TEXnica ArS

Rivista italiana
di TEX, LATEX e
tipografia digitale

31 • 2021

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 2021

ArsTEXnica

Rivista italiana di TEX e LATEX

Numero 31, Aprile 2021

Gruppo Utilizzatori Italiani di TEX

Editoriale

Francesco Biccari

Cari lettori, in questo numero di ArsTEXnica troverete due sorprese. La prima, di cui vi sarete già accorti, è la nuova veste grafica della rivista. Il formato passa da A4 a B5 e il testo viene distribuito su una singola colonna. In questo modo speriamo di agevolare la lettura anche agli utenti che la fruiscono in formato elettronico. Per lo stesso motivo abbandoniamo anche il Computer Modern a favore di un font con un taglio più moderno e di facile lettura su uno schermo, il Cochineal, realizzato da Michael Sharpe sulla base del Crimson di Sebastian Kosch. Nei prossimi numeri, in base alle opinioni dei lettori, altre modifiche più fini potrebbero essere messe in atto.

L'altra sorpresa è che in questo numero, per la seconda volta nella storia di ArsTEXnica, viene indetto un vero e proprio concorso: la GUIT *challenge*! I partecipanti dovranno cimentarsi nel preparare un pacchetto per elaborare e mettere in grafico dei dati sperimentali forniti in un file esterno. Il vincitore verrà annunciato nel prossimo GUIT meeting e vincerà il volume *TEX by topic* avvolto nella famosa carta da regalo del GUIT.

A tal proposito vi comunico che è stato deciso di tenere il prossimo GUIT meeting a Firenze, presso il Dipartimento di Fisica e Astronomia dell'Università degli Studi di Firenze. L'appuntamento è fissato per il giorno 16 ottobre 2021. Tutti noi speriamo che le condizioni relative alla pandemia di COVID-19 e alle vaccinazioni siano tali da poter svolgere il meeting in presenza. Se ciò non dovesse essere possibile, il meeting si svolgerà in remoto come, purtroppo, è già avvenuto per la passata edizione.

Per presentare un contributo al meeting dovete inviare il relativo articolo all'indirizzo `arstexnica@guit.it` entro il 31 agosto 2021. I contributi presentati dopo questa data potrebbero essere presi in considerazione per la pubblicazione ma non garantiscono la presentazione orale al meeting. La partecipazione è, come sempre, gratuita per tutti.

Ma vediamo cosa troverete in questo numero di ArsTEXnica.

Nel primo articolo si dà una risposta all'annosa questione: cosa fare quando si vogliono inserire immagini su due pagine o in generale si ha a che fare con immagini molto grandi? LATEX, con i comandi standard, non permette di fare molto in tal senso. Per colmare questa lacuna Claudio Beccari e Heinrich Fleck hanno realizzato un nuovo pacchetto, `swfigure`, che viene presentato qui per la prima volta. L'utente finale potrà finalmente gestire in maniera semplice, grazie a un singolo comando, tutte le situazioni più comuni che si presentano con le immagini di grandi dimensioni e non solo.

Anche il secondo articolo è firmato dal vulcanico Claudio Beccari. Qualche mese fa Claudio ha raccolto due richieste di aiuto: quella di un utente della *mailing list* ufficiale sulla sillabazione in TEX e quello di una sua cara amica che voleva ricomporre in LATEX un libro scritto da suo padre. Entrambi sono albanesi e una delle mancanze di TEX è l'assenza dei pattern per la cesura di alcune lingue, tra cui, appunto, l'albanese. Per cui, dopo una splendida introduzione storica sulla lingua albanese, l'articolo mostra tutti i dettagli tecnici affrontati per la realizzazione del pattern di sillabazione albanese, ora a disposizione di tutta la comunità TEX.

Il terzo articolo: spesso ci troviamo di fronte al problema di dover generare dei documenti tutti simili, come delle lettere d'invito, delle partecipazioni o, per rimanere in un ambito meno romantico, delle fatture. Cioè dei *template* in cui vanno sostituiti dei campi con dei valori presi, per esempio, da un database, eventualmente dopo averli opportunamente elaborati. Ad entrare nel merito è Giacomo Mazzamuto che ci spiega come creare questo tipo di documenti con LaTeX grazie al supporto di Python e Jinja.

Il quarto e ultimo articolo di questo numero di ArsTeXnica non tratta direttamente di LaTeX ma di un problema che molti di noi hanno sicuramente incontrato: cosa fare quando un'immagine raster non è di qualità sufficiente per l'inserimento nel nostro documento e non possiamo ricrearla o sostituirla con un'altra? Gianluca Pignalberi ci mostrerà che in questo caso può essere utile vettorializzare l'immagine. In particolare, si concentrerà su Potrace, un software di vettorializzazione a riga di comando fornito con Inkscape. Dopo un'introduzione al problema della vettorializzazione di immagini raster, verranno presentati degli esempi di applicazione reali.

Purtroppo questo editoriale si chiude con due tristi notizie. Il 16 aprile 2021 si è spento all'età di 81 anni Charles Matthew Geschke, co-fondatore, con John Warnock, di Adobe e co-autore, sempre insieme a Warnock, del linguaggio PostScript. L'altra triste notizia ci riguarda da vicino. Il 25 gennaio 2021 è venuto a mancare Gustavo Mezzetti, conosciuto dagli utenti del forum del GUIT come `letteracdp`. Tutto il GUIT si unisce al ricordo scritto per ArsTeXnica da Enrico Gregorio, suo collega e amico.

Francesco Biccari
Dipartimento di Fisica e Astronomia
Università degli Studi di Firenze
Firenze, Italia
biccari@gmail.com

Managing large figures

Claudio Beccari and Heinrich Fleck

Abstract Sometimes a document may require the insertion of very large images, such that the best solution might be to insert them so as to occupy a full two page spread without leaving any internal margin. Depending on the image aspect ratio it might be preferable to look for other solutions. It looks a simple problem, but the aspect ratio of the original image may set forth some problems that require special treatment. With package swfigure described in this paper we try to create a single command or environment that can handle several display modes.

Sommario In certe circostanze è necessario inserire in un documento delle immagini molto grandi, tali che la soluzione migliore sarebbe di inserirle in modo da occupare due pagine consecutive affacciate così da usare completamente anche il margine interno. Ma, a seconda del rapporto di forma potrebbe essere necessario ricorrere ad altre soluzioni. La cosa sembra semplice, ma il coefficiente di forma dell'immagine originale può dare luogo a problemi che richiedono un trattamento speciale. Con il pacchetto swfigure descritto in questo articolo si crea un singolo comando o ambiente che può gestire diversi modi di presentazione.

1. Introduction

Author H. Fleck already published on ArsTEXnica (Beccari e Fleck, 2012) a class style suitable for typesetting dictionaries; his main use at that time was to publish a dictionary on astronomy. He worked also on other such documents, some of which are in the public domain; he created also another dictionary on seamanship. In both dictionaries he needed to insert large figures, such as figure 1 or figure 2. Even if they are typeset at full page width, they do not show enough details; it would be helpful if this kind of images could be printed in a spread wide fashion. This package, swfigure, can do it and offers more functionalities; it is already installed with the TEX system distributions; the documentation of its code is in (Beccari, 2020a) and a small gallery of examples is in (Beccari, 2020b).

If the document is read on screen, there are around several PDF viewing programs that can display two PDF pages at a time, with the even numbered page on the left and the odd numbered one on the right; it is even possible to annihilate the little gap between the visible pages, so that the screen view displays exactly as a single page containing the spread. With package pdfpages (Matthias, 2020) it is even possible to reformat the output PDF file, so that each page is a complete spread of the original document.

If the document gets printed and bound in a proper way with sewn signatures, each spread of the document displays the adjacent half figures, so that the spine is almost invisible, and the two half pictures appear as a single one.

This is the problem: if we go into the details, apparently the whole procedure consists in splitting the original image into two halves, and to scale them so that their heights do not exceed the text body height, and their widths do not exceed the text body plus the internal

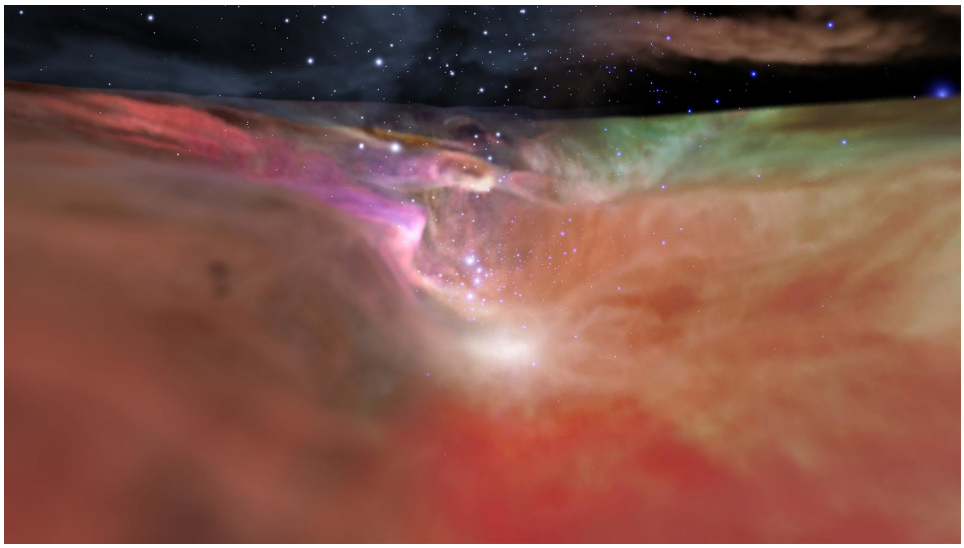


Figure 1. The Orion Valley (Hubble Space Telescope)

Figure 2. A reconstruction of a Greek trireme battle ship (Hellenic Navy Museum)

margin widths. Then such scaled images should be input so that the right or respectively the left sides touch the spine.

The possible scaling factors to reach the above goal are in general different, and in any case one of the two equals its upper limit, while the other one is smaller.

Experience shows that the width constraints are more stringent than the height ones; but the aspect ratio of the original picture might require special treatment.

Sometimes the aspect ratio of the original image is such that the above described display mode is not usable; it is evident that the spread-wide display mode is suited for images that have a height to width ratio (the aspect ratio) lower than 1. If this aspect ratio is slightly smaller than 1, it might be better to use a different display mode, for example a landscape rotated figure. If the aspect ratio is slightly higher than 1 the best display mode might be the standard full page mode. If the original image has an aspect ratio definitely larger or smaller than 1, and looks like a horizontal or vertical ribbon, other display modes might be preferable; if the image is already available in a separate A3 sized (PDF) page, yet another display mode is available to completely fill up the spread pages without leaving around any margin and omitting any resizing of the image;[1]; sometimes for certain slim figures a full paper height display might be preferable, while certain landscape images a full paper width display mode would be more suited.

The aim of the following section is precisely to describe eight different display modes that handle the above described situations, and to create a single command or environment that changes display mode according to an optional argument. In this way the user can choose a different mode without using different commands for each mode; the user can select the default spread-wide display mode and, according to the result, he can change the optional mode-argument and find out which is the best for that particular image.

2. Possible approaches

Here we give a detailed description to the *spread-wide solution* because it is quite tricky. Further on, we describe other possible solutions to the problem of managing large images with different aspect ratios; each of these solutions has advantages and disadvantages depending on the image aspect ratio. We examine several possible situations:

1. The spread-wide situation which forms the core of this paper; it occupies two facing pages with the image and its caption but without any text; the outside margins remain available. so that the spread facing pages may contain header, footer and, in principle, margin notes.
2. The normal full page figure that is defined by the LaTeX kernel.
3. The rotated full page rendering, that might be achieved also with the lscape package.
4. The text-wrapped rendering obtainable with the wrapfig package.
5. The spread-wide situation for pages containing a very long horizontal ribbon-like image at a two page spread top, and filled up with text in the remaining part of the pages.

1. We prefer to deal with ISO standard paper sizes, but the macros we develop are usable also with US paper sizes, although with less accuracy in filling the margins.

6. The total height situation where the image reaches the top and bottom margins of the page, but leaves some space for a lateral caption.
7. The total width situation where the image and its caption reach the lateral page borders, but some space remains underneath for some text.
8. The full spread situation that completely covers the spread pages, possibly without leaving any margin around.

Some of these situations allow freely floating objects; others allow to float them only to a certain extent and require some manual intervention by the user.

The whole process is performed by a single user command that accepts options; depending on the mode related option, it inserts the image with its caption according to one of the above described situations.

It goes by itself that in order to have a smooth output of such large figures, the figure stack should be empty and should remain empty after each one of such large images is shipped out to the output file. Therefore another figure may be declared several paragraphs after any such large image.

3. Spread-wide strategy

The strategy to solve the described spread-wide display mode is the following.

Let us refer to figure 3. The spread has two facing pages containing two text blocks separated by a distance equal to twice the internal margin; therefore the total space for the spread-wide image is a rectangle with an height equal to the text block height H, while the width W is as wide as twice the width of the text block T plus the internal margin width M. Its aspect ratio is therefore $A_1 = HW$.

The initial image dimensions are w and h with aspect ratio equal to $A_2 = hw$. In order to fit the image within the available spread area it is necessary to scale the image so that its scaled width w_s does not exceed W, *and* its scaled height h_s does not exceed H; at the same time the image should be as large as possible while satisfying the above constraints.

Therefore the scaling factor s_f should satisfy these constraints:

$$s_\mathrm{f} = \begin{cases} wW & \text{if } A_2 \leq A_1 \\ hH & \text{if } A_2 > A_1 \end{cases} \tag{1}$$

4. Problems

The reader should consider the aspect ratio $A_2 = hw$ of the original image and the aspect ratio $A_1 = HW = H2T\ M$ of the available space; the former, in principle, might be of any value; the latter is almost certainly smaller than one, depending on the page design and the paper size. Another element to consider is the text block aspect ratio $A_3 = HT$.

If A_2 is significantly larger than 1, the original figure is so slim that a spread wide rendering would not be suitable compared to the same figure in a single page. So avoid using this procedure with images with an aspect ratio $A_2 > A_3$, but use the sigle page rendering or the

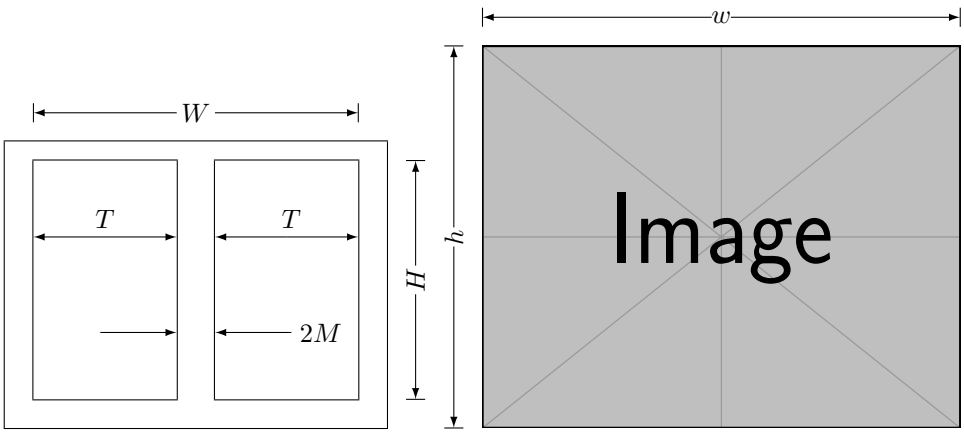


Figure 3. Schematics of the spread page layout and of a dummy image (from package mwe (Scharrer, 2018))

wrapped vertical rendering. If the A_2 is larger than, say, 2, then it would be preferable to use a text wrapped display mode: if A_2 is definitely smaller than A_1 the ribbon like image may find its place on top of a spread, with the rest of the page filled up with text. The display mode, therefore, must be a user choice.

On the opposite, if A_2 is slightly lower than 1, the original image is squarish; dividing it in two halves to set each half on the facing pages of a spread might yield poor results; may be a 90° rotated figure in a single page might be a better solution.

The spread wide solution is preferable when its A_2 aspect ratio is pretty close to A_1. The scaling will produce the best effects, and the spread wide rendering would be a valid solution; it is difficult to specify how close the aspect ratios should be: it depends on the figures themselves and on the inequalities of equation (1).

Another possible problematic situation is when A_2 is rather small, say, smaller than 0.5; the scaling would produce a narrow image strip across the spread, with a lot of white space around; probably this strip might look better if if was at the top of the page, and the rest of the page would be occupied by by the document text. There has been a paper on ArsTEXnica (De Marco e Dominici, 2008) where a similar problem has been solved with another approach; it actually does not address ribbon like images, therefore a different solution must be created.

A final situation is possible when the user needs to include a complex page that is fully and easily readable only if it is printed, or is directly compiled, in A3 format. Of course in this case it would be possible to find other solutions; for example the A3 sheet is printed by itself and then it is joined to the A4 sized document by simply gluing it to the bound document; or, for electronic ones, simply hyperlinking it from the main document. But for screen reading, the insertion as a full spread image, covering also the external/lateral margins would be a solution similar to that for a regular spread-wide one, except for the margins.

The conclusion of this section is simple: the swfigure package can solve the eight described display modes. In few words they are the following.

1. A_2 is very small, say, smaller than 0.5: use a spread-wide horizontal strip.
2. $A_2 \approx A_1$: use a full two page spread-wide rendering.
3. $A_1 < A_2 < A_3$: use a rotated figure in vertical landscape mode.
4. $A_2 \approx A_3$: use a normal full page figure.
5. A_2 is large, say, larger than 2: use a wrapped figure.
6. $A_2 \approx \sqrt{2}$ or $A_2 \approx \sqrt{0.5}$: depending on its contents, it might be preferable to use a full spread figure.

Of course the inequalities of the above enumeration should be evaluated with good sense by a human rather than by an algorithm. The swfigure package takes care of all these situations, but sometimes some human intervention is required in order to specify suitable options.

5. The caption

The LaTeX based typesetting programs can do all the above described processing; in a later section we show the details. But there is another delicate question; where do we put the figure caption?

The answer is not so easy; consider some well typeset magazines, such as, for example, the well known National Geographic, where often they publish beautiful pages much larger than the normal paper width; sometimes such pages are folded one or more times. Depending on the contents of the folded page, either they may label the single sub images, or they are forced to put the caption in the previous or following page.

The book by Gabrielli (1974) is an art book the pagination of which differs almost from page to page, depending on the images that have to be shown; some of them are spread wide images that sometimes allow some space for text or a very long descriptive caption; see for example the photograph in figure 4.

Another industrial art book by Sassi Perino e Faraggiana (1995) is peculiar, not only for its unusual contents, that mixes reproductions of the original design drawings by the engineers or architects who designed the bridges of Turin, but also for the pagination that contains several air borne spread wide images of some of those bridges; furthermore the final trimmed page shape is square; see for example the photograph in figure 5

Now the approach we took was to create something that did not need any page folding; on the contrary we maintained intact the outer margin (except in one case), therefore we decided to put the spread wide figure caption in the right margin, therefore always on the odd numbered page; furthermore we decided to typeset it rotated by an angle of 90°; we decided to choose a measure equal to the scaled figure height; since the spread wide is intended for large figures, we assume that, in spite of scaling the figure, the caption measure might turn out to be close to the text block height. In the full spread case, on the opposite, we decided to vertically typeset the caption close to the image right border overprinting the included image and, depending on this image contents, to choose a contrasting color.

If the caption is not longer than a few lines the result appears to be quite acceptable; of course in future versions we might introduce some modifications or variants in order to produce different results. The sixth, seventh and eighth described modes are already additions to the previous versions functionalities.

Figure 4. A painting in an historical palace of Saluzzo, Italy, from (Gabrielli, 1974)

Figure 5. A bridge on the Po river in Turin, Italy, from (Sassi Perino e Faraggiana, 1995)

6. Other problems

It goes by itself that the spread-wide rendering must start on an even numbered page; we cannot float two consecutive full page floating figures without controlling the parity of the page number of the first float.

