

L^AT_EX 2_ε による美文書作成入門

(写経版)

姫 伯邑考

2018 年 01 月 01 日

第 1 章

T_EX とその仲間

本章では T_EX とその仲間（pdfT_EX、X_YT_EX、LuaT_EX、pT_EX、upT_EX）との関係について紹介する。

1.1 T_EX とは？

T_EX は組版^{くみはん}ソフトである。組版（typesetting）とは印刷用語で、活字を組んで版^{はん}（印刷用の板）を作ることを意味する。T_EX はコンピュータでテキストと図版をうまく配置して、版にあたるもの（PDF あるいは PostScript ファイル）を出力する（タイプセットする）ためのソフトウェアである。

T_EX には次のような特徴がある。

- T_EX はオープンソースソフトウェアなので無料で入手することができ、自由に中身を調べたり改良することができる。商用利用も自由に行うことができる。
- T_EX は Windows でも、Mac や Linux などの UNIX 系 OS でも全く同じ動作をする。すなわち、入力と同じであれば、原理的には全く同じ出力が得られる。
- T_EX への入力はテキスト形式なので、普通のテキストエディタで読み書きすることができ、再利用・データベース化が容易である。
- 自動ハイフネーション、ペアカーニング（AV や To など相補的な形の文字を食い込ませる処理）、リガチャ（fi、fl、ffi、m などのような合字^{ごうじ}（対応フォントのみ））処理、独立行（ウィドウまたはオーファン）処理（段落の最初の行だけ、あるいは最後の行だけが別ページになることを抑制する処理）など、高度な組版技術が組み込まれている。
- 特に数式の組版については定評があり、数式をテキスト形式で表す際の事実上の標準となっている。

1.2 T_EX の読み方・書き方

T_EX の作者 Knuth 教授（Donald E. Knuth：1938 ～：計算機科学の分野で最も偉大な学者の 1 人）によれば、T_EX はギリシア語から命名したもので、最後の X は口の奥で発声する無音の「ハ」に近い音だそうだが、英語でこれに一番近い音は /k/ なので「テック」と読む人が多いようである。ドイツ語では「テッヒ」という発音が広く用いられている。

日本では、特に大学関係者の間では昔から「テフ」と呼び慣わされているが、英語圏で T_EX を覚えた人や出版関係者の間では「テック」という発音が広く用いられている。

T_EX は見ての通り E を少し下げて字間を詰めて表記する。このような文字の上げ下げや詰めは T_EX の得意とするところだが、これができない場合には TeX と表記することになっている（TEX や Tex とは表記しない）。

1.3 L^AT_EX とは？

L^AT_EX は DEC（現 HP）のコンピュータ科学者 Leslie Lamport^{*1}によって機能拡張された T_EX である。元々の T_EX と同様にオープンソースソフトとして配布されている。

L^AT_EX は日本では「ラテック」あるいは「ラテフ」と読まれる。英語圏では「レイテック」と発音されることが多いようである（アクセントの位置を圏点付きの太字で示してある）。

*1 Lamport はその後 2001 年に Microsoft Research に移籍。2013 年にはチューリング賞を受賞している。

最初の \LaTeX は 1980 年代に開発されたが、1993 年には $\text{\LaTeX} 2_{\epsilon}$ という新しい \LaTeX が開発され、今日では \LaTeX と言えばほぼ確実に $\text{\LaTeX} 2_{\epsilon}$ を指すようになっている（古い \LaTeX は $\text{\LaTeX} 2.09$ と呼ばれる）。本稿でも以下では $\text{\LaTeX} 2_{\epsilon}$ のことを単に \LaTeX と表記することにする。

\LaTeX は見ての通り A を小さく上付きにして表記する。文字の上げ下げができない場合は LaTeX と表記することになっている。同様に $\text{\LaTeX} 2_{\epsilon}$ と表記できない場合には $\text{LaTeX} 2_{\epsilon}$ と表記する。

\LaTeX の特徴は、文書の論理的な構造と視覚的なレイアウトを分けて考えることができることである。

例えば「はじめに」という ^{セクション} 節の見出しがあれば、文書ファイルに、

```
\section{はじめに}
```

のように記述しておく。この `\section{...}` という命令が、紙面上のデザイン、例えば「14 ポイントのゴシック体で左寄せ、前後の空きはそれぞれ何ミリを基準とし、何ミリ以内なら伸ばしてもよい…」というレイアウトに対応するといったことは、様式・判型ごとに別ファイル（クラスファイル・スタイルファイル）に登録されている。標準のクラスファイルのデザインが気に入らなければ、自由に変更することができる。クラスファイルだけ変更すれば、同じ文書ファイルでも違ったレイアウトで出力することができる。

仮に、文書ファイルに「ここは 14 ポイントのゴシック体で 3 行どり中央に…」などと書き込んでしまっても、後で組み方を変更しようとする際に原稿全体に手を入れなければならない。下手をすると、節ごとに見出しの体裁が違ってしまうことにもなりかねない。文書の再利用も難しくなる。

更に、 \LaTeX は章・節・図・表・数式などの番号を自動的に付けてくれるし、参考箇所には番号やページを自動挿入することができる。目次・索引・引用文献の処理まで自動的に行われる。また、^{はしり} 柱 も自動的に作成される。

このような便利な機能を備えるため、 \LaTeX 利用者が飛躍的に増え、 \TeX を使っているといっても実際には \LaTeX であることが多くなっている。

\LaTeX は \TeX のプログラミング機能（マクロ機能）を用いて作られたものである。一方、 \TeX 本体（マクロと区別するためにエンジンと呼ぶことがある）も改良され、特に日本では日本語の扱いに優れた pTeX というエンジンが広く用いられている。 pTeX 用にマクロを修正した \LaTeX が pLaTeX ($\text{pLaTeX} 2_{\epsilon}$) である。

\LaTeX は理系の論文や本の製作に広く用いられている。多くの論文誌や ^{アーカイブ} arXiv (<http://arxiv.org>) のようなプレプリントサーバは \LaTeX での論文投稿を推奨しており、理系の出版社は多くの本を \LaTeX で製作している。一例を挙げれば『岩波数学辞典』の最新版（第 4 版）は全て \LaTeX ($\text{pLaTeX} 2_{\epsilon}$) で製作されている。

Wikipedia も数式は \LaTeX 形式で記述することができる（実際には Wikipedia サーバ上の \LaTeX で数式部分を組版して画像に変換している）。

1.4 \TeX の処理方式

一般のワープロソフトと異なり、 \TeX は高度な最適化を行うため、段落の最後に 1 文字追加するだけで段落の最初の改行位置が変わることもあり得る。このような処理を、キーボードから 1 文字入力するごとに行うのはかなりのマシンパワーを必要とする。そのため、 \TeX ではキーを打つたびに画面上の印刷結果のイメージを更新する方式^{*2}ではなく、一括して全体を処理するバッチ処理方式を採用している。

伝統的な作業手順では、原稿は自分の使い慣れたソフト（テキストエディタ等）で記述し、テキストファイルとして保存しておく。これを後で \LaTeX で一括処理する。

^{*2} 画面表示と印刷イメージが同じ（What You See is What You Get）という意味で ^{ワイジワイグ} WYSIWYG 方式と呼ぶことがある。

例えば、 $\text{\LaTeX} 2_{\epsilon}$ では、

```
\documentclass{jsarticle}
\begin{document}
これはサンプルの文書である。
テキストファイル中では、
どこで改行しても構わない。
印刷結果の改行位置は自動的に決定される。

段落の切れ目には空の行を入れておく。
\end{document}
```

のようなテキストファイルを入力すると、

これはサンプルの文書である。テキストファイル中では、どこで改行しても構わない。印刷結果の改行位置は自動的に決定される。

段落の切れ目には空の行を入れておく。

のように出力される（ \backslash で始まる行は \LaTeX の命令である）。

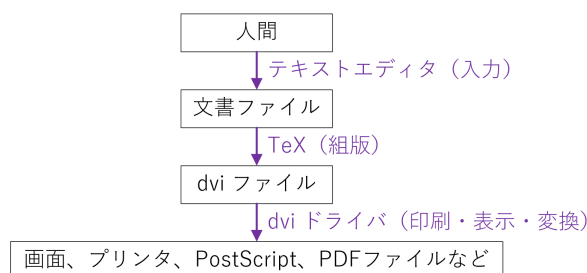
テキストファイルを用いることの利点は多数存在する。

- 文書入力 自分の使い慣れているソフトで行う方が楽である。どんな文書入力ソフトでもテキストファイルで保存することができるので、 \TeX は入力ソフトを選ばない。
- テキストファイルはコンピュータの機種に依存しない。どんなコンピュータや携帯端末でも、テキストファイルなら安全にやり取りすることができる。
- テキストファイルの変換は容易である。そのため、データベース出力や Web フォーム入力、XML 文書などから \TeX に変換して組版するといったことがよく行われている。
- テキストファイルで文書を用意する方が、コンピュータのパワーユーザの心理に適っているかもしれない。数式を考えなければ Adobe InDesign のようなリアルタイムで \TeX 同様の処理を行うレイアウトソフトが存在するし、数式を考えても WYSIWYG な数式エディタが存在する。しかし、これらは今のところ \TeX に置き換わりそうにない。

1.5 \TeX の出力

元々の \TeX は、組版結果を ^{ディーヴィーアイ} $\text{\text{dvi}}$ ファイルという中間ファイルに書き出す。 $\text{\text{dvi}}$ は device independent（装置に依存しない）という英語の略である。この $\text{\text{dvi}}$ ファイルを読み込んでパソコンの画面、各種プリンタ・写植機、及び PostScript や PDF などのファイルとして出力するための専用ソフト（ $\text{\text{dvi}}$ ドライバ・ $\text{\text{dvi}}$ ウェア）が、出力装置や出力ファイル形式ごとに用意されている。特に、画面出力用の $\text{\text{dvi}}$ ドライバのことを $\text{\text{dvi}}$ ビューアという。

この処理の流れを図示すると、次のようになる。



ところが欧米語圏では、後述する $\text{\text{pdf}\TeX}$ や $\text{\text{X}\TeX}$ 、 $\text{\text{Lua}\TeX}$ の登場によって、 \TeX から直接 PDF ファイルを出力することができるようになり、画面表示や印刷も全て PDF を通じて行うのが普通となっている。

日本語圏で広く用いられている $\text{\text{p}\TeX}$ や $\text{\text{up}\TeX}$ については PDF を直接出力することはできないが、 $\text{\text{dvipdfmx}}$ という高速な $\text{\text{dvi}}$ ドライバと組み合わせて PDF を作成することができる。

1.6 T_EX と日本語

日本語の文字数は非常に多く、多数のフォントに分割すれば T_EX でも扱えないこともないが、非常に厄介である。

本格的に T_EX を日本語化する試みはいくつか存在したが、今日広く使われているのは、かつての株式会社アスキーが開発した^{ビーテック}pT_EX、及びそれを Unicode 対応にした upT_EX (田中琢爾氏作) である。

日本語であれば文字は全て同じ幅なので単純に組んでいけばよいかというと、そうはいかない。次のような処理が必要となる。

- 句読点、終わり括弧（閉じ括弧）類、中黒（・）、繰返し（々）、感嘆符（！）、疑問符（？）が行頭に来ないようにする必要がある（行頭禁則処理）。促音文字（っ）、拗音文字（ゃゅょ）、長音記号（ー、音引き）などもなるべく行頭に來てほしくない。
- 同様に、始め括弧（開き括弧）類が行末に来ないようにする必要がある（行末禁則処理）。
- 括弧類、句読点が行頭、行末に来たときや、括弧類、句読点が連なったとき、空白が空きすぎて見えるので、詰める必要がある。
- 段落の最後の 1 文字と句読点になるのは、なるべく避けたいところである（文字ウィドウ処理）。

これらの条件を満足するためには、文字の間隔を微調整しなければならない（詰め処理、伸ばし処理）。しかし、調整すぎると文字の間隔が揃わず、かえって見苦しくなってしまう。例えば、拗促音文字（ゃゅょっ）はなるべく行頭に來ないほうがよいし、文字ウィドウもなるべく避けたいところだが、あまり厳密にこれらのルールを当てはめるとかえって不自然になることがある。そこで pT_EX では、どの文字が行頭に來ると何点減点、行末に來ると何点減点、文字ウィドウは何点減点、文字の間隔がどれだけ伸びると何点減点といった具合に点数を計算し、減点の合計が最小になるように組む。点数の配分は自由に調整することができる。

pT_EX の日本語組版が非常に優れているため広く使われるようになった一方、海外で普及している X_YT_EX や LuaT_EX を日本語で扱うための仕組みも次第に改良されてきている。

1.7 T_EX のライセンス

T_EX 関連のソフトウェアは全てオープンソースのライセンスで配布されており、商用利用も含めて自由に使うことができる（詳しくは各ソフトウェアのオンラインドキュメントを参照）。ターミナルに「`texdoc` [ソフトウェア名](#)」と打ち込めばオンラインドキュメントが表示される。

オリジナルの T_EX は、付加価値を付けたものを有償で販売することも自由である。但し、T_EX と完全な互換性を持たないものは T_EX と名乗ることを禁止されている。米国での T_EX の商標は American Mathematical Society (米国数学会) が登録しているが、これは無関係な人物に商標登録されることを防ぐためであり、T_EX を使う際に「T_EX は…の商標です」などと断る必要はない。

L^AT_EX は LPPL (L^AT_EX Project Public License, <https://www.latex-project.org/lppl/>) に従い、ファイル名さえ変えなければ改変したものの再配布も自由である。

pL^AT_EX 等については、かつての株式会社アスキーが開発したものだが、日本語 T_EX 開発コミュニティ (<https://texjp.org>) による「コミュニティ版」に移行している。これらは (修正) BSD ライセンスに従っており、オリジナルの著作権表示などを残す限り改変・再配布は自由である。

1.8 T_EX のディストリビューション

T_EX は Pascal をベースとした Knuth 教授の WEB^{*3} という「文芸的プログラミング」ツールで作成されているが、UNIX 上では通常は C 言語に変換してからコンパイルされる。これが現在の多くの T_EX の実装の起源である Web2c の由来である。

^{*3} World Wide Web の Web とは無関係である。

この Web2c をベースに Thomas Esser が集大成した teTeX という TeX ディストリビューション^{*4}が広く使われるようになり、日本では村上展之氏がこれに基づく ptetex を配布していた。

一方、Windows では角藤亮氏の W32TeX というディストリビューションが広く使われるようになった。

その後、 TeX Live という超巨大な集大成が広く使われるようになり、土村氏もこれに基づく ptexlive を開発した。これが現在では全て TeX Live 本体に取り込まれている。

1.9 これからの TeX

最初の TeX が開発されたのは 1978 年である。これだけ長い間安定して使われているソフトウェアは他に例を見ない。 TeX は組版の歴史における 1 つの不動点^{フィックスポイント}と言えるだろう。

しかし、既存の TeX に満足しては進歩がない。今日に至るまで、 TeX 本体（エンジン）及びマクロに、いろいろな拡張が行われている。エンジンの拡張としては次のようなものが挙げられる。

- $\epsilon\text{-TeX}$ ($\epsilon\text{-TeX}$) は、 TeX の種々のレジスタ（変数）の個数を拡張（256 個 → 32768 個）した他、右から左に組む機能などを追加したものである。現在では pTeX をはじめ殆どのシステムが $\epsilon\text{-TeX}$ 拡張を含んでいる。 LaTeX パッケージにも $\epsilon\text{-TeX}$ 拡張を仮定したものが増えている。
- TeX を日本語化した pTeX も、北川弘典氏により $\epsilon\text{-TeX}$ 拡張された ($\epsilon\text{-pTeX}$: $\epsilon\text{-pTeX}$)。
- 田中琢爾氏の upTeX (upTeX : 読み方はユーピー TeX またはユプ TeX) は、($\epsilon\text{-}$) pTeX の内部を Unicode 化したものである。
- Hàn Thế Thành 作の pdfTeX (pdfTeX) は、 TeX の出力形式を dvi ではなく PDF にし、 $\epsilon\text{-TeX}$ と同様の拡張を加えて、 microtypography という高度な組版アルゴリズムを組み込んだものである。現在では、オリジナルの TeX を置き換えて広く用いられている。
- Jonathan Kew 作の XeTeX (XeTeX) は元々 Mac の OS X 上で動作し、OS X 上の OpenType フォントをそのまま利用することができ、 PDF を出力する TeX だが、Windows や Linux にも移植されている。
- LuaTeX は pdfTeX に軽量スクリプト言語 Lua を組み込んだものである。Unicode 対応で、 XeTeX とは別な（OS に依存しない）方法で OpenType フォントに対応している。 pdfTeX の後継として今最も注目されているものである。 LuaTeX-jp プロジェクトの成果と組み合わせれば pTeX に置き換えて使うことができる（但し、実行速度がやや遅いのが難点である）。2005 年から開発が始まり、2016 年 9 月末にバージョン 1.0 がリリースされた。ただ、開発者たちは LaTeX ではなく後述の ConTeXt を主に使っているため、 LuaTeX の仕様が違って LaTeX で問題が生じるといったことが過去に多々あった。
- Clerk Ma による pTeX-ng は、Web2c ベースではなく C 言語で開発された Y&Y TeX から出発して pTeX 、 upTeX の機能を取り込んだもので PDF を直接出力するが、まだまだ開発中のものである。

一方、 TeX のマクロも $\text{LaTeX 2}_{\epsilon}$ ができてもう 20 年経過している。現在では $\text{LaTeX 2}_{\epsilon}$ 後継の LaTeX 3 の開発が（気長に）行われているところである。また、 LaTeX と全く異なるマクロパッケージ ConTeXt ^{コンテキスト}（作者は Hans Hagen）も特にヨーロッパでユーザ層を広げている。最新の ConTeXt は LuaTeX 上で動作する。

Web 上で LaTeX とほぼ互換の数式組版機能を Javascript、CSS、Web フォントで実現したものとして MathJax というものが存在する。Web ベースの帳票印刷システムでも、サーバ側で TeX 経由で PDF を作成し、クライアント側で印刷するといった用途で TeX が用いられることがある。

最近では、 Markdown という非常に簡単な形式で文書を記述し、必要に応じて LaTeX 、 HTML などに変換しようという流れがある。多様な形式間を相互変換する pandoc などのツールも開発されている。

高度な組版から PDF 作成まででき、Adobe-Japan1-6 の 2 万字（更には IPAmj フォントの 6 万字）を使いこなせるフリーソフトウェアは、今のところ TeX しか存在しない。他のソフトウェアの組版エンジンとしても、色々使いこなせると思われる。

^{*4} ディストリビューションとは、配布用にパッケージされたソフトウェア群のことである。

第 2 章

使ってみよう

本章の第 1 節では、 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ のインストールが不要な Web で $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ を利用する方法を説明する。第 2 節以降では、インストールと設定が完了しているパソコンで $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ を使う方法を説明する。

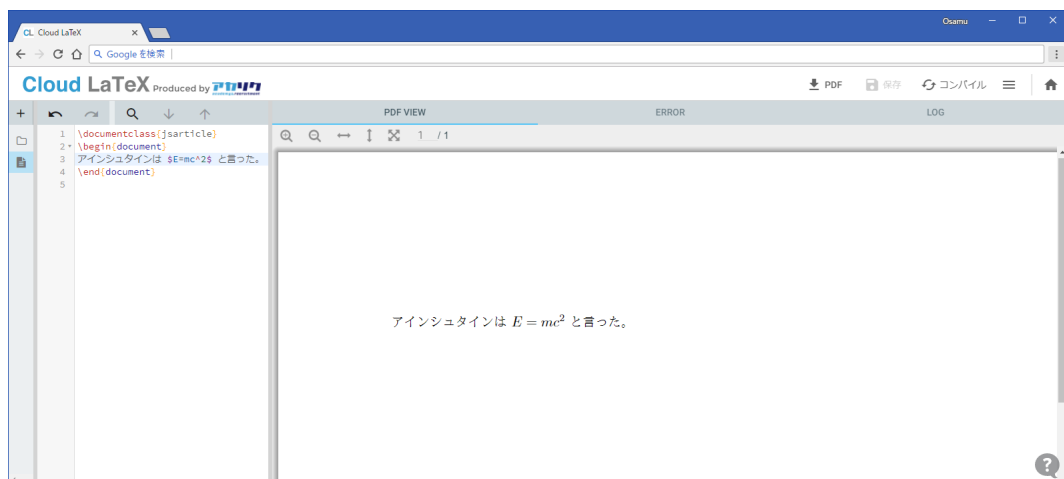
2.1 Web で $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ を使う

