GUIT

TEXnica ArS

Rivista italiana
di TEX, LATEX e
tipografia digitale

34 · 2023

Editoriale

Come scrivere un libro e sopravvivere

Semplificare LATEX con *ORG-mode* in Emacs

CJK Unicode strokes e radicals con MFLUA: uno studio preliminare

Mapping to individual characters in expl3

TeXShop, Version 5 (August 11, 2022)

Interactive content using TEX4ht

TEXnica Ars

GUIT – Gruppo Utilizzatori Italiani di TEX

ArsTEXnica è la pubblicazione ufficiale del GUIT

ArsTEXnica è la prima rivista italiana dedicata a TEX, a LATEX e alla tipografia digitale. Lo scopo che la rivista si prefigge è quello di diventare uno dei principali canali italiani di diffusione di informazioni e conoscenze sul programma ideato da Donald Knuth.

Le uscite hanno cadenza semestrale e sono pubblicate nei mesi di Aprile e Ottobre. In particolare, la seconda uscita dell'anno contiene gli Atti del Convegno Annuale (GUIT*meeting*).

Chiunque volesse collaborare con la rivista a qualsiasi titolo (recensore, revisore di bozze, grafico, etc.) può contattare la redazione all'indirizzo:

arstexnica@guitex.org.

Pubblicare su ArsTEXnica

La rivista è aperta al contributo di tutti coloro che vogliano partecipare con un proprio articolo. Questo dovrà essere inviato alla redazione di ArsTEXnica, per essere sottoposto alla valutazione di recensori. È necessario che gli autori utilizzino la classe di documento ufficiale della rivista; l'autore troverà raccomandazioni e istruzioni più dettagliate all'interno del file di esempio (`.tex`). Tutto il materiale è reperibile all'indirizzo web della rivista.

Gli articoli potranno trattare di qualsiasi argomento inerente al mondo di TEX e LATEX e non dovranno necessariamente essere indirizzati ad un pubblico esperto. In particolare tutorials, rassegne e analisi comparate di pacchetti di uso comune, studi di applicazioni reali, saranno bene accetti, così come articoli riguardanti l'interazione con altre tecnologie correlate.

Di volta in volta verrà fissato, e reso pubblico sulla pagina web, un termine di scadenza per la presentazione degli articoli da pubblicare nel numero in preparazione della rivista. Tuttavia gli articoli potranno essere inviati in qualsiasi momento e troveranno collocazione, eventualmente, nei numeri seguenti.

Nota sul Copyright

Associarsi a GUIT

Fornire il tuo contributo a quest'iniziativa come membro, e non solo come semplice utente, è un presupposto fondamentale per aiutare la diffusione di TEX e LATEX anche nel nostro paese. L'adesione al Gruppo prevede una quota di iscrizione annuale diversificata: 30,00 € soci ordinari, 20,00 (12,00) € studenti (junior), 75,00 € Enti e Istituzioni.

Indirizzi

Gruppo Utilizzatori Italiani di TEX
c/o Università degli Studi di Napoli Federico II
Dipartimento di Ingegneria Industriale
Via Claudio 21
80125 Napoli – Italia
`http://www.guitex.org`
guit@guitex.org

Redazione ArsTEXnica:
`http://www.guitex.org/arstexnica/`
arstexnica@guitex.org

ISSN 1828-2350 (Stampa)
ISSN 1828-2369 (Online)

15 Aprile 2023

GUIT meeting 2023

DICIANNOVESIMO CONVEGNO NAZIONALE
SU TEX, LATEX E TIPOGRAFIA DIGITALE

20 maggio 2023

Università di Roma Tor Vergata

Comitato organizzatore

Guido Milanese · Dipartimento di Scienze Storiche e Filologiche dell'Università Cattolica del Sacro Cuore; Grazia Messineo e Salvatore Vassallo · Dipartimento di Matematica per le scienze economiche, finanziarie ed attuariali dell'Università Cattolica del Sacro Cuore; Claudio Beccari, Massimiliano Dominici, Roberto Giacomelli, Gianluca Pignalberi e Luigi Scarso · Gruppo Utilizzatori Italiani di TEX.

Programma del convegno

TEX e la tipografia digitale in ambito umanistico: tra innovazione e durabilità delle tecnologie tipografiche.

Sessione mattutina

9:30 Inizio lavori sessione mattutina. Moderatore: Claudio Beccari.
9:40 Saluto del Professor Lorenzo Perilli, direttore del Dipartimento di Studi Letterari, Filosofici e di Storia dell'Arte dell'Università degli studi di Roma Tor Vergata.
10:00 *Come scrivere un libro e sopravvivere*. Grazia Messineo e Salvatore Vassallo.
10:45 *Semplificare LATEX con Org-Mode in Emacs*. Emmanuele Somma.
11:30 *CJK Unicode strokes e radicals con MFLUA: uno studio preliminare*. Luigi Scarso.
12:30 — Pausa pranzo (1 h 30 min).

Sessione pomeridiana

14:00 Inizio lavori sessione pomeridiana. Moderatore: Guido Milanese. Presenta gli interventi Claudio Beccari.
14:00 *Mapping individual characters in* expl3. Joseph Wright.
14:30 *TeXShop, Version 5 (August 11, 2022)*. Richard Koch.
15:00 *Interactive content using TEX4ht*. Richard Koch.

15:30 Riunione annuale del gruppo.
17:00 Chiusura dei lavori.

Evento organizzato dal Gruppo Utilizzatori Italiani di TEX in collaborazione con il Dipartimento di Studi Letterari, Filosofici e di Storia dell'Arte dell'Università degli studi di Roma Tor Vergata e con i Dipartimenti di Matematica per le scienze economiche, finanziarie ed attuariali e di Scienze Storiche e Filologiche dell'Università Cattolica del Sacro Cuore.

Note per i docenti: le conferenze rientrano nelle iniziative di formazione e aggiornamento dei docenti realizzate dalle Università automaticamente riconosciute dall'Amministrazione scolastica, secondo la normativa vigente (Direttiva n. 170 del 21/03/2016), e danno luogo — per gli insegnanti di ogni ordine e grado — agli effetti giuridici ed economici della partecipazione alle iniziative di formazione.

Note per gli studenti: le conferenze rientrano nelle tipologie di esperienze che danno luogo ai crediti formativi riconoscibili per l'esame di Stato (conclusivo del II ciclo di studi) come recita il D.M. 49 del 25.02.2000, nonché ad eventuali crediti formativi universitari.

I certificati di partecipazione saranno rilasciati direttamente dal Dipartimento di Scienze Storiche e Filologiche e inviati via mail ai singoli partecipanti. Le scuole di appartenenza degli insegnanti riconosceranno internamente i crediti.

La partecipazione è libera e gratuita, previa registrazione entro il 13 maggio.

Registrazione online: `www.guitex.org/home/meeting`

www.guitex.org

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

UNIVERSITÀ CATTOLICA del Sacro Cuore

ArsTEXnica

Rivista italiana di TEX e LATEX

Numero 34, Aprile 2023

5 **Editoriale**
Claudio Beccari
7 **Come scrivere un libro e sopravvivere**
Grazia Messineo, Salvatore Vassallo
21 **Semplificare LATEX con *ORG-mode* in Emacs**
Emmanuele F. Somma
49 **CJK Unicode strokes e radicals con MFLUA: uno studio preliminare**
Luigi Scarso
61 **Mapping to individual characters in expl3**
Joseph Wright
63 **TeXShop, Version 5 (August 11, 2022)**
Richard Koch
67 **Interactive content using TEX4ht**
Richard Koch

Gruppo Utilizzatori Italiani di TEX

Editoriale

Claudio Beccari

Scrivo questo editoriale in sostituzione del direttore prof. Francesco Biccari, impedito da impegni familiari irrevocabili. A nome suo esprimo i ringraziamenti del GUIT alle autorità delle Università di Roma Tor Vergata, e dell'Università Cattolica del Sacro Cuore per averci sostenuto ed ospitato in questa bella sede per riprendere i nostri Incontri annuali, i nostri GUIT Meeting, che, purtroppo, avevano subito notevoli inconvenienti e ritardi durante la pandemia.

Do il benvenuto ai Docenti delle Scuole di ogni ordine e grado, perché ritengo che troveranno interessanti gli argomenti che verrano trattati.

Il tema di questo convegno riguarda "TEX e la tipografia digitale in ambito umanistico: tra innovazione e durabilità delle tecnologie tipografiche".

Non c'è dubbio che il "Sistema TEX" sia stato e continui ad essere un una singolarità isolata nel campo delle Scienze umane, pur essendo nato nell'ambito della matematica e delle scienze naturali.

I contenuti delle scienze umane sono altrettanto difficili da trasmettere ai lettori di tutte le età; penso, per esempio, alle antologie adeguatamente commentate con diversi apparati di note; penso ai libri di storia, ai testi destinati all'insegnamento delle lingue antiche, come il latino e il greco. Per gli umanisti ci sono molti ambiti dove riferirsi a lingue esotiche con scritture da sinistra a destra, o a quelle che si scrivono da destra a sinistra, oppure in verticale…

Nella mattinata di questo convegno avremo tre presentazioni, una dedicata ai docenti che si cimentano con la scrittura di testi con l'obbligo di seguire regole tipografiche di solito usate come stile distintivo della casa editrice.

Un altro articolo dovrebbe essere particolarmente gradito dai docenti che usano i programmi liberi, a cominciare dal sistema operativo Linux e il programma di editing `emacs`; si tratta di un programma potentissimo e integrabile con altri programmi, in modo da estenderne le funzionalità in modo prezioso per l'utente, specialmente quando ha da scrivere tanto codice LATEX, per comporre i suoi testi.

Il terzo articolo riguarda il trattamento dei grafemi gestiti con l'ormai universale codifica Unicode; è facile gestire i grafemi formati da un solo glifo, e non da comporre mettendo insieme diversi glifi più semplici. I caratteri CJK, (acronimo per *Chinese, Japanese, Kirean*) sono particolarmente soggetti a questi procedimenti; se poi uno desidera disegnarsi i propri glifi, allora le conoscenze necessarie sono difficili da acquisire se non ci fossero articoli come quello presentato qui.

Nel pomeriggio presenterò (in italiano) il contenuto di tre articoli già pubblicati sulla rivista internazionale TUGboat.

Il primo, di nuovo, parla di accesso ai grafemi completi delle loro parti; prende ad esempio un caso scritto in bengalese; la singolarità di questa scrittura si presta benissimo per discutere il problema.

Gli altri due articoli sembrano destinati ai soli utenti di Apple Mac; l'articolo relativo al programma di editing TEXShop, adatto a lavorare con LATEX sulle macchine con il sistema Operativo MAC OS X, sembra limitato ad un uditorio limitato; ma quello che appare in questo programma, prima o poi viene adottato anche da altri. L'articolo importante per i docenti riguarda il processo di produrre testi per la stampa, in cui il concetto di pagina è essenziale, e testi per il Web, dove il concetto di pagina è assente, ma in entrambi i casi partendo dallo stesso file sorgente scritto in linguaggio LATEX.

Claudio Beccari
Professore emerito
del Politecnico di Torino

Come scrivere un libro e sopravvivere

Grazia Messineo e Salvatore Vassallo

Sommario Nel 2022, la scrittura a quattro mani di un manuale di Matematica per gli studenti del primo anno dei Corsi di Laurea in Economia ci ha posti di fronte a varie sfide, che possono essere suddivise in due macro aree: la necessità di usare un pacchetto che consentisse la scrittura collaborativa, magari con i file su cloud, con autori che lavorano in momenti diversi e parzialmente sovrapposti, e la necessità di soddisfare le richieste stilistiche imposte dalla casa editrice, alcune delle quali note a priori ed altre che sono emerse solo in fase di correzione delle bozze.

Abbiamo trovato, per molti di questi problemi, alcuni "trucchi" e soluzioni, che condividiamo in questo articolo con coloro che vogliono cimentarsi in un lavoro analogo, precisando che queste sono le soluzioni che hanno consentito a noi di sopravvivere e di produrre un testo che soddisfa tutti i requisiti imposti, ma che non sono forse le soluzioni ottimali e neppure le uniche soluzioni possibili ai problemi che ci si sono presentati.

Abstract In 2022, we wrote with two other colleagues a book of Calculus for first year students of Economy. Writing this book has posed various problems, which can be divided in two areas: the need of using a package to collaboratively write the book, possibly with the files saved on a cloud service, with authors working in different times, but sometimes at the same time; and the need to satisfy the stylistic requests of the editor, some of which already known and other known during the proofreading phase.

We found, for a great number of these problems, some tricks and solutions that we want to share in this paper with those willing to begin a similar job. Let us specify that these are the solutions which allowed us to survive and to produce a book satisfying all the characteristics imposed by the editor. Perhaps these are not the optimal solutions, nor the unique.

1. Introduzione

Nel 2022 con altri due colleghi abbiamo intrapreso la scrittura di un manuale di Matematica per gli studenti di Economia della nostra Università. Questo lavoro vuole essere il racconto della nostra "battaglia" con LaTeX per ottenere un risultato simile a quello desiderato dall'editore.

Le soluzioni che abbiamo trovato non sono necessariamente le migliori o le più efficaci, ma hanno portato al risultato atteso. Sicuramente molti altri utenti TeX hanno (o avrebbero) trovato altre soluzioni.

Il problema di adeguare la produzione di un file `pdf` agli standard editoriali richiesti è già stato affrontato: ad esempio Luca Pignalberi in Pignalberi (2015) mostra come ha modificato la classe `book` per adeguarla a varie collane dello stesso editore.

Nel nostro caso le modifiche dovevano riguardare solo il nostro manoscritto e, avendo già avuto alcuni rapporti con l'editore in precedenza, avevamo già alcune indicazioni.

Il nostro compito era quello di produrre un file `pdf` pronto per la stampa *senza* alcuni elementi come pagina iniziale, copyright, ecc. che formano le prime sei pagine. Sotto molti aspetti l'editore ci ha lasciato una certa libertà: scelta dei caratteri, del sezionamento, uso delle figure e delle tabelle, scelta degli indici ed altro ancora.

Quali sono stati i problemi che ci si sono posti? La lista è questa:

1. la possibilità di operare in più persone;
2. la geometria della pagina (larghezza e altezza del testo, altezza delle testatine e dei piè di pagina, larghezza e altezza della pagina);
3. la formattazione delle testatine e il formato dei nomi dei capitoli, sezioni, …;
4. la formattazione di alcuni elementi della pagina (rientri, note a piè di pagina, …);
5. rientri e spaziature degli ambienti;
6. centratura e corpo del testo delle figure;
7. la formattazione dell'intestazione di teoremi e simili;
8. la formattazione dell'indice, dell'indice analitico, dell'elenco delle figure e delle tabelle;
9. vedove ed orfani.

Alcuni di questi problemi, con alcune soluzioni, sono affrontati in Pantieri e Gordini (2017), altri sono discussi a più riprese sul forum del Gruppo Utilizzatori Italiani di TEX e su stackexchange.

Per il seguito è utile tener presente che nel testo sono presenti moltissime figure (più di 130) e 6 tabelle su circa 350 pagine.

Osserviamo preliminarmente che la nostra procedura per risolvere dei problemi utilizzando LATEX si compone generalmente di questi passaggi:

- ricerca di un pacchetto che faccia quello di cui abbiamo bisogno
- modifica del codice della classe o dei pacchetti utilizzati per ottenere il risultato richiesto
- creazione di codice *ad hoc*

In questo caso abbiamo utilizzato solo i primi due punti.

2. La scelta della classe e gli strumenti per la condivisione

Abbiamo deciso di utilizzare la classe suftesi di Ivan Valbusa (Valbusa, 2021b) per la sua flessibilità e la possibilità di modificare alcuni elementi strutturali del libro dalle opzioni della classe.

Poiché il lavoro è stato suddiviso tra più persone abbiamo deciso di condividere i file su Dropbox e di utilizzare il pacchetto subfiles: con questo pacchetto si crea un file principale in cui viene scritto il preambolo e vengono caricati dei "sottofile" (nel nostro caso capitoli). Ogni autore può compilare il proprio capitolo senza dover compilare tutto il libro e senza dover creare alcun preambolo. L'unico problema che presenta questa procedura è la mancanza dei riferimenti incrociati tra diversi capitoli e la numerazione non corretta degli ambienti, poiché ogni capitolo compilato autonomamente ha il numero zero.

Infine abbiamo deciso di utilizzare i font di default di suftesi, ovvero il magnifico Cochineal serif per i caratteri con grazie, Linux Biolinum per i caratteri senza grazie e Inconsolata per il carattere monospaziato.

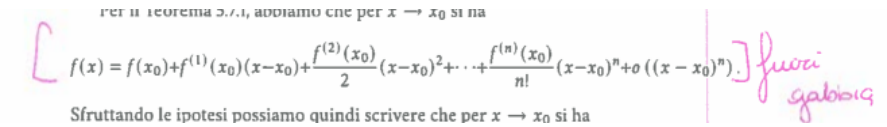
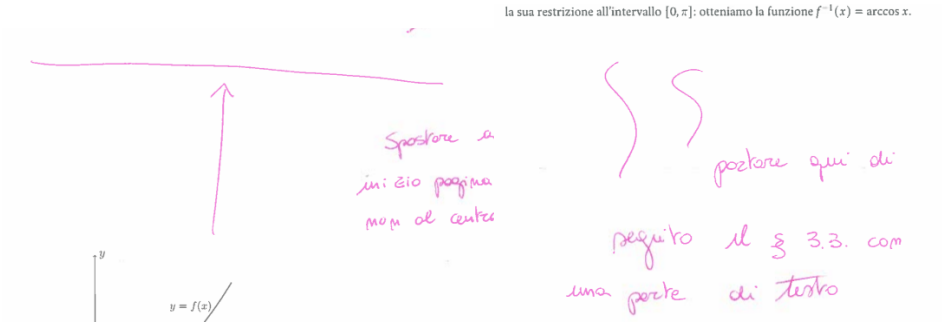
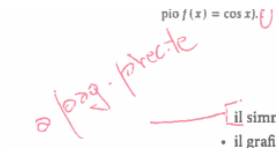


Figura 1. Formula fuori gabbia

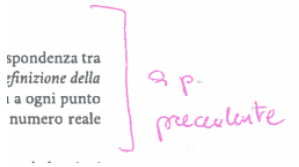


(a) Pagina vuota in alto

(b) Pagina vuota in basso



(c) Elenco spezzato male



(d) Paragrafo da spostare

Figura 2. Problemi "verticali"

3. La geometria della pagina

L'editore richiede una gabbia di 12.5×19 cm, ma con l'altezza del testo di 18.5 cm. Ovviamente per questo non ci sono stati problemi utilizzando il pacchetto geometry (Carlisle e Umeki, 2020).

Purtroppo però ci possono essere dei problemi tipografici che LaTeX da solo non può risolvere: usando la terminologia di Pantieri e Gordini ci sono problemi "orizzontali" e problemi "verticali". Queste problematiche sono emerse in fase di correzione delle bozze. Nel nostro caso i problemi "orizzontali" riguardavano quasi esclusivamente formule troppo lunghe (Figura 1) che sono state spezzate e alcune righe con spazi troppo larghi tra le parole per problemi di sillabazione risolti dando i consueti "suggerimenti" di sillabazione con il comando `\-`.

I problemi "verticali" sono stati di diversa natura: spazi troppo ampi a fine pagina (ovvero pagine troppo corte, si veda Figura 2), vedove ed orfani[1], problema di cui ci occuperemo in una sezione apposita (Sezione 9), elenchi o ambienti spezzati esteticamente male.

Il primo passo per rimuovere questi problemi è stato, come consigliato anche in Pantieri e Gordini (2017), quello di arretrare o avanzare di qualche capoverso l'inserimento di una o più

1. Ricordiamo che con il termine *vedove* in tipografia l'ultima riga di un capoverso che si trova all'inizio di una nuova pagina e con il termine *orfani* la prima riga di un capoverso in fondo ad una pagina.

figure. Spesso questo semplice accorgimento ha risolto il problema. Un'altra possibilità seguita è stata quella di ingrandire o rimpicciolire di poco (si veda anche la sezione 6) le dimensioni di una figura.

L'unica accortezza che abbiamo dovuto avere è stata quella di fare modifiche "locali" che, pur risolvendo il problema in una pagina, non si propagassero alle pagine successive.

Fortunatamente questi piccoli accorgimenti ci hanno permesso di risolvere quasi tutti i problemi del testo: restavano ancora un problema relativo all'indice analitico che affronteremo nella sezione 8.1 e un problema di "orfani".

4. Le testatine e i nomi dei capitoli

L'editore non richiede "piedini", ma solo "testatine" con

- su pagina dispari: numero e titolo della sezione a sinistra (in corsivo) e numero della pagina a destra (in tondo);
- su pagina pari: numero della pagina a sinistra (in tondo) e "Capitolo", numero e nome del capitolo a destra (in corsivo).

La classe suftesi permette di ridefinire le testatine utilizzando alcuni comandi interni oppure con i comandi di fancyhdr (si veda van Oostrum (2022)) che viene caricato dalla classe.

Nel nostro caso abbiamo inserito il codice[2]

```
\renewcommand{\chaptermark}[1]{%
\markboth{\smallrr\chaptername
\ \smallrr\thechapter.\ #1}{}}
\renewcommand{\sectionmark}[1]{%
\markright{\smallrr\thesection.\ #1}{}}
```

Purtroppo all'editore non è piaciuta la nostra scelta di usare le "old style figures" perché i numeri dei capitoli e delle pagine risultavano troppo piccoli, sia nelle testatine che nei titoli dei capitoli, e abbiamo dovuto ripiegare sui numeri "normali" (Figura 3). Per la formattazione dei comandi di sezionamento abbiamo scelto di usare uno stile simile a uno di quelli predefiniti di suftesi, rendendolo però in grassetto.

Un intervento più pesante è stato richiesto per le *subsections* e le *subsubsections*: in questo caso suftesi aggiunge uno spazio prima del titolo di sezionamento che l'editore non desiderava: abbiamo quindi dovuto intervenire direttamente con i comandi di titlesec (Bezos, 2021) per eliminare questo spazio (Figura 4).

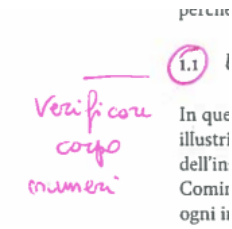
Altri problemi sulle testatine riguardavano gli indici (e ce ne occuperemo nella sezione 8) e la Prefazione.

La Prefazione è un capitolo non numerato e nella testatina deve quindi apparire solo la parola "Prefazione" che deve però apparire nell'indice. Quindi prima della Prefazione abbiamo dovuto ridefinire la testatina e la numerazione dei capitoli con

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

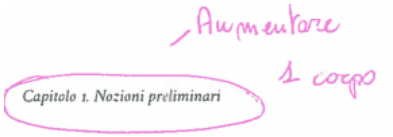
2. Il comando `\smallrr` è un comando del pacchetto fontsize sempre di Ivan Valbusa (Valbusa, 2021a).

(a) Numero del capitolo



(b) Numero di sezione

(c) Numero di pagina



(d) Testatina

Figura 3. I numeri scritti in "old style" non vanno bene

Figura 4. Nome della sottosezione

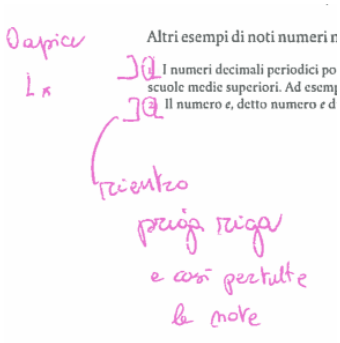
Ovviamente queste variazioni dovevano essere "locali" e non espandersi a tutto il testo e quindi è stato definito un ambiente apposito (si veda Bezos (2021)). Abbiamo quindi creato un semplice stile di pagina con fancyhdr che tenesse conto delle nostre esigenze

```
\fancypagestyle{prefazione}{%
\fancyhead{} % get rid of headers
\fancyfoot{}
\fancyhead[LE,RO]{\thepage}
\fancyhead[RE,LO]{\itshape{Prefazione}}
}
```

5. Note, rientri, elenchi

Ad una prima scrittura non ci siamo preoccupati della formattazione delle note a piè di pagina, ma successivamente l'editore ci ha fornito indicazioni più stringenti: era necessario un filetto di separazione (o un ampio spazio di separazione dal testo), il numero della nota doveva essere in apice, la prima riga doveva avere un rientro come il testo.