It would be possible to delay the output of such figures by using the afterpage package by means of its `\afterpage` command (Carlisle, 2014); the parity controls might be done with the command replacement text, and if the next page number is not even, the whole material to be delayed may be postponed again by means of another nested `\afterpage` command. David Carlisle, the author of afterpage, suggested to one of us[2] to proceed this way, but he warned that `\afterpage` is pretty delicate because it modifies the `\shipout` kernel macro; this is the macro that ships out and appends a complete page to the output file; but, if the next page contains a section header, afterpage may mess up things to the point of loosing its contents; it actually happened with nested `\afterpage` commands when we first approached the problems addressed in this paper.

In general, if it is necessary to use `\afterpage`, it is possible to move around the macro with its contents, by advancing or retracting its occurrence in the source file; in our case we thought it might be better to look for a different solution.

We believe that the author is the best judge to find a suitable position for these spread-wide figures. If close to the point where the user would like to output such figure there is a paragraph end, so that after such a paragraph the typesetting program reverts to vertical mode, the macro that produces the spread wide figure can be used freely; the macro starts with a `\clearpage` so that all queued floating objects are output, and a new page is ready to accept new material; but if the page is odd numbered, a blank page is output, and the user might be forced to move the macro somewhere else.

Unfortunately, in spite of using floating objects, the user clever intervention is still necessary.

7. Other approaches

We have already explained that there are other possible solutions to correctly handle large images.

The most simple one is to float the large image by scaling it to a large fraction of the text block while maintaining its aspect ratio into a float-only page; certainly this is the simplest solution; it requires just the standard command set:

```
\begin{figure}[p]
\centering
\includegraphics[width=\linewidth,height=⟨fraction⟩\textheight,%
    keepaspectratio]{⟨file name⟩}
\caption[⟨lof entry⟩]{⟨caption⟩}\label{⟨label⟩}
\end{figure}
```

2. Private communication.

The advantage is that the image and its caption float where LaTeX finds enough place to output the full page; of course ⟨*fraction*⟩ is a fractional number smaller than one so as to leave some space on the page for the caption and its space above to separate the caption from the image; this might be a good solution for figures that originally are moderately taller than wide. The disadvantage is that a really large image should be scaled to fit the page, and some details might become difficult to distinguish.

Another solution for images that are wider than tall would be to rotate them together with their caption so as to take advantage from the fact that the text block is generally taller than wide. It would be necessary to have available the lscape package (Carlisle, 2000) that allows a simple code such as this one.

```
\begin{figure}[p]
\begin{landscape}
\centering
\includegraphics[width=\linewidth,height=⟨fraction⟩\textheight,%
    keepaspectratio]{⟨file name⟩}
\captionoarglof entry{⟨caption⟩}\label{⟨label⟩}
\end{landscape}
\end{figure}
```

Again, ⟨*fraction*⟩ is a fractional number smaller than one, in order to reduce the scaled original image height so as to leave space for the caption and its label. Actually for these two solutions we preferred to produce the same effects by redefining what is necessary while using the same user command just with a different option. See below in section 10 the code description.

The next two possible solutions are a little trickier and require external packages already input by the swfigure one.

Thin and tall figures would produce a waste of space if inserted with the above methods; is would be more suitable to insert them as wrapped figures; therefore package wrapfig is required. It must be emphasised that wrapfig handles very well the position of a wrapped figure if it is surrounded by normal text, but various limitations exist if the figure falls next to any list or any text that is typeset in a special way. Moreover it does wrong calculations if the surrounding text contains a section header.

It is very difficult the change the page design so as to locally modify the text measure in order to fit a figure wrapped by *any* kind of text. Of course, almost nothing is impossible with LaTeX, but we did not try any other solution different from using the wrapfig package, since we wanted to maintain a single command to handle all such large figures. In section 10 the single user command is described with all its advantages and disadvantages.

Another situation deals with large width and small height figures. These figures require again a spread-wide solution, but done in such a way that the lower part of a page is filled up with the document text. Of course the left and right half images, containing also the caption below the right half one, should have the same dimensions otherwise the first text lines of the facing pages are not aligned.

Eventually there is the situation when a portrait image with an aspect ratio close to $\sqrt{2}$ or a landscape image with an aspect ratio close to $\sqrt{0.5}$ has to be inserted and occupies the full spread without leaving any margin around; in this case the caption must overwrite a small lateral portion of the image; it would not be unusual if the caption had to be typeset in a contrasting color so as to distinguish it from the average background figure color.

8. Examples

A couple of examples with the spread-wide representation are shown on page 11 where we put the screen shots of two spreads. The first one is scaled according to its height, while the second is scaled according to its width; notice that this is done by the software, because it is not the original image that has to be scaled, but each of the two half images that go into different (facing) pages.

In effects, notice that the aspect ratio of the two images is different; therefore the first image is dominated by its height for the proper scaling, while the second image is dominated by its width. This is visible by examining the space between the page headings and the image, and the space between the image right side and the caption on its right. The caption "bottom" is always aligned with the image bottom side. The folios in the respective page foots remain untouched by the image sizes and/or aspect ratios.

They are created with these simple commands:

```
\DFimage{OrionValley}{The Orion Valley}{dfig:ValleDiOrione}

\DFimage{TriremeGreca}[Roman trireme]{A Roman trireme where the beak
   is evidenced together with the three oar opening rows}%
   {dfig:trireme-greca}
```

The command whole syntax is the following:

```
\DFimage[⟨option⟩]{⟨file name⟩}[⟨short caption⟩]{⟨caption text⟩}[⟨label⟩]%
      (⟨fraction⟩)<⟨lines⟩>|⟨width test⟩|!⟨precaption⟩!
```

The argument ⟨*fraction*⟩ is optional and is delimited by round parentheses; it is not used by all ⟨*options*⟩; more details in section 10. The angle bracket delimited optional arguments ⟨*lines*⟩ and vertical bar delimited ⟨*width test*⟩ are used only when ⟨*option*⟩ is VS; again see 10. The exclamation point delimited ⟨*precaption*⟩ may contains any declaration that is going to be used to partially format the caption and is optionally used only with the full spread display mode.

Obviously the ⟨*file name*⟩ refers to the file that contains the original image; for spread-wide, horizontal slim, and full spread display-modes the command splits it in two exact halves to be set on each spread page. The ⟨*caption text*⟩ is the full caption and it is recommended to keep it not too long; in any case the optional⟨*short caption*⟩ is useful for the list of figures; the ⟨*label*⟩ is optional; it is just the argument to the `\label` macro; of course, if this argument is not specified the figure cannot be referenced. The ⟨*precaption*⟩ may be used only with the full spread display mode and may be used, for example, to select a caption color contrasting with the image background.

It is worth noting that on an A4 sized page, typeset in one column mode, inserting images in spread-wide fashion appears as an exaggerated effect. But, if these images are printed in a well bound document with sewn signatures, they make a beautiful scene.

If this document, or any other document containing these images, is read on the screen, and the PDF viewer allows to display a spread at a time, the result is even more impressive. If the PDF viewer allows to display the spread without any gap between the spread pages, it is just perfect. In facts, in figure 6 there are the screen shots taken on a Mac computer by using its Preview PDF viewer: it contains the spread pages of this very paper draft; we thought it would

Figure 6. The screen shots of the spreads containing the same images shown in figures 2 and 1

be more sober to display the screen shots rather than the real spreads. These screenshots and the following ones display a full Mac pane obtained with its Preview viewer, because is the only one we found that does not mark the internal page boundary with a gap or a black line.

We have experienced with several PDF viewers on the three common platforms, Mac, Linux, Windows; certainly the Preview.app on Mac can display gapless spreads. The Okular viewer on Linux displays a gapless spread with a very thin black line dividing the halves of the spread. The multi platform viewers Adobe Acrobat Reader DC as well as Adobe Acrobat Pro and PDF Studio Pro display two pages at a time with a very small gap, but start displaying pages from page 1, therefore the even numbered pages fall on the right and the spread-width images get split in two different views. PDF Studio Pro has a particular view setting *Cover Continuous* to display the cover in a single page and then all the following spreads have the even page on the left. Adobe Reader performs the same as PDF Studio Pro, provided that the document source file uses the hyperref package, and *pdfpagelayout=TwoColumnRight* is among its options.

Skim displays spreads correctly[3], but with a small gap between the facing pages. The internal viewers of TeXShop and TeXworks can display spreads in the correct way, but with a small gap between pages. TeX Pro Reader Lite correctly displays spreads wth a gap in between. WPS and Foxit Reader behave just as Adobe Acrobat Reader DC; TeXstudio does not display any spread. On Windows SumatraPDF correctly displays spreads with a little gap between the pages. Xpdf on all platforms displays page spreads as the Adobe programs do.

In spite of the large number of PDF viewers examined by the authors, not all of them behave the same way and, in particular, at least one of them does not display any spread.

The normal result performed by the LaTeX kernel commands, is shown in figures 1 and 2. So we do not insist on this result.

The rotated image and the slim vertical image are shown in the screen shots of figures 7 and 8.

An example of a slim and spread-wide image is shown by means of the screen shot of figure 9. The caption falls under the right side page, so that `\pageref` points to the second page; this feature implies a suitable wording in order to refer to the figure; in any case if the user reads the document by viewing a spread at a time, and the figure numbers are hyperlinked, the screen window displays correctly the whole spread. Notice that the text in the facing pages shown in figure 9 are perfectly aligned at their top.

Figures 11 and 12 display a Total Height display mode, and a Total Width display mode. Both are taken from odd numbered pages, but on even numbered ones the display is the specular one of that shown in these figures. The total width, depending on the figure aspect ratio leaves some space for normal text; while the Total Height one occupies the whole page.

Last but not least, figure 10 displays a screenshot of an A3 sized fake figure, rotated counterclockwise by 90°; it covers the full spread; the figure has its own margins, therefore the overwritten caption falls on its white space at the bottom; there is no need to color the caption with any contrasting color.

We do not go further by displaying other screenshots that prove the functionalities of the main macro we are talking about; we just summarise the eight display modes with the various schemes shown in figure 13; notice that each white framed rectangle shows the full

3. "Correctly", here and in the following text, means with the even numbered page on the left without any specific set-up as it is necessary with the Adobe viewers and PDF Studio.

Figure 7. Rotated figure

Figure 8. Slim vertical figure

trimmed two pages spread; internally, the white framed rectangles show the text blocks; the grey rectangles display the areas occupied by the images; the red rectangles display the areas occupied by the captions. If the red area is a slim vertical ribbon, the caption is typeset rotated 90° anticlockwise.

9. Command or environment?

As any LaTeX user knows, any command name may be used as an environment name. `\DFimage` is not an exception, but it is a little different from other such environments; in effects,it is prohibited to nest such environments so that it is up to the user to avoid nesting such DFimage environments and/or commands; the package does not provide any test to verify this situation. Moreover, when the command/environment is inserted within a paragraph, it is important that, when its processing is completed, the typesetting mode is restored.

This implies a choice from the user who should ask himself: "When the command is the best choice, and when the environment is more convenient?". We would suggest that the environment is preferable when the large figure appears on the page(s) together with some text; this implies the Vertical Slim and the Horizontal Slim display modes. Certainly it is not convenient with the Normal Figure and the Rotated Figure modes that produce fully floating objects; it is usable with the Spread Wide mode, because it can be used within a paragraph, in order to choose where to start a new page; but it is not really necessary; the Full Spread display mode should be used preferably only at the end of a chapter or at the end of a document, when the command is the best choice.

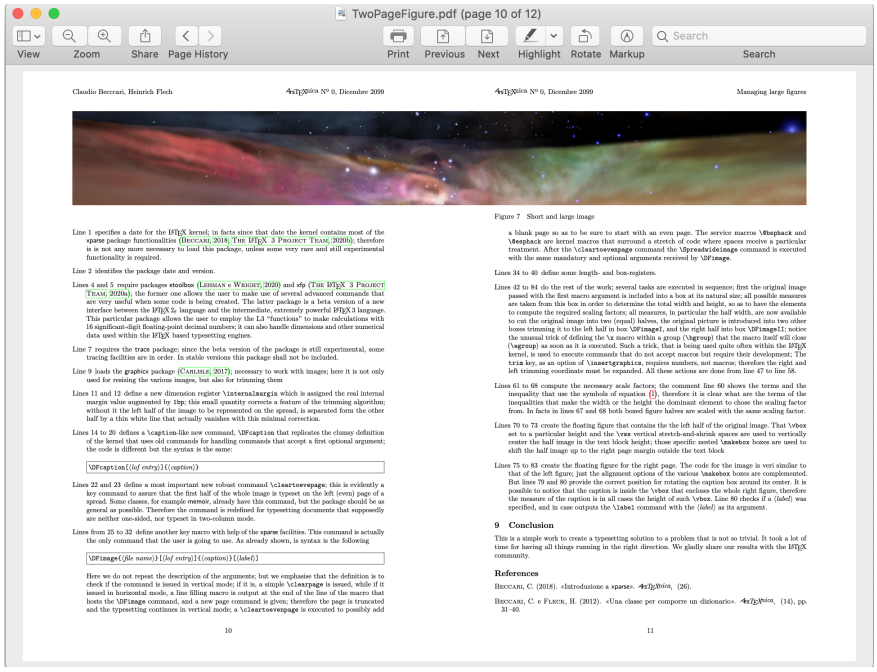


Figure 9. A short and large spread-with image

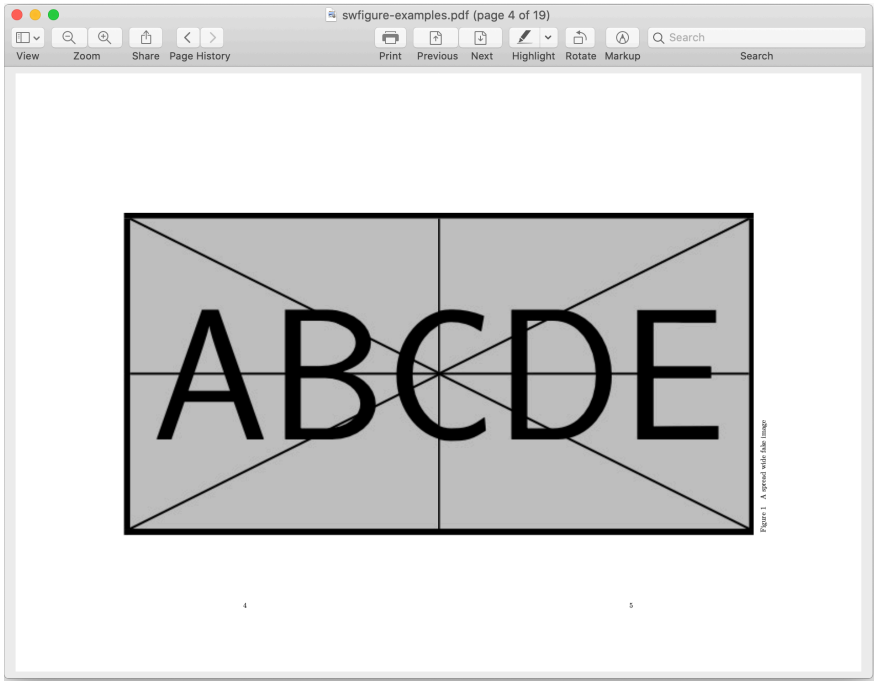


Figure 10. An A3 sized fake image

Figure 11. Total Height figure

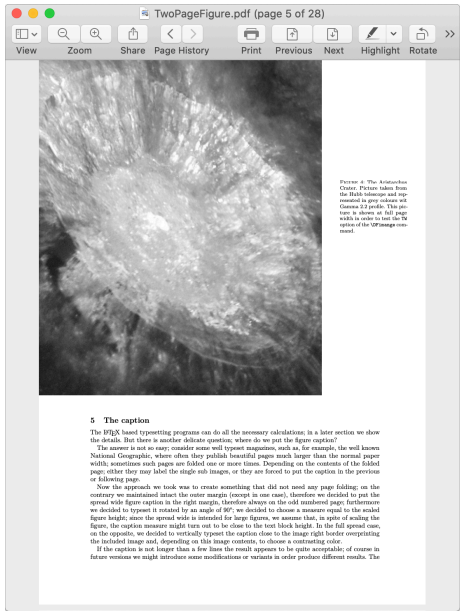


Figure 12. Total Width figure

10. The software

The package swfigure.sty is already available in any updated and complete installation of the TEX system which has available both the code documentation `swfigure.pdf` and a short user manual `swfigure-examples.pdf`; both are readable with the command line `texdoc` followed by either file name. Therefore we redirect the reader to those two documents. Here we describe the peculiar main environment, just to make an example of the type of code we used by employing the xparse facilities and the special string-list analysis LATEX 3 functions, useful to code the various display mode selections.

In facts the single `\DFimage` command or DFimage environment receive some mandatory and optional arguments that allow to typeset the eight varieties of each large image, as described in the previous sections. The default display mode is SW, the one for which this code was initially created; it refers to a spread-wide figure on two facing pages that do not contain any document text. By means of the argument ⟨*display mode*⟩ it is possible to create the other display modes:

```
\DFimage [⟨display mode⟩] {⟨image file name⟩} [⟨lof entry⟩] {⟨caption⟩} [⟨label⟩] %
    (⟨fraction⟩) <⟨lines⟩> | ⟨width test⟩ | ! ⟨precaption⟩ !
```

The various arguments have the following meanings.

⟨*display mode*⟩ This optional argument accepts eight values.

`SW` (Spread-Wide) This is the default display mode to produce a spread that contains only one large image; the caption is vertically set in the right margin; the scaling

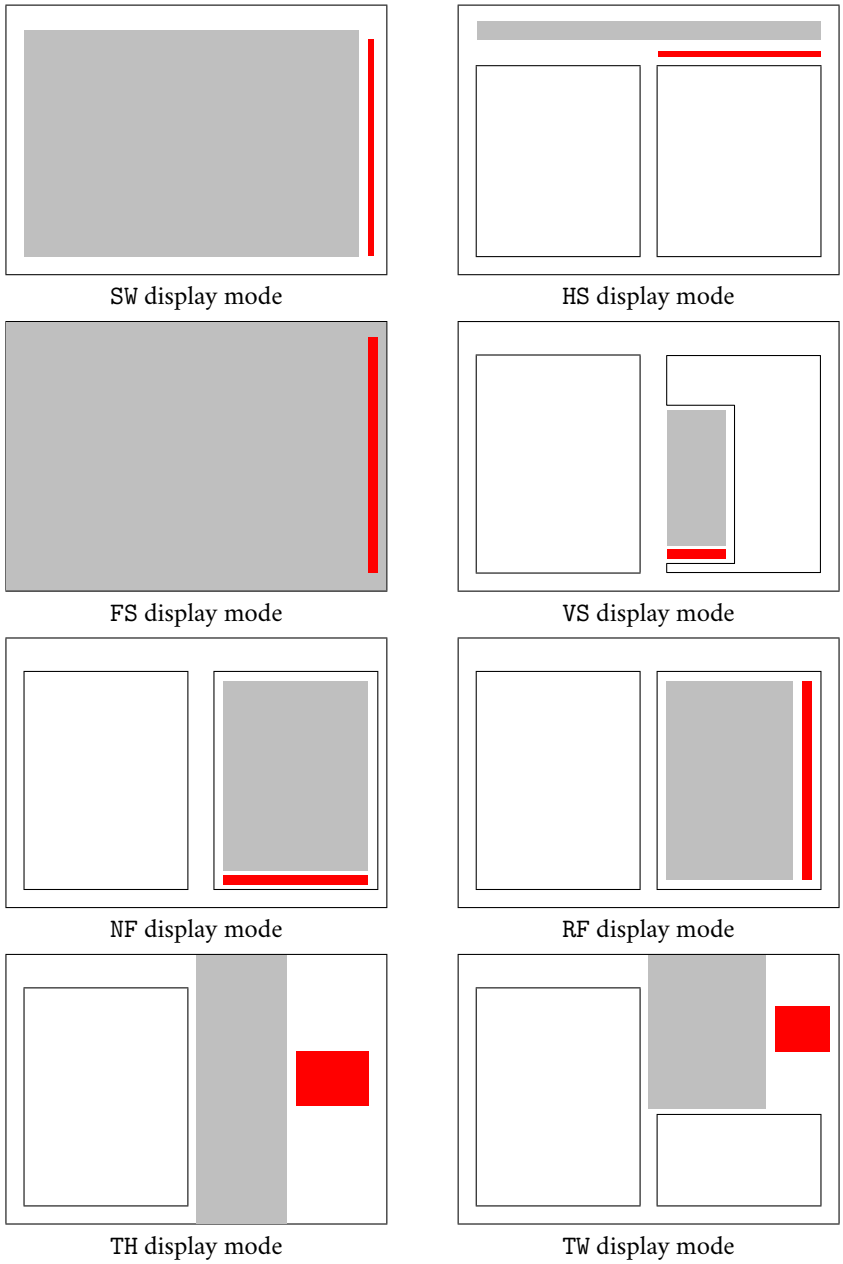


Figure 13. Schematic diagrams of the spreads containing the eight display modes: the grey area is the space occupied by the actual figure, while the red one is the space occupied by the caption.

of the half images is computed in base of the full image aspect ratio in such a way that it exceeds neither the text block height nor the two adjacent spread text block widths including the internal margins.

