

L_AT_EX e CSV

Massimiliano Dominici

Email mlgdominici@interfree.it
Address Pisa, Italy

Abstract Questo articolo presenta alcune tecniche e alcuni esempi per gestire, all'interno di documenti L_AT_EX, dati organizzati in tabelle di *comma separated values*. In particolare l'attenzione verrà focalizzata su due pacchetti scritti appositamente per facilitare questo compito: `datatool` e `pgfplots`.

1 Introduzione

CSV (*comma separated values*) è un particolare formato di file adatto a rappresentare strutture di dati in forma tabulare. Un file CSV è a tutti gli effetti un file di testo, con alcuni vincoli che stabiliscono l'interpretazione del suo contenuto. Questi vincoli sono i seguenti:

1. I dati sono organizzati in *record* e ogni *record* è suddiviso in *campi* o *colonne*;
2. ogni *record* è delimitato da un'interruzione di riga (l'ultimo *record* può anche esserne privo);
3. la prima riga può contenere un'intestazione, ovvero il nome assegnato a ciascuna colonna;
4. ogni *campo* all'interno di un *record*, o dell'intestazione, deve essere separato da una virgola; il numero di campi deve essere lo stesso per ogni *record* e per l'intestazione; un campo può anche essere vuoto, ovvero non contenere dati;
5. gli spazi, anche iniziali e finali, fanno parte del campo e devono essere preservati;
6. i doppi apici (") funzionano da carattere di *escaping*: se un *campo* contiene virgole, interruzioni di riga o doppi apici, deve essere racchiuso tra doppi apici; i doppi apici contenuti in un *campo* devono sempre essere raddoppiati.

In base a quanto detto sopra, il codice seguente è una corretta rappresentazione del contenuto di un file CSV:

Autore,Titolo,Anno,Casa Editrice
C. E. Gadda,La meccanica,1958,"Garzanti, Milano"

A questo punto, sono però necessarie alcune precisazioni. La struttura di un file CSV non è mai stata formalizzata in uno standard. I vincoli descritti sopra sono quelli riportati in un documento informale dello IETF (International Engineering Task Force) per il tipo MIME `text/csv` [IETF, 2005]. Tuttavia non tutte le applicazioni che gestiscono file CSV implementano lo stesso insieme di specifiche: in particolare quasi tutte disattendono quanto stabilito al punto 5 e scartano automaticamente gli spazi a inizio e fine campo. Se si vogliono preservare questi spazi è quindi bene racchiudere il *campo* tra doppi apici. Inoltre quasi tutte le applicazioni sono in grado di usare (in lettura e scrittura) caratteri diversi per i delimitatori di *campo*. Per esempio "punto e virgola" o "TAB", al posto della virgola.¹

Poiché un CSV (nel senso esteso specificato nella nota 1) è molto adatto a rappresentare dati di qualsiasi tipo in forma tabulare, i campi di applicazione in cui viene usato sono numerosi e di ambito differente. In particolare in questo articolo prenderemo in considerazione la gestione di semplici basi di dati e la manipolazione di tabelle di valori numerici provenienti da registrazioni di dati sperimentali o simulazioni numeriche. I dati organizzati nel CSV potranno essere stati scritti manualmente, o, più verosimilmente, provenire da un'applicazione. In quest'ultimo caso è probabile che si tratti di un gestore di basi di dati (Oracle, MySQL, PostgreSQL, ecc.), di un foglio di calcolo (MS Excel, OpenOffice Calc, ecc.) oppure di un programma per il calcolo numerico o simbolico (Matlab, Octave, Mathematica, Sagemath, ecc.).

1. In questo caso, bisognerebbe piuttosto parlare di "Fielded Text" (<http://www.fieldedtext.org/>). Ma siccome il "Fielded Text" copre casi più generali del CSV, e le varie implementazioni di quest'ultimo esistono ormai da molti anni, si continua a considerare CSV anche un insieme di dati separati da caratteri diversi dalla virgola.

2 Gestire semplici basi di dati

Il formato CSV si presta abbastanza bene a rappresentare semplici basi di dati di uso quotidiano in applicazioni da ufficio, come rubriche di indirizzi o registrazioni di punteggi scolastici. I dati possono essere compilati manualmente, oppure essere estratti da un foglio di calcolo o da una base di dati.

È bene far notare che il formato CSV è un formato “appiattito”, ovvero tutti i dati sono contenuti in un’unica tabella. Chi ha dimestichezza con le basi di dati sa che invece, in genere, la loro struttura è più complessa, organizzata in diverse tabelle coordinate tra loro. Il modello più diffuso è il cosiddetto *Relational Database Management System* (RDBMS), alla base di numerosi prodotti commerciali, *open source* o *free software*. Non è questa la sede adatta per illustrare tale modello; basterà dire che di solito le applicazioni che implementano questo tipo di basi di dati offrono la possibilità di esportare in formato CSV, quindi in un’unica tabella, dati provenienti dalle varie tabelle interne, opportunamente organizzati (per esempio filtrati secondo le indicazioni passate al programma, o ordinati in base a una o più colonne scelte dall’utente).

Per quanto riguarda i fogli di calcolo, invece, i dati sono solitamente già “appiattiti” all’origine e potrà essere necessario, al limite, solo l’ordinamento in base ad una colonna data.

Lo strumento migliore per gestire un CSV di questo tipo all’interno di un documento \LaTeX è il pacchetto `datatool`.

