestra di selezione dei dati, seguito da una virgola e da una descrizione breve della raccolta (opzionale).

Le righe seguenti contengono due elementi, separati da una virgola e racchiusi tra virgolette doppie. Il primo è il nome del file di dati (escludendo l'estensione `.gdt`), mentre il secondo è una breve descrizione del contenuto del file di dati. Dovrebbe esserci una riga come questa per ogni file di dati della raccolta.

I file di catalogo per le raccolte di file di comandi sono molto simili a quelli appena visti, tranne per il fatto che ogni riga del file contiene tre campi: il nome del file (senza l'estensione `.inp`), una breve descrizione del significato econometrico della serie di comandi contenuti nel file e una breve descrizione dei dati usati. Ecco un altro esempio: le prime righe del catalogo della raccolta di file di comandi "misc" inclusa nella distribuzione di gretl:

```
# Gretl: various sample scripts
"arma","ARMA modeling","artificial data"
"ects_nls","Nonlinear least squares (Davidson)","artificial data"
"leverage","Influential observations","artificial data"
"longley","Multicollinearity","US employment"
```

La procedura per creare la propria raccolta di dati e renderla disponibile agli utenti è la seguente:

1. Assemblare i dati, nel formato più comodo.
2. Convertire i dati in formato gretl e salvarli come file `gdt`. Probabilmente il modo più semplice consiste nell'importare i dati nel programma come testo semplice, CSV o formato foglio elettronico (MS Excel o Gnumeric) e quindi salvarli. Può essere utile aggiungere delle descrizioni delle singole variabili (usando il comando "Variabile, Modifica attributi") e delle informazioni sulle fonti dei dati (usando il comando "Dati, Modifica descrizione").
3. Scrivere un file di catalogo per la raccolta, usando un editor di testi.
4. Copiare i file di dati e il file di catalogo in una sottodirectory della directory dei dati (o utente) di gretl.
5. Se la raccolta deve essere distribuita ad altri utenti, creare un pacchetto contenente i file di dati e il catalogo, ad esempio sotto forma di file zip.

Se la raccolta creata non contiene dati proprietari, è possibile inviarla al curatore di gretl in modo che venga resa disponibile a tutti gli utenti del programma come pacchetto dati opzionale.

Capitolo 5

Modifica del campione

5.1 Introduzione

Questo capitolo affronta alcune questioni correlate alla creazione di sotto-campioni in un dataset.

È possibile definire un sotto-campione per un dataset in due modi diversi, che chiameremo rispettivamente “impostazione” del campione e “restrizione” del campione.

5.2 Impostazione del campione

Per “impostazione” del campione, si intende la definizione di un campione ottenuta indicando il punto iniziale e/o quello finale dell’intervallo del campione. Questa modalità è usata tipicamente con le serie storiche; ad esempio se si hanno dati trimestrali per l’intervallo da 1960:1 a 2003:4 e si vuole stimare una regressione usando solo i dati degli anni ’70, un comando adeguato è

```
smpl 1970:1 1979:4
```

Oppure se si vuole riservare la parte finale delle osservazioni disponibili per eseguire una previsione "fuori dal campione", si può usare il comando

```
smpl ; 2000:4
```

dove il punto e virgola significa “mantenere inalterata l’osservazione iniziale” (e potrebbe essere usato in modo analogo al posto del secondo parametro, indicando di mantenere inalterata l’osservazione finale). Per “inalterata” in questo caso si intende inalterata relativamente all’ultima impostazione eseguita con `smpl`, o relativamente all’intero dataset, se in precedenza non è ancora stato definito alcun sotto-campione. Ad esempio, dopo

```
smpl 1970:1 2003:4
smpl ; 2000:4
```

l’intervallo del campione sarà da 1970:1 a 2000:4.

È possibile anche impostare l’intervallo del campione in modo incrementale o relativo: in questo caso occorre indicare per il punto iniziale e finale uno spostamento relativo, sotto forma di numero preceduto dal segno più o dal segno meno (o un punto e virgola per indicare nessuna variazione). Ad esempio

```
smpl +1 ;
```

sposterà in avanti di un’osservazione l’inizio del campione, mantenendo inalterata la fine del campione, mentre

```
smpl +2 -1
```

sposterà l’inizio del campione in avanti di due osservazioni e la fine del campione indietro di una.

Una caratteristica importante dell’operazione di “impostazione del campione” descritta fin qui è che il sotto-campione creato risulta sempre composto da un insieme di osservazioni contigue. La struttura del dataset rimane quindi inalterata: se si lavora su una serie trimestrale, dopo aver impostato il campione la serie rimarrà trimestrale.

5.3 Restrizione del campione

Per "restrizione" del campione si intende la definizione di un campione ottenuta selezionando le osservazioni in base a un criterio Booleano (logico), o usando un generatore di numeri casuali. Questa modalità è usata tipicamente con dati di tipo cross-section o panel.

Si supponga di avere dei dati di tipo cross-section che descrivono il genere, il reddito e altre caratteristiche di un gruppo di individui e si vogliano analizzare solo le donne presenti nel campione. Se si dispone di una variabile dummy `genere`, che vale 1 per gli uomini e 0 per le donne, si potrebbe ottenere questo risultato con

```
smpl genere=0 --restrict
```

Oppure si supponga di voler limitare il campione di lavoro ai soli individui con un reddito superiore ai 50.000 euro. Si potrebbe usare

```
smpl reddito>50000 --restrict
```

Qui sorge un problema: eseguendo in sequenza i due comandi visti sopra, cosa conterrà il sotto-campione? Tutti gli individui con reddito superiore a 50.000 euro o solo le donne con reddito superiore a 50.000 euro? La risposta corretta è la seconda: la seconda restrizione si aggiunge alla prima, ossia la restrizione finale è il prodotto logico della nuova restrizione e di tutte le restrizioni precedenti. Se si vuole applicare una nuova restrizione indipendentemente da quelle applicate in precedenza, occorre prima re-impostare il campione alla sua lunghezza originaria, usando

```
smpl --full
```

In alternativa, è possibile aggiungere l'opzione `replace` al comando `smpl`:

```
smpl income>50000 --restrict --replace
```

Questa opzione ha l'effetto di re-impostare automaticamente il campione completo prima di applicare la nuova restrizione.

A differenza della semplice "impostazione" del campione, la "restrizione" del campione può produrre un insieme di osservazioni non contigue nel dataset originale e può anche modificare la struttura del dataset.

Questo fenomeno può essere osservato nel caso dei dati panel: si supponga di avere un panel di cinque imprese (indicizzate dalla variabile `impresa`) osservate in ognuno degli anni identificati dalla variabile `anno`. La restrizione

```
smpl anno=1995 --restrict
```

produce un dataset che non è più di tipo panel, ma cross-section per l'anno 1995. In modo simile

```
smpl impresa=3 --restrict
```

produce un dataset di serie storiche per l'impresa numero 3.

Per questi motivi (possibile non-contiguità nelle osservazioni, possibile cambiamento nella struttura dei dati) gretl si comporta in modo diverso a seconda che si operi una "restrizione" del campione o una semplice "impostazione" di esso. Nel caso dell'impostazione, il programma memorizza semplicemente le osservazioni iniziali e finali e le usa come parametri per i vari comandi di stima dei modelli, di calcolo delle statistiche ecc. Nel caso della restrizione, il programma crea una copia ridotta del dataset e la tratta come un semplice dataset di tipo cross-section non datato.[1]

[1] Con una eccezione: se si parte da un dataset panel bilanciato e la restrizione è tale da preservare la struttura di panel bilanciato (ad esempio perché implica la cancellazione di tutte le osservazioni per una unità cross-section), allora il dataset ridotto è ancora trattato come panel.

Se si vuole re-imporre un'interpretazione di tipo "serie storiche" o "panel" al dataset ridotto, occorre usare il comando `setobs`, o il comando dal menù "Dati, Struttura dataset", se appropriato.

Il fatto che una "restrizione" del campione comporti la creazione di una copia ridotta del dataset originale può creare problemi quando il dataset è molto grande (nell'ordine delle migliaia di osservazioni). Se si usano simili dataset, la creazione della copia può causare l'esaurimento della memoria del sistema durante il calcolo dei risultati delle regressioni. È possibile aggirare il problema in questo modo:

1. Aprire il dataset completo e imporre la restrizione sul campione.
2. Salvare su disco una copia del dataset ridotto.
3. Chiudere il dataset completo e aprire quello ridotto.
4. Procedere con l'analisi.

5.4 Campionamento casuale

Se si usano dataset molto grandi (o se si intende studiare le proprietà di uno stimatore), può essere utile estrarre un campione casuale dal dataset completo. È possibile farlo ad esempio con

```
smpl 100 --random
```

che seleziona 100 osservazioni. Se occorre che il campione sia riproducibile, occorre per prima cosa impostare il seme del generatore di numeri casuali, usando il comando `set`. Questo tipo di campionamento è un esempio di "restrizione" del campione: viene infatti generata una copia ridotta del dataset.

5.5 I comandi del menù Campione

Gli esempi visti finora hanno mostrato il comando testuale `set`, ma è possibile creare un sotto-campione usando i comandi del menù "Campione" nell'interfaccia grafica del programma.

I comandi del menù permettono di ottenere gli stessi risultati delle varianti del comando testuale `smpl`, con la seguente eccezione: se si usa il comando "Campione, Imposta in base a condizione..." e sul dataset è già stato impostato un sotto-campione, viene data la possibilità di preservare la restrizione già attiva o di sostituirla (in modo analogo a quanto avviene invocando l'opzione `replace` descritta nella sezione 5.3).

Capitolo 6

Grafici e diagrammi

6.1 Grafici gnuplot

Gretl richiama un programma separato, gnuplot, per generare i grafici. Gnuplot è un programma di grafica molto completo, con una miriade di opzioni, disponibile su www.gnuplot.info (si noti che la versione MS Windows di gretl include già gnuplot). gretl fornisce l'accesso, attraverso un'interfaccia grafica, a una parte delle opzioni di gnuplot, ma è possibile anche controllare l'aspetto di un grafico in tutti i suoi dettagli, se si vuole.

Mentre un grafico viene visualizzato, facendo clic sulla finestra del grafico si aprirà un menù pop-up con le seguenti opzioni:

- Salva come PNG: salva il grafico in formato Portable Network Graphics
- Salva come postscript: salva in formato encapsulated postscript (EPS)
- Salva come Windows metafile: salva in formato Enhanced Metafile (EMF).
- Salva alla sessione come icona: il grafico apparirà sotto forma di icona quando si seleziona "Finestra Icone" dal menù Visualizza
- Ingrandisci: permette di selezionare un'area all'interno del grafico per visualizzarla da vicino
- Stampa: permette di stampare il grafico direttamente (disponibile solo in Gnome e MS Windows)
- Copia negli appunti: (solo in MS Windows) permette di copiare il grafico per poi incollarlo in altri programmi Windows, come ad esempio MS Word [1]
- Modifica: apre una finestra che permette di modificare vari dettagli dell'aspetto del grafico
- Chiudi: chiude la finestra del grafico

Mostrare le etichette dei dati

Nel caso di semplici diagrammi a dispersione X-Y (con o senza la retta di regressione), sono disponibili altre opzioni se il dataset contiene "marcatori" (ossia etichette che identificano ogni osservazione)[2]. Quando il diagramma a dispersione è visualizzato, muovendo il puntatore del mouse su un punto dei dati, viene mostrata l'etichetta corrispondente. In modalità predefinita, queste etichette non compaiono nelle versioni stampate o copiate del grafico, e possono essere rimosse selezionando "Cancella le etichette dei dati" dal menù pop-up del grafico. Se si desidera rendere permanenti le etichette (cosicché siano visibili anche se il grafico è stampato o copiato), ci sono due opzioni:

- Per fissare le etichette che sono mostrate in un dato momento, selezionare "Fissa le etichette dei dati" dal menù pop-up del grafico.

[1] Per ottenere i risultati migliori quando si incollano grafici nelle applicazioni di MS Office, usare il comando "Modifica, Incolla speciale..." dell'applicazione e selezionare l'opzione "Immagine (Enhanced Metafile)".

[2] Per un esempio di dataset simili, si veda il file di Ramanathan `data4-10`: esso contiene dati sulle iscrizioni alle scuole private per i 50 stati degli USA, incluso Washington DC; i marcatori rappresentano i codici a due lettere per i vari stati.

- Per fissare le etichette per tutti i punti del grafico, selezionare “Modifica” dal menù pop-up e marcare la casella “Mostra tutte le etichette dei dati”. Questa opzione è disponibile solo se ci sono meno di 55 punti, e produrrà i risultati migliori se i punti del grafico non sono troppo addensati, altrimenti le etichette tenderanno a sovrapporsi.

Per rimuovere le etichette che sono state fissate in uno di questi due modi, basta selezionare “Modifica” dal menù pop-up e disattivare la casella “Mostra tutte le etichette dei dati”.

Opzioni avanzate

Se si conosce gnuplot e si desidera un controllo sull’aspetto del grafico più preciso di quello fornito dalla finestra di modifica del grafico (opzione “Modifica”), ci sono due possibilità:

- Una volta salvato il grafico come icona di sessione, facendo clic col tasto destro sull’icona si apre un altro menù pop-up. Una delle opzioni disponibili è “Comandi per modificare il grafico”, che apre una finestra di modifica con i comandi di gnuplot. È possibile modificare questi comandi e salvarli per il futuro, oppure inviarli direttamente a gnuplot (usando l’icona Esegui sulla barra degli strumenti nella finestra di modifica dei comandi).
- Un altro modo per salvare i comandi del grafico (o per salvare il grafico in formati diversi da EPS o PNG) è quello di aprire la finestra di modifica del grafico usando il comando “Modifica” nel menù pop-up del grafico e quindi facendo clic su “File”: verrà visualizzato un menù a discesa con i formati in cui è possibile salvare il grafico.

Per saperne di più su gnuplot si veda il manuale online o www.gnuplot.info. Si veda anche la voce `gnuplot` nella *Guida ai comandi di gretl* e i comandi `graph` e `plot` per generare dei semplici “grafici ASCII”.

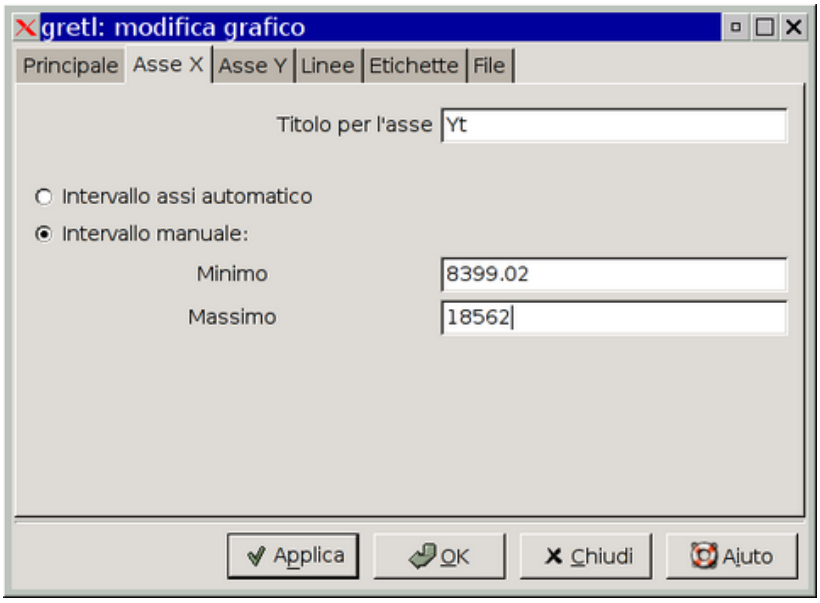


Figura 6.1: Finestra di modifica dei grafici gnuplot di gretl

6.2 Boxplot

I grafici boxplot non vengono generati con gnuplot, ma da gretl stesso.

Questi grafici (da Tukey e Chambers) mostrano la distribuzione di una variabile. La “scatola” centrale (box) racchiude il 50 per cento centrale dei dati, ossia è delimitata dal primo e dal terzo quartile. I “baffi” (whiskers) si estendono fino ai valori minimo e massimo. Una linea trasversale sulla scatola indica la mediana.

Nel caso dei grafici a tacca ("notches"), una tacca indica i limiti dell'intervallo di confidenza approssimato al 90 per cento per la mediana, ottenuto col metodo bootstrap (se la serie dei dati è molto lunga, potrebbe essere necessario un po' di tempo).

Facendo clic nella finestra del boxplot si ottiene un menù che permette di salvare il grafico come file encapsulated postscript (EPS) o come file postscript a piena pagina. Se si usa il sistema X Window è anche possibile salvare il grafico come file XPM, mentre in MS Windows è possibile copiarlo negli appunti in formato bitmap. Il menù dà anche la possibilità di mostrare un riepilogo in cinque numeri (minimo, primo quartile, mediana, terzo quartile, massimo) e un intervallo di confidenza per la mediana, se il boxplot è del tipo "a tacca".

Alcuni dettagli del funzionamento dei boxplot di gretl possono essere controllati attraverso un file testuale chiamato `.boxplotrc` (o in alternativa `plotconfig.txt`), che viene cercato, nell'ordine, nella directory di lavoro attuale, nella directory home dell'utente (che corrisponde alla variabile d'ambiente HOME) e nella directory utente di gretl (scelta attraverso il comando "Strumenti, Preferenze, Generali..."). Tra le opzioni che possono essere specificate in questo modo ci sono: il carattere da usare per l'output in postscript (deve essere un nome di font postscript valido; il valore predefinito è Helvetica), la dimensione del carattere in punti (sempre per l'output in postscript; il valore predefinito è 12), i valori minimo e massimo per l'asse y, la larghezza e l'altezza del grafico in pixel (valori predefiniti: 560 x 448), se occorre mostrare anche i valori numerici per i quartili e la mediana (l'impostazione predefinita non li mostra) e se occorre indicare separatamente gli outlier, ossia i punti che distano più di 1.5 volte il range interquartile dalla scatola centrale (l'impostazione predefinita non li mostra). Ecco un esempio:

```
font = Times-Roman
fontsize = 16
max = 4.0
min = 0
width = 400
height = 448
numbers = %3.2f
outliers = true
```

Sulla penultima riga, il valore associato a `numbers` è una stringa di formato "printf" come quelle usate nel linguaggio di programmazione C; se viene specificata, controlla il modo in cui vengono mostrati la mediana e i quartili accanto al boxplot, altrimenti questi valori non vengono mostrati. Nell'esempio, i valori verranno mostrati usando 3 cifre in totale e 2 cifre di precisione dopo il punto decimale.

Non occorre specificare tutte le opzioni, né importa l'ordine in cui vengono specificate. Le righe che non seguono la struttura "variabile = valore" vengono ignorate, così come le righe che iniziano con il carattere cancelletto, #.

Dopo ogni variabile specificata nel comando boxplot è possibile inserire un'espressione booleana tra parentesi per limitare il campione da utilizzare per la variabile in questione, avendo cura di inserire uno spazio tra il nome (o il numero) della variabile e l'espressione. Si supponga di avere una variabile `salario` con gli stipendi di uomini e donne, e una variabile dummy `GENERE` che vale 1 per gli uomini e 0 per le donne. In questo caso è possibile disegnare dei boxplot comparativi usando il seguente comando nella finestra di dialogo:

```
salario (GENERE=1) salario (GENERE=0)
```

Capitolo 7

Funzioni speciali in genr

7.1 Introduzione

Il comando `genr` offre un modo flessibile per definire nuove variabili. Il comando è documentato nella *Guida ai comandi di gretl*, mentre questo capitolo offre una discussione più approfondita di alcune delle funzioni speciali disponibili con `genr` e di alcune particolarità del comando.

7.2 Varianza di lungo periodo

Come è noto, la varianza della media di T variabili aleatorie $x_1, x_2, \ldots, x_T$ con uguale varianza σ^2 è pari a σ^2/T se i dati sono non correlati. In questo caso, la varianza campionaria di x_t rappresenta uno stimatore consistente.

Se però esiste correlazione seriale tra le x_t, la varianza di $\bar{X} = T^{-1}\sum_{t=1}^{T} x_t$ va stimata in modo diverso. Una delle statistiche più usate a questo scopo è uno stimatore kernel non parametrico con il kernel di Bartlett definito come

$$\hat{\omega}^2(k) = T^{-1}\sum_{t=k}^{T-k}\left[\sum_{i=-k}^{k} w_i(x_t - \bar{X})(x_{t-i} - \bar{X})\right], \tag{7.1}$$

dove l'intero k è definito come l'ampiezza della finestra, mentre i termini w_i sono i cosiddetti *pesi di Bartlett*, definiti come $w_i = 1 - \frac{|i|}{k+1}$. Si può dimostrare che, per k abbastanza alto, $\hat{\omega}^2(k)/T$ rappresenta uno stimatore consistente alla varianza di $\bar{X}$.

Gretl implementa questo stimatore usando la funzione `lrvar()`, che accetta due argomenti: la serie di cui si vuole stimare la varianza di lungo periodo e lo scalare k. Se k è negativo, viene usato il diffuso valore $T^{1/3}$.

7.3 Filtri per serie storiche

Un tipo di funzioni speciali di `genr` consente il filtraggio delle serie storiche. Oltre alle solite operazioni di ritardo e differenza, gretl fornisce anche la differenza frazionaria e due filtri usati spesso in macroeconomia per la scomposizione fra ciclo e trend: il filtro di Hodrick–Prescott e quello passa banda di Baxter–King.

Differenza frazionaria

Differenziare una serie storica per d volte è un'operazione ovvia quando d è un numero intero, ma può sembrare strana quando d è una frazione. Tuttavia, questa idea ha un ben preciso fondamento matematico: si consideri la funzione

$$f(z) = (1 - z)^{-d},$$

dove z e d sono numeri reali. L'espansione in serie di Taylor intorno a $z = 0$ mostra che

$$f(z) = 1 + dz + \frac{d(d+1)}{2}z^2 + \cdots$$

o, più sinteticamente,

$$f(z) = 1 + \sum_{i=1}^{\infty}\psi_i z^i$$

con

$$\psi_k = \frac{\prod_{i=1}^{k}(d+i-1)}{k!} = \psi_{k-1}\frac{d+k-1}{k}$$

La stessa espansione può essere usata con l'operatore ritardo, così che se definiamo

$$Y_t = (1-L)^{0.5}X_t$$

potrebbe essere considerata equivalente a

$$Y_t = X_t - 0.5X_{t-1} - 0.125X_{t-2} - 0.0625X_{t-3} - \cdots$$

In gretl questa trasformazione può essere eseguita con il comando

```
genr Y = fracdiff(X,0.5)
```

Il filtro di Hodrick–Prescott

Questo filtro è utilizzabile tramite la funzione `hpfilt()`, che accetta un argomento: il nome della variabile da processare.

Una serie storica y_t può essere scomposta in un trend, o componente di crescita g_t e in una componente ciclica c_t.

$$y_t = g_t + c_t, \quad t = 1, 2, \ldots, T$$

Il filtro di Hodrick–Prescott effettua questa scomposizione, minimizzando l'espressione seguente:

$$\sum_{t=1}^{T}(y_t - g_t)^2 + \lambda \sum_{t=2}^{T-1}\left((g_{t+1} - g_t) - (g_t - g_{t-1})\right)^2.$$

Il primo termine è la somma dei quadrati delle componenti cicliche $c_t = y_t - g_t$. Il secondo termine è un multiplo λ della somma dei quadrati delle differenze seconde della componente di trend. Questo secondo termine penalizza le variazioni nel tasso di crescita della componente di trend: maggiore è il valore di λ, maggiore sarà la penalizzazione, e quindi più regolare sarà la serie di trend.

Si noti che la funzione `hpfilt` in gretl produce la componente di ciclo, c_t, della serie originale. Se si vuole il trend depurato, basta sottrarre il ciclo dalla serie originale:

```
genr ct = hpfilt(yt)
genr gt = yt - ct
```

Hodrick e Prescott (1997) suggeriscono che un valore $\lambda = 1600$ sia ragionevole per dati trimestrali. Il valore predefinito in gretl è il quadrato della frequenza dei dati, moltiplicato per 100 (che dà appunto 1600 per dati trimestrali). Il valore può essere modificato con il comando `set` sul parametro `hp_lambda`. Ad esempio, `set hp_lambda 1200`.

Il filtro di Baxter e King

Questo filtro è utilizzabile tramite la funzione `bkfilt()`; anche questa accetta come unico argomento il nome della variabile da processare.

Si consideri la rappresentazione spettrale di una serie storica y_t:

$$y_t = \int_{-\pi}^{\pi} e^{i\omega}\mathrm{d}Z(\omega)$$

Per estrarre la componente di y_t che si trova tra le frequenze $\underline{\omega}$ e $\overline{\omega}$ potremmo applicare un filtro passa banda:

$$c_t^* = \int_{-\pi}^{\pi} F^*(\omega)e^{i\omega}\mathrm{d}Z(\omega)$$

dove $F^*(\omega) = 1$ per $\underline{\omega} < |\omega| < \overline{\omega}$ e 0 altrove. Ciò implicherebbe, nel dominio temporale, applicare alla serie un filtro con un numero infinito di coefficienti, cosa non desiderabile. Il filtro passa banda di Baxter e King applica a y_t un polinomio finito nell'operatore di ritardo $A(L)$:

$$c_t = A(L)y_t$$

dove $A(L)$ è definito come

$$A(L) = \sum_{i=-k}^{k} a_i L^i$$

I coefficienti a_i sono scelti in modo che $F(\omega) = A(e^{i\omega})A(e^{-i\omega})$ sia la migliore approssimazione di $F^*(\omega)$ per un dato k. Chiaramente, maggiore è k, migliore è l'approssimazione, ma poiché ciò implica scartare $2k$ osservazioni, di solito si cerca un compromesso. Inoltre, il filtro ha altre proprietà teoriche interessanti, tra cui quella che $a(1) = 0$, quindi una serie con una sola radice unitaria è resa stazionaria dall'applicazione del filtro.

In pratica, il filtro è usato di solito con dati mensili o trimestrali per estrarne la componente di "ciclo economico", ossia la componente tra 6 e 36 trimestri. I valori usuali per k sono 8 o 12 (o forse di più per serie mensili). I valori predefiniti per i limiti di frequenza sono 8 e 32, mentre il valore predefinito per l'ordine di approssimazione, k, è 8. È possibile impostare questi valori usando il comando `set`. La parola chiave per impostare i limiti di frequenza è `bkbp_limits`, mentre quella per k è `bkbp_k`. Quindi ad esempio, se si stanno usando dati mensili e si vuole impostare i limiti di frequenza tra 18 e 96, e k a 24, si può eseguire

```
set bkbp_limits 18 96
set bkbp_k 24
```

Questi valori resteranno in vigore per le chiamate alla funzione `bkfilt` finché non saranno modificati da un altro uso di `set`.

7.4 Dati panel

Variabili dummy

In uno studio panel, può nascere l'esigenza di costruire delle variabili dummy di uno dei seguenti tipi: (a) dummy che identificano ciascuna delle unità cross-section, o (b) dummy che identificano ciascuno dei periodi. Il primo tipo può essere usato per permettere all'intercetta della regressione di variare tra le unità cross-section, il secondo per permettere all'intercetta di variare tra i periodi.

Per creare questo tipo di dummy, è possibile usare le due funzioni speciali del menù Aggiungi, o del comando testuale `genr`.

1. "Dummy per unità" (comando testuale `genr unitdum`). Questo comando crea un insieme di variabili dummy che identificano le unità cross section. La variabile `du_1` avrà valore 1 in ogni riga dei dati che corrisponde a un'osservazione della prima unità cross section, e 0 altrove; `du_2` avrà valore 1 in ogni riga dei dati che corrisponde a un'osservazione della seconda unità cross section, e così via.
2. "Dummy temporali" (comando testuale `genr timedum`). Questo comando crea un insieme di variabili dummy che identificano i periodi. La variabile `dt_1` avrà valore 1 in ogni riga dei dati che corrisponde a un'osservazione del primo periodo, e 0 altrove; `dt_2` avrà valore 1 in ogni riga dei dati che corrisponde a un'osservazione del secondo periodo, e così via.

Se un dataset panel contiene l'anno di ogni osservazione all'interno della variabile `ANNO`, è possibile creare una dummy periodica per un anno particolare, ad esempio con `genr dum = (ANNO=1960)`. È anche possibile creare variabili dummy periodiche usando l'operatore modulo, `%`. Ad esempio, per creare una dummy che valga 1 ogni trenta osservazioni a partire dalla prima e 0 altrove, basta eseguire

```
genr index
genr dum = ((index-1)%30) = 0
```

Ritardi, differenze, trend

Se i periodi temporali sono distanziati in modo uniforme, è possibile usare valori ritardati delle variabili in una regressione panel (ma si veda la sezione 15.2; è anche possibile costruire differenze prime delle variabili.

Se un dataset è identificato correttamente come panel, gretl gestirà correttamente la generazione di questo tipo di variabili. Ad esempio, il comando `genr x1_1 = x1(-1)` creerà una variabile che contiene il primo ritardo di `x1`, laddove è disponibile, e il codice di valore mancante, laddove il ritardo non è disponibile (ad esempio nella prima osservazione per ogni gruppo). Quando si esegue una regressione che include questo tipo di variabili, il programma escluderà automaticamente le osservazioni mancanti.

Quando un dataset panel ha una dimensione temporale sostanziale, può essere utile includere un trend nell'analisi. Il comando `genr time` crea una variabile di nome `time` che assume valori compresi tra 1 e T per ogni unità, dove T è la lunghezza della dimensione temporale del panel. Per creare un indice che assuma valori compresi tra 1 e $m \times T$, dove m è il numero di unità nel panel, si usi invece `genr index`.

Statistiche descrittive per unità

Le funzioni `pmean()` e `psd()` possono essere usate per generare semplici statistiche descrittive (media e scarto quadratico medio) per una data variabile, calcolate per gruppo.

Supponendo di avere un dataset panel che comprende 8 osservazioni temporali per ciascuna di N unità o gruppi. Il comando

```
genr pmx = pmean(x)
```

crea una serie di questo tipo: i primi 8 valori (che corrispondono all'unità 1) contengono la media di `x` per l'unità 1, i successivi 8 valori contengono la media per l'unità 2 e così via. La funzione `psd()` funziona in modo simile. Lo scarto quadratico medio campionario per il gruppo i è calcolato come

$$s_i = \sqrt{\frac{\sum (x - \bar{x}_i)^2}{T_i - 1}}$$

dove T_i denota il numero di osservazioni valide su `x` per l'unità data, $\bar{x}_i$ denota la media di gruppo, e la somma viene fatta su tutte le osservazioni valide per il gruppo. Se però vale $T_i < 2$, lo scarto quadratico medio viene impostato pari a 0.

È interessante notare un uso particolare di `psd()`: se si vuole formare un sotto-campione di un panel che contenga solo quelle unità per cui la variabile `x` varia nel tempo, si può eseguire

```
smpl (psd(x) > 0) --restrict
```

Funzioni speciali per manipolare i dati

Oltre alle funzioni discusse sopra, ci sono alcune opzioni di `genr` particolarmente utili per manipolare i dati panel, soprattutto quando i dati sono stati importati da una fonte esterna e non sono nella forma corretta per l'analisi panel. Queste funzionalità sono spiegate nel Capitolo 4.

7.5 Ricampionamento e bootstrapping

Un'altra funzione particolare è il ricampionamento, con reimmissione, di una serie. Data una serie di dati originale `x`, il comando

```
genr xr = resample(x)
```

crea una nuova serie in cui ognuno degli elementi è estratto in modo casuale dagli elementi di `x`. Se la serie originale ha 100 osservazioni, ogni elemento di `x` è scelto con probabilità 1/100 ad ogni estrazione. L'effetto è quindi di "rimescolare" gli elementi di `x`, con la particolarità che ogni elemento di `x` può apparire più di una volta, o non apparire affatto, in `xr`.

L'uso principale di questa funzione è la costruzione di intervalli di confidenza o p-value con il metodo bootstrap. Ecco un semplice esempio: si supponga di aver stimato una semplice regressione OLS di y su x e di aver trovato che il coefficiente della pendenza abbia un rapporto t pari a 2.5 con 40 gradi di libertà. Il p-value a due code per l'ipotesi nulla che il parametro della pendenza sia pari a zero vale quindi 0.0166, usando la distribuzione $t(40)$. A seconda del contesto, però, potremmo dubitare del fatto che il rapporto tra il coefficiente e l'errore standard segua veramente una distribuzione $t(40)$. In questo caso, potremmo derivare un valore bootstrap per il p-value come mostrato nell'esempio 7.1.

Sotto l'ipotesi nulla che la pendenza rispetto a x sia pari a zero, y è uguale alla sua media più un termine di errore. Simuliamo y ricampionando i residui del modello OLS iniziale e ri-stimiamo il modello. Ripetiamo questa procedura un gran numero di volte e registriamo il numero di casi in cui il valore assoluto del rapporto t è maggiore di 2.5: la proporzione di questo numero di casi è il nostro valore bootstrap per il p-value. Per una buona discussione dei test basati sulla simulazione e sui metodi bootstrap, si veda Davidson e MacKinnon (2004, capitolo 4).

Esempio 7.1: Calcolo del p-value col metodo bootstrap

```
ols y 0 x
# salva i residui
genr ui = $uhat
scalar ybar = mean(y)
# numero delle replicazioni per il bootstrap
scalar replics = 10000
scalar tcount = 0
series ysim = 0
loop replics --quiet
  # genera i valori simulati di y ricampionando
  ysim = ybar + resample(ui)
  ols ysim 0 x
  scalar tsim = abs($coeff(x) / $stderr(x))
  tcount += (tsim > 2.5)
endloop
printf "Proporzione dei casi con |t| > 2.5 = %g\n", tcount / replics
```

7.6 Densità cumulate e p-value

Le due funzioni `cdf` e `pvalue` forniscono strumenti complementari per esaminare i valori di varie distribuzioni di probabilità: la normale standard, la t di Student, la χ^2, la F, la gamma, e la binomiale. La sintassi di queste funzioni è spiegata nella *Guida ai comandi di gretl*; in questa sede viene presentato un aspetto particolare riguardante la precisione dei risultati.

La funzione di ripartizione, o di densità cumulata (CDF), per una variabile casuale è l'integrale della densità della variabile, dal suo limite inferiore (tipicamente $-\infty$ o 0) fino a un certo valore x. Il p-value (almeno il p-value destro, a una coda, fornito dalla funzione `pvalue`) è la probabilità complementare, l'integrale da x al limite superiore della distribuzione, tipicamente $+\infty$.

In linea di principio non c'è bisogno di due funzioni distinte: dato un valore della funzione di ripartizione p_0 è possibile ricavare facilmente il p-value come $1 - p_0$ (o viceversa). In pratica, poiché il computer usa aritmetica a precisione finita, due funzioni non sono ridondanti. In gretl, come nella maggior parte dei programmi statistici, i numeri a virgola mobile sono rappresentati tramite dei "double" — valori a precisione doppia, che sono tipicamente memorizzati in 8 byte, o 64 bit. Visto che il numero di bit disponibili è fisso, i numeri in virgola mobile che possono essere rappresentati sono limitati: *i "double" non modellano esattamente la retta reale*. Tipicamente, i "double" possono rappresentare numeri che stanno all'incirca nell'intervallo $\pm 1.7977 \times 10^{308}$, ma con circa solo 15 cifre di precisione.

Supponiamo di essere interessati alla coda sinistra della distribuzione χ^2 con 50 gradi di libertà, ossia di voler conoscere il valore della CDF per $x = 0.9$. Vediamo la seguente sessione interattiva:

```
? genr p1 = cdf(X, 50, 0.9)
Generato lo scalare p1 (ID 2) = 8.94977e-35
? genr p2 = pvalue(X, 50, 0.9)
Generato lo scalare p2 (ID 3) = 1
? genr test = 1 - p2
Generato lo scalare test (ID 4) = 0
```

La funzione `cdf` ha prodotto un valore accurato, ma la funzione `pvalue` ha dato come risultato 1, da cui non è possibile ricavare il valore della CDF. Questo può sembrare sorprendente, ma si spiega considerando che se il valore di `p1` è corretto, il valore corretto di `p2` è $1-8.94977\times10^{-35}$, ma non c'è modo di rappresentare questo valore con un "double": richiederebbe oltre 30 cifre di precisione.

Ovviamente questo è un esempio estremo. Se il valore di x in questione non si trova alle estremità di una delle due code della distribuzione, le funzioni `cdf` e `pvalue` produrranno risultati complementari, come si vede da questo esempio:

```
? genr p1 = cdf(X, 50, 30)
Generato lo scalare p1 (ID 2) = 0.0111648
? genr p2 = pvalue(X, 50, 30)
Generato lo scalare p2 (ID 3) = 0.988835
? genr test = 1 - p2
Generato lo scalare test (ID 4) = 0.0111648
```

La morale è che se occorre esaminare valori estremi, occorre scegliere attentamente la funzione da usare, tenendo presente che valori molto vicini allo zero possono essere rappresentati con "double", mentre valori molto vicini a 1 possono non esserlo.

7.7 Gestione dei valori mancanti

Sono disponibili quattro funzioni speciali per gestire i valori mancanti. La funzione booleana `missing()` richiede come unico argomento il nome di una variabile e produce una serie con valore 1 per ogni osservazione in cui la variabile indicata ha un valore mancante, 0 altrove (ossia dove la variabile indicata ha un valore valido). La funzione `ok()` è il complemento di `missing`, ossia una scorciatoia per `!missing` (dove `!` è l'operatore booleano NOT). Ad esempio, è possibile contare i valori mancanti della variabile `x` usando

```
genr nmanc_x = sum(missing(x))
```

La funzione `zeromiss()`, che richiede anch'essa come unico argomento il nome di una serie, produce una serie in cui tutti i valori zero sono trasformati in valori mancanti. Occorre usarla con attenzione (di solito non bisogna confondere valori mancanti col valore zero), ma può essere utile in alcuni casi: ad esempio, è possibile determinare la prima osservazione valida di una variabile `x` usando

```
genr time
genr x0 = min(zeromiss(time * ok(x)))
```

La funzione `misszero()` compie l'operazione opposta di `zeromiss`, ossia converte tutti i valori mancanti in zero.

Può essere utile chiarire la propagazione dei valori mancanti all'interno delle formule di `genr`. La regola generale è che nelle operazioni aritmetiche che coinvolgono due variabili, se una delle variabili ha un valore mancante in corrispondenza dell'osservazione t, anche la serie risultante avrà un valore mancante in t. L'unica eccezione a questa regola è la moltiplicazione per zero: zero moltiplicato per un valore mancante produce sempre zero (visto che matematicamente il risultato è zero a prescindere dal valore dell'altro fattore).

7.8 Recupero di variabili interne

Il comando genr fornisce un modo per recuperare vari valori calcolati dal programma nel corso della stima dei modelli o della verifica di ipotesi. Le variabili che possono essere richiamate in questo modo sono elencate nella *Guida ai comandi di gretl*; qui ci occupiamo in particolare delle variabili speciali $test e $pvalue.

Queste variabili contengono, rispettivamente, il valore dell'ultima statistica test calcolata durante l'ultimo uso esplicito di un comando di test e il p-value per quella statistica test. Se non è stato eseguito alcun comando di test, le variabili contengono il codice di valore mancante. I "comandi espliciti di test" che funzionano in questo modo sono i seguenti: add (test congiunto per la significatività di variabili aggiunte a un modello); adf (test di Dickey–Fuller aumentato, si veda oltre); arch (test per ARCH); chow (test Chow per break strutturale); coeffsum (test per la somma dei coefficienti specificati); cusum (statistica *t* di Harvey–Collier); kpss (test di stazionarietà KPSS, p-value non disponibile); lmtest (si veda oltre); meantest (test per la differenza delle medie); omit (test congiunto per la significatività delle variabili omesse da un modello); reset (test RESET di Ramsey); restrict (vincolo lineare generale); runs (test delle successioni per la casualità); testuhat (test per la normalità dei residui) e vartest (test per la differenza delle varianze). Nella maggior parte dei casi, vengono salvati valori sia in $test che in $pvalue; l'eccezione è il test KPSS, per cui non è disponibile il p-value.

Un punto da tenere in considerazione a questo proposito è che le variabili interne $test e $pvalue vengono sovrascritte ogni volta che viene eseguito uno dei test elencati sopra. Se si intende referenziare questi valori durante una sequenza di comandi gretl, occorre farlo nel momento giusto.

Un'altra questione è data dal fatto che alcuni dei comandi di test di solito generano più di una statistica test e più di un p-value: in questi casi vengono salvati solo gli ultimi valori. Per controllare in modo preciso quali valori vengono recuperati da $test e $pvalue occorre formulare il comando di test in modo che il risultato non sia ambiguo. Questa nota vale in particolare per i comandi adf e lmtest.

- Di solito, il comando adf genera tre varianti del test Dickey–Fuller: una basata su una regressione che include una costante, una che include costante e trend lineare, e una che include costante e trend quadratico. Se si intende estrarre valori da $test o $pvalue dopo aver usato questo comando, è possibile selezionare la variante per cui verranno salvati i valori, usando una delle opzioni --nc, --c, --ct o --ctt con il comando adf.
- Di solito, il comando lmtest (che deve seguire una regressione OLS) esegue vari test diagnostici sulla regressione in questione. Per controllare cosa viene salvato in $test e $pvalue occorre limitare il test usando una delle opzioni --logs, --autocorr, --squares o --white.

Un aspetto comodo per l'uso dei valori immagazzinati in $test e $pvalue è dato dal fatto che il tipo di test a cui si riferiscono questi valori viene scritto nell'etichetta descrittiva della variabile generata. Per controllare di aver recuperato il valore corretto, è possibile leggere l'etichetta con il comando label (il cui unico argomento è il nome della variabile). La seguente sessione interattiva illustra la procedura.

```
? adf 4 x1 --c
Test Dickey-Fuller aumentati, ordine 4, per x1
ampiezza campionaria 59
ipotesi nulla di radice unitaria: a = 1
  test con costante
  modello: (1 - L)y = b0 + (a-1)*y(-1) + ... + e
  valore stimato di (a - 1): -0.216889
  statistica test: t = -1.83491
  p-value asintotico 0.3638
P-value basati su MacKinnon (JAE, 1996)
? genr pv = $pvalue
Generato lo scalare pv (ID 13) = 0.363844
```

```
? label pv
  pv=Dickey-Fuller pvalue (scalar)
```

7.9 Procedure numeriche

Esistono due funzioni particolarmente utili per costruire stimatori speciali, ossia `BFGSmax` (il massimizzatore BFGS, discusso nel Capitolo 17) e `fdjac`, che produce un'approssimazione del Jacobiano calcolata col metodo della differenza finita in avanti.

Il massimizzatore BFGS

La funzione `BFGSmax` accetta due argomenti: un vettore che contiene i valori iniziali di un insieme di parametri, e una stringa che specifica una chiamata a una funzione che calcola il criterio (scalare) da massimizzare, dati i valori attuali dei parametri e gli altri dati rilevanti. Se si tratta di una minimizzazione, questa funzione dovrebbe produrre il criterio con segno negativo. In caso di successo, `BFGSmax` produce il valore massimo del criterio e la matrice indicata come primo argomento contiene i valori dei parametri che producono il massimo. Ecco un esempio:

```
matrix X = { dataset }
matrix theta = { 1, 100 }'
scalar J = BFGSmax(theta, "Funzione(&theta, &X)")
```

Si assume che `Funzione` sia una funzione definita dall'utente (si veda il Capitolo 10) con una struttura di questo tipo:

```
function Funzione (matrix *theta, matrix *X)
  scalar val = ...  # Esegue dei calcoli
  return scalar val
end function
```

Esempio 7.2: Ricerca del minimo della funzione di Rosenbrock

```
function Rosenbrock(matrix *param)
  scalar x = param[1]
  scalar y = param[2]
  scalar f = -(1-x)^2 - 100 * (y - x^2)^2
  return scalar f
end function

nulldata 10

matrix theta = { 0 , 0 }

set max_verbose 1
M = BFGSmax(theta, "Rosenbrock(&theta)")

print theta
```

Il funzionamento del massimizzatore BFGS può essere regolato usando il comando `set` sulle variabili `bfgs_maxiter` e `bfgs_toler` (si veda il Capitolo 17). Inoltre, è possibile vedere i dettagli della massimizzazione assegnando un valore positivo alla variabile `max_verbose`, sempre con il comando `set`.

Spesso, per testare gli algoritmi di ottimizzazione si usa la funzione di Rosenbrock, chiamata anche "valle di Rosenbrock" o la "funzione a banana di Rosenbrock", visto che le linee di

contorno sono a forma di banana. Questa è definita come:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

La funzione ha un minimo globale in $(x, y) = (1, 1)$ dove vale $f(x, y) = 0$. L'Esempio 7.2 mostra uno script di gretl che cerca il minimo usando la funzione BFGSmax (mostrando i dettagli sul progresso del calcolo).

Calcolo di un Jacobiano

Gretl offre la possibilità di differenziare numericamente una funzione definita dall'utente, usando la funzione fdjac.

Questa funzione accetta due argomenti: una matrice $n \times 1$ che contiene i valori iniziali dei parametri e una stringa che specifica una chiamata a una funzione che calcola e produce una matrice $m \times 1$, dati i valori attuali dei parametri e gli altri dati rilevanti. In caso di successo, viene prodotta una matrice $m \times n$ che contiene il Jacobiano. Ad esempio,

```
matrix Jac = fdjac(theta, "Somma(&theta, &X)")
```

dove si assume che Somma sia una funzione definita dall'utente con la struttura seguente:

```
function Somma (matrix *theta, matrix *X)
  matrix V = ...  # Esegue dei calcoli
  return matrix V
end function
```

Questo può rivelarsi utile in vari casi: ad esempio, se si usa BFGSmax per stimare un modello, si potrebbe voler calcolare un'approssimazione numerica al Jacobiano rilevante per costruire una matrice di covarianza per le stime.

Un altro esempio è rappresentato dal metodo delta: se si ha uno stimatore consistente di un vettore di parametri $\hat{\theta}$ e una stima consistente della sua matrice di covarianza Σ, potrebbe essere necessario calcolare stime di una trasformazione nonlineare continua $\psi = g(\theta)$. In questo caso, un risultato standard della teoria asintotica è il seguente:

$$\left\{ \begin{array}{c} \hat{\theta} \xrightarrow{\text{p}} \theta \\ \sqrt{T}\left(\hat{\theta} - \theta\right) \xrightarrow{\text{d}} N(0, \Sigma) \end{array} \right\} \Longrightarrow \left\{ \begin{array}{c} \hat{\psi} = g(\hat{\theta}) \xrightarrow{\text{p}} \psi = g(\theta) \\ \sqrt{T}\left(\hat{\psi} - \psi\right) \xrightarrow{\text{d}} N(0, J\Sigma J') \end{array} \right\}$$

dove T è l'ampiezza del campione, mentre J è il Jacobiano $\left.\frac{\partial \psi}{\partial \theta}\right|_{\theta=\hat{\theta}}$.

Lo script 7.3 esemplifica questo caso: è tratto da Greene (2003), sezione 9.3.1. Le leggere differenze tra i risultati riportati nel testo originale e quelli prodotti da gretl sono dovuti al fatto che il Jacobiano è calcolato numericamente, invece che analiticamente come nel testo.

7.10 La trasformata discreta di Fourier

La trasformata discreta di Fourier è una trasformazione lineare invertibile di un vettore complesso. Quindi, se $\mathbf{x}$ è un vettore n-dimensionale il cui k-esimo elemento è $x_k = a_k + ib_k$, il risultato della trasformata discreta di Fourier è un vettore $\mathbf{f} = \mathcal{F}(\mathbf{x})$ il cui k-esimo elemento è

$$f_k = \sum_{j=0}^{n-1} e^{-i\omega_{j,k}} x_j$$

dove $\omega_{j,k} = 2\pi i \frac{jk}{n}$. Poiché la trasformazione è invertibile, il vettore $\mathbf{x}$ può essere ricavato da $\mathbf{f}$ usando la cosidetta trasformata inversa

$$x_k = \frac{1}{n} \sum_{j=0}^{n-1} e^{i\omega_{j,k}} f_j.$$

Esempio 7.3: Metodo delta

```
function MPC(matrix *param, matrix *Y)
  beta = param[2]
  gamma = param[3]
  y = Y[1]
  matrix ret = beta*gamma*y^(gamma-1)
  return matrix ret
end function

# William Greene, Econometric Analysis, 5e, Chapter 9
set echo off
set messages off
open greene5_1.gdt

# Usa OLS per inizializzare i parametri
ols realcons 0 realdpi --quiet
genr a = $coeff(0)
genr b = $coeff(realdpi)
genr g = 1.0

# Esegui NLS con derivate analitiche
nls realcons = a + b * (realdpi^g)
  deriv a = 1
  deriv b = realdpi^g
  deriv g = b * realdpi^g * log(realdpi)
end nls

matrix Y = realdpi[2000:4]
matrix theta = $coeff
matrix V = $vcv

mpc = MPC(&theta, &Y)
matrix Jac = fdjac(theta, "MPC(&theta, &Y)")
Sigma = qform(Jac, V)

printf "\nmpc = %g, std.err = %g\n", mpc, sqrt(Sigma)
scalar teststat = (mpc-1)/sqrt(Sigma)
printf "\nTest per MPC = 1: %g (p-value = %g)\n", \
        teststat, pvalue(n,abs(teststat))
```

La trasformata di Fourier è usata in varie situazioni, grazie a questa sua proprietà fondamentale: la convoluzione di due vettori può essere calcolata in modo efficiente moltiplicando gli elementi delle loro trasformate di Fourier e invertendo il risultato. Se

$$z_k = \sum_{j=1}^{n} x_j y_{k-j},$$

allora vale

$$\mathcal{F}(\mathbf{z}) = \mathcal{F}(\mathbf{x}) \odot \mathcal{F}(\mathbf{y}).$$

Ossia, $\mathcal{F}(\mathbf{z})_k = \mathcal{F}(\mathbf{x})_k \mathcal{F}(\mathbf{y})_k$.

Per calcolare la trasformata di Fourier, gretl usa la libreria esterna `fftw3` (si veda Frigo e Johnson 2003), che garantisce velocità e accuratezza estreme. Infatti il tempo di processore necessario a compiere la trasformazione è $O(n \log n)$ per ogni n. Ecco perché l'insieme di tecniche numeriche impiegate in `fftw3` è chiamato comunemente Trasformata *Veloce* di Fourier.

Gretl fornisce due funzioni matriciali[1] per calcolare la trasformata di Fourier e la sua inversa: `fft` e `ffti`. In realtà l'implementazione della trasformata di Fourier di gretl è un po' più specializzata: il valore di ingresso della funzione `fft` deve essere reale. Al contrario, `ffti` richiede un argomento complesso e produce un risultato reale. Ad esempio:

```
x1 = { 1 ; 2 ; 3 }
# Esegue la trasformazione
f = fft(a)
# Esegue la trasformazione inversa
x2 = ffti(f)
```

produce

$$x_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \qquad f = \begin{bmatrix} 6 & 0 \\ -1.5 & 0.866 \\ -1.5 & -0.866 \end{bmatrix} \qquad x_2 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

dove la prima colonna di f contiene la parte reale, mentre la seconda la parte complessa. In generale, se l'argomento di `fft` ha n colonne, il risultato ne ha $2n$, dove le parti reali sono contenute nelle colonne dispari, mentre le parti complesse in quelle pari. Se fosse necessario calcolare la trasformata di Fourier su molti vettori con lo stesso numero di elementi, è numericamente più efficiente raggrupparli in una matrice, piuttosto che eseguire `fft` separatamente per ogni vettore.

Ad esempio, si consideri la moltiplicazione di due polinomi:

$$\begin{aligned} a(x) &= 1 + 0.5x \\ b(x) &= 1 + 0.3x - 0.8x^2 \\ c(x) = a(x) \cdot b(x) &= 1 + 0.8x - 0.65x^2 - 0.4x^3 \end{aligned}$$

I coefficienti del polinomio $c(x)$ sono la convoluzione dei coefficienti di $a(x)$ e $b(x)$; il seguente codice per gretl illustra come calcolare i coefficienti di $c(x)$:

```
# Definizione dei due polinomi
a = { 1, 0.5, 0, 0 }'
b = { 1, 0.3, -0.8, 0 }'
# Calcolo delle trasformate
fa = fft(a)
fb = fft(b)
# Moltiplicazione complessa delle due trasformate
fc = cmult(fa, fb)
# Calcolo dei coefficienti di c usando la trasformata inversa
c = ffti(fc)
```

[1] Si veda il capitolo 12.

L'efficienza massima si otterrebbe raggruppando `a` e `b` in una matrice. Il vantaggio computazionale nell'esempio appena visto è trascurabile, ma nel caso di un gran numero di righe o colonne, è preferibile procedere nel modo seguente:

```
# Definizione dei due polinomi
a = { 1 ; 0.5; 0 ; 0 }
b = { 1 ; 0.3 ; -0.8 ; 0 }
# Calcolo congiunto delle trasformate
f = fft(a ~ b)
# Moltiplicazione complessa delle due trasformate
fc = cmult(f[,1:2], f[,3:4])
# Calcolo dei coefficienti di c usando la trasformata inversa
c = ffti(fc)
```

In econometria la trasformata di Fourier è usata per lo più nell'analisi delle serie storiche, ad esempio nel calcolo del periodogramma. Lo script 7.4 mostra come calcolare il periodogramma di una serie storica usando la funzione `fft`.

Esempio 7.4: Periodogramma usando la trasformata di Fourier

```
nulldata 50
# Genera un processo AR(1)
series e = normal()
series x = 0
x = 0.9*x(-1) + e
# Calcola il periodogramma
scale = 2*pi*$nobs
X = { x }
F = fft(X)
S = sumr(F.^2)
S = S[2:($nobs/2)+1]/scale
omega = seq(1,($nobs/2))' .* (2*pi/$nobs)
omega = omega ~ S
# Confronto con il comando pergm
pergm x
print omega
```

Capitolo 8

Variabili discrete

Quando una variabile può assumere solo un numero finito, tipicamente basso, di valori, essa si può chiamare *discreta*. Alcuni comandi di gretl si comportano in modo leggermente diverso quando sono usati su variabili discrete; in più, gretl fornisce alcuni comandi che si applicano solo alle variabili discrete. Nello specifico, i comandi `dummify` e `xtab` (si veda oltre) sono disponibili solo per variabili discrete, mentre il comando `freq` (distribuzione di frequenza) produce risultati differenti nel caso di variabili discrete.

8.1 Dichiarazione delle variabili discrete

Gretl usa una semplice euristica per decidere se una variabile deve essere considerata come discreta, ma è anche possibile marcare esplicitamente una variabile come discreta, nel qual caso il controllo automatico non viene eseguito.

L'euristica è la seguente: per prima cosa si controlla se tutti i valori della variabile sono "abbastanza arrotondati", ossia se sono almeno multipli interi di 0.25; se questa condizione è soddisfatta, si controlla se la variabile assume un numero di valori distinti che sia "abbastanza piccolo", per la precisione uguale o minore di 8. Se entrambe le condizioni sono soddisfatte, la variabile viene automaticamente considerata come discreta.

Se invece si vuole marcare esplicitamente una variabile come discreta, ci sono due modi:

1. Dall'interfaccia grafica, selezionare "Variabile, Modifica attributi" dal menù. Apparirà una finestra di dialogo che, se la variabile sembra adatta, contiene la casella "Tratta questa variabile come discreta". La stessa finestra di dialogo può essere richiamata dal menù contestuale (facendo clic col tasto destro su una variabile) o premendo il tasto F2;
2. Dall'interfaccia a riga di comando, usando il comando `discrete`. Il comando accetta uno o più argomenti, che possono essere variabili o liste di variabili. Ad esempio:

   ```
   list xlist = x1 x2 x3
   discrete z1 xlist z2
   ```

   In questo modo è possibile dichiarare più variabili discrete con un solo comando, cosa che al momento non è possibile fare usando l'interfaccia grafica. L'opzione `--reverse` inverte la dichiarazione, ossia rende continua una variabile discreta. Ad esempio:

   ```
   discrete pippo
   # ora pippo è discreta
   discrete pippo --reverse
   # ora piipo è continua
   ```

La variante a riga di comando è più potente, visto che consente di marcare una variabile come discreta anche se non viene giudicata adatta dal programma per questa trasformazione.

