Claudio Beccari

Il LaTeX Reference Manual
commentato

Versione v.2.9.10 del 2019-01-17

Associati anche tu al GUIT

Fai click per associarti

L'associazione per la diffusione di TEX in Italia, riconosciuta ufficialmente in ambito internazionale, si sostiene *unicamente* con le quote sociali.

Se anche tu trovi che questa guida tematica gratuita ti sia stata utile, il mezzo principale per ringraziare gli autori è diventare socio.

Divenendo soci si ricevono gratuitamente:

- l'abbonamento alla rivista ArsTEXnica;
- il DVD TEX Collection;
- un eventuale oggetto legato alle attività del GUIT.

L'adesione al GUIT prevede un quota associativa compresa tra 12,00 € e 70,00 € a seconda della tipologia di adesione prescelta ed ha validità per l'anno solare in corso.

Il LATEX Reference Manual commentato

Questo documento è curato da Claudio Beccari.

Introduzione

Questa guida tematica è una versione commentata della Guida scritta direttamente da Leslie Lamport nel 1994 quando è uscito LaTeX 2_ε, (Lamport, 1994).

Non è una versione commentata di tutto il manuale; questo è articolato in capitoli e in appendici: qui mi occuperò solo dell'appendice C, *Reference Manual*, dove Lamport descrive tutti i comandi che fanno parte del mark-up chiamato LaTeX. Il resto si trova descritto con maggiore o minore completezza in moltissime guide, anche in italiano, e il sito GuIt ne propone almeno due, di cui *L'arte di scrivere con LaTeX* (Pantieri e Gordini, 2018) rappresenta certamente un ottimo punto di partenza. In questa guida gli autori hanno fatto delle scelte di cosa presentare e di cosa omettere, e hanno allargato molto il loro discorso ad ambiti che rappresentano delle estensioni notevoli di LaTeX, specialmente nell'uso pratico della maggior parte dei lettori a cui è dedicato, cioè massimamente i laureandi universitari. Anch'io, qui, ho fatto delle selezioni non tanto su che cosa presentare quanto su che cosa approfondire e che cosa commentare; quindi posso dire che l'intero *Reference Manual* non è effettivamente riportato con completezza da nessuna parte, per lo meno in nessuna guida italiana; per un'ottima guida il lettore può riferirsi a quella scritta in inglese, *Guide to LaTeX* (Kopka e Daly, 2004), abbastanza concisa ma tale da coprire moltissime estensioni. Il lettore non scordi, per altro, che mentre la guida di Lamport è "stabile", l'intero sistema TeX è in forte evoluzione e comprende ora dei programmi che quando Lamport ha creato la seconda versione del suo mark-up erano ancora *in fieri*, così come moltissimi pacchetti di estensione. Tuttavia è opportuno avere a disposizione un manuale in cui siano toccati, sia pure non con la stessa profondità, tutti gli argomenti trattati nel *Reference Manual*.

Questa, quindi, non è una traduzione: è qualcosa di più perché è com-

mentata; è qualcosa di meno perché mancano alcune piccole parti che ho ritenuto a torto o a ragione di accennare appena. Per esempio parlo molto succintamente dell'ambiente *tabbing*[1]; ritengo che le intenzioni di Leslie Lamport nel produrre questo ambiente nel 1984, quando LaTeX era appena uscito, erano ottime, ma la pratica ha dimostrato che questo ambiente era troppo legato storicamente alle tabulazioni eseguite con le macchine da scrivere. Forse in qualche rara circostanza esso può essere utile, ma c'è certamente qualche modo più moderno di eseguire le tabulazioni, o meglio, di mostrare del testo strutturato che mostri meglio la struttura; qualche modo che non sia quello di comporre in un ambiente solo apparentemente facile da usare da parte di chi abbia reminiscenze di macchine da scrivere elettro-meccaniche; esso risulta fragile, così da non poter essere usato dentro altri ambienti o altri comandi. È probabile che il lettore esperto conosca molto di LaTeX ma non abbia mai sentito nominare l'ambiente *tabbing*; ciò giustifica la mia reticenza a parlarne.

L'ordine in cui Lamport presenta il suo materiale in un certo senso parte dalle strutture tipografiche più fini per salire a quelle più complesse, ma non sempre quest'ordine è rispettato; qualche volta si ha l'impressione che i vari argomenti siano trattati alla rinfusa, ma Lamport aveva invece una idea precisa per seguire quell'ordine. Io voglio però seguire la stessa disposizione anche se il lettore qua e là potrebbe sentirsi a disagio. Non lo voglio mettere a disagio; voglio solo fargli seguire la stessa logica di esposizione che ha usato Lamport nel suo testo. Tra l'altro lui ha scritto l'insieme dei comandi che costituiscono LaTeX in una sola appendice, *Reference Manual*, e anch'io non romperò questa unitarietà descrivendo e commentando la sua appendice spezzandola in più capitoli.

In fondo che male c'è se si parte spiegando prima i segni di interpunzione e le loro idiosincrasie, la divisione in capoversi per poi passare a strutture di più ampio respiro?

Vale la pena, però, di domandarsi, al di là del contenuto di informazione che qualunque documento dovrebbe avere, quale sia la sua articolazione a stampa. Perché con certi stili di stampa lo spazio che segue ogni segno di interpunzione può essere uguale o diverso; perché la spaziatura dopo

[1] In realtà nell'indice trovate qualcosa, ma oltre ad alcuni miei commenti, è riportato solo l'elenco dei comandi da usare in questo ambiente per simulare e usare gli arresti di tabulazione e poco più in merito all'uso simultaneo di certi comandi insieme ai comandi nativi per gli accenti.

il punto fermo *può* oppure *deve* essere diversa a seconda del ruolo che il punto fermo svolge. Questo giusto per cominciare da elementi di stampa piccoli piccoli, ai quali di solito si dà pochissima importanza; oppure si dà una importanza che discende dall'uso della macchina da scrivere, ma non è legato per davvero agli scopi della tipografia.

Prendiamo lo spazio dopo il punto fermo all'interno dello stesso capoverso. Esso può indicare la fine di un periodo o può indicare una abbreviazione; È "giusto" che lo spazio che segue un punto di abbreviazione sia più piccolo di quello che segue il punto di fine periodo?

Prendiamo il rientro della prima parola di un capoverso; questa è una pratica molto diffusa, perché marca in modo inequivocabile l'inizio di un capoverso; questo a sua volta termina con un "punto e a capo". Se mancasse il rientro, come si farebbe a distinguere l'inizio di un capoverso che ne segue un altro la cui ultima riga, compreso il punto finale, arriva a occupare tutta la larghezza dello specchio di stampa? Ma se le cose stanno così, che senso ha rientrare un capoverso preceduto da un titolo, non da un altro capoverso?

LaTeX non risponde a tutte queste domande, ma fornisce i mezzi per rispondere consapevolmente con un comportamento omogeneo e coerente, la massima dote della tipografia.

Prendiamo strutture più articolate: che significato hanno i comandi di sezionamento `\part`, `\chapter`, giù giù fino a `\subparagraph`? Se si prende un romanzo, molto probabilmente si tratta di una struttura eccessiva, ma come applichiamo questa struttura a un testo come la Bibbia, formata di Libri, divisi in capitoli, a loro volta in versetti? Come lo applichiamo a una raccolta di poesie? Come applichiamo questo schema a un codice legale, tanto per fare un esempio il Codice Civile, diviso in titoli, a loro volta divisi in articoli, generalmente formati di pochi capoversi, talvolta chiamati commi? O a un testo legislativo complesso come per esempio la Legge Finanziaria, con una struttura complessa, ma che scende fino ai commi costituiti da voci di elenchi numerati con lettere, spesso di un solo capoverso, ma sovente anche formati da più capoversi. Esistono classi come *memoir* che presentano una struttura di sezionamento ancora più ampia di quella standard di LaTeX, ma che spesso risulta di difficile interpretazione nell'uso dei vari comandi, e che in ogni caso va adattata, come la struttura standard, al documento che si sta scrivendo.

Che cosa significano le varie parti della composizione della matematica?

Perché l'ambiente *eqnarray* non deve venire usato? Perché se il forte di LaTeX è davvero la composizione della matematica, proprio con questa bisogna evitare di usare certi ambienti o certi comandi, e perché bisogna estenderne la funzionalità mediante i pacchetti come `amsmath` e `amssymb`?

Perché è così difficile personalizzare le strutture di composizione offerte da LaTeX standard e bisogna rivolgersi a pacchetti di estensione?

Bene: accennerò a questi problemi ma consiglio il lettore di rivolgersi a guide più ampie e approfondite, altrimenti questa guida tematica finisce con l'andare troppo fuori argomento e invece di esporre e commentare il linguaggio (o *mark up*) LaTeX, come l'ha creato Leslie Lamport, finisce col diventare un'altra guida onnicomprensiva.

Nei limiti del possibile cercherò di affiancare la parola inglese a quella italiana; visto che i comandi LaTeX riecheggiano la terminologia inglese, penso che questo non sia inutile. Come penso che non sia inutile se scrivendo in italiano io usi la parola capoverso che in inglese si chiama *paragraph*, e la parola paragrafo per indicare quello che in inglese si chiama *section*; quindi cerco di non farmi fuorviare dai "falsi amici".

Avvertenza Questo *LaTeX Reference Manual commentato* non è sostitutivo del testo originale di Leslie Lamport (Lamport, 1994); il mio testo può contenere degli errori e in ogni caso contiene delle opinioni personali; quindi la risorsa definitiva per l'uso di LaTeX è e resta il testo di Leslie Lamport.

Nota bene Il grosso di questo testo trae la sua origine da uno dei capitoli della cosiddetta Guida GuIT (Beccari, 2018); questa è un'opera collettiva dove ho svolto principalmente la funzione di curatore nell'assemblare contributi arrivati da diversi membri del GuIT; li ringrazio nuovamente qui tutti; sono elencati nella pagina del retrofrontespizio di quella guida. Ringrazio invece Marco Stara che molto gentilmente ha letto tutta questa guida con l'occhio del nuovo utente di LaTeX e mi ha segnalato sia i refusi sia le cose poco chiare.

Questa traduzione commentata è dotata di un indice analitico che riporta tutti i comandi, gli ambienti, le classi, i pacchetti citati in questa traduzione commentata. Nella versione elettronica contiene tutti i collegamenti ipertestuali che agevolano enormemente la "navigazione" nel documento.

Indice

Elenco delle tabelle

Elenco delle figure

LaTeX Reference Manual 1

Avvertenze generali In questo capitolo riepilogherò in modo molto succinto la sintassi di tutti i comandi LaTeX, anche quelli spesso non descritti in altre guide. L'ordine del materiale segue l'appendice C del manuale di Leslie Lamport, il creatore di LaTeX.

Userò spesso la dicitura *comandi nativi*; questi sono quelli definiti all'interno del motore di composizione e quindi non sono le macro che questo motore interpreta (infatti talvolta chiamerò *interprete* questo motore di composizione, sia esso pdftex o xetex o luatex). Spesso si potrebbe dire *comandi eseguibili*, ma non sarebbe sempre corretto. Userò talvolta i nomi *control sequence* o *sequenza di controllo*, anche se mi dispiace di dover usare queste "locuzioni" di diverse parole; nello stesso tempo mi rendo conto che la parola *comando* senza aggettivi non sempre è corretta e che non si distingue bene da *macro*, che è definita con il linguaggio LaTeX ma spesso è difficile distinguerla da un comando nativo.

Talvolta negli esempi potrei usare la sintassi nativa del motore di composizione, invece della sintassi LaTeX. Non è il caso di preoccuparsi; starò molto attento nello spiegare quello che quegli esempi vogliono dire.

Parentesi graffe e quadre Le parentesi graffe generalmente racchiudono o gli argomenti obbligatori dei comandi oppure un insieme di testo e di comandi e istruzioni che formano un gruppo; i comandi e le istruzioni mantengono la loro efficacia solo all'interno del gruppo, tranne alcuni comandi il cui effetto è globale; questi comandi globali sono:

`\newcounter`	`\newlength`	`\pagenumbering`	`\hyphenation`
`\setcounter`	`\newsavebox`	`\thispagestyle`	`\newtheorem`
`\addtocounter`		`\pagecolor`	

Le parentesi quadre delimitano gli argomenti facoltativi di molti comandi; bisogna stare attenti alle possibili contraddizioni fra queste parentesi e le parentesi quadre che compaiono nel testo; queste potrebbero essere scambiate per quelle. Al fine di evitare queste incomprensioni è opportuno racchiudere le parentesi quadre testuali (o le frasi racchiuse fra parentesi quadre) fra parentesi graffe. I comandi con cui potrebbero succedere queste incomprensioni sono i seguenti:

`\\`	`\linebreak`	`\nolinebreak`	`\newcounter`	`\twocolumn`
`\item`	`\pagebreak`	`\nopagebreak`	`\newtheorem`	`\suppressfloats`

Si scriverà per esempio:

```
\begin{itemize}
\item {[omissis]} il conte disse allora: "<Ah, ah!">
\item[{[omissis]}] serve per indicare l'omissione di una
        parte di testo.
...
\end{itemize
```

per produrre:

- [omissis] il conte disse allora: «Ah, ah!»

[omissis] serve per indicare l'omissione di una parte di testo.

...

Gli argomenti degli ambienti Quando si apre un ambiente, solo il comando di apertura può ricevere uno o più argomenti, mentre il comando di chiusura non ne può accettare alcuno. Se l'ambiente ha un nome che contiene un asterisco finale, questo va ripetuto anche nel comando di chiusura; l'asterisco non è un argomento, ma fa parte integrante del nome.

Comandi fragili e robusti Quasi tutti i comandi di LaTeX sono robusti, nel senso che eseguono quel che devono eseguire senza che questa operazione possa essere modificata dalla posizione che essi hanno, per esempio negli argomenti di altri comandi. Gli altri sono fragili e, per evitare che possano combinare pasticci, bisogna proteggerli con `\protect` premesso al loro nome.

Il LaTeX 3 Team sta facendo l'impossibile per eliminare e/o modificare i pochi comandi fragili che rimangono, tuttavia ce ne sono ancora alcuni.

I comandi fragili si possono comportare male quando sono usati come argomenti mobili di altri comandi; sono argomenti mobili quelli che devono venire elaborati in fasi successive, per esempio quando devono essere scritti nei file ausiliari e poi riletti ed elaborati in una fase successiva quando si rilancia LaTeX.

I comandi con argomenti mobili sono tipicamente quelli che scrivono qualcosa che deve poi comparire negli indici generale e/o analitico; nei glossari, nelle liste delle figure e/o delle tabelle, eccetera; gli argomenti di `\footnote` e di `\footnotetext`[1]; gli argomenti di `\typein` e di `\typeout` che consentono di creare dei file `.tex` interattivi; il contenuto dell'ambiente *letter* definito all'interno della classe *letter*; l'argomento di `\thanks` usato tipicamente nei titoli, specialmente per indicare le afferenze degli autori; l'argomento facoltativo di `\bibitem`; l'argomento delle *@-espressioni* nella definizione degli incolonnamenti degli ambienti *tabular* e *array*.

È opportuno usare `\protect` con parsimonia, per non correre il rischio di inserirlo dove proprio non ci vuole; conviene inserirlo a posteriori quando l'esecuzione della compilazione mostra errori irrecuperabili, spesso di difficile diagnosi, ma altrettanto spesso dovuti alla fragilità di qualche comando.

Il comando `\\` Serve sia per andare a capo durante la composizione del testo normale, eventualmente contenuto dentro ambienti che specificano il tipo di composizione (testi in display), sia nella composizione di tabelle o matrici, quando si vuole terminare una riga di quella struttura. I due usi possono entrare in collisione, specialmente nelle colonne definite con i descrittori di colonna `p`, `m` o `b`. Per evitare che un comando `\\` usato con il testo che deve venire composto in una colonna `p`, `m` o `b` di una tabella possa venire male interpretato come comando relativo alla tabella, basta racchiudere l'intero contenuto della cella dentro un gruppo formato dalla coppia di parentesi graffe `{...}`; alternativamente si può usare il comando `\newline`; meglio ancora usare `\tabularnewline` quando si vuole terminare una riga di celle in un ambiente *tabular* o *array*.

[1] È tipico il fatto che se si invia all'indice analitico un termine usando gli stessi comandi, ma una volta lo si fa nel testo e una volta in una nota, nell'indice analitico quel termine può apparire due volte con indicazioni di pagine diverse; è frutto delle operazioni che vengono svolte sugli argomenti fragili.

Quando LaTeX compone un capoverso, si eviti di usare due comandi \\ di seguito; si usi invece l'argomento facoltativo per aggiungere altro spazio; la sintassi del comando, lo si ricorda, è:

```
\\[⟨spazio⟩]
\\*[⟨spazio⟩]
```

dove la versione asteriscata impedisce un salto pagina prima della nuova riga; a causa di queste diverse modalità di andare a capo, il comando \\ è fragile.

Gli ambienti (anche nelle versioni asteriscate) e i comandi che accettano il comando \\, a parte il testo normale, sono i seguenti:

`\shortstack`	`\author`	`\title`	`\date`
eqnarray	*tabbing*	*array*	*tabular*

Gli ambienti *array* e *tabular* sono stati messi in evidenza perché, insieme con gli altri ambienti per comporre materiale tabulare, bisogna evitare che il segno di 'a capo' \\ del testo possa essere frainteso per un comando di fine riga \\ della tabella; questa possibile confusione è particolarmente insidiosa se la parte testuale viene composta con una delle dichiarazioni `\centering`, `\raggedright` o `\raggedleft`. Per evitare di confondere il significato di \\, si suggerisce di usare per l'*a capo* testuale il comando `\newline` oppure di finire una riga testuale all'interno di una cella con `\tabularnewline`.

Inoltre, se si specifica uno ⟨*spazio*⟩ come argomento di \\ all'interno di una tabella o di una matrice, e se questo è troppo piccolo, potrebbe non manifestarsi nessun effetto visibile, perché il piccolo spazio viene assorbito dalla profondità dello `\strut` (*pilastrino*[2]) che compare in ogni riga delle tabelle e delle matrici.

[2] Come suggerisce il nome, il *pilastrino* è un oggetto tipograficamente invisibile che garantisce una spaziatura minima fra due righe successive. Quando si componeva in piombo, il pilastrino era appunto un distanziatore – da non inchiostrare – per distanziare due righe di caratteri.

1.1 La struttura del file sorgente

```
\begin{filecontents}{⟨nome file⟩}
⟨contenuto del file⟩
\end{filecontents}
...
\documentclass[⟨lista di opzioni⟩]{⟨classe⟩}
⟨preambolo⟩
\begin{document}
⟨testo del documento⟩
\end{document}
```

La dichiarazione `\documentclass` può essere preceduta da uno o più ambienti *filecontents* che servono, in particolar modo, quando si inviano i file sorgente ad altri collaboratori, per specificare i pacchetti non standard da cui dipende la composizione del documento. Questi file vengono estratti dal file sorgente la prima volta che questo viene elaborato dal coautore e memorizzati con il loro ⟨*nome file*⟩ nella cartella corrente, cosicché quando la compilazione comincia per davvero, LaTeX dispone di tutti i file di cui ha bisogno, compresi, appunto, i file non standard allegati al file sorgente.

1.2 Periodi e capoversi

Si ricorda che "periodo" indica una frase di senso compiuto che inizia con una lettera maiuscola e termina con un punto fermo; "capoverso" indica una struttura formata generalmente di diversi periodi; quasi sempre la prima riga di un capoverso è rientrata e il punto finale dell'ultimo periodo è in realtà un "punto e a capo". Il capoverso in inglese si chiama *paragrah*; è molto importante non confondere la parola inglese con "paragrafo", che in italiano indica una sezione di testo preceduta da un titolino, spesso numerato; senza ulteriori specificazioni è il principale sezionamento di un capitolo, ma spesso si usa questa parola anche per sezionamenti subalterni del paragrafo vero e proprio.

1.2.1 Periodi

Apostrofi e virgolette

apostrofo: `'` ⟶ l'apostrofo
virgolette semplici rialzate: `‘`⟨*testo*⟩`’` ⟶ ‘testo’
virgolette doppie rialzate: `‘‘`⟨*testo*⟩`’’` ⟶ “testo”
virgolette caporali[3]: `"<`⟨*testo*⟩`">` ⟶ «testo»

Lineati

lineetta: `-`⟶ -
lineato medio per intervalli numerici: `--`⟶ –
lineato lungo per gli incisi e per i dialoghi: `---` ⟶ —

Spazi

spazio fine: `\,`
spazio interparola: `\␣`;
spazio interparola che impedisce di andare a capo: `~`
modificatore dello spazio dopo il punto di fine periodo: `\@`

Il comando `\@` inserito dopo una lettera maiuscola e prima di un punto fermo serve per evitare che questo sia scambiato per un punto di abbreviazione. L'interprete, infatti, ritiene che un punto dopo una lettera minuscola sia un punto di fine periodo, mentre un punto dopo una lettera maiuscola indichi una abbreviazione. Si osservi la spaziatura quando si scrive M. Rossi e quando, invece, si scrive m. rossi. In questo secondo caso, lo spazio che segue m. dovrebbe essere due volte più grande dello spazio che segue M. e comunque circa il doppio dello spazio interparola. Si confrontino ancora le due frasi:

Vado all'ufficio postale a pagare l'IMU. Poi torno a casa.
Vado all'ufficio postale a pagare l'IMU. Poi torno a casa.

Nel secondo caso *non* è stato inserito il comando `\@` dopo la parola IMU. La differenza è minima, ma un occhio esercitato la rileva. È per questo che nella composizione della bibliografia, dove le abbreviazioni sono numerose, si usa il prossimo comando `\frenchspacing` che rende uguali tutti gli spazi

[3] In verità, per le virgolette caporali sarebbe sufficiente usare le legature `<<` e `>>`; siccome in italiano non deve essere lasciato nessuno spazio dopo i caporali aperti e prima dei caporali chiusi, la scrittura `"<` e `">` assicura l'eliminazione dello spazio inavvertitamente inserito nel file sorgente.

dopo i segni di interpunzione, in modo che non ci sia da preoccuparsi se un punto indichi una abbreviazione o un punto fermo.

`\frenchspacing` serve per rendere uguali gli spazi dopo qualsiasi segno di interpunzione.

`\nonfrenchspacing` serve per aumentare gli spazi dopo i punti di interpunzione che marcano la fine della frase o del periodo, in particolare dopo i punti fermi. Questa è la situazione di default; secondo alcuni stimati book designer, la spaziatura alla francese sarebbe da evitare sempre; secondo altri dovrebbe essere l'unica ad essere usata. Si noti che questo intero documento è stato composto con la dichiarazione `\frenchspacing`.

Segni speciali

^	`\textasciicircum`	%	`\%`	{	`\{`	}	`\}`
~	`\textasciitilde`	&	`\&`	#	`\#`	_	`\_`
\	`\textbackslash`	$	`\$`				

Anche altri segni svolgono funzioni speciali oltre a quelle che uno si aspetterebbe; `'` seguito da cifre comprese fra 0 e 7 serve per indicare che il numero indicato da quella sequenza di cifre è da intendersi scritto in base ottale. In italiano, come in quasi tutte le altre lingue, le doppie virgolette diritte `"` si comportano come un carattere attivo che svolge diverse funzioni ma, se non sono attive o se vengono disattivate, esse possono precedere una stringa di cifre da 0 a 9, magari intercalate con le lettere maiuscole da A a F, e indicano che la stringa rappresenta un numero in base esadecimale. Quando, fuori del contesto matematico, il carattere speciale `^` è raddoppiato e seguito da una sola lettera maiuscola da A a Z o da pochi altri caratteri indica un codice ASCII compreso fra `00` e `31`; invece, quando è raddoppiato ed è seguito da una stringa di cifre da 0 a 9 magari intercalate con le lettere minuscole da 'a' a 'f', indica il carattere che nel font corrente ha per indirizzo quel valore esadecimale; con XeLaTeX e LuaLaTeX, che possono gestire anche i caratteri UNICODE con indirizzi formati da diverse cifre esadecimali, il carattere `^` può venire ripetuto tante volte quante sono le cifre esadecimali minuscole che seguono la serie di `^`. È molto raro che questi caratteri (apice, doppio apice e circonflesso) vengano usati durante il

normale uso di LaTeX, ma possono venire usati dall'utente avanzato quando si definisce delle macro personali particolari.

Loghi

`\LaTeX`	Logo di LaTeX
`\LaTeXe`	Logo di LaTeX 2_ε
`\TeX`	Logo di TeX

L'uso del pacchetto `metalogo` permette di scrivere anche altri loghi; fra gli altri quelli di XeTeX e XeLaTeX.

`\today` produce la data del giorno.

`\emph{`⟨*testo*⟩`}` Evidenzia il ⟨*testo*⟩ scrivendolo in corsivo all'interno di un contesto in tondo e in tondo all'interno di un contesto corsivo.

`\mbox{`⟨*testo*⟩`}` scrive il ⟨*testo*⟩ all'interno di una scatola orizzontale in modo che non possa, per esempio, essere divisa in sillabe in fin di riga; questo comando può essere usato anche in un contesto matematico, ma di fatto, se non ci sono motivi speciali, in matematica si possono usare sia i comandi di scelta dei font da usare in un contesto testuale, per esempio `\textrm`, rinunciando però alla prerogativa delle matematica di scegliere il font della giusta dimensione, oppure i comandi `\text` oppure `\intertext` messi a disposizione dal pacchetto `amsmath`; usando questo pacchetto, i comandi `\textit`, `\textsl`, eccetera, si possono usare anche in matematica e vengono rispettati i corpi dei pedici e degli apici.

1.2.2 Capoversi

Un capoverso si può iniziare o terminare con uno dei seguenti comandi; in ogni caso una o più righe assolutamente vuote nel file sorgente, senza neanche i segni di commento, servono per terminare un capoverso.

`\noindent` comincia un capoverso senza inserirne il tipico rientro.

`\indent` comincia un capoverso con il tipico rientro, anche quando normalmente esso non sarebbe inserito.

`\par` termina un capoverso e non è più necessario lasciare altre righe bianche.

I capoversi vengono composti secondo le seguenti giustezze:

`\textwidth` è la giustezza normale fissata dal file di classe; se la si vuole cambiare, bisognerebbe farlo solo nel preambolo.

`\linewidth` è la giustezza corrente; può essere minore o maggiore del valore generale `\textwidth` a seconda dell'ambiente all'interno del quale si sta componendo.

`\columnwidth` è la giustezza di una colonna, quando si sta componendo su più colonne.

`\columnsep` è lo spazio di separazione fra due colonne adiacenti.

`\columnseprule` è lo spessore del filetto verticale che separa le colonne quando si compone su due o più colonne; se questo spessore è nullo (default) il filetto è invisibile.

`\parindent` è l'ammontare del rientro del capoverso.

`\baselineskip` è l'avanzamento di riga, o *scartamento*, stabilito con la scelta del corpo del font che si sta usando, eventualmente modificato dal comando seguente.

`\linespread{`⟨*fattore*⟩`}` modifica l'avanzamento di riga mediante il coefficiente moltiplicativo ⟨*fattore*⟩; per attivare questa specifica deve essere impartito il comando `\selectfont` in modo esplicito o implicito (è implicito se si specifica un altro comando di cambiamento di font).

`\lineskip` interlinea aggiuntiva da aggiungere fra due righe adiacenti per evitare che interferiscano l'una con l'altra sovrapponendosi in parte.

`\lineskiplimit` valore minimo della distanza fra due righe successive, scendendo sotto il quale TEX introduce interlinea supplementare pari a `\lineskip`.

`\parskip` rappresenta uno spazio di interlinea supplementare da inserire prima di ogni nuovo capoverso; generalmente questo spazio è nullo; talvolta è rappresentato da una *lunghezza elastica* (vedi più avanti nel paragrafo 1.7.6) dotata di un valore naturale nullo, ma di un piccolo allungamento, talvolta anche di un piccolo accorciamento, che consente di giustificare le pagine affacciate anche se contengono un numero di righe leggermente diverso. Questo normalmente succede quando, per esempio, una formula di una certa dimensione verticale o un nuovo paragrafo deve cominciare verso la fine di una pagina ma, se non c'è abbastanza posto, l'oggetto viene spostato alla pagina successiva. La pagina rimarrebbe 'mozza' e, per evitare questo ine-

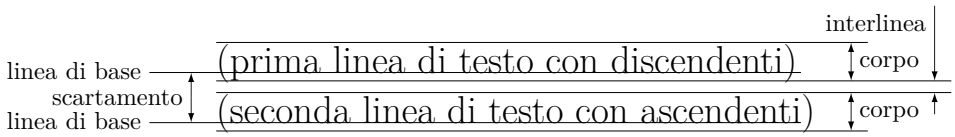

Figura 1.1 Corpo, interlinea, scartamento, linea di base

stetismo, il programma di composizione sfrutta l'elasticità di questi contrografismi per distribuire le righe della pagina in modo che le due pagine affacciate terminino allo stesso livello.

`\enlargethispage{`⟨*lunghezza*⟩`}` allunga la pagina della ⟨*lunghezza*⟩ specificata; per ovvi motivi conviene che questa lunghezza sia specificata con un numero intero e piccolo (1 o 2) di righe di testo che, in fondo alla pagina, richiedono un numero intero di avanzamenti di riga. Il comando asteriscato `\enlargethispage*{`⟨*lunghezza*⟩`}` prima di allungare la pagina cerca di comprimere gli spazi verticali presenti nella pagina stessa al fine di farci stare più materiale; in generale è più conveniente usare il comando asteriscato. Componendo a due colonne questo comando si riferisce solo alla colonna nel quale si trova. Componendo a due colonne `\enlargethispage` con un argomento negativo permette di accorciare una colonna di sinistra tanto da poterla "pareggiare" con la colonna adiacente di destra, se si sa introdurre il giusto numero di righe con cui accorciare la colonna. Componendo a una colonna, `\enlargethispage` permette di accorciare o allungare una colonna in modo da eliminare la presenza di righe vedove o di righe orfane (vedi quanto descritto nel paragrafo 1.14.3 nella pagina 116).

Tutta questa terminologia deve essere ben chiara; è meglio mostrare una figura dalla quale si evinca senza possibilità di dubbio che cosa significano le parole scartamento (`\baselineskip`), interlinea (`\lineskip`), soglia per l'interlinea (`\lineskiplimit`) in relazione al corpo (*body*) e agli ascendenti (`\lineheight`) e i discendenti (`\linedepth`). Si faccia quindi riferimento alla figura 1.1 e si dovrebbe anche capire perché interlinea e scartamento vogliono dire due cose diverse, anche se l'uso della parola interlinea per designare lo scartamento è diffusissimo, tanto che è anche utilizzata (purtroppo) anche nei word processor con l'interfaccia localizzata in italiano.

1.2.3 Note in calce

`\footnote[`⟨*numero*⟩`]{`⟨*testo*⟩`}`

Se viene specificato il ⟨*numero*⟩ non vi si può fare riferimento in modo simbolico attraverso i comandi `\ref`, `\label`, eccetera. Il contatore `footnote` viene rappresentato con un numero a esponente, tranne che nelle note interne all'ambiente *minipage* dove le note sono numerate con lettere minuscole, sempre a esponente. In certe circostanze è possibile separare il comando che inserisce il richiamo della nota dal comando che effettivamente la scrive:

`\footnotemark[`⟨*numero*⟩`]`
...
`\footnotetext[`⟨*numero*⟩`]{`⟨*testo*⟩`}`

Per i parametri stilistici di composizione si veda il prossimo paragrafo.

1.2.4 Note marginali

`\marginpar[`⟨*nota a sinistra*⟩`]{`⟨*nota a destra*⟩`}`

La ⟨*nota a sinistra*⟩ serve facoltativamente per specificare un testo con una modalità di composizione diverso da quello della ⟨*nota a destra*⟩.

I parametri stilistici con cui vengono composte le note in calce e quelle marginali sono le seguenti.

`\footnotesep` è una lunghezza che rappresenta l'altezza dello `\strut`[4] messo all'inizio di ogni nota; aumentando questa altezza, ogni nota viene staccata di più dalla precedente.

`\footnoterule` è il comando con il quale vengono collocati nella lista verticale di oggetti che compongono la pagina il filetto orizzontale e gli spazi adiacenti, in modo da marcare la separazione fra il testo e le note in calce. Complessivamente, il filetto e gli spazi adiacenti devono avere una altezza totale nulla, quindi bisogna usare anche dello spazio

[4] Continuo ad usare il nome e il comando `\strut`, che in italiano significa *pilastrino*, perché il lettore si abitui a questo comando che può usare anche in altre circostanze; per impostazione predefinita, lo `\strut` di LaTeX ha l'altezza e la profondità di una parentesi nel font corrente, ma chiunque può usare un pilastrino definito come un `\rule` di altezza e profondità specificate e larghezza nulla; per il comando `\rule` si veda più avanti.

'negativo'; infatti, la definizione di default di questo comando è la seguente:

```
\def\footnoterule{\kern-3\p@ \hrule \@width 2in \kern 2.6\p@}
```

Benché sia scritto con comandi protetti di basso livello, si capisce che prima si arretra di 3 pt, poi si inserisce un filetto lungo due pollici e spesso 0,4 pt (questo è il valore di default), e infine si avanza di 2,6 pt: si constata che $-3\,\text{pt} + 0,4\,\text{pt} + 2,6\,\text{pt} = 0\,\text{pt}$. Si può ridefinire questo comando con `\renewcommand`, ma la definizione deve sempre corrispondere a una serie di oggetti la cui altezza complessiva sia nulla.

`\marginparsep` è la larghezza dello spazio che separa le note marginali dal testo a cui sono affiancate.

`\marginparpush` è la distanza minima fra due note marginali consecutive.

`\marginparwidth` è la giustezza con cui sono composte le note marginali; va da sé che il margine dove deve comparire la nota deve essere sufficientemente largo e deve tenere conto anche di `\marginparsep` e di un buono spazio verso il taglio della pagina affinché, dopo il taglio, non si debba scoprire che anche parte della nota è stata tagliata.

`\reversemarginpar` è una istruzione booleana che imposta lo scambio del margine nel quale vengono composte le note marginali; agisce solo quando si compone a una sola colonna, perché a due colonne le note marginali della colonna di sinistra vengono composte comunque a sinistra, e quelle relative alla colonna di destra alla loro destra.

`\normalmarginpar` ristabilisce la composizione delle note marginali nel margine esterno.

1.2.5 Accenti e simboli speciali

Premesso che con l'uso del pacchetto `inputenc` (a cui sia specificata un'opportuna codifica di entrata – si consiglia sempre la codifica *utf8*) e della tastiera nazionale, la necessità di ricorrere ai comandi per gli accenti è fortemente diminuita, è tuttavia utile saper gestire i comandi più insoliti al fine di comporre correttamente anche parole in lingue straniere che fanno uso di segni che non compaiono sulla tastiera nazionale.

I comandi per gli accenti testuali sono raccolti nella tabella 1.1, mentre quelli per il contesto matematico sono raccolti nella tabella 1.3.

Tabella 1.1 I comandi per gli accenti testuali e i simboli speciali di molte lingue

`\‘{a}`	à	`\’{a}`	á	`\={a}`	ā
`\v{a}`	ǎ	`\^{a}`	â	`\"{a}`	ä
`\~{a}`	ã	`\u{a}`	ă	`\.{a}`	ȧ
`\H{a}`	a̋	`\t{oo}`	o͡o	`\c{t}`	ţ
`\d{a}`	ạ	`\b{a}`	a̲	`\r{u}`	ů
`\i`	ı	`\j`	ȷ	`\oe`	œ
`\OE`	Œ	`\ae`	æ	`\AE`	Æ
`\aa`	å	`\AA`	Å	`\o`	ø
`\O`	Ø	`\l`	ł	`\L`	Ł
`\ss`	ß	`?‘`	¿	`!‘`	¡
`\dag`	†	`\ddag`	‡	`\S`	§
`\P`	¶	`\copyright`	©	`\pounds`	£

È estremamente importante evitare di usare i comandi per gli accenti matematici quando si scrive del testo e viceversa; il modo matematico di composizione non riconosce le lettere accentate come simboli validi per la matematica; analogamente, il modo testo non riconosce i comandi per i diacritici matematici che vengono presi da una delle tre raccolte di font che servono per comporre la matematica “principale”, i pedici e gli apici di primo livello e i pedici e gli apici di secondo livello; la selezione di questi corpi non ha senso in modo testo e LaTeX produce una serie di vigorose proteste se si viola questa regola di distinguere la matematica dal testo. Questa regola può venire (apparentemente) violata solo e soltanto se si usa il pacchetto `amsmath` e solo negli argomenti dei comandi testuali per la scelta delle serie e delle forme, come `\textrm`, `\textup`, `\textsc`, eccetera; questi comandi, infatti, servono per inserire piccole parti testuali in un contesto matematico, per esempio:

$$x \geq 0 \quad \text{solo e soltanto se è} \quad x \not< 0$$

1.3 Suddivisione del testo e indici

1.3.1 Comandi di sezionamento

Tutti i comandi di sezionamento propriamente detti, e gli altri comandi simili hanno una sintassi comune:

comando[⟨*titolo breve*⟩]{⟨*titolo lungo*⟩}

Essi sono:

```
\part            \chapter    \section       \subsection
\subsubsection   \paragraph  \subparagraph  \caption
```

Il comando `\chapter` non è definito per la classe *article*; nessun comando di sezionamento è definito per la classe *letter*. Il livello di sezionamento `\part` non è obbligatorio; tutti gli altri comandi devono essere contenuti sotto un comando del livello precedente.

Il comando `\caption` non è propriamente un comando di sezionamento ma ne condivide molte funzionalità. Questo comando si può usare solo all'interno degli ambienti flottanti come *figure* e *table* o altri che si possono definire con il pacchetto `float`; `\caption` condivide con i comandi di sezionamento il fatto di comporre un "titolo", la didascalia da apporre all'oggetto flottante, ma anche di inviare tutto o parte di questo titolo ai file ausiliari per comporre gli elenchi delle figure, gli elenchi delle tabelle, gli elenchi degli altri oggetti flottanti creati con il pacchetto `float`.

Le parole che eventualmente vengono scritte nelle intestazioni, come 'Part' o 'Parte', 'Chapter' o 'Chapître', vengono impostate tramite l'opportuna chiamata al pacchetto `babel` con le opzioni per la lingua del documento. In ogni caso, esse sono contenute nei comandi `\partname`, `\chaptername`, `\figurename`, `\tablename`, eccetera, per cui possono venire ridefinite a piacere.

Ogni comando di sezionamento, oltre a comporre l'intestazione della divisione specifica mediante il ⟨*titolo lungo*⟩, scrive negli opportuni file accessori il ⟨*titolo breve*⟩ da inserire nei corrispondenti indici e, con certe classi, nelle testatine.

Ogni divisione del documento può venire numerata e può comparire nell'indice; questi due fatti sono controllati dai valori numerici contenuti nei due contatori `secnumdepth` e `tocdepth`. Le sezioni sono numerate se

Sezionamento	Livello	Macro che contiene il nome
`\part`	−1	`\partname`
`\chapter`	0	`\chaptername`
`\section`	1	
`\subsection`	2	
`\subsubsection`	3	
`\paragraph`	4	
`\subparagraph`	5	
`\caption`	—	`\figurename` `\tablename`

Tabella 1.2 Numeri associati al livello di sezionamento

il loro livello è associato a un numero minore o uguale a quello contenuto in `secnumdepth`; i loro titoli vanno a finire nell'indice se il loro livello corrisponde a un numero minore o uguale a quello contenuto in `tocdepth`; si veda la tabella 1.2.

1.3.2 Appendici

Va notato che quando si vogliono inserire delle appendici nel documento, queste continuano a essere composte iniziando con il loro titolo, cioè con l'esecuzione del comando `\chapter`; ma perché l'intestazione non contenga più il nome Chapter o Capitolo, ma Appendix o Appendice, bisogna inserire il comando `\appendix` prima della prima appendice; da quel momento in poi, i comandi `\chapter` intitolano solo le appendici, le quali sono anche distinte da un "numero" formato da una lettera maiuscola, cominciando con A per la prima appendice. Quindi, il comando `\appendix` non serve per cominciare una nuova appendice, ma serve per cominciare la sezione delle appendici quasi fosse una parte distinta del documento.

Fra i *book designer* più accreditati non c'è uniformità di posizione circa il trattamento delle appendici; alcuni dicono che devono essere numerate, sia pure con un sistema di numerazione diverso, altri che non devono

essere numerate. LaTeX permette di seguire sia l'una sia l'altra posizione. Infatti bisogna ricordare che la classe standard `book`, così come altre classi specifiche, contiene i comandi di "sezionamento" `\frontmatter`, `\mainmatter` e `\backmatter` (vedi però quanto specificato nella pagina 28).

Per impostazione predefinita, `\frontmatter` è la sezione iniziale di un documento molto strutturato, come un libro, dove i capitoli e le loro sezioni non sono numerati, ma sono elencati nell'indice generale; inoltre, le pagine sono numerate con numeri romani minuscoli[5]. I comandi di sezionamento asteriscati, invece, continuano a non modificare né l'indice né le testatine.

Sempre per impostazione predefinita, `\mainmatter` numera i capitoli e le loro sezioni e ne inserisce i titoli nell'indice e nelle testatine, ma numera le pagine con cifre arabe ricominciando da 1. Infine `\backmatter` non numera i capitoli e le loro sezioni ma ne mette i titoli nell'indice e nelle testatine; non cambia la numerazione delle pagine impostata nella *main matter*, che prosegue in cifre arabe senza ricominciare da 1; inoltre, in un documento stampato fronte/retro, i capitoli e le appendici vengono composti sempre cominciando da una pagina di destra (dispari) quando si compone la *main matter*; invece nella parte finale (*back matter*) è consentito cominciare questi sezionamenti sia a destra sia a sinistra, e questa è l'impostazione predefinita di `\backmatter` (a meno che non si sia specificata l'opzione di classe *openright*).

Per le appendici questa è una questione cruciale; secondo i *book designer* favorevoli alla numerazione delle appendici, queste vanno composte alla fine della *main matter*; per gli altri *book designer* le appendici vanno composte all'inizio della *back matter*.

Un po' di coerenza non guasta. Se il documento contiene una sola appendice, è evidente che il suo 'numero' letterale non serve per distinguerla da altre appendici; tanto vale non numerarla, mettendola quindi all'inizio della *back matter* e senza farla precedere dal comando `\appendix`, visto che non è numerata. Se invece il documento contiene una serie di appendici, esse vanno distinte l'una dall'altra con la loro numerazione letterale; perciò vanno composte alla fine della *main matter* facendole precedere dal comando `\appendix`.

[5] Siccome i romani non conoscevano le lettere minuscole, sarebbe preferibile che la numerazione romana fosse composta in minuscoletto, invece che in tondo minuscolo; a questo si può provvedere ridefinendo lo stile di composizione delle pagine.

Questa soluzione di buon senso può rendere difficile la composizione di una sola appendice che contenga oggetti numerati, come figure, equazioni, tabelle, e simili. In questi casi, il problema va risolto ridefinendo il modo di numerare questi oggetti. LaTeX ha tutti i mezzi necessari per eseguire queste ridefinizioni, ma tocca al compositore provvedere, perché LaTeX non lo fa in automatico.

