てふてふ

Friday, November 3rd, 2023

Systems and Control Laboratory, Department of Aeronautics and Astronautics,
Graduate School of Engineering, **Kyoto University**

H. Asakura*

* GitHub: https://github.com/ha-a

Table of Contents

1. はじめに

本資料は TEX や LATEX，latexmk の中身やコンパイルとはなにかについて，備忘録の意味も込めて概説するものです．インストールの方法などについて解説するわけではなく，これを読んだら環境構築が楽になるわけでもなければ，tex が上手に書けるようになるわけでもないです．これまで気にせずに使っていたけど，よく考えると何も知らなくて，たまに理不尽に言うことを聞いてくれないから怖い “アイツ” についてほんの少しでもその中身と歴史を知ろうというわけです．実際ネット上には，TEX と LATEX の区別がついてないものや，使ってはいけないもしくは強く推奨されないコマンドを紹介しているようなブログや Qiita 記事がたくさんあります．というか，当時正しくても今となっては正しくないものも多かったりします．そんなことを念頭に本資料ではなるべく，少なくとも執筆当初は嘘ではないこと，を書くようにしているつもりです．が，私もｶﾞﾁﾌﾟﾛではないので，間違えてることもあるかもしれません．もし間違いを見つけた場合はお近くの私までご連絡をお願いします．

2. TEX と LATEX

まず，TEX と LATEX は異なるものである．しかし，そもそも名前が似てるし，両者の区別が曖昧のままでも所望の文書を作成することはとりあえずできてしまうので，別にどっちでもいいという考え方もできる．が，違いがあるから名前が違うのである．ここでは TEX と LATEX の何が違うのかを説明し，言葉として TEX がつく様々なものについて，今後混乱・誤用しないようにざっくりと解説していく．

ところで，TEX は TEX であり，tex でも TEX でもなければ Tex でもない．しかし，場合によって TeX と表記されることはある．また，本資料では拡張子が *.tex となっているものについては tex ファイルなどと表記することにする．LATEX も同様で，LATEX かもしくは LaTeX である．ちなみに，TEX の発音はアルファベットの X /ɛks/として読むのではなく，無声軟口蓋摩擦音/x/と発音するのが正しい（諸説あるが）*.

2.1. TEX とはなにか

TEX とは 1978 年に Donald E. Knuth[1] が発表した “組版†ソフトウェアと組版言語” である[3]‡．つまり，“文章そのもの” と “文章の構造を指定する命令” が書かれたテキストファイル（この命令を記す言語も TEX という）を読み込み，その命令に従って文章を組版するソフトウェアが TEX である．組版結果は DVI 形式 (DeVice-Independent) に書き出されるが，この dvi ファイルは文書の見た目のレイアウト（紙面のどの位置にどの文字を配置する，などの情報）を画像形式・表示デバイス・プリンタにまったく依存しない形で記録している[4]．結局 TEX をとてもざっくりまとめると（Knuth の思う）美しい組版がデバイスによらない形式で出力される，ということになろう．

*日本語で表記するなら「てふ」であり，歴史的仮名遣い的にその発音は/tʃoː/である．
†原稿及びレイアウトの指定に従って，文字・図版・写真などを配置する作業の総称[2]．
‡由来はギリシャ語で美術や工芸を意味する “τέχνη”/téxni/

2.2. LATEX とはなにか

TEX での命令は「横方向に何センチメートルの空白を作る」のように非常に原始的なものなので，使い勝手が良くない．そこで TEX を使ってもっと簡単に論文やレポートを作成したいという要望から LATEX は開発された[5]．TEX の原始的な命令を組み合わせて新しい命令（マクロ）を作ることができる（`\def` みたいなやつ）が，TEX で文書作成する際には多量のマクロが必要になるため，それらの必要なマクロが一通り揃えられた “マクロ体系” の 1 つが LATEX である．

繰り返しになるが，この世にいくつも存在する TEX を楽に使うためのマクロ体系のひとつが LATEX である．現代の多くの `tex` ファイルはマクロが準備された LATEX で処理されることが前提であり，それを TEX で処理しようとして目的のものが得られることはまずない．次節で述べる通り，TEX にも LATEX にも沢山の種類がある．つまり，同じ LATEX でも pLATEX と LuaLATEX とでは用意されているマクロや定義に違いがあるため，`tex` ファイルが想定するそれと処理する LATEX が食い違っていればエラーが出るのも当然である．また，LATEX で定義されたマクロのおかげで成り立つ事象を TEX 全体で成り立つと理解するのは危険なのである．

2.3. その他のﾅﾝﾄｶ TEX とか TEXﾅﾝﾄｶとかﾅﾝﾄｶ TEXﾅﾝﾄｶとはなにか

TEX と LATEX 以外にも名前に TEX がつくものは数多くある．違いがわかるように簡単にまとめておこう[6]．

(1) TEX の仲間

組版を行うソフトウェアとして，もともとの TEX の機能が様々に拡張されたものがある．例えば，

- ε-TEX: オリジナルの TEX と 100% の互換性を保ちながら TEX の少々不便なところを補う実用性の高い拡張．
- pTEX: 縦組みに対応し，和文組版できる拡張 TEX．
- ε-pTEX: ε-TEX を取り入れた pTEX．
- upTEX: pTEX の Unicode*版．つまり，pTEX の内部コードを Unicode 化している．簡単に言ってしまえば扱える文字が増える．例えば Fig. 2.1 では同じ日本語で書かれた `tex` ファイルを pTEX と upTEX で処理した際の出力の違いである（正確には pTEX に対応したマクロ体系 pLATEX を利用した場合と upLATEX を利用した場合の違い）．pTEX の方では一部の文字（例えば踊り字「〻」や非常用漢字「沆瀣」など）が出力されていないことがわかる．Unicode を用いることでこの世に存在するより幅広い文字を扱えることになる．
- pdfTEX: ε-TEX の拡張．DVI ではなく，PDF を直接出力する．少しだけ触れたが，`dvi` ファイルとはデバイスに依存しない，TEX 固有の出力形式である．人間が読めるようになっているわけではないため，別に PDF などに変換する操作（`dvipdfmx` とかそういうの）が必要である．pdfTEX は出力としてこの `dvi` ではなく `pdf` ファイルを直接出力するため，そういった変換も不要なほか，ハイパーテキストリンクや目次といった PDF

*様々な言語の文字を単一の文字コードで取り扱うために開発されている文字コード[7],[8]

の機能を直接扱うことができる．

- XƎTEX(/ziːtɛx/)[9],[10] : ε-TEX の拡張．符号空間を Unicode 全体に拡大したもの（上述 upTEX の説明参照）．また，OpenType フォントに対応している．これはつまり，OS にインストールされているフォントをそのまま利用することが可能ということである[11]．もともとの TEX では組版を行うとき，組み立てる文字の情報を `tfm` ファイルと呼ばれるものから参照する．したがって好きなフォントを使うためにはそのための `tfm` ファイルを用意する必要があったのだが，XƎTEX ではその必要はなく，インストールさえしてしまえば好きなフォントが使える．
- LuaTEX: pdfTEX の後継．OpenType フォント，Unicode に対応しているほか，Lua という軽量スクリプト言語を利用できる．この Lua の利用によりあらゆるものがカスタマイズ可能となり，究極の柔軟性が得られる[12]．

