Il PostScript in LaTeX

Riccardo Nisi

Sommario

L'ambiente PSTricks con i suoi collegamenti al PostScript mi ha dato lo spunto per interessarmi a questo linguaggio cercando ulteriori occasioni per integrarlo con LaTeX, anche accrescendone il campo applicativo.

Abstract

PSTricks, along with his links to PostScript, gave me the chance to take an interest in this language, searching for further occasions to integrate it with LaTeX, and enrich its application scope.

1 Introduzione

I documenti TeX e LaTeX, da quelli semplici a quelli complessi, sono in grado di soddisfare le esigenze più varie e, se si considera la disponibilità dei numerosissimi *package*, anche quelle più particolari. Ad esempio la grafica, specialmente in ambiente scientifico, è ben risolta dai pacchetti PSTricks.

Certo per calcolare il codominio di una funzione reale l'ambiente di calcolo non può essere TeX. LaTeX crea il documento in cui si scrive del contesto della funzione e in cui la si stampa, ma è PSTricks che ricorrendo all'ambiente PostScript calcola le coordinate della funzione e traccia il grafico. LaTeX gestisce tutto così bene che quasi non ci si accorge della sortita. Per riconoscere la diversità dell'ambiente occorre leggere o pensare di scrivere l'equazione che calcola le coordinate della funzione o della curva, soprattutto con una versione non recentissima di PSTricks. In alcuni casi, tuttavia, quando si tratta di effettuare calcoli con delle distanze, è possibile utilizzare direttamente TeX, come negli esempi della punteggiata e della spirale archimedea in cui si confrontano l'uso della macro `\multido` con il calcolo effettuato da TeX e dal PostScript. Ma TeX limita il calcolo numerico agli interi. È proprio questo limite che, a partire dal caso di una elementare tabella che prevedeva oltre alla rappresentazione di dati anche il calcolo di alcune percentuali con cifre decimali, mi ha spinto a guardare al PostScript. In questo caso, come nei due precedenti, l'ho fatto comparando procedimento e risultato del PostScript con quello di TeX. Nei casi delle curve e del calcolo dei coefficienti angolari non c'è alternativa all'uso del PostScript.

Poiché l'approccio al PostScript, pur limitato ad alcuni aspetti semplici, ha significato per un autodidatta come me un notevole anche se stimolante impegno, ho cercato di raccontarlo provando a chiarire, nella misura parziale e provvisoria di cui sono stato capace, alcuni aspetti di quel linguaggio che a mio parere sono essenziali per iniziare a comprenderlo. Questo lo farò nella sezione 2.2 *Esecuzione del codice* e in maniera più particolareggiata nella presentazione degli esempi e nel loro commento.

2 Uno sguardo al PostScript

Il PostScript (Adobe System Inc., 1991; Utting, 1992), un vero e proprio linguaggio di programmazione, nasce per descrivere la pagina di un documento con una resa grafica ottimale. L'interprete PostScript utilizza gli oggetti del linguaggio immettendoli in una memoria particolare, lo stack o pila, in cui l'ultimo elemento immesso è il primo ad essere estratto: gli elementi sono letti ed elaborati nell'ordine inverso a quello in cui sono immessi. Proprio da questa gestione degli oggetti del linguaggio discende la sua caratteristica notazione postfissa in cui l'operatore segue gli operandi: `a b add`. Il primo elemento ad essere immesso nello stack è l'operando `a` (un numero reale), il secondo è il numero `b` e infine l'operatore `add`, che è il più alto dello stack, detto stack degli operandi. La lettura da destra a sinistra dice a noi e all'interprete PostScript che l'operatore ha i due operandi previsti e che l'operazione può essere eseguita: i tre elementi della operazione vengono tolti dallo stack e viene eseguita la somma, inserita a sua volta nello stack.

L'area su cui si appoggiano tutte le comunicazioni, per passare i parametri ed eseguire le operazioni del linguaggio, è lo stack degli operandi. Ogni oggetto su cui si deve intervenire deve essere inserito nello stack. Per oggetto si intende qualsiasi entità del linguaggio PostScript. Un oggetto semplice è contenuto in una determinata dimensione, la stessa per ogni oggetto semplice, che in genere è di 8 byte. Gli oggetti composti come le stringhe e gli array hanno il nome associato ad un puntatore che si riferisce alla locazione del contenuto. Nel caso di duplicazione, mentre un oggetto semplice duplica la coppia (*nome*, *valore*), un oggetto composto duplica solo il nome: un nuovo puntatore che si riferisce allo stesso indirizzo di memoria dell'oggetto originale. Intervenire sugli oggetti dello stack degli operandi è anche il modo più diretto e veloce di farlo. Si vedrà che creare variabili e procedure implica un doppio passaggio tra stack degli operandi e dizionari che rallenta il processo esecutivo.

I programmi PostScript sono scritti in caratteri ASCII e raggruppati in alcuni costrutti sintattici. Quando si lancia un programma, il flusso dei dati viene letto dallo scanner che ha il compito di convertire i costrutti incontrati (esaminati uno alla volta) in oggetti che passa all'interprete. Distingue il nome di una procedura e di una variabile inteso in senso letterale (con lo slash) da un nome eseguibile.

In tabella 1 presentiamo un elenco dei costrutti sintattici.

2.1 Gli operatori utilizzati

Il PostScript ha una collezione molto ampia di operatori che sono raccolti in un dizionario (system dictionary). Tra gli altri esistono operatori matematici, booleani, di controllo del flusso, di manipolazione degli stack, per la grafica, per il testo. I font sono raccolti in uno specifico dizionario.

Aritmetici e matematici

Gli operatori `add`, `sub`, `div` e `mul` prevedono due operandi razionali. In particolare, in `a b div`, `a` è il dividendo e `b` il divisore e in `a b sub`, `a` è il minuendo e `b` il sottraendo. Gli operatori `idiv` e `mod` prevedono due operandi interi; il primo restituisce la parte intera del quoziente il secondo il resto della divisione. `neg`, `abs` e `round` hanno un solo operando razionale; `sqrt` un operando razionale non negativo. Operatori matematici sono `sin`, `cos` e `atan`: l'operando dei primi due è un angolo in gradi sessagesimali, l'operando di `atan` è un razionale, accetta $\frac{n}{0}$, non accetta la frazione $\frac{0}{0}$ e restituisce un angolo in gradi sessagesimali.

Manipolazione dello stack

`pop`, `exch`, `dup` e `roll` sono adibiti a manipolare lo stack.
`pop` elimina l'elemento top dello stack.
Se gli operandi sono gli oggetti `a` e `b`, `exch` li restituisce nell'ordine inverso: `a b exch` restituisce `b a`.
`dup` duplica l'ultimo elemento dello stack degli operandi: il frammento di codice `10 dup mul` restituisce `100`.
L'operatore `roll n m` agisce sugli ultimi `n` elementi dello stack cambiandone `m` volte le posizioni con una sorta di rotazione. Il frammento di codice `a b c roll 3 1` restituisce `c a b`: agisce sugli ultimi 3 elementi dello stack (quelli rappresentati) facendoli scorrere di una posizione verso destra e recuperando a sinistra l'elemento che è dovuto uscire dal gruppo di tre. `a b c roll 3 2` effettua due spostamenti verso destra, restituendo `b c a`. `a b c roll 3 -1` esegue uno spostamento verso sinistra, restituendo `b c a`.

Operatori di controllo

Tra questi operatori, essenzialmente cicli e istruzioni condizionate, troviamo `if`, `ifelse`, `repeat` e `for`.
`bool Procedura if`: se `bool` è vero esegue `Procedura`.
`bool Procedura1 Procedura2 ifelse`: in caso `bool` sia vero esegue la `Procedura1`, altrimenti `Procedura2`.
`n repeat Procedura`: ripete `n` volte `Procedura`.
L'operatore `for` ha quattro argomenti: `iniz`, `incr`, `lim` e `Procedura`. Ad esempio, in `1 2 8 add for`, i primi tre argomenti dicono che a partire da 1 con incremento 2 si creano nuovi elementi senza che superino il valore limite 8. Gli elementi costruiti sono 1, 3, 5, 7. Questi valori sono passati uno alla volta alla procedura che in questo caso li addiziona calcolando la loro somma. Il ciclo restituisce il numero 16.

Operatori su stringhe

Questi sono: `string`, `stringwidth` e `show`.
`n string` crea una stringa vuota di `n` caratteri.
`(Stringa) show` stampa la `Stringa`.
`(Stringa) stringwidth` calcola preventivamente gli spostamenti che subisce il punto corrente per la stampa di `Stringa` nel font corrente. L'ascissa dello spostamento esprime l'ampiezza della stringa. Non c'è spostamento verticale.

Tabella 1: Elenco dei costrutti sintattici di PostScript

Costrutto	Significato
Numero	Interi e reali, rappresentati in base 10 o in altre basi riconosciute dallo scanner. Il numero decimale 100 in base 2 è 2#1100100, 8#144 in base 8, 16#64 in base 16.
Stringa	Sequenza di caratteri ASCII, anche su più righe, racchiusa da parentesi tonde. Ad esempio (sono una stringa) è una stringa.
Commento	Qualsiasi cosa compresa tra il carattere '%' e il fine linea.
Nome	È una stringa senza caratteri speciali (le parentesi). A231M08 è un nome. Una stringa preceduta dallo slash è un nome inteso in senso letterale. Un nome indica una variabile o una procedura ed è il tramite con cui il PostScript ricerca una procedura e/o una variabile nei dizionari. Un nome preceduto dallo slash come /A231M08 dice che il valore cui è associato non è stato ancora interpretato.
Procedura	Una sequenza di elementi (operatori, operandi, variabili) chiusi da graffe.
Array	È una collezione eterogenea di elementi racchiusi da parentesi quadre.