1.3.3 Indici

Per produrre gli indici si usano i comandi:

`\tableofcontents`	`\listoffigures`	`\listoftables`

Alcune classi non standard, come per esempio *memoir*, dispongono anche delle versioni asteriscate che differiscono dalle versioni non asteriscate per non inserire nulla nell'indice generale. In sostanza, l'uso dell'asterisco è conforme a quanto succede con i comandi di sezionamento. Tuttavia è bene ricordare che questa è un proprietà di *memoir*. Le classi standard dispongono solo dei comandi non asteriscati, i quali compongono gli indici per i quali sono stati creati, ma non inseriscono nulla nell'indice generale; sarebbe un po' buffo che l'indice generale contenesse una riga per informare il lettore che lo sta leggendo dove si trova l'indice generale...

Le informazioni per produrre questi indici sono contenute in tre file ausiliari le cui estensioni sono rispettivamente `.toc`, `.lof` e `.lot`. Per gli oggetti flottanti costruiti con il pacchetto `float` le estensioni delle rispettive liste sono definiti dallo stesso pacchetto `float`

Le versioni asteriscate dei comandi di sezionamento producono solo l'intestazione non numerata e non producono niente nell'indice, né nelle testatine. Il comando `\caption` non possiede la variante asteriscata, ma una tale variante è definita all'interno nell'ambiente *longtable*. Altri pacchetti, per esempio `caption` o `capt-of`, permettono di disporre del comando `\captionof` che richiede un primo argomento contenente il nome di un oggetto flottante, oltre al secondo argomento riservato alla didascalia vera e propria; in questo modo si possono scrivere didascalie simili a quelle degli oggetti flottanti; si veda la documentazione dei pacchetti citati.

Sebbene le versioni asteriscate non producano nessuna voce nei file indice, tuttavia le si può far comparire se si procede a mano inserendo

dopo i comandi espliciti asteriscati il comando `\addcontentsline` con la seguente sintassi:

`\addcontentsline{`⟨*estensione*⟩`}{`⟨*livello*⟩`}{`⟨*voce*⟩`}`

dove ⟨*estensione*⟩ è l'estensione del file ausiliario per l'indice specifico, ⟨*livello*⟩ è il nome (non il numero) del livello di sezionamento, espresso dallo stesso nome del comando, senza però il backslash; la ⟨*voce*⟩ è quanto viene scritto nel file indice. Generalmente, quanto viene scritto nell'indice è costituito dall'espressione

`\protect\numberline{`⟨*numero*⟩`}{`⟨*titolo*⟩`}`

dove ⟨*numero*⟩ è il numero della sezione, per esempio 3.8 per l'ottavo paragrafo del capitolo tre; va da sé che una sezione non numerata ha questo campo vuoto. Il ⟨*titolo*⟩ non è altro che il titolo (breve) della sezione.

Per le sezioni non numerate, lasciare vuoto il campo riservato al ⟨*numero*⟩ implica che il ⟨*titolo*⟩ viene incolonnato con gli altri titoli delle sezioni numerate. Se invece si omette tutta la parte `\protect\numberline{`⟨*numero*⟩`}`, allora il ⟨*titolo*⟩ viene allineato al margine sinistro.

Lo stesso effetto di non numerare le sezioni da un certo livello in poi ma di metterle nell'indice, si può ottenere senza nessun artificio usando i comandi `\frontmatter` e `\mainmatter` con le classi che ne dispongono. Fra le classi standard *report* e *article* non dispongono di tali comandi; per altre classi standard, non solo mancano i comandi di sezionamento, ma non ha nemmeno senso di parlare di indice, per esempio con la classe *letter*.

Come verrà spiegato più avanti, i comandi `\frontmatter`, `\mainmatter` e `\backmatter` sono dei particolari comandi di sezionamento che impostano diverse cose per un documento complesso; per esempio `\frontmatter` imposta la numerazione romana per le pagine iniziali; `\mainmatter` imposta la numerazione araba che ricomincia da 1. `\backmatter` non influisce sulla numerazione delle pagine, ma ha in comune con `\frontmatter` che entrambe queste parti del documento hanno sezioni non numerate, anche se vanno nell'indice; perciò in queste due parti preliminare e finale non è necessario, anzi è raccomandabile *non usare* i comandi di sezionamento asteriscati, che correttamente non numerano le rispettive sezioni, ma non mettono nemmeno nulla nell'indice.

Se si vuole simulare questo comportamento con classi che non dispongono di queste particolari dichiarazioni `\frontmatter`, `\mainmatter` e `\backmatter`, si possono dichiarare esplicitamente i contenuti dei contatori `secnumdepth` e `tocdepth` specificando:

```
\setcounter{secnumdepth}{-2}
\setcounter{tocdepth}{1}
```

dove con il primo statement si toglie la numerazione a qualunque comando di sezionamento, ma con il secondo si consente l'inserimento nell'indice dei titoli (brevi) dei comandi di sezionamento fino al livello '1', quello della "section", cioè del paragrafo.

Per modificare il contenuto di un indice con informazioni che non vengono raccolte mediante i comandi di sezionamento e dai comandi per generare le didascalie, si può usare il comando `\addtocontents` con la seguente sintassi:

`\addtocontents{`⟨*estensione*⟩`}{`⟨*materiale da aggiungere*⟩`}`

Questo comando talvolta è usato erroneamente al posto di `\addcontentsline` e viceversa. Bisogna quindi fare molta attenzione a non confonderli. L'⟨*estensione*⟩ da usare è la stessa di quella che si usa per aggiungere una voce a un determinato indice. Invece, il ⟨*materiale da aggiungere*⟩ può essere sia del testo, sia dei comandi da usare in modo verticale. Internamente viene usato dal nucleo di LaTeX per aggiungere spazio verticale prima delle voci che indicano la posizione dei capitoli e delle parti. Anche l'utente può servirsene, ma deve sapere bene che cosa sta aggiungendo e che, se il materiale aggiunto contiene dei comandi fragili o li protegge con `\protect`, va incontro a messaggi d'errore criptici.

Per l'indice analitico e il glossario, i vari elementi vengono raccolti mediante i rispettivi comandi `\index` e `\glossary`, ma il contenuto del loro argomento varia molto a seconda di come si intende usare l'informazione raccolta. In ogni caso, nessuna informazione viene raccolta se nel preambolo mancano i comandi `\makeindex` e, rispettivamente, `\makeglossary`; l'indice analitico e/o il glossario non vengono composti se non si dà esplicitamente il comando di comporli, di solito attraverso comandi messi a disposizione da pacchetti esterni. Altrimenti bisogna usare gli ambienti *theindex* e *theglossary* o specificando a mano il nome dei file che contengono

l'indice o il glossario già elaborati adeguatamente; si veda più avanti nella pagina 34 la breve descrizione del programma makeidx e di altri programmi analoghi che permettono di fare questa elaborazione preliminare).

1.4 Classi, pacchetti e stili delle pagine

Ogni documento ha una sua tipologia specifica che in LaTeX si chiama *classe*; LaTeX quindi dispone di molti file di classe che definiscono la struttura generale di ogni documento di una certa tipologia. LaTeX dispone anche di altri file raggruppati in *pacchetti*, formati generalmente da un file principale e da un numero variabile di file subalterni (anche nessuno), con i quali è possibile modificare le impostazioni della classe. Fra le impostazioni più importanti ci sono quelle che riguardano lo *stile* delle pagine.

1.4.1 Classe del documento

Le classi standard sono `book`, `report`, `article`, `letter`, `slides`, `minimal`, `proc`, `ltxdoc`; queste sono affiancate dalle numerosissime altre classi non standard che popolano gli archivi internazionali.

Le classi `book` serve per comporre libri. La classe `report` serve per comporre documenti chiamati "rapporti" che sovente contengono informazioni limitate ad argomenti molto specifici. La classe `article` serve per comporre articoli o comunque documenti brevi e poco strutturati. La classe `letter` serve per comporre lettere. La classe `slides` serviva per comporre le pagine di una presentazione da riprodurre mediante le vecchie "lavagne luminose" (epidiascopi), oggi non più in uso. La classe `minimal` è un canovaccio minimo per comporre un documento usato per collaudare le funzionalità di nuove macro definite dall'utente. La classe `proc` può servire per raccogliere diversi articoli e pubblicare gli atti (*proceedings*) di un convegno. La classe `ltxdoc` serve per comporre la documentazione dei file di classe o dei pacchetti in una forma dove il codice e il testo di documentazione sono trattati diversamente, al fine di poter estrarre il solo codice e disporre dei file di classe o dei pacchetti.

Le prime tre classi sono le più usate. Per la classe `letter` ci sono numerose altre classi non standard che possono comporre le lettere meglio di come lo fa questo file di classe. Le altre classi indicate sopra sono usate raramente o solo per scopi molto particolari.

La scelta della classe va eseguita con il comando:

`\documentclass[`⟨*opzioni*⟩`]{`⟨*classe*⟩`}`

Le opzioni valide per quasi tutte le classi standard sono le seguenti.

10pt, definisce il corpo di default per il documento; però può venire trasmessa anche ad altri pacchetti fra i quali `type1ec` il quale con questa opzione ricava i disegni di tutti i caratteri dai font in corpo 10 pt, ingrandendoli o rimpicciolendoli a seconda delle richieste del compositore. Questo modo di procedere è accettabile solo quando si sta lavorando con font vettoriali. Il file di uscita risulta meno ingombrante, la qualità peggiora un poco, e questo effetto è visibile ad occhio nudo. Tra l'altro si noti la differenza di resa dei font ingranditi o rimpiccioliti; qui si mostra lo stesso testo riportato al corpo 10 partendo da un font di corpo 5 e da un font di corpo 17: partendo da un font di corpo 5 e partendo da un font di corpo 17. L'effetto opposto si otterrebbe partendo da un font di corpo 10 riducendolo al corpo 5 o ingrandendolo al corpo 17.

11pt, *12pt*, servono per scegliere il corpo da 11 pt o da 12 pt come corpo normale.

letterpaper, *legalpaper*, *executivepaper*, *a4paper*, *a5paper*, *b5paper*, servono per scegliere il supporto della carta secondo i formati specificati con le sigle suddette, precisamente:

letter	$8,5\,\text{in} \times 11\,\text{in}$	A4	$210\,\text{mm} \times 297\,\text{mm}$
legal	$8,5\,\text{in} \times 14\,\text{in}$	A5	$148\,\text{mm} \times 210\,\text{mm}$
executive	$7,25\,\text{in} \times 10,5\,\text{in}$	B5	$176\,\text{mm} \times 250\,\text{mm}$

Siccome l'opzione di default è *letterpaper*, per scrivere in Europa è sempre necessario specificare un formato ISO, quasi sempre *a4paper*. Le dimensioni così specificate vengono passate anche al file `.pdf` nel caso che si usi pdfLaTeX, in quanto quel formato ha bisogno di conoscere le dimensioni del supporto di uscita. Il file `.dvi` a stretto rigore non ne ha bisogno, ma poi diventa necessario se questo tipo di file viene convertito nei file `.ps` oppure `.pdf`.

landscape, serve per scambiare fra di loro la larghezza e l'altezza del supporto.

final, *draft*, servono per specificare se si sta componendo la versione finale oppure una bozza; quando si compone una bozza, vengono evidenziate le righe fuori giustezza con un vistoso rettangolo nero e non vengono importate le figure, ma al loro posto viene segnato un rettangolo delle giuste dimensioni contenente al suo interno solo il nome del file grafico.

oneside, *twoside*, servono per specificare se si vuole comporre solo sul recto o anche sul verso della carta.

openright, *openany*, servono per specificare se i capitoli possono iniziare solo sulle pagine di destra oppure su qualunque pagina.

onecolumn, *twocolumn*, servono per specificare se si vuol comporre su un'unica colonna o su due colonne.

notitlepage, *titlepage*, servono per specificare se il titolo del documento va in testa alla prima pagina, nella quale comincia subito anche il testo, oppure se si deve riservare una pagina solo per il titolo.

openbib, serve per comporre la bibliografia (nelle classi che la riconoscono) con uno stile più aperto; generalmente le varie parti del riferimento bibliografico sono trattate come dei brevi capoversi senza rientro; però le loro righe dopo la prima risultano composte con un rientro, rispetto al margine sinistro, specificato mediante il parametro `\bibindent`.

leqno, serve per specificare che si desidera numerare le equazioni con il numero collocato a filo del margine sinistro della gabbia del testo. Se anche le equazioni vengono allineate a sinistra, potrebbe non esserci abbastanza spazio.

fleqn, serve per specificare che non si desiderano le equazioni centrate, ma si desidera che siano rientrate di una quantità fissa dal margine sinistro; questa quantità viene specificata mediante `\mathindent`, che di default è impostata al valore della rientranza del margine sinistro delle liste di primo livello. Forse è un po' poco, specialmente se i numeri identificativi delle equazioni (che possono diventare relativamente lunghi quando esse sono numerose) con l'opzione precedente ècollocato al margine sinistro dello specchio di stampa; si può sempre modificare `\mathindent` con il comando `\setlength`. Va notato, però, che con il valore di default di `\mathindent` non vengono introdotti altri valori di rientranza dei margini, e questo tipograficamente va bene, ma con numerose equazioni potrebbe verificarsi l'interferenza dell'etichetta dell'equazione con l'equazione stessa. Il pacchetto `amsmath` evita

sempre questa interferenza, perciò direi che sia meglio usare `amsmath` piuttosto che modificare `\mathindent`.

1.4.2 Pacchetti

La scelta del pacchetto si esegue con il comando `\usepackage` con la sintassi:

`\usepackage[`⟨*opzioni*⟩`]{`⟨*pacchetto*⟩`}[`⟨*data*⟩`]`

dove ⟨*data*⟩ è una data nella forma (*aaaa/mm/gg*) che rappresenta il limite inferiore della data corrispondente alla versione che si vuole usare; negli oltre 20 anni, da quando esiste LaTeX 2_ε, i vari pacchetti sono stati aggiornati così che le versioni successive non solo hanno (presumibilmente) meno errori e *funzionalità strane* degli stessi pacchetti nelle loro versioni precedenti, ma anche più funzionalità utili. Ecco che, specificando la data, si ha il vantaggio di ricevere un avviso del fatto che il pacchetto in uso non è aggiornato. Questo è il motivo per il quale qui si raccomanda spesso di usare una installazione completa e aggiornata del sistema TeX. I pacchetti standard sono:

`alltt` serve per definire l'ambiente *alltt* dove quasi tutti i caratteri speciali perdono il loro significato specifico e vengono usati come caratteri 'normali'. Questo ambiente è simile all'ambiente *verbatim*, ma i caratteri speciali `\`, `{` e `}` conservano le loro funzioni speciali.

`amsmath` costituisce la principale estensione per gli ambienti matematici; conviene sempre caricarlo assieme al pacchetto `amssymb`, che permette di accedere ad altri due alfabeti matematici i quali arricchiscono enormemente la dotazione di simboli matematici disponibili con LaTeX; vedi più avanti le tabelle 1.11 e 1.12.

`babel` consente di comporre in diverse lingue; attenzione: `babel` sceglie le specifiche della lingua, compresa la sillabazione, ma non è `babel` che carica la sillabazione; vedi la guida GuIt (Beccari, 2018) per una descrizione approfondita dell'algoritmo di sillabazione.

`color` serve per gestire i colori dei font e degli sfondi. Spesso è più conveniente usare il pacchetto esteso `xcolor` che a sua volta è costruito sopra `color`; non è quindi necessario invocarli entrambi.

`graphics` serve per gestire le manipolazioni grafiche. Benché sia il pacchetto che svolge il lavoro richiesto, per l'utente è più comodo riferirsi al pacchetto seguente.

`graphicx` gestisce le operazioni grafiche e gli effetti speciali con una interfaccia utente più efficace; in realtà è `graphicx` che si preoccupa di caricare `graphics` e di passargli i comandi tradotti nella sua sintassi specifica.

`ifthen` permette di eseguire un fine controllo logico con una interfaccia utente più semplice di quella che offrono i comandi nativi di TeX.

`latexsym` consente di usare i simboli speciali di LaTeX 2.09; questo pacchetto è disponibile più per compatibilità con il passato che altro; tutti quei simboli sono accessibili anche con il pacchetto `amssymb`; usando LaTeX 2_ε è decisamente sconsigliabile usare simultaneamente `latexsym` e `amssymb`, ma conviene usare solo quest'ultimo pacchetto.

`makeidx` serve per gestire comodamente la creazione dell'indice analitico. In alternativa si può usare uno dei pacchetti `imakeidx` oppure `indextools`, che gestiscono anche la produzione di indici analitici multipli e consentono di sfruttare la proprietà dei sistemi moderni di lanciare comandi di sistema, cosicché gli indici analitici possono essere prodotti direttamente con una sola esecuzione di pdflatex (o degli altri programmi XeLaTeX e LuaLaTeX).

`pict2e` estende le capacità grafiche dell'ambiente *picture* di LaTeX; si noti che le funzionalità di questo pacchetto sono già state descritte da Lamport stesso nel 1994, ma queste estensioni sono disponibili solo dal 2003; il pacchetto ha raggiunto una situazione abbastanza stabile solo dal 2004; nel 2009 gli sono state ulteriormente estese le funzionalità; si vede dunque come l'opzione della ⟨*data*⟩ non sia una cosa tanto insolita da usare. Quando questo documento è stato composto, l'ultimo aggiornamento di `pic2e` risaliva al 2014.

Il pacchetto riconosce da solo se si sta componendo con un motore di composizione che sfrutta il linguaggio PDF oppure PostScript e produce la sua uscita in modo che gli oggetti grafici sono in effetti disegnati nel file di uscita dagli interpreti di quei due linguaggi.

`showidx` consente di scrivere le voci raccolte con `\index` direttamente sulla pagina dove sono state trovate; durante la lavorazione di un documento, questo pacchetto talvolta si rivela utile per selezionare e confrontare le varie voci, sia per ridurre i duplicati sia per scegliere i

riferimenti più adatti. Per comporre la versione finale del documento bisogna evidentemente togliere l'invocazione a questo pacchetto.
Si noti che questo pacchetto mostra all'inizio della pagina, nello spazio destinato alle note marginali, l'elenco dei contenuti dei vari comandi `\index` trovati nella pagina. Per parole singole l'informazione può essere utile; se le informazioni inviate al file ausiliario `.idx` sono molto articolate, la stringa inviata al file può essere molto lunga, quindi quanto `showidx` mostra sulla pagina verosimilmente esce dai margini della "carta"; ne segue che queste lunghe informazioni non sono di nessuna utilità perché non possono venire lette completamente. Da quando il formato di uscita preferito è il PDF, l'utilità di questo pacchetto è fortemente diminuita.

1.4.3 Stili delle pagine

Ogni classe inizializza lo stile delle pagine normali secondo le prescrizioni stilistiche generali della classe stessa. Il comando per scegliere lo stile è

`\pagestyle{`⟨*stile*⟩`}`

Il comando

`\thispagestyle{`⟨*stile*⟩`}`

imposta lo stile della pagina solo per la pagina corrente.

Gli stili predefiniti nelle classi standard sono i seguenti.

`empty` sia il piedino sia la testatina sono vuoti.

`plain` contiene solo il piedino con il numero della pagina al centro.

`headings` il piedino è vuoto ma la testatina è elaborata e contiene sul verso delle pagine scritte con l'opzione *twoside* il titolo (breve) del capitolo e nella testatina del recto il titolo (breve) del paragrafo attivo all'inizio della pagina.
Questi titoli sono forniti dai comandi di sezionamento in modo automatico; se si desiderano testatine specifiche, che contengano elementi diversi o titolini correnti diversi, bisogna usare i comandi `\markright` e `\markboth`, oppure bisogna servirsi di pacchetti di estensione come, per esempio, `fancyhdr`. Oppure bisogna servirsi di classi diverse da quelle standard.

`myheadings` in senso generale è simile allo stile `headings`, salvo che le informazioni da inserire nelle testatine debbono essere fornite esplicitamente mediante i comandi `\markright` e `\markboth`.

La sintassi di questi comandi è la seguente:

`\markright{`⟨*titolino di destra*⟩`}`
`\markboth{`⟨*titolino di sinistra*⟩`}{`⟨*titolino di destra*⟩`}`

La numerazione delle pagine viene specificata con il comando

`\pagenumbering{`⟨*stile di numerazione*⟩`}`

dove gli stili di numerazione disponibili sono i seguenti:

`arabic` serve per la normale numerazione con cifre arabe.
`roman` serve per la numerazione romana in lettere minuscole.
`Roman` serve per la numerazione romana in lettere maiuscole.
`alph` serve per la numerazione alfabetica in lettere minuscole.
`Alph` serve per la numerazione alfabetica in lettere maiuscole.
`fnsymbol` non viene usato per numerare le pagine; serve per rappresentare i numeri con la stessa sequenza di simboli usata per le note, per esempio l'asterisco, la spada, la doppia spada e via di seguito. I simboli in totale sono nove, quindi non è possibile numerare più di 9 cose. Perciò questo tipo di numerazione non viene usato per le pagine, ma lo si usa abbastanza spesso per le note, in particolare per quelle del frontespizio.

Difficilmente si useranno numerazioni alfabetiche perché il numero totale delle pagine da numerare potrebbe superare 26, ma per le pagine le numerazioni arabe e romane vengono usate spesso.

Lo stile della pagina è vistosamente differente se si compone su una sola colonna o su due o più colonne; le classi standard consentono al massimo due colonne, ma mediante il pacchetto `multicol` non è difficile aumentarne il numero.

Quando però si usano le classi standard e si specifica di comporre a una sola colonna, è possibile comporre alcune pagine su due colonne, consentendo di comporre anche una intestazione a una colonna. Al contrario se si è specificata l'opzione per comporre su due colonne, allora è possibile comporre alcune pagine su una sola colonna. I comandi da usare sono:

```
\twocolumn[⟨intestazione a una colonna⟩]
\onecolumn
```

Entrambi i comandi iniziano sempre una nuova pagina, poi cominciano a comporre come dice il loro nome.

Ovviamente, lo stile della pagina dipende anche dalla giustezza e dalla sua relazione con il formato della carta, quindi con i margini. Le giustezze orizzontali e verticali sono specificate mediante le lunghezze `\textwidth` e `\textheight` e i margini sono specificati mediante le lunghezze `\oddsidemargin` per le pagine di destra, `\evensidemargin` per le pagine di sinistra, `\topmargin` per il margine superiore; i margini laterali sono comunque i margini sinistri, e le parole 'odd' e 'even' (dispari o pari) si riferiscono alla posizione della pagina in relazione alla sua numerazione, sapendo che il verso di tutte le pagine è sempre pari e il recto sempre dispari. Questi margini sono riferiti allo spigolo superiore sinistro del foglio di carta e tutti sono 'un pollice' di meno di quello che ci si aspetterebbe; questo dipende dal fatto che inizialmente era previsto che tutti i driver di uscita avrebbero inserito di default un ulteriore pollice di *offset*; si veda la figura 1.2.

Non è il caso di mettersi a giocare con questi margini; se proprio si deve modificare l'aspetto geometrico di una pagina, è meglio usare una classe che consenta di farlo, oppure il pacchetto `geometry`.

1.4.4 Il frontespizio

È vero che se uno deve comporre un frontespizio in modo decoroso, è opportuno che studi il layout e la scelta del corpo e dello stile dei font in modo professionale. Tuttavia, nei rapporti e negli articoli anche la modesta composizione di default operata dalla classi standard di LaTeX può essere accettabile.

Bisogna ovviamente specificare l'opzione *titlepage*; ma bisogna anche specificare nel preambolo un certo numero di informazioni.

`\title{`⟨*titolo*⟩`}` serve per specificare il titolo del documento; se questo titolo è sufficientemente lungo lo si può comporre su più righe inserendo il comando `\\`. Bisogna però ricordarsi di non andare a capo fra un articolo e il nome, fra una preposizione e il suo complemento, fra avverbi brevi, come 'non', e quanto segue; fa parte dell'eleganza della

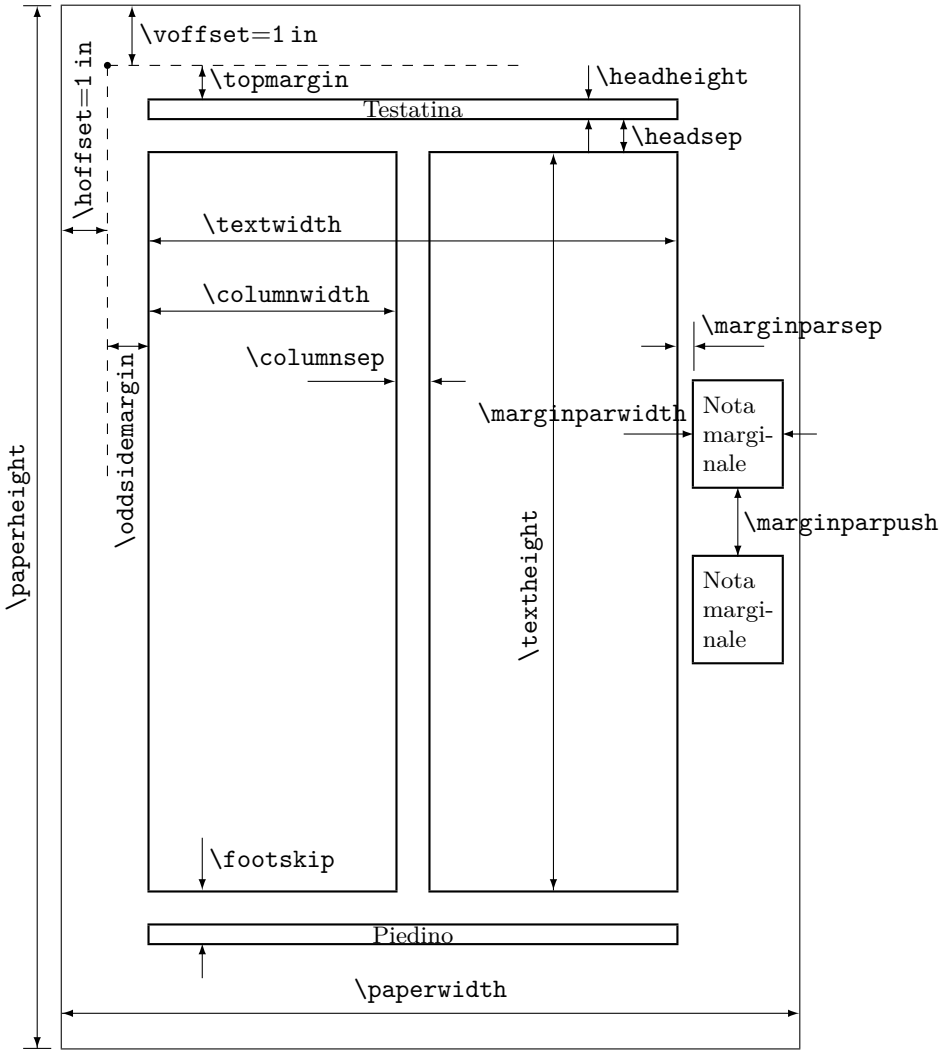

Figura 1.2 Geometria di una pagina dispari. La pagina è composta su due colonne; se viene composta in una sola colonna questa è larga quanto le due colonne compreso il loro spazio intercolonna. In una pagina pari è tutto scambiato fra destra e sinistra; `\oddsidemarign` riguarda lo spazio a sinistra del blocco del testo nelle pagine dispari, mentre `\evensidemargin` riguarda lo spazio a sinistra del blocco di testo in una pagina pari, entrambi a meno della quantità `\hoffset`.

composizione. Riassumendo: nei titoli non si deve mai andare a capo dopo le congiunzioni, gli articoli, le preposizioni e i brevi avverbi; per evitarlo si deve usare il comando di legatura costituito dalla tilde `~`.

`\author{`⟨*nomi*⟩`}` serve per specificare i nomi degli autori; ma se gli autori sono più di uno bisogna separarli con `\and`; si può andare a capo usando `\\`. Attenzione: per ciascun autore, sempre il o i nomi propri prima del o dei cognomi!

`\date{`⟨*testo*⟩`}` il testo di questo comando potrebbe essere una data, come suggerisce il nome del comando, ma potrebbe anche essere il nome di una conferenza, un luogo, qualunque informazione che permetta di identificare il documento oltre al suo titolo e ai suoi autori (i quali potrebbero aver prodotto un articolo con lo stesso titolo a un altro convegno...).

`\thanks{`⟨*testo*⟩`}` questo comando può essere appeso sia ai nomi degli autori, per esempio per specificarne l'afferenza, sia al titolo, per esempio per associargli il nome dell'ente finanziatore delle ricerca, sia al testo specificato con `\date`. Esso funziona come il comando per inserire le note a piè di pagina, solo che il riferimento di queste note non è né un numero né una lettera, ma un generico simbolo tratto da una lista che contiene l'asterisco, la spada, la spada doppia, eccetera (vedi la descrizione nella pagina 36).

Il tutto viene composto dando il comando `\maketitle`. Alternativamente e con risultati esteticamente variabili a seconda del gusto del compositore, si può comporre il frontespizio aprendo l'ambiente *titlepage* e scrivendoci dentro quello che si desidera, dove lo si desidera, con i font che si desiderano; talvolta la cosa produce buoni risultati, talaltra è meglio affidarsi alla composizione di default.

Nei rapporti e negli articoli, spesso il frontespizio può contenere anche un breve riassunto; in LaTeX questo viene composto mediante l'ambiente *abstract*. Il riassunto può comparire come prima cosa dell'articolo e se questo è composto su due colonne, il sunto si trova nella prima colonna, subito sotto il titolo e le informazioni composte con `\maketitle`.

Ovviamente classi diverse da quelle standard possono mettere a disposizione del compositore degli ambienti preconfezionati di *titlepage* che dispongono le informazioni desiderate in modo diverso, più elegante, includendo anche il retro del frontespizio, con le informazioni legali, il copyright,

gli avvisi di copia riservata o proibita, altre informazioni sulle persone che hanno collaborato a impaginare il testo, a fare i disegni, le fotografie, e simili. Alcune classi consentono anche di inserire nel retro del frontespizio sia il *barcode* classico a 10 cifre, sia quello moderno a 13 cifre. Bisogna però documentarsi accuratamente in merito alle classi alternative per scegliere quella che fa al caso proprio.

Va da sé che titoli, stile delle pagine, comandi di sezionamento e simili vanno modificati secondo le necessità, mediante l'uso di classi alternative; come abbiamo ricordato nell'introduzione, una Bibbia richiede certe strutture, un'antologia di poesie una struttura diversa; una edizione critica richiede diversi apparati di note, un codice legale un sezionamento completamente diverso. Una distribuzione non minimale del sistema TeX contiene molte decine di classi alternative a quelle standard, ma tutte le classi, standard o speciali, possono venire ulteriormente personalizzate mediante adeguati pacchetti di estensione, alcuni dei quali citati sopra. Questo non è un incentivo a sbizzarrirsi con classi qualsiasi e poco adatte al contenuto del documento. Al contrario, è un consiglio di documentarsi bene sulle classi disponibili e di studiare bene l'applicazione che se ne vuol fare.

Cito solo un esempio: il pacchetto di estensione `ClassicThesis`, che estende le funzionalità della classe speciale *`scrbook`*, produce un layout della pagina molto bello, con strutture composte in modo molto elegante; nominalmente è stato concepito per comporre belle tesi, ma può essere usato anche per comporre documenti diversi dalle tesi. I font che usa sono scelti con cura e contribuiscono non poco all'eleganza del documento. Ma questa estensione non è adatta a scrivere testi che riguardino le scienze che usano grandezze misurabili, grandezze fisiche, voglio dire, perché le norme ISO richiedono che vengano usati nelle formule certi font che permettano di distinguere correttamente i simboli delle grandezze fisiche dagli altri elementi che entrano a far parte delle equazioni fra grandezze. Certo, si possono usare altre classi che consentano di usare font più adeguati alla scrittura delle relazioni fra le grandezze fisiche, ma una parte dell'eleganza del testo, quella dovuta ai font, viene persa. Questo fatto, quindi, esclude l'uso di questo pacchetto da testi di fisica, ingegneria, chimica, biologia, eccetera, ma lascia libero il campo ai matematici che non sono vincolati dalle norme ISO (vedi la Guida GuIt (Beccari, 2018)).

1.5 Testi in display

1.5.1 Citazioni e poesie

Gli ambienti per i testi fuori testo o, come si suole dire, *in display*, sono:

```
\begin{quote} ⟨testo⟩ \end{quote}
\begin{quotation} ⟨testo⟩ \end{quotation}
\begin{verse} ⟨testo⟩ \end{verse}
```

L'ambiente *quote* è più adatto a citazioni fuori testo di estensione limitata, possibilmente (una frazione di) un solo capoverso dell'opera citata. L'ambiente *quotation* è adatto a citazioni più corpose, di almeno due capoversi.

Nell'ambiente *verse* si termina ogni verso con \\ e si termina ogni strofa come se fosse un capoverso, cioè lasciando una riga vuota nel file sorgente. Il titolo della poesia è opportuno che sia composto con un comando di sezionamento che normalmente impedisce di passare a una nuova pagina subito dopo il titolo.

Il pacchetto `verse` consente di personalizzare il modo di comporre poesie. Il pacchetto `ledmac` o i più recenti `eledmac` `reledmac` consentono di annotare la poesia numerando i versi e annotandoli in relazione al loro numero.

1.5.2 Liste

Gli ambienti predefiniti per le liste sono:

```
\begin{itemize}
\item[⟨contrassegno⟩] ⟨testo⟩
...
\end{itemize}
```

```
\begin{enumerate}
\item[⟨contrassegno⟩] ⟨testo⟩
...
\end{enumerate}
```

```
\begin{description}
\item[⟨voce da descrivere⟩] ⟨testo di descrizione⟩
...
\end{description}
```

In italiano la lista di tipo *itemize* si chiama lista *puntata* o lista *semplice* o *elencazione*; la lista di tipo *enumerate* si chiama *enumerazione*; la lista di tipo *description* si chiama anche descrizione, ma probabilmente sarebbe meglio chiamarla glossario; siccome il pacchetto `glossaries` definisce un altro ambiente per i glossari, qui ci limitiamo a chiamare questo ambiente "descrizione" come in inglese.

Per i primi due ambienti il ⟨*contrassegno*⟩ è davvero facoltativo; per l'ambiente *description*, benché appaia come argomento facoltativo, in realtà esso è obbligatorio, perché non avrebbe senso dare una descrizione di nulla. Tuttavia se non si specifica nulla, nemmeno le parentesi quadre, si comincia una nuova descrizione senza etichetta che può essere vista come un nuovo capoverso; può servire per iniziare una descrizione con un capoverso esplicativo; oppure per inserire fra due distinti gruppi di voci un capoverso che spieghi perché le voci successive sono di una categoria diversa. Generalmente è meglio inserire il capoverso esplicativo prima della descrizione e i "capoversi" fra i gruppi di voci stanno meglio fuori delle descrizioni, e quindi è meglio usare ambienti *description* distinti.

L'ambiente principale, con il quale sono costruite tutte le altre liste, però, è l'ambiente *list*, che è completamente configurabile in ogni dettaglio. La sintassi è:

```
\begin{list}{⟨contrassegno di default⟩}{⟨dichiarazioni⟩}
\item[⟨contrassegno personalizzato⟩] ⟨testo⟩
...
\end{list}
```

Il ⟨*contrassegno di default*⟩ è il modo di comporre il contrassegno che *list* produce quando non viene esplicitato un ⟨*contrassegno personalizzato*⟩ con il comando `\item`. Le ⟨*dichiarazioni*⟩, invece, sono una serie di istruzioni che specificano il modo di comporre eseguito da *list*. Ci sono numerose possibilità che è meglio descrivere una alla volta.

`\topsep` è la distanza che separa il testo che precede la lista dalla prima voce della lista. La stessa distanza viene posta dopo la chiusura della lista.

`\partopsep` è lo spazio ulteriore aggiunto prima e dopo la lista se questa comincia un nuovo capoverso, cioè se nel file sorgente una riga completamente vuota precede il comando di apertura dell'ambiente.

`\itemsep` è lo spazio aggiuntivo che viene messo prima di una voce della lista se questa è preceduta da una riga bianca.

`\parsep` Siccome dentro le liste i capoversi solitamente non vengono rientrati, allora si può usare un piccolo spazio aggiuntivo fra due capoversi appartenenti alla stessa voce; questo spazio di separazione fra i *paragraphs* si chiama appunto `\parsep`.

`\leftmargin` è la rientranza di ogni voce della lista rispetto al testo circostante; liste annidate hanno rientranze assolute (cioè rispetto alla gabbia di stampa) sempre maggiori; il file di classe specifica le varie successive rientranze (relative al margine sinistro del testo o della lista precedente) mediante i comandi `\leftmargini`, `\leftmarginii`, `\leftmarginiii` e `\leftmarginiv`.

`\rightmargin` è la distanza orizzontale fra il margine destro della lista corrente e il margine destro del testo che contiene la lista.

`\listparindent` è il rientro extra aggiunto (o anche tolto, se ha un valore negativo) alla rientranza di ogni riga di ciascuna voce, tranne la prima; tutte le classi standard hanno questo parametro impostato a zero, ma non è impossibile che in altre classi o in liste personalizzate non si possa usare questo parametro.

`\itemindent` è la rientranza della prima riga di ogni voce della lista. Può anche avere un valore negativo.

`\labelsep` è la distanza minima che separa il ⟨*contrassegno*⟩ dal resto del testo nella prima riga di ogni voce.

`\labelwidth` è la larghezza prevista per il ⟨*contrassegno*⟩; a seconda della personalizzazione, se questo contrassegno fosse più largo o più stretto di questa larghezza, esso viene fatto sporgere a sinistra, oppure viene fatto sporgere a destra ma cominciando il testo della voce più a destra in modo da mantenere costante lo spazio fra la fine del ⟨*contrassegno*⟩ e l'inizio della voce.

`\makelabel{`⟨*contrassegno*⟩`}` è la macro che effettivamente compone il contrassegno.

`\usecounter{`⟨*contatore*⟩`}` specifica il ⟨*contatore*⟩ da usare per numerare gli elementi della lista e ne rende il contenuto usabile dai comandi `\label` e `\ref`.

Alcune delle dimensioni descritte qui sopra sono riportate nella figura 1.3; quanto è indicato fra parentesi quadre nella figura indica gli spazi che vengono inseriti se la lista ha bisogno di ulteriori spazi nel caso che cominci un nuovo capoverso o che un nuovo capoverso cominci dopo la lista. Si noti che per impostazione predefinita `\listparindent` ha un valore nullo, mentre `\parsep` ha un piccolo valore positivo, per cui i diversi capoversi di una stessa voce risultano spaziati. Volendo, si potrebbe impostare `\parsep` a zero e specificare un buon rientro ad ogni capoverso dopo il primo; tuttavia, al fine di evitare righe orfane e vedove in una lista che contenga un cambio di pagina, è preferibile conservare l'impostazione predefinita che assicura una leggera allungabilità agli spazi fra i capoversi.

Vale la pena di provare a leggere la definizione dell'ambiente *enumerate* per capire meglio come si usi l'ambiente *list*. L'esempio che si porta riguarda l'ambiente *enumerate* della classe *book*: le sue definizioni sono sparse in diversi file; i valori numerici sono contenuti nel file `bk10.clo` per quando si compone con il corpo di default di 10 pt. Le definizioni vere e proprie dell'ambiente sono eseguite con comandi di basso livello e/o con comandi di servizio di LaTeX, ma si ritiene che siano abbastanza chiare da non richiedere una spiegazione, mentre se ne daranno per altri aspetti delle definizioni.

```
%                                   Definizioni in latex.ltx
\newcount\@enumdepth \@enumdepth = 0
\@definecounter{enumi}
\@definecounter{enumii}
\@definecounter{enumiii}
\@definecounter{enumiv}
%                                   Definizioni in book.cls
\renewcommand\theenumi{\@arabic\c@enumi}
\renewcommand\theenumii{\@alph\c@enumii}
\renewcommand\theenumiii{\@roman\c@enumiii}
\renewcommand\theenumiv{\@Alph\c@enumiv}
```

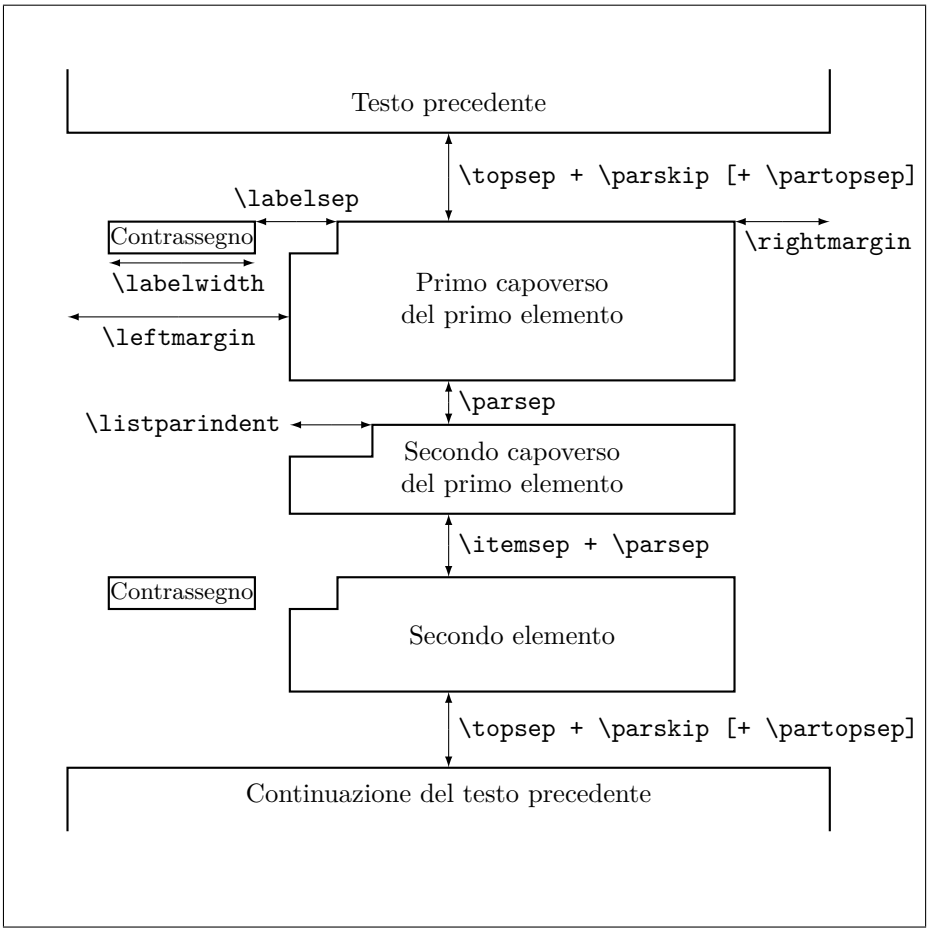

Figura 1.3 Alcune dimensioni che caratterizzano le liste secondo quanto spiegato nel testo

```
\newcommand\labelenumi{\theenumi.}
\newcommand\labelenumii{(\theenumii)}
\newcommand\labelenumiii{\theenumiii.}
\newcommand\labelenumiv{\theenumiv.}
\renewcommand\p@enumii{\theenumi}
\renewcommand\p@enumiii{\theenumi(\theenumii)}
\renewcommand\p@enumiv{\p@enumiii\theenumiii}
```

```
%                                    Definizioni in bk10.clo
\def\@listi{\leftmargin\leftmargini
            \parsep 4\p@ \@plus2\p@ \@minus\p@
            \topsep 8\p@ \@plus2\p@ \@minus4\p@
            \itemsep4\p@ \@plus2\p@ \@minus\p@}
\let\@listI\@listi
\@listi
\def\@listii {\leftmargin\leftmarginii
              \labelwidth\leftmarginii
              \advance\labelwidth-\labelsep
              \topsep    4\p@ \@plus2\p@ \@minus\p@
              \parsep    2\p@ \@plus\p@  \@minus\p@
              \itemsep   \parsep}
\def\@listiii{\leftmargin\leftmarginiii
              \labelwidth\leftmarginiii
              \advance\labelwidth-\labelsep
              \topsep    2\p@ \@plus\p@\@minus\p@
              \parsep    \z@
              \partopsep \p@ \@plus\z@ \@minus\p@
              \itemsep   \topsep}
\def\@listiv {\leftmargin\leftmarginiv
              \labelwidth\leftmarginiv
              \advance\labelwidth-\labelsep}
\def\@listv  {\leftmargin\leftmarginv
              \labelwidth\leftmarginv
              \advance\labelwidth-\labelsep}
\def\@listvi {\leftmargin\leftmarginvi
              \labelwidth\leftmarginvi
              \advance\labelwidth-\labelsep}
%                                    Definizioni in latex.ltx
\def\enumerate{%
  \ifnum \@enumdepth >\thr@@\@toodeep\else
    \advance\@enumdepth\@ne
    \edef\@enumctr{enum\romannumeral\the\@enumdepth}%
    \expandafter\list\csname label\@enumctr\endcsname
    {\usecounter\@enumctr\def\makelabel##1{\hss\llap{##1}}}%
  \fi}
```

```
\let\endenumerate =\endlist
```

Inizialmente si definiscono il contatore di livello e i quattro contatori per ciascuno dei quattro livelli; si ricorderà infatti che le liste, anche le enumerazioni, possono essere annidate l'una nell'altra, ma solo fino al quarto livello.