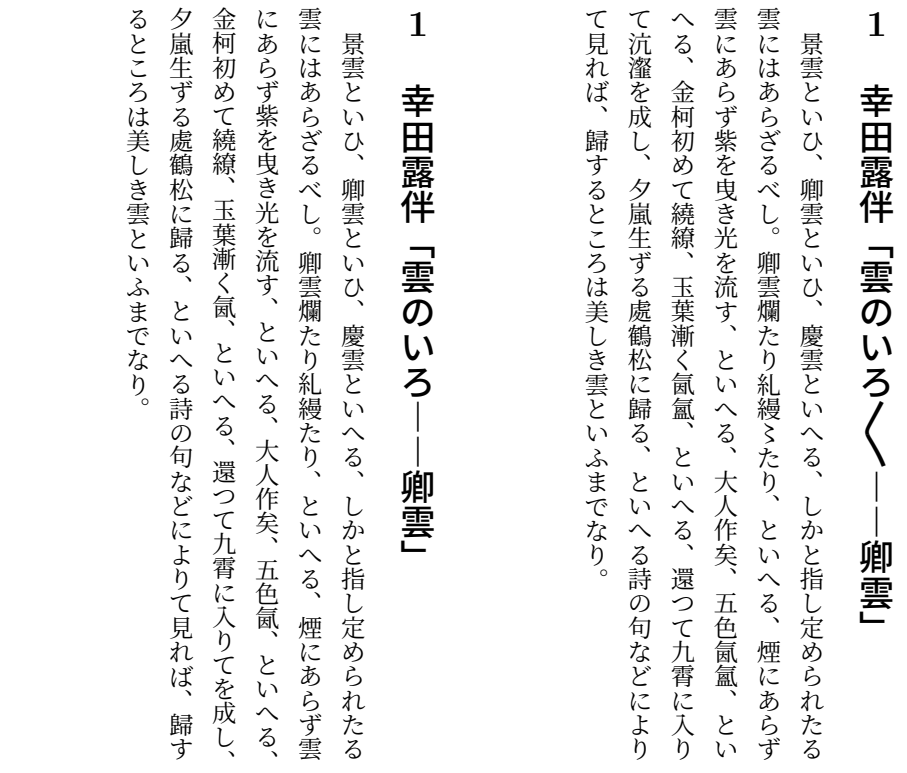
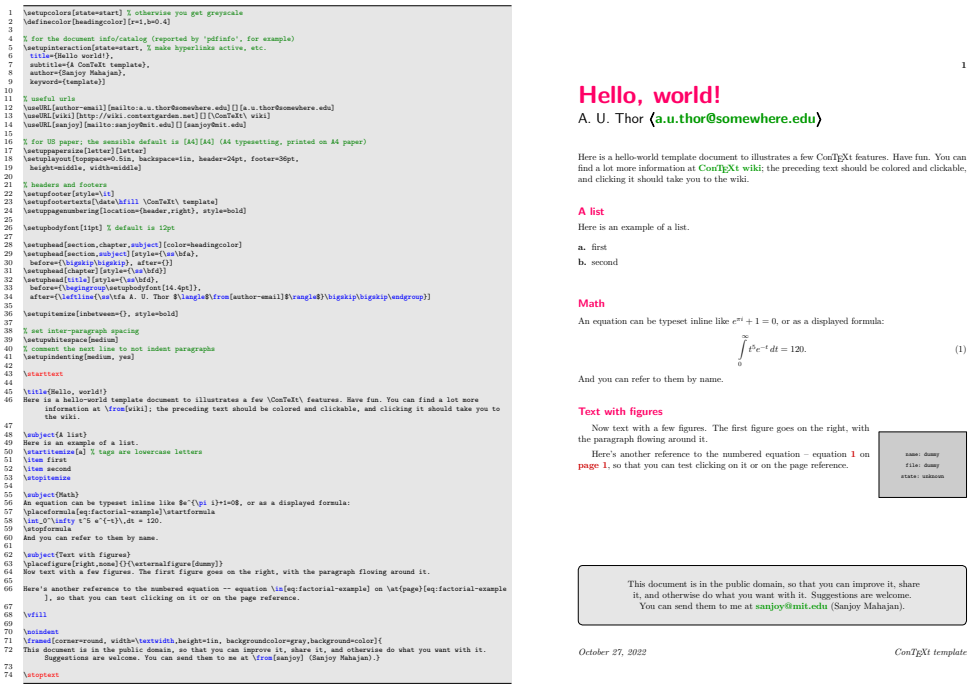


Fig. 2.1. pTEX engine (left) vs upTEX engine (right).[13]

(2) LATEX の仲間

TEX を使いやすくするためのマクロ体系は LATEX 以外にもあって，例えば以下のようなものがある：

- plain TEX: Knuth 自身が開発したもので，組版に最低限必要となる程度の機能しか持たない．
- pLATEX: pTEX に対応した LATEX．
- upLATEX
- pdfLATEX
- XELATEX
- LuaLATEX
- ConTEXt: 文書の体裁もユーザー自身によってすべて調整可能とすることを目標としているマクロ体系[14]．外部パッケージなしであらゆる部分の調整が可能となるようになっているため，LATEX で起こるようなパッケージ同士の衝突などはおこらない．Fig. 2.2 に ConTEXt を使った `tex` ファイルの例とその出力 `pdf` ファイルを示す．左のコードの中で赤く示した命令 (`\starttext` と `stoptext`) の間が本文であり，その前がいわゆるプリアンブルである．パッケージ等を用いず，すべての設定をユーザー自身が行う必要があり，プリアンブルが長くなっている印象である．一方，体裁は自分で好きなように決められることがそもそものコンセプトなので，セクション（ConTEXt 的には`\subject`）の色や図の位置などの調整は自由かつ思い通りに扱いやすい．



```
\setupcolors[state=start] % otherwise you get greyscale
\definecolor[headingcolor][r=1,b=0.4]

% for the document info/catalog (reported by 'pdfinfo', for example)
\setupinteraction[state=start, % make hyperlinks active, etc.
  title={Hello world!},
  subtitle={A ConTeXt template},
  author={Sanjoy Mahajan},
  keyword={template}]

% useful urls
\useURL[author-email][mailto:a.u.thor@somewhere.edu][][a.u.thor@somewhere.edu]
\useURL[wiki][http://wiki.contextgarden.net][][\ConTeXt\ wiki]
\useURL[sanjoy][mailto:sanjoy@mit.edu][][sanjoy@mit.edu]

% for US paper; the sensible default is [A4][A4] (A4 typesetting, printed on A4 paper)
\setuppapersize[letter][letter]
\setuplayout[topspace=0.5in, backspace=1in, header=24pt, footer=36pt,
  height=middle, width=middle]

% headers and footers
\setupfooter[style=\it]
\setupfootertexts[\date\hfill \ConTeXt\ template]
\setuppagenumbering[location={header,right}, style=bold]

\setupbodyfont[11pt] % default is 12pt

\setuphead[section,chapter,subject][color=headingcolor]
\setuphead[section,subject][style={\ss\bfa},
  before={\bigskip\bigskip}, after={}]
\setuphead[chapter][style={\ss\bfd}]
\setuphead[title][style={\ss\bfd},
  before={\\begingroup\setupbodyfont[14.4pt]},
  after={\leftline{\ss\tfa A. U. Thor $\langle$\from[author-email]$\rangle$}\bigskip\bigskip\endgroup}]

\setupitemize[inbetween={}, style=bold]

% set inter-paragraph spacing
\setupwhitespace[medium]
% comment the next line to not indent paragraphs
\setupindenting[medium, yes]

\starttext

\title{Hello, world!}
Here is a hello-world template document to illustrates a few \ConTeXt\ features. Have fun. You can find a lot more
    information at \from[wiki]; the preceding text should be colored and clickable, and clicking it should take you to
    the wiki.

\subject{A list}
Here is an example of a list.
\startitemize[a] % tags are lowercase letters
\item first
\item second
\stopitemize

\subject{Math}
An equation can be typeset inline like $e^{\pi i}+1=0$, or as a displayed formula:
\placeformula[eq:factorial-example]\startformula
\int_0^\infty t^5 e^{-t}\,dt = 120.
\stopformula
And you can refer to them by name.

\subject{Text with figures}
\placefigure[right,none]{}{\externalfigure[dummy]}
Now text with a few figures. The first figure goes on the right, with the paragraph flowing around it.

Here's another reference to the numbered equation -- equation \in[eq:factorial-example] on \at{page}[eq:factorial-example
    ], so that you can test clicking on it or on the page reference.

\vfill

\noindent
\framed[corner=round, width=\textwidth,height=1in, backgroundcolor=gray,background=color]{
This document is in the public domain, so that you can improve it, share it, and otherwise do what you want with it.
    Suggestions are welcome. You can send them to me at \from[sanjoy] (Sanjoy Mahajan).}

\stoptext
```

1

Hello, world!