2.1 Generalità su `datatool`

Il pacchetto `datatool` [Talbot, 2009], successore di `csvtools` [Talbot, 2007], è stato disegnato per poter creare, modificare, caricare e gestire delle semplici basi di dati. Per semplice intendiamo sia “dalla struttura semplificata” (con tutti i dati, generalmente, contenuti in un’unica tabella) che di ridotte dimensioni. \TeX infatti non è particolarmente efficiente nel gestire questo tipo di operazione.

Nel seguito dell’articolo ci occuperemo principalmente della gestione di file CSV creati esternamente, quindi non accenneremo, se non di sfuggita, ai comandi che si occupano dei primi due aspetti citati. Esamineremo, invece, nei dettagli quanto attiene agli ultimi due aspetti.

I primi due comandi che l'utente deve conoscere sono quelli relativi al caricamento e alla visualizzazione dei *database*.

```
\DTLloaddb[<opzioni>]
  {<nome_database>}{<nome_file>}
```

carica il *database* contenuto nel file *nome_file* assegnandogli il nome *nome_database*. Se non vengono specificate opzioni il comando assume implicitamente che la prima riga del file CSV contenga un'intestazione e di conseguenza contrassegnerà ogni colonna con la stringa contenuta nel corrispondente campo della prima riga. Le <opzioni> servono appunto a modificare questo comportamento:

noheader segnala che la prima riga non contiene l'intestazione ma valori normali;
keys accetta un insieme di valori, separati da virgole, e li usa per contrassegnare le varie colonne, sovrascrivendo quelle eventualmente presenti nell'intestazione;

headers accetta un insieme di valori, separati da virgole, e li usa per definire l'intestazione del *database* in fase di visualizzazione.

Immaginiamo di avere un ipotetico file *messaggi.csv* in cui sia riportato, mese per mese, il numero di messaggi postati nelle principali categorie del Forum G_{IT} nel periodo novembre 2008/febbraio 2009, ripartito per ogni singola categoria presa in considerazione:

```
TeX e LaTeX,235,212,289,321
ConTeXt,0,0,0,0
Altri programmi,24,18,19,31
Tipografia,13,11,18,22
Corsi e didattica,3,2,4,4
Edizioni critiche,2,0,5,3
```

Come si può vedere, il CSV non ha un'intestazione e, senza opzioni, `\DTLloaddb` interpreterebbe scorrettamente la prima riga. Il seguente comando, risolve il problema:

```
\DTLloaddb[noheader,
  keys={cat,nov,dic,gen,feb},
  headers={Categoria, Novembre 2008,
    Dicembre 2008,Gennaio 2009,
    Febbraio 2009}]{messaggi}{messaggi.csv}
```

Categoria	Novembre 2008	Dicembre 2008	Gennaio 2009	Febbraio 2009
T _E X e L ^A T _E X	235	212	289	321
ConTeXt	0	0	0	0
Altri programmi	24	18	19	31
Tipografia	13	11	18	22
Corsi e didattica	3	2	4	4
Edizioni critiche	2	0	5	3

Figura 1: Visualizzazione automatica del *database* messaggi.

Il file `messaggi.csv` segue lo standard per quanto riguarda l'uso dei separatori di campo, ma, se così non fosse, `datatool` fornisce i comandi `\DTLsetseparator{⟨char⟩}` e `\DTLsettabseparator` per cambiarlo in maniera opportuna.² Può anche rendersi necessario impostare il carattere che delimita i campi (in mancanza di specifiche: `""`); è possibile farlo con il comando `\DTLsetdelimiter{⟨char⟩}`.

Infine, nel caso che il file CSV contenga caratteri che L^AT_EX interpreta in maniera speciale (`$`, `&`, `%`, ecc.), il *database* va caricato con il comando `\DTLloadrawdb`, che ha la stessa sintassi e fa in modo che tali caratteri vengano interpretati correttamente.³

Per visualizzare in maniera semplice il precedente database, è sufficiente dare l'istruzione

```
\DTLdisplaydb{messaggi}
```

che dispone i dati in una tabella, come quella riportata nella figura 1. In caso di *database* corposi, la tabella può essere particolarmente lunga e occupare più pagine. Sarà opportuno, allora, usare una `longtable`, tramite il comando

```
\DTLdisplaylongdb[<opzioni>
  {<nome_database>}
```

dove le `⟨opzioni⟩` servono a impostare l'aspetto dei vari elementi dell'ambiente `longtable`.

Tornando alla figura 1, si può notare che le varie colonne sono allineate automaticamente in considerazione del tipo di dato contenuto. Il testo, per esempio,

2. Il primo dei due comandi serve per impostare il separatore a un carattere generico (per esempio `;`), il secondo nel caso speciale che il separatore sia il carattere `TAB`.

3. È perfettamente lecito introdurre nel file CSV comandi L^AT_EX, che verranno correttamente interpretati, per cui, per esempio, il primo campo della prima riga del file `messaggi.csv` può essere riscritto così: `"\TeX{} e \LaTeX"`.

è allineato a sinistra, mentre i numeri sono allineati a destra. `datatool`, infatti, è in grado di riconoscere quattro diversi tipi di dati: stringhe di testo, numeri interi, numeri reali, valuta.⁴ L'utente può modificare gli allineamenti predifiniti usando comandi della forma `\dtl<tipo_dato>align`, dove `<tipo_dato>` è uno dei seguenti: `string`, `int`, `real`, `currency`. Molti altri aspetti della tabella così ottenuta possono essere modificati in maniera limitata (presenza di filetti orizzontali o verticali, aspetto dell'intestazione, applicazione di particolari istruzioni a colonne contenenti un certo tipo di dato, ecc.). Tutto ciò è adeguatamente descritto nel manuale.