Successivamente, nella classe *book* si definiscono i modi di rappresentare i vari contatori di enumerazione; il primo contatore viene stampato mediante `\@arabic\c@enumi`, cioè in cifre arabe; nella lista, però, esso viene marcato come il numero seguito da un punto: `\theenumi.`

Per il primo livello basta così, ma per i livelli successivi ci vuole un prefisso; quando si cita il terzo elemento della seconda lista annidata, per esempio, non basta la scrittura del valore del secondo contatore, che viene stampato con una lettera mediante `\@alph\c@enumii`, ma nella lista questa lettera è racchiusa fra parentesi tonde `(\theenumii)`. Nella citazione del terzo elemento non basta che appaia 'c' oppure '(c)', ma ci vuole anche un prefisso che faccia riferimento alla prima lista; ecco quindi il prefisso `\theenumi` che fa sì che quell'elemento sia citato come '2(c)'. Lo stesso avviene per gli elementi delle altre liste interne.

Nel file `bk10.clo`, che contiene le dimensioni e i corpi specificati per quando si compone con l'opzione di default *10pt*, sono definiti alcuni dei parametri descritti per l'ambiente *list*. Per l'enumerazione di primo livello il tutto è contenuto nella definizione del comando `\@listi`; questi spazi si riferiscono a tutte le liste di primo livello, non solo alle enumerazioni; precisamente sono definiti `\parsep`, `\topsep` e `\itemsep`; sono tutte lunghezze elastiche, perché contengono la componente di allungamento, introdotta da `\@plus`, e la componente di accorciamento, introdotta da `\@minus`; il comando `\p@` indica una lunghezza predefinita del valore di un punto tipografico. Fra le definizioni contenute in `\@listi` compare anche `\leftmargin`, a cui viene assegnato il valore di `\leftmargini`; questo a sua volta è definito nella classe *book* come una quantità espressa in unità di misura `em`, legate, cioè, al corpo del font in uso in quel contesto (potrebbe anche essere una nota o una citazione scritta in corpo minore). Definizioni analoghe appaiono anche per i livelli interni di annidamento.

Infine viene la definizione vera e propria che ricorre all'ambiente *list* così come è contenuta in `latex.ltx`. In questa definizione, per prima cosa si verifica che il livello di annidamento già raggiunto non sia superiore a

tre; se così fosse, si emette un messaggio di errore mediante il comando `\@toodeep`, altrimenti si incrementa il contatore del livello di annidamento e si procede alle varie definizioni; si noti che per fare riferimento ai parametri corrispondenti al livello di annidamento corrente si usa il suo nome ottenuto agglutinando il prefisso `enum` con il valore del contatore di annidamento espresso in numeri romani minuscoli; questo risultato si ottiene con i comandi nativi `\csname` e `\endcsname`; in generale, l'utente che deve usare l'ambiente *list* per confezionare una lista speciale non ha bisogno di ricorrere a questi comandi nativi; tuttavia, all'occorrenza si può consultare il TeXbook.

Il primo argomento dell'ambiente *list* (attivato senza ricorrere a `\begin`, ma dando direttamente il comando 'interno' `\list`, non è racchiuso fra parentesi graffe, ma è il frutto di `\expandafter` e nuovamente di `\csname` e `\endcsname` che assieme agglutinano il prefisso `label` con il valore del contatore di annidamento espresso in numeri romani minuscoli.

Il secondo argomento dice di usare il contatore `\@enumctr` e dice di comporre l'etichetta di ogni voce mediante un numero composto in bandiera giustificata a destra, autorizzando anche di fuoriuscire dal margine di sinistra mediante l'accorgimento di far precedere il comando `\llap` (*left overlap*, sovrapponi a sinistra) con un blocco di gomma elastica di lunghezza naturale nulla e allungamento e accorciamento infiniti `\hss`.

Il comando di chiusura dell'ambiente *enumerate* è reso equivalente attraverso `\let` al comando di chiusura `\endlist` della generica lista.

L'esempio appena fatto, forse, è complicato, ma in realtà delucida tante cose che raramente sono descritte nella documentazione di LaTeX e mostra anche alcune tecniche di programmazione a livello di comandi nativi di TeX.

Un esempio più semplice. Si vuole comporre una ambiente simile a *description* che abbia la possibilità di definire il margine sinistro sulla base della lunghezza di una parola; potrebbe essere utile per un glossario, per esempio. Vogliamo anche che ogni chiave che introduce ogni voce sia scritta in carattere lineare, ma non sia nero. Vediamo prima come è definito l'ambiente *description*:

```
\newenvironment{description}
               {\list{}{\labelwidth\z@ \itemindent-\leftmargin
                        \let\makelabel\descriptionlabel}}
               {\endlist}
```

```
\newcommand*\descriptionlabel[1]{\hspace\labelsep
                                \normalfont\bfseries #1}
```

Dobbiamo quindi definire comandi simili, per esempio l'ambiente *descrizione*, che accetti un argomento sul quale prendere le misure del margine sinistro, e la sua etichetta `\etichettadescrizione`. Ecco come fare:

```
\newenvironment{descrizione}[1]%
   {\list{}{\settowidth{\labelwidth}{\normalfont\sffamily#1}%
         \let\makelabel\etichettadescrizione}}
   {\endlist}
\newcommand*{\etichettadescrizione}[1]{\hspace\labelsep
                  \normalfont\sffamily #1}
```

Come si vede, il compito era inizialmente più facile e la soluzione è decisamente più semplice.

1.5.3 Testo composto verbatim

Si possono usare due comandi e due ambienti per comporre del testo in modo verbatim[6]; questo modo di comporre è quello che serve quando bisogna esporre, per esempio, dei brani scritti in un linguaggio di programmazione nel quale si fa uso dei caratteri speciali di TEX (ovviamente con altri significati); oppure per esporre brani di programmazione in linguaggio TEX. I comandi sono:

`\verb`*simbolo*⟨*testo da riprodurre*⟩*simbolo* oppure `\verb*`*simbolo*⟨*testo da riprodurre*⟩*simbolo*

Il *simbolo* è un solo carattere e funziona da delimitatore del ⟨*testo da riprodurre*⟩; la versione senza asterisco riproduce gli spazi come spazi; la versione con l'asterisco riproduce gli spazi con il segno speciale ␣. Il ⟨*testo da riprodurre*⟩ deve comparire in una sola riga nel file sorgente.

Gli ambienti sono invece:

[6] Dal latino *verbatim*, che in questo caso significa *alla lettera*; in LATEX quindi, il testo sorgente non viene formattato né interpretato, ma viene scritto davvero lettera per lettera, tale e quale esso appare nel file sorgente.

```
\begin{verbatim}
⟨testo da riprodurre verbatim⟩
\end{verbatim}

\begin{verbatim*}
⟨testo da riprodurre verbatim⟩
\end{verbatim*}
```

Il ⟨*testo da riprodurre verbatim*⟩ si può svolgere su diverse righe, anzi di solito è composto di diverse righe. L'unica riga che non si può riprodurre è `\end{verbatim}`, con o senza asterisco. L'ambiente asteriscato e il comando asteriscato, riproducono lo spazio in modo visibile con il carattere ␣.

Il comando `\verb` è fragilissimo e non può apparire come argomento di nessun altro comando (nemmeno nella versione asteriscata); esistono però i pacchetti `verbdef`, `verbatim` e `fancyvrb` che aiutano a superare questo scoglio forniscono altri utili modi di comporre il testo in modo verbatim.

Il pacchetto standard `alltt` definisce un ambiente *alltt* da usare come l'ambiente *verbatim* dove tutti i caratteri sono diventati caratteri 'normali', salvo `\`, `{` e `}`.

1.6 Formule matematiche

1.6.1 Formule

La matematica in linea può essere delimitata dai seguenti delimitatori:

```
$ ⟨formula in linea⟩ $
\( ⟨formula in linea⟩ \)
\begin{math} ⟨formula in linea⟩ \end{math}
```

Si consiglia di non usare il primo metodo, anche se è più comodo da scrivere, perché si perde la diagnostica di LaTeX nel caso che ci si dimentichi di uno dei due delimitatori. L'ambiente *math*, come si può ben capire, non viene usato molto spesso. I delimitatori `\(` e `\)` dispongono della diagnostica, ma sono fragili. Invece i delimitatori `$`...`$` sono robusti, anche se mancano della diagnostica.

Il comando

```
\ensuremath{⟨formula⟩}
```

permette di comporre una ⟨*formula*⟩ in linea garantendo che essa venga composta correttamente anche se il comando viene emesso in modo testuale. È comodo per definire macro che debbono potersi usare sia nel modo testo sia nel modo matematico.

Usando il pacchetto `babel` è disponibile il comando:

`\textormath{`⟨*per il modo testo*⟩`}{`⟨*per il modo matematico*⟩`}`

che produce un effetto analogo a quello di `\ensuremath`, con una notevole differenza; usato per definire un comando da usare sia in modo testo sia in modo matematico, può eseguire composizioni diverse nell'uno o nell'altro modo senza che il compositore debba preoccuparsi di verificare il modo di composizione; `\ensuremath`, invece, compone il suo argomento sempre in modo matematico, all'occorrenza entrando ed uscendo da questo modo, al fine di lasciare TEX nello stesso stato nel quale si trovava prima dell'esecuzione del comando.

Per la matematica in display si hanno gli ambienti

`\[` ⟨*formula non numerata in display*⟩ `\]`
`\begin{displaymath}` ⟨*formula non numerata in display*⟩ `\end{displaymath}`
`\begin{equation}` ⟨*formula numerata in display*⟩ `\end{equation}`

I primi due ambienti non assegnano un numero alla ⟨*formula in display*⟩, mentre il terzo ambiente assegna un numero a detta formula. L'ambiente *displaymath* viene usato raramente visto che quello definito con `\[...\]` è molto più comodo. L'ambiente *equation* non ha sostituti.

LATEX offrirebbe anche gli ambienti *eqnarray* e *eqnarray** ma è preferibile non servirsene; al contrario, per comporre formule e sistemi di formule in display è molto meglio servirsi del pacchetto `amsmath`. I due ambienti citati soffrono di un difetto "storico" presente anche nella precedente versione 2.09 di LATEX; questo difetto consiste nel fatto che la spaziatura attorno alla seconda colonna, che solitamente contiene un operatore di relazione, non è coerente con la spaziatura corretta attorno agli operatori di relazione ma è, appunto, la spaziatura intercolonna di una tabella; si noti infatti che cosa succede con un piccolo sistema di due equazioni con due incognite composto con *eqnarray**:

$$\begin{aligned} 2x + y &= 33 \\ x + 2y &= 25 \end{aligned}$$

Lo stesso sistema composto con l'ambiente *align** del pacchetto `amsmath` ha invece la spaziatura corretta:

$$2x + y = 33$$
$$x + 2y = 25$$

e si vede in modo inequivocabile la differenza di spaziatura a cavallo del segno di 'uguale'; il secondo esempio ha la spaziatura corretta.

Dentro gli ambienti *equation* e *eqnarray* (all'interno di ciascuna riga) e, ovviamente, negli ambienti previsti da `amsmath`, è possibile inserire il comando `\label` così da definire una etichetta mnemonica per richiamare il numero della formula con `\ref` e/o il numero della pagina in cui cade la formula con `\pageref`. Dentro *eqnarray* si può usare `\nonumber` (`\notag` se si usa il pacchetto `amsmath`) per evitare che LaTeX assegni un numero a una espressione che non si vuole numerare. Questo ambiente consentirebbe anche di spezzare una singola lunga espressione matematica su più righe, ricorrendo anche al comando suddetto e a `\lefteqn`; ma per questo scopo è meglio servirsi degli opportuni ambienti predisposti apposta dal pacchetto `amsmath`.

Le espressioni in display usano un certo numero di parametri per comporre secondo diversi stili.

`\jot` serve per aggiungere altro spazio fra una equazione e l'altra in un sistema di equazioni; `\jot` è una lunghezza a cui si può attribuire un valore a piacere, senza esagerare; infatti bisogna sempre ricordare che 3 pt equivalgono a circa 1 mm e gli spazi verticali devono sempre rimanere piuttosto contenuti.

`\mathindent` serve per fissare la rientranza sinistra delle equazioni quando queste sono composte allineate a sinistra se si usa l'opzione *fleqn* per la classe del documento.

`\abovedisplayskip` serve per definire lo spazio verticale che precede una equazione in display; viene usato uno spazio verticale più piccolo quando la riga prima del display è molto corta e l'espressione matematica potrebbe dare l'impressione ottica di essere preceduta da uno spazio troppo grande.

`\abovedisplayshortskip` è appunto questo spazio verticale superiore più piccolo.

`\belowdisplayskip` è lo spazio verticale da inserire dopo una formula per distanziarla dal testo seguente.

`\belowdisplayshortskip` serve per inserire uno spazio verticale più piccolo se l'espressione matematica è seguita da una riga di testo molto corta.

Va da sé che i `...shortskip` non vengono usati quando si compone con l'opzione *fleqn* perché le espressioni sono tutte allineate a sinistra, a meno del loro rientro non molto importante (esso è predefinito pari al valore del rientro di un capoverso; se si usano formule numerate a sinistra potrebbe essere poco e conviene assegnargli un valore maggiore).

L'algoritmo con il quale LaTeX decide di usare gli 'skip' normali o quelli 'corti' dipende dallo spazio che rimarrebbe fra il righino superiore (o inferiore) e la formula centrata; se questo spazio è negativo, il righino e la formula si sovrapporrebbero e quindi vengono sicuramente impiegati gli 'skip' normali; se invece lo spazio che rimarrebbe fra le due entità (righino e formula) è maggiore di un certo limite specificato nella classe in uso, allora vengono usati gli 'skip' corti; non sono riuscito a trovare questa grandezza né nel file `latex.ltx` (che contiene il nucleo di LaTeX) né nei file di classe che ho esaminato, ma il capitolo 19 del TeXbook è esplicito sul fatto che questa grandezza deve separare il righino precedente dall'espressione matematica se fosse composta sulla stessa riga: vi si accenna con una indicazione qualitativa di un quadrato (cioè di una spaziatura larga quanto il corpo del carattere di testo nel contesto in cui appare l'espressione matematica.

Per comporre le espressioni matematiche, si usano spesso piccoli comandi o piccole strutture di cui è facile dimenticarsi.

Esponenti Gli esponenti si inseriscono mediante il comando `^`

`^{`⟨*esponente*⟩`}`

Con l'opzione *italian* di `babel` è disponibile la macro `\ap{`⟨*esponente*⟩`}` per inserire un'annotazione in tondo quando si è in modo matematico, un esponente con il font corrente quando si è in modo testo. Il nucleo di LaTeX per il modo testo prevede anche il comando `\textsuperscript`, valido sempre, anche senza l'uso di `babel`.

Apici L'apice $'$ e il doppio apice $''$ si inseriscono con uno o due apostrofi, che in modo matematico vengono composti come apici.

Pedici e deponenti I pedici vengono inseriti con il comando `_`

`_{`⟨*pedice*⟩`}`

Con l'opzione *italian* di `babel` è possibile usare la macro `\ped{`⟨*pedice*⟩`}` per inserire un pedice in tondo in modo matematico e per inserire un deponente con il font corrente in modo testo. Con alcune classi è disponibile il comando `\textsubscript` per svolgere la stessa funzione di `\ped` in modo testo anche quando non si usa l'opzione *italian* per `babel`.

Frazioni Le frazioni vengono composte con il comando

`\frac{`⟨*numeratore*⟩`}{`⟨*denominatore*⟩`}`

Bisogna ricordare che questo comando compone le frazioni in display come ci si aspetterebbe, ma le compone in modo testo sia nelle espressioni matematiche in linea, sia nelle sottoespressioni come per esempio un altro numeratore o denominatore; in questi casi è preferibile servirsi delle frazioni a barra obliqua, piuttosto che di quelle a barra orizzontale:

$$x = \frac{a + \frac{b}{c}}{d + \frac{e}{f}} \qquad \text{a confronto con} \qquad x = \frac{a + b/c}{d + e/f}$$

Radici Le radici quadrate o con altro ⟨*indice*⟩ si compongono con

`\sqrt[`⟨*indice*⟩`]{`⟨*radicando*⟩`}`

Se l'⟨*indice*⟩ vale `2` non lo si indica; esempi:

$$\sqrt{2} \approx 1,414 \qquad \sqrt[3]{2} \approx 1,256$$

Ellissi Le parti omesse (ellissi) vengono sostituite con i soliti tre puntini; in matematica ce ne sono di diversi tipi: `\dots` (...) si può usare sia in modo matematico sia in modo testo e produce i soliti puntini a livello della linea di base in modo testo, o a livello della linea di base o dell'asse matematico in modo matematico a seconda del contesto,

cioè se è preceduto o seguito da segni di interpunzione appoggiati alla linea di base oppure da operatori binari centrati sull'asse matematico; `\ldots` produce lo stesso effetto di collocare i tre puntini sulla linea di base, ma è da usare in modo matematico; `\cdots` ($\cdots$) produce i tre puntini allineati con il segno $-$, cioè sull'asse matematico della formula; `\vdots` ($\vdots$) produce tre puntini verticali che servono per sostituire in verticale gli elementi omessi da colonne di matrici; infine `\ddots` ($\ddots$) produce tre puntini in diagonale, utili per riempire una parte omessa dal cuore di una matrice.
Se si usa `\dots` in modo testo i tre puntini sono sempre sulla linea di base; ma se si usa il pacchetto `amsmath`, questo comando è ridefinito in modo che in matematica si comporti in modo intelligente, vale a dire che colloca i puntini sulla linea di base se almeno da un lato è contornato da un segno di interpunzione, mentre produce i puntini sull'asse matematico della formula se è contornato da almeno un operatore binario. Si confrontino i seguenti due casi, composti entrambi con `\dots`:

$$i = 1, 2, 3, \dots, n \qquad\qquad i = 1 + 2 + 3 + \dots + n$$

1.6.2 Simboli, accenti, delimitatori e grandi operatori

Tutti i simboli di qualunque genere usabili in matematica sono raccolti nelle tabelle 1.3–1.10; nelle tabelle 1.11–1.12, invece, sono raccolti gli ulteriori simboli disponibili con il pacchetto `amssymb`.

Bisogna notare che diversi pacchetti consentono di scrivere le lettere greche maiuscole e minuscole sia inclinate sia diritte; generalmente le maiuscole sono diritte per impostazione predefinita ma, "alla peggio", cioè senza caricare altri pacchetti, i comandi indicati nella tabella 1.6 sono sempre disponibili. Invece, per le lettere minuscole diritte bisogna necessariamente disporre dei pacchetti appositi; per esempio i pacchetti `txfonts` o `pxfonts` (per usare rispettivamente i font Times eXtended o i Palatino eXtended) o altri pacchetti che contengono comandi come `\alphaup`, `\betaup`, eccetera, con i quali è possibile disporre delle lettere minuscole greche diritte. I pacchetti `txfonts` e `pxfonts` sono stati sostituiti recentemente con coppie di pacchetti `newtxtext`, `newtxmath`, `newpxtext` e `newpxmath` rispettivamente; questi nuovi pacchetti sono decisamante migliorati rispetto ai precedenti.

Tabella 1.3 Accenti matematici e altri diacritici usati in matematica

`\hat{x}`	$\hat{x}$	`\check{x}`	$\check{x}$
`\breve{x}`	$\breve{x}$	`\acute{x}`	$\acute{x}$
`\grave{x}`	$\grave{x}$	`\tilde{x}`	$\tilde{x}$
`\bar{x}`	$\bar{x}$	`\vec{x}`	$\vec{x}$
`\dot{x}`	$\dot{x}$	`\ddot{x}`	$\ddot{x}$
`\overline{x+y}`	$\overline{x+y}$	`\underline{x+y}`	$\underline{x+y}$
`\overbrace{x\cdots y}`	$\overbrace{x\cdots y}$	`\underbrace{x\cdots y}`	$\underbrace{x\cdots y}$
`\overrightarrow{x-y}`	$\overrightarrow{x-y}$	`\overleftarrow{x-y}`	$\overleftarrow{x-y}$
`\widehat{x-y}`	$\widehat{x-y}$	`\widetilde{x-y}`	$\widetilde{x-y}$
`\mathring{x}`	$\mathring{x}$		

È opportuno osservare, fra i "diacritici" matematici, che i comandi `\overbrace` e `\underbrace` sono molto particolari; essi non servono solamente per mettere sopra o sotto una espressione matematica un graffa orizzontale, ma servono anche per applicare una annotazione all'espressione, una annotazione che di solito è costituita da un'altra breve espressione matematica; infatti la loro sintassi è rispettivamente:

```
\overbrace{⟨espressione matematica⟩}^{⟨annotazione⟩}
\underbrace{⟨espressione matematica⟩}_{⟨annotazione⟩}
```

e l'⟨*annotazione*⟩ (matematica) viene collocata rispettivamente sopra o sotto l'⟨*espressione matematica*⟩ principale nel corpo degli indici di primo livello.

Va da sé che se l'annotazione è testuale, l'⟨*annotazione*⟩ deve essere scritta in ambiente testo; quindi, se si sta usando il modulo `amsmath`, si può scrivere la nota nella forma `\text{`⟨*annotazione*⟩`}`; se invece non si sta usando `amsmath`, bisogna ricorrere a una "scatola", come `\mbox`, nella quale si deve anche specificare a mano il corpo del font usato per la nota, per esempio `\mbox{`⟨`\scriptsize` *annotazione*⟩`}`.

Sarebbe un errore usare il comando `\mathr` per scrivere in tondo l'annotazione, perché questa verrebbe scritta in tondo ma in modo matematico. Si confronti

Tabella 1.4 Delimitatori di dimensioni variabili

(	(	/	/	\langle	⟨	\rangle	⟩
)	)	\lfloor	⌊	\rfloor	⌋	\lgroup	(
[	[	\lceil	⌈	\rceil	⌉	\rgroup	)
]	]	\backslash	\	\vert	|	\bracevert	⎪
|	‖	\Vert	‖	\updownarrow	↕	\Updownarrow	⇕
|	|	\uparrow	↑	\Uparrow	⇑	\downarrow	↓
}	}	\arrowvert	⏐	\Arrowvert	‖	\Downarrow	⇓
{	{	\lmoustache	⎰	\rmoustache	⎱		

- "valori efficaci", scritto con `\mbox{valori efficaci}`, con
- "valoriefficaci", scritto con `\mathrm{valori efficaci}`.

Nelle tabelle da 1.7 a 1.10 alcuni simboli sono riportati su uno sfondo grigio; questi simboli ricorrevano, con il vecchio LaTeX 209, a una collezione speciale di simboli, che, per retrocompatibilità, oggi si potrebbero

Tabella 1.5 Grandi operatori di diverse grandezze

`\bigcap`	$\bigcap$	$\cap$	`\bigvee`	$\bigvee$	$\vee$
`\bigcup`	$\bigcup$	$\cup$	`\bigwedge`	$\bigwedge$	$\wedge$
`\bigodot`	$\bigodot$	$\odot$	`\coprod`	$\coprod$	$\amalg$
`\bigoplus`	$\bigoplus$	$\oplus$	`\int`	$\int$	$\int$
`\bigotimes`	$\bigotimes$	$\otimes$	`\oint`	$\oint$	$\oint$
`\bigsqcup`	$\bigsqcup$	$\sqcup$	`\prod`	$\prod$	$\prod$
`\biguplus`	$\biguplus$	$\uplus$	`\sum`	$\sum$	$\sum$

`\sqrt`	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$	$\sqrt{}$

invocare tramite il pacchetto `latexsym`; questa operazione è fortemente sconsigliata; è assai preferibile usare i font dell'American Mathematical Society (tabelle 1.11 e 1.12) disegnati meglio e coerenti con gli altri simboli matematici. Il pacchetto `latexsymb` serve solo per comporre vecchi documenti composti prima dell'avvento di LaTeX 2_ε; inoltre *bisogna evitare* di caricare simultaneamente il pacchetto `latexsym` assieme ai font dell'American Mathematical Society, anche se è possibile farlo; i risultati sono o scadenti, oppure si è caricata la memoria del programma di composizione con un pacchetto inutile di font matematici; siccome non si possono usare simultaneamente più di 16 alfabeti matematici, sprecare uno di questi alfabeti può talvolta essere un danno grave.

1.6.3 Impilare gli oggetti matematici

Una (breve) espressione può venire soprallineata con

`\overline{`⟨*espressione*⟩`}`

Analogamente, si può sottolineare una espressione con

Tabella 1.6 Le lettere greche in matematica con le loro varianti

Minuscole					
\alpha	α	\iota	ι	\rho	ρ
\beta	β	\kappa	κ	\sigma	σ
\gamma	γ	\lambda	λ	\tau	τ
\delta	δ	\mu	μ	\upsilon	υ
\epsilon	ϵ	\nu	ν	\phi	ϕ
\zeta	ζ	\xi	ξ	\chi	χ
\eta	η	o	o	\psi	ψ
\theta	θ	\pi	π	\omega	ω
Varianti delle minuscole					
\varepsilon	ε	\varpi	ϖ	\varsigma	ς
\vartheta	ϑ	\varrho	ϱ	\varphi	φ
Maiuscole					
\Gamma	Γ	\Xi	Ξ	\Phi	Φ
\Delta	Δ	\Pi	Π	\Psi	Ψ
\Theta	Θ	\Sigma	Σ	\Omega	Ω
\Lambda	Λ	\Upsilon	Υ		
Maiuscole corsive					
\mathit{\Gamma}	$\mathit{\Gamma}$	\mathit{\Xi}	$\mathit{\Xi}$	\mathit{\Phi}	$\mathit{\Phi}$
\mathit{\Delta}	$\mathit{\Delta}$	\mathit{\Pi}	$\mathit{\Pi}$	\mathit{\Psi}	$\mathit{\Psi}$
\mathit{\Theta}	$\mathit{\Theta}$	\mathit{\Sigma}	$\mathit{\Sigma}$	\mathit{\Omega}	$\mathit{\Omega}$
\mathit{\Lambda}	$\mathit{\Lambda}$	\mathit{\Upsilon}	$\mathit{\Upsilon}$		

`\underline{`⟨*espressione*⟩`}`

Anche gli accenti matematici possono venire impilati, ma si tratta semplicemente di racchiudere fra le graffe che delimitano l'argomento del primo accento una espressione già accentata.

Invece, `\stackrel` permette di costruire degli operatori di relazione impilando uno sopra l'altro due segni distinti, scelti fra i vari simboli disponibili:

`\stackrel{`⟨*elemento superiore*⟩`}{`⟨*elemento inferiore*⟩`}`

Tabella 1.7 Operatori binari

\amalg	⨿	\ast	∗	\bigcirc	◯
\bigtriangledown	▽	\bigtriangleup	△	\bullet	•
\cap	∩	\cdot	·	\circ	∘
\cup	∪	\dagger	†	\ddagger	‡
\diamond	⋄	\div	÷	\lhd	◁
\mp	∓	\odot	⊙	\ominus	⊖
\oplus	⊕	\oslash	⊘	\otimes	⊗
\pm	±	\rhd	▷	\setminus	\
\sqcap	⊓	\sqcup	⊔	\star	⋆
\times	×	\unlhd	⊴	\unrhd	⊵
\uplus	⊎	\vee	∨	\wedge	∧
\wr	≀				

Per esempio, il segno di "uguale per definizione" si può definire con

```
\newcommand\eqdef{\stackrel{\mathrm{def}}{=}}
```

per ottenere, per esempio:

$$\mathrm{e} \stackrel{\mathrm{def}}{=} 2,718\,281\,828\,459\,\ldots$$

1.6.4 Spaziatura matematica

Non si può raccomandare mai abbastanza l'opportunità di non spaziare le espressioni matematiche. I comandi di spaziatura hanno senso se e solo se la spaziatura di default non è adeguata ai simboli che compaiono uno accanto all'altro in una specifica espressione matematica. Se, quindi, si inserirà una qualche spaziatura, lo si farà solo dopo aver corretto le bozze controllando accuratamente che questa spaziatura sia praticamente impercettibile.

\,	spazio sottile	\:	spazio medio
\!	spazio sottile negativo	\;	spazio grande

Il comando \, può essere usato anche in modo testo. Naturalmente, si potrebbero anche usare \quad e \qquad oltre ai comandi di spaziatura del

Tabella 1.8 Operatori di relazione

\approx	≈	\asymp	≍	\bowtie	⋈
\cong	≅	\dashv	⊣	\doteq	≐
\downarrow	↓	\Downarrow	⇓	\equiv	≡
\frown	⌢	\ge	≥	\geq	≥
\gets	←	\gg	≫	\hookleftarrow	↩
\hookrightarrow	↪	\iff	⟺	\in	∈
\Join	⋈	\le	≤	\leadsto	⇝
\leftarrow	←	\Leftarrow	⇐	\leftharpoondown	↽
\leftharpoonup	↼	\leftrightarrow	↔	\Leftrightarrow	⇔
\leq	≤	\ll	≪	\longleftarrow	⟵
\Longleftarrow	⟸	\longleftrightarrow	⟷	\Longleftrightarrow	⟺
\longmapsto	⟼	\longrightarrow	⟶	\Longrightarrow	⟹
\mapsto	↦	\mid	∣	\models	⊨
\ne	≠	\nearrow	↗	\neq	≠
\ni	∋	\not=	≠	\nwarrow	↖
\parallel	∥	\perp	⊥	\prec	≺
\preceq	⪯	\propto	∝	\rightarrow	→
\Rightarrow	⇒	\rightharpoondown	⇁	\rightharpoonup	⇀
\rightleftharpoons	⇌	\searrow	↘	\sim	∼
\simeq	≃	\smile	⌣	\sqsubset	⊏
\sqsubseteq	⊑	\sqsupset	⊐	\sqsupseteq	⊒
\subset	⊂	\subseteq	⊆	\succ	≻
\succeq	⪰	\supset	⊃	\supseteq	⊇
\swarrow	↙	\to	→	\uparrow	↑
\Uparrow	⇑	\vdash	⊢		

Tabella 1.9 Operatori funzionali

Operatori senza limiti				
\arccos	\arcsin	\arctan	\arg	\cos
\cosh	\cot	\coth	\csc	\deg
\dim	\exp	\hom	\ker	\lg
\ln	\log	\sec	\sin	\sinh
\tan	\tanh			
Operatori con limiti				
\det	\gcd	\inf	\lim	\liminf
\limsup	\max	\min	\Pr	\sup

Tabella 1.10 Simboli matematici diversi

\aleph	ℵ	\angle	∠	\backslash	\
\bot	⊥	\Box	□	\clubsuit	♣
\Diamond	◇	\diamondsuit	♢	\ell	ℓ
\emptyset	∅	\exists	∃	\flat	♭
\forall	∀	\hbar	ℏ	\heartsuit	♡
\Im	ℑ	\imath	ı	\infty	∞
\jmath	ȷ	\mho	℧	\nabla	∇
\natural	♮	\neg	¬	\partial	∂
\prime	′	\Re	ℜ	\sharp	♯
\spadesuit	♠	\surd	√	\top	⊤
\triangle	△	\wp	℘	\\|	‖

modo testo. Tuttavia merita segnalare i comandi nativi `\mskip` e `\mkern` che accettano come argomenti (non racchiusi fra nessun tipo di parentesi) degli spazi espressi mediante le unità di misura chiamate `mu`, valide solo in matematica; 18 mu equivalgono a 1 em, ma in matematica l'unità 'em' cambia grandezza a seconda del font in uso, in particolare per il corpo principale della formula, per gli apici e i pedici di primo ordine e per quelli di secondo ordine.

1.6.5 Font matematici

Per avere una intera formula composta con caratteri medi o con caratteri neri, bisogna specificarlo prima di entrare in modo matematico:

```
\boldmath
⟨ambiente matematico⟩
\unboldmath
```

Altrimenti i vari simboli letterali e/o gli operatori possono essere resi con font diversi se si usano i comandi seguenti:

`\mathrm`	tondo	`\mathsf`	senza grazie
`\mathit`	*corsivo*	`\mathtt`	`spaziatura fissa`
`\mathbf`	**nero**	`\mathcal`	$\mathcal{CALLIGRAFICO}$

Tabella 1.11 Prima serie di simboli accessibili con il pacchetto `amsmath`

\ulcorner	⌜	\urcorner	⌝	\llcorner	⌞
\lrcorner	⌟	\dashrightarrow	⇢	\dashleftarrow	⇠
\widehat	̂	\widetilde	̃	\dasharrow	⇢
\rightleftharpoons	⇌	\leftrightharpoons	⇋	\angle	∠
\hbar	ℏ	\sqsubset	⊏	\sqsupset	⊐
\mho	℧	\square	□	\lozenge	◊
\vartriangleright	▷	\vartriangleleft	◁	\trianglerighteq	⊵
\trianglelefteq	⊴	\circleddash	⊝	\boxdot	⊡
\boxplus	⊞	\boxtimes	⊠	\boxminus	⊟
\blacksquare	■	\centerdot	·	\blacklozenge	⧫
\circlearrowright	↻	\circlearrowleft	↺	\Vdash	⊩
\Vvdash	⊪	\vDash	⊨	\twoheadrightarrow	↠
\twoheadleftarrow	↞	\leftleftarrows	⇇	\rightrightarrows	⇉
\upuparrows	⇈	\downdownarrows	⇊	\upharpoonright	↾
\downharpoonright	⇂	\upharpoonleft	↿	\downharpoonleft	⇃
\rightarrowtail	↣	\leftarrowtail	↢	\leftrightarrows	⇆
\rightleftarrows	⇄	\Lsh	↰	\Rsh	↱
\rightsquigarrow	⇝	\leftrightsquigarrow	↭	\looparrowleft	↫
\looparrowright	↬	\circeq	≗	\succsim	≿
\gtrsim	≳	\gtrapprox	⪆	\multimap	⊸
\therefore	∴	\because	∵	\doteqdot	≑
\triangleq	≜	\precsim	≾	\lesssim	≲
\lessapprox	⪅	\eqslantless	⪕	\eqslantgtr	⪖
\curlyeqprec	⋞	\curlyeqsucc	⋟	\preccurlyeq	≼
\leqq	≦	\leqslant	⩽	\lessgtr	≶
\backprime	‵	\risingdotseq	≓	\fallingdotseq	≒
\succcurlyeq	≽	\geqq	≧	\geqslant	⩾
\gtrless	≷	\vartriangleright	▷	\vartriangleleft	◁
\trianglerighteq	⊵	\trianglelefteq	⊴	\bigstar	★
\between	≬	\blacktriangledown	▼	\blacktriangleright	▶
\blacktriangleleft	◀	\vartriangle	△	\blacktriangle	▲
\triangledown	▽	\eqcirc	≖	\lesseqgtr	⋚
\gtreqless	⋛	\lesseqqgtr	⪋	\gtreqqless	⪌
\Rrightarrow	⇛	\Lleftarrow	⇚	\veebar	⊻
\barwedge	⊼	\doublebarwedge	⩞	\measuredangle	∡
\sphericalangle	∢	\varpropto	∝	\smallsmile	⌣
\smallfrown	⌢	\Subset	⋐	\Supset	⋑
\Cup	⋓	\Cap	⋒	\curlywedge	⋏
\curlyvee	⋎	\leftthreetimes	⋋	\rightthreetimes	⋌
\subseteqq	⫅	\supseteqq	⫆	\bumpeq	≏
\Bumpeq	≎	\lll	⋘	\ggg	⋙
\circledS	Ⓢ	\pitchfork	⋔	\dotplus	∔
\backsim	∽	\backsimeq	⋍	\complement	∁
\intercal	⊺	\circledcirc	⊚	\circledast	⊛

Tabella 1.12 Seconda serie di simboli accessibili con il pacchetto `amsmath`

`\lvertneqq`	$\lvertneqq$	`\gvertneqq`	$\gvertneqq$	`\nleq`	$\nleq$
`\ngeq`	$\ngeq$	`\nless`	$\nless$	`\ngtr`	$\ngtr$
`\nprec`	$\nprec$	`\nsucc`	$\nsucc$	`\lneqq`	$\lneqq$
`\gneqq`	$\gneqq$	`\nleqslant`	$\nleqslant$	`\ngeqslant`	$\ngeqslant$
`\lneq`	$\lneq$	`\gneq`	$\gneq$	`\npreceq`	$\npreceq$
`\nsucceq`	$\nsucceq$	`\precnsim`	$\precnsim$	`\succnsim`	$\succnsim$
`\lnsim`	$\lnsim$	`\gnsim`	$\gnsim$	`\nleqq`	$\nleqq$
`\ngeqq`	$\ngeqq$	`\precneqq`	$\precneqq$	`\succneqq`	$\succneqq$
`\precnapprox`	$\precnapprox$	`\succnapprox`	$\succnapprox$	`\lnapprox`	$\lnapprox$
`\gnapprox`	$\gnapprox$	`\nsim`	$\nsim$	`\ncong`	$\ncong$
`\diagup`	$\diagup$	`\diagdown`	$\diagdown$	`\varsubsetneq`	$\varsubsetneq$
`\varsupsetneq`	$\varsupsetneq$	`\nsubseteqq`	$\nsubseteqq$	`\nsupseteqq`	$\nsupseteqq$
`\subsetneqq`	$\subsetneqq$	`\supsetneqq`	$\supsetneqq$	`\varsubsetneqq`	$\varsubsetneqq$
`\varsupsetneqq`	$\varsupsetneqq$	`\subsetneq`	$\subsetneq$	`\supsetneq`	$\supsetneq$
`\nsubseteq`	$\nsubseteq$	`\nsupseteq`	$\nsupseteq$	`\nparallel`	$\nparallel$
`\nmid`	$\nmid$	`\nshortmid`	$\nshortmid$	`\nshortparallel`	$\nshortparallel$
`\nvdash`	$\nvdash$	`\nVdash`	$\nVdash$	`\nvDash`	$\nvDash$
`\nVDash`	$\nVDash$	`\ntrianglerighteq`	$\ntrianglerighteq$	`\ntrianglelefteq`	$\ntrianglelefteq$
`\ntriangleleft`	$\ntriangleleft$	`\ntriangleright`	$\ntriangleright$	`\nleftarrow`	$\nleftarrow$
`\nrightarrow`	$\nrightarrow$	`\nLeftarrow`	$\nLeftarrow$	`\nRightarrow`	$\nRightarrow$
`\nLeftrightarrow`	$\nLeftrightarrow$	`\nleftrightarrow`	$\nleftrightarrow$	`\divideontimes`	$\divideontimes$
`\varnothing`	$\varnothing$	`\nexists`	$\nexists$	`\Finv`	$\Finv$
`\Game`	$\Game$	`\eth`	$\eth$	`\eqsim`	$\eqsim$
`\beth`	$\beth$	`\gimel`	$\gimel$	`\daleth`	$\daleth$
`\lessdot`	$\lessdot$	`\gtrdot`	$\gtrdot$	`\ltimes`	$\ltimes$
`\rtimes`	$\rtimes$	`\shortmid`	$\shortmid$	`\shortparallel`	$\shortparallel$
`\smallsetminus`	$\smallsetminus$	`\thicksim`	$\thicksim$	`\thickapprox`	$\thickapprox$
`\approxeq`	$\approxeq$	`\succapprox`	$\succapprox$	`\precapprox`	$\precapprox$
`\curvearrowleft`	$\curvearrowleft$	`\curvearrowright`	$\curvearrowright$	`\digamma`	$\digamma$
`\varkappa`	$\varkappa$	`\Bbbk`	$\Bbbk$	`\hslash`	$\hslash$
`\backepsilon`	$\backepsilon$				

Si noti che il corsivo matematico, il corsivo testuale usato in matematica mediante il comando `\mathit`, e il corsivo per il testo sono font con proprietà completamente diverse; per esempio, il primo non contiene legature, per cui la parola *affine* verrebbe composta $affine$. Più dettagliatamente gli

esempi seguenti mostrano il diverso comportamento dei tre corsivi.

`\textit{affine, affine affine}`	*affine, affine affine*
`\( \mathit{affine, affine affine} \)`	$affine, affineaffine$
`\( affine, affine affine \)`	$affine, affineaffine$

Si noti che i comandi testuali, come `\textit`, possono essere usati anche in matematica producendo semplice testo; mentre i comandi che scelgono i font in matematica lo fanno solo per le lettere e gli altri segni equivalenti alle lettere. Il corsivo matematico trae i suoi segni da una polizza diversa da quella del corsivo testuale e non usa le legature. Si noti ancora che lo spazio dopo la virgola, quando si scrive usando il comando `\mathit` e quando si usa il corsivo matematico, appare esplicitamente perché, anche se nel contesto di questa guida si è usata la 'virgola intelligente', la virgola nei due casi non è seguita da una cifra e quindi si comporta coma un segno di punteggiatura con uno spazietto alla sua destra che, come si vede, è più piccolo dello spazio interparola che si presenta usando il comando `\textit`.

1.6.6 Stili di composizione

I quattro stili di composizione della matematica sono

```
\displaystyle
\textstyle
\scriptstyle
\scriptscriptstyle
```

Lo stile `\textstyle` differisce dal `\displaystyle` nel senso che è più raccolto in verticale: gli apici e i pedici sono più vicini all'asse matematico; i limiti superiori e inferiori sono composti accanto all'operatore come se fossero apici o pedici; le frazioni sono composte in `\scriptstyle` per mantenere limitato l'ingombro verticale in modo che non sia necessario spaziare le righe del testo. È per questo che bisogna stare attenti, specialmente con le frazioni, a non renderle troppo piccole; piuttosto che una frazione troppo piccola, è preferibile una barra obliqua, senza rimpicciolire numeratori e denominatori, nemmeno quando sono puramente numerici, come in $^2/_3$; questa pratica è 'deprecata' dalla norma UNI 2950.