In questo caso la soluzione è stata in un primo tempo un mix tra la flessibilità della classe suftesi e l'utilizzo di altri pacchetti: in particolare abbiamo usato l'opzione footnotestyle = superscript di suftesi e con il pacchetto footmisc (Mittelbach, 2022a) abbiamo sistemato gli altri problemi. Successivamente, senza avere l'ansia dei tempi stretti, abbiamo eliminato l'utilizzo di footmisc modificando leggermente il codice di Ivan Valbusa per le note con opzione superscript ridefinendo il testo della nota in



(a) Numero della nota e rientro

(b) Filetto della nota

```
\makeatletter
\renewcommand\@makefntext{%
\hskip\baselineskip
\textsuperscript{\@thefnmark}}
\makeatother
```

C'è da notare che nelle note la lunghezza `\parindent` è zero e il rientro è dato dalla lunghezza `\baselineskip`.

Utilizzando le opzioni di enumitem (Bezos, 2019) abbiamo poi definito come dovevano apparire di default gli elenchi puntati e numerati.

Il comando `\setlist[itemize,1]{label=\textbullet}` definisce l'etichetta degli elenchi puntati di primo livello con "•" mentre il comando `\setlist[itemize,2]{label=-}` definisce l'etichetta degli elenchi puntati di secondo livello con "-".

Analogamente per gli elenchi numerati abbiamo usato per il primo livello il comando `\setlist[enumerate,1]{label=(\arabic*),ref=(\arabic*)}` che definisce l'etichetta con gli usuali numeri tra parentesi tonde e la referenza all'item nello stesso modo, mentre per il secondo livello `\setlist[enumerate,2]{label=\emph{\alph*})}` che definisce l'etichetta con lettere minuscole tra parentesi tonde. Per il secondo livello abbiamo poi indicato tra le opzioni `ref=(\theenumi.\emph{\alph*})` in modo che la referenza a un item contenga anche l'etichetta del primo livello.

Si noti che i comandi definiscono anche come devono essere indicate le referenze agli item degli elenchi numerati (tra parentesi indicando l'etichetta dell'item e, per il secondo livello, anche l'etichetta del primo livello).

6. Gli ambienti flottanti

Mentre non abbiamo avuto grossi problemi con le tabelle, a parte il ridimensionamento di alcune righe e il posizionamento della didascalia che l'editore richiede in basso come per le figure, il posizionamento delle figure (ricordiamo che nel libro ci sono più di 130 figure) ha richiesto uno sforzo supplementare.

Iniziamo col dire che abbiamo usato il pacchetto subcaption (Sommerfeldt, 2022) per poter affiancare le figure e avere le relative didascalie e che tutte le figure erano file pdf esterni,

ognuno dei quali generato con pstricks o tikz ed il pacchetto standalone (Scharrer, 2018) per avere figure "ben ritagliate".

Ci si sono presentati subito alcuni problemi: figure troppo lontane dal testo a cui si riferiscono, didascalie troppo lunghe, problemi di centratura orizzontale, testo nelle figure troppo piccolo (spesso) o troppo grande.

Alcuni problemi sono stati risolti con il pacchetto placeins (Arseneau, 2005) ed altri con gli spostamenti delle figure come indicato nella sezione 3.

Per uniformare la grandezza del testo si è dovuto talvolta riscrivere il codice delle figure (alcune scritte usando tikz, altre pstricks).

I maggiori problemi si sono avuti con la centratura orizzontale delle figure e delle loro didascalie: alcune figure affiancate non risultavano ben centrate o le didascalie risultavano troppo corte o troppo lunghe.

Questi problemi a volte sorgevano dalla combinazione di alcuni fattori: un margine troppo ampio per le figure, parametri non corretti nei comandi del pacchetto subcaption che abbiamo usato.

Per poter affiancare le figure (ed altri ambienti flottanti) ci sono due possibilità: l'ambiente subcaptionblock e il comando `\subcaptionbox`. Per le immagini il comando `\subcaptionbox` ha questa sintassi (abbiamo volutamente ignorato alcuni parametri opzionali)

`\subcaptionbox{`⟨*didascalia*⟩`}[`⟨*larghezza*⟩`]`

`{`⟨`\includegraphics[`⟨*opzioni*⟩`]{`⟨*file da inserire*⟩`}`⟩`}`

L'ambiente subcaptionblock ha invece questa sintassi (anche in questo caso abbiamo volutamente ignorato alcuni parametri opzionali)

```
\begin{subcaptionblock}{larghezza}
\includegraphics[opzioni]{file immagine}
\caption{didascalia}
\end{subcaptionblock}
```

In entrambi i casi bisogna inserire il codice in un ambiente figure che ha una propria didascalia, ma `\subcaptionbox` inserisce il suo contenuto in una `\parbox` mentre subcaptionblock lo inserisce in una minipage.

Se si hanno due figure affiancate per avere la centratura corretta della didascalia della figura le due larghezze devono essere uguali.

È facile capire che è abbastanza facile avere situazioni "difficili" soprattutto se le figure non hanno margini ben bilanciati: in questo caso si è a volte proceduto inserendo spazi orizzontali in modo da ottenere una centratura corretta.

7. Teoremi

In questo caso c'era un unico problema: di default LaTeX numera i Teoremi in questo modo se non hanno un "nome"

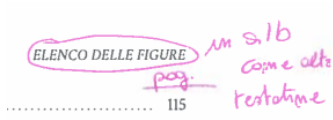
Teorema 1.1.

e in questo modo se hanno un "nome"

Teorema 1.1 (Teorema di Duck).

C'è da notare la differente posizione del punto nei due casi.

(a) Indice analitico



(b) Elenco delle Figure

Figura 6. Le testatine degli indici sono in maiuscolo

L'editore voleva, nel secondo caso, il punto dopo il numero. Questa semplice variazione tipografica non è semplicissima da implementare. La prima soluzione trovata, dovendo risolvere il problema in tempo brevissimo è stata la ridefinizione dello stile *plain* in modo da aggiungere il punto in ogni caso. Questa soluzione non era comunque soddisfacente perché l'intervento sul codice era molto pesante. Con maggiore calma si è quindi successivamente cercata un'altra soluzione, esaminando i pacchetti che permettono una personalizzazione degli ambienti "Teorema" e simili.

Alla fine abbiamo visto che il pacchetto thmtools (Chou, 2022) ci permetteva di ridefinire completamente gli ambienti, intervenendo in maniera molto fine su ogni aspetto dell'ambiente *teorema* e derivati (corollario, lemma, esempio, definizione, osservazione, …): i caratteri usati per i vari elementi (il corpo, il nome, il numero, …), la spaziatura tra gli elementi, i separatori.

8. Indici

In questa sezione ci occupiamo di tutti gli indici anche se ognuno di questi presenta sue specificità, ma alcuni problemi sono comuni.

Il primo problema è la testatina: sia l'elenco delle figure che quello delle tabelle che l'indice analitico presentano il loro nome nella testatina in lettere maiuscole (si veda la Figura 6). Il problema è facilmente risolubile modificando lo stile delle testatine. In questo caso abbiamo preso il codice di default suftesi aggiungendo un comando `\nouppercase` in modo da avere il formato normale, ad esempio, per l'indice analitico abbiamo:

```
\fancypagestyle{index}{%
\fancyhead{} % get rid of headers
\fancyfoot{}
\fancyhead[LE,RO]{\thepage}
\fancyhead[RE]
{\itshape{\nouppercase\leftmark}}
\fancyhead[LO]
{\itshape{\nouppercase\rightmark}}
}
```

8.1. L'indice analitico

Per la costruzione dell'indice analitico ci siamo appoggiati al pacchetto imakeidx (Gregorio, 2016) con l'utilizzo del programma di default makeindex (Lamport, 1987). Nel libro l'indice analitico è su due colonne, mentre il testo ordinario è su una colonna.

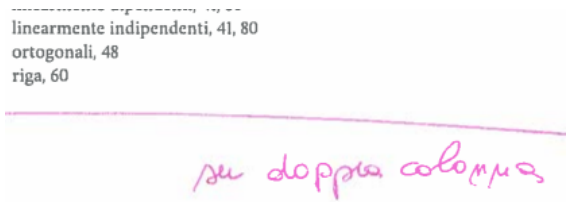


Figura 7. Le colonne dell'indice analitico non sono bilanciate

Il primo problema che si è presentato è stato quello dell'ultima pagina dell'indice analitico che non aveva le colonne bilanciate, come richiesto dall'editore, e, purtroppo, questa pagina non era l'ultima del libro (Figura 7). Ci sono molti pacchetti che permettono il bilanciamento di un testo su più colonne automaticamente ad esempio balance (Daly, 1999) e flushend (Tolušis, 2021). Purtroppo nessuno di questi pacchetti risolveva il nostro problema. Il pacchetto balance richiede di inserire il comando `\balance` dove inizia il testo da bilanciare e il comando `\nobalance` al termine: questo per noi voleva dire prima di `\printindex` e prima di `\printbibliography` rispettivamente. Purtroppo questo portava al bilanciamento dell'ultima pagina, ma sbilanciava due pagine precedenti.

Il pacchetto flushend semplicemente non bilanciava l'ultima pagina.

Il problema è stato risolto con il pacchetto pbalance (Lago, 2022) che deve solo essere caricato e non ha bisogno di comandi: il risultato è stato ottimo.

Un altro problema con l'indice analitico è stato la formattazione del testo: premettiamo che, a nostro parere, la documentazione dei file di stile dell'indice analitico risente pesantemente dell'età e dà per scontato che chi scrive sia più un programmatore che un utente.

Lo stile che avevamo scelto prevede

- la lettera identificativa in grassetto sul bordo sinistro;
- la voce in corsivo;
- le sotto-voci in tondo;
- il numero di pagina uniforme al testo precedente.

Questo stile è ben documentato e il file `.ist` che lo produce è

```
preamble
"\\begin{theindex}
\\renewcommand\\item%
{\\par\\normalfont\\itshape}
\\renewcommand\\subitem%
{\\par\\hspace{5mm}\\normalfont}"

headings_flag 1
heading_prefix "\\goodbreak\\textbf{"
heading_suffix "}\\par\\nobreak\\vskip%
\\smallskipamount\\nobreak"
```

(a) Indice **(b)** Elenco delle Figure

Figura 8. Sopra i numeri di pagina ci deve essere "pag."

```
symhead_positive "Simboli"
symhead_negative "Simboli"
numhead_positive "Numeri"
numhead_negative "Numeri"
```

Qui preamble indica ciò che andrà all'inizio del file `.idx` e, in questo caso determina il carattere usato per le voci (`\itshape`) e per le sotto-voci (`\norlmalfont`).

Gli altri comandi servono a indicare che il "nome" delle voci è la lettera iniziale maiuscola in grassetto.

L'editore ci ha chiesto i numeri di pagina in tondo: come inserire questa variazione nella riga del testo?

Qui nessuna guida ci era di aiuto: nella guida Chen e Harrison (1987) sono elencati molti parametri che indicano come si separano le voci, come le voci dalle sotto-voci, le sotto-voci tra di loro, una voce dal numero di pagina in cui appare. Non è però previsto un cambiamento di carattere tra voce e numero di pagina.

La nostra idea è stata allora quella di "forzare" uno dei parametri, in particolare il parametro `delim_0` che rappresenta il delimitatore tra una voce e il relativo numero di pagina che, di default, è la virgola, cioè `delim_0 ", ''`. Abbiamo modificato il delimitatore in `delim_0 ", \\upshape"` ottenendo così il numero di pagina in tondo.

8.2. Indice

L'indice era già formattato in parte secondo le norme editoriali: le opzioni tocstyle = dotted e compacttoc = par della classe suftesi hanno dato all'indice l'aspetto desiderato.

L'editore però aveva una richiesta che si trova pù volte anche su stackexchange: in ogni pagina, sopra alla colonna con il numero di pagina, doveva esserci "pag." e lo stesso per l'elenco delle figure e delle tabelle (si veda la Figura 8).

Noi avevamo già aggiunto "pag." alla prima pagina dell'indice con il solito comando

```
\addtocontents{toc}
{\protect\hfill{\vspace{1mm}\emph{pag.}\par}}
```

ma questo non risolveva il problema delle altre pagine.

La soluzione più consigliata su stackexchange sembra quella di aggiungere manualmente nel file `.toc` (e analogamente nei file `.lof` e `.lot`) la riga
`\protect\hfill{\vspace{1mm}\emph{pag.}\par}`.

Questa soluzione non era praticabile perché avrebbe implicato un intervento manuale per ogni compilazione e questo per qualsiasi autore avesse ricompilato: l'errore o la disattenzione erano dietro l'angolo!

Figura 9. Orfano

La nostra soluzione è stata quella di intervenire sulle testatine: poiché suftesi fa uso di fancyhdr è sufficiente definire uno stile di testatina che inserisca "pag." al posto giusto e che venga utilizzato solo per indice e elenco delle figure e delle tabelle. Il codice usato è:

```
\fancypagestyle{fancy}{%
\fancyhead{}
\fancyfoot{}
\setlength{\headheight}{38.06pt}
\fancyhead[LE]%
{\thepage\linebreak\linebreak}
\fancyhead[RO]%
{\thepage\linebreak\linebreak {\itshape pag.}}
\fancyhead[RE]%
{\itshape{\nouppercase\leftmark}%
\linebreak\linebreak pag.}
\fancyhead[LO]%
{\itshape{\nouppercase\rightmark}
\linebreak\linebreak}
}
```

I due comandi `\nouppercase\leftmark` e `\nouppercase\rightmark` sono necessari per l'elenco delle figure e delle tabelle perché in tali elenchi di default le testatine sono in maiuscolo.

9. Vedove ed orfani

Il lavoro svolto da LaTeX in questo caso è stato ottimo e l'editore non ha avuto molte segnalazioni e quasi tutte si sono sistemate modificando la posizione delle figure come indicato nella Sezione 3.

Esistono comunque alcuni pacchetti che facilitano la soluzione del problema e tra questi segnaliamo nowidow (Pinson, 2011) e widow-and-orphans (Mittelbach, 2022b).

Non è stato possibile risolvere un unico problema (Figura 9) in cui l'orfano è il nome di un ambiente.

Evidentemente LaTeX non interpreta la riga coma una riga isolata poiché è un paragrafo completo.

In questo caso anche l'utilizzo dei due pacchetti indicati non ha portato al risultato sperato e si è dovuto aggiungere uno spazio a fine pagina per mandare il testo sulla pagina successiva[3].

3. Si sarebbe potuto utilizzare il comando `\enlargethispage` che non conoscevamo: ringraziamo i revisori per avercelo segnalato, lo useremo nella seconda edizione del libro.

10. Conclusioni

Le case editrici stanno cominciando ad usare ed apprezzare LaTeX in misura molto più estesa di quanto avveniva una quindicina di anni fa, quando il suo uso era limitato, senza troppo entusiasmo, ai soli testi di matematica o discipline affini. Il file prodotto con LaTeX richiede in genere pochissime correzioni nell'impaginazione e negli altri aspetti "estetici", a patto che la composizione venga effettuata senza introdurre troppe forzature (figure ed altri oggetti "floating" forzati in una certa posizione, spaziature forzate sia in orizzontale che in verticale, ecc.).

D'altro canto, però, le diverse richieste tipografiche delle case editrici con le quali si collabora richiedono una certa capacità di "manipolare" ambienti, stili, ecc. che non è alla portata, spesso, dell'utente novizio. Qui si sono forniti alcuni "trucchi" per sopravvivere a questa sfida, senza la pretesa che le soluzioni trovate siano le migliori possibili.

Sarebbe auspicabile, per diffondere maggiormente l'utilizzo del programma, implementare, in collaborazione con le case editrici (almeno le maggiori), uno stile personalizzato (come quello già disponibile per molte riviste), che possa poi essere usato dagli autori senza troppi interventi.

Riferimenti bibliografici

Arseneau, Donald (2005). «The placeins package». CTAN:macros/latex/contrib/placeins/placeins-doc.pdf.

Bezos, Javier (2019). «Customizing lists with the enumitem package». CTAN:macros/latex/contrib/enumitem/enumitem.pdf.

— (2021). «The titlesec, titleps and titletoc packages». CTAN:macros/latex/contrib/titlesec/titlesec.pdf.

Carlisle, David e Hideo Umeki (2020). «The geometry package». CTAN:macros/latex/contrib/geometry/geometry.pdf.

Chen, Pehong e Michael A. Harrison (1987). «Index preparation and processing». CTAN:indexing/makeindex/paper/ind.pdf.

Chou, Yukai (2022). «Thmtools users' guide». CTAN:macros/latex/contrib/thmtools/doc/thmtools-manual.pdf.

Daly, Patrick W. (1999). «Balancing the two columns of text on the last page». CTAN:macros/latex/contrib/preprint/balance.pdf.

Gregorio, Enrico (2016). «The package imakeidx». CTAN:macros/latex/contrib/imakeidx/imakeidx.pdf.

Lago, Nelson (2022). «The pbalance (poor man's balance) package». CTAN:macros/latex/contrib/pbalance/pbalance.pdf.

Lamport, Leslie (1987). «Makeindex : An index processor for LaTeX». CTAN:indexing/makeindex/doc/makeindex.pdf.

Mittelbach, Frank (2022a). «footmisc — a portmanteau package for customizing footnotes in LaTeX». CTAN:macros/latex/contrib/footmisc/footmisc-doc.pdf.

— (2022b). «The widows-and-orphans package». CTAN:macros/latex/contrib/widow-and-orphans/widow-and-orphans-doc.pdf.

Pantieri, Lorenzo e Tommaso Gordini (2017). *L'Arte di scrivere con LaTeX*. http://www.lorenzopantieri.net/LaTeX_files/ArteLaTeX.pdf.

Pignalberi, Gianluca (2015). «LaTeX in un'editrice universitaria: come e perché book non è adatta e una possibile alternativa funzionante». *ArsTEXnica*, (19), pp. 33–41. http://www.guitex.org/home/numero-19-aprile-2015.

Pinson, Raphäel (2011). «The nowidow package». CTAN:macros/latex/contrib/nowidow/nowidow.pdf.

Scharrer, Martin (2018). «The standalone package». CTAN:macros/latex/contrib/standalone/standalone.pdf.

Sommerfeldt, Axel (2022). «The subcaption package». CTAN:macros/latex/contrib/caption/subcaption.pdf.

Tolušis, Sigitas (2021). «The flushend package». CTAN:macros/latex/contrib/sttools/flushend.pdf.

Valbusa, Ivan (2021a). «The fontsize package». CTAN:macros/latex/contrib/fontsize/fontsize.pdf.

— (2021b). «The suftesi document class». CTAN:macros/latex/contrib/suftesi/suftesi.pdf.

van Oostrum, Pieter (2022). «The fancyhdr and extramarks packages». CTAN:macros/latex/contrib/fancyhdr/fancyhdr.pdf.

Grazia Messineo
Università Cattolica Milano – IIS "Falcone-Righi" Corsico
grazia.messineo@unicatt.it

Salvatore Vassallo
Università Cattolica Milano
salvatore.vassallo@unicatt.it

Semplificare LaTeX con *ORG-mode* in Emacs

Emmanuele F. Somma

Sommario Nella stesura degli articoli accademici o tecnici si può ridurre la complessità nell'uso dei marcatori o delle configurazioni se si abbandona il *linguaggio di markup* di LaTeX e si adotta *ORG-mode* in Emacs, pur senza rinunciare all'elevata qualità tipografica del LaTeX. Quest'articolo presenta il markup ORG usato per l'esportazione verso LaTeX, e mostra gli attributi dedicati agli elementi più comuni degli articoli accademici, come collegamenti, tabelle, note, figure e riferimenti bibliografici. In appendice, come progetto pratico, sono documentate le scelte compiute dall'autore per seguire le regole tipografiche di ArsTeXnica nella consegna del presente articolo.

Abstract In writing academic or technical articles, you can reduce complexity by using markup or configurations, dropping the LaTeX *markup language* and adopting *ORG-mode* in Emacs while retaining the LaTeX high typographical quality. This article shows how the ORG markup exported to LaTeX works. Also, it explains the attributes of the most common elements of academic papers, such as links, tables, notes, figures, and bibliographic references. Finally, in the appendix, as a practical project, the author's choices for delivering this article following the typographical rules of ArsTeXnica are documented.

1. Introduzione

ORG e *Markdown* sono linguaggi di markup leggero (*lightweight markup language* o LML) per creare testi ben impaginati utilizzando solo un editor di testo. *ORG-mode*, o, come recita il suo nome completo il "Carsten's[1] outline-mode for keeping track of everything", è un ambiente testuale in cui tener traccia di tutto quello che torna utile nel normale processo di lavoro di un ricercatore (agende, liste di cose da fare, note sparse, mappe mentali, ecc.), ma la sua caratteristica più interessante è quella di poter *esportare* il contenuto dei file di testo in molti sistemi di impaginazione attraverso *back-end* di traduzione. LaTeX è uno di questi.

Il linguaggio *Markdown* fu creato da John Gruber e Aaron Swartz nel 2004 (Gruber, 2004); come evoluzione delle convenzioni già usate nella redazione di testi semplici, come messaggi di posta elettronica e newsletter, forum online, wiki o brevi pezzi di documentazione testuali. Comunemente considerato il capostipite, o quantomeno l'archetipo, degli LML, è sicuramente quello che ha saputo conquistare il maggior numero di citazioni online, ma in realtà nasce *dopo* alcune precedenti formulazioni di markup leggeri e *insieme* a molti altri pressoché simili. Ha però avuto il merito di razionalizzare ed estendere le disparate funzionalità solitamente presenti in un LML e rendere più accessibili i documenti, ma anche il difetto di essere implementato in molti dialetti lievemente incompatibili (Mailund, 2014; Voegler *et al.*, 2014; White, 2022; Leonard, 2016).

AsciiDoc, *Textile*, *reStructuredText*, *txt2tags* e *WikiText*, il linguaggio delle pagine di *Wikipedia*, cioè i più comuni *dialetti* di LML, hanno tutti una genesi pressoché contemporanea a

1. Carsten Dominik è Professore di Astronomia e direttore dell'Anton Pannekoek Institute for Astronomy della Facoltà di Scienze Naturali dell'Università di Amsterdam, è anche autore del popolare ambiente per citazioni LaTeX chiamato RefTeX.

partire dall'inizio degli anni 2000. Anche *ORG-mode* fa parte di questo gruppo: fu rilasciato per la prima volta nel 2003.

Tutti erano stati anticipati da linguaggi con minori potenzialità, come *BBcode*, *setext* e *POD* che risalgono alla metà degli anni '90.

Sulla scia del successo di *Markdown*, l'interesse sugli LML è cresciuto, esistono oggi una grande quantità di strumenti di lavoro basati su quest'approccio. La principale caratteristica di questi linguaggi è la semplicità di espressione, a scapito della varietà e della completezza delle funzionalità. Il loro uso non ha travalicato i limiti della redazione di testi brevi e poco strutturati nella produzione di pagine online. L'esempio più noto sono le pagine di Wikipedia redatte appunto con un LML specializzato, simile al Markdown.

ORG-mode è un'eccezione: permette lavorazioni più ampie e strutturate e pertanto si merita una presentazione anche per gli utenti LaTeX abituati alla produzione di testi di alta qualità.

L'esperienza di uso di un linguaggio di markup leggero, nel processo di lavoro di una redazione e poi nella completa impaginazione di una rivista basata su LaTeX, fu già raccontato nel 2009 sulle pagine di ArsTeXnica (Somma, 2009). In quel caso venne adottato un LML del 2002 chiamato BHL (*Brute to HTML and LaTeX*) di Bastien Guerry, diretto antenato del moderno *ORG-mode*. In un primo momento, BHL fu adottato per la redazione di articoli esportati in formato OpenOffice per la comunicazione tra redazione e impaginazione della rivista LinuxMagazine, poi venne esteso e ridenominato in TCHL, per includere le funzionalità utili alla redazione degli articoli tecnici in una nuova iniziativa editoriale dove venne calato in un flusso di lavoro questa volta completamente basato su LaTeX.