Questo modo di visualizzare un *database*, è comodo solo se non si deve operare in qualche modo sui dati, per organizzarli, filtrarli, manipolarli, ecc. A questo scopo è preferibile usare il comando `\DTLforeach`, che permette di scorrere tutti i *record* e operare sui campi ivi contenuti. La sua sintassi è la seguente:

```
\DTLforeach[<condizioni>]{<nome_database>}
  {<assegnazioni>}{<istruzioni>}
```

Le `<condizioni>` sono, appunto, opzioni di filtro da applicare ai dati (ad esempio ci interessa di visualizzare solo *record* che contengano, per un dato campo, valori maggiori di zero). Le `<assegnazioni>` costituiscono un modo, invece, per filtrare le colonne. Possiamo infatti essere interessati a visualizzare solo una parte dei campi riportati nel *database*; assegnando un particolare campo a un comando, possiamo richiamare il valore di quel campo, per ogni *record*, tramite il relativo comando. Le `<istruzioni>`, infine, determinano ciò che vogliamo fare con i dati in questione.

Possiamo, ad esempio, ricostruire (approssimativamente) la tabella illustrata nella figura 1, con l'aggiunta di una ulteriore colonna con i totali, per mezzo del seguente codice:

```
\begin{tabular}{lrrrrr}
  \bfseries Categoria &
  \bfseries Novembre 2008 &
  \bfseries Dicembre 2008 &
```

4. `datatool` è inoltre corredato di una serie di comandi accessibili all'utente, che permettono di identificare, comparare e manipolare questi quattro tipi di dati. In questo articolo non prenderemo in esame tali comandi, se non dove è necessario per la comprensione degli esempi. Per una panoramica completa si rimanda al manuale [Talbot, 2009].

Categoria	Novembre 2008	Dicembre 2008	Gennaio 2009	Febbraio 2009	Totale
T _E X e L ^A T _E X	235	212	289	321	1,057
ConTeXt	0	0	0	0	0
Altri programmi	24	18	19	31	92
Tipografia	13	11	18	22	64
Corsi e didattica	3	2	4	4	13
Edizioni critiche	2	0	5	3	10

Figura 2: Aggiunta di una colonna alla visualizzazione del *database* messaggi.

```

\bfseries Gennaio 2009 &
\bfseries Febbraio 2009 &
\bfseries Totale%
\DTLforeach*{messaggi}{%
  \cat=cat,\nov=nov,\dic=dic,%
  \gen=gen,\feb=feb}{%
  \
  \cat \gdef\tot{0} &
  \nov \DTLgadd{\tot}{\nov}{\tot} &
  \dic \DTLgadd{\tot}{\dic}{\tot} &
  \gen \DTLgadd{\tot}{\gen}{\tot} &
  \feb \DTLgadd{\tot}{\feb}{\tot} &
  \tot
}
\end{tabular}

```

Il risultato è visibile nella figura 2. Per generare i totali dell'ultima colonna, usiamo il comando `\DTLgadd`⁵ i cui tre argomenti obbligatori rappresentano rispettivamente: il registro in cui viene immagazzinato il risultato della somma e i due addendi. All'inizio di ogni riga azzeriamo il valore del totale. Poiché ci troviamo all'interno di una cella l'assegnazione deve essere globale (`\gdef`). Si noterà che `\DTLforeach` è chiamato, in questo caso, nella versione con asterisco. Questo significa che il *database* viene usato in modalità di sola lettura. In precedenza, infatti, è stato già accennato che `datatool` permette anche di creare e modificare un *database*. Se si vuole essere sicuri che nessuna modifica venga apportata al file è bene usare la versione con asterisco dei comandi che manipolano i *database*.

A questo punto, avendo esaminato gli strumenti da usare, possiamo introdurre un paio di esempi concreti.

5. `\DTLgadd` fa parte di una libreria di comandi per effettuare calcoli aritmetici in virgola fissa inclusa in `datatool` e basata sul pacchetto `fp`.

2.2 Primo esempio: *mail merging*

Le maggiori distribuzioni di T_EX contengono più di una classe disegnata per comporre lettere formali [Kohm and Morawski, 2009, Thompson, 2009, Lamport et al., 2008, Dekker, 2008, Mezzetti, 2006]. La maggior parte di queste forniscono anche meccanismi per produrre automaticamente le etichette con gli indirizzi, e qualcuna anche la possibilità di creare numerose istanze della stessa lettera verso diversi destinatari, il cui indirizzo viene estratto da un file esterno (*mail merging*). In caso queste funzionalità non venissero fornite dalla classe in questione, si possono usare pacchetti dedicati [Emmel, 2006, Rahtz et al., 2003, Veytsman, 1997, Braams, 1994].

Il problema è che, in ogni caso, il formato accettato da tutti questi pacchetti per il file che contiene la rubrica, non è compatibile con il formato CSV. In alcuni casi è possibile costringere l'applicazione in cui originariamente si trovano i dati ad esportare nel formato richiesto, ma in genere sarebbe più comodo poter lavorare con un file CSV. Grazie a datatool questo è possibile.

L'esempio che proporrò usa la classe `guitletter`. Questa è semplicemente una versione della classe standard `letter` personalizzata in modo da inserire nella testatina e nel piè di pagina il logo del G_UT e altre informazioni relative al gruppo.⁶ Tutto ciò che si può fare con `guitletter` si può fare anche con `letter`. In particolare vedremo come preparare automaticamente una lettera di benvenuto nel gruppo per diversi destinatari e come generare automaticamente le etichette con gli indirizzi da apporre sulle buste.