A. U. Thor ⟨a.u.thor@somewhere.edu⟩

Here is a hello-world template document to illustrates a few ConTEXt features. Have fun. You can find a lot more information at ConTEXt wiki; the preceding text should be colored and clickable, and clicking it should take you to the wiki.

A list

Here is an example of a list.

a. first
b. second

Math

An equation can be typeset inline like $e^{\pi i}+1=0$, or as a displayed formula:

$$\int_0^\infty t^5 e^{-t}\,dt = 120. \tag{1}$$

And you can refer to them by name.

Text with figures

Now text with a few figures. The first figure goes on the right, with the paragraph flowing around it.

Here's another reference to the numbered equation – equation 1 on page 1, so that you can test clicking on it or on the page reference.

name: dummy
file: dummy
state: unknown

This document is in the public domain, so that you can improve it, share it, and otherwise do what you want with it. Suggestions are welcome.
You can send them to me at sanjoy@mit.edu (Sanjoy Mahajan).

October 27, 2022 *ConTEXt template*

Fig. 2.2. Example of ConTEXt (left: source tex file, right: pdf file)

少し話は変わるが，LATEX のバージョンについても名前がついている：

- LATEX 2$_\varepsilon$: 現在（2023-11-03）の主流.
- LATEX2.09: 古いバージョン.
- LATEX3: 開発中の次期バージョン.

ここまで，組版ソフトウェアの TEX とその拡張，TEX を楽に使うためのマクロ体系の LATEX とその拡張および仲間について説明した．ここからは TEX の名前がついていて，TEX とともに使うもの*について見ていこう.

(3) 文献管理ソフト

`\cite` 等を使って文献の引用をする場合，そのタイトルや著者，出版年，出版社等々の文献情報を管理しなくてはならない．`thebibliography` 環境でひとつずつ（`\bibitem`）自分の手で文献情報を入力することも可能だが，数が増えると大変である上，表記に一貫性を持つための調整も手間である．そんなときに，LATEX と組み合わせて文献データベースから自動的に参考文献リストを作るためのツールが文献管理ソフトである．よく耳にするのは

- BIBTEX: 書誌情報のデータベース（`bib` ファイル）を用いて, 自動で参考文献リストを作成する. このデータベース `bib` ファイルのなかで各書誌情報はまずエントリごとに種別分けされ（`@article`, `book`, `proceedings`, etc.），それぞれの情報を保存する（`title`, `author`, `journal`, etc.）.
- pBIBTEX: pTEX に特化．日本語の文献を扱える.
- upBIBTEX

などがあるが，最近は他にも

- BibLATEX(+Biber) : LATEX で用いるパッケージ．そもそも BIBTEX は参考文献のフォーマットは `bst` ファイルというもので設定されるのだが，これまた編集が大変である．そのため，よりカスタマイズが容易なのが，本パッケージである．BibLATEX のバックエンド（文献のソートを担当）として `biber` か `bibtex` を用いることになるが，`biber` のほうが `bibtex` を用いるよりさらにカスタマイズの柔軟性が上がる[15].

が次世代文献管理ソフトとして挙げられる.

(4) 統合開発環境

TEX 周りの作業がしやすいように整備されたエディタ．これさえあれば TEX がつかえる，というものではない.

- TEXShop: macOS 専用の LATEX 統合環境．シンプルな UI と直感的な操作性が特徴.
- TEXworks: TEXShop をモデルにすべてのシステム上で実現すべく開発された LATEX 用統合環境.
- TEXstudio: 豊富な機能と高度な LaTeX サポートがあり，拡張性が高い．全システム上で動く.

*LATEX は TEX “を” 使うもの

(5) ディストリビューション

TEX がまともに使えるようになるためには TEX さえあればいい，わけではない．TEX なのか pTEX なのか LuaTEX なのか．LATEX なのか ConTEXt なのか．BIBTEX なのか `Biber` なのか．統合開発環境は使うのか．使うならどれか．さらに言えば，自分の文書作成では LuaLATEX のみ使うと決心したとしても，学会によっては upLATEX で動くことのみ想定したフォーマットを配っている場合もあり，そのフォーマットを LuaLATEX 用に書き換えるのはただ面倒で生産性があまりなく，こうなると最初から upLATEX を準備しておくのが望ましい．これ以外にも，特定のマクロ体系を更に拡張するための “パッケージ” が必要なことがほとんどで，それらも新たに必要になるたびインストールしなくてはならない．結局，TEX を使えるようになるためには非常に多くのものをインストールしなくてはならないが，一つ一つ個別にインストールするのは大変であり，これらを一括でインストールできるのがディストリビューションである．

日本国内で多く用いられているディストリビューションは

- TEXLive[16]

であるが，他にも

- MacTEX: TEXLive をベースにした macOS 専用のディストリビューション．
- BasicTEX: LATEX の文書作成に必要最低限なものが含まれる macOS 専用ディストリビューション．ε-pTEX などが含まれない．
- MiKTEX: Windows，macOS，Linux で動作するディストリビューション．日本語 pTEX が含まれない．

などが存在する．

2.4. 番外編：文書クラス

さて，TEX と名のつくものについて説明してきたが，ここで TEX を名前に含まないものの，種類がいくつかあってややこしいものとして，文書クラスについて解説する[17]．`tex` ファイルの一番最初に`\documentclass` とか書くあれである．

- `article`: 論文やレポートなどの一般的な文書．文章の構造は `part`, `section`, `subsection`, `subsubsection`, `paragraph`, `subparagraph` の 6 つのレベルに分けられる．
- `report`: 論文やレポートのより長い文書. 構造としては `article` に `chapter` が追加される (`part` と `section` の間)．また，（デフォルトの設定では）`part` が 1 ページ使って表示される（Part I ○○とだけ書かれたページができる)．
- `book`: 書籍．構造は `report` と同じだが，奇数ページと偶数ページのレイアウトが変わる．例えば，`part` の示すページは必ず奇数ページ（見開きの右側）になる．

- jclasses (`jarticle`, `jreport`, `jbook` etc.): 和文横書き対応の文書クラス．
- `tarticle`: 和文縦書き対応の `article`．
- jsclasses (`jsarticle` etc.): 日本語 (u)pLaTeX 用の新しい文書クラス[18]．JIS フォントメトリックを使用している．簡単に言えば「文字の組み方」が綺麗に，読みやすくなる．例えば下の例では，小さい文字が連続している部分で `jarticle` などの min10 フォントメトリックでは不自然に詰められ，美しくなかったり読みづらかったりするのが jsclasses では解消される[19]．他にも日本語中の英語の組み方などに差が出る．

ちょっとチャックをチェック
ちょっとチャックをチェック

Fig. 2.3. jclasses (above) vs jsclasses (below)