今日では、パソコンに $\text{T}_{\text{E}}\text{X}$ をインストールしなくても Web ブラウザ経由で $\text{T}_{\text{E}}\text{X}$ を利用することができる。Cloud $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ (<https://cloudlatex.io>) という本格的なサイトが存在するので、このサイトの利用を推奨する。これ以外に Share $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ (<https://ja.sharelatex.com>) や Overleaf (<https://www.overleaf.com> : 旧 write $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$) でも $\text{X}_{\text{E}}\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ や $\text{LuaL}^{\text{A}}\text{T}_{\text{E}}\text{X}$ により日本語の利用、IPAex フォントの埋め込みが可能である。

最初に Cloud $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ に新規登録（登録が完了していればログイン）する。

まずは新しいプロジェクトを作成する。例えば「Test」という名前のプロジェクトを作成したとする。プロジェクトに入ると「main.tex」というファイルがあるので、これに例えば次のように入力する。

```
\documentclass{jsarticle}
\begin{document}
アインシュタインは  $E=mc^2$  と言った。
\end{document}
```



自動コンパイルが有効になっていれば、PDF が作成され、右側にプレビューが表示される。正しくコンパイルすることができたら PDF をダウンロードする。正しくコンパイルすることができない場合には、右上の設定 (≡) をクリックして、 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ エンジンを「 $\text{p}^{\text{L}}\text{a}^{\text{T}}\text{E}_{\text{X}}$ 」に設定して「自動コンパイルを有効にする」をオフにしておき、必要に応じて「コンパイル」ボタンを押す方が軽快に作業することができる。

2.2 コマンドで行う方法

Windows のコマンドプロンプトや PowerShell、Windows 上のオープンソースソフト Cygwin のシェル、Mac や Linux 等のターミナル上でコマンドを打ち込んで $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ を利用する方法について説明する。

2.2.1 ターミナル（コマンドプロンプト・PowerShell）の起動

Windows のコマンドプロンプトや PowerShell は、Cortana や検索ボックスに「コマンド（または cmd）」「PowerShell」と打ち込んで検索して実行することができる。Cygwin をインストールした Windows の場合は mintty 等のターミナルを起動する。Mac の「ターミナル」は「アプリケーション」→「ユーティリティ」の中に存在する。以下では、これらをまとめて「ターミナル」と呼ぶことにする。

ターミナルを起動したら、まずは現在位置（カレントディレクトリ）に注意する。必要に応じて `cd` コマンドでディレクトリを移動する。

2.2.2 エディタの起動

L^AT_EX の文書ファイルを作成するには、テキストファイルを作成・編集するための「テキストエディタ」（単に「エディタ」とも言う）を用いる。

テキストエディタには多数の銘柄が存在する。T_EXworks や T_EXShop の編集用のウィンドウもテキストエディタであり、Windows の「メモ帳 (notepad)」や Mac の「テキストエディタ」は典型的なテキストエディタである。Word などのワープロソフトもテキストエディタとして使うことができる（保存の際に「テキストファイル」形式を選択する）。

T_EX の文書ファイル名は、例えば `ex1.tex` のように拡張子（ファイル名の末尾）を `tex` にするのが一般的である。以下では、`ex1.tex` という文書ファイルを作成し、それを pL^AT_EX で処理してみることにする。

Windows の場合、ターミナルから

```
notepad ex1.tex
```

のように打ち込むと「メモ帳」が起動する。「ファイル名 `ex1.tex` が見つかりません。新しく作成しますか?」と聞いてきたら [はい] と答える。

エディタを起動したら、最初は実験なので日本語は使わず、エディタの中に全て半角文字で次のように打ち込んでみる。これは「Hello, T_EX!」という文と「 $\int dx = x + C.$ 」という数式を出力するための入力である。

```
\documentclass{article}
\begin{document}
Hello,\TeX{}!
\[ \int dx = x + C. \]
\end{document}
```

打ち込み終わったら、エディタの起動時にファイル名を指定していない場合は `ex1.tex` というファイル名で保存する。保存するフォルダ（ディレクトリ）は必ず先程のターミナルの現在位置と同じにしておかなければならない。保存する際の文字コード（エンコーディング）は、従来の Windows ではシフト JIS が主流であったが、これからの時代は UTF-8 を推奨する。pL^AT_EX はシフト JIS、JIS (ISO-2022-JP)、EUC-JP、UTF-8 に対応している。

2.2.3 コマンドで L^AT_EX を起動する

`ex1.tex` を pL^AT_EX で処理するには、ターミナルに

```
platex ex1.tex
```

と打ち込む。但し、デフォルトの文字コードはシステムによって異なる（自動判断するものも存在する）。文字コードを例えば UTF-8 に指定して処理するには

```
platex -kanji=utf8 ex1.tex
```

と打ち込む。utf8 の他に euc、jis、sjis を指定することができる。

ここで行ったことは、文書ファイル `ex1.tex` を `dvi` ファイル `ex1.dvi` に変換する作業^{*1} である。

なお、画面に “No file `ex1.aux`.” というメッセージが出る場合があるが、これはエラーメッセージではないので無視して構わない。拡張子が `aux` のファイルは相互参照に使うもので、第 10 章で詳しく説明する。同じファイルを再度 `pLATEX` で処理すると、このメッセージは出なくなる。エラーメッセージが出て止まってしまった場合は、後述の「エラーが発生した場合」を参照すること。

正常に終了したら、文書ファイル `ex1.tex` が格納されているフォルダに次の 3 つの新しいファイルが作成されているはずである。

ex1.aux : `aux` ファイルもしくは補助 (auxiliary) ファイルと呼ばれるもの。テキストファイルなので「メモ帳」などのテキストエディタで読むことができる。`LATEX` の「相互参照」という機能を使わない場合は何の意味も持たない。この場合、直ちに削除して構わない。

ex1.dvi : これが肝心の組版結果 (`dvi` ファイル) である。このファイルに、どの文字をどのページのどの位置に配置するかという情報が書き込まれている。バイナリファイルなので通常のテキストエディタでは読むことはできない。

ex1.log : ログファイルと呼ばれるものである。実行の途中で画面に表示されるメッセージや、実行状態についての情報がここに書き込まれる。ログ (`log`) とは、航海日誌または一般に業務日誌を意味する英語である。テキストファイルなのでテキストエディタで読むことができる。エラーが生じた際に、その原因を調べるために用いるが、ここでは削除してしまって構わない。

2.2.4 PDF に変換する

`dvi` ファイルを `PDF` ファイルに変換するには `dvipdfmx` というコマンドを用いる。例えば、`ex1.dvi` を `ex1.pdf` に変換するには、ターミナルに次のように打ち込む^{*2}。

```
dvipdfmx ex1
```

こうして生成された `PDF` ファイルは `Adobe Reader` でもプレビューすることができるが、`Adobe Reader` は重く、しかも開いている `PDF` をロックするので、`PDF` を生成する前に閉じなければならず効率が悪い。`TEXShop` や `TEXworks` のような統合環境の `PDF` ビューアを使うか、あるいは単体の軽い `PDF` ビューアとして、`Windows` では `SumatraPDF`、`Mac` では `OS X` 付属の「プレビュー」などが使える。これらは `PDF` ファイルをロックせず、`PDF` ファイルが更新されると自動的に再読み込みを行ってくれる。

2.3 日本語のテスト

今度は日本語を入力して試してみる。エディタで先程の文書ファイル `ex1.tex` を次のように適当に日本語を含めた形に書き直し、上書き保存する。

```
\documentclass{jsarticle}
\begin{document}
こんにちは、\TeX{}!
\[ \int dx = x + \text{定数} . \]
\end{document}
```

最初の行の `article` が `jsarticle` になる。`js` を付けても付けなくても欧文・和文とも出力することができるが、標準の行間などが若干変化する。これを先程と同様に組版し、日本語部分も正しく表示できているかを確認する。もし、

```
! LaTeX Error: This file needs format 'pLaTeX2e'
but this is 'LaTeX2e'.
```

^{*1} この作業を「タイプセットする」あるいは「コンパイルする」という。コンパイル (compile) とは、コンピュータのソースコード (テキストファイル) をオブジェクトコード (バイナリファイル) に変換する作業を指す言葉である。これらを `TEX` の処理に当てはめて「`TEX` ソース」を `dvi` ファイルあるいは `PDF` ファイルに変換する作業を「コンパイルする」という。

^{*2} ここでは `platex` と `dvipdfmx` を順に実行したが、この 2 つをまとめて実行するコマンド `ptex2pdf` が存在する。`ptex2pdf` は「`ptex2pdf -l ex1`」のように用いる。

などと言って止まってしまうなら、 \TeX の設定が間違っていて英語版 \LaTeX (\pdf\LaTeX など) を呼び出してしまっている可能性がある。

\pLaTeX や \dvipdfmx の処理が正常に終わっても、PDF ビューアに日本語部分が表示されない場合は、日本語フォントを埋め込む設定がうまくできていないことが考えられる。設定を見直しても改善しないようなら、PDF ビューアが使用している環境に合わないことも考えられるので、別の PDF ビューアを試してみるとよい。

2.4 SyncTeX の使い方

SyncTeX はエディタ (\TeX ソース編集画面) と PDF プレビュー画面との相互リンクのための仕組みである。 \TeXworks や \TeXShop のような統合環境の他、PDF ビューアでは SumatraPDF、エディタでは Emacs などが対応している。

例えば、 \TeXworks ではエディタの画面を $\text{Ctrl} + \text{左クリック}$ すると PDF の該当箇所にジャンプし、逆に PDF の画面を $\text{Ctrl} + \text{左クリック}$ するとソースの該当箇所にジャンプする。

\TeX 側でも SyncTeX に対応している必要がある。 \pdfTeX や \W32TeX の \pTeX では以前から対応していたが、現在は Windows 版以外の \pTeX でも対応している。これらの \TeX を起動するコマンドに $-\text{synctex}=1$ オプションを付ければ SyncTeX が有効になる。

SumatraPDF のサイトには、SumatraPDF と Emacs (\AUCTeX) などのエディタを組み合わせる SyncTeX を動作させる方法が詳しく記載されている。

SyncTeX の仕組みは、 \TeX 側が入力・出力の位置の対応関係を「ファイル名.synctex.gz」という圧縮ファイルに書き出すだけである。PDF ファイルに余計な情報が挿入されることはない。

2.5 TeX2img の紹介

寺田侑祐氏 (Windows 版は安倍紀行氏) 作の \TeX2img は、 \LaTeX 出力を画像に変換するためのツールである。PDF、EPS、PNG、SVG、EMF など多様な画像形式をサポートしている。 \LaTeX で出力した数式を別のソフト (DTP ソフト、ワープロソフト、プレゼンソフト等) や Web ページに貼り付けたりするために便利である。Mac 版、Windows 版が存在する。使い方は、 \LaTeX ソースを貼り付けて「画像生成」ボタンを押すだけである。

詳しくは、 \TeX2img 配布サイト^{*3}を参照のこと。

2.6 エラーが発生した場合

本稿に登場する数行の例でも、あるいはもっと長い雛形でも、とにかく確実にタイプセットできる例を出発点として、1 つずつ新しい技を覚えていくこと。始めのうちは、1 つ命令を追加したら直ちにタイプセットしてみることを推奨する。そうすれば、どこでエラーが発生したのか自ずと分かるだろう。

\backslash で始まる命令の綴りを間違えた場合、エラーメッセージが現れ、入力待ちの状態となる。

メッセージ “! Undefined control sequence.” は、未定義 (undefined) の命令 (control sequence) が使われているという意味である。エラーが発生した際、次の処理を選ぶことができる。

- そのまま Enter を押せば、エラーを無視して処理を続行する。うまく行けばエラー箇所以外の処理ができるかもしれない。
- x (エックス) を入力して Enter を押せば \LaTeX による処理を中断する。 \TeXworks では \dvipdfmx による処理に移行するので、うまく行けばエラーの少し前までの PDF 出力が得られるかもしれない。x は \exit (終了) を意味する (間違えて quit のつもりで q を打つと、エラーを表示せずに続行する quiet モードになってしまう)。

^{*3} <https://tex2img.tech/>

2.6.1 不明なエラーが発生した場合

もし原因不明のエラーが発生して、対処法が分からない場合は、まずはエラーメッセージを Google などで検索してみるとよい。大抵、何らかのヒントが得られるだろう。

それでも分からない場合は、質問用掲示板にて質問する。

もっとも「インストールしたけれど動きません。どうしたらいいですか?」といった質問では、誰も答えてくれないだろう。例えば「Windows 10 に『美文書第 7 版』DVD からインストールした TeX Live 2016 で、TeXworks に○ページの例を入力して [タイプセット] ボタンを押すと次のようなエラーメッセージが出ます」といった具体的な質問であれば、そのとき時間のある有志の方が答えてくれるだろう。

肝心なのは、エラーが再現できる材料を全て提示することである。但し、もし何百行もあるファイルでエラーが発生したなら、それを全て送るのではなく、本質的でない部分を少しずつ削って短くし、これ以上短くするとエラーが発生しなくなる（あるいは別のエラーが発生する）ところまで短くして添付すること。これがいわゆる「エラーが再現する最小例」である。厳密に「最小」である必要はないが、このようにエラーを保ったままファイルを切り詰めることで、エラーの原因が自ずと明らかになることが多々ある。

2.7 texdoc の使い方

TeX Live のマニュアルは「texdoc」というコマンドで参照することができる。例えば、jsarticle の詳しい使い方を知りたければ、ターミナル上に次のように打ち込む。

```
texdoc jsarticle
```

これで「pLATEX_ε 新ドキュメントクラス」というマニュアルが画面に表示される。“texdoc jsarticle”だけでなく“texdoc jsbook”や“texdoc jsclasses”と打ち込んでも同じマニュアルが表示される*4。

*4 texdoc は候補となるマニュアルのファイル名などを元にもっともらしさの点数を付け、最高得点のマニュアルを 1 つ開く。検討対象になった他のマニュアルも参照したい場合には、texdoc -l で一覧から指定することができる。

第 3 章

L^AT_EX2e の基本

本章では、L^AT_EX (L^AT_EX 2_ε、pL^AT_EX 2_ε) の基本を一通り説明し、併せて upL^AT_EX、X_YL^AT_EX、LuaL^AT_EX にも触れる。

3.1 L^AT_EX2e の入力・印刷の完全な例

エディタで次のようなテキストファイルを作成する。ファイル名の拡張子は `tex` とする。ここではファイル名を `test.tex` としておく。

```
\documentclass{jsarticle}
\begin{document}
「何人ものニュートンがいた (There were several Newtons)」
と言ったのは、科学史家ハイルブロンである。同様にコーヘンは
「ニュートンは常に 2 つの貌を持っていた (Newton was always ambivalent)」と語っている。

近代物理学史上でもっとも傑出しもっとも影響の大きな人物が
ニュートンであることは、誰しも頷くことであろう。
しかしハイルブロンやコーヘンの言うようにニュートンは様々な、ときには相矛盾した顔を持ち、
その影響もまた時代とともに大きく変っていった。
\end{document}
```

保存したら L^AT_EX (pL^AT_EX 2_ε) で処理し、画面もしくはプリンタに出力する。

次のように出力できただろうか？

「何人ものニュートンがいた (There were several Newtons)」と言ったのは、科学史家ハイルブロンである。同様にコーヘンは「ニュートンは常に 2 つの貌を持っていた (Newton was always ambivalent)」と語っている。

近代物理学史上でもっとも傑出しもっとも影響の大きな人物がニュートンであることは、誰しも頷くことであろう。しかし、ハイルブロンやコーヘンの言うようにニュートンは様々な、ときには相矛盾した顔を持ち、その影響もまた時代とともに大きく変っていった。

上の入力例のように、途中で適当に **Enter** キーで改行するのが、昔からの T_EX 流の方法である。T_EX では、ほとんどの改行は無視されるだけである。但し、改行を 2 回続けて打つと（すなわち、空行があると）段落の区切りと解釈される。

段落の頭に全角スペースを入れなくても、自動的に段落の頭は 1 文字分だけ字下げされる。

\ (バックスラッシュ) で始まる文字列は、\ も含めていわゆる半角文字で入力する。Windows の和文フォントでは \ が ¥ と表示されるので注意を要する。本文の英語部分も半角文字で入力するが、日本語の句読点や括弧は全角文字を用いる。

PDF に埋め込まれる日本語のフォント (和文フォント) は、デフォルトでは IPAex またはヒラギノになっていると思われる。これを変更する方法は第 13 章を参照のこと。一方、英語のフォント (欧文フォント) は、標準では Computer Modern という書体となる。これを Times にするには、2 行目に次のような 1 行を追加する。

```
\documentclass{jsarticle}
\usepackage{newtxtext} % ←本文を Times、見出しを Helvetica に
\begin{document}
...
\end{document}
```

また、Palatino フォントにするには、次のようにする。

```
\documentclass{jsarticle}
\usepackage{newpxtext} % ←本文を Palatino、見出しを Helvetica に
\begin{document}
...
\end{document}
```

これらのフォントでエラーが起こるようなら、 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ のシステムが古いか、インストールがうまくいっていない可能性がある。

標準フォント (Computer Modern) と Times、Palatino では次のような違いがある。

標準 フォント： There were several Newtons. Newton was always ambivalent.

Times フォント： There were several Newtons. Newton was always ambivalent.

Palatino フォント： There were several Newtons. Newton was always ambivalent.

より多くの欧文フォントの例と設定方法については第 12 章を参照のこと。

3.2 最低限のルール

文書ファイルを作成する際に、とりあえず次のルールさえ知っていれば、ベタの文章なら間違いなく作成することができる。個々のルールについては後述する。

- 文書ファイル名の拡張子は `tex` とする。日本語のファイル名や空白や記号類はうまく扱えないことがあるので、英語のアルファベット・数字・アンダースコア (アンダーバー) に限るのが安全である。
- 文書の最初に半角で次のように記述する。

```
\documentclass{jsarticle}
```

これは、文書を `jsarticle` という名前の書式 (ドキュメントクラス) で組版することを意味する。ドキュメントクラスは、 $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 以外のソフトウェアで「スタイル」もしくは「テンプレート」と呼ばれるものに相当する。`jsarticle` は比較的新しいドキュメントクラスの 1 つである。これがインストールされていない場合は、

```
\documentclass{jarticle}
```

のように `jarticle` クラスを指定する。用紙が A4 判でない場合は、

```
\documentclass[a5paper]{jsarticle}
```

のように `a5paper`、`b4paper`、`b5paper` のいずれかを [] に囲んで指定する。

- 標準フォントで飽き足りない場合には、例えば次のようなフォント指定を行う (両方指定することはできない)。

```
\usepackage{newtxtext} % ← Times、 Helvetica を使う。
```

```
\usepackage{newpxtext} % ← Palatino、 Helvetica を使う。
```

- 次に、これから文書が始まることを意味する次の命令を記述する。

```
\begin{document}
```

- 文書の最後には、次の命令を記述する。

```
\end{document}
```

- これらの命令は全て半角の `\` で始まるが、Windows の和文フォントでは半角の円記号 \yen として表示される。
- 入力しやすいように、適当に Enter キーで改行して構わない。但し、Enter キーを 2 回続けて打つ (空の行を作る) と、そこが段落の区切りと見なされる。段落の頭では、自動的に全角 1 文字分の字下げ (インデント) が行われる。字下げしたくない段落には `\noindent` を付加する。
- 以下の半角文字は、そのままでは出力することができない。これらの文字を出力する方法は後述する。

```
# $ % & _ \ ^ ~ < > |
```

3.3 半角カナや機種依存文字を使うには

今までの pL^AT_EX の解説書には、次のようなことが記載されていた。

- いわゆる半角カナは使わず、全角カナを使うこと。
- 機種依存文字 (JIS 第 1・第 2 水準外の文字) は、そのままでは文字抜け・文字化けする可能性がある。

しかし、今日では、この制限を取り払う方法がいくつも存在する。一番簡単なのは pL^AT_EX の代わりに upL^AT_EX を使う方法である。upL^AT_EX を使うには `\documentclass[uplatex]{jsarticle}` のように uplatex オプション、または autodetect-engine オプションを付加する。また、文書ファイルは必ず文字コードを UTF-8 にして保存する。

タイプセットのコマンドは latex ではなく uplatex とする*¹。dvipdfmx による処理は今まで通りである。

3.4 ドキュメントクラス

L^AT_EX 2_ε の文書ファイルの最初の `\documentclass{...}` のような行は、ドキュメント (文書) のクラス (種類) を指定するものである。この中括弧 { } の中には、次のいずれかを指定する。頭に u が付くものは upL^AT_EX 用のドキュメントクラスである。