Le esigenze di una rivista tecnica commerciale sono però inferiori a quelle di un articolo accademico, più ampio e strutturato, con note, riferimenti bibliografici, matematica in linea ed equazioni numerate, nonché la gestione di molteplici lingue e eventualmente differenti alfabeti, e maggiore variabilità e complessità degli elementi *floating* (come figure, riquadri e tabelle).

BHL e in generale i linguaggi Markdown, non sono adatti per questi usi. È indispensabile rivolgersi a programmi di impaginazione ad-hoc e, nel caso dei markup, a linguaggi più sofisticati, come appunto LaTeX o differenti varianti di XML e SGML, al prezzo di maggiore prolissità del linguaggio e difficoltà di gestione dell'intero processo, o la necessità di adottare interfacce utente non sempre facilmente usabili e spesso costose.

Permettendo di sfruttare al meglio i vantaggi di entrambi gli approcci, il linguaggio di markup di *ORG-mode* può essere integrato con LaTeX per produrre documenti di qualità.

2. Breve confronto tra *markup*

Per fare tre soli esempi significativi della semplicità relativa di *ORG-mode* si consideri come sono realizzati: a) il corsivo di una parola, b) un ambiente per l'espressione di una citazione testuale e c) una tabella.

Nel caso del corsivo in LaTeX si usa un markup come

```
\emph{in corsivo}
```

impiegando una parola chiave (`'emph'`) e 6 caratteri differenti. Con *ORG-mode* si segue una sintassi testuale comune ed è sufficiente usare due volte lo stesso carattere `'/'`, come

```
/in corsivo/.
```

Per l'espressione di una citazione fuori linea, in LaTeX è possibile usare un ambiente *quote* o *quotation*:

```
\begin{quote}
« Lorem ipsum dolor sit amet, consectetur [...]
\end{quote}
```

In *ORG-mode* si usa il *blocco* `quote`.

```
#+BEGIN_QUOTE
« Lorem ipsum dolor sit amet, consectetur [...]
#+END_QUOTE
```

Ad esempio, volendo ottenere una citazione in lingua con un alfabeto non latino (es. greco) sarà sufficiente usare un paio di *attributi* per la definizione del linguaggio (`':environment foreigndisplayquote'` e `':options {greek}'`) e l'uso del giusto ambiente nel preambolo come:

```
#+LATEX_HEADER:\usepackage[autostyle=true]{csquotes}
```

e ottenere una cosa del genere:

> ἀγεωμέτρητος μηδεὶς εἰσίτω.

Infine, per l'impaginazione delle tabelle in LaTeX, tra parole chiave e caratteri di specifica, si usano sette differenti elementi, come in:

```
\begin{center}
\begin{tabular}{lll}
Intestazione 1 & Intestazione 2 & Intestazione 3\\
\hline
Cella 11 & Cella 12 & Cella 13\\
Cella 21 & Cella 22 & Cella 23\\
Cella 31 & Cella 32 & Cella 33\\
\end{tabular}
\end{center}
```

mentre la stessa tabella in notazione *ORG-mode* è ben più parsimoniosa[2]:

```
| Intestazione 1 | Intestazione 2 | Intestazione 3 |
|----------------+----------------+----------------|
| Cella 11       | Cella 12       | Cella 13       |
| Cella 21       | Cella 22       | Cella 23       |
| Cella 31       | Cella 32       | Cella 33       |
```

in ambedue i casi il risultato è identico:

Intestazione 1	Intestazione 2	Intestazione 3
Cella 11	Cella 12	Cella 13
Cella 21	Cella 22	Cella 23
Cella 31	Cella 32	Cella 33

La concisione del markup *ORG* non eguaglia la varietà del markup LaTeX, ma ci sono molti aspetti che si possono modificare nell'esportazione dei singoli elementi, in modo da coprire molti, se non tutti, i casi d'uso più comuni nella redazione di un articolo accademico.

2. Va anche specificato che *ORG-mode* ha un ambiente di redazione delle tabelle che permette di interagire con questi elementi testuali come se fossero un foglio di calcolo (con tanto di campi calcolati tramite formule).

3. Integrazione con Emacs

Il confronto tra *ORG-mode* e LaTeX può farsi secondo due linee: a) confronto tra il linguaggio di markup, accennato in precedenza e che si vedrà in maggior dettaglio dalla prossima sezione dell'articolo e b) confronto dell'integrazione con l'editor Emacs (Stallman, 1981).

LaTeX non impone l'utilizzo di uno specifico editor ma non è così per *ORG-mode*, che è strettamente legato all'editor Emacs. Esistono, è vero, implementazioni alternative[3], ma dal punto di vista della più produzione di articoli accademici *ORG-mode* e Emacs sono ancora un tutt'uno.

D'altro canto è anche vero che i *modi Emacs* per la redazione di file (e gruppi di file) LaTeX e `ORG` sono ambedue altrettanto potenti. Un eventuale confronto su questo punto si inoltrerebbe solo in scelte progettuali e dettagli implementativi dell'uno e dell'altro modo. Come ben sanno gli utenti di Emacs, quest'editor è particolarmente versato ed efficace nella redazione di testi per i vari linguaggi di markup, così come per quelli di programmazione. A parità di condizioni comunque non si può vantare alcuna sostanziale differenza nell'utilizzo dell'*ORG-mode* rispetto al LaTeX-*mode* di Emacs (Borkowski, 2018; Babenhauserheide, 2014).

Però, per utilizzare il sottosistema di impaginazione di *ORG-mode* per la produzione di documenti PDF, LaTeX deve essere comunque correttamente installato e configurato. Emacs, e di conseguenza *ORG-mode*, è in grado di adattarsi praticamente senza intervento a una installazione LaTeX ben realizzata, come quella che può ottenersi ormai dalle più comuni distribuzioni[4]. Il che porta, nella maggior parte dei casi comuni, ad avere il sistema `ORG` per LaTeX immediatamente disponibile dopo la corretta installazione di LaTeX e di Emacs.

Tuttavia, resta vero che, pur senza un sistema LaTeX installato, sarà sempre possibile produrre un impaginato esportando il testo del file ORG in altri formati (come Odt, HTML o testo base), e da questi, scontando la minore qualità in impaginazione, un PDF.

I file testuali con il markup `ORG` sono identificati dall'estensione `'.org'`. Com'è usuale, quando Emacs carica un file con un'estensione conosciuta si dispone ad interagirvi con il *modo principale* adeguato (è da questo che proviene il nome *ORG-mode*: è il *modo principale* con cui Emacs interagisce con i file `'.org'`).

Un modo principale in Emacs attiva uno o più menù appositi e una serie di *combinazioni di tasti* specifiche, che si possono utilizzare solo quando si è in editing su un file con quell'estensione. Ad esempio editando un file `'.org'` ci sarà un menu `'Org'`, e quando il cursore si trova all'interno di una tabella, comparirà anche un ulteriore menu per la redazione delle tabelle, per aggiungere colonne, righe, e altre operazioni, come già specificato[2].

In *ORG-mode* alcune funzioni interattive e le relative combinazioni di tasti sono basilari nel processo di lavoro. Ad esempio, `'C-c C-e'`, che corrisponde al comando `'M-x org-export-dispatch'`, apre il pannello con cui il testo può essere esportato in vari formati di *output*. Di qui si può scegliere il *back-end* di esportazione e quindi lanciare l'operazione di traduzione, ad esempio la combinazione `'l p'` crea un file PDF. Invece `'C-u C-c C-e'` ripete l'ultimo comando di impaginazione usato. Dopo un primo uso del precedente pannello, dove si selezione l'esportazione voluta, `'C-u C-c C-e'` diventa la combinazione di tasti più usata nella fase di impaginazione.

3. Si veda il vasto elenco di sistemi riportato (AA.VV., 2022). Per fare un unico esempio la visualizzazione come pagine web del markup `ORG` è stata integrata in Github che è diventato così un comodo mezzo di pubblicazione online del markup `ORG` senza neppure la mediazione di Emacs.
4. Si veda anche Giordano *et al.* (2013).

La combinazione completa ‘C-c C-e l p’ (legata a ‘M-x org-latex-export-to-pdf’) trasforma il testo ‘.org’ in LaTeX e poi lo impagina in PDF utilizzando pdfLaTeX (o il programma scelto dall’utente).

Se invece si usa ‘C-c C-e l o’, alla fine della compilazione si aprirà anche il programma di visualizzazione di sistema per mostrare il risultato dell’impaginazione in PDF.

Quindi, se ben configurato, *ORG-mode* compie tutte le necessarie, ed eventualmente molteplici, compilazioni necessarie a LaTeX per ottenere correttamente indici, bibliografie e riferimenti, senza che l’utente debba fare altro.

È utile personalizzare il processo di compilazione in PDF con il programma latekmk configurando appositamente la *variabile Emacs* org-latex-pdf-process in ~/.emacs:

```
(setq org-latex-pdf-process
      (list "latexmk -shell-escape -bibtex -f -pdf %f"))
```

Un’altra utile combinazione di tasti è ‘C-c C-,’ che apre un pannello da cui è possibile scegliere il tipo di blocco da inserire (ad esempio si può usare ‘e’ per un *esempio*, ‘s’ per un listato di programma in codice *sorgente*, ‘q’ per una *citazione*, ‘v’ per *versi* poetici, ecc.). Sono definite molte altre combinazioni di tasti per compiere attività specifiche, come inserire collegamenti (‘C-c C-l’), note a pié pagina (‘C-c C-x f’), citazioni (‘C-c C-x [’), e così via.

4. ORG per LaTeX

Per ottenere un file LaTeX a partire da uno ‘.org’ non c’è quindi bisogno di particolari operazioni se non quella di esportazione verso il file sorgente ‘.tex’ con ‘C-c C-e l l’.

Con *ORG-mode*, senza particolari configurazioni, il documento LaTeX esportato sarà un articolo (classe article). È possibile configurare in modo più raffinato questa esportazione.

Esistono due possibilità per configurare le operazioni di *ORG-mode*: a) attraverso il meccanismo della personalizzazione (o configurazione)[5] di Emacs, che si applica a tutti i documenti ORG o b) con la definizione di *direttive*[6] nel file ‘.org’ in lavorazione, che valgono ovviamente solo per il file in lavorazione.

La prima strategia è adatta a modifiche sull’intero ambiente di lavoro, e si basa sul meccanismo di personalizzazione del menu ‘Options->Customize->Group->Specific Group’,

5. In Emacs si considera una differenza tra la configurazione (*configuration*) e la personalizzazione (*customization*), per quanto l’obiettivo e il risultato finale può essere indistinguibile. Per *configurazione* si intende l’introduzione nel file di inizializzazione di Emacs (solitamente ‘~/.emacs’) di comandi in linguaggio di programmazione Lisp, solitamente molto semplici, per l’attivazione dei package e la definizione delle *variabili di personalizzazione* (o *customization variables*). Per *personalizzazione* invece si intende l’uso della struttura gerarchica dei pannelli di personalizzazione del menu ‘Options->Customize’ per poi definire le stesse *customization variabiles* e, a sua volta, generare una struttura all’interno del file ‘.emacs’ basata sulla funzione Lisp custom-set-variable. Le due modalità hanno risultati equivalenti, adottare l’una o l’altra forma, o un mix tra le due, è questione di preferenza dell’utente.
6. Si adotta in quest’articolo, per rendere più comprensibile la differenza, la convenzione (non comune) di denominare *direttive* le linee di configurazione presenti all’interno del file ‘.org’ (che di solito la documentazione chiama *variabili* o *comandi* ORG) riservando la dizione *variabili di personalizzazione* (o semplicemente *variabili Emacs*) alle *customization variables* a livello di editor. Le direttive saranno sempre rappresentate tra i caratteri ‘#+’ e un ‘:’, necessari per la loro definizione e saranno sempre riportate in maiuscolo, sebbene questo non sia strettamente necessario. Per rendere chiara anche la distinzione tra *variabili di personalizzazione*, configurabili dall’utente, e *funzioni interattive* di Emacs, cioè i comandi eseguibili dall’utente, al nome di queste ultime si premetterà sempre la combinazione di tasti ‘M-x’ necessaria ad attivarle.

indicando ‘org’ come gruppo, ovvero con il comando ‘M-x customize-group RET org’. In alternativa si possono introdurre apposite linee di comando Lisp nel file ‘~/.emacs’.

Invece, nel secondo caso si possono indicare direttamente nel file ‘.org’ le configurazioni usando le *direttive* del markup ORG che hanno la forma:

```
#+<CHIAVE>: <VALORE>
```

La chiave può essere in maiuscolo o minuscolo indifferentemente.

Quindi, lo stesso effetto della variazione della variabile ‘org-latex-default-class’, che sceglie quale *classe* LaTeX debba essere introdotta nella \documentclass in testa al file ‘.tex’, si ottiene usando la direttiva ‘#+LATEX_CLASS:’ nel file ‘.org’ :

```
#+LATEX_CLASS: report
```

e se si vogliono aggiungere delle opzioni alla classe si usa ‘#+LATEX_CLASS_OPTIONS:’:

```
#+LATEX_CLASS_OPTIONS: [12pt,oneside]
```

La direttiva ‘#+LATEX_COMPILER:’ sceglie il compilatore da utilizzare.

```
#+LATEX_COMPILER: xetex
```

Infine, una o più direttive ‘#+LATEX_HEADER:’ possono essere usate per introdurre linee *extra* all’interno del preambolo del documento LaTeX. Si ha così un modo comodo per aggiungere ulteriori *package* senza dover configurare ‘org-latex-default-packages-alist’, ma anche eventualmente per definire nuovi comandi o speciali configurazioni per LaTeX.

La conversione da ORG a LaTeX dipende dalla variabile ‘org-latex-classes’ che è così definita:

```
(("article" "\\documentclass[11pt]{article}"
  ("\\section{%s}" . "\\section*{%s}")
  ("\\subsection{%s}" . "\\subsection*{%s}")
  ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
  ("\\paragraph{%s}" . "\\paragraph*{%s}")
  ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))
 ("report" "\\documentclass[11pt]{report}"
  ("\\part{%s}" . "\\part*{%s}")
  ("\\chapter{%s}" . "\\chapter*{%s}")
  ("\\section{%s}" . "\\section*{%s}")
  ("\\subsection{%s}" . "\\subsection*{%s}")
  ("\\subsubsection{%s}" . "\\subsubsection*{%s}"))
 ("book" "\\documentclass[11pt]{book}"
  ("\\part{%s}" . "\\part*{%s}")
  ("\\chapter{%s}" . "\\chapter*{%s}")
  ("\\section{%s}" . "\\section*{%s}")
  ("\\subsection{%s}" . "\\subsection*{%s}")
  ("\\subsubsection{%s}" . "\\subsubsection*{%s}")))
```

Ogni elemento di questa lista, definito come:

```
(nome-classe inizio-del-preambolo elementi-di-segmentazione)
```

rappresenta una delle classi usuali dei documenti LaTeX (article, book e report).

Gli *elementi di segmentazione* della definizione sono coppie di comandi di titolazione, la prima da usare in caso di titoli numerati, la seconda per quelli non numerati. Come si vede, in article la segmentazione parte dalla section, mentre per book e report inizia da part, ma può ovviamente essere cambiata.

Nell’esportazione verso LaTeX, il *back-end* applicherà, prima di ogni altra cosa la classe definita a livello globale in ‘org-latex-default-class’ oppure nel documento con la

direttiva ‘#+LATEX_CLASS:’, le cui opzioni sono in ‘org-latex-default-class-options’ ovvero con la direttiva ‘#+LATEX_CLASS_OPTIONS:’. I package di default, elencati nella variabile ‘org-latex-default-packages-alist’ saranno inseriti successivamente e possono essere omessi usando la stringa speciale ‘[NO-DEFAULT-PACKAGES]’, in `inizio-del-preambolo` della classe definita in ‘org-latex-classes’.

Altri package, indicati in ‘org-latex-packages-alist’, saranno accodati dopo, e possono essere omessi con ‘[NO-PACKAGES]’ e infine saranno aggiunti gli ‘EXTRA’, cioè le linee definite nel file ‘.org’ con le direttive ‘#+LATEX_HEADER:’. Anche quest’insieme può essere escluso usando ‘[NO-EXTRA]’. Il preambolo di un articolo per ArsTeXnica può essere visionato in appendice a quest’articolo.

La variabile ‘org-latex-default-packages-alist’ di solito include i package inputenc, fontenc, hyperref e altri che servono alle varie funzionalità di *ORG-mode* e non dovrebbero mai essere esclusi.

Per lavorare con una differente classe LaTeX, ad esempio quella di arstexnica con cui è consegnato quest’articolo, sarà necessario configurare opportunamente ‘org-latex-classes’ come si vedrà in appendice.

Quando possibile, è sempre preferibile agire a livello del file ‘.org’, in modo anche da tener documentato nel file stesso le sue esigenze di compilazione e non dipendere da una configurazione generale che potrebbe cambiare nel tempo o tra differenti utilizzatori.

Le direttive dedicate a LaTeX sono solo quelle indicate, ma molte altre direttive generiche hanno effetto nell’esportazione.

Ad esempio, le direttive descrittive del documento come:

```
#+TITLE: Emacs ORG-Mode, LaTeX (e ArsTeXnica)
#+AUTHOR: Emmanuele F. Somma
#+EMAIL: ef.somma@exedre.org
#+CREATOR: Emacs 28.1 (Org mode 9.5.2)
#+DESCRIPTION: Un articolo per ArsTeXnica
#+KEYWORDS: Emacs ORG LaTeX impaginazione
```

sono utilizzate in modo appropriato dal *back-end* di esportazione

```
\author{Emmanuele F. Somma}
\date{\today}
\title{Emacs ORG-Mode, \LaTeX{} (e \ArsTeXnica{}) }
\hypersetup{
 pdfauthor={Emmanuele F. Somma},
 pdftitle={Emacs ORG-Mode, LaTeX (e ArsTeXnica)},
 pdfkeywords={Emacs ORG LaTeX impaginazione },
 pdfsubject={Un articolo per ArsTeXnica },
 pdfcreator={Emacs 28.2 (Org mode 9.5.5)},
 pdflang={Italian}}
```

Da notare anche la definizione della lingua con cui è scritto il documento:

```
#+LANGUAGE: it
```

Le lingue supportate sono nella variabile ‘org-latex-babel-language-alist’. La definizione della lingua però non include automaticamente il package LaTeX relativo in modo che l’utente possa scegliere se adottare babel o polyglossia, che si può definire nell’elenco generale dei package di default con il comando Lisp:

```
(add-to-list 'org-latex-packages-alist
             '("AUTO" "babel" t ("pdflatex" "xelatex" "lualatex")))
```

ovvero

```
(add-to-list 'org-latex-packages-alist
             '("AUTO" "polyglossia" t ("xelatex" "lualatex")))
```

Dove ‘AUTO’ indica la lingua scelta nella direttiva ‘#+LANGUAGE:’ e, come si vede, possono essere differenziate per motore di impaginazione.

In alternativa, il package linguistico si può inserire esplicitamente nel file ‘.org’ con una linea come:

```
#+LATEX_HEADER: \usepackage[italian,greek]{babel}
```

5. Opzioni del documento

Alcune opzioni del documento ORG hanno effetto sull’esportazione LaTeX. Le opzioni sono definite nel formato chiave:valore e inserite nella direttiva denominata ‘#+OPTIONS:’[7], ad esempio come segue:

```
#+OPTIONS: ':nil *:t -:t ::t <:t H:3 \n:nil ^:t arch:headline
#+OPTIONS: author:t broken-links:nil c:nil creator:nil
#+OPTIONS: d:(not "LOGBOOK") date:t e:t email:nil f:t inline:t num:t
#+OPTIONS: p:nil pri:nil prop:nil stat:t tags:t tasks:t tex:t
#+OPTIONS: timestamp:t title:t toc:t todo:t |:t
```

Le variabili booleane assumono i valori *vero*, se ’‘t’, e *falso*, se ‘nil’. Ecco alcune opzioni utili:

‘num:’ attiva la numerazione delle sezioni

‘toc:’ impagina la tavola dei contenuti (indice)[8]

‘|:’ attiva l’impaginazione delle tabelle

‘^:’ usa la sintassi TeX per apici e pedici [9]

‘-:’ permette la conversione delle stringhe speciali

‘f:’ attiva la conversione delle note (indicate come ‘[fn:...]’)

‘*:’ attiva le enfasi testuali (‘*’ per grassetto, ‘/’ per corsivo, ‘_’ per sottolineato)

‘TeX:’ permette le macro TeX nel testo.

‘author:’ include il nome dell’autore nell’impaginazione

‘email:’ include l’email dell’autore nell’impaginazione

‘creator:’ include le informazioni di creazione nell’impaginazione

7. Con la combinazione di tasti ‘C-c C-e #’ o il comando ‘M-x org-export-insert-default-template’ si inseriscono tutte le opzioni nel documento. Scegliendo ‘default’ si inseriscono le opzioni comuni a tutti i *back-end* e con ‘latex’, ‘html’, ecc. quelle relative a LaTeX, HTML e così via.
8. La tavola dei contenuti è impaginata dopo il titolo o nel punto in cui è inserita una linea composta unicamente dalla stringa ‘[TABLE-OF-CONTENTS]’.
9. Attenzione: non è un flag booleano ma va impostato come ‘^:{}’, in questo caso ‘a_{b}’ viene interpretato, mentre ‘a_b’ no.

6. Attributi degli elementi

Le direttive e le opzioni fin qui viste si applicano all'intero processo di esportazione e determinano come si forma l'impaginato. Normalmente *ORG-mode* esegue un'impaginazione *standard* degli elementi del documento LaTeX, che però non sempre è sufficiente agli scopi dell'autore. È necessario fornire delle specifiche particolari ai vari elementi da impaginare per renderli più rispondenti alle proprie necessità. Si possono fornire degli *attributi* ai singoli elementi da impaginare (come immagini, tabelle, ecc). Gli attributi sono definiti in ORG con una direttiva `'#+ATTR_<backend>:'`. Per LaTeX è `'#+ATTR_LATEX:'`.

Per lo stesso elemento si possono indicare più direttive di attributi, relative anche a differenti *back-end*, e saranno usate nelle relative esportazioni. In ogni direttiva `'#+ATTR_...'`, possono essere inclusi molti attributi, definiti nel formato `':<attributo> <valore>'`. Gli attributi degli elementi non vanno confusi cone le opzioni del documento definite nella direttiva `'#+OPTIONS:'` e definite come `'<opzione>:<valore>'`.

Per fare un solo esempio, se si vuole introdurre un'immagine nel flusso del testo è sufficiente usare l'espressione:

```
[[./img/arstexnica.png]]
```

si otterrà nel testo impaginato la seguente immagine, estesa alla dimensione della pagina:

È però anche possibile definire degli attributi per l'immagine, come ad esempio la dimensione o un eventuale angolo di rotazione, in questo modo:

```
#+ATTR_LATEX: :width 4cm :options angle=45
[[./img/arstexnica.png]]
```

e si otterrà questo:

Si può anche inserire l'immagine in un blocco galleggiante (*floating*) e fornirgli una didascalia con il comando generale valido per tutti i formati di esportazione `'#+CAPTION:'` come in:

```
#+CAPTION: Il logo di ArsTeXnica
#+ATTR_LATEX: :width 4cm
[[./img/lion.png]]
```

Figura 1. Il logo di ArsTEXnica

Molti elementi del markup ORG hanno attributi specifici, come quelli appena visti per le immagini. Uno dei più importanti è `':environment'` che seleziona l'ambiente da applicare al successivo elemento. Ad esempio:

```
#+ATTR_LATEX: :environment myverbatim
#+BEGIN_EXAMPLE
«  Lorem ipsum...
#+END_EXAMPLE
```

Applicherà al blocco di esempio l'environment myverbatim:

```
\begin{myverbatim}
« Lorem ipsum...
\end{myverbatim}
```

anche se in questo caso sarebbe ancora più opportuno usare la possibilità di indicare il blocco proprio con il nome dell'ambiente:

```
#+BEGIN_myverbatim
« Lorem ipsum...
#+END_myverbatim
```

Usando gli attributi è possibile specificare i dettagli della composizione tipografica senza ricorrere direttamente ai comandi LaTeX.