Conversione di operatori

L'operatore `cvs` converte un qualsiasi oggetto in una stringa con un numero di caratteri assegnato.

Operatori del dizionario

Tale operatore è `def`.
`/Nome-chiave Valore def` crea la coppia <`Nome-chiave`, `Valore`> e la immette nel dizionario corrente. Se il codice è `/a 5 def`, nel dizionario il nome `a` è associato al numero 5.

Operatori di stato per grafici

Sono `gsave`, `grestore` e `setgray`.
`gsave` salva una copia dello stato corrente dei grafici.
`grestore` recupera lo stato corrente dei grafici.
`r setgray` stabilisce l'intensità del grigio; $0 \leq$ `r` ≤ 1. `r` $= 0$ equivale a nero.

Sistema di coordinate

Sono `rotate` e `translate`.
`a rotate` ruota il sistema di riferimento in senso antiorario di `a` gradi sessagesimali.
`a b translate` sposta l'origine del sistema di coordinate di `a` unità nella direzione dell'asse x e di `b` unità in quella dell'asse y.

Costruzione e tracciamento di percorsi

Gli operatori sono `newpath`, `currentpoint`, `setlinewidth`, `moveto`, `rmoveto`, `lineto`, `rlineto`, `arc`, `fill`, `stroke` e `newpath`.
`newpath` vuota il percorso corrente e si prepara ad accoglierne uno nuovo.
`currentpoint` restituisce le coordinate del punto corrente.
`r setlinewidth` assegna lo spessore `r` alla linea da tracciare.
`x y moveto` assegna le coordinate `x` e `y` al punto corrente.
`dx dy rmoveto` sposta le coordinate del vigente punto corrente di (`dx`, `dy`).
`x y lineto` collega con un segmento il punto corrente con il punto (`x`, `y`).
`dx dy lineto` collega con un segmento il punto corrente al punto spostato rispetto a questo di (`dx`, `dy`).
`x y r ai af arc` traccia un arco riferito ad una circonferenza di centro (`x`, `y`) e raggio `r`. Le coordinate angolari iniziale e finale sono `ai` e `af`.
`fill` riempie un percorso chiuso con il colore attuale e cancella il punto corrente. Poiché `fill` cancella il punto corrente per riutilizzare il percorso, occorre salvare il punto stesso con `gsave` prima di utilizzare `fill`, per poi recuperarlo con `grestore`.
`stroke` traccia il percorso; `showpage` stampa la pagina corrente.

Operazioni su caratteri e font

Detti operatori sono `findfont`, `scalefont`, `setfont`, `currentfont` e `stringwidth`.
`/Times-Roman findfont` restituisce il dizionario che descrive il font `/Times-Roman`.
`12 scalefont` porta il font `/Times-Roman` descritto dal dizionario nella dimensione 1 pt alla dimensione richiesta di 12 punti.
`setfont` trasforma il font richiesto nel font corrente.
`currentfont` attiva il dizionario del font corrente.

Operazioni sulle procedure

L'operatore `bind` cerca ogni nome eseguibile della procedura nel dizionario; se lo trova, e il suo valore è un operatore, sostituisce il nome con l'operatore vero e proprio: l'interprete non dovrà cercare i nomi nei dizionari ed eseguirà direttamente le operazioni. La ricerca nel dizionario è stata fatta in via preliminare da `bind`.

2.2 Esecuzione del codice

Un codice costituito da operazioni PostScript che rispetti le proprietà degli operatori e il giusto tipo di operando immesso nello stack degli operandi viene direttamente eseguito (Adobe System Inc., 1984, 1988). Dopo l'esecuzione il codice è tolto dallo stack e sostituito dal risultato, se previsto. È l'approccio più diretto, indicato come il migliore da Adobe System Inc. (1988). Se il codice deve essere utilizzato più volte o per evitare una sua eccessiva complessità, si ricorre a definizioni di procedure e variabili. L'operatore da utilizzare è `def`. Le espressioni `/S 5 def` e `/P{5 3 sub 4 mul}def` definiscono la variabile `/S` e la procedura `/P`. Nel primo caso il PostScript, con l'operatore `def`, salva nelle due posizioni più alte del dizionario corrente la coppia (*nome della variabile*, *valore della variabile*). Immette per primo il nome (*key*) `S` e per secondo, nella posizione più alta, il valore `5`. Nome e valore sono due oggetti semplici. Nel caso della procedura, nel dizionario corrente sono salvati la coppia data dal nome `P` e dal valore dato dall'array `{5 3 sub 4 mul}`. Questa volta gli oggetti formano una coppia composta in cui il nome rappresenta un puntatore all'array. Se il programma lancia, ad esempio, il nome `P`, questo verrà cercato nei dizionari. Trovato, la coppia (*nome*, *valore*) viene inserita nello stack degli operandi. La ricerca del nome nel dizionario e l'inserimento nello stack dei dati e degli operatori richiede un tempo che sconsiglia l'abuso delle procedure.

I dizionari, quello di sistema con tutti gli operatori predefiniti e quello dell'utente in cui sono memorizzate le variabili e le procedure di un programma, sono memorizzati in un apposito stack, con il dizionario dell'utente nella parte più alta. Il dizionario più alto è il dizionario corrente. Se

l'utente definisce la variabile `/S 5 def`, gli oggetti `S` e `5` formano la coppia più alta del dizionario corrente. La sua collocazione nel dizionario non è diretta, come vedremo ritornando al momento in cui lo scanner incontra la definizione della variabile: i tre elementi della definizione sono inviati uno dopo l'altro, a partire dal primo a sinistra, nello stack degli operandi. Il primo è il nome `/S` inteso in senso letterale (non interpretato), poi il valore e infine l'operatore `def`. Dallo stack degli operandi `def` interpreta `/S` come nome immettendolo, privato dello slash, nel dizionario corrente in coppia col suo valore e liberando lo stack degli operandi. In maniera analoga agisce il `def` della procedura.

Può essere interessante seguire il comportamento di una procedura un po' speciale (Glenn C. Reid, 1990) che, oltre a prevedere operazioni da eseguire, ha il compito di definire alcune variabili:

```
/Stampa{
  /Stringa exch def
  /Y exch def
  /X exch def
   X Y moveto Stringa show
       }def
300 -20 (Sono qui) Stampa
```

Nello stesso programma è presente una invocazione alla procedura che le passa tre parametri.

Nella fase di lettura lo scanner inizia con lo scorrere il codice della procedura immettendolo sotto forma di costrutti sintattici non interpretati nello stack degli operandi. Qui risulta operativo soltanto il `def` della procedura. L'operatore `def` immette la procedura nel dizionario corrente (svuotando lo stack degli operandi): il nome, `Stampa`, non è più inteso in senso letterale, ed è associato con l'array delle operazioni ancora non interpretate. Lo scanner prosegue e termina il proprio compito con la lettura dell'invocazione alla procedura, che porta a due conseguenze:

1. i tre parametri che passa a se stessa e il nome della procedura sono immessi nello stack degli operandi, a partire dal primo a sinistra. Seguendo la regola che l'ultimo elemento immesso è il primo ad operare, l'interprete inizia dal nome `Stampa`.

2. trovato nel dizionario il nome `Stampa`, prosegue l'immissione nello stack degli operandi degli elementi dell'array estratti dal dizionario (il valore associato al nome `Stampa`). Il primo frammento di procedura immesso nello stack è la definizione più in alto (o più a sinistra), di seguito le altre due definizioni e infine il codice che caratterizza il compito della procedura.

Lo stack, pensato con una disposizione orizzontale, ha nel primo elemento a sinistra il primo elemento immesso e si presenta così:

```
300 -20 (Sono qui)/Stringa exch def
/Y exch def/X exch def X Y moveto
Stringa show
```

Leggendo lo stack, come vuole la notazione postfissa, da destra a sinistra, si incontra il primo operatore esecutivo nello `exch` che precede il nome `/Stringa`. Può infatti intervenire su due operandi invertendo il loro ordine. Considerando soltanto questo frammento di codice, `exch` lo fa diventare `/Stringa (Sono qui) def`. È la modalità in cui `def` diventa esecutivo immettendo nel dizionario corrente la coppia formata dal nome (non più inteso in senso letterale) `Stringa` e dal valore `(Sono qui)`. La prima varibile è stata definita e lo stack diventa:

```
300 -20 /Y exch def /X exch def
X Y moveto
Stringa show
```

Ora è operativo il codice `-20 /Y exch def` che `exch` trasforma in `/Y -20 def` che definisce, immettendola nel dizionario, la seconda variabile con il nome `Y` e il valore `-20`. Allo stesso modo sarà definita l'ultima variabile dalla coppia <`X`, `300`>. Nello stack è rimasto il codice `X Y moveto Stringa show`. Ora le tre variabili sono definite, lo stack è esecutivo: il codice stamperà nella posizione richiesta il testo previsto (vedi Sez. 6).