用途	欧文	和文 (旧・横)	和文 (旧・縦)	和文 (新・横)
論文・レポート	article	(u)jarticle	(u)tarticle	jsarticle
長い報告書	report	(u)jreport	(u)treport	-
本	book	(u)jbook	(u)tbook	jsbook

article の類は、論文やレポートなど、いくつかの節 (section) からなる文章のためのクラスである。book や report の類は、書籍などいくつかの章 (chapter) からなる文書のためのクラスである。jsarticle、jsbook が本稿で推奨する新しい横書きドキュメントクラスである。jsreport は用意されていないが `\documentclass[report]{jsbook}` とすれば jsreport 相当となる。

これら以外にも、いろいろなドキュメントクラスが出版社や学会などにより提供されている。用途にあったものを使用すればよい。

`\documentclass[a5paper]{jsarticle}` のような角括弧 [] の中は、ドキュメントクラスのオプションといい、必要に応じて次のような指定を行う。「デフォルト」と記述したものは無指定でそのようになるので、特に指定する必要はない。

- 10pt : 本文の欧文文字サイズを 10 ポイントにする (デフォルト)。
- 11pt : 本文の欧文文字サイズを 11 ポイントにする。
- 12pt : 本文の欧文文字サイズを 12 ポイントにする。

複数のオプションを指定する際には、

```
\documentclass[b5paper,12pt,papersize]{jsarticle}
```

のように、半角のコンマで区切る (順不同)。

次に示すのは用紙サイズのオプションである。デフォルトでは A4 版になる。

- a4paper : A4 判 (210 mm × 297 mm)
- a5paper : A5 判 (148 mm × 210 mm)
- b4paper : B4 判 (257 mm × 364 mm)
- b5paper : B5 版 (182 mm × 257 mm)
- papersize : 出力 PDF サイズを用紙サイズに合わせる。

残りは組み方のオプションである。

- twocolumn : 二段組にする。

*¹ uplatex と dvipdfmx を順に起動する ptex2pdf -u -l というコマンドも存在する。

3.5 プリアンブル

L^AT_EX 2_ε の文書ファイルは通常、

```
\documentclass{jsarticle}
\begin{document}
  (本文)
\end{document}
```

のように記述するが、`\documentclass{...}` と `\begin{document}` の間に更に細かい設定を記述することができる。この部分のことをプリアンブル (preamble、前口上) という。

例えば、ページ番号を振りたくない場合は、プリアンブルに `\pagestyle{empty}` と記述する。すなわち、

```
\documentclass{jsarticle}
\pagestyle{empty}
\begin{document}
  (本文)
\end{document}
```

のように記述することになる。

文書全体についてのフォント指定もプリアンブルで行う。例えば、欧文や数字を Times 系のフォントにするには、

```
\usepackage{newtxtext}
```

と記述する。このように記述することを「newtxtext パッケージを使う」と表現する。

3.6 文章の構造

文書はタイトル、著者名、章の見出し、節の見出し、段落、... のような構造を持っている。L^AT_EX で文書ファイル中に書き込むのは、このような文書の構造である。これに対して、いわゆるワープロソフトは文書の構造とレイアウト (文字サイズや書体) などが明確に分かれていない。

Web ページを制作するための HTML についても、L^AT_EX と同様のことが言える。HTML で表すのは文書の構造である。これを実際のレイアウトに結びつけているのは、CSS (スタイルシート) である。L^AT_EX において CSS に相当するものが、ドキュメントクラスやスタイルファイルである。

現在の多くのワープロソフトでは、スタイル指定により文書の構造とレイアウトを対応付けることができる。しかし、スタイルから逸脱することがあまりにも容易にできてしまうため、よほど注意しなければ不統一なレイアウトになってしまう恐れがある。

例えば、次の文書を考えてみる。

1 序章

1.1 チャーチルのメモ

1940 年、潰滅の危機に瀕した英国の宰相の座についたウィンストン・チャーチルは、政府各部署の長に次のようなメモを送った。

われわれの職務を遂行するには大量の書類を読まねばならぬ。その書類のほとんどすべてが長すぎる。時間が無駄だし、要点を見つけるのに手間がかかる。

同僚諸兄とその部下の方々に、報告書をもっと短くするようにご配慮願いたい。

この文書で「1 序章」は章の見出し、「1.1 チャーチルのメモ」は節の見出しである。また、この文書は3つの段落から成るが、最後の2つの段落は地の文書ではなく、引用した文書である。

L^AT_EX に入力する文書ファイルでは、これらを次のように表現する。

```
\documentclass{jsarticle}
\begin{document}
\section{序章}
\subsection{チャーチルのメモ}
1940 年、潰滅の危機に瀕した英国の宰相の座についたウィンストン・チャーチルは、政府各部署の長に次のようなメモを送った。
\begin{quotation}
われわれの職務を遂行するには大量の書類を読まねばならぬ。
その書類のほとんどすべてが長すぎる。時間が無駄だし、
要点を見つけるのに手間がかかる。

同僚諸兄とその部下の方々に、
報告書をもっと短くするようにご配慮願いたい。
\end{quotation}
\end{document}
```

この例では、文書の構造を記述するために次のような命令が使われている。

- `\section{...}` セクション（節）の見出し
- `\subsection{...}` サブセクション（小節）の見出し
- `\begin{quotation}` 引用の始まり
- `\end{quotation}` 引用の終わり

この他に必要に応じて次のような命令で文書の構造を明示する。

- `\part{...}` 第何部という部見出し
- `\chapter{...}` 章見出し（report、book、jreport、jbook、jsbook のみ）
- `\subsubsection{...}` サブサブセクション（小々節）の見出し
- `\subparagraph{...}` サブパラグラフ（小段落）の見出し
- `\begin{quote}` 短い引用の始まり（quotation 環境と違って段落の頭を字下げしない）
- `\end{quote}` 短い引用の終わり

段落も文書構造の1つである。段落の区切りは空行によって表される*2。

3.7 タイトルと概要

タイトルを出力するためには次の4つの命令を用いる。

- `\title{...}` 文書名の指定
- `\author{...}` 著者名の指定
- `\date{...}` 日付の指定
- `\maketitle` 文書名・著者名・日付の出力

`\title`、`\author`、`\date` は `\maketitle` の前ならどこに記述しても構わない。また、この3つの順序もどれが先でも構わない。実際にタイトルを出力するのは `\maketitle` である。

タイトル、著者名、日付が長い場合は `\\` で区切れば、そこで改行される。

```
\title{非常に長いタイトルを \\ 二行に分ける是非について}
\author{寿限無寿限無五劫の擦り切れ \\ 海砂利水魚の水行末}
\date{2016 年 07 年 08 日投稿 \\ 2016 年 09 日 10 日受理}
```

*2 強制改行のコマンド `\\` を段落の区切りに使うべきではない。

例えば、次のように記述する。

```
\documentclass{jsarticle}
\begin{document}
\title{理科系の作文技術}
\author{木下 是雄}
\date{1981 年 09 月 25 日}
\maketitle

\section{序章}
\subsection{チャーチルのメモ}
1940 年、潰滅の危機に瀕した英国の……
\end{document}
```

また、著者が複数人存在する場合は次のように `\and` で区切る。`\and` の直後には半角空白を入れなければならない。`\and` の直前の半角空白は無視される。

```
\author{アルファ \and ベーテ \and ガモフ}
```

タイトルや著者名に脚注を付ける場合は `\thanks` という命令で、

```
\author{湯川秀樹\thanks{京都大} \and 朝永振一郎\thanks{東京大}}
```

のようにする。なぜ、`\thanks` なのかというと、研究費を援助してくれた機関を著者名の脚注で記述するのが慣習となっているからである。

`\date` 指定を省略すれば、文書ファイルを \LaTeX で処理した日付を出力する。`article`、`jarticle`、`jsarticle` クラスでは、タイトルは本文第 1 ページの上部に出力される。もし、タイトルのページを独立に 1 ページ取りたいなら、

```
\documentclass[titlepage]{jsarticle}
```

のようにドキュメントクラスのオプションとして `titlepage` を指定する。

また、`\maketitle` の直後に `\begin{abstract}` と `\end{abstract}` で囲んで論文の要約（概要・梗概）を書いておけば、タイトルと本文の間に出力される（`titlepage` オプションを指定した場合は独立のページに出力される）。

3.8 入力ファイルに書ける文字

以上の説明で、数式や表などを含まない文章なら大抵扱えると思われる。ここでは、また最初に戻って、1 つ 1 つの文字について更に詳しく述べることにする。

\TeX で文書中に書いてそのまま出力することができる「安全な」欧文文字は、次の通りである。

```
! ' ( ) * + , - . / : ; = ? @ [ ] ^
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

これら以外に欧文用のスペース（いわゆる半角空白）が存在する。本来は見えない文字だが、本稿では便宜上「」と書くことがある。また、キーボード上には存在するが、入力してもそのまま出力されない欧文用文字には次のものがあり、それぞれ \TeX では特別な意味を持っている。

```
# $ % & _ { } \ ^ ~
```

次の 3 文字は \TeX のデフォルトの状態では数式モードでしか出力することができない。

```
< > |
```

これら以外に次の「文字」も存在する。

タブ：Tab キーを押して入力される文字である。これは半角空白と同じ意味となる。

改行：Enter キーを押して入力される文字である。欧文では半角空白と同じ意味となり、和文では無視される。2 回続けて打つと段落の区切りとして扱われる。

これら以外の文字は \ で始まる命令で表す。例えば、æ という文字は \ae と記述することで出力することができる。この種の命令については後述する。

pTeX では上記の他に、和文の文字（いわゆる全角文字）を扱うことができる。

3.9 打ち込んだ通りに出力する方法

半角文字の # \$ % & _ { } \ ^ ~ < > | はそのままではうまく出力することができない。笑顔のつもりで (^_^) などと書けばエラーとなってしまう。また、改行は無視され、半角のスペースは何個並べても 1 個分のスペースしか出力されない（これらのルールについては後述する）。

次のように \begin{verbatim}、\end{verbatim} で囲んだ部分は、入力画面の通りに出力される。この“verbatim”（ヴァーベイティム）は「文字通りに」という意味の英語である。

```
\documentclass{jsarticle}
\begin{document}
メールで用いられる記号類
\begin{verbatim}
:-)      元祖スマイル
^^;      冷や汗
\end{verbatim}
\end{document}
```

これを出力すると、次のようになる。

```
メールで用いられる記号類
:-)      元祖スマイル
^^;      冷や汗
```

半角文字は幅一定の typewriter 書体となるが、半角文字と全角文字の幅の比は一般に 1:2 にはならない。この verbatim 環境はコンピュータのプログラムを出力する際に便利である。

verbatim 環境は行単位だが、数文字なら \verb|...| という書き方を用いる。例えば、(^_^) と出力したいなら \verb|(^_^)| と記述する。区切りは縦棒 | に限らず、両側が同じなら \verb/(^_^)/ でも \verb"(^_^)" でも \verb@(^_^)@ でも構わない。

\begin{verbatim*}... \end{verbatim*} や \verb*|...| のように * 印を付けると、半角文字が ̣ という文字で出力される。なお、\chapter{...} や \section{...} のような \何々{...} 型の命令の { } の中では \verb 命令は大抵使えないので、特殊文字を出力するためには他の方法に依らなければならない。これは後述する \何々[...] 型の命令の [] の中でも同様である。

3.10 改行の扱い

ここでいう改行とは Enter キーを打つことである。

3.10.1 改行は無視される（行末が和文文字の場合）

InDesign などの DTP ソフトと異なり、TeX は通常は改行を無視するので、TeX 用の入力ファイルにはポエムのように改行を多量に入れる人が多いようである。その方が、Git などのバージョン管理システムで変更点がよくわかって便利

である。

但し、空の行（何も入力していない行、Enter だけの行）があると、**T_EX** はそこを段落の区切りと解釈する。すなわち、Enter キーを 1 度だけ打っても **T_EX** はそれを無視するが、2 度続けて Enter を打てば、空行ができるので、そこで段落が改まり、通常の設定では次の行の頭に 1 文字分の空白（字下げ、インデント）が入る（全角下がり）。

以下の例でこの点をよく理解しておくこと。

入力	改行は 無視される。
出力	改行は無視される。

入力	空行があると 段落が改まる。
出力	空行があると 段落が改まる。

3.10.2 改行は空白になる（行末が欧文文字の場合）

入力ファイルの改行は（空行でない限り）無視されると前述したが、これは和文文字の場合だけである。詳述すると、行の最後の文字が和文の文字（かな・漢字などのいわゆる全角文字）であれば、その直後に打った Enter キーは無視される。

ところが、行の最後の文字が半角文字（欧文文字）であれば、その直後に打った Enter キーは半角空白と同じ意味になる。少々ややこしいが、実際に欧文を入力してみれば、ごく自然なルールであることが理解できる。

次の例では半角文字 a の直後の Enter が空白になる。

入力	Here's a good example.
出力	Here's a good example.

なお、欧文の場合でも、空行があると **T_EX** はそこを段落の区切りであると解釈する。

3.11 注釈（コメント）

欧文では改行は空白となるが、場合によっては改行を単に無視してほしいこともある。このような場合は、最後の文字の直後に % を記述する。

入力	Supercalifragilistic% expialidocious!
出力	Supercalifragilisticexpialidocious!

この % は、その行のそれ以降を **T_EX** に無視させる特殊な命令である。この文字からは改行文字 Enter も含めて全て無視されるので、注釈（コメント）を記述するのに便利である。

3.12 空白の扱い

空白（スペース）には全角空白と半角空白が存在する。本稿では、紛らわしい場合には半角空白を と表記している。

エディタの画面上では半角文字 2 個 と全角空白 1 個とは区別が付きにくいですが、**T_EX** にとってはこれらの意味は全く異なる。全角空白を並べれば、単にその個数分の全角空白が出力されるだけである。

半角空白をいくつも出力するためには、\ で区切る。

なお、行頭・行末の半角空白「」は何個あっても無視される。

TeX の入力ファイルには、^じ地の文と組版命令が入り混ざっている。組版命令には様々なものが存在し、自前の命令を定義することもできる。

もし、半角空白を入れずに“この本は\LaTeXで書きました”と書いてしまうと「\LaTeXで書きました」の部分が命令と解釈されてしまい、エラー（誤り）となってしまう。

- この本は`\LaTeX`で書きました。← 半角空白を入れる。
- この本は`\LaTeX` ← 改行する。
で書きました。
- この本は`\LaTeX{}`で書きました。← `{}` を付ける。
- この本は`{\LaTeX}`で書きました。← `{}` で囲む。

\LaTeX, がんばれ!

$$\backslash\mathrm{LaTeX}_{\text{is awesome!}} \rightarrow \mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}_{\text{is awesome!}}$$

このような場合は、スペースを入れる命令 `_` を使うか、`{ }` で区切る。

<code>\LaTeX\isawesome!</code>	\rightarrow	L ^A T _E X is awesome!
<code>\LaTeX{}\isawesome!</code>	\rightarrow	L ^A T _E X is awesome!
<code>{\LaTeX}\isawesome!</code>	\rightarrow	L ^A T _E X is awesome!

`\$` も `\` で始まるので `TEX` の命令の一種だが、`\` にアルファベットではない記号・数字が付いてできた命令は `\LaTeX` のような命令と少し違った性質を持つ。

- これらの命令は \ の後ろに 1 文字しか付かない。例えば、\\$ という命令は存在するが \\$# や \$foo といった命令は存在しない。
- \\$ のような命令は \\$_29 のように空白で区切る必要はない。単に \$29 のように書く。もし、\$_29 のように空白を付けると、\$ 29 のように実際に空白が出力されてしまう。

3.15 特殊文字

\LaTeX という命令で L^AT_EX と出力され、\\$ という命令で \$ と出力された。この類の特殊記号・特殊文字を出力する命令の一部を以下に列挙しておく。

前述の通り、例えば Ångström を \AAngstr\om と書いたのでは、L^AT_EX は「\AAngstr という命令は存在しない」「\om という命令は存在しない」というエラーメッセージを吐くだけである。区切りの中括弧 {} を用いて、

\AA{}ngstr\o{}m もしくは {\AA}ngstr{\o}m

と書くか、半角空白で \AA_ngstr\o_m のように区切る。

入力	出力	入力	出力	入力	出力	入力	出力
\#	#	\copyright	©	\L	Ł	''\,	''
\\$	\$	\pounds	£	ss	ß	'\,''	'''
\%	%	\oe	œ	?‘	¿	-	-
\&	&	\OE	Œ	!‘	¡	--	—
_	—	\ae	æ	\i	ı	---	—
\{	{	\AE	Æ	\j	ĵ	\texttrgistered	®
\}	}	\aa	å	‘	‘	\texttrademark	™
\S	§	\AA	Å	’	’	\textasciitilde	~
\P	¶	\o	ø	“	“	\TeX	T _E X
\dag	†	\O	Ø	”	”	\LaTeX	L ^A T _E X
\ddag	‡	\l	ł	*	*	\LaTeXe	L ^A T _E X 2 _ε

\& のような記号で終わる命令には区切りは不要である（半角空白はそのまま出力される）。記号類は \& などと書く代わりに全角（和文）文字を用いても構わない（デザインは少々異なる）。

アスタリスク * を入力すると、標準では * のように上寄りに出力されるが、\$*\$ のように \$ で挟むと * のように中央に出力される。

--- は — のような欧文用のエムダッシュ（通常 1 [em] の長さのダッシュ）を出力する命令である。これは文中で間を置いて読むべきところ — 例えば、説明的な部分の区切り — に用いられる。

Remember, even if you with the rat race — you're still a rat.

以上の他に、特殊文字ではないが、今日の日付を出力する命令 \today は便利である。これは \date を省略した際の \maketitle と同様に、日付を出力する。

3.16 アクセント類

以下に、欧文で使用する種々のアクセント類を出力する命令を示す。

入力	出力	入力	出力	入力	出力	入力	出力
\'{}o	ó	\~{}o	õ	\v{}o	ö	\d{}o	ø
\'{}o	ó	\={}o	ō	\H{}o	ő	\b{}o	ö
\^{}o	ô	\.{}o	ò	\t{}oo	öo	\r{}a	å
\"{}o	ö	\u{}o	ü	\c{}o	ç	\k{}a	ą

例：Schr\"{}o{}dinger → Schrödinger、al-Khw\={}a{}rizm\~{}{i} → al-Khwārizmī

数式モードを用いれば、更に多くのアクセント記号を出力することができる。数式モードについては第 5 章を参照のこと。

3.17 書体を変える命令

前述した通り、 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ ではなるべく文書の構造を指定する命令のみを使い、書体や文字サイズを直接指定する命令を使うことは避けるべきである。しかし、とりあえずワープロ代わりに使いたい場合もあるであろうから、書体や文字サイズを変更する方法についても説明しておくことにする。

3.17.1 和文書体

和文書体については第 13 章で詳述するが、ここではとりあえずゴシック体に変える命令 $\text{\texttt{\texttt{...}}}$ について挙げておく。

入力 | $\text{\texttt{\texttt{ゴシック体}}}$ は見出しなどに用いる。

出力 | ゴシック体は見出しなどに用いる。

3.17.2 欧文書体

欧文書体については第 12 章で詳述するが、 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ でよく用いられる 7 書体について、簡単な指定方法を挙げておく。

$\text{\texttt{\texttt{Roman}}}$	Roman	本文（デフォルト）
$\text{\texttt{\texttt{Boldface}}}$	Boldface	見出し
$\text{\texttt{\texttt{Italic}}}$	<i>Italic</i>	強調、書名
$\text{\texttt{\texttt{Slanted}}}$	<i>Slanted</i>	<i>Italic</i> の代用
$\text{\texttt{\texttt{Sans Serif}}}$	Sans Serif	見出し
$\text{\texttt{\texttt{Typewriter}}}$	Typewriter	コンピュータの入力例
$\text{\texttt{\texttt{Small Caps}}}$	SMALL CAPS	見出し

これらの内、Roman 体はデフォルトなので $\text{\texttt{\texttt{...}}}$ を使う機会はほとんど無い。 $\text{\texttt{\texttt{Italic}}}$ の代わりに $\text{\texttt{\texttt{emph}}}$ という命令も使うことができる。この方が強調 (emphasis) していることがわかりやすい名前である。この $\text{\texttt{\texttt{emph}}}$ は、周囲がイタリック体であれば逆にローマン体に戻すはたらきを持つ。

Slanted はローマン体を機械的に斜めにしたもので、一般にはイタリック体であれば $\text{\texttt{\texttt{Italic}}}$ を用いるべきである。

3.18 文字サイズを変える命令

文字の大きさについても第 12 章で詳述するが、よく使われるのは次のように標準から相対的な大きさを指定する命令である。欧文フォントが 10 ポイントの場合の実サイズと共に挙げておく。

$\text{\texttt{\texttt{tiny}}}$	5.00 ポイント	見本 Sample
$\text{\texttt{\texttt{scriptsize}}}$	7.00 ポイント	見本 Sample
$\text{\texttt{\texttt{footnotesize}}}$	8.00 ポイント	見本 Sample
$\text{\texttt{\texttt{small}}}$	9.00 ポイント	見本 Sample
$\text{\texttt{\texttt{normalsize}}}$	10.00 ポイント (標準)	見本 Sample
$\text{\texttt{\texttt{large}}}$	12.00 ポイント	見本 Sample
$\text{\texttt{\texttt{Large}}}$	14.40 ポイント	見本 Sample
$\text{\texttt{\texttt{LARGE}}}$	17.28 ポイント	見本 Sample
$\text{\texttt{\texttt{huge}}}$	20.74 ポイント	見本 Sample
$\text{\texttt{\texttt{HUGE}}}$	24.88 ポイント	見本 Sample

この内、 $\text{\texttt{\texttt{normalsize}}}$ は標準の大きさなので特に指定する必要はない。これらの命令は、 $\text{\texttt{\texttt{small}}}$ 「小さな文字」のように、命令の区切りに半角空白を入れ、適用範囲を $\{ \}$ で囲む。

3.19 環境

`\begin{何々}` ... `\end{何々}` のように対になった命令を環境 (environment) という。例えば、`\begin{quote}` ... `\end{quote}` なら `quote` 環境という。環境の内側は一種の別天地であり、様々な設定が環境の外側とは異なる。例えば `quote` 環境なら左マージン (左余白) が周囲より広くなる。環境の中で書体などを変えても、環境の外には影響が及ぶことはない。

入力	ここは環境の外。 <code>\begin{quote}</code> ここは環境の中。ここで <code>\small</code> 文字サイズを変えても… <code>\end{quote}</code> 環境の外では元の書体に戻る。
出力	ここは環境の外。 ここは環境の中。ここで文字サイズを変えても… 環境の外では元の書体に戻る。

`quote` 以外によく使われる環境は次の通りである。

- `flashleft` 環境 左寄せ
- `flushright` 環境 右寄せ
- `center` 環境 センタリング (中央寄せ)

これらの環境の中で改行するには `\\` を用いる。

3.20 箇条書き

環境の例として、色々な箇条書きの方法を説明する。

3.20.1 itemize 環境

頭に `•` などの記号を付けた箇条書きである。

入力	出力
<code>\LaTeX</code> には、 <code>\begin{itemize}</code> <code>\item</code> 記号付き箇条書き <code>\item</code> 番号付き箇条書き <code>\item</code> 見出し付き箇条書き <code>\end{itemize}</code>	<code>L^AT_EX</code> には、 <code>•</code> 記号付き箇条書き <code>•</code> 番号付き箇条書き <code>•</code> 見出し付き箇条書き

の機能が存在する。 の機能が存在する。

入れ子にすると、標準の設定では次のように記号が変化する。

入力	出力
<code>\begin{itemize}</code> <code>\item</code> 第 1 レベルの箇条書き <code>\begin{itemize}</code> <code>\item</code> 第 2 レベルの箇条書き <code>\begin{itemize}</code> <code>\item</code> 第 3 レベルの箇条書き <code>\begin{itemize}</code> <code>\item</code> 第 4 レベルの箇条書き <code>\end{itemize}</code> <code>\end{itemize}</code> <code>\end{itemize}</code> <code>\end{itemize}</code> <code>\end{itemize}</code> <code>\end{itemize}</code>	 <code>•</code> 第 1 レベルの箇条書き – 第 2 レベルの箇条書き * 第 3 レベルの箇条書き ・ 第 4 レベルの箇条書き