`NF` (Normal Figure) This is the document-class mode to display an image, with the caption underneath. This is apparently useless, but it is handy to have a single command with several modes; if the user does not like a certain mode he can change it by simply changing its mode option. It is similar to the normal behaviour; actually it performs a little differently but the real feature is that it is provided by the same `\DFimage` command used to provide the other display modes; therefore changing mode implies the simple substitution of NF with another display mode option.

`RF` (Rotated Figure) In this mode the figure is rotated counterclockwise by 90° on a page that contains only the figure with its caption; the ⟨*fraction*⟩ argument may be used to scale the original image height so that it does not fill up the whole page width and some space is used for the caption; depending on the caption length the user might choose another larger or smaller value than the default one (preset to 0.8). See below.

`VS` (Vertical Strip) When the image is a high and thin strip, it is inserted as a narrow figure wrapped by the surrounding text. The vertical scaling ⟨*fraction*⟩ lets the user control the total height of the image with its caption underneath. The ⟨*lines*⟩ integer number can be used in order to correct the actual number of indented lines that the `\begin{wrapfigure}` computes for the figure vertical space. If the aspect ratio of the image to be displayed with this display mode is smaller than 2, the image is not displayed, and in its place a notice is printed in the document that explains why, and suggests the user to select a different display mode. For this mode it is convenient to use the environment in order to have the `\end` command after a suitable number of full paragraphs, so as to assure the correct indentations without bothering about other floating bodies. But the limitation of the aspect ratio not smaller than 2 may be ignored if the user directly employs the wrapfigure environment and lets the figure protrude in the outer margin so as to avoid a too short measure for the indented wrapping text.

`HS` (Horizontal Strip) When the original image is very large but relatively thin, then a spread is created with both half images well aligned at the top of the spread facing pages; the text underneath in both pages is also correctly aligned.

`FS` (Full Spread) This mode is suited to import single-page A3-sized documents without any scaling to fit the spread; it fills up the entire page with no margins around; it is an alternative to attach a folded A3 sheet to a well bound printed document; or to link an external PDF file from inside an A4-sized document. It does not require a special class, such as memoir, that can locally change the virtual paper size; neither it requires a special printer capable of changing paper size on demand of the document to be printed. The result with this display mode might be fundamental if the A3 sized PDF file contained a complicate drawing or a very large tabular material that might become unreadable if shrunk to A4 size.

`TH` (Total Height) This mode is suited to display a tall not so slim image that extends from the top page border to the bottom one; if the aspect ratio is just greater

than unity, but is not large enough, a warning message is sent to the `.log` file , but compilation continues; the caption is typeset in a box in the space available close to the external paper border. The image and the caption boxes are aligned according to their median line.

`TW` (Total Width) This mode is similar to the preceding one, but the image is scaled to a width that equals the sum of the internal margin width and 80% of the text one. The white space between the image and the caption is used for the caption and the objects are aligned as specified above for the full height image display mode. Under these boxes alignment there is place for some text, therefore the page cannot be flushed to the output by means of a `\clearpage` command.

If a ⟨*display mode*⟩ is misspelt, a warning appears in the log file together with the misspelt string and the image file name which it applied to.

⟨*image file name*⟩ The ⟨*image file name*⟩ can be written without specifying the extension; we recall the the allowed extensions are just `.pdf`, `.eps`, `.jpg`, `.png`, `.mps`, the last one of this list belonging to images created with METAPOST. Other graphic files with other extensions may be used, but they require special settings; see the documentation Carlisle (2017).

⟨*lof entry*⟩ The ⟨*lof entry*⟩ is the short caption used to enter a shortened caption text into the list of figures. It is obviously optional, as it is for the standard `\caption` command; its use is recommended when the full caption is longer than one line.

⟨*caption*⟩ The `\caption` argument is the text of the figure caption; we recommend to maintain it short; in most cases two or three lines should be sufficient; with the Vertical Strip mode the caption is typeset in a narrow text block with a small measure, so that the caption text should be carefully handled and possibly hyphenated by the user. With this mode the ⟨*precaption*⟩ optional argument may be used if ragged right typesetting is desired.

⟨*label*⟩ The ⟨*label*⟩ is optional; if it is used, it amounts to whatever the user would write as the `\label` argument; obviously without this argument is is impossible to make reference to the (numbered) figure with the usual mechanism of `\label`, `\ref`, `\pageref` and similar commands provided by other packages.

⟨*fraction*⟩ This optional argument should be delimited by normal parentheses `( )`; it is a fractional number not greater than one; as specified in some previous description items, it may be used with some display modes, and neglected in other ones; its default value is 0.8; the user can specify a different value in order to fine tune the height of vertical objects: rotated images or vertical strips. For the Full With display mode this parameter may be used to fine tune the lateral caption width; in this case it can never exceed unity and we suggest to use values close to the default value 0.8. This parameter can be used also to fine tune the caption with in the `TW` mode. Its default value in general is adequate, but the user may choose values that are not too different from the default value, provided it does not exceed unity. According with the experience we accumulated with many tests, the useful range is between 0.7 and 0.9.

⟨*lines*⟩ In the Vertical Strip display mode the whole work is done by package wrapfig by means of its environment wrapfigure. This environment opening command accepts several arguments, one of which specifies the total amount of lines to be indented to make space for the wrapped figure. Sometimes the environment computes a smaller or

larger value than necessary; it depends from many factors, but the most important of which, probably, is that the native command `\hangindent` handles lines, not vertical spaces, therefore the contents of the wrapping text may cause such "errors". In order to correct this error, the ⟨*lines*⟩ (angle bracket `< >` delimited) optional argument (preset to zero) may be specified in order to correct the computed number of lines by adding or subtracting an integer number so as to get the wrapped figure correctly spaced from the wrapping text. With the tests we made, we found situations where a correction of −1 lines was necessary, and other ones where the correction amounted to 2 lines.

⟨*width test*⟩ is used only in the Vertical Strip display mode; it is the vertical-bar delimited optional argument (`| |`); its default value is 0.25; it is the fraction of the text block width under which the scaling of the wrapped image width should not go below; in facts, by using the ⟨*fraction*⟩ optional value, the image gets shorter than that ⟨*fraction*⟩ of the text block *height*; but by reducing its height, also its width gets smaller; if such width gets too small, the ⟨*width test*⟩ controls that it is not so small that the caption cannot be typeset in a decent way. The preset width test fraction 0.25 assures that the caption measure does not get smaller that one fourth of the text block width; the user can select a different value not greater than 0.5 but as low as zero. Evidently whatever ⟨*fraction*⟩ is selected, the scaled image width never becomes negative, therefore the zero value always produces a false test, equivalent to 'no-test'; on the opposite, the larger this ⟨*width test*⟩ value, the most probable becomes the failure of the test, and therefore a notice is printed in place of the image.

⟨*precaption*⟩ is an optional argument delimited by exclamation marks (`! !`). With the Full Spread display mode there is no place for captions, because the whole space of the spread pages is occupied by the imported material; if this material has its own margins, the vertical caption may not need any special treatment, as in figure 10. But if the imported material does not have a white margin, the caption overwrites a small part of the imported "image", and might become unreadable if typeset with the default black color; this option allows the user to choose a different contrasting color so as to let the caption text emerge form the background; the color should be specified by means of the full declaration `\color{`⟨*color*⟩`}`. Besides the color the user can specify a displacement, a different font, a justification command, and any other declaration that makes sense with a caption.

The actual short code for the environment DFimage is the following:

```
\NewDocumentEnvironment{DFimage}%
  { O{SW} m O{#4} m o D(){0.8} D<>{0} D||{0.25} D!!{} }%
{%
  \SetList{\DisplayModeList}{SW,HS,VS,FS,NF,RF,TH,TW}%
  \TestList{\DisplayModeList}{#1}%
  {\csuse{#1figure}{#2}[#3]{#4}[#5](#6)<#7>|#8|!#9!%
    \fptest{\CompStrings{\@currenv}{DFimage}}{}{\reset@tmode}%
  }%
  {%
    \DFwarning[#1]{#2}[#3]{#4}
  }%
}%
```

```
{%
  \aftergroup\reset@tmode
}%

\NewDocumentCommand\DFwarning{ o m o m }{%
\PackageWarning{swfigure}%
    {*****************************************\MessageBreak
      Option #1\space is not valid. Nothing done \MessageBreak
                                                 \MessageBreak
      Image #2 was not processed                 \MessageBreak
     *****************************************\MessageBreak
    }%
}
```

Notice the LaTeX 3 environment signature with a total of nine arguments, two of which are mandatory and seven ad variously delimited optional argument with or without initial default values; all these preset values have been just described above. Note the third argument; it is the `lof` entry corresponding to the particular environment or command; its default value is the same of the fourth argument, the caption text; this greatly simplifies the handling of the optional `lof` entry implemented in the LaTeX kernel `\caption` command.

The `\SetList` command is a user interface to an internal LaTeX 3 function to create a list of strings; here the strings are the eight valid display mode codes.

The `\TestList` command is user interface to another LaTeX 3 function that tests if the specified display mode is in the list; if the test is true the macro obtained by glueing the text mode code to the word "figure" is called with all the macro arguments except the first one already used. Therefore all the mandatory and the optional arguments are passed to the actual macros that implement the particular display mode, even those arguments that shall not be used by the macro. On the opposite, if the specified display mode is not in the list, it was misspelt, and the warning macro `\DFwarning` is executed; this macro, as explained before, does not output any image, and in its place it writes the reason why: the display mode was wrong and the corresponding image has not been processed.

The environment closing commands just execute the "reset the text printing mode" after the possible environment end; actually the eight display mode specific macros do the same; therefore using the command or the environment is almost the same thing. It goes by itself that some switches are set by the specific macros, so that these final environment commands do not produce errors if the command was used in place of the environment or vice versa.

11. Conclusion

The code in the swfigure package contain various special artifices that were necessary to realise the goal of managing large images. This task was not a simple one; this code resulted sufficient to create typesetting solutions to a problem that is not so trivial. It took a lot of time and several code subversions to fine tune every detail so that all display modes produce the desired results. Some little "features" still remain in the code, but substantially this beta version might remain stable at user level. We gladly share our results with the LaTeX community and

ask the possible users to address comments, suggestions, and bug notices to the first author of this paper; we did our best, but bugs may still be hiding somewhere in our code.

References

Beccari, Claudio (2020a). «The swfigure package — Managing large and spread wide figures». PDF document. Readable with command `texdoc swfigure`.

— (2020b). «The swfigure package — Usage examples». PDF document. Readable with command `texdoc swfigure-examples`.

Beccari, Claudio e Heinrich Fleck (2012). «Una classe per comporre un dizionario». *ArsTEXnica*, (14), pp. 31–40.

Carlisle, David (2000). «The lscape package». PDF document. Readable with command `texdoc lscape`.

— (2014). «The afterpage package». PDF document. Readable with command `texdoc afterpage`.

Carlisle, D.P. (2017). «Packages in the graphics bundle». PDF document. Readable with command `texdoc graphicx`.

De Marco, Agostino e Massimilano Dominici (2008). «longmedal: un pacchetto per medaglioni divisi su più pagine». *ArsTEXnica*, (6), pp. 86–92.

Gabrielli, Noemi (1974). *L'arte nell'antico marchesato di Saluzzo*. Istituto BAncario San Paolo, Torino.

Matthias, Andreas (2020). «The pdfpages package». PDF document. Readable with command `texdoc pdfpages`.

Sassi Perino, Angia e Giorgio Faraggiana (1995). *I trentasei ponti di Torino*. Edizioni del Capricorno, Torino.

Scharrer, Martin (2018). «The mwe package». PDF document. Readable with command `texdoc mwe`.

Claudio Beccari
claudio.beccari@gmail.com
Heinrich Fleck
heinrich.fleck@yahoo.it

Albanian hyphenation

Claudio Beccari

Abstract After a short historical review of the Albanian language the procedure used to create the Albanian hyphenation pattern file is described.

Sommario Dopo una breve esposizione storica della lingua albanese viene descritta la procedura per crearne il file di pattern per la cesura.

1. Introduction

Sometimes in mid 2020 the TEX Hyphen group received a message asking instructions to create the Albanian hyphenation patterns, since this language, at that time, was the only one among those written by means of the extended Latin alphabet that was missing its hyphenation patterns and was typeset with the "nohyphen" metalanguage settings, therefore no hyphenation at all.

I offered the Albanian young man asking for instructions about patterns all the support I could; I sent him some example procedures and many papers on hyphenation, but after that I did not see any Albanian pattern file in the TEX Live distribution and its upgrade procedures. May be they are stil under processing by the TEX Hyphen group, but I tried to experiment myself with a language that I do not know at all.

Albanian has nothing to do with western European languages, and very few people know that language, unless they are Albanian themselves. There are a little more than three million people in Albania who use that language; several hundred thousands autochthonous Albanian in the neighbouring countries, even across the Adriatic see; there are about three million recent emigrants resident all over the world; approximately there are a total of about eight million people around the world who speak, read, and write Albanian.

The Albanian literature is quite recent, compared with the literature of the other mediterranean countries; according to the linguists the first recorded writings in Albanian, or in "proto Albanian" date back to the XIII century. Printed records are very few until the XIX century. Therefore typographical hyphenations is quite recent.

Modern grammars, as usual[1] are very soft on syllabification[2]. Some modern dictionaries often offer both the pronunciation and syllabification of each lemma. But hyphenation dictio-

1. I already complained several times about the fact that grammars generally skip hyphenation and syllabification; if they do, that address elementary school children, that are starting to work with written language; at maximum they repeat the same rules for junior high school students, but always on an elementary bases that are not sufficient for a correct hyphenation practice. Difficult languages such as English, with its numerous varieties, have available huge lists of hyphenated words; romance languages have phonetic rules that are more or less complicated and may refer to the word etymology, but this often requires higher linguistic competence. Unfortunately linguistic scholars consider syllabification a topic too simple for their studies and generally discard it. There are exceptions, though: a famous Italian linguist was the chairman of the Italian Standardisation Office that published the regulation UNI 6461 (1969) some 50 years ago. Such regulation is still valid to day, but it fails with the many new words that entered modern Italian: they have a foreign stem or are compound with foreign words. Any daily newspaper reader can spot hyphenation errors almost every day with such "anomalous" words.
2. According to a trend among the TEX Hyphen Team, "syllabification" is a grammatical notion, while "hyphenation" refers to word breaking at the end of printed lines in order to justify the lines in a printed text.

naries, containing only hyphenated words are very rare, and do exist when the rules are too many to be applied by average educated people; the motto "too many rules equals no rule" is valid with these languages. Of course the Albanian situation appears to be in line with the above description.

At the beginning of TEX a PhD student, Liang (1983), who was working with Knuth (1996), made his doctoral thesis on computer operated hyphenation; his original PhD thesis is not available any more, but it is possible to find on Internet a scanned copy, where the reader can find a lot of information on the data structure of the pattern hash, but very little on the actual process of building a pattern set suitable for a computer: a procedure to process a hyphenation dictionary in order to obtain a set of patterns.

The program Liang wrote was named `patgen` and apparently is available also with TEX Live and the other TEX system distributions, but even with TEX Live the documentation for its use is either absent or so specific that it can't be used with languages different than English; let me remind that English is the only Western Language for which the 26 26 lower and upper case ascii glyphs are sufficient; I would say that all other western language use diacritics; but even English writers have to use them when citing foreign names, but probably the standard TEX accent commands are sufficient for such rare instances of foreign names; unfortunately those accent macros highjack the hyphenation process.

Liang initially worked with a 25 000 entry hyphenation dictionary, just to prove the validity of his thesis approach; the actual pattern file contained in any TEX distribution was prepared with a larger hyphenation dictionary that contained about 100 000 entries; in spite of this, tug (the international TEX Users Group) maintains a list of English hyphenation exception that are not handled by the default pattern set.

Nevertheless nowadays the Unicode encoding, and its transcode UTF-8, is the standard to input source TEX files even when working with pdfLATEX, irrespective of the language being used. Such encodings are very useful with the totality of the other western languages that use the extended Latin alphabet. Albanian is one of such languages. The original `patgen` program cannot be used with non-ascii glyphs, but there exists a version, `patgen2`, that was used when the Omega typesetting program was still maintained by Haralambous (2009); the instructions are not so simple to apply to Unicode encoded fonts[3] glyphs.

Therefore it was necessary to find a different approach to create the Albanian hyphenation patterns. This paper describes such an approach. I anticipate that I followed the same approach I used to test Latin hyphenation while I was trying to create a specific hyphenation pattern set to be used in the composition of Gregorian chants.[4]

2. Short historical description of the Albanian language

The Albanian language has been classified as an Indo-European one by the German linguist Franz Bopp in mid XVIII century. Before, it was considered a language by itself.

3. Translation: I was not able to use it with Unicode encoded glyphs.
4. Eventually I gave up, because of the different points of view between German, French and Italian coworkers, that were unable to find a common base. I admit that I was unsuited for this work because I was working with hyphenation in mind, while the other team members had not only syllabification in mind, but also the chant music; the German-French team continued and is still continuing that work.

Table 1. The 36 letters of the Albanian alphabet

A a	B b	C c	Ç ç	D d	Dh dh	E e	Ë ë	F f	G g	Gj gj	H h
I i	J j	K k	Ll	LL ll	M m	N n	Nj nj	O o	P p	Q q	R r
RR rr	S s	Sh sh	T t	Th th	U u	V v	X x	Xh xh	Y y	Z z	Zh zh

Linguists found several varieties in the actual Albanian language territory that includes also Kosova and some neighbouring areas in Montenegro, Northern Makedonia, Greece, and Italy. At the same time the Slavic languages and Greek are the obvious adstrate ones from where many words entered into the common Albanian language.

Other adstrate languages derived from occupations by other people, particularly important Turkish and Italian. This influenced also the way of writing; it is reported that Albanian in the past was written with Latin, Greek, Cyrillic, and Turko-Arabic alphabets. On the modern Albanian territory two main regional languages are spoken and used to be written: the northern Gheg and the southern Tosk. Outside the country the Arbëresh variety survives in central southern Italy; it is not due to a recent Albanian immigration; it is used by the descendants of the followers of Gjergj Kastrioti, known as Skanderbeg, a national hero who fought against the Ottomans in the XV century; eventually he was awarded some counties in central southern Italian peninsula by Ferdinand II, king of Naples. This variety of Albanian is the one that was used in the last part of medieval times and is spoken still today in some areas of the Italian Adriatic coast.