- ltjsclasses (`ltjsarticle` etc.): jsclasses を LuaLaTeX 用に改変したもの．
- bxjsclasses (`bxjsarticle` etc.): jsclasses の派生で，(u)pLaTeX 依存の部分を切り離したもの．つまり，LuaLaTeX など他のエンジンを用いても綺麗な日本語文書を作成できる．
- `jlreq`:「日本語組版処理の要件」*を満たす LuaLaTeX，(u)pLaTeX 用の文書クラス．
- `beamer`: スライド．「Beamer のすゝめ」も読んでね．

3. 再三のコンパイルと自動化

ここまで `tex` ファイルから文書出力を得ることを曖昧に “処理” と表現してきた．この “処理” とは実際にはどのような “処理” を行っているのかについて説明していく．具体的には，pdfTeX 系なのか否かや相互参照・文献情報の有無によって必要な処理の種類・順番や回数が変わる．最近では `latexmk` というコマンドのおかげで，このことを意識しなくてもいいことが多いが，この処理について知っておくことは `latexmk` 丸投げよりも効率がいい処理も可能で，デバッグの役にも立つので，知っておいて損はないと思う．なお，`latexmk` という名前からも分かる通り，今後の話は LaTeX まわりの話である．

3.1. tex ファイルから pdf ファイルまで

`tex` ファイルから `pdf` ファイルを出力するまでの処理の流れについて簡単に説明する[22]．なお，以下では作成した `tex` ファイルの名前を `hoge.tex` とする．ユーザは `hoge.tex` の中身に従い，適切なマクロ体系を選択する．というよりも，日本語を使うのか，Unicode を使うのか，好みのフォントを選びたいのか，といった目的に応じて選択すべきマクロ体系があり，それに合わせて `hoge.tex` を作成する．本節では，例として LuaLaTeX+upBibTeX を用いることにする．

*標準化団体 W3C (World Wide Web Consortium) の技術ノート “Requirements for Japanese Text Layout”[20],[21] のことで，字詰めやルビ，ページ構成や図表に関してなどあらゆる規定が書かれている．

(1) はじめてのらてふこんぱいる

選択したマクロ体系で処理をする:

```
lualatex hoge.tex
```

これにより，次のファイルが生成される:

- 出力ファイル
  - `hoge.dvi`: LaTeX, (u)pLaTeX などを用いた場合
  - `hoge.pdf`: pdfLaTeX, XeLaTeX, LuaLaTeX などを用いた場合
- 補助ファイル `hoge.aux`: 相互参照・ラベルの情報を保存する．
- ログファイル `hoge.log`: コンパイル時の記録．エラーや警告も保存される．

このとき，もともと相互参照に関する `hoge.aux` を読み込まずに `hoge.tex` のコンパイルを行ったため，今回の出力で適切な相互参照はできていない．相互参照するべきところ（`\ref`）では "**??**" が表示され，`hoge.log` には参照が見つからなかった旨の警告*が記録される．これは一度の処理を高速・省メモリで終えるために，1 回あたりのコンパイルでは `tex` ファイルを最初から最後まで順に読み込みながら出力を行っているからである．このコンパイルの最中に補助ファイル（`hoge.aux`）にラベルを前から順に保存していく．

次のステップは文書の中に相互参照が含まれるかどうかによって以下のように場合分けされる:

	次の処理
参照したい文献`\cite` がある場合	(2) 参考文献処理
参考文献以外の相互参照`\ref` のみある場合	(3) 相互参照を処理する LaTeX コンパイル
相互参照も参考文献もない場合	(5) 文書の pdf 出力を得る

(2) 参考文献処理

所望の形に整形された参考文献の一覧を作る:

```
upbibtex hoge.aux
```

具体的には，(1) はじめてのらてふこんぱいる によって `hoge.aux` ファイル内に保存された文献リストの雛形から所望の形の文献リストが `hoge.bbl` に生成される[23]．`hoge.bbl` の中身は `thebibliography` 環境であり，BibTeX 等を使わずに手動で`\bibitem` などとちまちま書いていくアレである．つまり，`hoge.bbl` にはそれぞれの情報を含めた文献項目（`bibitem` とか `entry`）が所望の形式で出力できるように収められている．

次のステップは (3) 文献リストを出力する LaTeX コンパイル．

*LaTeX Warning: There were undefined references.

(3) 文献リストを出力する LaTeX コンパイル

`hoge.bbl` を使って，文献リストを出力しつつ，補助ファイル `hoge.aux` に文献に関わる相互参照情報を保存する．

```
lualatex hoge.tex
```

ちなみに，(1) で作られた相互参照は今回の LaTeX コンパイル時に読み込まれた `hoge.aux` を使って，解決される．なぜならこのコンパイル時にはすでに 1 回目のコンパイルで作成された `hoge.aux` があり，そこに文献参照以外の必要な相互参照の情報が保存されているからである．

次のステップは (4) 相互参照を処理する LaTeX コンパイル．

(4) 相互参照を処理する LaTeX コンパイル

`hoge.aux` の相互参照を解決し，`hoge.dvi` か `hoge.pdf` を出力する:

```
lualatex hoge.tex
```

補助 `aux` ファイルの中に前回保存したラベルがあるため，その番号を採用・出力していく．御存知の通り，出力が `dvi` ファイルになるか `pdf` ファイルになるかはマクロ体系として何を使うかに依存する．直接 `pdf` ファイルが出力される系統を用いているのであれば，ここで文書は完成である．

マクロ体系	出力	次の処理
LaTeX, (u)pLaTeX	`hoge.dvi`	(5) 文書の pdf 出力を得る
pdfLaTeX, XeLaTeX, LuaLaTeX	`hoge.pdf`	-

(5) 文書の pdf 出力を得る

すでに述べたが，`dvi` ファイルとは DeVice-Independent file のことで，文書の見た目のレイアウトを画像形式・表示デバイス・プリンタなどにまったく依存しない形で記録している[24]．これは人間が読むように設計されておらず，DVI ドライバと呼ばれる別のプログラムに入力することで，画像として表示したり，文書形式に変換したりできる．主な DVI ドライバとしては次のようなものがある:

- `dvipdfmx`: `dvi`→`pdf`．日本語文書作成においては最もよく使われている．そもそも海外では pdfLaTeX とその周辺が主流であり，`dvi` ファイルは生成されないのである．
- `dvips`: `dvi`→`ps` (PostScript)．この `ps` ファイルとは `pdf` ファイルの基礎となったものである[25]．
- `dviout`: (u)pTeX 対応の DVI プレビューア．最近では推奨されない．

無限ループ

文書作成の具体的な流れはここまでで説明したとおりである．基本的な相互参照にしろ，文献の参照にしろ，ラベルを適切に補助 aux ファイルに保存できれば，その後もう一回コンパイルすることですべての参照がきちんと対応して出力される．つまり\label，\ref があって，文献参照が必要ない（もしくは bbl ファイルに変更がない）場合には 2 回のコンパイルによって出力が得られることになる．これは基本的に間違ってはいないが，実際には 2 回では済まない場合がある．具体的には，参照したいラベル先が不明でとりあえず “**??**” と出力されていた部分に，ラベルがついている数字や文献を表す記号などが正しく代入されることで改ページの場所や図表の位置が変わってしまった場合，ラベルと番号の対応付けが変わってしまい，参照番号が正しく振られていないことになってしまうことがある*．そこで次のような収束しないおもちゃを作ることができる[26]:

```
\documentclass{article}
\pagenumbering{roman}
\setlength{\textwidth}{5em}
\setlength{\textheight}{2\baselineskip}
\begin{document}
  \setcounter{page}{999}
  \noindent The Page Number \pageref{foo} is\label{foo} shaking up and down.
\end{document}
```