```
\end{itemize}
```

また、`\item[★]` のようにすると、その部分だけ記号を変えることができる。

3.20.2 enumerate 環境

頭に番号を付けた箇条書きである。

入力	出力
<code>\LaTeX</code> には、	\LaTeX には、
<code>\begin{enumerate}</code>	
<code>\item</code> 記号付き箇条書き	1. 記号付き箇条書き
<code>\item</code> 番号付き箇条書き	2. 番号付き箇条書き
<code>\item</code> 見出し付き箇条書き	3. 見出し付き箇条書き
<code>\end{enumerate}</code>	

の機能が存在する。 の機能が存在する。

入れ子にすると、標準の設定では次のように番号の付け方が変化する。

入力	出力
<code>\begin{enumerate}</code>	
<code>\item</code> 第 1 レベルの箇条書き	1. 第 1 レベルの箇条書き
<code>\begin{enumerate}</code>	
<code>\item</code> 第 2 レベルの箇条書き	(a) 第 2 レベルの箇条書き
<code>\begin{enumerate}</code>	
<code>\item</code> 第 3 レベルの箇条書き	i. 第 3 レベルの箇条書き
<code>\begin{enumerate}</code>	
<code>\item</code> 第 4 レベルの箇条書き	A. 第 4 レベルの箇条書き
<code>\end{enumerate}</code>	
<code>\end{enumerate}</code>	
<code>\end{enumerate}</code>	

3.20.3 description 環境

左寄せ太字で見出しを付けた箇条書きである。例えば、次のような入力を用意すると、

```
\begin{center}
  \large 第 7 回\LaTeX 勉強会の開催について（案内）
\end{center}
下記の通り行いますので、
万障お繰り合わせの上、ご参集下さい。
\begin{center} 記 \end{center}
\begin{description}
\item[日時] 2017 年 01 月 24 日 午後 3 時
\item[場所] 当社 2 階会議室
\item[用意するもの] 技術評論社『\LaTeX 美文書作成入門』（特に第 3 章をよく読んでおいて下さい）
\end{description}
\begin{flushright} 以上 \end{flushright}
```

次のような出力が得られる。

第 7 回 \LaTeX 勉強会の開催について（案内）
下記の通り行いますので、万障お繰り合わせの上、ご参集下さい。

日時 2017 年 01 月 24 日 午後 3 時

場所 当社 2 階会議室

用意するもの 技術評論社『 \LaTeX 美文書作成入門』(特に第 3 章をよく読んでおいて下さい)

以上

このように `\begin{description} ... \end{description}` を用いる。それぞれの箇条の頭には `\item[見出し]` を付加する。見出しの直後で改行したい場合は、単に強制改行 `\\` を入れてもうまくいかない。次のように `\mbox{}` という見えない箱を入れることでうまくいくようになる。

```
\begin{description}
\item[日時] \mbox{} \\
    2017 年 01 月 24 日 午後 3 時
\item[場所] \mbox{} \\
    当社 2 階会議室
\end{description}
```

3.21 長さの単位

\TeX で使用することができる長さの単位には次のものが存在する。最後の 4 つは \pTeX 、 \upTeX のみで使用可能である。

cm センチメートル (1 cm = 10 mm)
mm ミリメートル
in インチ (1 in = 2.54 cm)
pt ポイント (72.27 pt = 1 in)
pc パイカ (1 pc = 12 pt)
bp ビックポイント (72 bp = 1 in、DTP ポイント)
sp スケールドポイント (65536 sp = 1 pt)
em 文字サイズの公称値 (元来は “M” の幅)
ex 現在の欧文フォントの “x” の高さ (公称値)
zw 現在の和文フォントのボディの幅 (ベタ組み時の字送り量)
zh 現在の和文フォントの高さ (使わない方がいい)
Q 級 (1 Q = 0.25 mm)
H 歯 (1 H = 0.25 mm)

印刷関係ではポイント (point、pt、ポ) という単位をよく用いる。ポイントの定義は国によって若干の違いはあるが、 \TeX では 1 ポイントを $1/72.27$ インチと定義している。日本工業規格 (JIS) の 1 ポイント = 0.3514 mm と実質的には同じである。DTP ソフトや Word などでは、1 ポイントを $1/72$ インチ (\TeX でいうところのビックポイント) としている。写植機で使われてきた級数 (歯数) は和製の単位で、1 級 (Q) = 1 歯 (H) = 0.25 mm である。1 mm の $1/4$ (Quarter) なので Q という。文字の大きさには級を、送りの指定には歯を用いる慣習となっている。

欧文の書籍では 10 ポイントの文字を使うことが多く、 \LaTeX でも欧文 10 ポイントが標準となっている。この 10 ポイントというのは、活版印刷では活字の上下幅、すなわち詰めものをしないで活字を詰めた場合の行送り量である。

コンピュータのフォントでは、どの長さが 10 ポイントなのかははっきりしないのだが、 \TeX 標準付属の Computer Modern Roman 体の 10 ポイント (cmr10) では、括弧 () の上下幅がちょうど 10 ポイントになっている (ベースラインから上 7.5 pt、下 2.5 pt)。また、たまたま cmr10 の数字 2 文字分の幅も 10 ポイントになっている (cmr9 の数字 2 文字分の幅は 9 ポイントではない)。

この 10 ポイントの欧文に合わせる和文の文字として jsarticle 等では 13 Q (約 9.2469 pt)、jarticle 等では 9.62216 pt (約 13.527 Q) と定義されている。すなわち、jsarticle 等では 1 zw は 9.2469 pt、jarticle 等では 1 zw は 9.62216 pt である。

zw が全角の幅 (width) であるのに対して、zh は元来は全角の高さ (height) だが、歴史的な理由により、伝統的な pTeX の和文フォントメトリックでは 1 zh は 1 zw よりわずかに小さい値になっている (ほぼ $1\text{ zw} = 1.05\text{ zh}$)。現在では 1 zh の値には特に意味がないので使わない方がよいだろう。なお、otf パッケージの新しいフォントメトリックでは正方形 (正確に $1\text{ zw} = 1\text{ zh}$) となっている。

3.22 空白を出力する命令

以下で長さを書いた部分は 1zw や 12.3mm など \TeX で使うことのできる単位を付けた数を書き込む。

左右に空白（スペース）を入れるには次のどちらかの命令を用いる。

- `\hspace{長さ}` 行末・行頭では出力されない。
- `\hspace*{長さ}` 行末・行頭でも出力される。`\hspace*{1zw}` は全角空白を 1 つ入れることと同値である。

段落間などに空白（スペース）を入れるには次のどちらかの命令を用いる。

- `\vspace{長さ}` ページ頭・ページ末では出力されない。段落間に余分の空白を入れる際によく用いられる。
- `\vspace*{長さ}` ページ頭・ページ末でも出力される。図を貼り込むスペースを空ける際などに用いられる。

例えば、0.5 行分のアキを入れるには `\vspace{0.5\baselineskip}` とする。

3.23 脚注への書き込み

このページの下*3にあるような脚注を出力するには `\footnote{...}` という命令を用いて、次のように記述する。

このページの下\footnote{これが脚注である。}_にあるような脚注…

本稿では \footnote{...} の直後に上の例のように半角空白「」を入れるか、あるいは、

このページの下\footnote{これが脚注である。}\n
にあるような脚注…

のように行末に \ を入れる方式を推奨している。行末の \ は半角空白と同じ意味となる。行末に \ を入れないと、その直前が全角文字なので空白は出力されない。

欧文では、文の途中の \footnote は、

Gee\footnote{Note.}\textvisiblespace\verb'whiz.

のように単語に密着し、後ろは空白、あるいは改行にする。和文の場合は、

かくかく\footnote{脚注。}。しかじか

のように句読点の直前に \footnote を入れるのが一般的である。

3.24 罫線の類

L^AT_EX 標準の下線・枠線などを引く機能を挙げておく。より複雑な効果は第 7 章などを参照のこと。

- `\underline{...}` で下線を引くことができる。例えば、ほげ (`\underline{ほげ}`) となる。L^AT_EX 標準の下線では途中で改行することができない。別の方法については後述する。
- `\hrulefill` で水平の罫線を引くことができる。例えば、


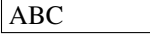

のようになる。



- `\dotfill` で水平の点線を引くことができる。例えば、
.....
のようになる。
- `\fbox{...}` で囲み枠を描くことができる。例えば、`\fbox{ABC}` で

ABC

 となる。

*³ これが脚注である。

- `\framebox[w]{...}` で幅 w の囲み枠を描くことができる。例えば、`\framebox[2cm]{ABC}` で  となる。同様に、`\framebox[2cm][l]{ABC}` で左揃え 、`\framebox[2cm][r]{ABC}` で右揃え  となる。
- `\rule[d]{w}{h}` で中身の詰まった長方形を描くことができる。 w は幅、 h は下端からの高さで、オプションの d はベースラインから下端までの距離である。 d 、 w 、 h を調節して任意の太さの縦罫・横罫を引くことができる。また、しばしば $w = 0$ として見えない縦罫の支柱を作り、行送りを微調整するために用いられる。

`\fbox` や `\framebox` の枠と中身の隙間は `\fboxsep` という長さで決まる。この長さはデフォルトでは 3pt だが、例えば `\setlength{\fboxsep}{0mm}` とすれば  のように隙間が無くなる。また、`\fbox` や `\framebox` の枠の太さは `\fboxrule` という長さで決まるが、この長さはデフォルトでは 0.4pt である。例えば `\setlength{\fboxrule}{0.8pt}` とすれば  のように太くなる。

第 4 章

パッケージと自前の命令

TeX には自前の命令（マクロ）を定義する機能が備わっている。LaTeX はこの機能を用いて TeX を拡張したものである。この機能を使えば、LaTeX を更に拡張することができる。

自前の命令は文書ファイルに直接書き込むこともできるが、パッケージ化して別ファイルに保存しておくこともできる。LaTeX 2_ε には様々な出来合いのパッケージが付属している。

本章では既存のパッケージの利用の仕方と、自分でパッケージを作成する方法を解説する。

4.1 パッケージ

パッケージとは LaTeX の機能を簡単に拡張するための仕組みである。例えば、ルビ（フリガナ）を振りたいとする。LaTeX にはルビを振る命令は存在しないので、何らかの方法で LaTeX を拡張しなければならない。このような場合に利用するのがパッケージである。

ルビを振る命令はさまざまなパッケージで定義されているが、ここでは `okumacro` というパッケージを使うことにする。そのためには、文書ファイルのプリアンブルに次のように記述する。

```
\usepackage{okumacro}
```

こうしておけば、ルビを振る命令 `\ruby` を使うことができるようになる。

入力	<pre>\documentclass{jsarticle} \usepackage{okumacro} \begin{document} \ruby{奥}{おく}\ruby{村}{むら}\ruby{晴}{はる}\ruby{彦}{ひこ}氏作のパッケージである。 \end{document}</pre>
----	---

出力	<div style="display: flex; align-items: center;"><div style="font-size: 0.8em; margin-right: 5px;">おくむらはるひこ</div>奥村晴彦氏作のパッケージである。</div>
----	---

`\usepackage{okumacro}` と記述すると LaTeX は `okumacro.sty` というファイルを読み込んで、その中で定義された命令を取り込む。すなわち、パッケージ `okumacro` の実体は `okumacro.sty` という名前のファイルである。もし、このファイルがコンピュータ上に存在しないと、次のようなエラーメッセージが出力される。

```
! LaTeX Error: File 'okumacro.sty' not found.
```

```
Type X to quit or <RETURN> to proceed.
```

```
or enter new name.(Default extension: sty)
```

```
Enter file name:
```

このようなエラーが出力されるのは `okumacro` の綴りを間違えているか、あるいは実際に `okumacro.sty` というファイルがインストールされていない可能性が考えられる。もし本当にインストールされていない場合は、インターネットで `okumacro.sty` をダウンロードして、どこかに格納すればよい。格納する場所は、現在の LaTeX 文書ファイル（ソースファイル）と同じフォルダの中が一番簡単である（pLaTeX 関連のファイルを格納する一般的な場所については、TeX Live のディレクトリ構造を理解している必要があるため後述する）。

4.2 節

L^AT_EX にて用紙の左右中央に、

記

と出力するには、

```
\begin{center} 記 \end{center}
```

と記述する。この入力を簡単にするために、`\記` という自前の命令を定義してみることにする。このような命令は別ファイルに記述するのが一般的だが、ここではとりあえず文書ファイルの中で、その命令を使いたい場所より前（例えばプリアンブル）に次のように `\記` の定義を記述しておく。

```
\newcommand{\記}{\begin{center} 記 \end{center}}
```

この `\newcommand` という命令は、新しい（new）命令（command）を定義するための命令である。上のように記述しておけば、それ以降 `\記` と記述すれば `\begin{center} 記 \end{center}` と記述するのと全く同じ意味となる。

L^AT_EX の命令のことを一般にコマンド（command）あるいは制御綴（せいぎょつづり control sequence）というが、このような自前の命令のことを特にマクロ（macro）とすることがある。

もう1つ例を挙げておく。小さい文字で弊社と何度も書く必要があるなら、次のように `\弊社` という命令を定義しておく。

```
\newcommand{\弊社}{\small 弊社}
```

右側の括弧は二重にしなければならない。単に、

```
\newcommand{\弊社}{\small 弊社}
```

としてしまつては、`\small 弊社` と記述したことと同じになり `\small` を閉じる括弧がないため、これ以降、文書の最後まで小さい文字になってしまう。また、この命令を使う際に、

```
\弊社では、...
```

と書いてしまつては、`\弊社では` という命令が未定義であるというエラーとなるため、

- `\弊社` では
- `\弊社{}` では
- `{\弊社}` では

のいずれかの書き方をする必要がある。

4.3 パッケージを作成する

マクロがいくつか作成できたら、自分用のパッケージに登録しておこう。

エディタを起動して、例えば `mymacros.sty` という名前のファイルを作成する。ファイル名は任意だが、拡張子は `sty` にしておく。このファイルに自分が定義した命令を並べて書き込んでおく。例えば、次のようにする。

```
\newcommand{\弊社}{\small 弊社}
\newcommand{\記}{\begin{center} 記 \end{center}}
```

この `mymacros.sty` は、とりあえずはカレントディレクトリ（文書ファイルの存在するフォルダ）に置いておけばよい。個々の文書ファイルでは、次のようにプリアンブルの `\usepackage` 命令でこのパッケージを読み込んで使用する。

```
\usepackage{mymacros}
```

このような命令が充実すればするほど、タイピングの量やレイアウトを考える必要が減り、文書の論理構造に集中できるようになる。

文書ファイルに、

```
\begin{center} 記 \end{center}
```

と記述することは、ワープロソフトと同様に、文書のレイアウトを指定していることになる。これに対して、`\記` と記述すれば、そこから文書の「記」という要素が始まるという文書の構造を示したことになる。

尚、読み込むパッケージが複数存在する場合は、

```
\usepackage{multicol}
\usepackage{mymacros}
```

のように記述しても、

```
\usepackage{multicol,mymacros}
```

のように並べて記述しても構わない。

4.4 命令の名前の付け方

命令の名前は `\FooBar` のような英字でも、`\命令` のような漢字でも、両者の混合でも構わない^{*1}。大文字と小文字は区別されるので、`\foo` と `\F00` は全く別の命令となる。

但し、`\foo_bar` や `\a4` のような記号・数字を含む命令は通常定義することはできない。例外として、句読点・括弧などの記号類や数字 1 文字だけからなる命令は定義することができる。例えば、`\3` という命令は定義することができる。しかし、`\33` や `\3K` や `\K3` は定義することはできない。

`\foo` や `\命令` のようなアルファベットや漢字の命令では、直後の半角空白は区切りの役割をするだけだが、`\3` のような数字・記号 1 文字の命令では、直後の半角空白は空白として出力される。

同じ名前の命令が既に存在する場合は `\newcommand` は使えない。既に存在する命令の定義を変更するには `\newcommand` の代わりに `\renewcommand` を用いる。`\renewcommand` を用いれば、自分で定義した命令でも、`LATEX` で定義されている命令でも定義を変更することができる。

4.5 自前の環境

`\begin{quote} ... \end{quote}` のような、`\begin` で始まり `\end` で終わる命令を環境 (environment) ということは前述した。環境も自前で定義することができる。環境を定義するための命令は `\newenvironment` である。これは、

```
\newenvironment{なにになに}{かくかく}{しかじか}
```

の形で用いる。これで、

```
\begin{なにになに} → {かくかく}
\end{なにになに}   → しかじか}
```

という意味となる。もう少し実用的な例として、先に定義した `\記` という命令を環境に拡張してみる。

```
\newenvironment{記}
{ \begin{center} 記 \end{center} \begin{description} }
{ \end{description} }
```

このように宣言しておく、その後では、

```
\begin{記} = { \begin{center} 記 \end{center} \begin{description} }
\end{記} = \end{description}
```

という等式が成り立つ。これで、

^{*1} 実際には、その命令の機能を類推することができる名前を付けるべきである。

```
\begin{記}
\item[日時] 2013 年 02 月 02 日 午後 2 時
\item[場所] 第 2 会議室
\end{記}
```

などと記述することができる。

尚、既に存在する命令と同じ名前の環境を定義することはできない。すなわち、先の例において `\newcommand{\記}{...}` のように命令を定義してあると `\newenvironment{記}{...}{...}` という定義は行うことができない。既に存在する定義を上書きしたい場合は `\newenvironment` の代わりに `\renewenvironment` という命令を用いる。

4.6 引数をとるマクロ

ゴシック体で `ほげほげ` と出力するためには、`\textgt{ほげほげ}` と記述する。このような命令の直後の `{}` で囲んだ部分を、その命令の引数 (argument) という。

引数を付けて使う命令のことを「引数をとる命令」という。引数をとる命令も `\newcommand` や `\renewcommand` で定義することができる。例えば、小さいゴシック体で出力する命令 `\sg` を定義してみる。