In 1909 a new spelling of Albanian and in 1972 a new Albanian "koinè" language was agreed on for the whole nation; this koinè does not supersede the regional languages, but is generally understood by everybody while its orthography is the official one and is used all over the country.

3. The Albanian letters

The alphabet of modern Albanian contains 36 '"letters" shown in table 1.

The word "letter" is quoted because it conflicts with the meaning we usually associate to this name; we usually call the two character signs with the name of digraphs; such single or two character signs ("letters" according to the Albanian terminology) represent phonemes. In the following, I will try to avoid the use of the word letter, in order to avoid misunderstandings, and use the words character or digraph depending on how many characters form each sign.

Apparently there are no diphthongs as in other languages: the diphthongs generally are groups of two vowels of which either the first or the second is an unstressed closed or semi closed vowel, which in these cases plays the role of a semiconsonant. The character "j" plays the role of a semiconsonant when preceded or followed by a vowel, but it plays the role of a diacritical sign when it is part of a digraph; therefore sometimes the role of a "j" is ambiguous for what concerns a computer; this device does not know the pronunciation of each word but operates only on its spelling, therefore on the character string that forms a word; for creating pattern files this ambiguity is a problem. Furthermore the language uses the apocope or elision that is handled with the apostrophe without any spaces before or after it.

Anybody who would like to know which phonemes the characters and the digraphs represent, should consult a bilingual dictionary to and from Albanian; more simply the

blog at `http://www.albanianlanguage.net/` (in English) or the Wikipedia page at `https://it.wikipedia.org/wiki/Lingua_albanese` (in Italian, but pretty well done) offer lots of information.

4. The approach to make Albanian patterns

What is explained above partially explain the premises of my approach.

1. Without any available document written in a language that I could decipher, this first obstacle is over if I could find somebody who not only knows the Albanian language, but also its grammar. Luckily enough, I have been knowing Sabina Kolici for many years; she was born and raised in Albania, where she got a university degree[5] in Albanian Literature; She has been living in Italy for more than 20 years. I proposed her to help me in this task and she accepted with enthusiasm.
2. We needed a text to work on; Ms Kolici chose a couple of chapters from an Albanian novel that she could get as a computer file, so as to use it as a test file; it is not important the name of the novel, because any book would have been usable for our purpose; Ms Kolici selected it because that novel did not contain special names connected to this or that professional discipline. Ms Kolici processed for me an initial file where the words were rearranged one per line.
3. I further processed the initial file in order to get rid of any hyphens that could have remained in such a file; I got rid of quotation marks, parentheses, punctuation signs, and the like; I lowercased all words and entered this long list of about 6000 words, into a spreadsheet column; with the facilities of the spreadsheet processing program, I sorted all words and eliminated all duplicates; the list was reduced to about 2600 words.
4. I created an initial `hyph-sq.tex`[6] where I introduced an initial set of pattern concerned with each and all single or double consonant signs at the beginning or at the end of words: i.e. a list of `.b2 2b. .c2 2c. .ç2 2ç.)....y2 2y. .z2 2z. .zh2 2zh. -- a very elementary set.`
5. I created a LuaLaTeX source file that would load the above pattern file and process the word list, in order to print out all input words in the hyphenated form. More on this file in the following section.
6. I would pass the hyphenated word list to Ms Kolici who would correct the wrong hyphen points and would send me the corrected list; I would modify the `hyph-sq.tex` file and repeat the process from the preceding step.

Of course this process would terminate when the corrected hyphenated word list would not contain any errors.

Too much work done by hand? Well, yes and no; missing an intelligent program that could compare the hyphenated word list with an initial hyphenated word list (as `patgen` or `patgen2` could have done) it is difficult to do it in a different way; moreover with this

5. Her degree is equivalent to a master degree of the European 3+2+3 university levels; this agreement was undersigned by the European member countries about 20 years ago; other European countries, although not members of the European Union, adopted the same system.
6. The string "sq" is the ISO code for Albanian: *Shqip* in Albanian.

do-rësh-kri-min	du-ro-i	ë-mës	fat-ke-që-si-a
do-re-zës	dy	em-rin	fat-ke-qe-ve
do-run-ti-na	dy-fish-të	en-de	fe-je-së
do-run-ti-në	dy-ja	ën-dë-rron-te	fe-je-sën
do-run-ti-nën	dy-ja-ve	ëndrre	fe-mër
do-run-ti-nës	dy-ja-vor	e-nësh	fë-mi-jët

Figure 1. A specimen of the output hyphenated word list

procedure the corrected word list and the corrected pattern list are processed by humans, who may also produce errors, but certainly are more intelligent than a program. Moreover `patgen` and `patgen2` require repetitions that do not reach the zero error solution, but a solution with the least number of errors. Our procedure stops when errors are absent.

I would not claim that our procedure is infallible; certainly it is not applicable to languages that do not mark the pronunciation; for example it is not applicable to English that contains homographs that are pronounced differently according to the role the word plays in a sentence; noun vs. verb; noun vs. adjective, and so on. A typical example is "to record" and "the record", but I could list many more. If accents were used, the words would not be homographs: "to recòrd" vs. "the rècord". Of course I am not invoking the use of stress accents in English (although this procedure was taken with ecclesiastic Latin); I am just remarking that a program that hyphenates words basing its rules on the spelling, cannot correctly hyphenate homographs that are pronounced and hyphenated in a different way depending on the role they play in a sentence.

5. The hyphenation testing source file

Here is the code for the LuaLaTeX testing file.

```
% !TEX TS-program = LuaLaTeX
% !TEX encoding = UTF-8 Unicode
\documentclass[12pt]{article}
\usepackage{fontspec}
 \defaultfontfeatures{Ligatures={NoCommon,
 NoDiscretionary, NoHistoric, NoRequired,
 NoContextual}}
\setmainfont{CMU serif}

\usepackage{luacode}
\usepackage{testhyphens}
\usepackage{multicol}

\begin{luacode}
local patfile = io.open('./hyph-sq.tex')
langobject = lang.new()
lang.patterns(langobject, patfile:read('*all'))
patfile:close()
\end{luacode}
```

```
\patterns{
2’2
.a1jo. a1a
1b .b2 2b. b2l 2bsh
1c .c2 2c. 2cj 2cn 2ct
1ç .ç2 2ç. 2çs ç2k
1d d2h .d2 2d. d2j 2dn d2r 2drr 2dt d2shmd2h 2dh. 2dhj2dht 2dhsh 2dhj 2dht
   dh2r dh2j
e1a e3ll e1u
ë1a
1f .f2 2f. f2l f2r 2fs 2ft 3f2sh 2f2t.
1g .g2 2g. g2j 2gj. 2gjv 2gl 2gm 2gr 2gt
1h .h2 2h. 2hd 2hj 2hm 2hn 2ht 2hrr
i1a i1e i1u .i2k3i .i2k3j
1j2 .j2 2j. 2j3c2 2j3d 2j3m 2j3p 2j3r 2j3t 2j3v 2j3s 2jf. j4tp 2jt. j3sh2m
1k .k2 2k. k2j 2kl 2km 2kth. k2r 2kt 2ks 2ksh
1l .l2 2l. 2lb 2lç 2lf 2lj 2lm 2ln l3n2g 2ls 2lt
l2l2 4ll. 2ll3s 4ll3z 2ll3k 4ll3gj 2ll3n 2ll3t 4ll3z
1m .m2 2m. m2b mb2j mb2l mb2r m2j 2m3n2d 2mt 2mr 2m3sh2 2m4sh. 2m1v
1n .n2 2n. .ng2r 2nc 2nd n2dm n2dv n2d3sh 2ng 2nk 2nsp 2nsh n3sh2m 2nt 2nv
   2nx 2nz n2j 2njt 2nj. 2njv
o1i
1p .p2 2p. p2j 2pn 2pt p2je. 2ps p2r pa2s3her .pe2r3af .pë2r3af
1q .q2 2q. 2qj 2qk 2qm 2qn 2qt q2v
1r .r2 2r. 2rt 2rb2 2r2b3r 2rc 2rç 2rd 2rc2rd 2rf 2rg  2rh .ri3n2d 2rk 2rl
   2rm 2rn r2n3d2 2rp 2rq 2rs 2r3sh2m 2rdh r2dht 2r3dr 2rj 2rv 2rz
r2r .rr2 2rr. 2rrj 2rrk 2rrm 2rrn 2rrt 2rrs
1s .s2 2s. 2sb 2sc 2sd 2sf 2sg 2sj 2sk 2sm 2sn sn2k 2sp 2ssh 2st st2r
   3s2je2ll sk2ll s2ve.
s2h 2sh. .sh4 2shm 3sh2mj 2shj .sh2j sh2k 2shk. sh2n shn4d sh2p .sh2q
    4sh3k2r 2shq 2sh3nj 2shpr 3sh4pj 3sh4pr 2shr 2shs 2sht .sh2t 3sh2te.
1t .t2 4t. 2tk t2j 2tm 2tn 2tp 2t3sh2m t2r 2tv
t2h .th2 2th. 2thç 2ths 2thç 2thf 2thm 2tht
u1ar. u1a u1e
1v .v2 2v. 2vr v2j
1x .x2 2x.
x2h
y1
1z .zb2r .z2 2z. 2zm 2zn 2zj 2zs 2zt 2zv z3sh2m
z2h .zh2 2zhd
}
```

The obtained Albanian pattern file

```
20
21 \language=%
22 \directlua{tex.sprint(lang.id(langobject))}
23
24 \advance\textwidth 60mm
25 \advance\oddsidemargin -30mm
26 \advance\textheight 50mm
27 \advance\topmargin -20mm
28
29 \begin{document}
30
31 Language: \the\language
32
33 \bigskip
34
35 \begin{multicols}{4}
36 \lefthyphenmin=1
37 \righthyphenmin=1
38 \begin{checkhyphens}
39 acarime
40 acarimi
41 afërta
42 ...
43 zyrë
44 zyrtarë
45 \end{checkhyphens}
46 \end{multicols}
47 \end{document}
```

Some comments are in order.

1. Line 1 and 2 are the so called "magic lines" that inform the editor that the file has to be saved with the UTF-8 transcode encoding, and the the file has to be processed with LuaLaTeX. Those shell editors that understand such "magic lines" act as specified according to such lines, and use the LuaLaTeX typesetting engine even if their default engine is a different one. If the editor does not understand these lines, the user can understand them and s/he can act accordingly.
2. Line 3 is obvious.
3. Lines from 4 to 8 use fontspec to specify which fonts have to be used for the output; in this case we select the normal Computer Modern OpenType fonts for the main (serifed) font.
4. Lines from 10 to 12 specify the packages to load: luacode is to allow to use Luacode in this input file; testhyphens is a package to print hyphenated words according to the patterns of the current language; multicol prints the text in a specified number of columns.
5. Lines from 14 to 23 are the specific lines that involve the `hyph-sq.tex` Albanian language pattern file and lines 22 and 23 set such pattern file as the default. This is

the special feature of LuaLaTeX we are exploiting; pdfLaTeX and XeLaTeX require that all hyphenation pattern file are preloaded while constructing their specific kernel `.fmt` format file; LuaLaTeX, on the opposite, can load hyphen pattern files even at run time.

6. Lines from 25 to 28 are low level commands to enlarge the typesetting area; by so doing and selecting to typeset the results in four columns we are pretty sure that no hyphenated word is going to be split on two or more lines.
7. Line 32 prints out the number of the current language; missing this line, one cannot be sure that the new pattern files have been correctly loaded and are the current ones in force.
8. On line 36 the molticols environment is opened with a specification of four columns; it shall be closed by the end of the file, in this numbered listing at line 47.
9. In lines 37 and 38, the minimum lengths of the first and last syllables of each word are both set to one character; actually these numbers shall not be the default values for actual typesetting; such actual values are specified by `albanian.ldf` or `gloss-albanian.ldf`. Here we set these values to 1 in order to verify that digraphs don't get split by the hyphenation process.
10. On line 39 the checkhyphens environment is opened and shall be closed at the end of the word list. This is the specific environment that hyphenates each word at *all its hyphen points*, and outputs the corresponding string so as to have the full word hyphenation to check with the "hand made" controlling process; human eyes are very good in noticing wrong break points.

A specimen of the output file is shown in figure 1 on page 5; the actual complete file is 15 pages long.

6. The final pattern file

The described procedure worked fine; the final hyphenation pattern file for Albanian turned out as shown on page 6.[7] The total number of patterns resulted to be 310, not so different from the corresponding number for some other romance languages, although Albanian does not belong to that language class.

It must be noticed that the pattern set shown on page 6 does not produce any errors with the 2600 words used to make them up. If tested with other texts it might still produce some hyphenation errors. However this is normal; at the beginning all the other pattern files produced errors and the corresponding pattern files had to be updated; nevertheless we are confident that this pattern set may produce correct results with the majority of texts that use common words. We hope that the community of Albanian LaTeX users continues to upgrade the Albanian pattern sets while they use the hyphenation utility with their actual documents. If they do not, each author has to create his/her hyphenation exception lists; of course this is not terrible, but it is what happens when a LaTeX user community is too small and/or does not work as a community. We hope that the Albanian community will be active in caring and maintaining their language facilities provided by the TeX system.

7. The pattern file is built by hand; the program that has to prove its correctness does not care if each line contains many patterns, but the pattern files distributed with any TeX installation have one pattern per line. Here, in order to save space, we selected to show several patterns per line.

7. Example

Here we show a short list of fully hyphenated Albanian words:

be-së-shke-lë-sin bu-da-lla-llëk di-nji-te-tin dëm-tu-a-ra do-rësh-kri-min fi-llu-an
go-di-tje-je ha-men-djesh he-rë-pas-her-shme kë-mbën-gul-ja ko-rri-do-rit mby-llu-ra
mi-rë-sje-lljes ngul-çi-mës një-ko-hë-sisht pshe-rë-ti-u s'u-lë-ri-u shp je-go-hej
u-dhë-hiq-nin zma-dhu-ar

Such words are a small sample of the results obtained with the used procedure where hyphenation was performed with both end syllable lengths set to unity. In the real usage of the Albanian language such parameters would be set to 2.

Conclusion

The `hyph-sq.tex` file has been submitted to the TEX Hyphen group; they are going to extract from this single file the necessary variants to be used with the other typesetting systems, as they usually do for all languages.

Acknowledgments

Creating Albanian pattern files requires a good knowledge of the language; I know how to make pattern files, but I do not know Albanian. My deep thanks are therefore due to Ms Sabina Kolici who contributed to this work with her constant support.

References

Beccari, Claudio (2014). «Greek and Latin hyphenation — Recent advances». *ArsTEXnica*, (18), pp. 87–96.

Carette, François e Arthur Reutenauer (2015). «Polyglossia: an alternative to Babel for XeLaTeX and LuaLaTeX». `http://mirrors.ctan.org/macros/latex/contrib/polyglossia/polyglossia.pdf`.

Haralambous, Yannis (2009). «A small tutorial on the multilingual features of PatGen2». PDF document. Readable with line command `texdoc patgen2`.

Knuth, Donald E. (1996). *The TEX book*. Addison Wesley, Reading, Mass., 16ª edizione.

Liang, Frank (1983). *Word Hy-phen-a-tion by Com-put-er*. Tesi di Laurea, Stanford University.

UNI 6461 (1969). *Divisione delle parole in fin di linea*. Ente Italiano di Unificazione, Milano.

Claudio Beccari
claudio.beccari@gmail.com

Generazione di documenti LaTeX con Python e Jinja

Giacomo Mazzamuto

Sommario Questo articolo illustra come creare documenti LaTeX con Jinja, un motore di *template* per Python, mettendo in evidenza le potenzialità fornite dall'avere a disposizione un linguaggio semplice e ricco come Python all'interno di un documento LaTeX. L'articolo è composto da un *tutorial* che dimostra le funzionalità di base di Jinja e da un esempio di configurazione più avanzata finalizzato a rendere più gradevole la sintassi di Jinja all'interno di un documento LaTeX. Infine, è mostrato l'esempio della generazione di una fattura.

Abstract This short article describes how to use Jinja, a template engine for Python, to create LaTeX documents harnessing the power of a simple and rich language such as Python. The article begins with a tutorial showing the basic usage of Jinja, followed by a more advanced configuration aiming at making Jinja's syntax more LaTeX-friendly. Finally, an example involving the generation of an invoice is shown.

1. Introduzione

Jinja[1] è un motore per modelli (o *template engine*) per Python[2]. Come molti *template engine*, Jinja è nato in ambito web dove viene usato principalmente per creare pagine HTML in modo dinamico da parte di un server di *backend*. La sua sintassi si ispira a quella del motore di template di Django, un popolare *web framework* anch'esso scritto in Python.

In generale, un *template engine* non è altro che un sistema per generare *file* di testo in base a un modello di partenza — il *template* appunto — e trova quindi applicazione nella generazione dinamica di documenti di testo di qualsiasi tipo, siano essi pagine HTML, documenti XML oppure sorgenti LaTeX. Per un'introduzione più dettagliata sul concetto di *template* si veda Giacomelli e Pignalberi (2015). Personalmente mi è capitato di utilizzare Jinja persino per la generazione in serie di documenti SVG, il diffuso formato di grafica vettoriale che, infatti, è un formato testuale, essendo basato su XML. Tutto quello che si può fare con Jinja e LaTeX si può ottenere, per esempio, anche con LuaLaTeX grazie al linguaggio di *scripting* Lua. Tuttavia, date la popolarità e la semplicità di Python, l'ampiezza della sua libreria standard e la vastità di pacchetti disponibili, vale la pena di esplorare le potenzialità offerte dall'unione di questi due mondi: Python e LaTeX. Sarà infatti molto facile creare documenti LaTeX con dati prelevati dinamicamente da un *database* o da Internet, o semplicemente si potrà utilizzare Python come linguaggio di *scripting* all'interno di documenti LaTeX.