Si noti che marcare una variabile come discreta non ne modifica il contenuto. È quindi responsabilità dell'utente usare correttamente questa funzione. Per ricodificare una variabile continua in classi, è possibile usare il comando `genr` e le sue funzioni aritmetiche come nell'esempio seguente:

```
nulldata 100
# genera una variabile con media 2 e varianza 1
genr x = normal() + 2
```

```
# suddivide in 4 classi
genr z = (x>0) + (x>2) + (x>4)
# ora dichiara z come discreta
discrete z
```

Quando si marca una variabile come discreta, questa impostazione viene ricordata dopo il salvataggio del file.

8.2 Comandi per le variabili discrete

Il comando `dummify`

Il comando `dummify` prende come argomento una serie x e crea delle variabili dummy per ognuno dei valori distinti presenti in x, che deve essere stata dichiarata discreta in precedenza. Ad esempio:

```
open greene22_2
discrete Z5 # marca Z5 come discreta
dummify Z5
```

L'effetto di questi comandi è quello di generare 5 nuove variabili dummy, i cui nomi vanno da `DZ5_1` fino a `DZ5_5`, che corrispondono ai diversi valori presenti in `Z5`. Ossia, la variabile `DZ5_4` vale 1 dove `Z5` vale 4, e 0 altrove. Questa funzionalità è disponibile anche nell'interfaccia grafica, con il comando del menù "Aggiungi, dummy per le variabili discrete selezionate".

Il comando `dummify` può essere usato anche con la sintassi seguente:

```
list dlist = dummify(x)
```

In questo modo, vengono create non solo le variabili dummy, ma anche una lista che può essere usata in seguito (si veda la sezione 11.1). L'esempio seguente calcola le statistiche descrittive per la variabile `Y` in corrispondenza di ogni valore di `Z5`:

```
open greene22_2
discrete Z5 # marca Z5 come discreta
list foo = dummify(Z5)
loop foreach i foo
  smpl $i --restrict --replace
  summary Y
end loop
smpl full
```

Poiché `dummify` genera una lista, può essere usato direttamente in comandi che accettano una lista come input, come `ols`. Ad esempio:

```
open greene22_2
discrete Z5 # marca Z5 come discreta
ols Y 0 dummify(Z5)
```

Il comando `freq`

Il comando `freq` mostra le frequenze assolute e relative per una variabile. Il modo in cui le frequenze vengono calcolate dipende dal carattere discreto o continuo della variabile. Questo comando è disponibile anche nell'interfaccia grafica, usando il comando del menù "Variabile, Distribuzione di frequenza".

Per variabili discrete, le frequenze sono contate per ogni diverso valore assunto dalla variabile. Per le variabili continue, i valori sono raggruppati in "classi" e quindi le frequenze sono calcolate per ogni classe. Per impostazione predefinita, il numero di classi è calcolato in funzione del numero di osservazioni valide nel campione selezionato al momento, come mostrato nella Tabella 8.1, ma quando il comando viene invocato attraverso la voce del menù "Variabile, Grafico frequenze", questa impostazione può essere modificata dall'utente.

Osservazioni	Classi
$8 \leq n < 16$	5
$16 \leq n < 50$	7
$50 \leq n \leq 850$	$\lceil \sqrt{n} \rceil$
$n > 850$	29

Tabella 8.1: Numero di classi per varie ampiezze campionarie

Ad esempio, il codice seguente

```
open greene19_1
freq TUCE
discrete TUCE # marca TUCE come discreta
freq TUCE
```

produce questo risultato

```
Lettura del file dati /usr/local/share/gretl/data/greene/greene19_1.gdt
Periodicità: 1, oss. max.: 32,
Intervallo delle osservazioni: 1-32

5 variabili elencate:
  0) const    1) GPA      2) TUCE     3) PSI      4) GRADE

? freq TUCE

Distribuzione di frequenza per TUCE, oss. 1-32
Numero di intervalli = 7, media = 21,9375, scarto quadratico medio = 3,90151

       Intervallo        P.med.  Frequenza    Rel.     Cum.

         <  13,417     12,000        1      3,12%    3,12% *
    13,417 - 16,250    14,833        1      3,12%    6,25% *
    16,250 - 19,083    17,667        6     18,75%   25,00% ******
    19,083 - 21,917    20,500        6     18,75%   43,75% ******
    21,917 - 24,750    23,333        9     28,12%   71,88% **********
    24,750 - 27,583    26,167        7     21,88%   93,75% *******
         >= 27,583     29,000        2      6,25%  100,00% **

Test per l'ipotesi nulla di distribuzione normale:
Chi-quadro(2) = 1,872 con p-value 0,39211

? discrete TUCE # marca TUCE come discreta

? freq TUCE
Distribuzione di frequenza per TUCE, oss. 1-32

          Frequenza    Rel.     Cum.

  12           1      3,12%    3,12% *
  14           1      3,12%    6,25% *
  17           3      9,38%   15,62% ***
  19           3      9,38%   25,00% ***
  20           2      6,25%   31,25% **
  21           4     12,50%   43,75% ****
  22           2      6,25%   50,00% **
  23           4     12,50%   62,50% ****
  24           3      9,38%   71,88% ***
  25           4     12,50%   84,38% ****
  26           2      6,25%   90,62% **
  27           1      3,12%   93,75% *
```

```
  28           1      3,12%   96,88% *
  29           1      3,12%  100,00% *

Test per l'ipotesi nulla di distribuzione normale:
Chi-quadro(2) = 1,872 con p-value 0,39211
```

Come si può vedere dall'esempio, viene calcolato automaticamente un test Doornik–Hansen per la normalità. Il test è soppresso per le variabili discrete che assumono un numero di valori distinti minore di 10.

Questo comando accetta due opzioni: `--quiet`, per evitare la stampa dell'istogramma, e `--gamma`, per sostituire il test di normalità con il test non parametrico di Locke, la cui ipotesi nulla è che i dati seguano una distribuzione Gamma.

Se occorre salvare i valori distinti di una variabile discreta, è possibile usare il comando matriciale `values()` (si veda il capitolo 12).

Il comando `xtab`

Il comando `xtab` può essere invocato in uno dei due modi seguenti. Per primo:

```
xtab lista-y ; lista-x
```

dove `lista-y` e `lista-x` sono liste di variabili discrete; esso produce tabulazioni incrociate per ognuna delle variabili nella `lista-y` (per riga) rispetto a ognuna delle variabili nella `lista-x` (per colonna). In secondo luogo:

```
xtab lista-x
```

In questo secondo caso, viene generata una tabulazione incrociata completa, ossia ogni variabile nella `lista-x` è tabulata rispetto ad ogni altra variabile. Nell'interfaccia grafica, questo comando è utilizzabile attraverso la voce "Tabulazione incrociata" nel menù Visualizza, che è attiva se sono state selezionate almeno due variabili.

Ecco un esempio di uso:

```
open greene22_2
discrete Z* # Marca Z1-Z8 come discrete
xtab Z1 Z4 ; Z5 Z6
```

che produce questo risultato

```
Tabulazione incrociata di Z1 (righe) rispetto a Z5 (colonne)

       [   1][   2][   3][   4][   5]  TOT.

[   0]    20    91    75    93    36    315
[   1]    28    73    54    97    34    286

TOTALE    48   164   129   190    70    601

Test chi-quadro di Pearson = 5,48233 (4 df, p-value = 0,241287)

Tabulazione incrociata di Z1 (righe) rispetto a Z6 (colonne)

       [   9][  12][  14][  16][  17][  18][  20]  TOT.

[   0]     4    36   106    70    52    45     2    315
[   1]     3     8    48    45    37    67    78    286

TOTALE     7    44   154   115    89   112    80    601
```

```
Test chi-quadro di Pearson = 123,177 (6 df, p-value = 3,50375e-24)

Tabulazione incrociata di Z4 (righe) rispetto a Z5 (colonne)

       [   1][   2][   3][   4][   5]  TOT.

[   0]    17    60    35    45    14    171
[   1]    31   104    94   145    56    430

TOTALE    48   164   129   190    70    601

Test chi-quadro di Pearson = 11,1615 (4 df, p-value = 0,0248074)

Tabulazione incrociata di Z4 (righe) rispetto a Z6 (colonne)

       [   9][  12][  14][  16][  17][  18][  20]  TOT.

[   0]     1     8    39    47    30    32    14    171
[   1]     6    36   115    68    59    80    66    430

TOTALE     7    44   154   115    89   112    80    601

Test chi-quadro di Pearson = 18,3426 (6 df, p-value = 0,0054306)
```

Il test χ^2 di Pearson per l'indipendenza viene mostrato automaticamente, a patto che tutte le celle abbiano frequenze attese nell'ipotesi di indipendenza pari almeno a 10^{-7}. Poiché spesso si considera valida questa statistica se la frequenza attesa nell'ipotesi di indipendenza supera 5 per almeno l'80 per cento delle celle, se questa condizione non è soddisfatta, viene mostrato un messaggio di avvertimento.

Inoltre, le opzioni `--row` o `--column` fanno in modo che vengano mostrate le percentuali di riga o di colonna.

Se si vuole incollare il risultato di `xtab` in qualche altra applicazione, ad esempio un foglio di calcolo, è utile usare l'opzione `--zeros`, che scrive il numero zero nelle celle con frequenza pari a zero, invece di lasciarle vuote.

Capitolo 9

Costrutti loop

9.1 Introduzione

Il comando `loop` permette di specificare un blocco di comandi gretl da ripetere più volte. Questa funzionalità è utile in particolare per le simulazioni Monte Carlo, per il bootstrapping delle statistiche test e per altre procedure di stima iterativa. La forma generale di un loop, o ciclo, è:

```
loop espressione di controllo [ --progressive | --verbose | --quiet ]
   corpo del loop
endloop
```

Sono disponibili cinque tipi di espressione di controllo, come si vede nella sezione 9.2.

Non tutti i comandi di gretl sono disponibili all'interno di un loop: i comandi che non sono accettabili in questo contesto sono mostrati nella Tabella 9.1.

Tabella 9.1: Comandi non utilizzabili in un loop

function include nulldata run setmiss

In modalità predefinita, il comando `genr` all'interno di un loop opera in modo silenzioso (senza mostrare informazioni sulle variabili generate). Per ottenere informazioni da `genr` è possibile specificare l'opzione `--verbose` del comando `loop`. L'opzione `--quiet` sopprime il consueto riepilogo del numero di iterazioni eseguite, cosa desiderabile quando i loop sono annidati.

L'opzione `--progressive` del comando `loop` modifica il comportamento dei comandi `print`, `store` e di alcuni comandi di stima, rendendoli più comodi per l'uso in analisi di tipo Monte Carlo (si veda la sezione 9.3).

Le sezioni che seguono descrivono le varie forme di espressioni di controllo e forniscono alcuni esempi di uso dei loop.

☞ Se occorre eseguire un'analisi Monte Carlo con molte migliaia di ripetizioni, possono verificarsi problemi di memoria e di tempo di calcolo. Un modo per minimizzare l'uso delle risorse di sistema consiste nell'eseguire lo script usando il programma a riga di comando, gretlcli, redirigendo i risultati su un file.

9.2 Varianti di controllo del loop

Loop limitati

Il modo più semplice di controllare un loop consiste nello specificare direttamente il numero di volte che il ciclo deve essere ripetuto, in un cosiddetto "loop limitato". Il numero di ripetizioni può essere una costante numerica, ad esempio `loop 1000`, o può essere letto da una variabile, come in `loop volte`.

Se il numero delle ripetizioni è indicato da una variabile, ad es. `volte`, la variabile dovrebbe essere uno scalare intero: se si usa una serie, viene letto il primo valore, e se questo non è intero, viene troncata la sua parte decimale. Si noti che `volte` viene valutata solo una volta, quando il loop viene impostato.

Loop di tipo while

Un secondo tipo di espressione di controllo consiste nell'uso del comando `while` seguito da una disuguaglianza, in cui il termine a sinistra è il nome di una variabile predefinita, mentre il lato destro può essere una costante numerica o il nome di un'altra variabile predefinita. Ad esempio:

```
loop while differ > .00001
```

L'esecuzione del ciclo di comandi continuerà fintanto che: a) la condizione specificata rimane vera, e b) il numero di iterazioni non eccede il valore contenuto nella variabile interna `loop_maxiter`; il valore predefinito è pari a 250, ma è possibile specificare un valore diverso usando il comando `set` (si veda la *Guida ai comandi di gretl*).

Se il termine a destra della disuguaglianza è una variabile, essa viene valutata all'inizio di ogni nuova iterazione del ciclo.

Loop con indice

Un terzo tipo di controllo di un loop utilizza la speciale variabile indice `i`. In questo caso, vengono specificati valori iniziali e finali per `i`, che viene incrementata di uno ogni volta che viene eseguito il ciclo. La sintassi è la seguente: `loop i=1..20`.

La variabile indice può essere usata all'interno del corpo del loop, in uno dei modi seguenti: è possibile accedere al valore di `i` (si veda l'esempio 9.4), oppure è possibile usare la sua rappresentazione come stringa `$i` (si veda l'esempio 9.5).

I valori iniziale e finale per l'indice possono essere indicati in forma numerica, o come riferimento a variabili predefinite. Nell'ultimo caso, le variabili vengono valutate una volta, quando il loop viene impostato. Inoltre, con dataset di serie storiche è possibile indicare i valori iniziale e finale sotto forma di date, ad esempio: `loop i=1950:1..1999:4`.

Questo tipo di loop è particolarmente utile usato insieme alla funzione matriciale `values()` quando occorre compiere un'operazione per ciascun valore di una variabile discreta (si veda il capitolo 8). Si consideri l'esempio seguente:

```
open greene22_2
open greene22_2
discrete Z8
v8 = values(Z8)
n = rows(v8)
n = rows(v8)
loop for i=1..n
  scalar xi = v8[$i]
  smpl (Z8=xi) --restrict --replace
  printf "mean(Y | Z8 = %g) = %8.5f, sd(Y | Z8 = %g) = %g\n", \
    xi, mean(Y), xi, sd(Y)
end loop
```

In questo caso valutiamo la media condizionale e lo scarto quadratico medio della variabile `Y` per ciascun valore della variabile `Z8`.

Loop di tipo foreach

Anche il terzo tipo di controllo usa la variabile interna `i`, che in questo caso può assumere valori solo all'interno di una lista specifica di stringhe. Il loop è eseguito una volta per ogni stringa presente nella lista, agevolando l'esecuzione di operazioni ripetitive su un gruppo di variabili. Ecco un esempio:

```
loop foreach i mele pere pesche
  print "$i"
endloop
```

Questo loop verrà eseguito tre volte, mostrando “mele”, “pere” e “pesche” ad ogni iterazione.

Per eseguire un loop su una lista di variabili contigue nel dataset, è possibile indicare i nomi della prima e dell’ultima variabile nella lista, separate da “`..`”, invece di dover indicare tutti i nomi. Ad esempio, ipotizzando di avere 50 variabili `AK`, `AL`, ..., `WY`, che contengono livelli di reddito per gli stati degli USA, per stimare una regressione del reddito sul tempo per ognuno degli stati, basta eseguire:

```
genr time
loop foreach i AL..WY
   ols $i const time
endloop
```

Questo tipo di loop può essere usato anche per costruire cicli sugli elementi di una *lista definita dall’utente* (si veda il capitolo 11). Ad esempio:

```
list ylist = y1 y2 y3
loop foreach i ylist
   ols $i const x1 x2
endloop
```

Loop di tipo for

L’ultimo tipo di controllo usa una forma semplificata dell’istruzione `for` del linguaggio di programmazione C. L’espressione di controllo si compone di tre parti, separate da punto e virgola. La prima parte specifica una condizione iniziale, espressa per mezzo di una variabile di controllo; la seconda parte imposta una condizione di continuazione (espressa in funzione della stessa variabile di controllo), mentre la terza parte specifica un incremento (o un decremento) per la variabile di controllo, da applicare ogni volta che il ciclo viene eseguito. L’intera espressione deve essere racchiusa tra parentesi. Ad esempio:

```
loop for (r=0.01; r<.991; r+=.01)
```

La variabile `r` assumerà i valori 0.01, 0.02, ..., 0.99 nel giro di 99 iterazioni. Si noti che a causa della precisione limitata dell’aritmetica in virgola mobile usata dal computer, può dover essere necessario usare una condizione di continuazione come quella mostrata sopra, `r<.991`, invece della più “naturale” `r<=.99` (usando numeri in doppia precisione su un processore x86, quando ci si aspetta che `r` valga 0.99, potrebbe in realtà valere 0.990000000000001).

Esistono altre regole per i tre componenti dell’espressione di controllo:

1. La condizione iniziale deve avere forma X1 = Y1, dove Y1 deve essere una costante numerica o una variabile predefinita. Se la variabile X1 non esiste, viene creata automaticamente.
2. La condizione di continuazione deve avere la forma X1 *operatore* Y2, dove l’*operatore* può essere <, >, <= o >= e Y2 deve essere una costante numerica o una variabile predefinita (nel caso in cui sia una variabile, essa viene valutata ad ogni esecuzione del ciclo).
3. L’espressione che indica l’incremento o il decremento deve avere la forma X1 += DELTA, oppure X1 -= DELTA, dove DELTA è una costante numerica o una variabile predefinita (nel secondo caso, essa viene valutata solo una volta, quando il loop viene impostato)

9.3 La modalità progressiva

Usando l’opzione `--progressive` nel comando loop, viene abilitata una modalità speciale in alcuni comandi, ossia in `print`, `store` e nei comandi di stima “semplice”, ossia i comandi che: (a) stimano una sola equazione (non un sistema di equazioni), e (b) utilizzano una sola dichiarazione (al contrario di comandi che usano blocchi di dichiarazioni, come `nls` e `mle`). L’esempio classico è il comando `ols`; altri esempi sono `tsls`, `wls`, `logit` e così via.

La modalità speciale ha questi effetti:

Comandi di stima: i risultati di ogni iterazione della stima non vengono mostrati. Al contrario, una volta terminato il loop si ottiene un elenco dei seguenti valori: (a) il valore medio di ognuno dei coefficienti stimati, calcolato su tutte le iterazioni; (b) lo scarto quadratico medio relativa a questa media; (c) il valore medio dell'errore standard stimato per ogni coefficiente; (d) lo scarto quadratico medio degli errori standard stimati. Tutto ciò ha senso solo se ogni iterazione del loop contiene un elemento di casualità.

`print`: se si usa questo comando per mostrare il valore di una variabile, questo non viene mostrato ad ogni iterazione. Al contrario, alla fine del loop si ottiene il valore medio e lo scarto quadratico medio della variabile, calcolata su tutte le iterazioni del ciclo. Questa funzione è utile per le variabili che assumono un singolo valore per ogni iterazione, ad esempio la somma dei quadrati degli errori di una regressione.

`store`: questo comando scrive i valori delle variabili specificate, ad ogni iterazione del loop, nel file indicato, ossia, tiene traccia completa del valore delle variabili in tutte le iterazioni. Ad esempio, si potrebbero salvare le stime dei coefficienti per poi studiarne la distribuzione di frequenza. È possibile usare il comando `store` solo una volta all'interno di un loop.

9.4 Esempi di loop

Esempio di procedura Monte Carlo

Un semplice esempio di uso della modalità "progressiva" per realizzare una procedura Monte Carlo è mostrato in esempio 9.1.

Esempio 9.1: Un semplice loop di tipo Monte Carlo

```
nulldata 50
seed 547
genr x = 100 * uniform()
# Apre un loop "progressivo", da ripetere 100 volte
loop 100 --progressive
   genr u = 10 * normal()
   # Costruisce la variabile dipendente
   genr y = 10*x + u
   # Esegue la regressione OLS
   ols y const x
   # Definisce variabili per i coefficienti e R-quadro
   genr a = $coeff(const)
   genr b = $coeff(x)
   genr r2 = $rsq
   # Mostra le statistiche su queste variabili
   print a b r2
   # Salva i coefficienti in un file
   store coeffs.gdt a b
endloop
```

Questo loop mostrerà le statistiche di riepilogo per le stime di "a", "b" e R^2 lungo le 100 iterazioni. Dopo aver eseguito il loop, è possibile aprire con gretl il file `coeffs.gdt`, che contiene le stime dei singoli coefficienti durante tutte le iterazioni, ed esaminare nel dettaglio la distribuzione di frequenza delle stime.

Il comando `nulldata` è utile per le procedure Monte Carlo: invece di aprire un "vero" dataset, `nulldata 50` (ad esempio) apre un finto dataset da 50 osservazioni, che contiene solo la costante e una variabile indice. Successivamente è possibile aggiungervi variabili usando il comando `genr`. Si veda il comando `set` per informazioni su come generare numeri pseudo-casuali in modo ripetibile.

Minimi quadrati iterati

L'esempio 9.2 usa un loop di tipo "while" per replicare la stima di una funzione di consumo non lineare nella forma

$$C = \alpha + \beta Y^{\gamma} + \epsilon$$

presentata in Greene (2000, Esempio 11.3). Questo script è compreso nella distribuzione di gretl con il nome `greene11_3.inp`; è possibile aprirlo usando il comando del menù "File, Comandi, File di esempio, Greene...".

L'opzione `--print-final` per il comando `ols` fa sì che non vengano mostrati i risultati della regressione per ogni iterazione, ma solo quelli dell'ultima iterazione del loop.

Esempio 9.2: Funzione di consumo non lineare

```
open greene11_3.gdt
# Esegue la regressione OLS iniziale
ols C 0 Y
genr essbak = $ess
genr essdiff = 1
genr beta = $coeff(Y)
genr gamma = 1
# Itera OLS finché la somma dei quadrati degli errori converge
loop while essdiff > .00001
   # Genera le variabili linearizzate
   genr C0 = C + gamma * beta * Y^gamma * log(Y)
   genr x1 = Y^gamma
   genr x2 = beta * Y^gamma * log(Y)
   # Esegue la regressione OLS
   ols C0 0 x1 x2 --print-final --no-df-corr --vcv
   genr beta = $coeff(x1)
   genr gamma = $coeff(x2)
   genr ess = $ess
   genr essdiff = abs(ess - essbak)/essbak
   genr essbak = ess
endloop
# Mostra le stime dei parametri usando i "nomi giusti"
noecho
printf "alfa = %g\n", $coeff(0)
printf "beta  = %g\n", beta
printf "gamma = %g\n", gamma
```

L'esempio 9.3 mostra come sia possibile usare un loop per stimare un modello ARMA usando la regressione "prodotto esterno del gradiente" (OPG - "outer product of the gradient") discussa da Davidson e MacKinnon nel loro *Estimation and Inference in Econometrics*.

Esempi di loop con indice

L'esempio 9.4 mostra un loop con indice, in cui il comando `smpl` contiene la variabile indice `i`. Si supponga di avere un dataset di tipo panel, con osservazioni su alcuni ospedali per gli anni dal 1991 al 2000 (dove l'anno dell'osservazione è indicato da una variabile chiamata `anno`). Ad ogni iterazione, restringiamo il campione a un certo anno e calcoliamo statistiche di riepilogo sulla dimensione cross-section per le variabili da 1 a 4.

L'esempio 9.5 illustra un loop indicizzato per sostituire stringhe.

Alla prima iterazione, la variabile `V` verrà impostata a `COMP1987` e la variabile dipendente per il comando `ols` sarà `PBT1987`. All'iterazione successiva, `V` verrà ridefinita come `COMP1988` e la variabile dipendente della regressione sarà `PBT1988`, e così via.

Esempio 9.3: ARMA 1, 1

```
open armaloop.gdt

genr c = 0
genr a = 0.1
genr m = 0.1

series e = 1.0
genr de_c = e
genr de_a = e
genr de_m = e

genr crit = 1
loop while crit > 1.0e-9

   # Errori di previsione "one-step"
   genr e = y - c - a*y(-1) - m*e(-1)

   # Log-verosimiglianza
   genr loglik = -0.5 * sum(e^2)
   print loglik

   # Derivate parziali degli errori di previsione rispetto a c, a e m
   genr de_c = -1 - m * de_c(-1)
   genr de_a = -y(-1) -m * de_a(-1)
   genr de_m = -e(-1) -m * de_m(-1)

   # Derivate parziali di l rispetto a c, a e m
   genr sc_c = -de_c * e
   genr sc_a = -de_a * e
   genr sc_m = -de_m * e

   # Regressione OPG
   ols const sc_c sc_a sc_m --print-final --no-df-corr --vcv

   # Aggiorna i parametri
   genr dc = $coeff(sc_c)
   genr c = c + dc
   genr da = $coeff(sc_a)
   genr a = a + da
   genr dm = $coeff(sc_m)
   genr m = m + dm

   printf "  constant        = %.8g (gradient = %#.6g)\n", c, dc
   printf "  ar1 coefficient = %.8g (gradient = %#.6g)\n", a, da
   printf "  ma1 coefficient = %.8g (gradient = %#.6g)\n", m, dm

   genr crit = $T - $ess
   print crit
endloop

genr se_c = $stderr(sc_c)
genr se_a = $stderr(sc_a)
genr se_m = $stderr(sc_m)

noecho
print "
printf "constant = %.8g (se = %#.6g, t = %.4f)\n", c, se_c, c/se_c
printf "ar1 term = %.8g (se = %#.6g, t = %.4f)\n", a, se_a, a/se_a
printf "ma1 term = %.8g (se = %#.6g, t = %.4f)\n", m, se_m, m/se_m
```

Esempio 9.4: Statistiche panel

```
open ospedali.gdt
loop i=1991..2000
  smpl (anno=i) --restrict --replace
  summary 1 2 3 4
endloop
```

Esempio 9.5: Sostituzione di stringhe

```
open bea.dat
loop i=1987..2001
  genr V = COMP$i
  genr TC = GOC$i - PBT$i
  genr C = TC - V
  ols PBT$i const TC V
endloop
```

Capitolo 10

Funzioni definite dall'utente

10.1 Definizione di una funzione

A partire dalla versione 1.3.3, gretl contiene un meccanismo per definire funzioni all'interno di uno script. Questa funzionalità ha subito alcune modifiche prima di raggiungere un assetto stabile ed estensibile, ma pensiamo che la versione presente in gretl 1.6.1 costituisca una solida base per gli sviluppi futuri.

Occorre definire una funzione prima di poterla utilizzare. La sintassi per farlo è la seguente:

```
function nome-funzione(parametri)
   corpo della funzione
end function
```

Il *nome-funzione* identifica la funzione in modo univoco: deve iniziare con una lettera, può essere lungo al massimo 31 caratteri (eventuali caratteri in più verranno troncati) e non può contenere spazi. Se si tenta di definire una funzione con lo stesso nome di un comando di gretl.

I *parametri* di una funzione vanno indicati sotto forma di lista separata da virgole. I parametri possono essere di uno dei seguenti tipi:

Tipo	Descrizione
`bool`	variabile scalare usata come interruttore Booleano
`int`	variabile scalare usata come numero intero
`scalar`	variabile scalare
`series`	serie di dati
`list`	lista di serie
`matrix`	matrice o vettore
`string`	variabile stringa o stringa letterale

Ogni elemento della lista di parametri deve includere due termini: per prima cosa un indicatore di tipo, quindi il nome con cui il parametro verrà riconosciuto all'interno della funzione. Ecco un esempio:

```
function funzione(series y, list xvars, bool verboso)
```

Ognuno di questi indicatori di tipo, con l'eccezione di `list` e `string`, può essere modificato facendo precedere da un asterisco il parametro ad esso associato, come in questo esempio

```
function funzione(series *y, scalar *b)
```

Il significato di questa modifica è spiegato nella sezione 10.4 e ha a che fare con l'uso dei puntatori nel linguaggio di programmazione C. Inoltre, i parametri possono essere modificati con il tag `const` (anche in questo caso, si veda la sezione 10.4).

Oltre a questi elementi richiesti, la specificazione di un parametro `scalare` può includere fino a tre ulteriori informazioni: un valore minimo, uno massimo e un valore predefinito. Questi valori aggiuntivi devono seguire direttamente il nome del parametro, devono essere racchiusi tra parentesi quadre e i singoli elementi devono essere separati dal carattere due punti. Ad esempio, ipotizzando di avere un parametro intero chiamato `ordine` per cui si vuole specificare un valore minimo di 1, un massimo di 12 e un valore predefinito di 4, si può scrivere

```
int ordine[1:12:4]
```

Per omettere uno dei tre valori aggiuntivi, basta lasciare vuoto il campo corrispondente. Ad esempio, `[1::4]` specifica un minimo di 1 e un valore predefinito di 4, senza porre limiti al valore massimo.

Per parametri di tipo `bool` è possibile specificare come valori predefiniti 1 (vero) o 0 (falso), come in questo esempio:

```
bool verboso[0]
```

È possibile definire funzioni che non hanno parametri (quelle che in alcuni linguaggi di programmazione vengono chiamate "routine"). In questo caso, occorre usare la parola chiave `void` al posto dell'elenco dei parametri:

```
function funzione2(void)
```

Quando una funzione viene chiamata, i parametri vengono istanziati usando gli argomenti indicati nella chiamata della funzione. Vengono fatti dei controlli automatici per assicurarsi che il numero degli argomenti contenuti in una chiamata di funzione corrisponda al numero di parametri, e che i tipi degli argomenti corrispondano ai tipi specificati nella definizione della funzione; se qualcuna di queste condizioni è violata, viene segnalato un errore. Una precisazione: è consentito omettere degli argomenti alla fine della lista, a patto che i valori predefiniti siano specificati nella definizione della funzione. Più precisamente: il controllo consiste nell'assicurarsi che il numero degli argomenti sia almeno uguale al numero dei parametri *richiesti* e non superiore al numero totale dei parametri.

È possibile indicare uno scalare, una serie, o una matrice come argomento di una funzione, sia specificando il nome di una variabile preesistente, oppure utilizzando un'espressione che, valutata, restituisce una variabile del tipo appropriato. Gli scalari possono essere indicati anche sotto forma di valori numerici, mentre le liste devono essere indicate per nome.

Il *corpo della funzione* è composto da comandi `gretl` o funzioni definite dall'utente (ossia, le funzioni possono essere nidificate). Una funzione può chiamare sé stessa (ossia, le funzioni possono essere ricorsive). Se il corpo della funzione può contenere chiamate ad altre funzioni non può però contenere definizioni di altre funzioni, ossia non è possibile definire una funzione all'interno di un'altra funzione.

10.2 Chiamata di una funzione

Una funzione utente viene chiamata, o invocata, usando il suo nome, eventualmente seguito da argomenti tra parentesi; se si usano due o più argomenti, vanno separati da virgole. L'esempio seguente mostra una chiamata di funzione che rispetta la definizione della funzione stessa.

```
# Definizione della funzione
function ols_ess(series y, list xvars)
  ols y 0 xvars --quiet
  scalar myess = $ess
  printf "ESS = %g\n", myess
  return scalar myess
end function
# Script principale
open data4-1
list xlist = 2 3 4
# Chiamata della funzione (il valore restituito viene qui ignorato)
ols_ess(price, xlist)
```

La chiamata della funzione contiene due argomenti: il primo è una serie di dati specificata per nome, e il secondo è una lista di regressori. Si noti che la funzione produce la variabile `myess` come risultato, ma in questo esempio esso è ignorato. Una nota a margine: se si desidera

che una funzione calcoli alcuni valori che hanno a che fare con una regressione, ma non si è interessati ai risultati completi della regressione, è possibile usare l'opzione `--quiet` con il comando di stima, come visto sopra.

Un secondo esempio mostra una chiamata di funzione che assegna il valore prodotto dalla funzione ad alcune variabili:

```
# Definizione di funzione
function get_uhat(series y, list xvars)
  ols y 0 xvars --quiet
  series uh = $uhat
  return series uh
end function
# Script principale
open data4-1
list xlist = 2 3 4
# Chiamata di funzione
series resids = get_uhat(price, xlist)
```

10.3 Cancellazione di una funzione

Se si vuole cancellare dalla memoria una funzione definita in precedenza, è possibile usare le parole chiave `delete` o `clear`, come in

```
function myfunc delete
function get_uhat clear
```

Si noti che ridefinendo una funzione, la vecchia funzione viene automaticamente sovrascritta, quindi raramente è necessario cancellare una funzione in modo esplicito.

10.4 Programmazione delle funzioni

Variabili o puntatori

Le serie, gli scalari, e le matrici che sono argomenti di una funzione possono essere passati alla funzione in due modi: "così come sono", oppure come puntatori. Si consideri l'esempio seguente:

```
function triplo1(series x)
  series ret = 3*x
  return series ret
end function

function triplo2(series *x)
  series ret = 3*x
  return series ret
end function
```

Queste due funzioni sono quasi identiche (e producono lo stesso risultato): l'unica differenza sta nel fatto che `triplo1` richiede una serie come argomento, come in `triplo1(serie)`, mentre `triplo2` richiede un *puntatore* a una serie, come in `triplo2(&serie)`.

Perché questa distinzione? Ci sono due motivi principali: il più importante è la modularità, il secondo le prestazioni.

L'idea di modularità nasce dal fatto che isolare le funzioni dal resto di uno script è di solito una buona idea. Uno dei tanti benefici di questo approccio consiste nel fatto che le funzioni sono facilmente riutilizzabili in altri contesti. Per ottenere la modularità, *le variabili create in una funzione sono locali a quella funzione, e vengono distrutte quando la funzione termina la sua esecuzione*, a meno che esse siano rese disponibili come valori di ritorno, e che questi siano "raccolti" o assegnati nella chiamata della funzione.

Inoltre, le funzioni non hanno accesso alle variabili dell'"ambiente esterno" (ossia le variabili che esistono nello script da cui la funzione è chiamata), a meno che queste siano passate esplicitamente alla funzione come argomenti.

Nel caso predefinito, quando una variabile viene passata a una funzione come argomento, la funzione ottiene una *copia* della variabile esterna, quindi il valore della variabile nell'ambiente esterno non viene modificato dalle operazioni che avvengono all'interno della funzione. Invece, l'uso dei puntatori consente a una funzione e all'ambiente di "cooperare" in modo che una variabile esterna possa essere modificata dalla funzione. Questo meccanismo consente a una funzione di restituire più di un valore (anche se una funzione può restituire direttamente al massimo una variabile, come spiegato in seguito). Il parametro in questione viene contrassegnato col prefisso * nella definizione della funzione, mentre l'argomento corrispondente viene contrassegnato col prefisso complementare & nella chiamata. Ad esempio:

```
function get_uhat_and_ess(series y, list xvars, scalar *ess)
  ols y 0 xvars --quiet
  ess = $ess
  series uh = $uhat
  return series uh
end function
# Script principale
open data4-1
list xlist = 2 3 4
# Chiamata di funzione
scalar SSR
series resid = get_uhat_and_ess(price, xlist, &SSR)
```

In questo caso, alla funzione viene passato l'*indirizzo* della variabile scalare `SSR` a cui viene assegnato un valore (usando il nome `ess`). Per chi ha familiarità con la programmazione in C, si noti che non è necessario (né possibile) "dereferenziare" la variabile in questione nella funzione usando l'operatore *. Per accedere al contenuto della variabile nell'ambiente esterno è sufficiente usare il nome della variabile senza prefissi.

Un parametro di "indirizzo" di questo tipo può essere utile per offrire informazioni opzionali alla chiamata (ossia, l'argomento corrispondente non è strettamente necessario, ma sarà usato se presente). In questo caso, al parametro andrà assegnato un valore predefinito `null` e la funzione dovrebbe controllare se alla chiamata è stato fornito un argomento corrispondente, usando la funzione `isnull()`. Ad esempio, ecco la funzione vista in precedenza, modificata in modo da rendere opzionale l'indicazione del valore di `ess`.

```
function get_uhat_and_ess(series y, list xvars, scalar *ess[null])
  ols y 0 xvars --quiet
  if !isnull(ess)
     ess = $ess
  endif
  series uh = $uhat
  return series uh
end function
```

Se alla chiamata non si ha interesse per il valore di `ess`, occorre usare `null` al posto di un vero argomento:

```
series resid = get_uhat_and_ess(price, xlist, null)
```

Usare i puntatori può essere utile anche per ottimizzare le prestazioni: anche se una variabile non viene modificata all'interno della funzione, può essere una buona idea passarla come puntatore se occupa molta memoria. Altrimenti, il tempo impiegato da gretl per trascrivere il valore della variabile nella copia locale potrebbe influire sostanzialmente sul tempo di esecuzione dell'intera funzione.

L'esempio 10.1 mostra un caso estremo. Definiamo due funzioni che producono il numero di righe di una matrice (un'operazione abbastanza veloce). La prima funzione accetta la matrice

come argomento, la seconda come puntatore alla matrice. Valutiamo le due funzioni usando una matrice con 2000 righe e 2000 colonne; su un sistema tipico, i valori a virgola mobile occupano 8 byte di memoria, quindi lo spazio occupato dalla matrice è di circa 32 megabyte.

Eseguendo il codice dell'esempio 10.1 verrà prodotto un risultato simile al seguente (i numeri esatti dipenderano dalla macchina su cui si esegue il codice):

```
  Tempo impiegato:
          senza puntatori (copia) = 3.66 secondi,
          con puntatori (non copia) = 0.01 secondi.
```

Esempio 10.1: Confronto di prestazioni: valori contro puntatori

```
function a(matrix X)
  r = rows(X)
  return scalar r
end function

function b(matrix *X)
  r = rows(X)
  return scalar r
end function

nulldata 10
set echo off
set messages off
X = zeros(2000,2000)
r = 0

set stopwatch
loop 100
  r = a(X)
end loop
fa = $stopwatch

set stopwatch
loop 100
  r = b(&X)
end loop
fb = $stopwatch

printf "Tempo impiegato:\n\
\tsenza puntatori (copia) = %g secondi,\n\
\tcon puntatori (non copia) = %g secondi.\n", fa, fb
```

Argomenti lista

L'uso di una lista come argomento di una funzione dà modo di fornire a una funzione un gruppo di argomenti il cui numero non è noto al momento della scrittura della funzione, ad esempio un insieme di regressori o di strumenti. All'interno della funzione, la lista può essere passata a comandi come `ols`, oppure può essere "espansa" usando un costrutto loop `foreach`. Ad esempio, si supponga di avere una lista `X` e di voler calcolare lo scarto quadratico medio di ogni variabile della lista:

```
  loop foreach i X
     scalar sd_$i = sd($i)
  end loop
```

Quando a una funzione viene passata una lista di variabili, in realtà riceve una copia della lista; al contrario, le variabili referenziate dalla lista sono direttamente accessibili dalla funzione, in modo simile a quando si passa uno scalare o una serie come "puntatore", come descritto in precedenza. Passare una lista a una funzione è quindi un altro modo per consentire a una funzione di modificare i dati quando questa viene chiamata, piuttosto che produrre semplicemente un valore di uscita. Se le variabili *non* verranno modificate nella funzione, è una buona idea sottolineare questo fatto usando l'identificatore `const` nella lista dei parametri, come in questo esempio:

```
function funzione (scalar y, const list X)
```

Quando una lista è marcata come `const`, ogni tentativo di rinominare, cancellare o sovrascrivere i valori originali delle variabili nella lista produrrà un errore.

Se un argomento lista per una funzione è opzionale, esso va indicato facendolo seguire da un valore predefinito `null`, nel modo seguente:

```
function myfunc (scalar y, list X[null])
```

In questo caso, se la chiamata passa `null` come argomento di lista, la lista `X` all'interno della funzione sarà vuota. Questa condizione può essere riconosciuta usando la funzione `nelem()`, che restituisce il valore 0 se una lista è vuota (questo meccanismo può essere usato anche per controllare se è stata fornita una lista vuota come argomento).

Argomenti stringa

È possibile usare stringhe come argomenti, ad esempio per gestire in modo flessibile i nomi di variabili create all'interno di una funzione. Nell'esempio che segue, la funzione `movavg` produce una lista che contiene due medie mobili costruite a partire da una serie iniziale, con i nomi delle nuove variabili formati usando l'argomento stringa.

```
function movavg (series y, string vname)
   series @vname_2 = (y+y(-1)) / 2
   series @vname_4 = (y+y(-1)+y(-2)+y(-3)) / 4
   list retlist = @vname_2 @vname_4
   return list retlist
end function

open data9-9
list malist = movavg(nocars, "nocars")
print malist --byobs
```

L'ultima riga dello script stamperà due variabili chiamate `nocars_2` e `nocars_4`. Per i dettagli sulla gestione delle stringhe, si veda il capitolo 11.

Recuperare il nome degli argomenti

Le variabili passate come argomenti a una funzione vengono chiamate all'interno della funzione col nome dei parametri corrispondenti. Ad esempio, all'interno di questa funzione:

```
function somefun (series y)
```

abbiamo la serie chiamata `y`. In alcuni casi può però essere utile disporre del nome delle variabili che sono state passate come argomenti. Questo risultato si ottiene usando la funzione `argname`, che accetta come unico argomento il nome di un parametro della funzione e restituisce una stringa. Ecco un esempio:

```
function namefun (series y)
  printf "La serie passata come 'y' si chiamava %s\n", argname(y)
end function
```

```
open data9-7
namefun(QNC)
```

L'output è il seguente:

```
La serie passata come 'y' si chiamava QNC
```

Occorre notare che questo meccanismo non funziona sempre: gli argomenti di una funzione possono essere variabili senza nome, create al volo, come in `somefun(log(QNC))` o `somefun(CPI/100)`. In questo caso, la funzione `argname` produce una stringa vuota. Quando si usa questo meccanismo all'interno di una funzione occorre quindi controllare il valore prodotto da `argname` usando la funzione `isstring()`, che produce 1 per una stringa non vuota, o 0 altrimenti.

Valori di uscita

Le funzioni possono non produrre alcun valore (limitandosi ad esempio a stampare un risultato), oppure possono produrre una singola variabile: una scalare, una serie, una lista o una matrice (al momento le funzioni non possono produrre stringhe). Il valore di uscita è specificato con una dichiarazione all'interno del corpo della funzione che comincia con la parola chiave `return`, seguita dall'indicatore del tipo e dal nome della variabile (in modo analogo a quanto avviene per la lista dei parametri di una funzione). All'interno di una funzione può esserci solo una di queste dichiarazioni. Ecco un esempio di dichiarazione return valida:

```
return scalar SSR
```

Per fare in modo che una funzione produca più di una variabile in uscita, è possibile farle produrre una lista, ossia è possibile definire alcune variabili all'interno della funzione e accorparle in una lista; in questo caso esse non vengono distrutte all'uscita della funzione. Ecco un semplice esempio, che illustra anche la possibilità di impostare delle etichette descrittive per le variabili generate in una funzione.

```
function make_cubes (list xlist)
   list cubes = null
   loop foreach i xlist --quiet
      series $i3 = $i^3
      setinfo $i3 -d "cube of $i"
      list cubes += $i3
    end loop
    return list cubes
end function

open data4-1
list xlist = price sqft
list cubelist = make_cubes(xlist)
print xlist cubelist --byobs
labels
```

Si noti che la dichiarazione `return` *non* indica alla funzione di produrre il valore (terminare) nel punto in cui essa appare nel corpo della funzione. Piuttosto, essa specifica quale variabile è disponibile per l'assegnazione quando la funzione terminerà, cosa che può avvenire solo quando: a) viene raggiunta la fine del codice che definisce la funzione, oppure b) gretl produce un errore, o c) si incontra una dichiarazione `funcerr`.

La parola chiave `funcerr`, che può essere seguita da una stringa contenuta fra virgolette doppie, fa terminare una funzione con un messaggio di errore. Se si fornisce una stringa, questa viene stampata all'uscita, altrimenti viene mostrato un messaggio di errore generico. Questo meccanismo consente all'autore di una funzione di anticipare un possibile errore di esecuzione e/o di offrire un messaggio di errore più specifico. Ad esempio:

```
if nelem(xlist) = 0
   funcerr "xlist non può essere vuota"
end if
```

Controllo degli errori

Quando gretl legge e "compila" una definizione di funzione, esegue un controllo degli errori minimale: controlla che il nome della funzione sia accettabile e che non si tenti di definire una funzione all'interno di una funzione (si veda la sezione 10.1). Se il corpo della funzione contiene comandi non validi, verrà segnalato solo quando la funzione viene chiamata.

Stampa dei risultati

Durante l'esecuzione di una funzione, il meccanismo con cui gretl mostra i comandi e i risultati delle operazioni di creazione di nuove variabili, viene disattivato. Per riattivarlo (ad esempio se si vuole fare il debug del codice di una nuova funzione), basta usare uno dei seguenti comandi (o entrambi) all'interno nella funzione:

```
set echo on
set messages on
```

10.5 Pacchetti di funzioni

A partire dalla versione 1.6.0, gretl contiene un meccanismo per creare pacchetti di funzioni e renderli disponibili agli altri utenti. Questa funzionalità è ancora sperimentale, ma è già utilizzabile seguendo le istruzioni contenute in questa sezione.

Caricamento in memoria di una funzione

Ci sono vari modi per caricare una funzione in memoria:

- Se si ha un file script che contiene definizioni di funzioni, aprendo il file ed eseguendolo;
- Creando un file script da zero, includendo almeno una definizione di funzione ed eseguendo lo script;
- Aprendo il terminale di gretl e inserendo una definizione di funzione in modalità interattiva. Questo metodo non è particolarmente raccomandato: è probabilmente più comodo definire una funzione in modalità non interattiva.

Ad esempio, si supponga di voler creare un pacchetto con una funzione che produce l'incremento percentuale di una serie storica. Basta aprire un file script e digitare

```
function pc(series y)
  series foo = diff(y)/y(-1)
  return series foo
end function
```

Ora si può eseguire la funzione e controllare che funzioni correttamente, facendo alcuni test; ad esempio si può aprire il terminale e digitare:

```
genr x = uniform()
genr dpcx = pc(x)
print x dpcx --byobs
```

Si dovrebbe vedere qualcosa di simile alla figura 10.1. La funzione sembra essere corretta, quindi è possibile procedere al passo successivo.

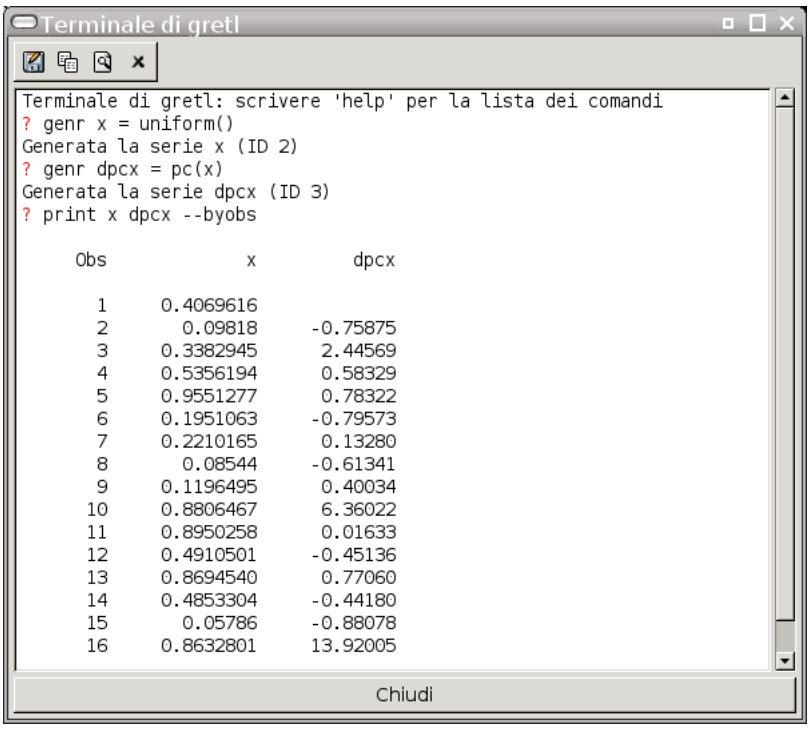


Figura 10.1: Controllo del funzionamento di una funzione

Creazione di un pacchetto

Usando l'interfaccia grafica del programma, nel menù "File, Funzioni", si trovano quattro voci: "Sul sistema locale", "Sul server di gretl", "Modifica pacchetto", "Nuovo pacchetto".

Selezionare il comando "Nuovo pacchetto" (il comando funziona solo quando è stata caricata in memoria almeno una funzione definita dall'utente); nella prima finestra di dialogo occorre selezionare:

- Una funzione pubblica da impacchettare;
- Zero o più funzioni ausiliarie "private".

Le funzioni pubbliche sono direttamente disponibili per gli utenti, quelle private fanno parte del meccanismo che lavora "dietro le quinte" in un pacchetto di funzioni.

Facendo clic su "OK" apparirà una seconda finestra di dialogo (si veda la figura 10.2), in cui occorre inserire le informazioni sul pacchetto (al momento: l'autore, la versione, la data e una breve descrizione), oltre che un testo di aiuto che descrive l'interfaccia pubblica della funzione. Si ha l'opportunità di modificare il codice delle funzioni, selezionandole dal menù e facendo clic su "Modifica codice della funzione". Infine, è possibile scegliere di caricare il pacchetto sul server di gretl appena lo si salva, selezionando l'apposita casella.

Facendo clic su "OK", si otterrà una finestra di salvataggio del file, che suggerirà di salvare il file in una directory chiamata `functions`, collocata sotto la directory di sistema di gretl (se si ha il permesso di scrittura su di essa), oppure nella directory utente di gretl. Questo è il posto suggerito per salvare i pacchetti di funzioni, visto che il programma li cercherà automaticamente qui, al momento di aprire questo tipo di file.

Ovviamente, il comando "File, Funzioni, Modifica pacchetto" permette di modificare un pacchetto precedentemente salvato.

Qualche informazione sui file dei pacchetti di funzioni: per impostazione predefinita essi hanno l'estensione `.gfn`, e, a differenza dei file di comandi di gretl, sono file in formato XML che contengono il codice delle funzioni e le informazioni aggiunte dal creatore del pacchetto. Gli

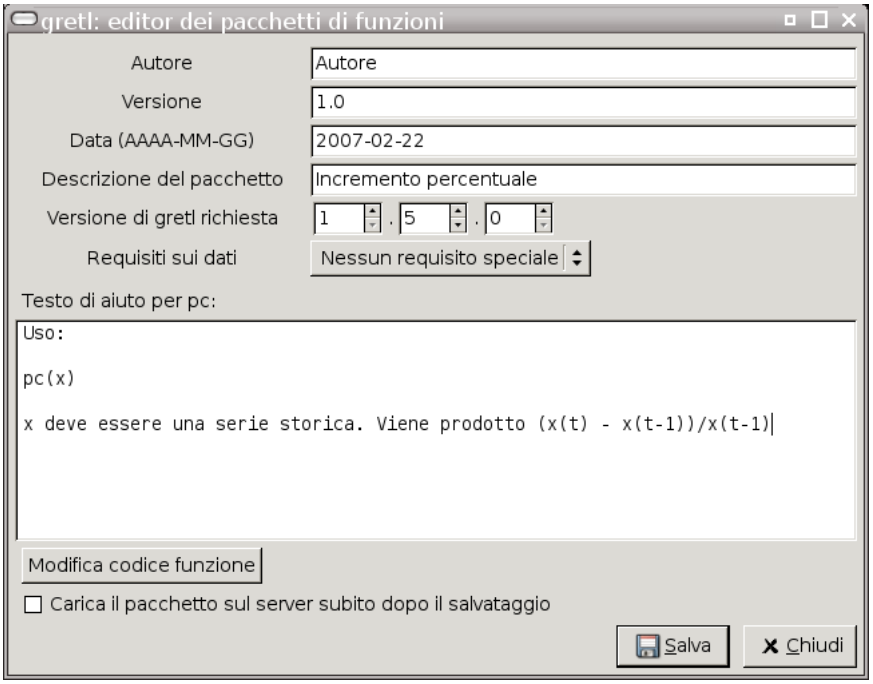


Figura 10.2: La finestra di modifica di un pacchetto

utenti esperti possono anche scrivere questi file da zero, invece di usare l'editor di pacchetti, ma la maggior parte delle persone troverà più comodo usare quest'ultimo. Si noti che i caratteri speciali XML nel codice della funzione vanno commentati, ad esempio & va rappresentato come `&`. Inoltre, alcuni elementi della sintassi delle funzioni differiscono dalla loro rappresentazione standard all'interno degli script di comandi: i parametri e i valori di uscita (se esistono) vanno rappresentati in XML. Sostanzialmente, la funzione viene analizzata e caricata in modo veloce usando la libxml.

Caricamento di un pacchetto

Perché impacchettare le funzioni in questo modo? Per scoprirlo, proviamo a chiudere gretl, a riaprirlo, e ad andare nel menù "File, Funzioni, Sul sistema locale". Se le operazioni precedenti sono andate a buon fine, troveremo il pacchetto che avevamo creato, insieme alla sua breve descrizione. Facendo clic su "Info", si ottiene una finestra con tutte le informazioni disponibili sul pacchetto; facendo clic sull'icona "Visualizza codice" della barra degli strumenti di questa nuova finestra, si aprirà una finestra che mostra il codice delle funzioni contenute nel pacchetto. Tornando alla finestra "Pacchetti funzioni", facendo clic sul nome di un pacchetto, le funzioni verranno caricate, pronte per essere eseguite, usando il pulsante "Chiama".

Dopo aver caricato le funzioni contenute nel pacchetto, aprendo il terminale di gretl, sarà possibile richiamare il testo di aiuto relativo ad una delle nuove funzioni caricate, se esso esiste, con il comando `help funzione`, dove `funzione` è il nome della funzione pubblica del pacchetto caricato.

In modo simile, è possibile consultare e caricare i pacchetti di funzioni disponibili sul server di gretl, selezionando "File, Funzioni, Sul server di gretl".

Una volta che un pacchetto è installato sulla macchina locale, è possibile usare le funzioni contenute attraverso l'interfaccia grafica, come descritto sopra, o quella a riga di comando (in uno script o nel terminale), caricando la funzione con il comando `include` e specificando come argomento il nome del pacchetto, compresa l'estensione `.gfn`.

Per continuare con l'esempio, caricare il file `np.gdt` (uno dei dataset di esempio forniti con gretl). Si supponga di voler calcolare il tasso di crescita per la variabile `iprod` usando la funzione appena creata e di salvare il risultato in una serie chiamata `pippo`.

Figura 10.3: Uso di un pacchetto

Basta selezionare "File, Funzioni, Sul sistema locale" e comparirà un elenco dei pacchetti installati, compreso quello appena creato. Selezionandolo e facendo clic su "Esegui" (o facendo doppio clic sul nome del pacchetto), apparirà una finestra simile a quella della figura 10.3. Facendo clic su "Ok", la serie `pippo` verrà generata (si veda la figura 10.4). Per fare in modo che la nuova variabile compaia nell'elenco delle variabili nella finestra principale, può essere necessario usare il comando "Dati, Aggiorna finestra" (o semplicemente premere il tasto "r").