```
\newcommand{\sg}[1]{\textgt{\small #1}}
```

使うときは `\sg{ほげほげ}` のようにする。

もう 1 つの例として、ア のように、小さいゴシック体を幅 1cm の長方形で囲む命令 `\f{...}` を定義してみる。

```
\newcommand{\f}[1]{\framebox[1cm]{\textgt{\small #1}}}
```

ここで用いた `\framebox[1cm]{何々}` は、幅 1cm の枠で囲んで「何々」を出力する命令である。これで、

大化の改新は `\f{ア}` 年である。 → 大化の改新は ア 年である。

となる。このように、引数をとる命令 (マクロ) は、

```
\newcommand{\命令の名前}[引数の個数]{定義内容}
```

の形式で定義する。命令の定義内の `#1` が引数で置き換えられる。引数がいくつも存在する場合は、`#1` が第 1 引数、`#2` が第 2 引数、… に置き換えられる。引数は 9 個まで使うことができる。

4.7 マクロの引数の制約

`\newcommand` でも `\newenvironment` でも同じことができる場合、どちらを用いたらよいだろうか？ 例えば、

```
\newcommand{\sg}[1]{\textgt{\small #1}}
これは\sg{ほげほげ}です。
```

とすると、

```
\newenvironment{sg}{\small}{}
これは\begin{sg}ほげほげ\end{sg}です。
```

とするのは、同じことのように見える。しかし、マクロの引数の中では `\verb` など一部の命令が使えないという制約が存在する。先程の `\sg` 命令 (マクロ) において、

```
\sg{冷汗 \verb|^_|}
```

としようとするエラーが発生する。一方、環境の方では、

```
\begin{sg}冷汗 \verb|^_| \end{sg}
```

としてもエラーは発生せず、正しく処理される。

`\section` など既存の命令の引数にも `\verb` は使うことができない。目的がタイプライタ体で出力するだけなら、代わりに `\texttt{...}` を使うことができる。

`\texttt{...}` でうまく出力することができない特殊文字は `\symbol{文字コード}` という命令で出力することができる。例えば、`\` (バックスラッシュ) は文字コードが 5C (16 進) なので、`\texttt{\symbol{"5C}}` とすることでタイプライタ体のバックスラッシュを出力することができる。

L^AT_EX 2_ε では `lrbox` 環境を併用することで `\verb` を命令の引数の中で使うことができる。但し、文字の大きさは状況に応じて変化しないため、次の例のように `\section` 中で使うなら `\Large` にしておく必要がある。

```
\newsavebox{\mybox}      % mybox という箱を作る
\begin{lrbox}{\mybox}    % mybox という箱の中に \Large\verb|\TeX| を入れる
  \Large\verb|\TeX|
\end{lrbox}
\section{\usebox{\mybox} コマンドについて}
```

4.8 ちょっと便利なマクロ

いくつかの便利な小物マクロを紹介しておく。これらの命令は L^AT_EX ではなく裸の T_EX や plain T_EX の知識に基づいている。本節で紹介する命令やその改良版は、T_EX Live に含まれる `okumacro` というパッケージに含まれている。

4.8.1 丸囲み文字

丸囲み文字①②③… は、JIS X 0208 ベースの符号化 (シフト JIS、EUC-JP、ISO-2022-JP) では機種依存文字となり Windows と Mac 間などで文字化けを起こす。これらは `utf` パッケージ、もしくは `otf` パッケージを用いれば、`\ajMaru` という命令で出力することができる。また、`pifont` パッケージでも利用可能である。

自前で丸印と数字を合成して ① ② … を出力するためには、次のような命令を定義しておき `\MARU{1}` `\MARU{2}` … とする。

```
\newcommand{\MARU}[1]
  {\ooalign{\hfil#1\/\hfil\crrc\raise.167ex\hbox{\mathhexbox20D}}{}}
```

4.8.2 時候の挨拶

次の命令 `\挨拶` は「拝啓 陽春の候、ますますご清栄のこととお喜び申し上げます」のような挨拶をその月に合わせて出力する。`\month` には L^AT_EX で処理した月 (1~12) が格納される。`\ifcase` は数値 0、1、2、… によって条件分岐する命令である。

```
\def\挨拶{\noindent 拝啓\hskip+1zw}\ifcase\month\or
  厳寒\or 春寒\or 早春\or 陽春\or 新緑\or 向暑\or
  猛暑\or 残暑\or 初秋\or 中秋\or 晩秋\or 初冬\fi
  の候、ますますご清栄のこととお喜び申し上げます。}
```

4.8.3 曜日

今日は曜_{アット}曜日です と記述すると「今日は金曜日です」のように、L^AT_EX で処理した日の曜日を出力する。この命令の定義には Zeller の公式^{*2} というものを用いている。

`\@tempcnta`、`\@tempcntb` は L^AT_EX 内部でよく用いられる一時的な (temporary な) 利用のためのカウンタ (整数値の変数) である。このように `@` (`at` マーク) を含む名前は、パッケージファイル中では自由に使うことができるが、L^AT_EX 文書ファイル中で使うには `\makeatletter` と `\makeatother` で囲っておかなければならない。

^{*2} 『C 言語による最新アルゴリズム辞典』(技術評論社、1991 年)、『Java によるアルゴリズム辞典』(技術評論社、2003 年) 参照


```

\makeatletter
\newcommand{\曜}{\@tempcnta=\year \@tempcntb=\month
\ifnum \@tempcntb < 3
\advance \@tempcnta by -1
\advance \@tempcntb by 12
\fi
\multiply \@tempcntb by 13
\advance \@tempcntb by 8
\divide \@tempcntb by 5
\advance \@tempcntb by \@tempcnta
\divide \@tempcnta by 4
\advance \@tempcntb by \@tempcnta
\divide \@tempcnta by 25
\advance \@tempcntb by -\@tempcnta
\divide \@tempcnta by 4
\advance \@tempcntb by \@tempcnta
\advance \@tempcntb by \day
\@tempcnta = \@tempcntb
\divide \@tempcntb by 7
\multiply \@tempcntb by 7
\advance \@tempcnta by -\@tempcntb
\ifcase \@tempcnta 日\or 月\or 火\or 水\or 木\or 金\or 土\or
\fi}}
\makeatother

```

`\advance`、`\multiply`、`\divide` は和・積・商の値を求める命令である。`\ifnum` は数値の比較を行う命令である。

4.8.4 均等割り

均等割りの命令は次のようにして定義することができる。但し、使用できるは和文だけである。

```

\newcommand{\kintou}[2]{%
\leavemode
\hbox to #1{%
\kanjiskip=0pt plus 1fill minus 1fill \xkanjiskip=\kanjiskip #2
}
}

```

`\leavemode` は $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ の「垂直モード vertical mode を抜ける」ための命令で、`\hbox`（水平ボックス：horizontal box）を段落の最初でも使えるようにするオマジナイである。`\leavemode \hbox to 5zw {ほげ}` で「ほげ」という文字を 5zw の箱の中に書き込むという意味になるが、均等割りにするために和文文字間のアキ `\kanjiskip`、和文・欧文文字間のアキ `\xkanjiskip` を標準で 0 ポイント、それにプラスマイナス 1 [fill]（いくらでも伸びる値）に設定している。

この命令は次のように用いる。

入力 | 5 文字の幅に 4 文字を `\kintou{5zw}{均等割り}` する

出力 | 5 文字の幅に 4 文字を均 等 割 りする

この命令を使うと、3 文字分の幅に 4 文字を無理やり詰め込むこともできる。

4.8.5 圈点

けんでん
圈点を打つマクロである。`\kenten{強調}` とすると 強 調 となる。`\. 強 \. 調`（強 調）として打つ圈点より大きくなる。

以下の定義では、垂直モード（段落が始まっていない状態）なら（\ifvmode）段落を開始し（\leavevmode）、そうでなければ\kanjiskip だけのアキを入れる。「・」といくらでも縮むグルー \hss とを入れた幅ゼロ（\z@ は 0pt と同義）の箱を箱レジスタ 1 に代入し、その高さ（\ht）を 0.63zw にして、\@kenten に制御を移す。 \@kenten は tall recursion という方法を用いて各文字をループし、実際に圈点を振る。

```
\makeatletter
\def\kenten#1{%
  \ifvmode\leavevmode\else\hskip\kanjiskip\fi
  \setbox1 = \hbox to \z@{·\hss}% 「・」は全角の中黒
  \ht1 = .63zw
  \@kenten#1\end}
\def\@kenten#1{%
  \ifx#1\end \let\next = \relax \else
    \raise.63zw\copy1\nobreak #1\hskip\kanjiskip\relax
    \let\next = \@kenten
  \fi\next}
\makeatother
```

4.8.6 倍角ダッシュ

半角のマイナスを“--”のように 2 つ連続して入力すると—のような欧文のエヌダッシュが出力される。また、“---”のように 3 つ連続で入力すると、――のような欧文のエムダッシュが出力される。

和文の——のような倍角ダッシュ（2 倍ダシ）は JIS コード 213D の—を 2 つ並べても出力することができるが、フォントによっては隙間が空いてしまうことがある。次のようにすれば隙間を無くすることができる。

```
\def\――{\kern-.5zw—\kern-.5zw—}
```

これで海\――山 とすると「海——山」となる。同様のマクロが okumacro パッケージで定義されている。

第 5 章

数式の基礎

TeX を開発した Knuth 教授は数学者でもあり、数式関係の TeX の機能は抜群である。本章では、LaTeX の標準機能による数式の書き方を説明し、次章では amsmath パッケージによる高度な数式の書き方を説明する。

5.1 数式の基本

例えば、

アインシュタインは $E = mc^2$ と言った。

と出力するには、LaTeX では、

```
\documentclass{jsarticle}
\begin{document}
アインシュタインは  $E=mc^2$  と言った。
\end{document}
```

と記述する。この \$ (ドル記号) で挟まれた部分が数式である。

E や m や c のようなアルファベットが、数式中では *E* や *m* や *c* のような数式用フォント (イタリック体) で出力される。また、^ (ハット) に続く文字が「上付き文字」 (superscript) になる。

数式にはもう 1 種類ある。

アインシュタインは
$$E = mc^2$$

と言った。

のような別行立ての数式、あるいは別行数式 (displayed formula, display math) と呼ばれるものである。これは、

```
\documentclass{jsarticle}
\begin{document}
アインシュタインは \[ E=mc^2 \] と言った。
\end{document}
```

のように \[...\] でサンドイッチする。

5.2 数式用のフォント

LaTeX では標準で Computer Modern フォントが使われる。本文を Times 系のフォントにするには、プリアンブルに、

```
\usepackage{newtxtext}
```

と書けばよいのだが、本文だけでなく数式も Times 系のフォントにするためには、

```
\usepackage{newtxtext,newtxmath}
```

と記述する。

同様に、本文・数式が Palatino フォントにするためには、プリアンブルに、

```
\usepackage{newpxtext,newpxmath}
```

と記述する。あるいは、少々デザインは異なるが、

```
\usepackage{mathpazo}
```

と記述しても本文・数式が Palatino フォントになる。

5.3 数式の書き方の詳細

数式モードでは、次のように半角空白を入れても出力は変わらない。

```
$a + (- b) = a - b$ →  $a + (-b) = a - b$ 
```

```
$a+(-b)=a-b$ →  $a + (-b) = a - b$ 
```

また、数式用の書体 *xyz* (`xyz`) は、本文用イタリック体 *xyz* (*xyz*) とは微妙に異なることがある。特に、文字間の間隔が異なるため、本文のイタリック体の代わりに数式モードを用いると次のようにおかしいことになる。

(正) `\textit{difference}` → *difference*

(誤) `$difference$` → *difference*

更に、約物（句読点や括弧類）以外の文字と数式の間には、半角の空白「`\`」を入れるのが慣習となっている。

方程式「`$f(x)=0$`」の解 ← 通常は半角空白を入れる。

方程式「`$f(x)=0$`、`$g(x)=0$`」の解 ← 約物との間には空白を入れない。

5.4 上付き文字・下付き文字

累乗（一般に上付き文字） x^2 は `x^2` と書く。しかし、 x^{10} と出力するつもりで `x^{10}` と書くと `x^{10}` となってしまう。ここは、`x^{10}` と書かなければならない。また、 a_n のような添字（一般に下付き文字）を付けるには `a_n` のように書く。これも添字が 2 文字以上なら `a_{ij}` のようなグループ化が必要となる。

いくつかの複雑な例を挙げておく。

<code>\$2^{2^2}\$</code>	→ 2^2
<code>\$a^{k_{ij}}\$</code>	→ $a^{k_{ij}}$
<code>\$\mathrm{^{137m}Ba}\$</code>	→ $^{137\text{m}}\text{Ba}$
<code>\$\mathrm{^{137}_{\hphantom{0}55}Cs}\$</code>	→ $^{137}_{55}\text{Cs}$
<code>\$R^{\rho}_{\sigma\mu\nu}\$</code>	→ $R^{\rho}_{\sigma\mu\nu}$
<code>\$R^{\rho}_{\hphantom{\rho}\sigma\mu\nu}\$</code>	→ $R^{\rho}_{\sigma\mu\nu}$

5.5 別行立ての数式

前述したように、`\[...\]` で囲めば別行立ての数式となる。無指定では、別行立ての数式は行の中央に配置される。左端から一定の距離に配置するには、ドキュメントクラスのオプションに `fleqn` を指定する。すなわち、文書ファイルの最初の行を、

```
\documentclass[fleqn]{...}
```

のようにする。左端からの距離を全角 1 文字分にするには、更に、

```
\setlength{\mathindent}{1zw}
```

のように指定する。本稿ではこの指定を行っている。

数式番号を付けるには、`\[...\]` の代わりに `equation` 環境を用いて、

```
別行とは……とは、
\begin{equation}
  y = ax^2 + bx + c
\end{equation}
のように……
```

のように記述する。右側に数式番号が

$$y = ax^2 + bx + c \quad (1)$$

のように自動的に出力される。章に分かれた本 (jsbook ドキュメントクラス等) の場合は、第 3 章の最初の数式なら (3.1) のようになる。

数式番号は標準では右側に付加される。左側に付けたい場合は、

```
\documentclass[leqno]{...}
```

のように leqno オプションを付加する。

5.6 和・積分

和の記号 \sum を出力する命令は `\sum` である。

$$\sum_{k=1}^n a_k = a_1 + a_2 + a_3 + \cdots + a_n$$

と出力するには、

```
\[ \sum_{k=1}^n {a_{k}} = a_{1}+a_{2}+a_{3}+\cdots+a_{n} \]
```

と記述する。この `_{k=1}` や `^n` は上下の添字を付ける命令と同じだが、 \sum のような特殊な記号については、別行立ての数式として使ったときに限り、添字は記号の上下に付加される。

同じ `\sum` でも、本文中で、

```
和  $\sum_{k=1}^n {a_{k}}$  を求めよ。
```

と書くと、

```
和  $\sum_{k=1}^n a_k$  を求めよ。
```

のよう上下限の付き方が変わる。本文中で $\sum_{k=1}^n a_k$ のように別行立て数式のような和記号を使いたい場合には、

```

$$\sum_{k=1}^n {a_{k}}$$

```

のように `\displaystyle` という命令を用いる。逆に、別行立ての数式で本文中のような記号 $\sum_{k=1}^n a_k$ を出力するには、

```
\[ \textstyle \sum_{k=1}^n {a_{k}} \]
```

のように `\textstyle` という命令を用いる。

`\displaystyle`、`\textstyle` を使うと、和記号・積分記号・分数の大きさ、添字の位置などが変わる。大きさを変えずに、添字の付き方だけを変えたい場合には `\limits`、`\nolimits` を用いる。

$$\begin{aligned} \[\textstyle \sum_{k=1}^n \] &\rightarrow \sum_{k=1}^n \\ \[\textstyle \sum\limits_{k=1}^n \] &\rightarrow \sum_{k=1}^n \\ \[\sum\nolimits_{k=1}^n \] &\rightarrow \sum_{k=1}^n \end{aligned}$$

積分記号 \int は `\int` という命令で出力する。これも和記号と同様に上下限を `_` `^` で指定する。例えば、`\int_{0}^1` は、別行立て数式では \int_0^1 、本文中では \int_0^1 のようになる。

5.7 分数

分数 (fraction) を書く命令は `\frac{分子}{分母}` である。例えば、

```
\[ y=\frac{1+x}{1-x} \]
```

 と書けば、

$$y = \frac{1+x}{1-x}$$

と出力される。本文中で $y = \frac{1+x}{1-x}$ と書けば $y = \frac{1+x}{1-x}$ のように小さめの字となる。しかし、これは $y = (1+x)/(1-x)$ と書いて $y = (1+x)/(1-x)$ とする方がよいスタイルであるとされている。どうしても $y = \frac{1+x}{1-x}$ のように大きい分数を本文中で使いたい場合は、

```

$$y = \frac{1+x}{1-x}$$

```

のように記述する。逆に、別行立ての数式を本文中の数式の形式にするには `\textstyle` を用いる。

なお、第 6 章で説明する `amsmath` パッケージには、大きい分数を出力する `\dfrac`、小さい分数を出力する `\tfrac` が定義されているので、そちらを使う方が便利である。

5.8 字間や高さの微調整

数式中の字間は多くの場合 `TeX` が正しく判断してくれる。例えば、 $\mathbf{x-y}$ と書いた際と $\mathbf{-y}$ と書いた際には $-$ と y の間隔は異なるのが正しいのだが、`TeX` は正しくこれを判断してくれる。しかし、`TeX` の判断には限界がある。例えば、 $\mathbf{f(x,y)dx dy}$ と書くと $\mathbf{f(x,y)dxdy}$ となってしまう、意味の上での区切りがわかりにくくなってしまう。このような場合、 $\mathbf{f(x,y) dx dy}$ のように若干の空きを入れるには $\mathbf{f(x,y)\,dx\,dy}$ のように `\,` を適宜挿入する。同様に、 $\mathbf{\sqrt{2}x}$ より $\mathbf{\sqrt{2}\,x}$ と書く方がよいだろう。

数式中に強制的にスペースを入れる命令は `\,` 以外にもいろいろ存在する。まず、`\quad` は本文に 10 ポイントの文字を使っているなら 10 ポイントの空きを入れる命令である。`\quad` は数式中でも本文中でも使用可能である。この `\quad` の他に、次の命令が用意されている。

`\qqad` `\quad` の 2 倍。
`\,` `\quad` の 3/18 ほど。
`\>` `\quad` の 4/18 ほど (数式モードのみ)。
`\;` `\quad` の 5/18 ほど (数式モードのみ)。
`\!` `\quad` の -3/18 ほど (数式モードのみ)。

上で「ほど」と書いたのは、状況に応じて若干伸び縮みするからである。`\>` は足し算の $+$ の両側の空き、`\;` は等号 $=$ の両側の空きに相当する。最後の `\!` は負の空き、すなわち後戻りを意味する。これ以外に、数式で `\,` を使うと、本文の半角スペース `\,` 相当の空きが挿入される (`\quad` の 1/3 ~ 1/4)。`\,` などの空白は $\mathbf{i,j}$ のように使ったときと $\mathbf{a_{i,j}}$ のように添字の中で使ったときとで、長さが変化する。

2 重積分 \iint は、単純に `\int\int` と書くと $\int\int$ のように積分記号の間隔が広くなりすぎてしまう。`\!` を 2 ~ 3 個挿めばうまくいく。但し、これらは第 6 章で説明する `amsmath` パッケージで定義されている `\iint` という命令を使った方が簡単である。3 重積分 `\iiint` などと同様である。`newtxmath`、`newpmath` パッケージでも `\iint`、`\iiint` などが定義されている。

文字の高さも微調整するとよい場合がある。例えば、

```

$$\sqrt{g} + \sqrt{h}$$

```

と書くと、 $\sqrt{g} + \sqrt{h}$ のように根号 (ルート) の高さが揃いになるので、`\mathstrut` という命令を用いて、

```

$$\sqrt{\mathstrut{g}} + \sqrt{\mathstrut{h}}$$

```

のように書けば、 $\sqrt{g} + \sqrt{h}$ のように多少ましになるが、フォントによってはこれでも揃いになる。更に凝った方法は第 6 章で紹介する。

5.9 式の参照

`LATEX` は数式に自動的に番号を振ってくれるが、書き手は数式を番号ではなく適当な名前で管理する方が便利である。

例えば、

$$E = mc^2 \quad (12)$$

という数式があったとする。この数式の番号は (12) だが、追加・削除したり順番を変えたりすると番号は変わってしまう。そこで、この数式に例えば eq:Einstein という名前を付けて、この名前で管理すると便利である。それには `\label` という命令を用いて、