In questo articolo vedremo come utilizzare Jinja per generare dei semplici sorgenti LaTeX, ed esploreremo alcune opzioni di configurazione per renderlo più "amichevole" in quest'ambito

1. https://palletsprojects.com/p/jinja/
2. https://www.python.org

d'uso. Si presuppone che l'utente abbia già installato un interprete Python sul proprio sistema, mentre per installare Jinja è sufficiente eseguire questo comando:

```
pip install -U Jinja2
```

2. Guida all'uso

Consideriamo il seguente "modello" di documento LaTeX:

```
\documentclass{article}
\begin{document}
L'autore di {{myvar.title}} è
{{myvar.author}}.
\end{document}
```

Nell'esempio qui sopra, si può facilmente riconoscere che si tratta di un modello perché alcune porzioni di esso (`{{myvar.title}}` e `{{myvar.author}}`) sono dei segnaposti che verranno sostituiti dalle corrispondenti variabili quando il *template* sarà "compilato". Utilizziamo il seguente *script* Python per compilare il *template*:

```
from jinja2.nativetypes import \
    (NativeEnvironment, Environment)
from jinja2.loaders import \
    FileSystemLoader

env = Environment(
    loader=FileSystemLoader('.'),
)

mydict = {
    'author': 'James Joyce',
    'book': 'Dubliners',
}

templ = env.get_template('template.tex')
templ = templ.stream(myvar=mydict)
templ.dump('output.tex')
```

Commentiamo il codice passo passo. Dopo aver importato i pacchetti e le classi necessarie, alle righe 6–8 si inizializza Jinja configurandolo affinché carichi i modelli cercandoli nel *filesystem*, in particolare nella cartella corrente (`'.'`). Alle righe 10–13 si definisce un *dizionario* con due chiavi (`author` e `book`). Alla riga 15 si carica il *template*, supponendo di aver salvato il documento LaTeX di cui sopra in un file chiamato `template.tex`. Infine, alla riga 16 si "compila" il template avendo cura di passare tutte le variabili di cui abbiamo bisogno (in questo caso, soltanto `myvar=mydict`), e il risultato della compilazione sarà salvato in un file chiamato `output.tex` (riga 17). Dopo aver eseguito questo *script*, il file `output.tex` così generato conterrà il seguente documento LaTeX:

```
\documentclass{article}
\begin{document}
L'autore di Dubliners è James Joyce.
\end{document}
```

Come si può vedere, i due segnaposti sono stati sostituiti con i corrispondenti valori del *dizionario* `mydict` che abbiamo definito nello *script* Python e che abbiamo reso disponibile al *template* come variabile chiamata `myvar`. Questo è un esempio illustrativo, dove per motivi di semplicità i valori da sostituire sono definiti (*hard-coded*) direttamente nello script; più probabilmente, in un caso reale di utilizzazione, i dati sarebbero prelevati da un qualche tipo di banca dati come un *database* MySQL, o anche semplicemente letti da un file. Le doppie parentesi graffe racchiudono un'*espressione*, al loro interno sono cioè ammesse semplici espressioni Python come mostrato in questo esempio:

```
\begin{document}
Il cubo di 4 è {{4**3}}
La mia stringa in maiuscolo:
{{'prova'.upper()}}.
\end{document}
```

che diventa:

```
\begin{document}
Il cubo di 4 è 64.
La mia stringa in maiuscolo: PROVA.
\end{document}
```

Complichiamo leggermente le cose passando al motore di *template*, anziché un singolo *dizionario* come fatto sopra, una *lista* (`books`) contenente due *dizionari*:

```
books = [
    {'author': 'James Joyce',
     'title': 'Dubliners'},
    {'author': 'Donald Knuth',
     'title': 'The Art of Computer '
              'Programming'}
]
templ = env.get_template('template.tex')
templ = templ.stream(books=books)
templ.dump('output.tex')
```

Modifichiamo inoltre il file `template.tex` in questo modo:

```
\begin{document}                                    1
{% for b in books %}                                2
L'autore di {{b.title}} è {{b.author}}.             3
{% endfor %}                                        4
\end{document}                                      5
```

Mentre come abbiamo già visto le doppie parentesi graffe delimitano un'*espressione*, la parentesi graffa seguita dal segno di percentuale delimita un *blocco* che racchiude uno *statement* — ad esempio: un ciclo `for` come in questo caso, oppure un costrutto condizionale (`if`). Dopo aver compilato il *template*, il risultato sarà:

```
\begin{document}

L'autore di Dubliners è James Joyce.

L'autore di The Art of Computer
Programming è Donald Knuth.

\end{document}
```

Si notino le linee vuote che vengono inserite nel documento finale. Poiché in LaTeX la linea vuota ha il ben preciso significato di iniziare un nuovo capoverso, può essere necessario fare attenzione a questo comportamento che in ogni caso è personalizzabile agendo su specifiche opzioni di configurazione di Jinja, oppure si possono semplicemente rimuovere gli accapo alle righe 2–4 del *template* (cioè scrivendo tutto su una o due righe anziché su tre come nell'esempio).

Concludiamo questa breve introduzione vedendo come chiamare funzioni Python direttamente dal *template*. Supponiamo di avere una funzione `cubed(x)` che restituisce il cubo di un numero. Occorre innanzitutto "registrare" la funzione presso Jinja prima di compilare il template:

```
def cubed(x):
    return x**3

env.globals.update(cubed=cubed)
```

In questo modo, il seguente *template*:

```
\begin{document}
Il cubo di 4 è {{cubed(4)}}.
\end{document}
```

diventerà:

```
\begin{document}
Il cubo di 4 è 64.
\end{document}
```

Ovviamente il documento così prodotto deve poi essere compilato con `pdflatex`, ma non è difficile modificare lo *script* Python in modo che, oltre a generare il documento `.tex` (magari in una cartella temporanea), si occupi anche di compilare il documento finale in PDF, ad esempio usando il modulo `subprocess` della libreria standard di Python per eseguire `pdflatex` oppure `latexmk -pdf`. Uno *script* così fatto sarebbe perfetto come *microservizio*[3] nel *backend*

3. Nelle moderne architetture *cloud*, i microservizi sono un modo di strutturare il *backend* in una rete di piccoli servizi indipendenti, ciascuno dedicato ad assolvere un compito specifico e scritto nel linguaggio di programmazione più appropriato per quel compito.

di un sito web che deve generare documenti PDF dinamicamente in base ai dati degli utenti, come ad esempio un attestato o una fattura.

3. Personalizzazione di Jinja

Gli esempi mostrati fino ad ora sono scritti nella sintassi predefinita di Jinja. Tuttavia, è possibile configurare Jinja cambiandone la sintassi in modo che questa si amalgami meglio con quella di LaTeX. Il mescolamento delle sintassi di LaTeX e di Jinja, infatti, può non risultare gradevole e inoltre non è ben gestito dagli *editor* di testo che supportano la colorazione della sintassi. Ad esempio, nella sezione precedente abbiamo visto che un *blocco* è delimitato dalla coppia di caratteri costituita da una parentesi graffa e da un segno di percentuale. Siccome in LaTeX il segno di percentuale introduce un commento, il nostro *editor* riconoscerebbe lo *statement* come un commento ma solo a partire dal segno %, escludendo quindi il primo carattere (la parentesi graffa aperta); è per questo motivo che, anche in questo articolo, la parentesi graffa dello *statement* è mostrata in verde mentre il resto del rigo è in grigio come se fosse un commento. Per quanto riguarda le *espressioni*, la doppia parentesi graffa non è dissonante con la sintassi di LaTeX, ma si può pensare di renderla comunque più esplicita.

Fortunatamente, Jinja offre molte opzioni di configurazione e personalizzazione, sfruttando le quali è ad esempio possibile ridefinire alcuni elementi della sua sintassi in stile LaTeX:

```
1 env = Environment(
2     loader=FileSystemLoader('.'),
3     variable_start_string='\PYEXP{',
4     variable_end_string='}',
5     block_start_string='\PYBLOCK{',
6     block_end_string='}',
7     line_statement_prefix='%%',
8 )
```

Con questa configurazione, l'espressione `{{4**3}}` si scriverebbe come `\PYEXP{4**3}`, camuffandola quindi come un comando LaTeX. Lo stesso si può dire per lo *statement* che adesso si scriverebbe come:

```
\PYBLOCK{for b in books}
L'autore di \PYEXP{b.title} è
\PYEXP{b.author}.
\PYBLOCK{endfor}
```

C'è però un modo ancora più conciso e gradevole per scrivere uno *statement* in un *template* LaTeX. Jinja consente infatti di considerare una riga come uno *statement* se questa inizia con un determinato prefisso, che nella configurazione predefinita è il carattere cancelletto. Nella configurazione di cui sopra, alla riga 7 abbiamo specificato che intendiamo usare un doppio segno di percentuale come prefisso per i cosiddetti *line statement*. In questo modo non c'è ambiguità tra un normale commento LaTeX, che inizia con un singolo carattere %, e uno *statement*, che invece comincia con %% ma che viene comunque interpretato come commento dall'*editor* di testo come mostrato nell'esempio seguente:

```
\begin{document}
%% for b in books
L'autore di \PYEXP{b.title} è
\PYEXP{b.author}.
%% endfor
% normale commento LaTeX
\end{document}
```

che diventa:

```
\begin{document}
L'autore di Dubliners è James Joyce.
L'autore di The Art of Computer
Programming è Donald Knuth.
% normale commento LaTeX
\end{document}
```

Si noti che, quando si usano i *line statement* come in questo esempio, nel documento finale non vengono inserite le righe vuote prima e dopo il *blocco*, a differenza di quanto visto nella sezione precedente.

Nella pagina seguente è riportato il codice completo di un piccolo esempio che raccoglie quanto visto fino ad ora, il quale per ragioni di brevità non rende giustizia alle potenzialità offerte dal poter accedere, direttamente in un documento LaTeX, a elaborazioni comunque complesse scritte in un linguaggio agevole come Python. Il listato 1 mostra sia il *template* sorgente (in alto) sia il risultato della compilazione tramite lo *script* riportato nel listato 2 (in basso). Si noti, in aggiunta al già discusso ciclo `for`, l'uso del condizionale `if`.

Nella sezione che segue vedremo invece un esempio un po' più articolato incentrato sulla generazione di una fattura, un tipo di documento che per sua natura potrebbe richiedere un'elevata complessità algoritmica, per meglio mettere in luce quali vantaggi si possano ottenere dall'usare insieme Jinja e LaTeX in un contesto pratico.

4. Un esempio: generazione di una fattura

In questa sezione vediamo un esempio relativamente più complicato: la generazione di una fattura. Cercheremo comunque di mantenere un livello piuttosto semplice, focalizzandoci unicamente sul *template* e sull'interazione con Python, senza porre in questa sede particolare attenzione alla resa tipografica del documento.

In un contesto reale è facile immaginare che i dati necessari per compilare la fattura sarebbero estratti da un *database* tramite una *query* opportuna. Ai fini di questa dimostrazione, non ha importanza soffermarci su come lo *script* Python recuperi i dati di interesse (ad esempio, se tramite una connessione diretta al database, oppure tramite una richiesta HTTP a un server, o ancora prelevandoli da un foglio di calcolo in formato Excel o OpenDocument usando il pacchetto `pyexcel`). Fortunatamente, la ricchezza della libreria standard di Python e l'esistenza di librerie dedicate facilita non di poco questo compito. Per questo esempio,

Template sorgente:

```
\documentclass{article}

\begin{document}
Il cubo di 4 è \PYEXP{cubed(4)}.
%% if myvar == 42
Hai vinto.
%% else
Hai perso.
%% endif

%% for item in mylist
L'autore di \PYEXP{item.title} è
\PYEXP{item.author}.
%% endfor

% normale commento LaTeX
\end{document}
```

Template "compilato":

```
\documentclass{article}

\begin{document}
Il cubo di 4 è 64.
Hai vinto.
L'autore di Dubliners è James Joyce.
L'autore di The Art of Computer
Programming è Donald Knuth.
% normale commento LaTeX
\end{document}
```

Listato 1. In alto: *template* sorgente. In basso: *template* compilato con il codice del listato 2.

supponiamo che i dati siano memorizzati sul disco in un *file* in formato YAML[4] come mostrato nel listato 3. Una volta caricato il *file* usando il pacchetto `PyYAML`, i dati saranno accessibili in Python sotto forma di un *dizionario*.

Per questo esempio useremo il *template* mostrato nel listato 4 e lo *script* mostrato nel listato 5. Il documento finale compilato con LaTeX è mostrato in figura 1. Come si può vedere, il *template* rappresenta una lettera con diversi elementi che sono riempiti dinamicamente tra cui: il numero e la data della fattura (riga 6), il nome e l'indirizzo del destinatario (righe 9–10),

4. YAML, acronimo ricorsivo per "YAML Ain't Markup Language", è un formato per la serializzazione dei dati pensato per essere facilmente leggibile e modificabile dagli esseri umani. Più probabilmente, nelle comunicazioni tra differenti servizi, quindi tra macchine, i dati sarebbero serializzati in formato JSON: JavaScript Object Notation.

```
from jinja2.nativetypes import \
    (NativeEnvironment, Environment)
from jinja2.loaders import \
    FileSystemLoader

env = Environment(
    loader=FileSystemLoader('.'),
    variable_start_string='\PYEXP{',
    variable_end_string='}',
    line_statement_prefix='%%',
)

def cubed(a):
    return a**3

env.globals.update(cubed=cubed)

mylist = [
    {
    'author': 'James Joyce',
    'book': 'Dubliners',
    },
    {
    'author': 'Donald Knuth',
    'book': 'The Art of Computer '
            'Programming',
    },
]
myvar = 42

templ = env.get_template('template.tex')
templ = templ.stream(
    mylist=mylist,myvar=myvar)
templ.dump('output.tex')
```

Listato 2. Esempio di *script* Python per compilare il *template* Jinja mostrato nel listato 1.

le righe della tabella contenenti il dettaglio dei libri acquistati (righe 18–24). Si possono notare gli *statement* `for` e `if` rispettivamente alle righe 18 e 21, evidenziate in azzurro. In particolare il condizionale `if` viene usato per aggiungere alla tabella una voce riguardante l'eventuale confezione regalo. All'interno delle espressioni, demarcate da `\PYEXP{}`, figurano chiamate a funzioni Python (riga 10) e operazioni matematiche (riga 20).

Passando adesso allo *script* Python, si possono notare alcuni blocchi di codice già incontrati in precedenza, come la personalizzazione di Jinja alle righe 15–20. Inoltre:

```yaml
invoice:
  number: 42
  date: 2021-03-28
customer:
  name: Mario Rossi
  address: |
    Viale delle Idee, 1
    50100 Firenze
books:
  - title: The Art of Computer Programming
    author: Donald Knuth
    price: 65.49
    gift: false
    quantity: 2
  - title: 'Just for Fun: The Story of an Accidental Revolutionary'
    author: Linus Torvalds
    price: 14.64
    gift: true
    quantity: 1
  - title: The Cathedral and the Bazaar
    author: Eric S. Raymond
    price: 9.24
    gift: false
    quantity: 1
```

Listato 3. Un esempio di documento YAML con i dati per la generazione di una fattura.

- alla riga 7 si imposta il *locale* italiano.
- alla righe 9–13 si definiscono due funzioni ausiliarie `P()` e `D()`, rispettivamente per la formattazione dei prezzi e delle date. Come abbiamo già visto, alla riga 30 queste vengono "registrate" nell'ambiente Jinja per poter essere usate direttamente nel *template*.
- alle righe 22–23 si carica il documento YAML mostrato nel listato 3 e contenente i dati per la fatturazione.
- alle righe 25–28 si calcola il costo totale della fattura, sommando i prezzi dei singoli articoli e aggiungendo le commissioni per le confezioni regalo.
- alla righe 32–35 si definisce un *dizionario* di variabili aggiuntive che vogliamo rendere disponibili nel *template*.
- alla riga 38 si "compila" il template, passandogli tutte le variabili di cui abbiamo bisogno. Si notino i doppi asterischi con cui si fa il cosiddetto *unpacking* dei due *dizionari* `data` e `my_vars`. In questo modo le tre *chiavi* del *dizionario* corrispondente ai dati del listato 3 (`invoice`, `customer` e `books`), insieme alle due *chiavi* del *dizionario* `my_vars`, sono direttamente disponibili nel *template* come se fossero variabili singole.

```
\documentclass[DIV=13]{scrletter}
\usepackage{eurosym} \let\texteuro\euro
\date{}
\setkomavar{fromname}{The GuIT Bookshop}
\setkomavar{fromaddress}{Via Claudio 21 \\ 80125 Napoli}
\setkomavar{title}{Fattura n. \PYEXP{invoice.number} del
  \PYEXP{D(invoice.date)}}

\begin{document}
  \begin{letter}{\PYEXP{customer.name} \\
  \PYEXP{customer.address.replace('\n', ' \\\\ ')}}
    \opening{}
    \footnotesize
    \begin{tabular}{lrrr}
      \hline
      \textbf{Titolo} & \textbf{Prezzo unitario} & \textbf{Quantità}
      & \textbf{Totale} \\
      \hline
      %% for book in books
        \PYEXP{book.title} & \PYEXP{P(book.price)} &
        \PYEXP{book.quantity} & \PYEXP{P(book.price * book.quantity)}\\
        %% if book.gift
          \hfill\textit{confezione regalo} & & &
            \PYEXP{P(gift_wrap_price)} \\
        %% endif
      %% endfor
      \hline
      \textbf{Totale} & & & \textbf{\PYEXP{P(total)}} \\
      \hline
    \end{tabular}
  \end{letter}
\end{document}
```

Listato 4. Un semplice *template* per una fattura.

5. Conclusioni

In questo articolo abbiamo visto come usare Jinja per la generazione dinamica di documenti LaTeX. Jinja è un motore ricco di funzionalità per le quali si rimanda alla sua documentazione ufficiale che è molto ben scritta. Per quanto riguarda la sintassi da usare all'interno dei *template* si può fare riferimento alla *Template Designer Documentation*[5]; per quanto riguarda l'interfaccia di programmazione Python si può fare riferimento alla *API Documentation*[6].

5. https://jinja.palletsprojects.com/en/2.11.x/templates/
6. https://jinja.palletsprojects.com/en/2.11.x/api/

```python
import locale
import yaml

from jinja2.nativetypes import (NativeEnvironment, Environment)
from jinja2.loaders import FileSystemLoader

locale.setlocale(locale.LC_ALL, 'it_IT.utf8')

def P(x):
    return locale.currency(x)

def D(x):
    return x.strftime('%d %B %Y')

env = Environment(
    loader=FileSystemLoader('.'),
    variable_start_string='\PYEXP{',
    variable_end_string='}',
    line_statement_prefix='%%',
)

with open('db.yaml', 'r') as f:
    data = yaml.safe_load(f)

gift_wrap_price = 2

total = sum([book['price'] * book['quantity'] for book in data['books']])
total += sum([book['gift'] for book in data['books']]) * gift_wrap_price

env.globals.update(P=P, D=D)

my_vars = {
    'total': total,
    'gift_wrap_price': gift_wrap_price,
}

template = env.get_template('invoice.tex')
template.stream(**data, **my_vars).dump('output.tex')
```

Listato 5. *Script* per compilare il *template* mostrato nel listato 4.

The GuIT Bookshop
Via Claudio 21
80125 Napoli

The GuIT Bookshop, Via Claudio 21 , 80125 Napoli

Mario Rossi
Viale delle Idee, 1
50100 Firenze

Fattura n. 42 del 28 marzo 2021

Titolo	**Prezzo unitario**	**Quantità**	**Totale**
The Art of Computer Programming	€ 65,49	2	€ 130,98
Just for Fun: The Story of an Accidental Revolutionary	€ 14,64	1	€ 14,64
confezione regalo			€ 2,00
The Cathedral and the Bazaar	€ 9,24	1	€ 9,24
Totale			**€ 156,86**

Figura 1. Esempio di fattura generata con Jinja.

Riferimenti bibliografici

Giacomelli, Roberto e Gianluca Pignalberi (2015). «Generare documenti LaTeX con diversi linguaggi di programmazione». *ArsTEXnica*, p. 40.

Giacomo Mazzamuto
Consiglio Nazionale delle Ricerche – Istituto Nazionale di Ottica
(CNR-INO)
g.mazzamuto+ctan@gmail.com

Vettorizzazione di immagini raster (parte I) per recuperare situazioni tipograficamente critiche

Gianluca Pignalberi

Sommario Esistono determinati casi in cui trasformare un'immagine raster nell'equivalente vettoriale permette un risultato tipografico migliore. La versione di Potrace inclusa in Inkscape aiuta in quei determinati casi. Vediamo nel primo di due articoli i casi più comuni e come agire di volta in volta.

Abstract Specific cases exist when converting a raster image to the corresponding vector one allows a tipographically better result. The Potrace version included in Inkscape helps in those cases. In this first paper out of two, we are going to list the most common cases and show how to process them.

1. Introduzione

I documenti dotati di apparato iconografico hanno spesso immagini fotografiche. Queste immagini hanno la naturale rappresentazione nel formato *raster* (a matrice di pixel[1]): a ogni pixel della matrice corrisponde un *quanto* della scena catturata. Un quanto è una porzione di scena, tanto più piccola quanti più punti contiene la matrice. Per visualizzare un quanto, immaginiamo di sovrapporre una griglia alla scena da fotografare. Tanto più la griglia è fitta, tanti più sono i quanti, cioè le porzioni nelle quali la scena è stata suddivisa; tanti piú sono i quanti, quanto maggiore sarà la definizione, l'accuratezza dell'immagine. I punti delle comuni immagini raster[2] immagazzinano le componenti luminosa e cromatica rilevate in ognuno dei quanti in cui è stata suddivisa la scena. Nel caso in cui un'immagine di questo tipo presenti problemi in fase di stampa, si potrà provare a postprocessarla per ridurre l'incidenza dei problemi. Le procedure comuni in elaborazione delle immagini prevedono di operare su immagini raster e di restituire il risultato sotto forma di immagine raster. Un'ottima discussione della teoria e della pratica di alcuni di questi metodi si trova in Zamperoni (1990).