L'esempio è stato considerato per aver il modo di seguire come si sviluppa l'azione di un codice PostScript. Pensando solo alla esecuzione più diretta il codice sarebbe stato:

```
300 -20 moveto (Sono qui) show
```

3 La pagina LaTeX e l'ospite PostScript

L'utilizzo del PostScript in un documento LaTeX (Goossens *et al.*, 2007) implica, tra le altre cose, il coordinamento degli output, che hanno necessità di apparire in continuità.

Se si tratta di un grafico, come nei casi della punteggiata e della curva archimedea, il modo più diretto è quello di utilizzare la macro `\code` all'interno di un comando `\pscustom` del pacchetto PSTricks.

Il comando `\pscustom` è predisposto ad interfacciare PSTricks, e quindi LaTeX, col PostScript. Ad esempio `\pscustom` trasforma l'unità di misura del PostScript, che è $\frac{1}{72}$ inch, direttamente in 1 pt di TeX, che è $\frac{1}{72.27}$ inch. Per TeX 1 pt del PostScript è un bigpoint (1 bp). Entro `\pscustom` la macro `\code` apre una finestra sul PostScript, e il grafico prodotto sarà del tutto inserito nel documento LaTeX. Ad esempio, nella sez. 4.1.3 il PostScript costruisce una circonferenza con il raggio di 42.5197 pt (bp), equivalenti a 1.5 cm. Ma l'output, che avviene in ambiente LaTeX, propone un raggio che pur rimanendo di 42.5197 pt

misura 1.4944 cm. Solo se è necessario che l'immagine rispetti le dimensioni in centimetri bisognerà moltiplicare i 42.5197 pt del PostScript per il fattore di conversione 1.00375, immettendo nel PS i 42.6791 pt (1.5056 cm) che in TEX saranno ridotti a 1.5 cm.

Con la macro `\pstverb` di PSTricks si inserisce una pagina PostScript nella pagina LATEX. L'origine della pagina PostScript è il punto corrente della pagina LATEX[1].

4 Curve mediante cicli

Partendo dal tracciamento di punti distribuiti su una circonferenza e da quello dei caratteri 'A' disposti lungo una spirale archimedea, si vuole mostrare come i cicli che disegnano la circonferenza e la spirale possono essere ottenuti in maniera diretta con la macro `\multido` del PSTricks o, in modo più consapevole e flessibile, con il ciclo `loop ... repeat` di TEX e con quello `n ... repeat` di PostScript.

Per disegnare la punteggiata, il ciclo `\multido` e quello standard di TEX ripetono il comando `\psdot`, quello PostScript effettua ripetute rotazioni del sistema di riferimento. `\psdot` usa coordinate polari, come conferma il ';' che separa i suoi parametri.

Un po' diverso è `\nput`, che costruisce le prime due versioni della spirale archimedea, e che può essere pensato come un'asta rigida di lunghezza variabile: una estremità è legata al nodo di riferimento, l'estremità libera incontra idealmente il carattere `A` nella sua metà più alta; la base di `A` è parallela alla retta dell'asta. `\cnode` crea il nodo attorno a cui `\nput` ruota e distanzia l'etichetta A.

4.1 Circonferenza punteggiata

Ad ogni variazione della coordinata angolo, crescente con incremento costante, il comando `\psdot` stampa un punto all'estremo del raggio, rotante e di lunghezza costante. Agendo sulle opzioni di pspicture — con `dotstyle=square,triangle,...` — il punto può essere sostituito da altri oggetti. Nella seconda versione della punteggiata, il ciclo `\loop` e i comandi `\ifnum` e `\advance` appartengono al codice TEX.

4.1.1 La punteggiata con `\multido`

Nella prima versione della punteggiata (esempio 5.14.1 Goossens *et al.* (2007)) la variazione dell'angolo e l'esecuzione del comando `\psdot` sono gestite dalla macro `\multido`. La `i` della variabile `\iAmp` prepara la macro a ricevere numeri interi.

Il seguente codice

1. Questo avviene nell'esempio della tabella in cui il punto corrente dato da `\vskip` è l'origine della pagina PostScript (vedi Sez. 6.2). Similmente, nel caso del calcolo del coefficiente angolare dell'asintoto dell'iperbole (vedi Sez. 5.3)

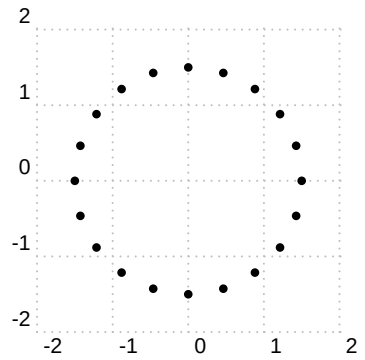

Figura 1: Circonferenza punteggiata ottenuta con `\multido`

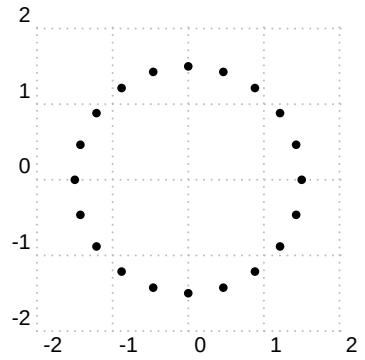

Figura 2: Circonferenza punteggiata ottenuta con un ciclo standard TEX

```
\begin{pspicture}[showgrid=true]
                 (-2,-2)(2,2)
  \multido{\iAmp=18+18}{20}
           {\psdot(1.5;\iAmp)}
\end{pspicture}
```

genera il disegno di figura 1.

4.1.2 La punteggiata e il codice TEX

Il ciclo `\loop ... \repeat` traccia 20 punti cambiando 19 volte la direzione del raggio con un passo di 18°. Il primo punto con il raggio nella direzione 18°, l'ultimo nella direzione 360°. Quest'ultimo è tracciato quando il contatore raggiunge il valore 342. Anche `\ifnum` e `\advance` sono codice TEX.

Di seguito il codice che genera il disegno di figura 2.

```
\begin{pspicture}[showgrid=true]
                    (-2,-2)(2,2)
  \newcount\cont \cont=0
   \loop
     \ifnum\cont<343 \advance
           \cont by 18
       \psdot(1;\the\cont)
   \repeat
\end{pspicture}
```

4.1.3 La punteggiata e il codice PostScript

Meno convenzionale è il caso del codice PostScript. Qui il punto, disegnato come un cerchio pieno di

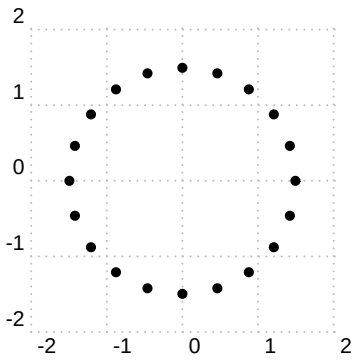

FIGURA 3: Circonferenza punteggiata ottenuta a partire dal codice PostScript

raggio 1.4 pt, è rappresentato 20 volte nella stessa posizione 42.5197, 0). I 42.5197 pt PostScript equivalgono a 1.5 cm, ma nell'ambiente LATEX che ospita la punteggiata misurano, come già detto, 1.4944 cm. Il comando per disegnare la circonferenza è `arc` e i suoi cinque argomenti sono $\mathrm{x}_c, \mathrm{y}_c, \mathrm{r}, \mathrm{ang}_i, \mathrm{ang}_f$. Il comando `rotate` di PostScript agisce sugli assi cartesiani. Sono loro a ruotare di 18° e facilitano le cose, almeno numericamente. Certo, se vivessimo sull'asse x vedremmo disegnare un solo punto. Ad evitare tracciamenti di segmenti il punto corrente coincide col centro del punto.

Il codice seguente

```
\begin{pspicture}[showgrid=true]
                    (-2,-2)(2,2)
 \pscustom{%
  \code{
        42.5197 0 moveto
        20{42.5197 0 1.5 0 360 arc
        gsave fill grestore
        18 rotate
        stroke} repeat
             }}
\end{pspicture}
```

fa ottenere il disegno di figura 3

4.2 Spirale archimedea

Nel caso della spirale archimedea disegnata con il carattere 'A' il punto di riferimento (centro della spirale) è un nodo tracciato come un cerchio pieno da `\cnode` che stabilisce: coordinate del centro, lunghezza del raggio e nome del nodo. Sia `\multido` che `loop ... repeat` spostano e ruotano insieme al raggio il carattere 'A', stampato disponendolo secondo una spirale archimedea con il comando `\nput` (coordinate polari). Detto comando prende come riferimento il nodo e assegna sia la direzione del raggio, variabile di una quantità costante ad ogni passo del ciclo, che la distanza della circonferenza perimetrale (r = 4 pt) dall'etichetta, vista come `labelsep` e crescente con uno step costante. Stabilito ciò, può stampare l'etichetta alla giusta

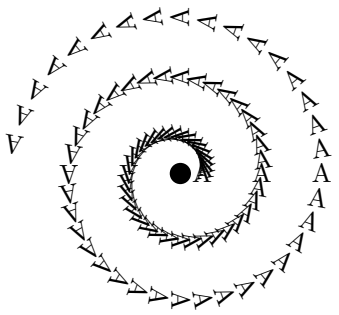

FIGURA 4: Spirale archimedea ottenuta col comando `\multido`

distanza e orientazione. La prima distanza tra il centro del nodo e il carattere 'A' è $(4 + 0.625)$ pt.