```
\begin{equation}
  E = mc^2 \label{eq:Einstein}
\end{equation}
```

と書いておく。この数式番号を参照したいときには、命令 `\ref` を用いる。また、数式のページを参照したいときには、命令 `\pageref` を用いる。例えば、

```
\pageref{eq:Einstein} ページの式 (\ref{eq:Einstein}) によれば...
```

とすれば “89 ページの式 (12) によれば...” のように出力される。参照する側とされる側のどちらが先にあっても構わない。但し、このような参照機能を用いる際には、文書ファイルを \LaTeX で少なくとも 2 回処理することが必要となる。例えば、foo.tex という文書ファイルなら、 \LaTeX は 1 回目の実行で補助ファイル foo.aux に参照表を書き出し、2 回目の実行で foo.aux から参照番号を拾い出す。

5.10 括弧類

括弧類（区切り文字：delimiters）には次のような種類が存在する。まず、左右の区別があるものは以下の通りである。

入力	出力	入力	出力	入力	出力
(x)	(x)	\{ x \}	{x}	\lceil x \rceil	[x]
[x]	[x]	\lfloor x \rfloor	[x]	\langle x \rangle	\rangle

次は左右の区別のないものである。

入力	出力	入力	出力	入力	出力
/	/	\uparrow	↑	\updownarrow	↕
\backslash	\	\Uparrow	↗	\Updownarrow	↕
		\downarrow	↓		
\		\Downarrow	⇓		

これらを少し大きくするには、前に `\big` を付ける。但し、左括弧の類は `\bigl`、右括弧の類は `\bigr` とする方がバランスがよくなる。また、2 項関係を表す記号を大きくするには `\bigm` を付ける。

$$\begin{aligned} \text{\texttt{\$}\bigl| |x| + |y| \bigr|} &\rightarrow ||x| + |y|| \\ \text{\texttt{\$}\bigl\lfloor \sqrt{X} \bigr\rfloor} &\rightarrow \lfloor \sqrt{X} \rfloor \\ \text{\texttt{\$}\bigl(x - f(x) \bigr)\bigm/ \bigl(x + f(x) \bigr)} &\rightarrow (x - f(x)) / (x + f(x)) \end{aligned}$$

`\big` より大きくするには `\Big`、`\bigg`、`\Bigg` を付ける。`\Bigl`、`\biggl`、`\Biggl`、`\Bigr`、`\biggr`、`\Biggr`、`\Bigm`、`\biggm`、`\Biggm` についても同様である。

次のように `\left`、`\right` を用いれば、区切りの大きさが自動的に選択される。

$$\begin{aligned} \text{\texttt{\$}\left(x \right)} &\rightarrow (x) \\ \text{\texttt{\$}\left(x^2 \right)} &\rightarrow (x^2) \\ \text{\texttt{\$}\displaystyle\left(\frac{A}{B} \right)} &\rightarrow \left(\frac{A}{B}\right) \end{aligned}$$

`\left` と `\right` は必ずペアで用いる。片方だけ括弧を付けたい場合には、

$$\text{\texttt{\$}\left(x^2 \right.} \rightarrow (x^2$$

のように、もう片方はピリオド (.) にしておく。この場合のピリオドは出力されない。

あまり古くないシステム (ϵ -TeX 拡張されたもの) では、次のような `\middle` という命令も定義されている。

$$\displaystyle{\left(\frac{A}{B}\middle/\frac{C}{D}\right)} \rightarrow \left(\frac{A}{B}\middle/\frac{C}{D}\right)$$

5.11 ギリシア文字

数式モードで扱うことができるギリシア文字を挙げておく。

小文字は英語名の前に `\` を付けるだけである。但し、*o* (omicron) だけは英語のオーと同じなので特に用意されていない。

入力	出力	入力	出力	入力	出力	入力	出力
<code>\alpha</code>	α	<code>\eta</code>	η	<code>\nu</code>	ν	<code>\tau</code>	τ
<code>\beta</code>	β	<code>\theta</code>	θ	<code>\xi</code>	ξ	<code>\upsilon</code>	υ
<code>\gamma</code>	γ	<code>\iota</code>	ι	<code>o</code>	o	<code>\phi</code>	ϕ
<code>\delta</code>	δ	<code>\kappa</code>	κ	<code>\pi</code>	π	<code>\chi</code>	χ
<code>\epsilon</code>	ϵ	<code>\lambda</code>	λ	<code>\rho</code>	ρ	<code>\psi</code>	ψ
<code>\zeta</code>	ζ	<code>\mu</code>	μ	<code>\sigma</code>	σ	<code>\omega</code>	ω

一部のギリシア文字 (小文字) には変体文字が用意されている。

入力	出力	入力	出力	入力	出力
<code>\varepsilon</code>	ε	<code>\varpi</code>	ϖ	<code>\varsigma</code>	ς
<code>\vartheta</code>	ϑ	<code>\varrho</code>	ϱ	<code>\varphi</code>	φ

大文字は、次の 11 通り以外は英語のアルファベットの大文字と同じである。

入力	出力	入力	出力	入力	出力	入力	出力
<code>\Gamma</code>	Γ	<code>\Lambda</code>	Λ	<code>\Sigma</code>	Σ	<code>\Psi</code>	Ψ
<code>\Delta</code>	Δ	<code>\Xi</code>	Ξ	<code>\Upsilon</code>	Υ	<code>\Omega</code>	Ω
<code>\Theta</code>	Θ	<code>\Pi</code>	Π	<code>\Phi</code>	Φ		

数式中のギリシア文字は、慣習に従って小文字だけ斜体となる。大文字も斜体にしたい場合は、第 6 章の `amsmath` パッケージを用いて、例えば `\varDelta` と書けば Δ が出力される。

小文字を立体にしたい場合は一部の文字については `textcomp` パッケージで出力することができる (例: `\textmu` で μ)。もっと広範囲に立体を使いたい場合は `\usepackage{upgreek}` として `\up...` という命令を用いる。例えば、 μ の立体は `\upmu` (μ) である。

5.12 筆記体

大文字の筆記体には数式モードで `mathcal` という命令で記述する。標準では次のようなフォントになる。

$$\mathcal{ABCDEFGHIJKLMNOPQRSTUVWXYZ} \rightarrow \mathcal{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$$

物理でハミルトニアンやラグランジュを \mathcal{H} や \mathcal{L} のようにかっこよく書くには、プリアンブルに、

```
\usepackage{mathrsfs}
```

と書いておき、`\mathscr{H}` のように記述する。このフォントは RSFS (Ralph Smith's Formal Script) と言う。

5.13 2 項演算子

2 項演算子とは足し算、引き算の記号の仲間である。各々を単独で用いたり、 $\pm a$ (`\pm a`) のように単項演算子として用いたりすることもできる。

入力	出力	入力	出力	入力	出力	入力	出力
+	+	<code>\circ</code>	○	<code>\vee</code>	∨	<code>\oplus</code>	⊕
-	-	<code>\bullet</code>	•	<code>\wedge</code>	∧	<code>\ominus</code>	⊖
<code>\pm</code>	±	<code>\cdot</code>	·	<code>\setminus</code>	∖	<code>\otimes</code>	⊗
<code>\mp</code>	∓	<code>\cap</code>	∩	<code>\wr</code>	⌢	<code>\oslash</code>	⊘
<code>\times</code>	×	<code>\cup</code>	∪	<code>\diamond</code>	◇	<code>\odot</code>	⊙
<code>\div</code>	÷	<code>\uplus</code>	⊕	<code>\bigtriangleup</code>	△	<code>\bigcirc</code>	◯
*	*	<code>\sqcap</code>	⊓	<code>\bigtriangledown</code>	▽	<code>\dagger</code>	†
<code>\ast</code>	*	<code>\sqcup</code>	⊔	<code>\triangleleft</code>	◁	<code>\ddagger</code>	‡
<code>\star</code>	★			<code>\triangleright</code>	▷	<code>\amalg</code>	⊔

5.14 関係演算子

関係演算子とは等号 =、不等号 <、> の仲間である。まず、左向きと右向きの区別のあるものを挙げる。

入力	出力	入力	出力	入力	出力	入力	出力
<	<	>	>	<code>\supset</code>	⊃	<code>\supseteq</code>	⊇
<code>\leq</code>	≤	<code>\geq</code>	≥	<code>\subseteq</code>	⊆	<code>\supseteq</code>	⊇
<code>\prec</code>	<	<code>\succ</code>	>	<code>\sqsubseteq</code>	⊆	<code>\sqsupseteq</code>	⊇
<code>\preceq</code>	≤	<code>\succeq</code>	≥	<code>\vdash</code>	⊢	<code>\dashv</code>	⊥
<code>\ll</code>	≪	<code>\gg</code>	≫	<code>\in</code>	∈	<code>\ni</code>	∋
				<code>\notin</code>	∉		

次は、左向きと右向きの区別がないものである。

入力	出力	入力	出力	入力	出力	入力	出力
=	=	<code>\sim</code>	∼	<code>\propto</code>	∝	<code>\parallel</code>	∥
<code>\equiv</code>	≡	<code>\simeq</code>	≈	<code>\models</code>	⊨	<code>\bowtie</code>	⋈
<code>\neq</code>	≠	<code>\asymp</code>	≈	<code>\perp</code>	⊥	<code>\smile</code>	∪
<code>\doteq</code>	≐	<code>\approx</code>	≈	<code>\mid</code>		<code>\frown</code>	∩
		<code>\cong</code>	≅	:	:		

≐ などの記号は AMSFonts (第 6 章) の `\doteqdot` を用いる。斜線を重ねるには `\not` を冠する。

$$\begin{array}{cc} \text{入力} & \text{出力} \\ \hline \$x \not\equiv y$ & $x \neq y$ \end{array}$$

`\{x \mid x \leq 1\}` とすると $\{x|x \leq 1\}$ のようにバランスが悪くなる。以下のように記述するのが望ましい。

$$\{x \mid x \leq 1\} \rightarrow \{x|x \leq 1\}$$

`\mid` を少し大きくしたい場合は `\bigm\mid` ではなく `\big|` とする。両側のスペースも入る。

一方で、`\middle|` とすれば (ε-TeX 拡張された新しい TeX や pTeX では) 括弧に合わせて大きくなるが、両側に \; と同じ幅のスペースが入らない。これを解決する方法の 1 つは、

```
\newcommand{\relmiddle}[1]{\mathrel{\middle#1\mathrel{}}}
```

と定義しておき、

$$\displaystyle \left\{ x \relmiddle| x \leq \frac{1}{2} \right\} \rightarrow \left\{ x \middle| x \leq \frac{1}{2} \right\} \text{ とする。}$$

コロンも数式の中では関係演算子扱いとなり、両側に \; 相当の空白が入る。このような空白を出力したくない場合には、`\:` のように中括弧で囲む。

次のような場合に、適切な空白を入れてくれる `\colon` という命令も用意されている。

$$\begin{aligned} \$f : \quad A \to B\$ &\rightarrow f : A \rightarrow B \\ \$f{:}\backslash \quad A \to B\$ &\rightarrow f{:} A \rightarrow B \\ \$f\backslash\colon A \to B\$ &\rightarrow f : A \rightarrow B \end{aligned}$$

5.15 矢印

矢印は、括弧類で挙げたもの以外に、次のものが用意されている。

入力	出力	入力	出力
<code>\leftarrow</code> (gets)	\leftarrow	<code>\longleftarrow</code>	\longleftarrow
<code>\Leftarrow</code>	\Leftarrow	<code>\Longleftarrow</code>	\Longleftarrow
<code>\rightharpoonup</code> (<code>\to</code>)	\rightarrow	<code>\longrightharpoonup</code>	\rightarrow
<code>\Rightarrow</code>	\Rightarrow	<code>\Longrightarrow</code>	\Longrightarrow
<code>\leftrightharpoonup</code>	\leftrightarrow	<code>\longleftrightarrow</code>	\longleftrightarrow
<code>\Leftrightarrow</code>	\Leftrightarrow	<code>\Longleftrightarrow</code>	\Longleftrightarrow
<code>\mapsto</code>	\mapsto	<code>\longmapsto</code>	\mapsto
<code>\hookrightarrow</code>	\hookrightarrow	<code>\hookrightarrow</code>	\hookrightarrow
<code>\leftharpoonup</code>	\leftarrow	<code>\rightharpoonup</code>	\rightarrow
<code>\leftharpoondown</code>	\leftarrow	<code>\rightharpoondown</code>	\rightarrow

なお、`\iff` も `\Longleftrightarrow` と同じ記号 \iff を出力するが、両側の空きは `\iff` の方が広くなる。

入力	出力	入力	出力	入力	出力
<code>\nearrow</code>	\nearrow	<code>\swarrow</code>	\swarrow	<code>\rightleftharpoons</code>	\rightleftharpoons
<code>\searrow</code>	\searrow	<code>\nwarrow</code>	\nwarrow		

5.16 雑記号

上記以外に、以外に次のようなものが用意されている。

入力	出力	入力	出力	入力	出力
<code>\aleph</code>	\aleph	<code>\prime</code>	\prime	<code>\neg</code> (<code>\lnot</code>)	\neg
<code>\hbar</code>	\hbar	<code>\emptyset</code>	\emptyset	<code>\flat</code>	\flat
<code>\imath</code>	\imath	<code>\nabla</code>	∇	<code>\natural</code>	\natural
<code>\jmath</code>	\jmath	<code>\surd</code>	\surd	<code>\sharp</code>	\sharp
<code>\ell</code>	ℓ	<code>\top</code>	\top	<code>\clubsuit</code>	\clubsuit
<code>\wp</code>	\wp	<code>\bot</code>	\bot	<code>\diamondsuit</code>	\diamondsuit
<code>\Re</code>	\Re	<code>\angle</code>	\angle	<code>\heartsuit</code>	\heartsuit
<code>\Im</code>	\Im	<code>\triangle</code>	\triangle	<code>\spadesuit</code>	\spadesuit
<code>\partial</code>	∂	<code>\forall</code>	\forall		
<code>\infty</code>	∞	<code>\exists</code>	\exists		

5.17 latexsym で定義されている文字

L^AT_EX 2.09 では標準で使えた次の記号が、L^AT_EX 2_ε ではオプションとなっている。これらの記号を使うには、プリアンブルに `\usepackage{latexsym}` と書いておく。なお、ほぼ同じ記号が AMSFonts (第 6 章参照) にも用意されている。次の 3 つの表において、括弧内は該当する AMSFonts (amssymb パッケージでの名前) である。今後は、この AMSFonts を用いた方がいいだろう。

まず、2 項演算子から挙げておく。

入力	出力	入力	出力
<code>\lhd (\vartriangleleft)</code>	\triangleleft	<code>\unlhd (\trianglelefteq)</code>	\trianglelefteq
<code>\rhd (\vartriangleright)</code>	\triangleright	<code>\unrhd (\trianglerighteq)</code>	\trianglerighteq

関係演算子は以下の通りである。

入力	出力	入力	出力
<code>\sqsubset (\sqsubset)</code>	\sqsubset	<code>\sqsupset (\sqsupset)</code>	\sqsupset
<code>\Join (\bowtie)</code>	\bowtie		

それ以外の記号は以下の通りである。

入力	出力	入力	出力
<code>\leadsto (\rightsquigarrow)</code>	\rightsquigarrow	<code>\Diamond (\lozenge)</code>	\lozenge
<code>\Box (\square)</code>	\square	<code>\mho (\mho)</code>	\mho

5.18 mathcomp で定義されている文字

`textcomp` パッケージの記号を数式モードで出力するための `mathcomp` パッケージによる記号である。デフォルトのフォントは Computer Modern Roman だが、例えば `\usepackage[ppl]{mathcomp}` とすれば、Palatino (ppl) フォントになる。

入力	出力	入力	出力
<code>\tcohm</code>	Ω	<code>\tcmu</code>	μ
<code>\tcdegree</code>	$^{\circ}$	<code>\tccelsius</code>	$^{\circ}\text{C}$
<code>\tcoerthousand</code>	‰		

45° を昔は `$45^{\circ}` と書いていたが、今はテキストモードでは `textcomp` の `\textdegree`、数式モードでは `mathcomp` の `\tcdegree` 使うべきである。尚、 $^{\circ}\text{C}$ は単位記号なので、数字との間に若干のスペース `\,` を入れるのが正しいとされている。

5.19 大きな記号

和・積分の類である。

入力	出力	入力	出力	入力	出力
<code>\sum</code>	Σ	<code>\bigcap</code>	\bigcap	<code>\bigodot</code>	\bigodot
<code>\prod</code>	\prod	<code>\bigcup</code>	\bigcup	<code>\bigotimes</code>	\bigotimes
<code>\coprod</code>	\coprod	<code>\bigsqcup</code>	\bigsqcup	<code>\bigoplus</code>	\bigoplus
<code>\int</code>	\int	<code>\bigvee</code>	\bigvee	<code>\biguplus</code>	\biguplus
<code>\oint</code>	\oint	<code>\bigwedge</code>	\bigwedge		

5.20 log 型関数と mod

`log` のような関数はイタリック体ではなく、アップライト体（立体）で書く。`log x` と出力するつもりで `$log x$` と書くと `log x` のような見苦しい出力になってしまう。正しくは `\log` という命令を用いて `$\log x$` と書く。

この種の関数は多数存在する。

入力	出力	入力	出力	入力	出力	入力	出力
<code>\arccos</code>	\arccos	<code>\csc</code>	\csc	<code>\ker</code>	\ker	<code>\min</code>	\min
<code>\arcsin</code>	\arcsin	<code>\deg</code>	\deg	<code>\lg</code>	\lg	<code>\Pr</code>	\Pr
<code>\arctan</code>	\arctan	<code>\det</code>	\det	<code>\lim</code>	\lim	<code>\sec</code>	\sec
<code>\arg</code>	\arg	<code>\dim</code>	\dim	<code>\liminf</code>	\liminf	<code>\sin</code>	\sin
<code>\cos</code>	\cos	<code>\exp</code>	\exp	<code>\limsup</code>	\limsup	<code>\sinh</code>	\sinh
<code>\cosh</code>	\cosh	<code>\gcd</code>	\gcd	<code>\ln</code>	\ln	<code>\sup</code>	\sup
<code>\cot</code>	\cot	<code>\hom</code>	\hom	<code>\log</code>	\log	<code>\tan</code>	\tan

\coth coth \inf inf \max max \tanh tanh

これらの演算子のうち上限・下限をとるものは `^` と `_` で指定する。

$$\begin{aligned} \text{\texttt{\$}\lim_{x \to \infty} f(x)\text{\texttt{\$}}} &\rightarrow \lim_{x \rightarrow \infty} f(x) \\ \text{\texttt{\$}\displaystyle\lim_{x \to \infty} f(x)\text{\texttt{\$}}} &\rightarrow \lim_{x \rightarrow \infty} f(x) \end{aligned}$$

log 型関数と似たものに次の 2 種類の `mod` がある。`\bmod` は 2 項 (binary) 演算子の `mod` である。`\pmod` は括弧付き (parenthesized) の `mod` である。

$$\begin{aligned} \text{\texttt{\$m \bmod n\text{\texttt{\$}}}} &\rightarrow m \bmod n \\ \text{\texttt{\$a \equiv b \pmod{n}\text{\texttt{\$}}}} &\rightarrow a \equiv b \pmod{n} \end{aligned}$$

第 6 章においてこの類の追加と、これに類似の命令を新たに定義する方法を説明する。

5.21 上下に付けるもの

数式モードだけで使うことができるアクセント記号である。

入力	出力	入力	出力	入力	出力	入力	出力
<code>\hat{a}</code>	\hat{a}	<code>\acute{a}</code>	\acute{a}	<code>\bar{a}</code>	\bar{a}	<code>\ddot{a}</code>	\ddot{a}
<code>\check{a}</code>	\check{a}	<code>\grave{a}</code>	\grave{a}	<code>\vec{a}</code>	\vec{a}		
<code>\breve{a}</code>	\breve{a}	<code>\tilde{a}</code>	\tilde{a}	<code>\dot{a}</code>	\dot{a}		

i 、 j にアクセント記号を付ける場合は、 i (`\imath`)、 j (`\jmath`) を用いて、例えば、 \tilde{i} (`\text{\texttt{\$}\tilde{\imath}\text{\texttt{\$}}}`) のようにする。

次は、伸縮自在の上下の棒の類である。

入力	出力	入力	出力
<code>\overline{x+y}</code>	$\overline{x+y}$	<code>\overbrace{x+y}</code>	$\overbrace{x+y}$
<code>\underline{x+y}</code>	$\underline{x+y}$	<code>\underbrace{x+y}</code>	$\underbrace{x+y}$
<code>\widehat{xyz}</code>	\widehat{xyz}	<code>\overrightarrow{\mathrm{OA}}</code>	$\overrightarrow{\mathrm{OA}}$
<code>\widetilde{xyz}</code>	\widetilde{xyz}	<code>\overleftarrow{\mathrm{OA}}</code>	$\overleftarrow{\mathrm{OA}}$

以上のうち `\widehat`、`\widetilde` はある程度しか伸びない。これらは重ねたり、入れ子にしたりすることができる。

`\overbrace`、`\underbrace` は和記号と同じような添字の付き方をする。

$$\begin{aligned} \text{\texttt{\$}\overbrace{a + \cdots + z}^{26}\text{\texttt{\$}}} &\rightarrow \overbrace{a + \cdots + z}^{26} \\ \text{\texttt{\$}\underbrace{a + \cdots + z}_{26}\text{\texttt{\$}}} &\rightarrow \underbrace{a + \cdots + z}_{26} \end{aligned}$$

記号の上に式を乗せるには、`\stackrel{式}{記号}` とする。出来上がった記号は関係演算子として扱われる。

$$\begin{aligned} \text{\texttt{\$}\stackrel{f}{\to}\text{\texttt{\$}}} &\rightarrow \overset{f}{\rightarrow} \\ \text{\texttt{\$}\stackrel{def}{=}\text{\texttt{\$}}} &\rightarrow \overset{\mathrm{def}}{=} \end{aligned}$$

ここで `\mathrm` は数式モード中の文字の書体をローマン体変える命令である。

5.22 数式の書体

数式中でも次のように書体を変えることができる。

入力	出力	入力	出力
<code>x + \mathrm{const}</code>	$x + \mathrm{const}$	<code>H(x)</code>	$H(x)$
<code>x\,,\,\mathrm{cm}^2</code>	$x\,,\,\mathrm{cm}^2$	<code>\mathrm{H}(x)</code>	$\mathrm{H}(x)$
<code>x_\mathrm{max}</code>	x_{max}	<code>\mathcal{H}(x)</code>	$\mathcal{H}(x)$

<code>\mathbf{x}</code>	\mathbf{x}	<code>\mathsf{H}(x)</code>	$H(x)$
<code>\mathit{diff}(x)</code>	$diff(x)$	<code>\mathtt{H}(x)</code>	$H(x)$

`diff` のような 1 語となったものは `\mathit{diff}` とする。単に、`$diff(x)$` とすると $diff(x)$ のような見苦しい出力になってしまう*1。

数式モード中で通常のローマン体の欧文を出力するには `\mathrm` を使う方法と `\textrm` を使う方法がある。`\mathrm` では数式用のローマン体フォントになり、`\textrm` では本文用のローマン体フォントになる。数式用フォントでは `\mathrm{for all}` と書いても空白は入らない。`\mathrm{for\ all}` 又は `\textrm{for all}` とする必要がある。

`\textrm` の代わりに `\mbox` を使うこともできる。しかし、これらでは添字中でも文字サイズが変わらないので、`amsmath` パッケージの `\text` を用いた方がいいだろう。

数式中で和文を使いたい場合は、`\textmc` (明朝体)、`\textgt` (ゴシック体) を用いるか、`amsmath` パッケージの `\text` を用いる。

数式中の太字 (ボールド体) は `\mathbf` で出力することができる。例えば、 α (太字の `\alpha`) なら `\bm{\alpha}`、 ∇ (太字の `\nabla`) なら `\bm{\nabla}` のようにして出力する。また、同じ太字を何度も使う必要がある場合は、