L'inclusione di immagini raster nei documenti è spesso l'unica soluzione conosciuta da alcuni autori, ma non è sempre quella migliore. Grafici, schemi e disegni al tratto trovano nel formato vettoriale il veicolo perfetto. Il formato vettoriale descrive gli elementi rappresentati nell'immagine per mezzo di formule. Il suo punto di forza, l'indipendenza dal ridimensionamento, è anche il suo punto debole: più oggetti ci sono da rappresentare, maggiore sarà

1. Le immagini raster sono anche note col nome di *bitmap*, mappa di bit.
2. Il dato rappresentato nelle immagini raster dipende da cosa vogliamo rappresentare. Quelle cosiddette "comuni" sono immagini di intensità, ma ecco che esistono immagini di profondità, con punti tanto più chiari — o più scuri — quanto più vicini sono all'osservatore; immagini di calore, in cui il colore dei punti dipende dalla loro temperatura; immagini altimetriche, in cui il colore rappresenta un'altezza o una profondità. L'elenco potrebbe continuare.

l'occupazione di spazio in memoria, di massa o meno. Un bosco, uno scorcio o una foto aerea di una città potrebbero essere ingestibili, se rappresentati in formato vettoriale, a causa dell'enorme numero di oggetti presenti; nelle immagini raster, invece, la quantità d'informazione dipende da quanti punti è composta l'immagine e da quante sfumature di colore sono possibili. Nei casi in cui l'immagine non abbia troppe sfumature, come nei disegni al tratto, o abbia solo elementi geometrici o quasi, come nei grafici e negli schemi, il formato vettoriale si rivela parco di occupazione di spazio in memoria e presenterà una figura tecnicamente esente da problemi in stampa: non sarà sgranata, né impastata, né sfocata.

Il presente articolo mostra come in alcune occasioni la trasformazione di immagini raster "difettose" in immagini vettoriali ha recuperato (o avrebbe potuto recuperare) brillantemente delle situazioni critiche, segnatamente la stampa di immagini confuse a causa delle piccole dimensioni e/o bassa risoluzione. Dopo una breve introduzione alla vettorizzazione[3] di immagini raster e a Potrace (nella versione inclusa in Inkscape), i paragrafi 4 e 5 mostreranno diversi casi d'uso che fungeranno da conciso manuale operativo, costituendo così il nocciolo dell'articolo.

2. Inseguimento dei contorni, ovvero della vettorizzazione di immagini raster

La pagina Wikipedia (2021) elenca un buon numero di programmi in grado di vettorizzare immagini raster mettendone a confronto le caratteristiche, i pro e i contro. Tra i programmi in elenco figura Potrace, un programma a riga di comando che prende in input un'immagine raster in bianco e nero (in uno tra i pochi formati riconosciuti) e ne restituisce l'equivalente vettoriale che può essere salvata in `.eps` (Encapsulated PostScript). Su Selinger (2019) si legge il perché di queste scelte e si scopre che gli sviluppatori di Inkscape, includendo Potrace nel loro programma, ne hanno esteso la funzionalità alle immagini a colori.

Lo stesso sito mostra alcuni esempî completi, e generosi di spiegazioni, ottenuti o meno in abbinamento al programma supplementare mkbitmap.

Ma la parte utile a capire come funziona Potrace è data dal documento Selinger (2003). Rimandando il lettore interessato al suo studio integrale, diamo qui la traduzione dell'inizio della sezione 2:

> L'algoritmo di Potrace, trasforma una bitmap in un contorno vettoriale in diversi passi. Nel primo passo, la bitmap viene decomposta in un insieme di percorsi che formano i contorni tra le aree nere e quelle bianche. Nel secondo passo, ogni percorso è approssimato mediante un poligono ottimale. Nel terzo passo, ogni poligono viene trasformato in un contorno senza spigoli. In un quarto passo, opzionale, la curva risultante viene ottimizzata unendo tra loro segmenti consecutivi di curve di Bézier, dove ciò è possibile. Infine, viene generato l'output nel formato voluto.

Insomma, Potrace prende un'immagine raster, ne insegue i contorni mediante poligoni, quindi fa un aggiustamento dei vertici, un'analisi degli angoli e uno *smoothing* (opzionale, un'ottimizzazione delle curve) e restituisce l'output vettoriale. L'articolo mostra anche una

3. Il verbo *vettorizzare* è l'italianizzazione del verbo inglese *to vectorize*, voce presente sul Collins. Questa la traduzione della definizione ivi riportata: (di grafica computerizzata) convertire da una rappresentazione a matrice di punti a una rappresentazione vettoriale.

comparazione di tempi di elaborazione e risultati tra Potrace e AutoTrace, entrambi molto in favore del primo. È pur vero che un caso non fa statistica, ma sembra un bel biglietto da visita.

3. Breve manuale d'uso di Potrace

Sapendo di dover vettorizzare delle immagini a colori o a toni di grigio, dovrò necessariamente usare Potrace nella versione inclusa in Inkscape, quindi vediamo direttamente come usare questa versione. Sebbene l'articolo mostri i reali risultati ottenuti con una vecchia versione di Inkscape (la 0.92), il lettore tenga presente che la versione stabile corrente di Inkscape è la 1.0. Questa mostra una diversa organizzazione delle finestre di dialogo di Potrace e vi introduce la vettorizzazione *pixel art* (Vinícius dos Santos Oliveira *et al.*, 2019), basata su `libdepixelize` e tenuta in precedenza come voce separata. Un futuro articolo descriverà esclusivamente la vettorizzazione *pixel art*.

La prima cosa da fare è importare in un documento l'immagine che dobbiamo vettorizzare. Inkscape non ha le restrizioni sui formati di Potrace a riga di comando; in ogni caso, possiamo sempre ricorrere a un programma di fotoritocco per la conversione di formato da un raster "esoterico" a un raster più comune. Appena l'immagine è visibile nel documento, la selezioniamo e selezioniamo la voce del menù Path → Trace bitmap. Si aprirà una finestra simile a quella della figura 1. Appena avremo impostato tutti i valori utili all'elaborazione, premeremo OK per avere poco dopo il risultato sotto forma di un'immagine vettoriale *sovrapposta* all'originale raster. Siamo pronti a salvare il frutto dell'elaborazione in uno dei formati vettoriali conosciuti da Inkscape. Attenzione in fase di salvataggio: entrambe le immagini rimangono selezionate e finiranno nel file salvato, a meno di non escludere l'originale raster.

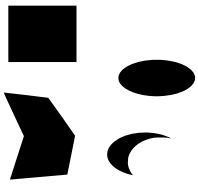
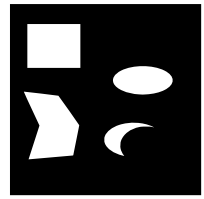
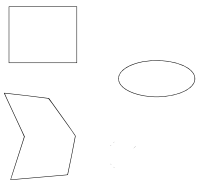
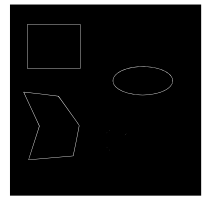
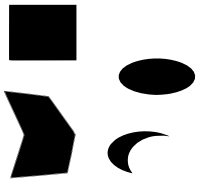
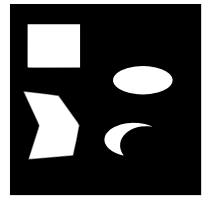
La finestra per il tracciamento delle bitmap ha tre linguette: Modalità (*Mode*), Opzioni (*Options*) e Meriti (*Credits*).[4] L'ultima contiene un semplice messaggio che indica in Potrace il vettorizzatore incluso in Inkscape; la seconda permette di scegliere di eseguire o meno, e in base a quali valori, alcuni passi preliminari alla vettorizzazione; la prima, infine, permette di stabilire il principio di vettorizzazione.

Le opzioni comprendono il *despeckle* (*Suppress speckles*), cioè l'eliminazione di blocchi di pixel isolati che normalmente costituiscono rumore impulsivo (valori ammessi per la dimensione dei blocchi: da 0 a 1000), lo *smoothing* (*Smooth corners*) degli angoli acuminati, cioè un leggero arrotondamento (valori ammessi per la soglia, cioè per un arrotondamento più o meno marcato: da 0.00 a 1.34), e l'ottimizzazione del percorso (*Optimize paths*) tramite giunzione di segmenti di curve di Bézier adiacenti (valori ammessi per la tolleranza, cioè la minore o maggiore ottimizzazione che si ripercuote sul numero di nodi nel percorso: da 0.00 a 5.00). In tutti i casi trattati per il manuale, non c'è mai stato bisogno di cambiare i valori di default per avere un ottimo risultato in stampa.

Torniamo al principio di vettorizzazione. La prima decisione per tracciare un contorno, cioè vettorizzare un'immagine raster, riguarda la classe delle scansioni: singole o multiple. Una scansione singola (*Single scan*) genera un unico percorso, cioè un'immagine in bianco e nero; una scansione multipla (*Multiple scans*) ne genera un gruppo, cioè un'immagine a toni di grigio o a colori (fino a 256). Tutto ciò indipendentemente dal numero di oggetti sconnessi all'interno dell'immagine.

4. Meriti è l'inusitata, ma corretta, traduzione del termine inglese *Credits* usato in questo contesto.

Tabella 1. Tabella riassuntiva dei risultati della vettorizzazione a singola scansione con Inkscape. Le immagini della terza colonna sembrano più piccole perché lo sfondo è ineliminabile. In quelle della seconda colonna lo sfondo è stato eliminato, allo stesso modo di un *crop*.

Metodo	Risultato	Negativo	Note
Brightness cutoff (taglio di luminosità)			La soglia minima per vettorizzare tutti e quattro gli oggetti è 0.830
Edge detection (estrazione dei contorni)			Nessuna soglia è in grado di estrarre il contorno dell'oggetto giallo. La soglia 0 copre l'intero rettangolo
Color quantization (quantizzazione del colore)			Il numero minimo di colori per vettorizzare i quattro oggetti è 11

La classe di scansioni costituisce un insieme di operazioni alternative, che dunque possiamo selezionare una per volta. Per generare immagini in bianco e nero possiamo operare un taglio della luminosità (*Brightness cutoff*, con una soglia compresa tra 0 e 1), oppure un'estrazione dei contorni (*Edge detection*) o, infine, una quantizzazione del colore (*Color quantization*). In aggiunta a ognuna delle tre operazioni, possiamo invertire l'immagine (*Invert image*). La tabella 1 mostra i risultati della singola scansione applicata all'immagine sintetica della figura 2.

Per generare immagini a colori abbiamo a disposizione le operazioni alternative "livelli di luminosità" (*Brightness steps*), "colori" (*Colors*) e "grigi" (*Grays*). Per tutte queste operazioni esiste un unico parametro (*scans*) che indica il numero n di scansioni a cui sottoporre l'immagine di input. La prima operazione genererà n oggetti pari al numero n dei livelli di luminosità. La seconda e la terza genereranno rispettivamente un'immagine a colori e una a toni di grigio e n sarà esattamente il numero di colori (o grigi) da generare. Nel caso in esame, chiederemo che $n = 5$ perché ci sono 4 oggetti colorati e lo sfondo di un quinto colore (bianco, in questo caso). La tabella 2 riporta i risultati ottenuti. Non è prevista la generazione di un'immagine inversa ma possiamo combinare tre diverse operazioni opzionali: sfocatura gaussiana preliminare (*Smooth*), impilamento dele scansioni (*Stack scans*, per evitare spazî vuoti tra una regione di colore e l'altra) e rimozione dello strato di sfondo (*Remove background*) alla fine dell'operazione.

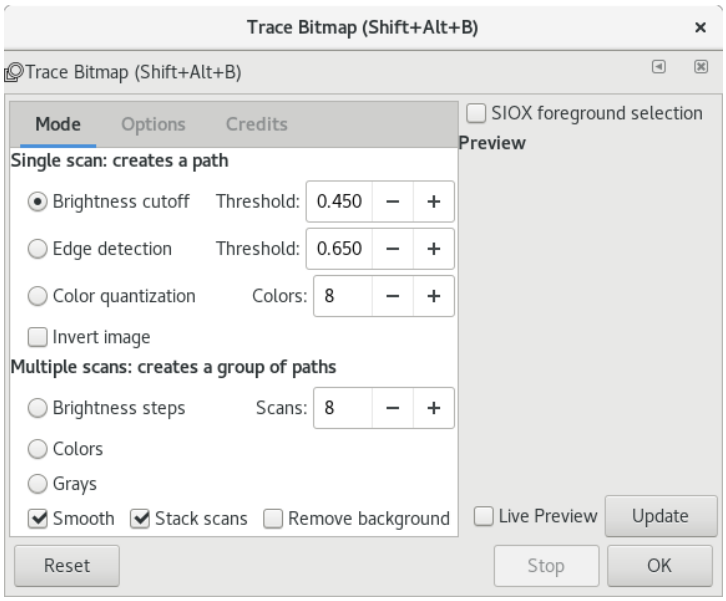


Figura 1. Finestra di Inkscape per il controllo di Potrace.

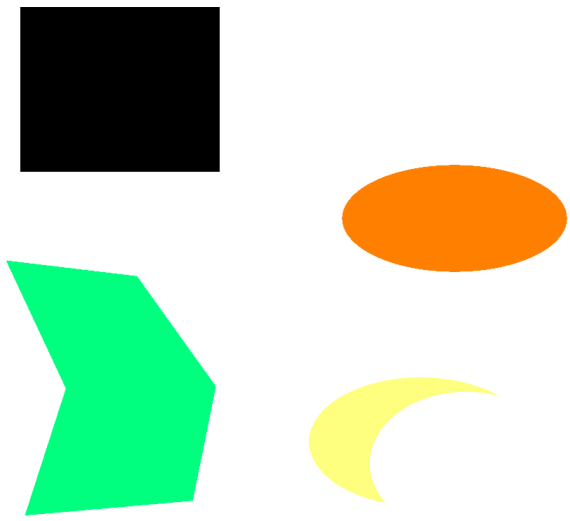
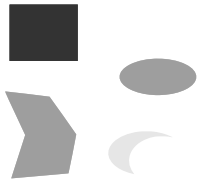
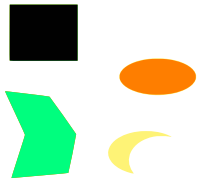
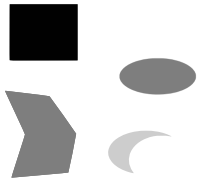


Figura 2. Immagine sintetica costituita da forme geometriche colorate con colori più o meno luminosi.

Tabella 2. Tabella riassuntiva dei risultati della vettorizzazione a scansione multipla con Inkscape.

Metodo	Risultato	Note
Brightness steps (livelli di luminosità)		La soglia minima per vettorizzare tutti e quattro gli oggetti è 11 colori (scansioni)
Colors (colori)		La soglia minima per vettorizzare tutti e quattro gli oggetti è 5 colori (scansioni)
Grays (grigi)		La soglia minima per vettorizzare tutti e quattro gli oggetti è 5 colori (scansioni)

Ora che abbiamo una cognizione minima del funzionamento di Potrace in Inkscape, siamo pronti a procedere.

4. Una situazione ipotetica

Per portare a termine la composizione della copertina del libro *L'arnale* di Iano Lanz,[5] ho ricevuto dall'autore un'immagine *clip art* comprata da iStock (la figura 3 mostra quella reperibile in formato vettoriale). Di tale immagine si possono comprare (pagandone uno per volta) file raster di diverse dimensioni oppure un file vettoriale `.eps`. Non essendo l'autore esperto di immagini digitali, ha comprato l'immagine più piccola in commercio, quando a parità di prezzo è sempre preferibile comprarne una enorme da rimpicciolire, piuttosto che una piccola da ingrandire; meglio ancora il formato vettoriale, quando previsto. Quest'immagine, delle dimensioni di 170×170 pixel, era buona al massimo per un'anteprima su un sito. Le soluzioni possibili e alternative erano due: la prima — ricomprare dal sito per una decina d'euro l'immagine vettoriale; la seconda — elaborare l'immagine piccola per ottenerne una versione ingrandita e utilizzabile senza problemi. In realtà c'è una terza "soluzione": prendere l'anteprima dell'immagine stessa, di 1024×1024 pixel e piena di filigrane, e tentarne una ripulitura senza passare per la vettorizzazione.

5. Il nome è chiaramente uno pseudonimo.

Figura 3. Immagine vettoriale per la copertina del romanzo *L'Arnale* di Iano Lanz.

Parafrasando il principio di economia o di parsimonia, altrimenti noto come *rasoio di Occam*, possiamo affermare che la soluzione migliore è quella che, a parità di qualità dell'immagine, costa meno oppure che, a parità di costo, offre l'immagine più pulita.

Ovviamente, in questo specifico caso la soluzione migliore è stata ricomprare l'immagine: ogni elaborazione di una delle due raster citate costerebbe enormemente di più. Ma, per pura accademia, vedremo come applicare il metodo di recupero della qualità: potrebbe essere utile in casi reali come quelli descritti nella prossima sezione.

4.1. Quanti artefatti da cancellare

L'immagine JPEG di 170×170 pixel fornitami era piena di artefatti dovuti alla compressione *lossy* del formato, tanto più evidenti quanto più si rimpicciolisce l'immagine.

Prima di passare l'immagine al vettorizzatore, occorre tentare di migliorarne la qualità. Nel caso in esame useremo GIMP. Dopo una serie di tentativi compresa tra i 20 e i 30 — e qui mi è mancato un algoritmo genetico simile a quello sviluppato per la mia tesi di laurea e per il modesto pugno di articoli da essa prodotti, l'ultimo dei quali è PIGNALBERI *et al.* (2003): era in grado di generare diverse combinazioni di parametri da dare in input a un segmentatore e di stabilire tramite euristica la qualità del risultato, cioè dell'immagine segmentata —, la sequenza di passi migliore è stata quella mostrata dalla figura 4. Il risultato non è proprio esaltante. Le didascalie riportano tutte le operazioni fatte.

4.2. Osservazioni

Come già svelato, la soluzione migliore è stata acquistare una nuova copia dell'immagine, ovviamente in versione vettoriale. Le dimensioni esigue dell'immagine fornita dall'autore, e i suoi artefatti di compressione, hanno reso difficile, imperfetta e bisognosa di molto *preprocessing* una procedura di lavoro che normalmente funziona senza alcun problema.

5. Una situazione reale

Al momento della stesura dell'articolo, l'editrice CLUT sta ultimando il *Manuale di Scienza e Tecnologia del Legno* (BONAMINI e UZIELLI, 2021). Gli autori hanno realizzato tutte le immagini del manuale: molte sono disegni Autocad, ma una buona parte sono dei disegni al tratto in toni di grigio e dotati di scritte. Per un disguido tecnico, la cartella con gli originali dei disegni è stata persa. Questo è un brutto guaio:

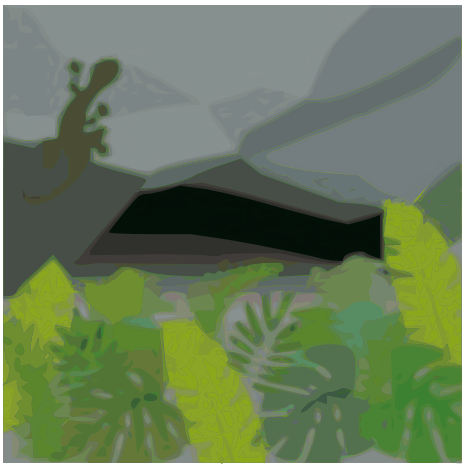
- le immagini fornite inizialmente (`.tif` e `.png`) erano troppo piccole per andare negli scavi predisposti (compresi tra i 9 e i 12 cm). Ingrandirle rendeva evidente la *pixelatura*;
- le immagini fornite in un secondo tempo, ingrandite dagli autori del manuale, sembravano aver risolto il problema: viste a video, sembravano quasi perfette. La stampa ne ha però rivelato tutti i difetti;
- molte delle immagini contengono elementi "didascalici": segmenti, frecce e scritte. Il formato nativo del programma usato per realizzare le immagini avrebbe permesso di mantenere elementi diversi su diversi livelli. Ciò avrebbe reso immediato eliminare gli elementi didascalici aggiunti in un secondo tempo al disegno e che risultavano altrettanto sgranati quanto il disegno;
- gli elementi didascalici, impressi nel disegno, sono stati ridimensionati con le immagini. Ciò ha fatto perdere uniformità all'insieme, specie in quelle immagini che, ridimensionate, presentano scritte esageratamente grandi.