7. Collegamenti esterni, interni e identificazione degli elementi

Un URI, riportato testualmente nel documento, o incluso in parentesi angolari (`'<LINK>'`) ovvero in doppie parentesi quadre (`'[[LINK]]'`), viene impaginato come collegamento attivo in LaTeX (es. `http://orgmode.org`). Il formato generico di un hyperlink è:

```
[[LINK][DESCRIZIONE]]
```

In questo caso, la descrizione sarà impaginata come testo, e il link permetterà di accedere alla risorsa. Esistono precise regole per rappresentare correttamente, all'interno dei link, alcuni elementi che necessitano del carattere di escape (`'\'`), come ad esempio le parentesi quadre, e il carattere di escape stesso. Il modo migliore di inserire e modificare un URI nel testo è usare la combinazione di tasti messi a disposizione da Emacs (`'C-c C-l'`) che si occupa automaticamente di quest'aspetto.

Oltre ai collegamenti esterni, in tutti gli schemi usuali (‘`http`’, ‘`ftp`’, ‘`email`’, ecc.), *ORG-mode* gestisce anche collegamenti *interni al documento*, come i riferimenti a figure e tabelle, alle sezioni strutturali e addirittura ai singoli punti di una elencazione.

Un elemento dell’articolo, come tabella o figura, può essere identificato con la direttiva ‘`#+NAME:`’ messa prima dell’elemento. Con tale nome può essere richiamata nei riferimenti interni del documento, ad esempio:

```
#+NAME: Tav1
| Col1 | Col2 |
|------+------|
| A    | B    |

Come indicato nella *Tavola [[Tav1]]*.
```

che sarà impaginato così:

Col1	Col2
A	B

Come indicato nella **Tavola 7**.

È possibile anche fare riferimento ai singoli punti di una elencazione numerata:

```
1. primo punto
2. secondo punto
3. <<p3>>terzo punto

Come indicato nel punto [[p3]].
```

che produce:

1. primo punto
2. secondo punto
3. terzo punto

Come indicato nel punto 3.

È infine anche possibile collegarsi ad un titolo di sezione con la notazione ‘`*Paragrafo [[*Matematica]]*`’ (ottenendo **Paragrafo 10**). L’opzione di numerazione (‘`num:t`’) deve essere attiva.

8. Alcuni tipi di blocco importanti

Esistono alcuni tipi di blocchi specializzati delimitati dalle direttive ‘`#+BEGIN_<tipo>`’ e ‘`#+END_<tipo>`’.

Nella linea di apertura del blocco possono anche essere definite delle opzioni che specificano il comportamente che *ORG-mode* deve usare nella gestione del blocco.

Ci sono blocchi di tipo:

'abstract' il blocco `abstract` gioca un ruolo speciale perché viene composto come sommario del documento (*abstract*), secondo le regole della classe usata nel documento LaTeX. Ad esempio il blocco `abstract` del presente articolo è:

```
#+BEGIN_ABSTRACT
Nella stesura degli articoli [...]
#+END_ABSTRACT
```

'center' blocco per centrare il contenuto nella pagina;

'export' blocco particolarmente importante, la chiave di specifica indica un sistema di *back-end* (`html`, `latex`, ecc.). Il contenuto sarà incluso solo nei file esportati per quel *back-end*. Quindi per trasferire un pezzo di codice LaTeX dal file '`.org`' a quello '`.tex`' basta usare:

```
#+BEGIN_EXPORT latex
 <codice LaTeX> ...
#+END_EXPORT
```

'example' blocco che presenta un esempio rappresentato in verbatim;

```
#+BEGIN_EXAMPLE
| Intestazione 1 | Intestazione 2 | Intestazione 3 |
|----------------+----------------+----------------|
| Cella 11       | Cella 12       | Cella 13       |
| Cella 21       | Cella 22       | Cella 23       |
| Cella 31       | Cella 32       | Cella 33       |
#+END_EXAMPLE
```

'quote' blocco per riportare citazioni testuale fuori linea;

'src' blocco che include codice sorgente nei più comuni linguaggi di programmazione esplicitati nella prima opzione della direttiva `begin`. Un aspetto rilevante dei documenti `ORG` è che questi blocchi di codice non servono solo per la rappresentazione di un esempio (per i quali esiste anche il tipo di blocco `example`), ma possono essere resi *attivi*, quindi *ORG-mode* con il supporto dei sistemi esterni o interni di interpretazione dei linguaggi può realmente eseguire il codice del blocco e riportare all'interno del documento il risultato dell'elaborazione con la direttiva '`#+RESULT:`' apposta subito dopo il blocco `src`. In alternativa il codice può essere esportato in file sorgente esterni, e successivamente interpretato o compilato. Questo permette a un documento '`.org`' di diventare un *notebook* di *literate programming* attivo alla stregua dei classici strumenti tangle e weave, con cui lo stesso compilatore TeX fu scritto da Donald Knuth (Knuth, 1984; Knuth e Levy, 1993), o i più moderni progetti di *notebook interface* come il progetto Jupyter (Fruchart *et al.*, 2022). Per la rappresentazione in LaTeX del blocco `src` si usa l'ambiente verbatim, ma è anche possibile scegliere anche la rappresentazione con listings o minted configurando la variabile Emacs '`org-latex-listings`'. Un esempio è:

```
#+BEGIN_SRC python
 def fibonacci(n):
     if n <= 1:
         return n
     else:
         return fibonacci(n-1) + fibonacci(n-2)
#+END_SRC
```

'verse' blocco per riportare poesie e versi, come:

```
#+BEGIN_VERSE
There once was a student of science,
Whose essays casted off in decadence
She then used Org to write,
And LaTeX to impaginate,
So her colleagues admired in silence.
#+END_VERSE
```

9. LaTeX dentro il markup ORG

ORG è un formato che tende a rappresentare, inalterato, il contenuto testuale, eccezion fatta per gli elementi di markup posti nelle posizioni o nelle combinazioni previste. Tutto ciò che non è riconosciuto come markup viene quindi comunque trasferito, tal quale, al file esportato.

È possibile includere elementi specifici del sistema di impaginazione sottostante, che sarà poi interpretato dal *back-end*. Per fare un esempio, sarà possibile includere un qualsiasi comando LaTeX, diciamo '`\raggedright`', per ottenere il comportamento LaTeX desiderato (posto che l'opzione '`TeX:t`' sia definita).

Lavorando con un unico *back-end*, non avendo quindi necessità di esportare il risultato in qualche altro formato, non c'è bisogno di usare il blocco `export`. Se invece si vuole sfruttare la capacità di *ORG-mode* di esportare il contenuto in molteplici formati è necessario delimitare l'esportazione al solo *back-end* desiderato, usando i blocchi `export` quando necessario.

Se si devono inserire pezzi più brevi si può usare un'espressione in linea inclusa dentro una doppia coppia di simboli '`@`' seguita da '`latex:`' come in '`@@latex: codice arbitrario LaTeX@@`'. Questo codice sarà inserito solo nell'esportazione per il *back-end* selezionato. Allo stesso modo si può fare in altri *back-end* come:

```
...questo documento è impaginato con @@latex: \LaTeX{}@@@@html: HTML@@
   e non in @@latex: HTML@@@@html: \LaTeX{}@@
```

Inserendo questa frase nel presente articolo, quando il lettore avrà di fronte l'impaginato in PDF saprà che questo documento è impaginato con LaTeX e non in HTML, se lo vedesse su una pagina web, esportata direttamente da *ORG-mode* in HTML, leggerebbe il contrario.

Se non è necessario un intero blocco e basta una singola linea, si può usare con la direttiva '`#+LATEX:`' in questo modo:

```
#+LATEX: \newpage
```

10. Matematica

Uno dei punti di forza che rendono giustamente orgogliosi i sostenitori di LaTeX è l'avanzata capacità di comporre documenti con espressioni matematiche, sia in linea che in evidenza. Da questo punto di vista, ORG usa l'approccio più lineare: la notazione LaTeX per la matematica. Quindi dentro il sorgente '`.org`' un'espressione come la seguente:

```
\[ \hat{y} = \hat{\beta}_{0}+\hat{\beta}_{1}x_{1}+
\hat{\beta}_{2}x_{2}+... +\hat{\beta}_{p}x_{p}  \]
```

produce questo:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + ... + \hat{\beta}_p x_p$$

Se si vuole ottenere un'equazione numerata sarà sufficiente usare un blocco `equation` (o qualsiasi altro *environment* solitamente utilizzato in LaTeX, come displaymath, eqnarray, ecc.

Anche per la matematica in linea è possibile utilizzare semplicemente la notazione LaTeX, come ad esempio `'\( x^{2+y} \)'` per ottenere x^{2+y}. Da questo punto di vista quindi *ORG-mode* non aggiunge e non sottrae nulla a LaTeX[10].

Va senza dire che laddove il *back-end* lo prevede, verrà composta correttamente la matematica anche nell'esportazione in formati differenti da LaTeX, ad esempio HTML come si può constatare usando la combinazione di tasti `'C-c C-e h O'`.

11. Tabelle

Le tabelle sono state già brevemente introdotte in precedenza, è necessario solo richiamare solo alcuni attributi come:

`':environment'` specifica l'ambiente per il *back-end* LaTeX (come tabularx, longtable, array, ecc.);

`':float'` impagina la tabella come galleggiante, come `'sideways'` o `'multicolum'` per ottenerla rispettivamente ruotata o su più colonne;

`':mode'` modifica il modo con cui la tabella è esportata, può assumere i valore di `'table'`, `'math'`, `'inline-math'`, `'verbatim'` o `'tabbing'`;

`':placement'` posizionamento della tabella (che assume i valori `h t b p ! H` proprio come in LaTeX).

Altre opzioni riguardano l'esportazione in ambiente matematico (`':math-prefix'`, `':math-suffix'` e `':math-arguments'`), l'ampiezza (`':spread'` e `':width'`) e gli switch come `':booktabs'` che migliorano la gestione dell'impaginazione, `':center'` per il posizionamento centrato e `':rmlines'` per rimovere tutte le linee tranne la prima per le tabelle impaginate nel formato previsto da `'table.el'`.

Questo formato di tabella, denominato `'table.el'`, è un formato alternativo di Emacs un po' meno parsimonioso di quello standard `ORG` perché prevede di indicare tutti gli incroci delle celle ma che permette di comporre tabelle in modo più complesso del sistema semplificato di *ORG-mode*. Ad esempio, una tabella con celle unite, come questa:

A		B
10	11	
20	21	22

10. Si ricorda che l'utilizzo della notazione TeX, `'$...$'` o `'$$...$$'`, per le equazioni, pur possibile, è sconsigliato in LaTeX e, a maggior ragione, in *ORG-mode*.

viene composta nel file testuale come:

```
+-----+-----+-----+
| A         | B   |
+-----+-----+     |
| 10  | 11  |     |
+-----+-----+-----+
| 20  | 21  | 22  |
+-----+-----+-----+
```

12. Note a pié pagina

ORG-mode permette due tipi di espressioni per le note.

1. Nel primo caso si esprime un'etichetta in linea nel formato `'[fn:NOME]'` e la nota stessa sarà espressa come paragrafo autonomo, solitamente a fine file o della sezione considerata, in formato `'[fn:NOME] Testo della nota.'`[11]. La definizione deve iniziare con la prima parentesi quadra nella prima colonna della riga, senza spazi precedenti, e termina all'inizio di una nuova nota o di una sezione o con due righe vuote.

2. Il secondo caso permette l'inserimento di una nota in linea con il formato `'[fn::Testo della nota in linea...]'`, si noti il doppio `':'`. Si può usare anche la notazione `'[fn:NOME: Testo in linea]'` se è necessario riferirsi alla nota anche in un altra posizione, come è avvenuto nel presente articolo per la nota riportata alla fine di questa frase, usata due volte già altrove.[2]

La numerazione delle note avviene in automatico.

Se invece di note a piè pagina si vogliono ottenere note a fine documento si possono introdurre le seguenti direttive LaTeX dove si vuole che compaiano le note, al termine del documento:

```
#+LATEX_HEADER: \usepackage{endnotes}
#+LATEX_HEADER: \renewcommand{\footnote}{\endnote}
#+LATEX: \theendnotes
```

13. Bibliografie e riferimenti

L'espressione dei collegamenti e la redazione di elenchi di riferimenti bibliografici è uno dei compiti fondamentali nella scrittura di articoli scientifici e per lungo tempo *ORG-mode* non ha avuto un proprio sistema *standard* per esprimerli. Da qualche anno questa situazione è cambiata in una maniera, però, che potrebbe non soddisfare del tutto l'autore abituato a LaTeX.

Il modo più *moderno* di indicare le citazioni bibliografiche è affidato al sottosistema `org-cite` (presente nella libreria `'oc.el'`), già incluso nelle moderne distribuzioni di *ORG-mode*.

`Org-cite'` si basa sulla definizione di *processori delle citazioni*, il cui compito è offrire diverse funzionalità tra cui:

11. Testo della nota.

- colorazione dei collegamenti e presentazione dei tooltip al passaggio del mouse;
- azioni al click dell'utente nell'editor;
- inserimento e modifica delle citazioni;
- esportazione verso formati differenti.

Il meccanismo delle citazioni di *ORG-mode*, che è stato pensato per essere *agnostico al back-end*, introduce terna composta da

```
(processore stile-bibliografico stile-di-citazione)
```

Lo *stile di citazione* si basa su una stringa basata su un modello uniforme composto come segue:

```
[cite/s/v:@key1;@key2;...]
```

dove i due *specificatori* `'/s'` e `'/v'` sono opzionali e rappresentano gli indicatori di *stile* e di *variante* della citazione. Possono essere usati per selezionare una citazione senza autore (`'na'` per `'noauthor'`), cioè solo con la data, o in una variante con le maiuscole iniziali (`'cf'` per `'caps-full'`). Equivalgono ai tanti comandi di citazione presenti in biblatex (come `\cite`, `\citetitle`, `\citeyear`, ecc.).

Invece `'@key1'`, `'@key2'` indicano le *chiavi di citazione*, che possono essere una sola (nel qual caso il punto e virgola finale non è necessario) o molte, separate da punti e virgola. Ogni chiave di citazione è composta dal carattere `'@'` seguito dall'etichetta del riferimento usata nel file `'.bib'`, anticipata da un eventuale prefisso, che sarà impaginato prima del riferimento e seguita prima da un *localizzatore*, ovvero un indicatore di pagina, di capitolo o di volume o altri elementi identificabili nel riferimento e, dopo uno spazio, un suffisso. Questi elementi sono opzionali. Un esempio generico di chiave di citazione è quindi composta nel modo seguente:

```
prefisso @key1 localizzatore suffisso
```

All'inizio dell'intera stringa di citazione potrebbe esserci un prefisso senza chiave di citazione, che ha il ruolo di prefisso comune a tutte le citazioni. Allo stesso modo potrebbe esserci anche un suffisso comune a tutte le citazioni. In definitiva una citazione espressa come:

```
[citep/text/full:Vedi;@hegel1807fenomenologia pp. 184-6]
ovvero
[citep:@hegel1807fenomenologia]
```

potrebbe essere composta come:

> (Vedi Hegel, La Fenomenologia dello Spirito (1807) pp. 184-6)
> ovvero
> (Hegel, 1807)

I processori attualmente presenti sono quattro, ma i due effettivamente utili sono:

`'csl'` per l'esportazione testuale delle citazioni in tutti i *back-end*;

`'bibtex'` solo per LaTeX (ne esiste uno `'natbib'` equivalente).

Il processore ‘bibtex’ usa LaTeX, per cui ‘[cite:@hegel1807fenomenologia pp. 184-6]’ diventa ‘\cite[184-6]{hegel1807fenomenologia}’. In questo caso LaTeX userà anche bibtex (o biber a scelta dell’utente) per generare le citazioni. Il rovescio della medaglia nell’uso di questo processore è che *non* può essere usato per impaginare in formati differenti da LaTeX.

Per avere un documento ‘.org’ universale bisogna ricorrere al processore ‘csl’ (Citation Style Language). CSL è un linguaggio di specifica delle citazioni e delle bibliografie basato su XML ed è usato da alcuni noti programmi di gestione dei riferimenti bibliografici come Zotero, Mendeley o Papers e fu sviluppato inizialmente all’interno dell’OpenOffice Bibliographic Project.

La scelta di uno o l’altro approccio si basa sulla volontà di affidarsi o meno a bibtex per la gestione delle citazioni in LaTeX. Se si usa csl, sia le citazioni che i riferimenti bibliografici saranno composti in modo puramente testuale, secondo lo standard bibliografico scelto: non ci saranno nel file né comandi di citazione \cite, né un \printbibliography che legga il file creato da bibtex.

Chi sa *guidare* bene bibtex, specie nel selezionare le varianti citazionali o se deve gestire campi meno noti del file ‘.bib’ (come ad esempio i campi ‘crossref’ o ‘related’, usate per indicare volumi correlati o traduzioni in lingua), sa quanto sia potente affidarsi al sistema citazione di biblatex. L’approccio agnostico al *back-end* di csl potrebbe quindi non risultare adeguato ai più raffinati cultori di LaTeX.

Va anche citato un package alternativo, creato da John Kitchin, servito originariamente da modello per org-cite, denominato org-ref. Pur essendo un package da installare tramite ‘M-x package-list-package’, è ancor oggi il sistema più utilizzato per la composizione delle citazioni e bibliografie in LaTeX.

Org-ref va confrontato con il processore bibtex (o natbib) di org-cite ma contiene molte funzionalità in più che non hanno trovato ancora posto in org-cite, e forse non lo troveranno mai. È in generale più raffinato nel rapporto con LaTeX. Purtroppo, il tentativo di Kitchin di integrare org-ref in org-cite, con un progetto chiamato org-ref-cite, sembra essere ormai ad un punto morto. Invece è stata recentemente rilasciata una nuova, ancora più funzionale, versione 3.0 di org-ref.

La sintassi per le citazioni di Org-ref è la stessa che usa org-cite, tranne per il carattere ‘&’ nella chiave di citazione invece di ‘@’.

Perdendo quindi ogni compatibilità con *back-end* differenti da quello LaTeX, org-ref è la più efficace scelta per la composizione dei riferimenti e delle bibliografie di *ORG-mode*. D’altro canto, org-cite migliorerà nel tempo, pur avendo dei limiti strutturali che Kitchin considera invalicabili. In prospettiva adottare org-cite potrebbe creare documenti più manutenibili nel tempo.

Comunque, org-cite fa riferimento ad una specifica direttiva ‘#+BIBLIOGRAPHY:’ che indica il file bibliografico da tenere in considerazione, invece org-ref risolve il problema semplicemente aggiungendo le opportune righe in testa al file LaTeX, come:

```
#+LATEX_HEADER: \usepackage[citestyle=authoryear-icomp,bibstyle=authoryear, hyperref=true,
#+LATEX_HEADER:             backref=true,maxcitenames=3,url=true,backend=biber,natbib=true] {biblatex}
#+LATEX_HEADER: \addbibresource{tests/test-1.bib}
```

e infine un \printbibliography al termine.

14. Conclusioni: perché usare o non usare *ORG-mode*

Al termine di questa presentazione introduttiva ad *ORG-mode*, si può tracciare un bilancio sulle motivazioni che possono portare ad usare o non usare il markup `ORG` al posto di quello LaTeX.

L'uso di *ORG-mode* è vantaggioso per i seguenti motivi:

1. Il formato testuale più snello e pulito di ORG, aiutano la leggibilità del sorgente, l'integrazione in un *workflow* di pubblicazione meno oneroso e una maggiore semplicità di conservazione (Schöch, 2014);

2. La possibilità di creare documenti esportabili in differenti formati, permette la pubblicazione contemporanea su differenti media nonché una più semplice collaborazione con coautori che utilizzino formati differenti;

3. La presenza di blocchi sorgente attivi, che possono produrre risultati in modo programmatico, rende possibile un approccio di ricerca orientato alla *Open Science*, e alla produzione di risultati scientifici riproducibili (Leha e Beissbarth, 2011; Schulte e Davison, 2011; Stanisic *et al.*, 2015). La soluzione di *ORG-mode* è notevolmente più semplice di quelle possibili direttamente con LaTeX (Bar e Wang, 2021).

D'altro canto vi sono anche alcune chiare controindicazioni:

1. Nessuna rivista o editore accetta il formato nativo `'.org'`, la consegna dell'articolo dovrà essere fatta necessariamente in un formato differente (LaTeX o OpenOffice).

2. Se un autore alle prime armi con LaTeX può trovare vantaggioso imparare direttamente il markup `ORG` e contemporaneamente l'utilizzo di Emacs, ottenendo così in un sol colpo un ambiente di ricerca e pubblicazione, peraltro di ottima qualità, un autore già molto competente con LaTeX, con eventualmente già una propria catena di strumenti di sviluppo rodata, potrebbe considerare superfluo, o addirittura controproducente, l'impegno per ottenere qualche modesto vantaggio, dal suo punto di vista.

3. *ORG-mode* di fatto obbliga ad usare Emacs come editor, che non necessariamente potrebbe essere apprezzato.

Il dato di fatto, che un po' quest'articolo vuole dimostrare nei fatti, è che *se si vuole* usare Emacs e *ORG-mode* per consegnare un articolo in LaTeX, seguendo le linee guida di una rivista o di un editore è *sempre* possibile farlo. Questo è vero per LaTeX, che è l'oggetto di quest'articolo, ma anche se si deve consegnare in uno dei formati per *word-processor* (`docx`, `odt`, ecc.).

ORG-mode rappresenta un approccio aggiuntivo, integrativo e non sostitutivo di LaTeX, di cui si può far tesoro nel propria cassetta degli attrezzi da ricercatore.

Di una cosa si può sicuri: c'è da scommettere sulla longevità di questo progetto, come è stato per LaTeX o Emacs.

A. Usare ORG-mode con ArsTeXnica

Per poter consegnare ad ArsTeXnica quest'articolo redatto in *ORG-mode*, è stato necessario attenersi ai requisiti di redazione, in particolare adottare la classe arstexnica e rispettare le linee guida riportate nel kit (GuIT, 2022).

È una di quelle cose che sono più facili da fare che da documentare.

A.1. Classe arstexnica

La prima cosa da fare è stato quindi definire arstexnica come classe da usare nell'esportazione indicando la direttiva '`#+LATEX_CLASS:`' nel file '`.org`':

```
#+LATEX_CLASS: arstexnica
```

Ciò non è però sufficiente. Bisogna configurare opportunamente '`org_latex_classes`' per aggiungere la classe `arstexnica`, ovvero una terna:

```
(nome-classe inizio-del-preambolo elementi-di-segmentazione)
```

Il comando Lisp da inserire in '`~/.emacs`' è quindi:

```
(with-eval-after-load 'ox-latex
   (add-to-list 'org-latex-classes
                '("arstexnica"
                  "\\documentclass{arstexnica}"
                  ("\\section{%s}" . "\\section{%s}")
                  ("\\subsection{%s}" . "\\subsection{%s}")
                  ("\\subsubsection{%s}" . "\\subsubsection{%s}")
                  ("\\paragraph{%s}" . "\\paragraph*{%s}")
                  ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))))
```

Già a questo punto, un'esportazione da `.org` permette una buona visualizzazione dell'articolo.

Il file prodotto però non rispecchia esattamente quello dell'esempio riportato nel kit con il nome `name.tex`.

Il preambolo dell'articolo generato da *ORG-mode* è differente da quello in ArsTeXnica. Per renderlo uguale è necessario configurare alcune variabili di Emacs.

Il modello principale di un articolo ArsTeXnica, contenuto nel file `name.tex` del kit, ha una fondamentale differenza con il modello di *ORG-mode*, permette la compilazione con differenti motori di impaginazione per LaTeX, specificamente pdfLaTeX, XeLaTeX o LuaLaTeX.