この文書は，999 ページ始まりで，横 5 文字強，縦 2 行の本文スペースに文章を書き連ねている．\label は “is” の場所を示し，そのページ数を\pageref でローマ数字で表記している．実際にコンパイルすると，

i. はじめてのこんぱいるでは，補助 aux ファイルが存在しないため，\pageref で参照すべきものがなく，“**??**” が出力される．その後\label で “is” のページ番号 999 が補助ファイルに保存される．
ii. 参照が解決していないので，2 度目のコンパイルを行う．今度は先程作った補助ファイルがあるので，\pageref 部分で適切に “is” の位置（999）がローマ数字 cmxcix で表示される．その結果，“is” は次の 1000 ページ目に移動してしまい，\label でこの 1000 ページの情報が再び補助 aux ファイルに保存される．
iii. “**??**” はないが，参照先と参照元があってないので，3 度目のコンパイルを行う．今度は\pageref では先程保存した 1000 ページの情報をローマ数字 m で表示する．今度は “is” が 999 ページ目に戻ってしまい，補助ファイルには再び 999 ページの情報が保存される．
　というということで以下 ii. と iii. 繰り返し．

この tex ファイルをコンパイルした結果（のうち最初のページ，つまり 999 ページ）を Fig. 3.1 に示す．$n \in \mathbb{Z}_+$ 回コンパイルしたあとの出力において，ラベルが付けられる “is” のページ番号とそれを参照しているはずの \pageref{foo}の出力は必ず一致しない．このおもちゃに面白い以上の意味は見いだせないが，相互参照のために aux ファイルが必要で，同様に複数回のコンパイルが必要なこともわかるであろう．

*LaTeX Warning: Label(s) may have changed. Rerun to get cross-references right.

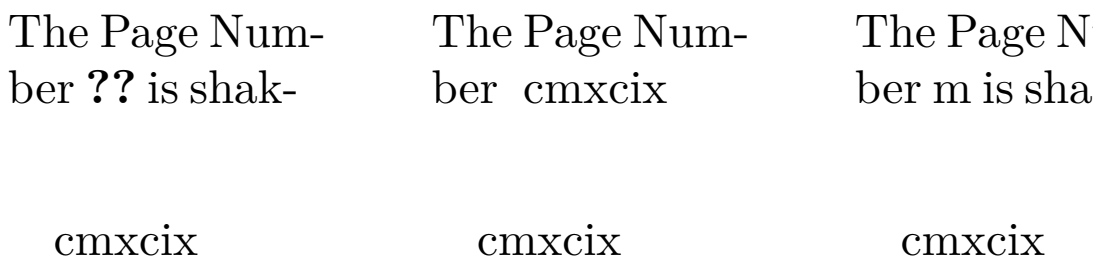


Fig. 3.1. The example of not converging cross-referencing. Left: former page (page 999) of first compilation results. Center: That of even numbered compilations. Right: that of odd numbered compilations.

3.2. 自動化

ここまででわかるように，文献を含む相互参照をきちんと機能させるためには，補助ファイル `hoge.aux` の存在も，それを使って複数回コンパイルすることも必要不可欠である．一方で，そのコンパイル回数はある程度目処がつくとは言え，正確に予測することは困難である．このような状況で，変更のたびに参照が失敗している箇所がないか確認しながら自分の手でコンパイルしていくのは大変面倒である．

この「複数回コンパイル等に起因する煩雑さ」を解決・軽減する方法として有名なのはもちろん

- `latexmk`: 現在最も有名な自動化ツール．複数回の実行に加え，BibTeX 等の関連ツールの実行も自動で行える．`latexmk` は `aux` ファイルの内容が変化したときにコンパイルし直すことになっている．例えば `aux` ファイルに時刻の情報を含むようなコードを書けば，コンパイルのたびに `aux` ファイルが変化することになる*.

であるが，自動化ツールも何種類か存在する．いくつかを下記に示すが「煩雑だな」と感じるのは同じでもそれを解消するための設計指針や思想は異なっているものも多い．

- `arara`: 手順を書いたファイルを事前に用意して，ビルドする．有名どころ（`pdflatex` や `biber` などなど）は事前に用意されている．
- `ptex2pdf`: `(u)platex` と `dvipdfmx` をまとめて実行する．
- `llmk`: Light Latex MaKe.

*latexmk では上限回数が設定されており，これにより無限の彼方に飛ばされることはない[27].

4. 結局なにをどうするのか

ここまでで，TEX があって，その拡張としていくつか種類があり，TEX たちを楽に使うための LATEX も何種類か存在することがわかった．`tex` ファイル中では文書クラスも適宜選ばなくてはならない．文献管理の方法も一つではなく，コンパイルを自動化してくれるツールも（一つシェア率が高いものがあるとは言え）他にも選択肢がある．もちろん，日本語を扱えないものや明らかに目的が違うものなどもあるため，その中から選択していく必要はあるが，それでも選択肢が何個もあるのは確かである．では，我々はなにを選択するべきなのか．

結論

LuaLATEX + upBIBTEX (or BibLATEX) + bxjsclasses + `latexmk`

これは結局カスタマイズ性を優先している方法である．その中で個人的には一番「特別なにも考えなくてもいいレシピ」である．LuaLATEX を使う理由としては

- Unicode に対応しているので，出力できない文字を気にしなくて良い．
- `dvi` ファイルという TEX 固有のよくわからないものを生成する必要もない．

などが挙げられるが，さらにもう少し TEX を自在に扱いたいと思ったとして

- OpenType 対応なので，フォントの設定が楽．
- 扱いが難しいとされる原始的な TEX 言語を扱わなくても Lua によって柔軟なカスタマイズが可能．

などもメリットであろう[28]．一方で，デメリットとして LuaLATEX は処理が遅いことが挙げられるが，以前は 20 倍程度遅かったところ，最近では 2 倍程度に抑えられており[29]，今後さらに速くなっていくと考えられる．

逆に (u)pLATEX を使わない理由としては，欧文と和文でフォントの扱いが違うためにせっかくの文書が美しくならなかったり，近いうちにまともに動かなくなるなんて話もあったり[30],[31]*するからである．

一方で upBIBTEX を用いるのは簡単に

- 学会であれば基本的に参考文献のフォーマットは決まっている（`bst` ファイルが配布される）．
- 自分で参考文献のところを必要以上に凝りたいと思うことが少ない．

ために，比較的処理の遅い BibLATEX を用いる必要性を感じないからである．しかし，せっかくなら細かいところまで気にしたい，のであれば BibLATEX を用いるのもよいだろう．

先に述べたコンパイルの煩雑さを回避するためにも自動化は必要不可欠であり，結局広く使われている `latexmk` をここではおすすめしている．というのも

*これは先述した開発中の LATEX 次期バージョン LATEX3 では pTEX に対応しない可能性があるからである．他の新しい手段があるのに，日本だけでガラパゴス化されたものに対応しないかもしれないというのも理解できる．しかしまあ，おそらく結局誰かがどうにかして実際に使えなくなることはないのだろうが，それでもそのようなリスクを抱えるものをわざわざ使う理由も特にない．

- 広く使われているからこそ，使い方などの情報が豊富に見つかる．
- Visual Studio Code などで公式にサポートされている*．
- 一度設定ファイル `latexmkrc` を作ってしまえば，なにも考えずにその後使いまわすこともできる．

からである．

一方で，もしコンパイルの速さを優先するなら，

upLaTeX + upBibTeX + `dvipdfmx` + jsclasses (+ `latexmk`)

なども考えられよう．この場合，いつでも使える `latexmk` で自動化せずに，必要な回数コンパイルするレシピを作って処理するのがより速いだろう．例えばただの授業の課題など，体裁にこだわる必要のない，たかだか数ページのものを作るのならコンパイルの速度を優先するのがいい．ただ，どうせコンパイル中も編集は行えることを加味すれば LuaLaTeX を利用するのがいいように思える．

5. ちょっとした Tips

ここまでで，基本的な TeX の実行について幾分イメージがつきやすくなったと思う．これより先は「へえ、そうだったのか」と個人的に思ったような TeX 周りの何事かを完全に備忘録として記していく．

5.1. LuaLaTeX でなんとやら

(1) フォント

本資料でもフォントの調整は行っている．このように途中で変えるのもお手の物である．

Table 5.1. Fonts used in this document.