Per risolvere in un colpo solo tutte queste controindicazioni all'uso delle immagini ricevute, gli autori avrebbero dovuto ridisegnare tutto e fornire ogni immagine nelle dimensioni minime di 1500×1500 pixel. Per evitare questo lavoro, senz'altro non breve, la vettorizzazione è una panacea. Vediamo dunque un caso pratico di elaborazione completa di una delle immagini incluse nel citato manuale. Ovviamente la procedura proposta dovrà essere considerata come indicativa e non certo come un algoritmo valido per qualunque immagine.

(a) Immagine originale con artefatti.

(b) Sfocatura ottenuta con l'operatore *Gaussian blur*, raggio di sfocatura 2×2 pixel e metodo IIR (*Infinite Impulse Response*).

(c) Nitidezza ripristinata mediante l'operazione di posterizzazione con 31 colori anziché con i filtri specifici.



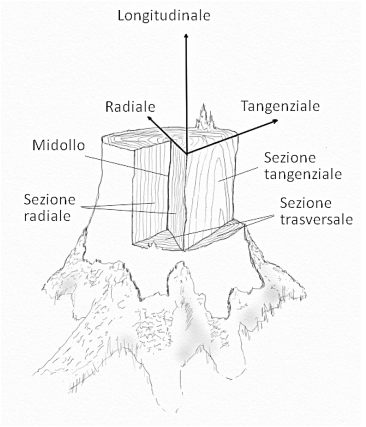
(d) Tracciamento in modalità colore (36 scansioni, *smooth* deselezionato, soglia di arrotondamento degli angoli 0.50 e di ottimizzazione dei percorsi 5.00).

Figura 4. Elaborazione passo-passo dell'immagine minuscola.

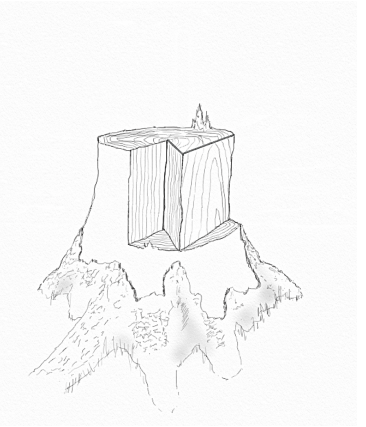
5.1. Passo 1: analisi della figura originale

La figura 5(a) mostra una delle figure originali fornite dagli autori. In questo caso "originale" significa "fornita coi file del libro" e non "nel formato di disegno originario".

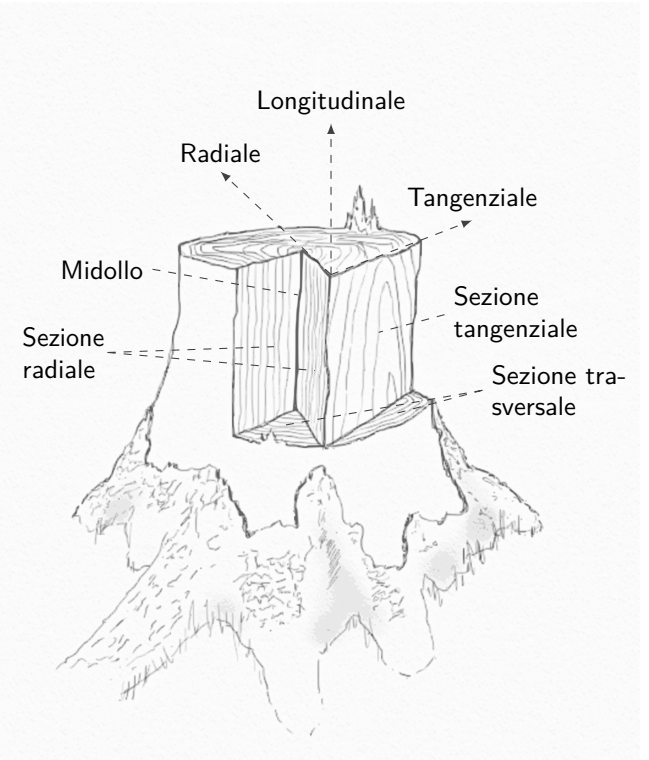
Quali sono i motivi che vietano moralmente il suo uso in un'opera a stampa? Li elencheremo in ordine casuale: i segmenti sono molto scalettati; gli assi sono molto, forse troppo neri rispetto al grigio del disegno, gli altri segmenti sono troppo simili per colore e spessore al



(a) Una delle figure realizzate per il Manuale del legno.



(b) L'immagine di partenza dopo la pulitura.



(c) L'immagine vettorizzata è stata dotata delle scritte ed è pronta a sostituire l'originale nel libro. Qui è presentata nelle esatte dimensioni del manuale.

Figura 5. Una delle figure del Manuale di Scienza e Tecnologia del Legno, prima e dopo la "trasformazione".

tratto usato nel disegno; le scritte, il cui font sarebbe quello usato nel libro, appaiono troppo sgranate e grigie, al pari del disegno complessivo; il disegno vero e proprio, molto pregevole, non è per niente resistente agli ingrandimenti proprio per quanto detto in precedenza.

Prima di ottenere un'immagine adatta alla stampa dobbiamo fare un po' di pulizia, cioè eliminare le scritte e le righe che le collegano alle parti del disegno.

5.2. Passo 2: ripulitura dell'immagine

La prima cosa che toglieremo sono le scritte. Queste, tutte poste sullo sfondo, si eliminerebbero facilmente con lo strumento "gomma per cancellare" di GIMP. Questa, di fatto, è un pennello che dipinge col colore dello sfondo (background) anziché con quello del tratto (foreground). Purtroppo, però, lo sfondo non è uniforme e dunque la strategia da adottare sarà diversa: si ricorre al "timbro clone", uno speciale pennello che dipinge con quello che trova lungo un percorso selezionato. Se, dunque, selezioniamo una porzione dello sfondo scabro del disegno, dipingeremo sulle scritte con tale sfondo.

Puliti i tratti di segmento che insistono sullo sfondo con lo stesso metodo, vanno poi eliminati quelli sovrapposti al disegno. In pratica, bisogna ricostruire le parti di disegno "cancellate" nella sovrapposizione di elementi. La ricostruzione può essere fatta con varî strumenti. Ognuno sceglierà quelli a lui più confacenti. Io mi sono limitato al timbro clone, e al correttore, o levigatore, cioè lo strumento che ammorbidisce le asperità del colore dei pixel adiacenti in accordo a un'area precedentemente selezionata. La figura 5(b) mostra il disegno dopo la ripulitura.

5.3. Passo 3: vettorizzazione

A questo punto siamo pronti a vettorizzare l'immagine; quindi, a rimetterle le scritte cancellate. Per la prima operazione useremo le scansioni multiple di Potrace incluso in Inkscape. Nel caso in esame, ci serve connettere delle regioni in base ai toni di grigio, da lì la selezione della relativa operazione nelle scansioni multiple. Limitare a otto il numero di toni di grigio finali serve a non allungare inutilmente l'elaborazione e ad avere un'immagine ancora intelleggibile: nel poco spazio a disposizione per l'immagine stampata su pagina troppi toni sono indistinguibili.

Il risultato finale, dopo aver sovraimposto i segmenti e le scritte con LaTeX, è visibile nella figura 5(c).

Per vedere la differenza tra l'immagine di partenza e quella di arrivo, vediamo la stessa porzione presa dalla figura prima della vettorizzazione e dopo di essa (figura 6). Ora abbiamo la certezza che l'immagine stampata non soffrirà di tutti i difetti evidenziati.

6. Ulteriori suggerimenti

Non c'è bisogno di ricordare che ogni immagine necessità di una strategia *ad hoc*. Diamo qui una lista di suggerimenti validi a grandi linee per la maggior parte dei casi:

- le immagini senza elementi extra (scritte, segmenti o frecce aggiunte in un secondo momento) non hanno bisogno di ripulitura: si può passare direttamente alla vettorizzazione;

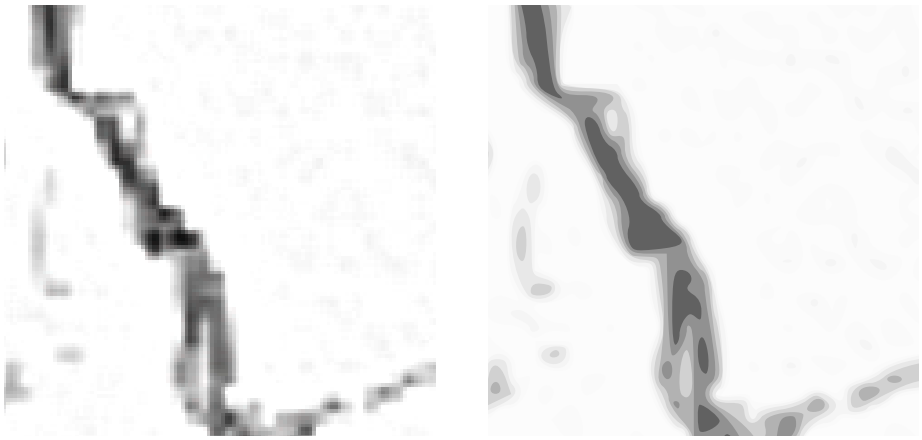


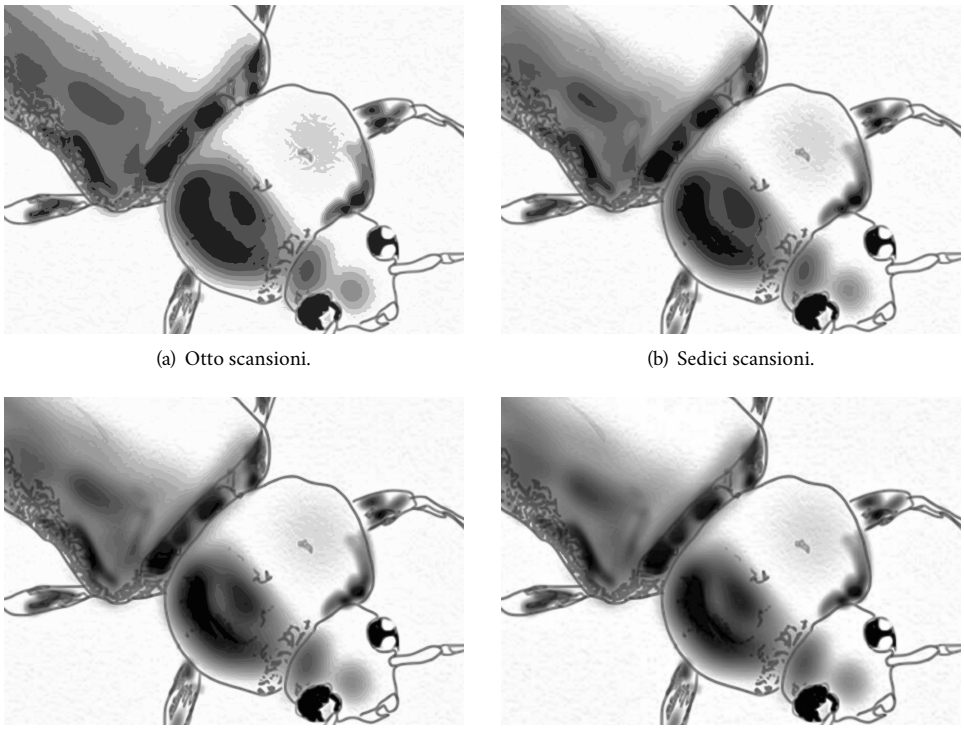
Figura 6. Ingrandendo molto è apprezzabile la differente qualità dell'originale immagine raster e della vettoriale finale.

- le immagini ricche di sfumature andranno vettorizzate con un numero di scansioni che non mostri stacchi evidenti tra i toni (vedi la figura 7);
- le immagini con regioni di colore uniforme possono "soffrire" degli effetti dell'antialias nelle zone di cambio di colore. Può essere utile fare in modo che il colore uniforme riempia per bene tali regioni;
- per preservare la disposizione dei testi voluta dagli autori, è meglio iniziare dall'ultimo passo: si include l'immagine raster nel documento e si posizionano scritte, segmenti e frecce sovrapponendole quanto più possibile alle originali, quindi si passa alla ripulitura e alla vettorizzazione;
- in alcuni casi, non sarà consigliabile eliminare l'immagine raster originale dal salvataggio. La prossima sezione darà un esempio non tanto dell'utilità, quanto della necessità di mantenere anche l'immagine raster.

7. Quando salvare raster e vettoriale insieme

Durante la lavorazione del citato manuale, di alcune immagini non ho potuto salvare la sola versione vettoriale. Cerchiamo di comprenderne i motivi.

Il manuale citato mostra un planisfero (raster) che, una volta ripulito e uniformato nei riempimenti, appare come nella figura 8. Per la vettorizzazione, la prima soluzione che ci viene in mente è la solita scansione multipla dei toni di grigio. Ma, come vediamo nella figura 9(a), i bordi di quel tono sono inaccettabili. Una soluzione subordinata è provare sempre le scansioni multiple ma con la soglia di luminosità. La figura 9(b) mostra che sono troppi gli artefatti introdotti sui contorni per rendere accettabile l'uso di tale risultato. Dunque bisognerà ricorrere a uno dei metodi a scansione singola. Il primo a cui si pensa è, senz'altro,



(a) Otto scansioni.

(b) Sedici scansioni.

(c) Trentadue scansioni.

(d) Sessantaquattro scansioni.

Figura 7. Resa delle sfumature con diverso numero di scansioni.

l'estrazione dei contorni, se non ci si ricorda che questo operatore, praticamente, ispessisce i contorni da rilevare nel caso in esame. Il risultato, mostrato nella figura 9(c), esplicita quanto supposto in teoria. Infine, proviamo col taglio della luminosità. Se teniamo la soglia entro un certo valore, l'unico elemento a essere vettorizzato è il bordo. Se cancellassimo la sottostante immagine raster, otterremmo un'immagine dei soli contorni neri, senza più grigi. Ecco perché salveremo l'immagine risultante lasciando dietro di essa l'originale raster ripulita.

8. Conclusioni

Può capitare che un'immagine raster fornita con un manoscritto non sia utilizzabile in tipografia perché dà problemi in stampa. Una delle strategie utili per evitare di dover rifare l'immagine da zero (perché è impossibile, o perché è un processo lungo o perché è molto costoso) è di convertire tale immagine da raster (matrice di pixel) a vettoriale (insieme di descrizioni geometriche del contenuto della foto). Nell'articolo ne abbiamo visti alcuni esempî. Di questi sono fornite sia la descrizione delle operazioni fatte, sia le motivazioni dell'adozione di tali operazioni, sia i risultati permessi.

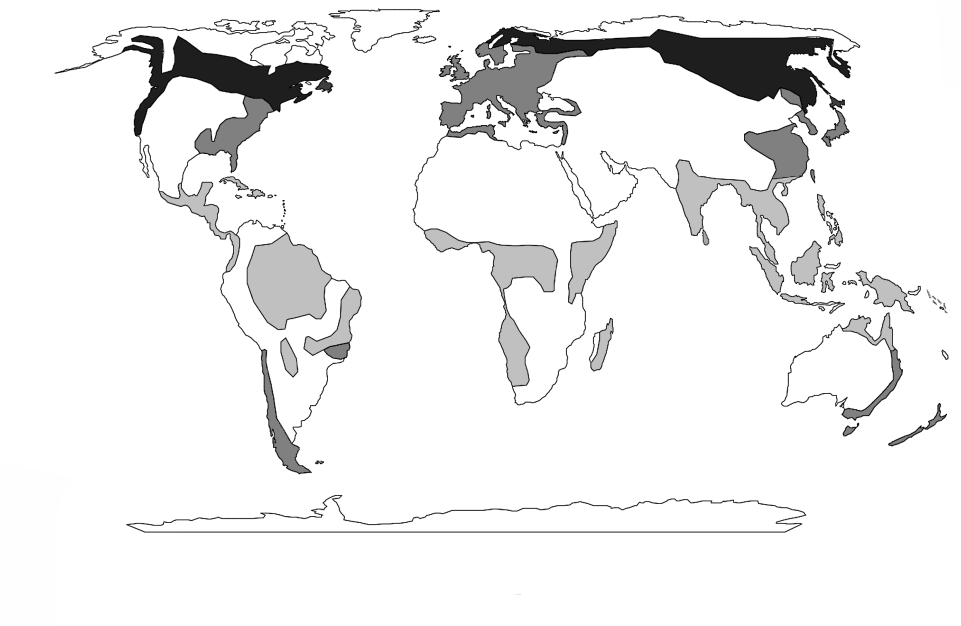


Figura 8. Un planisfero pronto per la vettorizzazione.

Ringraziamenti

L'autore desidera ringraziare l'architetto Michele Ruffino della CLUT e i professori Gabriele Bonamini e Luca Uzielli per aver concesso amichevolmente la pubblicazione delle immagini tratte dal manuale citato nell'articolo. Bonamini ha anche evidenziato delle mancanze fondamentali e ha ispirato quello che sarà la seconda parte dell'articolo.

Un ringraziamento di prammatica agli anonimi revisori che sono stati in grado di scovare nei minimi dettagli tutto quanto poteva migliorare la qualità dell'articolo.

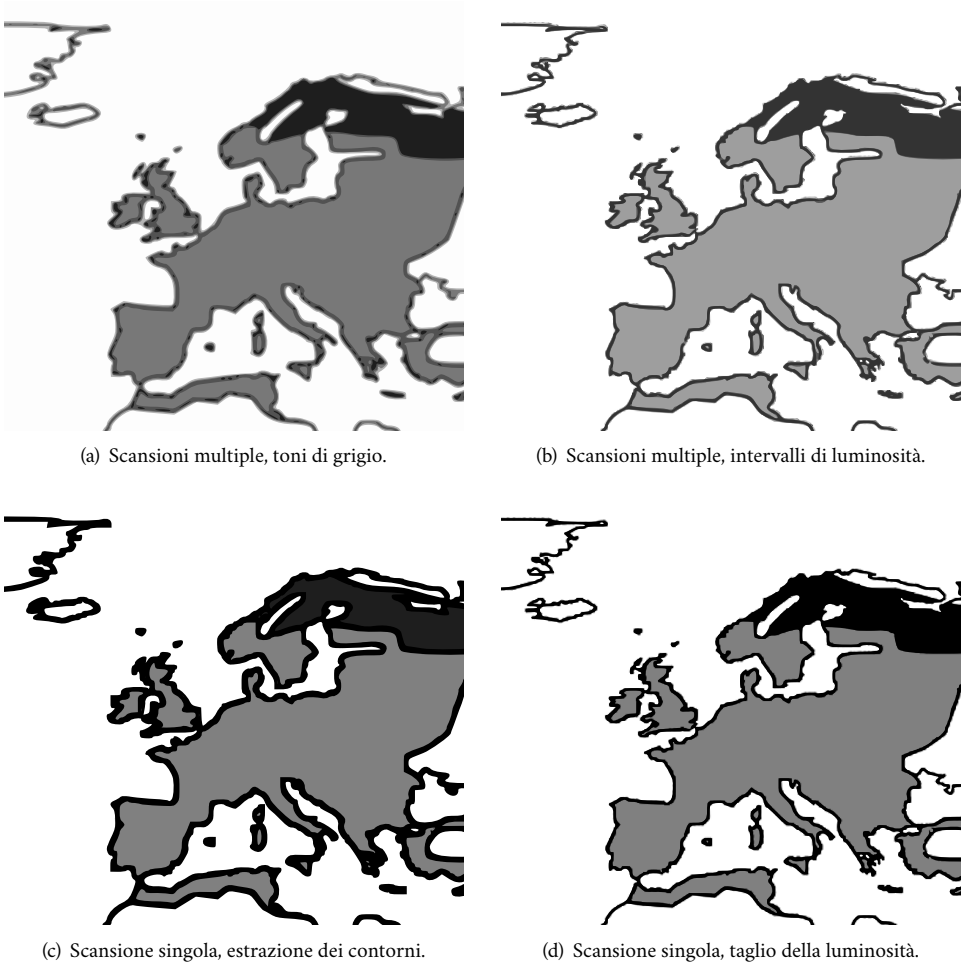
Riferimenti bibliografici

Bonamini, Gabriele e Luca Uzielli (2021). *Manuale di Scienza e Tecnologia del Legno*. CLUT, Torino. In corso di pubblicazione.