1.7 Definizioni, numeri e programmazione

1.7.1 Comandi di definizione

Nuove macro possono essere definite, ridefinite o rese disponibili, se non lo sono già, mediante i comandi:

```
\newcommand{⟨comando⟩}[⟨numero argomenti⟩][⟨default⟩]{⟨definizione⟩}
\renewcommand{⟨comando⟩}[⟨numero argomenti⟩][⟨default⟩]{⟨definizione⟩}
\providecommand{⟨comando⟩}[⟨numero argomenti⟩][⟨default⟩]{⟨definizione⟩}
```

Sono tutti comandi fragili, ma del resto non sembra opportuno usarli negli argomenti di altri comandi. La loro sintassi è autoesplicativa, ma conviene ricordare alcuni punti.

Innanzi tutto, i tre comandi suddetti definiscono macro "lunghe", cioè che possono ricevere come argomenti un testo formato da diversi capoversi; se al comando di definizione si aggiunge un asterisco, la definizione è "corta", cioè nessuno dei suoi argomenti può contenere qualcosa che possa essere o rappresentare un comando di fine capoverso. L'uso dell'asterisco è altamente raccomandabile; uno degli errori frequenti, infatti, è quello di dimenticare la parentesi graffa chiusa dopo avere scritto un argomento di una certa lunghezza. È chiaro che si tratta di un errore, e il programma di composizione lo segnala ma se il comando è "corto", lo segnala subito al primo segno di fine di capoverso; se invece il comando è lungo, il messaggio d'errore può arrivare alla fine del file (rendendo quindi difficile trovare dove ci si è dimenticati di inserire la graffa chiusa) oppure può provocare la saturazione del buffer d'entrata, il che genera messaggi d'errore di difficile comprensione. Tutte le volte che si definisce un nuovo comando, ci si ricordi sempre di questo fatto e si scelga di inserire l'asterisco per tutti i comandi che richiedono argomenti se si è sicuri che non devono essere formati da più di un capoverso.

La macro definita con questi comandi accetta uno e un solo argomento facoltativo dichiarato con ⟨*default*⟩; nella ⟨*definizione*⟩, questo argomento è sempre il primo. Se non c'è la coppia di quadre con il valore ⟨*default*⟩ (eventualmente vuoto) non ci sono argomenti facoltativi. All'interno della ⟨*definizione*⟩, gli argomenti sono tutti indicati con il loro numero d'ordine preceduto dal segno '`#`': `#1`, `#2`, fino eventualmente a `#9`. Nessun comando può accettare più di nove argomenti.

`\newcommand` produce un messaggio d'errore se la macro da definire è già definita; al contrario, `\renewcommand` produce un messaggio d'errore se la macro da definire non esiste ancora; `\providecommand` definisce la macro se essa non è stata ancora definita, ma se esiste già non fa nulla.

Vale la pena di ricordare che un grosso gruppo di lavoro sta lavorando da alcuni anni alla definizione di una sintassi più robusta di quella di LaTeX 2_ε, così da poter definire degli spazi di nomenclatura tali da ridurre, se non eliminare, ogni possibilità di conflitto fra vari pacchetti e che costituirà l'ossatura su cui si baserà il futuro LaTeX 3; il linguaggio si chiama (per ora) L3 ed è già parte integrante di molti pacchetti di servizio che si usano in continuazione senza che ci rendiamo conto che dietro le quinte c'è un linguaggio diverso; si suggerisce di leggere più avanti il paragrafo 1.16.2, dove si usa indirettamente il nuovo linguaggio tramite un pacchetto che consente di definire nuovi comandi o ambienti, o di ridefinirne di vecchi, con una sintassi che è sostanzialmente diversa da quella esposta qui, ma moto più potente e tale da permettere caratterizzazioni molto sottili di ciascun argomento.

1.7.2 Comandi per la definizione di ambienti

Gli ambienti sono definiti o ridefiniti mediante i comandi

```
\newenvironment{⟨nome⟩}[⟨numero argomenti⟩][⟨default⟩]%
                    {⟨comandi di apertura⟩}{⟨comandi di chiusura⟩}
\renewenvironment{⟨nome⟩}[⟨numero argomenti⟩][⟨default⟩]%
                    {⟨comandi di apertura⟩}{⟨comandi di chiusura⟩}
```

In sostanza, la definizione di un ambiente equivale alla definizione di due comandi: \⟨*nome*⟩ e `\end`⟨*nome*⟩; questo impone delle limitazioni ai nomi degli ambienti, in particolare ⟨*nome*⟩ deve essere diverso da "relax", da "end", e da qualunque altro ⟨*nome*⟩ di cui sia già definita una macro col ⟨*nome*⟩ preceduto da `\` o da `\end`; la diagnostica di LaTeX sotto questo aspetto è ottima ma, per esempio, si rimane perplessi se si volesse definire un ambiente di ⟨*nome*⟩ pari a `group`, ma è corretto che sia così, perché è vero che non esiste nel nucleo di LaTeX il comando `\group`, ma esiste il comando `\endgroup`.

Per gli ambienti solo il comando di apertura può ricevere un argomento facoltativo, come se fosse definito con `\newcommand`.

1.7.3 Teoremi

LaTeX mette a disposizione dei comandi per definire degli ambienti al fine di comporre gli enunciati dei teoremi o di altri enunciati dello stesso genere. Il comando `\newtheorem` consente due forme di definizione:

```
\newtheorem{⟨nome⟩}{⟨titolino⟩}[⟨contatore dominante⟩]
\newtheorem{⟨nome⟩}[⟨numerato come⟩]{⟨titolino⟩}
```

dove:

⟨*nome*⟩ rappresenta il nome dell'ambiente; potrà essere, per esempio, 'teorema', 'lemma', 'definizione', eccetera. Per cui, se ⟨*nome*⟩ si riferisce a un `teorema`, il suo enunciato verrà racchiuso all'interno di `\begin{teorema}` ⟨*enunciato*⟩ `\end{teorema}`.

⟨*titolino*⟩ è la parola che identifica l'enunciato; potrà essere 'Teorema', 'Lemma', 'Definizione', eccetera. Non si confonda il nome dell'ambiente con il titolino dell'enunciato.

⟨*contatore dominante*⟩ è il contatore del capitolo, se si vuole che la numerazione del nuovo enunciato sia iniziata da 1 ogni nuovo capitolo; sarà il contatore dei paragrafi, se si desidera che la numerazione ricominci da 1 ogni nuovo paragrafo; eccetera.

⟨*numerato come*⟩ è invece il nome dell'ambiente del quale si vuole condividere la numerazione. Questo deve essere già stato definito. Si potrebbe, per esempio, numerare con una sola sequenza numerica sia i teoremi sia i lemmi sia i corollari, mentre, probabilmente, si preferirebbe numerare separatamente le definizioni, le congetture, eccetera.

Se non si specificano i contatori facoltativi, questi comandi definiscono un nuovo contatore con lo stesso nome dell'ambiente che può venire stampato usando il comando `\the`*nome*. Nella stessa maniera i comandi `\label` e `\ref` usano il nuovo nome del contatore per rendere accessibile ai riferimenti incrociati anche questi enunciati. Se invece si specifica il contatore ⟨*numerato come*⟩, non viene introdotto nessun nuovo contatore, ma si usa quello già esistente. Infine se si specifica il nome del ⟨*contatore dominante*⟩, il comando `\the`*nome* produrrà in stampa il numero del contatore relativo all'ambiente preceduto dal numero stampato del ⟨*contatore dominante*⟩; se, per esempio,

il ⟨*contatore dominante*⟩ fosse il paragrafo, allora il tredicesimo enunciato del quarto paragrafo del decimo capitolo verrà stampato nella forma 10.4.13.

Fra i pacchetti resi disponibili dalla American Mathematical Society c'è il pacchetto `amsthm`, che estende le possibilità di definire pseudoambienti del tipo dei teoremi con altri comandi e altre impostazioni molto utili.

1.7.4 Numeri, lunghezze e spazi

Il sistema TEX, per comporre, svolge una quantità di calcoli sia usando numeri veri e propri, sia usando lunghezze e spazi.

TEX fa calcoli solo con numeri interi; gli unici numeri non interi che esso gestisce sono i fattori moltiplicativi delle lunghezze o degli spazi. Qui di seguito vedremo come esistano registri per conservare i numeri, come agire sui numeri e sui loro registri, chiamati *contatori*.

TEX usa le lunghezze e dispone di registri per la conservazione di informazioni metriche; questi registri non hanno nomi particolari, come i contatori per i numeri, ma possono essere *chiamati per nome*, come si fa per i contatori.

Gli spazi sono speciali lunghezze e i loro registri sono anch'essi un po' speciali; sono chiamati anche "lunghezze elastiche", perché hanno la proprietà di potersi allungare o accorciare rispetto alla loro lunghezza naturale; sono costituiti dalla terna di tre lunghezze la prima rappresenta la lunghezza naturale, la seconda l'ammontare assoluto di cui essa può allungarsi e la terza l'ammontare assoluto con cui può accorciarsi.

Per operare con i numeri e le lunghezze rigide o elastiche, TEX dispone di un certo numero di comandi nativi. L'estensione etex, incorporata nelle versioni più recenti di pdftex, di xetex e di luatex, consente di eseguire certe operazioni in modo molto più comodo; anzi, luatex può anche eseguire calcoli molto complessi con numeri a virgola mobile grazie all'interprete del linguaggio lua che esso incorpora. Altrimenti, non disponendo di una versione recente di pdftex o degli altri motori di composizione, bisogna ricorrere al pacchetto `calc`, alla cui documentazione si rimanda il lettore.

1.7.5 Numeri

A parte il modo di scrivere i numeri già visto nella pagina 36, LATEX consente di fare semplici conti con numeri interi e/o con le misure delle

lunghezze.

`\newcounter` con la sintassi:

> `\newcounter{`⟨*nome*⟩`}[`⟨*contatore dominante*⟩`]`
>
> permette di definire un nuovo contatore per numeri interi chiamato ⟨*nome*⟩, asservito, e quindi azzerato, quando il ⟨*contatore dominante*⟩ viene incrementato. Il ⟨*nome*⟩ e il ⟨*contatore dominante*⟩ hanno lo stesso significato descritto per i teoremi.

`\setcounter` con la sintassi:

> `\setcounter{`⟨*nome*⟩`}{`⟨*valore*⟩`}`
>
> inserisce la quantità numerica ⟨*valore*⟩ nel contatore che ha quel ⟨*nome*⟩.

`\addtocounter` con la sintassi:

> `\addtocounter{`⟨*nome*⟩`}{`⟨*valore*⟩`}`
>
> aggiunge il ⟨*valore*⟩ specificato al contenuto del contatore ⟨*nome*⟩. Il ⟨*valore*⟩ può anche essere negativo, quindi di fatto LaTeX esegue una somma algebrica.

`\value` con la sintassi:

> `\value{`⟨*nome*⟩`}`
>
> recupera il contenuto del contatore ⟨*nome*⟩ per passarlo ad altri comandi o per rendere stampabile il numero contenuto dentro quel contatore.

`\stepcounter` con la sintassi:

> `\stepcounter{`⟨*nome*⟩`}`
>
> incrementa il contatore ⟨*nome*⟩ di una unità.

`\refstepcounter` con la sintassi:

> `\refstepcounter{`⟨*nome*⟩`}`

oltre ad incrementare il contatore ⟨*nome*⟩ di una unità, lo rende accessibile al comando `\label`, così che si possa fare riferimento ai valori del contatore mediante le chiavi usate da `\label` e `\ref`.

`\arabic` con la sintassi:

`\arabic{`⟨*nome*⟩`}`

trasforma il contenuto del contatore ⟨*nome*⟩ nell'equivalente stringa di cifre arabe.

`\roman` con la sintassi:

`\roman{`⟨*nome*⟩`}`

trasforma il contenuto del contatore ⟨*nome*⟩ nell'equivalente stringa di cifre romane minuscole; secondo me è infelice la scelta delle cifre tonde minuscole, mentre sarebbe meglio ricorrere sempre alle cifre minuscole del maiuscoletto; questo per altro non è disponibile in tutte le famiglie, serie e forme, quindi l'utente deve provvedere a mano caso per caso, oppure può definire un falso maiuscoletto per i font che ne mancano, semplicemente riducendo il maiuscolo al corpo corrispondente a `\scriptsize`; si riconosce a colpo d'occhio che si tratta di un "falso", ma talvolta è meglio che niente.

`\Roman` con la sintassi:

`\Roman{`⟨*nome*⟩`}`

trasforma il contenuto del contatore ⟨*nome*⟩ nell'equivalente stringa di cifre romane maiuscole.

`\alph` con la sintassi:

`\alph{`⟨*nome*⟩`}`

trasforma il contenuto del contatore ⟨*nome*⟩ nella lettera dell'alfabeto latino minuscolo corrispondente alla posizione indicata dal contatore; ovviamente, in questo modo si può rappresentare il valore del contatore solo finché il suo contenuto è positivo ma non superiore a 26, essendo 26 le lettere dell'alfabeto latino.

`\Alph` con la sintassi:

`\Alph{`⟨*nome*⟩`}`

trasforma il contenuto del contatore ⟨*nome*⟩ nella lettera dell'alfabeto latino maiuscolo corrispondente alla posizione indicata dal contatore; ovviamente, in questo modo si può rappresentare il valore del contatore solo finché il suo contenuto è positivo ma non superiore a 26, essendo 26 le lettere dell'alfabeto latino.

`\fnsymbol` LaTeX contiene una lista ordinata di simboli: ∗, †, ‡, §, ¶, ‖, ∗∗, ††, ‡‡, dalla quale il comando `\fnsymbol`, con la sintassi:

`\fnsymbol{`⟨*nome*⟩`}`

trae il simbolo corrispondente alla posizione indicata dal contatore ⟨*nome*⟩; questo comando serve solo per i richiami di nota del comando `\thanks`, che può essere usato solo per comporre la lista degli autori, il titolo e la data quando si usa il comando `\maketitle` o quando si usa l'ambiente *titlepage*. Indicando le note con questi simboli, si possono rendere univoche le note con un numero positivo, non superiore a 9, visto che la lista contiene solo nove simboli.

Quando si definisce un nuovo contatore ⟨*nome*⟩, LaTeX definisce anche il modo di stamparlo attraverso il comando `\the`⟨*nome*⟩; in altre parole, se si definisce il nuovo contatore `pippo` mediante il comando:

```
\newcounter{pippo}
```

viene contemporaneamente definito il comando `\thepippo`, in generale in modo molto semplice[7]:

```
\newcommand\thepippo{\arabic{pippo}}
```

Il compositore è libero di ridefinire il modo di scrivere il contatore, specialmente se è asservito a un altro contatore; a titolo di esempio scriverà:

```
\newcounter{pippo}[pluto]
\renewcommand*\thepippo{\arabic{pluto}.\arabic{pippo}}
```

[7] In realtà la definizione normalmente viene eseguita ricorrendo ai comandi nativi, che sono più veloci da eseguire in quanto non hanno bisogno di essere interpretati.

se vuole che il contatore `pippo`, supponendo che valga 3, quando il contatore `pluto` vale 5, venga stampato nella forma 5.3.

Vale la pena di segnalare che il comando nativo `\the` può precedere il nome di qualunque registro interno dell'interprete per trasformarne il contenuto in una stringa di caratteri che ne indica il contenuto. Questo comando nativo non è applicabile ai nomi dei contatori LaTeX, perché i loro nomi non individuano direttamente il registro, come invece succede con le lunghezze. Quando si definisce un contatore LaTeX, viene associato un particolare registro numerico, il primo disponibile, a una sequenza formata con il ⟨*nome*⟩ preceduto alla stringa `\c@`. Quindi, il nome del contatore LaTeX `pippo` serve per definire la sequenza `\c@pippo` che viene associata all'indirizzo numerico del registro interno (di cui il compositore non deve preoccuparsi). Il comando `\the` dovrebbe quindi precedere la sequenza `\c@pippo` così:

```
\the\c@pippo
```

per produrne la stringa di cifre che ne indica il contenuto. Data la presenza del segno `@`, che normalmente non è una lettera, anche se può essere usato come lettera all'interno dei file di classe o di quelli di estensione, questo modo di procedere è fortemente sconsigliato; tra l'altro, è molto più comodo e più robusto rispetto agli errori ricorrere ai comandi specifici, come per esempio `\arabic`, per scrivere il contenuto dei contatori LaTeX.

Infine, va segnalato che qualche volta è necessario modificare gli asservimenti dei contatori come sono definiti nei file di classe, oppure definire nuovi contatori asserviti a contatori definiti dagli utenti.

Per fare queste cose, il nucleo di LaTeX fornisce un paio di comandi `\@addtoreset` e `\@removefromreset`; i loro nomi sono espliciti, ma sono protetti dal carattere `@` che ne rende l'uso delicato, in quanto bisogna ricorrere ai comandi `\makeatletter` e `\makeatother`. Il pacchetto `chngcntr` aggira questo problema definendo i comandi `\counterwithin` e `\counterwithout`; la loro sintassi è la seguente:

`\counterwithin{`⟨*contatore subordinato*⟩`}{`⟨*contatore principale*⟩`}`
`\counterwithout{`⟨*contatore subordinato*⟩`}{`⟨*contatore principale*⟩`}`

Alcune classi come *memoir* aggiungono queste macro alla loro dotazione predefinita, ridefinendole in modo compatibile con gli altri loro comandi e

senza bisogno di caricare il pacchetto suddetto. Con la distribuzione del 2018 di TeX Live, i comandi `\counterwithin` e `\counterwithout` sono stati aggiunti nel nucleo di LaTeX, per cui non è più necessario ricorrere al pacchetto `chngcntr` come nel passato; però, per compatibilità, se si carica questo pacchetto con una distribuzione successiva al 2017, il programma di composizione semplicemente lo ignora, visto che già dispone di quei comandi.

Per esempio: si vogliono numerare le equazioni subordinate alla numerazione dei paragrafi e si vuole usare la classe *book*, dove esse sono già subordinate alla numerazione dei capitoli. Bisogna rimuovere, quindi, il contatore `equation` dalla lista di reset del contatore `chapter` e poi aggiungerlo alla lista di reset del contatore `section`; il codice da usare diventa perciò:

```
\counterwithout{equation}{chapter}
\counterwithin{equation}{section}
\renewcommand*{\theequation}{\thesection.\arabic{equation}}
```

Dopo queste impostazioni specificate nel preambolo del proprio documento, la terza equazione del secondo paragrafo del quarto capitolo viene numerata 4.2.3.

1.7.6 Lunghezze

Le lunghezze esplicite, quelle, cioè, che si esprimono con una misura e una unità di misura, sono costituite da una sequenza di cifre decimali, separate dal separatore decimale che in teoria potrebbe anche essere la virgola; siccome nella sintassi LaTeX la virgola svolge anche altri ruoli, è preferibile usare sempre e soltanto il punto decimale quando si esprimono le misure delle lunghezze. Un numero che contenga una parte intera nulla potrebbe cominciare con il punto decimale, omettendo lo zero che dovrebbe precederlo; si sconsiglia questa abbreviazione del tutto irrilevante ma che facilita gli errori. Il numero può essere preceduto dal segno positivo o negativo; quello positivo è evidentemente facoltativo; quello negativo invece non lo è. Se, in seguito alla sostituzione di espressioni e argomenti nelle macro, la misura risultasse preceduta da diversi segni, il numero di segni negativi si comporta come ci si aspetterebbe, nel senso che ognuno cambia segno a quanto segue, quindi un numero dispari di segni negativi porta a

un risultato complessivamente negativo, mentre un numero pari porta a un risultato positivo.

Le lunghezze definite con i comandi LaTeX sono sempre lunghezze elastiche, anche se i loro allungamenti e accorciamenti sono nulli; di fatto si comportano come lunghezze rigide, a meno che nella definizione non vengano specificati anche l'allungamento e/o l'accorciamento.

Unità di misura Le unità di misura accettabili da LaTeX sono le seguenti:

- `cm` centimetri.
- `mm` millimetri.
- `in` pollici.
- `pt` punti tipografici ($1\,\text{pt} = (1/72,27)\,\text{in} = 0,3515\,\text{mm}$; $72,27\,\text{pt} = 1\,\text{in}$; $2.84528\,\text{pt} = 1\,\text{mm}$).
- `pc` pica ($1\,\text{pc} = 12\,\text{pt}$).
- `bp` punto PostScript o *big point* ($1\,\text{bp} = (1/72)\,\text{in} = 0,3528\,\text{mm}$).
- `sp` *scaled point* (la frazione 2^{-16} di un punto tipografico).
- `dd` punto didot (1 dd=0,3760 mm=1,070 pt).
- `cc` cicero (1 cc=12 dd).
- `nd` nuovo punto didot (1 nd= 0.375 mm) disponibile con il motore pdftex.
- `nc` nuovo cicero (1 nc=12 nd) disponibile con il motore pdftex.
- `ex` *x-height*, occhio mediano del carattere corrente, ovvero altezza della lettera 'x'.
- `em` *em-width*, larghezza della 'M', convenzionalmente circa uguale al corpo del font corrente. In realtà, font diversi dello stesso corpo possono avere valori di `1em` diversi, nessuno dei quali legato all'effettiva larghezza della lettera M; per esempio: con il corpo di 10 pt il font con grazie, il font senza grazie e il font monospaziato medi diritti hanno i valori di `1em` che valgono rispettivamente 10.0pt, 10.0pt, 10.5pt; le rispettive larghezze della lettera M sono 9.17pt, 8.75pt, 5.24998pt.

`\fill` è una lunghezza elastica (gomma) di lunghezza naturale nulla ma infinitamente allungabile.

`\stretch{`⟨*moltiplicatore*⟩`}` è una lunghezza elastica (gomma) di lunghezza naturale nulla ma allungabile infinitamente quanto `\fill` moltiplicata per ⟨*moltiplicatore*⟩; questo è un numero decimale con segno facolta-

tivo; se il segno è negativo, questa lunghezza diventa 'infinitamente' accorciabile.

`\newlength{`⟨*comando*⟩`}` definisce una nuova lunghezza identificata con ⟨*comando*⟩; questo è il nome di un tipico comando LaTeX formato da una stringa letterale preceduta dal backslash, oppure da un solo segno non letterale preceduto dal backslash. Si possono definire fino a 256 lunghezze; in realtà, il motore di composizione pdftex (sufficientemente recente) accetta la definizione di un numero molto maggiore di lunghezze. Per poter allocare registri di lunghezza in numero maggiore di 256, è necessario caricare il pacchetto `etex`. È vero che i registri con un indirizzo superiore a 255 sono accessibili anche con i comandi di basso livello (nativi) del motore di composizione, ma è sconsigliabile farlo a meno di non essere in condizioni protette e purché si sia consoci di quello che si sta facendo.

`\setlength{`⟨*comando*⟩`}{`⟨*lunghezza*⟩`}` serve per assegnare la ⟨*lunghezza*⟩ esplicita al registro-lunghezza identificato con ⟨*comando*⟩.

`\addtolength`⟨*comando*⟩⟨*lunghezza*⟩ serve per incrementare il valore di lunghezza contenuto nel registro ⟨*comando*⟩ della quantità ⟨*lunghezza*⟩, che può essere sia positiva, sia negativa; nel caso, si ha una sottrazione.

`\settowidth`, `\settoheight` e `\settodepth` seguono la sintassi seguente:

```
\settowidth{⟨comando⟩}{⟨testo⟩}
\settoheight{⟨comando⟩}{⟨testo⟩}
\settodepth{⟨comando⟩}{⟨testo⟩}
```

Ognuna di queste istruzioni assegna al registro-lunghezza ⟨*comando*⟩ rispettivamente la larghezza, l'altezza o la profondità della stringa che costituisce il ⟨*testo*⟩.

1.7.7 Spaziature

Gli spazi orizzontali e verticali possono venire inseriti a mano con i seguenti comandi:

```
\hspace{⟨lunghezza⟩}
\hspace*{⟨lunghezza⟩}
\vspace{⟨lunghezza⟩}
\vspace*{⟨lunghezza⟩}
```

I comandi con asterisco impediscono che gli spazi siano eliminati alla fine o all'inizio di una riga oppure all'inizio o alla fine di una pagina, come invece avviene con gli spazi 'semplici'. Questa possibilità di essere eliminati è essenziale per la buona composizione di una pagina e delle sue righe, ma può non essere quello che si desidera quando si compone dentro una scatola oppure quando si eseguono composizioni speciali: si pensi, per esempio, a un frontespizio dove non si vuole inserire il primo elemento scritto all'inizio della gabbia, ma lo si desidera spostare in basso per l'equilibrio della composizione.

Molto comodi risultano i comandi abbreviati

```
\bigskip
\medskip
\smallskip
```

che equivalgono rispettivamente a:

```
\vspace{\bigskipamount}
\vspace{\medskipamount}
\vspace{\smallskipamount}
```

Le lunghezze indicate con i tre comandi `\...skipamount` sono definite nei file di classe; generalmente esse hanno i valori di 12 pt, 6 pt e 3 pt.

Il comando

`\addvspace{`⟨*lunghezza*⟩`}`

aggiunge spazio verticale tenendo conto di eventuale spazio verticale già inserito implicitamente da comandi precedenti, così che alla fine lo spazio complessivo aggiunto ammonti esattamente alla ⟨*lunghezza*⟩ specificata.

I comandi `\hfill` e `\vfill` equivalgono rispettivamente ai comandi `\hspace{\fill}` e `\vspace{\fill}`.

1.7.8 Operazioni fra numeri e grandezze

Le recenti estensioni del programma pdftex includono tutte le estensioni del precedente programma etex; una descrizione completa è contenuta nel documento `/textmf-distr/doc/etex/base/etex_man.pdf`. Quello che ora ci interessa di più è la sezione 3.5 di quel documento, intitolata *Expressions*.

Vediamo dunque che ora pdftex è in grado di calcolare alcune espressioni matematiche direttamente in memoria (nei registri della CPU del calcolatore che si sta usando). Il risultato di queste espressioni è usabile in qualunque momento sia lecito usare un risultato numerico o dimensionale, per esempio assegnandolo a un registro o inserendolo in una espressione logica di confronto fra numeri o fra lunghezze.

Le espressioni dimensionali `\dimexpr` e quelle numeriche `\numexpr` possono venire mescolate; non è sempre obbligatorio terminare ogni espressione di ciascun tipo con `\relax`, ma è consigliabile. Esistono espressioni anche per le lunghezze elastiche e per le lunghezze matematiche: `\glueexpr` e `\muexpr`; nelle espressioni si possono usare le parentesi tonde col solito significato. L'esempio che il testo propone è il seguente:

```
\ifdim\dimexpr (2pt-5pt)*\numexpr 3-3*13/5\relax+34pt/2<\wd20
```

Vi compare una espressione mista dimensionale che contiene al suo interno una espressione numerica; il comando `\relax` serve per terminare l'espressione numerica; quella dimensionale termina con il primo carattere non valido in una espressione, vale a dire che termina subito prima del segno di minore `<`.

Le operazioni valide all'interno delle espressioni sono i segni delle quattro operazioni aritmetiche: `+`, `-`, `*`, e `/`

Le operazioni numeriche sono eseguite fra numeri interi (o con l'indicazione di contatori TeX oppure contatori LaTeX passati come argomento di `\value`). Le divisioni, quindi, sono divisioni intere, cioè forniscono solo la parte intera del quoziente che potremmo calcolare con una calcolatrice ordinaria (o anche a mano).

Le operazioni dimensionali sono le stesse e sono del tutto equivalenti a quelle che potrebbero venire eseguite usando i comandi nativi di pdftex, vale a dire `\advance` per la somma algebrica, `\multiply` per la moltiplicazione e `\divide` per la divisione intera.

Ci sono però due novità.

1. Mentre le operazioni di somma, sottrazione e moltiplicazione fra numeri interi danno sempre un risultato intero, l'operazione di divisione con quoziente intero implica sempre la perdita di un po' di informazione; con i comandi nativi di pdftex le divisioni vengono sempre troncate; con le espressioni numeriche e dimensionali di etex i risultati

vengono arrotondati, quindi il risultato che si ottiene può essere per eccesso o per difetto.

2. Le operazioni di scalamento di una lunghezza, cioè della moltiplicazione di una lunghezza per una frazione, sono intese come il prodotto della lunghezza da scalare per il numeratore della frazione il cui risultato viene poi diviso per il denominatore della frazione; numeratore e denominatore possono essere numeri interi (eventualmente contenuti in contatori) oppure altre due lunghezze; non possono essere uno un numero e l'altro una lunghezza, ma devono essere entrambi dello stesso tipo.

L'operazione di scalamento viene fatta mettendo il risultato della prima moltiplicazione in un registro della CPU di 64 bit, il doppio di una normale parola di 4 byte, e il risultato della successiva divisione viene riportato a una parola di 32 bit, cioè di 4 byte. Nel fare questo viene eseguito l'arrotondamento.

Vediamo un esempio: se la giustezza `\textwidth` dovesse venire divisa per sei al fine di comporre a sei colonne, bisogna togliere cinque volte lo spazio intercolonna e dividere il risultato per sei; il tutto va poi assegnato alla lunghezza `\columnwidth`; scriveremo allora[8]:

```
\newlength\Numer \newlength\Denom
...
\setlength\Numer{1pt} \setlength\Denom{6pt}
\columnwidth=\dimexpr(\textwidth -5\columnsep)*\Numer/\Denom\relax
```

Si noti che moltiplicare per 1 pt e dividere per 6 pt non è la stessa cosa di moltiplicare per 1 e dividere per 6. Nella codifica interna, la lunghezza di 1 pt è data dal numero intero di *scaled points* corrispondenti appunto a 1 pt, quindi da un numero binario formato da un 1 seguito da sedici zeri binari; analogamente, il numero sei è rappresento dalle tre cifre binarie `110` seguite da sedici zeri binari; la prima moltiplicazione pertanto sposta a sinistra di sedici posizioni il numero del risultato della prima sottoespressione, poi viene diviso per sei e il risultato viene scalato a destra di sedici posizioni, portando a un suo eventuale arrotondamento. Moltiplicare semplicemente per uno e dividere semplicemente per sei non porta con sé nessuno scorrimento delle cifre binarie e l'arrotondamento di un numero binario che dispone di meno

[8] Usando la sintassi di basso livello di pdftex.

cifre. Infatti, eseguendo l'operazione indicata sopra, l'espressione calcolata con `\dimexpr` vale 62,08069 pt, mentre quella calcolata con gli operatori precedenti all'introduzione delle estensioni di etex vale 62,08067 pt. In questo caso la differenza è trascurabile, ma in altri casi potrebbe non esserlo.

Si noti, infine, che le operazioni di scalamento vengono eseguite correttamente se si usano i registri lunghezza (o i registri numerici) piuttosto che le grandezze esplicite; o meglio, per il numeratore della frazione che rappresenta il fattore di scala si possono usare anche grandezze esplicite, mentre per il denominatore è preferibile, per non dire necessario, usare solo dei registri lunghezza.

Vale la pena di indicare alcune particolarità dei calcoli che pdftex esegue e dei registri in cui salva certe quantità; è importante anche al fine di rendersi conto della delicatezza dei calcoli e dell'eventualità di eseguirne alcuni che portano o alla perdita di ogni informazione o all'*overflow*.

I registri interi, cioè i contatori, sono parole di 4 byte e contengono 32 bit o cifre binarie. Un bit è riservato per il segno, quindi restano 31 bit per rappresentare i numeri. Ne segue che il massimo numero intero vale $2^{31} - 1 = 2\,147\,483\,647$; chiaramente, un numero sufficientemente grande da dubitare di poterlo superare.

I registri per le lunghezze sono ugualmente dei registri interi dove viene memorizzato il numero di *scaled points* corrispondenti. Si tratta di parole di 4 byte che contengono 32 bit; due di questi bit servono per il segno e per altre informazioni specifiche relative alle lunghezze; restano quindi 30 bit per memorizzare la grandezza espressa mediante un numero intero di *scaled points* il cui valore assoluto massimo può quindi arrivare a $1\,073\,741\,823$. Siccome 2^{16} sp corrispondono a 1 pt, la massima lunghezza espressa in punti che può venire memorizzata vale 16 383,99999 pt. Questo valore, tra l'altro, viene memorizzato nel registro di lunghezza `\maxdimen` nel caso il compositore/programmatore volesse eseguire dei confronti o utilizzare questo numero speciale.

Questa massima lunghezza corrisponde a poco più di 5758 mm, quasi sei metri, e sembra che anche questo valore in tipografia sia difficilmente raggiungibile. Purtroppo non è vero. Si pensi infatti ai calcoli che bisognerebbe fare per determinare una lunghezza che sia in proporzione con

un'altra lunghezza con lo stesso rapporto di altre due lunghezze:

$$\frac{l_1}{l_2} = \frac{L_1}{L_2} \qquad \text{cioè} \qquad l_1 = l_2 \frac{L_1}{L_2}$$

Per fare un esempio pratico, si supponga di avere a disposizione una carta di formato non standard con base L_2 e altezza L_1 da specificare nel preambolo assegnandone i valori ai registri di lunghezza `\paperwidth` e `\paperheight`; in base al font usato, si determina che la leggibilità è ottimale se la base della griglia interna `\textwidth` vale l_2; quanto deve valere `\textheight` affinché il testo sia inscritto in un rettangolo simile al rettangolo della carta?

Con le espressioni di dimensione che abbiamo appena visto possiamo determinare:

```
\textheight=\dimexpr \textwidth * \paperheight / \paperwidth \relax
```

e pdftex ci esegue i calcoli senza problemi grazie ai risultati interni conservati un un registro di due parole, cioè di 64 bit.

Cosa sarebbe successo se avessimo usato i comandi nativi `\multiply` e `\divide`? Per rendercene conto è meglio fare un esempio numerico. Supponiamo che la carta abbia le dimensioni di 176 mm per 250 mm (la carta UNI B5). Il calcolo della giustezza sulla base dei font che andiamo ad usare ci dice che `\textwidth` dovrebbe valere 120 mm. Eseguendo i calcoli con una calcolatrice tascabile (a virgola mobile) avremmo[9]:

$$\begin{aligned} x &= l_2 \cdot L_1 = 120 \times 250 = 30000 \gg 5758 \\ l_1 &= x / L_2 = 30000/176 = 170.454545 \end{aligned}$$

Come si vede, il risultato finale è perfettamente accettabile, ma la moltiplicazione intermedia eccede le possibilità dei registri interni di pdftex e quindi si ha un *overflow*.

[9]Non si badi al fatto che le indicazioni contengano solo le misure e non le unità di misura. È voluto per due motivi: (*a*) l'elaboratore usa solamente numeri, e (*b*) qui non si vuole annoiare il lettore con lunghe scritture di sequenze di bit o di cifre esadecimali o ottali; il concetto è quello di mostrare come si svolgono i conti, anche se qui usiamo le misure in millimetri, non dimentichiamoci che i contatori numerici non possono contenere un numero equivalente di millimetri superiore a poco più di 5758.

Ah, se è così, allora eseguiamo prima la divisione e poi la moltiplicazione:

$$y = l_2/L_2 = 120/176 = 0$$
$$l_1 = y \cdot L_1 = 0 \times 250 = 0$$

ma in questo modo la divisione intera di un numero per un numero più grande ci dà la parte intera del quoziente, che vale evidentemente zero e perdiamo ogni informazione.

Solo usando le espressioni estese mediante il comando `\dimexpr` possiamo arrivare al risultato senza problemi, ma dobbiamo stare attenti. Infatti conviene sempre stimare i conti che si devono fare (come fatto sopra) affinché si possa essere sicuri che, nonostante la moltiplicazione sia effettuata salvando il risultato in un registro temporaneo a 64 bit, non venga poi eseguita una divisione con un divisore troppo piccolo. Il calcolo che è stato descritto sopra diventa[10]:

```
{\makeatletter
\textwidth=120mm \relax
\textheight=
   \dimexpr \textwidth * \paperheight / \paperwidth \relax
\dimen@=0.3514598\textheight
\strip@pt\dimen@\,mm}
```

otteniamo il valore 170.45253 mm che presenta, per gli arrotondamenti eseguiti, una differenza di qualche unità sulla sesta cifra significativa rispetto al calcolo eseguito con molta maggior precisione con una macchinetta tascabile[11].

[10] Questo calcolo è racchiuso in un gruppo, cosicché la assegnazioni di valori alla grandezze in gioco restano locali e, usciti dal gruppo, i valori preesistenti vengono ripristinati. Questo testo è composto su carta B5, quindi i valori di `\paperheight` e di `\paperwidth` sono già specificati dalla classe. Il valore 0.35145980 serve per trasformare una misura in punti nel corrispondente valore in millimetri, ed è dato dal calcolo $(25,4\,\text{mm/in})/(72,27\,\text{pt/in}) = 0.35145980\,\text{mm/pt}$; `\dimen@` è un registro dimensionale temporaneo usato solo per scrivere il valore in millimetri della giustezza; il comando `\strip@pt` serve per convertire il contenuto del registro dimensionale in una stringa di caratteri da cui toglie le unità di misura 'pt'.

[11] Anche se non è difficile stimare a mente a quanti millimetri corrisponde una lunghezza espressa in punti tipografici (basta dividere il numero di punti per tre e si ha una stima approssimata per difetto con un 'errore' del 5%), io uso spesso questa piccolissima

Ecco, quindi, dove nasce l'enorme comodità dell'operazione di scalamento eseguita da pdftex in un registro interno di 64 bit. Questi inconvenienti (*overflow* o perdita di significato dei dati) possono però succedere anche con pdftex quando si esegue la divisione finale, se il divisore è troppo piccolo; in questo caso, pdftex non può fare a meno di emettere un messaggio di errore, ma si può proseguire con le dita incrociate; i calcoli saranno sbagliati, quindi andrà corretto l'errore prima di ottenere risultati ragionevoli, ma lo si potrà fare senza perdere tutte le informazioni accessorie che pdftex è in grado di raccogliere o di generare durante la sua esecuzione.

Inconvenienti di questo genere possono succedere più sovente di quanto si immagini, quindi è opportuno fare sempre una analisi preventiva dei calcoli da eseguire, almeno come ordini di grandezza, per non ritrovarsi senza la possibilità di lasciare fare i conti a pdftex con macro capaci di funzionare con qualunque insieme di valori "ragionevoli" degli argomenti.

1.7.9 Il pacchetto `ifthen`

Il pacchetto `ifthen` permette di accedere ai comandi nativi di controllo del flusso delle informazioni di cui TEX è capace. I comandi messi a disposizione da questo pacchetto sono i seguenti.

`\ifthenelse` con la sintassi:

`\ifthenelse{`⟨*test*⟩`}{`⟨*esegui se è vero*⟩`}{`⟨*esegui se è falso*⟩`}`

esegue il ⟨*test*⟩ e se questo ⟨*test*⟩ restituisce il valore 'vero' allora vengono passati al flusso di informazioni da elaborare i token che formano il contenuto del primo argomento dopo il test, ⟨*esegui se è vero*⟩. Altrimenti, vengono eseguiti i token che formano il testo di ⟨*esegui se è falso*⟩.
I ⟨*test*⟩ che si possono eseguire sono i seguenti.

confronto numerico con la sintassi:

⟨$numero_1$⟩ **`operatore`** ⟨$numero_2$⟩

espressione per mostrare i valori di lunghezza espressi in millimetri, visto che io e, suppongo, la totalità dei lettori abbiamo più facilmente la sensazione delle dimensioni espresse in millimetri; in ogni caso, tutti disponiamo di righe o righelli graduati in millimetri, e assai pochi dispongono di un tipometro graduato in punti tipografici inglesi.

dove *operatore* è uguale a `>`, oppure `=`, oppure `<`; i numeri ⟨*numero*$_i$⟩ possono essere i contenuti di due contatori o di un contatore e di una quantità esplicita. Per i contatori LaTeX bisogna specificarne il valore attraverso la macro `\value`.

CONFRONTO DI STRINGHE il comando `\equal` con la sintassi

```
\equal{⟨stringa1⟩}{⟨stringa2⟩}
```

confronta due stringhe e se sono assolutamente identiche il test è vero, altrimenti è falso.

CONFRONTO DI LUNGHEZZE il comando `\lengthtest` con la sintassi

```
\lengthtest{⟨lungh1 operatore lungh2⟩}
```

confronta due lunghezze (generalmente almeno una delle due è contenuta in un registro-lunghezza) e restituisce il valore 'vero' se le due lunghezze stanno nella relazione implicata dall'*operatore*. Questo può essere un solo segno matematico fra `=`, `>`, oppure `<`.

CONFRONTO DI PARITÀ il comando `\isodd` con la sintassi

```
\isodd{⟨numero⟩}
```

controlla se un numero (generalmente contenuto in un contatore) sia dispari.

VERIFICA DI DEFINIZIONE il comando `\isundefined` con la sintassi

```
\isundefined{⟨\comando⟩}
```

restituisce il valore vero se ⟨*comando*⟩ non è mai stato definito.

STATO DI UNA VARIABILE BOOLEANA il comando `\boolean` controlla lo stato di una variabile booleana; la sintassi per gestire queste variabili è la seguente.

```
\newboolean{⟨variabile booleana⟩}
\provideboolean{⟨variabile booleana⟩}
\setboolean{⟨variabile booleana⟩}{⟨stato⟩}
\boolean{⟨variabile booleana⟩}
```

dove ⟨*stato*⟩ è una delle due parole `true` (vero) oppure `false` (falso).

In realtà, `\boolean` è in grado di verificare lo stato anche delle variabili booleane interne a LaTeX e anche di quelle definite con i comandi elementari di TeX o di Plain TeX. Con quest'ultimo si

definisce un nuovo comando logico con `\newif`, il quale accetta come argomento il comando e contemporaneamente definisce due altri comandi per impostare lo stato vero o falso della variabile booleana. In pratica, se si volesse definire una nuova variabile booleana '`test`', con il pacchetto `ifthen` si dovrebbero usare i comandi `\newboolean`, oppure `\provideboolean` che esegue la stessa definizione che esegue `\newboolean` senza però ripeterla se la ⟨*variabile booleana*⟩ esiste già:

```
\newboolean{test}
...
\setboolean{test}{true}
...
\ifthenelse{\boolean{test}}{\textbf{Pippo}}{\textit{Pluto}}
```

Se invece si usassero i comandi di Plain TEX (accessibili anche quando si usa LATEX) si dovrebbe scrivere

```
\newif\iftest
...
\testtrue
...
\iftest\textbf{Pippo}\else\textit{Pluto}\fi
```

Si è fatto questo esempio non tanto per invitare a usare i comandi elementari di Plain TEX, quanto per permettere di capire come funzionano i test elementari che si trovano scritti a piene mani nei comandi definiti nei file di formato, di classe e di estensione. Usati male, i comandi di Plain TEX possono comportarsi in modo bizzarro; i test gestiti con il pacchetto `ifthen` sono più affidabili, ma non sono privi qualche bizzarria; i test gestiti con le macro del pacchetto `etoolbox` sono i migliori.

Segnalo anche che le distribuzioni moderne del sistema TEX includono le estensioni di ε-TEX. Tra le altre cose vengono introdotti molti nuovi comandi condizionali, e per conoscerli rimando alla documentazione mediate il comando `texdoc etex`.

operatori logici `\and`, `\or` e `\not` permettono di mettere insieme diverse frasi logiche da collegare fra di loro; l'intera frase comprendente gli operatori deve essere racchiusa fra `\(` e `\)`.