Per comodità dividiamo in più parti l'analisi del codice, cominciando dal preambolo:

```
\documentclass[12pt,color]{guitletter}
\usepackage[italian]{babel}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{datatool}

\DTLloaddb{indirizzi}{indirizzi.csv}

\signature{Massimiliano Dominici}
\city{Pisa}
```

6. La classe non è disponibile pubblicamente.

```
\date{\today}
\subject{Adesione al \GuITtext}
```

```
\makelabels
```

Conosciamo già il comando `\DTLloaddb` che serve, in questo caso, a caricare la nostra rubrica degli indirizzi:

```
Nome,Cognome,Via,Numero,Città,Prov,CAP
Mario,Bianchi,Piazza Cavour,7,Novi Ligure,AL,15067
Carlo,Rossi,Via Roma,8,Noto,SR,96017
Franco,Verdi,Corso Garibaldi,3,Senigallia,AN,60017
```

Le altre istruzioni riguardano la composizione delle lettere e delle etichette: `\signature`, `\city`, `\today` e `\subject` fanno parte delle informazioni su mittente, oggetto, ecc., mentre `\makelabels` specifica che vogliamo anche stampare le etichette con gli indirizzi dei destinatari.

Il corpo del documento, è invece il seguente:

```
\begin{document}

\DTLforeach*{indirizzi}{%
  \Nome=Nome,\Cognome=Cognome,%
  \Via=Via,\Numero=Numero,%
  \Citta=Città,\Prov=Prov,\CAP=CAP}{%
\begin{letter}{\Nome\ \Cognome\\
              \Via, \Numero\\
              \CAP\ -- \Citta\ (\Prov)}

\opening{Caro socio,}

<testo della lettera>

\closing{Cordialmente,}

\end{letter}
}

\end{document}
```

Ogni singola lettera deve essere racchiusa all'interno di un ambiente `letter`. Mentre `\opening` e `\closing` si limitano ad inserire le formule di saluto, formattandole adeguatamente, le informazioni da variare per ciascuna lettera si trovano nell'argomento obbligatorio dell'ambiente. Proprio perché tali informazioni devono variare, le inseriamo sotto forma di *registri*. Di volta in volta, leggendo i vari *record* della rubrica, il comando `\DTLforeach` assegna a tali registri il valore del campo corrispondente, generando tre lettere (una per *record*).

Le etichette vengono generate automaticamente, al momento della chiamata `\end{document}`. L'istruzione `\makeLabels` che abbiamo visto in precedenza, infatti, abilita la scrittura delle informazioni relative ai vari indirizzi sul file `.aux`, dove `\end{document}` può leggerle e poi provvedere a stamparle.

2.3 Secondo esempio: produrre schede bibliografiche

In questo secondo esempio mostreremo che è possibile, fino a un certo punto, operare con diversi file CSV come se fossero le tabelle di un *database* relazionale. Attenzione: questo tipo di approccio è, in genere, sconsigliabile, ed è preferibile, come detto in precedenza, organizzare appropriatamente i dati al momento dell'esportazione in CSV dall'applicazione in cui i dati sono contenuti, e lavorare quindi su un'unica tabella. Pianificare questa operazione in funzione di ciò che si vuole ottenere è fondamentale per avere buoni risultati.

Tuttavia, allo scopo di esplorare le potenzialità di `datatool`, e in considerazione del fatto che non sempre l'utente è in grado di organizzare da sé tali dati, ma può trovarsi nella condizione di dover lavorare su materiale fornito da altri, daremo anche un esempio dell'uso contemporaneo di più file CSV in relazione l'uno con l'altro.

Una possibile applicazione di questo principio è la costruzione di schede bibliografiche (per esempio per una biblioteca personale) a partire da tre file in cui siano contenute, rispettivamente, le informazioni riguardanti gli *autori*, le *case editrici* e i singoli *titoli*. Uno degli scopi di un *database* relazionale è quello di evitare la ridondanza dei dati: è più comodo, infatti, dover gestire le informazioni relative, ad esempio, a un autore in una singola collocazione e richiamare poi queste informazioni tramite un riferimento.

I primi due file CSV riportati nella figura 3 non contengono riferimenti a dati presenti in altri file, ma il terzo CSV, invece, contiene due campi (`RefAutore`

Id, Nome, Cognome	Id, Nome, Luogo
001, Carlo Emilio, Gadda	001, UTET, Torino
002, Ryunosuke, Akutagawa	002, Garzanti, Milano
003, Francesco, De Sanctis	003, Adelphi, Milano,
004, Louis, Stevenson	004, TEA, Milano
005, Jan, Tschichold	005, BUR, Milano
006, Matthew P., Shiel	006, Mondadori, Milano
007, Giorgio, Manganelli	007, Bompiani, Milano
008, Galileo, Galilei	008, Edizioni Sylvestre Bonnard, Milano
009, , Erodoto	

Codice, Titolo, RefAutore, Anno, RefEditrice, Note
001, Storia della letteratura italiana, 003, 2006, 005,
002, Weir di Hermiston, 004, 2000, 006,
003, Le storie, 009, 2006, 001, 2 volumi
004, La Meccanica, 001, 1999, 002,
005, Un fulmine sul 220, 001, 2005, 002,
006, La nube purpurea, 006, 2004, 003,
007, Rashomon e altri racconti, 002, 2008, 004,
008, Opere, 008, 2005, 001,
009, Teatro, 007, 2008, 007,
010, La forma del libro, 005, 2003, 008,

Figura 3: File CSV rappresentanti tabelle di un database bibliografico.

e RefEditrice) che prevedono un riferimento agli altri due file. Il riferimento si effettua tramite il codice dell'autore, o della casa editrice corrispondenti, per come sono specificati nell'apposito campo del relativo file. Così, nella seconda riga di titoli.csv, il terzo campo (RefAutore) riporta il valore 003 che corrisponde, nel file autori.csv all'autore Francesco de Sanctis. Nella sesta riga di titoli.csv, il quinto campo (RefEditrice) riporta il valore 002 che corrisponde, nel file case_editrici.csv alla casa editrice Garzanti.