4.2.1 Spirale archimedea con `\multido`

`\multido`, che gestisce due variabili indipendenti, traccia facilmente la spirale (esempio 6.2.54 GOOSSENS *et al.* (2007)). In questo esempio la macro `\multido` esegue 90 volte il comando `\nput` e, con una rotazione `rot` che va da 0° a 900°, percorre due rotazioni complete più metà. Le due assegnazioni sono la rotazione `rot=\nA`, e la distanza `labelsep=\rB` pt che separa la circonferenza del nodo dalla etichetta. La prima variabile di `\multido`, `\nA`, accetta solo numeri interi, mentre la seconda variabile, `\rB`, accetta numeri reali; le due variabili sono indipendenti e, avendo entrambe un valore che varia in maniera costante, anche il loro rapporto lo è. Ed è sul valore 16 di tale rapporto che ci si baserà per tracciare con il codice TEX una seconda spirale uguale alla prima. È da notare che `\nput` utilizza due volte il valore assegnato da `\multido` alla variabile angolare `\nA`: la prima volta come l'angolo che assegna la rotazione dell'etichetta, la seconda volta come direzione del raggio. La prima `A` di `\nput` si riferisce al nodo, la seconda all'etichetta.

La figura 4 è ottenuta con il seguente codice:

```
\begin{pspicture}(5,3.5)
  \cnode*(3,1){4pt}{A}
  \multido{\nA=0+10,
           \rB=0+0.625}{90}{
    \nput[rot=\nA,
      labelsep=\rB pt]{\nA}{A}{A}}
\end{pspicture}
```

4.2.2 Spirale archimedea e il codice TEX

Il compito di `loop ... repeat`, che ha una sola vera variabile (intera), presenta una certa artificiosità ma nello stesso tempo è più flessibile di `\multido`. `\cont` è un contatore adeguato a rappresentare i valori interi degli angoli, ma non può rappresentare la lunghezza del raggio, che ha valori razionali. Per questo motivo il valore intero di `\cont` è passato sotto forma di distanza alla variabile razionale `\dimen2`, che può essere divi-

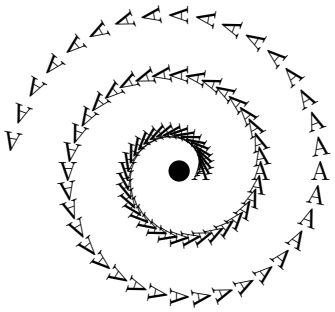

FIGURA 5: Spirale archimedea, identica alla prima, ma con diverso codice

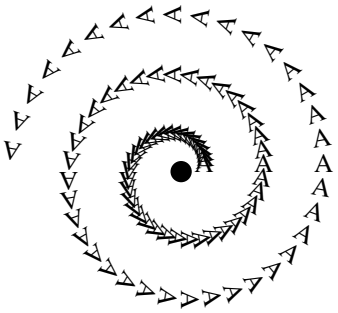

FIGURA 6: Spirale ottenuta con codice PostScript

sa. Ponendo a 10 il passo di `\cont` e dividendo `\dimen2` per 16 si ottiene che le variabili abbiano gli stessi valori del caso precedente e che le due spirali siano sovrapponibili.

Il codice TEX permette anche di mostrare come la rigidità del blocco {*raggio–box del carattere-etichetta*} presente nella spirale è costruita imponendo all'angolo di rotazione dell'etichetta lo stesso valore della rotazione del raggio. Se infatti si definisce un secondo contatore, lo si fa avanzare, ad esempio, con il passo 5 e si assegna il valore di questo contatore alla variabile angolare del raggio, si ottiene che il raggio ruota 1,25 angoli giro mentre l'etichetta effettua 2,5 rotazioni complete.

In figura 5 è mostrata la spirale ottenuta con il programma di seguito riportato:

```
\begin{pspicture}(6,3)
  \newcount\cont \cont=0
  \cnode*(3,0){4pt}{A}
  \loop
   \ifnum\cont<900
   \nput[rot=\the\cont,
      labelsep=\dimen2]
   {\the\cont}{A}{A}
   \advance
   \cont by 10\dimen2=\cont pt
   \divide\dimen2 by 16
  \repeat
\end{pspicture}
```

4.2.3 Spirale archimedea e il codice PostScript

Più semplice il tracciamento col PostScript. Anche in questo caso la rotazione del sistema di riferimento facilita le cose e crea una vera e propria unità del blocco {*raggio–carattere A*}. Qui il raggio è anche base per il carattere A, che risulta più alto rispetto ai due casi precedenti.

Basta pensare di tracciare 90 volte la stringa `(A)`, sull'asse x, a partire dalla posizione 5, di incrementare ad ogni passo la distanza dall'origine di 0.625 bp e di ruotare il sistema di riferimento di 10° in senso antiorario: 22.5 bp in una rotazione di 360°. Il compito è affidato alla procedura `/spiraledia`. Il ciclo `for` passa al corpo della procedura i valori dell'angolo di rotazione, a partire da 0 con incrementi di 10, senza superare il valore 890; il duplicato di 10, diviso per 16 e sommato a 5, dà l'ascissa del punto in cui stampare la stringa `(A)`. Qui non c'è il nodo di riferimento: il cerchio nell'origine è per uniformità grafica con le altre due versioni. La distanza costante tra due spire successive è 22.5 bp (0.8 cm). Il carattere /Times-Roman del PostScript non è uguale al Computer Modern Roman di LATEX.

Il seguente codice disegna la spirale di figura 6:

```
\begin{pspicture}[showgrid=false]
    (-3,-1)(2,2.8)
  \pscustom{\code{
   /Times-Roman findfont 10
     scalefont  setfont
   /spiraledia{0 10 899{dup gsave
     rotate 0 0 moveto 16 div
     5 add 0 rmoveto (A) show
     grestore}for}def
     spiraledia 0 0 moveto
     0 0 4 0 360 arc fill
     showpage }}
\end{pspicture}
```

5 Ellissi e iperboli

Rimanendo nell'ambito delle curve notissime, le ellissi e le iperboli permettono da un lato di toccare due fra le modalità con cui PSTricks traccia le curve e dall'altro di esaminare con una certa attenzione e gradualità il linguaggio PostScript per elaborare le coordinate dei punti della curva e predisporne l'output.

PSTricks traccia una curva in coordinate parametriche, $\big(x(t), y(t)\big)$, con la macro `\parametricplot`, e in coordinate polari, $\big(\alpha, r(\alpha)\big)$, con `\psplot`. In quest'ultimo caso la scelta si esplicita con l'assegnazione `polarplot=true` come argomento di `\psset` o di `\psplot`. La lingua per il calcolo è il PostScript.

5.1 Ellisse in coordinate parametriche

Le coordinate parametriche dell'ellisse ($a > b$) possono essere ottenute pensando di tagliare le circonferenze inscritta (raggio b) e circoscritta (raggio a) all'ellisse con una retta uscente dall'origine e

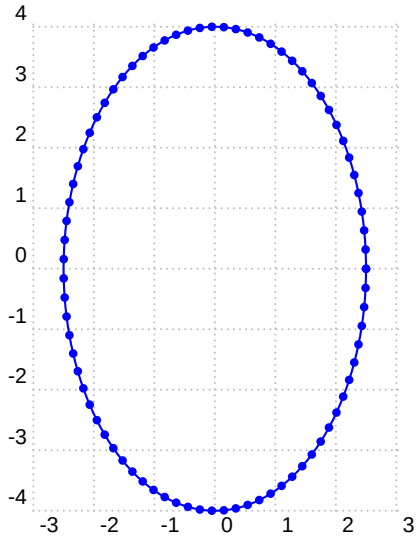

Figura 7: Ellisse in coordinate parametriche

formante un angolo t con l'asse x. L'ascissa dell'intersezione con la circonferenza esterna e l'ordinata di quella con la circonferenza interna si proiettano su uno stesso punto dell'ellisse il cui raggio forma un angolo t con l'asse x; le coordinate del punto, $(a\cos t, b\sin t)$, verificano l'equazione cartesiana dell'ellisse.

Dati $a = 2.5$ e $b = 4$, nell'ellisse in coordinate parametriche l'argomento di `\parametricplot` è `t cos 2.5 mul t sin 4 mul`, in cui è immediato riconoscere $(2.5\cos t, 4\sin t)$.