```
\bmdefine{\balpha}{\alpha}
```

のように定義すれば `\balpha` で α を出力することができるようになる。

`newtxmath` や `mathpazo` 等の数式フォントを変更するパッケージと併用する場合は、`\usepackage{bm}` はそれらのパッケージの後に記述する。

5.23 ISO/JIS の数式組版規則

昔からの数式の組版規則では、

- 数字はローマン体にする ($3.I4$ ではなく 3.14)。
- 複数文字からなる名前はローマン体にする ($\sin x$ ではなく $\sin x$)。
- 単位記号はローマン体にする ($3m$ ではなく $3\,m$)。

というルールになっている。また、 $\sin x$ の \sin と x の間や、 $3\,m$ の 3 と m の間には少しだけスペースを入れる。但し、 $\sin(a+b)$ のように括弧が来る場合はスペースを空けない。

例として『岩波数学辞典』第 3 版 (1985 年) や、全編 `TEX` で組まれた第 4 版 (2007 年) では、事象 E の確率 $P(E)$ 、事象 ϵ の起こる確率 $\Pr(\epsilon)$ のような書き方をしている。例外として、ユニタリ群 $U(n)$ に対して特殊ユニタリ群 $SU(n)$ のように、複数文字でも群や体の名前はイタリック体になっている。 $GF(n)$ 、 $Spin(n)$ も同様である。

しかし、ISO や JIS の流儀では、1 文字でも演算子や定数はローマン体にするになっている。例えば、数学流の微分の書き方 dx に対して、こちらの流儀では $\mathrm{d}x$ (`\mathrm{d}x`) のように立てて書く。また、自然対数の底などの定数も、こちらの流儀では e や π ではなく $e = 2.718\cdots$ 、 $\pi = 3.14\cdots$ のように立てて書く。どちらが正しいというわけではないので、投稿論文の決まりに従えばよい。

5.24 プログラムやアルゴリズムの組版

プログラムの組版は `verbatim` 環境を用いるのが最も簡単である。`verbatim` だけでは左寄せになるので、更に `quote` で囲むか、あるいは `jsverb` パッケージの `verbatim` 環境を用いる。後者の場合、`\setlength\verbatimleftmargin{3zw}` などのようにして左マージンを設定することができる。なお、タブコードは半角空白 1 個分として扱われるので、予め適当な個数の空白に置換しておく。

*1 これは $d \times i \times f \times f(x)$ と解釈されてしまうためである。

```

\begin{quote}
\setlength{\baselineskip}{12pt}
\verb'\begin{verbatim}
sum1 = sum2 = 0 ;
for (k = 1; k <= 10; k++) {
    sum1 += k ; sum2 += k * k ;
}
\end{verbatim}
\end{quote}

```

行送り (`\baselineskip`) を 12 ポイントにしたのは、和文の本文の行送り (15 ~ 16 ポイント) では行間が空きすぎになってしまうからである。

5.25 array 環境

`array` 環境は、第 8 章の `tabular` 環境とほぼ同じものだが、`tabular` 環境が本文中で使われるのに対して、`array` 環境は数式中で使われるというところが異なる。例えば、数式中で次のように記述すると、

```

\begin{equation}
\begin{array}{lcr}
abc & & abc & & abc \\
x & & y & & z
\end{array}
\end{equation}

```

以下の出力が得られる。

$$\begin{array}{lcr} abc & & abc & & abc \\ x & & y & & z \end{array}$$

このように、`\begin{array}` に続く中括弧 `{ }` の中に、各列の揃え方を並べる。中央揃えは `c`、左揃えは `l`、右揃えは `r` である。`array` 環境の本体では、各列は `&` で区切り、各行は `\\` で区切る。

従来の \LaTeX では、この `array` 環境に装飾を施して行列の類を出力していた。例えば、`\left(...\right)` で囲めば、括弧付きの行列となる。

```

\begin{equation}
A = \left(
\begin{array}{@{\,}\,}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{m1} & a_{m2} & \cdots & a_{mn}
\end{array}
\right)
\end{equation}

```

上のようになると、以下の出力が得られる。

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

行指定を `{cccc}` ではなく `{@{\,}\,}` としたのは、括弧と中身の間の空気を調節するためのトリックである。

@{} でいったん余分な空白を除いてから、\, でほんの少しの空白を入れ直している。

第 6 章で述べる amsmath パッケージを使えば、もっと素直に行列を書くことができる。ただ、array 環境は次のように行列の中に罫線を引くときに便利である。縦罫線は |、横罫線は \hline で描くことができる。

```
\begin{equation}
A = \left(
\begin{array}{@{\,}\,c|ccc@{\,}\,}
a_{11} & 0 & \cdots & 0 & \\
0 & a_{22} & \cdots & a_{2n} & \\
\vdots & \vdots & \ddots & \vdots & \\
0 & a_{m2} & \cdots & a_{mn} & \\
\end{array}
\right)
\end{equation}
```

これで以下の出力が得られる。

$$A = \left(\begin{array}{c|ccc} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{m2} & \cdots & a_{mn} \end{array} \right)$$

第 6 章

高度な数式

英国数学会（American Mathematical Society）が開発した `amsmath` パッケージと `AMSFonts` を用いた高度な数式の書き方を説明する。

6.1 `amsmath` と `AMSFonts`

Leslie Lamport が `LaTeX` を開発しているとき、米国数学回（American Mathematical Society）は Michael Spivak による `AMS-TeX` という数学論文記述に特化したマクロパッケージの開発を後押ししていた。`AMS-TeX` の数式記述能力は素晴らしいものだったが、世の中ではより使いやすい `LaTeX` が主流になってきて、`LaTeX` の枠内で `AMS-TeX` の数式記述能力を使いたいという要望が高まった。そこで、`LaTeX 3` プロジェクトチームの Frank Mittelbach と Rainer Schöpf が中心となって、`AMS-LaTeX` が開発された。

`AMS-LaTeX` のバージョン 1.1 までは `amstex.sty` というファイルが核となっていたが、これはまだ `AMS-TeX` 時代のしがらみをとどめており、`LaTeX` の流儀と一致しない部分が存在した。しかし、`AMS-LaTeX 1.2` で内容が一新され、`LaTeX 2ε` の枠内で使用することができるようになった。また、後述の `AMSFonts` パッケージとの役割分担も見直され、フォント関係の命令は `AMSFonts` に移された。1999 年 12 月の `AMS-LaTeX 2.0` からは“`AMS-LaTeX` = `amsmath` パッケージ + 米国数学会用クラスファイル群”という性質が確立された。

高度な数式を扱う `LaTeX` 文書では、`amsmath` と `AMSFonts` が標準的に用いられる。`AMSFonts` を使うためのパッケージは `amssymb` なので、プリアンブルには、

```
\usepackage{amsmath,amssymb}
```

と記述しておく。

`AMSFonts` の含まれるいろいろな数学記号の表を挙げておく。数学記号では、まず 2 項演算子に次のようなものが用意されている。

入力	出力	入力	出力	入力	出力
<code>\boxdot</code>	⊠	<code>\Cap</code>	⋈	<code>\circleddash</code>	⊖
<code>\boxplus</code>	⊕	<code>\curlywedge</code>	⋈	<code>\divideontimes</code>	⋈
<code>\boxtimes</code>	⊗	<code>\curlyvee</code>	⋈	<code>\lessdot</code>	⋈
<code>\centerdot</code>	⋅	<code>\leftthreetimes</code>	⋈	<code>\gtrdot</code>	⋈
<code>\boxminus</code>	⊖	<code>\rightthreetimes</code>	⋈	<code>\ltimes</code>	⋈
<code>\veebar</code>	⋈	<code>\dotplus</code>	+	<code>\rtimes</code>	⋈
<code>\barwedge</code>	⋈	<code>\intercal</code>	⋈	<code>\smallsetminus</code>	⋈
<code>\doublebarwedge</code>	⋈	<code>\circledcirc</code>	⊙		
<code>\Cup</code>	⊃	<code>\circledast</code>	⊛		

次に関係演算子を挙げる。

入力	出力	入力	出力	入力	出力
<code>\circlearrowright</code>	↻	<code>\upharpoonleft</code>	↵	<code>\gtrapprox</code>	⋈
<code>\circlearrowleft</code>	↺	<code>\downharpoonleft</code>	↵	<code>\multimap</code>	⋈

<code>\rightleftharpoons</code>	\Rightarrow	<code>\rightarrowtail</code>	\rightarrowtail	<code>\therefore</code>	\therefore
<code>\leftrightharpoons</code>	\Leftrightarrow	<code>\leftarrowtail</code>	\leftarrowtail	<code>\because</code>	\because
<code>\Vdash</code>	\Vdash	<code>\leftrightarrows</code>	\leftrightarrows	<code>\doteqdot</code>	\doteqdot
<code>\Vvdash</code>	\Vvdash	<code>\rightleftarrows</code>	\rightleftarrows	<code>\triangleq</code>	\triangleq
<code>\vDash</code>	\vDash	<code>\Lsh</code>	\Lsh	<code>\precsim</code>	\precsim
<code>\twoheadrightarrow</code>	\twoheadrightarrow	<code>\Rsh</code>	\Rsh	<code>\lesssim</code>	\lesssim
<code>\twoheadleftarrow</code>	\twoheadleftarrow	<code>\rightsquigarrow</code>	\rightsquigarrow	<code>\lessapprox</code>	\lessapprox
<code>\leftleftarrows</code>	\leftleftarrows	<code>\leftrightsquigarrow</code>	\leftrightsquigarrow	<code>\eqslantless</code>	\eqslantless
<code>\rightrightarrows</code>	\rightrightarrows	<code>\looparrowleft</code>	\looparrowleft	<code>\eqslantgtr</code>	\eqslantgtr
<code>\upuparrows</code>	\upuparrows	<code>\looparrowright</code>	\looparrowright	<code>\curlyeqprec</code>	\curlyeqprec
<code>\downdownarrows</code>	\downdownarrows	<code>\circeq</code>	\circeq	<code>\curlyeqsucc</code>	\curlyeqsucc
<code>\upharpoonright</code>	\upharpoonright	<code>\succsim</code>	\succsim		
<code>\downharpoonright</code>	\downharpoonright	<code>\gtrsim</code>	\gtrsim		

以下の関係演算子も用意されている。

入力	出力	入力	出力	入力	出力
<code>\preccurlyeq</code>	\preccurlyeq	<code>\trianglerighteq</code>	\trianglerighteq	<code>\smallsmile</code>	\smallsmile
<code>\leqq</code>	\leqq	<code>\trianglelefteq</code>	\trianglelefteq	<code>\smallfrown</code>	\smallfrown
<code>\leqslant</code>	\leqslant	<code>\between</code>	\between	<code>\Subset</code>	\Subset
<code>\lessgtr</code>	\lessgtr	<code>\blacktriangleright</code>	\blacktriangleright	<code>\Supset</code>	\Supset
<code>\risingdotseq</code>	\risingdotseq	<code>\blacktriangleleft</code>	\blacktriangleleft	<code>\subteq</code>	\subteq
<code>\fallingdotseq</code>	\fallingdotseq	<code>\vartriangle</code>	\vartriangle	<code>\supseteq</code>	\supseteq
<code>\succcurlyeq</code>	\succcurlyeq	<code>\eqcirc</code>	\eqcirc	<code>\bumpeq</code>	\bumpeq
<code>\geqq</code>	\geqq	<code>\lesseqgtr</code>	\lesseqgtr	<code>\Bumpeq</code>	\Bumpeq
<code>\geqslant</code>	\geqslant	<code>\gtreqless</code>	\gtreqless	<code>\lll</code>	\lll
<code>\gtrless</code>	\gtrless	<code>\lesseqqgtr</code>	\lesseqqgtr	<code>\ggg</code>	\ggg
<code>\sqsubset</code>	\sqsubset	<code>\gtreqqless</code>	\gtreqqless	<code>\pitchfork</code>	\pitchfork
<code>\sqsupset</code>	\sqsupset	<code>\Rrightarrow</code>	\Rrightarrow	<code>\backsim</code>	\backsim
<code>\vartriangleright</code>	\vartriangleright	<code>\Lleftarrow</code>	\Lleftarrow	<code>\backsimeq</code>	\backsimeq
<code>\vartriangleleft</code>	\vartriangleleft	<code>\varpropto</code>	\propto		

更に、以下の関係演算子も用意されている。

入力	出力	入力	出力	入力	出力
<code>\lvertneqq</code>	\lvertneqq	<code>\precnapprox</code>	\precnapprox	<code>\nvDash</code>	\nvDash
<code>\gvertneqq</code>	\gvertneqq	<code>\succnapprox</code>	\succnapprox	<code>\nVDash</code>	\nVDash
<code>\nleq</code>	\nleq	<code>\lnapprox</code>	\lnapprox	<code>\ntrianglerighteq</code>	\ntrianglerighteq
<code>\ngeq</code>	\ngeq	<code>\gnapprox</code>	\gnapprox	<code>\ntrianglelefteq</code>	\ntrianglelefteq
<code>\nless</code>	\nless	<code>\nsim</code>	\nsim	<code>\ntriangleright</code>	\ntriangleright
<code>\ngtr</code>	\ngtr	<code>\ncong</code>	\ncong	<code>\ntriangleleft</code>	\ntriangleleft
<code>\nprec</code>	\nprec	<code>\varsubsetneq</code>	\varsubsetneq	<code>\nleftarrow</code>	\nleftarrow
<code>\nsucc</code>	\nsucc	<code>\varsupsetneq</code>	\varsupsetneq	<code>\nrightarrow</code>	\nrightarrow
<code>\lneqq</code>	\lneqq	<code>\subsetneqq</code>	\subsetneqq	<code>\nLeftarrow</code>	\nLeftarrow
<code>\gneqq</code>	\gneqq	<code>\supsetneqq</code>	\supsetneqq	<code>\nRightarrow</code>	\nRightarrow
<code>\nleqslant</code>	\nleqslant	<code>\subsetneq</code>	\subsetneq	<code>\nleftrightarrow</code>	\nleftrightarrow
<code>\ngeqslant</code>	\ngeqslant	<code>\supsetneq</code>	\supsetneq	<code>\nLeftrightarrow</code>	\nLeftrightarrow
<code>\lneq</code>	\lneq	<code>\varsubsetneqq</code>	\varsubsetneqq	<code>\eqsim</code>	\eqsim
<code>\gneq</code>	\gneq	<code>\varsupsetneqq</code>	\varsupsetneqq	<code>\shortmid</code>	\shortmid
<code>\npreceq</code>	\npreceq	<code>\subsetneq</code>	\subsetneq	<code>\shortparallel</code>	\shortparallel
<code>\nsucceq</code>	\nsucceq	<code>\supsetneq</code>	\supsetneq	<code>\thicksim</code>	\thicksim
<code>\precnsim</code>	\precnsim	<code>\subsetneq</code>	\subsetneq	<code>\thickapprox</code>	\thickapprox
<code>\succnsim</code>	\succnsim	<code>\supsetneq</code>	\supsetneq	<code>\approxeq</code>	\approxeq

<code>\lnsim</code>	\lesssim	<code>\nparallel</code>	\nparallel	<code>\succapprox</code>	\gtrsim
<code>\gnsim</code>	\gtrsim	<code>\nmid</code>	\nmid	<code>\precapprox</code>	\lessapprox
<code>\nleqq</code>	\nleqq	<code>\nshortmid</code>	\nshortmid	<code>\curvearrowleft</code>	\curvearrowleft
<code>\ngeqq</code>	\ngeqq	<code>\nshortparallel</code>	\nshortparallel	<code>\curvearrowright</code>	\curvearrowright
<code>\precneqq</code>	\precneqq	<code>\nvdash</code>	\nvdash	<code>\backepsilon</code>	ϵ
<code>\succneqq</code>	\succneqq	<code>\nVdash</code>	\nVdash		

以下は、その他の記号である。

入力	出力	入力	出力	入力	出力
<code>\square</code>	\square	<code>\measuredangle</code>	\measuredangle	<code>\mho</code>	\mho
<code>\blacksquare</code>	\blacksquare	<code>\spherivalangle</code>	\spherivalangle	<code>\eth</code>	\eth
<code>\lozenge</code>	\lozenge	<code>\circledS</code>	\circledS	<code>\beth</code>	\beth
<code>\blacklozenge</code>	\blacklozenge	<code>\complement</code>	\complement	<code>\gimel</code>	\gimel
<code>\backprime</code>	\backprime	<code>\diagup</code>	\diagup	<code>\daleth</code>	\daleth
<code>\bigstar</code>	\bigstar	<code>\diagdown</code>	\diagdown	<code>\digamma</code>	\digamma
<code>\blacktriangledown</code>	\blacktriangledown	<code>\varnothing</code>	\varnothing	<code>\varkappa</code>	\varkappa
<code>\blacktriangle</code>	\blacktriangle	<code>\nexists</code>	\nexists	<code>\Bbbk</code>	\Bbbk
<code>\triangledown</code>	\triangledown	<code>\Finv</code>	\Finv	<code>\hslash</code>	\hslash
<code>\angle</code>	\angle	<code>\Game</code>	\Game	<code>\hbar</code>	\hbar

以下は、ギリシア語の斜体である。

入力	出力	入力	出力	入力	出力
<code>\varGamma</code>	\varGamma	<code>\varXi</code>	\varXi	<code>\varPhi</code>	\varPhi
<code>\varDelta</code>	\varDelta	<code>\varPi</code>	\varPi	<code>\varPsi</code>	\varPsi
<code>\varTheta</code>	\varTheta	<code>\varSigma</code>	\varSigma	<code>\varOmega</code>	\varOmega
<code>\varLambda</code>	\varLambda	<code>\varUpsilon</code>	\varUpsilon		

6.2 いろいろな記号

6.2.1 ドイツ語 (Fraktur)

\mathfrak{ABC} は `\mathfrak{ABC}` のようにして出力する。

6.2.2 黒板太文字

\mathbf{ABC} は `\mathbf{ABC}` のようにして出力する。

6.2.3 文脈に応じてサイズが変わるテキスト

`\text` は数式中にテキストをはさむために用いる。`\mbox` とは異なり、文脈に応じてフォントのサイズが変わる。

$$A_{\text{max}} = \text{some constant} \rightarrow A_{\text{max}} = \text{some constant}$$

6.2.4 賢い点々

数式の中で点々 (... の類) は、通常 `\dots` と書くだけで後続の記号から種類を判断してくれることになっている。

$$\begin{aligned} a_1, a_2, \dots, a_n &\rightarrow a_1, a_2, \dots, a_n \\ a_1 + a_2 + \dots + a_n &\rightarrow a_1 + a_2 + \dots + a_n \\ a_1 a_2 \dots a_n &\rightarrow a_1 a_2 \dots a_n \\ \int \dots \int &\rightarrow \int \dots \int \end{aligned}$$

最後の例は後述の `\idotsint` を利用した方がいいだろう。後続の記号がない場合や、うまくいかない場合は、次のような命令で区別する*1。

*1 これらは標準の L^AT_EX で用意されている `\ldots`、`\cdots` 命令に代わるもので、前後の空白が微妙に調節されている。

$\$a_{\{1\}}, \backslashdotsc\$ \rightarrow a_1, \dots$ ([commas](#))
 $\$a_{\{1\}} + \backslashdotssb\$ \rightarrow a_1 + \dots$ ([binary operation/relations](#))
 $\$a_{\{1\}} \backslashdotssm\$ \rightarrow a_1 \cdots$ ([multiplications](#))
 $\$\int \backslashint \backslashdotssi\$ \rightarrow \int \dots$ ([integrals](#))

6.2.5 長さが自由に伸びる矢印

両側に矢印が付いたのも以外は普通の L^AT_EX でも利用することができる。

$\$\overrightarrow{A}\$ \rightarrow \overrightarrow{A}$
 $\$\overleftarrow{A}\$ \rightarrow \overleftarrow{A}$
 $\$\overleftrightarrow{A}\$ \rightarrow \overleftrightarrow{A}$
 $\$\underrightarrow{A}\$ \rightarrow \underrightarrow{A}$
 $\$\underleftarrow{A}\$ \rightarrow \underleftarrow{A}$
 $\$\underleftrightarrow{A}\$ \rightarrow \underleftrightarrow{A}$

矢印と文字の間を離したいときは $\$\overleftrightarrow{\mathstrut x}\$$ のように \mathstrut を用いる。

6.2.6 いくらでも伸びる矢印

$\$xrightarrow{\text{xyz}}\$, $\$xrightarrow[abc]{\text{xyz}}\$$ は文字の付いた自由に伸びる矢印である。 $\$xrightarrow{\text{xyz}}\$$ のようにすると $\xrightarrow{\text{xyz}}$ 、 $\$xrightarrow[abc]{\text{xyz}}\$$ とすると $\xrightarrow[abc]{\text{xyz}}$ のようになる。$

入力	$\$ \backslash \text{text}\{\text{foo.tex}\} \backslash xrightarrow{\backslash \text{text}\{\text{platex}\}} \backslash \text{text}\{\text{foo.dvi}\} \backslash xrightarrow{\backslash \text{text}\{\text{dvipdfmx}\}} \backslash \text{text}\{\text{foo.pdf}\} \$$
出力	$\text{foo.tex} \xrightarrow{\text{platex}} \text{foo.dvi} \xrightarrow{\text{dvipdfmx}} \text{foo.pdf}$

6.2.7 数学用アクセント

$\$\hat{\hat{A}}\$$ のようにネスト（入れ子）にしても、 \hat{A} のように正しい位置に出力される。

入力	出力	入力	出力	入力	出力	入力	出力
$\backslash \text{Hat}\{A\}$	\hat{A}	$\backslash \text{Check}\{A\}$	\check{A}	$\backslash \text{Tilde}\{A\}$	\tilde{A}	$\backslash \text{Acute}\{A\}$	\acute{A}
$\backslash \text{Grave}\{A\}$	\grave{A}	$\backslash \text{Dot}\{A\}$	\dot{A}	$\backslash \text{Ddot}\{A\}$	\ddot{A}	$\backslash \text{Breve}\{A\}$	\breve{A}
$\backslash \text{Bar}\{A\}$	\bar{A}	$\backslash \text{Vec}\{A\}$	\vec{A}				

6.2.8 上につく点

次の最初の 2 つは `amsmath` パッケージを使わなくても利用可能である。

入力	出力	入力	出力	入力	出力	入力	出力
$\backslash \text{dot}\{x\}$	\dot{x}	$\backslash \text{ddot}\{x\}$	\ddot{x}	$\backslash \text{dddot}\{x\}$	\dddot{x}	$\backslash \text{ddddot}\{x\}$	\ddddot{x}

6.2.9 多重積分記号

$\backslash \text{int}\backslash \text{int}$ とすると間が空きすぎてしまうため、以下の命令が用意されている。

入力	出力	入力	出力	入力	出力	入力	出力
$\backslash \text{iint}$	\iint	$\backslash \text{iiint}$	\iiint	$\backslash \text{iiiint}$	\iiiiiint	$\backslash \text{idotsint}$	$\int \cdots \int$

6.2.10 数式中の空白

数式中の空白は $\backslash \text{mspace}$ という命令を用いて $\backslash \text{mspace}\{5\text{mu}\}$ のようにして出力する。 mu という単位（math units）は em （‘m’の幅）の $1/18$ である。負の値を指定することも可能である。

6.2.11 \smash

`\smash{...}` は高さ、深さをゼロにつぶす命令だが、`amsmath` パッケージで更に `\smash[t]{...}` や `\smash[b]{...}` でそれぞれ高さ、深さだけをゼロにすることができる。次の例では、 y のルートだけ下に伸びすぎるのを防ぐために使っている。

`\sqrt{x} + \sqrt{y}` $\rightarrow \sqrt{x} + \sqrt{y}$
`\sqrt{x} + \sqrt{\smash[b]{y}}` $\rightarrow \sqrt{x} + \sqrt{y}$

高さ・深さを揃えるための数式用の支柱 `\mathstrut` については前述したが、これを `\smash[b]` と組み合わせると便利である。

`\newcommand{\ssqrt}[1]{\sqrt{\smash[b]{\mathstrut #1}}}`
`\ssqrt{g} + \ssqrt{h}` $\rightarrow \sqrt{g} + \sqrt{h}$

6.2.12 演算子

L^AT_EX 標準の log 型関数に加えて、次の演算子が追加されている。

入力	出力	入力	出力	入力	出力
<code>\injl</code>	<code>injl</code>	<code>\varinjl</code>	$\lim\limits\rightarrow$	<code>\varliminf</code>	\liminf
<code>\projl</code>	<code>projl</code>	<code>\varprojl</code>	$\lim\limits\leftarrow$	<code>\varlimsup</code>	\limsup

この種の命令（マクロ）を更に追加することもできる。例えば、`\cosec x` と書いて $\operatorname{cosec} x$ と出力したければ、プリアンプルに次のように記述しておく。