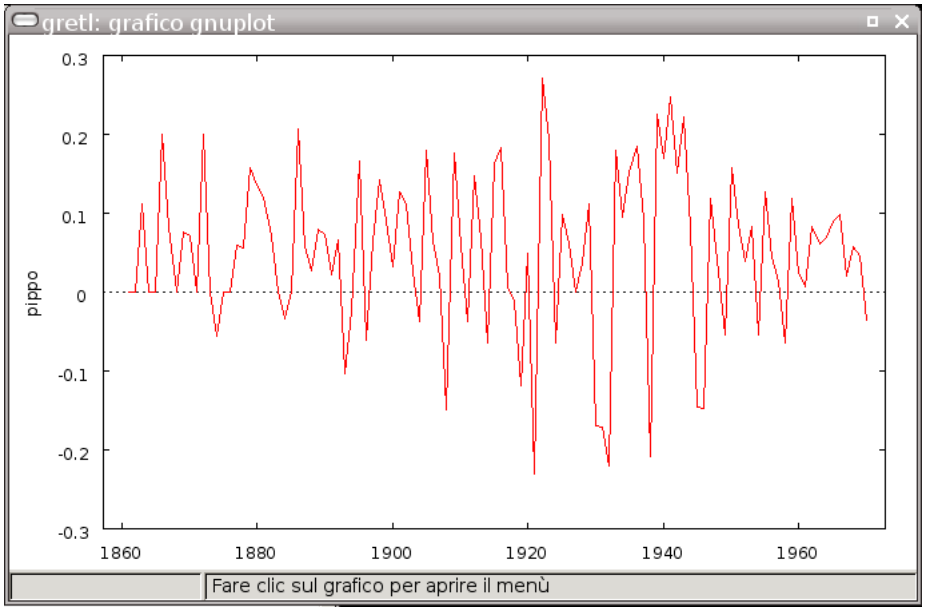


Figura 10.4: Incremento percentuale della produzione industriale

In alternativa, lo stesso risultato si ottiene con lo script

```
include pc.gfn
open np
pippo = pc(iprod)
```

Capitolo 11

Liste e stringhe definite dall'utente

11.1 Liste definite dall'utente

Molti comandi di gretl richiedono come argomenti una o più liste di variabili. Per semplificare la loro gestione negli script di comandi, in particolare nelle funzioni definite dall'utente, gretl offre la possibilità di creare *liste definite dall'utente*.

Creazione e modifica di una lista

Per creare una lista occorre usare il comando `list`, seguito dal nome della lista, un segno di uguale, e, in alternativa, `null` (per creare una lista vuota) o il nome di una o più variabili da inserire nella lista. Ad esempio:

```
list xlist = 1 2 3 4
list reglist = income price
list empty_list = null
```

È disponibile una forma speciale: se si usa la parola chiave `dataset` al posto di una lista vera e propria di variabili, verrà costruita una lista che contiene tutte le serie di dati del dataset in uso (tranne che per la costante predefinita `const`).

Il nome della lista deve iniziare con una lettera e deve essere composto interamente da lettere, numeri o il carattere trattino basso (non da spazi). La lunghezza massima del nome è di 15 caratteri. Quando si aggiungono variabili a una lista è possibile riferirsi a essa per nome o per numero identificativo.

Una volta creata, una lista viene "ricordata" per l'intera durata della sessione di gretl e può essere usata nel contesto di qualsiasi comando gretl che accetta liste di variabili. Un semplice esempio è la specificazione di una lista di regressori:

```
list xlist = x1 x2 x3 x4
ols y 0 xlist
```

Le liste possono essere modificate in due modi. Per *ridefinire* una lista si usa la stessa sintassi usata per crearla. Ad esempio

```
list xlist = 1 2 3
list xlist = 4 5 6
```

Dopo il secondo comando, `xlist` contiene solo le variabili 4, 5 e 6.

Per *accodare* o *premettere* variabili a una lista esistente, è possibile usare il nome di una lista all'interno del comando di definizione della lista stessa. Ad esempio, è possibile scrivere

```
list xlist = xlist 5 6 7
list xlist = 9 10 xlist 11 12
```

Un altro modo per accodare un termine a una lista consiste nell'usare l'operatore +=, come in

```
list xlist += cpi
```

Per eliminare una variabile da una lista, basta usare l'operatore -=:

```
list xlist -= cpi
```

Interrogazione delle liste

È possibile determinare se una variabile sconosciuta rappresenta una lista usando la funzione `islist()`.

```
series xl1 = log(x1)
series xl2 = log(x2)
list xlogs = xl1 xl2
genr is1 = islist(xlogs)
genr is2 = islist(xl1)
```

Il primo comando `genr` assegnerà il valore 1 a `is1` visto che `xlogs` in effetti è una lista. Il secondo comando genr assegnerà il valore 0 a `is2`, visto che `xl1` è una serie di dati, non una lista.

È anche possibile determinare il numero di elementi che compongono una lista usando la funzione `nelem()`.

```
list xlist = 1 2 3
genr nl = nelem(xlist)
```

Lo scalare `nl` avrà un valore di 3, visto che `xlist` contiene 3 membri.

È possibile mostrare i membri di una lista come illustrato in questa sessione interattiva:

```
? list xlist = x1 x2 x3
 # list xlist = x1 x2 x3
Added list 'xlist'
? list xlist print
 # list xlist = x1 x2 x3
```

Si noti che `print xlist` produrrà un effetto diverso, ossia mostrerà i valori di tutte le variabili contenute in `xlist` (come ci si può aspettare).

Liste di variabili trasformate

Data una lista di variabili, è possibile generare liste che contengono trasformazioni di queste variabili usando una forma speciale dei comandi `logs`, `lags`, `diff`, `ldiff`, `sdiff` o `square`. In pratica basta far seguire questi comandi dal nome della lista tra parentesi. Ad esempio:

```
list xlist = x1 x2 x3
list lxlist = logs(xlist)
list difflist = diff(xlist)
```

Quando si genera una lista di *ritardi* in questo modo, è possibile specificare il massimo ordine di ritardi, inserendolo per primo tra parentesi e separandolo dal nome della lista con una virgola. Ad esempio

```
list xlist = x1 x2 x3
list laglist = lags(2, xlist)
```

oppure

```
scalar order = 4
list laglist = lags(order, xlist)
```

Questi comandi riempiranno `laglist` col numero specificato di ritardi delle variabili presenti in `xlist`. Come avviene per il normale comando `lags` è possibile omettere l'ordine, che sarà così determinato automaticamente a seconda della frequenza dei dati. Un altro modo speciale di utilizzare questi comandi consiste nell'indicare tra parentesi il nome di una singola variabile al posto di quello di una lista, come in

```
series lx = log(x)
list laglist = lags(4, lx)
```

Si noti che la sintassi normale per questi comandi, ad esempio `logs`, è semplicemente

```
logs x1 x2 x3
```

Se `xlist` è una lista, è anche possibile scrivere

```
logs xlist
```

ma questo comando non salverà i logaritmi in una lista; per avere questo risultato occorre usare la sintassi

```
list loglist = logs(xlist)
```

Controllo dei valori mancanti

Gretl offre varie funzioni per riconoscere e gestire i valori mancanti (si veda la *Guida ai comandi di gretl* per i dettagli). In questa sede è utile ricordare che la funzione `ok()` può essere usata con un argomento lista. Ad esempio:

```
list xlist = x1 x2 x3
series xok = ok(xlist)
```

Dopo questi comandi, la serie `xok` avrà valore 1 per le osservazioni in cui nessuna variabile tra `x1`, `x2`, e `x3` ha un valore mancante, e valore 0 per le osservazioni che non soddisfano questa condizione.

11.2 Stringhe definite dall'utente

In alcuni casi può essere utile salvare una stringa (ossia una sequenza di caratteri) sotto forma di variabile che possa essere riutilizzata. Gretl offre questa funzionalità a partire dalla versione 1.6.0, ma alcune delle caratteristiche mostrate di seguito sono disponibili solo a partire dalla versione 1.7.2.

Per *definire* una variabile stringa è possibile usare uno dei comandi seguenti: `string` o `sprintf`. Il comando `string` è il più semplice: basta scrivere, ad esempio

```
string s1 = "Qualcosa da ricordare"
string s2 = getenv("HOME")
string s3 = @pippo + 10
```

Il primo campo dopo `string` è il nome della variabile che verrà creata, quindi segue il segno di uguale, infine una specificazione della stringa da salvare. Quest'ultima può essere la parola chiave `null`, per produrre una stringa nulla, oppure può assumere una delle forme seguenti:

- una stringa letterale (racchiusa tra virgolette doppie)
- il nome di una variabile stringa esistente, prefissata con `@`
- una funzione che produce una stringa (si veda oltre)
- una delle forme viste finora, seguita da + e un valore intero che indica una posizione
- una delle forme viste finora, seguita da uno spazio, seguito da un altro campo simile

Il ruolo del valore intero di posizione permette di usare una sotto-stringa dell'elemento precedente, a partire dal carattere specificato. Se il valore del parametro di posizione è maggiore della lunghezza della stringa in questione, viene prodotto un messaggio di errore.

Nel caso in cui vengano usate due specificazioni separate da spazio, la stringa risultante è la concatenazione dei due elementi.

Il comando `sprintf` è più flessibile. Funziona esattamente come `printf` di gretl, tranne per il fatto che la stringa "di formato" deve essere preceduta dal nome di una variabile stringa. Ad esempio,

```
scalar x = 8
sprintf pippo "var%d", x
```

Per *recuperare il valore* di una variabile stringa, basta usare il nome della variabile preceduto dal carattere "at", `@`.

In molti contesti la notazione `@` è trattata come una "macro", ossia in un comando gretl una sequenza di caratteri che segue il simbolo `@` viene trattata come il nome di una variabile stringa e il valore della variabile viene sostituito sulla riga di comando, come avviene in questo esempio di sessione interattiva:

```
? scalar x = 8
 scalar x = 8
Generato lo scalare x (ID 2) = 8
? sprintf pippo "var%d", x
Salvata la stringa 'pippo'
? print "@foo"
var8
```

Si noti l'effetto delle virgolette doppie nella riga `print "@foo"`. La riga

```
? print @foo
```

non stamperebbe un "`var8`" letteralmente, come visto sopra. La riga verrebbe processata e letta come

```
print var8
```

Quindi stamperebbe i valori della variabile `var8`, se esiste, oppure un messaggio di errore.

In alcuni contesti specifici, comunque, è naturale trattare le variabili `@` come variabili a tutti gli effetti, e gretl si comporta proprio così. Questo avviene quando le variabili appaiono:

- come argomenti dei comandi `printf` e `sprintf`.
- nella parte destra di un'istruzione `string`.
- come argomenti della funzione `isstring` (si veda sotto).

Ecco un esempio di uso delle stringhe come argomenti di `printf`:

```
? string vstr = "varianza"
Salvata stringa 'vstr'
? printf "vstr: %12s\n", @vstr
vstr:     varianza
```

Si noti che `@vstr` non dovrebbe essere messa tra virgolette in questo contesto. Allo stesso modo con

```
? string copy = @vstr
```

Stringhe predefinite

Oltre alle stringhe definite dall'utente, gretl contiene alcune variabili stringa predefinite che possono essere utili per chi scrive funzioni che includono comandi shell. Le stringhe predefinite sono mostrate nella Tabella 11.1.

`@gretldir`	La directory di installazione di gretl
`@workdir`	La directory di lavoro di gretl per l'utente
`@gnuplot`	Percorso, o nome, dell'eseguibile gnuplot
`@tramo`	Percorso, o nome, dell'eseguibile tramo
`@x12a`	Percorso, o nome, dell'eseguibile x-12-arima
`@tramodir`	Directory dei dati di tramo
`@x12adir`	Directory dei dati di x-12-arima

Tabella 11.1: Variabili stringa predefinite

Lettura delle stringhe dall'ambiente

In aggiunta, è possibile importare all'interno delle stringhe di gretl dei valori che sono definiti nell'ambiente esterno. Per fare questo occorre usare la funzione `getenv`, che richiede come argomento il nome di una variabile di ambiente. Ad esempio:

```
? string user = getenv("USER")
Salvata la stringa 'user'
? string home = getenv("HOME")
Salvata la stringa 'home'
? print "La home directory di @user è @home"
La home directory di cottrell è /home/cottrell
```

Per controllare se la variabile di ambiente richiamata da `getenv` è non-vuota, si può usare la funzione `isstring`, come in questo esempio

```
? string temp = getenv("TEMP")
Salvata la stringa vuota 'temp'
? scalar x = isstring(@temp)
Generato lo scalare x (ID 2) = 0
```

Si noti che `isstring` significa "è una stringa che contiene effettivamente qualcosa".

Al momento la funzione `getenv` può essere usata solo nella parte destra di un'istruzione `string`, come negli esempi visti sopra.

Lettura delle stringhe usando la shell

Se in gretl sono stati abilitati i comandi shell, è possibile catturare i risultati dei comandi usando la sintassi

`string` *nome-stringa* = `$(`*comando-shell*`)`

Ossia, si include il comando shell tra parentesi precedute dal carattere dollaro.

Lettura delle stringhe da file

È possibile leggere il contenuto di un file e scriverlo all'interno di una variabile stringa usando la sintassi

`string` *nome-stringa* = `readfile(`*nome-file*`)`

Il campo *nome-file* può includere variabili stringa. Ad esempio

```
string qnc = readfile(@x12adir/QNC.rts)
```

La funzione `strstr`

Questa funzione viene usata nel modo seguente

`string` *nome-stringa* = `strstr(`*s1*`,` *s2*`)`

L'effetto è quello di cercare all'interno di *s1* la prima occorrenza di *s2*. Se non viene trovato alcun risultato, viene prodotta una stringa vuota, altrimenti viene prodotta la porzione di *s1* che inizia per *s2*. Ad esempio:

```
? string hw = "hello world"
Saved string as 'hw'
? string w = strstr(@hw, "o")
Saved string as 'w'
? print "@w"
o world
```

Capitolo 12

Operazioni con le matrici

12.1 Creazione di matrici

È possibile creare una matrice usando uno dei metodi seguenti:

1. Specificando direttamente i valori scalari che compongono la matrice, in forma numerica, per riferimento a variabili scalari preesistenti, o usando valori calcolati;
2. fornendo una lista di serie di dati;
3. fornendo una *lista personalizzata* di serie;
4. usando una formula simile a quelle usate con il comando `genr`, in cui una nuova matrice viene definita in termini di matrici e/o scalari esistenti, oppure attraverso funzioni speciali.

Per specificare una matrice *direttamente in termini di scalari*, la sintassi è la seguente:

```
matrix A = { 1, 2, 3 ; 4, 5, 6 }
```

La matrice viene definita per righe successive; gli elementi di ogni riga sono separati da virgole e le righe sono separate da punti e virgola. L'intera espressione va racchiusa tra parentesi graffe. Gli spazi tra le parentesi non sono significativi. L'espressione vista sopra definisce una matrice 2×3. Ogni elemento deve essere un valore numerico, il nome di una variabile scalare preesistente, o un'espressione che, valutata, produce uno scalare. Per ottenere la matrice trasposta, basta aggiungere un apostrofo (') direttamente dopo la parentesi graffa che chiude l'espressione.

Per specificare una matrice *in termini di serie di dati* la sintassi è la seguente:

```
matrix A = { x1, x2, x3 }
```

dove i nomi delle variabili sono separati da virgole. Oltre ai nomi di variabili esistenti, è possibile usare espressioni che producono una serie una volta valutate. Ad esempio, data una serie `x`, si può scrivere

```
matrix A = { x, x^2 }
```

Ogni variabile rappresenta una colonna (e può esserci solo una variabile per ogni colonna). Non è possibile usare il carattere di punto e virgola come separatore di riga in questo caso: se si vuole che le serie siano disposte per righe, occorre usare il simbolo di trasposizione. L'intervallo dei valori dei dati inclusi nella matrice dipende dall'impostazione corrente dell'intervallo del campione.

Si noti che mentre le funzioni statistiche di gretl sono in grado di gestire valori mancanti, ciò non vale per le funzioni che trattano le matrici: *quando si costruisce una matrice partendo da serie che contengono valori mancanti, le osservazioni per cui almeno una delle serie contiene valori mancanti sono saltate.*

Invece di fornire una lista esplicita di variabili, è possibile fornire il *nome di una lista personalizzata* (si veda il capitolo 11), come nell'esempio seguente:

```
list xlist = x1 x2 x3
matrix A = { xlist }
```

Se si usa il nome di una lista, le serie che la costituiscono vengono usate per comporre le colonne, come risulta naturale in un contesto econometrico; se si vuole che vengano usate come righe, basta usare il simbolo di trasposizione.

Un caso speciale di costruzione di una matrice usando una lista di variabili è il seguente:

```
matrix A = { dataset }
```

In questo modo, si costruisce una matrice usando tutte le serie del dataset corrente, salvo la costante (variabile 0). Quando si usa questa lista speciale, essa deve essere il solo elemento nella definizione della matrice `{...}`. È tuttavia possibile creare una matrice che includa la costante, oltre alle altre variabili, usando la concatenazione per colonne (si veda oltre), come in questo esempio:

```
matrix A = {const}~{dataset}
```

È possibile creare nuove matrici, o sostituire matrici esistenti, usando varie trasformazioni, come per gli scalari e le serie di dati. I prossimi paragrafi spiegano nel dettaglio questi meccanismi.

La sintassi

```
matrix A = {}
```

crea una matrice vuota, ossia una matrice con zero righe e zero colonne. Si veda la sezione 12.6 per una discussione di questo oggetto.

☞ I nomi delle matrici devono soddisfare gli stessi requisiti dei nomi delle variabili in gretl: il nome non può essere più lungo di 15 caratteri, deve iniziare per una lettera e deve essere composto da lettere, numeri e il carattere trattino basso.

12.2 Selezione di sotto-matrici

È possibile selezionare delle sotto-matrici a partire da una matrice usando la sintassi:

A[*righe*,*colonne*]

dove *righe* può avere una delle seguenti forme:

Vuoto	seleziona tutte le righe
Un valore intero	seleziona la riga identificata dal numero
Due interi separati dal carattere due punti	seleziona un intervallo di righe
Il nome di una matrice	seleziona le righe specificate dai valori della matrice

Rispetto alla seconda opzione, il valore intero può essere indicato numericamente, attraverso il nome di una variabile scalare esistente, o con un'espressione che, valutata, produce uno scalare. Con l'ultima opzione, la matrice indicata nel campo *righe* deve avere dimensioni $p \times 1$ o $1 \times p$ e deve contenere valori interi nell'intervallo da 1 a n, dove n è il numero di righe da selezionare dalla matrice principale.

L'uso del parametro *colonne* è simile, *mutatis mutandis*. Ecco alcuni esempi.

```
matrix B = A[1,]
matrix B = A[2:3,3:5]
matrix B = A[2,2]
matrix idx = { 1, 2, 6 }
matrix B = A[idx,]
```

Il primo esempio seleziona la prima riga dalla matrice A; il secondo seleziona una sotto-matrice 2×3; il terzo seleziona uno scalare, mentre il quarto seleziona le righe 1, 2 e 6 dalla matrice A.

In aggiunta, c'è una specificazione di indice predefinita, `diag`, seleziona la diagonale principale di una matrice quadrata, come in `B[diag]`, dove `B` è quadrata.

È possibile usare la selezione di sotto-matrici sia a destra sia a sinistra in una formula che genera una matrice. Ecco un esempio di uso della selezione nella parte destra, per estrarre una sotto-matrice 2×2 B da una matrice 3×3 A:

```
matrix A = { 1, 2, 3; 4, 5, 6; 7, 8, 9 }
matrix B = A[1:2,2:3]
```

Ed ecco un esempio di selezione sulla sinistra. La seconda riga nell'esempio scrive una matrice identità 2×2 nell'angolo inferiore destro della matrice 3×3 A. La quarta riga rimpiazza la diagonale di A con valori 1.

```
matrix A = { 1, 2, 3; 4, 5, 6; 7, 8, 9 }
matrix A[2:3,2:3] = I(2)
matrix d = { 1, 1, 1 }
matrix A[diag] = d
```

12.3 Operatori matriciali

Per le matrici sono disponibili i seguenti operatori binari:

+	addizione
-	sottrazione
*	moltiplicazione matriciale
'	premoltiplicazione per la trasposta
/	"divisione" matriciale (si veda oltre)
~	concatenazione per colonne
|	concatenazione per righe
**	prodotto di Kronecker
=	test per l'uguaglianza

Inoltre, i seguenti operatori (operatori "punto") funzionano elemento per elemento:

```
.*   ./   .^   .=   .>   .<
```

Ecco qualche spiegazione per i casi meno ovvi.

Per l'addizione e la sottrazione matriciale, in generale le due matrici devono avere le stesse dimensioni, salvo il caso in cui uno dei termini è una matrice 1×1 o uno scalare. In questo caso, lo scalare viene implicitamente trasformato in una matrice con le corrette dimensioni, i cui elementi sono tutti pari al valore dello scalare. Ad esempio, se A è una matrice $m \times n$ e k è uno scalare, i comandi

```
matrix C = A + k
matrix D = A - k
```

producono entrambi delle matrici $m \times n$, con elementi $c_{ij} = +a_{ij} + k$ e $d_{ij} = a_{ij} - k$ rispettivamente.

Per "moltiplicazione per la trasposta" si intende, ad esempio, che

```
matrix C = X'Y
```

ha come risultato il prodotto fra X-trasposta e Y. In effetti, l'espressione `X'Y` ha lo stesso significato di `X'*Y` (che pure è un comando valido).

La "divisione" tra matrici, A/B è algebricamente equivalente a $B^{-1}A$ (pre-moltiplicazione per mezzo dell'inversa del "divisore"). Quindi in linea di principio le due espressioni seguenti sono equivalenti:

```
matrix C = A / B
matrix C = inv(B) * A
```

dove `inv` è la funzione di inversione tra matrici (si veda oltre). Però la prima forma può essere più accurata della seconda, perché la soluzione è ottenuta tramite la scomposizione LU, senza calcolare esplicitamente la matrice inversa.

Nella *moltiplicazione per elementi*, scrivendo

```
matrix C = A .* B
```

il risultato dipende dalle dimensioni di A e B. Se A è una matrice $m \times n$ e B è una matrice $p \times q$.

- Se $m = p$ e $n = q$, C sarà una matrice $m \times n$ con $c_{ij} = a_{ij} \times b_{ij}$. Questo tipo di prodotto è tecnicamente noto come *prodotto di Hadamard*.
- Se $m = 1$ e $n = q$, oppure $n = 1$ e $m = p$, C sarà una matrice $p \times q$ con $c_{ij} = a_k \times b_{ij}$, dove $k = j$ se $m = 1$, altrimenti $k = i$.
- Se $p = 1$ e $n = q$, oppure $q = 1$ e $m = p$, C sarà una matrice $m \times n$ con $c_{ij} = a_{ij} \times b_k$, dove $k = j$ se $p = 1$, altrimenti $k = i$.
- Se non è soddisfatta alcuna delle condizioni precedenti, il prodotto non è definito e viene segnalato un errore.

Ad esempio, se A è un vettore riga con lo stesso numero di colonne di B, le colonne di C sono le colonne di B moltiplicate per i corrispondenti elementi di A. Si noti che questa convenzione rende superfluo, nella maggior parte dei casi, usare matrici diagonali per eseguire trasformazioni usando la moltiplicazione ordinaria tra matrici: se $Y = XV$, dove V è diagonale, è molto più comodo dal punto di vista computazionale ottenere Y usando l'istruzione

```
matrix Y = X .* v
```

dove `v` è un vettore riga che contiene la diagonale di V.

La divisione per elementi e l'elevamento a potenza per elementi funzionano in modo analogo alla moltiplicazione per elementi, sostituendo $\times$ con $\div$, o con l'operazione di elevamento a potenza, nella spiegazione precedente.

Nella *concatenazione per colonne* di una matrice A $m \times n$ e di una matrice B $m \times p$, il risultato è una matrice $m \times (n + p)$. Ossia:

```
matrix C = A ~ k
```

produce $C = \left[\begin{array}{cc} A & B \end{array}\right]$.

La *concatenazione per righe* di una matrice A $m \times n$ e di una matrice B $p \times n$ produce una matrice $(m + p) \times n$.

```
C = A | B
```

produce $C = \left[\begin{array}{c} A \\ B \end{array}\right]$.

12.4 Operatori matrice-scalare

Per una matrice A e uno scalare k, sono disponibili gli operatori mostrati nella Tabella 12.1 (addizione e sottrazione sono state presentate nella sezione 12.3 ma sono incluse nella tabella per completezza). Inoltre, per una matrice quadrata A e un intero $k \geq 0$, `B = A^k` produce una matrice B che è A elevata alla potenza k. Si noti che l'operatore `**` non può essere usato al posto di `^` a questo scopo, perché in un contesto matriciale esso è riservato per il prodotto di Kronecker.

Espressione	*Risultato*
`matrix B = A * k`	$b_{ij} = ka_{ij}$
`matrix B = A / k`	$b_{ij} = a_{ij}/k$
`matrix B = k / A`	$b_{ij} = k/a_{ij}$
`matrix B = A + k`	$b_{ij} = a_{ij} + k$
`matrix B = A - k`	$b_{ij} = a_{ij} - k$
`matrix B = k - A`	$b_{ij} = k - a_{ij}$
`matrix B = A % k`	$b_{ij} = a_{ij}$ modulo k

Tabella 12.1: Operatori matrice–scalare

12.5 Funzioni matriciali

Creazione e I/O						
	`I`	`mnormal`	`mread`	`mwrite`	`muniform`	`ones`
	`seq`	`zeros`				
Forma/dimensione						
	`cols`	`diag`	`dsort`	`mlag`	`mshape`	`rows`
	`sort`	`transp`	`unvech`	`vec`	`vech`	
Elemento per elemento						
	`abs`	`atan`	`cnorm`	`cos`	`dnorm`	`exp`
	`gamma`	`int`	`lngamma`	`log`	`qnorm`	`sin`
	`sqrt`	`tan`				
Algebra matriciale						
	`cholesky`	`det`	`eigengen`	`eigensym`	`fft`	`ffti`
	`infnorm`	`inv`	`invpd`	`ginv`	`ldet`	`mexp`
	`nullspace`	`onenorm`	`qform`	`qrdecomp`	`rank`	`rcond`
	`svd`	`tr`	`cmult`	`cdiv`		
Statistica						
	`cdemean`	`imaxc`	`imaxr`	`iminc`	`iminr`	`mcorr`
	`mcov`	`maxc`	`maxr`	`meanc`	`meanr`	`minc`
	`minr`	`mols`	`mxtab`	`princomp`	`sumc`	`sumr`
	`sd`	`values`				

Tabella 12.2: Funzioni matriciali suddivise per categoria

La tabella 12.2 elenca le funzioni matriciali fornite da gretl (una versione ordinata in ordine alfabetico della tabella è fornita alla fine di questo capitolo, si veda la Tabella 12.5). Per *la trasformazione elemento per elemento* delle matrici sono disponibili le seguenti funzioni: `log`, `exp`, `sin`, `cos`, `tan`, `atan`, `int`, `abs`, `sqrt`, `dnorm`, `cnorm`, `qnorm`, `gamma` e `lngamma`. Queste funzioni operano in modo analogo a quando sono usate nel contesto del comando `genr`. Ad esempio, se una matrice `A` è già stata definita,

```
matrix B = sqrt(A)
```

genera una matrice tale che $b_{ij} = \sqrt{a_{ij}}$. Tutte queste funzioni richiedono una sola matrice come argomento, o un'espressione che si risolve in una singola matrice.

Si noti che per calcolare la "radice quadrata di una matrice" occorre la funzione `cholesky` (si veda oltre); inoltre, la funzione `exp` calcola l'esponenziale elemento per elemento, quindi *non* produce l'esponenziale della matrice, a meno che la matrice non sia diagonale; per calcolare l'esponenziale della matrice, si usi `mexp`.

Le funzioni `sort`, `dsort` e `values`, utilizzabili con le serie di dati, possono essere usate anche con le matrici. In questo caso, l'argomento di queste funzioni deve essere un vettore ($p \times 1$ o $1 \times p$). Per `sort` e `dsort`, il valore restituito è un vettore che contiene gli elementi del vettore originale riordinati in ordine di grandezza crescente (`sort`) o decrescente (`dsort`). Per `values` il risultato è un vettore che contiene i valori distinti del vettore originale, riordinati in ordine crescente.

Infine, ci sono funzioni dedicate in modo specifico alle matrici, che è possibile suddividere nelle categorie seguenti:

1. Quelle che richiedono come argomento una sola matrice e producono uno scalare.
2. Quelle che richiedono come argomento una sola matrice (e in alcuni casi un parametro aggiuntivo) e producono una matrice.
3. Quelle che richiedono come argomento uno o due valori e producono una matrice.
4. Quelle che richiedono come argomento due matrici e producono una matrice.
5. Quelle che richiedono come argomento una o più matrici e producono una o più matrici.

Questi gruppi di funzioni vengono presentati nell'ordine.

Funzioni da matrice a scalare

Le funzioni che richiedono come argomento una sola matrice e producono uno scalare sono:

`rows`	numero di righe
`cols`	numero di colonne
`rank`	rango
`det`	determinante
`ldet`	log-determinante
`tr`	traccia
`onenorm`	norma-1
`infnorm`	norma infinita
`rcond`	reciproco del numero di condizione

L'argomento di queste funzioni può essere il nome di una matrice esistente o un'espressione che si risolve in una matrice. Si noti che le funzioni `det`, `ldet` e `tr` richiedono una matrice quadrata. La funzione `rank` è calcolata usando la decomposizione in valori singolari (SVD).

Le funzioni `onenorm` e `infnorm` restituiscono rispettivamente la norma-1 e la norma infinita di una matrice. La prima è il massimo, tra le colonne della matrice, della somma dei valori assoluti degli elementi della colonna; la seconda è il massimo, tra le righe, della somma dei valori assoluti degli elementi della riga. La funzione `rcond` restituisce il reciproco del numero di condizione per una matrice simmetrica definita positiva.

Funzioni da matrice a matrice

Le funzioni che richiedono come argomento una sola matrice e producono una matrice sono:

`sumc`	somma per colonna	`sumr`	somma per riga
`meanc`	media per colonna	`meanr`	media per riga
`sd`	scarto quadratico medio per colonna		
`mcov`	matrice di covarianza	`mcorr`	matrice di correlazione
`cholesky`	scomposizione di Cholesky	`mexp`	esponenziale matriciale
`inv`	inversa	`invpd`	inversa di una matrice definita positiva
`ginv`	inversa generalizzata	`diag`	diagonale principale
`transp`	trasposta	`cdemean`	sottrazione della media delle colonne
`vec`	elementi come vettore colonna	`vech`	elementi del triangolo inferiore come vettore colonna
`unvech`	annulla l'operazione `vech`	`mlag`	ritardo o anticipo per le matrici
`nullspace`	spazio nullo destro	`princomp`	componenti principali
`maxc`	massimi delle colonne (valori)	`maxr`	massimi delle righe (valori)
`imaxc`	massimi delle colonne (indici)	`imaxr`	massimi delle righe (indici)
`minc`	minimi delle colonne (valori)	`minr`	minimi delle righe (valori)
`iminc`	minimi delle colonne (indici)	`iminr`	minimi delle righe (indici)
`fft`	trasformata discreta di Fourier	`ffti`	trasformata discreta inversa di Fourier

Come per il gruppo precedente di funzioni, l'argomento deve essere il nome di una matrice esistente o un'espressione che si risolve in una matrice.

Per una matrice A $m \times n$, `sumc(A)` produce un vettore riga con le n somme per colonna, mentre `sumr(A)` produce un vettore colonna con le m somme per riga. `meanc(A)` produce un vettore riga con le n medie per colonna e `meanr(A)` un vettore colonna con le m medie per riga. `sd(A)` produce un vettore riga che contiene gli scarti quadratici medi delle n colonne (senza la correzione per i gradi di libertà).

Inoltre, per una matrice A $m \times n$, la famiglia di funzioni `max` e `min` può produrre una matrice $m \times 1$ (le varianti `r`, che selezionano il valore estremo di ogni riga), o una matrice $1 \times n$ (le varianti `c`, che selezionano il valore estremo di ogni colonna). I vettori `max` contengono i valori massimi di ogni riga o colonna, mentre quelli `min` contengono i valori minimi. Le varianti delle funzioni prefissate con `i` (ad es. `imaxc`) producono non i valori ma gli indici (a partire da 1) degli elementi di valore massimo o minimo.

Per una matrice A $T \times k$, `mcov(A)` e `mcorr(A)` producono entrambe delle matrici simmetriche $k \times k$, nel primo caso contenenti le varianze (sulla diagonale) e le covarianze delle variabili nelle colonne di A, e nel secondo caso, le correlazioni delle variabili.

Per una matrice A $n \times n$, la funzione `mexp(A)` produce una matrice $n \times n$ che contiene l'esponenziale matriciale

$$e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!} = \frac{I}{0!} + \frac{A}{1!} + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots$$

(questa serie converge sicuramente).

La funzione `cholesky` calcola la scomposizione di Cholesky L di una matrice simmetrica definita positiva A: $A = LL'$; L è triangolare inferiore (contiene zeri al di sopra della diagonale).

La funzione `diag` restituisce la diagonale principale di una matrice $n \times n$ A come vettore colonna, ossia come vettore n-dimensionale v tale che $v_i = a_{ii}$.

La funzione `cdemean` applicata a una matrice A $m \times n$ restituisce una matrice B $m \times n$ tale che $b_{ij} = a_{ij} - \bar{A}_j$, dove $\bar{A}_j$ indica la media della colonna j di A.

La funzione `vec` applicata a una matrice A $m \times n$ restituisce un vettore colonna di lunghezza mn formato impilando le colonne di A.

La funzione `vech` applicata a una matrice A $n \times n$ restituisce un vettore colonna di lunghezza $n(n+1)/2$ formato impilando gli elementi del triangolo inferiore di A, colonna per colonna. Si

noti che A deve essere quadrata e, perché l'operazione abbia senso, simmetrica. La funzione `unvech` esegue l'operazione inversa, producendo una matrice simmetrica.

La funzione `mlag` richiede due argomenti: una matrice e uno scalare che indica un ordine di ritardo, m. Applicata a una matrice A $T \times k$, questa funzione produce una matrice B $T \times k$ tale che

$$b_{ij} = \begin{cases} a_{i-m,j} & 1 \leq i - m \leq T \\ 0 & \text{otherwise} \end{cases}$$

Ossia, le colonne di B sono versioni ritardate delle colonne di A, in cui i valori mancanti vengono sostituiti da zeri. L'ordine m può essere negativo, e in questo caso invece di ritardi verranno generati anticipi delle variabili.

La funzione `nullspace` produce X, lo spazio nullo destro di una matrice A (che si assume avere rango pieno): X soddisfa la condizione $A \cdot X = 0$.

La funzione `invpd` produce l'inversa di una matrice simmetrica e definita positiva. Da usare con cautela: è più veloce della funzione standard `inv`, ma gretl non controlla se la matrice è simmetrica; occorre esserne sicuri.

La funzione `ginv` produce l'inversa generalizzata, o pseudoinversa (chiamata anche inversa di Moore–Penrose) di una matrice rettangolare A. Essa soddisfa le seguenti equazioni:

$$\begin{aligned} AA^+A &= A \\ A^+AA^+ &= A^+ \\ (AA^+)' &= AA^+ \\ (A^+A)' &= A^+A \end{aligned}$$

La funzione `princomp` richiede due argomenti: una matrice X $T \times k$ e uno scalare p, tale che $0 < p \leq k$. Si assume che X contenga T osservazioni per ognuna delle k variabili (serie). Il valore prodotto è una matrice P $T \times p$ che contiene le prime p componenti principali di X. Gli elementi di P sono calcolati come

$$P_{tj} = \sum_{i=1}^{k} Z_{ti}\, v_i^{(j)}$$

dove Z_{ti} è il valore standardizzato della variabile i nell'osservazione t, $Z_{ti} = (X_{ti} - \bar{X}_i)/\hat{\sigma}_i$, e $v^{(j)}$ è l'autovettore j-esimo della matrice di correlazione degli X_i, con gli autovettori ordinati per valore decrescente dei corrispondenti autovalori.

Le funzioni `fft` e `ffti` producono la trasformata di Fourier reale discreta e la sua inversa. Se `X` è una matrice $n \times k$, `fft(X)` è una matrice $n \times 2k$ che contiene la parte reale della trasformata nelle colonne dispari e la parte immaginaria in quelle pari. Al contrario, `ffti` richiede un argomento $n \times 2k$ e produce un risultato $n \times k$. Si veda la Sezione 7.10 per alcuni esempi.

Funzioni di riempimento delle matrici

Le funzioni che richiedono come argomento uno o due valori interi e producono una matrice sono:

`I(`n`)`	matrice identità $n \times n$
`zeros(`m`,`n`)`	matrice nulla $m \times n$
`ones(`m`,`n`)`	matrice $m \times n$ con tutti gli elementi pari a 1
`muniform(`m`,`n`)`	matrice $m \times n$ con elementi casuali uniformi
`mnormal(`m`,`n`)`	matrice $m \times n$ con elementi casuali normali
`seq(`a`,`b`)`	vettore riga che contiene i numeri da a a b

Le dimensioni m e n, o nel caso di `seq` i valori estremi a e b, possono essere indicate numericamente, per riferimento a variabili scalari pre-esistenti, o con espressioni che, valutate, producono scalari.

Le funzioni matriciali `muniform` e `mnormal` riempiono la matrice con valori estratti dalla distribuzione uniforme (0–1) e dalla distribuzione normale standard.

La funzione `seq` genera una sequenza di numeri interi da a a b inclusi, crescente se $a < b$ o decrescente se $a > b$.

Funzioni di ristrutturazione delle matrici

È possibile creare una matrice anche ri-strutturando gli elementi di una matrice preesistente, usando la funzione `mshape`; essa richiede tre argomenti: la matrice iniziale A e le righe e colonne della matrice finale, rispettivamente r e c. Gli elementi di A vengono letti per colonne e scritti nella matrice finale con lo stesso metodo; se A contiene meno elementi di $n = r \times c$, essi sono ripetuti ciclicamente, se invece A contiene più elementi, sono usati solo i primi n.

Ad esempio:

```
matrix a = mnormal(2,3)
a
matrix b = mshape(a,3,1)
b
matrix b = mshape(a,5,2)
b
```

produce

```
?   a
a

      1.2323      0.99714     -0.39078
     0.54363      0.43928     -0.48467

?   matrix b = mshape(a,3,1)
Generata la matrice b
?   b
b

      1.2323
     0.54363
     0.99714

?   matrix b = mshape(a,5,2)
Sostituita la matrice b
?   b
b

      1.2323     -0.48467
     0.54363       1.2323
     0.99714      0.54363
     0.43928      0.99714
    -0.39078      0.43928
```

Funzioni da coppie di matrici a matrici

La funzione `qform` costruisce una forma quadratica in una matrice A e una matrice simmetrica X conformabile. Il comando

```
B = qform(A, X)
```

calcola $B = AXA'$. Questo risultato viene calcolato in modo più efficiente rispetto al comando alternativo `B = A*X*A'`. Inoltre, il risultato è simmetrico per costruzione.

Le funzioni `cmult` e `cdiv` calcolano rispettivamente il prodotto e la divisione complessi di due matrici A e B che rappresentano numeri complessi. Queste matrici devono avere lo stesso

numero di righe, n, e una o due colonne. La prima colonna contiene la parte reale, mentre la seconda (se presente) la parte immaginaria. Il valore prodotto è una matrice $n \times 2$, o se il risultato non ha parte immaginaria, un vettore di dimensione n.

La funzione `mxtab` ha essenzialmente lo stesso ruolo del comando `xtab`, tranne per il fatto che produce una matrice come risultato. La funzione accetta come argomenti due matrici $T \times 1$, x e y, e produce una matrice che contiene la tabulazione incrociata dei valori contenuti in x (per riga) e y (per colonna), che devono essere valori interi. È anche possibile usare due serie come argomenti.

Funzioni da matrici a matrici

Le funzioni che richiedono come argomento una o più matrici e producono una o più matrici sono:

`qrdecomp`	scomposizione QR
`eigensym`	auto-analisi di una matrice simmetrica
`eigengen`	auto-analisi di una matrice generica
`mols`	OLS matriciale
`svd`	decomposizione in valori singolari (SVD)

La sintassi per le funzioni `qrdecomp`, `eigensym` e `eigengen` segue la forma

```
matrix B = func(A, &C)
```

Il primo argomento, `A`, rappresenta i dati in ingresso, ossia la matrice di cui è richiesta la scomposizione o l'analisi. Il secondo argomento deve essere il nome di una matrice esistente, preceduto da `&` (per indicare l'"indirizzo" della matrice in questione), nel qual caso un risultato ausiliario viene scritto in quella matrice, oppure la parola chiave `null`, nel qual caso il risultato non è mostrato o è scartato.

Nel caso in cui il secondo argomento venga indicato, la matrice specificata sarà sovrascritta con il risultato della funzione (non è richiesto che la matrice preesistente abbia le dimensioni corrette per ricevere il risultato della funzione).

La funzione `eigensym` calcola gli autovalori, e opzionalmente gli autovettori destri, di una matrice simmetrica $n \times n$. Gli autovalori sono restituiti direttamente in un vettore colonna di lunghezza n; se vengono richiesti gli autovettori, sono restituiti in una matrice $n \times n$. Ad esempio:

```
matrix V
matrix E = eigensym(M, &V)
matrix E = eigensym(M, null)
```

Nel primo caso `E` contiene gli autovalori di `M` e `V` contiene gli autovettori. Nel secondo, `E` contiene gli autovalori, ma gli autovettori non vengono calcolati.

La funzione `eigengen` calcola gli autovalori, e opzionalmente gli autovettori, di una matrice generica $n \times n$. Gli autovalori vengono restituiti direttamente in una matrice $n \times 2$, in cui la prima colonna contiene le componenti reali e la seconda quelle immaginarie.

Se vengono richiesti gli autovettori (ossia il secondo argomento di `eigengen` non è `null`), essi vengono restituiti in una matrice $n \times n$. La disposizione delle colonne in questa matrice è particolare: gli autovettori sono disposti nello stesso ordine degli autovalori, ma gli autovettori reali occupano una colonna, mentre quelli complessi ne occupano due (la parte reale nella prima colonna); il numero totale di colonne è sempre n, visto che l'autovettore coniugato viene saltato. L'esempio 12.1 dovrebbe chiarire la situazione.

La funzione `qrdecomp` calcola la scomposizione QR di una matrice $m \times n$ A: $A = QR$, dove Q è una matrice ortogonale $m \times n$ e R è una matrice $n \times n$ triangolare superiore. La matrice Q è prodotta direttamente, mentre R può essere recuperata attraverso il secondo argomento. Ecco due esempi:

Esempio 12.1: Autovalori e autovettori complessi

```
set seed 34756

matrix v
A = mnormal(3,3)

/* Genera gli autovalori e autovettori */
l = eigengen(A,&v)
/* L'autovalore 1 è reale, 2 e 3 sono complessi coniugati */
print l
print v

/*
  La colonna 1 contiene il primo autovettore (reale)
*/

B = A*v[,1]
c = l[1,1] * v[,1]
/* B dovrebbe essere pari a c */
print B
print c


/*
  Le colonne 2:3 contengono le parti reale e immaginaria dell'autovettore 2
*/

B = A*v[,2:3]
c = cmult(ones(3,1)*(l[2,]),v[,2:3])
/* B dovrebbe essere pari a c */
print B
print c
```

```
matrix R
matrix Q = qrdecomp(M, &R)
matrix Q = qrdecomp(M, null)
```

Nel primo esempio, la matrice triangolare R è salvata come R; nel secondo, R è scartata. La prima delle righe nell'esempio precedente mostra una "dichiarazione semplice" di una matrice: R viene dichiarata come matrice, ma non gli viene assegnato alcun valore esplicito. In questo caso, la variabile è inizializzata a una matrice 1×1 il cui unico elemento vale zero.

La sintassi per svd è

```
matrix B = func(A, &C, &D)
```

La funzione svd calcola la decomposizione in valori singolari (totale o parziale) della matrice reale A $m \times n$. Sia $k = \min(m, n)$. La decomposizione è

$$A = U\Sigma V'$$

dove U è una matrice ortogonale $m \times k$, Σ è una matrice diagonale $k \times k$ e V è una matrice ortogonale $k \times n$[1]. Gli elementi diagonali di Σ sono i valori singolari di A, sono reali e non negativi, e sono prodotti in ordine decrescente. Le prime k colonne di U e V sono i vettori singolari destro e sinistro di A.

La funzione svd produce i valori singolari in un vettore di ordine k. I vettori singolari sinistri e/o destri si possono ottenere indicando valori non nulli per il secondo e/o terzo argomento della funzione. Ad esempio:

```
matrix s = svd(A, &U, &V)
matrix s = svd(A, null, null)
matrix s = svd(A, null, &V)
```

Nel primo caso si otterranno entrambi gli insiemi di vettori singolari; nel secondo caso si otterranno solo i valori singolari, nel terzo si ottengono i vettori singolari destri ma non viene calcolata U. *Nota bene*: quando il terzo argomento è non nullo, in realtà viene prodotto V'. Per ricostruire la matrice originale a partire dalla sua SVD, basta eseguire:

```
matrix s = svd(A, &U, &V)
matrix B = (U.*s)*V
```

Infine, la sintassi per mols è

```
matrix B = mols(Y, X, &U)
```

La funzione produce le stime OLS ottenuta regredendo la matrice Y $T \times n$ sulla matrice X $T \times k$, ossia una matrice $k \times n$ che contiene $(X'X)^{-1}X'Y$. Viene usata la decomposizione di Cholesky. La matrice U, se non è nulla, è usata per salvare i residui.

Lettura e scrittura di matrici con file testuali

Le due funzioni mread e mwrite possono essere usate per semplici operazioni di input/output matriciale, ad esempio permettendo a gretl di scambiare dati con altri programmi.

La funzione mread accetta un parametro stringa: il nome del file (testuale) da cui leggere la matrice. Il file in questione deve sottostare alle regole seguenti:

1. Le colonne devono essere separate da spazi o caratteri tab.
2. Il separatore decimale deve essere il carattere punto "`.`".

[1] Questa non è la sola definizione della SVD: alcuni definiscono U come $m \times m$, Σ come $m \times n$ (con k elementi della diagonale diversi da zero) e V come $n \times n$.

3. La prima riga del file deve contenere due interi, separati da spazio o tab, che indicano rispettivamente il numero di righe e di colonne.

In caso di errore (ad esempio se il file è mal formattato o non accessibile), viene prodotta una matrice vuota (si veda la sezione 12.6).

La funzione complementare `mwrite` produce file di testo formattati nel modo descritto sopra. Il separatore di colonna è il carattere tab, in modo da facilitare l'importazione in fogli elettronici. L'uso è descritto nell'esempio 12.2. Le matrici esportate con il comando `mwrite` possono essere facilmente lette con altri programmi: la tabella seguente riassume i comandi necessari per leggere una matrice `A` da un file chiamato `a.mat` in alcuni programmi di uso comune[2].

Programma	Codice di esempio
GAUSS	`tmp[] = load a.mat;`
	`A = reshape(tmp[3:rows(tmp)],tmp[1],tmp[2]);`
Octave	`fd = fopen("a.mat");`
	`[r,c] = fscanf(fd, "%d %d", "C");`
	`A = reshape(fscanf(fd, "%g", r*c),c,r)';`
	`fclose(fd);`
Ox	`decl A = loadmat("a.mat");`
R	`A = as.matrix(read.table("a.mat", skip=1))`

Esempio 12.2: Input/output di matrici con file testuali

```
nulldata 64
scalar n = 3
string f1 = "a.csv"
string f2 = "b.csv"

matrix a = mnormal(n,n)
matrix b = inv(a)

err = mwrite(a, f1)

if err != 0
  fprintf "Errore nella scrittura di %s\n", f1
else
  err = mwrite(b, f2)
endif

if err != 0
  fprintf "Errore nella scrittura di %s\n", f2
else
  c = mread(f1)
  d = mread(f2)
  a = c*d
  printf "La matrice seguente dovrebbe essere una matrice identità\n"
  print a
endif
```

[2] Gli utenti di Matlab possono trovare utile l'esempio in Octave, visto che i due programmi sono per lo più compatibili fra loro.

12.6 Matrici vuote

Lo scopo principale del concetto di matrice vuota è quello di permettere all'utente di definire un punto di partenza per successive operazioni di concatenazione. Ad esempio, se X è una matrice già definita di dimensione qualsiasi, i comandi

```
matrix A = {}
matrix B = A ~ X
```

producono una matrice B identica a X.

Da un punto di vista algebrico, l'idea di una matrice vuota può essere giustificata facendo riferimento agli spazi vettoriali: se una matrice è un insieme ordinato di vettori, A={} rappresenta l'insieme vuoto. Di conseguenza, le operazioni come l'addizione e la moltiplicazione non hanno un senso chiaro (si potrebbe sostenere che non ne hanno affatto), ma le operazioni che riguardano la cardinalità di questo insieme (ossia la dimensione dello spazio generato da A) hanno senso.

La Tabella 12.3 mostra le operazioni consentite con le matrici vuote; tutte le altre operazioni matriciali generano un errore se viene fornita una matrice vuota tra gli argomenti. In linea con l'interpretazione vista sopra, alcune funzioni matriciali producono una matrice vuota sotto certe condizioni: le funzioni `diag, vec, vech, unvech` quando l'argomento è una matrice vuota; le funzioni `I, ones, zeros, mnormal, muniform` quando uno o più argomenti sono pari a 0; la funzione `nullspace` quando il suo argomento ha rango pieno (per colonne).

Funzione	*Valore prodotto*
`A', transp(A)`	A
`rows(A)`	0
`cols(A)`	0
`rank(A)`	0
`det(A)`	NA
`ldet(A)`	NA
`tr(A)`	NA
`onenorm(A)`	NA
`infnorm(A)`	NA
`rcond(A)`	NA

Tabella 12.3: Funzioni consentite su una matrice vuota A

12.7 Matrici accessorie

Oltre alle funzioni matriciali viste sopra, esistono vari "accessori" che permettono di salvare una copia di alcune matrici che vengono generate automaticamente dal programma quando viene stimato un modello. Sono presentati nella Tabella 12.4.

Quando questi accessori sono utilizzati senza farli precedere da alcun prefisso, producono i risultati dell'ultimo modello stimato, se esiste. Altrimenti, se sono prefissati dal nome di un modello salvato in precedenza, separato da un punto (.), producono i risultati dal modello specificato. Ecco alcuni esempi:

```
matrix u = $uhat
matrix b = m1.$coeff
matrix v2 = m1.$vcv[1:2,1:2]
```

Il primo comando produce i residui dell'ultimo modello; il secondo produce il vettore dei coefficienti del modello m1, mentre il terzo (che usa il meccanismo della selezione di sotto-matrici descritto in precedenza) produce una porzione della matrice di covarianza del modello m1.

`$coeff`	vettore dei coefficienti stimati
`$stderr`	vettore degli errori standard stimati
`$uhat`	vettore dei residui
`$yhat`	vettore dei valori stimati
`$vcv`	matrice di covarianza (si veda il testo)
`$rho`	coefficienti di autoregressione per il processo di errore
`$jalpha`	matrice α (loading) della procedura di Johansen
`$jbeta`	matrice β (vettori di cointegrazione) della procedura di Johansen
`$jvbeta`	matrice di covarianza per gli elementi non vincolati di β della procedura di Johansen

Tabella 12.4: Matrici accessorie per i dati provenienti da modelli

Se il "modello" in questione è in realtà un sistema (un VAR, un VECM, o un sistema di equazioni simultanee), `$uhat` produce la matrice dei residui (una colonna per equazione) e `\$vcv` produce la matrice di covarianza tra le equazioni. Nel caso speciale di un VAR o un VECM, `$coeff` produce la matrice dei coefficienti (una colonna per equazione) e `$compan` la matrice associata. Al momento gli altri accessori non sono disponibili per i sistemi di equazioni.

Dopo aver stimato un modello vettoriale a correzione d'errore con la procedura di Johansen, sono disponibili anche le matrici `jalpha` e `$jbeta`. Esse hanno un numero di colonne pari al rango di cointegrazione scelto, quindi il prodotto

```
matrix Pi = $jalpha * $jbeta'
```

produce la stima a rango ridotto di $A(1)$. Poiché β è identificato automaticamente attraverso la normalizzazione di Phillips (si veda la sezione 21.5), i suoi elementi non vincolati possiedono una matrice di covarianza che può essere recuperata attraverso l'accessorio `$jvbeta`.

12.8 Conflitti tra nomi

Le matrici condividono lo spazio dei nomi consentiti con le serie di dati e le variabili scalari. In altre parole, non possono esistere due oggetti di questo tipo con lo stesso nome. È un errore tentare di cambiare il tipo di una variabile esistente; ad esempio:

```
scalar x = 3
matrix x = ones(2,2) # Errore!
```

È comunque possibile cancellare o rinominare una variabile esistente, e quindi riutilizzare il nome per una variabile di diverso tipo:

```
scalar x = 3
delete x
matrix x = ones(2,2) # Corretto!
```

12.9 Creazione di una serie di dati da una matrice

Il capitolo 12.1 descrive come creare una matrice da una o più serie di dati. In alcuni casi può essere necessario dover fare l'operazione inversa, ossia copiare valori da una matrice a una serie. La sintassi da usare è

`series` *`serie`* `=` *`matrice`*

dove *`serie`* è il nome della serie da creare e *`matrice`* è il nome della matrice da cui copiare i valori (che può essere seguita da un'espressione di selezione). Ecco alcuni esempi:

```
series s = x
series u1 = U[,1]
```

Si assume che x e U siano matrici preesistenti. Nel secondo esempio, la serie u1 è formata dalla prima colonna della matrice U.

Affinché questa operazione funzioni, la matrice (o il risultato dell'operazione di selezione matriciale) deve essere un vettore con lunghezza pari alla lunghezza del dataset attuale, n, oppure alla lunghezza dell'intervallo del campione attuale, n'. Se $n' < n$, verranno estratti dalla matrice solo n' elementi; se la matrice comprende n elementi, vengono usati i n' valori a partire dall'elemento t_1, dove t_1 rappresenta l'osservazione iniziale dell'intervallo del campione. Ad ogni valore della serie che non proviene dalla matrice viene assegnato il codice di valore mancante.

12.10 Matrici e liste

Per facilitare la manipolazione di liste di variabili (si veda il capitolo 11), è possibile convertire liste in matrici e viceversa. Nella sezione precedente 12.1 è stato presentato il modo di creare una matrice a partire da una lista di variabili, come in

```
matrix M = { listname }
```

Questa formulazione, con il nome della lista tra parentesi graffe, crea una matrice le cui colonne contengono le variabili indicate nella lista. Di seguito viene presentato un altra operazione: dicendo

```
matrix M = listname
```

(senza le parentesi graffe) si ottiene un vettore riga i cui elementi sono i numeri identificativi delle variabili della lista. Questo caso speciale di generazione di matrici non può essere utilizzato all'interno di altre espressioni: la sintassi deve essere quella appena vista, ossia la semplice assegnazione di una lista a una matrice.

Per lavorare nella direzione opposta, è possibile includere una matrice nella parte destra di un'espressione che definisce una lista, come in

```
list Xl = M
```

dove M è una matrice che deve avere le dimensioni adatte per la conversione, ossia, deve essere un vettore riga o colonna che contenga valori interi e non negativi, nessuno dei quali ecceda il massimo numero identificativo di una variabile (serie o scalare) nel dataset attuale.

L'esempio 12.3 illustra l'uso di questo tipo di conversione per "normalizzare" una lista, spostando la costante (variabile 0) al primo posto.

12.11 Eliminazione di matrici

Per eliminare una matrice, basta scrivere

```
delete M
```

dove M è il nome della matrice da eliminare.

12.12 Stampa di matrici

Per stampare una matrice, basta indicare solo il suo nome su una riga, oppure usare il comando print:

```
matrix M = mnormal(100,2)
M
print M
```

12.13 Esempio: OLS usando le matrici

L'esempio 12.4 mostra come usare i metodi matriciali per replicare la funzionalità OLS di gretl.