Con datatool è possibile usare filtri sui valori di tali campi per connettere i dati contenuti nei vari file CSV.

Il codice seguente produce come risultato una serie di schede bibliografiche (una per pagina), come quelle riportate nella figura 4.

```

\newcounter{scheda}
\setcounter{scheda}{0}

\DTLforeach*{autori}{%
  \AutId=Id,\Nome=Nome,\Cognome=Cognome}{%
  \DTLforeach*[\AutId=\RefAutore]{titoli}{%
    \RefAutore=RefAutore,%
    \RefEditore=RefEditrice,\Titolo=Titolo,%
    \Anno=Anno,\Note=Note,\Codice=Codice}{%
  \DTLforeach*[\EdId=\RefEditore]{editori}{%
    \EdId=Id,\NomeEditore=Nome,\Luogo=Luogo}{%
    \vspace*{\stretch{1}}
    \stepcounter{scheda}
    \centering
    \fbox{
      \begin{tabular}[t]{%
        >{\columncolor[gray]{.8}}r
        >{\raggedright\arraybackslash}p{3.5cm}
      }
      \multicolumn{2}{1}{Scheda n° \thescheda}\
      \midrule
      Autore: &
        \Nome
        \CheckEmpty{\Nome}{}{\ }
        \Cognome\
      \midrule
      Titolo: & \Titolo\
      \midrule
      Editore: &
        \NomeEditore
        \CheckEmpty{\NomeEditore}{}
        {\CheckEmpty{\Luogo}{}{, }}
        \Luogo\
      \midrule
      Anno: & \Anno\
      \midrule
      Note: & \Note\
      \midrule
      Codice: & \Codice\
      \midrule
    \end{tabular}
  }
}

```

Scheda n° 1	Scheda n° 2	Scheda n° 3
Autore: Ryunosuke Akutagawa	Autore: Francesco De Sanctis	Autore: Erodoto
Titolo: Rashomon e altri racconti	Titolo: Storia della letteratura italiana	Titolo: Le storie
Editore: TEA, Milano	Editore: BUR, Milano	Editore: UTET, Torino
Anno: 2008	Anno: 2006	Anno: 2006
Note:	Note:	Note: 2 volumi
Codice: 007	Codice: 001	Codice: 003
Scheda n° 4	Scheda n° 5	Scheda n° 6
Autore: Carlo Emilio Gadda	Autore: Carlo Emilio Gadda	Autore: Galileo Galilei
Titolo: La Meccanica	Titolo: Un fulmine sul 220	Titolo: Opere
Editore: Garzanti, Milano	Editore: Garzanti, Milano	Editore: UTET, Torino
Anno: 1999	Anno: 2005	Anno: 2005
Note:	Note:	Note:
Codice: 004	Codice: 005	Codice: 008
Scheda n° 7	Scheda n° 8	Scheda n° 9
Autore: Giorgio Manganelli	Autore: Matthew P. Shiel	Autore: Louis Stevenson
Titolo: Teatro	Titolo: La nube purpurea	Titolo: Weir di Hermiston
Editore: Bompiani, Milano	Editore: Adelphi, Milano	Editore: Mondadori, Milano
Anno: 2008	Anno: 2004	Anno: 2000
Note:	Note:	Note:
Codice: 009	Codice: 006	Codice: 002
	Scheda n° 10	
	Autore: Jan Tschichold	
	Titolo: La forma del libro	
	Editore: Edizioni Sylvestre Bonnard, Milano	
	Anno: 2003	
	Note:	
	Codice: 010	

Figura 4: Schede bibliografiche.

```

\vspace*{\stretch{1}}
\clearpage
}
}
}

```

Il risultato è stato ottenuto annidando tre diverse chiamate a `\DTLforeach` (una per file CSV) e imponendo la condizione, nei cicli interni, che i valori dei campi di riferimento del file CSV principale (`titoli.csv`) coincidessero con il valore del campo di identificazione degli altri due file CSV. Ogni scheda ha un numero progressivo di identificazione, ottenuto con `\thescheda` che visualizza il valore corrente del contatore che tiene traccia dell'avanzamento nella "produzione" di schede.

Le schede sono ordinate per autore. `datatool` consente infatti di imporre ai *database* caricati un ordinamento, con il comando `\DTLsort`. Le impostazioni usate nel nostro caso sono state le seguenti:

```
\DTLloaddb{autori}{autori.csv}
\DTLloaddb{editori}{case_editrici.csv}
\DTLloaddb{titoli}{titoli.csv}

\DTLsort*{Cognome,Nome}{autori}
\DTLsort*{Nome,Luogo}{editori}
\DTLsort*{Titolo,Anno,Codice}{titoli}
```

Come si nota dal codice precedente, l'ordinamento può interessare anche più campi. Ovviamente, nel nostro caso, l'ordinamento principale è quello imposto dal ciclo più esterno.