Usage	Font Name
English main	TeX Gyre Termes
Japanese main	Hiragino Mincho ProN
English sans serif	Verdana
Japanese sans serif	Hiragino Kaku Gothic Pro
Typewriter	New Computer Modern Mono10
Footer	3270 Narrow
English in `section` and `subsection`	American Typewriter

*実は VScode 上でのエラーや警告の拾い方は latexmk だけ特別扱いされる[32]．

(2) Lua コード

LuaLaTeX の大きな特徴の一つはもちろん “Lua” である．Lua は，高速な動作と高い移植性，組み込みの容易さが特徴であるスクリプト言語である[33],[34]．派生の LuaJIT は動的型付けのスクリプト言語で最速言語である（らしい）．

そんな高速な Lua を TeX にからめて使っていく[35],[36],[37],[38]．もちろんプログラム言語なので普通に計算を行うことができて，わざわざ別のなにかを立ち上げることなく，計算して結果を出すことができる．例えば

```
$\sqrt{2}=$ \directlua{tex.sprint(math.sqrt(2))}
```

などと書けば

$$\sqrt{2} = 1.4142135623731$$

と表示される．

他にも引数を持つ自作関数を\newcommand でつくると，本文中で\sumSquare{}{}と書く度に，引数の小さい方から大きい方までの整数二条和を計算させられる．

```
\newcommand{\sumSquare}[2]{%
  \directlua{
    function sumSquareFunc(x,y)
      if x > y then
        x, y = y, x
      end
      local s = 0
      for i = x, y do
        s = s + i^2
      end
      tex.sprint(math.floor(s))
    end
  }%
  \directlua{ sumSquareFunc(#1,#2) }%
}
```

\sumSquare{1}{10} $= 385$

\sumSquare{-2}{1} $= 6$

\sumSquare{50}{-50} $= 85850$

もちろん文字列も扱える．もし間違えて猫を犬にしてしまったときは，以下の自作関数\displace を用いて，\displace{dog}{cat}{dog cat dog cat}と書けば，“dog cat dog cat” を “ cat cat cat cat” と猫だらけにできる．

```
\begin{luacode*}
  function myDisplace(str_before, str_after, sentence)
    local sentence_after = ""
    local l = #str_before - 1
    local i = 0
    while true do
      if string.sub(sentence, i, i+l) == str_before then
        sentence_after = sentence_after .. str_after
        i = i + l
      else
        sentence_after = sentence_after .. string.sub(sentence,i,i)
      end
      i = i + 1
      if i > #sentence then break end
    end
    tex.sprint(sentence_after)
  end
\end{luacode*}
\newcommand{\displace}[3]{ \directlua{myDisplace("#1","#2","#3")} }
```

LuaLaTeX は Lua を LaTeX と絡めて使えるため，文書の様式・組版を変えることもできる．

```
\documentclass[twocolumn]{article}
\usepackage{lipsum}
\usepackage{luacode}
\setlength{\columnsep}{1cm}
\pagestyle{empty}

\begin{luacode*}
vibrateLine = function (head, group, size)
  i = 1.5
  for list in node.traverse(head) do
    i = i + 0.13
    if list.id == node.id("hlist")
    then
      list.shift = list.shift - (1200000 * (math.sin (i)))
    end
  end
  return head
end
\end{luacode*}

\directlua{%
  luatexbase.add_to_callback('vpack_filter',vibrateLine,"strict")%
}

\begin{document}
  \lipsum[1-10]
\end{document}
```

Fig. 5.1. Example of Lua coding[36]

5.2. LaTeX で使える長さについて

(1) 単位

長さの単位としてはオーソドックスでなものが多い．ここで示したもの以外にも様々あるが，使いたい場面がない．

(2) 環境に依存する長さ

一方, 環境依存で定義された長さもいくつかある[39]. 特に図の挿入時には紙面の横幅に対する割合などを指定すれば，見た目の調整が楽である．定義から分かる通り，基本的な長さとしては`\paperwidth`>`\textwidth`≥`\linewidth`である．実際，これらの長さ（の半分）の線を引いてみる

_____ .5`\paperwidth`

_____ .5`\textwidth`

_____ .5`\linewidth` (in one column)

_____ .5`\linewidth` (in 2 cols)

_____ .5`\columnwidth`

Table 5.2. List of length units available in LaTeX.

unit	name	description
pt	point	1 pt $\simeq$ 0.35 mm
pc	pica	1 pc = 12 pt
mm	millimeter	1 mm $\simeq$ 2.85 pt
in	inch	1 in = 25.4 mm = 72.27 pt
em	/ɛm/	the width of an “M” in the current font
ex	/ɛks/	the height of an “x” in the current font
zw	zenkaku width	the width of a zenkaku-kanji in the current font
zh	zenkaku height	the height of a zenkaku-kanji in the current font

Table 5.3. List of common length macros.

macros	description
`\paperwidth`	width of the page
`\paperheight`	height of the page
`\textwidth`	width of the text on the page
`\textheight`	height of the text on the page
`\linewidth`	width of the line in the current environment
`\columnwidth`	width of the column
`\columnsep`	distance between columns
`\parindent`	indentation of paragraphs
`\parskip`	extra space between paragraphs
`\baselineskip`	vertical distance between lines in a paragraph

Table 5.4. List of space control macros.

macros	definition
`\␣`	0.33 em
`~`	0.33 em (non-breakable)
`\,`	3/18 em
`\:`	4/18 em
`\;`	5/18 em
`\quad`	1 em
`\qquad`	2 em
`\!`	−3/18 em
`\hspace{}`	flexible horizontal space
`\vspace{}`	vertical

(3) 空白

空白の制御を行うときにいつも忘れるのでメモしておく[40]．「カンマ < コロン < セミコロン」の順に縦に伸びていくので，これで覚えればいいか．

空白関連でもう一つ．TeX 処理の際には基本的にピリオドがあると終止符だと認識し，その後ろには大きめの空白が入る．しかし例外があり，大文字の後ろのピリオドは終止符認識しない（略語などに対応するため）．したがって，大文字の単語が文末にくると，終止符にも関わらず例外として処理されてしまい，次の文との間の空白が本来より小さくなってしまう．そこでピリオドの直前に大文字以外のなにかを挟めばよいため，例えば`\@`．などが用いられる．

6. おわりに

TEX と LATEX の違いから始まり，その他の TEX と名のつくものについて整理してきた．さらに tex ファイルから pdf ファイルを作るまでの処理の流れも概説した．確かに，文章を書いてるだけなのに（Microsoft Word と違って），よくエラーが出る．しかしそこで苦手意識を持って諦めてほしくない．少しでも TEX やらその周辺の仕組み・大枠を知ることで，よくわからん謎のエラーに対して，そのエラーがとりあえずどこらへんの絡まりが原因なのかを推測できるようになっているはずである．私の個人的な願いとしては，せっかく TEX を使うなら，満足できる美しさを目指してぜひとも色々カスタマイズして自分好みになるようにしていってほしい．

そんなこれからの楽しい TEX ライフの一助となれば幸いです．

References

[1] “Don Knuth’s home page.” URL: https://www-cs-faculty.stanford.edu/~knuth/ (visited on Oct. 26, 2022) (cit. on p. 3).

[2] “組版 - Wikipedia.” URL: https://ja.wikipedia.org/wiki/%E7%B5%84%E7%89%88 (visited on Oct. 25, 2022) (cit. on p. 3).