Con il seguente codice otteniamo l'ellisse in coordinate parametriche visibile in figura 7.

```
\def\Ellissey{t cos 2.5 mul
              t sin 4 mul}
\begin{pspicture}[showgrid=true]
                  (-3,-4)(3,4)
 \psset{linestyle=solid,unit=.8cm}
 \parametricplot[plotstyle=curve,
   plotpoints=80,showpoints=true,
   linecolor=blue]{0}{360}
                 {\Ellissey}
\end{pspicture}
```

5.2 Ellisse in coordinate polari

Per ricavare le coordinate polari $\big(\alpha, r(\alpha)\big)$ dell'ellisse si considera che la conica abbia il centro di simmetria nell'origine del sistema di riferimento. Le coordinate del punto del raggio vettore (modulo r e angolo α con l'asse x) sull'ellisse sono $(r\cos\alpha, r\sin\alpha)$. Sostituendo nell'equazione cartesiana dell'ellisse si ricava $r = \frac{ab}{\sqrt{b^2\cos^2\alpha + a^2\sin^2\alpha}}$. Il radicando è positivo $\forall\alpha$; la curva, se $a > b$, esiste ed è continua in $|\ x\ | \leq a$ (se $a < b \Rightarrow$ sarà continua in $|\ x\ | \leq b$) e non ha complicazioni di tracciamento. In `\psplot` l'equazione del raggio è espressa in codice PostScript; i semiassi `a` e `b` hanno i valori 4 e 2,5: il contrario di quanto fatto nell'ellisse in coordinate parametriche. L'esempio

Codice 1: Codice dell'ellisse in coordinate polari

```
1  \resetOptions
2  \psset{linestyle=solid,
3      unit=0.7cm,polarplot=true}
4  \begin{pspicture}(-4,-1)(4,3)
5    \psaxes[ticks=all]{->}(0,0)
6                    (-4.5,-3)(4.5,3)
7    \psplot[plotstyle=curve,
8      showpoints=false]{0}{360}
9     {/a 4 def /b 2.5 def a b mul
10     x cos dup mul b dup mul mul
11     x sin dup mul a dup mul mul
12     add sqrt  div}
13 \end{pspicture}
```

del tracciamento dell'ellisse in coordinate polari è utile per il confronto con quello più problematico della iperbole.

Nel codice dell'ellisse in coordinate polari l'angolo è espresso dalla variabile x, che sostituisce α, come vediamo nelle righe 9–12 del codice 1.

Interpretando il codice PostScript del raggio dell'ellisse, scritto secondo la Reversed Polish Notation (omologa al modo in cui i dati del PostScript sono salvati in uno stack, disposti in una pila in cui l'ultimo elemento salvato è il primo ad essere estratto), il primo operatore a destra è `div`, che agisce su un dividendo e un divisore, due numeri al momento non disponibili. La posizione che dovrebbe essere occupata dal divisore ospita l'operatore `sqrt`. E alla sua sinistra si incontra l'operatore `add`, che aspetta due addendi. Occorre individuarli scorrendo sempre a sinistra e tenendo presente che in questo contesto sono definiti gli operandi `a`, `b` e l'ampiezza `x`. Si incontra subito un `mul` che agisce su due fattori. Il primo è dato dal frammento di codice `a dup mul`, ossia a^2, il secondo fattore è `x sin dup mul`, ossia $\sin^2 x$. Considerando anche il primo `mul` incontrato, si ottiene il primo addendo che è $a^2\sin^2 x$. Analogamente si individua il secondo addendo che è $b^2\cos^2 x$. La somma dei due addendi, entrambi positivi, è il radicando. Volendo a questo punto fare una istantanea del contenuto dello stack si ottiene `a b mul` $(b^2\cos^2 x + a^2\sin^2 x)$

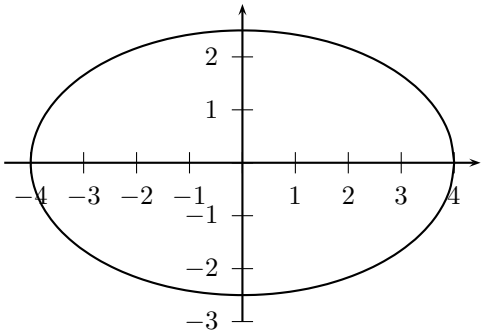

Figura 8: Ellisse in coordinate polari

`sqrt div` dove l'espressione algebrica tra parentesi tonde sta ad indicare il suo valore memorizzato in quella posizione dello stack. L'operatore `sqrt` ne estrae la radice quadrata, che è il divisore che si stava aspettando. Scorrendo il codice ancora a sinistra è facile individuare in `a b mul` il codice del dividendo, espresso nello stack dal valore di ab. Alla fine lo stack occupa una sola locazione con il valore dell'espressione $\frac{ab}{\sqrt{b^2\cos^2 x + a^2\sin^2 x}}$, cioè il risultato del calcolo per il valore assegnato da `\psplot` alla variabile angolo.

In figura 8 vediamo l'ellisse disegnata dal codice mostrato in codice 1.

5.3 Iperbole in coordinate polari

L'equazione del raggio polare dell'iperbole è $r = \frac{ab}{\sqrt{b^2\cos^2\alpha - a^2\sin^2\alpha}}$ e, poiché il radicando è una differenza, la curva non è continua, ma esiste per $\mid x \mid \geq a$ e α interno agli angoli tra gli asintoti contenenti l'asse x, e il suo tracciamento richiede attenzione. Infatti la macro `\psplot` che la traccia deve, in un certo senso, essere guidata dall'utente. Se si prova a far calcolare il raggio in $0 \leq \alpha \leq 360$, come per l'ellisse, o in intervalli non congrui, si creano incompatibilità che bloccano la compilazione. Occorre inoltre tracciare i due rami di iperbole separatamente assegnando due intervalli per la variabile α basati sulle direzioni degli asintoti. Per determinare tali direzioni evitando la calcolatrice ho usato un po' di codice PostScript immesso nel documento LaTeX con la macro `\pstverb` di PSTricks. Questa garantisce che l'output sia in una pagina PostScript collegata al punto corrente della pagina LaTeX, creato nell'esempio da `\vskip 2cm`, che diviene l'origine della pagina PostScript. Il codice PostScript calcola gli angoli tra gli asintoti e l'asse x e li stampa in un box accanto all'iperbole permettendo di scegliere gli intervalli per `\psplot`: più gli estremi degli intervalli sono vicini agli angoli fra gli asintoti e l'asse x più i rami di iperbole saranno lunghi. Poiché gli estremi degli archi nel cui intervallo calcolare il raggio polare e tracciare la curva sono alloctoni, è opportuno che `\psplot` effettui un controllo di positività sul radicando che qui, diversamente dall'ellisse, è dato da una differenza. Gli opposti coefficienti angolari degli asintoti obbligano al loro tracciamento separato portando a quattro il numero dei tracciamenti. Nei vari `\psplot` secondo richiesta vengono assegnati i parametri a e c dell'iperbole, definiti il parametro b e il radicando. `\psplot` riconosce solo variabili e procedure definite nel proprio ambiente: pur se identiche vanno ripetute ogni volta.

Il codice riportato di seguito, e commentato successivamente, si divide in due macroblocchi: da riga 2 a riga 35 è il blocco PostScript che effettua i calcoli e traccia il box dei risultati; da riga 38 a riga 81 invece è il codice PSTricks che disegna l'iperbole e gli asintoti. Detto codice genera la figura 9.

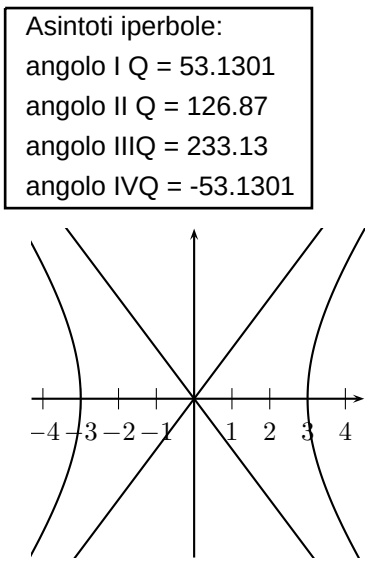

FIGURA 9: Iperbole in coordinate polari e codice PostScript

```
1  \pstverb{
2    /LM 10 def
3    /Helvetica findfont 10
4     scalefont setfont
5    /nstr 8 string def
6    /a 3 def
7    /b 4 def
8    /arctg1{b a atan}def
9    /arctg2{b a neg atan}def
10   /arctg4{b neg a atan 360 sub}def
11   /arctg3{b neg a neg atan}def
12   /prt-n{nstr cvs show}def
13   /prtArctg1{( angolo I
14                 Q = ) show
15      arctg1 prt-n }def
16   /prtArctg2{( angolo II
17                 Q  = ) show
18      arctg2 prt-n}def
19   /prtArctg3{( angolo III
20                 Q  = ) show
21      arctg3 prt-n}def
22   /prtArctg4{( angolo IV
23                 Q = ) show
24      arctg4 prt-n}def
25   LM  15 moveto
26   ( Asintoti iperbole:) show
27   LM    0 moveto prtArctg1
28   LM  -15 moveto prtArctg2
29   LM  -30 moveto prtArctg3
30   LM  -45 moveto prtArctg4
31   newpath 5 25 moveto 5 -50 lineto
32   120 -50 lineto 120 25 lineto
33   5 25 lineto 1 setlinewidth
34   stroke showpage
35 }
36 \vskip 2cm
37 \resetOptions
38 \psset{linestyle=solid,unit=.5cm,
39        polarplot=true}
```

```
\begin{pspicture*}(-4.3,-4.2)
                          (4.5,4.5)
\psaxes[ticks=x]{->}(0,0)
            (-4.3,-4.2)(4.5,4.5)
\psplot[plotstyle=curve]{-50}{50}
{
  /a 3 def /c 5 def
  /b{c dup mul a dup mul
     sub sqrt} def
  /rdcn{b dup mul x cos dup mul
     mul a dup mul x sin dup mul
     mul sub}def
  rdcn  0 gt{a b mul rdcn
             sqrt div} if
}
\psplot[plotstyle=curve]{127}
                           {230}
{
     /a 3 def /c 5 def /b{c dup
       mul a dup mul sub sqrt} def
     /rdcn{b dup mul x cos dup mul
       mul a dup mul x sin dup
 mul
       mul sub}def
   rdcn  0 gt{a b mul rdcn sqrt
                div}{} ifelse
}
\psset{polarplot=false}
\psplot[plotpoints=30]{-4}{4.5}
{
   /a 3 def /c 5 def
   /b{c dup mul a dup mul sub
      sqrt} def
    b a div x mul}
\psplot[plotpoints=30]{-4.1 }
                      {3.5 }
{
   /a 3 def /c 5 def
   /b{c dup mul a dup mul sub
      sqrt} def
    b a div x mul neg
}
\end{pspicture*}
```