`\whiledo` consente di descrivere e realizzare un ciclo 'while'; la sintassi è

`\whiledo{`⟨*test*⟩`}{`⟨*ciclo*⟩`}`

Questo comando ripete il ⟨*ciclo*⟩ fino a quando il ⟨*test*⟩, inizialmente vero, diventa falso. Va da sé che, prima di iniziare la ripetizione di ⟨*ciclo*⟩, gli elementi da cui dipende ⟨*test*⟩ devono essere inizializzati in modo tale che ⟨*test*⟩ sia vero. Il ⟨*ciclo*⟩ deve contenere delle istruzioni o dei comandi che prima o poi rendano il ⟨*test*⟩ falso, altrimenti LaTeX entra in un ciclo infinito e non ne esce più.

A titolo di esempio dell'uso di `\ifthenelse`, nel nucleo di LaTeX viene definito il comando `\cleardoublepage` mediante la sintassi di basso livello TeX:

```
\def\cleardoublepage{\clearpage\if@twoside \ifodd\c@page\else
    \hbox{}\newpage\if@twocolumn\hbox{}\newpage\fi\fi\fi}
```

Usando il pacchetto `ifthen`, questa definizione potrebbe essere tradotta in:

```
\newcommand*\cleardoublepage%
{\clearpage
  \ifthenelse{\(\boolean{@twoside}\and\not\isodd{\value{page}}\)}{%
      \mbox{}\newpage
      \ifthenelse{\boolean{@twocolumn}}{\mbox{}\newpage}{}%
                                    }{}%
}
```

La scrittura è un poco più complessa, ma si vedono meglio gli effetti e la natura dei test; il primo controlla se stiamo componendo fronte e retro e se siamo su una pagina pari; è chiaro che se non stiamo componendo fronte e retro, saltare una pagina per ricominciare da una pagina dispari, una pagina di destra, non ha molto senso; inoltre, è chiaro che se la pagina è dispari non bisogna fare nulla. Infine controlla se si sta componendo su due colonne: in questo caso esegue comunque un `\newpage` che, a due colonne, vuol dire di terminare una colonna e ricominciare a comporre nella colonna successiva[12]. L'incolonnamento dei contenuti delle varie clausole e delle relative parentesi graffe permette di seguire meglio i successivi passi.

[12] Il secondo `\ifthenelse` può essere omesso se il comando precedente, invece di `\newpage`, fosse nuovamente `\clearpage`.

Per chi volesse cimentarsi nella scrittura di macro per costruire file di estensione o di classe, suggerirei di servirsi del pacchetto `etoolbox`, che mette a disposizione comandi dalle simili funzionalità ma robusti. Anche se esiste un pacchetto `xifthen` che estende la funzionalità del pacchetto `ifthen`, io ritengo che le macro di `etoolbox` siano di gran lunga migliori e coprano situazioni che vanno ben al di là della gestione dei test e della logica booleana.

Va notato che anche le estensioni etex incorporate in pdftex, permettono di eseguire molto efficacemente diversi test in modo molto semplice; per esempio, per sapere se la macro `\pippo` è già stata definita, con il linguaggio nativo di TEX bisognerebbe scrivere:

```
\expandafter\ifx\csname pippo\endcsname\relax A\else B\fi
```

Con il linguaggio etex avremmo invece:

```
\ifcsname pippo\endcsname B\else A\fi
```

Con il pacchetto `ifthen` avremmo:

```
\ifthenelse{\ifundefined{\pippo}}{A}{B}
```

Con il pacchetto `etoolbox` avremmo:

```
\ifdef\pippo{B}{A}
```

Per controllare il rapporto esistente fra due numeri è comodo il comando `\unless`, definito nelle estensioni di etex, che nega il risultato di un test. È particolarmente indicato quando si devono fare test su numeri e lunghezze: infatti, i controlli che si possono fare su queste entità sono solo quelle indicate da `=`, `>` e `<`, e non sono disponibili comandi come $\geq$ $\leq$ o $\neq$; infatti, per controllare se un valore è maggiore o uguale a un altro, con i test normali non si può se non scambiando l'operatore di confronto; quindi se si vuole fare qualcosa se il valore contenuto nel contatore `pippo` è superiore o uguale al valore contenuto nel contatore `pluto`, si può certamente scrivere

`\ifdim \pippo<\pluto\else`⟨*fai qualcosa*⟩`\fi`

ma personalmente trovo più chiara la scrittura:

`\unless\ifdim \pippo<\pluto`⟨*fai qualcosa*⟩`\fi`

Con il pacchetto `ifthen` si potrebbe usare l'operatore logico `\not`. Con il pacchetto `etoolbox` i confronti fra numeri e fra grandezze sono ancora più numerosi di quelli che si possono eseguire con Plain TeX, `etex`, `ifthen`.

Per documentarsi meglio sulle estensioni di `etex`, si rinvia il lettore alla sua documentazione contenuta nel file `etex-man.pdf`; per documentarsi sul pacchetto `etoolbox`, la sua documentazione si trova nel file `etoolbox.pdf`.

1.8 Figure, tabelle e altri oggetti flottanti

I comandi per rendere flottanti le figure e le tabelle sono descritti nei prossimi paragrafi.

1.8.1 Figure e tabelle

`\begin{figure}[`⟨*posizione*⟩`]` ⟨*figura*⟩ `\end{figure}`
`\begin{figure*}[`⟨*posizione*⟩`]` ⟨*figura*⟩ `\end{figure*}`
`\begin{table}[`⟨*posizione*⟩`]` ⟨*tabella*⟩ `\end{table}`
`\begin{table*}[`⟨*posizione*⟩`]` ⟨*tabella*⟩ `\end{table*}`

I comandi senza asterisco vanno bene sia componendo a una colonna, sia componendo a due colonne. I comandi con l'asterisco inseriscono l'oggetto a larghezza piena in testa a una pagina composta a due colonne; componendo a una colonna, si può usare indifferentemente la versione con o quella senza asterisco.

Le posizioni possibili sono

`t`	in testa alla pagina
`b`	al piede della pagina
`h`	'qui', se possibile...
`p`	in una pagina di soli oggetti flottanti

Con tutte le classi standard, se l'argomento facoltativo ⟨*posizione*⟩ non è specificato, le posizioni di default sono `tbp`.

La lista delle posizioni in realtà accetta anche `!`, che invita LaTeX a 'mettercela tutta' per collocare l'oggetto nel posto più vicino a dove è stato definito.

LaTeX può mettere un filetto sotto una figura in testa alla pagina, oppure sopra una figura in calce alla pagina; può mettere anche un filetto sotto

una figura a giustezza piena in testa a una pagina composta a due colonne. I comandi per mettere questi filetti sono `\topfigrule`, `\botfigrule` e `\dblfigrule`; di default questi tre comandi sono equivalenti a `\relax`, quindi non fanno niente. Ogni classe, oppure ogni compositore, li può ridefinire in modo che mettano qualcosa, generalmente un filetto, ma potrebbe essere una qualunque decorazione, nella posizione a cui si riferiscono, ma questo qualcosa deve avere altezza *nulla* allo stesso modo del filetto che separa le note in calce dal testo sovrastante. La definizione di questo filetto `\footnoterule` è la seguente:

```
\def\footnoterule{\kern-3\p@ \hrule \@width 2in \kern 2.6\p@}
```

e basta definire i comandi di questi filetti cambiando il nome `\footnoterule` con `\topfigrule`, o `\botfigrule`, o `\dblfigrule`; per esempio, mettendo un filetto che attraversa l'intero specchio di stampa, si possono definire:

```
\def\topfigrule{\kern2.6\p@ \hrule \kern -3\p@}
\def\botfigrule{\kern-3\p@ \hrule  \kern 2.6\p@}
\def\dblfigrule{\kern2.6\p@ \hrule \kern -3\p@}
```

Si noti l'espediente per fare sembrare il filetto di altezza nulla; si arretra di 3pt, si mette un filetto spesso 0,4pt, e si avanza di 2,6pt; complessivamente non si è avanzato in verticale di nessuno spazio. Gli spostamenti in avanti o all'indietro di 2,6pt e di 3pt sono messi strategicamente in modo che lo spostamento positivo separi il filetto da ciò che esso segue o precede.

LaTeX, durante la costruzione della pagina di uscita, segue dei criteri di ottimalità per collocare gli oggetti flottanti, eventualmente in attesa di essere usati appena nella pagina c'è abbastanza spazio, oppure quando viene usato il comando `\clearpage`. Questi criteri implicano il rispetto di certe frazioni minime o massime di spazio dedicato a questa o quella parte della pagina, nonché il numero massimo di oggetti che possono essere collocati in una pagina. Le regole che LaTeX segue sono descritte qui di seguito, ma è chiaro che se si vuole che LaTeX faccia diversamente da come è stato programmato, bisogna comprendere appieno queste regole e bisogna capire come funzionano i parametri massimi o minimi che vi presiedono.

1. LaTeX colloca un oggetto flottante nel primo posto che non viola le regole seguenti, salvo che, se è specificato il codice `h`, questo ha la precedenza sul codice `t`.

2. LaTeX non collocherà mai un oggetto flottante in una pagina che viene prima di quella dove compare il testo del file sorgente adiacente all'ambiente di flottaggio.
3. LaTeX colloca le figure in ordine e lo stesso fa con le tabelle, per cui non può succedere che la figura 22 appaia prima della figura 21.
   Va notato, però, che quando si compone a due colonne (senza usare il pacchetto `multicol`[13], ma usando la specifica LaTeX `\twocolumn`), possono formarsi pagine che hanno figure o tabelle a giustezza piena contemporaneamente a una o più figure o tabelle con la giustezza di una colonna. In questi casi, visto che gli oggetti a piena giustezza vengono collocati solo in testa[14], mentre gli oggetti con la giustezza di una colonna possono essere collocati sia in testa alla colonna, sia in calce, sia in una posizione qualsiasi consentita dai parametri di posizione, può succedere che l'oggetto in testa abbia un numero successivo a quello di un oggetto in colonna. Per ovviare a questo inconveniente era necessario usare un piccolo pacchetto di correzioni del nucleo di LaTeX, `fixltx2e`, che eseguiva alcune modifiche alla routine di emissione della pagina composta, onde evitare questa e poche altre apparenti incongruenze presenti nel nucleo standard; dalla versione 2015 delle distribuzioni del sistema TeX non è più necessario ricorrere a questo pacchetto, perché le sue funzionalità sono incorporate nel nucleo di LaTeX, quindi non può più accadere che un oggetto flottante in testa abbia un numero superiore a un oggetto flottante della stessa specie in colonna.
4. LaTeX colloca gli oggetti flottanti solamente nelle posizioni specificate nell'argomento facoltativo ⟨*posizione*⟩ o, in mancanza di questo, in una delle tre posizioni di default `tbp`.
5. LaTeX non colloca mai un oggetto flottante in una pagina che non contiene abbastanza spazio, quindi non produce mai una '`Overfull vbox`' a causa degli oggetti flottanti. Be', a meno che un oggetto flottante di per sé non sia così grande da eccedere lo spazio disponibile; a questo l'utente deve provvedere preventivamente mediante le opportune operazioni di scalamento e gli altri artifici che si possono

[13] Questo pacchetto non consente di mettere figure all'interno delle colonne.

[14] Mediante il pacchetto `dblfloatfix`, sarebbe possibile mettere gli oggetti flottanti a piena pagina anche in calce alla pagina, non solo in testa.

mettere in opera grazie alle macro `\scalebox` e `\resizebox` definite dal pacchetto `graphicx`.

6. I vincoli imposti dai parametri stilistici tipici di ogni classe non vengono mai violati; quando viene specificato `!`, vengono rilassati i vincoli imposti dai parametri che si riferiscono a pagine contenenti sia testo sia oggetti flottanti, mentre LaTeX continua a rispettare i vincoli imposti alle pagine che contengono solamente oggetti flottanti. Per queste pagine di soli oggetti flottanti le regole vengono ignorate quando si emettono i comandi `\clearpage` oppure `\cleardoublepage`, e, ovviamente, quando si incontra `\end{document}`, perché questi comandi e la specifica che il documento è terminato ordinano a LaTeX di svuotare le code di tutto ciò che si trova ancora in memoria.
   Ci si ricordi che `\cleardoublepage` e `\end{document}` eseguono loro stessi un comando `\clearpage`; inoltre, i comandi di sezionamento a livello più alto, come `\part` e `\chapter`, eseguono loro stessi il comando `\cleardoublepage`.
7. Se fra i codici di posizionamento compare solo `h` con o senza il punto esclamativo, e per qualche motivo il programma non lo può inserire proprio nella pagina e nel luogo richiesto, lo accoda alla lista degli oggetti in attesa e da quel momento in poi lo tratta come un oggetto flottante da inserire nella posizione `t`. Se il motivo per non mettere l'oggetto nella posizione richiesta era la sua dimensione verticale, e questa preclude la posizione `t`, quell'oggetto blocca la coda fino a quando il programma non incontra esplicitamente o implicitamente il comando `\clearpage`.
8. I limiti dimensionali, di cui parlerà fra poco, devono essere rispettati da ogni oggetto flottante, non solo da quelli col posizionamento `h`; se un qualsiasi oggetto flottante non li rispetta, blocca la coda.

Quando si specifica la ⟨*posizione*⟩, bisogna dare abbastanza possibilità a LaTeX di fare il suo mestiere, altrimenti l'oggetto che non ha altre possibilità che una sola, blocca le code fino alla fine del documento o alla prima esecuzione di `\clearpage`; questi comandi vengono emessi automaticamente quando si inizia un nuovo capitolo, ma, a parte che è brutto vedere tutte, o quasi tutte, le figure di un capitolo accumulate alla sua fine, bisogna ricordare che la memoria di LaTeX è programmata per memorizzare

solamente 52 oggetti flottanti[15]; se dovessero accodarsene di più, quelli in eccesso verrebbero persi e verrebbe emesso un messaggio che avvisa del fatto, ma è una magra consolazione. Si esamini per altro la possibilità di usare il pacchetto `morefloats`.

I comandi che si possono usare dentro gli ambienti mobili tipicamente sono:

`\caption` con la sintassi

> `\caption[`⟨*didascalia breve*⟩`]{`⟨*didascalia*⟩`}`
>
> dove, se non si specifica la ⟨*didascalia breve*⟩, questa viene automaticamente resa identica alla didascalia 'lunga'; potrebbe non essere una buona idea quella di inviare alla lista delle figure o delle tabelle l'intera didascalia, specialmente se questa contiene uno o più periodi dopo il primo che svolge il compito di titolo, mentre i periodi successivi svolgono il compito di fornire maggiori delucidazioni sull'oggetto specifico. Si sottolinea che `\caption` può essere usato solo dentro gli ambienti mobili. Se si vuole mettere una didascalia in un ambiente fisso si può ricorrere al comando `\captionof` fornito da diverse classi e/o pacchetti, il più semplice dei quali è `capt-of`.

`\label` può essere usato, anzi è conveniente che sia usato, per ogni figura e per ogni tabella; il comando deve essere collocato dopo il comando `\caption` perché, finché `\caption` non viene eseguito, all'oggetto flottante non è ancora stato assegnato un numero.

`\suppressfloats` con la sintassi

> `\suppressfloats[`⟨*posizione*⟩`]`
>
> può essere collocato fuori degli ambienti di flottaggio al fine di evitare che LaTeX metta altri oggetti nella stessa ⟨*posizione*⟩ specificata come argomento facoltativo; se non si specifica nulla, tutti gli oggetti flottanti sono esclusi dalla pagina corrente. Se però nella ⟨*posizione*⟩ del comando di apertura degli ambienti *figure* o *table* compare `!`, allora `\suppressfloats` viene ignorato.

[15] Prima del 2015, il numero massimo di float in coda era solamente 18.

I parametri dimensionali e numerici che regolano il deflusso degli oggetti flottanti dalle rispettive code sono i seguenti.

`topnumber` è il nome del contatore che contiene il numero *massimo* di oggetti flottanti che possono essere collocati in testa alla pagina.

`\topfraction` è la *massima* frazione di pagina destinata agli oggetti flottanti in testa a una pagina, se questa contiene anche del testo.

`bottomnumber` è il contatore che contiene il numero *massimo* di oggetti flottanti in calce alla pagina.

`\bottomfraction` è la *massima* frazione di pagina destinata agli oggetti flottanti in calce a una pagina, se questa contiene anche del testo.

`totalnumber` è il contatore che contiene il *massimo* numero di oggetti flottanti che possono comparire in una pagina di testo, indipendentemente dal fatto che che siano in testa o in calce o in mezzo al testo.

`\textfraction` per una pagina che contenga anche del testo, è la frazione *minima* della pagina destinata al testo.

`\floatpagefraction` in una pagina di soli oggetti flottanti rappresenta la *minima* frazione di pagina che deve essere occupata da questi oggetti; se cioè si specifica il parametro di posizione `p` per un oggetto flottante e questo è troppo piccolo, esso viene trattenuto in memoria finché non si trova un altro oggetto flottante dello stesso genere e con lo stesso attributo di posizione che assieme possano superare questo valore minimo specificato.

`dbltopnumber` è il contatore che contiene il numero *massimo* di oggetti flottanti a piena pagina da mettere in testa a una pagina con il testo composto su due colonne.

`\dbltopfraction` la *massima* frazione di pagina a giustezza piena da destinare agli oggetti flottanti in testa alla pagina quando si compone a due colonne.

`\dblfloatpagefraction` è la *minima* frazione di pagina da occupare con oggetti flottanti a giustezza piena quando si compone una pagina di soli oggetti flottanti in un testo composto a due colonne. È, insomma, l'analogo di `\floatpagefraction` che, invece, vale quando si compone il testo a una sola colonna.

`\floatsep` è la distanza *minima* da riempire con uno spazio verticale incolore fra due oggetti flottanti consecutivi.

Tabella 1.13 Parametri nella classe *book* composta in corpo 10 per gestire gli oggetti flottanti

Numeri		**Frazioni**	
`topnumber`	2	`\topfraction`	0,7
`bottomnumber`	1	`\bottomfraction`	0,3
`totalnumber`	3	`\textfraction`	0,2
`dbltopnumber`	2	`\floatpagefraction`	0,5
		`\dbltopfraction`	0,7
		`\dblfloatpagefraction`	0,5

Separatori	
`\floatsep`	`12pt plus 2pt minus 2pt`
`\textfloatsep`	`20pt plus 2pt minus 4pt`
`\intextsep`	`12pt plus 2pt minus 2pt`
`\dblfloatsep`	`12pt plus 2pt minus 2pt`
`\dbltextfloatsep`	`20pt plus 2pt minus 4pt`

`\textfloatsep` è la distanza *minima* da interporre fra gli oggetti flottanti in testa o in calce e il testo adiacente.

`\intextsep` è la distanza *minima* fra un oggetto flottante a centro pagina e il testo che lo precede e il testo che lo segue.

`\dblfloatsep` è la distanza *minima* fra due oggetti flottanti consecutivi in testa a una pagina composta a due colonne.

`\dbltextfloatsep` è la distanza *minima* posta fra gli oggetti flottanti in testa a una pagina composta a due colonne e il testo sottostante.

I contatori si impostano con i comandi specifici, in particolare `\setcounter`; le frazioni `\...fraction` sono valori decimali, in generale fratti e minori dell'unità che vengono conservati dentro le macro con i rispettivi nomi, e quindi tutti si reimpostano, per modificarne i valori di default, mediante `\renewcommand`. I separatori `\...sep` sono lunghezze e si impostano con il comando `\setlength`. Ogni classe definisce i suoi particolari valori per questi parametri; per la classe *book* composta in corpo 10 i parametri sono riportati nella tabella 1.13.

Va detto che i 'Numeri' e le 'Frazioni' indicati nelle tabelle 1.13 non

devono necessariamente essere coerenti; sembra strano, ma è comprensibile. Per i 'Numeri', per esempio, `totalnumber` sembra essere la somma di `topnumber` e `bottomnumber`; se però `totalnumber` fosse posto al valore `2`, questo vorrebbe dire che *al massimo* ci possono essere `2` oggetti flottanti in testa e uno in calce, ma in ogni caso non possono essere tutti e tre presenti, perché nella pagina ce ne possono essere *al massimo* due. Lo stesso vale per le frazioni; esse rappresentano dei limiti per delle disuguaglianze; questi limiti non debbono necessariamente avere per somma l'unità; l'importante è che queste disuguaglianze, esaminate a una a una secondo una precisa sequenza, siano tutte rispettate via via che LaTeX esegue la routine di uscita quando provvede alla collocazione degli oggetti flottanti.

Il compositore si ricordi, specialmente per quel che riguarda le frazioni, che nel computo dell'ingombro dell'oggetto flottante bisogna tenere conto anche dello spazio destinato alla didascalia e del suo spazio di separazione. Bisogna anche tenere conto degli spazi che separano ogni oggetto inserito nella pagina da quanto li circonda, sia esso un altro oggetto o del testo. Spesso il compositore è deluso dalla rigidezza con cui LaTeX gestisce gli oggetti flottanti; in realtà LaTeX, come tutti i programmi, si comporta come gli è stato prescritto, in particolare, in questo caso, come richiesto dai 'Numeri', dalle 'Frazioni' e dai 'Separatori' che compaiono nella tabella 1.13. Se non piacciono quei valori, li si può cambiare, ma il compositore stia attento che la cura non sia peggio del male. Piuttosto, egli dedichi la sua attenzione al dimensionamento degli oggetti flottanti, all'eliminazione di spazi inutili al loro contorno, all'eliminazione delle parti inutili delle fotografie che si trovano spessissimo ai margini laterali o verticali; insomma, usi appropriatamente le possibilità offerte dalle opzioni di `\includegraphics`, perché è con quelle che si curano davvero sia il deflusso degli oggetti flottanti dalle rispettive code, sia la qualità del documento prodotto.

1.8.2 Note marginali

I comandi che portano alla composizione e alla collocazione delle note marginali sono i seguenti:

`\marginpar[`⟨*nota di sinistra del testo*⟩`]{`⟨*nota di destra del testo*⟩`}`
`\reversemarginpar`
`\normalmarginpar`

La posizione normale delle note quando si compone a una colonna è il margine esterno; quando si compone a due colonne è il margine adiacente alla colonna alla quale la nota si riferisce, quindi il margine di sinistra per la colonna di sinistra e il margine di destra per la colonna di destra; e questo avviene indipendentemente dal fatto che ci si trovi in una pagina di sinistra o di destra. È evidente che se si vuole comporre a due colonne e si vogliono usare le note marginali, bisogna che entrambi i margini ai lati dello specchio di stampa siano sufficientemente larghi da poter contenere dette note; ci vuole quindi un layout di pagina sicuramente non standard, da predisporre con il pacchetto `geometry`.

`\reversemarginpar` scambia i margini per le note composte ai margini di testi composti a una sola colonna e `\normalmarginpar` ripristina la collocazione di default. Questi due comandi sono inattivi quando si compone a due colonne. L'uso di `\reversemarginpar` implica un generoso margine interno; il margine interno nella geometria della pagina è solitamente minore del margine esterno, quindi anche l'uso di questo comando implica un disegno della pagina particolare e diverso dai disegni standard. Benché quindi sia possibile usare questo comando, è raccomandabile pensarci due volte prima di usarlo senza conoscerne le implicazioni.

Per il testo delle note che dovrebbero apparire a sinistra del testo a cui si riferiscono potrebbe essere applicata la dichiarazione `\raggedleft` per avere una composizione in bandiera giustificata a destra. Tuttavia, questo è consigliabile solo quando le annotazioni marginali sono sempre molto brevi in modo da non superare una riga. Cambiare giustificazione da nota a nota sarebbe una caduta di uniformità e di stile. D'altra parte, le note di più righe giustificate solo a destra sono leggermente più laboriose da leggere rispetto alle note giustificate solo a sinistra; il compositore ci pensi bene prima di decidere come comporre le note; eventualmente si crei una macro che componga sempre entrambi i testi di sinistra e di destra uniformemente con lo stesso stile, in modo da non dover operare a mano così da evitare i possibili errori di disomogeneità.

I parametri che governano la collocazione delle note, a parte il margine 'normale' o 'scambiato', sono:

`\marginparwidth` è la giustezza delle note marginali e deve essere evidentemente minore della larghezza del margine fisico a lato del testo.

`\marginparsep` è lo spazio di separazione fra il testo e il blocchetto della nota marginale.

`\marginparpush` è lo spazio con cui una nota marginale viene spostata in basso rispetto alla fine della nota marginale precedente; talvolta questo obbliga a portare la nota nella pagina successiva, ma viene emesso un avvertimento sullo schermo e nel file `.log`. Per riparare queste situazioni non tanto desiderabili, bisogna necessariamente riformulare il testo principale a cui la nota in questione si riferisce, oppure, bisogna accorciare il testo della nota marginale precedente.

Un avvertimento: LaTeX tratta le note marginali alla stessa stregua degli oggetti flottanti; questo vuol dire che usa le stesse scatole per conservare in memoria gli oggetti flottanti; siccome le scatole a disposizione sono solo 18, l'uso di frequenti note marginali e di molte figure e/o tabelle aumenta il rischio di esaurire la memoria a disposizione e di perdere degli oggetti flottanti.

Dal 2015 si può fare uso del pacchetto `morefloats` che estende 'infinitamente' il numero di oggetti flottanti in attesa di essere emessi dalla memoria del calcolatore; permette di arrivare fino al massimo consentito dalla memora del calcolatore in uso. Il sistema TeX che sto usando dispone di una memoria di tre milioni di parole di 4 byte per svolgere il suo lavoro, ma per comporre questo documento ne usa solamente circa un decimo.

1.9 Incolonnamenti

1.9.1 L'ambiente *tabbing*

Quando TeX 78 nacque, le macchine da scrivere meccaniche ed elettriche erano ancora diffusissime e le tabulazioni eseguite con il tasto di tabulazione, nonché con gli arresti di tabulazione, erano familiari a tutti. Tuttavia l'ambiente *tabbing*, nato, forse, proprio per affidarsi a qualcosa di noto a qualunque dattilografo, presenta dei notevoli inconvenienti, primo fra i quali quello di essere un ambiente fragile che non può essere annidato dentro altri ambienti.

La sintassi è quella comune a qualunque ambiente:

`\begin{tabbing}` ⟨*testo da tabulare*⟩ `\end{tabbing}`

I comandi per gestire gli arresti di tabulazione sono i seguenti.

`\=` serve per impostare un arresto di tabulazione; spesso conviene scrivere una riga modello da *non* comporre se si ricorre al comando seguente.

`\kill` non compone la riga che precede questa parola chiave, ma conserva le informazioni di impostazione degli arresti di tabulazione che la riga conteneva.

`\>` serve per spostare la composizione al successivo arresto di tabulazione.

`\\` comincia una nuova riga riportando il contatore degli arresti di tabulazione al valore iniziale, generalmente zero.

`\+` incrementa di una unità il valore iniziale del contatore di tabulazione così che da questo momento in poi e fino ad ordine contrario, tutte le righe risultano rientrate e incolonnate sotto un arresto presente nella riga precedente.

`\-` decrementa di una unità il valore iniziale del contatore di tabulazione; contrasta l'effetto di `\+`.

`\<` torna indietro di un arresto di tabulazione e può essere usato solamente all'inizio di una riga.

`\'` serve per spostare tutto il contenuto di una 'colonna' a filo del margine destro del suo campo di tabulazione.

`\`` serve per comporre il contenuto della sua colonna a filo del margine sinistro del suo arresto di tabulazione.

`\pushtabs` manda in memoria le posizioni degli attuali arresti di tabulazione, consentendo di impostarne di diversi.

`\poptabs` serve per richiamare dalla memoria un insieme di arresti di tabulazione precedentemente memorizzati.

`\a=` `\a'` e `\a`` agiscono come `\=`, `\'` e `\`` per comporre gli accenti macron (lungo), acuto e grave rispettivamente, perché questi comandi, come si vede sopra, sono stati ridefiniti all'interno dell'ambiente per svolgere funzioni diverse. Il compositore che usa il pacchetto `inputenc` non se ne preoccupi e continui a usare tranquillamente le lettere accentate della sua tastiera, perché quel pacchetto provvede alla compatibilità senza che il compositore debba preoccuparsene più di tanto. Il compositore deve purtroppo occuparsene se e solo se la sua tastiera è priva di segni accentati e se il suo *shell editor* non consente di usare combinazioni di tasti per inserire direttamente nel file `.tex` i segni accentati, oppure, infine, se ha impostato il suo *shell editor* in modo che salvi i file in modo tale che contengano solo caratteri ASCII codificati con 7 bit, così da assicurare la massima portabilità

cross-platform.

Va però notato quanto segue. Prima del 2018, la distribuzione di TEX Live (e di MiKTEX) era tale che quando incontrava un carattere diverso dai caratteri ASCII, produceva effetti strani durante la composizione mediante pdflatex, se non era stato usato il pacchetto `inputenc` con l'opzione giusta per la codifica. Dal 2018 TEX Live e, probabilmente, MiKTEX, interpretano i caratteri diversi da quelli ASCII come se fossero codificati con la codifica *utf8*; questo fatto di per sé non elimina certi effetti strani; ma gli editor moderni sono quasi tutti preimpostati per salvare i file con la codifica *utf8*, per cui se si usano i caratteri nazionali della propria tastiera e l'editor è correttamente impostato, non si verificano più i fatti strani che si manifestavano nel passato. Questo non vuol dire che il pacchetto `inputenc` sia diventato inutile; serve ancora in molte circostanze e quando si devono ricomporre file sorgente datati, cioè creati con codifiche diverse prima del 2018. In sostanza dal 2018 è del tutto inutile ricorrere a codifiche come *latin1*, *latin9*, *ansinew*, *cp1252*, *applemac* e simili, come si faceva prima di quell'anno; le specifiche per quelle codifiche obsolete compaiono però ancora in documenti datati.

In generale io suggerirei di specificare la codifica di ogni file sorgente mediante le righe magiche: sempre e comunque, sia nei file datati, sia nei nuovi file; per approfondire l'argomento, si consulti la guida tematica (Beccari e Gordini, 2018).

1.9.2 Gli ambienti *array* e *tabular*

Le sintassi sono:

```
\begin{array}[⟨posizione⟩]{⟨colonne⟩} ⟨righe⟩ \end{array}
\begin{tabular}[⟨posizione⟩]{⟨colonne⟩} ⟨righe⟩ \end{tabular}
\begin{tabular*}{⟨larghezza⟩}[⟨posizione⟩]{⟨colonne⟩}
    ⟨righe⟩
\end{tabular*}
```

La ⟨*posizione*⟩ serve per stabilire l'allineamento della tabella o della matrice con il 'testo' circostante; la ⟨*larghezza*⟩ serve per imporre una larghezza specificata alla tabella. I descrittori delle ⟨*colonne*⟩ sono certi codici che ora si esporranno; le ⟨*righe*⟩ sono i contenuti delle celle orizzontali contenenti il testo o la matematica da comporre. *array* si usa solo in ambiente matematico;

gli ambienti *tabular* e *tabular** solo in modo testo; sia gli uni sia gli altri ambienti possono contenere singole celle contenenti 'testo' dell'altra specie, ma bisogna specificarlo per ciascuna di queste celle; in ambiente *tabular* basta usare gli appositi comandi per passare alla matematica; in ambiente *array* si possono usare i segni di dollaro che agiscono da interruttori/deviatori; se si è già in modo matematico (come avviene per ogni cella di un *array*) il primo segno di dollaro passa in modo testo e il secondo riporta al modo matematico.

I codici di posizione sono

`t` serve per allineare la riga di testa con il testo circostante.
`b` serve per allineare la riga di base con quella del testo circostante.
`c` non necessita di essere espresso, perché l'allineamento centrato è quello di default.

I descrittori delle ⟨*colonne*⟩ e i loro separatori sono i seguenti.

`l` colonna con i contenuti allineati a sinistra.
`c` colonna con i contenuti centrati.
`r` colonna con i contenuti allineati a destra.
`|` filetto verticale di separazione fra colonne adiacenti.
`\vline` lo stesso filetto quando deve essere inserito in una *@-espressione*.
`@{`⟨*testo*⟩`}` si tratta di un particolare costrutto, detto *@-espressione*, con il quale si specifica che ⟨*testo*⟩ sostituisce completamente il separatore ordinario fra due colonne adiacenti. Questo significa che se i contenuti delle due celle adiacenti devono essere in qualche modo distanziati, allora gli spaziatori devono essere inseriti dentro la *@-espressione*. Un elemento importante delle *@-espressioni* sono le dichiarazioni di spaziatura con elasticità 'infinita' da usare con l'ambiente *tabular** per consentire che queste particolari tabelle si possano estendere fino a raggiungere la larghezza specificata.
`\extracolsep{`⟨*larghezza*⟩`}` specificato dentro una *@-espressione* serve a dichiarare che lo spaziatore con la ⟨*larghezza*⟩ esplicitata venga inserito a sinistra del contenuto di tutte le celle *che seguiranno* sulla stessa riga; quindi *non* di questa e di tutte le celle che seguiranno, ma solo di quelle che seguiranno. Tipicamente, questo comando viene inserito nella prima *@-espressione* che compare a sinistra della prima

cella di una riga; quindi, questa cella non verrà mai allargata per raggiungere la giustezza desiderata.

`\fill` è una larghezza naturalmente nulla ma dotata di allungamento infinito, tipicamente usata come argomento di `\extracolsep`.

`p{`⟨*larghezza*⟩`}` serve per descrivere una colonna composta come un (breve) capoverso la cui prima riga è allineata con quella delle celle adiacenti e avente una giustezza pari a ⟨*larghezza*⟩. Il testo delle colonne composte con questo descrittore non può contenere direttamente il comando `\\`, perché LaTeX non potrebbe sapere se esso si riferisce alla cella o all'intera riga di celle. Perciò è necessario racchiudere questo comando dentro un ambiente, come *minipage* o un altro *tabular*, oppure dentro una scatola `\parbox` esplicita; oppure dentro il raggio di azione di comandi come `\centering`, `\raggedright` oppure `\raggedleft`, purché a loro volta siano contenuti dentro un gruppo delimitato da parentesi graffe oppure siano contenuti all'interno di altri ambienti. Se come terminatore della riga di celle si usa il comando `\tabularnewline` non ci sono più ambiguità.

`*{`⟨*numero*⟩`}{`⟨*descrittori*⟩`}` è una forma abbreviata per ripetere ⟨*numero*⟩ volte le stessa sequenza di ⟨*descrittori*⟩ delle colonne. Una **-espressione* ne può contenere un'altra; si consiglia di non eccedere con queste forme stenografiche innestate l'una nell'altra, altrimenti dopo un po' non si capisce più che cosa si sia specificato. Piuttosto è meglio rifarsi alle possibilità offerte dal pacchetto `array` e dal suo comando `\newcolumntype`.

I comandi che si possono usare dentro gli ambienti di incolonnamento sono i seguenti.

`\multicolumn` con la sintassi

`\multicolumn{`⟨*numero*⟩`}{`⟨*descrittore*⟩`}{`⟨*cella*⟩`}`

serve per comporre un'unica ⟨*cella*⟩ con il ⟨*descrittore*⟩ specificato che occupi ⟨*numero*⟩ celle adiacenti. Il ⟨*descrittore*⟩, oltre alla collocazione con una delle varie lettere chiave (compresa la `p`), può contenere filetti verticali e persino *@-espressioni*. Normalmente il contenuto di `\multicolumn` rimpiazza completamente tutte le ⟨*numero*⟩ celle adiacenti, compresi i loro separatori destri; quindi, il separatore sinistro

non dovrebbe mai essere indicato come contenuto del ⟨*descrittore*⟩, a meno che il gruppo di celle non comprenda anche la prima cella di una riga.

`\hline` serve per tirare un filetto orizzontale attraverso tutta la tabella o tutta la matrice. Esso può essere eseguito solo come primo elemento di una tabella, oppure dopo il comando di fine riga `\\`, oppure dopo un altro comando `\hline`; in quest'ultimo caso vengono tracciati due filetti ravvicinati attraverso tutta la tabella.

`\cline{`⟨*col₁-col₂*⟩`}` serve per tracciare un filetto orizzontale solo sotto le colonne ⟨*col₁-col₂*⟩; per esempio in una tabella a 7 colonne, il comando `\cline{2-6}` traccia un filetto orizzontale sotto le 5 colonne centrali; non si possono inserire due identici comandi `\cline` uno di seguito all'altro per raddoppiare il filetto, ma si possono usare per 'sottolineare' colonne distinte: per esempio, nella tabella a 7 colonne dell'esempio precedente `\cline{2-3}\cline{5-6}` 'sottolinea' solo le colonne 2 e 3 con un filetto e, allineato con il primo, un secondo filetto 'sottolinea' le colonne 5 e 6.

È opportuno sottolineare che la buona tipografia non usa affatto i filetti verticali e usa molto poco i filetti orizzontali; per questo scopo è conveniente il pacchetto `booktabs`, che definisce alcuni pochi filetti orizzontali di spessori diversi e dotati delle loro spaziature verticali, in modo da essere collocati convenientemente in testa, in calce e in mezzo alla tabella; tipicamente se ne usa uno spesso in testa, uno sottile dopo la riga delle intestazioni delle colonne ed eventualmente uno spesso al piede.

Si raccomanda l'uso del pacchetto `array` già citato; esso arricchisce le lettere descrittrici dei tipi di colonne con `m{`⟨*larghezza*⟩`}` e `b{`⟨*larghezza*⟩`}`, che assomigliano a `p{`⟨*larghezza*⟩`}` ma risultano allineati con il contenuto delle celle adiacenti sulla loro mezzeria o sulla riga di base dell'ultima riga. Quello che è maggiormente importante è che quel pacchetto permette di definire nuove lettere descrittive di colonne, in particolare permette di aggiungere all'inizio di ogni cella e, rispettivamente, alla fine di ogni colonna delle dichiarazioni che permettono di alterare il contenuto delle celle di una data colonna, senza bisogno di ripeterle ad ogni cella; per esempio in una tabella testuale una intera colonna può contenere dati matematici; allora, per esempio, una colonna di celle dal contenuto matematico centrato verrebbe dichiarata con questa descrizione:

```
\begin{tabular}{... >{$}c<{$}...}
```

e gli switch matematici costituiti dai segni di dollaro è come se fossero ripetuti all'inizio e alla fine di ogni cella di quella colonna. Se capita spesso di usare questo tipo di colonna, si può definire:

```
\newcolumntype{C}{>{$}c<{$}}
```

e poi dichiarare i descrittori di colonna con

```
\begin{tabular}{...C...}
```

Come si vede è molto comodo e con una sola definizione che vale per tutto il documento, cosicché, se fosse necessario, basterebbe modificare solo la definizione del "nuovo tipo di colonna" senza bisogno di andare a cercare tutte le tabelle dove si è specificato quel tipo di composizione.

I parametri compositivi di tabelle e matrici sono i seguenti.

`\arraycolsep` è metà dello spazio di separazione fra due celle consecutive di una matrice.

`\tabcolsep` e metà dello spazio di separazione fra due celle consecutive di una tabella.

`\arrayrulewidth` è lo spessore dei filetti orizzontali o verticali che si possono inserire in tabelle e matrici.

`\doublerulesep` è la spaziatura verticale o orizzontale fra due filetti consecutivi orizzontali o verticali.

`\arraystretch` è una macro, che può venire ridefinita con `\renewcommand`, e che rappresenta il fattore di scala con cui viene ingrandito o rimpicciolito lo strut (pilastrino) che si trova sempre nella prima cella di ogni riga delle tabelle o delle matrici. Il valore di default di questo fattore di scala è l'unità.

1.10 I file ausiliari e i loro comandi

1.10.1 I file del sistema TEX

LATEX fa riferimento a un gran numero di file, dei quali spesso il compositore non si accorge nemmeno. A parte i file sorgente con estensione `.tex` o `.ltx`, ci sono anche i file con le estensioni seguenti; il nome proprio dei file corrispondenti a un medesimo documento è costantemente il nome proprio

del file principale e questo nome è conservato dentro la variabile `\jobname` che può essere usata anche dal compositore o dal programmatore che scrive le macro.

`.aux` serve per conservare tutte le informazioni che riguardano i riferimenti incrociati; quando si usano i comandi `\include`, questi file conservano anche tutte le informazioni relative al file incluso in modo da poterne simulare la compilazione quando questo file non sia incluso nella lista di file da compilare contenuta nell'argomento di `\includeonly`. Il comando `\nofiles` sopprime la scrittura di tutti i file di questo tipo.

`.bbl` questo file viene scritto da BibTeX e da biber, ma viene poi letto da LaTeX per comporre la bibliografia quando si usi il comando `\bibliography`.

`.dvi` è il formato di default dell'uscita compilata di tutti i documenti elaborati con il programma tex; era e resta il formato di default del programam latex; esso fa riferimento ai font a matrici di pixel e generalmente il programma per rendere leggibile agli umani questo file è incluso nella distribuzione del sistema TeX; pdflatex, che viene usato per produrre direttamente l'uscita in formato PDF, è in grado di produrre l'uscita in formato DVI se si pone all'inizio del file `.tex` la specificazione: `\pdfoutput=0`. Oggi questo formato può essere evitato nella quasi totalità dei casi, se non altro perché con pdflatex si possono evitare i font a matrici di punti.

`.glo` è il file prodotto da LaTeX quando il comando `\makeglossary` sia attivo; esso contiene tutte le voci `\glossaryentry` prodotte dai vari comandi `\glossary` inseriti nei file sorgente.

`.idx` è il file prodotto da LaTeX quando sia attivo il comando `\makeindex`; esso contiene tutte le voci `\indexentry` prodotte dai vari comandi `\index` inseriti nel file sorgente.

`.ind` questo file viene scritto dal programma makeindex, ma viene riletto da LaTeX quando deve comporre l'indice analitico.

`.lof` contiene le informazioni per comporre l'indice delle figure.

`.log` contiene tutta le registrazione di quanto è successo durante l'esecuzione di LaTeX, in particolare le informazioni relativamente dettagliate concernenti gli errori e gli avvertimenti, i font usati, quelli che sono stati creati al volo, quelli mancanti, quelli sostituiti, eccetera.

`.lot` contiene le informazioni per comporre la lista delle tabelle.

`.toc` contiene le informazioni per comporre l'indice generale.

Altri pacchetti possono generare altri file ausiliari; per esempio, se si usa il pacchetto `hyperref` per generare i collegamenti ipertestuali, allora viene generato un file ausiliario con l'estensione `.out` che contiene alcune informazioni relative all'ancora e al bersaglio di questi collegamenti ipertestuali.

1.10.2 I riferimenti incrociati

I comandi per assegnare una parola chiave agli oggetti da citare e/o per richiamarne il valore o la pagina sono:

`\label{`⟨*chiave*⟩`}`
`\ref{`⟨*chiave*⟩`}`
`\pageref{`⟨*chiave*⟩`}`

Il comando `\label` è fragile; sarebbe meglio, anche se è possibile, non inserirlo mai all'interno di altri comandi, nemmeno i comandi di sezionamento o i comandi per le didascalie. I contatori che possono essere citati con la chiave sono quelli che sono stati incrementati con `\refstepcounter`; in ogni caso, `\label` associa alla chiave il valore dell'ultimo contatore in ordine di tempo che è stato incrementato con quel comando.