Il blocco di codice che specifica questo comportamento è il seguente:

```
 \ifbool{PDFTeX}{%                                   pdfLaTeX
    \usepackage[T1]{fontenc}
    \usepackage[utf8]{inputenc}
    \usepackage[english, italian]{babel}
  }{\ifbool{XeTeX}{%                                 XeLaTeX
    \usepackage{polyglossia}
    \setmainlanguage[babelshorthands]{italian}
    \PolyglossiaSetup{italian}{indentfirst=false}
    \setotherlanguage{english}
    }{\ifbool{LuaTeX}{%                              LuaLaTeX
      \usepackage{polyglossia}
      \setmainlanguage[babelshorthands]{italian}
      \PolyglossiaSetup{italian}{indentfirst=false}
      \setotherlanguage{english}
      }{%
      \ArsTeXnicaError\endinput
      }
    }
 }
```

```
20
21  \usepackage{cochineal}
22  \ifbool{PDFTeX}{%
23      \usepackage[varqu,varl,var0]{inconsolata}
24    }{%
25      \fontspec[StylisticSet={1,2,3},Scale=MatchLowercase]{inconsolata}
26  }
27
28  \usepackage[scale=.9,type1]{cabin}
29  \usepackage[cochineal,vvarbb]{newtxmath}
30  \usepackage[cal=boondoxo]{mathalfa}
31
32  \usepackage{microtype}
33  \usepackage{natbib}
34  \usepackage{graphicx}
35  \usepackage{hyperref}
```

Listing 1. Il preambolo del file di consegna di un articolo per ArsTEXnica presente nel kit per gli autori

Nei due blocchi alle linee 1–19 e 21–26 si danno tre casi:

1. Se il motore è pdfLaTeX i package utilizzati sono fontenc, inputenc e per la gestione della lingua babel nonché, come specificato nella seconda parte il font a passo fisso inconsolata.

2. Se il motore è XeTeX ovvero LuaTeX, non sono caricati i package fontenc e inputenc e per la gestione linguistica viene usato polyglossia, inoltre il font inconsolata è dato per già caricato e se ne specificano le caratteristiche con il comando `\fontspec`.

3. Nel caso che il motore non sia nessuno dei tre si emette un `\ArsTeXnicaError` e si blocca l'impaginazione.

Nella parte successiva del preambolo (linee 28–35) vi sono altri package di cui solo graphicx e hyperref sono già presenti nel preambolo generato da *ORG-mode*.

Il codice prodotto da *ORG-mode* non gestisce questa alternativa tra i motori di impaginazione. Si possono possono fare due scelte:

1. Dando per scontato che la rivista utilizzi solo pdfLaTeX per l'impaginazione, si lasciano le configurazioni di *ORG-mode* come sono e si inseriscono i pacchetti mancanti (cochineal, inconsolata, cabin, newtxmath, mathalfa, microtype enatbib) in apposite direttive `'#+LATEX_HEADER:'` in testa al file `'.org'`. È la scelta più semplice.

2. Provare a replicare completamente il file `name.txt`

Si seguirà, ovviamente, la seconda scelta. Per giungere al risultato aggiungere nella configurazione globale della traduzione della classe arstexnica tutto il preambolo del file principale di ArsTEXnica. disabilitando i package di default e aggiuntivi presenti in *ORG-mode*, ma non quelli `[EXTRA]`.

Oltre a ciò va gestito, direttamente all'interno del file `'.org'`, sia la questione del doppio abstract, in lingua inglese, che la definizione dello stile bibliografico relativo a *natbib*.

Infine vanno risolti alcuni dettagli, come la corretta impaginazione del nome ArsTEXnica e l'utilizzo di alcune macro utili.

In definitiva la ridefinizione della variabile `'org-latex-classes'` diventa solamente un po' più complessa:

```
1          (with-eval-after-load 'ox-latex
2             (add-to-list 'org-latex-classes
3                          '("arstexnica"
4                            "\\documentclass{arstexnica}
5  [NO-DEFAULT-PACKAGES]
6  [NO-PACKAGES]
7    % Il codice che segue rende possibile usare uno qualsiasi tra i
8    % principali motori di tipocomposizione pdfLaTeX, XeLaTeX, LuaLaTeX.
9    % Si prega di non modificare il preambolo se non è strettamente
10   % necessario.  È consentito aggiungere altre lingue e font particolari
11   % ove l'argomento dell'articolo lo richieda. Altre personalizzazioni
12   % vanno fatte nei file `\\jobname-package.tex' e
13   % `\\jobname-command.tex'.
14   %
15   % The following code allows the use of any typesetting
16   % engine among pdfLaTeX, XeLaTeX, or LuaLaTeX.
17   % Please, don't change the preamble unless it is strictly necessary.
18   % You can add other languages or fonts if it is required by
19   % the subject of the paper. Other customisations should be added to
20   % the files `\\jobname-package.tex' and `\\jobname-command.tex'.
21   %
22   \\ifbool{PDFTeX}{%                                    pdfLaTeX
23       \\usepackage[T1]{fontenc}
24       \\usepackage[utf8]{inputenc}
25       \\usepackage[english, italian]{babel}
26     }{\\ifbool{XeTeX}{%                                 XeLaTeX
27       \\usepackage{polyglossia}
28       \\setmainlanguage[babelshorthands]{italian}
29       \\PolyglossiaSetup{italian}{indentfirst=false}
30       \\setotherlanguage{english}
31       }{\\ifbool{LuaTeX}{%                              LuaLaTeX
32         \\usepackage{polyglossia}
33         \\setmainlanguage[babelshorthands]{italian}
34         \\PolyglossiaSetup{italian}{indentfirst=false}
35         \\setotherlanguage{english}
36         }{%
37         \\ArsTeXnicaError\\endinput
38         }
39       }
40   }
41
42   \\usepackage{cochineal}
43   \\ifbool{PDFTeX}{%
44       \\usepackage[varqu,varl,var0]{inconsolata}
45     }{%
46       \\fontspec[StylisticSet={1,2,3},Scale=MatchLowercase]{inconsolata}
47   }
48   \\usepackage[scale=.9,type1]{cabin}
49   \\usepackage[cochineal,vvarbb]{newtxmath}
50   \\usepackage[cal=boondoxo]{mathalfa}
51
52   \\usepackage{microtype}
53   \\usepackage{natbib}
54   \\usepackage{graphicx}
55   \\usepackage{hyperref}
56   [EXTRA]
57    "
58                            ("\\section{%s}" . "\\section{%s}")
59                            ("\\subsection{%s}" . "\\subsection{%s}")
60                            ("\\subsubsection{%s}" . "\\subsubsection{%s}")
61                            ("\\paragraph{%s}" . "\\paragraph*{%s}")
62                            ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))))
```

In pratica, si è spostata qui la prima parte del file `name.tex`, disabilitando al contempo l'aggiunta dei package di default alle linee 5 e 6. Ora, la generazione del file `.tex` da parte di *ORG-mode*, per la classe arstexnica, replicherà esattamente il preambolo nel file `name.tex`, a meno di quello che si dirà più oltre.

A.2. Doppio abstract

ArsTEXnica ha un doppio abstract per gestire la presentazione in due lingue, italiano e inglese. Una volta definito il primo abstract si possono usare due strategie, direttamente all'interno del file `.org`, la prima prevede di inserire tutto il necessario per il secondo abstract in lingua inglese in un blocco di esportazione specifico per il LaTeX.

```
#+BEGIN_EXPORT latex
  \begin{otherlanguage}{} % Second language
  \begin{abstract}
  In writing academic...
  \end{abstract}
  \end{otherlanguage}
#+END_EXPORT
```

La principale controindicazione di questa strategia è che il testo inserito nell'abstract non sarà processato da *ORG-mode* ma copiato tal quale, quindi eventuali markup non saranno trasformati, ad esempio le enfasi o la rappresentazione del nome LaTeX, e dovranno essere espressi direttamente in markup LaTeX.

L'alternativa è quella di non usare un blocco di esportazione, ma includere il secondo blocco di abstract tra due linee di comando per LaTeX che aprono e chiudono l'ambiente otherlanguage, in questo modo:

```
#+LATEX: \begin{otherlanguage}{english}
#+BEGIN_ABSTRACT
In writing academic ...
#+END_ABSTRACT
#+LATEX: \end{otherlanguage}
```

In questo caso il contenuto del secondo blocco `abstract` godrà anche della trasformazione del markup.

A.3. Macro e ArsTEXnica, il logo

La corretta impaginazione del logo di ArsTEXnica è un dettaglio che non si poteva proprio trascurare. È stata anche la cosa più difficile da fare.

Una soluzione più facile avrebbe comportato l'uso del comando LaTeX direttamente nel sorgente `.org`, oppure l'uso della direttiva `'#+MACRO:'`, ma questo avrebbe voluto dire dover scrivere nel testo `'\ArsTeXnica{}'` nel primo caso o `'{{{ArsTeXnica()}}}'` nel secondo. Orribile!

Per quelle macro che « si usano spesso quando si parla del sistema TEX » (GuIT, 2022, p. 4) come `\pkgname`, `\clsname`, `\optname`, e via dicendo si possono anche definire della direttive `'#+MACRO:'` in *ORG-mode*, in questo modo:

```
#+MACRO: envname @@latex:\envname{$1}@@
```

e poi usarle con `'{{{envname(verbatim)}}}'`.

Quest'approccio avrebbe senso se si avesse come obiettivo quello di ottenere un documento impaginabile anche in altri *back-end* oltre al LaTeX, nel qual caso la macro `ORG` si dovrebbe cambiare inserendo alternative per ogni *back-end*, ad esempio per poter gestire anche HTML, si potrebbe fare una cosa del genere:

```
#+MACRO: envname @@latex:\envname{$1}@@@@html:<b class="envname">$1</b>@@
```

e poi definire opportunamente un `css` in HTML per la classe `envname`, eventualmente con una modalità di rappresentazione comune per tutti i nomi degli ambienti.

Solo in questo caso l'aumentata complessità nell'uso della citazione sarebbe giustificata.

Il presente articolo è destinato solo per la consegna in formato LaTeX ad ArsTeXnica, quindi la scelta migliore è utilizzare direttamente le macro LaTeX nel testo `.org`. Quindi per ottenere il nome di un ambiente come verbatim, si scriverebbe semplicemente `'\envname{verbatim}'`, proprio come in LaTeX.

Ma per il nome ArsTeXnica, si adotterà un approccio differente, anche solo per dimostrare perché l'approccio a codice aperto e la programmazione in Lisp, rendono Emacs un editor di livello superiore a tutti gli altri, adattabile alle necessità dell'utente fin dentro i suoi intimi dettagli (Lewis *et al.*, 1997; Glickstein, 1997; Monnier e Sperber, 2020).

Si può notare che le stringhe testuali `'LaTeX'` e `'TeX'` presenti nel file '.org' sono automaticamente trasformate dal *back-end* LaTeX nel loro equivalente comando LaTeX, ovvero `'\LaTeX{}'` e `'\TeX{}'`. Si vuole ottenere lo stesso risultato anche con `'ArsTeXnica'`.

Purtroppo quest'operazione non è immediata o configurabile in qualche modo, perché è compiuta all'interno di una funzione Lisp non personalizzabile. Bisogna intervenire nella programmazione della libreria `'ox-latex.el'`.

Qui ci vengono in soccorso le caratteristiche proprie di *apertura* del linguaggio Lisp, come ambiente interpretato dove una funzione può essere sostituita, semplicemente ridefinendola.

Con un po' di ricerca nell'archivio del codice sorgente delle librerie di *ORG-mode* si scopre che la funzione *incriminata* è `'org-latex-plain-tex'` così definita in `'ox-latex.el'`. La sua definizione è questa:

```
(defun org-latex-plain-text (text info)
  "Transcode a TEXT string from Org to LaTeX.
TEXT is the string to transcode.  INFO is a plist holding
contextual information."
  (let* ((specialp (plist-get info :with-special-strings))
         (output
          ;; Turn LaTeX into \LaTeX{} and TeX into \TeX{}.
          (let ((case-fold-search nil))
            (replace-regexp-in-string
             "\\<\\(?:La\\)?TeX\\>" "\\\\\&{}"
             ;; Protect ^, ~, %, #, &, $, _, { and }.  Also protect \.
             ;; However, if special strings are used, be careful not
             ;; to protect "\" in "\-" constructs.
             (replace-regexp-in-string
              (concat "[%$#&{}_~^]\\|\\\\" (and specialp "\\([^-]\\|$\\)"))
              (lambda (m)
                (cl-case (string-to-char m)
                  (?\\ "$\\\\backslash$\\1")
                  (?~ "\\\\textasciitilde{}")
                  (?^ "\\\\^{}")
                  (t "\\\\\&")))
              text)))))
    ;; Activate smart quotes.  Be sure to provide original TEXT string
    ;; since OUTPUT may have been modified.
    (when (plist-get info :with-smart-quotes)
      (setq output (org-export-activate-smart-quotes output :latex info text)))
    ;; Convert special strings.
    (when specialp
      (setq output (replace-regexp-in-string "\\.\\.\\." "\\\\ldots{}" output)))
    ;; Handle break preservation if required.
    (when (plist-get info :preserve-breaks)
      (setq output (replace-regexp-in-string
                    "\\(?:[ \t]*\\\\\\)?[ \t]*\n"
                    (concat org-latex-line-break-safe "\n")
                    output nil t)))
    ;; Return value.
    output))
```

Questa è una funzione di traduzione che trasforma il testo base dal markup ORG a LaTeX, limitatamente ad alcuni elementi che non possono essere espressi tal quali e, tra le altre cose, traduce anche 'TeX' e 'LaTeX'. Questo è il punto giusto dove far trasformare anche 'ArsTeXnica'.

È necessario modificare l'espressione regolare alla riga 10 del precedente listato dove l'espressione regolare di riconoscimento ("\\<\\(?:La\\)?TeX\\>") cattura la stringa 'TeX', eventualmente preceduta da 'La' e la sostituisce con il pattern di sostituzione ("\\\&{}"), cioè antepone alla stringa riconosciuta uno '\' (slash), a cui fa seguire la stringa catturata nel riconoscimento, che si indica come "\&", seguita infine dalle parentesi graffe aperte e chiuse (cioè, si passa da 'LaTeX' a '\LaTeX{}'). Gli slash del pattern di sostituzione devono essere protetti a loro volta con uno slash quindi il pattern definitivo, presente alla fine della linea 10 nel listato, avrà ben 6 slash di seguito: "\\\\\\&{}". Oltre alle tante parentesi, questi *trenini* di slash sono una delle caratteristiche meno apprezzabili del linguaggio Emacs Lisp.

Per riconoscere anche 'ArsTeXnica', quindi basta modificare l'espressione regolare di riconoscimento, in questo modo:

```
"\\<\\(?:La\\|Ars\\)?TeX\\(nica\\)?\\>"
```

In realtà, a questo punto, si possono aggiungere anche le altre sigle definite nel file 'arsacro.sty' incluso nel kit di ArsTeXnica completando la tabella:

Stringa *ORG-mode*	Formattata con stringa in *ORG-mode* (es 'TeX')	Fomattata con macro LaTeX (es. '\TeX')
'TeX'	TeX	TeX
'LaTeX'	LaTeX	LaTeX
'ArsTeXnica'	ArsTeXnica	ArsTeXnica
'PCTeX'	PC TeX	PC TeX
'pcTeX'	pcTeX	pcTeX
'pdfTeX'	pdfTeX	pdfTeX
'pdfLaTeX'	pdfLaTeX	pdfLaTeX
'PiC'	PiC	PiC
'PiCTeX'	PiCTeX	PiCTeX
'plain'	plain	plain
'SliTeX'	SliTeX	SliTeX

L'espressione regolare finale sarà quindi:

```
"\\<\\(?:La\\|Ars\\|pc\\|PC\\|pdf\\|pdfLa\\|PiC\\|Sli\\)?TeX\\(nica\\)?\\|plain\\|PiC\\>"
```

Il problema è che non c'è un modo per iniettare questa variazione di espressione regolare all'interno della funzione 'org-latex-plain-tex', però si può *completamente sostituire* questa funzione, ridefinendola, con la nuova espressione regolare, e includendola ad esempio nel file di configurazione iniziale di Emacs ('~/.emacs'). Meglio ancora sarebbe definire una *customization variable* di Emacs, con un nome significativo (ad esempio 'org-latex-plain-subs-regexp') ed introdurre nella nuova funzione questa variabile, in modo che possa essere successivamente configurabile dall'utente.

```
(defcustom org-latex-plain-subs-regexp
      "\\<\\(\\(?:La\\|Ars\\|pc\\|PC\\|pdf\\|pdfLa\\|PiC\\|Sli\\)?TeX\\(nica\\)?\\|plain\\|PiC\\)\\>"
```

```
  "Regular expression that recognizes strings to be transformed
   into LaTeX commands."
  :group 'org-export-latex
  :type '(string :tag "Regular Expression")
  :safe #'stringp)
```

La nuova funzione da inserire nel file '`~/.emacs`' è quindi:

```
1  (defun org-latex-plain-text (text info)
2    "Transcode a TEXT string from Org to LaTeX.
3  TEXT is the string to transcode.  INFO is a plist holding
4  contextual information."
5    (let* ((specialp (plist-get info :with-special-strings))
6           (output
7            ;; Turn LaTeX into \LaTeX{} and TeX into \TeX{}.
8            (let ((case-fold-search nil))
9              (replace-regexp-in-string
10              org-latex-plain-subs-regexp
11              "\\\\\\&{}"
12              ;; Protect ^, ~, %, #, &, $, _, { and }.  Also protect \.
13              ;; However, if special strings are used, be careful not
14              ;; to protect "\" in "\-" constructs.
15              (replace-regexp-in-string
16               (concat "[%$#&{}_~^]\\|\\\\" (and specialp "\\([^-]\\|$\\)"))
17               (lambda (m)
18                 (cl-case (string-to-char m)
19                   (?\\ "$\\\\backslash$\\1")
20                   (?~ "\\\\textasciitilde{}")
21                   (?^ "\\\\^{}")
22                   (t "\\\\\\&")))
23               text)))))
24     ;; Activate smart quotes.  Be sure to provide original TEXT string
25     ;; since OUTPUT may have been modified.
26     (when (plist-get info :with-smart-quotes)
27       (setq output (org-export-activate-smart-quotes output
28                                                      :latex info text)))
29     ;; Convert special strings.
30     (when specialp
31       (setq output (replace-regexp-in-string "\\.\\.\\."
32                                              "\\\\ldots{}" output)))
33     ;; Handle break preservation if required.
34     (when (plist-get info :preserve-breaks)
35       (setq output (replace-regexp-in-string
36                     "\\(?:[ \t]*\\\\\\)?[ \t]*\n"
37                     (concat org-latex-line-break-safe "\n")
38                     output nil t)))
39     ;; Return value.
40     output))
```

Rispetto alla precedente è cambiata solo la riga 10, dove non c'è più l'espressione regolare precedente, ma la variabile personalizzata definita in precedenza. A questo punto nel testo '.org' sarà possibile utilizzare '`ArsTeXnica`' per ottenere ArsTeXnica.

L'ultimo problema è che non esiste una macro LaTeX chiamata `\ArsTeXnica` poiché il kit ne defisce solo una chiamata `\Ars`. A questo punto, come consigliato dalle stesse linee guida (GuIT, 2022, p. 5), si può però aggiungere una definizione (o nel preambolo o direttamente nel file '`.org`' con una direttiva '`#+LATEX_HEADER:`'):

```
#+LATEX_HEADER: \let\ArsTeXnica\Ars
```

A.4. Ultimi dettagli: `hyperref` *e ambiente* `article`

Uno dei piccoli problemi generati dall'algoritmo standard di esportazione di *ORG-mode* è dovuto all'automatica creazione della definizione dei metadati del file PDF nel comando

\hypersetup, che invece non vanno definiti per la consegna a ArsTEXnica.

```
\hypersetup{
 pdfauthor={Emmanuele F. Somma},
 pdftitle={Emacs ORG-Mode, LaTeX (e ArsTeXnica)},
 pdfkeywords={Emacs ORG LaTeX impaginazione },
 pdfsubject={Un articolo per ArsTeXnica },
 pdfcreator={Emacs 28.2 (Org mode 9.5.5)},
 pdflang={Italian}}
```

Va quindi posta a ‘nil’ la variabile ‘org-latex-hyperref-template’, cosa che si può fare anche direttamente nel file ‘.org’ utilizzando la direttiva ‘#+BIND:’:

```
#+BIND: org-latex-hyperref-template nil
```

Infine, tutto il testo dell’articolo deve essere racchiuso da un ambiente article che è specifico di ArsTEXnica. Per risolvere facilmente il problema anche qui si useranno due direttive ‘#+LATEX:’, una all’inizio e una alla fine del testo.

```
#+LATEX: \begin{article}
...
#+LATEX: \end{article}
```

A.5. E infine… la consegna

Dopo le ultime revisioni e i controlli, arrivato il momento della consegna, viene da chiedersi: « Ora, cosa consegnare? »

Attualmente, ArsTEXnica non accetta sorgenti ‘.org’: sarà quindi necessario generare il sorgente LaTeX con ‘C-c C-e l l’ e includerlo insieme a tutti gli elementi aggiuntivi necessari, come le immagini, in un pacchetto zip e infine consegnarlo alla redazione e al processo di peer-review.

Ecco fatto! Buon peer review… e buona lettura a tutti[12].

Riferimenti bibliografici

AA.VV. (2022). «Org mode tools!» https://orgmode.org/worg/org-tools/. Visitato il 1 apr 2023.

Babenhauserheide, Arne (2014). «Tutorial: Writing papers for acpd using emacs org-mode». https://www.draketo.de/files/howto-write-for-acpd-with-emacs.pdf. Visitato il 1 apr 2023.

Bar, Haim e HaiYing Wang (2021). «Reproducible science with latex». *Journal of Data Science*, **19** (1), pp. 111–125. https://arxiv.org/abs/2010.01482.

Borkowski, Marcin (2018). «Texing in emacs». **39** (1). https://www.tug.org/TUGboat/tb39-1/tb121borkowski-emacs.pdf.

Fruchart, Mathilde, Benjamin Guinhouya, Sylvia Pelayo, Christian Vilhelm e Antoine Lamer (2022). «Jupyter notebooks for introducing data science to novice users». *Studies in Health Technology and Informatics*, **294**, pp. 823–824. https://doi.org/10.3233/shti220598.

12. Il codice .org di questo articolo è disponibile nel repository https://gitlab.com/exedre/arstexnica-orgmode.

Giordano, Mosè, Orlando Iovino e Matteo Leccardi (2013). *Guida pratica all'uso di GNU Emacs e AUCTEX*. GuIt. https://github.com/GuITeX/guidaemacsauctex.

Glickstein, Bob (1997). *Writing GNU Emacs Extensions: Editor Customizations and Creations with Lisp*. O'Reilly Media, Inc.

Gruber, John (2004). «Markdown: Syntax». http://daringfireball.net/projects/markdown/syntaxp. Visitato il 1 apr 2023.

GuIt (2022). «Istruzioni per gli autori». https://www.guitex.org/home/en/for-authors. Visitato il 1 apr 2023.

Knuth, Donald Ervin (1984). «Literate programming». *The computer journal*, **27** (2), pp. 97–111. https://doi.org/10.1093/comjnl/27.2.97.

Knuth, Donald Ervin e Silvio Levy (1993). *The CWEB System of Structured Documentation*. Addison-Wesley Professional. http://www-cs-faculty.stanford.edu/~knuth/cweb.html.

Leha, Andreas e Tim Beissbarth (2011). «The emacs org-mode: Reproducible research and beyond». In *The R User Conference, useR! 2011 August 16-18 2011 University of Warwick, Coventry, UK*. p. 28. https://www.r-project.org/conferences/useR-2011/abstracts/010411-lehaandreas.pdf.

Leonard, Sean (2016). «Guidance on markdown: Design philosophies, stability strategies, and select registrations». RFC 7764, IETF. https://www.rfc-editor.org/rfc/rfc7764.txt.

Lewis, Bil, Daniel LaLiberte, Richard Stallman *et al.* (1997). *GNU Emacs Lisp Reference Manual*. Free Software Foundation. https://www.gnu.org/software/emacs/manual/html_node/elisp/index.html.

Mailund, Thomas (2014). *Introducing Markdown and Pandoc: Using Markup Language and Document Converter*. Apress.

Monnier, Stefan e Michael Sperber (2020). «Evolution of emacs lisp». *Proceedings of the ACM on Programming Languages*, **4** (HOPL), pp. 1–55. https://dl.acm.org/doi/10.1145/3386324.

Schöch, Christof (2014). «The right tool for the job: Five collaborative writing tools for academics». Blog post. https://blogs.lse.ac.uk/impactofsocialsciences/2014/04/04/five-collaborative-writing-tools-for-academics/. Visitato il 1 apr 2023.