Esempio 12.3: Manipolazione di una lista

```
function normalize_list (matrix *x)
  # Se la matrice (che rappresenta una lista) contiene la variabile 0
  # ma non nella prima posizione, questa viene spostata nella prima posizione

  if (x[1] != 0)
     scalar k = cols(x)
     loop for (i=2; i<=k; i++) --quiet
        if (x[i] = 0)
            x[i] = x[1]
            x[1] = 0
            break
         endif
     end loop
  end if
end function

open data9-7
list Xl = 2 3 0 4
matrix x = Xl
normalize_list(&x)
list Xl = x
```

Esempio 12.4: OLS usando le matrici

```
open data4-1
matrix X = { const, sqft }
matrix y = { price }
matrix b = invpd(X'X) * X'y
printf "vettore dei coefficienti stimati\n"
b
matrix u = y - X*b
scalar SSR = u'u
scalar s2 = SSR / (rows(X) - rows(b))
matrix V = s2 * inv(X'X)
V
matrix se = sqrt(diag(V))
printf "errori standard stimati\n"
se
# Confronto con la funzione OLS di gretl
ols price const sqft --vcv
```

abs	atan	cdemean	cdiv	cholesky	cmult
cnorm	cols	cos	det	diag	dnorm
dsort	eigengen	eigensym	exp	fft	ffti
gamma	ginv	I	imaxc	imaxr	iminc
iminr	infnorm	int	inv	invpd	ldet
lngamma	log	mcorr	mcov	maxc	maxr
meanc	meanr	mexp	minc	minr	mlag
mnormal	mols	mread	mshape	muniform	mwrite
mxtab	nullspace	onenorm	ones	princomp	qform
qnorm	qrdecomp	rank	rcond	rows	seq
sin	sort	sqrt	sumc	sumr	svd
tan	tr	transp	unvech	values	vec
vech	zeros				

Tabella 12.5: Elenco alfabetico delle funzioni matriciali

Capitolo 13

Esempi svolti

Questo capitolo spiega come eseguire alcuni compiti comuni (e altri meno comuni) usando il linguaggio di scripting di gretl. Alcune delle tecniche mostrate, ma non tutte, sono utilizzabili anche attraverso l'interfaccia grafica del programma; sebbene questa possa sembrare più intuitiva e facile da utilizzare a prima vista, incoraggiamo gli utenti a sfruttare le potenzialità del linguaggio di scripting di gretl, dopo aver preso confidenza col programma.

13.1 Gestione dei dataset

Periodicità "strane"

Problema: si hanno dati rilevati ogni 3 minuti a partire dalle ore 9, ossia ogni ora è suddivisa in 20 periodi.

Soluzione:

```
setobs 20 9:1 --special
```

Commento: ora le funzioni come `sdiff()` (differenza "stagionale") o i metodi di stima come l'ARIMA stagionale funzioneranno come ci si aspetta.

Aiuto, i miei dati sono all'incontrario!

Problema: Gretl si aspetta che le serie storiche siano in ordine cronologico (l'osservazione più recente per ultima), ma sono stati importati da una fonte esterna dei dati ordinati in modo inverso (l'osservazione più recente per prima).

Soluzione:

```
setobs 1 1 --cross-section
genr ordinamento = -obs
dataset sortby ordinamento
setobs 1 1950 --time-series
```

Commento: la prima riga è richiesta solo se il dataset in uso è interpretato come serie storiche: occorre rimuovere questo tipo di interpretazione che (ovviamente) non permette di usare il comando `dataset sortby`. Le due righe successive invertono l'ordine dei dati, usando come variabile di ordinamento il negativo della variabile indice interna `obs`. L'ultima riga è solo un esempio: imposta l'interpretazione del dataset come serie storiche annuali che iniziano dall'anno 1950.

Se si ha un dataset ordinato correttamente per tutte le variabili tranne una, è possibile correggere l'ordinamento di una sola variabile nel modo seguente:

```
genr x = sortby(-obs, x)
```

Eliminare osservazioni mancanti in modo selettivo

Problema: si ha un dataset con molte variabili e si vuole restringere il campione a quelle osservazioni per cui non ci sono osservazioni mancanti per nessuna delle variabili `x1`, `x2` e `x3`.

Soluzione:

```
list X = x1 x2 x3
genr sel = ok(X)
smpl sel --restrict
```

Commento: ora è possibile salvare il file con il comando `store` per preservare una versione ristretta del dataset.

Operazioni diverse a seconda dei valori di una variabile

Problema: si ha una variabile discreta `d` e si vuole eseguire alcuni comandi (ad esempio, stimare un modello) suddividendo il campione a seconda dei valori di `d`.

Soluzione:

```
matrix vd = values(d)
m = rows(vd)
loop for i=1..m
  scalar sel = vd[i]
  smpl (d=sel) --restrict --replace
  ols y const x
end loop
smpl full
```

Commento: L'ingrediente principale è un loop, all'interno del quale è possibile eseguire tutte le istruzioni volute per ogni valore di `d`, a patto che siano istruzioni il cui uso è consentito all'interno di un loop.

13.2 Creazione e modifica di variabili

Generazione di una variabile dummy per una specifica osservazione

Problema: Generare $d_t = 0$ per tutte le osservazioni tranne una, per cui vale $d_t = 1$.

Soluzione:

```
genr d = (t="1984:2")
```

Commento: La variabile interna `t` viene usata per riferirsi alle osservazioni sotto forma di stringa, quindi se si ha un campione cross-section si può usare semplicemente `d = (t="123")`; ovviamente, se il dataset ha delle etichette per i dati si può usare l'etichetta corrispondente. Ad esempio, se si apre il dataset `mrw.gdt`, fornito con `gretl`, si può generare una variabile dummy per l'Italia usando

```
genr DIta = (t="Italy")
```

Si noti che questo metodo non richiede di creare uno script: è possibile inserire i comandi appena visti usando il comando dell'interfaccia grafica "Aggiungi/Definisci nuova variabile".

Generazione di un ARMA(1,1)

Problema: Generare $y_t = 0.9y_{t-1} + \varepsilon_t - 0.5\varepsilon_{t-1}$, con $\varepsilon_t \sim NIID(0,1)$.

Soluzione:

```
alpha = 0.9
theta = -0.5
series e = normal()
series y = 0
series y = alpha * y(-1) + e + theta * e(-1)
```

Commento: L'istruzione `series y = 0` è necessaria perché l'istruzione successiva valuta `y` ricorsivamente, quindi occorre impostare `y[1]`. Si noti che occorre usare la parola chiave `series`, invece di scrivere `genr y = 0` o semplicemente `y = 0`, per assicurarsi che `y` sia una serie e non uno scalare.

Assegnazione condizionale

Problema: Generare y_t secondo la regola seguente:

$$y_t = \begin{cases} x_t & \text{for} \quad d_t = 1 \\ z_t & \text{for} \quad d_t = 0 \end{cases}$$

Soluzione:

```
series y = d ? x : z
```

Commento: ci sono varie alternative a quella presentata. La prima è quella di forza bruta usando i loop. Un'altra, più efficiente ma ancora subottimale, è quella di usare `y = d*x + (1-d)*z`. L'operatore di assegnazione condizionale ternario non solo rappresenta la soluzione numericamente più efficiente, ma è anche quella di più semplice lettura, una volta che si è abituati alla sua sintassi, che per alcuni lettori può ricordare quella della funzione `=IF()` nei fogli di calcolo.

13.3 Trucchi utili

Dummy di interazione

Problema: si vuole stimare il modello $y_i = \mathbf{x}_i\beta_1 + \mathbf{z}_i\beta_2 + d_i\beta_3 + (d_i \cdot \mathbf{z}_i)\beta_4 + \varepsilon_t$, dove d_i è una variabile dummy, mentre $\mathbf{x}_i$ e $\mathbf{z}_i$ sono vettori di variabili esplicative.

Soluzione:

```
list X = x1 x2 x3
list Z = z1 z2
list dZ = null
loop foreach i Z
  series d$i = d * $i
  list dZ = dZ d$i
end loop

ols y X Z d dZ
```

Commento: incredibile cosa si può fare con la sostituzione delle stringhe, vero?

Volatilità percepita

Problema: avendo dati raccolti ogni minuto, si vuole calcolare la "realized volatility" per ogni ora come $RV_t = \frac{1}{60}\sum_{\tau=1}^{60} y_{t:\tau}^2$. Il campione parte da 1:1.

Soluzione:

```
smpl full
genr time
genr minute = int(time/60) + 1
genr second = time % 60
setobs minute second --panel
genr rv = psd(y)^2
setobs 1 1
smpl second=1 --restrict
store foo rv
```

Commento: qui facciamo credere a gretl che il dataset sia di tipo panel, dove i minuti sono le "unità" e i secondi sono il "tempo"; in questo modo possiamo utilizzare la funzione speciale `psd()` (panel standard deviation). Quindi eliminiamo semplicemente tutte le osservazioni tranne una per minuto e salviamo i dati risultanti (`store foo rv` significa "salva nel file di dati `foo.gdt` la serie `rv`").

Parte II

Metodi econometrici

Capitolo 14

Stima robusta della matrice di covarianza

14.1 Introduzione

Si consideri (ancora una volta) il modello di regressione lineare

$$y = X\beta + u \tag{14.1}$$

dove y e u sono vettori di dimensione T, X è una matrice $T \times k$ di regressori, e β è un vettore di parametri di dimensione k. Come è noto, lo stimatore di β dato dai minimi quadrati ordinari (OLS) è

$$\hat{\beta} = (X'X)^{-1}X'y \tag{14.2}$$

Se la condizione $E(u|X) = 0$ è soddisfatta, questo stimatore è non distorto; sotto condizioni meno restrittive, lo stimatore è distorto ma consistente. È semplice mostrare che quando lo stimatore OLS non è distorto (ossia quando $E(\hat{\beta} - \beta) = 0$), la sua varianza è

$$\mathrm{Var}(\hat{\beta}) = E\left((\hat{\beta} - \beta)(\hat{\beta} - \beta)'\right) = (X'X)^{-1}X'\Omega X(X'X)^{-1} \tag{14.3}$$

dove $\Omega = E(uu')$ è la matrice di covarianza dei termini di errore.

Sotto l'ipotesi che i termini di errore siano indipendenti e identicamente distribuiti (iid), si può scrivere $\Omega = \sigma^2 I$, dove σ^2 è la varianza (comune) degli errori (e le covarianze sono zero). In questo caso, la (14.3) si riduce alla "classica" formula,

$$\mathrm{Var}(\hat{\beta}) = \sigma^2(X'X)^{-1} \tag{14.4}$$

Se la condizione iid non è soddisfatta, ne derivano due conseguenze. Per prima cosa è possibile costruire uno stimatore più efficiente di quello OLS, ad esempio un qualche tipo di stimatore FGLS (Feasible Generalized Least Squares). Inoltre, la semplice formula "classica" per la varianza dello stimatore dei minimi quadrati non è più corretta, e quindi gli errori standard da essa derivati (ossia le radici quadrate degli elementi sulla diagonale della matrice definita dalla 14.4) non sono strumenti corretti per l'inferenza statistica.

Nella storia recente dell'econometria ci sono due approcci principali al problema rappresentato dagli errori non iid. L'approccio "tradizionale" consiste nell'usare uno stimatore FGLS. Ad esempio, se l'ipotesi iid viene violata a causa di una dipendenza di tipo temporale tra i termini di errore, e se si ha ragione di pensare che questo si possa modellare con un processo di autocorrelazione del prim'ordine, si potrebbe utilizzare un metodo di stima AR(1), come quello di Cochrane–Orcutt, o di Hildreth–Lu, o di Prais–Winsten. Se il problema sta nel fatto che la varianza dell'errore non è costante tra le osservazioni, si potrebbe stimare la varianza come funzione delle variabili indipendenti e usare quindi i minimi quadrati ponderati, prendendo come pesi i reciproci delle varianze stimate.

Mentre questi metodi sono tuttora utilizzati, un approccio alternativo sta guadagnando favore: usare lo stimatore OLS ma calcolare gli errori standard (o più in generale le matrici di covarianza) in modo che siano robusti rispetto alle deviazioni dall'ipotesi iid. Questo approccio è spesso associato all'uso di grandi dataset, abbastanza grandi da suggerire la validità della proprietà di consistenza (asintotica) dello stimatore OLS, ed è stato reso possibile anche dalla disponibilità di sempre maggiori potenze di calcolo: il calcolo degli errori standard robusti e l'uso di grandi dataset erano compiti scoraggianti fino a qualche tempo fa, ma ora non pongono alcun problema. Un punto a favore di questo approccio consiste nel fatto che, mentre la stima FGLS offre un vantaggio in termini di efficienza, spesso richiede di fare delle ipotesi statistiche aggiuntive, che potrebbero non essere giustificate, che potrebbe essere difficile testare, e che potrebbero

mettere in discussione la consistenza dello stimatore; ad esempio, l'ipotesi di "fattore comune" che è implicata dalle tradizionali "correzioni" per i termini di errore autocorrelati.

Introduction to Econometrics di James Stock e Mark Watson illustra questo approccio in modo comprensibile agli studenti universitari: molti dei dataset usati sono composti da migliaia o decine di migliaia di osservazioni, la stima FGLS è poco considerata, mentre si pone l'enfasi sull'uso di errori standard robusti (in effetti la discussione degli errori standard classici nel caso di omoschedasticità è confinata in un'appendice).

Può essere utile passare in rassegna le opzioni fornite da gretl per la stima robusta della matrice di covarianza. Il primo punto da notare è che gretl produce errori standard "classici" come risultato predefinito (in tutti i casi tranne quello della stima GMM). In modalità a riga di comando (o negli script) è possibile ottenere gli errori standard robusti aggiungendo l'opzione `--robust` ai comandi di stima. Se si usa l'interfaccia grafica, le finestre di dialogo per la specificazione dei modelli contengono una casella "Errori standard robusti", insieme a un pulsante "Configura" che viene attivato se si seleziona la casella. Premendo il pulsante si ottiene una finestra (raggiungibile anche attraverso il menù principale: Strumenti → Preferenze → Generali → HCCME), da cui è possibile scegliere tra diverse varianti di stima robusta, e anche rendere predefinita la stima robusta.

Le specifiche opzioni disponibili dipendono dalla natura dei dati in esame (cross-section, serie storiche o panel) e anche, in qualche misura, dalla scelta dello stimatore (anche se finora si è parlato di errori standard robusti in relazione allo stimatore OLS, questi possono essere usati anche con altri stimatori). Le successive sezioni di questo capitolo presentano argomenti caratteristici di ognuno dei tre tipi di dati appena ricordati. Dettagli ulteriori riguardanti la stima della matrice di covarianza nel contesto GMM si trovano nel capitolo 18.

Per concludere questa introduzione, ricordiamo ancora quello che gli "errori standard robusti" possono e non possono garantire: possono fornire un'inferenza statistica asintoticamente valida in modelli che sono correttamente specificati, ma in cui gli errori non sono iid. Il termine "asintotico" significa che questo approccio può essere poco utile su piccoli campioni. Il termine "correttamente specificati" significa che non si ha una bacchetta magica: se il termine di errore è correlato con i regressori, le stime dei parametri sono distorte e inconsistenti, gli errori standard robusti non possono risolvere questo problema.

14.2 Dati cross-section e HCCME

Con dati cross-section, la causa più comune di violazione dell'ipotesi iid è data dall'eteroschedasticità (varianza non costante)[1]. Il alcuni casi è possibile fare delle ipotesi plausibili sulla forma specifica dell'eteroschedasticità e quindi applicare una correzione ad hoc, ma di solito non si sa con che tipo di eteroschedasticità si ha a che fare. Vogliamo quindi trovare uno stimatore della matrice di covarianza delle stime dei parametri che mantenga la sua validità, almeno dal punto di vista asintotico, anche in caso di eteroschedasticità. Che questo sia possibile non è ovvio a priori, ma White (1980) ha mostrato che

$$\widehat{\mathrm{Var}}_{\mathrm{h}}(\hat{\beta}) = (X'X)^{-1}X'\hat{\Omega}X(X'X)^{-1} \tag{14.5}$$

fa al caso nostro (come al solito in statistica dobbiamo dire "sotto alcune condizioni", ma in questo caso le condizioni non sono molto restrittive). $\hat{\Omega}$ è una matrice diagonale i cui elementi diversi da zero possono essere stimati usando i quadrati dei residui OLS. White ha chiamato la (14.5) uno stimatore HCCME (heteroskedasticity-consistent covariance matrix estimator).

Davidson e MacKinnon (2004, capitolo 5) offrono una discussione utile di alcune varianti dello stimatore HCCME di White. Chiamano HC_0 la variante originale della (14.5), in cui gli elementi diagonali di $\hat{\Omega}$ sono stimati direttamente con i quadrati dei residui OLS, $\hat{u}_t^2$ (gli errori standard associati sono chiamati spesso "errori standard di White"). Le varie estensioni dell'approccio di White hanno in comune un punto: l'idea che i quadrati dei residui OLS siano probabilmente "troppo piccoli" in media. Questa idea è piuttosto intuitiva: le stime OLS dei parametri, $\hat{\beta}$, per

[1] In alcuni contesti speciali, il problema può essere invece l'autocorrelazione spaziale. Gretl non ha funzioni per gestire questo caso, che quindi verrà trascurato in questa trattazione.

costruzione soddisfano il criterio che la somma dei quadrati dei residui

$$\sum \hat{u}_t^2 = \sum \left(y_t - X_t\hat{\beta}\right)^2$$

è minimizzata, dati X e y. Si supponga che $\hat{\beta} \neq \beta$. È quasi certo che sia così: anche se OLS non è distorto, sarebbe un miracolo se i $\hat{\beta}$ calcolati da un campione finito fossero esattamente uguali a β. Ma in questo caso la somma dei quadrati dei veri errori (non osservati), $\sum u_t^2 = \sum (y_t - X_t\beta)^2$ è certamente maggiore di $\sum \hat{u}_t^2$. Le varianti di HC_0 partono da questo punto nel modo seguente:

- HC_1: applica una correzione per gradi di libertà, moltiplicando la matrice HC_0 per $T/(T-k)$.
- HC_2: invece di usare $\hat{u}_t^2$ per gli elementi diagonali di $\hat{\Omega}$, usa $\hat{u}_t^2/(1-h_t)$, dove $h_t = X_t(X'X)^{-1}X_t'$, il t^{esimo} elemento diagonale della matrice di proiezione, P, che ha la proprietà che $P \cdot y = \hat{y}$. La rilevanza di h_t sta nel fatto che se la varianza di tutti gli u_t è σ^2, il valore atteso di $\hat{u}_t^2$ è $\sigma^2(1-h_t)$, o in altre parole, il rapporto $\hat{u}_t^2/(1-h_t)$ ha un valore atteso di σ^2. Come mostrano Davidson e MacKinnon, $0 \leq h_t < 1$ per ogni t, quindi questa correzione non può ridurre gli elementi diagonali di $\hat{\Omega}$ e in generale li corregge verso l'alto.
- HC_3: Usa $\hat{u}_t^2/(1-h_t)^2$. Il fattore aggiuntivo $(1-h_t)$ nel denominatore, relativo a HC_2, può essere giustificato col fatto che le osservazioni con ampia varianza tendono a esercitare una grossa influenza sulle stime OLS, così che i corrispondenti residui tendono ad essere sottostimati. Si veda Davidson e MacKinnon per ulteriori dettagli.

I rispettivi meriti di queste varianti sono stati analizzati sia dal punto di vista teorico che attraverso simulazioni, ma sfortunatamente non c'è un consenso preciso su quale di esse sia "la migliore". Davidson e MacKinnon sostengono che l'originale HC_0 probabilmente si comporta peggio delle altre varianti, tuttavia gli "errori standard di White" sono citati più spesso delle altre varianti più sofisticate e quindi per motivi di comparabilità, HC_0 è lo stimatore HCCME usato da gretl in modo predefinito.

Se si preferisce usare HC_1, HC_2 o HC_3, è possibile farlo in due modi. In modalità script, basta eseguire ad esempio

```
set hc_version 2
```

Con l'interfaccia grafica, basta andare nella finestra di configurazione di HCCME come mostrato sopra e impostare come predefinita una delle varianti.

14.3 Serie storiche e matrici di covarianza HAC

L'eteroschedasticità può essere un problema anche con le serie storiche, ma raramente è l'unico, o il principale, problema.

Un tipo di eteroschedasticità è comune nelle serie storiche macroeconomiche, ma è abbastanza semplice da trattare: nel caso di serie con una forte tendenza, come il prodotto interno lordo, il consumo o l'investimento aggregato, e simili, alti valori della variabile sono probabilmente associati ad alta variabilità in termini assoluti. Il rimedio ovvio, usato da molti studi macroeconomici, consiste nell'usare i logaritmi di queste serie, al posto dei livelli. A patto che la variabilità *proporzionale* di queste serie rimanga abbastanza costante nel tempo, la trasformazione logaritmica è efficace.

Altre forme di eteroschedasticità possono sopravvivere alla trasformazione logaritmica e richiedono un trattamento distinto dal calcolo degli errori standard robusti. Ad esempio l'*eteroschedasticità autoregressiva condizionale* riscontrabile ad esempio nelle serie dei prezzi di borsa, dove grandi disturbi sul mercato possono causare periodi di aumento della volatilità; fenomeni come questo giustificano l'uso di specifiche strategie di stima, come nei modelli GARCH (si veda il capitolo 20).

Nonostante tutto questo, è possibile che un certo grado di eteroschedasticità sia presente nelle serie storiche: il punto chiave è che nella maggior parte dei casi, questa è probabilmente combinata con un certo grado di correlazione seriale (autocorrelazione), e quindi richiede un trattamento speciale. Nell'approccio di White, $\hat{\Omega}$, la matrice di covarianza stimata degli u_t, rimane

diagonale: le varianze, $E(u_t^2)$, possono differire per t, ma le covarianze, $E(u_t u_s)$, sono sempre zero. L'autocorrelazione nelle serie storiche implica che almeno alcuni degli elementi fuori dalla diagonale di $\hat{\Omega}$ possono essere diversi da zero. Questo introduce una complicazione evidente e un ulteriore termine da tenere presente: le stime della matrice di covarianza che sono asintoticamente valide anche in presenza di eteroschedasticità e autocorrelazione nel processo di errore vengono definite HAC (heteroskedasticity and autocorrelation consistent).

Il tema della stima HAC è trattato in termini più tecnici nel capitolo 18, qui cerchiamo di fornire un'intuizione basilare. Iniziamo da un commento generale: l'autocorrelazione dei residui non è tanto una proprietà dei dati, quanto il sintomo di un modello inadeguato. I dati possono avere proprietà persistenti nel tempo, ma se imponiamo un modello che non tiene conto adeguatamente di questo aspetto, finiamo con avere disturbi autocorrelati. Al contrario, spesso è possibile mitigare o addirittura eliminare il problema dell'autocorrelazione includendo opportune variabili ritardate in un modello di serie storiche, o in altre parole specificando meglio la dinamica del modello. La stima HAC *non* dovrebbe essere considerata il primo strumento per affrontare l'autocorrelazione del termine di errore.

Detto questo, la "ovvia" estensione dello stimatore HCCME di White al caso di errori autocorrelati sembra questa: stimare gli elementi fuori dalla diagonale di $\hat{\Omega}$ (ossia le autocovarianze, $E(u_t u_s)$) usando, ancora una volta, gli opportuni residui OLS: $\hat{\omega}_{ts} = \hat{u}_t \hat{u}_s$. Questo approccio sembra giusto, ma richiede una correzione importante: cerchiamo uno stimatore *consistente*, che converga verso il vero Ω quando l'ampiezza del campione tende a infinito. Campioni più ampi permettono di stimare più elementi di ω_{ts} (ossia, per t e s più separati nel tempo), ma *non* forniscono più informazione a proposito delle coppie ω_{ts} più distanti nel tempo, visto che la massima separazione nel tempo cresce anch'essa al crescere della dimensione del campione. Per assicurare la consistenza, dobbiamo confinare la nostra attenzione ai processi che esibiscono una dipendenza limitata nel tempo, o in altre parole interrompere il calcolo dei valori $\hat{\omega}_{ts}$ a un certo valore massimo di $p = t - s$ (dove p è trattato come una funzione crescente dell'ampiezza campionaria, T, anche se non è detto che cresca proporzionalmente a T).

La variante più semplice di questa idea consiste nel troncare il calcolo a un certo ordine di ritardo finito p, che cresce ad esempio come $T^{1/4}$. Il problema è che la matrice $\hat{\Omega}$ risultante potrebbe non essere definita positiva, ossia potremmo ritrovarci con delle varianze stimate negative. Una soluzione a questo problema è offerta dallo stimatore di Newey–West (Newey e West, 1987), che assegna pesi declinanti alle autocovarianze campionarie, man mano che la separazione temporale aumenta.

Per capire questo punto può essere utile guardare più da vicino la matrice di covarianza definita nella (14.5), ossia,

$$(X'X)^{-1}(X'\hat{\Omega}X)(X'X)^{-1}$$

Questo è noto come lo stimatore "sandwich". La fetta di pane è $(X'X)^{-1}$, ossia una matrice $k \times k$, che è anche l'ingrediente principale per il calcolo della classica matrice di covarianza. Il contenuto del sandwich è

$$\underset{(k\times k)}{\hat{\Sigma}} = \underset{(k\times T)}{X'} \underset{(T\times T)}{\hat{\Omega}} \underset{(T\times k)}{X}$$

Poiché $\Omega = E(uu')$, la matrice che si sta stimando può essere scritta anche come

$$\Sigma = E(X'u\, u'X)$$

che esprime Σ come la covarianza di lungo periodo del vettore casuale $X'u$ di dimensione k.

Dal punto di vista computazionale, non è necessario salvare la matrice $T \times T$ $\hat{\Omega}$, che può essere molto grande. Piuttosto, si può calcolare il contenuto del sandwich per somma, come

$$\hat{\Sigma} = \hat{\Gamma}(0) + \sum_{j=1}^{p} w_j \left(\hat{\Gamma}(j) + \hat{\Gamma}'(j)\right)$$

dove la matrice $k \times k$ di autocovarianza campionaria $\hat{\Gamma}(j)$, per $j \geq 0$, è data da

$$\hat{\Gamma}(j) = \frac{1}{T} \sum_{t=j+1}^{T} \hat{u}_t \hat{u}_{t-j} X_t' X_{t-j}$$

e w_j è il peso dato dall'autocovarianza al ritardo $j > 0$.

Rimangono due questioni. Come determiniamo esattamente la massima lunghezza del ritardo (o "larghezza di banda") p dello stimatore HAC? E come determiniamo esattamente i pesi w_j? Torneremo presto sul (difficile) problema della larghezza di banda, ma per quanto riguarda i pesi, gretl offre tre varianti. Quella predefinita è il kernel di Bartlett, come è usato da Newey e West. Questo stabilisce che

$$w_j = \begin{cases} 1 - \frac{j}{p+1} & j \leq p \\ 0 & j > p \end{cases}$$

in modo che i pesi declinino linearmente mentre j aumenta. Le altre due opzioni sono il kernel di Parzen e il kernel QS (Quadratic Spectral). Per il kernel di Parzen,

$$w_j = \begin{cases} 1 - 6a_j^2 + 6a_j^3 & 0 \leq a_j \leq 0.5 \\ 2(1 - a_j)^3 & 0.5 < a_j \leq 1 \\ 0 & a_j > 1 \end{cases}$$

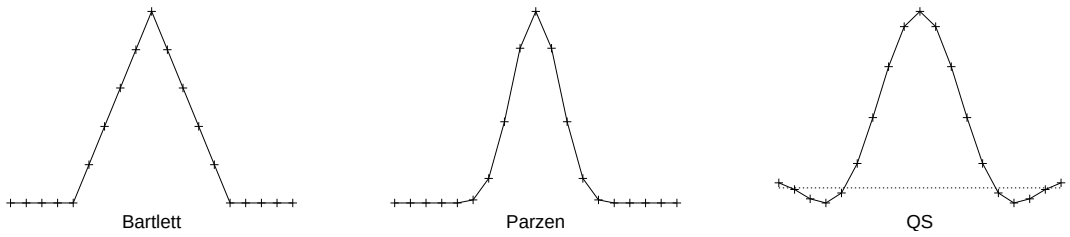
dove $a_j = j/(p+1)$, mentre per il kernel QS

$$w_j = \frac{25}{12\pi^2 d_j^2} \left(\frac{\sin m_j}{m_j} - \cos m_j \right)$$

dove $d_j = j/p$ e $m_j = 6\pi d_i/5$.

La figura 14.1 mostra i pesi generati da questi kernel per $p = 4$ e j che va da 1 a 9.

Figura 14.1: Tre kernel per HAC



In gretl è possibile scegliere il kernel usando il comando `set` col parametro `hac_kernel`:

```
set hac_kernel parzen
set hac_kernel qs
set hac_kernel bartlett
```

Scelta della larghezza di banda HAC

La teoria asintotica sviluppata da Newey, West ed altri ci dice in termini generali come la larghezza di banda HAC, p, deve crescere in relazione all'ampiezza campionaria, T, ossia dice che p dovrebbe crescere proporzionalmente a qualche potenza frazionaria di T. Purtroppo questo non è di molto aiuto quando nella pratica si ha a che fare con un dataset di ampiezza fissa. Sono state suggerite varie regole pratiche, due delle quali sono implementate da gretl. L'impostazione predefinita è $p = 0.75T^{1/3}$, come raccomandato da Stock e Watson (2003). Un'alternativa è $p = 4(T/100)^{2/9}$, come raccomandato in Wooldridge (2002b). In entrambi i casi si prende la parte intera del risultato. Queste varianti sono chiamate rispettivamente `nw1` e `nw2` nel contesto del comando `set` col parametro `hac_lag`. Ossia, è possibile impostare la versione data da Wooldridge con il comando

```
set hac_lag nw2
```

Come mostrato nella Tabella 14.1 la scelta tra `nw1` e `nw2` non causa rilevanti differenze.

È anche possibile specificare un valore numerico fisso per p, come in

T	p (nw1)	p (nw2)
50	2	3
100	3	4
150	3	4
200	4	4
300	5	5
400	5	5

Tabella 14.1: Larghezza di banda HAC: confronto tra due regole pratiche

```
set hac_lag 6
```

Inoltre è possibile impostare un valore diverso per il kernel QS (visto che questo non deve essere necessariamente un valore intero). Ad esempio:

```
set qs_bandwidth 3.5
```

Prewhitening e scelta della larghezza di banda basata sui dati

Un approccio alternativo per trattare l'autocorrelazione dei residui consiste nell'attaccare il problema da due fronti. L'intuizione alla base di questa tecnica, nota come *VAR prewhitening* (Andrews e Monahan, 1992) può essere illustrata con un semplice esempio. Sia x_t una serie di variabili casuali con autocorrelazione del prim'ordine

$$x_t = \rho x_{t-1} + u_t$$

Si può dimostrare che la varianza di lungo periodo di x_t è

$$V_{LR}(x_t) = \frac{V_{LR}(u_t)}{(1-\rho)^2}$$

Nella maggior parte dei casi, u_t è meno autocorrelato di x_t, quindi dovrebbe richiedere una minore larghezza di banda. La stima di $V_{LR}(x_t)$ può quindi procedere in tre passi: (1) stimare ρ; (2) ottenere una stima HAC di $\hat{u}_t = x_t - \hat{\rho} x_{t-1}$; (3) dividere il risultato per $(1-\rho)^2$.

Applicare questo approccio al nostro problema implica stimare un'autoregressione vettoriale (VAR) di ordine finito sulle variabili vettoriali $\xi_t = X_t \hat{u}_t$. In generale, il VAR può essere di ordine qualsiasi, ma nella maggior parte dei casi è sufficiente l'ordine 1; lo scopo non è quello di produrre un modello preciso per ξ_t, ma solo quello di catturare la maggior parte dell'autocorrelazione. Quindi viene stimato il VAR seguente

$$\xi_t = A\xi_{t-1} + \varepsilon_t$$

Una stima della matrice $X'\Omega X$ può essere ottenuta con

$$(I - \hat{A})^{-1} \hat{\Sigma}_\varepsilon (I - \hat{A}')^{-1}$$

dove $\hat{\Sigma}_\varepsilon$ è uno stimatore HAC, applicato ai residui del VAR.

In gretl è possibile usare il prewhitening con

```
set hac_prewhiten on
```

Al momento non è possibile calcolare un VAR iniziale con un ordine diverso da 1.

Un ulteriore miglioramento di questo approccio consiste nello scegliere la larghezza di banda in base ai dati. Intuitivamente, ha senso che la larghezza di banda non tenga conto soltanto dell'ampiezza campionaria, ma anche delle proprietà temporali dei dati (e anche del kernel scelto). Un metodo non parametrico di scelta è stato proposto da Newey e West (1994) ed è spiegato bene e in modo sintetico da Hall (2005). Questa opzione può essere abilitata in gretl con il comando

```
set hac_lag nw3
```

ed è abilitata in modo predefinito quando si seleziona il prewhitening, ma è possibile modificarla utilizzando un valore numerico specifico per `hac_lag`.

Anche il metodo basato sui dati proposto da Newey–West non identifica univocamente la larghezza di banda per una data ampiezza del campione. Il primo passo consiste nel calcolare una serie di covarianze dei residui, e la lunghezza di questa serie è una funzione dell'ampiezza campionaria, ma solo per un certo multiplo scalare; ad esempio, è data da $O(T^{2/9})$ per il kernel di Bartlett. Gretl usa un multiplo implicito pari a 1.

14.4 Problemi speciali con dati panel

Visto che i dati panel hanno sia caratteristiche di serie storiche sia caratteristiche di dati cross-section, ci si può aspettare che in generale la stima robusta della matrice di covarianza debba richiedere di gestire sia l'eteroschedasticità che l'autocorrelazione (l'approccio HAC). Inoltre ci sono altre caratteristiche dei dati panel che richiedono attenzione particolare:

- La varianza del termine di errore può differire tra le unità cross-section.
- La covarianza degli errori tra le unità può essere diversa da zero in ogni periodo temporale.
- Se non si rimuove la variazione "between", gli errori possono esibire autocorrelazione, non nel senso classico delle serie storiche, ma nel senso che l'errore medio per l'unità i può essere diverso da quello per l'unità j (questo è particolarmente rilevante quando il metodo di stima è pooled OLS).

Gretl al momento offre due stimatori robusti per la matrice di covarianza da usare con dati panel, disponibili per modelli stimati con effetti fissi, pooled OLS, e minimi quadrati a due stadi. Lo stimatore robusto predefinito è quello suggerito da Arellano (2003), che è HAC a patto che il panel sia del tipo "n grande, T piccolo" (ossia si osservano molte unità per pochi periodi). Lo stimatore di Arellano è

$$\hat{\Sigma}_{\mathrm{A}} = (X'X)^{-1} \left(\sum_{i=1}^{n} X_i' \hat{u}_i \hat{u}_i' X_i \right) (X'X)^{-1}$$

dove X è la matrice dei regressori (con le medie di gruppo sottratte, nel caso degli effetti fissi), $\hat{u}_i$ denota il vettore dei residui per l'unità i, e n è il numero delle unità cross-section. Cameron e Trivedi (2005) difendono l'uso di questo stimatore, notando che il classico HCCME di White può produrre errori standard artificialmente bassi in un contesto panel, perché non tiene conto dell'autocorrelazione.

Nei casi in cui l'autocorrelazione non è un problema, lo stimatore proposto da Beck e Katz (1995) e discusso da Greene (2003, capitolo 13) può essere appropriato. Questo stimatore, che tiene conto della correlazione contemporanea tra le unità e l'eteroschedasticità per unità, è

$$\hat{\Sigma}_{\mathrm{BK}} = (X'X)^{-1} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} \hat{\sigma}_{ij} X_i' X_j \right) (X'X)^{-1}$$

Le covarianze $\hat{\sigma}_{ij}$ sono stimate con

$$\hat{\sigma}_{ij} = \frac{\hat{u}_i' \hat{u}_j}{T}$$

dove T è la lunghezza della serie storica per ogni unità. Beck e Katz chiamano gli errori standard associati "Panel-Corrected Standard Errors" (PCSE). Per usare questo stimatore in gretl basta eseguire il comando

```
set pcse on
```

Per reimpostare come predefinito lo stimatore di Arellano occorre eseguire

```
set pcse off
```

Si noti che a prescindere dall'impostazione di `pcse`, lo stimatore robusto non è usato a meno che non si aggiunga l'opzione `--robust` ai comandi di stima, o non si selezioni la casella "Robusto" nell'interfaccia grafica.

Capitolo 15

Dati panel

15.1 Stima di modelli panel

Minimi quadrati ordinari

Lo stimatore più semplice per i dati panel è quello "pooled OLS" (ossia i minimi quadrati ordinari utilizzando allo stesso modo tutte le osservazioni del campione). In genere questo stimatore non fornisce risultati ottimali, ma rappresenta un metro di paragone per stimatori più complessi.

Quando si stima un modello usando i minimi quadrati ordinari (OLS) su dati panel, è disponibile un tipo di test aggiuntivo: nel menù `Test` della finestra del modello è il comando "Diagnosi panel", mentre nella versione a riga di comando del programma è il comando `hausman`.

Per eseguire questo test, occorre specificare un modello senza alcuna variabile dummy relativa alle unità cross-section. Il test confronta il semplice modello OLS con le due principali alternative: il modello a effetti fissi e quello a effetti casuali. Questi due modelli sono descritti nella prossima sezione.

I modelli a effetti fissi e casuali

A partire dalla versione 1.6.0 di gretl, i modelli a effetti fissi e a effetti casuali per i dati panel possono essere stimati separatamente. Nell'interfaccia grafica, queste opzioni si trovano nel menù "Modello/Panel/Effetti fissi e casuali", mentre nell'interfaccia a riga di comando occorre usare il comando `panel`, con l'eventuale opzione `--random-effects`.

Di seguito viene spiegata la natura di questi modelli e il modo con cui possono essere stimati in gretl.

La specificazione "pooled OLS" si può scrivere come

$$y_{it} = X_{it}\beta + u_{it} \tag{15.1}$$

dove y_{it} è l'osservazione della variabile dipendente per l'unità cross section i nel periodo t, X_{it} è un vettore $1 \times k$ di variabili indipendenti osservate per l'unità i nel periodo t, β è un vettore $k \times 1$ di parametri, e u_{it} è un termine di errore o di disturbo specifico all'unità i nel periodo t.

I modelli a effetti fissi e casuali rappresentano due modi di scomporre l'errore unitario u_{it}. Per il modello a *effetti fissi* possiamo scrivere $u_{it} = \alpha_i + \varepsilon_{it}$, che produce

$$y_{it} = X_{it}\beta + \alpha_i + \varepsilon_{it} \tag{15.2}$$

Ossia, scomponiamo u_{it} in una componente specifica all'unità e invariante nel tempo, α_i, e un errore specifico all'osservazione, ε_{it}[1].

Gli α_i sono quindi trattati come parametri fissi (intercette y specifiche per le unità) da stimare. Questo può essere fatto includendo una variabile dummy per ogni unità cross-section (e sopprimendo la costante comune), ossia applicando il cosiddetto metodo "Least Squares Dummy Variables" (LSDV). In alternativa, è possibile procedere sottraendo le medie di gruppo da ognuna delle variabili e stimando un modello senza costante. Nell'ultimo caso la variabile dipendente si può scrivere come

$$\tilde{y}_{it} = y_{it} - \bar{y}_i$$

[1] È possibile individuare un'altra componente di u_{it}, ossia w_t, uno shock che sia specifico al periodo temporale, ma comune a tutte le unità. Per semplicità non consideriamo questa possibilità.

La "media di gruppo", $\bar{y}_i$, è definita come

$$\bar{y}_i = \frac{1}{T_i} \sum_{t=1}^{T_i} y_{it}$$

dove T_i è il numero di osservazioni per l'unità i; in modo simile si possono riscrivere le variabili indipendenti. Date le stime dei parametri, $\hat{\beta}$, ottenute usando questi dati espressi in termini di differenza dalla media, possiamo ottenere le stime degli α_i usando

$$\hat{\alpha}_i = \frac{1}{T_i} \sum_{t=1}^{T_i} \left(y_{it} - X_{it}\hat{\beta} \right)$$

Questi due metodi (LSDV oppure usare le differenze dalla media) sono numericamente equivalenti. gretl sceglie il secondo approccio, che consiste nel prendere le differenze dalla media dei dati. Se si ha un piccolo numero di unità cross-section, un grande numero di osservazioni temporali per ogni unità, e un grande numero di regressori, è più economico in termini di memoria utilizzare il metodo LSDV, che può essere implementato manualmente in questo modo:

```
genr unitdum
ols y x du_*
```

Si veda il Capitolo 7 per i dettagli su `unitdum`.

Le stime $\hat{\alpha}_i$ non vengono mostrate fra i risultati del modello tipicamente prodotti da gretl (possono essere molto numerose, e tipicamente non sono molto interessanti), tuttavia è possibile recuperarle dopo aver stimato il modello a effetti fissi, se si vuole. Nell'interfaccia grafica, basta andare nel menù "Salva" della finestra del modello e scegliere "costanti per-unità". Nella modalità a riga di comando, basta eseguire `genr` *nome* = `$ahat`, dove *nome* è il nome che si vuole dare alla nuova serie.

Per il modello a *effetti casuali* possiamo scrivere $u_{it} = v_i + \varepsilon_{it}$, così che il modello diventa

$$y_{it} = X_{it}\beta + v_i + \varepsilon_{it} \tag{15.3}$$

Al contrario del modello a effetti fissi, invece di trattare gli v_i come parametri fissi, li trattiamo come estrazioni casuali da una certa distribuzione di probabilità.

Il famoso teorema di Gauss–Markov, secondo cui OLS è il miglior stimatore lineare non distorto (best linear unbiased estimator, BLUE), dipende dall'ipotesi che il termine di errore sia indipendente e identicamente distribuito (IID). Nel contesto panel l'ipotesi IID significa che $E(u_{it}^2)$, in relazione all'equazione 15.1, sia pari a una costante, σ_u^2, per ogni i e t, mentre la covarianza $E(u_{is}u_{it})$ sia pari a zero dove $s \neq t$ e la covarianza $E(u_{jt}u_{it})$ sia pari a zero dove $j \neq i$.

Se queste ipotesi non sono soddisfatte (ed è probabile che non sia così nel contesto di dati panel), OLS non è lo stimatore più efficiente. È possibile raggiungere una maggior efficienza usando i minimi quadrati generalizzati (GLS) e tenendo conto della struttura di covarianza del termine di errore.

Si considerino le osservazioni sulla stessa unità i in due diversi periodi s e t. Per l'ipotesi precedente, si può concludere che $\mathrm{Var}(u_{is}) = \mathrm{Var}(u_{it}) = \sigma_v^2 + \sigma_\varepsilon^2$, mentre la covarianza tra u_{is} e u_{it} è data da $E(u_{is}u_{it}) = \sigma_v^2$.

In notazione matriciale, è possibile raggruppare tutte le osservazioni T_i per l'unità i nel vettore $\mathbf{y}_i$ e scriverlo come

$$\mathbf{y}_i = \mathbf{X}_i\beta + \mathbf{u}_i \tag{15.4}$$

Il vettore $\mathbf{u}_i$, che include tutti i disturbi per l'unità i, ha una matrice di varianza–covarianza pari a

$$\mathrm{Var}(\mathbf{u}_i) = \Sigma_i = \sigma_\varepsilon^2 I + \sigma_v^2 J \tag{15.5}$$

dove J è una matrice quadrata con tutti gli elementi pari a 1. Si può mostrare che la matrice

$$K_i = I - \frac{\theta}{T_i} J,$$

dove $\theta = 1 - \sqrt{\frac{\sigma_\varepsilon^2}{\sigma_\varepsilon^2 + T_i\sigma_v^2}}$, ha la seguente proprietà:

$$K_i \Sigma K_i' = \sigma_\varepsilon^2 I$$

Ne consegue che il sistema trasformato

$$K_i \mathbf{y}_i = K_i \mathbf{X}_i \beta + K_i \mathbf{u}_i \tag{15.6}$$

soddisfa le condizioni di Gauss–Markov, e la stima OLS della (15.6) fornisce un'inferenza efficiente. Ma poiché

$$K_i \mathbf{y}_i = \mathbf{y}_i - \theta \bar{\mathbf{y}}_i$$

la stima GLS è equivalente a OLS usando le variabili espresse in termini di "quasi-differenze" dalla media; ossia, sottraendo alle variabili una frazione θ della loro media. Si noti che per $\sigma_\varepsilon^2 \to 0$, $\theta \to 1$, mentre per $\sigma_v^2 \to 0$, $\theta \to 0$. Ciò significa che se tutta la varianza è attribuibile agli effetti individuali, lo stimatore ottimale è quello a effetti fissi; se, al contrario, gli effetti individuali sono trascurabili, allora lo stimatore ottimale si rivela essere il semplice OLS "pooled".

Per implementare l'approccio GLS occorre calcolare θ, che a sua volta richiede una stima delle varianze σ_ε^2 e σ_v^2 (spesso chiamate rispettivamente varianza "within", interna, e varianza "between", esterna, in quanto la prima si riferisce alla variazione all'interno di ogni unità cross-section, mentre la seconda alla variazione tra le unità). In letteratura sono stati suggeriti vari modi di stimare queste grandezze (si veda Baltagi 1995); gretl usa il metodo di Swamy e Arora (1972): σ_ε^2 è stimata dalla varianza residua dal modello a effetti fissi, e la somma $\sigma_\varepsilon^2 + T_i\sigma_v^2$ è stimata come T_i moltiplicato per la varianza residua dallo stimatore "between",

$$\bar{y}_i = \bar{X}_i \beta + e_i$$

L'ultima regressione è implementata costruendo un dataset che comprende le medie di gruppo di tutte le variabili rilevanti.

Scelta dello stimatore

Che modello panel si deve usare? A effetti fissi o casuali?

Un modo per rispondere a questa domanda consiste nell'esaminare la natura del dataset. Se il panel comprende osservazioni su un numero fisso e relativamente piccolo di unità (ad esempio, i paesi membri dell'Unione Europea), potrebbe essere consigliabile adottare il modello a effetti fissi. Se invece comprende osservazioni su un gran numero di individui selezionati in modo casuale (come in molti studi epidemiologici o longitudinali in genere), è più appropriato il modello a effetti casuali.

A parte questa regola euristica, occorre tener conto di alcune considerazioni statistiche:

1. Alcuni dataset panel contengono variabili i cui valori sono specifici all'unità cross-section, ma che non variano nel tempo. Se si vuole includere queste variabili nel modello, la soluzione a effetti fissi non è utilizzabile. Quando questa viene implementata con l'approccio delle variabili dummy, sorge un problema di perfetta collinearità tra le variabili che non cambiano nel tempo e le dummy caratteristiche delle unità. Usando l'approccio della sottrazione delle medie di gruppo, le variabili che non cambiano nel tempo finiscono per avere valore zero.
2. Un vincolo simile esiste per lo stimatore a effetti casuali. Questo stimatore in effetti è una media ponderata (in senso matriciale) dello stimatore "pooled OLS" e dello stimatore "between". Se si hanno osservazioni su n unità e k variabili indipendenti, e se $k > n$ lo stimatore "between" non è definito, visto che abbiamo solo n osservazioni effettive, e quindi non è definito neanche lo stimatore a effetti casuali.

Se non si ricade in una delle limitazioni appena viste, la scelta tra effetti fissi e casuali può essere espressa in termini di due *desiderata* econometrici: l'efficienza e la consistenza.

Da un punto di vista puramente statistico, potremmo dire che c'è un trade-off tra robustezza ed efficienza. Nell'approccio a effetti fissi, l'unica ipotesi che facciamo sugli "effetti di gruppo"

(ossia sulle differenze nella media tra i gruppi, che non variano nel tempo) è che essi esistano, e questa ipotesi può essere testata, come vedremo. Di conseguenza, una volta che questi effetti sono annullati prendendo le deviazioni dalle medie di gruppo, i parametri rimanenti possono essere stimati.

D'altra parte, l'approccio a effetti casuali cerca di modellare gli effetti di gruppo come estrazioni da una certa distribuzione di probabilità, invece di rimuoverli. Ciò richiede che gli effetti individuali siano rappresentabili come parte del termine di disturbo, ossia come variabili casuali a media nulla, non correlate con i regressori.

Di conseguenza, lo stimatore a effetti fissi "funziona sempre", ma al costo di non essere in grado di stimare l'effetto dei regressori che non variano nel tempo. L'ipotesi più ricca, lo stimatore a effetti casuali, permette di stimare i parametri dei regressori che non variano nel tempo, e di stimare in modo più efficiente i parametri dei regressori che variano nel tempo. Questi vantaggi dipendono però dalla validità delle ipotesi aggiuntive che sono state fatte. Se, ad esempio, c'è motivo di credere che gli effetti individuali possano essere correlati con alcune delle variabili esplicative, lo stimatore a effetti casuali sarebbe inconsistente, mentre quello a effetti fissi sarebbe valido. È esattamente su questo principio che si basa il test di Hausman, descritto in seguito: se gli stimatori a effetti fissi e casuali concordano, nel limite dei consueti margini di errore statistici, non c'è motivo di ritenere invalide le ipotesi aggiuntive, e di conseguenza, non c'è motivo di *non* usare il più efficiente stimatore a effetti casuali.

Test sui modelli panel

Se si stima un modello a effetti fissi o casuali usando l'interfaccia grafica, si noterà che il numero di comandi disponibili nel menù "Test" della finestra del modello è abbastanza limitato. I modelli panel hanno alcune complicazioni che rendono difficile implementare tutti i test che si è soliti utilizzare con i modelli stimati su dati cross-section o su serie storiche.

Tuttavia, assieme alle stime dei parametri dei modelli panel vengono mostrati alcuni test specifici dei modelli panel.

Quando si stima un modello usando gli *effetti fissi*, si ottiene automaticamente un test F per l'ipotesi nulla che le unità cross-section abbiano un'intercetta comune, ossia che tutti gli α_i siano uguali, nel qual caso il modello "pooled" (15.1), con una colonna di valori 1 inclusa nella matrice X, è adeguato.

Quando si stima un modello usando gli *effetti casuali*, vengono presentati automaticamente i test di Breusch–Pagan e quello di Hausman.

Il test di Breusch–Pagan è la controparte del test F menzionato sopra. L'ipotesi nulla è che la varianza di v_i nell'equazione (15.3) sia pari a zero; se questa ipotesi non viene rifiutata, si può concludere che il semplice modello "pooled" è adeguato.

Il test di Hausman verifica la consistenza delle stime GLS. L'ipotesi nulla è che queste stime siano consistenti, ossia che sia soddisfatto il requisito di ortogonalità di v_i e di X_i. Il test è basato su una misura, H, della "distanza" tra le stime a effetti fissi e quelle a effetti casuali, costruita in modo che sotto l'ipotesi nulla essa segue una distribuzione χ^2 con gradi di libertà pari al numero di regressori che variano nel tempo contenuti nella matrice X. Se il valore di H è "alto", significa che lo stimatore a effetti casuali non è consistente, e quindi il modello a effetti fissi è preferibile.

Ci sono due modi per calcolare H, il metodo della differenza matriciale e il metodo di regressione. La procedura per il primo metodo è la seguente:

- Raccogliere le stime a effetti fissi in un vettore $\tilde{\beta}$ e le corrispondenti stime a effetti variabili in $\hat{\beta}$, quindi formare il vettore differenza $(\tilde{\beta} - \hat{\beta})$.
- Formare la matrice di covarianza del vettore differenza come $\mathrm{Var}(\tilde{\beta} - \hat{\beta}) = \mathrm{Var}(\tilde{\beta}) - \mathrm{Var}(\hat{\beta}) = \Psi$, dove $\mathrm{Var}(\tilde{\beta})$ e $\mathrm{Var}(\hat{\beta})$ sono stimati con le matrici di varianza campionaria rispettivamente del modello a effetti fissi e casuali [2].

[2] Hausman (1978) mostra che la covarianza della differenza assume questa forma semplice quando $\hat{\beta}$ è uno stimatore efficiente e $\tilde{\beta}$ è inefficiente.

- Calcolare $H = \left(\tilde{\beta} - \hat{\beta}\right)' \Psi^{-1} \left(\tilde{\beta} - \hat{\beta}\right)$.

Date le efficienze relative di $\tilde{\beta}$ e $\hat{\beta}$, la matrice Ψ "dovrebbe" essere definita positiva, nel qual caso H è positivo, ma in campioni finiti ciò non è garantito e ovviamente un valore negativo del χ^2 non è ammissibile. Il metodo della regressione evita questo potenziale problema. La procedura è la seguente:

- Considerare il modello a effetti casuali come il modello vincolato e calcolare la sua somma dei quadrati dei residui come SSR_r.
- Stimare via OLS un modello non vincolato, in cui la variabile dipendente è la quasi-differenza dalla media di y e i regressori includono sia le quasi-differenze dalla media delle X (come nel modello a effetti casuali) e le differenze dalla media di tutte le variabili che variano nel tempo (ossia i regressori degli effetti fissi); calcolare la somma dei quadrati dei residui di questo modello come SSR_u.
- Calcolare $H = n\left(\text{SSR}_r - \text{SSR}_u\right)/\text{SSR}_u$, dove n è il numero totale di osservazioni usate. In questa variante, H non può essere positivo, visto che aggiungere altri regressori al modello a effetti casuali non può aumentare la SSR.

Errori standard robusti

Per la maggior parte degli stimatori, gretl offre la possibilità di calcolare una stima della matrice di covarianza che sia robusta rispetto all'eteroschedasticità e/o all'autocorrelazione (e di conseguenza anche errori standard robusti). Nel caso di dati panel, sono disponibili stimatori robusti della matrice di covarianza robusta per i modelli pooled e a effetti fissi, ma al momento non per il modello a effetti casuali. Per i dettagli, si veda la sezione 14.4.

15.2 Modelli panel dinamici

Quando tra i regressori di un modello panel compare un ritardo della variabile dipendente nascono problemi speciali. Si consideri una variante dinamica del modello pooled (15.1):

$$y_{it} = X_{it}\beta + \rho y_{it-1} + u_{it} \tag{15.7}$$

Per prima cosa, se l'errore u_{it} include un effetto di gruppo, v_i, ne consegue che y_{it-1} è correlato con l'errore, visto che il valore di v_i influisce su y_i per tutti i t. Ciò significa che OLS applicato alla (15.7) sarà inconsistente, oltre che inefficiente. Il modello a effetti fissi annulla gli effetti di gruppo e quindi aggira questo particolare problema, ma ne rimane uno più sottile, che si applica sia alla stima a effetti fissi che a quella a effetti casuali. Si consideri la rappresentazione del modello a effetti fissi in termini di differenza dalle medie, applicata al modello dinamico

$$\tilde{y}_{it} = \tilde{X}_{it}\beta + \rho\tilde{y}_{i,t-1} + \varepsilon_{it}$$

dove $\tilde{y}_{it} = y_{it} - \bar{y}_i$ e $\varepsilon_{it} = u_{it} - \bar{u}_i$ (o $u_{it} - \alpha_i$, usando la notazione dell'equazione 15.2). Il problema è che $\tilde{y}_{i,t-1}$ sarà correlato con ε_{it} attraverso la media di gruppo, $\bar{y}_i$. Il disturbo ε_{it} influenza y_{it} direttamente, che influenza $\bar{y}_i$, che, per costruzione, influenza il valore di $\tilde{y}_{it}$ per ogni t. Lo stesso problema sorge per l'operazione di quasi-differenza dalle medie usata nel modello a effetti casuali. Gli stimatori che ignorano questa correlazione saranno consistenti solo se $T \to \infty$ (caso in cui l'effetto marginale di ε_{it} sulla media di gruppo di y tende a svanire).