Il comando `\CheckEmpty`, costruito a partire da *macro* fornite dal pacchetto `datatool`, serve ad evitare la presenza di spazi o punteggiatura spuri. La sua definizione è la seguente:

```
\newcommand\CheckEmpty[3]{%
  \ifthenelse{\DTLiseq{#1}{}}{#2}{#3}%
}
```

Infine, il codice sopra esposto richiede di caricare i pacchetti `booktabs` e `colortbl` per le rifiniture della tabella e i tocchi di colore.

3 Tabelle di dati numerici

Come abbiamo accennato nell'introduzione, i vari formati di esportazione sotto forma di matrici di valori, da parte di numerose applicazioni di calcolo numerico, possono essere assimilati, agli effetti pratici, a dei CSV. La loro utilità sta nel fatto che possono essere usati come formato di interscambio tra applicazioni diverse e, per ciò che ci riguarda più da vicino, per tracciare grafici con \LaTeX .

Di norma i vari *software* che abbiamo citato nell'introduzione (vedi sezione 1) hanno anche la capacità di generare grafici (tramite funzionalità interne, o appoggiandosi a programmi come `GNUPLOT`). Tuttavia la possibilità di poter controllare l'aspetto tipografico delle illustrazioni, in modo da avere una stretta integrazione con il testo è fondamentale per ottenere un risultato estetico di ottima qualità (si veda a questo proposito [De Marco \[2008\]](#)).

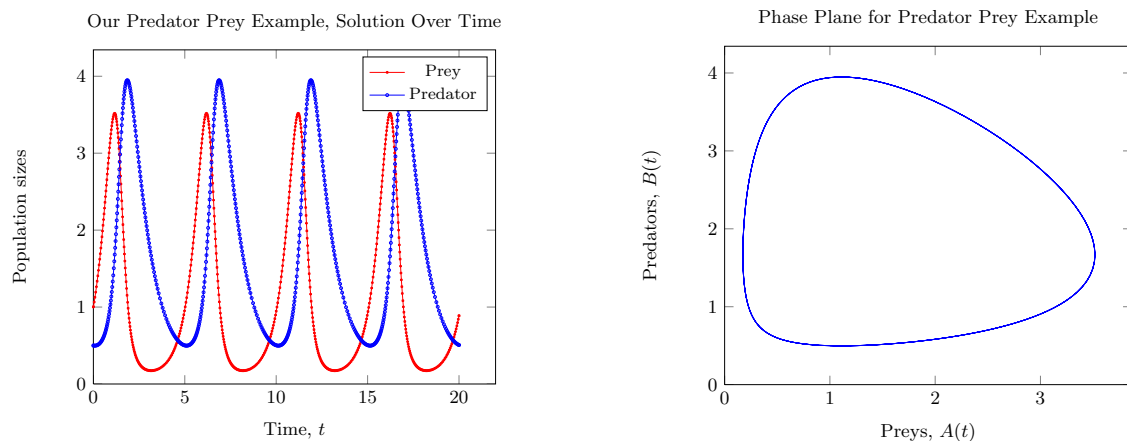


Figura 5: Equazioni di Lotka-Volterra

3.1 Il pacchetto pgfplots

Il pacchetto `pgfplots` [Feuersänger, 2009a] si appoggia al sistema grafico PGF/TikZ [Tantau, 2008] per tracciare grafici a partire da tabelle di coordinate che possono essere passate alle *macro* che si occupano del tracciamento secondo diverse modalità: esplicitamente, implicitamente sotto forma di espressione matematica, delegando a `GNUPLOT`⁷ il calcolo esplicito, caricandole da un file esterno.

L'ambiente principale è `axis`, che traccia assi e griglia secondo le indicazioni dell'utente, e all'interno del quale è possibile aggiungere i singoli grafici tramite il comando `\addplot`. I tipi di grafici supportati sono numerosi e comprendono istogrammi, diagrammi di dispersione, diagrammi cumulativi, ecc.

Come tutti i pacchetti che si appoggiano a PGF, e come PGF stesso, l'uso di `pgfplots` è configurabile praticamente all'infinito. In questa sede ci limiteremo a mostrare solo alcune possibili modifiche alle impostazioni predefinite, rimandando ad un'attenta lettura del manuale per chi volesse acquistare dimestichezza con le varie opzioni.

3.2 Esempio: importare dati da GNU Octave

GNU Octave⁸ è un programma *free software* per l'analisi numerica, parzialmente compatibile con Matlab e corredato di funzionalità per il calcolo matriciale. Il programma avvia una *shell* all'interno della quale è possibile passare le istruzioni all'interprete, il quale restituisce (implicitamente o esplicitamente) un *output* sotto forma matriciale. Le istruzioni possono essere anche preparate in un file esterno, sotto forma di funzioni o di *script*, ed essere poi richiamate dalla *shell*. I risultati ottenuti possono essere visualizzati sotto forma di grafici oppure esportati in file esterni sotto diversi formati. Quello che ci interessa è il formato ASCII. I valori vengono espressi in notazione scientifica, che *pgfplots* è in grado di comprendere, e organizzati in tabelle.