Pignalberi, Gianluca, Rita Cucchiara, Luigi Cinque e Stefano Levialdi (2003). «Tuning range image segmentation by genetic algorithm». *EURASIP Journal on Applied Signal Processing*, pp. 780–790.

Selinger, Peter (2003). «Potrace: a polygon-based tracing algorithm».

— (2019). «Potrace. transforming bitmaps into vector graphics». potrace.sourceforge.net.



(a) Scansioni multiple, toni di grigio.

(b) Scansioni multiple, intervalli di luminosità.

(c) Scansione singola, estrazione dei contorni.

(d) Scansione singola, taglio della luminosità.

Figura 9. Prove di vettorizzazione di una mappa.

Vinícius dos Santos Oliveira, Nicolas Dufour, Kris De Gussem e Gellért Gyuris (2019). «Inkscape tutorial: Tracing Pixel Art». https://inkscape.org/it/doc/tutorials/tracing-pixelart/tutorial-tracing-pixelart.html.

Wikipedia (2021). «Comparison of raster-to-vector conversion software». https://en.wikipedia.ord/wiki/Comparison_of_raster-to-vector_software.

Zamperoni, Piero (1990). *Metodi dell'elaborazione digitale delle immagini*. Masson, Milano.

Gianluca Pignalberi
g.pignalberi@gmail.com

Ricordo di Gustavo Mezzetti

Enrico Gregorio

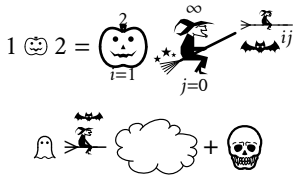
Il mese di gennaio 2021 è terminato con una tristissima notizia: Gustavo Mezzetti non è più tra noi.

Conoscevo Gustavo da più di un quarto di secolo perché si laureò con relatore Adalberto Orsatti, il 'maestro', che era stato anche mio relatore e 'advisor' per la tesi di dottorato. Gustavo si appassionò degli stessi argomenti e prese la mia tesi come punto di partenza per la sua.

A margine dell'attività scientifica, lo introdussi al mondo di TEX: lo incuriosiva molto e lo studiò a fondo. Durante il suo dottorato scrisse una classe di documenti per LATEX, ancora oggi disponibile su CTAN, cioè `letteracdp`.

Sul forum del GuIT Gustavo era appunto noto come `letteracdp` mentre su TEX.StackExchange il suo nome era `GuM`. La sua perizia e generosità nell'aiutare con risposte spesso conclusive sono note a tutti e i suoi interventi restano come eredità preziosa.

E come non menzionare `halloweenmath`?



Grazie, Gustavo, da tutto il mondo della matematica e di TEX.

Enrico

GuIT challenge (italiano)

GuIT members

Sommario Per la seconda volta nella storia di ArsTEXnica viene indetta una GuIT *challenge*. La sfida è nata nel gruppo Telegram "GuIT members" e consiste nell'elaborare tramite LaTeX un file di dati sperimentali relativi a un fenomeno fisico lineare, per collocare i punti rappresentativi su un diagramma cartesiano insieme alla retta di regressione, stampando i parametri calcolati. Un ricco premio per i vincitori!

1. Introduzione

Il fisico Mario Rossi sta studiando un fenomeno presumibilmente lineare ed esegue misure in laboratorio adatte a verificare la sua ipotesi; misura la grandezza x che produce il fenomeno e misura anche una delle caratteristiche y che il fenomeno presenta per effetto della stimolazione x. L'incertezza di misura di x coincide sostanzialmente con l'errore massimo dello strumento con cui viene misurata; l'incertezza di y è invece molto maggiore di quella strumentale poiché il fenomeno è influenzato da altre sollecitazioni sulle quali Mario Rossi non ha nessun controllo e che quindi costituiscono delle sorgenti di incertezza casuali. Percentualmente, l'incertezza sui valori di x è molto minore rispetto a quella sulle y.

Mario riporta i valori di x e y prima di tutto in una tabella. Per comodità ha memorizzato questi dati anche in un file, liberamente scaricabile all'indirizzo `https://github.com/GuITeX/ars-contest/tree/master/02-regression`. Per mostrare la sua struttura, nella Tabella 1 vengono riportate le prime 25 righe di questo file. I dati nel file sono organizzati nel formato CSV (*comma separated values*), cioè in un file di testo ASCII composto di due colonne di dati separati dalla virgola, in cui la prima colonna è occupata dai valori delle x e la seconda colonna dai valori delle y. Il carattere di a capo è un *carriage return* e un *line feed*.

Successivamente Mario riporta in grafico i dati della tabella per giudicare se a occhio nudo i punti seguono ragionevolmente un andamento lineare oppure no; a questo proposito egli calcola anche i parametri della retta di regressione e traccia sul diagramma anche la retta di regressione in modo da avere immediatamente la percezione visiva della qualità dei risultati ottenuti.

Disgraziatamente i calcoli per determinare i parametri della retta di regressione sono noiosi e ripetitivi, e sarebbe opportuno poterli fare in automatico contemporaneamente al posizionamento delle coppie x, y sul diagramma stesso.

Essendo un utente di LaTeX, pensa di poter prendere due piccioni con una fava: usare cioè LaTeX per disegnare il grafico con i punti sperimentali rappresentati dai punti x, y e allo stesso tempo per calcolare i parametri a e b della retta di regressione $y = ax \;\; b$ cercata e di conseguenza anche disegnarla sullo stesso grafico.

Purtroppo però Mario Rossi non è così esperto di LaTeX e ha quindi chiesto al GuIT un aiuto.

Tabella 1. Prime 25 righe del file dei dati sperimentali. La prima colonna sono le x, la seconda colonna le y. Il file completo (100 righe) è liberamente scaricabile all'indirizzo `https://github.com/GuITeX/ars-contest/tree/master/02-regression`.

```
0.104,0.243
0.237,0.052
0.331,0.201
0.401,0.592
0.537,0.638
0.603,0.617
0.725,0.885
0.807,0.943
0.950,0.782
1.020,0.905
1.139,1.162
1.232,1.192
1.343,1.413
1.439,1.486
1.539,1.506
1.608,1.437
1.746,1.884
1.819,1.807
1.928,1.729
2.029,2.098
2.142,2.135
2.249,2.033
2.320,2.313
2.423,2.305
2.509,2.512
. . . . . .
```

2. La competizione

Per aiutare Mario Rossi si deve realizzare un pacchetto che abbia almeno le seguenti caratteristiche:

- Una volta lanciato, chiede nel terminale il nome del file dei dati da analizzare
- Deve essere in grado di analizzare un file come quello riportato all'indirizzo `https://github.com/GuITeX/ars-contest/tree/master/02-regression` ma con un numero generico di dati sperimentali, anche non ordinati in ordine crescente.
- In output restituisce un file pdf contenente il grafico dei dati sperimentali e la retta di regressione lineare. Deve inoltre stampare i coefficienti della retta.
- Deve sfruttare solo le funzionalità del nucleo di LaTeX e con l'uso del minimo di pacchetti gestiti dal LaTeX Team, escludendo, quindi, i pacchetti forniti dalla miriade di altri utenti
- Si può ricorrere ad interfacce con i moduli interni scritti in LaTeX3, ma già presenti nelle distribuzioni complete e aggiornate del sistema TeX. Si possono usare sola-

mente pacchetti contenuti nel kernel di LaTeX o nelle cartelle `/tex/latex/base/`, `/tex/latex/tools/`, `/tex/latex/grpahics/` e `/tex/latex/etoolbox/`.

Funzionalità aggiuntive dal punto di vista grafico e/o dell'interfaccia sono ben accette e concorreranno al giudizio finale. Funzionalità aggiuntive dal punto di vista del calcolo statistico (incertezza sui parametri della retta, coefficiente di determinazione del fit per quantificare la bontà dell'ipotesi di linearità tra x e y, ecc. . .) possono ovviamente essere inserite, ma serviranno solamente a dirimere il giudizio finale a parità di altri fattori.

3. Chi può partecipare e come

Qualunque persona che usi LaTeX, italiana o straniera, può partecipare a questa competizione. Sono esclusi sia l'attuale che il precedente direttore di ArsTeXnica.

Per partecipare bisogna inviare il proprio pacchetto zippato all'indirizzo `arstexnica@guitex.org` **entro e non oltre il 31 agosto 2021**.

I file contenuti nell'archivio `.zip` devono essere tutti anonimi, cioè non devono contenere né le sue generalità né altre informazioni che lo possano identificare, perché il file verrà poi girato dal direttore di ArsTeXnica ai membri della commissione giudicatrice. L'unico che conoscerà l'identità di tutti i partecipanti sarà il direttore di ArsTeXnica.

Il pacchetto `.zip` deve contenere un singolo file `.tex` con il codice. Deve contenere inoltre un brevissimo file di testo, che descriva i punti di forza del pacchetto, la strategia e i punti più caratteristici del codice.

Il file deve essere compilabile con `pdflatex` o `xelatex`. Si può usare anche `lualatex` ma in questo caso i partecipanti non gareggeranno con gli altri che non possono usare Lua, ma faranno parte di una competizione separata di soli utenti Lua.

L'uso di ConTeXt è escluso, come qualunque altro programma di composizione che non faccia uso del linguaggio LaTeX. È ulteriormente vietato l'uso di software esterno attraverso operazioni di shell o di `\write18`.

4. Il giudizio, i risultati e i premi

La commissione giudicatrice sarà composta dal direttore di ArsTeXnica e da tre membri scelti dal direttore fra i componenti del Comitato Scientifico di ArsTeXnica. Chiaramente potranno essere scelti solamente i componenti del Comitato che non partecipano come concorrenti.

Uno dei test fondamentali che verranno condotti sui codici ricevuti sarà quello di dare loro in input un file di testo con un formato identico a quello fornito in questo bando, ma con valori diversi delle x e delle y.

I risultati saranno pubblicati sul numero di ArsTeXnica di ottobre 2021.

I premi consistono in copie del testo *TEX by topic* adeguatamente avvolte nella carta regalo con il simbolo del GUIT.

GUIT members
arstexnica@guitex.org

GUIT challenge (english)

GUIT members

Sommario For the second time in the history of ArsTEXnica a GUIT *challenge* is launched. The challenge was born in the Telegram group "GUIT members". It consists in the elaboration, by LATEX, of a file containing a set of experimental data of a linear phenomenon. The goal is to draw the experimental points on a cartesian diagram together with the regression line, and to print the calculated parameters of the line. A rich prize for the winners!

1. Introduction

The physicist Mario Rossi is investigating a phenomenon, presumably linear, and he performs measurements in his laboratory to verify his hypothesis; he measures the quantity x which generates the phenomenon and he measures also one of the characteristics y showed by the phenomenon under the effect of the stimulation x. The measurement uncertainty on x is equal to the maximum error of the instrument used to measure it; the uncertainty on y is instead much larger than the instrumental one since the phenomenon is affected by other causes on which Mario Rossi has no control and therefore they are sources of random uncertainties. The relative uncertainty of x is much smaller than the relative uncertainty of y.

First of all, Mario reports the values of x and y in a table. For convenience he store these data in a file, freely downloadable at the URL `https://github.com/GuITeX/ars-contest/tree/master/02-regression`. In order to show the structure of the data, in Table 1 the first 25 lines of this file are reported. The data in the file are organized in the CSV (*comma separated values*) format, that is a ASCII text file composed by two columns of data, separated by a comma, in which the first column reports the values of x while the second column the values of y. The newline character is a *carriage return* and a *line feed*.

Subsequently Mario graphs the data of the table to judge if the points reasonably follow a linear trend or not; in this regard he computes the parameters of the regression line and he draws this line on the graph in order to judge the quality of the obtained results.

Unfortunately, the calculations to determine the parameters of the regression line are boring and repetitive. It would be appropriate to perform them automatically, by positioning, at the same time, the experimental data on the same graph.

Being a LATEX user, he thinks to kill two birds with one stone: using LATEX to draw the graph with the experimental data consisting in the x, y points and, at the same time, to compute the parameter a e b of the regression line $y = ax \;\; b$, and finally to draw also this line on the same graph.

But sadly, Mario Rossi is not so expert in LATEX and therefore he asked GUIT a help.

Tabella 1. First 25 lines of the experimental data file. The first column reports the x's, the second column the y's. The full file (100 lines) is freely downloadable from the URL `https://github.com/GuITeX/ars-contest/tree/master/02-regression`.

```
0.104,0.243
0.237,0.052
0.331,0.201
0.401,0.592
0.537,0.638
0.603,0.617
0.725,0.885
0.807,0.943
0.950,0.782
1.020,0.905
1.139,1.162
1.232,1.192
1.343,1.413
1.439,1.486
1.539,1.506
1.608,1.437
1.746,1.884
1.819,1.807
1.928,1.729
2.029,2.098
2.142,2.135
2.249,2.033
2.320,2.313
2.423,2.305
2.509,2.512
. . . . . .
```

2. The challenge

In order to help Mario Rossi a package must be prepared, having, at least, the following features:

- Once launched, it asks in the terminal the file name to be analyzed
- It must be able to process a file like the one reported at the URL `https://github.com/GuITeX/ars-contest/tree/master/02-regression` but with a generic numbero of experimental data, even not ordered in ascending or descending order.
- The output must consists in a pdf file containing the graph with the experimental data and the regression line. The parameters of the regression line must be printed too.
- It can only use the functionalities of the LaTeX core and the least possible number of packages managed by the LaTeX Team, therefore excluding the many packages provided by other users
- Interfaces with internal modules written in LaTeX3 can be used, if already present in the full and updated TeX distributions. Only packages contained in the LaTeX kernel can

be used or in the folders `/tex/latex/base/`, `/tex/latex/tools/`, `/tex/latex/graphics/` and `/tex/latex/etoolbox/`.

Additional functionalities from the graphical and interface point of view are welcome and they will contribute to the final judgement. Additional functionalities from the statistics point of view (uncertainty of the line parameters, determination coefficient to quantify the goodness of fit and the hyphothesis of linearity between x and y, etc. . .) can be obvioysly implemented, but they will only serve to settle the final judgementall other factors being equal.

3. Who can participate and how

Any LaTeX user, italian or foreign, can participate to this contest. Current and previous ArsTeXnica editors are excluded.

In order to participate, the candidates must send their zipped package to the email address `arstexnica@guitex.org` **not later than 31 august 2021**.

The files contained in the `.zip` archive must be all anonymous, which means they must not contain any generality nor any other information useful to identify the participant. This is necessary because the package will be forwarded to the members of the selection board by the ArsTeXnica editor. The only person knowing the identity of all the participants will be the ArsTeXnica editor.

The `.zip` package must contain a single `.tex` file with the code. It must contain also a very short text file describing the strengths of the package, the strategy and the most importaint points of the code.

The file must be compiled by `pdflatex` or `xelatex`. Also `lualatex` can be used but in this case the participants will not compete with the others non Lua users, but they will be part of another separate competition of only Lua users.

The usage of ConTeXt is excluded, like any other typesetting program not using the LaTeX language. Moreover, the usage of external software, for example by shell commands or `\write18` is forbidden.

4. Evaluation, results, prizes

The selection board will be composed by the ArsTeXnica editor and by three members of the ArsTeXnica Scientific Committee chosen by the editor. Clearly, only the Committee members who will not participate to the contest can be chosen.

One of the fundamental test adopted on the received packages will be running the code on an input data file, with the same format as the one described in this call, but with different values of the x's and the y's.

The final decision will be published in the ArsTeXnica volume of october 2021.

The prizes consist in copies of the book *TEX by topic* properly wrapped in the gift paper reporting the GUIT logo.

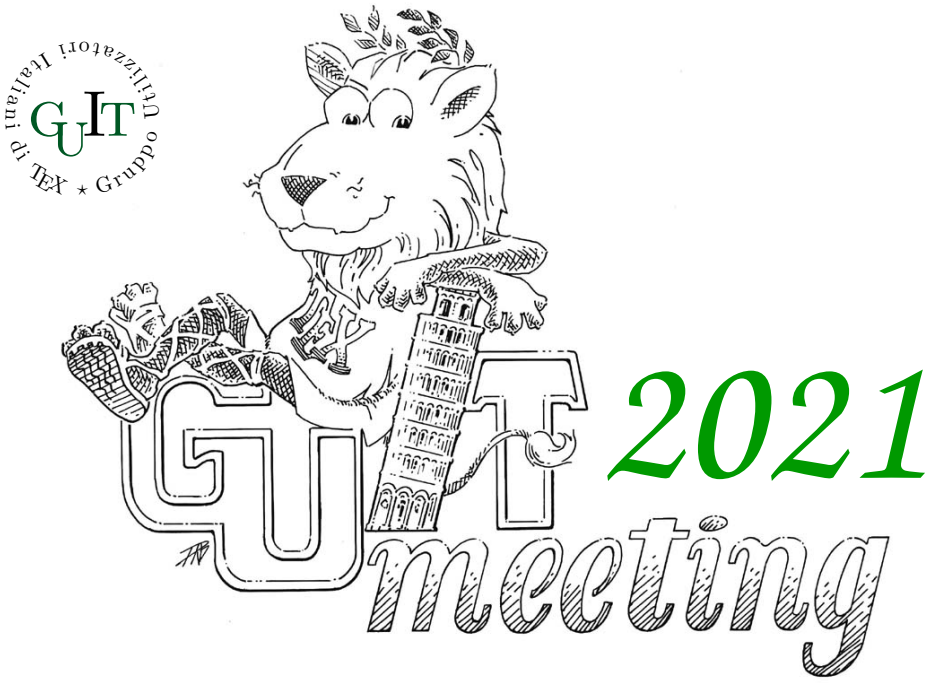
GUIT members
`arstexnica@guitex.org`

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.



Diciottesimo convegno nazionale su TEX, LATEX e tipografia digitale

Firenze
16 ottobre 2021

Call for Papers

Firenze è stata scelta come sede per il diciottesimo Convegno annuale su TEX, LATEX e tipografia digitale organizzato dal Gruppo Utilizzatori Italiani di TEX. Il Convegno sarà un momento di ritrovo e di confronto per la comunità LATEX italiana, tramite una serie di interventi atti sia a contribuire all'arricchimento sia a supportarne lo sviluppo.
Attenzione: per effetto delle disposizioni di sicurezza circa l'epidemia di Coronavirus, l'evento potrebbe essere tenuto per via telematica.
Maggiori informazioni sul Convegno e sulle modalità di presentazione degli interventi saranno disponibili all'indirizzo:

https://www.guitex.org/home/guit-meeting-2021/


10031
9 771828 236001