Commenti al codice

Il codice del precedente paragrafo è stato ripulito dai commenti, che vengono qui ampliati. Per ogni intervallo di righe di codice si dà una spiegazione sommaria di quanto avviene.
Righe 2–35: Codice PostScript per calcolare gli angoli tra asintoti e asse x e tracciare il box con i risultati.
Righe 2–4: Margine a sinistra 10 pt e definizione del font.
Righe 5–7: Stringa vuota di 8 caratteri.
Righe 8–11: Calcola $\arctan b/a$ per i quadranti 1º, 2º, 4º e 3º.
Riga 12: Converte il numero in stringa e lo stampa.
Righe 13–24: Prepara le stringhe col valore dell'arctan.
Righe 25–30: Crea vari punti correnti per stampare i risultati.
Righe 31–35: Traccia il box dei risultati.
Righe 38–81: Codice PSTricks per tracciare l'iperbole in coordinate polari.
Righe 38–43: Ambiente pspicture e tracciamento assi.
Righe 44–54: Definizioni, positività radicando, tracciamento iperbole 1º e 4º quadrante.
Righe 55–65: Definizioni, positività radicando, tracciamento iperbole 2º e 3º quadrante.
Righe 66–81: Definizioni e tracciamento asintoti.

6 Tabelle calcolanti[2]

LATEX, i pacchetti in circolazione, e l'ambiente newenvironment sono in grado di costruire tabelle per ogni esigenza. Ma non prevedono alcuna reale autonomia circa la elaborazione di quantità numeriche presenti nelle tabelle. Il principale motivo per questa carenza è probabilmente dovuto al codice TEX i cui operatori aritmetici agiscono solo su contatori numerici interi. Effettua calcoli su contatori dimensionali con valori razionali restituendo una dimensione, anche nella divisione tra contatore dimensionale e numerico. Al contrario il PostScript prevede operatori che effettuano calcoli coi numeri reali ma non ha una gestione diretta delle tabelle; occorre definire procedure specifiche. Per mostrare i rispettivi comportamenti si considera il caso di una semplice tabella che rappresenti i punteggi ottenuti da tre gruppi in una ipotetica competizione e che sappia calcolare e rappresentare risultati e percentuali.

Nel primo caso il calcolo delle percentuali, espresse da numeri interi, richiede un elevato numero di contatori e operazioni. Con il PostScript il calcolo delle percentuali con due decimali significativi si ottiene con poche procedure mentre è impegnativa la costruzione delle tabelle che mostrano i dati.

6.1 Tabelle LATEX che calcolano con TEX

Il primo esempio è una tabella che espone i punteggi associati a tre gruppi; questa calcola le rispettive percentuali con i comandi TEX ed in particolare con `\divide` che opera la divisione restituendo il quoziente troncato.

Le percentuali che si ottengono troncando agli interi il quoziente delle divisioni tra i singoli punteggi moltiplicati per 100 e il punteggio complessivo (tolti i casi del dividendo multiplo intero del divisore) sono sempre inferiori al valore effettivo, fino ad una unità (0,99...). Ma il risultato della divisione può essere migliorato con l'utilizzo dei resti. Se il resto supera il duecentesimo del pun-

2. Il mio è un approccio empirico elementare, lontano e indipendente dal lavoro sapientemente fondato di Roberto Giacomelli, *Una tabella che fa i calcoli*, ArsTEXnica, ottobre 2008.

teggio complessivo (pari allo 0,5%) la percentuale assegnata aumenta di un punto, altrimenti rimane invariata. Le percentuali corrette, sempre espresse solo dagli interi, hanno ora una più equilibrata incertezza di ± 0,5%, anche se il centesimo della somma che viene utilizzato per correggere l'effetto del primo troncamento è anche esso troncato: nel caso dei Bianchi porta alla contraddizione di un resto superiore al centesimo della somma dei punti.

Guardando alla quantità di codice sembra un calcolo complicato, ma lo sembra per la numerosità dei contatori necessari. Occorre solo un po' di pazienza. Il tutto è ben compendiato dalla tabella.

```
\newcount\Rossi \newcount\Rcento
\newcount\Bianchi \newcount\Vcento
\newcount\Verdi \newcount\Rcento
\newcount\Rg \newcount\Vg
\newcount\Bg \newcount\sumcento
\newcount\sumduecento
\newcount\Rr \newcount\Vr
\newcount\Rv \newcount\Br
\newcount\Rb \newcount\Rb
\newcount\sum \newcount\sumcentoB
\newcount\sumcentoR
\newcount\sumcentoV
\Rossi=18769
\Verdi=9876
\Bianchi=14728
\Rcento = \Rossi
\Vcento = \Verdi
\Bcento = \Bianchi
\Rr     = \Rossi
\Vr     = \Verdi
\Br     = \Bianchi
\multiply\Rcento by 100
%     (-> \Rcento = 1876900)
\multiply\Vcento by 100
%     (-> \Vcento =  987600)
\multiply\Bcento by 100
%     (-> \Bcento = 1472800)
\sum=0
  \advance\sum by \Rossi
  \advance\sum by \Verdi
  \advance\sum by \Bianchi
%     (-> \sum = 43373)
\sumduecento=\sum
  \divide\sumduecento by 200
%     (-> \sumduecento = 216)
\sumcento=\sum
  \divide\sumcento by 100
%     (-> \sumcento = 433)
  \sumcentoR=\sumcento
  \sumcentoV=\sumcento
  \sumcentoB=\sumcento
  \divide\Rcento by \sum
%     (-> \Rcento = 43)
  \divide\Vcento by \sum
%     (-> \Vcento = 22)
  \divide\Bcento by \sum
%      (-> \Bcento = 33)
  \Rg=\Rcento \Vg=\Vcento
  \Bg=\Bcento
 \multiply\sumcentoR by -\Rcento
%    (-> \sumcentoR = -18619)
 \advance\Rr by \sumcentoR
%    (-> \Rr = 150)
 \ifnum \Rr>\sumduecento %(->
false )
   \advance\Rcento by 1\fi
%    (-> \Rcento = 43)
 \multiply\sumcentoV by -\Vcento
%    (-> \sumcentoV =  -9526)
 \advance\Vr by \sumcentoV
%    (-> \Vr = 350)
 \ifnum \Vr>\sumduecento  %(->
true )
   \advance\Vcento by 1\fi
%    (-> \Vcento = 23)
 \multiply\sumcentoB by -\Bcento
%    (-> \sumcentoB =  -14289)
 \advance\Br by \sumcentoB
%    (->\Br = 439)
 \ifnum \Br>\sumduecento  %(->
true )
   \advance\Bcento by 1\fi
%   (-> \Bcento = 34)
```

TABELLA 2: Tabella con le percentuali calcolate con TEX

Gruppo	Pnt	% gr	$\frac{\Sigma}{200}$	Rst	% crt
Rossi	18769	43%	216	150	43%
Verdi	9876	22%	216	350	23%
Bianchi	14728	33%	216	439	34%

Commenti al codice

Righe 1–6: Contatori per il calcolo della percentuale grezza.
Righe 7–12: Contatori per il calcolo della percentuale corretta.
Righe 13–15: Dati.
Righe 16–38: Calcolo percentuale grezza (le righe commentate — % — indicano un risultato parziale).
Righe 39–49: Percentuali grezze espresse da `\Rg \Vg \Bg`.
Righe 50–70: Calcolo percentuali corrette espresse da `\Rcento \Vcento \Bcento`; i resti sono espressi da `\Rr \Vr \Br`. Nella tabella 2 è visibile il risultato del codice appena analizzato, ottenuta con il sorgente nel codice 2.

6.2 Il PostScript che calcola e costruisce tabelle

In questa sezione si è scelto di immettere tutto il codice nell'ambiente creato dal comando `\pstverb`. Questo crea una pagina PostScript in cui immettere l'output elaborato collegandolo alla pagina

Codice 2: Codice della tabella con valori calcolati con TEX

```
\begin{tabular}{lrrcrr}
 prcnt = \%
\toprule
 Gruppo&Punti& prc gr&$\frac{S}
           {200}$&Rst &prc crrt\\
\midrule
 Rossi&\the\Rossi&\the\Rg prcnt&
     \the\sumduecento &
     \the\Rr&\the\Rcento prcnt\\
 Verdi&\the\Verdi&\the\Vg prcnt&
     \the\sumduecento &
     \the\Vr&\the\Vcento prcnt\\
 Bianchi&\the\Bianchi&\the\
      Bg prcnt&\the\sumduecento &
     \the\Br&\the\Bcento prcnt\\
\bottomrule
\end{tabular}
```

LATEX. L'origine della pagina PostScript è il punto corrente della pagina LATEX. Nel caso specifico quello determinato da `\vglue 170pt`.