Una strategia per affrontare questo problema e ottenere stime consistenti di β e ρ è stata proposta da Anderson e Hsiao (1981). Invece di prendere la differenza dalla media dei dati, essi suggeriscono di prendere la prima differenza della (15.7), un modo alternativo per annullare gli effetti di gruppo:

$$\Delta y_{it} = \Delta X_{it}\beta + \rho\Delta y_{i,t-1} + \eta_{it} \tag{15.8}$$

dove $\eta_{it} = \Delta u_{it} = \Delta(v_i + \varepsilon_{it}) = \varepsilon_{it} - \varepsilon_{i,t-1}$. Non è ancora sufficiente, data la struttura dell'errore η_{it}: il disturbo $\varepsilon_{i,t-1}$ influenza sia η_{it} che $\Delta y_{i,t-1} = y_{it} - y_{i,t-1}$. Il passo successivo è quello di trovare uno strumento per l'espressione "contaminata" $\Delta y_{i,t-1}$. Anderson e Hsiao

suggeriscono di usare $y_{i,t-2}$ o $\Delta y_{i,t-2}$, che sono entrambi non correlati con η_{it} a patto che gli errori sottostanti, ε_{it}, non siano serialmente correlati.

Lo stimatore di Anderson–Hsiao non è fornito come funzione interna di gretl, visto che le funzionalità di gretl per la gestione dei ritardi e delle differenze nei dati panel lo rendono una semplice applicazione di regressione con le variabili strumentali: si veda l'esempio 15.1, basato su uno studio dei tassi di crescita dei paesi (Nerlove 1999)[3].

Esempio 15.1: Lo stimatore Anderson–Hsiao per un modello panel dinamico

```
# La Penn World Table usata da Nerlove
open penngrow.gdt
# Effetti fissi (per confronto)
panel Y 0 Y(-1) X
# Effetti casuali (per confronto)
panel Y 0 Y(-1) X --random-effects
# Differenza di tutte le variabili
diff Y X
# Anderson-Hsiao, usando Y(-2) come strumento
tsls d_Y d_Y(-1) d_X ; 0 d_X Y(-2)
# Anderson-Hsiao, usando d_Y(-2) come strumento
tsls d_Y d_Y(-1) d_X ; 0 d_X d_Y(-2)
```

Sebbene lo stimatore di Anderson–Hsiao sia consistente, non è il più efficiente: non usa pienamente gli strumenti disponibili per $\Delta y_{i,t-1}$, né tiene conto della struttura dell'errore η_{it}. Dei miglioramenti sono stati proposti da Arellano e Bond (1991) e da Blundell e Bond (1998). La versione attuale di gretl permette di utilizzare il metodo di Arellano–Bond, si veda la documentazione del comando `arbond`.

15.3 Esempio: la Penn World Table

La Penn World Table (homepage a pwt.econ.upenn.edu) è un ricco dataset panel macroeconomico, che comprende 152 paesi sull'arco temporale 1950–1992. I dati sono disponibili in formato gretl: si veda la pagina dei dati di gretl (i dati sono liberamente scaricabili, anche se non sono distribuiti nel pacchetto principale di gretl).

L'esempio 15.2 apre il file `pwt56_60_89.gdt`, un sottoinsieme della Penn World Table che contiene dati per 120 paesi negli anni 1960–89, su 20 variabili, senza osservazioni mancanti (il dataset completo, anch'esso compreso nel pacchetto pwt per gretl, contiene molte osservazioni mancanti). Viene calcolata la crescita del PIL reale sul periodo 1960–89 per ogni paese, e viene regredita sul livello del PIL reale dell'anno 1960, per analizzare l'ipotesi della "convergenza" (ossia di una crescita più veloce da parte dei paesi che partono da una situazione peggiore).

[3] Si veda anche la pagina dei benchmark di Clint Cummins, http://www.stanford.edu/~clint/bench/.

Esempio 15.2: Uso della Penn World Table

```
open pwt56_60_89.gdt
# Per l'anno 1989 (l'ultima oss.) il ritardo 29 dà 1960, la prima oss.
genr gdp60 = RGDPL(-29)
# Calcola la crescita totale del PIL reale sui 30 anni
genr gdpgro = (RGDPL - gdp60)/gdp60
# Restringi il campione a una cross-section del 1989
smpl --restrict YEAR=1989
# convergenza: i paesi che partono più indietro crescono di più?
ols gdpgro const gdp60
# risultato: no! Proviamo una relazione inversa?
genr gdp60inv = 1/gdp60
ols gdpgro const gdp60inv
# Ancora no. Proviamo a trattare l'Africa in modo speciale?
genr afdum = (CCODE = 1)
genr afslope = afdum * gdp60
ols gdpgro const afdum gdp60 afslope
```

Capitolo 16

Minimi quadrati non lineari

16.1 Introduzione ed esempi

gretl supporta i minimi quadrati non lineari (NLS - nonlinear least squares), usando una variante dell'algoritmo Levenberg–Marquardt. L'utente deve fornire la specificazione della funzione di regressione, e, prima ancora di fare questo, occorre "dichiarare" i parametri da stimare e fornire dei valori iniziali. È anche possibile indicare analiticamente delle derivate della funzione di regressione rispetto a ognuno dei parametri. La tolleranza usata per fermare la procedura iterativa di stima può essere impostata con il comando `set`.

La sintassi per la specificazione della funzione da stimare è la stessa usata per il comando `genr`. Ecco due esempi, che includono anche le derivate.

Esempio 16.1: Funzione di consumo da Greene

```
nls C = alfa + beta * Y^gamma
    deriv alfa = 1
    deriv beta = Y^gamma
    deriv gamma = beta * Y^gamma * log(Y)
end nls
```

Esempio 16.2: Funzione non lineare da Russell Davidson

```
nls y = alfa + beta * x1 + (1/beta) * x2
    deriv alfa = 1
    deriv beta = x1 - x2/(beta*beta)
end nls
```

Si notino i comandi `nls` (che indica la funzione di regressione), `deriv` (che indica la specificazione di una derivata) e `end nls`, che conclude la specificazione e avvia la stima. Se si aggiunge l'opzione `--vcv` all'ultima riga, verrà mostrata la matrice di covarianza delle stime dei parametri.

16.2 Inizializzazione dei parametri

Prima di eseguire il comando `nls`, occorre definire dei valori iniziali per i parametri della funzione di regressione. Per farlo, è possibile usare il comando `genr` (o, nella versione grafica del programma, il comando "Variabile, Definisci nuova variabile").

In alcuni casi, in cui la funzione non lineare è una generalizzazione di un modello lineare (o di una sua forma ristretta), può essere conveniente eseguire un comando `ols` e inizializzare i parametri a partire dalle stime OLS dei coefficienti. In relazione al primo degli esempi visti sopra, si potrebbe usare:

```
ols C 0 Y
genr alfa = $coeff(0)
genr beta = $coeff(Y)
genr gamma = 1
```

E in relazione al secondo esempio si userebbe:

```
ols y 0 x1 x2
genr alfa = $coeff(0)
genr beta = $coeff(x1)
```

16.3 Finestra di dialogo NLS

Probabilmente il modo più comodo di formulare i comandi per una stima NLS consiste nell'usare uno script per gretl, ma è possibile anche procedere interattivamente, selezionando il comando "Minimi quadrati non lineari" dal menù "Modello, Modelli non lineari". In questo modo, si aprirà una finestra di dialogo in cui è possibile scrivere la specificazione della funzione (opzionalmente preceduta da linee `genr` per impostare i valori iniziali dei parametri) e le derivate, se sono disponibili. Un esempio è mostrato nella figura 16.1. Si noti che in questo contesto non occorre scrivere i comandi `nls` e `end nls`.

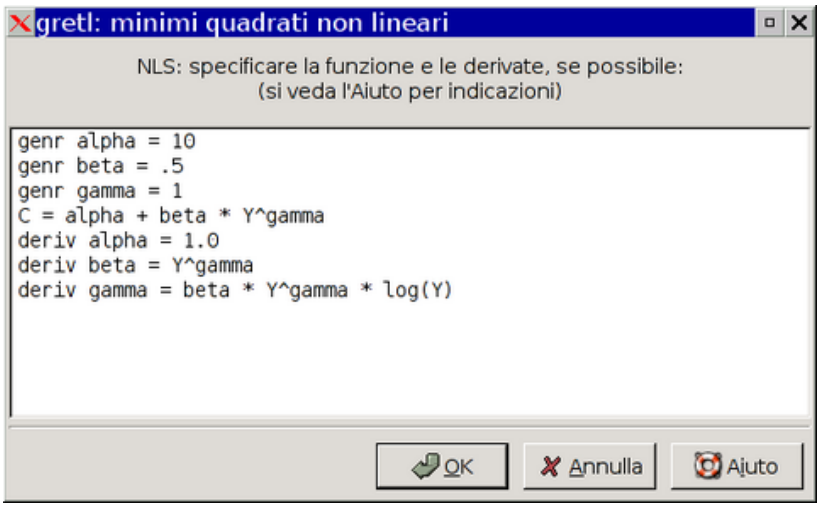


Figura 16.1: Finestra di dialogo NLS

16.4 Derivate analitiche e numeriche

Se si è in grado di calcolare le derivate dalla funzione di regressione rispetto ai parametri, è consigliabile indicarle come mostrato negli esempi precedenti. Se ciò non è possibile, gretl calcolerà delle derivate approssimate numericamente. Tuttavia, le proprietà dell'algoritmo NLS in questo caso potrebbero non essere ottimali (si veda la sezione 16.7).

Questo vien fatto usando l'istruzione `params`, seguita da una lista di identificatori per i parametri da stimare. In questo caso, gli esempi fatti sopra diventano:

```
nls C = alpha + beta * Y^gamma
    params alpha beta gamma
end nls
```

```
nls y = alpha + beta * x1 + (1/beta) * x2
    params alpha beta
end nls
```

Se vengono fornite delle derivate analitiche, ne viene controllata la coerenza con la funzione non lineare data. Se esse sono chiaramente scorrette, la stima viene annullata con un messaggio di errore. Se le derivate sono "sospette", viene mostrato un messaggio di avvertimento, ma la stima prosegue. Questo avvertimento può essere causato da derivate scorrette, ma può anche essere dovuto a un alto grado di collinearità tra le derivate.

Si noti che non è possibile mischiare derivate numeriche e analitiche: se si indicano espressioni analitiche per una derivata, occorre farlo per tutte.

16.5 Arresto della procedura

La procedura di stima NLS è iterativa: l'iterazione viene arrestata quando si verifica una qualunque delle seguenti condizioni: viene raggiunto il criterio di convergenza, o si supera il massimo numero di iterazioni impostato.

Se k denota il numero di parametri da stimare, il numero massimo di iterazioni è $100 \times (k + 1)$ quando si indicano le derivate analitiche, mentre è $200 \times (k + 1)$ quando si usano le derivate numeriche.

Preso un numero ϵ piccolo a piacere, la convergenza si ritiene raggiunta se è soddisfatta almeno una delle due condizioni seguenti:

- Entrambe le riduzioni, effettiva o prevista, della somma dei quadrati degli errori sono minori di ϵ.
- L'errore relativo tra due iterazioni consecutive è minore di ϵ.

Il valore predefinito per ϵ è pari alla precisione della macchina elevata alla potenza $3/4$[1], ma può essere modificato usando il comando `set` con il parametro `nls_toler`. Ad esempio

```
set nls_toler .0001
```

imposterà il valore di ϵ a 0.0001.

16.6 Dettagli sul codice

Il motore sottostante la stima NLS è basato sulla suite di funzioni minpack, disponibile su netlib.org. Nello specifico, sono usate le seguenti funzioni minpack:

`lmder`	Algoritmo Levenberg–Marquardt con derivate analitiche
`chkder`	Controllo delle derivate analitiche fornite
`lmdif`	Algoritmo Levenberg–Marquardt con derivate numeriche
`fdjac2`	Calcolo del Jacobiano approssimato finale se si usano le derivate numeriche
`dpmpar`	Determinazione della precisione della macchina

In caso di successo nell'iterazione Levenberg–Marquardt, viene usata una regressione Gauss–Newton per calcolare la matrice di covarianza per le stime dei parametri. Se si usa l'opzione `--robust`, viene calcolata una variante robusta. La documentazione del comando `set` spiega le opzioni disponibili a questo proposito.

[1] Su una macchina Intel Pentium a 32-bit, il valore è pari a circa 1.82×10^{-12}.

Poiché i risultati NLS sono asintotici, si può discutere sulla necessità di applicare una correzione per i gradi di libertà nel calcolo dell'errore standard della regressione (e di quello delle stime dei parametri). Per confrontabilità con OLS, e seguendo il ragionamento in Davidson e MacKinnon (1993), le stime calcolate in gretl *usano* una correzione per i gradi di libertà.

16.7 Accuratezza numerica

La tabella 16.1 mostra i risultati dell'uso della procedura NLS di gretl sui 27 "Statistical Reference Dataset" forniti dal National Institute of Standards and Technology (NIST) statunitense, per il test del software di regressione non lineare.[2] Per ogni dataset, il file di test indicano due valori iniziali per i parametri, quindi il test completo riporta 54 stime. Sono stati eseguiti due test completi, uno usando derivate analitiche e uno usando approssimazioni numeriche; in entrambi i casi si è usata la tolleranza predefinita.[3]

Sulle 54 stime, gretl non riesce a produrre una soluzione in 4 casi, se vengono usate le derivate analitiche, e in 5 casi se vengono usate le approssimazioni numeriche. Dei quattro fallimenti in modalità derivate analitiche, due sono dovuti alla non convergenza dell'algoritmo Levenberg–Marquardt dopo il numero massimo di iterazioni (su `MGH09` e `Bennett5`, entrambi descritti dal NIST come di "alta difficoltà") e due sono dovuti ad errori di intervallo (valori in virgola mobile al di fuori dei limiti) occorsi durante il calcolo del Jacobiano (su `BoxBOD` e `MGH17`, rispettivamente descritti come di "alta difficoltà" e di "media difficoltà"). In modalità approssimazione numerica, l'ulteriore caso di fallimento è rappresentato da `MGH10` ("alta difficoltà", massimo numero di iterazioni raggiunto).

La tabella mostra informazioni su vari aspetti dei test: numero di fallimenti, numero medio di iterazioni richieste per produrre una soluzione, e due tipi di misura dell'accuratezza dei risultati per i parametri e per i loro errori standard.

Per ognuno dei 54 test eseguiti in ogni modalità, se è stata prodotta una soluzione, sono state confrontate le stime dei parametri ottenute da gretl con i valori certificati dal NIST. È stata definita la variabile "numero minimo di cifre corrette" per una data stima come il numero di cifre significative per cui la *meno accurata* delle stime di gretl coincide con il valore certificato. La tabella mostra i valori medio e minimo di questa variabile, calcolati sulle stime che hanno prodotto una soluzione; la stessa informazione è fornita per gli errori standard stimati.[4]

La seconda misura di accuratezza mostrata è la percentuale di casi, tenendo conto di tutti i parametri di tutte le stime giunte a buon fine, in cui la stima di gretl concorda con il valore certificato per almeno 6 cifre significative, che sono mostrate in modo predefinito nei risultati delle regressioni di gretl.

Usando derivate analitiche, i valori dei casi peggiori sia per le stime dei parametri che per gli errori standard sono stati migliorati a 6 cifre corrette restringendo il valore di tolleranza a 1.0e−14. Usando derivate numeriche, la stessa modifica del limite di tolleranza ha innalzato la precisione dei valori peggiori a 5 cifre corrette per i parametri e 3 cifre per gli errori standard, al costo di un fallimento in più nella convergenza.

Si noti la tendenziale superiorità delle derivate analitiche: in media le soluzioni ai problemi dei test sono state ottenute con molte meno iterazioni e i risultati sono più accurati (in modo evidente per gli errori standard stimati). Si noti anche che i risultati a 6 cifre mostrati da gretl non sono affidabili al 100 per cento per i problemi non lineari difficili (in particolare se si usano derivate numeriche). Tenendo presente questi limiti, la percentuale dei casi in cui i risultati sono accurati alla sesta cifra o più sembra sufficiente per giustificarne l'utilizzo in questa forma.

[2] Per una discussione dell'accuratezza di gretl nella stima di modelli lineari, si veda l'appendice C.

[3] I dati mostrati nella tabella derivano dalla versione 1.0.9 di gretl, compilato con gcc 3.3, collegato a glibc 2.3.2 ed eseguito in Linux su un PC i686 (IBM ThinkPad A21m).

[4] Per gli errori standard, dalle statistiche mostrate nella tabella, è stato escluso l'outlier costituito da `Lanczos1`, che rappresenta un caso strano, composto da dati generati con un adattamento quasi esatto; gli errori standard sono di 9 o 10 ordini di grandezza più piccoli dei coefficienti. In questo caso gretl riesce a riprodurre gli errori standard solo per 3 cifre (con derivate analitiche) e per 2 cifre (con derivate numeriche).

Tabella 16.1: Regressione non lineare: i test NIST

	Derivate analitiche	*Derivate numeriche*
Fallimenti in 54 test	4	5
Iterazioni medie	32	127
Media del "numero minimo di cifre corrette", stima dei parametri	8.120	6.980
Valore minimo del "numero minimo di cifre corrette", stima dei parametri	4	3
Media del "numero minimo di cifre corrette", stima degli errori standard	8.000	5.673
Valore minimo del "numero minimo di cifre corrette", stima degli errori standard	5	2
Percentuale delle stime corrette a 6 cifre, stima dei parametri	96.5	91.9
Percentuale delle stime corrette a 6 cifre, stima degli errori standard	97.7	77.3

Capitolo 17

Stima di massima verosimiglianza

17.1 Stima di massima verosimiglianza con gretl

La stima di massima verosimiglianza (maximum likelihood, ML) è una pietra angolare delle procedure moderne di inferenza. Gretl fornisce un modo per implementare questa metodologia per un grande numero di problemi di stima, usando il comando `mle`. Seguono alcuni esempi.

Introduzione

Per illustrare gli esempi seguenti, inizieremo con un breve ripasso degli aspetti basilari della stima di massima verosimiglianza. Dato un campione di ampiezza T, è possibile definire la funzione di densità[1] per l'intero campione, ossia la distribuzione congiunta di tutte le osservazioni $f(\mathbf{Y};\theta)$, dove $\mathbf{Y} = \{y_1, \ldots, y_T\}$. La sua forma è determinata da un k-vettore di parametri sconosciuti θ, che assumiamo contenuti in un insieme Θ, e che possono essere usati per stimare la probabilità di osservare un campione con qualsiasi data caratteristica.

Dopo aver osservato i dati, i valori di $\mathbf{Y}$ sono noti, e questa funzione può essere valutata per tutti i possibili valori di θ. Quando usiamo y_t come argomento e θ come parametro, la funzione è interpretabile come una densità, mentre è preferibile chiamarla funzione di *verosimiglianza* quando θ è considerato argomento della funzione e i valori dei dati $\mathbf{Y}$ hanno il solo compito di determinarne la forma.

Nei casi più comuni, questa funzione possiede un massimo unico, la cui posizione non viene alterata dal fatto di considerare il logaritmo della verosimiglianza (ossia la log-verosimiglianza): questa funzione si esprime come

$$\ell(\theta) = \log f(\mathbf{Y};\theta)$$

Le funzioni di log-verosimiglianza gestite da gretl sono quelle in cui $\ell(\theta)$ può essere scritta come

$$\ell(\theta) = \sum_{t=1}^{T} \ell_t(\theta)$$

che è vero nella maggior parte dei casi di interesse. Le funzioni $\ell_t(\theta)$ vengono chiamate contributi di log-verosimiglianza.

Inoltre, la posizione del massimo è ovviamente determinata dai dati $\mathbf{Y}$. Ciò significa che il valore

$$\hat{\theta}(\mathbf{Y}) = \underset{\theta\in\Theta}{\text{Argmax}}\, \ell(\theta) \tag{17.1}$$

è una qualche funzione dei dati osservati (ossia una statistica), che ha la proprietà, sotto alcune condizioni deboli, di essere uno stimatore consistente, asintoticamente normale e asintoticamente efficiente, di θ.

In alcuni casi è possibile scrivere esplicitamente la funzione $\hat{\theta}(\mathbf{Y})$, ma in generale ciò non è sempre vero, e il massimo va cercato con tecniche numeriche. Queste si basano spesso sul fatto che la log-verosimiglianza è una funzione continuamente differenziabile di θ, e quindi nel massimo le sue derivate parziali devono essere tutte pari a 0. Il *vettore gradiente*, o *vettore degli score*, è una funzione con molte proprietà interessanti dal punto di vista statistico, che

[1] Stiamo supponendo che i nostri dati siano una realizzazione di variabili casuali continue. Per variabili discrete, la trattazione rimane valida, riferendosi alla funzione di probabilità invece che a quella di densità. In entrambi i casi, la distribuzione può essere condizionale su alcune variabili esogene.

verrà denotata con $\mathbf{g}(\theta)$. È un k-vettore con elemento tipico

$$g_i(\theta) = \frac{\partial \ell(\theta)}{\partial \theta_i} = \sum_{t=1}^{T} \frac{\partial \ell_t(\theta)}{\partial \theta_i}$$

I metodi basati sul gradiente possono essere illustrati brevemente:

1. scegliere un punto $\theta_0 \in \Theta$;
2. valutare $\mathbf{g}(\theta_0)$;
3. se $\mathbf{g}(\theta_0)$ è "piccolo", fermarsi; altrimenti calcolare un vettore di direzione $d(\mathbf{g}(\theta_0))$;
4. valutare $\theta_1 = \theta_0 + d(\mathbf{g}(\theta_0))$;
5. sostituire θ_0 con θ_1;
6. ricominciare dal punto 2.

Esistono molti algoritmi di questo tipo; si differenziano nel modo con cui calcolano il vettore di direzione $d(\mathbf{g}(\theta_0))$, per assicurarsi che sia $\ell(\theta_1) > \ell(\theta_0)$ (in modo che prima o poi si arrivi a un massimo).

Il metodo usato da gretl per massimizzare la log-verosimiglianza è un algoritmo basato sul gradiente, noto come metodo di **BFGS** (Broyden, Fletcher, Goldfarb e Shanno). Questa tecnica è usata in molti pacchetti statistici ed econometrici, visto che è ritenuta valida e molto potente. Ovviamente, per rendere operativa questa tecnica, deve essere possibile calcolare il vettore $\mathbf{g}(\theta)$ per ogni valore di θ. In alcuni casi, la funzione $\mathbf{g}(\theta)$ può essere vista esplicitamente in termini di $\mathbf{Y}$. Se questo non è possibile, o è troppo difficile, il gradiente può essere valutato numericamente.

La scelta del valore iniziale θ_0 è cruciale in alcuni contesti e ininfluente in altri. In generale è consigliabile far partire l'algoritmo da valori "sensibili", quando è possibile. Se è disponibile uno stimatore consistente, di solito è una scelta valida ed efficiente: ci si assicura che per grandi campioni il punto di partenza sarà probabilmente vicino a $\hat{\theta}$ e la convergenza sarà raggiunta in poche iterazioni.

Il numero massimo di iterazioni consentite per la procedura BFGS, e la relativa tolleranza per valutare la convergenza, possono essere impostati usando il comando `set`: le variabili relative sono `bfgs_maxiter` (valore predefinito: 500) e `bfgs_toler` (valore predefinito: la precisione della macchina elevata alla potenza 3/4).

Matrice di covarianza ed errori standard

Per impostazione predefinita, la matrice delle stime dei parametri è basata sul prodotto esterno del gradiente (OPG, Outer Product of the Gradient), ossia:

$$\widehat{\mathrm{Var}}_{\mathrm{OPG}}(\hat{\theta}) = \left(G'(\hat{\theta})G(\hat{\theta})\right)^{-1}$$

dove $G(\hat{\theta})$ è la matrice $T \times k$ dei contributi al gradiente. Sono disponibili due altre possibilità: se si usa l'opzione `--hessian`, la matrice di covarianza viene calcolata con un'approssimazione numerica dell'Hessiana alla convergenza. Se si usa l'opzione `--robust`, viene usato lo stimatore "sandwich" di quasi-massima verosimiglianza:

$$\widehat{\mathrm{Var}}_{\mathrm{QML}}(\hat{\theta}) = H(\hat{\theta})^{-1}G'(\hat{\theta})G(\hat{\theta})H(\hat{\theta})^{-1}$$

dove H denota l'approssimazione numerica dell'Hessiana.

17.2 Stima di una Gamma

Si supponga di avere un campione di T osservazioni indipendenti e identicamente distribuite da una distribuzione Gamma. La funzione di densità per ogni osservazione x_t è

$$f(x_t) = \frac{\alpha^p}{\Gamma(p)} x_t^{p-1} \exp(-\alpha x_t) \tag{17.2}$$

La log-verosimiglianza per l'intero campione può essere scritta come il logaritmo della densità congiunta di tutte le osservazioni. Visto che queste sono indipendenti e identiche, la densità congiunta è il prodotto delle densità individuali, e quindi il suo logaritmo è

$$\ell(\alpha, p) = \sum_{t=1}^{T} \log \left[\frac{\alpha^p}{\Gamma(p)} x_t^{p-1} \exp(-\alpha x_t) \right] = \sum_{t=1}^{T} \ell_t \tag{17.3}$$

dove

$$\ell_t = p \cdot \log(\alpha x_t) - \gamma(p) - \log x_t - \alpha x_t$$

e $\gamma(\cdot)$ è il logaritmo della funzione gamma. Per stimare i parametri α e p con la massima verosimiglianza, occorre massimizzare (17.3) rispetto ad essi. Il frammento di codice da eseguire in gretl è

```
scalar alpha = 1
scalar p = 1
scalar p = 1
mle logl =  p*ln(alpha * x) - lngamma(p) - ln(x) - alpha * x
end mle
```

I due comandi

```
alpha = 1
p = 1
```

sono necessari per assicurarsi che le variabili `p` e `alpha` esistano prima di tentare il calcolo di `logl`. Il loro valore sarà modificato dall'esecuzione del comando `mle` e sarà sostituito dalle stime di massima verosimiglianza se la procedura è andata a buon fine. Il valore iniziale è 1 per entrambi; è arbitrario e non conta molto in questo esempio (ma si veda oltre).

Il codice visto può essere reso più leggibile, e leggermente più efficiente, definendo una variabile in cui memorizzare $\alpha \cdot x_t$. Questo comando può essere inserito nel blocco `mle` nel modo seguente:

```
scalar alpha = 1
scalar p = 1
scalar p = 1
mle logl =  p*ln(ax) - lngamma(p) - ln(x) - ax
series ax = alpha*x
params alpha p
end mle
```

In questo caso, è necessario includere la riga `params alpha p` per impostare i simboli `p` e `alpha` separatamente da `ax`, che è una variabile generata temporaneamente, e non un parametro da stimare.

In un semplice esempio come questo, la scelta dei valori iniziali è quasi ininfluente, visto che l'algoritmo convergerà a prescindere dai valori iniziali. In ogni caso, stimatori consistenti basati sul metodo dei momenti per p e α possono essere ricavati dalla media campionaria m e dalla varianza V: visto che si può dimostrare che

$$E(x_t) = p/\alpha \qquad V(x_t) = p/\alpha^2$$

segue che gli stimatori

$$\begin{aligned} \tilde{\alpha} &= m/V \\ \tilde{p} &= m \cdot \tilde{\alpha} \end{aligned}$$

sono consistenti e quindi appropriati da usare come punti di partenza per l'algoritmo. Lo script per gretl diventa quindi

```
scalar m = mean(x)
scalar alpha = m/var(x)
scalar p = m*alpha
scalar p = m*alpha
mle logl =  p*ln(ax) - lngamma(p) - ln(x) - ax
series ax = alpha*x
params alpha p
end mle
```

Un'altro fatto di cui tener conto è che talvolta i parametri sono vincolati all'interno di certi intervalli: in questo caso, ad esempio, sia α che p devono essere numeri positivi. Gretl non controlla queste condizioni: è responsabilità dell'utente assicurarsi che la funzione venga sempre valutata in un punto ammissibile dello spazio dei parametri, durante la ricerca iterativa del massimo. Un metodo efficace per far questo consiste nel definire una variabile per controllare che i parametri siano ammissibili e impostare la log-verosimiglianza come indefinita se il controllo fallisce. Si veda il seguente esempio, che usa l'operatore di assegnazione condizionale:

```
scalar m = mean(x)
scalar alpha = m/var(x)
scalar p = m*alpha
scalar p = m*alpha

mle logl = check ? p*ln(ax) - lngamma(p) - ln(x) - ax : NA
  series ax = alpha*x
  scalar check = (alpha>0) & (p>0)
params alpha p
end mle
```

17.3 Funzioni di costo con frontiera stocastica

Quando si modella una funzione di costo, talvolta è utile incorporare esplicitamente nel modello statistico il fatto che le imprese possano essere inefficienti, così che il costo osservato si discosta dal valore teorico non solo per l'eterogeneità (non osservata) tra le imprese, ma anche perché due imprese, uguali sotto tutti gli aspetti, potrebbero operare a diversi regimi di efficienza. In questo caso, possiamo scrivere

$$C_i = C_i^* + u_i + v_i$$

dove C_i è qualche indicatore dei costi variabili, C_i^* è il suo valore "teorico", u_i è un termine di disturbo a media zero e v_i è il termine che modella l'inefficienza, che si suppone essere non negativo, per il suo significato.

Spesso si sceglie una specificazione lineare per C_i^*; ad esempio, la funzione di costo Cobb–Douglas si ottiene quando C_i^* è una funzione lineare dei logaritmi dei prezzi degli input e delle quantità di output.

Il modello con *frontiera stocastica* è un modello lineare del tipo $y_i = x_i\beta + \varepsilon_i$ in cui il termine di errore ε_i è la somma di u_i e v_i. Un postulato tipico è che siano $u_i \sim N(0, \sigma_u^2)$ e $v_i \sim |N(0, \sigma_v^2)|$. Se si ipotizza anche l'indipendenza tra u_i e v_i, è possibile dimostrare che la funzione di densità di ε_i ha la forma:

$$f(\varepsilon_i) = \sqrt{\frac{2}{\pi}}\Phi\left(\frac{\lambda\varepsilon_i}{\sigma}\right)\frac{1}{\sigma}\phi\left(\frac{\varepsilon_i}{\sigma}\right) \tag{17.4}$$

dove $\Phi(\cdot)$ e $\phi(\cdot)$ sono, rispettivamente, la funzione di distribuzione e quella di densità della normale standard, $\sigma = \sqrt{\sigma_u^2 + \sigma_v^2}$ e $\lambda = \frac{\sigma_u}{\sigma_v}$.

Di conseguenza, la log-verosimiglianza per una osservazione ha la seguente forma (a meno di una costante non rilevante):

$$\ell_t = \log \Phi\left(\frac{\lambda \varepsilon_i}{\sigma}\right) - \left[\log(\sigma) + \frac{\varepsilon_i^2}{2\sigma^2}\right]$$

Quindi il modello di funzione di costo Cobb-Douglas con frontiera stocastica è descritto dalle seguenti equazioni:

$$\begin{aligned}
\log C_i &= \log C_i^* + \varepsilon_i \\
\log C_i^* &= c + \sum_{j=1}^{m} \beta_j \log y_{ij} + \sum_{j=1}^{n} \alpha_j \log p_{ij} \\
\varepsilon_i &= u_i + v_i \\
u_i &\sim N(0, \sigma_u^2) \\
v_i &\sim \left| N(0, \sigma_v^2) \right|
\end{aligned}$$

In molti casi, si intende assicurarsi che l'omogeneità della funzione di costo rispetto ai prezzi sia valida per costruzione; visto che questa condizione equivale a $\sum_{j=1}^{n} \alpha_j = 1$, l'equazione vista sopra per C_i^* può essere riscritta come

$$\log C_i - \log p_{in} = c + \sum_{j=1}^{m} \beta_j \log y_{ij} + \sum_{j=2}^{n} \alpha_j (\log p_{ij} - \log p_{in}) + \varepsilon_i \tag{17.5}$$

Questa equazione può essere stimata con OLS, ma con due inconvenienti: per prima cosa lo stimatore OLS per l'intercetta c non è consistente, perché il termine di disturbo ha media diversa da zero; in secondo luogo, gli stimatori OLS per gli altri parametri sono consistenti, ma inefficienti, a causa della non-normalità di ε_i. Entrambi i problemi possono essere risolti stimando la (17.5) con la massima verosimiglianza. Tuttavia, la stima OLS è un modo veloce e comodo per ricavare dei valori iniziali per l'algoritmo di massima verosimiglianza.

L'esempio 17.1 mostra come implementare il modello descritto finora. Il file `banks91` contiene parte dei dati usati in Lucchetti, Papi e Zazzaro (2001).

17.4 Modelli GARCH

Gretl gestisce i modelli GARCH attraverso una funzione interna, tuttavia può essere istruttivo vedere come possono essere stimati usando il comando `mle`.

Le equazioni seguenti mostrano l'esempio più semplice di modello GARCH(1,1):

$$\begin{aligned}
y_t &= \mu + \varepsilon_t \\
\varepsilon_t &= u_t \cdot \sigma_t \\
u_t &\sim N(0,1) \\
h_t &= \omega + \alpha \varepsilon_{t-1}^2 + \beta h_{t-1}.
\end{aligned}$$

Poiché la varianza di y_t dipende dai valori passati, per scrivere la funzione di log-verosimiglianza non basta sommare i logaritmi delle densità per le singole osservazioni. Come spesso accade nei modelli di serie storiche, y_t non può essere considerato indipendente dalle altre osservazioni del campione, di conseguenza, la funzione di densità per l'intero campione (la densità congiunta di tutte le osservazioni) non è semplicemente il prodotto delle densità marginali.

La stima di massima verosimiglianza, in questi casi, si effettua considerando le densità *condizionali*, quindi quello che si massimizza è una funzione di verosimiglianza condizionale. Se definiamo il set informativo al tempo t come

$$F_t = \{y_t, y_{t-1}, \ldots\},$$

Esempio 17.1: Stima di una funzione di costo con frontiera stocastica

```
open banks91

# Funzione di costo Cobb-Douglas

ols cost const y p1 p2 p3

# Funzione di costo Cobb-Douglas con vincoli di omogeneità

genr rcost = cost - p3
genr rp1 = p1 - p3
genr rp2 = p2 - p3

ols rcost const y rp1 rp2

# Funzione di costo Cobb-Douglas con vincoli di omogeneità
# e inefficienza

scalar b0 = $coeff(const)
scalar b1 = $coeff(y)
scalar b2 = $coeff(rp1)
scalar b3 = $coeff(rp2)

scalar su = 0.1
scalar sv = 0.1

mle logl = ln(cnorm(e*lambda/ss)) - (ln(ss) + 0.5*(e/ss)^2)
  scalar ss = sqrt(su^2 + sv^2)
  scalar lambda = su/sv
  series e = rcost - b0*const - b1*y - b2*rp1 - b3*rp2
  params b0 b1 b2 b3 su sv
end mle
```

la densità di y_t condizionata a F_{t-1} è normale:

$$y_t|F_{t-1} \sim N[\mu, h_t].$$

Per le proprietà delle distribuzioni condizionali, la densità congiunta può essere fattorizzata nel modo seguente

$$f(y_t, y_{t-1}, \ldots) = \left[\prod_{t=1}^{T} f(y_t|F_{t-1})\right] \cdot f(y_0)$$

Se trattiamo y_0 come fissato, il termine $f(y_0)$ non dipende dai parametri sconosciuti, e quindi la log-verosimiglianza condizionale può essere scritta come la somma dei contributi individuali

$$\ell(\mu, \omega, \alpha, \beta) = \sum_{t=1}^{T} \ell_t \tag{17.6}$$

dove

$$\ell_t = \log\left[\frac{1}{\sqrt{h_t}}\phi\left(\frac{y_t - \mu}{\sqrt{h_t}}\right)\right] = -\frac{1}{2}\left[\log(h_t) + \frac{(y_t - \mu)^2}{h_t}\right]$$

Lo script seguente mostra una semplice applicazione di questa tecnica, che usa il file di dati `djclose`, uno dei dataset di esempio forniti con gretl, che contiene dati giornalieri per l'indice azionario Dow Jones.

```
open djclose

series y = 100*ldiff(djclose)

scalar mu = 0.0
scalar omega = 1
scalar alpha = 0.4
scalar beta = 0.0

mle ll = -0.5*(log(h) + (e^2)/h)
  series e = y - mu
  series h = var(y)
  series h = omega + alpha*(e(-1))^2 + beta*h(-1)
  params mu omega alpha beta
end mle
```

17.5 Derivate analitiche

Il calcolo del vettore degli score è essenziale per lavorare col metodo BFGS. In tutti gli esempi precedenti non è stata data alcuna formula esplicita per il calcolo dello score, ma sono stati indicati all'algoritmo dei gradienti valutati numericamente. Il calcolo numerico dello score per il parametro i-esimo è effettuato tramite un'approssimazione finita della derivata, ossia

$$\frac{\partial \ell(\theta_1, \ldots, \theta_n)}{\partial \theta_i} \simeq \frac{\ell(\theta_1, \ldots, \theta_i + h, \ldots, \theta_n) - \ell(\theta_1, \ldots, \theta_i - h, \ldots, \theta_n)}{2h}$$

dove h è un numero piccolo.

In molte situazioni questo metodo è abbastanza efficiente e accurato, ma potrebbe esserci la necessità di evitare le approssimazioni e specificare una funzione esatta per le derivate. Come esempio, si consideri lo script seguente:

```
nulldata 1000

genr x1 = normal()
genr x2 = normal()
genr x3 = normal()
```

```
genr ystar = x1 + x2 + x3 + normal()
genr y = (ystar > 0)

scalar b0 = 0
scalar b1 = 0
scalar b2 = 0
scalar b3 = 0

mle logl = y*ln(P) + (1-y)*ln(1-P)
  series ndx = b0 + b1*x1 + b2*x2 + b3*x3
  series P = cnorm(ndx)
  params b0 b1 b2 b3
end mle --verbose
```

Vengono generate artificialmente 1000 osservazioni per un modello probit ordinario[2]: y_t è una variabile binaria, che assume valore 1 se $y_t^* = \beta_1 x_{1t} + \beta_2 x_{2t} + \beta_3 x_{3t} + \varepsilon_t > 0$ e 0 altrove. Quindi, $y_t = 1$ con probabilità $\Phi(\beta_1 x_{1t} + \beta_2 x_{2t} + \beta_3 x_{3t}) = \pi_t$. La funzione di probabilità per un'osservazione può essere scritta come

$$P(y_t) = \pi_t^{y_t}(1-\pi_t)^{1-y_t}$$

Visto che le osservazioni sono indipendenti e identicamente distribuite, la log-verosimiglianza è semplicemente la somma dei contributi individuali. Quindi

$$\ell = \sum_{t=1}^{T} y_t \log(\pi_t) + (1-y_t)\log(1-\pi_t)$$

L'opzione `--verbose` alla dine del comando `end mle` produce un resoconto dettagliato delle iterazioni compiute dall'algoritmo BFGS.

In questo caso, la derivazione numerica funziona abbastanza bene, tuttavia è anche facile calcolare lo score in modo analitico, visto che la derivata $\frac{\partial \ell}{\partial \beta_i}$ può essere scritta come

$$\frac{\partial \ell}{\partial \beta_i} = \frac{\partial \ell}{\partial \pi_t} \cdot \frac{\partial \pi_t}{\partial \beta_i}$$

ed è facile vedere che

$$\begin{aligned} \frac{\partial \ell}{\partial \pi_t} &= \frac{y_t}{\pi_t} - \frac{1-y_t}{1-\pi_t} \\ \frac{\partial \pi_t}{\partial \beta_i} &= \phi(\beta_1 x_{1t} + \beta_2 x_{2t} + \beta_3 x_{3t}) \cdot x_{it} \end{aligned}$$

Il blocco `mle` nello script precedente può quindi essere modificato nel modo seguente:

```
mle logl = y*ln(P) + (1-y)*ln(1-P)
  series ndx = b0 + b1*x1 + b2*x2 + b3*x3
  series P = cnorm(ndx)
  series tmp = dnorm(ndx)*(y/P - (1-y)/(1-P))
  deriv b0 = tmp
  deriv b1 = tmp*x1
  deriv b2 = tmp*x2
  deriv b3 = tmp*x3
end mle --verbose
```

Si noti che il comando `params` è stato sostituito da una serie di comandi `deriv`, che hanno la doppia funzione di identificare i parametri rispetto a cui ottimizzare e di fornire un'espressione analitica per i rispettivi elementi del vettore degli score.

[2] Ancora una volta ricordiamo che gretl contiene il comando interno `probit` (si veda il capitolo 22.1), ma in questo caso usiamo il modello probit come esempio per la procedura

17.6 Debug del codice mle

Finora sono stati presentati i comandi principali che possono essere usati all'interno di un blocco `mle`, ossia

- comandi ausiliari per generare variabili ausiliarie;
- comandi `deriv` per specificare il gradiente rispetto ad ognuno dei parametri;
- un comando `params` per identificare i parametri nel caso in cui non vengano specificate le derivate analitiche.

Per fare il debug del codice contenuto nei blocchi `mle`, è possibile usare un altro tipo di comandi: è possibile stampare il valore di una variabile rilevante nel corso di ogni iterazione. Questo meccanismo è più limitato rispetto al classico comando `print`: in questo caso, la parola chiave `print` può essere seguita dal nome di una sola variabile (una scalare, una serie, o una matrice).

Nell'ultimo degli esempi visti sopra, è stata generata una variabile chiamata `tmp`, usata come base per le derivate analitiche. Per tenere traccia dell'andamento di questa variabile, si può aggiungere un comando print nel blocco `mle`, nel modo seguente:

```
series tmp = dnorm(ndx)*(y/P - (1-y)/(1-P))
print tmp
```

Capitolo 18

Stima GMM

18.1 Introduzione e terminologia

Il metodo dei momenti generalizzato (Generalized Method of Moments, GMM) è un metodo di stima generale e molto potente, che racchiude in pratica tutte le tecniche di stima parametrica usate in econometria. È stato introdotto in Hansen (1982) e in Hansen e Singleton (1982); un'eccellente e approfondita trattazione si trova nel capitolo 17 di Davidson e MacKinnon (1993).

Il principio di base su cui si basa il GMM è abbastanza semplice: si supponga di voler stimare un parametro scalare θ basandosi su un campione $x_1, x_2, \ldots, x_T$, con θ_0 a indicare il "vero" valore di θ. Considerazioni teoriche (di natura statistica o economica) suggeriscono di considerare valida una relazione come la seguente:

$$E\left[x_t - g(\theta)\right] = 0 \Leftrightarrow \theta = \theta_0, \tag{18.1}$$

con $g(\cdot)$ funzione continua e invertibile. Ossia, esiste una funzione dei dati e del parametro, con la proprietà di avere valore atteso pari a zero se e solo se essa è valutata in corrispondenza del vero valore del parametro. Ad esempio, i modelli economici con aspettative razionali conducono spesso ad espressioni come la (18.1).

Se il modello di campionamento per gli x_t è tale per cui vale una versione della legge dei grandi numeri, segue

$$\bar{X} = \frac{1}{T}\sum_{t=1}^{T} x_t \xrightarrow{\text{p}} g(\theta_0);$$

quindi, visto che $g(\cdot)$ è invertibile, la statistica

$$\hat{\theta} = g^{-1}(\bar{X}) \xrightarrow{\text{p}} \theta_0,$$

così che $\hat{\theta}$ è uno stimatore consistente di θ. Un modo diverso per ottenere lo stesso risultato consiste nello scegliere, come stimatore di θ, il valore che minimizza la funzione obiettivo

$$F(\theta) = \left[\frac{1}{T}\sum_{t=1}^{T}(x_t - g(\theta))\right]^2 = \left[\bar{X} - g(\theta)\right]^2; \tag{18.2}$$

il minimo è ovviamente raggiunto in $\hat{\theta} = g^{-1}(\bar{X})$, visto che l'espressione tra parentesi quadre vale 0.

Il ragionamento precedente può essere generalizzato come segue: si supponga che θ sia un vettore a n dimensioni e che valgano m relazioni come

$$E\left[f_i(x_t, \theta)\right] = 0 \quad \text{for } i = 1 \ldots m, \tag{18.3}$$

dove $E[\cdot]$ è un valore atteso condizionato su un insieme di p variabili z_t chiamate *strumenti*. Nel semplice esempio visto sopra, si ha $m = 1$ e $f(x_t, \theta) = x_t - g(\theta)$, e l'unico strumento usato è $z_t = 1$. Quindi, deve anche essere vero che

$$E\left[f_i(x_t, \theta) \cdot z_{j,t}\right] = E\left[f_{i,j,t}(\theta)\right] = 0 \quad \text{for } i = 1 \ldots m \quad \text{and} \quad j = 1 \ldots p; \tag{18.4}$$

L'equazione (18.4) è detta *condizione di ortogonalità*, o *condizione sul momento*. Lo stimatore GMM è definito come il minimo della forma quadratica

$$F(\theta, W) = \bar{\mathbf{f}}' W \bar{\mathbf{f}}, \tag{18.5}$$

dove $\bar{\mathbf{f}}$ è un vettore $(1 \times m \cdot p)$ che contiene la media delle condizioni di ortogonalità e W è una qualche matrice simmetrica e definita positiva, nota come matrice dei *pesi*. Una condizione necessaria per l'esistenza del minimo è la condizione di ordine $n \leq m \cdot p$.

La statistica

$$\hat{\theta} = \underset{\theta}{\text{Argmin}}\, F(\theta, W) \tag{18.6}$$

è uno stimatore consistente di θ qualunque sia la scelta di W. Tuttavia, per raggiungere la massima efficienza asintotica, W deve essere proporzionale all'inversa della matrice di convarianza di lungo periodo fra le condizioni di ortogonalità; se W non è nota, è sufficiente uno stimatore consistente.

Queste considerazioni portano a definire la seguente strategia nella pratica:

1. Scegliere una matrice W definita positiva e calcolare lo stimatore GMM *a 1 passo* $\hat{\theta}_1$. Alcune scelte tipiche per W sono $I_{m \cdot p}$ oppure $I_m \otimes (Z'Z)^{-1}$.
2. Usare $\hat{\theta}_1$ per stimare $V(f_{i,j,t}(\theta))$ e usare la sua inversa come matrice dei pesi. Lo stimatore risultante $\hat{\theta}_2$ è chiamato stimatore *a 2 passi*.
3. Ri-stimare $V(f_{i,j,t}(\theta))$ per mezzo di $\hat{\theta}_2$ e ottenere $\hat{\theta}_3$; iterare fino alla convergenza. Asintoticamente questi passi successivi non sono necessari, visto che lo stimatore a due passi è consistente ed efficiente, ma lo stimatore iterato ha spesso proprietà migliori sui piccoli campioni e dovrebbe essere indipendente dalla scelta di W fatta al passo 1.

Nel caso speciale in cui il numero dei parametri n sia pari al numero delle condizioni di ortogonalità $m \cdot p$, lo stimatore GMM $\hat{\theta}$ è lo stesso per qualsiasi scelta della matrice dei pesi W, quindi il primo passo è sufficiente; in questo caso la funzione obiettivo vale 0 al minimo.

Se, al contrario, vale $n < m \cdot p$, occorre il secondo passo (o le successive iterazioni) per raggiungere l'efficienza, e lo stimatore così ottenuto può essere molti diverso, su campioni finiti, dallo stimatore a un passo. Inoltre, il valore della funzione obiettivo al minimo, scalato opportunamente per il numero delle osservazioni, produce la *statistica J di Hansen*, che può essere interpretata come una statistica test che si distribuisce come una χ^2 con $m \cdot p - n$ gradi di libertà sotto l'ipotesi nulla di corretta specificazione. Si veda il capitolo 17.6 di Davidson e MacKinnon (1993), per i dettagli.

Nelle sezioni seguenti verrà mostrato come queste nozioni sono implementate in gretl usando alcuni esempi.

18.2 Minimi quadrati ordinari (OLS) come GMM

È istruttivo iniziare con un esempio volutamente semplice: si consideri il modello lineare $y_t = x_t\beta + u_t$. Anche se si è soliti interpretarlo come la somma di una certa "parte sistematica" e di un certo "disturbo", un'interpretazione più rigorosa consiste nell'*ipotesi* che la media condizionale $E(y_t|x_t)$ sia lineare e nella *definizione* di u_t as $y_t - E(y_t|x_t)$.

Dalla definizione di u_t, segue che $E(u_t|x_t) = 0$, quindi disponiamo della seguente condizione di ortogonalità:

$$E[f(\beta)] = 0, \tag{18.7}$$

dove $f(\beta) = (y_t - x_t\beta)x_t$. Le definizioni date nella sezione precedente diventano quindi le seguenti:

- θ è β;
- lo strumento è x_t;
- $f_{i,j,t}(\theta)$ is $(y_t - x_t\beta)x_t = u_t x_t$; la condizione di ortogonalità è interpretabile come il requisito che i regressori siano non correlati con i disturbi;
- W può essere una qualsiasi matrice simmetrica definita positiva, visto che il numero dei parametri uguaglia quello delle condizioni di ortogonalità. Scegliamo ad esempio I.

- La funzione $F(\theta, W)$ in questo caso è

$$F(\theta, W) = \left[\frac{1}{T} \sum_{t=1}^{T} (\hat{u}_t x_t) \right]^2$$

  ed è semplice vedere perché OLS e GMM coincidono: la funzione obiettivo GMM e quella OLS sono minimizzate rispetto alla stessa grandezza, la somma dei quadrati dei residui. Si noti però che le due funzioni non sono uguali: al minimo vale $F(\theta, W) = 0$ mentre la somma dei quadrati dei residui minimizzata vale zero solo nel caso di relazione lineare perfetta.

Il codice contenuto nell'esempio 18.1 usa il comando `gmm` di gretl per rendere operative le considerazioni fatte sopra.

Esempio 18.1: OLS via GMM

```
/* Inizializzazioni varie */
series e = 0
scalar beta = 0
matrix V = I(1)

/* Esecuzione della stima */
gmm
  series e = y - x*beta
  orthog e ; x
  weights V
  params beta
end gmm
```

Gli ingredienti necessari per la stima GMM vengono forniti a gretl in un blocco di comandi che inizia con `gmm` e finisce con `end gmm`, all'interno del quale è obbligatoria la presenza di tre elementi:

1. una o più dichiarazioni `orthog`
2. una dichiarazione `weights`
3. una dichiarazione `params`

I tre elementi vanno forniti nell'ordine presentato sopra.

Le dichiarazioni `orthog` sono usate per specificare le condizioni di ortogonalità e devono seguire la sintassi

```
orthog x ; Z
```

dove `x` può essere una serie, matrice, o una lista di serie, e `Z` può anche essa essere una serie, lista o matrice. Nell'esempio 18.1, la serie `e` contiene i "residui" e la serie `x` contiene il regressore. Se `x` fosse stata una lista (o una matrice), la dichiarazione `orthog` avrebbe generato una condizione di ortogonalità per ogni elemento (o colonna) di `x`. Si noti la struttura della condizione di ortogonalità: si assume che il termine a sinistra del punto e virgola rappresenti una quantità che dipende dai parametri stimati (e che va quindi aggiornata durante la stima iterativa), mentre il termine a destra è una funzione costante dei dati.

La dichiarazione `weights` specifica la matrice iniziale dei pesi, e la sua sintassi è evidente. La dichiarazione `params` specifica i parametri rispetto a cui il criterio GMM va minimizzato: segue le stesse regole usate per i comandi `mle` e `nls`.

Il valore minimo viene cercato tramite ottimizzazione numerica usando il metodo BFGS (si veda la sezione 7.9 e il capitolo 17). Il progresso della procedura di ottimizzazione può essere visualizzato aggiungendo l'opzione `--verbose` alla riga `end gmm`. Ovviamente nell'esempio appena visto la stima GMM non è la scelta più naturale, visto che il metodo OLS fornisce facilmente una soluzione esatta senza dover ricorrere all'ottimizzazione.

18.3 Minimi quadrati a due stadi (TSLS) come GMM

Avvicinandoci al dominio proprio della stima GMM, consideriamo ora i minimi quadrati a due stadi (two-stage least squares, TSLS) come un caso della stima GMM.

La stima TSLS si usa quando occorre stimare un modello lineare della forma $y_t = X_t\beta + u_t$, ma in cui una o più delle variabili nella matrice X sono potenzialmente endogene (correlate con il termine di errore u). Procediamo identificando un insieme di strumenti, Z_t, che possono spiegare le variabili endogene X ma che sono plausibilmente non correlati con u. La classica procedura a due stadi consiste nel (1) regredire gli elementi endogeni di X su Z e (2) stimare l'equazione che interessa, avendo sostituito gli elementi endogeni di X con i relativi valori stimati nella (1).

Una prospettiva alternativa è fornita dal metodo GMM. Si definisce il residuo $\hat{u}_t$ come $y_t - X_t\hat{\beta}$, al solito. Ma invece di basarsi su $E(u|X) = 0$ come in OLS, la stima si basa sulla condizione $E(u|Z) = 0$. In questo caso è naturale basare la matrice iniziale dei pesi sulla matrice di covarianza degli strumenti. L'esempio 18.2 presenta un modello preso da *Introduction to Econometrics* di Stock e Watson. La domanda di sigarette è modellata come funzione lineare dei logaritmi del prezzo e del reddito; il reddito è trattato come esogeno, mentre il prezzo è considerato endogeno; due misure della tassazione sono usate come strumenti. Poiché si hanno due strumenti e una variabile endogena, il modello è sovraidentificato, quindi la matrice dei pesi influenzerà la soluzione. Una parte dei risultati di questo script è mostrata in 18.3. Gli errori standard stimati con GMM sono robusti per impostazione predefinita: se usassimo l'opzione `--robust` con il comando `tsls` otterremmo risultati identici[1].

18.4 Opzioni per la matrice di covarianza

La matrice di covarianza dei parametri stimati dipende dalla scelta di W attraverso l'equazione

$$\hat{\Sigma} = (J'WJ)^{-1}J'W\Omega WJ(J'WJ)^{-1} \tag{18.8}$$

dove J è un termine Jacobiano

$$J_{ij} = \frac{\partial \bar{f}_i}{\partial \theta_j}$$

e Ω è la matrice di covarianza di lungo periodo delle condizioni di ortogonalità.

Gretl calcola J tramite differenziazione numerica (non c'è modo di specificare un'espressione analitica per J al momento). Per Ω serve una stima consistente: la scelta più semplice consiste nell'usare la matrice di covarianza campionaria delle f_t:

$$\hat{\Omega}_0(\theta) = \frac{1}{T}\sum_{t=1}^{T} f_t(\theta)f_t(\theta)' \tag{18.9}$$

Questo stimatore è robusto rispetto all'eteroschedasticità, ma non rispetto all'autocorrelazione. Una variante robusta rispetto all'eteroschedasticità e all'autocorrelazione (HAC, heteroskedasticity autocorrelation consistent) si può ottenere usando il kernel di Bartlett, o altri kernel. Una sua versione univariata è usata nella funzione `lrvar()`, si veda l'equazione (7.1). La versione multivariata è riassunta nell'equazione (18.10).

$$\hat{\Omega}_k(\theta) = \frac{1}{T}\sum_{t=k}^{T-k}\left[\sum_{i=-k}^{k} w_i f_t(\theta)f_{t-i}(\theta)'\right], \tag{18.10}$$

Gretl calcola in modo predefinito la matrice di covarianza HAC quando si stima un modello GMM su serie storiche. È possibile controllare il kernel e la larghezza di banda (ossia il valore di k nella 18.10) usando il comando `set`. Si veda il capitolo 14 per una discussione approfondita della stima HAC. È anche possibile dire a gretl di *non* usare la versione HAC con il comando

```
set force_hc on
```

[1] Il file di dati usato in questo esempio è tratto dal pacchetto che contiene altri dati dal libro di Stock e Watson, disponibile su http://gretl.sourceforge.net/gretl_data_it.html.