Schulte, Eric e Dan Davison (2011). «Active documents with org-mode». *Computing in Science & Engineering*, **13** (3), pp. 66–73. https://ieeexplore.ieee.org/document/5756277.

Somma, Emmanuele (2009). «Il respawn di Infomedia (LaTeX-based)». *ArsTeXnica*, (8), pp. 92–101. http://www.guitex.org/home/numero-8.

Stallman, Richard M. (1981). «Emacs the extensible, customizable self-documenting display editor». In *SIGPLAN SIGOA Symposium on Text Manipulation*. https://dl.acm.org/doi/10.1145/872730.806466.

Stanisic, Luka, Arnaud Legrand e Vincent Danjean (2015). «An effective git and org-mode based workflow for reproducible research». *ACM SIGOPS Operating Systems Review*, **49** (1), pp. 61–70.

Voegler, Jens, Jens Bornschein e Gerhard Weber (2014). «Markdown–a simple syntax for transcription of accessible study materials». In *Computers Helping People with Special Needs: 14th International Conference, ICCHP 2014, Paris, France, July 9-11, 2014, Proceedings, Part I*. Springer, pp. 545–548. https://link.springer.com/chapter/10.1007/978-3-319-08596-8_85.

White, Jason JG (2022). «Using markup languages for accessible scientific, technical, and scholarly document creation». *Journal of Science Education for Students with Disabilities*, **25** (1), p. 5. https://scholarworks.rit.edu/jsesd/vol25/iss1/5/.

Emmanuele F. Somma
Banca d'Italia
Dipartimento Economia e Statistica
Servizio Struttura Economica
ef.somma@exedre.org

CJK Unicode strokes e radicals con MFLua: uno studio preliminare

Luigi Scarso

Sommario Presentiamo un metodo originale per la generazione di un font Opentype per il blocco Unicode *Kangxi Radicals* e *CJK Strokes* del sistema di scrittura cinese utilizzando MFLua e alcuni programmi ausiliari. Dato il suo carattere non definitivo, il font generato non è inteso per essere usato in normali applicazioni come usuale per altri font.

Abstract We present an original method for generating an Opentype font for the Unicode block *Kangxi Radicals* and *CJK Strokes* of the Chinese writing system using MFLua and some auxiliaries programs. Given its non-definitive status, the generated font is not intended to be used in normal applications as usual for other fonts.

1. Introduzione

A circa trent'anni dalla pubblicazione della prima versione (la versione 1.0.0 è datata Ottobre 1991, la 1.1.0 è del giugno 1993 e corrisponde alla prima versione dello standard ISO ISO/IEC 10646-1:1993) Unicode è lo standard de-facto per la codifica di ogni carattere usato per la scrittura di testi. Il repertorio potenziale è di circa 1,1 milioni di caratteri e l'ultima versione (la 15.0.0 del settembre 2022, Unicode Standard (2022)) ne codifica in totale 149.186, poco meno del 15% del totale. Lo standard definisce inoltre 161 collezioni di lettere ed altri segni scritti (*scripts*) che sono utilizzate in almeno un sistema di scrittura: ad esempio lo script `Latin` comprende 1.481 caratteri usati nella lingua italiana, francese, tedesca (ma non solo), mentre lo script `Armenian` comprende 96 caratteri ed è usato solo nella lingua armena; lo script `Han`, usato per il cinese, giapponese e coreano, contiene 98.408 caratteri. Infine esistono 3 scripts, `Common`, `Inherited` e `Unknown` per gestire i caratteri comuni (ad esempio i numerali o i segni diacritici) e quelli non ancora assegnati, in modo tale che ogni carattere appartenga esattamente ad uno e solo uno script.

Il repertorio potenziale è più correttamente chiamato spazio Unicode (*Unicode space*) ed è, semplicemente, l'insieme di numeri naturali da 0 a 1.114.111 inclusi (quindi in totale 1.114.112 numeri); ciascun numero è chiamato *code point* ed è indicato con una particolare notazione esadecimale *U+xxxx*, in modo tale che lo spazio Unicode è formato dai code point `U+0000`…`U+10FFFF`. Lo spazio Unicode è partizionato in sette gruppi di code point di cui quattro (`Graphic`, `Format`, `Control` e `Private-use`) possono essere assegnati ai caratteri astratti (*abstract characters*), mentre tre (`Surrogate`, `Noncharacter` e `Reserved`) sono a vario titolo riservati e non individuano mai un carattere astratto. Da notare che i sette gruppi non hanno la stessa cardinalità e neppure sono necessariamente

intervalli sequenziali di code point. Unicode riserva il termine blocco (*block*) ad un intervallo sequenziale di code point che contiene un multiplo di 16 di code point e che inizia sempre ad un code point multiplo di 16. Un blocco ha un nome (e.g. `Basic Latin` o `Arrows`) che tendenzialmente indica l'area semantica dei code point; due blocchi non si sovrappongono mai.

Il termine *abstract character* in Unicode ha significato di "carattere sorgente" : è l'*unità di informazione utilizzata per l'organizzazione, il controllo o la rappresentazione di dati testuali*, e proviene dal mondo "esterno" ad Unicode: è il carattere che, direttamente o meno, dovrà essere rappresentato dallo standard. Il caso più semplice si ha quando lo standard assegna *stabilmente* ad un carattere astratto un nome ed una "rappresentazione pittorica" (immagine): a questo punto il carattere diventa codificato (*encoded character*), ed è quindi possibile assegnargli un code point: ad esempio la lettera maiuscola `A` ha code point `U+0041`, nome `LATIN CAPITAL LETTER A` ed immagine [A]. Nella pratica la tripla <code point, nome, immagine> dovrebbe risolvere (o quantomeno rendere accettabile) la potenziale circolarità/ambiguità che si può presentare in certi casi nella definizione (come appunto per la lettera `A`) ed in questo modo possiamo indicare un carattere (codificato) in modo univoco con il suo code point oppure con il suo nome, mentre per l'immagine permane un margine di incertezza: per la lettera `A` abbiamo ad esempio le potenziali immagini [A], [**A**], [A], [`A`].

Il caso più complesso si ha quando un carattere astratto viene indicato da una sequenza di due o più code point: la lettera à può essere indicata sia con `U+00E0 LATIN SMALL LETTER A WITH GRAVE` (perché ha già un suo code point assegnato) od in modo equivalente con

`U+0061 LATIN SMALL LETTER A` + `U+0300 COMBINING GRAVE ACCENT` ([`a`] + [̀])

mentre nel caso (inventato per lo scopo) di à̯ abbiamo solamente :

`U+0061 LATIN SMALL LETTER A` +

`U+032F COMBINING INVERTED BREVE BELOW` +

`U+030F COMBINING DOUBLE GRAVE ACCENT` ([`a`] + [̯] + [̏]). Tecnicamente à̯ è un carattere astratto che non ha un code point ma diventa una sequenza Unicode grazie alla combinazione valida di una lettera e due diacritici (tre code point) ed è associata all'immagine à̯ – ma è un caso: infatti con questo particolare font a spaziatura fissa LuaLaTeX è "in grado" di comporre dinamicamente la suddetta sequenza Unicode, ma già cambiando font à̯ diventa à̯□ e questo perché, molto probabilmente, non esiste un glifo per à̯. Il termine glifo, almeno per come è usato in quest'articolo, fa infatti riferimento ad una singola "unità" di un particolare font resa ad una certa dimensione metrica: A, **A**, A, `A` sono quattro glifi di quattro font diversi, resi a 12pt, della medesima "unità" A – che per convenzione si chiama *carattere* A del font. Esiste un parallelo con il *grafema* (Grafema, 2022), l'"unità" elementare di un sistema di scrittura, che nel caso dell'alfabeto italiano è ancora rappresentato graficamente tramite un carattere; il sistema grafematico italiano (SGI, 2022) ha 21 grafemi, le lettere <a>…<z> (j, k, x, y, w sono grafemi ma non dell'alfabeto italiano) senza distinzione tra maiuscole, minuscole e lettere accentate (gli accenti sono solo acuto, grave e circonflesso). Il termine carattere ha quindi significati diversi ma fortemente correlati:

1. il grafema <a>, indicato graficamente col carattere a, corrisponde al glifo **a** del carattere a di un particolare font ed in Unicode è il carattere codificato `U+0061 LATIN SMALL LETTER A`;

2. il grafema <a> può essere anche scritto come <A>, ma sono due glifi differenti, **a** e **A**, ed in Unicode sono due code point differenti, `U+0061` e `U+0041`;

3. i grafemi <f><i> in alcuni font sono un unico glifo **fi** ed in altri no (e.g. `fi`); nel primo caso in Unicode abbiamo `U+FB01 LATIN SMALL LIGATURE FI`, nel secondo caso due code point distinti, `U+0066 LATIN SMALL LETTER F` + `U+0069 LATIN SMALL LETTER I` – ma possono essere usati anche per il primo caso;

4. l'accento grave ` non è un grafema ma la maggior parte dei font hanno il carattere accento grave, i.e. il glifo `; in Unicode è il carattere codificato `U+0060 GRAVE ACCENT` – che non va confuso con `U+0300 COMBINING GRAVE ACCENT` anche se le loro immagini sono simili;

5. à̗ non è un grafema, non è il glifo di un qualche carattere di un font, ed in Unicode è un carattere astratto che viene rappresentato come una sequenza di code point, come visto precedentemente.

Unicode quindi non è solo un repertorio di caratteri ma anche di precise *istruzioni tipografiche* come nel caso della combinazioni di lettere e diacritici: un programma che legge un flusso di code point dovrebbe essere in grado di eseguire correttamente queste istruzioni. Come si è visto, non sempre questo è il caso ed anzi anche à̗ non appare dal punto di vista tipografico del tutto corretta (la posizione del diacritico superiore e le spaziature verticali sono discutibili); in Unicode tuttavia non sono presenti code point utili per il corretto posizionamento ai fini tipografici, che rimane quindi a discrezione del programma, ma in definitiva la miglior soluzione è che il font abbia il glifo associato al carattere richiesto. Il caso delle combinazioni con diacritici poi non è l'unico: Unicode ha diciassette code point per le tipiche spaziature tipografiche, che sono però di semplice implementazione. Un secondo aspetto è il numero elevato di caratteri codificati. Un font ha una "unità stilistica" intrinseca che lo contraddistingue e che dovrebbe essere mantenuta nei suoi glifi, ma con 149.186 caratteri il compito è decisamente impegnativo ed anche considerando il fatto che un font, in questo contesto, è un font *digitale*, i.e. un programma, le sue capacità sono limitate.

In definitiva ciò che è auspicabile è che una applicazione per la creazione di font abbia, come METAFONT, la capacità di *programmare* un font – e naturalmente di produrre un font di un formato valido, che al giorno d'oggi è Opentype.

2. MFLua

MFLua nasce dalla constatazione che METAFONT è un linguaggio di programmazione ad alto livello per la progettazione e la produzione di font, il cui elemento fondamentale, la

curva di Bézier, è di tipo cubico mentre il font prodotto è di tipo bitmap – e non vettoriale come OpenType. Proprio a causa del tipo bitmap METAFONT è stato progressivamente abbandonato ed in relativamente pochi casi (rispetto ai font globalmente prodotti) sostituito da METAPOST, la cui sintassi è molto simile ma a cui manca la parte relativa alla gestione del tipo bitmap. Notevole in questo senso il caso della famiglia di font `Latin Modern` che conferma l'aspetto fondamentale dei `Computer Modern`: il font inteso come programma e quindi sia la possibilità di creare un carattere astratto da cui far discendere istanze diverse ma compatibili in senso stilistico (e quindi famiglie di font, da cui il prefisso META) sia creare efficacemente un numero elevato di glifi. L'osservazione fondamentale è che le informazioni necessarie per produrre un font vettoriale sono in realtà presenti anche in METAFONT– ma solo nel file di `log`, e solo attivate con la macro `\loggingall`; una post-elaborazione del `log` risulta, in generale, complicata.

Seguendo l'approccio di LuaTEX, MFLua è una modifica del programma sorgente di METAFONT ottenuta inserendo in punti opportuni delle nuove istruzioni che, in sostanza, memorizzano proprio quelle informazioni altrimenti visibili solo nel `log` e le rendono disponibili al programmatore. Per agevolare la post-elaborazione si utilizza un linguaggio di programmazione diverso da METAFONT, Lua– esattamente come in LuaTEX, ma differisce da LuaTEX in due fondamentali principi di design:

1. le nuove istruzioni non modificano lo stato interno di MFLua; sono istruzioni cioè che leggono un dato, ma non modificano dati, perché si vuole mantenere la totale compatibilità con METAFONT;

2. per limitare il numero di istruzioni inserite, le eventuali informazioni mancanti per quanto possibile vengono gestite in post-elaborazione da Lua. L'inserimento di una nuova istruzione nel sorgente di MFLua comporta infatti non solo la ricompilazione del sorgente in un nuovo eseguibile, ma anche un rallentamento dell'esecuzione complessiva del programma. Oltre a ciò si vuole favorire lo sviluppo di font MFLua che diano come risultato finale un font OpenType *anche utilizzando, se necessario altri programmi*.

Si può descrivere il funzionamento tipico di MFLua nel seguente modo: durante la compilazione di un font sorgente l'interprete Lua viene richiamato dalle nuove istruzioni per memorizzare i vari dati (tipicamente le curve di Bézier ed eventualmente i bitmap dei glifi) e giusto prima di uscire viene eseguita una funzione Lua `end_program` (che risiede in un file esterno `end_program.lua`) tramite il quale l'utente può gestire i dati raccolti e produrre un font in formato OpenType. Se il font sorgente utilizza solo outlines, le stesse outlines si ritroveranno immutate in `end_program` – e questo è il tipico flusso di lavoro di un font designer: utilizzare le outlines come elementi costitutivi del proprio carattere. È possibile inoltre utilizzare la libreria MetaType1 (metatype1, 2015) che risolve i problemi di intersezione tra outlines ed aggiungere un importante livello di astrazione nella programmazione del font. MFLua però non impone come tradurre i dati memorizzati in `end_program` in un font OpenType: questo è il compito del *backend*, ed è interamente compito dell'utente implementare il backend desiderato. In altre parole in MFLua `end_program.lua` non fa parte del macro formato ma è equivalente ad una

file di macro specifiche del font – è l'utente che sceglie se è un file ad uso generale o meno per i suoi progetti. Infatti, un backend particolarmente interessante è il formato ttFont del programma `ttx`, parte del package `fontTools` (fontTools, 2023), che è un file XML che descrive completamente un file Opentype. Il programma `ttx` legge un file ttFont e lo trasforma in un file Opentype (e viceversa) ed in Lua è relativamente semplice scrivere una funzione che trasformi le outlines in formato ttFont (che per brevità indicheremo anche come formato ttx). Ma si tratta comunque di una scelta opinabile: il formato ttx non è diffuso tra i formati Opentype ad alto livello.

Si può quindi delineare per un dato progetto il flusso di lavoro seguente: si utilizza una singola cartella di lavoro per il progetto la quale contiene

- i files per produrre il formato `base` utilizzato da MFLua ed il formato compilato; il formato cioè è specifico del progetto, anche se è in generale prodotto da copie verbatim di `mfluaplain.mf` e `mfluamodes.mf` (questi files si trovano in `texmf-dist/metafont/mflua` e `texmf-dist/scripts/mflua` della distribuzione TeXLive 2023);
- i files Lua come `end_program.lua` ed eventuali altri files che contengono informazioni sul font come autore, data etc;
- i files di macro in MFLua del font, con i singoli caratteri di cui si specificano *solo outlines*;
- i files Lua del backend ttx, atti a produrre il font in formato ttx *per lo specifico progetto*;

Una tipica iterazione consiste nel lanciare il programma `mflua` sul font in formato MFLua che deve terminare con un file ttx valido seppur incompleto e trasformare il file ttx in un file Opentype tramite il comando `ttx -f <filename>.ttx`, in modo da ottenere sempre un file finale valido. L'iterazione finale deve produrre il font che si considera definitivo.

Una fondamentale componente di METAFONT è la *penna*, che simula il tracciato di una penna ellittica o poligonale che si muove lungo una curva di Bézier. Purtroppo non è né semplice né robusto ricavare nel backend le outline finali di un tracciato mentre in linea di principio è più semplice ricavare le outline dal bitmap finale del carattere (che viene comunque prodotto): per questo motivo MFLua incorpora il programma `potrace` (Potrace, 2019), che ricava da una immagine bitmap le outline cubiche di Bèzier dei contorni. Il flusso di lavoro è simile al precedente: il compito del backend in questo caso è l'analisi del file *generic font* che contiene i bitmap di ciascun carattere, in modo da ottenere con la libreria interna `potrace` le outline (con il corretto orientamento) di ciascun glifo da passare poi alla subroutine ttx. Il metodo si può applicare anche quando si hanno solo outline, e benché semplice (ed in teoria valido per ogni font scritto in METAFONT) ha una importante conseguenza negativa: le curve cubiche ottenute con `potrace` non sono direttamente legate alla curva su cui si muove la penna – in sostanza, non si ha un buon controllo delle outline finali che tendono anzi ad essere composte da un numero eccessivo di cubiche.

Esistono infine altre tre componenti che fanno parte di MFLua e sono (in ordine di importanza):

1. una nuova primitiva **runscript** <string primary>, che chiama l'interprete Lua sulla stringa di ingresso e poi si comporta come **scantokens**. In questo modo è possibile incapsulare in un sorgente METAFONT frammenti di codice Lua, ma a causa dell'implementazione piuttosto semplificate delle stringhe in METAFONT è spesso preferibile salvare il codice Lua in un file esterno (e.g `myfile.lua`) e chiamare **runscript** con `dofile`, e.g.

   ```
   runscript("dofile([[myfile.lua]])");
   ```

2. la libreria `Lpeg`, utile per il parsing di files di input;

3. la libreria `otfcc` (otfcc, 2022), che converte una descrizione ad alto livello di un font direttamente in Opentype (e viceversa). Si tratta di una estensione di MFLua ancora a livello sperimentale, il cui uso per il momento non è consigliato.

3. Unicode e scripts dell'Asia orientale

Il capitolo **East Asia** della versione 15.0.0 dello standard Unicode è dedicato agli scripts (che ricordiamo sono collezioni di lettere ed altri segni scritti) dell'Asia orientale, in cui emerge il ruolo preponderante della storia della Cina continentale nella definizioni dei caratteri codificati. Lo script principale infatti è *Han* con 98.408 caratteri ed è lo script che ha più code point di tutti gli altri (ed il secondo è *Hangul*, il sistema di scrittura attuale del linguaggio coreano, con 11.739 caratteri). *Han* è frequentemente indicato con il termine CJK per *Chinese-Japanese-Korean* ed indica un insieme di logogrammi condivisi dal linguaggi cinese, giapponese e coreano. Tale condivisione (spesso indicata col termine *Han unification* o *Unihan*) però non è stata esente da critiche in quanto diversi caratteri codificati non hanno una immagine universalmente accettata: ad esempio in `U+5203 CJK UNIFIED IDEOGRAPH-5203` 刃 il tratto 丶 dovrebbe essere sostituito da 丿 (Han unification, 2023). Se la differenza grafica appare minima è fondamentale sottolineare l'importanza dello studio dei logogrammi fin dalla prima età scolare, e la complessa organizzazione di tale studio: cambiare la grafica di un segno ha conseguenze difficili da gestire. Volendo fare un esempio "occidentale" la differenza grafica tra `i ı` è minima, ma sono chiaramente percepite come segni completamente diversi (`ı` è una lettera dell'alfabeto turco) – anche se nella scrittura manuale il puntino sulla i non sempre viene messo. Identico discorso per `A` e `Λ` – due lettere completamente diverse (`Λ` è la lettera lambda maiuscola dell'alfabeto greco).

L'elenco degli scripts relativi all'Asia orientale è il seguente:

- *Han* (98.408 code point), *Tangut* (6,914 code point), *Yi* (1.220 code point), *Khitan Small Script* (471 code point), *Nüshu* (397 code point), *Miao* (149 code point), *Bopomofo* (77 code point), e *Lisu* (49 code point) riconducibili all'area geografica cinese continentale ed insulare;

- *Hangul* (11.739 code point) riconducibili all'area geografica coreana;
- *Hiragana* (381 code point) e *Katakana* (321 code point) riconducibili all'area geografica giapponese.

Non tutti sono sistemi di scrittura logografici – *Miao* ad esempio è un *alfasillabario* o *abugida*, un sistema di scrittura ibrido in cui i grafemi fondono una consonante e una vocale intrinseca; *Tangut* e *Khitan Small Script* sono scritture antiche, e *Bopomofo* è un alfabeto per annotare od insegnare la fonetica del cinese.

Il sistema di scrittura cinese, rappresentato dallo script *Han*, è tra i più importanti sistemi logografici attualmente esistenti. A differenza di un alfabeto, in cui l'unità elementare è il grafema, in un sistema logografico l'unità elementare, il *logogramma*, rappresenta un *morfema*, ovvero il più piccolo elemento dotato di significato. Un logogramma quindi ha sia un valore fonetico che un valore semantico, a differenza di un grafema che ha solo un valore fonetico (e.g. <b> si legge "bi" in italiano, ma non ha un valore semantico). Il termine *ideogramma* indica in realtà un tipo di logogramma che codifica un concetto astratto, mentre *pittogramma* è la rappresentazione stilizzata di un oggetto fisico; entrambi sono presenti nello script *Han* ma la maggior parte dei logogrammi è di tipo *radicale-fonetico*, dove il radicale è un logogramma semantico e *fonetico* è un logogramma per la pronuncia. Lo standard Unicode prevede numerosi blocchi dedicati ai logogrammi, ma due sono particolarmente importanti nel contesto di questo articolo:

1. il blocco *CJK Strokes*, code point `U+31C0`…`U+31EF`:
   ㇀ ㇁ ㇂ ㇃ ㇄ ㇅ ㇆ ㇇ ㇈ ㇉ ㇊ ㇋ ㇌ ㇍ ㇎ ㇏
   ㇐ ㇑ ㇒ ㇓ ㇔ ㇕ ㇖ ㇗ ㇘ ㇙ ㇚ ㇛ ㇜ ㇝ ㇞ ㇟
   ㇠ ㇡ ㇢ ㇣
2. il blocco *Kangxi Radicals*, code point `U+2F00`…`U+2FDF`:
   一 丨 丶 丿 乙 亅 二 亠 人 儿 入 八 冂 冖 冫 几
   凵 刀 力 勹 匕 匚 匸 十 卜 卩 厂 厶 又 口 囗 土
   士 夂 夊 夕 大 女 子 宀 寸 小 尢 尸 屮 山 巛 工
   己 巾 干 幺 广 廴 廾 弋 弓 彐 彡 彳 心 戈 戶 手
   支 攴 文 斗 斤 方 无 日 曰 月 木 欠 止 歹 殳 毋
   比 毛 氏 气 水 火 爪 父 爻 爿 片 牙 牛 犬 玄 玉
   瓜 瓦 甘 生 用 田 疋 疒 癶 白 皮 皿 目 矛 矢 石
   示 禸 禾 穴 立 竹 米 糸 缶 网 羊 羽 老 而 耒 耳
   聿 肉 臣 自 至 臼 舌 舛 舟 艮 色 艸 虍 虫 血 行
   衣 襾 見 角 言 谷 豆 豕 豸 貝 赤 走 足 身 車 辛
   辰 辵 邑 酉 釆 里 金 長 門 阜 隶 隹 雨 青 非 面
   革 韋 韭 音 頁 風 飛 食 首 香 馬 骨 高 髟 鬥 鬯

鬲 鬼 魚 鳥 鹵 鹿 麥 麻 黃 黍 黑 黹 黽 鼎 鼓 鼠
鼻 齊 齒 龍 龜 龠

Infatti un logogramma dello script *Han* può essere decomposto in uno o più radicali Kangxi, ed un radicale può essere a sua volta decomposto in uno più strokes. Quindi il blocco *CJK Strokes* contiene i tratti grafici elementari di un logogramma *Han* organizzati in radicali – ed è appunto questa organizzazione che induce ad una programmabilità del logogramma stesso.