Dato che si tratta di un esempio dimostrativo il numero delle righe e colonne è minimo. La loro numerosità non è però un problema.

La tabella con le percentuali a due decimali — i centesimi incerti e arrotondati — mostra i risultati del calcolo che si risolve con poche procedure. Una per sommare i punteggi ed una per ogni gruppo (riga) della tabella per calcolare le percentuali. Nella seconda definizione (righe 16–21 a lato), si nota che il punteggio del singolo gruppo è moltiplicato per 10000 prima di essere diviso per il punteggio complessivo. Se volessimo calcolare una percentuale dovremmo moltiplicare il punteggio parziale per 100 e poi dividere per il punteggio complessivo. Si avrebbe un numero con molti decimali di cui solo due devono rimanere dopo aver approssimato i centesimi. Gli strumenti di calcolo del PostScript hanno l'operatore di arrotondamento, `round`, che agisce approssimando gli interi ed eliminando tutte le cifre decimali. Se nel frattempo anziché per 100 si è moltiplicato il punteggio parziale per 10000, `round` non avrà approssimato le unità ma i centesimi in quanto per riportare le cose al loro posto si dovrà dividere il quoziente ottenuto (un intero) per 100. Il codice per le variabili e le procedure di calcolo è quello riportato nelle righe 12–21 del codice 3.

I risultati ottenuti sono mostrati in due tabelle (entrambe in tabella 3) che allineano in maniera diversa i dati delle colonne: nella prima tutte le colonne sono imbandierate a sinistra, nella seconda la prima colonna è imbandierata a sinistra, le altre a destra. Le intestazioni della prima tabella sono centrate, quelle della seconda sono allineate con i dati. In codice 3 troviamo l'intero codice usato, che commenteremo nei successivi paragrafi.

Codice 3: Codice della tabella con valori calcolati con PostScript

```
\vglue 170pt
\pstverb
{
/grassetto/Helvetica-bold findfont
  10 scalefont def
/Helv/Helvetica findfont 10
  scalefont def
/helv/Helvetica findfont 30
  scalefont def
/nstr 8 string def
/prt-n{nstr cvs show}def
/lstA 127856 def
/lstB 6473 def
/lstC 48265 def
/somma {lstA lstB add lstC add}def
/percA{lstA 10000 mul somma div
       round 100 div}def
/percB{lstB 10000 mul somma div
       round 100 div}def
/percC{lstC 10000 mul somma div
       round 100 div}def
/intestazionegCentro{
   grassetto setfont (Gruppo)
   dup dup stringwidth  pop
   50 exch sub 2 div 0 rmoveto
   show stringwidth pop 50 exch
   sub 2 div 0 rmoveto
   (Punti) dup dup stringwidth
pop
   50 exch sub 2 div 0 rmoveto
   show stringwidth pop 50 exch
   sub 2 div 0 rmoveto
   (Percnt) dup dup stringwidth
   pop 50 exch sub 2 div 0
   rmoveto show stringwidth
   pop 50 exch sub 2 div 0
   rmoveto}def
%/intestazionegCCC
%{grassetto setfont
% /diff1{(Gruppo) stringwidth pop
%        50 exch sub 2 div}def
% /diff2{(Punti)  stringwidth pop
%        50 exch sub 2 div}def
% /diff3{(Percnt) stringwidth pop
%        50 exch sub 2 div}def
% diff1 0 rmoveto  (Gruppo) show
diff1 0 rmoveto
% diff2 0 rmoveto  (Punti)  show
diff2 0 rmoveto
% diff3 0 rmoveto  (Percnt) show
diff3 0 rmoveto
%}def
/intestazionegSSS{
 grassetto setfont
 (Gruppo) dup show stringwidth pop
 50 exch sub 0 rmoveto (Punti)
```

```
 dup show stringwidth pop 50 exch
 sub 0 rmoveto (Percnt) show}def
/rigaAgs{Helv setfont (Listone A)
 dup show stringwidth pop 50 exch
 sub 0 rmoveto lstA nstr cvs dup
 show stringwidth pop 50 exch sub
 0 rmoveto percA nstr
 cvs show}def
/rigaBgs{Helv setfont (Lista B)
 dup show stringwidth pop 50 exch
 sub 0 rmoveto lstB nstr cvs dup
 show stringwidth pop 50 exch sub
 0 rmoveto percB nstr cvs show}def
/rigaCgs{Helv setfont (Lista C)
 dup show stringwidth pop 50 exch
 sub 0 rmoveto lstC nstr cvs dup
 show stringwidth pop 50 exch sub
 0 rmoveto percC nstr cvs show}def
0 146 moveto intestazionegCCC
0 130 moveto rigaAgs
0 115 moveto rigaBgs
0 100 moveto rigaCgs
newpath -6 156 moveto 156 156
lineto  0.8 setlinewidth  stroke
newpath -4 141 moveto 154 141
lineto  0.6 setlinewidth  stroke
newpath -6  95 moveto 156  95
lineto  0.8 setlinewidth  stroke
showpage

/intestazionegSDD{
   grassetto setfont
   (Gruppo) dup show stringwidth
   pop 50 exch sub 0 rmoveto
   (Punti) dup stringwidth pop
   50 exch sub  0 rmoveto show
   (Percnt)dup stringwidth pop
   50 exch sub  0 rmoveto show
   }def
/rigaAgSDD{
   Helv setfont (Listone A)
   dup show stringwidth pop 50
   exch sub 0 rmoveto lstA nstr
   cvs dup stringwidth pop 50
   exch sub  0 rmoveto show percA
   nstr cvs dup stringwidth pop
   50 exch sub  0 rmoveto show
   }def
/rigaBgSDD{
   Helv setfont (Lista B)
   dup show stringwidth pop 50
   exch sub 0 rmoveto lstB nstr
   cvs dup stringwidth pop 50
   exch sub  0 rmoveto show percB
   nstr cvs dup stringwidth pop
   50 exch sub  0 rmoveto show
   }def
/rigaCgSDD{
   Helv setfont (Lista C)
   dup show stringwidth pop 50
   exch sub 0 rmoveto lstC nstr
   cvs dup stringwidth pop 50
   exch sub  0 rmoveto show percC
   nstr cvs dup stringwidth pop 50
   exch sub  0 rmoveto show
   }def
0 60 moveto intestazionegSDD
0 45 moveto rigaAgSDD
0 30 moveto rigaBgSDD
0 15 moveto rigaCgSDD
%newpath -6 70 moveto 156 70
%lineto 156 9 lineto -6 9 lineto
%closepath .8 setlinewidth
%stroke
%newpath -5 55 moveto  155 55
%lineto  0.6 setlinewidth  stroke
%newpath 50 69 moveto 50 8
%lineto stroke
%newpath 105 69 moveto 105 8
%lineto stroke
newpath -6 70 moveto 156 70
lineto  0.8 setlinewidth
stroke
newpath -4 55 moveto 154 55
lineto 0.6 setlinewidth stroke
newpath -6 9 moveto 156 9
lineto  0.8 setlinewidth stroke
showpage
}
```

Prima tabella

Le righe 4–21 contengono le definizioni preliminari, i dati e i relativi calcoli.

Considerato che le definizioni della riga delle intestazioni e dei dati non assegnano il punto corrente iniziale e che utilizzano, con `rmoveto`, gli spostamenti relativi, il codice che le invoca per stamparle può assegnare il punto corrente a una qualsiasi posizione della pagina.

L'operatore `stringwidth` calcola e mette nello stack gli spostamenti relativi a cui sarebbe sottoposto il punto corrente dalla stampa della stringa considerata, tenendo conto anche delle caratteristiche e delle dimensioni del font utilizzato: nella posizione più alta dello stack lo spostamento verticale, che è zero e che viene eliminato dal successivo `pop`, e subito sotto lo spostamento orizzontale. Nel fare queste operazioni viene cancellata la stringa a cui l'operatore `stringwidth` si è riferito.

Nelle righe 22–36 si trova il codice per formattare i titoli delle colonne, centrati. Per centrare una stringa nella dimensione di una colonna occorre calcolare la differenza tra le ampiezze della colonna (50) e della stringa (`stringwidth pop`) e assegnarne una metà (`stringwidth pop 50 exch sub 2 div`) all'ascissa di `rmoveto` per determinare il punto da cui iniziare la stampa della stringa e l'altra metà per spostare, con `rmoveto`, il punto

corrente all'inizio della seconda colonna. In questo fare si utilizza tre volte la stringa: una stampa e due `stringwidth` e questa tripla disponibilità si ottiene una volta dalla stringa stessa, le altre dalla doppia duplicazione della stringa `(Gruppo)`.

Nelle righe 37–48 è presente un codice simile, che usa le procedure. Il loro uso chiarisce il flusso dei dati, ma rallenta l'esecuzione. Da riga 49 a riga 70 possiamo vedere l'imbandieramento a sinistra sia dei titoli (49–60) che dei dati(61–70).