1.10.3 Suddivisione del file sorgente

Non è conveniente scrivere un unico file sorgente, specialmente se il documento da comporre contiene molte pagine, molti capitoli, eccetera. Conviene predisporre un 'master file' che contenga solo il preambolo e la lista dei file componenti; si veda lo schema sintattico riportato qui di seguito.

```
\documentclass[⟨opzioni⟩]{⟨classe⟩}
\usepackage[⟨opzioni⟩]{⟨pacchetto⟩}[⟨data⟩]
...
\includeonly{⟨lista di file⟩}
\begin{document}
\include{⟨primo file⟩}
\include{⟨secondo file⟩}
...
\end{document}
```

La ⟨*lista di file*⟩ è un elenco di nomi di file separati da virgole, ma senza spazi spuri prima o dopo le virgole. La sequenza di comandi `\include`, ognuno con il suo nome di file, esegue la compilazione dei file, tranne di quelli che *non* sono elencati nella ⟨*lista di file*⟩. Al limite si può compilare un file alla volta; i file già compilati, anche se non vengono più elaborati, hanno lasciato le loro tracce nei rispettivi file `.aux`, quindi, a meno che non siano intervenute modifiche, LaTeX riesce a risolvere tutti i riferimenti incrociati e a mantenere la coerenza dei numeri delle pagine e di tutti i contatori specificati con i comandi specifici di LaTeX.

È chiaro che prima di terminare la lavorazione di un documento è importante eseguire la compilazione di tutto il documento, cosicché si è completamente sicuri che tutti i riferimenti e tutti i valori dei contatori e delle pagine siano corretti; questo lavoro va fatto comunque per predisporre alla fine della lavorazione la compilazione dell'indice analitico.

Ci si ricordi che ogni comando `\include` per prima cosa esegue un comando `\clearpage` che svuota le code di oggetti flottanti e comincia la compilazione del nuovo file da includere su una pagina nuova. Se *non* è questo quello che si vuole ottenere, si faccia uso del comando

```
\input{⟨file⟩}
```

che non consente la compilazione selettiva, ma consente di proseguire la compilazione del testo dal punto preciso nel quale si trovava LaTeX nel momento in cui ha cominciato a eseguire il comando `\input`.

Per risolvere le dipendenze di un dato documento da certi file di macro, è disponibile l'unico ambiente che può precedere il comando `\documentclass`

```
\begin{filecontents}{⟨nome file⟩}
⟨corpo del file⟩
\end{filecontents}
```

È disponibile anche l'ambiente asteriscato; la differenza consiste in questo: quando *filecontents* viene *eseguito* durante la composizione di un dato documento, controlla che un file con il ⟨*nome file*⟩ specificato esista; se esiste, l'ambiente non fa nulla salvo scrivere sullo schermo e nel file `.log` che il file esiste già; se invece il file non esiste, questo ambiente lo crea scrivendovi dentro il ⟨*corpo del file*⟩ e salvandolo con il nome ⟨*nome file*⟩. Nel fare questo, al fine di identificare il tipo di file e la sua origine, esso scrive in testa al file anche dei commenti in stile LaTeX, cioè con un segno `%` all'inizio della riga. Se non si vogliono questi commenti, si usi l'ambiente asteriscato.

Il comando `\listfiles` serve per elencare a schermo e nel file `.log` tutti i nomi dei file che vengono via via aperti (tranne quelli di servizio) in modo che si possa seguire e controllare che l'esecuzione della compilazione proceda regolarmente.

1.11 Bibliografia e citazioni

Salvo disporre di file bibliografici esterni e di fare uso dei programmi BibTeX o biber, LaTeX dispone dell'ambiente *thebibliography* per comporre gli elenchi bibliografici e assegnare loro una chiave di riferimento.

```
\begin{thebibliography}{⟨cifre⟩}
\bibitem[⟨etichetta⟩]{⟨chiave⟩} ⟨voce bibliografica⟩
...
\end{thebibliography}
```

Le voci bibliografiche vengono richiamate con il comando

```
\cite[⟨testo⟩]{⟨chiave⟩}
oppure
\cite{⟨lista di chiavi⟩}
```

La ⟨*chiave*⟩ è una informazione interna di LaTeX che serve a LaTeX stesso e al compositore per riferirsi a una particolare voce bibliografica. Quando

questa viene citata, viene citata o con l'⟨*etichetta*⟩, se è stata specificata, oppure con il numero progressivo dell'elenco bibliografico; in ogni caso, numero o etichetta vengono racchiusi entro parentesi quadre.

Con `\cite` si possono fare citazioni multiple, semplicemente specificando una ⟨*lista di chiavi*⟩ formata dalle chiavi specifiche dei riferimenti da citare separate da virgole; per questo motivo, le chiavi possono essere formate con qualunque carattere, esclusa la virgola. Se si cita un'opera sola, la citazione contiene l'⟨*etichetta*⟩, o il numero, seguiti da ⟨*testo*⟩; per esempio: avendo attribuito a un riferimento l'etichetta `TUG2007`, il comando di citazione `\cite[cap.~2]{TUG2007}` produrrà qualcosa come [12, cap. 2].

Per la bibliografia, specialmente se contiene numerose voci, è consigliabile servirsi dei programmi BibTeX o biber, che garantiscono la composizione uniforme delle varie citazioni e, tramite file di estensione, consentono anche di usare stili di citazione diversi, come per esempio lo stile 'autore-anno'. Oggi è disponibile il pacchetto `biblatex` che dispone di molti moduli di configurazione e permette di comporre bibliografie molto ben fatte, senza bisogno di cercare di scriversi un file di stile bibliografico da soli, cosa che è particolarmente difficile, visto che il linguaggio usato, diverso dal linguaggio TeX, è particolarmente ostico.

Usando questi metodi automatici che si affidano ai programmi esterni BibTeX o biber per estrarre dai database *solo* le opere citate, si può usare il comando `\nocite` per elencare le etichette delle opere non espressamente citate ma da inserire ugualmente nella bibliografia, o addirittura si può usare un asterisco invece delle etichette per citare tutte le opere contenute nei database bibliografici usati da quei programmi per estrarre le informazioni delle opere da indicare nella bibliografia. La sintassi è la seguente:

`\nocite{`⟨*lista di etichette*⟩`}`

oppure

`\nocite{*}`

1.12 Indice analitico e glossario

Per comporre uno o più indici analitici è opportuno servirsi del programma makeindex; quel programma serve anche per compilare un glossario. Qui si richiamano i comandi necessari.

1.12.1 Indice analitico

La composizione dell'indice analitico avviene mediante l'ambiente *theindex*

```
\begin{theindex}
⟨contenuto dell'indice analitico⟩
\end{theindex}
```

Più precisamente, visto che l'indice analitico non viene mai composto a mano, ma tramite il programma esterno makeindex invece di esplicitare il contenuto completo dell'ambiente *theindex*, basta semplicemente importare il file `.ind` generato da makeindex sulla base della raccolta dei dati da indicizzare che LaTeX ha scritto nel file `.idx`:

```
\input{\jobname.ind}
```

Più comodamente, si usa il pacchetto `makeidx` e si ordina la composizione dell'indice analitico mediante il comando `\printindex`.

Questo procedimento, però fa capire che per generare l'indice analitico bisogna eseguire i programmi in quest'ordine.

1. Si esegue pdflatex sul file principale del documento che contiene la dichiarazione `\makeindex` nel preambolo, mentre il file principale e, specialmente, i file inclusi contengono i vari comandi `\index` che raccolgono i dati da indicizzare; questa raccolta di informazioni viene scritta nel file `\jobname.idx`.
2. Si esegue il programma makeindex; questo legge il file `\jobname.idx` e un file di stile (che generalmente ha estensione `.ist`) con cui comporre l'indice analitico; mette in ordine alfabetico e gerarchico tutte le voci dell'indice eliminando i duplicati, ma raccogliendo i numeri della pagina in cui ogni voce compare, e scrive il risultato, compresa l'apertura e la chiusura dell'ambiente *theindex* nel suo file di uscita `\jobname.ind`.
3. Si lancia nuovamente pdflatex, che legge il file `\jobname.ind` e compone l'indice analitico nel documento in lavorazione.

Perciò è ancora più comodo se si usa o il pacchetto `imakeidx` o `indextools`, che consentono di sfruttare la proprietà dei sistemi TeX moderni di eseguire dei comandi di sistema così da poter generare uno o più indici analitici con

una sola esecuzione di pdflatex (la prima indicata nella procedura descritta sopra).

Però, che si usi `makeidx` o `imakeidx` o `indextools`, per raccogliere le voci da indicizzare bisogna usare i seguenti comandi:

`\makeindex` inserito nel preambolo ordina di eseguire effettivamente la raccolta delle voci attraverso il comando `\index`, che altrimenti sarebbe completamente inerte.

`\index` serve per raccogliere le voci da indicizzare con la sintassi

`\index{`⟨*voce*⟩`}`

Il comando `\index` deve essere scritto senza spazi interposti alla fine della parola di testo che si vuole indicizzare. La ⟨*voce*⟩ non contiene solo la parola da indicizzare, ma anche le ulteriori informazioni di cui necessita il programma makeindex per formattare convenientemente la voce come principale, oppure secondaria, oppure di terzo livello, per scegliere il font con cui scrivere la voce, per scegliere la chiave di indicizzazione, per segnalare, nel caso di voci secondarie o di terzo livello, sotto quale voci primarie esse debbano essere elencate, per dare uno stile al numero di pagina, eccetera; si veda a questo proposito la documentazione del programma makeindex.

Chiaramente, se si usa o il pacchetto `imakeidx` o `indextools` per produrre diversi indici analitici, bisogna specificare con un parametro facoltativo in quale indice si deve porre la voce particolare marcata con `\index`; per questo è meglio consultare la documentazione di `imakeidx` o di `indextools`.

1.12.2 Glossario

Per comporre un glossario bisogna procedere in modo simile a quello che si usa per l'indice analitico; i comandi che permettono di raccogliere le voci sono

`\makeglossary` inserito nel preambolo, consente l'effettiva raccolta delle voci mediante l'attivazione del comando `\glossary`, che altrimenti sarebbe totalmente inattivo.

`\glossary` serve per raccogliere le voci da inserire nel glossario, con la sintassi:

`\glossary{`⟨*voce*⟩`}`

Il pacchetto `glossaries` aiuta molto l'autore nel predisporre la composizione di un glossario; si rinvia il lettore alla documentazione di questo pacchetto. Il pacchetto `nomencl` svolge funzioni simili; sotto certi aspetti è più comodo da usare rispetto a `glossaries`, il cui inconveniente maggiore è proprio quello di voler essere capace di fare troppe cose. Anche `acronym` può essere preso in considerazione per questi scopi, anche se è esplicitamente dedicato alla spiegazione e all'uso degli acronimi.

1.13 Compilazione interattiva

LaTeX consente un certo livello di interattività durante la compilazione; a parte gli errori, che chiedono al compositore di intervenire, LaTeX consente anche di emettere dei messaggi sul terminale e di leggere le risposte del compositore.

`\typeout` con la sintassi

`\typeout{`⟨*messaggio*⟩`}`

consente di scrivere un messaggio sullo schermo del terminale; questo messaggio può avere gli scopi più diversi, ma non consente nessuna interattività; piuttosto può preparare un dialogo interattivo.

`\typein` con la sintassi

`\typein[`⟨*comando*⟩`]{`⟨*messaggio*⟩`}`

emette il ⟨*messaggio*⟩ sullo schermo; se è specificato anche il ⟨*comando*⟩, il programma si arresta e attende che l'operatore scriva qualche cosa sulla tastiera (presumibilmente quanto il ⟨*messaggio*⟩ suggerisce di scrivere fra una rosa di possibili scelte); non appena il compositore preme il tasto di "invio", il testo introdotto viene assegnato al ⟨*comando*⟩, esattamente come se il tutto eseguisse le seguenti operazioni

`\typeout{`⟨*messaggio*⟩`}`
`\newcommand{`⟨*comando*⟩`}{`⟨*testo introdotto*⟩`}`

Il ⟨*comando*⟩ deve essere una valida sequenza di controllo, cioè una stringa esclusivamente letterale preceduta dal backslash, oppure un solo segno non letterale preceduto dal backslash.
Sta poi alla costruzione del file esaminare il ⟨*testo introdotto*⟩ e, a seconda di quel che contiene, eseguire diverse operazioni; in questo senso è possibile un certo grado di interattività fra il programma e l'operatore.

1.14 Interruzione di riga e di pagina

LaTeX permette di indicare manualmente sia comandi per interrompere una riga o una pagina in un punto preciso, sia per vietarlo. Per fare questo, esso agisce in genere sulle *penalità*; questi sono valori che l'utente o la classe possono specificare per agire in determinate circostanze. I comandi che nei paragrafi seguenti incoraggiano una interruzione di riga o di pagina inseriscono delle penalità negative di valore assoluto più o meno grande; i comandi che le impediscono inseriscono delle penalità positive più o meno grandi. Le routine interne per dividere un capoverso in righe o per dividere una bozza non impaginata in pagine tengono conto di queste penalità per svolgere il loro compito. Si tenga presente che per il motore di composizione una penalità di valore assoluto 10 000 o superiore significa "infinito", perciò il comando specificato non è più un suggerimento più o meno forte di interrompere o di non interrompere una riga o una pagina, ma diventa un ordine incondizionato.

1.14.1 Interruzione di riga

I comandi

```
\linebreak[⟨numero⟩]
\nolinebreak[⟨numero⟩]
```

consentono o vietano, rispettivamente, di interrompere una riga. Se si specifica il ⟨*numero*⟩, la loro azione è rinforzata o indebolita a seconda del valore del numero stesso. Il numero 0 rappresenta un debole incoraggiamento a operare, mentre il numero 4 rappresenta l'indicazione di eseguire incondizionatamente l'azione specificata; l'assenza del numero equivale a specificare il numero 4. Attenzione: `\linebreak` interrompe la linea ma

lascia che LaTeX cerchi di giustificarla, per cui se la gomma interparola deve essere allargata troppo, viene emesso un messaggio di `Underfull hbox`.

Una riga può venire interrotta anche con i comandi

```
\newline
\\[⟨spazio⟩]
\\*[⟨spazio⟩]
```

Essi permettono di interrompere una riga per andare decisamente a capo. Non viene eseguito nessun tentativo di giustificare la riga, nemmeno se manca poco al margine destro. La forma asteriscata impedisce un fine pagina subito dopo.

Per tutti questi comandi la possibilità di eseguire la cesura delle parole in fin di riga è essenziale. Per altro, in certe circostanze non basta consentire la cesura, ma bisogna variare le tolleranze di composizione, cioè bisogna accontentarsi di una maggiore bruttezza di una o più righe. Ciò si ottiene con i comandi

```
\sloppy
\fussy
```

Il primo rende più tollerante il programma di composizione accettando bruttezze molto alte, al limite "infinite". Il secondo ripristina i valori normali.

Questi due comandi non impostano solo il valore di `\tolerance` ma impostano anche le lunghezze `\hfuzz` e `\vfuzz`; questi controllano se il programma di composizione emette dei warning di righe (`\hbox`) che sporgono nel margine destro o di pagine (`\vbox`) che sforano nel margine inferiore. Se queste scatole non sforano più del valore impostato con queste due lunghezze, non viene emesso nessun messaggio; essi sono preimpostati entrambi al valore di `0.5pt`, valore piuttosto piccolo a causa del quale i messaggi possono essere numerosi. È vero che in un testo composto bene non dovrebbero esserci sforamenti orizzontali o verticali di nessun genere; tuttavia talvolta succede e, ovviamente, dipende dal contenuto del testo che si sta componendo. In questa guida, che contiene molte parti composte col carattere teletype, per il quale la cesura in fin di riga è interdetta, in prima battuta gli sforamenti erano numerosi; rimaneggiando le frasi che contenevano questi sforamenti li si è evitati quasi tutti; alcuni piccoli sforamenti sono tuttavia ancora presenti.

1.14.2 Interruzione di pagina

I comandi

```
\pagebreak[⟨numero⟩]
\nopagebreak[⟨numero⟩]
```

Agiscono come `\linebreak` ma si riferiscono alla pagina, non alla riga. Il ⟨*numero*⟩ ha lo stesso significato con 0 che indica in blando incoraggiamento ad eseguire, e 4 che ordina di eseguire incondizionatamente il comando stesso.

Per l'interruzione di pagina non si può agire sulla cesura, ma si può temporaneamente allungare la pagina rispetto al normale:

```
\enlargethispage{⟨lunghezza⟩}
\enlargethispage*{⟨lunghezza⟩}
```

Il comando senza asterisco allunga semplicemente la pagina; per ovvi motivi ⟨*lunghezza*⟩ dovrebbe essere un multiplo intero di `\baselineskip`, dell'avanzamento di riga normale con il font di default. Tuttavia, il comando asteriscato cerca di far entrare nella pagina quante righe ⟨*lunghezza*⟩ consente di contenere, ma allungando la pagina il meno possibile, cercando di stringere il più possibile gli spazi bianchi verticali. Specialmente componendo sul verso e sul recto, è importante che i blocchi di testo siano della stessa altezza; se i margini laterali sono consistenti, una riga in più in una delle due pagine si nota appena, tuttavia sarebbe meglio evitarlo. Il comando asteriscato potrebbe essere la soluzione se la pagina nella quale esso compare ha abbastanza spazi bianchi.

I comandi

```
\newpage
\clearpage
\cleardoublepage
```

chiudono la pagina corrente; gli ultimi due scaricano anche le eventuali code degli oggetti flottanti che potrebbero ancora contenere; il comando `\cleardoublepage` eventualmente emette anche una pagina bianca al fine di consentire di iniziare la pagina successiva sul recto. `\cleardoublepage` è tipicamente il primo comando che viene eseguito implicitamente quando si

esegue il comando di inizio di un nuovo capitolo e anche quando si immette un file con `\include`. Componendo a due colonne, `\newpage` chiude la colonna corrente e prosegue la composizione sulla colonna successiva.

Sono disponibili anche dei comandi presi a prestito da Plain TEX, ma ancora disponibili con LATEX:

`\smallbreak` esegue un salto pagina solo se prima era stato introdotto nel testo uno spazio verticale pari a `\smallskipamount`.

`\medbreak` esegue un salto pagina solo se prima era stato introdotto nel testo uno spazio verticale pari a `\medskipamount`.

`\bigbreak` esegue un salto pagina solo se prima era stato introdotto nel testo uno spazio verticale pari a `\bigskipamount`.

`\goodbreak` indica a LATEX che questo è un buon punto per spezzare la pagina; LATEX effettivamente esegue il salto se manca poco alla fine naturale della pagina, se, cioè, non resta una pagina pesantemente mozza, ma le mancano solo poche righe.

`\filbreak` sempre se manca poco alla fine della pagina, questo comando indica a LATEX che può eseguire un salto pagina inserendo anche un po' di gomma verticale in modo che non vengano emessi messaggi di `Underfull vbox`; nello stesso tempo, se il punto non è buono per interrompere la pagina `\filbreak`, non fa nulla.

Si consiglia di usare questi comandi solo alla fine della lavorazione di un documento per correggere a mano alcune piccole cose che LATEX da solo non è capace di gestire. I casi in cui il compositore è costretto a ricorrere a questi comandi e a quelli di LATEX sono rarissimi in testi 'normali'; possono essere frequenti in testi che contengono molta matematica. Tuttavia `\enlargethispage`, con o senza asterisco, consente di rimediare a un capitolo la cui ultima riga o ultime due righe cadono nell'ultima pagina, peggio ancora se questa pagina è dispari. Se la pagina è pari, questa eventualità non è tanto bella ma può essere accettata; se esse cadono su una pagina dispari, questa resta quasi bianca e il capitolo successivo inizia in generale di nuovo su una pagina dispari, con il risultato che appaiono due pagine bianche o quasi bianche di seguito; decisamente da evitare.

Inoltre ,secondo me, i comandi `\goodbreak` e `\filbreak` non sono sufficientemente affidabili; li si può usare a proprio rischio o pericolo, ma è inutile chiedersi perché talvolta LATEX consideri la seconda riga di una

pagina un punto quasi alla fine della pagina. È raro, ma con quei comandi succede. In generale, anche se sono disponibili, è meglio evitare le macro di Plain TeX.

Piuttosto, se i capoversi finali del capitolo problematico sono sufficientemente lunghi, si può convincere pdftex a comporli su una o due righe in più o in meno, senza forzare allargamenti o accorciamenti improponibili degli spazi interparola, specificando il comando nativo `\looseness` seguito da un (piccolo) numero positivo o negativo; per esempio scrivendo `\looseness=1` dentro o in assoluta adiacenza a un capoverso, pdftex cercherà di allungare quel solo capoverso di una riga purché il farlo non aumenti la 'bruttezza' del capoverso oltre alla tolleranza valida in quel momento. Il valore di `\looseness` viene rimesso a zero automaticamente alla fine del capoverso, quindi vale solo per quel capoverso per il quale è stato specificato. Perché `\looseness` abbia effetto è necessario che il capoverso sia abbastanza lungo e che l'ultima riga, senza l'uso di `\looseness` e un parametro positivo, sia abbastanza lunga al fine di avere un capoverso più lungo di una riga; al contrario, per accorciare il capoverso di una riga è necessario che il capoverso sia sufficientemente lungo, ma che l'ultima riga sia piuttosto corta. Questo capoverso è stato composto con il parametro pari a 1.

1.14.3 Righe orfane e vedove

Una riga vedova è l'ultima riga di un capoverso dopo un fine pagina, che la lascia da sola all'inizio della pagina successiva; anche le interruzioni della composizione di un capoverso per consentire l'inserimento di materiale fuori testo (un'equazione, un testo in display, un titolo di paragrafo e simili) possono dar luogo a righe vedove. Una riga orfana è la prima riga di un capoverso o della parte di un capoverso che appare dopo del materiale in display, che risulta essere l'ultima riga di una pagina. Per ricordare queste due definizioni valgono le frasi seguenti: "una vedova ha un passato, ma non un futuro" e "un'orfana ha un futuro ma non un passato".

L'algoritmo di divisione di una bozza non impaginata in pagine tiene conto di queste righe facendo riferimento a certe penalità imposte dalla classe del documento o dall'utente che si chiamano `\clubpenalty` e `\widowpenalty`. È evidente che la seconda penalità si riferisce alle righe 'widow' (vedova), ma, mentre è meno ovvio che la prima si riferisca alle

righe ‘club’, non è generalmente noto che in tipografia la parola inglese *club* indichi le righe orfane.

In questo testo la classe impone la penalità per le righe orfane pari a 3000, e quella per le righe vedove pari a 3000. Come si vede, si tratta di valori positivi inferiori a 10 000 che scoraggiano l’interruzione di pagina dopo una vedova o prima di una orfana, ma non sono ordini assoluti.

1.15 Scatole

Una *scatola* è un oggetto che può contenere diversi segni, anche un testo formato da diverse righe, ma questo testo, una volta inscatolato, viene trattato da LaTeX come se fosse un oggetto unico e non lo spezza mai alla fine di una riga o di una pagina. I comandi per mettere del ⟨*testo*⟩ dentro una scatola sono

```
\mbox{⟨testo⟩}
\makebox[⟨larghezza⟩][⟨posizione⟩]{⟨testo⟩}
\fbox{⟨testo⟩}
\framebox[⟨larghezza⟩][⟨posizione⟩]{⟨testo⟩}
```

I comandi che cominciano con la lettera ‘f’ inseriscono una cornice attorno al testo; quelli che cominciano con la lettera ‘m’ non inseriscono nessuna cornice. Se la ⟨*larghezza*⟩, che è facoltativa, viene specificata, il testo viene inserito dentro una scatola di quella ⟨*larghezza*⟩, altrimenti i comandi che accettano questo argomento facoltativo si comportano come quelli che non lo richiedono. Se viene specificata la ⟨*larghezza*⟩, allora si può specificare anche la posizione mediante una sola delle lettere seguenti: `l` per collocare il testo a sinistra, `r` per collocare il testo a destra, `s` per distribuire il testo, allargando o restringendo lo spazio interparola, lungo tutta la ⟨*larghezza*⟩; se non viene specificata nessuna posizione, il testo risulta centrato dentro la ⟨*larghezza*⟩. Si confrontino i casi seguenti.

```
\framebox[50mm][l]{testo a sinistra}
\framebox[50mm][r]{testo a destra}
\framebox[50mm]{testo centrato}
\framebox[50mm][s]{testo spaziato}
```

testo a sinistra

testo a destra

testo centrato

testo spaziato

Lo spessore della linea che forma la cornice delle scatole è conservata nel parametro dimensionale `\fboxrule` e la distanza della cornice dal testo è conservata dentro il parametro dimensionale `\fboxsep`. Attenzione: lo spessore di `1` pt per la cornice dà luogo a una cornice molto scura: testo. Quindi non è il caso di specificare spessori maggiori se non a ragion veduta.

Si può sostituire l'indicazione esplicita della ⟨*larghezza*⟩ nelle scatole che l'accettano mediante quattro nuovi comandi che si riferiscono tutti alle dimensioni della scatola composta con `\mbox` che contiene lo stesso ⟨*testo*⟩; essi sono:

`\height` ovvero l'altezza della scatola composta con `\mbox{`⟨*testo*⟩`}`
`\depth` ovvero la profondità di quella scatola
`\width` ovvero la larghezza di quella scatola
`\totalheight` ovvero la somma di `\height` e di `\depth`.

Per inserire del testo dentro scatole che contengono materiale 'verticale' (in pratica una successione verticale di righe, uno o più capoversi), si possono usare il comando `\parbox` o l'ambiente *minipage*:

`\parbox[`⟨*pos*⟩`][`⟨*altezza*⟩`][`⟨*pos.interna*⟩`]{`⟨*giustezza*⟩`}{`⟨*testo*⟩`}`

oppure

`\begin{minipage}[`⟨*pos*⟩`][`⟨*altezza*⟩`][`⟨*pos.interna*⟩`]{`⟨*giustezza*⟩`}`
⟨*testo*⟩
`\end{minipage}`

dove il parametro ⟨*pos*⟩ indica l'allineamento della scatola con il testo circostante; `t` indica che la riga di base della prima riga della scatola è allineata con la riga di base del testo circostante; `b` indica che la linea di base dell'ultima riga della scatola è allineata con la riga di base del testo circostante; per default, cioè senza specificare nulla, l'asse matematico della scatola è allineato con l'asse matematico del testo circostante. Facoltativamente si può specificare l'⟨*altezza*⟩ della scatola; se questa altezza viene specificata, allora si può usare il terzo parametro facoltativo ⟨*pos.interna*⟩; le lettere di posizionamento sono le stesse, ma si riferiscono a dove il testo contenuto nella scatola viene collocato rispetto ai bordi ideali superiore e inferiore della scatola stessa. Va infine specificata la ⟨*giustezza*⟩ della scatola, in pratica la sua larghezza e il ⟨*testo*⟩ che essa deve contenere.

La differenza sostanziale fra il comando e l'ambiente è la seguente: il comando `\parbox` non comincia a comporre il ⟨*testo*⟩ al suo interno finché non l'ha letto tutto. Tutto ciò non avviene con l'ambiente *minipage* il quale gestisce il ⟨*testo*⟩ in modo differente. In compenso, *minipage*, in quanto ambiente, richiede una 'amministrazione' un poco più complessa che richiede un maggior tempo di elaborazione. Parlando di microsecondi, non è il caso di formalizzarsi troppo, visti i numerosissimi vantaggi che offre l'ambiente rispetto al comando. L'ambiente *minipage* può anche contenere delle note interne alla 'paginetta' che esso compone; queste note vengono composte con i soliti comandi; solo che la numerazione delle note avviene con lettere corsive in posizione di apice, in modo da distinguerle bene dalle note di piè di pagina, generalmente composte con esponenti numerici o con simboli non letterali.

Si noti che se il contenuto di una *minipage* comincia con una equazione in display, essa conserva il suo spazio che la precede; se non si vuole questo spazio si cominci la *minipage* con `\vspace{-\abovedisplayskip}`.

Il comando `\rule` compone una scatola particolare: essa, infatti, è completamente nera. La sintassi è:

`\rule[`⟨*rialzo*⟩`]{`⟨*base*⟩`}{`⟨*altezza*⟩`}`

La ⟨*base*⟩ e l'⟨*altezza*⟩ sono la base e l'altezza del rettangolo nero; se una delle due è nulla il rettangolo è invisibile, ma l'oggetto continua a mantenere l'altra dimensione; se la base è nulla si ottiene uno *strut* o, in italiano, un *pilastrino*. Il rettangolo nero (o invisibile) risulta rialzato di ⟨*rialzo*⟩ se questa lunghezza è positiva o ribassato se essa è negativa.

Infine, un comando serve per collocare del ⟨*testo*⟩ in una scatola che viene alzata o abbassata a piacere e le si possono assegnare dimensioni a piacere, indipendentemente dal testo che la scatola contiene.

`\raisebox{`⟨*rialzo*⟩`}[`⟨*altezza*⟩`][`⟨*profondità*⟩`]{`⟨*testo*⟩`}`

La ⟨*profondità*⟩ può venire specificata solo se si specifica anche l'⟨*altezza*⟩.

LaTeX consente di definire dei registri/scatola dove conservare del materiale inscatolato; i comandi da usare per impiegare questi registri sono i seguenti:

`\newsavebox` con la sintassi:

```
\newsavebox{⟨comando⟩}
```

alloca un registro/scatola e lo associa al ⟨*comando*⟩, cioè a un backslash seguito da una stringa puramente letterale, oppure a un backslash seguito da un solo carattere non letterale.

`\sbox` con la sintassi:

```
\sbox{⟨comando⟩}{⟨testo⟩}
```

carica con il ⟨*testo*⟩ il registro/scatola identificato da ⟨*comando*⟩ precedentemente definito; qui e nel seguito di questo paragrafo con ⟨*testo*⟩ indicherò qualunque cosa: del testo in linea, un testo composto con un comando `\parbox` o con un ambiente *minipage*; un disegno; una formula; insomma qualunque cosa che produca dei segni visibili nel file di uscita.

`\savebox` con la sintassi:

```
\savebox{⟨comando⟩}[⟨larghezza⟩][⟨posizione⟩]{⟨testo⟩}
```

carica con il ⟨*testo*⟩ il registro/scatola identificato da ⟨*comando*⟩; la ⟨*larghezza*⟩ è facoltativa come per `\makebox` e `\framebox`; se non la si specifica, il comando si comporta come `\sbox`; se la si specifica ,il testo viene collocato dentro la scatola nella ⟨*posizione*⟩ specificata allo stesso modo di come succede con `\makebox` e `\framebox`. Nel manuale originale non è documentato, ma questo comando si può usare con una sintassi diversa anche dentro l'ambiente *picture* a cui si rimanda.

lrbox è un ambiente che si comporta quasi come `\sbox`; la sintassi è

```
\begin{lrbox}{⟨comando⟩}
⟨testo da inscatolare⟩
\end{lrbox}
```

L'uso dell'ambiente consente di dare il comando di apertura all'interno dei comandi di apertura di un altro ambiente, e il comando di chiusura all'interno dei comandi di chiusura dell'altro ambiente; certamente bisogna rispettare l'ordine di apertura e di chiusura dei vari ambienti ma, nonostante queste piccole attenzioni in più, questo ambiente risulta molto utile nella definizione o nell'uso di altri ambienti.

`\usebox` con la sintassi

`\usebox{`⟨*comando*⟩`}`

permette di copiare in uscita il contenuto del registro/scatola ⟨*comando*⟩; serve quindi per usare il suo contenuto per scrivere o disegnare qualcosa nel file di uscita; il registro/scatola non viene svuotato, ma continua a contenere ciò che conteneva prima dell'uso del comando `\usebox`.

Sapendo quello che si sta facendo, i registri/scatola possono venire anche chiamati con un indirizzo assoluto, ma bisogna essere sicuri di essere all'interno di un gruppo o di un ambiente che non faccia uso di quel registro ma, in ogni caso, che all'uscita dal gruppo o dall'ambiente il registro venga ripristinato al suo contenuto originale.

Presento qui un esempio di uso di queste scatole. Voglio produrre un ambiente in cui posso mettere due (piccole) figure affiancate, ognuna con la sua didascalia; prevedo quindi, all'interno dell'ambiente *figure*, di avere una sola scatola (in realtà non necessaria, ma utile concettualmente) che contenga altre quattro scatole sistemate in due file e due colonne; le scatole della stessa colonna hanno la stessa larghezza e le scatole della stessa riga hanno lo stesso allineamento verticale; si faccia riferimento alla figura 1.4.

È facilissimo predisporre le quattro *minipage*, o i quattro comandi `\parbox`, in modo da riempire i quattro spazi in cui è divisa la figura; l'unica "difficoltà" consiste nel fissare le larghezze delle scatole di ciascuna colonna allo stesso valore e l'allineamento delle scatole della prima riga alla posizione `b` (*bottom*)mentre l'allineamento delle scatole della seconda riga deve essere posto alla posizione `t` (*top*). È semplicissimo costruire un ambiente in modo tale che vengano rispettati i vincoli suddetti. Si noti che, a mia conoscenza, non esiste nessun pacchetto che definisca un ambiente del genere; quindi essere capaci di usare le scatole per definire quello che serve è una cosa alla portata di tutti e facile da realizzare. Poi si può ricamare sulla prima soluzione che viene in mente e, usando i vari comandi della sintassi di LaTeX, diventa relativamente facile specificare un ambiente che operi in modo del tutto automatico. Il risultato di questa operazione può essere verificato nelle figure affiancate 1.6 e 1.7 nella pagina 137.

Per esempio, si potrebbe usare un codice simile a quello riportato nella pagina 123.

Figura 1.4 Scatole per mettere due figure affiancate con le rispettive didascalie. Il bordo rosso indica la scatola "virtuale" che contiene le due figure affiancate con le loro didascalie.

Ovviamente è sempre possibile usare `\newenvironment` per definire un nuovo ambiente che faccia buona parte del lavoro, in particolare che accetti due parametri che contengano le giustezze sinistra e destra, e `\newcommand` per definire alcuni comandi utili per caricare le scatole col materiale che devono contenere.

Personalmente preferisco usare questo metodo per affiancare due figure scorrelate l'una con l'altra; tanto che hanno due didascalie distinte numerate indipendentemente; si tenga presente che le giustezze della figura e della didascalia di sinistra sono specificabili indipendentemente dalle grandezze omologhe di destra. Mi sono definito un nuovo ambiente *duefigure* che contiene le definizioni che consentono di riempire le quattro scatole di cui si è fatto uso anche nella sintassi della pagina 123; il lettore non avrà difficoltà a ricreare il mio ambiente, che si usa semplicemente così:

Nel preambolo si definiscono quattro scatole

```
\newbox{\figuraS} \newbox{\figuraD}
\newbox{\didascaliaS} \newbox{\didascaliaD}
```

Nel testo si usa il codice seguente.

```
\begin{figure}
\begin{lrbox}{\figuraS}
\begin{minipage}[b]{⟨giustezza sinistra⟩}
⟨figura di sinistra⟩
\end{minipage}
\end{lrbox}
%
\begin{lrbox}{\didascaliaS}
\begin{minipage}[t]{⟨giustezza sinistra⟩}
⟨didascalia di sinistra⟩
\end{minipage}
\end{lrbox}
%
\begin{lrbox}{\figuraD}
\begin{minipage}[b]{⟨giustezza destra⟩}
⟨figura di destra⟩
\end{minipage}
\end{lrbox}
%
\begin{lrbox}{\didascaliaD}
\begin{minipage}[t]{⟨giustezza destra⟩}
⟨didascalia di destra⟩
\end{minipage}
\end{lrbox}
%
\usebox{\figuraS} \hfill\usebox{\figuraD}
\usebox{\didascaliaS}\hfill\usebox{\didascaliaD}
\end{figure}
```

```
\begin{duefigure}[⟨posizione⟩]{⟨giustezza sinistra⟩}{⟨giustezza destra⟩}
\firstfigurebox{⟨materiale per la figura di sinistra⟩}
\firstcaptionbox{⟨didascalia di sinistra⟩}
\secondfigurebox{⟨materiale per la figura di destra⟩}
\secondcaptionbox{⟨didascalia di destra⟩}
\end{duefigure}
```

I quattro comandi indicati nello schema sintattico precedente possono venire scritti in un ordine qualsiasi, cioè non è obbligatorio seguire lo stesso ordine indicato in quello schema sintattico. L'argomento facoltativo ⟨*posizione*⟩ contiene i soliti codici di posizione per le figure o le tabelle.

Esiste il pacchetto `subfig` per mettere in un unico ambiente *figure* diverse figure, ognuna con la sua breve didascalia numerata con una lettera, e l'intero gruppo di piccole immagini ha una didascalia generale che ne spiega la presenza assieme. Il pacchetto dimensiona le didascalie parziali alle dimensioni orizzontali di ogni sotto figura; questo richiede un poco di attenzione, perché è difficile comporre bene con giustezze piccole, di due o tre centimetri.

1.16 Disegni e colori

L'ambiente *picture* nativo di LaTeX non è in grado di fare altro che semplici disegni, con l'inclinazione dei segmenti e dei vettori molto limitata dagli speciali font che servono per disegnarli; anche i cerchi sia pieni sia vuoti sono un problema, se non nelle situazioni più semplici; i rettangoli ad angoli arrotondati soffrono delle stesse limitazioni dei cerchi; le linee arbitrarie possono essere eseguite solo con le curve di Bézier quadratiche, quindi con certe limitazioni sulle quali qui non è il caso di insistere.

Dal 2003 è disponibile il pacchetto `pict2e`, che toglie tutte queste limitazioni; esso ha assunto una conformazione stabile dal 2004; quindi si raccomanda caldamente di usare quel pacchetto se si desiderano eseguire semplici disegni. Nel 2009, il pacchetto `pict2e` è stato ulteriormente arricchito di altri comandi che permettono ora di comporre disegni piuttosto elaborati. Si noti che il pacchetto `pict2e` è già stato descritto da Leslie Lamport nella sua guida del 1994 che descrive LaTeX 2_ε, quindi, benché sia necessario ricorrere a un pacchetto di estensione, *picture* arricchito con `pict2e` deve essere considerato come parte integrante di LaTeX 2_ε.

Se si vuole disegnare qualcosa di più elaborato ancora, si ricorra al pacchetto `pgf` e al suo ambiente *tikzpicture*; questo pacchetto consente di disegnare praticamente qualsiasi cosa. Sempre basato sul pacchetto `pgf` esiste il pacchetto `pgfplots` che consente di disegnare dei bellissimi diagrammi bidimensionali e tridimensionali; bisogna prenderci un po' la mano, ma vale la pena.

Naturalmente, i disegni necessitano dell'uso dei colori, così come, una volta creati, o importati dall'esterno, possono avere bisogno di altre manipolazioni, come il cambiamento di scala, la rotazione e simili.

1.16.1 Colori e gestione delle immagini

LaTeX è dotato di una serie di pacchetti che consentono di eseguire diverse cose: oltre all'importazione di file grafici esterni, essi mettono a disposizione la gestione del colore e la manipolazione di scatole varie, che normalmente non possono essere eseguite da LaTeX (se non passando attraverso il formato PostScript) ma possono invece venire eseguite direttamente da pdfLaTeX. L'intera serie di pacchetti si trova nella cartella `.../tex/latex/graphics/`, mentre la documentazione si trova in `.../doc/latex/graphics/`. Il lettore è invitato a documentarsi sul file `grfguide.pdf`; qui si riportano solo i comandi più importanti.

Si noti che fra questi pacchetti c'è `color` per la gestione del colore; i colori possono essere chiamati per nome o possono essere descritti a livello più basso mediante la specificazione del modello del colore e le componenti dalle quali in questo modello un dato colore è caratterizzato. Il pacchetto `xcolor`, da usare in sostituzione di `color`, estende moltissimo queste funzionalità; si rimanda alla documentazione di `xcolor` per avere quanti più modelli di colore si possono usare con questo pacchetto e come si possano definire nuovi colori semplicemente mescolando in proporzioni diverse alcuni colori già definiti; vale la pena di leggere la documentazione di `xcolor` anche per imparare molte cose sulla colorimetria, argomento di solito riservato agli specialisti.

La gestione delle scatole e l'importazione di figure esterne è gestita da altri pacchetti: di questi si è citato qui sempre e soltanto `graphicx`, che è il più comodo da usare, in quanto esso dispone di una interfaccia a parole chiave, rispetto al file che contiene le macro vere e proprie di gestione della grafica, `graphics`. Quest'ultimo può essere usato da solo, ma la sintassi per

fargli eseguire le cose che sono così semplici con `graphicx` diventa spesso abbastanza complessa.

`\scalebox{`⟨*scala-orizz.*⟩`}[`⟨*scala-vert.*⟩`]{`⟨*testo*⟩`}` serve per inserire ⟨*testo*⟩ dentro una scatola per poi scalarla del fattore di scala orizzontale; se il fattore di scala verticale non viene specificato, l'ingrandimento verticale avviene con lo stesso fattore di scala orizzontale, altrimenti la scatola viene deformata in maniera diversa così da ottenere effetti particolari.

`\resizebox{`⟨*larghezza*⟩`}{`⟨*altezza*⟩`}{`⟨*testo*⟩`}` esegue in pratica la stessa operazione di `\scalebox`, solo che invece di usare dei fattori di scala, usa le dimensioni finali effettive del ⟨*testo*⟩. Il comando asteriscato agisce in modo da scalare l'altezza totale (altezza più profondità) al valore specificato. Una delle due dimensioni può essere sostituita da un punto esclamativo; in questo caso, l'altra dimensione fissa il fattore di scala complessivo in modo che il rapporto di forma non sia modificato.

`\rotatebox[`⟨*lista di chiavi*⟩`]{`⟨*angolo*⟩`}{`⟨*testo*⟩`}` serve per ruotare la scatola che contiene il ⟨*testo*⟩ di un ⟨*angolo*⟩ specificato (in gradi) in senso antiorario; il ⟨*testo*⟩ può essere qualunque cosa: del testo vero e proprio, una figura, una tabella, ...

Nella ⟨*lista di chiavi*⟩ si può anche specificare mediante *origin* il punto attorno al quale eseguire la rotazione; si possono definire delle coordinate, ma ci sono alcuni punti speciali che possono essere rappresentati mediante codici letterali: `t`, `b`, `r`, `l` e `B` rappresentano le solite iniziali di 'top', 'bottom', 'right', 'left', ma `B` è un codice nuovo rispetto a quelli già incontrati altre volte: esso specifica la linea di 'Base', la 'Baseline', la linea, cioè, che attraversa orizzontalmente la scatola in corrispondenza del punto di riferimento; queste lettere sono da usare a due a due; per esempio: `tl` per indicare l'angolo in alto a sinistra, `Br` per indicare l'intersezione della linea di base con il lato destro della scatola, eccetera. Si osservino le tre diverse posizioni che assume la lettera 'Q' quando la si ruoti di 90° attorno ai tre possibili punti di rotazione di sinistra: '`bl`' Q, '`Bl`' Q e '`tl`' Q. Generalmente, per ruotare una scritta in verticale, per esempio per metterla a fianco di un asse verticale in un disegno, è conveniente eseguire la rotazione attorno al punto `Bl`.

`\reflectbox{`⟨*text*⟩`}` gira la scatola che contiene il ⟨*testo*⟩ attorno al suo asse verticale in modo che appaia come riflessa allo specchio.