Esempio 18.2: TSLS via GMM

```
\begin{scode}
open cig_ch10.gdt
# Prezzo medio reale al lordo delle tasse sulla vendita
genr ravgprs = avgprs / cpi
# Tassa sulle sigarette media reale
genr rtax = tax / cpi
# Totale delle tasse medie reali
genr rtaxs = taxs / cpi
# Tassa sulla vendita media reale
genr rtaxso = rtaxs - rtax
# Logaritmi del consumo, prezzo, reddito
genr lpackpc = log(packpc)
genr lravgprs = log(ravgprs)
genr perinc = income / (pop*cpi)
genr lperinc = log(perinc)
# Restrizione del campione alle osservazioni del 1995
smpl --restrict year=1995
# Equazione (10.16) usando tsls
list xlist = const lravgprs lperinc
list zlist = const rtaxso rtax lperinc
tsls lpackpc xlist ; zlist --robust

# Impostazioni per gmm
matrix Z = { zlist }
matrix W = inv(Z'Z)
series e = 0
scalar b0 = 1
scalar b1 = 1
scalar b2 = 1

gmm e = lpackpc - b0 - b1*lravgprs - b2*lperinc
  orthog e ; Z
  weights W
  params b0 b1 b2
end gmm
```

Esempio 18.3: TSLS via GMM: risultati parziali

```
Modello 1: stime TSLS usando le 48 osservazioni 1-48
Variabile dipendente: lpackpc
Strumenti: rtaxso rtax
Errori standard robusti per l'eteroschedasticità, variante HC0

      VARIABILE      COEFFICIENTE       ERRORE STD.   STAT T   P-VALUE

  const                 9.89496          0.928758     10.654  <0.00001 ***
  lravgprs             -1.27742          0.241684     -5.286  <0.00001 ***
  lperinc               0.280405         0.245828      1.141   0.25401

Modello 2: stime GMM a un passo usando le 48 osservazioni 1-48
e = lpackpc - b0 - b1*lravgprs - b2*lperinc

      PARAMETRO       STIMA              ERRORE STD.   STAT T   P-VALUE

  b0                    9.89496          0.928758     10.654  <0.00001 ***
  b1                   -1.27742          0.241684     -5.286  <0.00001 ***
  b2                    0.280405         0.245828      1.141   0.25401

  Criterio GMM = 0.0110046
```

18.5 Un esempio reale: il Consumption Based Asset Pricing Model

Per illustrare l'implementazione di GMM in gretl, replichiamo l'esempio contenuto nel capitolo 3 di Hall (2005). Il modello da stimare è un'applicazione classica del metodo GMM, e fornisce un esempio in cui le condizioni di ortogonalità non derivano da considerazioni di tipo statistico, ma dalla teoria economica.

Un individuo razionale che deve allocare il proprio reddito tra consumo e investimento in un'attività finanziaria deve in pratica scegliere il sentiero di consumo lungo tutto l'arco della propria vita, visto che l'investimento si trasforma in consumo futuro. Si può dimostrare che il sentiero di consumo ottimale deve soddisfare la condizione seguente:

$$pU'(c_t) = \delta^k E\left[r_{t+k}U'(c_{t+k})|\mathcal{F}_t\right], \tag{18.11}$$

dove p è il prezzo dell'attività finanziaria, $U(\cdot)$ è la funzione di utilità individuale, δ è il saggio di sconto soggettivo dell'individuo e r_{t+k} è il tasso di rendimento dell'attività tra il tempo t e il tempo $t+k$. $\mathcal{F}_t$ è il *set informativo* al tempo t; l'equazione (18.11) afferma che l'utilità "persa" al tempo t acquistando l'attività finanziaria invece che beni di consumo deve essere bilanciata da un corrispondente aumento nell'utilità futura (scontata) del consumo finanziato dal rendimento dell'investimento. Poiché il futuro è incerto, l'individuo considera la propria aspettativa, condizionata alle informazioni conosciute al momento in cui viene fatta la scelta.

Non si fa alcuna ipotesi sulla natura dell'attività finanziaria, quindi l'equazione (18.11) vale per qualsiasi attività, ed è possibile costruire un sistema di equazioni come la (18.11) per ogni attività di cui si osserva il prezzo.

Se si accettano le seguenti ipotesi

- L'intera economia può essere riassunta da un unico individuo rappresentativo e immortale
- La funzione $U(x) = \frac{x^\alpha - 1}{\alpha}$ rappresenta fedelmente le preferenze dell'individuo

imponendo $k = 1$, l'equazione (18.11) implica la seguente per ogni attività j:

$$E\left[\delta \frac{r_{j,t+1}}{p_{j,t}}\left(\frac{C_{t+1}}{C_t}\right)^{\alpha-1} \middle| \mathcal{F}_t\right] = 1, \tag{18.12}$$

dove C_t è la funzione di consumo aggregata e α e δ sono l'avversione al rischio e il saggio di sconto dell'individuo rappresentativo. In questo caso, è facile vedere come i parametri

"profondi" α e δ possono essere stimati tramite GMM usando

$$e_t = \delta \frac{r_{j,t+1}}{p_{j,t}} \left(\frac{C_{t+1}}{C_t} \right)^{\alpha-1} - 1$$

come condizione dei momenti, mentre qualsiasi variabile nota al tempo t può servire da strumento.

Esempio 18.4: Stima del Consumption Based Asset Pricing Model

```
open hall.gdt
set force_hc on

scalar alpha = 0.5
scalar delta = 0.5
series e = 0

list inst = const consrat(-1) consrat(-2) ewr(-1) ewr(-2)

matrix V0 = 100000*I(nelem(inst))
matrix Z = { inst }
matrix V1 = $nobs*inv(Z'Z)

gmm e = delta*ewr*consrat^(alpha-1) - 1
  orthog e ; inst
  weights V0
  params alpha delta
end gmm

gmm e = delta*ewr*consrat^(alpha-1) - 1
  orthog e ; inst
  weights V1
  params alpha delta
end gmm

gmm e = delta*ewr*consrat^(alpha-1) - 1
  orthog e ; inst
  weights V0
  params alpha delta
end gmm --iterate

gmm e = delta*ewr*consrat^(alpha-1) - 1
  orthog e ; inst
  weights V1
  params alpha delta
end gmm --iterate
```

Nel codice di esempio contenuto in 18.4, viene replicata una parte della tabella 3.7 contenuta in Hall (2005). La variabile `consrat` è definita come rapporto dei consumi pro capite (in servizi e beni non durevoli) in mesi successivi per gli USA, e `ewr` il rapporto rendimento–prezzo di un'attività finanziaria fittizia costruita prendendo la media di tutte le azioni quotate al NYSE. Il set degli strumenti contiene la costante e due ritardi di ogni variabile.

Il comando `set force_hc on` nella seconda riga dello script ha il solo scopo di replicare l'esempio: come spiegato in precedenza, forza gretl a calcolare la varianza di lungo periodo delle condizioni di ortogonalità secondo l'equazione (18.9) piuttosto che secondo la (18.10).

Si esegue `gmm` quattro volte: la stima a un passo per ognuna delle due matrici iniziali dei pesi, quindi la stima iterativa a partire da ognuno dei due insiemi di pesi iniziali. Visto che il numero delle condizioni di ortogonalità (5) è maggiore di quello dei parametri stimati (2), la scelta

Esempio 18.5: Stima del Consumption Based Asset Pricing Model: risultati

```
Modello 1: stime GMM a un passo usando le 465 osservazioni 1959:04-1997:12
e = d*ewr*consrat^(alpha-1) - 1

      PARAMETRO       STIMA             ERRORE STD    STAT-T   P-VALUE

  alpha                -3.14475          6.84439      -0.459   0.64590
  d                     0.999215         0.0121044    82.549   <0.00001 ***

  Criterio GMM = 2778.08

Modello 2: stime GMM a un passo usando le 465 osservazioni 1959:04-1997:12
e = d*ewr*consrat^(alpha-1) - 1

      PARAMETRO       STIMA             ERRORE STD    STAT-T   P-VALUE

  alpha                 0.398194         2.26359       0.176   0.86036
  d                     0.993180         0.00439367  226.048   <0.00001 ***

  Criterio GMM = 14.247

Modello 3: stime GMM iterato usando le 465 osservazioni 1959:04-1997:12
e = d*ewr*consrat^(alpha-1) - 1

      PARAMETRO       STIMA             ERRORE STD    STAT-T   P-VALUE

  alpha                -0.344325         2.21458      -0.155   0.87644
  d                     0.991566         0.00423620  234.070   <0.00001 ***

  Criterio GMM = 5491.78
  Test J: Chi-quadro(3) = 11.8103 (p-value 0.0081)

Modello 4: stime GMM iterato usando le 465 osservazioni 1959:04-1997:12
e = d*ewr*consrat^(alpha-1) - 1

      PARAMETRO       STIMA             ERRORE STD    STAT-T   P-VALUE

  alpha                -0.344315         2.21359      -0.156   0.87639
  d                     0.991566         0.00423469  234.153   <0.00001 ***

  Criterio GMM = 5491.78
  Test J: Chi-quadro(3) = 11.8103 (p-value 0.0081)
```

dei pesi iniziali fa la differenza, e in effetti si notano differenze sostanziali tra le stime a un passo (modelli 1 e 2). D'altra parte, la procedura di iterazione riduce queste differenze quasi totalmente (modelli 3 e 4).

Parte dei risultati viene mostrata nella 18.5. Occorre notare che il test J porta a rifiutare l'ipotesi di specificazione corretta, e questo forse non è sorprendente, viste le ipotesi eroiche richieste per passare dal principio microeconomico contenuto nell'equazione (18.11) al sistema aggregato che viene in effetti stimato.

18.6 Avvertenze

Nonostante la sua semplicità concettuale, il metodo GMM è probabilmente tra i più fragili disponibili in econometria. Il numero di scelte non banali che occorre fare è abbastanza alto, e su campioni finiti ognuna di esse può avere conseguenze drammatiche sul risultato finale. Alcuni dei fattori che possono influenzare i risultati sono i seguenti:

1. Le condizioni di ortogonalità possono essere scritte in più di un modo: ad esempio, se $E(x_t - \mu) = 0$, vale anche $E(x_t/\mu - 1) = 0$. È possibile che diverse specificazioni delle condizioni sui momenti conducano a diversi risultati.

2. Come accade con altri algoritmi di ottimizzazione numerica, possono verificarsi casi in cui la funzione obiettivo è quasi piatta in alcune direzioni, oppure ha più di un minimo. Il metodo BFGS di solito è adeguato, ma non garantisce di fornire sempre la soluzione migliore.

3. Gli stimatori a un passo e, in misura minore, quello a due passi, possono essere sensibili a dettagli apparentemente insignificanti, come il fatto di ri-scalare gli strumenti. Anche la scelta sulla matrice dei pesi iniziali può avere conseguenze sensibili.

4. Se i dati sono serie storiche, non c'è una regola precisa sul numero di ritardi da usare nel calcolo della matrice di covarianza di lungo periodo (si veda la sezione 18.4). Il nostro consiglio è di fare alcuni tentativi, visto che i risultati possono variare di molto a seconda della scelta. Le versioni future di gretl conterranno più opzioni sulla stima della matrice di covarianza.

Un'importante conseguenza di tutti questi aspetti è può essere molto difficile replicare i risultati di alcuni studi noti, se non si conoscono tutti i dettagli relativi alla procedura di stima usata.

Capitolo 19

Criteri di selezione dei modelli

19.1 Introduzione

In alcuni contesti, l'econometrico deve scegliere tra modelli alternativi basandosi su test di ipotesi formali. Ad esempio, si può scegliere un modello più generale rispetto ad uno più ristretto, se la restrizione in questione può essere formulata sotto forma di ipotesi nulla testabile e l'ipotesi nulla viene rifiutata da un apposito test.

In altri contesti si ha bisogno invece di un criterio di selezione dei modelli che tenga conto da una parte dell'accuratezza dell'adattamento ai dati, o della verosimiglianza del modello, e dall'altra parte della sua parsimonia. È necessario mantenere questo equilibrio perché l'aggiunta di variabili a un modello può solo aumentare la sua capacità di adattamento o la sua verosimiglianza, ma è possibile che ciò avvenga anche se le variabili aggiuntive non sono veramente rilevanti per il processo che ha generato i dati.

Il più famoso tra questi criteri di selezione, per modelli lineari stimati con i minimi quadrati, è l'R^2 corretto,

$$\bar{R}^2 = 1 - \frac{\text{SSR}/(n-k)}{\text{TSS}/(n-1)}$$

dove n è il numero di osservazioni nel campione, k denota il numero di parametri stimati, SSR e TSS denotano rispettivamente la somma dei quadrati dei residui e la somma dei quadrati della variabile dipendente. Confrontata con il classico coefficiente di determinazione R^2

$$R^2 = 1 - \frac{\text{SSR}}{\text{TSS}}$$

la versione "corretta" penalizza l'inclusione di variabili aggiuntive, a parità di altre condizioni.

19.2 Criteri di informazione

Un criterio più generale, che segue un'impostazione simile, è il "criterio di informazione di Akaike" (AIC) del 1974. La formulazione originale di questa misura è

$$\text{AIC} = -2\ell(\hat{\theta}) + 2k \tag{19.1}$$

dove $\ell(\hat{\theta})$ rappresenta la massima log-verosimiglianza come funzione del vettore delle stime dei parametri, $\hat{\theta}$, e k (come sopra) indica il numero di "parametri indipendenti all'interno del modello". In questa formulazione, con AIC correlato negativamente alla verosimiglianza e positivamente al numero dei parametri, il ricercatore mira a minimizzare il suo valore.

L'AIC può generare confusione, dal momento che sono diffuse varie versioni per il suo calcolo; ad esempio, Davidson e MacKinnon (2004) ne presentano una versione semplificata,

$$\text{AIC} = \ell(\hat{\theta}) - k$$

che vale quanto l'originale moltiplicata per -2: in questo caso, ovviamente, si cercherà di massimizzare l'AIC.

Nel caso di modelli stimati con i minimi quadrati, la log-verosimiglianza può essere scritta come

$$\ell(\hat{\theta}) = -\frac{n}{2}(1 + \log 2\pi - \log n) - \frac{n}{2}\log \text{SSR} \tag{19.2}$$

Sostituendo (19.2) in (19.1) otteniamo

$$\text{AIC} = n(1 + \log 2\pi - \log n) + n \log \text{SSR} + 2k$$

che può essere scritta anche come

$$\text{AIC} = n \log\left(\frac{\text{SSR}}{n}\right) + 2k + n(1 + \log 2\pi) \tag{19.3}$$

Alcuni autori semplificano la formula nel caso di modelli stimati con i minimi quadrati. Ad esempio William Greene scrive

$$\text{AIC} = \log\left(\frac{\text{SSR}}{n}\right) + \frac{2k}{n} \tag{19.4}$$

Questa variante può essere derivata da (19.3) dividendo per n e sottraendo la costante $1 + \log 2\pi$. Ossia, chiamando AIC_G la versione proposta da Greene, abbiamo

$$\text{AIC}_G = \frac{1}{n}\text{AIC} - (1 + \log 2\pi)$$

Infine, Ramanathan offre un'altra variante:

$$\text{AIC}_R = \left(\frac{\text{SSR}}{n}\right) e^{2k/n}$$

che è l'esponenziale della versione di Greene.

All'inizio, gretl usava la versione di Ramanathan, ma a partire dalla versione 1.3.1 del programma, viene usata la formula originale di Akaike (19.1), e più specificamente (19.3) per i modelli stimati con i minimi quadrati.

Anche se il criterio di Akaike è progettato per favorire la parsimonia, non lo fa in modo eccessivo. Ad esempio, se abbiamo due modelli annidati con rispettivamente $k - 1$ e k parametri, e se l'ipotesi nulla che il parametro k valga 0 è vera, per grandi campioni l'AIC tenderà comunque a far preferire il modello meno parsimonioso in circa il 16 per cento dei casi (si veda Davidson e MacKinnon, 2004, capitolo 15).

Un criterio alternativo all'AIC che non risente di questo problema è il "Criterio di informazione Bayesiana" (BIC) di Schwarz (1978). Il BIC può essere scritto (in modo simile alla formulazione di Akaike di AIC) come

$$\text{BIC} = -2\ell(\hat{\theta}) + k \log n$$

Il prodotto di k per $\log n$ nel BIC significa che la penalizzazione per l'aggiunta di parametri addizionali aumenta con l'ampiezza campionaria. Ciò assicura che, asintoticamente, un modello troppo esteso non verrà mai scelto al posto di un modello parsimonioso ma correttamente specificato.

Un'altra alternativa all'AIC che tende a favorire modelli più parsimoniosi è il criterio di Hannan–Quinn, o HQC (Hannan e Quinn, 1979). Volendo essere coerenti con la formulazione usata finora, può essere scritto nel modo seguente:

$$\text{HQC} = -2\ell(\hat{\theta}) + 2k \log\log n$$

Il calcolo di Hannan–Quinn si basa sulla regola del logaritmo iterato (si noti che l'ultimo termine è il logaritmo del logaritmo dell'ampiezza campionaria). Gli autori affermano che questa procedura fornisce un "procedura di stima consistente in senso forte per l'ordine di una autoregressione", e che "confrontata con altre procedure consistenti in senso forte, questa sottostimerà l'ordine in modo minore".

Gretl mostra AIC, BIC e HQC (calcolati nel modo spiegato sopra) per la maggior parte dei modelli. Quando si interpretano questi valori occorre sempre ricordarsi se sono calcolati in modo da essere massimizzati o minimizzati. In gretl essi sono sempre calcolati per essere minimizzati: valori minori sono da preferire.

Capitolo 20

Modelli per serie storiche

20.1 Introduzione

Questo capitolo e il successivo discutono i modelli per serie storiche. Questo capitolo si concentra sui modelli ARIMA, i test per radici unitarie e i modelli GARCH, mentre il successivo tratta la cointegrazione e i modelli a correzione d'errore.

20.2 Modelli ARIMA

Rappresentazione e sintassi

Il comando `arma` effettua la stima di modelli autoregressivi integrati a media mobile (ARIMA). Questi modelli possono essere scritti come

$$\phi(L)y_t = \theta(L)\epsilon_t \tag{20.1}$$

dove $\phi(L)$ e $\theta(L)$ sono polinomi nell'operatore ritardo, L, definito in modo che $L^n x_t = x_{t-n}$, e ϵ_t è un processo di rumore bianco. Il contenuto esatto di y_t, del polinomio AR $\phi()$ e del polinomio MA $\theta()$ verrà spiegato in seguito.

Media del processo

Il processo y_t mostrato nell'equazione (20.1) ha media zero, se non si danno altre indicazioni. Se il modello deve essere applicato a dati reali, è necessario includere un termine per gestire la possibilità che y_t abbia una media diversa da zero. Ci sono due modi possibili per rappresentare processi con media nulla: uno consiste nel definire μ_t come la media *non condizionale* di y_t, ossia il valore centrale della sua distribuzione marginale. Quindi, la serie $\tilde{y}_t = y_t - \mu_t$ ha media 0, e il modello (20.1) si applica a $\tilde{y}_t$. In pratica, assumendo che μ_t sia una funzione lineare di alcune variabili osservabili x_t, il modello diventa

$$\phi(L)(y_t - x_t\beta) = \theta(L)\epsilon_t \tag{20.2}$$

Questo viene talvolta chiamato un "modello di regressione con errori ARMA"; la sua struttura può risultare più chiara se lo rappresentiamo usando due equazioni:

$$\begin{aligned} y_t &= x_t\beta + u_t \\ \phi(L)u_t &= \theta(L)\epsilon_t \end{aligned}$$

Il modello appena presentato viene talvolta chiamato "ARMAX" (ARMA + variabili esogene), anche se questo nome sembra più appropriato per un altro tipo di modello: un altro modo per includere un termine di media nella (20.1) consiste nel basare la rappresentazione sulla media *condizionale* di y_t, ossia il valore centrale della distribuzione di y_t *dato il proprio passato*. Assumendo, ancora, che questa possa essere rappresentata come combinazione lineare di qualche variabile osservabile z_t, il modello diventerebbe

$$\phi(L)y_t = z_t\gamma + \theta(L)\epsilon_t \tag{20.3}$$

La formulazione (20.3) ha il vantaggio che γ può essere immediatamente interpretato come il vettore degli effetti marginali delle variabili z_t sulla media condizionale di y_t. E aggiungendo i ritardi di z_t a questa specificazione è possibile stimare dei *modelli di funzione di trasferimento* (che generalizzano gli ARMA aggiungendo gli effetti delle variabili esogene distribuiti nel tempo).

Gretl fornisce un modo per stimare entrambe le forme. I modelli scritti come nella (20.2) vengono stimati con la massima verosimiglianza; i modelli scritti come nella (20.3) vengono stimati con la massima verosimiglianza condizionale (per maggiori informazioni su queste opzioni, si veda la sezione "Stima" in seguito).

Nel caso speciale in cui $x_t = z_t = 1$ (ossia il modello include una costante ma nessuna variabile esogena) le due specificazioni discusse finora diventano

$$\phi(L)(y_t - \mu) = \theta(L)\epsilon_t \tag{20.4}$$

e

$$\phi(L)y_t = \alpha + \theta(L)\epsilon_t \tag{20.5}$$

rispettivamente. Queste formulazioni sono essenzialmente equivalenti, ma se rappresentano lo stesso processo, ovviamente μ e α non sono numericamente identici; piuttosto:

$$\alpha = \left(1 - \phi_1 - \ldots - \phi_p\right)\mu$$

La sintassi di gretl per stimare la (20.4) è semplicemente

```
arma p q ; y
```

Gli ordini di ritardo AR e MA, `p` e `q`, possono essere indicati come numeri o come scalari definiti in precedenza. Il parametro μ può essere omesso se necessario, aggiungendo l'opzione `--nc` ("no constant") al comando. Se occorre stimare la (20.5), bisogna aggiungere al comando l'opzione `--conditional`, come in

```
arma p q ; y --conditional
```

Generalizzando questo principio alla stima della (20.2) o della (20.3), si ottiene che

```
arma p q ; y const x1 x2
```

stimerebbe il modello seguente:

$$y_t - x_t\beta = \phi_1\left(y_{t-1} - x_{t-1}\beta\right) + \ldots + \phi_p\left(y_{t-p} - x_{t-p}\beta\right) + \epsilon_t + \theta_1\epsilon_{t-1} + \ldots + \theta_q\epsilon_{t-q}$$

dove in questo caso $x_t\beta = \beta_0 + x_{t,1}\beta_1 + x_{t,2}\beta_2$.

Usando l'opzione `--conditional`, come in

```
arma p q ; y const x1 x2 --conditional
```

si stimerebbe il modello seguente:

$$y_t = x_t\gamma + \phi_1 y_{t-1} + \ldots + \phi_p y_{t-p} + \epsilon_t + \theta_1\epsilon_{t-1} + \ldots + \theta_q\epsilon_{t-q}$$

Idealmente, la situazione descritta finora potrebbe essere sistematizzata scrivendo una specificazione più generale che comprende tutte le alternative, ossia

$$\phi(L)\left(y_t - x_t\beta\right) = z_t\gamma + \theta(L)\epsilon_t; \tag{20.6}$$

Una possibile generalizzazione del comando `arma` permetterebbe all'utente di specificare se, per ogni metodo di stima, certe variabili esogene vanno trattate come x_t o come z_t, ma non siamo ancora a questo livello (né lo è la maggior parte degli altri pacchetti software).

Modelli stagionali

Quando si analizzano serie storiche che mostrano un marcato andamento stagionale, è opportuno usare una struttura di ritardi più flessibile. Il modello (20.1) può essere espanso nel seguente

$$\phi(L)\Phi(L^s)y_t = \theta(L)\Theta(L^s)\epsilon_t. \tag{20.7}$$

In questi casi, è disponibile una versione ampliata della sintassi, ossia:

```
arma p q ; P Q ; y
```

dove p e q rappresentano gli ordini AR e MA non stagionali, mentre P e Q gli ordini stagionali. Ad esempio

```
arma 1 1 ; 1 1 ; y
```

stima il modello seguente:

$$(1 - \phi L)(1 - \Phi L^s)(y_t - \mu) = (1 + \theta L)(1 + \Theta L^s)\epsilon_t$$

Se y_t è una serie trimestrale (e quindi $s = 4$), l'equazione precedente può essere scritta in modo più esplicito come

$$y_t - \mu = \phi(y_{t-1} - \mu) + \Phi(y_{t-4} - \mu) - (\phi \cdot \Phi)(y_{t-5} - \mu) + \epsilon_t + \theta\epsilon_{t-1} + \Theta\epsilon_{t-4} + (\theta \cdot \Theta)\epsilon_{t-5}$$

Un tale modello è noto come "modello ARMA stagionale moltiplicativo".

Buchi nella struttura dei ritardi

Il modo standard di specificare un modello ARMA in gretl è usando gli ordini AR e MA, ossia rispettivamente p e q. In questo caso vengono inclusi tutti i ritardi a partire dall'ordine 1 fino all'ordine specificato. In alcuni casi, occorre includere solo alcuni specifici ritardi AR e/o MA; questo risultato può essere ottenuto in due modi.

- È possibile costruire una matrice che contiene i ritardi desiderati (numeri interi positivi) e fornire il nome della matrice al posto di p o q.
- È possibile fornire una lista di ritardi separati da spazi e racchiusi tra parentesi graffe, al posto di p o q.

Gli esempi seguenti illustrano le due possibilità:

```
matrix pvec = {1, 4}
arma pvec 1 ; y
arma {1 4} 1 ; y
```

Entrambi i comandi specificano un modello ARMA in cui vengono usati i ritardi AR 1 e 4, ma non i 2 e 3.

Questo meccanismo è disponibile solo per la componente non stagionale della specificazione ARMA.

Differenziazione e ARIMA

La discussione svolta finora presuppone che la serie storica y_t sia già stata soggetta a tutte le trasformazioni ritenute necessarie per assicurarne la stazionarietà (si veda anche la sezione 20.3). La differenziazione è la più comune tra queste trasformazioni, e gretl fornisce un meccanismo per includere questo passo nel comando arma: la sintassi

```
arma p d q ; y
```

stima un modello ARMA(p, q) su $\Delta^d y_t$. È funzionalmente equivalente a

```
series tmp = y
loop for i=1..d
  tmp = diff(tmp)
end loop
arma p q ; tmp
```

tranne che per quanto riguarda la previsione dopo la stima (si veda oltre).

Quando la serie y_t viene differenziata prima di effettuare l'analisi, il modello viene chiamato ARIMA (la "I" sta per "Integrato"); per questo motivo, gretl fornisce anche il comando `arima` come alias per `arma`.

La differenziazione stagionale è trattata in modo simile, con la sintassi

```
arma p d q ; P D Q ; y
```

dove `D` è l'ordine di differenziazione stagionale. Così, il comando

```
arma 1 0 0 ; 1 1 1 ; y
```

produrrebbe le stesse stime dei parametri date da

```
genr dsy = sdiff(y)
arma 1 0 ; 1 1 ; dsy
```

dove usiamo la funzione `sdiff` per creare una differenza stagionale (ad esempio per dati trimestrali: $y_t - y_{t-4}$).

Stima

Il metodo di stima predefinito per i modelli ARMA è quello della massima verosimiglianza esatta (sotto l'ipotesi che il termine di errore sia distribuito normalmente), usando il filtro di Kalman insieme all'algoritmo di massimizzazione BFGS. Il gradiente della log-verosimiglianza rispetto alle stime dei parametri è approssimato numericamente. Questo metodo produce risultati che sono direttamente confrontabili con quelli di molti altri pacchetti software. La costante ed eventuali altre variabili esogene sono trattate come mostrato nell'equazione (20.2). La matrice di covarianza per i parametri è calcolata usando un'approssimazione numerica dell'Hessiana alla convergenza.

Il metodo alternativo, utilizzabile con l'opzione `--conditional`, è quello della massima verosimiglianza condizionale (CML), noto anche come "somma dei quadrati condizionale" (si veda Hamilton 1994, p. 132). Questo metodo è stato esemplificato nello script 9.3 e ne verrà qui data solo una breve descrizione. Data un'ampiezza campionaria T, il metodo CML minimizza la somma dei quadrati degli errori di previsione "one-step-ahead" (per il periodo successivo) generata dal modello per le osservazioni $t_0, \ldots, T$. Il punto di partenza t_0 dipende dall'ordine dei polinomi AR nel modello. Il metodo numerico di massimizzazione usato è il BHHH, e la matrice di covarianza è calcolata usando una regressione di Gauss–Newton.

Il metodo CML è quasi equivalente a quello della massima verosimiglianza in ipotesi di normalità; la differenza sta nel fatto che le prime $(t_0 - 1)$ osservazioni sono considerate fisse ed entrano nella funzione di verosimiglianza solo come variabili condizionanti. Di conseguenza, i due metodi sono asintoticamente equivalenti sotto le consuete ipotesi, tranne per il fatto, discusso sopra, che la nostra implementazione CML tratta la costante e le variabili esogene come mostrato nell'equazione (20.3).

I due metodi possono essere confrontati nell'esempio seguente

```
open data10-1
arma 1 1 ; r
arma 1 1 ; r --conditional
```

che produce le stime mostrate nella Tabella 20.1. Come si può vedere, le stime di ϕ e θ sono abbastanza simili. Le costanti riportate sono molto diverse, come ci si può aspettare; in proposito, si veda la discussione delle equazioni (20.4) e (20.5). In ogni caso, dividendo la costante CML per $1 - \phi$ si ottiene 7.38, che non è molto distante dalla stima ML di 6.93.

Tabella 20.1: Stime ML e CML

Parametro	ML		CML	
μ	6.93042	(0.923882)	1.07322	(0.488661)
ϕ	0.855360	(0.0511842)	0.852772	(0.0450252)
θ	0.588056	(0.0986096)	0.591838	(0.0456662)

Convergenza e inizializzazione

I metodi numerici usati per massimizzare la verosimiglianza per i modelli ARMA non sempre convergono. I valori iniziali dei parametri possono influire sul raggiungimento della convergenza e sul raggiungimento del vero massimo della funzione di verosimiglianza. Gretl utilizza uno dei due seguenti meccanismi per inizializzare i parametri, a seconda della specificazione del modello e del metodo di stima scelto.

1. Stima di un un modello AR puro usando i minimi quadrati (non lineari, se il modello lo richiede, altrimenti OLS). Impostazione del parametro AR secondo i risultati di questa regressione e impostazione dei parametri MA a un valore positivo ma piccolo (0.0001).
2. Il metodo di Hannan–Rissanen: per prima cosa, stima di un modello autoregressivo usando OLS e salvataggio dei residui. Quindi stima di un secondo modello OLS, aggiungendo appropriati ritardi dei residui della prima regressione, per ottenere le stime dei parametri MA.

Per vedere i dettagli della procedura di stima ARMA, basta aggiungere l'opzione `--verbose` al comando. Verrà mostrato un messaggio sul metodo di inizializzazione scelto, oltre che i valori dei parametri e della log-verosimiglianza ad ogni iterazione.

Oltre a questi meccanismi automatici di inizializzazione, l'utente può specificare manualmente un insieme di valori di partenza, attraverso il comando `set`: il primo argomento deve essere la parola chiave `initvals`, mentre il secondo dev'essere il nome di una matrice pre-specificata che contiene i valori iniziali. Ad esempio

```
matrix start = { 0, 0.85, 0.34 }
set initvals start
arma 1 1 ; y
```

La matrice specificata deve avere un numero di parametri pari a quello del modello: nell'esempio precedente ci sono tre parametri, visto che il modello contiene implicitamente una costante. La costante, se presente, è sempre indicata per prima, altrimenti, l'ordine in cui vanno indicati i parametri è lo stesso della specificazione nel comando `arma` o `arima`. Nell'esempio, la costante è impostata a zero, ϕ_1 a 0.85 e θ_1 a 0.34.

Per far tornare gretl ad usare l'inizializzazione automatica, basta eseguire il comando `set initvals auto`.

Stima con X-12-ARIMA

In alternativa alla stima di modelli ARMA usando le proprie funzionalità interne, gretl offre l'opzione di usare il programma esterno X-12-ARIMA per l'aggiustamento stagionale, curato dallo U.S. Census Bureau.

Gretl include un modulo che si interfaccia con X-12-ARIMA: traduce i comandi `arma` dalla sintassi vista sopra in una forma riconosciuta da X-12-ARIMA, esegue il programma e recupera i risultati per ulteriori analisi in gretl. Per usare questa funzionalità, occore installare X-12-ARIMA separatamente: sul sito di gretl, http://gretl.sourceforge.net/, sono disponibili pacchetti per MS Windows e GNU/Linux.

Per invocare X-12-ARIMA come motore di stima, basta usare l'opzione `--x-12-arima`, come in questo esempio:

```
arma p q ; y --x-12-arima
```

Come accade con le stime effettuate da gretl, per impostazione predefinita viene usata la massima verosimiglianza esatta, ma è possibile usare la massima verosimiglianza condizionale con l'opzione `--conditional`. Comunque, occorre notare che quando si usa X-12-ARIMA in modalità di massima verosimiglianza condizionale, quanto detto sopra a proposito dei diversi modi di trattare la media del processo y_t *non si applica*. Ossia, quando si usa X-12-ARIMA l'equazione stimata è la (20.2), a prescindere dal fatto che la stima sia condotta con la massima verosimiglianza esatta o condizionale.

Previsione

I modelli ARMA sono spesso usati a scopo previsionale. La componente autoregressiva, in particolare, offre la possibilità di prevedere l'andamento del processo "fuori dal campione" su un certo orizzonte temporale.

Gretl supporta le previsioni basate su modelli ARMA usando il metodo illustrato da Box e Jenkins (1976)[1]. L'algoritmo di Box e Jenkins produce un insieme di coefficienti AR integrati che tiene conto delle operazioni di differenziazione (stagionale e/o non stagionale) sulla variabile dipendente in un contesto ARIMA, rendendo così possibile una previsione per il livello della variabile originale. Per contrasto, se si differenzia una serie manualmente e poi si applica ARMA alla serie differenziata, le previsioni saranno riferite alla serie differenziata, non al suo livello. Questo punto è illustrato dall'Esempio 20.1. Le stime dei parametri sono identiche per i due modelli; le previsioni sono diverse ma coerenti tra loro: la variabile `fcdiff` emula la previsione ARMA (statica, per il periodo successivo all'interno del campione, e dinamica, fuori dal campione).

20.3 Test per radici unitarie

Il test ADF

Il test ADF (Augmented Dickey–Fuller) è implementato in gretl sotto forma della statistica t su φ nella regressione seguente:

$$\Delta y_t = \mu_t + \varphi y_{t-1} + \sum_{i=1}^{p} \gamma_i \Delta y_{t-i} + \epsilon_t. \tag{20.8}$$

Questa statistica test è probabilmente il più famoso e utilizzato test per radici unitarie. È un test a una coda la cui ipotesi nulla è $\varphi = 0$, mentre quella alternativa è $\varphi < 0$. Sotto l'ipotesi nulla, y_t deve essere differenziata almeno una volta per raggiungere la stazionarietà. Sotto l'ipotesi alternativa, y_t è già stazionaria e non richiede differenziazione. Quindi, grandi valori negativi della statistica test portano a rifiutare l'ipotesi nulla.

Un aspetto peculiare di questo test è che la sua distribuzione limite non è standard sotto l'ipotesi nulla: inoltre, la forma della distribuzione, e quindi i valori critici per il test, dipendono dalla forma del termine μ_t. Un'eccellente analisi di tutti i casi possibili è contenuta in Hamilton (1994), ma il soggetto è trattato anche in qualsiasi testo recente sulle serie storiche. Per quanto riguarda gretl, esso permette all'utente di scegliere la specificazione di μ_t tra quattro alternative:

μ_t	Opzione del comando
0	`--nc`
μ_0	`--c`
$\mu_0 + \mu_1 t$	`--ct`
$\mu_0 + \mu_1 t + \mu_1 t^2$	`--ctt`

Queste opzioni non sono mutualmente esclusive e possono essere usate insieme; in questo caso, la statistica verrà calcolata separatamente per ognuno dei casi. La scelta predefinita in

[1] Si veda in particolare il loro "Programma 4" alle pagine 505 e seguenti.

Esempio 20.1: Previsione ARIMA

```
open greene18_2.gdt
# Logaritmo del PIL nominale trimestrale degli USA, da 1950:1 a 1983:4
genr y = log(Y)
# La sua differenza prima
genr dy = diff(y)
# Teniamo 2 anni per la previsione fuori dal campione
smpl ; 1981:4
# Stima con ARIMA
arima 1 1 1 ; y
# Previsione su tutto il periodo
smpl --full
fcast fc1
# Torniamo al campione ristretto ed eseguiamo ARMA sulla differenza prima di y
smpl ; 1981:4
arma 1 1 ; dy
smpl --full
fcast fc2
genr fcdiff = (t<=1982:1)*(fc1 - y(-1)) + (t>1982:1)*(fc1 - fc1(-1))
# Confronto delle previsioni sull'ultimo periodo
smpl 1981:1 1983:4
print y fc1 fc2 fcdiff --byobs
```

Il risultato dell'ultimo comando è:

```
                y          fc1          fc2       fcdiff
1981:1      7.964086     7.940930      0.02668      0.02668
1981:2      7.978654     7.997576      0.03349      0.03349
1981:3      8.009463     7.997503      0.01885      0.01885
1981:4      8.015625     8.033695      0.02423      0.02423
1982:1      8.014997     8.029698      0.01407      0.01407
1982:2      8.026562     8.046037      0.01634      0.01634
1982:3      8.032717     8.063636      0.01760      0.01760
1982:4      8.042249     8.081935      0.01830      0.01830
1983:1      8.062685     8.100623      0.01869      0.01869
1983:2      8.091627     8.119528      0.01891      0.01891
1983:3      8.115700     8.138554      0.01903      0.01903
1983:4      8.140811     8.157646      0.01909      0.01909
```

gretl è quella di usare la combinazione `--c --ct --ctt`. Per ognuno dei casi, vengono calcolati p-value approssimativi usando l'algoritmo descritto in MacKinnon 1996.

Il comando di gretl da usare per eseguire il test è `adf`; ad esempio

```
adf 4 x1 --c --ct
```

calcola la statistica test come statistica-t per φ nell'equazione 20.8 con $p = 4$ nei due casi $\mu_t = \mu_0$ e $\mu_t = \mu_0 + \mu_1 t$.

Il numero di ritardi (p nell'equazione 20.8) deve essere scelto per assicurarsi che la (20.8) sia una parametrizzazione abbastanza flessibile per rappresentare adeguatamente la persistenza di breve termine di Δy_t. Scegliere un p troppo basso può portare a distorsioni di dimensione nel test, mentre sceglierlo troppo alto porta a una perdita di potenza del test. Per comodità dell'utente, il parametro p può essere determinato automaticamente. Impostando p a un numero negativo viene attivata una procedura sequenziale che parte da p ritardi e decrementa p fino a quando la statistica t per il parametro y_p supera 1.645 in valore assoluto.

Il test KPSS

Il test KPSS (Kwiatkowski, Phillips, Schmidt e Shin, 1992) è un test per radici unitarie in cui l'ipotesi nulla è l'opposto di quella del test ADF: l'ipotesi nulla è che la serie sia stazionaria, mentre l'ipotesi alternativa è che la serie sia $I(1)$.

L'intuizione alla base di questa statistica test è molto semplice: se y_t può essere scritta come $y_t = \mu + u_t$, dove u_t è un qualche processo stazionario a media nulla, non solo la media campionaria di y_t fornisce uno stimatore consistente di μ, ma la varianza di lungo periodo di u_t è un numero finito. Nessuna di queste proprietà è valida nel caso dell'ipotesi alternativa.

Il test si basa sulla seguente statistica:

$$\eta = \frac{\sum_{i=1}^{T} S_t^2}{T^2 \bar{\sigma}^2} \tag{20.9}$$

dove $S_t = \sum_{s=1}^{t} e_s$ e $\bar{\sigma}^2$ è una stima della varianza di lungo periodo di $e_t = (y_t - \bar{y})$. Sotto l'ipotesi nulla, questa statistica ha una distribuzione asintotica ben definita (non standard), che non dipende da parametri di disturbo ed è stata tabulata con metodi di simulazione. Sotto l'ipotesi alternativa, la statistica diverge.

Di conseguenza, è possibile costruire un test a una coda basato su η, dove H_0 è rifiutata se η è maggiore del valore critico desiderato; gretl fornisce i quantili del 90%, 95%, 97.5% e 99%.

Esempio di uso:

```
kpss m y
```

dove `m` è un intero che rappresenta la larghezza di banda, o la dimensione della finestra usata nella formula per stimare la varianza di lungo periodo:

$$\bar{\sigma}^2 = \sum_{i=-m}^{m} \left(1 - \frac{|i|}{m+1}\right) \hat{\gamma}_i$$

I termini $\hat{\gamma}_i$ denotano le autocovarianze empiriche di e_t dall'ordine $-m$ fino al m. Affinché questo stimatore sia consistente, m deve essere abbastanza grande da accomodare la persistenza di breve periodo di e_t, ma non troppo grande se paragonato all'ampiezza campionaria T. Nell'interfaccia grafica di gretl, il valore predefinito è pari alla parte intera di $4\left(\frac{T}{100}\right)^{1/4}$.

Il concetto visto sopra può essere generalizzato al caso in cui y_t è stazionario attorno a un trend deterministico. In questo caso, la formula (20.9) rimane invariata, ma la serie e_t è definita come residui della regressione OLS di y_t su una costante e un trend lineare. Questa seconda forma del test si ottiene aggiungendo l'opzione `--trend` al comando `kpss`:

```
kpss n y --trend
```

Si noti che in questo caso la distribuzione asintotica del test è diversa, e i valori critici riportati da gretl sono corretti di conseguenza.

I test di cointegrazione

FIXME discuss Engle—Granger here, and refer forward to the next chapter for the Johansen tests.

20.4 ARCH e GARCH

Il fenomeno dell'eteroschedasticità rappresenta la varianza non costante del termine di errore in un modello di regressione. L'eteroschedasticità condizionale autoregressiva (ARCH) è un fenomeno specifico dei modelli per serie storiche, in cui la varianza del termine di errore presenta un comportamento autoregressivo, ad esempio, la serie presenta periodi in cui la varianza dell'errore è relativamente ampia e periodi in cui è relativamente piccola. Questo tipo di comportamento si verifica spesso sui mercati finanziari: una notizia inaspettata può provocare periodi di maggior volatilità del mercato.

Un processo di errore ARCH di ordine q può essere rappresentato come

$$u_t = \sigma_t \varepsilon_t; \qquad \sigma_t^2 \equiv \mathrm{E}(u_t^2 | \Omega_{t-1}) = \alpha_0 + \sum_{i=1}^{q} \alpha_i u_{t-i}^2$$

dove le ε_t sono indipendenti e identicamente distribuite (iid) con media zero e varianza uno, e dove σ_t è la radice quadrata di σ_t^2. Ω_{t-1} denota il set informativo al tempo $t-1$ e σ_t^2 è la varianza condizionale: ossia, la varianza condizionata all'informazione che risale al tempo $t-1$ e precedente.

È importante notare la differenza tra un processo ARCH e un normale processo di errore autoregressivo. Quest'ultimo, nel caso più semplice (del primo ordine), può essere scritto come

$$u_t = \rho u_{t-1} + \varepsilon_t; \qquad -1 < \rho < 1$$

dove le ε_t sono iid con media zero e varianza costante σ^2. Con un errore AR(1), se ρ è positivo un valore positivo di u_t tenderà ad essere seguito, con probabilità maggiore di 0.5, da un valore positivo u_{t+1}. Con un processo di errore ARCH, un errore u_t dal valore assoluto elevato tenderà ad essere seguito da valori a loro volta elevati, ma senza assumere che i valori successivi siano dello stesso segno. La presenza di processi ARCH nelle serie finanziarie è un "fatto stilizzato" ed è coerente con l'ipotesi di efficienza dei mercati; d'altra parte, un comportamento autoregressivo dei prezzi dei titoli violerebbe l'efficienza del mercato.

È possibile testare per l'esistenza di un ARCH di ordine q nel modo seguente:

1. Stimare il modello in esame con OLS e salvare i quadrati dei residui, $\hat{u}_t^2$.
2. Eseguire una regressione ausiliaria in cui i quadrati dei residui sono regrediti su una costante e su q ritardi propri.
3. Trovare il valore TR^2 (ampiezza campionaria moltiplicata per R^2 non corretto) per la regressione ausiliaria.
4. Confrontare il valore TR^2 con la distribuzione χ^2 con q gradi di libertà, e se il p-value è "abbastanza piccolo" rifiutare l'ipotesi nulla di omoschedasticità, in favore dell'ipotesi alternativa dell'esistenza di un processo ARCH(q).

Questo test è implementato in gretl con il comando `arch`. Questo comando può essere eseguito dopo la stima OLS di un modello di serie storiche, o selezionandolo dal menù "Test" della finestra del modello (sempre dopo una stima OLS). Verrà mostrato il risultato del test, e, se il valore TR^2 dalla regressione ausiliaria ha un p-value minore di 0.10, vengono mostrate anche le stime ARCH. Queste stime sono sotto forma di minimi quadrati generalizzati (Generalized Least Squares, GLS), in particolare di minimi quadrati ponderati, dove i pesi sono

inversamente proporzionali agli scarti quadratici medi stimati degli errori, $\hat{\sigma}_t$, derivati dalla regressione ausiliaria.

Inoltre, il test ARCH è disponibile dopo aver stimato un'autoregressione vettoriale (VAR). In questo caso però non viene eseguita la stima GLS.

GARCH

Il semplice processo ARCH(q) è utile per introdurre il concetto generale di eteroschedasticità condizionale nelle serie storiche, ma si è rivelato insufficiente per il lavoro empirico. La dinamica della varianza dell'errore consentita dal modello ARCH(q) non è abbastanza ricca per rappresentare l'andamento tipico dei dati finanziari. Oggi è di uso più comune il modello ARCH generalizzato, o GARCH.

La rappresentazione della varianza di un processo nel modello GARCH è abbastanza (ma non esattamente) simile alla rappresentazione ARMA del livello di una serie storica. La varianza al tempo t può dipendere sia dai valori passati della varianza, sia dai valori passati dei quadrati degli errori, come mostra il seguente sistema di equazioni:

$$y_t = X_t\beta + u_t \tag{20.10}$$

$$u_t = \sigma_t\varepsilon_t \tag{20.11}$$

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i u_{t-i}^2 + \sum_{j=1}^{p} \delta_i \sigma_{t-j}^2 \tag{20.12}$$

Come nel caso precedente, ε_t è una variabile iid con varianza unitaria. X_t è una matrice di regressori (o nel caso più semplice un vettore con elementi pari a 1, che consente una media di y_t diversa da zero). Si noti che se $p = 0$, il GARCH si riduce a un ARCH(q): la generalizzazione è incorporata nei termini δ_i che moltiplicano i valori precedenti della varianza dell'errore.

In linea di principio, l'innovazione sottostante, ε_t, potrebbe seguire una qualsiasi distribuzione di probabilità, e oltre all'ovvia candidata rappresentata dalla normale, o Gaussiana, anche la distribuzione t è stata usata in questo contesto. Al momento gretl gestisce solo il caso in cui ε_t viene ipotizzata Gaussiana. Però se si usa l'opzione `--robust` del comando `garch`, lo stimatore usato da gretl per la matrice di covarianza può essere considerato uno stimatore di quasi massima verosimiglianza anche con disturbi non normali. Si veda oltre per le altre opzioni che riguardano la matrice di covarianza GARCH.

Esempio:

```
garch p q ; y const x
```

dove `p` ≥ 0 e `q` > 0 denotano i rispettivi ordini di ritardo come mostrati nell'equazione (20.12). Questi valori possono essere forniti in forma numerica o come nomi di variabili scalari preesistenti.

Stima GARCH

La stima dei parametri di un modello GARCH non è per nulla un compito semplice. Si consideri l'equazione 20.12: la varianza condizionale in ogni periodo, σ_t^2, dipende dalla varianza condizionale nei periodi precedenti, ma σ_t^2 non è osservata e deve essere stimata con qualche tipo di procedura di massima verosimiglianza. Gretl usa il metodo proposto da Fiorentini, Calzolari e Panattoni (1996)[2] che è stato adottato come metro di paragone nello studio dei risultati GARCH di McCullough e Renfro (1998). Esso usa le derivate analitiche prime e seconde della log-verosimiglianza, adotta un algoritmo del gradiente misto, sfruttando la matrice informativa nelle prime iterazioni, e quindi passa all'Hessiano in prossimità della massima verosimiglianza (questo andamento può essere verificato usando l'opzione `--verbose` del comando `garch` di gretl).

[2] L'algoritmo si basa sul codice Fortran rilasciato dagli autori nell'archivio del *Journal of Applied Econometrics* ed è usato per gentile concessione del Professor Fiorentini.

Sono disponibili varie opzioni per il calcolo della matrice di covarianza delle stime dei parametri ottenute con il comando `garch`. Per prima cosa, è possibile scegliere tra uno stimatore "standard" e uno "robusto". La scelta predefinita è quella dell'Hessiano, a meno che non si usi l'opzione `--robust`, nel cui caso viene usato lo stimatore QML. Una scelta più dettagliata è disponibile usando il comando `set`, come mostrato nella tabella 20.2.

Tabella 20.2: Opzioni per la matrice di covarianza GARCH

Commando	*Effetto*
`set garch_vcv hessian`	Usa l'Hessiano
`set garch_vcv im`	Usa la matrice di informazione
`set garch_vcv op`	Usa il prodotto esterno del gradiente
`set garch_vcv qml`	Stimatore QML
`set garch_vcv bw`	Stimatore "sandwich" di Bollerslev–Wooldridge

Non è infrequente, nella stima di un modello GARCH, che il calcolo iterativo delle stime fallisca nel raggiungere la convergenza. Affinché un modello GARCH abbia senso, sono necessari vincoli stringenti sui valori ammissibili dei parametri, e non sempre esiste un insieme di valori all'interno dello spazio dei parametri per cui la verosimiglianza viene massimizzata.

I vincoli in questione possono essere spiegati riferendosi al più semplice (e più comune) modello GARCH, dove $p = q = 1$. Nel modello GARCH(1, 1), la varianza condizionale è

$$\sigma_t^2 = \alpha_0 + \alpha_1 u_{t-1}^2 + \delta_1 \sigma_{t-1}^2 \tag{20.13}$$

Prendendo il valore atteso non condizionale della (20.13) si ottiene

$$\sigma^2 = \alpha_0 + \alpha_1 \sigma^2 + \delta_1 \sigma^2$$

così che

$$\sigma^2 = \frac{\alpha_0}{1 - \alpha_1 - \delta_1}$$

Affinché questa varianza non condizionale esista, occorre che $\alpha_1 + \delta_1 < 1$, e quindi occorre richiedere $\alpha_0 > 0$.

Un motivo comune per la non convergenza delle stime GARCH (ossia, un motivo comune per la non esistenza di valori α_i e δ_i che soddisfano le condizioni precedenti e nello stesso tempo massimizzano la verosimiglianza dei dati) è la cattiva specificazione del modello. È importante rendersi conto che il modello GARCH in sé prevede *solamente* che la volatilità dei dati dipende dal tempo. Se la *media* della serie in questione non è costante, o se il processo di errore non è solo eteroschedastico ma è anche autoregressivo, è necessario tener conto di questi fatti per formulare un modello appropriato. Ad esempio, può essere necessario dover prendere la differenza prima della variabile in questione e/o aggiungere opportuni regressori X_t al modello, come mostrato nella (20.10).

Capitolo 21

Cointegrazione e modelli vettoriali a correzione d'errore

21.1 Introduzione

I concetti correlati di cointegrazione e correzione d'errore sono stati al centro della ricerca in macroeconometria negli ultimi anni. L'aspetto interessante del Modello Vettoriale a Correzione di Errore (VECM) consiste nel fatto che permette al ricercatore di inserire una rappresentazione di relazioni di equilibrio economico in una specificazione abbastanza ricca basata sulle serie storiche. Questo approccio supera l'antica dicotomia tra i modelli strutturali, che rappresentavano fedelmente la teoria macroeconomica ma non si adattavano ai dati, e l'analisi delle serie storiche, che era più precisa nel riprodurre l'andamento dei dati, ma di difficile, se non impossibile, interpretazione in termini di teoria economica.

L'idea basilare della cointegrazione è strettamente collegata al concetto di radici unitarie (si veda la sezione 20.3). Si supponga di avere un insieme di variabili macroeconomiche di interesse, e di non poter rifiutare l'ipotesi che alcune di queste variabili, considerate individualmente, siano non-stazionarie. In particolare, si supponga che un sottoinsieme di queste variabili siano individualmente integrate di ordine 1, o I(1), ossia che non siano stazionarie, ma che la loro differenza prima sia stazionaria. Dati i problemi di tipo statistico che sono associati all'analisi dei dati non stazionari (ad esempio il problema della regressione spuria), l'approccio tradizionale in questo caso consiste nel prendere la differenza prima delle variabili prima di procedere con l'analisi statistica.

In questo modo però, si perde informazione importante. Può darsi che mentre le variabili sono I(1) prese singolarmente, esista una loro combinazione lineare che sia invece stazionaria, ossia I(0) (potrebbe esserci anche più di una combinazione lineare). In altri termini, mentre l'insieme delle variabili è libero di muoversi nel tempo, esistono comunque delle relazioni che legano fra di loro le variabili; è possibile interpretare queste relazioni, o *vettori di cointegrazione* come condizioni di equilibrio.

Ad esempio, ipotizziamo di scoprire che la quantità di moneta, M, il livello dei prezzi, P, il tasso di interesse nominale, R, e l'output, Y, siano tutti I(1). Secondo la teoria standard della domanda di moneta, dovremmo comunque aspettarci una relazione di equilibrio tra la quantità di moneta reale, il tasso d'interesse e l'output; ad esempio

$$m - p = \gamma_0 + \gamma_1 y + \gamma_2 r \qquad \gamma_1 > 0, \gamma_2 < 0$$

dove le variabili in minuscolo indicano i logaritmi. In equilibrio si ha quindi

$$m - p - \gamma_1 y - \gamma_2 r = \gamma_0$$

Nella realtà non ci si aspetta che questa condizione sia soddisfatta in ogni periodo, ma occorre ammettere la possibilità di disequilibri di breve periodo. Ma se il sistema ritorna all'equilibrio dopo un disturbo, ne consegue che il vettore $x = (m, p, y, r)'$ è limitato da un vettore di cointegrazione $\beta' = (\beta_1, \beta_2, \beta_3, \beta_4)$, tale che $\beta' x$ è stazionario (con una media pari a γ_0). Inoltre, se l'equilibrio è caratterizzato correttamente dal semplice modello visto sopra, si ha $\beta_2 = -\beta_1$, $\beta_3 < 0$ e $\beta_4 > 0$. Queste proprietà sono testabili attraverso l'analisi di cointegrazione.

Questa analisi consiste tipicamente in tre passi:

1. Test per verificare il numero di vettori di cointegrazione, ossia il *rango di cointegrazione* del sistema.

2. Stima di un VECM di rango appropriato, non soggetto ad altre restrizioni.
3. Test dell'interpretazione dei vettori di cointegrazione come condizioni di equilibrio, usando le restrizioni sugli elementi di questi vettori.

Le sezioni seguenti approfondiscono ognuno dei passi, aggiungendo altre considerazioni econometriche e spiegando come implementare l'analisi usando gretl.

21.2 Modelli vettoriali a correzione di errore (VECM) come rappresentazione di un sistema cointegrato

Si consideri un VAR di ordine p con una parte deterministica data da μ_t (tipicamente un polinomio nel tempo). È possibile scrivere il processo n-variato y_t come

$$y_t = \mu_t + A_1 y_{t-1} + A_2 y_{t-2} + \cdots + A_p y_{t-p} + \epsilon_t \tag{21.1}$$

Ma poiché $y_{t-1} \equiv y_t - \Delta y_t$ and $y_{t-i} \equiv y_{t-1} - (\Delta y_{t-1} + \Delta y_{t-2} + \cdots + \Delta y_{t-i+1})$, è possibile riscrivere l'equazione nel modo seguente:

$$\Delta y_t = \mu_t + \Pi y_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + \epsilon_t, \tag{21.2}$$

dove $\Pi = \sum_{i=1}^{p} A_i$ e $\Gamma_k = -\sum_{i=k}^{p} A_i$. Questa è la rappresentazione VECM della (21.1).