4. CJK Strokes e Kangxi Radicals con MFLua

Come probabilmente intuibile un progetto che si ponga come scopo la produzione di un font Opentype per "la lingua cinese" deve necessariamente iniziare con una fase di raccolta di documentazione e precisazione degli obiettivi da raggiungere. La complessità dell'argomento, e la forte e viva tradizione tipografica della lingua cinese impongono di affrontare il progetto scomponendolo in parti precise e facilmente traducibili in passi operativi. In questo caso si è ritenuto opportuno procedere come segue:

1. scegliere l'attuale standard Unicode come riferimento, in particolare per quanto riguarda l'immagine dei glifi;

2. tra le varie risorse disponibili preferire, quando possibile, Wikipedia e Wikimedia in lingua inglese. Sono risorse facilmente accessibili e permettono un confronto tramite una lingua neutra e nota;

3. produrre fin dalla prima iterazione un font Opentype valido, anche se non usabile per scopi pratici. La complessità intrinseca di un font (di *qualunque* font) infatti rischierebbero in questa fase di predominare su altri aspetti, principalmente la formazione di una base documentale sufficientemente solida per motivare le scelte successive.

Per quanto riguarda il font Opentype il primo passo fondamentale è il suo *typeface* ovvero come è visualmente percepito rispetto a font analoghi. La scelta è ricaduta sullo stile calligrafico cinese Regolare (o Standard) (*Regular script*, Regular script (2023)) nato in Cina intorno al 200 a.C. ma affermatosi attorno al VII secolo. È lo stile più comune negli scritti moderni e il terzo più comune nelle pubblicazioni (dopo gli stili Ming e gotico, che vengono utilizzati esclusivamente in stampa) e viene talvolta riportato con in nomi *Kaiti* (da 楷體 (kǎitǐ)), *Kaishu* o *Kaisho*. Il secondo passo fondamentale è quali glifi produrre, e la scelta è ricaduta sui blocchi *CJK Strokes* e *Kangxi Radicals*, in quanto componenti costitutivi per gli altri logogrammi.

La scelta successiva dipende da quale direzione si vuole privilegiare: la programmabilità del glifo versus la produzione di un font completo in cui ogni glifo però è codice a se stante. Si è scelta quest'ultima, in base alla considerazione che è più opportuno avere prima un quadro di insieme completo che funga da riferimento, in modo da rilevare

punti critici che possono passare inosservati (e.g. 巛 sembra composto da tre ripetizioni identiche dello stesso tratto, ma non è esattamente cosi nel Regular script; tuttavia in un logogramma che contiene 巛 come radicale forse si può assumere l'uguaglianza dei tratti senza pregiudicarne la resa).

Il passo finale, a questo punto, è piuttosto immediato:

1. si sono individuati in Wikimedia i sorgenti in formato svg in stile Regular script dei blocchi *CJK Strokes* (CJK strokes, 2023) e *Kangxi Radicals* CJKV radicals (2023);
2. ciascuno glifo in formato svg è stato convertito in bitmap, ottenendo i files `31C0.pbm` …`31EF.pbm` per il blocco *CJK Strokes* e `2F00.pbm`…`2FDF.pbm` per il blocco *Kangxi Radicals*;
3. con uno script MFLua ogni bitmap è stato convertito in outline METAFONT, usando la libreria interna `potrace`. Si tratta di un procedimento duale rispetto a quanto visto in precedenza: invece di usare `potrace` per ricavare le outline da un font bitmap prodotto da MFLua, lo si usa per ricavare da un bitmap le outline in formato METAFONT– le problematiche comunque non cambiano.
4. infine ogni outline in formato METAFONT è semplicemente l'input del relativo carattere, di dimensioni 1000×1000.

Ad esempio il carattere `uni31c0` nel font Opentype finale è il glifo di `U+31C0` ed ha codice sorgente:

```
begincharacter("","0x31C0",1000,1000,0);
path p[];
%% 1 outline(s)
input "./glyphs-strokes/outlines/31C0";
%%
p1:= reverse p1;
fill (p1 shifted (500-bmp31C0.x/2,0*bmp31C0.y/2));
endcharacter;
```

dove `./glyphs-strokes/outlines/31C0.mf` è

```
p1 := (38.771,13.750)
   ..controls (22.497,29.664) and (9.829,43.722)..(4.837,51.405)
   ..controls (1.867,55.976) and (1.000,58.302)..(1.000,61.700)
   ..controls (1.000,68.050) and (3.836,70.556)..(12.087,71.499)
   ..controls (33.252,73.920) and (86.171,96.196)..(188.500,145.760)
   ..controls (248.706,174.921) and (381.060,241.677)..(421.500,263.278)
   ..controls (429.750,267.685) and (449.270,277.775)..(464.878,285.700)
   ..controls (494.577,300.779) and (498.150,301.978)..(498.840,297.095)
   ..controls (499.106,295.208) and (497.704,293.296)..(492.340,288.233)
   ..controls (481.810,278.295) and (450.841,255.480)..(415.000,231.260)
   ..controls (273.222,135.447) and (109.007,26.388)..(84.066,11.478)
   ..controls (73.013,4.870) and (63.127,1.026)..(57.154,1.012)
   ..controls (57.154,1.012) and (51.808,1.000)..(51.808,1.000)
   ..controls (51.808,1.000) and (38.771,13.750)..(38.771,13.750)
   --cycle;
```

Infine le outline di ogni carattere sono state convertite dal backend in formato `ttx` e quindi in formato OpenType, come precedentemente descritto e convalidato tramite `FontValidator.exe` (HinTak, 2023). In fig. 1 è possibile vedere il risultato dei glifi prodotti.

In fig. 2 è possibile vedere chiaramente l'eccesso di cubiche rispetto alla versione più semplice ottenuta con Fontforge (Fontforge, 2023) per il glifo del code point `U+2F2E` – ma è una situazione comune ad ogni glifo.

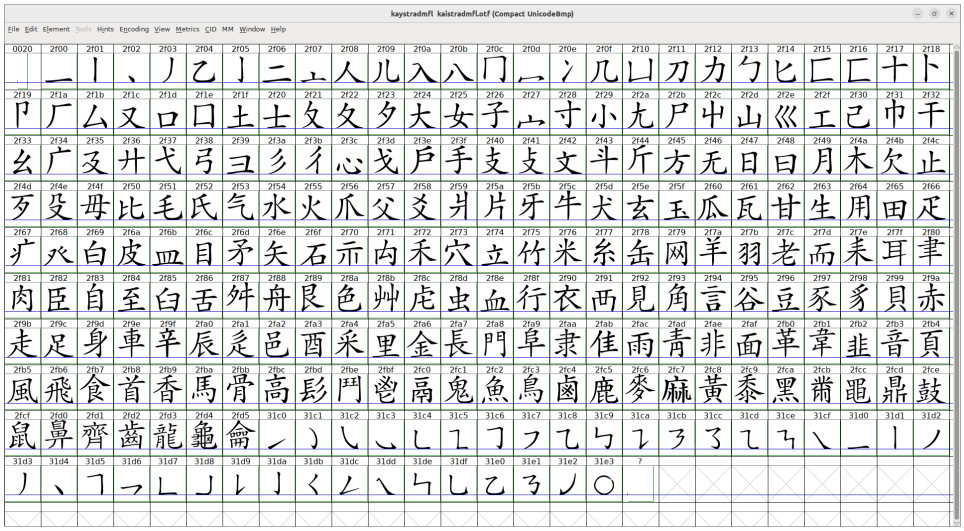


Figura 1. I glifi prodotti.

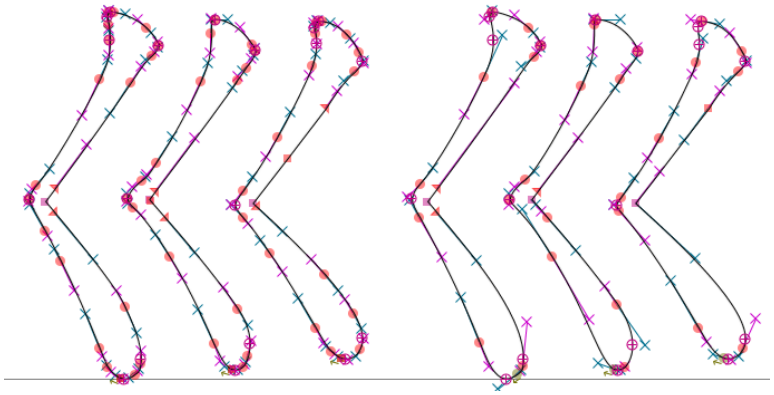


Figura 2. Il glifo originale di `U+2F2E` vs. il glifo semplificato.

5. Conclusioni

La creazione di un font per un sistema di scrittura complesso come quello cinese deve obbligatoriamente iniziare con una estesa documentazione che per scelta si è deciso di far gravitare attorno alla standard Unicode ed al formato Opentype. Il primo risultato mostra una coerenza stilistica dei glifi in linea con lo stile Regular script e costituisce quindi un buon quadro di riferimento, ma allo stesso tempo evidenza l'insufficiente qualità delle outline ed in generale dei parametri del font, che devono essere raffinati. Questo sarà l'oggetto della successiva iterazione: la semplificazione delle curve del blocco *CJK Strokes* e la produzione di un font migliore. Una possibile soluzione è in qualche modo prospettata dalla fig. 2 – utilizzare un programma come Fontforge per produrre il font finale. È sicuramente una opzione valida, ma non completamente soddisfacente: sebbene il typeface non sia originale sia perché esistono altri font Regular script probabilmente migliori, sia perché le outline provengono direttamente da fonti esterne, l'originalità a cui si tende consiste essenzialmente nel codice sorgente del glifo in formato METAFONT e nella sua programmabilità, cosa che ancora a questo stadio non è presente. Il passo successivo alla semplificazione è la composizione dei radicali tramite strokes. Ancora dalla fig. 2 si deduce che non è semplice come può apparire ad un primo sguardo: i tre strokes sono simili, ma non copie identiche traslate. In generale, la decomposizione in strokes è un tema noto e ben documentato, quindi il problema è capire come la programmabilità del glifo si inserisca correttamente in questo contesto.

Riferimenti bibliografici

CJK strokes (2023). «Stroke (CJK character)». https://en.wikipedia.org/wiki/Stroke_(CJK_character). [Online; accessed 17-April-2023].

CJKV radicals (2023). «Category:CJKV radicals and their variants in regular script style (in SVG format)». https://commons.wikimedia.org/w/index.php?title=Category:CJKV_radicals_and_their_variants_in_regular_script_style_(in_SVG_format). [Online; accessed 17-April-2023].

Fontforge (2023). «FontForge». https://fontforge.org/en-US/. [Online; accessed 17-April-2023].

fontTools (2023). «fontTools». https://pypi.org/project/fonttools/. [Online; accessed 17-April-2023].

Grafema (2022). «Grafema». https://it.wikipedia.org/wiki/Grafema. [Online; accessed 17-April-2023].

Han unification (2023). «Han unification». https://en.wikipedia.org/wiki/Han_unification. [Online; accessed 17-April-2023].

HinTak (2023). «Font-Validator». https://github.com/HinTak/Font-Validator. [Online; accessed 17-April-2023].

metatype1 (2015). «metatype1 – Generate Type 1 fonts from METAPOST». https://ctan.org/pkg/metatype1?lang=en. [Online; accessed 17-April-2023].

otfcc (2022). «otfcc». https://github.com/caryll/otfcc. [Online; accessed 17-April-2023].

Potrace (2019). «Potrace». https://potrace.sourceforge.net/. [Online; accessed 17-April-2023].

Regular script (2023). «Regular script». https://en.wikipedia.org/wiki/Regular_script. [Online; accessed 17-April-2023].

SGI (2022). «Sistema grafematico italiano». https://it.wikipedia.org/wiki/Sistema_grafematico_italiano. [Online; accessed 17-April-2023].

Unicode Standard (2022). «The Unicode Standard, Version 15.0.0». https://www.unicode.org/versions/Unicode15.0.0/. [Online; accessed 17-April-2023].

Luigi Scarso
luigi.scarso@gmail.com

Mapping to individual characters in expl3*

Joseph Wright

Abstract Humans see single glyphs even when they are made up of several commands, for example a letter followed by a self combining diacritic mark; humans call such visible signs graphemes. This package describes how to separate single graphemes in strings made up of Unicode encoded glyphs.

It is natural to think that separating a word up into individual characters is an easy operation. It turns out that for the computer this isn't really the case. If we look at a system that natively understands Unicode (like XeTeX or LuaTeX), most of the time one "character" is stored as one codepoint. A codepoint is a single character entity for a Unicode programme. For example, if we take the input "café" it is made up of four codepoints:

```
U+0063 (LATIN SMALL LETTER C)
U+0061 (LATIN SMALL LETTER A)
U+0066 (LATIN SMALL LETTER F)
U+00E9 (LATIN SMALL LETTER E WITH ACUTE)
```

So we could, in XeTeX/LuaTeX, use a simple mapping to grab one character at a time from this word and do stuff with it. However, that's not always the case. Take for example "Spın̈al Tap": the dotless-i is a single codepoint, but there is no codepoint for an umlauted-n. Instead, that is represented by two codepoints: a normal n and a combining umlaut. As a user, it's clear that we'd want to get a single "character" here. So there's clearly more work to do.

Luckily, this is not just a TeX problem and the Unicode Consortium have thought about it for us. They provide a data file and rules that describe how to divide input into *graphemes*: "user perceived characters". So "all" that is needed is to examine the input using these rules, and to divide it up so that "characters" stay together.

For pdfTeX, there's an additional wrinkle: it uses bytes, not codepoints, and so if we use a naïve TeX mapping, we would divide up any codepoint outside the ASCII range into separate bytes: not good. Luckily, the nature of codepoints is predictable: all that is needed is to examine the first byte and collect the right number of further bytes to re-combine into a valid codepoint.

This work isn't something the average end user wants to do. Luckily, they don't have to as the LaTeX team have worked on this and created a suitable set of expl3 functions to do it: `\text_map_function:nN` and `\text_map_inline:nn`.

*This paper has been published on *TUGboat* (43:3) 2022 by Joseph Wright, who kindly allowed to reproduce it on 4rsTeXnica; the Direction of 4rsTeXnica acknowleges his contribution and thanks him for his kindness. This paper was reformatted to comply with the one column 4rsTeXnica style on B5-sized paper by Claudio Beccari. Moreover C.B. could not use the same `NotoSansBengali-VariableFont_wdth,wght.ttf` font; he could use instead the `NotoSansBengali.ttf` one, but the results are visibly the same.

For example, we can do (mapping each character to printing itself in parentheses):

```
\ExplSyntaxOn
\text_map_inline:nn
  { Spıñal ~ Tap } { (#1) }
\ExplSyntaxOff
```

and get

(S)(p)(ı)(ñ)(a)(l)( )(T)(a)(p)

in any TEX engine —assuming we are set up to *print* the characters, of course. Getting the right fonts is an independent issue from parsing the input.

Taking a more "serious" example (and one that is going to use LuaTEX for font reasons), we might want to map over Bangla text: I'm going to use নন্দকিন্দ্র as my example. Our `\text_map_inline:nn` function divides up the characters correctly:

(ন্)(দ্)(র)(কি)(ন্)(দ্)(র)

In contrast, the generic `expl3` token-list function `\tl_map_inline:nn` gives:

(ন)(◌্)(দ)(◌্)(র)(ক)(◌ি)(ন)(◌্)(দ)(◌্)(র),

which is a very odd result. In short: Unicode characters are neither bytes nor tokens.

(If you want to try that demo yourself, you'll need a document preamble that can work properly with Bangla: I'm using

```
\usepackage{fontspec}
\newfontface\harfbangla
  {NotoSansBengali-VariableFont_wdth,wght.ttf}
  [Renderer = HarfBuzz, Script = Bengali]
```

then using `\harfbangla` in a brace pair around my demonstrations. Finding a monospaced font that properly renders Bangla is . . . tricky.)

So, as you can see, mapping to "real" text is easy with expl3: you just need to know that the tools are there.

Joseph Wright
Northampton, United Kingdom
joseph.wright@morningstar2.co.uk

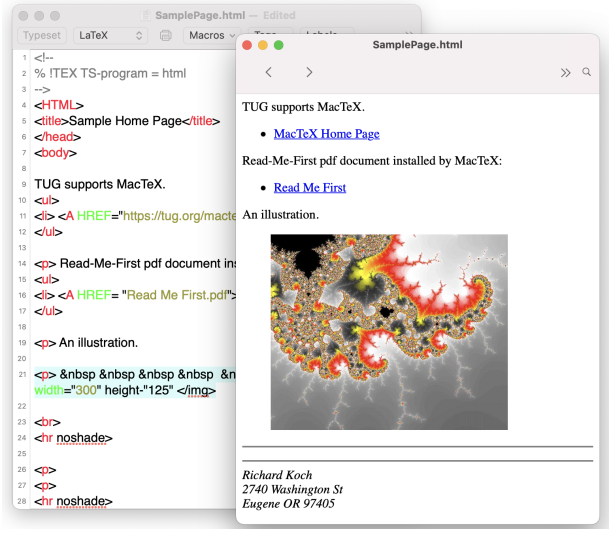
TeXShop, Version 5 (August 11, 2022)

Richard Koch

Sommario TeXShop provides a preview window for each open document, showing pdf output. Version 5 of the program provides an extra preview window for html output. Therefore, TeXShop authors can use typesetting engines which convert source files to pdf, or html, or both. Examples include TeX4ht, which accepts LaTeX source and outputs html, PreTeXt, which accepts XML source and outputs either pdf or html, and pure html source, which can be displayed as a live web page.

1. HTML Source

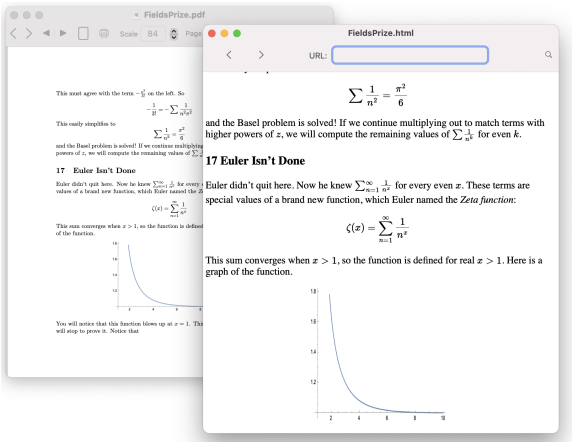
Pictured below is the most straightforward example. The window on the left contains html source and the window on the right shows the resulting web page. Links to local or remote web pages work, as do links to pdf files, illustrations, and all other standard web content. After revising the source file, a user can type command-T, the standard TeXShop shortcut to typeset, and the web page will instantly update.



2. TeX4ht

Pictured next is an editing session using TeX4ht. To save space, the source window is not shown. The window on the left contains pdflatex output and the window on the right contains

TeX4ht output. The TeX4ht engine, a very short shell script, typesets the source twice, once with pdflatex for the pdf preview and once with TeX4ht for the html preview. If the user edits the source and typesets again, both windows instantly update. In the illustration, both windows were resized to be very small. The resizing changed the magnification in the pdf window, but reflowed the text in the html window.



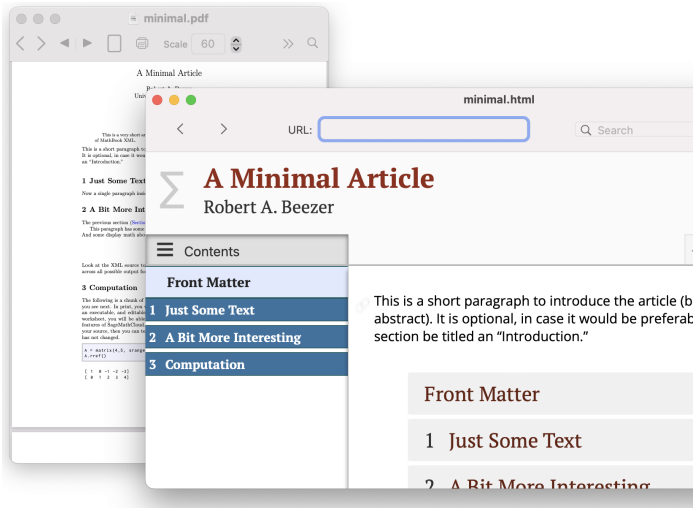
3. PreTeXt

PreTeXt is a project managed by Robert Beezer at the University of Puget Sound. Pictured on the next page is a typical editing session. The source window (not shown) contains xml source for document markup, but LaTeX source for mathematics. Typesetting calls the xsltproc program twice, once to convert the source to pdf, and once to convert it to html. The pdf is shown on the left and the html is shown on the right. PreTeXt deliberately formats these outputs differently, so the pdf looks like standard TeX and the html looks like a web project. A key advantage of the PreTeXt project is support for many forms of user interactions in the html page.

4. TeXShop Engines

TeXShop typesetting engines are simple shell scripts stored in ~/Library/TeXShop/Engines. A TeXShop user can navigate to this spot within the program, open files there, and edit them. Thus the user has complete control over the exact typesetting method used. In TeXShop 5.0, engine scripts can contain additional commands like

- **!TEX-pdfPreview**
- **!TEX-htmlPreview**
- **!TEX-bothPreview**



After typesetting, the first of these lines tells TeXShop to search for a file in the source directory with the same name as the source file and extension "pdf". If this file exists, it is opened in the pdf Preview window. The other commands work similarly.

5. Help Recalling HTML Commands

Several items in the TeXShop Help menu provide help recalling LaTeX commands. A new menu provides a list of html commands. The purpose of each command is listed, the appropriate tags are shown, and a short usage sample is provided. This help document can be typeset with command-T for easy viewing of examples explaining how to link to a pdf file, how to link to an external web page, how to display a movie, and how to display a UTube video.

If the user needs a command not listed, they can search for it on Google. Then they can enter a new item in TeXShop Help, because the help file is editable. TeXShop will remember the edit, so the Help list will expand as the user adds items.

6. Purpose of New Additions

Several talks at past TUG Conferences predicted the demise of TeX in five or six years, because faculty members need to provide interactive source material. I thought these talks were ridiculous. Then Covid struck, and faculty colleagues had to switch to remote learning in a week. The pessimistic talks had a point.

In my "dream setup", a faculty member would write lecture notes in Latex or XML and these notes would be typeset by an engine which outputs both pdf and html. The pdf document would contain the course in final polished form, so students could understand the logical flow of the lectures. The html document would contain interactive material: movies, questions with student input, etc., so students would learn how mathematics is actually created.

Source code could provide different output for pdf and html, using a construct like:

```
\ifpdf
    Here is an image.
\else
     -- html code embedding a movie--
 \fi
```

Maybe PreTeXt will become this system; maybe TeX4ht will be expanded for it, maybe new projects will be invented in the future. TeXShop is available when these engines are ready.

Richard Koch
koch@math.uoregon.edu

Interactive content using TEX4ht*

Richard Koch

Abstract TEX4ht converts LATEX source into web pages. This article explains how to add interactive content to these pages. The techniques should work on all computer platforms.

1. Introduction

Let me begin with three vignettes.

I started attending TUG conferences in 2001, and along with expected talks there were a few surprises. In 2005, an expert from England predicted that TEX would survive for four more years and then be replaced. He was teaching in the Open University system where students work remotely, and he wanted to include interactive content in his lectures. I thought the talk was nonsense. Then COVID hit.

The 2004 Practical TEX conference was held at Fisherman's Wharf in San Francisco, and included a talk by Ernest Prabhakar, an Apple engineer. After that talk, Prabhakar met with Mac users and others including Hans Hagen, all sitting around a large conference table. Hans was trying to convince Apple to allow Java programs to run in their pdf viewer so interactive elements could be added. I sat next to Prabhakar and got to see how he operates. He was fully engaged in the conversation, but simultaneously he was surfing the web –the fastest surfer I have ever seen. Eventually he said to Hans, "It appears to me that you are the only one in the world writing Java in pdf files."

I'm one of those users who updates TEX Live daily while drinking my morning coffee. Sometime in 2022 I noticed that `tex4ht` was on every day's update list. So I wrote the TEX Live mailing list asking that this bug be fixed. To my surprise, I was told that the updates were genuine; Michal Hoftich, who maintains TEX4ht, makes updates almost daily.