`\includegraphics[`⟨*lista di chiavi*⟩`]{`⟨*file grafico*⟩`}` serve per inserire una scatola che contiene l'immagine descritta nel ⟨*file grafico*⟩ elaborata secondo le numerose chiavi che si possono specificare, ognuna con il suo valore; se si specifica il *bounding box*, il rettangolo 'circoscritto', la versione asteriscata di questo comando permette di tagliare tutto quanto sporge fuori da detto rettangolo. Per questo scopo talvolta è meglio specificare le coordinate del `viewport` o le dimensioni delle strisce da ritagliare con `trim` e poi specificare la chiave `clip`; in questo modo si può avere un controllo migliore di quanto si sta eseguendo.

`\definecolor{`⟨*nome*⟩`}{`⟨*modello*⟩`}{`⟨*valore*⟩`}` serve per dare un ⟨*nome*⟩ a un colore specificato mediante il suo ⟨*modello*⟩ e il ⟨*valore*⟩ dei parametri che in quel modello identificano il colore scelto; il pacchetto `color` considera come modelli di colore i seguenti:

`gray` è il modello di colore grigio identificato da un solo valore, un numero decimale nell'intervallo $0-1$; 0 vuol dire nero e 1 significa bianco; ogni valore intermedio specifica una certa gradazione di grigio.

`rgb` è il modello di colore degli schermi: rosso, verde, blu. I colori sono additivi e il valore che specifica un dato colore in questo modello è dato da tre numeri decimali separati da virgole, tutti e tre nell'intervallo $0-1$; 0 vuol dire 'assenza' di quella particolare componente di colore, 1 significa che quella componente di colore è totalmente satura. Per esempio $1, 0, 0$ indica il rosso saturo.

`cmyk` è il modello della quadricromia a stampa con i colori sottrattivi; ogni colore è identificato da quattro componenti, corrispondenti ciascuna ai pesi per ciascuno dei colori fondamentali ciano (celeste), magenta (lilla), giallo, e nero. Ogni componente è pesata con un numero decimale nell'intervallo $0-1$ e la quaterna di valori decimali costituisce una lista di valori separati da virgole.

Con ogni modello di colore, sono predefiniti i seguenti colori: `white`, `black`, `red`, `green`, `blue`, `yellow`, `cyan`, `magenta`. Il driver specifico, a seconda se si passi attraverso il programma dvips oppure se si proceda con pdfLaTeX, può definire altri colori. I nomi dei colori predefiniti con il driver dvips sono contenuti nel file `dvipsnam.def` in

`.../tex/latex/graphics/`.

`\color{`⟨*nome*⟩`}` dichiara che da questo momento in poi si userà il colore con il ⟨*nome*⟩ specificato. La sua azione viene delimitata dai gruppi o dagli ambienti.

`\textcolor{`⟨*nome*⟩`}{`⟨*testo*⟩`}` serve per colorare il ⟨*testo*⟩ con il colore denominato ⟨*nome*⟩.

`\colorbox{`⟨*sfondo*⟩`}{`⟨*testo*⟩`}` agisce come `\fbox`, ma il ⟨*testo*⟩ viene composto su uno sfondo colorato con il colore di nome ⟨*sfondo*⟩.

`\fcolorbox{`⟨*bordo*⟩`}{`⟨*sfondo*⟩`}{`⟨*testo*⟩`}` agisce come `\colorbox` salvo che la cornice della scatola non è nera ma colorata con il colore di nome ⟨*bordo*⟩. I parametri `\fboxsep` e `\fboxrule`, che definiscono lo spazio di separazione fra bordo e ⟨*testo*⟩ e lo spessore del bordo, sono gli stessi che regolano le caratteristiche della scatola incorniciata con `\fbox`.

`\pagecolor{`⟨*nome*⟩`}` questo comando globale assegna alle pagine intere uno sfondo con il colore denominato ⟨*nome*⟩ e questo colore permane finché non si dichiara un colore differente (eventualmente `white`).

1.16.2 L'ambiente *picture*

L'ambiente *picture* per fare semplici disegni al tratto sta a *figure* come *tabular* sta a *table*; forse questo paragone serve a comprendere meglio la differenza fra figura e immagine.

L'ambiente *originale*, nato insieme al vecchio LaTeX 209, era molto limitato nella scelta delle linee che potevano essere tracciate, perché venivano usati dei piccoli segmenti tratti da una polizza di caratteri specialmente predisposti per questo scopo; per quanto potessero essere ben disegnati, questi piccoli segmenti erano pur sempre in numero limitato.

Siccome esiste l'estensione standard `pict2e` di LaTeX, già descritta da Lamport nella sua guida del 1994, nel seguito mi riferirò solo a quanto si può fare con questo ambiente integrato con l'estensione `pict2e`. Bisogna invocare questa estensione mettendo nel preambolo il codice seguente:

```
\usepackage[⟨opzioni⟩]{pict2e}[2009/06/01]
```

dove `[`⟨*opzioni*⟩`]` può essere una o più fra le opzioni disponibili, la maggior parte delle quali serve per definire il tipo di driver che verrà usato per tradurre il risultato in una forma comprensibile agli umani. Per lo più

servono solo in casi particolarissimi e rari, ma ci sono altre tre opzioni interessanti:

original serve a impostare `pict2e` in modo che sia perfettamente equivalente al vecchio LaTeX; evidentemente serve solo per compatibilità con il passato.

ltxarrows serve per specificare a `pict2e` che le punte delle frecce devono essere disegnate come fa LaTeX in modo di compatibilità con il passato; siccome quelle frecce con punta triangolare leggermente concave sono molto gradevoli, il compositore può decidere di usare tali punte di freccia, anche se non introducono nessun elemento di novità nello stile del disegno.

pstarrows serve invece per disegnare le punte delle frecce come un aereo stealth, cioè come un aereo con le ali a delta; questa è la maniera di disegnare le frecce usata anche dal pacchetto `PSTricks`, eccellente pacchetto di estensione che si affida completamente al driver `dvips` per produrre un file di tipo PostScript.

Se non si specificano opzioni, il pacchetto cerca se esiste il file di configurazione `pict2e.cfg` e sceglie le opzioni che quel file riesce a definire in base al programma eseguito; inoltre, sceglie di default le frecce tradizionali di LaTeX.

A parte questi preliminari, l'ambiente *picture* definisce uno spazio grafico in cui l'unità di misura è definita dal parametro `\unitlength`, che di default vale 1 pt. Specificando un valore diverso è facilissimo scalare il disegno in modo da ingrandirlo o da rimpicciolirlo. Come trucco personale io preferisco definire per ogni ambiente *figure*, prima di aprire l'ambiente *picture*, un'unità di misura dipendente dal font in uso, per esempio 1 ex, così che cambiando il corpo del font anche il disegno viene scalato proporzionalmente. In altre circostanze, scelgo come unità di misura un centesimo della giustezza; in questo modo sono sicuro che, se non supero il valore di 100 unità in orizzontale, il disegno certamente non esce fuori dei margini. La sintassi da usare per impostare l'unità di misura è:

```
\setlength{\unitlength}{⟨lunghezza⟩}
```

Dopo questa prima operazione, l'ambiente *picture* con le sue dimensioni apparenti e il suo offset, nonché ogni distanza o lunghezza a cui si faccia

riferimento all'interno di questo ambiente, è indicata mediante un numero, senza specificare le unità di misura, perché queste, è sottinteso, sono identificate da `\unitlength`[16].

La sintassi di apertura dell'ambiente *picture* è la seguente:

```
\begin{picture}(⟨base⟩,⟨altezza⟩)(⟨x-offset⟩,⟨y-offset⟩)
...
\end{picture}
```

Si notino le parentesi tonde per specificare le coordinate (in multipli dell'unità di misura) che si intendono usare; la prima coppia di parentesi tonde contiene le dimensioni della base e dell'altezza 'apparenti' del disegno; *apparenti* significa che il disegno verrà inserito dentro un rettangolo di quelle dimensioni, ma alcune parti del disegno potrebbero fuoriuscire da quel rettangolo, al punto che sarebbe possibile (e di fatto lo è) dichiarare entrambe le dimensioni pari a zero.

La seconda coppia di parentesi è facoltativa; se c'è, essa indica la posizione dello spigolo inferiore sinistro del suddetto rettangolo rispetto all'origine degli assi x e y; questi sono gli assi rispetto ai quali vengono specificate le coordinate degli oggetti da collocare nel disegno. La figura 1.5 dovrebbe mostrare il significato di quanto esposto in questo capoverso.

Ogni oggetto da collocare nel disegno viene collocato mediante il comando `\put`; se l'oggetto deve essere collocato un certo numero di volte in posizioni regolarmente spaziate, si può usare il comando `\multiput`; le sintassi sono le seguenti:

[16] Il pacchetto `picture` di Oberdiek estende l'uso dell'ambiente *picture* anche al caso in cui le grandezze dimensionali siano specificate con le loro unità di misura; il pacchetto è compatibile con `pict2e`, ma deve essere caricato dopo, perché ridefinisce alcune macro. Molto comodo specialmente quando si vuole fare riferimento alle dimensioni di cose già esistenti nella pagina, come per esempio `\textwidth` o `\textheight` e simili dimensioni della geometria della pagina; trasformarle in quantità compatibili con `\unitlength` non sarebbe impossibile, ma è sicuramente noioso. Permette anche di usare i comandi `\heightof`, `\widthof`, nonché le le espressioni dimensionali introdotte da `\dimexpr`.

Esiste anche il pacchetto `xpicture` di Furster, che permette di fare disegni geometrici simili a quelli descritti qui, ma che risulta particolarmente comodo per scrivere testi adatti alle scuole secondarie superiori, con rette, parabole, iperboli, ellissi, polinomi di vario genere, e simili.

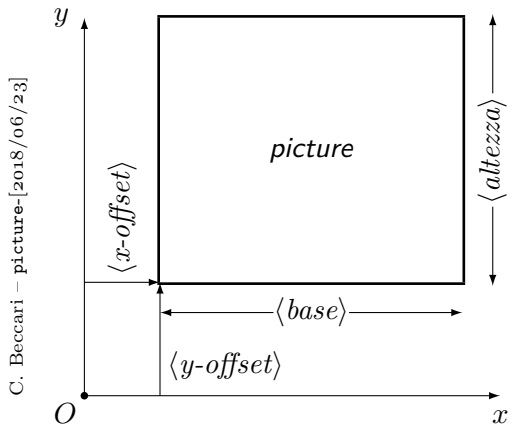

Figura 1.5 Relazione fra le dimensioni della figura e l'origine degli assi; i due offset possono anche essere negativi, così che l'origine degli assi cada dentro la figura.

```
\put(⟨x⟩,⟨y⟩){⟨oggetto⟩}
\multiput(⟨x-ini⟩,⟨y-ini⟩)(⟨x-step⟩,⟨y-step⟩){⟨numero⟩}{⟨oggetto⟩}
```

Le coordinate ⟨*x-ini*⟩ e ⟨*y-ini*⟩ sono le coordinate del primo punto; le coordinate ⟨*x-step*⟩ e ⟨*y-step*⟩ sono gli incrementi da dare ad x e ad y ogni volta che si deve inserire una nuova istanza dell'oggetto; il ⟨*numero*⟩ indica il numero totale di istanze dell'oggetto; infine ⟨*oggetto*⟩ specifica che cosa collocare nel disegno.

Per quanto riguarda `\put`, è bene precisare due o tre cose: gli spazi dopo la graffa chiusa sono ignorati del tutto, quindi non vengono introdotti spazi spuri dopo la graffa chiusa di questo comando anche se non si inserisce un segno di commento adiacente a destra della graffa chiusa; le coordinate `(`⟨*x*⟩`,`⟨*y*⟩`)` sono relative al punto di riferimento del codice circostante e ciò permette di annidare diversi comandi `\put` uno dentro l'⟨*oggetto*⟩ dell'altro senza problemi, così da poter dislocare facilmente oggetti che non si trovino nella posizione desiderata; questa proprietà può essere usata anche per dislocare uno o più comandi `\multiput`. Infine, l'⟨*oggetto*⟩ di `\put` e `\multiput` può contenere ogni genere di dichiarazione e di altri comandi validi in modo orizzontale; le dichiarazioni hanno valore solo per l'⟨*oggetto*⟩

specifico che le contiene. Nei codici degli esempi di questo paragrafo sono mostrate diverse applicazioni di queste proprietà.

Gli oggetti collocabili sono linee, vettori (frecce), cerchi, dischi, testi semplici o riquadrati con linee continue o tratteggiate; i testi possono essere collocati intelligentemente rispetto al rettangolo ideale che li contiene. Possono poi essere collocati degli 'ovali', cioè dei rettangoli con gli spigoli arrotondati, dentro ai quali può essere collocato del testo; possono essere fatti semplici disegni tracciando delle curve mediante l'uso di curve di Bézier di secondo o di terzo grado. La sintassi che descrive ognuno di questi oggetti è la seguente:

```
\line(⟨x-pend⟩,⟨y-pend⟩){⟨lunghezza⟩}
\vector(⟨x-pend⟩,⟨y-pend⟩){⟨lunghezza⟩}
\circle{⟨diametro⟩}
\circle*{⟨diametro⟩}
\makebox(⟨base⟩,⟨altezza⟩)[⟨posizione⟩]{⟨testo⟩}
\framebox(⟨base⟩,⟨altezza⟩)[⟨posizione⟩]{⟨testo⟩}
\dashbox{⟨lung-trattino⟩}(⟨base⟩,⟨altezza⟩)[⟨posizione⟩]{⟨testo⟩}
\oval[⟨raggio⟩](⟨base⟩,⟨altezza⟩)[⟨parte⟩]
\savebox{⟨control sequence⟩}(⟨base⟩,⟨altezza⟩)[⟨posizione⟩]{⟨testo⟩}
\usebox{⟨control sequence⟩}
```

I nomi dei parametri dovrebbero essere abbastanza chiari, ma vanno specificate alcune cose particolari.

La lunghezza dei segmenti e dei vettori è la componente orizzontale, cioè la proiezione sull'asse x del segmento o del vettore; se questo è completamente verticale, allora e solo allora la lunghezza coincide con quella del segmento o del vettore.

I coefficienti di pendenza dei segmenti e dei vettori rappresentano le proiezioni lungo l'asse x e y rispettivamente di un tratto di segmento o di vettore; siccome queste proiezioni non possono essere numeri arbitrariamente grandi, è evidente che conviene scegliere i valori più piccoli conformi con queste limitazioni arrotondando ai valori interi più vicini. Per un vettore verticale che ha la punta della freccia in alto, questi coefficienti formeranno la coppia `(0,1)`, ma si potrebbe indicare anche `(0,999)` con il medesimo risultato; per ovvi motivi sarebbe meglio usare la prima scrittura. Per i segmenti, che non hanno una punta di freccia che ne indichi la direzione, questa va intesa come la direzione del segmento dal suo estremo collocato

con `\put` all'altro estremo; in altre parole, il comando `\put` colloca alla sua coordinata il primo punto del segmento, da cui parte il segmento con la pendenza specificata.

Il comando `\circle` specifica una circonferenza completa di cui si specifica il ⟨*diametro*⟩; il suo punto di riferimento, messo in posizione con `\put`, è il suo centro. Il comando `\circle*` specifica invece un disco completo, non solo il contorno, quindi disegna un cerchio 'nero' con il ⟨*diametro*⟩ specificato.

I comandi `\makebox`, `\framebox` e `\dashbox` definiscono tre scatole, la prima senza che ne venga disegnato il contorno, la seconda con il contorno disegnato, la terza con il contorno tratteggiato con trattini lunghi ⟨*lung-trattino*⟩. Il comando `\put` mette in posizione il loro punto di riferimento, cioè il vertice inferiore sinistro. All'interno di questi rettangoli, o scatole, può essere posto del testo che viene collocato rispetto ai bordi ideali, disegnati o tratteggiati conformemente ai parametri di ⟨*posizione*⟩; questi sono le solite lettere `t`, `b`, `l`, `r` e `c`, con il rispettivo significato di 'top', 'bottom', 'left', 'right' e 'center'; per ogni oggetto si possono specificare al massimo due parametri di posizione[17], ricordando che 'center' è sempre il valore di default.

È particolarmente comodo il comando `\makebox` con dimensioni di ⟨*base*⟩ e ⟨*altezza*⟩ nulle, perché il ⟨*testo*⟩ che vi viene inserito risulta collocato con precisione rispetto allo spigolo inferiore sinistro, ma senza spazio alcuno interposto; non ha importanza che le dimensioni apparenti della scatola siano nulle; è importante che il testo sia collocato con un riferimento preciso, indipendentemente dalla presenza di ascendenti o discendenti.

Per il comando `\oval` è possibile specificare il raggio minimo del quarto di cerchio che costituisce lo spigolo; per altro, le dimensioni di ⟨*base*⟩ e ⟨*altezza*⟩ di questo speciale rettangolo a spigoli arrotondati si riferiscono al rettangolo completo; ma la specifica facoltativa di ⟨*parte*⟩ permette di scegliere quali angoli mostrare o quale coppia di angoli adiacenti mostrare, così che specificando raggio, metà della base e metà dell'altezza uguali è possibile disegnare solo un quarto di circonferenza oppure metà circonferenza; al solito, la ⟨*parte*⟩ è specificata con le lettere `l`, `t`, `b` e `r` (`c` non avrebbe

[17] Ovviamente non contrastanti. È ovvio che non si possono specificare simultaneamente `t` e `b`, perché lo stesso oggetto non può essere collocato allo stesso tempo in testa e al piede dentro la scatola specificata.

senso), cosicché `t` permette di disegnare solo la metà di sopra, `tl` solo il quarto in alto a sinistra.

I comandi `\savebox` e `\usebox` si servono di una scatola definita precedentemente all'uso dell'ambiente *picture* (sempre mediante il comando `\newsavebox` descritto nella pagina 120) e si usano come i corrispondenti comandi descritti in quelle pagine; cambia solo un pochino la sintassi di `\savebox`, perché richiede gli stessi argomenti che si usano per `\makebox` e `\frambeox`. Risulta molto comodo per salvare in una scatola delle parti ripetitive del disegno per poterle usare svariate volte in punti del disegno non facilmente gestibili con `\multiput`. Pur essendo presenti nel nucleo di LaTeX da molti anni, questi comandi non sono mai stati documentati nella guida di Lamport (1994).

Nell'ambito dell'ambiente *picture* è possibile specificare due spessori predefiniti per le linee da tracciare, `\thinlines` e `\thicklines`; si può specificare un valore assoluto di spessore (espresso con le unità di misura) mediante:

`\linethickness{`⟨*spessore assoluto*⟩`}`

e questo comando modifica lo spessore sia delle linee rette comunque inclinate, sia delle linee curve, siano esse archi di circonferenza oppure curve di Bézier quadratiche o cubiche.

Infine, i comandi `\qbezier` e `\cbezier` consentono di mettere in posizione delle curve di secondo o di terzo grado rispettivamente, dette *spline* di Bézier, senza bisogno di ricorrere ai comandi `\put` o `\multiput`. La loro sintassi è la seguente:

`\qbezier(`⟨x_1⟩`,`⟨y_1⟩`)(`⟨x_2⟩`,`⟨y_2⟩`)(`⟨x_3⟩`,`⟨y_3⟩`)`
`\cbezier(`⟨x_1⟩`,`⟨y_1⟩`)(`⟨x_2⟩`,`⟨y_2⟩`)(`⟨x_3⟩`,`⟨y_3⟩`)(`⟨x_4⟩`,`⟨y_4⟩`)`

Per la curva di secondo grado, descritta da `\qbezier`, (x_1, y_1) rappresentano le coordinate del punto da cui la curva parte in direzione del punto avente le coordinate (x_2, y_2); invece (x_3, y_3) rappresentano le coordinate del punto di arrivo con la direzione che proviene da (x_2, y_2).

Per la curva di terzo grado, descritta da `\cbezier`, questa parte da (x_1, y_1) in direzione (x_2, y_2) e arriva al punto (x_4, y_4) con la direzione proveniente da (x_3, y_3).

Scegliendo accuratamente i punti iniziali e finali, nonché le direzioni delle tangenti, si possono facilmente disegnare semplici diagrammi come mostrato nelle figure 1.6 e 1.7. I punti di controllo delle curve di Bézier sono rappresentati con dei puntini colorati relativi alla curva a cui si riferiscono; sono altresì disegnate le spezzate che uniscono i tre o i quattro punti delle curve di Bézier, che definiscono anche il *convex hull* dentro il quale si svolge ciascuna curva.

Va notato che nel 2009 il pacchetto `pict2e` è stato fortemente arricchito di nuove funzionalità che non sono delle semplici aggiustatine agli stessi comandi della versione tradizionale di LaTeX, ma sono delle estensioni decisamente importanti.

Innanzi tutto è stata definita una nuova routine di divisione fra numeri fratti che permette di dividere numeri di qualsiasi grandezza (purché inferiori in modulo a 16 384, sia dividendo, sia divisore, sia quoziente); per cui, la restrizione sul fatto che inizialmente (nella versione di `pict2e` del 2004) i coefficienti di pendenza di segmenti e vettori dovessero essere rappresentati da numeri interi inferiori in modulo a 1000 è stata rimossa; ora si possono usare tranquillamente numeri fratti e se, per esempio, si vuole una pendenza di 37°, il cui coseno vale 0, 79864 e il cui seno vale 0, 60182, si possono usare direttamente questi due numeri senza dover ricorrere all'artificio di moltiplicarli per 1000 e poi usare la sola parte intera arrotondata; non cambia molto da un punto di vista pratico, ma evita di dover fare calcoli e arrotondamenti.

Inoltre, sono stati aggiunti altri comandi, alcuni dei quali tracciano il loro disegno quando siano messi come argomento di `\put` e altri che sono assoluti, e cioè, come per le curve di Bézier, non hanno bisogno dell'intermediario del comando `\put`. Nello schema seguente sono evidenziati i comandi che richiedono di essere messi in posizione con `\put`; non è vietato usare il comando `\put` anche per quelli che non ne hanno bisogno, perché può servire per dislocare gli oggetti di quella parte di disegno che richiede uno spostamento, senza costringere l'utente a ricalcolare le coordinate che caratterizzano quegli oggetti.

Comandi da usare con `\put`:

`\arc[`⟨⟨*angolo1*⟩,⟨*angolo2*⟩⟩`]{`⟨*raggio*⟩`}`
`\arc*[`⟨⟨*angolo1*⟩,⟨*angolo2*⟩⟩`]{`⟨*raggio*⟩`}`

Comandi che non necessitano di `\put`:

`\Line(`⟨x_1⟩`,`⟨y_1⟩`)(`⟨x_2⟩`,`⟨y_2⟩`)`
`\polyline(`⟨x_1⟩`,`⟨y_1⟩`)...(`⟨x_n⟩`,`⟨y_n⟩`)`
`\polygon(`⟨x_1⟩`,`⟨y_1⟩`)...(`⟨x_n⟩`,`⟨y_n⟩`)`
`\polygon*(`⟨x_1⟩`,`⟨y_1⟩`)...(`⟨x_n⟩`,`⟨y_n⟩`)`

I comandi `\arc`, con o senza asterisco, generano un arco con il centro nella coordinata specificata dalle coordinate del comando `\put`, di raggio pari a ⟨*raggio*⟩, che inizia nell'angolo ⟨*angolo1*⟩ e termina nell'angolo ⟨*angolo2*⟩; gli angoli sono specificati in gradi e con il segno positivo si indica la rotazione in senso antiorario. La versione asteriscata riempie di colore (di default nero) il settore sotteso dall'angolo.

Il comando `\polyline` genera una spezzata che collega in successione gli n punti (x_1, y_1), (x_2, y_2), ..., (x_n, y_n); il numero di punti è arbitrario, perché il comando itera il proprio funzionamento fino ad esaurire la lista delle coordinate.

Il comando `\polygon` genera una spezzata chiusa nel senso che arrivato al punto n-esimo aggiunge ancora un segmento fino al primo punto della lista. Il comando `\polygon*` si comporta nello stesso modo, ma riempie di colore (di default nero) l'interno del poligono.

Inoltre, sono stati definiti nuovi comandi per specificare come devono essere formati gli estremi dei segmenti (tagliati dritti o arrotondati) e come debbano essere raccordati fra di loro i segmenti che formano le spezzate (se si prolungano fino a incontrarsi i lati che delimitano i due segmenti da unire, oppure se questo prolungamento viene troncato a una certa distanza dal punto di intersezione dei loro assi); per i dettagli si veda la documentazione. Si noti che con linee sottili questi dettagli sono virtualmente impercettibili, ma con linee relativamente spese la differenza c'è e si vede. Ricordiamoci che con `pict2e` si possono variare a piacere gli spessori di tutte le linee, quindi non è difficile ottenere segmenti di un certo spessore; semplicemente, si cerchi di non dimenticare che una linea di uno spessore di circa mezzo millimetro ($\approx 1,5\,\mathrm{pt}$) è una linea nerissima; si veda la figura 1.8.

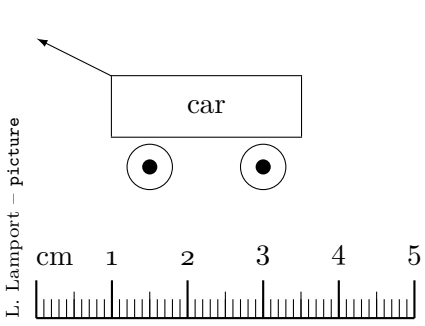

Figura 1.6 Il semplice disegno usato da Leslie Lamport per descrivere le potenzialità dell'ambiente *picture*

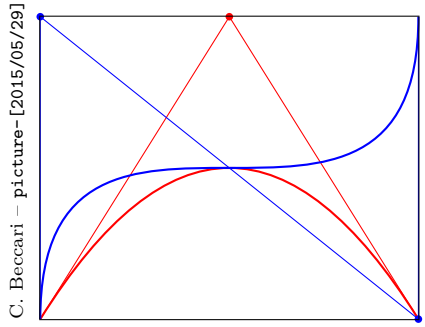

Figura 1.7 Due curve di Bézier di secondo e di terzo grado tracciate nell'ambiente *picture*

Usando i colori e i nuovi comandi di `pict2e` non è difficile disegnare il diagramma della figura 1.8.

Dopo aver definito nel preambolo il nuovo contatore `cms`, il codice per disegnare la figura 1.8 è il seguente:

```
\begin{figure}
\unitlength=8.5mm
\centering
\begin{picture}(11,11)(-1,-1)
\linethickness{0.35pt}
\multiput(0,0)(0.5,0){21}{\line(0,1){10}}
\multiput(0,0)(0,0.5){21}{\line(1,0){10}}
\linethickness{0.5pt}
\multiput(0,0)(1,0){11}{\line(0,1){10}}
\multiput(0,0)(0,1){11}{\line(1,0){10}}
\linethickness{0.7pt}
\setcounter{cms}{0}
\put(0,0){\framebox(10,10){}}
\multiput(0,-0.5)(1,0){11}{%
  \makebox(0,0)[b]{\arabic{cms}}\stepcounter{cms}}
\put(10.2,0){\rotatebox{90}{%
  \tiny C. Beccari -- \texttt{picture - 2009/06/01}}}
\put(5,-1){\makebox(0,0)[b]{Tempo in secondi}}
```

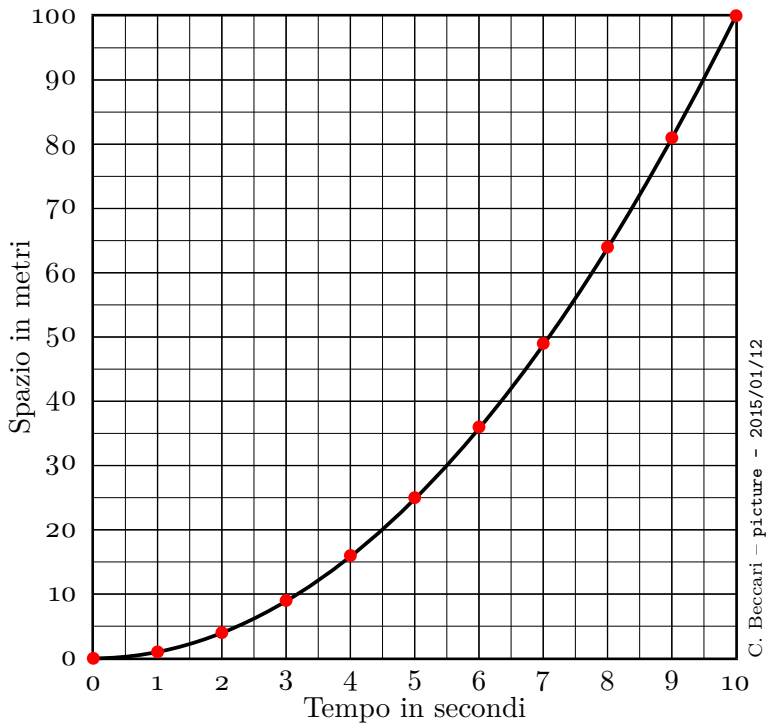

Figura 1.8 Moto uniformemente accelerato

```
\setcounter{cms}{0}
\multiput(-0.25,0)(0,1){11}{%
  \makebox(0,0)[r]{\arabic{cms}\addtocounter{cms}{10}}}
\put(-0.9,5){\rotatebox{90}{%
                    \makebox(0,0)[b]{Spazio in metri}}}
\linethickness{1.4pt}
\cbezier(0,0)(4.5,0)(8,6)(10,10)
\setcounter{cms}{-1}
\makeatletter
\multiput(0,0)%
  (1,\strip@pt\dimexpr\value{cms}\p@/10\relax){11}%
  {\color{red}\circle*{0.2}\addtocounter{cms}{2}}
\makeatother
```

```
\end{picture}
\caption{Moto uniformemente accelerato}%
  \label{fig:motoaccelerato}
\end{figure}
```

Vi sono molti aspetti un po' particolari, come per esempio l'uso delle scatole `\makebox` con le dimensioni dichiarate nulle; questo è un espediente che si usa spesso e serve per poter collocare il contenuto della scatola in una posizione precisa rispetto ai suoi bordi (invisibili), mantenendo nello stesso tempo un completo controllo sul punto di riferimento dell'oggetto da mettere in posizione o anche da ruotare, come la legenda della scala delle ordinate.

L'ultimo comando `\multiput` contiene argomenti un po' insoliti per definire l'incremento dell'ordinata di ciascun punto; usa il comando interno `\strip@pt`, il quale a sua volta contiene la "non-lettera" `@`, che non può far parte normalmente dei nomi delle sequenze di controllo; però la si può usare in un tratto di codice racchiuso fra `\makeatletter` e `\makeatother`; Lamport stesso indica questa operazione nel corpo del suo manuale. L'incremento delle ordinate è calcolato mediante la nota regola che la differenza di due quadrati di basi intere consecutive è un numero dispari; il calcolo è eseguito 'strappando via" le unità di misura da una espressione dimensionale, poiché abbiamo bisogno di ordinate espresse con numeri decimali fratti e non interi.

Sono disponibili anche i comandi che permettono di fare la cosiddetta *turtle graphics*; questa locuzione inglese si rifà a un gioco per bambini che consisteva nel prendere una tartaruga (domestica), nel metterla in un punto e poi nel farle fare un certo percorso allettandola in una direzione o in un'altra con del cibo. Questo procedimento in realtà è usato molto in ogni programma grafico quando si costruiscono percorsi andando da un punto iniziale verso una serie di punti intermedi seguendo delle linee curve o dritte secondo precise istruzioni.

Tutto ciò si può fare anche con i comandi di `pict2e`, che mette a disposizione del compositore i comandi seguenti:

```
\moveto(⟨x0 ⟩,⟨y0⟩)
\lineto(⟨xi ⟩,⟨yi⟩)
\curveto(⟨xi1 ⟩,⟨yi1⟩)(⟨xi2 ⟩,⟨yi2⟩)(⟨xi ⟩,⟨yi⟩)
\circlearc[⟨N⟩]{⟨x⟩}{⟨y⟩}{⟨R⟩}{⟨α1⟩}{⟨α2⟩}
\closepath
\strokepath
\fillpath
```

È chiaro dai nomi che `\moveto` mette la tartaruga, cioè la penna, nel punto iniziale; `\lineto` sposta la penna dall'ultimo punto raggiunto al punto specificato con un percorso rettilineo; `\curveto` sposta la penna dall'ultimo punto raggiunto fino al terzo punto specificato mediante un arco di curva di Bézier controllato dai punti di controllo specificati con le due coppie di coordinate precedenti.

Gli altri comandi sono leggermente diversi o hanno funzioni particolari per la gestione dei percorsi; `\circlearc` richiede di specificare un numero $\langle N \rangle$ con il quale si definisce il tipo di percorso iniziale a partire dall'ultimo punto raggiunto, per poi disegnare un arco di cerchio di centro ($\langle x \rangle$,$\langle y \rangle$) e raggio $\langle R \rangle$, che si sviluppa dall'angolo $\langle \alpha_1 \rangle$ all'angolo $\langle \alpha_2 \rangle$ in senso antiorario; per i dettagli si rinvia alla documentazione di `pict2e`. `\closepath` serve per chiudere il percorso congiungendo con un segmento l'ultimo punto specificato con il punto iniziale; non è necessario usare questo comando se l'ultimo comando usato specificava o portava a un punto finale coincidente con il punto iniziale.

Con questi comandi è stato solo specificato il percorso, ma bisogna dire al programma che cosa farne: `\strokepath` lo disegna effettivamente come una linea dello spessore specificato di default, oppure specificato con i comandi normali dell'ambiente *picture*; invece, `\fillpath` riempie il percorso chiuso con il colore predefinito (nero) o con quello definito con i comandi del pacchetto `xcolor` (oppure `color`).

Collocare qualcosa sulla pagina

Con questo titolo un po' criptico mi riferisco alla necessità di collocare una legenda una frase, un disegno, un "qualcosa", in una posizione precisa sulla pagina. È lo stesso problema affrontato da diversi pacchetti, per esempio, per mettere una filigrana sotto il testo di una o di tutte le pagine. Ma

l'utente potrebbe avere bisogno di collocare altri oggetti in posizioni precise, per esempio un logo, una scritta, una annotazione particolare.

Bisogna tenere presente se l'oggetto da collocare debba essere sotto il testo della pagina, oppure se faccia parte del testo. Farò un esempio con qualcosa da mettere sotto il testo. Vogliamo scrivere in diagonale la parola "BOZZA" sotto il testo componendola con un colore chiaro che non disturbi la lettura del testo sovrastante. Il problema di collocare qualcosa allo stesso livello del testo è più semplice, ma la tecnica è la stessa.

Bisogna trovare un punto fisso di riferimento della pagina che venga collocato prima del testo dalla routine di composizione della pagina. Questa routine mette in posizione prima le testatine, poi il blocco del testo e infine i piedini, quindi bisogna stare attenti a questi tre livelli; per qualcosa che deve stare sotto il testo bisogna affidarsi alle testatine, per qualcosa che deve stare sopra si può ricorrere ai piedini.

Possiamo scegliere la testatina o il piedino a ragion veduta; ma bisogna mettere un segnaposto nella testatina o nel piedino che possa ricevere la definizione di ciò che va scritto. Supponiamo di voler modificare solo le pagine composte con lo stile `headings`; potremmo ridefinire questo stile, e la via più semplice è quella di usare il pacchetto `fancyhdr`. Scriveremo quindi nel preambolo il codice seguente[18]:

```
\usepackage{fancyhdr}
\fancyhead[RO]{\segnaposto}
\fancyhead[RE]{\segnaposto}
\fancypagestyle{headings}{\fancyfoot[L]{\segnapiede}}
\let\segnaposto\empty
\let\segnapiede\empty
```

per mettere un "segnaposto" nella parte di destra della testatina, il cui angolo inferiore destro diventa quindi il punto di riferimento; possiamo mettere anche un altro "segnaposto' nel piedino per mettere altre informazioni. Inizialmente definiamo il `\segnaposto` e il `\segnapiede` affinché non contengano nulla (`\empty`); ma quando vogliamo che compaia la parola

[18] La definizione di `\fancyhead` per le pagine pari e dispari va adattata alla particolare classe, perché per qualche classe quell'area della testatina potrebbe essere già occupata da altro; quindi bisogna metterci anche le altre informazioni, lasciando per ultimo il comando `\segnaposto`.

"BOZZA" nelle pagine che vengono emesse, rendiamo questo comando equivalente al testo che vogliamo inserire, e quando vogliamo omettere la scritta, lo rendiamo equivalente a `\empty`

Aggiungiamo ora la definizione del comando che contiene la parola BOZZA, mettendolo in posizione con i comandi dell'ambiente *picture*; Preventivamente facciamoci un'idea dell'inclinazione della diagonale dello specchio di stampa; se questo ha altezza h e base b (misurando su una pagina composta e includendo nell'altezza anche il piedino e la testatina – operazione da eseguire con un semplicissimo righello) basta usare la calcolatrice che ogni sistema operativo mette a disposizione dei suoi utenti ed eseguire il calcolo $\arctan(h/b)$; misurando su una schermata di questa guida ridotta in modo da mostrare sullo schermo l'intera pagina, risulta $b = 57\,\text{mm}$ e $h = 90\,\text{mm}$; l'inclinazione della tangente è pertanto di $57{,}65°$, arrotondabile a $58°$, ma non è necessario farlo. Con lo stesso righello e la stessa approssimazione si può misurare la diagonale, pari a $107\,\text{mm}$. Il comando `\bozza` diventa quindi[19]:

```
\newcommand\bozza{\begin{picture}(0,0)\put(0,0){%
   \rotatebox{57.65}{\makebox(0,0)[r]{%
      \makebox[107mm]{\color{red!20!white}%
      \fontsize{40mm}{40mm}\selectfont BOZZA}}}}
\end{picture}}
```

Per poter usare la definizione di miscela di colori usata nel comando color, bisogna aver caricato nel preambolo il pacchetto `xcolor`; la miscela di rosso e di bianco indicata in quella miscela indica un colore rosso molto chiaro, che non è proprio rosa ma quasi.

Al momento in cui si vuole la scritta diagonale basta emettere il comando

```
\let\segnaposto\bozza
```

e, al contrario, quando si vuole cessare di scrivere quella scritta diagonale, basta dare il comando

[19] Per impostare font di dimensioni enormi come in questo caso, bisogna usare i comandi di basso livello di LaTeX; inoltre bisogna che il documento faccia uso di font vettoriali scalabili a qualunque corpo; i Latin Modern usati in questa guida lo consentono, ma ovviamente non sono gli unici; questo non si può invece fare con i font Computer Modern e i font CM-Super, perché sono accessibili solo in una serie discreta di corpi; vedi il paragrafo 1.17.2 nella pagina 158.

```
\let\segnaposto\empty
```

È chiaro che si può definire un comando simile a `\bozza`, ma con un contenuto diverso, e scrivere qualunque altra cosa in qualunque posizione della pagina; se si usa anche il pacchetto `picture`, si possono anche indicare le misure esplicite, non in multipli di `\unitlength`, cosa che rende le operazioni molto più semplici; per esempio:

```
\newcommand\compostoil[1][\today]{\begin{picture}(0,0)
\put(0,-2\baselineskip){Composto il #1}\end{picture}}
```

Fatto ciò, possiamo rendere il comando `\segnapiede` equivalente a `\compostoil` quando vogliamo mettere nella riga sotto all'eventuale numero di pagina l'indicazione di quando il testo è stato composto; specificando l'argomento facoltativo, possiamo specificare una data precisa da non cambiare di giorno in giorno; quindi scriveremo

```
\let\segnapiede\compostoil
```

se la data odierna è lo scopo dell'annotazione; oppure scriveremo

```
\renewcommand\segnapiede{\compostoil{29 febbraio 2015}}
```

se vogliamo specificare una data precisa (nell'esempio la data è volutamente una data inesistente). In entrambi i casi rimettere

```
\let\segnapiede\empty
```

annulla l'impostazione dell'annotazione al piede della pagina

Disegnare ellissi piene e vuote

Usare i comandi introdotti dal pacchetto `pict2e` per disegnare curve particolari è relativamente semplice, ma è utile per creare macro che le disegnino senza dover reinventare il modo di farlo ogni volta che se ne ha bisogno.

Proviamo a vedere come costruire una macro che disegni un'ellisse con il centro dove si vuole, con i semiassi desiderati e con la figura ruotata di un angolo specificato.

Abbiamo diverse possibilità, ma ci sono comodi i comandi del pacchetto `graphicx` per eseguire lo scalamento di un cerchio o di un arco di cerchio e

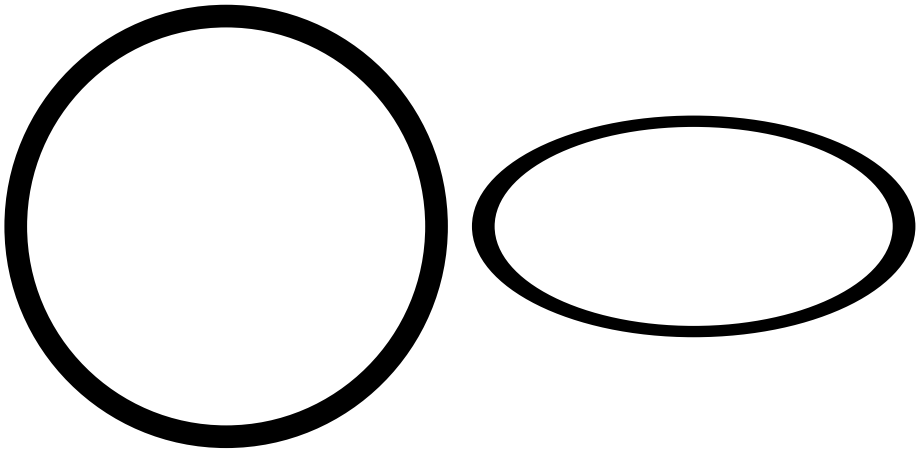

Figura 1.9 Sgradevole effetto causato dallo scalamento

per la sua rotazione; useremo quindi i comandi `\resizebox` e `\rotatebox` di quel pacchetto; in realtà useremo solo il secondo comando e vediamo subito il perché. Con riferimento alla figura 1.9, vediamo che cosa succede a usare i comandi di scalamento, e per renderlo più evidente si è usato un cerchio con il bordo molto spesso.

Si vede, dunque, che lo scalamento influisce anche sullo spessore della linea che traccia il contorno del cerchio; con linee più sottili queste potrebbero talvolta diventare troppo sottili e non essere più visibili, almeno nelle parti più sottili; noi invece vogliamo un'ellisse tracciata con un contorno di spessore costante.

Possiamo allora determinare le coordinate dei punti guida di archi di ellisse che si sviluppino su un quadrante; partiamo dall'arco di cerchio di 90° che si sviluppa nel primo quadrante; siccome l'ellisse non è altro che un cerchio scalato diversamente nelle direzioni dei due assi, diventa molto semplice dai punti caratteristici di una curva di Bézier relativi a un quarto di circonferenza di raggio unitario ricavare i punti caratteristici di un arco di ellisse del primo quadrante di semiasse orizzontale a e semiasse verticale b; per il cerchio ci riferiamo alla figura 1.10.