Questo, per esempio, è l'inizio di un file del genere:

```
0.00000000e+00 1.00000000e+00 5.00000000e-01
5.66993145e-03 1.00797044e+00 4.99726732e-01
4.51639430e-02 1.06532387e+00 4.98401027e-01
8.51085701e-02 1.12669546e+00 4.98125368e-01
1.25796018e-01 1.19282722e+00 4.99007906e-01
1.67235284e-01 1.26407660e+00 5.01190251e-01
2.06413960e-01 1.33517539e+00 5.04527388e-01
2.43682108e-01 1.40627123e+00 5.08935002e-01
```

Il file immagazzina i risultati di un'analisi numerica effettuata su equazioni di Lotka-Volterra (note anche come equazioni preda-predatore, in quanto forniscono un modello matematico per un ecosistema in cui interagiscono due specie animali: una come predatore, l'altra come preda).

Si noterà che il formato differisce da quello di un CSV standard: infatti il separatore di campo è in questo caso uno spazio (o più spazi successivi). Tuttavia, come premesso nell'introduzione, in questo articolo consideriamo un formato CSV in senso esteso (si veda la nota 1). *pgfplots*, d'altra parte, è in grado di interpretare correttamente anche CSV veri e propri ed altri formati con separatori non standard. Per una lista completa si consulti il manuale. Le righe precedute da # o % vengono automaticamente saltate (per mantenere compatibilità con file

7. <http://www.gnuplot.info/>.

8. <http://www.gnu.org/software/octave/>.

generati da GNUPLOT), e la prima riga viene considerata come intestazione che definisce i nomi delle colonne se contiene almeno un valore non numerico.⁹

Gli script di Octave usati per generare i dati sono i seguenti:

```
t0 = 0;
tf = 20;
init_vals = [1; 0.5];
options=odeset('AbsTol',1.e-12,'RelTol',
  1.e-9,'InitialStep',2,'MaxStep',2);
[t,x] = ode45(@pred_prey_odes,[t0,tf],
  init_vals,options);
A = [t,x];
save -ascii PredPrey.dat A

function deriv_vals = pred_prey_odes(t,x)
deriv_vals = zeros(size(x));
deriv_vals(1) = 2*x(1) - 1.2*x(1).*x(2);
deriv_vals(2) = -1*x(2) + 0.9*x(1).*x(2);
```

e sono stati adattati a partire da quelli presenti su questa pagina *web*: http://paws.wcu.edu/emcnelis/NCSI_Houston08.html. L'adattamento è consistito nel sostituire le istruzioni per generare i grafici con quelle per salvare i dati nel file `PredPrey.dat`.

Tali dati sono stati poi importati in un file `.tex`, in cui era stato precedentemente caricato il pacchetto `pgfplots`.

In primo luogo tracciamo il grafico con l'andamento delle soluzioni nel tempo:

```
\begin{tikzpicture}
  \begin{axis}[ymin=0,xmin=0,
    xlabel={Time, $t$},
    ylabel={Population sizes},
    title={Our Predator Prey Example,
      Solution Over Time},
    legend entries={Prey,Predator}]
    \addplot[red,mark=asterisk,
```

9. Appoggiandosi sul pacchetto `pgfplotstable` [Feuersänger, 2009b] per la lettura di tabelle di valori da file esterni, `pgfplots` riconosce i quattro tipi di dati che hanno senso per il primo: stringhe di caratteri alfanumerici, interi, reali e date. Tuttavia è in grado di usare realmente solo gli ultimi tre. Come si vede dall'esempio riportato sopra, i reali possono essere espressi anche in notazione scientifica.

```

        mark options={scale=.35}]
    plot table[x index=0,
        y index=1] {PredPrey.dat};
    \addplot[blue,mark=o,
        mark options={scale=.35}]
    plot table[x index=0,
        y index=2] {PredPrey.dat};
\end{axis}
\end{tikzpicture}

```

I grafici vengono tracciati, come abbiamo detto sopra, tramite il comando `\addplot`. Poiché ci troviamo di fronte a un file che contiene più di due colonne di dati, dobbiamo aggiungere la specificazione `table`, che ci permette di scegliere di volta in volta le colonne da usare per le coordinate.¹⁰ Le colonne possono essere identificate da una stringa in un'intestazione, oppure, se il file ne è privo, da un indice progressivo, come in questo caso. Scegliamo per il grafico della prima soluzione le colonne 1 e 2 (indici 0 e 1) e per il grafico della seconda le colonne 1 e 3 (indici 0 e 2). Le altre opzioni di `\addplot` servono a personalizzare l'aspetto visuale del grafico (colore, stile e dimensioni dei marcatori, ecc.). Altre personalizzazioni sono state effettuate sull'aspetto degli elementi "esterni" della figura: assi, etichette, legenda, e sono contenute nell'argomento opzionale dell'ambiente `axis`.

Tracciamo anche, in una illustrazione a parte, il diagramma nel piano delle fasi, scegliendo le colonne 2 e 3 (indici 1 e 2):

```

\begin{tikzpicture}
  \begin{axis}[ymin=0,xmin=0,
    xlabel={Preys,  $A(t)$ },
    ylabel={Predators,  $B(t)$ },
    title={Phase Plane for Predator Prey Example},
    line width=.1pt]
    \addplot[blue,mark=none]
    table[x index=1,y index=2] {PredPrey.dat};
  \end{axis}
\end{tikzpicture}

```

In questo caso abbiamo ommesso i marcatori (`mark=none`) e modificato lo spessore delle linee. Dopo aver aggiunto, nel preambolo, alcune impostazioni globali per cambiare le dimensioni dei caratteri

10. Altrimenti avremmo potuto usare la specificazione `file`.

```

\pgfplotsset{
  tick label style={font=\small},
  label style={font=\small},
  title style={font=\small},
  legend style={font=\footnotesize},
}

```

otteniamo finalmente le due illustrazioni contenute nella figura 5.