L'interpretazione della (21.2) dipende in modo cruciale da r, il rango della matrice Π.

- Se $r = 0$, i processi sono tutti I(1) e non cointegrati.
- Se $r = n$, Π è invertibile e i processi sono tutti I(0).
- La cointegrazione accade in tutti i casi intermedi, quando $0 < r < n$ e Π può essere scritta come $\alpha\beta'$. In questo caso, y_t è I(1), ma la combinazione $z_t = \beta' y_t$ è I(0). Se, ad esempio, $r = 1$ e il primo elemento di β fosse -1, si potrebbe scrivere $z_t = -y_{1,t} + \beta_2 y_{2,t} + \cdots + \beta_n y_{n,t}$, che è equivalente a dire che

  $$y_{1_t} = \beta_2 y_{2,t} + \cdots + \beta_n y_{n,t} - z_t$$

  è una relazione di equilibrio di lungo periodo: le deviazioni z_t possono non essere pari a zero, ma sono stazionarie. In questo caso, la (21.2) può essere scritta come

  $$\Delta y_t = \mu_t + \alpha\beta' y_{t-1} + \sum_{i=1}^{p-1} \Gamma_i \Delta y_{t-i} + \epsilon_t. \tag{21.3}$$

  Se β fosse noto, z_t sarebbe osservabile e tutti i restanti parametri potrebbero essere stimati con OLS. In pratica, la procedura stima per prima cosa β e tutto il resto poi.

Il rango di Π viene analizzato calcolando gli autovalori di una matrice ad essa strettamente legata che ha rango pari a quello di Π, ma che per costruzione è simmetrica e semidefinita positiva. Di conseguenza, tutti i suoi autovalori sono reali e non negativi e i test sul rango di Π possono quindi essere condotti verificando quanti autovalori sono pari a 0.

Se tutti gli autovalori sono significativamente diversi da 0, tutti i processi sono stazionari. Se, al contrario, c'è almeno un autovalore pari a 0, allora il processo y_t è integrato, anche se qualche combinazione lineare $\beta' y_t$ potrebbe essere stazionaria. All'estremo opposto, se non ci sono autovalori significativamente diversi da 0, non solo il processo y_t è non-stazionario, ma vale lo stesso per qualsiasi combinazione lineare $\beta' y_t$; in altre parole non c'è alcuna cointegrazione.

La stima procede tipicamente in due passi: per prima cosa si esegue una serie di test per determinare r, il rango di cointegrazione. Quindi, per un certo rango vengono stimati i parametri dell'equazione (21.3). I due comandi offerti da gretl per compiere queste operazioni sono rispettivamente `coint2` e `vecm`.

La sintassi di `coint2` è

```
coint2 p listay [ ; listax [ ; listaz ] ]
```

dove `p` è il numero di ritardi nella (21.1), `listay` è una lista che contiene le variabili y_t, `listax` è una lista opzionale di variabili esogene, e `listaz` è un'altra lista opzionale di variabili esogene il cui effetto è ipotizzato essere confinato alle relazioni di cointegrazione.

La sintassi di `vecm` è

```
vecm p r listay [ ; listax [ ; listaz ] ]
```

dove `p` è il numero di ritardi nella (21.1), `r` è il rango di cointegrazione, e le liste `listay`, `listax` e `listaz` hanno la stessa funzione che hanno nel comando `coint2`.

Entrambi i comandi supportano opzioni specifiche per trattare la componente deterministica μ_t; queste sono illustrate nella sezione seguente.

21.3 Interpretazione delle componenti deterministiche

L'inferenza statistica nell'ambito dei sistemi cointegrati dipende dalle ipotesi fatte a proposito dei termini deterministici, per cui si possono individuare i ben noti "cinque casi".

Nell'equazione (21.2), il termine μ_t di solito assume la forma seguente:

$$\mu_t = \mu_0 + \mu_1 \cdot t.$$

Affinché il modello si adatti nel modo migliore alle caratteristiche dei dati, occorre risolvere una questione preliminare. I dati sembrano seguire un trend deterministico? In caso positivo, si tratta di un trend lineare o quadratico?

Una volta stabilito questo, bisogna imporre restrizioni coerenti su μ_0 e μ_1. Ad esempio, se i dati non esibiscono un trend evidente, ciò significa che Δy_t vale zero in media, quindi è ragionevole assumere che anche il suo valore atteso sia zero. Scriviamo l'equazione (21.2) come

$$\Gamma(L)\Delta y_t = \mu_0 + \mu_1 \cdot t + \alpha z_{t-1} + \epsilon_t, \tag{21.4}$$

dove si ipotizza che $z_t = \beta' y_t$ sia stazionario e quindi possieda momenti finiti. Prendendo i valori attesi non condizionati, otteniamo

$$0 = \mu_0 + \mu_1 \cdot t + \alpha m_z.$$

Visto che il termine al primo membro non dipende da t, il vincolo $\mu_1 = 0$ pare appropriato. Per quanto riguarda μ_0, ci sono solo due modi per rendere vera l'espressione vista sopra: $\mu_0 = 0$ con $m_z = 0$, oppure μ_0 esattamente uguale a $-\alpha m_z$. Questa ultima possibilità è meno restrittiva nel senso che il vettore μ_0 può non essere pari a zero, ma è vincolato ad essere una combinazione lineare delle colonne di α. Ma in questo caso μ_0 può essere scritto come $\alpha \cdot c$, si può scrivere la (21.4) come

$$\Gamma(L)\Delta y_t = \alpha \begin{bmatrix} \beta' & c \end{bmatrix} \begin{bmatrix} y_{t-1} \\ 1 \end{bmatrix} + \epsilon_t.$$

La relazione di lungo periodo contiene quindi un'intercetta. Questo tipo di restrizione è scritta di solito nel modo seguente:

$$\alpha_\perp' \mu_0 = 0,$$

dove $\alpha_\perp$ è lo spazio nullo sinistro della matrice α.

È possibile dare un'intuizione del problema per mezzo di un semplice esempio. Si consideri una serie x_t che si comporta nel modo seguente:

$$x_t = m + x_{t-1} + \varepsilon_t$$

dove m è un numero reale e ε_t è un processo "rumore bianco" ("white noise"): x_t è quindi una "passeggiata aleatoria" ("random walk") con deriva pari a m. Nel caso particolare in cui $m = 0$, la deriva scompare e x_t è una passeggiata aleatoria pura.

Si consideri ora un altro processo y_t, definito da

$$y_t = k + x_t + u_t$$

dove, ancora, k è un numero reale e u_t è un processo a rumore bianco. Poiché u_t è stazionario per definizione, x_t e y_t sono cointegrate, ossia la loro differenza

$$z_t = y_t - x_t = k + u_t$$

è un processo stazionario. Per $k = 0$, z_t è un semplice rumore bianco a media zero, mentre per $k \neq 0$ il processo z_t è un rumore bianco con media diversa da zero.

Dopo alcune semplici sostituzioni, le due equazioni precedenti possono essere rappresentate congiuntamente come un sistema VAR(1)

$$\begin{bmatrix} y_t \\ x_t \end{bmatrix} = \begin{bmatrix} k+m \\ m \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y_{t-1} \\ x_{t-1} \end{bmatrix} + \begin{bmatrix} u_t + \varepsilon_t \\ \varepsilon_t \end{bmatrix}$$

o in forma VECM

$$\begin{aligned} \begin{bmatrix} \Delta y_t \\ \Delta x_t \end{bmatrix} &= \begin{bmatrix} k+m \\ m \end{bmatrix} + \begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_{t-1} \\ x_{t-1} \end{bmatrix} + \begin{bmatrix} u_t + \varepsilon_t \\ \varepsilon_t \end{bmatrix} = \\ &= \begin{bmatrix} k+m \\ m \end{bmatrix} + \begin{bmatrix} -1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} y_{t-1} \\ x_{t-1} \end{bmatrix} + \begin{bmatrix} u_t + \varepsilon_t \\ \varepsilon_t \end{bmatrix} = \\ &= \mu_0 + \alpha\beta' \begin{bmatrix} y_{t-1} \\ x_{t-1} \end{bmatrix} + \eta_t = \mu_0 + \alpha z_{t-1} + \eta_t, \end{aligned}$$

dove β è il vettore di cointegrazione e α è il vettore dei "loading" o "aggiustamenti".

Possiamo ora considerare tre casi possibili:

1. $m \neq 0$: in questo caso x_t ha un trend, come abbiamo appena visto; ne consegue che anche y_t segue un trend lineare perché in media si mantiene a una distanza fissa da x_t pari a k. Il vettore μ_0 non ha restrizioni.

2. $m = 0$ e $k \neq 0$: in questo caso, x_t non ha un trend, e di conseguenza neanche y_t. Tuttavia, la distanza media tra y_t e x_t è diversa da zero. Il vettore μ_0 è dato da

   $$\mu_0 = \begin{bmatrix} k \\ 0 \end{bmatrix}$$

   che è non nullo, quindi il VECM mostrato sopra ha un termine costante. La costante, tuttavia è soggetta alla restrizione che il suo secondo elemento deve essere pari a 0. Più in generale, μ_0 è un multiplo del vettore α. Si noti che il VECM potrebbe essere scritto anche come

   $$\begin{bmatrix} \Delta y_t \\ \Delta x_t \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & -1 & -k \end{bmatrix} \begin{bmatrix} y_{t-1} \\ x_{t-1} \\ 1 \end{bmatrix} + \begin{bmatrix} u_t + \varepsilon_t \\ \varepsilon_t \end{bmatrix}$$

   che incorpora l'intercetta nel vettore di cointegrazione. Questo è il caso chiamato "costante vincolata".

3. $m = 0$ e $k = 0$: questo caso è il più vincolante: chiaramente, né x_t né y_t hanno un trend, e la loro distanza media è zero. Anche il vettore μ_0 vale 0, quindi questo caso può essere chiamato "senza costante".

Nella maggior parte dei casi, la scelta tra queste tre possibilità si basa su un misto di osservazione empirica e di ragionamento economico. Se le variabili in esame sembrano seguire un trend lineare, è opportuno non imporre alcun vincolo all'intercetta. Altrimenti occorre chiedersi se ha senso specificare una relazione di cointegrazione che includa un'intercetta diversa da

zero. Un esempio appropriato potrebbe essere la relazione tra due tassi di interesse: in generale questi non hanno un trend, ma il VAR potrebbe comunque avere un'intercetta perché la differenza tra i due (lo "spread" sui tassi d'interesse) potrebbe essere stazionaria attorno a una media diversa da zero (ad esempio per un premio di liquidità o di rischio).

L'esempio precedente può essere generalizzato in tre direzioni:

1. Se si considera un VAR di ordine maggiore di 1, l'algebra si complica, ma le conclusioni sono identiche.
2. Se il VAR include più di due variabili endogene, il rango di cointegrazione r può essere maggiore di 1. In questo caso α è una matrice con r colonne e il caso con la costante vincolata comporta che μ_0 sia una combinazione lineare delle colonne di α.
3. Se si include un trend lineare nel modello, la parte deterministica del VAR diventa $\mu_0 + \mu_1$. Il ragionamento è praticamente quello visto sopra, tranne per il fatto che l'attenzione è ora posta su μ_1 invece che su μ_0. La controparte del caso con "costante vincolata" discusso sopra è un caso con "trend vincolato", così che le relazioni di cointegrazione includono un trend, ma la differenza prima delle variabili in questione no. Nel caso di un trend non vincolato, il trend appare sia nelle relazioni di cointegrazione sia nelle differenze prime, il che corrisponde alla presenza di un trend quadratico nelle variabili (espresse in livelli).

Per gestire i cinque casi, gretl fornisce le seguenti opzioni per i comandi `coint2` e `vecm`:

μ_t	*Opzione*	*Descrizione*
0	`--nc`	Senza costante
$\mu_0, \alpha'_\perp \mu_0 = 0$	`--rc`	Costante vincolata
μ_0	(predefinito)	Costante non vincolata
$\mu_0 + \mu_1 t, \alpha'_\perp \mu_1 = 0$	`--crt`	Costante + trend vincolato
$\mu_0 + \mu_1 t$	`--ct`	Costante + trend non vincolato

Si noti che le opzioni viste sopra sono mutualmente esclusive. Inoltre, è possibile usare l'opzione `--seasonal` per aggiungere a μ_t delle dummy stagionali centrate. In ogni caso, i p-value sono calcolati con le approssimazioni indicate in Doornik (1998).

21.4 I test di cointegrazione di Johansen

I due test di cointegrazione di Johansen vengono usati per stabilire il rango di β, in altre parole il numero di vettori di cointegrazione del sistema. Essi sono il test "λ-max", per le ipotesi sui singoli autovalori, e il test "traccia", per le ipotesi congiunte. Si supponga che gli autovalori λ_i siano ordinati dal maggiore al minore. L'ipotesi nulla per il test "λ-max" sul i-esimo autovalore è che sia $\lambda_i = 0$. Invece, il test traccia corrispondente considera l'ipotesi che sia $\lambda_j = 0$ per ogni $j \geq i$.

Il comando `coint2` di gretl esegue questi due test; la voce corrispondente nel menù dell'interfaccia grafica è "Modello, Serie Storiche, COINT - Test di cointegrazione, Johansen".

Come nel test ADF, la distribuzione asintotica dei test varia a seconda della componente deterministica μ_t incluso nel VAR (si veda la sezione 21.3). Il codice seguente usa il file di dati `denmark`, fornito insieme a gretl, per replicare l'esempio proposto da Johansen nel suo libro del 1995.

```
open denmark
coint2 2 LRM LRY IBO IDE --rc --seasonal
```

In questo caso, il vettore y_t nell'equazione (21.2) comprende le quattro variabili `LRM`, `LRY`, `IBO`, `IDE`. Il numero dei ritardi equivale a p nella (21.2) (ossia, il numero dei ritardi del modello scritto in forma VAR). Di seguito è riportata parte dell'output:

```
Test di Johansen:
Numero di equazioni = 4
Ordine dei ritardi = 2
Periodo di stima: 1974:3 - 1987:3 (T = 53)

Caso 2: costante vincolata
Rango Autovalore Test traccia p-value   Test Lmax  p-value
   0    0,43317     49,144 [0,1284]     30,087 [0,0286]
   1    0,17758     19,057 [0,7833]     10,362 [0,8017]
   2    0,11279     8,6950 [0,7645]     6,3427 [0,7483]
   3   0,043411     2,3522 [0,7088]     2,3522 [0,7076]
```

Sia il test traccia, sia quello λ-max portano ad accettare l'ipotesi nulla che il più piccolo autovalore valga 0 (ultima riga della tabella), quindi possiamo concludere che le serie non sono stazionarie. Tuttavia, qualche loro combinazione lineare potrebbe essere I(0), visto che il test λ-max rifiuta l'ipotesi che il rango di Π sia 0 (anche se il test traccia dà un'indicazione meno netta in questo senso, con un p-value pari a 0.1284).

21.5 Identificazione dei vettori di cointegrazione

Il problema centrale nella stima dell'equazione (21.2) consiste nel trovare una stima di Π che abbia rango r per costruzione, così che possa essere scritta come $\Pi = \alpha\beta'$, dove β è la matrice che contiene i vettori di cointegrazione e α contiene i coefficienti di "aggiustamento" o "loading", per cui le variabili endogene rispondono a una deviazione dall'equilibrio nel periodo precedente.

Senza ulteriore specificazione, il problema ha molte soluzioni (in effetti ne ha infinite). I parametri α e β sono sotto-identificati: se tutte le colonne di β sono vettori di cointegrazione, anche qualsiasi loro combinazione lineare arbitraria è un vettore di cointegrazione. In altre parole, se $\Pi = \alpha_0\beta_0'$ per specifiche matrici α_0 e β_0, allora Π è anche uguale a $(\alpha_0\Sigma)(\Sigma^{-1}\beta_0')$ per qualsiasi matrice conformabile e non singolare Σ. Per trovare una soluzione unica, è quindi necessario imporre alcune restrizioni su α e/o β. Si può dimostrare che il numero minimo di restrizioni necessarie per garantire l'identificazione è pari a r^2. Un primo passo banale consiste nel normalizzare un coefficiente per ogni colonna rendendolo pari a 1 (o a -1, a seconda dei gusti), il che aiuta anche a interpretare i restanti coefficienti come parametri delle relazioni di equilibrio; tuttavia, questo basta solo nel caso in cui $r = 1$.

Il metodo usato da gretl in modo predefinito è chiamato "normalizzazione di Phillips", o "rappresentazione triangolare"[1]. Il punto di partenza consiste nello scrivere β in forma partizionata, come in

$$\beta = \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix},$$

dove β_1 è una matrice $r \times r$ e β_2 è $(n - r) \times r$. Assumendo che β_1 abbia rango pieno, β può essere post-moltiplicata da β_1^{-1}, ottenendo

$$\hat{\beta} = \begin{bmatrix} I \\ \beta_2\beta_1^{-1} \end{bmatrix} = \begin{bmatrix} I \\ -B \end{bmatrix},$$

I coefficienti prodotti da gretl sono i $\hat{\beta}$, mentre B è nota come matrice dei coefficienti non vincolati. Nei termini della relazione di equilibrio sottostante, la normalizzazione di Phillips

[1] Per fare confronti con altri studi, potrebbe essere necessario normalizzare β in modo diverso. Usando il comando `set` è possibile scrivere `set vecm_norm diag` per scegliere la normalizzazione che scala le colonne della β originaria in modo che valga $\beta_{ij} = 1$ per $i = j$ e $i \leq r$, come è mostrato nella sezione empirica di Boswijk e Doornik (2004). Una soluzione alternativa è `set vecm_norm first`, che scala β in modo che gli elementi della prima riga siano pari a 1. Per sopprimere del tutto la normalizzazione basta usare `set vecm_norm none`, mentre per tornare all'impostazione predefinita `set vecm_norm phillips`.

esprime il sistema di r relazioni di equilibrio come

$$\begin{aligned} y_{1,t} &= b_{1,r+1}y_{r+1,t} + \ldots + b_{1,n}y_{n,t} \\ y_{2,t} &= b_{2,r+1}y_{r+1,t} + \ldots + b_{2,n}y_{n,t} \\ &\vdots \\ y_{r,t} &= b_{r,r+1}y_{r+1,t} + \ldots + b_{r,n}y_{r,t} \end{aligned}$$

dove le prime r variabili sono espresse come funzione delle restanti $n - r$.

Anche se la rappresentazione triangolare assicura la soluzione del problema statistico della stima di β, le relazioni di equilibrio che risultano possono essere difficili da interpretare. In questo caso, l'utente potrebbe voler cercare l'identificazione specificando manualmente il sistema di r^2 vincoli che gretl userà per produrre una stima di β.

Come esempio, si consideri il sistema di domanda di moneta presentato nella sezione 9.6 di Verbeek (2004). Le variabili usate sono `m` (il logaritmo della quantità di moneta reale M1), `infl` (l'inflazione), `cpr` (i tassi di interesse sui "commercial paper"), `y` (il logaritmo del PIL reale) e `tbr` (i tassi di interesse sui "Treasury bill")[2].

La stima di β si può ottenere con questi comandi:

```
open money.gdt
smpl 1954:1 1994:4
vecm 6 2 m infl cpr y tbr --rc
```

e la parte rilevante dei risultati è questa:

```
Stime Massima verosimiglianza usando le osservazioni 1954:1-1994:4 (T = 164)
Rango di cointegrazione = 2
Caso 2: costante vincolata

Vettori di cointegrazione (errori standard tra parentesi)

m           1.0000       0.0000
           (0.0000)     (0.0000)
infl        0.0000       1.0000
           (0.0000)     (0.0000)
cpr         0.56108      -24.367
           (0.10638)     (4.2113)
y          -0.40446     -0.91166
           (0.10277)     (4.0683)
tbr        -0.54293       24.786
           (0.10962)     (4.3394)
const       -3.7483       16.751
           (0.78082)     (30.909)
```

L'interpretazione dei coefficienti della matrice di cointegrazione β sarebbe più semplice se si potesse associare un significato ad ognuna delle sue colonne. Questo è possibile ipotizzando l'esistenza di due relazioni di lungo periodo: una equazione di domanda di moneta

$$\mathtt{m} = c_1 + \beta_1\mathtt{infl} + \beta_2\mathtt{y} + \beta_3\mathtt{tbr}$$

e una equazione per premio sul rischio

$$\mathtt{cpr} = c_2 + \beta_4\mathtt{infl} + \beta_5\mathtt{y} + \beta_6\mathtt{tbr}$$

[2] Questo dataset è disponibile nel pacchetto verbeek; si veda la pagina http://gretl.sourceforge.net/gretl_data_it.html.

che implica che la matrice di cointegrazione può essere normalizzata come

$$\beta = \begin{bmatrix} -1 & 0 \\ \beta_1 & \beta_4 \\ 0 & -1 \\ \beta_2 & \beta_5 \\ \beta_3 & \beta_6 \\ c_1 & c_2 \end{bmatrix}$$

Questa rinormalizzazione può essere compiuta per mezzo del comando `restrict`, da eseguire dopo il comando `vecm`, oppure, se si usa l'interfaccia grafica, scegliendo la voce dal menù "Test, Vincoli lineari". La sintassi per specificare i vincoli è abbastanza intuitiva[3]:

```
restrict
  b[1,1] = -1
  b[1,3] = 0
  b[2,1] = 0
  b[2,3] = -1
end restrict
```

che produce

```
Vettori di cointegrazione (errori standard tra parentesi)

m          -1.0000       0.0000
           (0.0000)     (0.0000)
infl     -0.023026     0.041039
        (0.0054666)   (0.027790)
cpr         0.0000      -1.0000
           (0.0000)     (0.0000)
y          0.42545    -0.037414
          (0.033718)    (0.17140)
tbr      -0.027790       1.0172
        (0.0045445)   (0.023102)
const       3.3625      0.68744
           (0.25318)     (1.2870)
```

21.6 Restrizioni sovra-identificanti

Uno degli scopi dell'imporre restrizioni su un sistema VECM è quello di raggiungere l'identificazione del sistema. Se queste restrizioni sono semplici normalizzazioni, esse non sono testabili e non dovrebbero avere effetti sulla verosimiglianza massimizzata. Tuttavia, si potrebbe anche voler formulare vincoli su β e/o α che derivano dalla teoria economica che è alla base delle relazioni di equilibrio; restrizioni sostanziali di questo tipo sono quindi testabili attraverso il metodo del rapporto di verosimiglianza.

Gretl può testare restrizioni lineari generali nella forma

$$R_b \text{vec}(\beta) = q \tag{21.5}$$

e/o

$$R_a \text{vec}(\alpha) = 0 \tag{21.6}$$

Si noti che la restrizione su β può essere non-omogenea ($q \neq 0$) ma la restrizione su α deve essere omogenea. Non sono supportate le restrizioni non-lineari, né quelle incrociate tra β e α. Nel caso in cui $r > 1$ queste restrizioni possono essere in comune tra tutte le colonne di β (o

[3] Si noti che in questo contesto, stiamo trasgredendo la convenzione usata di solito per gli indici delle matrici, visto che usiamo il primo indice per riferirci alla *colonna* di β (il particolare vettore di cointegrazione). Questa è la pratica usata di solito nella letteratura sulla cointegrazione, visto che sono le colonne di β (le relazioni di cointegrazione o gli errori di equilibrio) che interessano in modo particolare.

α) o possono essere specifiche a certe colonne di queste matrici. Questo è il caso discusso in Boswijk (1995) e in Boswijk e Doornik (2004, capitolo 4.4).

Le restrizioni (21.5) e (21.6) possono essere scritte in forma esplicita come

$$\text{vec}(\beta) = H\phi + h_0 \tag{21.7}$$

e

$$\text{vec}(\alpha') = G\psi \tag{21.8}$$

rispettivamente, dove ϕ e ψ sono i vettori parametrici liberi associati con β e α rispettivamente. Possiamo riferirci collettivamente a tutti i parametri liberi come θ (il vettore colonna formato dalla concatenazione di ϕ e ψ). Gretl usa internamente questa rappresentazione per testare le restrizioni.

Se la lista delle restrizioni passata al comando `restrict` contiene più restrizioni di quelle necessarie a garantire l'identificazione, viene eseguito un test LR; inoltre è possibile usare l'opzione `--full` con il comando `restrict`, in modo che vengano mostrate le stime complete per il sistema vincolato (inclusi i termini Γ_i), e che il sistema vincolato diventi così il "modello attuale" ai fini di ulteriori test. In questo modo è possibile eseguire test cumulativi, come descritto nel capitolo 7 di Johansen (1995).

Sintassi

La sintassi completa per specificare le restrizioni è un'estensione di quella mostrata nella sezione precedente. All'interno di un blocco `restrict...end restrict` è possibile usare dichiarazione della forma

combinazione lineare di parametri = *scalare*

dove la combinazione lineare di parametri comprende la somma ponderata di elementi individuali di β o α (ma non di entrambi nella stessa combinazione); lo scalare al secondo membro deve essere pari a 0 per combinazioni che riguardano α, ma può essere qualsiasi numero reale per combinazioni che riguardano β. Di seguito vengono mostrati alcuni esempi di restrizioni valide:

```
b[1,1] = 1.618
b[1,4] + 2*b[2,5] = 0
a[1,3] = 0
a[1,1] - a[1,2] = 0
```

Una sintassi speciale è riservata al caso in cui una certa restrizione deve essere applicata a tutte le colonne di β: in questo caso, viene usato un indice per ogni termine `b` e non si usano le parentesi quadre. Quindi la sintassi seguente

```
restrict
  b1 + b2 = 0
end restrict
```

corresponde a

$$\beta = \begin{bmatrix} \beta_{11} & \beta_{21} \\ -\beta_{11} & -\beta_{21} \\ \beta_{13} & \beta_{23} \\ \beta_{14} & \beta_{24} \end{bmatrix}$$

La stessa convenzione viene usata per α: quando si usa solo un indice per ogni termine `a`, la restrizione viene applicata a tutte le r righe di α, o in altre parole le variabili indicate sono debolmente esogene. Ad esempio la formulazione

```
restrict
  a3 = 0
  a4 = 0
end restrict
```

specifica che le variabili 3 e 4 non rispondono alla deviazione dall'equilibrio nel periodo precedente.

Infine, è disponibile una scorciatoia per definire restrizioni complesse (al momento solo in relazione a β): è possibile specificare R_b e q, come in $R_b \text{vec}(\beta) = q$, dando i nomi di matrici definite in precedenza. Ad esempio,

```
matrix I4 = I(4)
matrix vR = I4**(I4~zeros(4,1))
matrix vq = mshape(I4,16,1)
restrict
  R = vR
  q = vq
end restrict
```

impone manualmente la normalizzazione di Phillips sulle stime di β per un sistema con rango di cointegrazione 4.

Un esempio

Brand e Cassola (2004) propongono un sistema di domanda di moneta per l'area Euro, in cui postulano tre relazioni di equilibrio di lungo periodo:

Domanda di moneta	$m = \beta_l l + \beta_y y$
Equazione di Fisher	$\pi = \phi l$
Teoria delle aspettative sui tassi di interesse	$l = s$

dove m è la domanda di moneta reale, l e s sono i tassi di interesse a lungo e a breve termine, y è l'output e π è l'inflazione[4]. I nomi di queste variabili nel file di dati di gretl sono rispettivamente `m_p`, `rl`, `rs`, `y` e `infl`.

Il rango di cointegrazione ipotizzato dagli autori è pari a 3, e ci sono 5 variabili, per un totale di 15 elementi nella matrice β. Per l'identificazione sono richieste $3 \times 3 = 9$ restrizioni, e un sistema esattamente identificato avrebbe $15 - 9 = 6$ parametri liberi. Tuttavia, le tre relazioni di lungo periodo ipotizzate contengono solo 3 parametri liberi, quindi il rango di sovraidentificazione è 3.

L'esempio 21.1 replica la tabella 4 a pagina 824 dell'articolo di Brand e Cassola[5]. Si noti che viene usato l'accessorio `$lnl` dopo il comando `vecm` per salvare la log-verosimiglianza non vincolata e l'accessorio `$rlnl` dopo il comando `restrict` per la sua controparte vincolata.

L'esempio continua nello script 21.2, dove vengono eseguiti ulteriori test per controllare (a) se l'elasticità al reddito nell'equazione di domanda di moneta è pari a 1 ($\beta_y = 1$) e (b) se la relazione di Fisher è omogenea ($\phi = 1$). Visto che è stata usata l'opzione `--full` con il comando `restrict` iniziale, è possibile applicare le restrizioni aggiuntive senza dover ripetere quelle precedenti. Il secondo script contiene alcuni comandi `printf` che non sono strettamente necessari, servono solo a formattare meglio i risultati. I dati portano a rifiutare entrambe le ipotesi aggiuntive, con p-value di 0.002 e 0.004.

Un altro tipo di test eseguito spesso è quello dell'"esogeneità debole". In questo contesto, si dice che una variabile è debolmente esogena se tutti i coefficienti nella riga corrispondente della matrice α sono pari a zero. In questo caso, la variabile non reagisce alle deviazioni da alcuno degli equilibri di lungo periodo e può essere considerata una forza autonoma dal resto del sistema.

[4] Una formulazione tradizionale dell'equazione di Fisher invertirebbe i ruoli delle variabili nella seconda equazione, ma questo dettaglio non è rilevante in questo contesto; inoltre, la teoria delle aspettative sui tassi implica che la terza relazione di equilibrio dovrebbe includere una costante per il premio di liquidità. In ogni caso, visto che in questo esempio il sistema è stimato con il termine costante non vincolato, il premio per la liquidità viene assorbito nell'intercetta del sistema e scompare da z_t.

[5] Fatta eccezione per quelli che sembrano alcuni errori di stampa nell'articolo.

Esempio 21.1: Stima di un sistema di domanda di moneta con vincoli su β

Input:

```
open brand_cassola.gdt

# Alcune trasformazioni
m_p = m_p*100
y = y*100
infl = infl/4
rs = rs/4
rl = rl/4

# Replica della tabella 4 a pagina 824
vecm 2 3 m_p infl rl rs y -q
genr ll0 = $lnl

restrict --full
  b[1,1] = 1
  b[1,2] = 0
  b[1,4] = 0
  b[2,1] = 0
  b[2,2] = 1
  b[2,4] = 0
  b[2,5] = 0
  b[3,1] = 0
  b[3,2] = 0
  b[3,3] = 1
  b[3,4] = -1
  b[3,5] = 0
end restrict
genr ll1 = $rlnl
```

Output parziale:

```
Log-verosimiglianza non vincolata (lu) = 116.60268
Log-verosimiglianza vincolata (lr) = 115.86451
2 * (lu - lr) = 1.47635
P(Chi-quadro(3) > 1.47635) = 0.68774

Beta (vettori di cointegrazione, errori standard tra parentesi)

m_p        1.0000       0.0000       0.0000
          (0.0000)     (0.0000)     (0.0000)
infl       0.0000       1.0000       0.0000
          (0.0000)     (0.0000)     (0.0000)
rl         1.6108     -0.67100       1.0000
         (0.62752)   (0.049482)     (0.0000)
rs         0.0000       0.0000      -1.0000
          (0.0000)     (0.0000)     (0.0000)
y         -1.3304       0.0000       0.0000
        (0.030533)     (0.0000)     (0.0000)
```

Esempio 21.2: Test ulteriori sul sistema di domanda di moneta

Input:

```
restrict
  b[1,5] = -1
end restrict
genr ll_uie = $rlnl

restrict
  b[2,3] = -1
end restrict
genr ll_hfh = $rlnl

# Replica della tabella 5 a pagina 824
printf "Test su zero restrizioni nello spazio di cointegrazione:\n"
printf "  test LR, rango = 3: chi^2(3) = %6.4f [%6.4f]\n", 2*(llO-ll1), \
        pvalue(X, 3, 2*(llO-ll1))

printf "Elasticità al reddito unitaria: test LR, rango = 3:\n"
printf "  chi^2(4) = %g [%6.4f]\n", 2*(llO-ll_uie), \
        pvalue(X, 4, 2*(llO-ll_uie))

printf "Omogeneità nell'ipotesi di Fisher:\n"
printf "  test LR, rango = 3: chi^2(4) = %6.3f [%6.4f]\n", 2*(llO-ll_hfh), \
        pvalue(X, 4, 2*(llO-ll_hfh))
```

Output:

```
Test su zero restrizioni nello spazio di cointegrazione:
  test LR, rango = 3: chi^2(3) = 1.4763 [0.6877]
Elasticità al reddito unitaria: test LR, rango = 3:
  chi^2(4) = 17.2071 [0.0018]
Omogeneità nell'ipotesi di Fisher:
  test LR, rango = 3: chi^2(4) = 15.547 [0.0037]
```

Il codice nell'Esempio 21.3 esegue questo test per ognuna delle variabili, replicando la prima colonna della Tabella 6 a pagina 825 di Brand e Cassola (2004). I risultati mostrano che l'ipotesi di esogeneità debole può essere accettata per i tassi di interesse a lungo termine e il PIL reale (rispettivamente con p-value 0.07 e 0.08).

Esempio 21.3: Test per l'esogeneità debole

Input:

```
restrict
  a1 = 0
end restrict
ts_m = 2*(ll0 - $rlnl)

restrict
  a2 = 0
end restrict
ts_p = 2*(ll0 - $rlnl)

restrict
  a3 = 0
end restrict
ts_l = 2*(ll0 - $rlnl)

restrict
  a4 = 0
end restrict
ts_s = 2*(ll0 - $rlnl)

restrict
  a5 = 0
end restrict
ts_y = 2*(ll0 - $rlnl)

loop foreach i m p l s y --quiet
  printf "\Delta $i\t%6.3f [%6.4f]\n", ts_$i, pvalue(X, 6, ts_$i)
end loop
```

Output (variabile, test LR, p-value):

```
\Delta m        18.111 [0.0060]
\Delta p        21.067 [0.0018]
\Delta l        11.819 [0.0661]
\Delta s        16.000 [0.0138]
\Delta y        11.335 [0.0786]
```

Identificazione e testabilità

Un punto che può risultare poco chiaro a proposito delle restrizioni sui VECM è che l'identificazione (la restrizione identifica il sistema?) e la testabilità (la restrizione è testabile?) sono questioni abbastanza distinte. Le restrizioni possono essere identificanti ma non testabili; in modo meno ovvio, possono anche essere testabili ma non identificanti.

Questo può essere visto facilmente in un sistema a rango 1. La restrizione $\beta_1 = 1$ è identificante (identifica la scala di β) ma, essendo un semplice cambiamento di scala, non è testabile. D'altra parte, la restrizione $\beta_1 + \beta_2 = 0$ è testabile — il sistema con questo requisito imposto avrà quasi sicuramente una minore verosimiglianza massimizzata — ma non è identificante; lascia ancora aperta la scelta della scala di β.

Abbiamo visto sopra che il numero di restrizioni deve essere pari ad almeno r^2, dove r è il rango di cointegrazione. Questa è una condizione necessaria ma non sufficiente. In effetti, quando $r > 1$ può essere abbastanza difficile giudicare quando un certo insieme di restrizioni è

identificante. Gretl usa il metodo suggerito da Doornik (1995), dove l'identificazione è valutata tramite il rango della matrice di informazione.

Può essere dimostrato che per le restrizioni del tipo della (21.7) e della (21.8) la matrice di informazione ha lo stesso rango della matrice Jacobiana

$$\mathcal{J}(\theta) = \left[(I_p \otimes \beta)G : (\alpha \otimes I_{p_1})H \right]$$

Una condizione sufficiente per l'identificazione è che il rango di $\mathcal{J}(\theta)$ sia pari al numero dei parametri liberi. Il rango di questa matrice è valutato esaminando i suoi valori singolari in un punto scelto a caso dello spazio dei parametri. Agli scopi pratici trattiamo questa condizione come se fosse sia necessaria che sufficiente, ossia tralasciamo i casi speciali in cui l'identificazione potrebbe essere raggiunta senza che questa condizione sia soddisfatta[6].

21.7 Metodi di soluzione numerica

In generale, il problema della stima ML per il VECM vincolato non ha una soluzione in forma chiusa, quindi il massimo deve essere cercato con metodi numerici[7]. In alcuni casi, la convergenza può risultare difficile da raggiungere, e gretl fornisce vari modi di risolvere il problema.

Algoritmi switching e LBFGS

Sono disponibili due metodi di massimizzazione in gretl: quello predefinito è l'algoritmo di switching proposto in Boswijk e Doornik (2004); l'alternativa è una variante a memoria limitata dell'algoritmo BFGS, che usa derivate analitiche: LBFGS. Per usare questo metodo occorre aggiungere l'opzione `--lbfgs` al comando `restrict`.

L'algoritmo di switching funziona massimizzando esplicitamente la verosimiglianza ad ogni iterazione rispetto a $\hat{\phi}$, $\hat{\psi}$ e $\hat{\Omega}$ (la matrice di covarianza dei residui). Questo metodo condivide una caratteristica con la classica procedura degli autovalori di Johansen, ossia: può gestire un insieme di restrizioni che non identificano completamente i parametri.

D'altra parte, LBFGS richiede che il modello sia completamente identificato, quindi in alcuni casi occorrerà aggiungere alle restrizioni che interessano alcune normalizzazioni con lo scopo di identificare i parametri. Ad esempio, si può usare in tutto o in parte la normalizzazione di Phillips (si veda la sezione 21.5).

Né l'algoritmo di switching né quello LBFGS garantiscono di raggiungere la soluzione ML globale[8]: l'ottimizzatore potrebbe fermarsi su un massimo locale (o, nel caso dell'algoritmo di switching, su un punto di sella).

La soluzione (o la sua mancanza) può essere sensibile al valore iniziale scelto per θ. Per impostazione predefinita, gretl sceglie un valore iniziale usando un metodo deterministico basato su Boswijk (1995), ma sono disponibili due altre opzioni: è possibile aggiustare l'inizializzazione usando il metodo della ricottura simulata (con l'opzione `--jitter`), oppure è possibile fornire esplicitamente un valore iniziale per θ.

Il metodo di inizializzazione predefinito è il seguente:

1. Calcolare la ML non vincolata $\hat{\beta}$ usando la procedura di Johansen.
2. Se la restrizione su β è non-omogenea, usare il metodo proposto da Boswijk (1995):

$$\phi_0 = -[(I_r \otimes \hat{\beta}_\perp)' H]^+ (I_r \otimes \hat{\beta}_\perp)' h_0 \tag{21.9}$$

[6] Si veda Boswijk e Doornik (2004, pp. 447–8) per la discussione di questo problema.

[7] L'eccezione è costituita dal caso in cui le restrizioni sono omogenee, sono comuni a tutti i β o a tutti gli α (nel caso $r > 1$), e includono solo β o solo α. Queste restrizioni sono gestite con il metodo degli autovalori modificati proposto da Johansen (1995): risolviamo direttamente lo stimatore ML, senza aver bisogno di metodi iterativi.

[8] Nello sviluppo del codice per il test dei VECM in gretl sono stati considerati un certo numero di "casi difficili" individuati da varie fonti. Vogliamo ringraziare Luca Fanelli dell'Università di Bologna e Sven Schreiber della Goethe University di Francoforte, per aver suggerito vari test per il codice VECM di gretl

dove $\hat{\beta}'_\perp \hat{\beta} = 0$ e A^+ denota l'inversa di Moore–Penrose di A. Altrimenti

$$\phi_0 = (H'H)^{-1} H' \mathrm{vec}(\hat{\beta}) \tag{21.10}$$

3. $\mathrm{vec}(\beta_0) = H\phi_0 + h_0$.
4. Calcolare la ML non vincolata $\hat{\alpha}$ condizionale su β_0, come da Johansen:

$$\hat{\alpha} = S_{01}\beta_0(\beta_0' S_{11}\beta_0)^{-1} \tag{21.11}$$

5. Se α è vincolata da $\mathrm{vec}(\alpha') = G\psi$, allora $\psi_0 = (G'G)^{-1}G' \mathrm{vec}(\hat{\alpha}')$ e $\mathrm{vec}(\alpha_0') = G\psi_0$.

Metodi di inizializzazione alternativi

Come è stato detto in precedenza, gretl offre la possibilità di impostare l'inizializzazione usando la ricottura simulata.

L'idea di base è questa: si inizia in un certo punto dello spazio parametrico, e per ognuna di n iterazioni (al momento $n = 4096$) si sceglie a caso un punto all'interno di un certo raggio di distanza dal precedente e si determina la verosimiglianza nel nuovo punto. Se la verosimiglianza è maggiore, si salta nel nuovo punto, altrimenti si salta con probabilità P (e si rimane nel punto precedente con probabilità $1 - P$). Man mano che le iterazioni procedono, il sistema "si raffredda" gradualmente, ossia il raggio delle perturbazioni casuali si riduce, così come la probabilità di fare un salto quando la verosimiglianza non aumenta.

Nel corso di questa procedura vengono valutati molti punti dello spazio dei parametri, cominciando dal punto scelto dalla procedura deterministica, che chiameremo θ_0. Uno di questi punti sarà il "migliore" nel senso che produce la più alta verosimiglianza: lo si chiami θ^*. Questo punto può avere o non avere una verosimiglianza maggiore di quella di θ_0. La procedura ha inoltre un punto finale, θ_n, che può essere o non essere il "migliore".

La regola seguita da gretl per scegliere un valore iniziale per θ basandosi sulla ricottura simulata è questa: usare θ^* se $\theta^* > \theta_0$, altrimenti usare θ_n. Ossia: se otteniamo un miglioramento nella verosimiglianza tramite la ricottura simulata, ne approfittiamo; d'altra parte, se non otteniamo un miglioramento usiamo comunque la ricottura per la randomizzazione del punto iniziale. La sperimentazione ha indicato che quest'ultimo metodo può aiutare.

Se l'inizializzazione deterministica fallisce, l'alternativa alla randomizzazione consiste nello scegliere esplicitamente dei valori iniziali. L'utente può farlo passando un vettore predefinito al comando `set` con il parametro `initvals`, come nell'esempio

```
set initvals myvec
```

I dettagli dipendono dall'algoritmo scelto. Nel caso dell'algoritmo di switching ci sono due opzioni per sceglire i valori iniziali. Quella più comoda (per la maggior parte delle persone supponiamo) consiste nello specificare una matrice che contiene $\mathrm{vec}(\beta)$ seguito da $\mathrm{vec}(\alpha)$. Ad esempio:

```
open denmark.gdt
vecm 2 1 LRM LRY IBO IDE --rc --seasonals

matrix BA = {1, -1, 6, -6, -6, -0.2, 0.1, 0.02, 0.03}
set initvals BA
restrict
  b[1] = 1
  b[1] + b[2] = 0
  b[3] + b[4] = 0
end restrict
```

In questo esempio tratto da Johansen (1995) il rango di cointegrazione è 1 e ci sono 4 variabili, ma il modello contiene una costante vincolata (opzione `--rc`) così che β ha 5 elementi. La matrice α ha 4 elementi, uno per equazione. Quindi la matrice `BA` può essere letta come

$$(\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \alpha_1, \alpha_2, \alpha_3, \alpha_4)$$

L'altra opzione, obbligatoria quando si usa LBFGS, consiste nello specificare i valori iniziali in termini dei parametri non vincolati, ϕ e ψ. Questo metodo è meno ovvio da comprendere: come ricordato sopra, la restrizione in forma implicita $R\text{vec}(\beta) = q$ ha la forma esplicita $\text{vec}(\beta) = H\phi + h_0$, dove $H = R_\perp$, lo spazio nullo destro di R. Il vettore ϕ è più corto, per il numero di restrizioni, di $\text{vec}(\beta)$. L'utente attento capirà cosa occorre fare. L'altro punto di cui tenere conto è che se α non è vincolato, la lunghezza *effettiva* di ψ è 0, visto che è ottimale calcolare α con la formula di Johansen, condizionale su β (equazione 21.11 sopra). Segue un esempio:

```
open denmark.gdt
vecm 2 1 LRM LRY IBO IDE --rc --seasonals

matrix phi = {-8, -6}
set initvals phi
restrict --lbfgs
  b[1] = 1
  b[1] + b[2] = 0
  b[3] + b[4] = 0
end restrict
```

In questa formulazione più economica, l'inizializzatore specifica solo i due parametri liberi di ϕ (5 elementi in β meno 3 restrizioni). Non c'è motivo di dare valori a ψ visto che α non è vincolato.

Rimozione della scala

Si consideri una versione più semplice della restrizione discussa nella sezione precedente, ossia,

```
restrict
  b[1] = 1
  b[1] + b[2] = 0
end restrict
```

Questa restrizione comprende un vincolo sostanziale e testabile — che la somma di β_1 e β_2 sia pari a zero — e una normalizzazione, o scalatura, $\beta_1 = 1$. Sorge la questione se non sia più semplice e affidabile massimizzare la verosimiglianza senza imporre $\beta_1 = 1$[9]. Se così fosse, potremmo tenere nota di questa normalizzazione, rimuoverla allo scopo di massimizzare la verosimiglianza, e quindi re-imporla scalando il risultato.

Purtroppo non è possibile dire in anticipo se una "rimozione di scala" di questo tipo darà risultati migliori per un certo problema di stima, ma nella maggior parte dei casi sembra essere così. Gretl quindi opera una rimozione di scala nei casi in cui è fattibile, a meno che

- l'utente non lo vieti, usando l'opzione `--no-scaling` del comando restrict; oppure
- l'utente fornisca uno specifico vettore di valori iniziali; oppure
- l'utente scelga l'algoritmo LBFGS per la massimizzazione.

La rimozione di scala viene giudicata non fattibile se ci sono restrizioni incrociate tra le colonne di β, o se ci sono restrizioni non-omogenee che coinvolgono più di un elemento di β.

In aggiunta, l'esperienza ha suggerito che la rimozione della scala non è consigliabile se il sistema è esattamente identificato con le normalizzazioni incluse, quindi in questo caso non viene eseguita. Per "esattamente identificato" si intende che il sistema non sarebbe identificato se una qualsiasi delle restrizioni fosse rimossa. Da questo punto di vista, l'esempio visto sopra non è esattamente identificato, visto che rimuovendo la seconda restrizione non si intaccherebbe l'identificazione; gretl in questo caso eseguirebbe la rimozione di scala a meno che l'utente non indichi altrimenti.

[9] Dal punto di vista numerico è più semplice. In linea di principio non dovrebbe fare differenza.

Capitolo 22

Variabili dipendenti discrete e censurate

22.1 Modelli logit e probit

Capita spesso che si voglia specificare e stimare un modello in cui la variabile dipendente non è continua ma discreta. Un esempio tipico è quello di un modello in cui la variabile dipendente è lo stato lavorativo di un individuo (1 = occupato, 0 = disoccupato). Un modo comodo per formalizzare questa situazione consiste nel considerare la variabile y_i come una variabile aleatoria di Bernoulli e analizzarne la distribuzione condizionata alle variabili esplicative x_i, ossia

$$y_i = \begin{cases} 1 & P_i \\ 0 & 1 - P_i \end{cases} \tag{22.1}$$

dove $P_i = P(y_i = 1|x_i)$ è una funzione delle variabili esplicative x_i.

Nella maggior parte dei casi, la funzione P_i è una funzione di ripartizione F, applicata a una combinazione lineare delle x_i. Nel modello probit, si usa la funzione di ripartizione normale, mentre il modello logit usa la funzione logistica $\Lambda()$. Quindi si ha

$$\text{probit} \qquad P_i = F(z_i) = \Phi(z_i) \tag{22.2}$$

$$\text{logit} \qquad P_i = F(z_i) = \Lambda(z_i) = \frac{1}{1 + e^{-z_i}} \tag{22.3}$$

$$z_i = \sum_{j=1}^{k} x_{ij}\beta_j \tag{22.4}$$

dove z_i è chiamata la funzione *indicatrice*. Si noti che in questo caso, i coefficienti β_j non possono essere interpretati come derivate parziali di $E(y_i|x_i)$ rispetto a x_{ij}. Comunque, per un dato valore di x_i è possibile calcolare il vettore delle "pendenze", ossia

$$\text{slope}_j(\bar{x}) = \left.\frac{\partial F(z)}{\partial x_j}\right|_{z=\bar{z}}$$

Gretl calcola automaticamente le pendenze assegnando a ogni variabile esplicativa un valore pari alla sua media.

Un modo equivalente di formulare questo modello consiste nell'ipotizzare l'esistenza di una variabile non osservata y_i^* che può essere descritta nel modo seguente:

$$y_i^* = \sum_{j=1}^{k} x_{ij}\beta_j + \varepsilon_i = z_i + \varepsilon_i \tag{22.5}$$

Si osserva $y_i = 1$ quando $y_i^* > 0$ e $y_i = 0$ altrimenti. Se si assume ε_i come normale, si ottiene il modello probit, mentre il modello logit assume che la funzione di densità di ε_i sia

$$\lambda(\varepsilon_i) = \frac{\partial \Lambda(\varepsilon_i)}{\partial \varepsilon_i} = \frac{e^{-\varepsilon_i}}{(1 + e^{-\varepsilon_i})^2}$$

gretl stima sia il modello probit che quello logit col metodo della massima verosimiglianza; poiché le equazioni degli score non hanno una soluzione in forma chiusa, vengono usate procedure di ottimizzazione numerica. La maggior parte delle volte queste richiedono poche iterazioni per raggiungere la convergenza, ma è possibile visualizzare i dettagli dell'algoritmo di massimizzazione usando l'opzione `--verbose`.

Esempio 22.1: Stima di semplici modelli logit e probit

```
open greene19_1

logit GRADE const GPA TUCE PSI
probit GRADE const GPA TUCE PSI
```

Come esempio, riproduciamo i risultati esposti nel capitolo 21 di Greene (2000), dove viene valutata l'efficacia di un programma per insegnare l'economia osservando i miglioramenti nei voti degli studenti. Eseguendo il codice contenuto nell'esempio 22.1 si ottengono i seguenti risultati:

```
Modello 1: Stime Logit usando le 32 osservazioni 1-32
Variabile dipendente: GRADE

      VARIABILE      COEFFICIENTE       ERRORE STD    STAT T       PENDENZA
                                                                 (alla media)
  const               -13,0213          4,93132      -2,641
  GPA                   2,82611         1,26294       2,238      0,533859
  TUCE                  0,0951577       0,141554      0,672      0,0179755
  PSI                   2,37869         1,06456       2,234      0,449339

  Media di GRADE = 0,344
  Numero dei casi 'previsti correttamente' = 26 (81,2%)
  f(beta'x) nella media delle variabili indipendenti = 0,189
  Pseudo-R-quadro di McFadden = 0,374038
  Log-verosimiglianza = -12,8896
  Test del rapporto di verosimiglianza: Chi-quadro(3) = 15,4042 (p-value 0,001502)
  Criterio di informazione di Akaike (AIC) = 33,7793
  Criterio bayesiano di Schwarz (BIC) = 39,6422
  Criterio di Hannan-Quinn (HQC) = 35,7227

             Previsto
               0    1
  Effettivo 0  18    3
            1   3    8

Modello 2: Stime Probit usando le 32 osservazioni 1-32
Variabile dipendente: GRADE

      VARIABILE      COEFFICIENTE       ERRORE STD    STAT T       PENDENZA
                                                                 (alla media)
  const                 -7,45232        2,54247      -2,931
  GPA                    1,62581        0,693883      2,343      0,533347
  TUCE                   0,0517288      0,0838903     0,617      0,0169697
  PSI                    1,42633        0,595038      2,397      0,467908

  Media di GRADE = 0,344
  Numero dei casi 'previsti correttamente' = 26 (81,2%)
  f(beta'x) nella media delle variabili indipendenti = 0,328
  Pseudo-R-quadro di McFadden = 0,377478
  Log-verosimiglianza = -12,8188
  Test del rapporto di verosimiglianza: Chi-quadro(3) = 15,5459 (p-value 0,001405)
  Criterio di informazione di Akaike (AIC) = 33,6376
  Criterio bayesiano di Schwarz (BIC) = 39,5006
  Criterio di Hannan-Quinn (HQC) = 35,581

             Previsto
```

```
                0    1
Effettivo 0    18    3
          1     3    8
```

In questo contesto, la funzione accessoria `$uhat` assume un significato speciale: produce i residui generalizzati come sono definiti in Gourieroux *et al* (1987), che possono essere interpretati come stimatori non distorti dei disturbi latenti ε_t. Questi sono definiti come

$$u_i = \begin{cases} y_i - \hat{P}_i & \text{per il modello logit} \\ y_i \cdot \frac{\phi(\hat{z}_i)}{\Phi(\hat{z}_i)} - (1 - y_i) \cdot \frac{\phi(\hat{z}_i)}{1-\Phi(\hat{z}_i)} & \text{per il modello probit} \end{cases} \tag{22.6}$$

Tra l'altro, i residui generalizzati sono spesso usati a scopo diagnostico; ad esempio, è molto facile costruire un test per variabili omesse equivalente al test LM usato tipicamente nel contesto della regressione lineare: l'esempio 22.2 mostra come eseguire un test per l'aggiunta di una variabile.

Esempio 22.2: Test per l'aggiunta di una variabile in un modello probit

```
open greene19_1

probit GRADE const GPA PSI
series u = $uhat

ols u const GPA PSI TUCE -q
printf "Test per l'aggiunta della variabile TUCE:\n"
printf "Rsq * T = %g (p. val. = %g)\n", $trsq, pvalue(X,1,$trsq)
```

Modelli ordinati

Questi modelli sono semplici variazioni sui normali modelli logit/probit, utilizzati di solito nei casi in cui la variabile dipendente assume valori discreti e ordinati, non necessariamente quantitativi. Ad esempio, questi modelli possono essere applicati per analizzare casi in cui la variabile dipendente è un giudizio qualitativo come "Buono", "Medio", "Scarso". Ipotizzando di avere p categorie, la probabilità che l'individuo i ricada nella j-esima categoria è dato da

$$P(y_i = j|x_i) = \begin{cases} F(z_i + \mu_0) & \text{per } j = 0 \\ F(z_i + \mu_j) - F(z_i + \mu_{j-1}) & \text{per } 0 < j < p \\ 1 - F(z_i + \mu_{p-1}) & \text{per } j = p \end{cases} \tag{22.7}$$

I parametri ignoti μ_j sono chiamati "punti di taglio" e sono stimati insieme ai β'. Ai fini dell'identificazione, μ_0 è ipotizzato pari a 0. In termini della variabile non osservata y_i^*, il modello può essere espresso in modo equivalente come $P(y_i = j|x_i) = P(\mu_{j-1} \leq y_i^* < \mu_j)$.

Esempio 22.3: Modello probit ordinato

```
open pension.gdt
series pctstck = pctstck/50
discrete pctstck
probit pctstck const choice age educ female black married finc25 finc35 \
  finc50 finc75 finc100 finc101 wealth89 prftshr
```

Per applicare questi modelli, la variabile dipendente deve essere marcata come discreta e il suo valore minimo deve essere pari a 0. L'esempio 22.3 riproduce la stima proposta nel capitolo 15 di Wooldridge (2002a). Si noti che gretl non fornisce un comando separato per i modelli ordinati: i comandi `logit` e `probit` stimano automaticamente le versioni ordinate se la variabile dipendente non è binaria (a patto che sia stata marcata in precedenza come discreta).

Dopo aver stimato modelli ordinati, la variabile `$uhat` contiene i residui generalizzati, come avviene per i modelli binari; in più, la variabile `$yhat` contiene $\hat{z}_i$, così è possibile calcolare una stima non distorta della variabile latente y_i^* semplicemente facendo l'addizione delle due.