Per seguire la dinamica operativa del PostScript (sempre in relazione alle righe 22 - 36 dell'intestazione centrata) bisogna guardare al codice della prima colonna come ad un codice inserito nello stack degli operandi, dall'alto verso il basso, o più comodamente da destra verso sinistra. Nella prima lettura a partire da destra si nota che per trovare un operatore esecutivo è necessario scorrere fino al `dup` situato alla destra della stringa `(Gruppo)`, di cui fa una copia. Le ultime tre posizioni ora sono `(Gruppo) (Gruppo) dup`: è operativo `dup` che crea una copia della seconda stringa. Le ultime cinque posizioni dello stack diventano: `(Gruppo)(Gruppo)(Gruppo) stringwidth pop`. L'operatore `stringwidth` memorizza l'incremento che avrebbero le coordinate del punto corrente dalla stampa della terza stringa, che toglie dallo stack (l'incremento dell'ascissa è ovviamente l'ampiezza della stringa); `pop` estrae l'ordinata del punto corrente. A questo punto le ultime posizioni dello stack sono `(Gruppo)(Gruppo) stringwidth 50 exch sub 2 div`; `exch` può scambiare i suoi operandi, `sub` sottrarli e `2 div` calcolare la metà della differenza, che è l'ascissa di `rmoveto` che, per comodità, è riportato nel codice tra virgolette. Andando ancora a guardare alla parte bassa dello stack il codice, che è `(Gruppo)(Gruppo) "ascissa" rmoveto 0 show`, crea il nuovo punto corrente da cui iniziare la stampa della stringa che gli è contigua, eliminandola dallo stack. Che nella parte più bassa ora è `(Gruppo) stringwidth pop 50 exch sub 2 div 0 rmoveto` e ha il compito di portare il punto corrente all'inizio della seconda colonna. Lo fa aggiungendo all'ascissa del punto corrente, cioè in coda alla stringa, la seconda metà della differenza di cui si è detto. Nella parte bassa dello stack è ora presente il codice della seconda colonna, uguale in tutto a quello della prima tranne che per la stringa che è diversa. Lo stesso per la terza colonna.

Da riga 55 a riga 70 troviamo il codice che predispone l'imbandieramento a sinistra dei dati della prima tabella e da riga 71 a riga 81 il codice per stampare i titoli centrati, i dati imbandierati a sinistra e le tre filettature orizzontali.

Di seguito si commenta la dinamica operativa delle righe 55–60 riguardanti l'imbandieramento a sinistra della prima riga di dati. Si sceglie il font da utilizzare, si immette la prima stringa `(Listone A)`, si duplica per utilizzarla due volte (`dup show stringwidth`): per stamparla (`dup show`) e misurarne la lunghezza con `stringwidth pop` che estrae la lunghezza della stringa. Si immette la larghezza della prima colonna (50) e poiché si deve spostare il punto corrente verso destra, nell'ascissa in cui inizia la seconda colonna, lo spazio rimasto alla destra della stringa dovrà essere espresso da un numero positivo: per questo si scambiano i ruoli del minuendo e del sottraendo con `exch` e si assegna a questa differenza il ruolo di spostamento relativo (`'dx' 0 rmoveto`). Lo 0 per l'ordinata ci dice che questa non cambia valore. Ora il punto corrente è l'ascissa del punto zero della seconda colonna. La seconda immissione della riga, `lstA`, è un numero e dovrà essere trasformato in stringa che, per i motivi detti, sarà duplicata e stampata iniziando dalla posizione attuale. La trasformazione in stringa avviene con la procedura `nstr` e l'operatore `cvs`. La prima, `nstr`, usa l'operatore `string` per creare una stringa vuota di una determinata lunghezza, nello specifico di otto caratteri. Il secondo operatore, `cvs`, trasforma il numero in una stringa al massimo di otto caratteri. Ora, come già fatto nella prima colonna, si calcola la lunghezza della nuova stringa (`stringwidth`), si elimina l'ordinata, si immette la larghezza della seconda colonna (50) e si calcola lo spazio rimasto alla destra della stringa per poter portare il nuovo punto corrente allo zero della terza colonna (`stringwidth pop 50 exch sub 0 rmoveto`). Lo stesso per la terza immissione (`percA`).

Tabella 3: Le due tabelle generate interamente col PostScript

Gruppo	Punti	Percnt
Listone A	127856	70.02
Lista B	6473	3.55
Lista C	48265	26.43

Gruppo	Punti	Percnt
Listone A	127856	70.02
Lista B	6473	3.55
Lista C	48265	26.43

Seconda tabella

Il codice per questa tabella, da riga 83 a a 141, funziona allo stesso modo, con l'evidente differenza di imbandieramento. La parte commentata traccerebbe una filettatura per il contorno e la linea orizzontale interna.

Per la prima colonna (giustificata a sinistra) vale quanto detto per tabella precedente e porta il punto corrente allo zero della seconda colonna. La seconda immissione (giustificata a destra) è un numero

che dovrà essere trasformato in stringa e duplicato, se ne misurano le dimensioni, si elimina l'ordinata dalla sommità dello stack, si immette la larghezza della seconda colonna, si sottrae dalla larghezza della colonna la larghezza della stringa e si fa di questo valore il punto corrente per iniziare la stampa della stringa, che andrà a terminare nel punto zero della terza colonna facendone il nuovo punto corrente (`lstA nstr cvs dup stringwidth pop 50 exch sub 0 rmoveto show`). La stessa cosa per la terza colonna.

7 Procedura che si autodefinisce e l'operatore bind

Il filo del mio discorso, che potrebbe ritenersi completato con l'*excursus* sulle tabelle, ha invece lasciato in sospeso l'argomento della procedura con due livelli di definizione, che non è stato verificato. Inoltre non volevo chiudere senza neanche accennare a quell'operatore `bind` presente almeno nella metà delle pagine di Adobe System Inc. (1988). Sarebbe stato come non lasciare neanche un esile *legame* con il PostScrpt.

Diciamo che riprendere quell'esempio e completarlo con l'operatore `bind` è quel legame.

Non ritornerò sulla procedura di cui ho già detto molte cose. Accennerò rapidamente soltanto al comportamento del nuovo operatore la cui esecutività consiste nell'andare a cercare nel dizionario i nomi presenti nella procedura. Se il nome cercato è nel dizionario, `bind` interviene sulla procedura legando il nome al corrispondente oggetto operatore. Questo ricorsivamente per ogni nome presente nella procedura, anche all'interno di un array, o in presenza di incapsulamenti. Se il nome non è nel dizionario o non è di un operatore, `bind` non agisce. Quando la procedura sarà eseguita, l'interprete troverà operatori di cui conosce le modalità di intervento e le azioni che sono in grado di compiere (definite nel dizionario e che, in coppia con il nome, costituiscono il valore dell'operatore), e l'esecuzione sarà immediata.

Ad esempio, il seguente codice è esemplificativo di quanto detto. Il risultato è visibile immediatamente dopo.

```
/helv/Helvetica findfont 30
  scalefont def
/F{
   /Stringa exch def
   /Y exch def
   /X exch def
  X Y moveto  Stringa show
   }bind def
 helv setfont
 0 -50 (?<- Sono qui) F
```

?<- Sono qui

8 Ringraziamenti

Per me si è trattato di una prima volta e non avrei potuto organizzare alcunché senza i riferimenti e la disponibilità del *The TEXBook* di Donald Knuth, il *LATEX* di Claudio Beccari, il *TEX* di Gianni Gilardi, *The LATEX Companion* di Frank Mittelbach ed altri, il *Guide to LATEX* di Helmuth Kopka, senza il *Tutorial and Cookbook* di Adobe e senza gli *Appunti di Programmazione* in LaTeX e TeX di Enrico Gregorio.

Riconosco anche che senza la disponibilità — non dichiarata ma espressa dalla qualità e apertura dei rapporti — di Massimiliano Dominici e di Gianluca Pignalberi non mi sarei così impegnato nel cercare di portare avanti il mio lavoro. Innanzi tutto ringrazio il mio anonimo Editor per aver saputo vedere elementi di validità nel mio scritto. Un ulteriore ringraziamento voglio fare a Gianluca Pignalberi e all'anonimo revisore per essere riusciti, con un gran lavoro e occhio di lince, con suggerimenti e anche il recupero di un paio di errori, a dare al mio scritto, che era nato per essere letto dagli amici, la forma (spero sia insieme a un po' di sostanza) di articolo.

Riferimenti bibliografici

Adobe System Inc. (a cura di) (1984). *PostScript Language, Tutorial and Cookbook*. Addison-Wesley, Reading, MA, USA. The BlueBook.

— (1988). *Postscript language program design*. Addison-Wesley, Reading, MA, USA. The GreenBook.

— (1991). *PostScript Language, Reference Manual*. Addison-Wesley, Reading, MA, USA. The RedBook.

Glenn C. Reid (1990). *Thinking in PostScrpt*. Addison-Wesley, Reading, MA, USA.

Goossens, M., Mittelbach, F., Rahtz, S., Roegel, D. e Voss, H. (2007). *The LATEX Graphics Companion Second Edition*. Addison-Wesley, Reading, MA, USA.

Utting, I. (1992). «Postscript tutorial and reference». Technical report. URL `http://www.cs.kent.ac.uk/pubs/1992/109`.

▷ Riccardo Nisi
`r underscore nisi at tin dot it`