4 Raffronto tra datatool e pgfplots

Come si può arguire dalla scelta degli esempi, benché entrambi i pacchetti siano disegnati per gestire tabelle di dati organizzati in *record* e *campi*, essi sono tuttavia indirizzati a diversi ambiti di applicazioni. Questo nonostante entrambi, accanto alle funzionalità fondamentali (visualizzazione e manipolazione di basi di dati per datatool e tracciamento di grafici per pgfplots) consentano anche, tramite pacchetti “ancillari” distribuiti insieme a quello principale, di eseguire compiti secondari che finiscono per sovrapporsi. datatool, infatti, permette di visualizzare il contenuto di un *database* anche sotto la forma di un semplice grafico, o di un istogramma o di un diagramma circolare (grafico a torta), grazie ai pacchetti dataplot, databar e datapie. D’altra parte pgfplotstable può caricare, manipolare e visualizzare una tabella contenente dati di diverso tipo (testi, numeri interi e reali, date).

Il fattore discriminante nell’uso dell’uno o dell’altro pacchetto rimane il tipo di dati con i quali si deve lavorare e la loro organizzazione. Se i dati sono costituiti da matrici di valori numerici da visualizzare con grafici, pgfplots è la scelta più opportuna, scelta che diventa addirittura obbligata se i dati sono forniti in notazione scientifica, che datatool non è in grado di riconoscere. D’altra parte, se lo scopo è quello di manipolare e visualizzare dati di diverso tipo organizzati in un *database*, datatool risulta essere lo strumento più adatto, grazie alla flessibilità di comandi come `\DTLforeach`, alla libreria di funzioni per confrontare stringhe di caratteri di ogni tipo, e, conseguentemente, alla possibilità di applicare filtri ai dati da visualizzare.

5 Ringraziamenti e note

L'impulso iniziale a scrivere questo articolo lo devo a Maurizio W. Himmelmann e all'esigenza di trovare un sistema "comodo" per gestire in via \TeX nica la corrispondenza del \GUIT . Per i sensibili miglioramenti rispetto alla versione iniziale dell'articolo devo ringraziare l'anonimo recensore e la redazione di *Ars \TeX nica*.

È superfluo far notare, infine, che tutti i dati esposti nella sezione 2, tranne quelli relativi alle schede bibliografiche, sono inventati di sana pianta e che ogni eventuale riferimento a fatti e persone realmente esistenti è puramente casuale.

Riferimenti bibliografici

Johannes Braams. Creating a mailing. <http://www.ctan.org/get/macros/latex/contrib/mailling/mailling.pdf>, 1994.

Agostino De Marco. Gestione avanzata delle figure in \LaTeX . *Ars \TeX nica*, (6):10–27, Ottobre 2008. ISSN 1828-2350. URL <http://www.guit.sssup.it/arstexnica.php>.

Wybo Dekker. The isodoc class. <http://www.ctan.org/get/macros/latex/contrib/isodoc/isodoc.pdf>, 2008.

Thomas Emmel. Making labels, visiting cards, pins and flash-cards with \LaTeX . <http://www.ctan.org/get/macros/latex/contrib/ticket/doc/manual.pdf>, 2006.

Christian Feuersänger. Manual for package PGFPLOTS. <http://www.ctan.org/get/graphics/pgf/contrib/pgfplots/doc/latex/pgfplots/pgfplots.pdf>, 2009a.

Christian Feuersänger. Manual for package PGFPLOTS TABLE . <http://www.ctan.org/get/graphics/pgf/contrib/pgfplots/doc/latex/pgfplots/pgfplotstable.pdf>, 2009b.

IETF. Rfc4180. <http://tools.ietf.org/html/rfc4180>, Ottobre 2005.

Markus Kohm and Jens-Uwe Morawski. Koma script. the guide. <http://www.ctan.org/get/macros/latex/contrib/koma-script/scrguide.pdf>, 2009.

- Leslie Lamport, Frank Mittelbach, and Rainer Schöpf. Standard letter document class for L^AT_EX2e version. <http://www.ctan.org/get/macros/latex/base/letter.dtx>, 2008.
- G. Mezzetti. The c.d.p. bundle. <http://www.ctan.org/get/macros/latex/contrib/cdpbundl/cdpbundl.dtx>, 2006.
- Sebastian Rahtz, Leonor Barroca, Grant Gustafson, and Julian Gilbey. A package for making sticky labels in L^AT_EX. <http://www.ctan.org/get/macros/latex/contrib/labels/labels.pdf>, 2003.
- Nicola Talbot. csvtools v1.24 : A L^AT_EX2e package providing access to data saved in a csv file. <http://www.ctan.org/tex-archive/obsolete/macros/latex/contrib/csvtools/doc/csvtools.pdf>, 2007.
- Nicola Talbot. datatool v 2.02: Databases and data manipulation. <http://www.ctan.org/get/macros/latex/contrib/datatool/datatool.pdf>, 2009.
- Till Tantau. The tikz and pgf packages. manual for version 2.00. <http://www.ctan.org/get/graphics/pgf/base/doc/generic/pgf/pgfmanual.pdf>, 2008.
- Paul A. Thompson. A new letter, fax, memo document class for L^AT_EX2e. <http://www.ctan.org/get/macros/latex/contrib/newlrm/manual.pdf>, 2009.
- Boris Veytsman. Printing envelopes and labels in L^AT_EX2e: Envlab package. user guide. <http://www.ctan.org/get/macros/latex/contrib/envlab/envlab.pdf>, 1997.