Logit multinomiale

Quando la variabile dipendente non è binaria e non ha un ordinamento naturale, si usano modelli *multinomiali*. Gretl non fornisce ancora un'implementazione interna per questo tipo di modelli, ma alcuni casi semplici possono essere gestiti usando il comando `mle` command (si veda il capitolo 17). Di seguito viene presentato un esempio di modello logit multinomiale. Si assuma che la variabile dipendente y_i possa avere valori interi $0, 1, \dots p$. La probabilità che sia $y_i = k$ è data da

$$P(y_i = k|x_i) = \frac{\exp(x_i\beta_k)}{\sum_{j=0}^{p} \exp(x_i\beta_j)}$$

Ai fini dell'identificazione del modello, uno dei valori deve essere preso come quello "di riferimento"; di solito si assume che valga $\beta_0 = 0$, nel cui caso

$$P(y_i = k|x_i) = \frac{\exp(x_i\beta_k)}{1 + \sum_{j=1}^{p} \exp(x_i\beta_j)}$$

e

$$P(y_i = 0|x_i) = \frac{1}{1 + \sum_{j=1}^{p} \exp(x_i\beta_j)}.$$

L'esempio 22.4 riproduce la Tabella 15.2 di Wooldridge (2002a), che si basa su dati sulle scelte di carriera contenuti in Keane e Wolpin (1997). La variabile dipendente è lo stato occupazionale di un individuo (0 = studente; 1 = non studente e non occupato; 2 = occupato) e le variabili esplicative sono il livello di educazione e l'esperienza lavorativa (in livelli e nei quadrati), oltre a una variabile binaria "oscura". Il dataset completo è di tipo panel, ma l'analisi presentata di seguito riguarda solo i dati del 1987. Per i dettagli sulle operazioni matriciali usate in questo script, si veda il capitolo 12.

22.2 Il modello Tobit

Il modello Tobit viene usato quando la variabile dipendente di un modello è censurata[1]. Si ipotizzi che una variabile latente y_i^* possa essere descritta come

$$y_i^* = \sum_{j=1}^{k} x_{ij}\beta_j + \varepsilon_i,$$

dove $\varepsilon_i \sim N(0, \sigma^2)$. Se le y_i^* fossero osservabili, i parametri del modello potrebbero essere stimati con i minimi quadrati ordinari. Al contrario, si supponga di poter osservare y_i, definita come

$$y_i = \begin{cases} y_i^* & \text{for} \quad y_i^* > 0 \\ 0 & \text{for} \quad y_i^* \leq 0 \end{cases} \tag{22.8}$$

In questo caso, regredire y_i sulle x_i non produce stime consistenti dei parametri β, perché la media condizionale $E(y_i|x_i)$ non è pari a $\sum_{j=1}^{k} x_{ij}\beta_j$. Come si può dimostrare, nemmeno

[1] Stiamo assumendo che i dati siano censurati per quanto riguarda i valori inferiori a zero. I casi di censura per valori maggiori di zero, o in corrispondenza di valori diversi da zero, possono essere trattati facilmente ridefinendo la variabile dipendente. Il caso più generale di censura da due lati non è contemplato automaticamente da gretl, ma è possibile stimare tali modelli usando il comando `mle` (si veda il capitolo 17).

Esempio 22.4: Logit multinomiale

```
function mlogitlogprobs(series y, matrix X, matrix theta)

  scalar n = max(y)
  scalar k = cols(X)
  matrix b = mshape(theta,k,n)

  matrix tmp = X*b
  series ret = -ln(1 + sumr(exp(tmp)))

  loop for i=1..n --quiet
    series x = tmp[,i]
    ret += (y=$i) ? x : 0
  end loop

  return series ret

end function

open Keane.gdt
status = status-1 # dep. var. must be 0-based
smpl (year=87 & ok(status)) --restrict

matrix X = { educ exper expersq black const }
scalar k = cols(X)
matrix theta = zeros(2*k, 1)

mle loglik = mlogitlogprobs(status,X,theta)
  params theta
end mle --verbose --hessian
```

restringere il campione alle osservazioni diverse da zero non produrrebbe stime consistenti. La soluzione sta nello stimare i parametri con la massima verosimiglianza. La sintassi è semplicemente

```
tobit depvar indvars
```

Come al solito, è possibile visualizzare i progressi dell'algoritmo di massimizzazione usando l'opzione `--verbose` e `$uhat` contiene i residui generalizzati. Si noti che in questo caso il residuo generalizzato è definito come $\hat{u}_i = E(\varepsilon_i | y_i = 0)$ per le osservazioni censurate, quindi l'uguaglianza $\hat{u}_i = y_i - \hat{y}_i$ vale solo per le osservazioni non censurate, ossia quando vale $y_i > 0$.

Un'importante differenza tra lo stimatore Tobit e quello OLS è che le conseguenze della non-normalità del termine di disturbo sono molto più severe, visto che la non-normalità implica la non consistenza per lo stimatore Tobit. Per questo motivo, fra i risultati del modello Tobit viene mostrato anche il test di normalità di Chesher–Irish (1987).

Modello di selezione del campione

Nel modello di selezione del campione (noto anche come modello "Tobit II"), esistono due variabili latenti:

$$y_i^* = \sum_{j=1}^{k} x_{ij}\beta_j + \varepsilon_i \tag{22.9}$$

$$s_i^* = \sum_{j=1}^{p} z_{ij}\gamma_j + \eta_i \tag{22.10}$$

e la regola di osservazione è data da

$$y_i = \begin{cases} y_i^* & \text{for} \quad s_i^* > 0 \\ \blacklozenge & \text{for} \quad s_i^* \leq 0 \end{cases} \tag{22.11}$$

In questo contesto, il simbolo ◆ indica che per alcune osservazioni non sono disponibili dati per y: y_i può essere 0 o mancante, o assumere qualsiasi altro valore. Di solito si usa una variabile dummy d_i per escludere le osservazioni censurate.

Una delle applicazioni più popolari di questo modello in econometria prevede un'equazione dei salari e un'equazione della partecipazione alla forza lavoro: viene osservato solo il salario delle persone occupate. Se y_i^* e s_i^* fossero (condizionalmente) indipendenti, non ci sarebbe motivo per non usare lo stimatore OLS per stimare l'equazione (22.9); ma in altri casi, lo stimatore OLS non produce stime consistenti dei parametri β_j.

Visto che l'indipendenza condizionale tra y_i^* e s_i^* è equivalente all'indipendenza condizionale tra ε_i e η_i, si può modellare la co-dipendenza tra ε_i e η_i come

$$\varepsilon_i = \lambda\eta_i + v_i;$$

sostituendo l'espressione precedente nella (22.9), si ottiene il modello che viene effettivamente stimato:

$$y_i = \sum_{j=1}^{k} x_{ij}\beta_j + \lambda\hat{\eta}_i + v_i,$$

quindi l'ipotesi per cui la censura non ha importanza equivale all'ipotesi $H_0 : \lambda = 0$, che può essere testata facilmente.

I parametri possono essere stimati con la massima verosimiglianza nell'ipotesi di normalità congiunta di ε_i e η_i, ma un metodo alternativo molto usato è il cosidetto stimatore *Heckit*, che prende il nome da Heckman (1979). La procedura può essere schematizzata nel modo seguente: per prima cosa viene stimato un modello probit per l'equazione (22.10); quindi, vengono inseriti i residui generalizzati nell'equazione (22.9) per correggere gli effetti della selezione del campione.

Gretl fornisce il comando `heckit` per eseguire la stima; la sintassi è

```
heckit y X ; d Z
```

dove `y` è la variabile dipendente, `X` è una lista di regressori, `d` è una variabile dummy che vale 1 per le osservazioni non censurate, e `Z` è una lista di variabili esplicative per l'equazione di censura.

Visto che quello della massima verosimiglianza è il metodo scelto più spesso, per impostazione predefinita gretl calcola le stime ML. Le stime Heckit a due passi si ottengono usando l'opzione `--two-step`. Dopo la stima, la variabile `$uhat` contiene i residui generalizzati. Come nel modello Tobit ordinario, i residui sono pari alla differenza tra i valori y_i effettivi e stimati solo per le osservazioni non censurate (quelle per cui vale $d_i = 1$).

L'esempio 22.5 mostra due stime dal dataset usato in Mroz (1987): la prima replica la tabella 22.7 di Greene (2003)[2], mentre la seconda replica la tabella 17.1 di Wooldridge (2002a)

Esempio 22.5: Modello Heckit

```
open mroz.gdt

genr EXP2 = AX^2
genr WA2 = WA^2
genr KIDS = (KL6+K618)>0

# Greene's specification

list X = const AX EXP2 WE CIT
list Z = const WA WA2 FAMINC KIDS WE

heckit WW X ; LFP Z --two-step
heckit WW X ; LFP Z

# Wooldridge's specification

series NWINC = FAMINC - WW*WHRS
series lww = log(WW)
list X = const WE AX EXP2
list Z = X NWINC WA KL6 K618

heckit lww X ; LFP Z --two-step
```

[2] Si noti che le stime prodotte da gretl non coincidono con quelle contenute nel libro, ma con quelle elencate nella pagina degli errata corrige del libro: http://pages.stern.nyu.edu/~wgreene/Text/Errata/ERRATA5.htm.

Parte III

Dettagli tecnici

Capitolo 23

Gretl e TEX

23.1 Introduzione

TEX, sviluppato inizialmente da Donald Knuth della Stanford University e poi migliorato da centinaia di sviluppatori di tutto il mondo, è il punto di riferimento per la composizione di testi scientifici. Gretl fornisce vari comandi che consentono di vedere un'anteprima o di stampare i risultati delle analisi econometriche usando TEX, oltre che di salvare i risultati in modo da poter essere successivamente modificati con TEX.

Questo capitolo spiega in dettaglio le funzionalità di gretl che interessano TEX. La prima sezione descrive i menù nell'interfaccia grafica, la sezione 23.3 discute vari modi di personalizzare i risultati prodotti da TEX, la sezione 23.4 spiega come gestire la codifica dei caratteri per le lingue diverse dall'inglese, mentre la sezione 23.5 fornisce alcune indicazioni per conoscere meglio TEXe installarlo, se ancora non lo si ha sul proprio computer. Per chiarezza: TEX non è incluso nella distribuzione standard di gretl, è un pacchetto separato, che comprende vari programmi e file di supporto.

Prima di procedere, può essere utile chiarire brevemente il modo in cui viene prodotto un documento usando TEX. La maggior parte di questi dettagli non necessitano di alcuna preoccupazione da parte dell'utente, visto che sono gestiti automaticamente da gretl, ma una comprensione di base dei meccanismi all'opera può aiutare nella scelta delle opzioni a disposizione.

Il primo passo consiste nella creazione di un file "sorgente" testuale, che contiene il testo o le espressioni matematiche che compongono il documento, assieme ad alcuni comandi mark-up che regolano la formattazione del documento. Il secondo passo consiste nel fornire il file a un motore che esegue la formattazione vera e propria. Tipicamente, si tratta di:

- un programma chiamato latex, che genera un risultato in formato DVI (device-independent), oppure
- un programma chiamato pdflatex, che genera un risultato in formato PDF[1].

Per visualizzare le anteprime, si può utilizzare un visualizzatore DVI, (tipicamente xdvi sui sistemi GNU/Linux) o PDF (ad esempio Adobe Acrobat Reader o xpdf, a seconda di come è stato processato il sorgente. Se si sceglie la via del DVI, c'è un ultimo passo per creare un documento stampabile, che tipicamente richiede il programma dvips per generare un file in formato PostScript. Se si sceglie la via del PDF, il documento è pronto per essere stampato.

Sulle piattaforme MS Windows e Mac OS X, gretl richiama pdflatex per processare il file sorgente e si aspetta che il sistema operativo sia in grado di eseguire un programma per visualizzare il file PDF; il formato DVI non è supportato. Su GNU/Linux, la scelta predefinita è per il formato DVI, ma se si preferisce usare il PDF basta seguire queste istruzioni: dal menù "Strumenti, Preferenze, Generali", nella sezione "Programmi", impostare il "Comando per compilare i file TeX" a pdflatex; inoltre assicurarsi che il "Comando per visualizzare i file PDF" sia ben impostato.

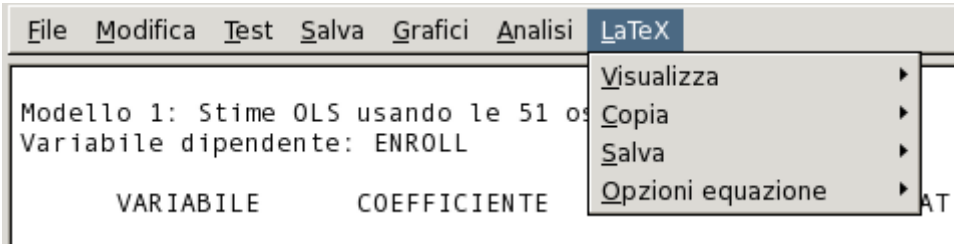
[1] Gli utenti più esperti hanno dimestichezza anche con il cosiddetto "plain TEX", che viene processato dal programma tex. La maggioranza degli utenti di TEX, comunque, usa le macro fornite dal programma LATEX, sviluppato inizialmente da Leslie Lamport. Gretl non supporta il plain TEX.

23.2 I comandi TEXnei menù

La finestra del modello

La maggior parte dei comandi di gretl relativi a TEX si trova nel menù "LaTeX" della finestra del modello, che contiene le voci "Visualizza", "Copia", "Salva" e "Opzioni equazione" (si veda la Figura 23.1).

Figura 23.1: Il menù LATEX della finestra del modello



Ognuna delle prime tre voci, a sua volta, contiene due opzioni, intitolate "Tabella" ed "Equazione". Selezionando "Tabella", il modello viene rappresentato in forma tabulare, consentendo la rappresentazione più completa ed esplicita dei risultati; l'esempio mostrato nella Tabella 23.1 è stato creato proprio scegliendo il comando "Copia, Tabella" di gretl (per brevità sono state omesse alcune righe).

Tabella 23.1: Esempio dell'output LATEX tabulare

Modello 1: Stime OLS usando le 51 osservazioni 1-51
Variabile dipendente: ENROLL

Variabile	Coefficiente	Errore Std.	statistica t	p-value
const	0.241105	0.0660225	3.6519	0.0007
CATHOL	0.223530	0.0459701	4.8625	0.0000
PUPIL	−0.00338200	0.00271962	−1.2436	0.2198
WHITE	−0.152643	0.0407064	−3.7499	0.0005

Media della variabile dipendente	0.0955686
D.S. della variabile dipendente	0.0522150
Somma dei quadrati dei residui	0.0709594
Errore standard dei residui ($\hat{\sigma}$)	0.0388558
R^2	0.479466
$\bar{R}^2$ corretto	0.446241
$F(3, 47)$	14.4306

L'opzione "Equazione" si spiega da sé: i risultati della stima vengono scritti sotto forma di equazione, nel modo seguente

$$\widehat{\text{ENROLL}} = \underset{(0.066022)}{0.241105} + \underset{(0.04597)}{0.223530}\,\text{CATHOL} - \underset{(0.0027196)}{0.00338200}\,\text{PUPIL} - \underset{(0.040706)}{0.152643}\,\text{WHITE}$$

$$T = 51 \quad \bar{R}^2 = 0.4462 \quad F(3, 47) = 14.431 \quad \hat{\sigma} = 0.038856$$

(errori standard tra parentesi)

La differenza tra i comandi "Copia" e "Salva" è di due tipi. Innanzitutto, "Copia" copia il sorgente TEX negli appunti, mentre "Salva" richiede il nome di un file in cui verrà salvato il sor-

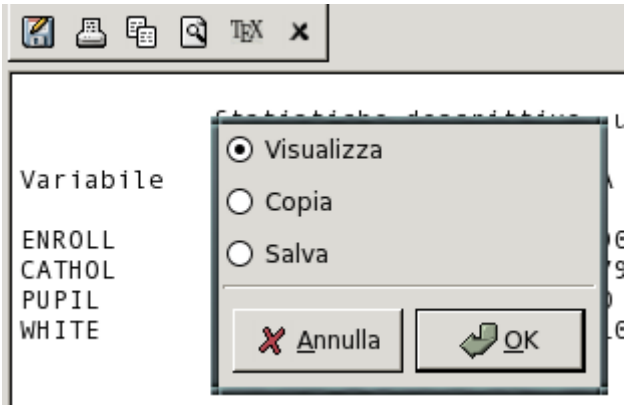
gente. In secondo luogo, con "Copia" il materiale viene copiato come "frammento" di file TEX, mentre con "Salva" viene salvato come file completo. La differenza si spiega tenendo conto che un file sorgente TEX completo comprende un preambolo che contiene comandi come `\documentclass`, che definisce il tipo di documento (ad esempio articolo, rapporto, libro, ecc.) e come `\begin{document}` e `\end{document}`, che delimitano l'inizio e la fine del documento. Questo materiale aggiuntivo viene incluso quando si sceglie "Salva", ma non quando si sceglie "Copia", visto che in questo caso si presume che l'utente incollerà i dati in un file sorgente TEX già provvisto del preambolo richiesto.

I comandi del menù "Opzioni equazione" sono chiari: determinano se, quando si stampa il modello sotto forma di equazione, verranno mostrati gli errori standard o i rapporti t tra parentesi sotto le stime dei parametri. L'impostazione predefinita è quella di mostrare gli errori standard.

Altre finestre

Altre finestre usate per la visualizzazione dei risultati contengono dei comandi per visualizzare, copiare e salvare in formato TEX. Nel caso di finestre che contengono una barra degli strumenti grafica, basta fare clic sul pulsante TEX; la Figura 23.2 mostra questa icona (la penultima della barra degli strumenti), insieme alla finestra di dialogo che appare quando si preme il pulsante.

Figura 23.2: L'icona TEX e la sua finestra di dialogo



Un aspetto del supporto TEX di gretl che può risultare utile per le pubblicazioni scientifiche è la possibilità di esportare la "tabella modelli" (si veda la sezione 3.4). Un esempio è mostrato nella Tabella 23.2.

23.3 Personalizzare l'output di TEX

È possibile operare su tre aspetti: modificare l'aspetto dell'output prodotto da gretl nella modalità di anteprima LATEX, modificare la formattazione delle tabelle dei modelli create con il comando `tabprint`, e incorporare l'output di gretl nei propri file TEX.

La modalità anteprima

Per quanto riguarda l'*anteprima* dei documenti LATEX, è possibile controllare l'aspetto dell'output di gretl usando un file chiamato `gretlpre.tex`, che deve trovarsi nella propria directory utente di gretl (si veda la *Guida ai comandi di gretl*). Il contenuto di questo file verrà usato come "preambolo" per il sorgente TEX, e il suo valore predefinito è il seguente:

```
\documentclass[11pt]{article}
\usepackage[latin1]{inputenc}
\usepackage{amsmath}
\usepackage{dcolumn,longtable}
```

Tabella 23.2: La tabella modelli in formato TEX

Stime OLS
Variabile dipendente: ENROLL

	Modello 1	Modello 2	Modello 3
const	0.2907**	0.2411**	0.08557
	(0.07853)	(0.06602)	(0.05794)
CATHOL	0.2216**	0.2235**	0.2065**
	(0.04584)	(0.04597)	(0.05160)
PUPIL	−0.003035	−0.003382	−0.001697
	(0.002727)	(0.002720)	(0.003025)
WHITE	−0.1482**	−0.1526**	
	(0.04074)	(0.04071)	
ADMEXP	−0.1551		
	(0.1342)		
n	51	51	51
$\bar{R}^2$	0.4502	0.4462	0.2956
ℓ	96.09	95.36	88.69

Errori standard tra parentesi
* indica significatività al livello del 10 per cento
** indica significatività al livello del 5 per cento

```
\begin{document}
\thispagestyle{empty}
```

Si noti che sono richiesti i pacchetti `amsmath` e `dcolumn` (per alcuni tipi di risultati è richiesto anche il pacchetto `longtable`). Modificando questo file, è possibile, ad esempio, cambiare il carattere usato, o la dichiarazione `documentclass`, oppure ancora includere un pacchetto alternativo.

La riga `\usepackage[latin1]{inputenc}` viene automaticamente modificata se gretl si trova ad essere eseguito su un sistema che usa la codifica caratteri UTF-8, si veda la sezione 23.4 seguente.

Inoltre, se occorre comporre l'output di gretl in più di una lingua, è possibile preparare file specializzati per ognuna delle lingue da usare, chiamandoli con il nome `gretlpre_xx.tex`, dove `xx` sta per le prime due lettere della variabile di ambiente `LANG`, che indica la lingua usata dal sistema. Ad esempio, se si sta utilizzando il programma in lingua polacca, usando `LANG=pl_PL`, gretl compirà le operazioni seguenti per formare il preambolo di un file sorgente TEX:

1. Cercare un file chiamato `gretlpre_pl.tex` nella directory utente di gretl.
2. Se esso non viene trovato, cercare un file chiamato `gretlpre.tex`.
3. Se esso non viene trovato, usare il preambolo predefinito visto sopra.

Al contrario, supponendo di utilizzare di solito gretl in una lingua diversa dall'inglese ma di voler talvota creare dei documenti in inglese, basta creare un file `gretlpre_en.tex`, che verrà usato per comporre il preambolo quando gretl è eseguito con la localizzazione inglese (ossia, ad esempio con `LANG=en_US`).

Opzioni a riga di comando

Dopo aver stimato un modello in uno script o interattivamente (usando il terminale di gretl o la versione a riga di comando del programma gretlcli) è possibile usare i comandi `tabprint` o `eqnprint` per stampare il modello rispettivamente in formato tabulare o sotto forma di equazione. Questi comandi sono presentati nella *Guida ai comandi di gretl*.

È possibile modificare la visualizzazione del formato tabulare usato da gretl specificando un formato di riga personalizzato con l'opzione `--format` del comando `tabprint`. La stringa di formato deve essere racchiusa tra virgolette doppie e preceduta da un segno di uguale, senza spazi. La composizione della stringa di formato è la seguente: ci sono quattro campi, che rappresentano il coefficiente, l'errore standard, il rapporto t e il p-value. Questi campi vanno separati usando barre verticali e possono contenere una specificazione di formato per valori numerici nello stile della funzione `printf`, oppure possono essere lasciati in bianco, in modo da sopprimere la visualizzazione del campo nella rispettiva colonna dela tabella (con l'unico vincolo che non è possibile lasciare in bianco tutti i campi). Ecco alcuni esempi:

```
--format="%.4f|%.4f|%.4f|%.4f"
--format="%.4f|%.4f|%.3f|"
--format="%.5f|%.4f||%.4f"
--format="%.8g|%.8g||%.4f"
```

La prima specificazione stampa i valori di tutte le colonne usando 4 cifre decimali. La seconda sopprime il p-value e mostra il rapporto t con 3 cifre decimali. La terza omette il rapporto t, mentre l'ultima omette il rapporto t e mostra sia il coefficiente che l'errore standard con 8 cifre significative.

Una volta che si imposta un formato in questo modo, esso viene ricordato e usato per tutta la sessione di lavoro. Per tornare ad usare il formato predefinito, basta usare la parola chiave `--format=default`.

Modifiche ulteriori

Dopo aver incollato l'output TEX di gretl nel proprio documento, o dopo averlo salvato in un file ed aperto nel proprio editor, è naturalmente possibile modificarlo a proprio piacimento. Di solito il codice TEX generato dai programmi automaticamente è difficile da comprendere, ma quello generato da gretl è fatto in modo da essere il più possibile comprensibile e facile da modificare. Inoltre, non utilizza alcun pacchetto che non sia compreso nelle distribuzioni standard di TEX: come detto sopra, a parte le classi di documento standard di LATEX, gli unici pacchetti richiesti sono `amsmath`, `dcolumn` e `longtable`, che sono compresi praticamente in qualsiasi distribuzione TEX abbastanza completa.

23.4 Codifica dei caratteri

Se si utilizza gretl su un sistema configurato per la lingua inglese non si dovrebbero incontrare problemi di codifica dei caratteri, ma se occorre generare documenti TEX che contengono caratteri accentati (o comunque non appartenenti alla codifica ASCII) possono essere utili le indicazioni che seguono.

Gretl genera l'output TEX usando la codifica di caratteri impostata sul sistema locale. Se si usa una codifica appartenente alla famiglia ISO-8859 non dovrebbero esserci problemi, ma se si usa la codifica Unicode UTF-8 (impostazione predefinita su alcuni sistemi GNU/Linux recenti) occorre assicurarsi che il proprio sistema TEX possa gestire l'input UTF-8, ossia installare il pacchetto latex-ucs, che potrebbe non essere stato installato automaticamente al momento dell'installazione di TEX.

Se gretl viene eseguito in un ambiente che usa la codifica UTF-8, vengono usate le seguenti righe nel preambolo dei file TEX:

```
\usepackage{ucs}
\usepackage[utf8x]{inputenc}
```

23.5 Installazione e utilizzo di TEX

Ovviamente in questa sede non è possibile fornire un'esposizione dettagliata di questi argomenti, ma solo alcuni riferimenti utili.

Praticamente ogni distribuzione GNU/Linux comprende una serie di pacchetti per TEX, che in alcuni casi sono anche installati per scelta predefinita. Si raccomanda di controllare la documentazione della propria distribuzione. Per quanto riguarda MS Windows, esistono varie versioni di TEX: una delle più famose è MiKTEX disponibile su http://www.miktex.org/. Per Mac OS X, una buona implementazione è iTEXMac, su http://itexmac.sourceforge.net/. Un punto di partenza essenziale per le risorse TEX disponibili online è costituito dal Comprehensive TEX Archive Network (CTAN) su http://www.ctan.org/.

Per imparare TEX, esistono molte guide utili, sia online che cartacee; in lingua italiana segnaliamo il sito del Gruppo Utenti Italiani di TeX e LaTeX: http://www.guit.sssup.it/.

Un'eccellente fonte di livello avanzato è *The LATEX Companion* (Goossens *et al.*, 2004).

Capitolo 24

Risoluzione dei problemi

24.1 Segnalazione dei bug

Le segnalazioni dei bug sono benvenute. È difficile trovare errori di calcolo in gretl (ma questa affermazione non costituisce alcuna sorta di garanzia), ma è possibile imbattersi in bug o stranezze nel comportamento dell'interfaccia grafica. Si tenga presente che l'utilità delle segnalazioni aumenta quanto più si è precisi nella descrizione: cosa *esattamente* non funziona, in che condizioni, con quale sistema operativo? Se si ricevono messaggi di errore, cosa dicono esattamente?

24.2 Programmi ausiliari

Come detto in precedenza, gretl richiama alcuni altri programmi per eseguire alcune operazioni (gnuplot per i grafici, LaTeX per la stampa ad alta qualità dei risultati delle regressioni, GNU R). Se succede qualche problema durante questi collegamenti esterni, non è sempre facile per gretl fornire un messaggio di errore abbastanza informativo. Se il problema si verifica durante l'uso di gretl con l'interfaccia grafica, può essere utile avviare gretl da un terminale invece che da un menù o da un'icona del desktop. Se si usa il sistema X window, è sufficiente avviare gretl dal prompt dei comandi di un `xterm`, mentre se si usa MS Windows occorre digitare `gretlw32.exe` da una finestra di terminale, o "Prompt di MS-DOS", usando le opzioni `-g` o `--debug`. Il terminale conterrà quindi messaggi di errore aggiuntivi.

Si tenga anche presente che nella maggior parte dei casi gretl assume che i programmi in questione siano disponibili nel "percorso di esecuzione" (path) dell'utente, ossia che possano essere invocati semplicemente con il nome del programma, senza indicare il percorso completo[1]. Quindi se un certo programma non si avvia, conviene provare ad eseguirlo da un prompt dei comandi, come descritto di seguito.

	Grafica	*Stampa*	*GNU R*
Sistema X window	gnuplot	latex, xdvi	R
MS Windows	wgnuplot.exe	pdflatex	RGui.exe

Se il programma non si avvia dal prompt, non è un problema di gretl: probabilmente la directory principale del programma non è contenuta nel path dell'utente, oppure il programma non è stato installato correttamente. Per i dettagli su come modificare il path dell'utente, si veda la documentazione o l'aiuto online per il proprio sistema operativo.

[1] L'eccezione a questa regola è costituita dall'invocazione di gnuplot in MS Windows, dove occorre indicare il percorso completo del programma.

Capitolo 25

L'interfaccia a riga di comando

25.1 Gretl sul terminale

Il pacchetto gretl include il programma a riga di comando gretlcli. Su Linux è possibile eseguirlo in una finestra di terminale (xterm, rxvt, o simili), o in un terminale testuale. Su MS Windows può essere eseguito in una finestra di terminale (quella che di solito viene chiamata "prompt di MS-DOS"). gretlcli ha un file di aiuto, a cui si accede inserendo il comando "help" al prompt. Inoltre, può essere usato in modalità batch, inviando i risultati direttamente a un file (si veda la *Guida ai comandi di gretl*).

Se gretlcli è linkato alla libreria readline (ciò avviene automaticamente nella versione MS Windows, si veda anche l'appendice B), è possibile richiamare e modificare i comandi già eseguiti, inoltre è disponibile il completamento automatico dei comandi. Per muoversi nella lista dei comandi già eseguiti è possibile usare i tasti Freccia Su/Giù, mentre per muoversi sulla riga di comando è possibile usare i tasti Freccia a destra/sinistra e le scorciatoie in stile Emacs[1]. Le scorciatoie più usate sono:

Scorciatoia	*Effetto*
`Ctrl-a`	Va all'inizio della riga
`Ctrl-e`	Va alla fine della riga
`Ctrl-d`	Cancella il carattere a destra

dove "`Ctrl-a`" significa premere il tasto "`a`" mentre si tiene premuto il tasto "`Ctrl`". Quindi se si vuole correggere qualcosa all'inizio della riga di comando, *non occorre* usare il tasto di cancellazione all'indietro su tutta la riga, basta saltare all'inizio della riga e aggiungere o cancellare i caratteri desiderati. Inserendo le prime lettere del nome di un comando e premendo il tasto Tab, readline cercherà di completare automaticamente il nome del comando. Se esiste un solo modo di completare le lettere inserite, verrà inserito il comando corrispondente, altrimenti premendo Tab una seconda volta verrà visualizzata una lista delle alternative possibili.

25.2 La modalità non interattiva

Probabilmente, il modo più utile per svolgere grosse quantità di lavoro con gretlcli consiste nell'utilizzare la modalità "batch" (non interattiva), in cui il programma legge ed esegue uno script di comandi, scrivendo il risultato su un file. Ad esempio:

```
gretlcli -b filescript > fileoutput
```

Il *filescript* fornito come argomento al programma deve specificare al proprio interno il file di dati su cui operare, usando il comando `open filedati`. Se non si utilizza l'opzione `-b` (che sta per "batch"), il programma resterà in attesa di input al termine dell'esecuzione dello script.

[1] Questi sono i valori predefiniti per le scorciatoie da tastiera, che possono essere modificate seguendo le istruzioni contenute nel manuale di readline.

Parte IV

Appendici

Appendice A

Dettagli sui file di dati

A.1 Formato interno di gretl

Il formato dati usato internamente da Gretl per i dataset è basato su XML (extensible mark-up language). I file di dati sono conformi al semplice DTD (document type definition) contenuto nel file `gretldata.dtd`, fornito con la distribuzione di Gretl e installato nella directory dei dati di sistema (ad es. `/usr/share/gretl/data` su Linux). I file di dati possono essere non compressi o compressi con gzip; oltre ai valori dei dati, essi contengono altre informazioni aggiuntive, come i nomi e le descrizioni delle variabili, la frequenza dei dati e così via.

La maggior parte degli utenti probabilmente non avrà bisogno di leggere o scrivere questi file, se non usando Gretl stesso, ma se si vuole manipolarli usando altri strumenti, può essere utile esaminare il DTD e qualcuno dei file di esempio forniti: il file `data4-1.gdt` dà un semplice esempio, il file `data4-10.gdt` invece contiene anche delle etichette per le osservazioni.

A.2 Formato tradizionale di ESL

Per compatibilità all'indietro, Gretl può elaborare anche file di dati nel formato "tradizionale" usato dal programma ESL di Ramanathan. In questo formato (che era quello predefinito nelle versioni di Gretl precedenti alla 0.98) un dataset è rappresentato da due file: uno contiene i dati veri e propri, l'altro contiene la descrizione dei dati e le modalità per la loro lettura. In particolare:

1. *Dati veri e propri*: una matrice rettangolare di numeri separati da spazi vuoti. Ogni colonna rappresenta una variabile, ogni riga un'osservazione per ognuna delle variabili (come in un foglio elettronico). Le colonne dei dati possono essere separate da spazi o caratteri tab. Il nome del file deve avere l'estensione `.gdt` e il file di solito è di tipo ASCII (testo semplice), ma può essere anche compresso con gzip, per occupare meno spazio su disco. È possibile inserire commenti in un file di dati: se una riga comincia con un carattere cancelletto (#), l'intera riga viene ignorata (questo comportamento è coerente con i file di dati di gnuplot e di octave).
2. *Descrizione*: il file di dati deve essere accompagnato da un file di descrizioni, che ha lo stesso nome di base del file di dati, ma l'estensione `.hdr`. Questo file contiene, nell'ordine:
   - (Opzionale) *commenti* sui dati, introdotti dalla stringa di apertura `(*` e conclusi dalla stringa `*)`; ognuna di queste stringhe deve essere presente da sola su una riga.
   - (Richiesta) lista dei *nomi delle variabili* presenti nel file, separati da spazio bianco. I nomi sono limitati a 8 caratteri, devono iniziare con una lettera e possono contenere solo caratteri alfanumerici e il carattere trattino basso, _. La lista può continuare su più righe e deve essere chiusa da un punto e virgola `;`.
   - (Richiesta) riga *osservazioni*, nel formato `1 1 85`. Il primo elemento indica la frequenza dei dati (1 per dati non datati o annuali, 4 per trimestrali, 12 per mensili). Gli altri due elementi indicano le osservazioni iniziale e finale: di solito questi saranno pari a 1 e al numero delle osservazioni, per i dati non datati. Per le serie storiche, è possibile usare date nella forma `1959.1` (trimestrale, una cifra dopo il punto) o `1967.03` (mensile, due cifre dopo il punto). Si veda il capitolo 15 per l'uso speciale di questa riga nel caso dei dati panel.
   - La parola chiave `BYOBS`.

Ecco un esempio di un file di descrizione dei dati ben scritto.

```
(*
  DATA9-6:
  Dati su log(moneta), log(reddito) e tasso di interesse USA.
  Fonte: Stock and Watson (1993) Econometrica
  (dati grezzi) Il periodo è 1900-1989 (dati annuali).
  Dati composti da Graham Elliott.
*)
lmoneta lreddito tassint ;
1 1900 1989 BYOBS
```

Il corrispondente file di dati contiene tre colonne di dati, ognuna con 90 osservazioni. Ci sono altre tre caratteristiche del formato dati "tradizionale" degne di nota.

1. Se la parola chiave BYOBS è sostituita da BYVAR ed è seguita dalla parola chiave BINARY, significa che il corrispondente file di dati è in formato binario. Questo tipo di file di dati può essere scritto da gretlcli usando il comando `store` con l'opzione `-s` (precisione singola) o l'opzione `-o` (precisione doppia).
2. Se BYOBS è seguita dalla parola chiave MARKERS, Gretl si aspetta un file di dati in cui la *prima colonna* contiene stringhe (lunghe al massimo 8 caratteri) usate per identificare le osservazioni. Queste possono essere utili nel caso dei dati di tipo cross-section in cui le unità di osservazione sono identificabili: regioni, stati, città ecc. Un altro caso è quello delle serie storiche irregolari, come ad esempio i dati giornalieri delle quotazioni azionarie, in cui mancano i giorni in cui non avvengono contrattazioni: in questo caso, le osservazioni possono essere marcate con una stringa che rappresenta la data, come ad esempio 10/01/98 (si ricordi il limite di 8 caratteri). Si noti che le opzioni BINARY e MARKERS sono mutualmente esclusive, e si noti anche che i "marcatori" non sono considerati come una variabile: questa colonna non compare nell'elenco delle variabili nel file di descrizione dei dati.
3. Se esiste un file con lo stesso nome di base del file di dati e di quello delle descrizioni, ma con l'estensione `.lbl`, esso viene letto e usato per riempire le etichette descrittive per le serie di dati. Il formato del file di etichette è semplice: ogni riga contiene il nome di una variabile (indicata nel file di descrizione) seguito da uno o più spazi, seguito dall'etichetta descrittiva. Ecco un esempio: `prezzo Indice dei prezzi auto nuove, anno base 1982`

Se si vuole salvare i dati nel formato tradizionale, occorre usare l'opzione `-t` con il comando `store`, nella versione a riga di comando del programma o nella finestra del terminale della versione grafica del programma.

A.3 Dettagli sui database binari

Un database di Gretl consiste di due parti: un file indice ASCII (con l'estensione `.idx`) che contiene informazioni sulle serie, e un file binario (con estensione `.bin`) che contiene i dati veri e propri. Ecco due esempi di voci contenute in un file `idx`:

```
GOM910  Indice composto da 11 indicatori principali (1987=100)
M 1948.01 - 1995.11  n = 575
currbal Bilancia dei pagamenti: parte corrente; corretta per la stagionalità
Q 1960.1 - 1999.4 n = 160
```

Il primo campo è il nome della serie, il secondo è una descrizione della serie (al massimo di 128 caratteri). Sulla seconda riga, il primo campo è un codice di frequenza: `M` per dati mensili, `Q` per dati trimestrali, `A` per dati annuali, `B` per dati giornalieri-lavorativi (giornalieri con cinque giorni a settimana) e `D` per dati giornalieri (sette giorni a settimana). Al momento non vengono accettati altri codici di frequenza. Quindi segue la data iniziale (con due cifre che seguono il punto nei dati mensili, una per i dati trimestrali, nessuna per quelli annuali), uno spazio, un

trattino, un altro spazio, la data finale, la stringa "`n = `" e il numero delle osservazioni. Nel caso di dati giornalieri, le date iniziale e finale devono essere indicate nella forma `YYYY/MM/DD`. Questo formato deve essere rispettato alla lettera.

Opzionalmente, la prima riga del file indice può contenere un breve commento (al massimo 64 caratteri) a proposito della fonte e del tipo dei dati, che segue un carattere cancelletto. Ad esempio:

```
# Banca Centrale Europea (tassi di interesse)
```

Il corrispondente database binario contiene i valori dei dati, rappresentati come "float", ossia come numeri in virgola mobile a precisione singola, che tipicamente occupano quattro byte ciascuno. I numeri sono immagazzinati "per variabile", così che i primi *n* numeri sono le osservazioni della variabile 1, i successivi *m* sono le osservazioni per la variabile 2 e così via.

Appendice B

Compilare gretl

B.1 Requisiti

Gretl è scritto nel linguaggio di programmazione C, aderendo nel modo più stretto possibile allo standard C ISO/ANSI (C90), anche se l'interfaccia grafica e alcune altre componenti devono fare uso necessariamente di estensioni specifiche per certe piattaforme.

Il programma è sviluppato in ambiente Linux. La libreria condivisa e il client a riga di comando dovrebbero essere compilabili su qualsiasi piattaforma che supporti l'ISO/ANSI C e abbia installate le librerie elencate nella Tabella B.1. Se il sistema dispone anche della libreria GNU readline, essa sarà usata per gretcli, fornendo una riga di comando molto più comoda da utilizzare. Si veda la home page di readline.

Libreria	*Funzione*	*Sito web*
zlib	compressione dei dati	info-zip.org
libxml2	manipolazione XML	xmlsoft.org
LAPACK	algebra lineare	netlib.org
FFTW3	Trasformata veloce di Fourier	fftw.org
glib-2.0	Varie utilità	gtk.org

Tabella B.1: Librerie richieste per compilare gretl

Il client grafico dovrebbe essere compilabile e utilizzabile su ogni sistema che, oltre ai requisiti visti sopra, offra la libreria GTK nella versione 2.4.0 o superiore (si veda gtk.org)[1].

Gretl usa gnuplot per produrre grafici. È possibile trovare gnuplot a gnuplot.info. Al momento della scrittura di questo manuale, la versione ufficiale più recente di gnuplot è la 4.2 (di marzo 2007). La versione MS Windows di Gretl comprende la versione 4.2 di gnuplot per Windows; sul sito web di Gretl è possibile trovare un pacchetto rpm di gnuplot 3.8j0 per sistemi Linux x86.

Alcune funzionalità di Gretl fanno uso di parti della libreria gtkextra di Adrian Feguin. Le parti rilevanti di questo pacchetto sono incluse (in forma leggermente modificata) con la distribuzione sorgente di gretl.

Una versione binaria del programma è disponibile per la piattaforma Microsoft Windows (Windows 98 o superiore). Questa versione è cross-compilata su Linux usando mingw (il compilatore GNU C, gcc, portato per l'uso con win32) e collegata alla libreria C di Microsoft C, `msvcrt.dll`. Utilizza il port di GTK 2.0 su win32 di Tor Lillqvist. L'installatore per Windows (libero, open-source) è a cura di Jordan Russell (jrsoftware.org).

Speriamo che gli utenti con conoscenze di programmazione possano considerare Gretl abbastanza interessante e degno di essere migliorato ed esteso. La documentazione dell'API libgretl non è ancora completa, ma è possibile trovare alcuni dettagli seguendo il link "Libgretl API docs" sul sito web di Gretl. Chiunque sia interessato allo sviluppo di gretl è invitato a iscriversi alla mailing list gretl-devel.

[1] Fino alla versione 1.5.1, gretl poteva essere compilato anche usando GTK 1.2, ma il supporto per questa libreria è stato abbandonato a partire dalla versione 1.6.0 di gretl.

B.2 Istruzioni per la compilazione

Questa sezione contiene istruzioni che permettono di compilare gretl a un utente con una conoscenza di base di un sistema di tipo Unix. La procedura è stata testata su una nuova installazione della distribuzione Debian Etch e dovrebbe funzionare anche su altre distribuzioni Linux (specialmente quelle derivate da Debian, come Ubuntu e simili). Su altri sistemi simili a Unix, come MacOSX e BSD, potrebbero esserci differenze maggiori nella procedura.

In questo esempio guidato, compileremo gretl insieme alla sua documentazione completa. Questa scelta implica alcuni requisiti in più, ma in cambio permette di modificare anche i file della documentazione, come l'aiuto in linea o i manuali.

Installare i programmi di base

Assumiamo che le utilità GNU di base siano già installate sul sistema, insieme a questi altri programmi:

- Un sistema TEX/LATEX(`tetex` o `texlive` vanno benissimo)
- Gnuplot
- ImageMagick

Assumiamo anche che l'utente abbia i privilegi di amministratore e sappia come installare i pacchetti. Gli esempi seguenti mostrano l'uso del comando shell `apt-get`, ma possono essere replicati anche con comandi come `aptitude`, `dselect`, con il programma a interfaccia grafica `synaptic`. Gli utenti di distribuzioni Linux che usano pacchetti rpm (es. Red Hat/Fedora, Mandriva, SuSE) possono consultare la pagina dipendenze sul sito di gretl.

Il primo passo consiste nell'installare il compilatore C e le utilità correlate. Su un sistema Debian, occorre installare i seguenti pacchetti:

```
apt-get install gcc autoconf automake1.9 libtool flex bison gcc-doc \
libc6-dev libc-dev libgfortran1 libgfortran1-dev gettext pkgconfig
```

Quindi occorre installare i pacchetti "di sviluppo" (dev) per le librerie usate da gretl:

Libreria	*Comando*
GLIB	`apt-get install libglib2.0-dev`
GTK 2.0	`apt-get install libgtk2.0-dev`
PNG	`apt-get install libpng12-dev`
XSLT	`apt-get install libxslt1-dev`
LAPACK	`apt-get install lapack3-dev`
FFTW	`apt-get install fftw3-dev`
READLINE	`apt-get install libreadline5-dev`
GMP	`apt-get install libgmp3-dev`

GMP è opzionale, ma raccomandata. I pacchetti dev di queste librerie sono necessari per *compilare* gretl, mentre per *eseguire* il programma occorrono anche i pacchetti normali (non-dev) delle librerie; la maggior parte di questi pacchetti fa già parte delle installazioni standard. Per abilitare funzionalità aggiuntive, come il supporto audio, occorre installare altre librerie.

Procurarsi il sorgente: pacchetto o CVS

A questo punto è possibile iniziare a compilare il sorgente. È possibile procurarselo in due modi: scaricando l'ultimo pacchetto sorgente rilasciato, o scaricando la versione di gretl contenuta attualmente nell'archivio CVS (Concurrent Versions System). La versione CVS è di solito più recente, può contenere correzioni di bug presenti nell'ultimo pacchetto rilasciato, ma può anche

contenere nuovi bug (o addirittura non essere compilabile) e codice "sperimentale". Se si vuole partecipare allo sviluppo e al test del programma, è raccomandabile usare la versione CVS di gretl.

Per lavorare col pacchetto sorgente:

1. Scaricare il pacchetto dei sorgenti di Gretl da gretl.sourceforge.net.
2. Decomprimere il pacchetto. Se si dispone delle utilità GNU, usare il comando `tar xvfz gretl-N.tar.gz` (sostituire `N` con il numero di versione specifico del file scaricato).
3. Spostarsi nella directory del codice sorgente di gretl appena creata (ad es. `gretl-1.6.6`).
4. Procedere alla sezione seguente "Configurare il sorgente".

Per lavorare con CVS occorre per prima cosa installare il programma client `cvs`, se non si trova già sul proprio sistema. Può essere utile consultare il sito web di CVS www.nongnu.org/cvs, le istruzioni per l'uso di CVS su SourceForge contenute nella pagina CVS di SourceForge, e le istruzioni specifiche su gretl nella pagina CVS di Gretl.

Quando si scarica il codice da CVS *per la prima volta*, occorre decidere dove salvarlo. Ad esempio si può creare una directory chiamata `cvs` nella propria home directory, aprire un terminale, fare `cd` in questa directory ed eseguire i comandi seguenti:

```
cvs -d:pserver:anonymous@gretl.cvs.sourceforge.net:/cvsroot/gretl login
cvs -z3 -d:pserver:anonymous@gretl.cvs.sourceforge.net:/cvsroot/gretl co -P gretl
```

Dopo il primo comando verrà richiesta una password: basta premere invio. Dopo il secondo comando, `cvs` creerà una sotto-directory chiamatat `gretl` che conterrà il codice sorgente.

Ogni volta che si vuole *aggiornare la propria copia del sorgente*, basta andare nella directory `gretl` ed eseguire

```
cvs update -d -P
```

Da questo punto in poi, assumendo di essere nella directory `gretl`, è possibile seguire le istruzioni valide anche per il sorgente contenuto nel pacchetto.

Configurare il sorgente

Il successivo comando da eseguire è `./configure`; si tratta di uno script sofisticato che cerca gli strumenti disponibili sul sistema e prepara la compilazione. Il comando `configure` accetta molte opzioni: eseguendo

```
./configure --help
```

si ottiene l'elenco di quelle disponibili. Un'opzione che può essere utile usare è `--prefix`, che modifica il percorso principale in cui verrà installato il programma; il valore predefinito è `/usr/local`. Ad esempio usando

```
./configure --prefix=/usr
```

i file verranno installati sotto il percorso `/usr`. Un'altra opzione ha a che fare col supporto per il desktop gnome, che gretl offre in modalità predefinita. Se non si è interessati alle funzionalità specifiche di gnome è possibile usare l'opzione `--without-gnome` di `configure`.

Per compilare anche la documentazione, c'è un'altra opzione di `configure`:

```
./configure --enable-build-doc
```

Esempio B.1: Risultato di `./configure --enable-build-doc`

```
Configuration:

  Installation path:                    /usr/local
  Use readline library:                 yes
  Use gnuplot for graphs:               yes
  Use PNG for gnuplot graphs:           yes
  Use LaTeX for typesetting output:     yes
  Gnu Multiple Precision support:       yes
  MPFR support:                         no
  LAPACK support:                       yes
  FFTW3 support:                        yes
  Build with GTK version:               2.0
  Script syntax highlighting:           yes
  Use installed gtksourceview:          yes
  Build with gnome support:             no
  Build gretl documentation:            yes
  Build message catalogs:               yes
  Gnome installation prefix:            NA
  X-12-ARIMA support:                   yes
  TRAMO/SEATS support:                  yes
  Experimental audio support:           no

Now type 'make' to build gretl.
```

Quando si esegue il comando, si vedranno i risultati di una serie di controlli, e se tutto funziona correttamente, un riassunto finale simile a quello mostrato nell'Esempio B.1.

☞ Se si usa CVS, è una buona idea ri-eseguire lo script `configure` dopo ogni operazione di aggiornamento del sorgente. Non è sempre necessario, ma alcune volte lo è; in ogni caso non è dannoso. A questo proposito, può essere comodo creare un piccolo script di shell che esegue `configure` con le opzioni che si è soliti utilizzare.

Compilare e installare

Siamo quindi pronti per iniziare la vera e propria compilazione: questa si ottiene col comando `make`, che compila tutti i file sorgenti nell'ordine corretto. Basta semplicemente eseguire

```
make
```

Questo passo della procedura richiederà qualche minuto e produrrà molti messaggi sullo schermo. Alla fine, per installare la copia di gretl appena prodotta, basterà eseguire:

```
make install
```

Sulla maggior parte dei sistemi, il comando `make install` richiede di avere i privilegi di amministratore di sistema. Quindi occorrerà fare login come `root` prima di eseguirlo, oppure occorrerà usare il comando `sudo`:

```
sudo make install
```

Appendice C

Accuratezza numerica

Gretl usa aritmetica a doppia precisione, ad eccezione del plugin per precisione multipla invocabile con il comando del menù "Modello, Altri modelli lineari, Minimi quadrati in alta precisione" che rappresenta i valori in virgola mobile usando un numero di bit indicato dalla variabile di ambiente GRETL_MP_BITS (valore predefinito: 256).

Le equazioni normali dei minimi quadrati sono risolte in modo predefinito usando la decomposizione di Cholesky, che è abbastanza accurata per la maggior parte delle applicazioni. Nel caso le variabili indipendenti esibiscano un'alto grado di collinearità, gretl adotta automaticamente la decomposizione QR; inoltre l'utente può scegliere di usare sempre la decomposizione QR.

Il programma è stato testato abbastanza approfonditamente con i dataset di riferimento forniti dal NIST (il National Institute of Standards and Technology statunitense) e un rapporto completo dei risultati si trova sul sito web di Gretl (seguendo il link "Accuratezza numerica").

Ad oggi, sono state pubblicate due recensioni che hanno preso in esame l'accuratezza di gretl: Giovanni Baiocchi e Walter Distaso (2003), e Talha Yalta e Yasemin Yalta (2007). Siamo grati a questi autori per il loro accurato esame del programma. I loro commenti hanno suggerito vari miglioramenti, come l'uso del codice cephes di Stephen Moshier per calcolare i p-value e altre quantità relative alle distribuzioni di probabilità (si veda netlib.org), modifiche alla formattazione dei risultati delle regressioni per assicurare che il programma mostri un numero coerente di cifre significative, soluzione di alcuni problemi di compilazione della versione MS Windows di gretl (che in passato era leggermente meno accurata della versione Linux).

La versione attuale di Gretl comprende un "plugin" che esegue la suite NIST di test della regressione lineare. È possibile trovarlo nel menù "Strumenti" della finestra principale. Quando viene eseguito questo test, un'introduzione spiega qual è il risultato atteso: se si esegue il test e si ottengono risultati diversi da quelli attesi, si prega di inviare un bug report a cottrell@wfu.edu.

Tutte le statistiche di regressione sono mostrate con 6 cifre significative nella versione attuale di Gretl (tranne quando viene usato il plugin per precisione multipla, e allora i risultati sono mostrati con 12 cifre). Se occorre esaminare un valore più da vicino, basta per prima cosa salvarlo (si veda il comando genr) e poi visualizzarlo usando il comando printf.

Appendice D

Altro software libero utile

Le possibilità offerte da gretl sono molte e in continua espansione, tuttavia può esserci l'esigenza di compiere operazioni non disponibili in gretl, oppure di confrontare i risultati con quelli di altri programmi. Se si è interessati a funzionalità complementari disponibili nell'ambito del software libero, o open source, raccomandiamo i programmi seguenti, di cui riportiamo la descrizione fornita nei rispettivi siti web.

- **GNU R** r-project.org: "R è un sistema per il calcolo statistico e i grafici. Consiste in un linguaggio e in un ambiente di lavoro che fornisce grafici, un debugger, l'accesso alle funzioni del sistema e la possibilità di eseguire programmi contenuti all'interno di script... È compilabile ed eseguibile su un'ampia varietà di piattaforme UNIX, Windows e MacOS". Commento: esistono molti pacchetti aggiuntivi per R, che coprono le principali aree dell'analisi statistica.
- **GNU Octave** www.octave.org: "GNU Octave è un linguaggio ad alto livello, dedicato principalmente al calcolo numerico. Fornisce una comoda interfaccia a riga di comando per risolvere numericamente problemi lineari e non lineari, e per eseguire altre operazioni numeriche usando un linguaggio per lo più compatibile con quello di Matlab. Può anche essere utilizzato in modalità non interattiva".
- **JMulTi** www.jmulti.de: "JMulTi è stato progettato in origine come strumento per alcune procedure econometriche tipiche dell'analisi delle serie storiche, di difficile utilizzo e reperibilità in altri pacchetti, come l'analisi impulso-risposta con intervalli di confidenza bootstrap per modelli VAR/VEC. In seguito, sono state aggiunte altre funzionalità che permettono di condurre un'analisi completa". Commento: JMulTi è un programma java con interfaccia grafica; per poterlo usare è necessario installare un java run-time environment.

Come già detto sopra, Gretl offre la possibilità di esportare i dati nei formati di Octave e R. Nel caso di Octave, il dataset di Gretl viene salvato come matrice singola X: una volta importati i dati in Octave, se si vuole, è possibile eliminare la matrice X, si veda il manuale di Octave per i dettagli. Per quanto riguarda R, il file dei dati esportati conserva qualsiasi struttura di serie storiche riconoscibile da gretl. Le serie sono salvate come strutture individuali: i dati vanno importati in R con il comando `source()`.

Inoltre, Gretl dispone di una funzione per trasferire velocemente dati in R. Il menù "Strumenti" di Gretl contiene il comando "Avvia GNU R", che salva il dataset in uso in formato R (nella directory utente di gretl) e lo carica avviando una nuova sessione di R. Il modo esatto in cui R viene avviato dipende dalla variabile interna `Rcommand` di gretl, il cui valore deve essere impostato nel menù "Strumenti, Preferenze". Il comando predefinito è `RGui.exe` in MS Windows, mentre in X è `xterm -e R`. Si noti che questa stringa può contenere al massimo tre elementi separati da spazi: ulteriori elementi saranno ignorati.

Appendice E

Elenco degli URL

Quello che segue è un elenco degli URL citati nel testo.

Estima (RATS) http://www.estima.com/

FFTW3 http://www.fftw.org/

Home page del desktop Gnome http://www.gnome.org/

Libreria GNU Multiple Precision (GMP) http://swox.com/gmp/

Home page di GNU Octave http://www.octave.org/

Home page di GNU R http://www.r-project.org/

Manuale di GNU R http://cran.r-project.org/doc/manuals/R-intro.pdf

Home page di Gnuplot http://www.gnuplot.info/

Manuale di Gnuplot http://ricardo.ecn.wfu.edu/gnuplot.html

Pagina dei dati di Gretl (versione italiana) http://gretl.sourceforge.net/gretl_data_it.html

Home page di Gretl (versione italiana) http://gretl.sourceforge.net/gretl_italiano.html

Home page di GTK+ http://www.gtk.org/

Home page del port di GTK+ per win32 http://www.gimp.org/~tml/gimp/win32/

Home page di InfoZip http://www.info-zip.org/pub/infozip/zlib/

Home page di JMulTi http://www.jmulti.de/

JRSoftware http://www.jrsoftware.org/

Home page di Mingw (gcc per win32) http://www.mingw.org/

Minpack http://www.netlib.org/minpack/

Penn World Table http://pwt.econ.upenn.edu/

Home page di Readline http://cnswww.cns.cwru.edu/~chet/readline/rltop.html

Manuale di Readline http://cnswww.cns.cwru.edu/~chet/readline/readline.html

Home page di Xmlsoft http://xmlsoft.org/