2. PDF and HTML fifteen years later

The Fisherman's Wharf conference was 18 years ago, and some issues are clearer with the passage of time. Today every computer platform has excellent software to display pdf files, and every computer platform has an up-to-date web browser. It seems clear that pdf is the right format for static documents, and that html is the right format for documents with interactive content. Other formats may emerge, but that will only happen if an activity cannot

*This paper was already published by Richard Koch on *TUGboat* 43:2(2022); the author kindly authorised to republish it on ArsTEXnica. The Editorial Board thanks him very much for his kindness and acknowledges his authorship. Claudio Beccari repaginated his paper in oder to cope with the ArsTEXnica typographical style; C.B. assumes all responsibility for any error he might have introduce in the paper.

be supported by pdf or html. (Although pdf has facilities for interactivity, they are rather infrequently used compared to interactive html.)

3. A TeXShop detour

I wrote TeXShop, a front end for TEX on the Macintosh. TeXShop is relevant here only because it explains how I was led to reexamine TEX4ht.

Typesetting in TeXShop is controlled by "engine files", small shell scripts that users can edit which call TeX binaries. After typesetting, an engine searches the source directory for a pdf file with the same name as the source and opens it in a pdf preview window if found. This August I added code which searches for an html file with the same name as the source, and opens it in an active web window if found. These windows are created using the Cocoa programming API's, so they are part of TeXShop rather than external Mac applications like Preview and Safari.

With this change, it is easy to "typeset" html files. Similarly, it is easy to support PreTEXt, a project where authors write xml source and then convert the source to pdf, html, and other formats.

So it was natural to try TEX4ht, which accepts a LATEX source file and outputs html (among other things). TeXShop now has a typesetting engine which typesets the source twice, once with TEX4ht and once with pdflatex. The TEX4ht output is opened in a web viewer and the pdf output is opened in a pdf preview.

I had seen demonstrations of TEX4ht given by Eitan Gurari, the original author of TEX4ht. Indeed at that 2004 conference at Fisherman's Wharf, Prabhakar's talk was immediately followed by a talk by Gurari on TEX4ht. At the time, TEX4ht was outputting mathematics using pictures, and the results were a little crude.

In the years since then, MathML was invented, and then MathJax was created and provided beautiful rendering of MathML code. TEX4ht adopted these technologies.

I selected a 20-page set of lecture notes, with extensive mathematical equations and many illustrations. The document used `hyperref`, `amsmath`, and other packages. I typeset it with TEX4ht, producing html. Typesetting was fast –and the html output was amazing! The mathematical equations were crisp and clear, the illustrations were fine; to tell the truth, I doubted that I was seeing html. As a test, I resized both windows. The text in the pdf window shrank since the pdf had been configured to "fit in window". The text in the html window reflowed.

4. Interactive content

TEX4ht therefore allows you to convert old and new LATEX static documents into web documents. But can you add interactive content to these documents? Yes, as this article will demonstrate.

Select an old document you have lying around the house. You'll be able to add interaction to it by the end of the next two sections. Don't typeset immediately because a couple of steps are needed. Both are given in these two sections.

First we need a method to write source code which will only appear in the html version of the document. The following code does the trick:

```
\ifx\HCode\undefined
  % source for pdf document
\else
  <!-- source for html document -->
\fi
```

The `\HCode` tested here is a command that appears only in TEX4ht. Some web documents recommend the `ifpdf` package, but that fails when typesetting with XETEX.

Next, we need to switch from writing LATEX code to writing html code which TEX4ht will insert directly into the final document without processing. The following code suffices:

```
\ifx\HCode\undefined
  % source for pdf document
\else
  Initial words for html document.
  \begin{html}
     <!-- direct html input -->
  \end{html}
\fi
```

Finally we need something interactive. We'll use a piece of SageMath code, which is explained in a later section. Putting all this together, add the following lines to your document, creating a new section in the web version.

```
\ifx\HCode\undefined
\else
 \section{An Experiment}
 \begin{html}
  <div class="compute">
  <script type="text/x-sage">
    plot(sin(x), (x, 0, 2*pi))
  </script></div>
 \end{html}
\fi
```

Running this, we discover a minor problem. TEX4ht does not understand the command `\begin{html}`, and your web browser does not understand the lines calling Sage.

5. The header

To solve the problem, we must add the following header immediately after `\begin{document}`, before any other code is inserted. (The columns in *TUGboat* are narrow; the long `sagemath.org` url below needs to be on one line, and other source lines would usually be combined. Also, the source for this article is available from the article's web page.)

```
\ifx\HCode\undefined
\else
 % declare environment html:
 \ScriptEnv{html}
 {\ifvmode\IgnorePar\fi\EndP
   \NoFonts\hfill\break}
 {\EndNoFonts}
\fi

\ifx\HCode\undefined
\else
 \begin{html}
  <script src="https://sagecell.sagemath.org/static/embedded_sagecell.js">
  </script>
  <script>
  // Make div with id `mycell' being a Sage cell
  sagecell.makeSagecell({
    inputLocation:  '#mycell',
    template:  sagecell.templates.minimal,
    evalButtonText: 'Activate'});
  // Make div with class `compute' a Sage cell
  sagecell.makeSagecell({
    inputLocation: 'div.compute',
    evalButtonText: 'Evaluate'});
  </script>
 \end{html}
\fi
```

This header is divided into two parts, each preceded by an `\HCode` test so it is only active when typeset by TEX4ht. The first defines `\begin{html}` for TEX4ht. The second defines the Sage commands for the browser. Notice that the second command is also preceded by `\begin{html}` and thus is inserted directly into the final html document.

Now your source document has everything required for interaction, so go ahead and typeset it with TEX4ht. The recommended way to typeset is

```
make4ht sourcefile.tex "mathjax"
```

In the pdf version of the document, nothing changed. The html version will contain an additional section pictured below. Notice the button labeled “Evaluate” (fig. 1). When this button is pushed, the display changes to the form shown in figure 2 on page 72.

But there’s more. The SageMath code shown in both images is editable. If this entry is changed to

```
plot(log(x), (x, .1, 10))
```

a graph of the logarithm is plotted, and if it is changed to

```
plot(sin(x) + cos(3*x)/2, (x, 0, 2*pi))
```

an alternate periodic function is plotted. Therefore the four lines of code we added for Sage did not just provide a plot of the sine function. It provided a plotting machine which readers can use to plot any function!

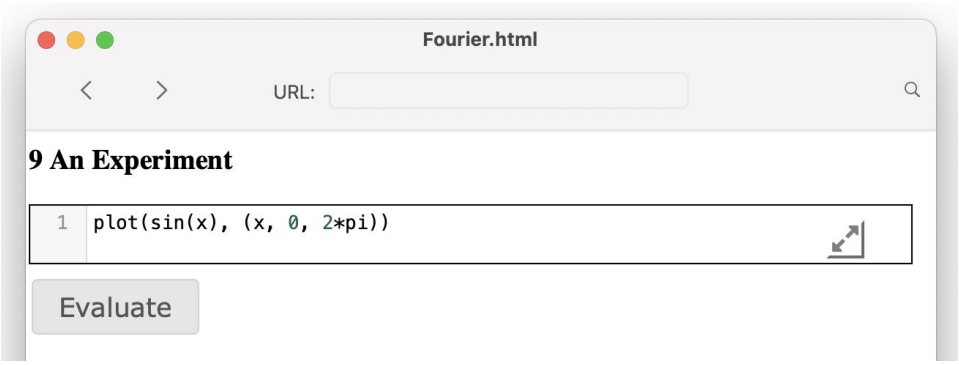


Figure 1. Sage Evaluate button and editable function

6. SageMath

SageMath is an open source alternative to the computer algebra systems Magma, Maple, Mathematica, and MATLAB. The project was created by William Stein, a mathematician at the University of Washington, and first released on February 24, 2005. See https://sagemath.org and https://wiki.sagemath.org. Sage is mostly written in Python, but it integrates many previous open source projects written in C, Lisp, and Fortran. Among these are Gap, Macaulay, Maxima, Octave, and R. The program has been used for serious research on elliptic curves, finite groups, and many other areas, and has an active support group.

The Sage web site has install packages for most major computer platforms. However, the use of Sage described in the previous section does not depend on installing SageMath, either for the author or for the reader on the web. Instead, Sage maintains a server which can run Sage over the web. See https://sagecell.sagemath.org and other links from that page for details.

Our previous Sage example contains a single line to plot a function. However, that line can be replaced by an arbitrary Sage program, which can be several pages long. We list several examples. All of these examples come from web pages at the Sage site; I have just copied and pasted code by others.

Here are two examples of calls to Sage:

```
\begin{html}
 <div class="compute">
 <script type="text/x-sage">
  x, y = var('x,y')
  plot3d(sin(x^2 - y^2), (x,-2, 2), (y,-2,2))
 </script></div>
\end{html}
```

and

```
\begin{html}
 <div class="compute">
```

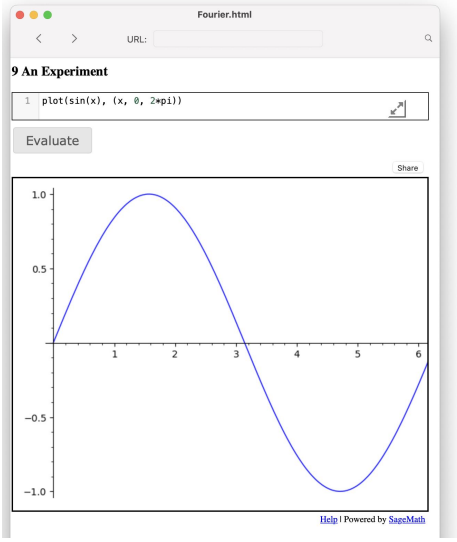


Figure 2. Original plot results

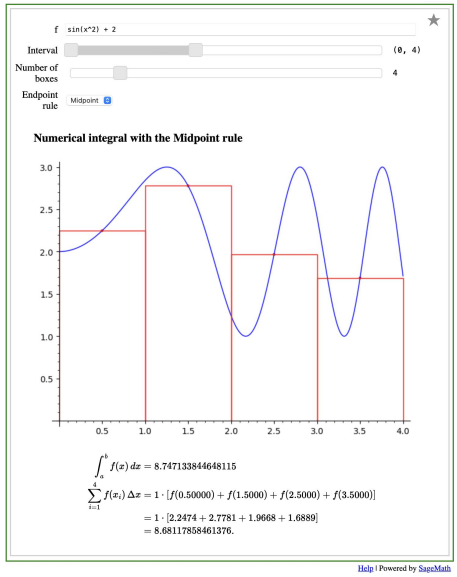


Figure 3. Numerical integration example with Sage

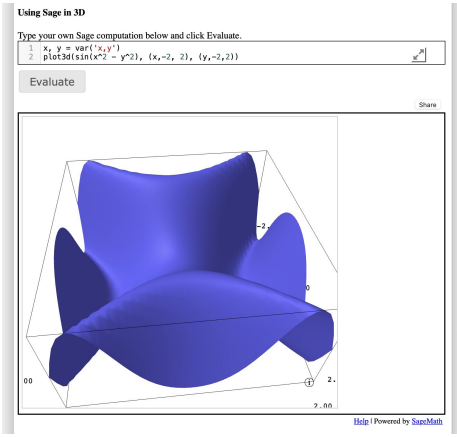


Figure 4. A fancy plot, which in html output can be transformed in real time

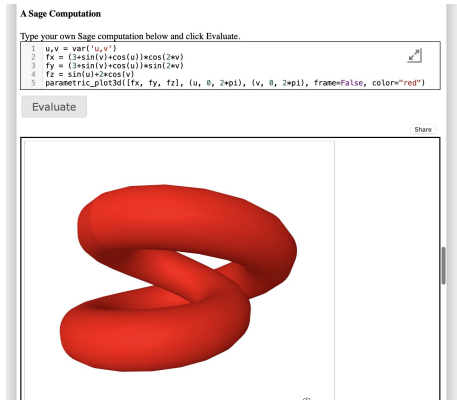


Figure 5. A fancy plot, which in html output can be transformed in real time

```
 <script type="text/x-sage">
u,v = var('u,v')
fx = (3+sin(v)+cos(u))*cos(2*v)
fy = (3+sin(v)+cos(u))*sin(2*v)
fz = sin(u)+2*cos(v)
parametric_plot3d([fx, fy, fz], (u, 0, 2*pi),
       (v, 0, 2*pi), frame=False, color="red")
 </script></div>
\end{html}
```

The output from executing these commands is shown on page 72 (figures 4 and 5). However, this (pdf) article doesn't show the most amazing thing. If these objects are grabbed with the mouse, they rotate and magnify instantly in real time.

For the mathematically inclined, I show two more examples on pages 72 and 74, without giving the Sage code, which can be found on various Sage web sites. Figure 3 illustrates numerical integration. The function can be set by the reader, the number of division points can be set, and the algorithm determining the top of each rectangle can use the value of the function at the left, right, or middle, or the maximum or minimum value. Since Sage can integrate symbolically, the exact value of the integral is shown at the bottom, and finally the numerical approximation is computed and shown.

In figure 6 the Taylor series of a function selected by the user is computed, and both the function and its approximation are plotted.

7. YouTube videos

Did you know that if you right-click while playing a YouTube video, a contextual menu appears allowing you to "copy embed code", which can be pasted into a web page?

I found a lecture by John Maynard, one of the four Field's Prize winners at the International Congress of Mathematicians for 2022. It is fun to watch this video for the depth and clarity of his mathematics. I confess that I also watched because Maynard is left-handed and we lefties need to stick together. I don't understand a word of the code which YouTube provided when I clicked, but I added it to a TEX4ht source page and it worked. Figure 7 shows a frame of the video, and here is some of the source code, as copied from YouTube:

```
\begin{html}
<iframe width="928" height="522"
 src="https://www.youtube.com/embed/kQqBeuk_xQw"
 title="13. Large gaps between primes ..."
 frameborder="0"
 allow="accelerometer; autoplay; ..."
 allowfullscreen></iframe>
\end{html}
```

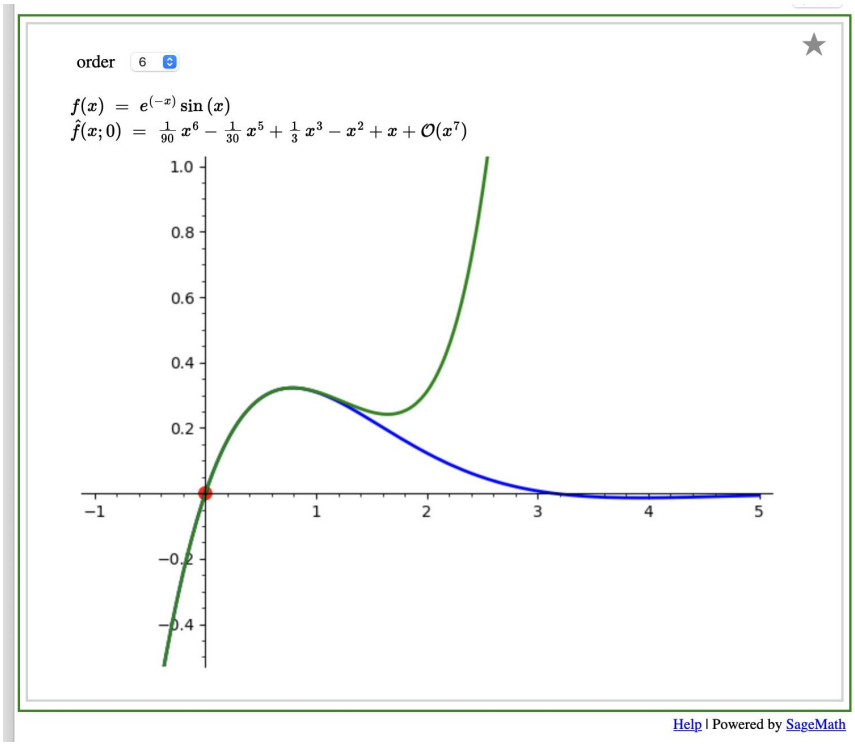


Figure 6. Taylor series of a user-selected function

Figure 7. Frame from YouTube video, playable live in html output

8. Mathematics

Often authors ask a question of readers and provide a multiple choice answer. If the reader answers correctly, they are told to go to the next section; otherwise new text appears explaining why their answer was incorrect.

In a mathematical text, both the question and the various answers will likely contain mathematical formulas. But recall that the interactive material is being written in html and inserted directly in the final document without processing. Are authors expected to write the mathematics in MathML? If they write in LATEX, TEX4ht cannot convert the code to MathML because it doesn't touch the author's html.

The happy answer is that authors *can* directly write LATEX code, even inside the `{html}` environment we've defined. Give it a try by inserting the following code. Recall that inline math can be defined by either a pair of `$` signs or a `\(` and `\)` pair. Display math can be defined by a pair of `$$` signs or a `\[` and `\]` pair.

```
\ifx\HCode\undefined
\else
 \section{New Experiment}
 \begin{html}
   <p>This sentence has <b>bold</b>
   and <i>italic</i> text.</p>
   <p>Also \(y = \sqrt{x^2 + 1}\) and
   $$\int_0^\infty e^{-x^2} \ dx =
   {{\sqrt{\pi}} \over 2}$$</p>
\end{html}
\fi
```

Typeset and you will see the output below (right margin has been truncated).

10 New Experiment

This sentence has **bold** and *italic* text.

Also $y = \sqrt{x^2 + 1}$ and

$$\int_0^\infty e^{-x^2}\,dx = \frac{\sqrt{\pi}}{2}$$

But how is this possible, since source inside an "html pair" is inserted directly in the output without processing?

9. Calling TEX4ht

Originally TEX4ht output small pictures for inline and displayed mathematics. Eitan Gurari unexpectedly died in 2009, and TUG paid him the ultimate compliment by keeping his program alive. Now it is actively maintained by Michal Hoftich.

Due to new developments in MathML and MathJax, there are many ways to call TEX4ht when it is asked to typeset. Let us concentrate on the three most important methods.

Calling TEX4ht using the call

```
make4ht source.tex "mathml"
```

causes TEX4ht to insert MathML code for inline and display equations. This MathML is then rendered by the browser.

Calling TEX4ht using the call

```
make4ht source.tex "mathml,mathjax"
```

causes TEX4ht to insert MathML code for inline and display equations, but call MathJax to render the resulting code.

Calling TEX4ht using the call

```
make4ht source.tex "mathjax"
```

causes TEX4ht to insert LATEX code for inline and display equations, and call MathJax to render the resulting code.

Note that MathJax can render both MathML and LATEX code when it discovers equations in an html document.

On my computer, mathematical rendering using the first method is not as clear as rendering with the other two methods. Integral signs are too small and there are other minor flaws. The first and third methods understand LATEX input for interactive content, but the second does not. These experiments suggest that the third method is the most desirable for interactive code.

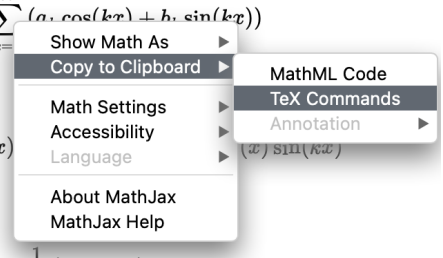
My initial experiments did not go well with the third method. Inline equations were fine, but displayed equations were rendered with static images. Then one day I tried the alternate `\[` notation rather than `$$` and everything worked. I reported this to Michal, and the *very next day* he fixed TEX4ht so both notations are rendered with MathJax. (The LATEX developers do recommend `\[...\]`, by the way.) Please update your TEX Live distribution and typeset using the third method.

10. A MathJax perk

By now, perhaps you have typeset your own document with TEX4ht and MathJax. Select an equation and right click on it. A contextual menu opens offering to copy the equation to the clipboard as either "MathML" or "TeX Commands". Here's a picture:

Theorem 4 (Fourier) *If $f(x)$ takes real values and we write*

$$f(x) = \frac{a_0}{2} + \sum_{k=} (a_k \cos(kx) + b_k \sin(kx))$$



then

$$a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(kx)$$

and

$$c_k = \frac{1}{2} (a_k - i b_k)$$

Select "TeX Commands", copy, and paste somewhere else. You will obtain the LaTeX code for the equation. This code can be copied into any other LaTeX source document.

This remarkably useful feature comes from MathJax and is not available if you call TEX4ht using the first method. Moreover, the menu will offer MathML code, but not LaTeX code, if you call TEX4ht using the second method. But the third method of calling TEX4ht gives LaTeX code.

11. Installing documents on the server

Suppose you typeset a document named `Sample` with TEX4ht and produce `Sample.html`. How should this file be put on a server? The answer is tricky because `Sample.html` itself will not contain any images, so any needed image files must be provided separately. Moreover, TEX4ht generates a support file `Sample.css`, which is also required.

Thus it is convenient to put all illustrations in a folder, named (say) `Graphics`, and refer to these illustrations in the LaTeX source using the pattern `Graphics/plot1` (LaTeX will automatically look for usual image extensions, using whichever is found). Then the web server should contain `Sample.html`, `Sample.css`, and the `Graphics` folder.

12. Using the work of other people

Nothing in this document comes from me. When I discovered that TEX4ht produces completely acceptable web pages, I wondered if it would accept html code and send it unmodified to the html document. I asked Karl Berry, who thought it was possible and asked Michal Hoftich. Michal sent the method described here, but I didn't believe it was sufficiently general. So I started writing a sample document showing that the method could not display math, or handle YouTube videos, or accept Sage code. My sample simply proved the opposite.

I do not know a single MathML tag. I knew the American Mathematical Society recommended MathJax, but didn't know why. I don't understand how these technologies work.

Several years ago I downloaded Sage. But I didn't know that web pages could access a server so students who had never installed Sage could still read web pages with Sage content. When I realized that, I used Sage to graph simple functions. When it displayed a 3D graph and let me rotate it interactively, I almost fell off my chair.

It is strange that I had to learn these lessons over again, because LaTeX is a crucial tool for me and yet I have never read TeXbook; TeX macros are crucial for my life and yet I don't know how to write a macro. We can do things in our lives because of the independent work of thousands of people.

13. PDF and HTML in mathematics

When I was a college sophomore, I took an abstract algebra course from W. Wistar Comfort. His lectures were crystal clear; you could copy the board, read the notes at home, and see every step in its proper logical order.

Later I took courses with a more rough and tumble atmosphere; the instructor seemed to be inventing right in front of our eyes, and sections of the board would be crossed out when a better idea presented itself.

Both lecture styles worked, showing the dual nature of mathematics. To me, pdf is for the final crystalline form of mathematics, and html is for the rough and tumble way it is invented. Euclid is pdf, but Legendre is html, and Euler is both.

Richard Koch
koch@math.uoregon.edu

Progetto grafico di copertina
a cura di Ivan Valbusa

Immagine di copertina:
Typesetting in wood
(Raphael Schaller, 2016)
https://unsplash.com/photos/GkinCd2enIY

Questa rivista è stata prodotta
dal Gruppo Utilizzatori Italiani di TEX
usando esclusivamente software libero.

Versione elettronica per la diffusione via web.

ArsTEXnica – Call for Paper

La rivista è aperta al contributo di tutti coloro che vogliano partecipare con un proprio articolo. Questo dovrà essere inviato alla redazione di ArsTEXnica, per essere sottoposto alla valutazione di recensori entro e non oltre il 14 Agosto 2023. È necessario che gli autori utilizzino la classe di documento ufficiale della rivista; l'autore troverà raccomandazioni e istruzioni più dettagliate all'interno del file d'esempio (.tex).

Gli articoli potranno trattare di qualsiasi argomento inerente al mondo di LATEX e non dovranno necessariamente essere indirizzati ad un pubblico esperto. In particolare tutorial, rassegne e analisi comparate di pacchetti di uso comune, studi di applicazioni reali, saranno bene accetti, così come articoli riguardanti l'interazione con altre tecnologie correlate.

Di volta in volta verrà fissato, e reso pubblico sulla pagina web `http://www.guitex.org/arstexnica/`, un termine di scadenza per la presentazione degli articoli da pubblicare nel numero in preparazione della rivista. Tuttavia gli articoli potranno essere inviati in qualsiasi momento e troveranno collocazione, eventualmente, nei numeri seguenti.

Chiunque, poi, volesse collaborare con la rivista a qualsiasi titolo (recensore, revisore di bozze, grafico, etc.) può contattare la redazione all'indirizzo arstexnica@guitex.org.


30034
9 771828 236001