[3] Arthur Reutenauer. “A brief history of TEX, volume II.” In: *EuroBachoTEX 2007* (2008), p. 68 (cit. on p. 3).

[4] “TEX と LATEX の違い - ラング・ラグー.” URL: https://blog.wtsnjp.com/2016/12/19/tex-and-latex/ (visited on Oct. 26, 2022) (cit. on p. 3).

[5] “LATEX- TEX Wiki.” URL: https://texwiki.texjp.org/LaTeX#f2915fa5 (visited on Oct. 26, 2022) (cit. on p. 4).

[6] “TEX Wiki.” URL: https://texwiki.texjp.org/ (visited on Oct. 25, 2022) (cit. on p. 4).

[7] “Unicode - Wikipedia.” URL: https://ja.wikipedia.org/wiki/Unicode (visited on Oct. 26, 2022) (cit. on p. 4).

[8] 師茂樹. “Unicode とのつきあい方—漢字文化圏を中心に—.” In: コンピュータ & エデュケーション 27 (2009), pp. 12–17 (cit. on p. 4).

[9] “XƎTEX - Wikipedia.” URL: https://ja.wikipedia.org/wiki/XeTeX (visited on Oct. 26, 2022) (cit. on p. 5).

[10] “XƎTEX - TEX Wiki.” URL: https://texwiki.texjp.org/?XeTeX (visited on Oct. 27, 2022) (cit. on p. 5).

[11] “XƎLATEX で日本語する件について.” URL: http://zrbabbler.sp.land.to/xelatex.html (visited on Oct. 26, 2022) (cit. on p. 5).

[12] 八登崇之. “日本人の知らない TEX” 2010 (cit. on p. 5).

[13] 幸田露伴. “雲のいろいろ.” 反省雑誌, 1987 (cit. on p. 5).

[14] “ConTEXt - TEX Wiki.” URL: https://texwiki.texjp.org/?ConTeXt (visited on Oct. 26, 2022) (cit. on p. 6).

[15] “Biber (LaTeX) - Wikipedia.” URL: https://en.wikipedia.org/wiki/Biber_(LaTeX) (visited on Oct. 30, 2022) (cit. on p. 7).

[16] “TeX Live をホンキで語る - Acetaminophen’s diary.” URL: https://acetaminophen.hatenablog.com/entry/texadvent2016-20161205#main-7 (visited on Oct. 27, 2022) (cit. on p. 8).

[17] “標準的なクラスファイルの一覧 - TEXWiki.” URL: https://texwiki.texjp.org/?%E3%82%AF%E3%83%A9%E3%82%B9%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB%E4%B8%80%E8%A6%A7 (cit. on p. 8).

[18] 奥村晴彦. “LATEX 2ε 新ドキュメントクラス.” In: (2023). URL: https://mirror.math.princeton.edu/pub/CTAN/macros/jptex/latex/jsclasses/jsclasses.pdf (cit. on p. 9).

[19] “LaTeX の「アレなデフォルト」傾向と対策 - Qiita.” URL: https://qiita.com/zr_tex8r/items/297154ca924749e62471 (cit. on p. 9).

[20] “日本語組版処理の要件（日本語版）.” URL: https://www.w3.org/TR/2020/NOTE-jlreq-20200811/ (cit. on p. 9).

[21] “JIS X 4051 - Wikipedia.” URL: https://ja.wikipedia.org/wiki/JIS_X_4051 (cit. on p. 9).

[22] “LaTeX コンパイル等 - Yamamoto’s Laboratory.” URL: http://www.yamamo10.jp/yamamoto/comp/latex/run/run.php (visited on Oct. 27, 2022) (cit. on p. 9).

[23] “参考文献の書き方 - 星野光.” URL: https://www.eng.u-hyogo.ac.jp/faculty/hoshino/pc/latex/bibtex/ (visited on Oct. 27, 2022) (cit. on p. 10).

[24] “DVI - Wikipedia.” URL: https://ja.wikipedia.org/wiki/DVI_(%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB%E3%83%95%E3%82%A9%E3%83%BC%E3%83%9E%E3%83%83%E3%83%88) (visited on Oct. 27, 2022) (cit. on p. 11).

[25] “PostScript - Wikipedia.” URL: https://ja.wikipedia.org/wiki/PostScript (visited on Oct. 27, 2022) (cit. on p. 11).

[26] “LATEX のの相互参照はいつでも解決（収束）するのか？.” URL: https://qiita.com/kuroky_plus/items/28fc1f4e5610aa5c956e (visited on Oct. 27, 2022) (cit. on p. 12).

[27] “TeX の実行あれこれ - 雑記帳.” URL: https://blog.miz-ar.info/2016/12/running-tex/ (visited on Oct. 27, 2022) (cit. on p. 13).

[28] “「日本語 LaTeX」が多すぎる件について.” URL: https://www.slideshare.net/zr-tex8r/latex-239371115 (visited on Oct. 30, 2022) (cit. on p. 14).

[29] “LuaTeX が実はそんなに遅くないかも知れない件 - マクロツイーター.” URL: https://zrbabbler.hatenablog.com/entry/20170730/1501402087 (visited on Oct. 30, 2022) (cit. on p. 14).

[30] “pLaTeX が本格的にやばいかもという話 - Acetaminophen’s diary.” URL: https://acetaminophen.hatenablog.com/entry/2021/06/18/022108 (visited on Oct. 30, 2022) (cit. on p. 14).

[31] “【詳細】どうして pLaTeX は LaTeX3 でピンチになるのか - LuaLaTeX Lab.” URL: https://lualatexlab.blog.fc2.com/blog-entry-44.html (cit. on p. 14).

[32] “LaTeX Workshop で llmk してみたらアレだった件.” URL: https://zrbabbler.hatenablog.com/entry/2020/09/23/190840 (visited on Oct. 30, 2022) (cit. on p. 15).

[33] “Lua - Wikipedia.” URL: https://ja.wikipedia.org/wiki/Lua (cit. on p. 16).

[34] “高速スクリプト言語 [Lua] を始めよう.” URL: https://ie.u-ryukyu.ac.jp/~e085739/lua.hajime.html (cit. on p. 16).

[35] “徹底攻略！ LuaLaTeX で Lua コードを「書く」ためのコツ.” URL: https://qiita.com/zr_tex8r/items/af2905bc93ac2c936a67 (cit. on p. 16).

[36] “TeX で使うプログラミング言語まとめ.” URL: https://zenn.dev/k16/articles/e7690b036e534c#luatex%E3%81%AElua (cit. on pp. 16, 17).

[37] “Pattern syntax using luacode.” URL: https://tex.stackexchange.com/questions/499915/pattern-syntax-using-luacode (cit. on p. 16).

[38] “思わず Lua で LaTeX してみた.” URL: http://zrbabbler.sp.land.to/lualatexlua.html (cit. on p. 16).

[39] Kent McPherson. “Displaying page layout variables.” In: (2021). URL: https://ftp.jaist.ac.jp/pub/CTAN/macros/latex/required/tools/layout.pdf (cit. on p. 17).

[40] “spacing in LaTeX.” URL: https://www.sascha-frank.com/spacing.html (cit. on p. 18).