L'equazione dell'arco di curva di Bézier con i nodi estremi P_1 e P_2 e i

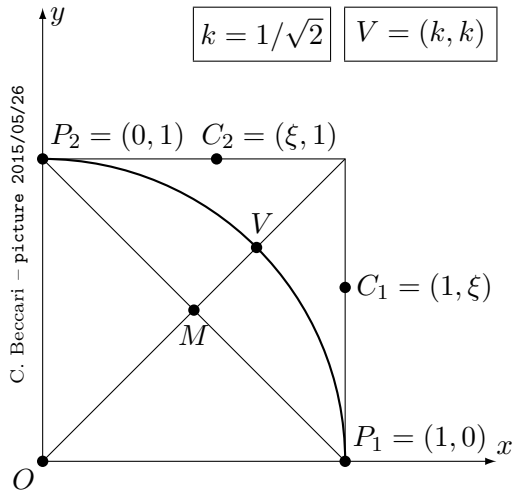

Figura 1.10 Arco di cerchio di 90° nel primo quadrante

punti di controllo C_1 e C_2 ha equazione:

$$P = P_1(1-t)^3 + 3C_1(1-t)^2t + 3C_2(1-t)t^2 + P_2t^3$$

dove il parametro t, variando da 0 a 1, fa spostare il punto P da P_1 a P_2 partendo e arrivando con le tangenti controllate dai punti di controllo C_1 e C_2. Bisogna determinare il valore di x in modo che il vertice $P = V$ dell'arco sia esattamente alle coordinate indicate nella figura, cioè a distanza unitaria dall'origine degli assi e a metà esatta dell'arco. È intuitivo che per raggiungere al metà del percorso, t debba valere $1/2$; semplici calcoli permettono di verificare che l'intuizione è esatta e che il parametro ξ cercato vale

$$\xi = \frac{4}{3}\left(\sqrt{2} - 1\right) = 0,552\,285$$

Scalando le distanze orizzontali di quanto vale il semiasse a e le distanze verticali di quanto vale il semiasse b, i nodi e i punti di controllo dell'arco di ellisse che giace nel primo quadrante diventano

$$\begin{aligned} P_1 &= (a, 0) \\ C_1 &= (a, b\xi) \\ C_2 &= (a\xi, b) \\ P_2 &= (0, b) \end{aligned}$$

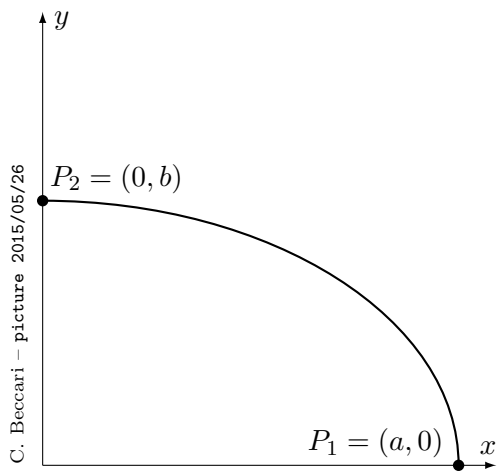

Figura 1.11 Quarto di ellisse nel primo quadrante

e l'arco di ellisse corrispondente appare disegnato nella figura 1.11.

Per gli archi di ellisse degli altri tre quadranti non resta che procedere per simmetria e ricavarne le coordinate dei nodi e dei punti di controllo; ciò fatto, diventa abbastanza semplice creare le cinque macro che ci servono per disegnare l'ellisse completa; più precisamente, vogliamo creare una macro con la sintassi seguente:

`\ellisse(`⟨x_C⟩`,`⟨y_C⟩`){`⟨*semiasse x*⟩`}{`⟨*semiasse y*⟩`}`

Questa macro è soltanto quella di base; la macro che l'utente userà avrà altri argomenti da passare a questa e per provvedere alla rotazione e alla colorazione dell'interno e/o del bordo. Avremo quindi bisogno di due macro.

1. La macro con la quale l'utente decide se tracciare solo il contorno o riempirla di colore tramite l'uso di un asterisco facoltativo.
2. Mediante un argomento facoltativo l'utente deve specificare di quanto vuole ruotare l'ellisse attorno al suo centro.
3. La stessa macro può provvedere alla posizione del centro; di default queste indicano l'origine, ma se l'utente specifica coordinate non nulle, la macro deve poter spostare l'ellisse in modo che il suo centro sia nel punto specificato dall'utente.

4. La stessa macro deve ricevere obbligatoriamente i due semiassi da passare alla macro `\ellisse`.
5. Sempre la stessa macro deve anche poter leggere le specifiche dei colori con cui eventualmente colorare l'interno e/o il contorno dell'ellisse; se questi colori non vengono specificati, l'interno resta bianco e l'eventuale contorno è nero.
6. Infine ci vuole la macro `\ellisse` che gestisca i dati che le vengono passati dalla prima macro.

La seconda macro è semplice, sulla base della spiegazione geometrica data sopra; deve usare i comandi di base per disegnare i quattro archi di ellisse che giacciono nei quattro quadranti attorno all'origine. Riceve solo i due semiassi e provvede a calcolare il fattore di scala individuato dalla macro `\x`, che permette di determinare le coordinate dei punti di controllo di ognuno dei quattro archi di ellisse.

```
\def\ellisse#1#2{%
\bgroup\def\a{#1}\def\b{#2}%
\dimendef\x=256 \x=0.552285\p@
\edef\ax{\strip@pt\dimexpr\a\x\relax}%
\edef\bx{\strip@pt\dimexpr\b\x\relax}%
\moveto(\a,0)
\curveto(\a,\bx)(\ax,\b)(0,\b)
\curveto(-\ax,\b)(-\a,\bx)(-\a,0)
\curveto(-\a,-\bx)(-\ax,-\b)(0,-\b)
\curveto(\ax,-\b)(\a,-\bx)(\a,0)
\fillstroke
\egroup}
```

L'unica macro 'nuova' è `\fillstroke`; le verrà assegnato un significato dalla prima macro e servirà a disegnare solo il contorno o a riempire di colore solo l'interno dell'ellisse.

La seconda macro è un po' più complessa. Se si vogliono usare i comandi nativi del sistema TEX, bisognerebbe creare tre o quattro macro che provvedono a separare e a leggere via via gli elementi facoltativi e quelli obbligatori.

Si fa molto meglio e molto prima a creare questa macro ricorrendo alle funzionalità del pacchetto `xparse` che sfrutta il linguaggio LATEX 3.

Ellissi disegnate ricorrendo al linguaggio L3

Il nuovo linguaggio L3 del futuro LaTeX 3 è allo studio per definire un ambiente di programmazione più robusto e in grado di gestire ogni possibile necessità che oggi viene risolta ricorrendo a una moltitudine di pacchetti esterni. Ovviamente qui non si scende nei dettagli e si rinvia il lettore alla documentazione del nuovo linguaggio, per esempio leggendo il file `expl3.pdf` (terminale: `texdoc expl3`).

Qui invece si userà una nuova interfaccia, basata su L3, costituita dal pacchetto `xparse` alla cui documentazione si rinvia il lettore per i molti dettagli che qui sarebbero fuori luogo; ci accontenteremo di descrivere qualche comando e alcuni dei parametri di cui il pacchetto fa uso; vedremo come il problema di disegnare un'ellissi arbitraria e arbitrariamente orientata e colorata possa essere completata con solo un comando di interfaccia fra l'utente e la macro `\ellisse` già descritta sopra.

Il comando che si descriverà è `\DeclareDocumentCommand`; esso è definito da `xparse`; è evidente che prima di usare questo nuovo comando bisogna avere caricato il pacchetto `xparse`. La sua sintassi è semplice:

`\DeclareDocumentCommand{`⟨*comando*⟩`}{`⟨*argomenti*⟩`}{`⟨*definizione*⟩`}`

Serve per definire un nuovo ⟨*comando*⟩, o a ridefinire un ⟨*comando*⟩ già definito, con il testo sostitutivo costituito dalla ⟨*definizione*⟩; fin qui non c'è nulla di nuovo se non per una piccola differenza di terminologia. La novità è costituita dagli ⟨*argomenti*⟩; questi sono una lista di descrittori che specificano se sono obbligatori o facoltativi; se sono 'normali' o delimitati; quali valori devono assumere quando siano facoltativi e non vengano specificati nella chiamata, eccetera. Qui non li si descrive tutti, ma si descrivono solo quelli che verranno effettivamente usati; tra parentesi si indica il nome o aggettivo inglese che ha dato origine alla sigla del particolare descrittore.

- `s` (star) specifica un asterisco facoltativo; se viene specificato, l'argomento corrispondente vale `true` e può venire controllato con la macro `\IfBooleanTF`. Più o meno corrisponde all'asterisco facoltativo di `\newcommand`.
- `D`⟨t_1⟩⟨t_2⟩`{`⟨*default*⟩`}` (delimited) specifica un argomento facoltativo delimitato a sinistra dal token ⟨t_1⟩ e a destra dal token ⟨t_2⟩, e il cui valore predefinito è ⟨*default*⟩; in qualche modo le coordinate che si

passano al comando `\put` potrebbero essere descritte dal descrittore `D()`, solo che per `\put` le coordinate sono obbligatorie, mentre qui si sta descrivendo un argomento delimitato facoltativo.

`O{`⟨*default*⟩`}` (optional) descrive un argomento facoltativo 'normale', cioè delimitato da parentesi quadre; ⟨*default*⟩ ne rappresenta il valore predefinito.

`o` (optional) descrive un argomento facoltativo senza valore predefinito. Si usa `\IfValueTF` per verificare se opzionalmente sia stato attribuito un valore. Vedi meglio sotto.

`m` (mandatory) specifica un argomento obbligatorio, come quelli 'normalmente' racchiusi fra parentesi graffe.

Definiamo la seconda macro, che è effettivamente l'unica che l'utente dovrebbe usare, `\Xellisse` in questo modo:

```
\NewDocumentCommand{\Xellipse}%
  { s D(){0,0} O{0} m m O{} o}%
    {\IfBooleanTF#1{\let\fillstroke\fillpath}%
                   {\let\fillstroke\strokepath}%
    \put(#2){\rotatebox{#3}{#6\ellipse@{#4}{#5}}%
    \IfValueTF{#7}{\let\fillstroke\strokepath
    #7\ellisse{#4}{#5}}{}}%
    }
```

Si riconosce subito che questo comando provvede a raccogliere tutti gli argomenti obbligatori e facoltativi come è stato descritto nella sottosezione precedente, ma connette direttamente l'utente con la macro `\ellisse` che effettivamente disegna l'ellisse; la sintassi di questa macro è semplicemente la seguente

`\Xellisse`⟨*⟩`(`⟨x_C⟩`,`⟨y_C⟩`)[`⟨*angolo*⟩`]{`⟨a⟩`}{`⟨b⟩`}[`⟨*dichiarazione*$_1$⟩`][`⟨*dichiarazione*$_2$⟩`]`

dove, però l'asterisco è facoltativo; la sua presenza viene verificata con il test `\IfBoleanTF` e implica il riempimento del colore specificato in ⟨*dichiarazione*$_1$⟩; le coordinate del centro sono facoltative `(`⟨x_C⟩`,`⟨y_C⟩`)` ma sono racchiuse fra parentesi tonde e il loro valore predefinito è `(0,0)`; non specificare queste coordinate implica l'uso di `\put` per mettere in posizione l'ellisse; omettendo `\put` l'ellisse avrà il suo centro collocato

nell'origine degli assi della tela. L'⟨*angolo*⟩, facoltativo ma con un valore di default nullo, specifica un angolo di rotazione dell'ellisse attorno al suo centro (positivo in senso antiorario). Poi, ⟨*a*⟩ (orizzontale) e ⟨*b*⟩ (verticale) sono i semiassi dell'ellisse; segue il primo parametro facoltativo costituito dalla ⟨*dichiarazione*$_1$⟩, che potrebbe essere la specificazione dello spessore del contorno, da specificare con un comando come `\thicklines` oppure `\linethickness{spessore}`; potrebbe anche essere la dichiarazione del colore con cui riempire l'intera ellisse. L'ultimo parametro, solo se viene specificato, può contenere una ⟨*dichiarazione*$_2$⟩, eventualmente anche nulla (semplicemente mediante `[]`), ma che permette di sovrapporre alla prima una seconda ellisse di cui si traccia solo il contorno; se la ⟨*dichiarazione*$_2$⟩ è nulla, il contorno viene disegnato nel colore e con lo spessore di default in quella fase del disegno, altrimenti essa può contenere l'una e/o l'altra specificazione di colore e di spessore. Va da sé che la ⟨*dichiarazione*$_2$⟩ ha senso se si specifica anche l'asterisco; ma non produce segnalazioni d'errore perché comunque sovrappone un secondo ellisse col solo bordo sopra il primo ellisse.

Con una sola macro di poche righe si fa ciò che con le definizioni eseguibili con i comandi nativi di TeX, avrebbero richiesto una sequenza, relativamente lunga ma molto delicata, di macro interagenti l'una con l'altra.

Con queste definizioni ci si può sbizzarrire con tutte le ellissi che si vogliono, come per esempio nella figura 1.12 create con il codice seguente:

```
\begin{figure}\unitlength=0.0097\textwidth\raggedleft
\begin{picture}(100,30)
\put(0,0){%
\put(0,0){\vector(1,0){30}}\put(28,2){$x$}
\put(0,0){\vector(0,1){30}}\put(2,28){$y$}
\Xellisse(15,15)[45]{15}{10}}
%
\put(35,0){%
\put(0,0){\vector(1,0){30}}\put(28,2){$x$}
\put(0,0){\vector(0,1){30}}\put(2,28){$y$}
\Xellisse*(15,15)[-45]{15}{7.5}}
%
\put(70,0){\definecolor{arancio}{rgb}{0.9686,0.5725,0.1922}
```

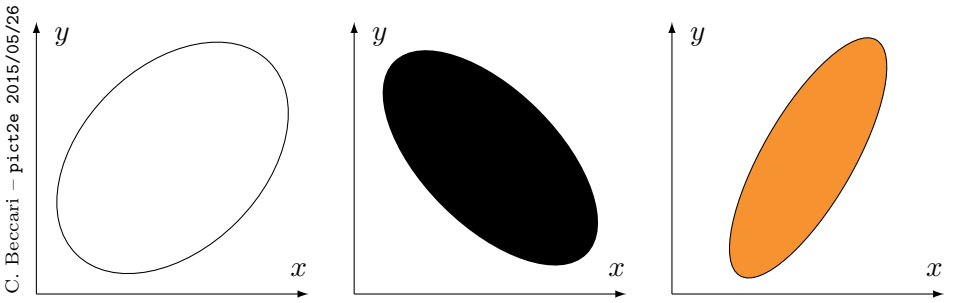

Figura 1.12 Diverse ellissi

```
\put(0,0){\vector(1,0){30}}\put(28,2){$x$}
\put(0,0){\vector(0,1){30}}\put(2,28){$y$}
\put(0,0){\color{arancio}\Xellisse*(15,15)[60]{15}{5}}
\Xellisse(15,15)[60]{15}{5}}
%
\put(-1.5,0){\rotatebox{90}{\makebox(0,0)[lb]{\tiny C. Beccari
     -- \texttt{pict2e 2015/05/26}}}}
\end{picture}
\caption{Diverse ellissi}
\label{fig:diverse-ellissi}
\end{figure}
```

Si noti che nel terzo esempio non si sono usati gli ultimi due argomenti facoltativi contenenti le dichiarazioni dei colori, ma si è specificato il colore arancio direttamente nell'argomento di `\put`; così facendo perché l'ellisse di colore arancio abbia il bordo nero, bisogna disegnare due ellissi con gli stessi parametri, ma con la prima creata con il comando asteriscato e preceduto dalla dichiarazione del colore, mentre la seconda ha il solo contorno disegnato con il colore predefinito (nero). Di per sé, il comando `\Xellisse` non ha bisogno di essere messo in posizione con il comando `\put`, ma in questo caso l'oggetto del comando `\put` contiene anche la dichiarazione del colore e la mantiene confinata al solo suo argomento. In questo esempio si vede anche come usare i comandi `\put` annidati per disegnare oggetti simili dislocandoli esattamente per farli stare sulla tela del disegno nelle posizioni desiderate senza bisogno di ricalcolare le coordinate dei vari oggetti. Tuttavia, come si può ben immaginare, quest'uso è un

po' artificioso e suscettibile di errori o dimenticanze. Ecco perché conviene servirsi degli ultimi due argomenti facoltativi. Il procedimento corretto viene illustrato qui di seguito.

Nella figura 1.13 si possono vedere alcune ellissi ruotate, colorate, col bordo colorato; si possono confrontare con il codice seguente usato per disegnarle.

```
\begin{picture}(115,100)(-30,-20)
\put(-35,-20){\framebox(116,100){}}
\put(-35,0){\vector(1,0){116}}
\put(0,-20){\vector(0,1){100}}
\put(0,35){\Xellipse*[120]{35}{20}}[\color{black!25!white}]%   grigio
\Xellisse(35,35)[45]{35}{20}%                          inclinato di 45°
\Xellisse{35}{20}%                               nell'origine degli assi
\Xellisse(35,35)[90]{35}{20}[\linethickness{1pt}]%         bordo nero
\put(70,35){\Xellisse*{10}{20}[\color{red!50!white}][%
   \linethickness{2mm}\color{blue}]}%         interno rosa, bordo blu
\put(20,70){\Xellisse{20}{10}[%
   \linethickness{2pt}\color{green!60!black}]}%           bordo verde
\Xellisse*(20,35)[-90]{5}{10}[\color{yellow!80!black}]% interno beige
\end{picture}
```

Considerazioni sul disegno programmato

Si noti questo semplice fatto: sono occorse soltanto circa una dozzina di righe di codice per definire le macro necessarie per disegnare ellissi variamente collocate sulla tela del disegno e variamente orientate; infatti, con il linguaggio L3 è stato possibile semplificare notevolmente. Quel che si vuole notare, infatti, è come l'essere in grado di definirsi le macro opportune semplifica molto il lavoro di composizione, non solo per fare disegni, ma in senso del tutto generale.

Programmi di disegno più avanzati, come quelli forniti dai pacchetti `pgf` con il suo ambiente *tikzpicture* (pacchetto `TikZ`) e con il pacchetto `pgfplots`, fanno cose molto più elaborate; praticamente permettono di disegnare qualunque cosa; se una dozzina di righe di codice sono necessarie per disegnare delle semplici ellissi, si immagini la complessità di quei pacchetti che dispongono di ogni strumento possibile per disegnare tutto quello che si desidera.

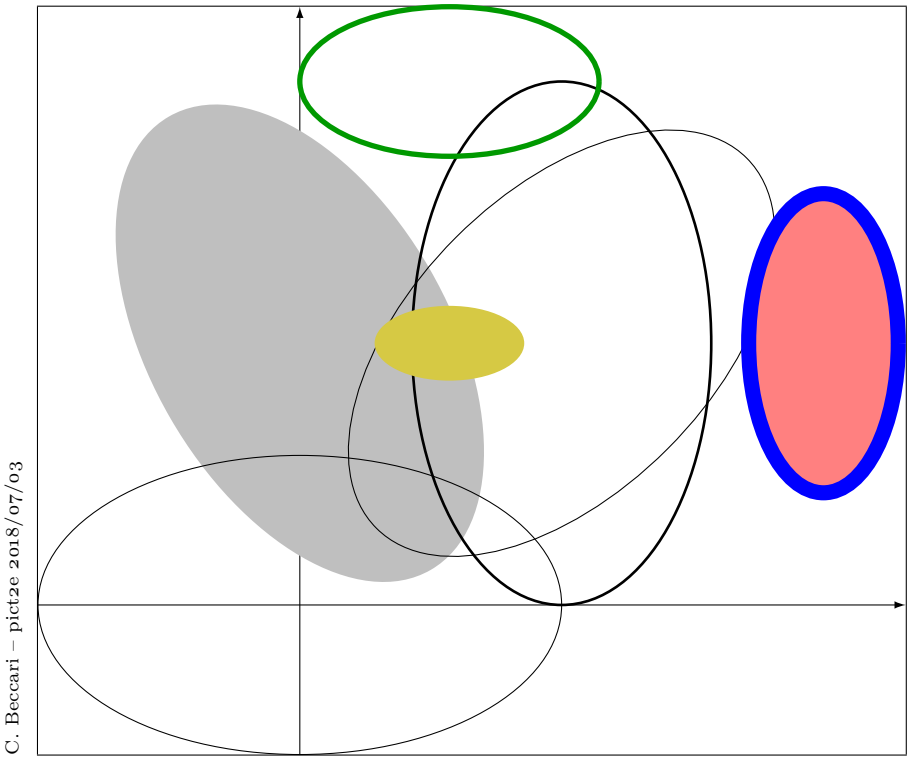

Figura 1.13 Ellissi variamente orientate e colorate con i bordi di diversi spessori

Disegno di un circuito elettrico e di una funzione di trasferimento

Un circuito elettrico Questo paragrafo è frutto dell'esperienza personale, e siccome il codice per disegnare circuiti contiene circa 3800 righe, è evidente che non può essere riportato qui. Tuttavia vorrei mostrare quello che si può fare con l'ambiente *picture* esteso con il pacchetto `pict2e`.

Nella figura 1.14 è disegnato un filtro elettrico; il codice utente per disegnarlo è semplice perché le complicazioni sono tutte nascoste nelle macro del pacchetto per disegnare circuiti; tuttavia vale la pena riportarlo per mostrare quanto poco occorra per completare il disegno:

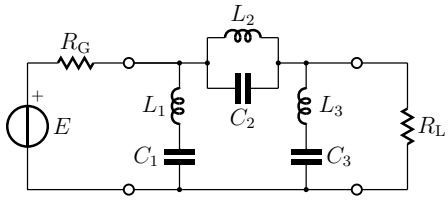

Figura 1.14 Un filtro elettrico eliminabanda (C. Beccari – 2015/03/31)

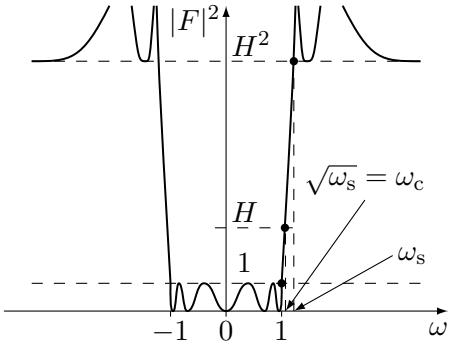

Figura 1.15 Modulo della funzione caratteristica di un filtro passabasso (C. Beccari – 2015/03/31)

```
\begin{circuito}(75,35)
\hconnect(0,0)(19,0)\HPolo(20,0)(65,0)
\polo(20,25)[30,25]\polo(65,25)[75,25]
\hconnect(66,0)(75,0)
\R(75,0)(75,25){R\ped{L}}
\E(0,0)(0,25){E}
\R(0,25)(19,25){R\ped{G}}
\serie*(30,0)(21,25)C{C_1}-L{L_1}
\parallelo(30,25)(55,25)L{L_2}-C{C_2}
\serie(55,0)(64,25)C{C_3}-L{L_3}
\nodi(55,0)(55,25)(30,25)(30,0)
\end{circuito}
```

Si tratta di 12 righe in tutto; ogni comando si spiega da solo, anche se un minimo di documentazione deve essere acquisita e letta; con `TikZ` esiste il modulo specifico per i circuiti elettrici `circuitikz`, ma nonostante la potenza di `TikZ`, a me pare che i circuiti disegnati con le mie macro siano più belli; è una questione di gusti, naturalmente; ed è logico che a me piacciano i miei disegni, ma non sottovaluto la potenza anche di quel modulo `circuitikz` che, tra l'altro, è in grado di disegnare componenti elettrici che io non ho previsto e quindi non ho implementato nel mio codice.

Una curva di risposta in frequenza Nella figura 1.15 è riportata la curva di risposta in frequenza (qualitativa) di un filtro passabasso. Il disegno è stato fatto completamente con le funzionalità dell'ambiente *picture* e con le macro (ri)definite dal pacchetto `pict2e`. Qui riporto il codice che, nella sua prima parte, contiene anche alcune macro che vanno messe nel preambolo del documento:

```
%------------------------------ nel preambolo del documento
\makeatletter
\def\SplitCoord(#1,#2)#3#4{\def#3{#1\p@}\def#4{#2\p@}}
\def\SubVect#1from#2to#3{%
\expandafter\SplitCoord\expandafter(#1)\@tA\@tB
\expandafter\SplitCoord\expandafter(#2)\@tC\@tD
\edef#3{\strip@pt\dimexpr\@tC-\@tA\relax,%
   \strip@pt\dimexpr\@tD-\@tB\relax}}
%
\def\Vector(#1,#2){\ifdim#1\p@=\z@
    \def\@tA{#2}\ifdim#2\p@<\z@\def\@tA{-#2}\fi
\else
    \def\@tA{#1}\ifdim#1\p@<\z@\def\@tA{-#1}\fi
\fi
\vector(#1,#2){\@tA}}
%
\def\VECTOR(#1)(#2){\begingroup
\SubVect#1from#2to\@tempa
\expandafter\put\expandafter(#1){%
   \expandafter\Vector\expandafter(\@tempa)}%
\endgroup\ignorespaces}
%
\def\Zbox(#1)[#2]#3{\put(#1){\makebox(0,0)[#2]{$#3$}}}
%---------------------------------- nel corpo del documento
\unitlength=0.0125\hsize
\begin{picture}(80,60)(-40,-5)
  \VECTOR(-40,0)(40,0)\Zbox(40,-2)[tr]{\omega}
  \VECTOR(0,-1)(0,55)\Zbox(-1,55)[tr]{|F|^2}
  \multiput(-35,5)(4,0){18}{\line(1,0){2}}
  \Zbox(2,7)[bl]{1}\multiput(-35,45)(4,0){18}{\line(1,0){2}}
  \Zbox(1,46)[bl]{H^2}\multiput(-2,15)(4,0){4}{\line(1,0){2}}
  \Zbox(1,16)[bl]{H}\Line(-10,0)(-10,-1)\Zbox(0,-2)[t]{0}
```

```
  \Zbox(-10,-2)[t]{-1}\Line(10,0)(10,-1)\Zbox(10,-2)[t]{1}
{\linethickness{0.75pt}
  \moveto(-12.5,55)\curveto(-12,40)(-12,40)(-10,5)
  \curveto(-10.1,0)(-9.75,0)(-9.5,0)\curveto(-9,0)(-9,5)(-8.5,5)
  \curveto(-7.75,5)(-7.75,0)(-7,0)\curveto(-5.5,0)(-5.5,5)(-4,5)
  \curveto(-2,5)(-2,0)(0,0)\strokepath
%
  \moveto(-13,55)\curveto(-13.75,45)(-13.75,45)(-14.5,45)
  \curveto(-15.75,45)(-15.75,45)(-17,55)\strokepath
%
  \cbezier(-21,55)(-26,45)(-28,45)(-35,45)
%
  \moveto(12.5,55)\curveto(12,40)(12,40)(10,5)
  \curveto(10.1,0)(9.75,0)(9.5,0)\curveto(9,0)(9,5)(8.5,5)
  \curveto(7.75,5)(7.75,0)(7,0) \curveto(5.5,0)(5.5,5)(4,5)
  \curveto(2,5)(2,0)(0,0)\strokepath
%
  \moveto(13,55)\curveto(13.75,45)(13.75,45)(14.5,45)
  \curveto(15.75,45)(15.75,45)(17,55)\strokepath
%
  \cbezier(21,55)(26,45)(28,45)(35,45)
}%
  \put(10.6,15){\circle*{1.5}}\put(12.2,45){\circle*{1.5}}
  \put(10,5){\circle*{1.5}}
  \multiput(12.2,0)(0,4){11}{\line(0,1){2}}
  \multiput(10.7,0)(0,4){4}{\line(0,1){2}}
  \VECTOR(25,20)(10.7,0)\VECTOR(30,10)(12.2,0)
  \Zbox(25,21)[b]{\sqrt{\omega\ped{s}}=\omega\ped{c}}
  \Zbox(31,10)[lc]{\omega\ped{s}}
\end{picture}
```

I comandi definiti nel preambolo servono per risparmiare un po' di fatica all'utente: `\Zbox` permette di specificare solo il necessario; in particolare, visto che serve per etichettare in modo matematico certi elementi del disegno, imposta direttamente il modo matematico per il suo argomento. `\VECTOR` è una macro un po' più complessa che disegna un vettore specificando solo le coordinate dei punti di inizio e di fine, così da risparmiare all'utente la fatica di calcolare i parametri da passare ai comandi nativi di *picture*. Il codice suddetto mostra chiaramente quanto si può fare con i modesti strumenti

dell'ambiente *picture* esteso con il pacchetto `pict2e`. Leggendo questo documento al calcolatore e ingrandendo i disegni con un ingrandimento corrispondente almeno ad un fattore 10, si constata che si tratta di disegni vettoriali, che non mostrano nessuna seghettatura dei contorni all'aumentare delle dimensioni del disegno; le figure a matrici di pixel non godono di questa proprietà e si vedono (abbastanza) bene solo se riprodotti con mezzi che abbiano essi stessi una densità di pixel maggiore o uguale alla quella dell'immagine.

Considerazioni sull'ambiente *picture*

L'ambiente *picture* è sì elementare, ma con le estensioni introdotte dal pacchetto `pict2e` e dalle macro che ogni utente può definire nel preambolo del suo documento, non è inferiore a molti altri software di disegno esterni al sistema TEX; ha il vantaggio che ogni informazione letterale o matematica che compare nel disegno è eseguita con gli stessi caratteri del resto del testo. Naturalmente non è l'unico mezzo per ottenere questo risultato, come d'altra parte si ricava da quanto esposto precedentemente e ribadito qui di seguito.

Non va dimenticato che per eseguire disegni molto più elaborati e rappresentazioni grafiche bi- o tri-dimensionali esistono i pacchetti `TikZ` e `pgfplots` che fanno cose molto più sofisticate; questi pacchetti sono compatibili sia con il programma latex (uscita in formato DVI, da trasformare in formato PS) sia con pdflatex, xelatex e lualatex (uscita finale in formato PDF); è vero che non sono pacchetti di base, ma lo stanno diventando di fatto e di diritto per l'uso diffusissimo che se ne sta facendo e per la vitalità che essi stanno dimostrando arricchendosi di continuo di nuove funzionalità.

1.17 Selezione dei caratteri

I comandi standard di LATEX che hanno a che vedere con la scelta dei font verranno descritti qui di seguito. Per gestire i font in modo più professionale, anzi, per caricare altre collezioni di font e per gestirle al meglio, è necessario documentarsi nella guida `fntguide.pdf`.

1.17.1 Scegliere famiglia, forma e serie

I comandi per scegliere la forma e/o la serie da cui trarre i font per la composizione sono raccolti nella tabella 1.14; nella tabella 1.15 sono invece appaiate le dichiarazioni e i corrispondenti comandi. Per la selezione del corpo si veda il prossimo paragrafo; solo la codifica non può essere cambiata mediante comandi di alto livello; se si vuole approfondire l'argomento si consulti la guida `fntguide.pdf` già citata.

Le dichiarazioni non possono essere usate in matematica; i comandi possono essere usati in matematica, e sono anche soggetti alla variazione automatica di corpo che compete agli indici e ai pedici di primo e di secondo livello *se e solo se* viene caricato anche il pacchetto `amsmath`, ma i risultati possono essere pessimi se i font matematici e quelli testuali non si accordano; per cui se ne sconsiglia l'uso, e si raccomanda, invece, di usare il corretto comando matematico `\mathrm` per i pedici e gli apici che normalmente contengono solo una parola o una abbreviazione, sempre scritti senza accenti. Si può ricorrere anche al comando `\text` fornito dal pacchetto `amsmath` e al corrispondente comando `\intertext` per intercalare del testo ad espressioni matematiche incolonnate. Nell'argomento di questi due comandi, se fosse necessario, si può ricorrere tanto ai comandi quanto alle dichiarazioni. Vale la pena di ricordare che tanto `\text` quanto `\intertext` usano il font testuale, non il tondo matematico o simili altri font matematici. Ci si ricordi, quindi, della necessità di scegliere i font testuali e quelli matematici in modo che si accordino gli uni con gli altri; di default, le collezioni di font CM, EC, LM, CM-super ed altre simili , si accordano sia in modo testo sia in modo matematico.

1.17.2 Scegliere il corpo

La dimensione dei caratteri è legata a nomi che fanno riferimento al corpo 'normale', quello, cioè, usato di default; per i corpi inferiori a 10 pt, la successione dei corpi procede di 1 pt in 1 pt; per i corpi superiori si procede con una successione geometrica di ragione 1,2; l'opzione di classe *11pt* corrisponde a un corpo di 10,95 pt, ma il numero magico 10,95 altro non è che $10 \times \sqrt{1,2}$. I comandi per la scelta dei corpi relativi a quello normale sono raccolti nella tabella 1.16. Questi comandi non possono venire usati in matematica, perché la matematica ha i simboli principali, non quelli per i

`\mdseries`	Serie media	`\upshape`	Forma diritta
`\bfseries`	**Serie nera**	`\itshape`	*Forma corsiva*
`\rmfamily`	Tondo con grazie	`\slshape`	*Forma inclinata*
`\sffamily`	Lineare senza grazie	`\scshape`	Maiuscoletto
`\ttfamily`	`Monospaziato`	`\normalfont`	Font di default

Tabella 1.14 Dichiarazioni per la scelta di famiglia, serie e forma

`\textmd{`⟨*testo*⟩`}`	`\mdseries`	`\textup{`⟨*testo*⟩`}`	`\upshape`
`\textbf{`⟨*testo*⟩`}`	`\bfseries`	`\textit{`⟨*testo*⟩`}`	`\itshape`
`\textrm{`⟨*testo*⟩`}`	`\rmfamily`	`\textsl{`⟨*testo*⟩`}`	`\slshape`
`\textsf{`⟨*testo*⟩`}`	`\sffamily`	`\textsc{`⟨*testo*⟩`}`	`\scshape`
`\texttt{`⟨*testo*⟩`}`	`\ttfamily`	`\textnormal{`⟨*testo*⟩`}`	`\normalfont`

Tabella 1.15 Corrispondenza fra comandi e dichiarazioni

Corpo normale	10 pt	11 pt	12 pt
`\tiny`	5.00 pt	6.00 pt	7.00 pt
`\scriptsize`	7.00 pt	8.00 pt	8.00 pt
`\footnotesize`	8.00 pt	9.00 pt	10.00 pt
`\small`	9.00 pt	10.00 pt	10.95 pt
`\normalsize`	10.00 pt	10.95 pt	12.00 pt
`\large`	12.00 pt	12.00 pt	14.40 pt
`\Large`	14.40 pt	14.40 pt	17.28 pt
`\LARGE`	17.28 pt	17.28 pt	20.74 pt
`\huge`	20.74 pt	20.74 pt	24.88 pt
`\Huge`	24.88 pt	24.88 pt	24.88 pt

Tabella 1.16 Dichiarazioni di corpo

pedici e gli apici di primo o secondo livello, composti nello stesso corpo del testo circostante.

Non tutte le famiglie contengono tutte le serie, per esempio la famiglia dei font monospaziati non contiene la serie nera. Non tutte le serie contengono tutte le forme; per esempio, la serie nera non contiene il maiuscoletto. Non tutte le forme contengono tutti i corpi; in questo caso il programma di composizione sceglie fra i corpi disponibili quello che maggiormente si avvicina al corpo richiesto.

Questa mancanza di alcuni corpi può essere un problema con i font bitmapped; lo è di meno con i font vettoriali, ma lo è certamente per quei font che sono disponibili solo per corpi discreti, come specificato nei loro file di descrizione (file con estensione `.fd`), mentre non ci sono problemi con i font accompagnati da file di descrizione che specificano quale font fra quelli disponibili sia da scalare per produrre il corpo desiderato; perché questo abbia successo, è necessario che i font siano vettoriali e che i loro file di descrizione specifichino gli intervalli di corpi che possono essere realizzati con continuità a partire da uno dei font disponibili il cui corpo sia il più vicino a quello desiderato. Fra questi font sono da segnalare i font Latin Modern, perché esistono solo in forma vettoriale, mentre per i font Extended Cork encoding (EC), dove sono disponibili sia i font bitmapped sia quelli vettoriali, l'unico file di descrizione di ciascuna famiglia non può che specificare valori di corpi discreti e non può specificare intervalli di ingrandimento; l'operazione di ingrandimento potrebbe essere fatta con i font EC vettoriali, ma non può essere fatta agevolmente con i font bitmapped.

Se si stanno usando font vettoriali e si vogliono impostare corpi diversi da quelli indicati nella tabella 1.16, e se i file di descrizione lo consentono mediante la specificazione di intervalli, non di valori discreti, allora si possono usare i comandi LaTeX di basso livello con la sintassi

`\fontsize{`⟨*corpo*⟩`}{`⟨*scartamento*⟩`}\selectfont`

dove ⟨*corpo*⟩ è una indicazione in punti tipografici (se non si esplicitano le unità di misura) o una indicazione assoluta (se si esplicitano le unità di misura) per indicare il corpo desiderato; ⟨*scartamento*⟩ è una quantità in punti o assoluta a seconda se si specificano le unità di misura; ⟨*scartamento*⟩ indica la distanza delle linee di base di due righe consecutive composte con

i caratteri di quel ⟨*corpo*⟩; di solito lo si specifica del 20% maggiore del corpo, ma per scopi speciali può anche avere un valore maggiore o minore, anche minore del corpo; in quest'ultimo caso si parla di righe *sterlineate*, dove i caratteri si sovrappongono in parte. Ho usato queste impostazioni nell'esempio del paragrafo 1.16.2.

Tutto ciò è oscuro? È possibile, ma generalmente gli utenti non cambiano corpo così alla leggera rispetto ai numerosi valori discreti già previsti e mostrati nella tabella 1.16. In questa guida si sono usati i font Latin Modern (estesi, grazie al pacchetto `cfr-lm`) scalabili con continuità (o, per essere più precisi, in modo lineare a tratti) e si sono anche usati corpi, per esempio di 10.5 pt, non ottenibili con altri font della dotazione di base di font del sistema TEX (Computer Modern e Extended Cork encoding)

1.17.3 Corpi testuali e matematici

Si ricorda che fra i comandi disponibili (ma raramente usati) è disponibile il comando per associare i quattro corpi che servono per il testo e gli indici superiori e inferiori di primo e di secondo ordine per la matematica:

`\DeclareMathSizes{`⟨*testo*⟩`}{`⟨*textsize*⟩`}{`⟨*subscriptsize*⟩`}{`⟨*subsubscriptsize*⟩`}`

dove al corpo testuale ⟨*testo*⟩ vengono associati i tre corpi per la matematica: ⟨*textstyle*⟩ per i simboli delle espressioni, ⟨*subscriptstyle*⟩ per gli indici e i pedici di primo livello e ⟨*subsubscriptstyle*⟩ per gli indici e i pedici di secondo livello.

Queste dichiarazioni sono più utili di quanto si pensi, perché talvolta nelle classi standard è desiderabile apportare qualche correzione, specialmente per i corpi più piccoli e più grandi.

1.17.4 Simboli speciali

Qualunque segno di qualunque font può venire stampato prescindendo dalle sue caratteristiche particolari; il comando

`\symbol`⟨*indirizzo*⟩

permette di accedere a qualunque simbolo di qualunque collezione di caratteri mediante il suo ⟨*indirizzo*⟩; il carattere viene stampato prescindendo dalle sue caratteristiche speciali o dall'esistenza di un comando specifico per

usarlo; l'unica questione è conoscerne l'indirizzo decimale, oppure ottale, oppure esadecimale. Attenzione: gli indirizzi si riferiscono alla polizza di font usati per la composizione, non ai caratteri introdotti o introducibili con la tastiera; per esempio, le due parentesi graffe nell'encoding di entrata hanno indirizzi 123 e 125 rispettivamente: se però si usano questi indirizzi con un font con codifica OT1, si ottiene – e ″, il che corrisponde perfettamente con quanto si può dedurre dall'esame delle collezioni di caratteri; per una consultazione rapida si veda la Guida Tematica di Beccari e Gordini (2018).

Questo fatto è collegato al modo diverso di interpretare i caratteri immessi con la tastiera nel file sorgente `.tex` e soggetti alla codifica di entrata specificata con l'opzione fornita al pacchetto `inputenc`, rispetto alla codifica usata per i font di uscita, specificata con l'opzione passata al pacchetto `fontenc`. Si veda più dettagliatamente questa questione nella Guida Tematica di Beccari e Gordini (2018).

Postfazione

Sotto questo titolo pomposo vorrei esprimere alcune considerazioni personali.

L'appendice C *Reference Manual* di Leslie Lamport consiste di 62 pagine; io espando questo stesso materiale a circa 150 pagine, più del doppio. Forse ho esagerato, ma nel commentare il mark up di LaTeX ci ho messo anche i molti anni di esperienza, visto che uso questo mark up dal 1986. Avrei potuto essere molto più ricco di dettagli, ma poi invece di scrivere un semplice commento avrei scritto una guida completa. Il lettore mi perdoni se ho tralasciato alcune parti importanti o se ritiene che abbia esposto delle considerazioni personali errate.

Ho ricevuto molto feedback dai lettori; li ringrazio tutti moltissimo, perché mi hanno permesso di eliminare molti refusi e di sfoltire alcune parti oppure di approfondirne altre. Grazie al feedback dei lettori nelle prossime versioni cercherò di eliminare le molte imperfezioni residue.

Nello stesso tempo vorrei ringraziare Leslie Lamport; il suo mark up LaTeX ha lasciato una impronta profonda di incommensurabile utilità nella scrittura *tecnico-scientifica*, denominazione che vorrei non venisse limitata in modo restrittivo alle scienze che fanno uso di grandezze misurabili; LaTeX ha già aiutato moltissimi filologi, letterati, storici, giuristi, economisti, linguisti, economisti, oltre che matematici, fisici, ingegneri, chimici e simili. Il suo mark up è talmente vivo e solido che per arrivare alla prossima versione 3 saranno passati tre o quattro lustri da quando vi si è cominciato a lavorare. Oggi (2019) sono già disponibili alcune parti della prossima versione e sono già incluse in molte distribuzioni del sistema TeX; i pacchetti più recenti fanno già uso di queste parti "sperimentali" ma ormai abbastanza consolidate. Lamport ha seminato un piccolo grande seme: oggi è diventato un grandissimo albero.

Bibliografia

Beccari, C. (2018). «Introduzione all'arte della composizione tipografica con LaTeX». PDF document. URL http://www.guitex.org/home/images/doc/guidaguit-a4.pdf.

Beccari, C. e Gordini, T. (2018). «Introduzione alle codifiche in entrata e in uscita». http://www.guitex.org/home/images/doc/GuideGuIT/introcodifiche.pdf.

Berry, K. (2013). «The TeX Live guide». PDF document. Leggibile con `texdoc texlive` nella distribuzione TeX Live.

Gregorio, E. (2010). «LaTeX, breve guida ai pacchetti di uso più comune». PDF document. URL http://profs.sci.univr.it/~gregorio/egtex.html.

Kopka, H. e Daly, P. W. (2004). *Guide to LaTeX*. Addison Wesley, 4ª edizione.

Lamport, L. (1994). *LaTeX: A document preparation system – User guide and reference manual*. Addison Wesley, 2ª edizione.

Mittelbach, F. *et al.* (2006). «LaTeX font encodings». PDF document. Leggibile con `texdoc encguide` nella distribuzione TeX Live.

Pantieri, L. e Gordini, T. (2018). «L'arte di scrivere con LaTeX». PDF document. URL http://www.lorenzopantieri.net/LaTeX_files/ArteLaTeX.pdf.

Indice analitico

B

C

D

E

F

G

H

I

J

K

L

M

N

P

Q

R

S

T

U

V

W

X

Z