Appendices

A. はじめてのらてふかんきょうづくり

- Windows（要検証）
  i. TeX Live のページ（https://www.tug.org/texlive/acquire-netinstall.html）から install-tl-windows.exe をダウンロード．
  ii. ダウンロードしたファイルを実行．のち，待ち．
- Mac
  i. Homebrew をダウンロード．

```
/bin/bash -c "$(curl -fsSL
        https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

  ii. MacTeX をインストール．のち，待ち．

```
brew install --cask mactex-no-gui
sudo tlmgr update --self --all
sudo tlmgr paper a4
```

iii. ここで一旦確認．例えば main.tex を作って，コンパイル： latex main.tex dvipdfmx main.dvi

```
\documentclass{article}
\begin{document}
    Hello, \LaTeX!
\end{document}
```

iv. LuaLaTeX も確認．lualatex main_lua.tex

```
\documentclass[a4paper,lualatex,ja=standard]{bxjsarticle}
\begin{document}
  Hello, \LaTeX! こんにちは，\LaTeX!
\end{document}
```

v. upBibTeX も確認． main_lua_bib.tex と ref.bib

```
\documentclass[a4paper,lualatex,ja=standard]{bxjsarticle}
\bibliographystyle{unsrt}
\begin{document}
  Hello, \LaTeX! こんにちは，\LaTeX! \cite{ref_item}
  \bibliography{ref}
\end{document}
```

```
@article{ref_item,
  title={Title},
  author={Author},
  journal={Journal},
  year={2000},
}
```

を作って，コンパイル：

```
lualatex main_lua_bib.tex
upbibtex main_lua_bib.aux
lualatex main_lua_bib.tex
lualatex main_lua_bib.tex
```

vi. `latexmk` を使うための設定ファイル `latexmkrc` を作成．LuaLaTeX なら以下．
upLaTeX なら最終行の `pdf_mode` を 3 にする．

```
#!/usr/bin/env perl
$latex     = 'uplatex  %O -synctex=1 -file-line-error -halt-on-error %S';
$pdflatex  = 'pdflatex %O -synctex=1 -file-line-error -halt-on-error %S';
$lualatex  = 'lualatex %O -synctex=1 -file-line-error -halt-on-error %S';
$xelatex   = 'xelatex  %O -synctex=1 -file-line-error -halt-on-error %S';
$biber     = 'biber    %O --bblencoding=utf8 -u -U --output_safechars %B';
$bibtex    = 'upbibtex %O %B';
$dvipdf    = 'dvipdfmx %O -o %D %S';
$pdf_mode  = 4; # or 3 (for upLaTeX)
```

vii. もっかい確認．`latexmk main.tex` でコンパイル．`main.tex` には相互参照や文献参照を追加してもいい．`pdf_mode` には注意．

viii. Visual Studio Code をインストール．

ix. VSCode で拡張機能 LaTeX Workshop をインストール．

x. VSCode でコマンドパレットから `settings.json` を開き，以下を追加する：

```
"latex-workshop.latex.tools": [
  {
      "name": "latexmk_tool",
      "command": "latexmk",
      "args": [
          "-outdir=%OUTDIR%",
          "%DOC%"
      ],
  },
],
"latex-workshop.latex.recipes": [
  {
      "name": "latexmk",
      "tools": [
          "latexmk_tool"
      ]
  },
],
"latex-workshop.latex.option.maxPrintLine.enabled": false,
```

xi. 最後の確認．VSCode で `main.tex` を開き，コマンドパレットから `Build LaTeX project` を実行．

これでとりあえず upLaTeX でも LuaLaTeX でもその他のなにかでもローカル環境で動く！ はず（`latexmkrc` は文書に合わせて適宜変える必要あり）．ここから先はすべて個人の好みである．例えば

- VSCode の設定ファイル `settings.json` の `recipes` と `tools` は好きなように組合せて作れるため，
  - ログファイルを出力しないようにする（高速）
  - LuaLaTeX + `latexmk`
  - upLaTeX + `latexmk`
  - upLaTeX + upBibTeX + upLaTeX×2 + `dvipdfmx`

とかを用意しておき，適宜ショートカットで切り替えながら使う．
- `latexmk` でコンパイルが終わるたび，中間ファイルを全消去する．
- 出力を別フォルダにすることで，中間ファイルでフォルダがごちゃごちゃするのを抑制する．
- 保存するたびに自動でコンパイルさせる．
- 出力 PDF の表示を VSCode のプレビューで行う．もしくは外部アプリで行う．

などなど色々工夫ができる．

B. なぜ LaTeX なのか

これまでずっと LaTeX を使う人に向けて，その説明をしてきた．では，そもそも LaTeX を使う理由はあるのか．

よく LaTeX を使うと数式が綺麗と言われる．一昔前までは実際そうだったのであろう．しかし，実際「美しさ」ではもはやほとんど差がないことが多い．というのも，以前 X，もとい Twitter で「LaTeX で作った数式」「Word で作った数式」「Word で LaTeX に似るように作った数式」の 3 つを見分けるポストがあり，私はまんまと外した (名誉のために言っておくと，この文を書いているときに再挑戦したらきちんと正解した)．このポストの「これは LaTeX」のアンケートの結果も「Word で LaTeX に似るように作った数式」が半数以上の票で 1 位だった．つまり，似せようと思えば，ほとんどの人が気にならないくらいには Word でも綺麗に数式を作れるのである．

では，なぜ LaTeX を使うのか．

私は結局「美しさと一貫性」だと思っている．数式に限らず，文書全体の綺麗さが LaTeX にはある．例えば，2 段組みはきちんと半分ずつか．3 段組みは正確に 3 分割か．横並びの 2 つの図はきちんと対称に並んでいるか．横幅が違う 2 つの図を同じ縮小率で並べられるか．キャプションとの距離感は適切か．相互参照は正確に更新されているか．箇条書きのスタイルは全ページで同じか．フォントはどうか．スライドのスタイルは統一されているか．余白は同じか．前ページと同じブロックの位置はズレていないか．これらはすべて，Word でも気をつけていれば実現できることであろう．しかしたくさんの資料を作らなくてはならなかったり，長大な文を書く時にこれらをすべて気にすることはできるだろうか．こういった部分がデフォルトである程度綺麗な LaTeX を用いる大きな魅力だと感じる．

デフォルトで綺麗というのは，特に複数人での作業のときそのメリットが現れるだろう．一人で凝るのであれば，自分が気をつければいい．しかし，複数人で一つの文書を作成するとき，統一感を持って美しく成り立たせるためには相当な労力が必要であろう．さらに，予稿集のように各人が提出したものを一つにまとめる場合，確実に統一性を持たせるには一体どれだけのルールが必要だろうか．ページサイズ，余白，日英のフォントとサイズを本文・タイトル・著者名・見出しなどなどすべて指定し，同じだけ行間の指定も必要である．雛形を配布したとして，図表の位置や見出し・キャプションと本文との距離感はどうだろうか．これらの注意をすべての人が完璧に守れるだろうか．こういった複数人での作業において，LaTeX の真価が発揮される．デフォルトで十分綺麗がゆえ，もしくはスタイルファイルを配ってしまえば (詳しい人以外は) いじろうにもいじることはできないため，統一感を持たせることができる．統一感のおかげで読み手の注意を無駄に損なわないことは無視できない効果があるだろう．

もう一つ LaTeX を用いるメリットを挙げるなら，それは文書の構造を意識できることである．(頑張れば)

`\section` などを用いずにまったく同じ見た目の文書は作成できる．しかしそうではなく，`\section` などのコマンドや`\begin{itemize}`のような環境が用意されている，というのはユーザに「ここで節を分けるんだ」「ここで並列にものを並べるのがいい」と意識しながら書くことを促す．書き手が文書の構造をしっかり理解していることは，「文書の中身の一貫性」にもつながると思う．結局 LaTeX の良さは美しさと一貫性なのである．

文章が綺麗で一貫性があることは，シンプルに読みやすさにつながるだろう．それだけで他の人が読みたいと思う，かはわからないけど，少なくとも「見た目で読む気が失せる」ということはない．文書が美しいことは，人にものを読ませる最低ラインで，だから私は LaTeX を使うし，使ってほしい．

List of Figures

List of Tables