```
\DeclareMathOperator{\cosec}{cosec}
```

また、

```
\DeclareMathOperator*{\argmax}{\mathrm{arg}\,max}
```

のように `*` を付ければ、

`\displaystyle{\argmax_{\theta}}` $\rightarrow \arg\max_{\theta}$

のように別行立て数式中で真下・真上に下限・上限が付くようになる。

マクロが定義できない場合は、`operatorname` もしくは `operatorname*` を用いる。

`\operatorname{cosec} x` $\rightarrow \operatorname{cosec} x$
`\operatorname*{\mathrm{arg}\,min}_{x} f(x)` $\rightarrow \arg\min_x f(x)$

6.3 行列

`amsmath` パッケージによる行列には、次のものが用意されている。

入力	出力
<code>\begin{matrix} a & b \\ c & d \end{matrix}</code>	$\begin{matrix} a & b \\ c & d \end{matrix}$
<code>\begin{pmatrix} a & b \\ c & d \end{pmatrix}</code>	$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$
<code>\begin{bmatrix} a & b \\ c & d \end{bmatrix}</code>	$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$
<code>\begin{Bmatrix} a & b \\ c & d \end{Bmatrix}</code>	$\begin{Bmatrix} a & b \\ c & d \end{Bmatrix}$

ちなみに、連分数は次のような書き方もすることがある。

入力 `\begin{equation}`

$$b_0 + \frac{c_1}{b_1 + \frac{c_2}{b_2 + \frac{c_3}{b_3 + \frac{c_4}{b_4 + \dots}}}}$$

`\end{equation}`

出力 $b_0 + \frac{c_1}{b_1 + \frac{c_2}{b_2 + \frac{c_3}{b_3 + \frac{c_4}{b_4 + \dots}}}}$

入力 `\begin{equation}`

$$b_0 + \frac{c_1}{b_1 + \frac{c_2}{b_2 + \frac{c_3}{b_3 + \frac{c_4}{b_4 + \dots}}}}$$

`\end{equation}`

出力 $b_0 + \frac{c_1}{b_1 + \frac{c_2}{b_2 + \frac{c_3}{b_3 + \frac{c_4}{b_4 + \dots}}}}$

6.4.2 2 項係数

2 項係数は `\binom{a}{b}` と書けば、本文中では $\binom{a}{b}$ のようなテキストスタイル、別行立て数式中では $\binom{a}{b}$ のようなディスプレイスタイルで出力される。必ずテキストスタイルで出力する `\tbinom`、必ずディスプレイスタイルで出力する `\tbinom` も用意されている。

6.4.3 一般の分数

分数や 2 項係数を含む一般の分数を出力する命令は、

`\genfrac{左括弧}{右括弧}{棒の太さ}{スタイル}{分子}{分母}`

である。「棒の太さ」は何も記入しなければ通常の分数の棒になるが、棒を出力したくない場合には `0pt` と書き込む。「スタイル」は通常は何も記入しないが、0 ~ 3 までの数字を書き込むと次のような特定にスタイルで出力することを意味する。

- 0 ... `\displaystyle` (別行立て数式のスタイル)
- 1 ... `\textstyle` (本文中の数式のスタイル)
- 2 ... `\scriptstyle` (添字のスタイル)
- 3 ... `\scriptscriptstyle` (添字の添字のスタイル)

例えば、

`\genfrac{}{}{}{}{a}{b}` → $\frac{a}{b}$
`\genfrac{\{}{\}}{0pt}{}{i}{j,k}` → $\left\{ \begin{matrix} i \\ j,k \end{matrix} \right\}$

のようになる。括弧が片側にしかない場合は逆側はピリオド (.) を指定する。

`\genfrac{.}{\}}{0pt}{}{a}{b}` → $\frac{a}{b)}$

6.5 別行立ての数式

数式番号の付いた別行立ての数式を出力するには `equation` 環境を用いる (これは標準の `LATEX` と同じである)。

入力 `\begin{equation}`

$$E = mc^2$$

`\end{equation}`

出力 | $E = mc^2$ (6)

数式番号が不要な場合は `equation*` 環境を用いる。

入力 | `\begin{equation*}`
 $E = mc^2$
`\end{equation*}`

出力 | $E = mc^2$

標準的でない数式番号は `\tag` で付ける。例えば、(*) という番号を付けるには `\tag{**}` とする*2。

入力 | `\begin{equation}`
 $E = mc^2$ `\tag{**}`
`\end{equation}`

出力 | $E = mc^2$ (*)

`\tag{...}` の中身は本文用のフォントで組まれる。数式番号に括弧を付けたくない場合は `\tag*{...}` とする。

複数の数式を並べるには `gather` 環境を用いる。数式の区切り（改行）は `\\` である。最後の行には `\\` は付けない。

入力 | `\begin{gather}`
 $(a + b)^2 = a^2 + 2ab + b^2$ `\\`
 $(a - b)^2 = a^2 - 2ab + b^2$ `\notag \\`
 $(a + b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$
`\end{gather}`

出力 | $(a + b)^2 = a^2 + 2ab + b^2$ (7)

$(a - b)^2 = a^2 - 2ab + b^2$

$(a + b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$ (8)

各行に数式番号が付加されるが、番号を付けたくない行は、最後 (`\\` の直前) に `\notag` と書いておく（他の数式環境でも同様である）。

`gather` の代わりに `gather*` とすると、全ての行に数式番号が付かなくなる。環境名に `*` を付けると番号が付かなくなるのは、他の数式環境でも同様である。

改行の命令 `\\` を例えば `\\[-3pt]` に変えると、改行の幅が通常より 3 ポイント小さくなる。和文（行送り 15 ～ 17 ポイント）と欧文（行送り 12 ～ 13 ポイント）の一般的な行送りの違いを考えれば、和文の中の数式は改行を `\\[-3pt]` 程度にする方が適切かもしれない（他の数式環境でも同様）。

`align` 環境は `&` で位置を揃えることができる。各行に番号が付く。番号が不要な行には `\notag` を用いる。

入力 | `\begin{align}`
 $\sinh^{-1} x$ `&= \log(x + \sqrt{x^2 + 1})` `\notag \\`
 $= x - x^3/6 + 3x^5/40 + \dots$
`\end{align}`

出力 | $\sinh^{-1} x = \log(x + \sqrt{x^2 + 1})$
 $= x - x^3/6 + 3x^5/40 + \dots$

(9)

どの行にも番号が不要なら `align*` を用いる。

位置を揃えた複数行の数式全体の中央に番号を振るには、`split` 環境もしくは `aligned` 環境で位置を揃え、全体を他の数式環境の中に入れて番号を振る。

*2 `**` は本来は数式モードの掛け算の記号なので、正しい使い方ではないかもしれない。これが気になる場合は `textcomp` パッケージの `\textasteriskcentered` を用いる。

入力	<pre>\begin{equation} \begin{split} \sinh^{-1} x &= \log(x + \sqrt{x^2 + 1}) \notag \\ &= x - x^3/6 + 3x^5/40 + \dots \end{split} \end{equation}</pre>	
出力	$\sinh^{-1} x = \log(x + \sqrt{x^2 + 1})$ $= x - x^3/6 + 3x^5/40 + \dots$	(10)

上の例で数式番号を出力したのは `equation` 環境の方である。`split` 自身は数式番号を出力しない（よって `split*` は存在しない）。

`aligned` も `split` とほぼ同じ用途に用いることができるが、こちらはより柔軟性に富み、枠 `\fbox` に入れたり、オプション `[t]` や `[b]` を付けて揃え位置を上下に動かしたりできるので、箇条書きの番号と揃えるときも便利である。

`align` 環境の類は各行に複数の `&` があっても構わない。各行の偶数番目の `&` は式を区切るために用いられる。

入力	<pre>\begin{align*} \sin A &= y/r & \cos A &= x/r & \tan A &= y/x \\ \cot A &= x/y & \sec A &= r/x & \csc A &= r/y \end{align*}</pre>	
出力	$\begin{array}{lll} \sin A = y/r & \cos A = x/r & \tan A = y/x \\ \cot A = x/y & \sec A = r/x & \csc A = r/y \end{array}$	

数式どうしの間隔を自分で制御するには `alignat{数式の個数}` を用いる。「数式の個数」とは各列の数式の個数（偶数番目の `&` の個数 + 1）の最大値である。これは次のような場合に便利である。

入力	<pre>\begin{alignat}{2} (a+b)^2 &= a^2+2ab+b^2 & \quad & \quad & \text{展開する} \\ &= a(a+2a)+b^2 & \quad & \quad & \text{\\$a\\$ でくくる} \end{alignat}</pre>	
出力	$(a+b)^2 = a^2 + 2ab + b^2 \quad \text{展開する}$ $= a(a+2a) + b^2 \quad a \text{ でくくる}$	(11) (12)

数式の途中に文章を割り込ませるには `\intertext` を用いる。

入力	<pre>\begin{align} s_{1} &= a_{1}, \\ s_{2} &= a_{1} + a_{2}, \\ \intertext{一般に、} s_{n} &= a_{1} + a_{2} + \cdots + a_{n} \end{align}</pre>	
出力	$s_1 = a_1,$ $s_2 = a_1 + a_2,$ <p>一般に、</p> $s_n = a_1 + a_2 + \cdots + a_n$	(13) (14) (15)

揃え位置のない複数行にわたる 1 つの数式は `multline` で記述する。

入力	<pre>\begin{multline} a + b + c + d + e + f + g + h + i + j + k \\ + l + m + n + o + p + q + r + s + t + u + v \\ + v + x + y + z + \alpha + \beta + \gamma + \delta \end{multline}</pre>	
----	---	--

	<code>\end{multline}</code>	
出力	$ \begin{aligned} &a + b + c + d + e + f + g + h + i + j + k \\ &\quad + l + m + n + o + p + q + r + s + t + u + v \\ &\quad + v + x + y + z + \alpha + \beta + \gamma + \delta \end{aligned} $	(16)

最初の行は左に寄り、最後の行は右に寄る。それ以外の行は、標準では左右中央に並ぶ（`fleqn` オプションを付ければ左から一定距離に並ぶ）が、強制的に右に寄せたい行は `\shoveright{...}`、左に寄せたい行は `\shoveleft{...}` で囲む（囲む範囲は改行 `\` の直前まで）。

左右に寄る場合、`\multlinegap` だけ余白が入る。これは標準で 10pt だが、`\setlength{\multlinegap}{20pt}` のようにして変更することができる。

数式中の `\` では改ページされない。改ページを許すには、`\` の直前に `\displaybreak[0]` と書いておく。この `[0]` を `[1]`、`[2]`、`[3]` と変更すると改ページのしやすさが次第に増し `\displaybreak[4]` では必ず改ページする。単に `\displaybreak` と書けば `\displaybreak[4]` と同じ意味になる。

全ての `\` について同じ改ページのしやすさを設定するには、プリアンブルに `\allowdisplaybreak[1]` などと記述しておく。オプションのパラメータの値は 0 から 4 までで、0 では改ページせず、4 に近づくほど改ページしやすくなる。この場合、改ページしたくない改行は `\`* で表す。

L^AT_EX では、例えば `\label{Einstein}` というラベルを貼った数式を参照する際、

式~(`\ref{Einstein}`) では…

のように書くが、`amsmath` パッケージでは、

式~`\eqref{Einstein}` では…

という命令も用意されている。`\eqref` の方が括弧やイタリック補正が組み込まれており便利である。

第 7 章

グラフィック

L^AT_EX 文書の中に写真や、他のツールで描いた図などを挿入することができる。昔は L^AT_EX に図を挿入するといえば EPS 形式がほとんどだったが、今日では PDF 形式（ラスター画像なら JPEG や PNG 形式）で挿入する方が速くトラブルもない。

逆に、L^AT_EX で組んだ文章や数式などを、他のソフトに PDF などのベクトル形式で挿入するすることも可能である。文字を変形したり、文字や背景に色を付けたりする方法も本章で扱う。

7.1 L^AT_EX と図

L^AT_EX だけで図を描く方法は多数存在する。いくつか例を挙げる。

- L^AT_EX 標準の `pictuer` 環境（現在は推奨しない）。
- `pict2e` パッケージの `picture` 環境。
- PostScript ベースの `PSTricks` というパッケージ群（PDF ベースのワークフローでは推奨しない）。
- `TikZ`
- 大熊一弘氏の `emath` パッケージ。

これらは全てコマンド（文字による命令）で図を描くため、人によっては敷居が高いと感じるかもしれない。

より簡単な方法として、Adobe Illustrator や Inkscape などのドローツール、PowerPoint や Keynote などのスライド作成ソフト、Excel や R などの統計グラフ機能を持ったソフトを活用して図を描き、PDF 形式で保存して L^AT_EX 文書に挿入することができる。複雑な表も Excel で組んで PDF で保存することで、同様に挿入することができる（Excel の図表が美しいかどうかは別問題だが）。

デジカメで撮影した写真やスキャン画像、Windows のペイントや GIMP、Photoshop などで描いた画像は、JPEG や PNG 形式のままで挿入することができる。

7.2 L^AT_EX での図の読み込み方

L^AT_EX 文書への図の挿入方法は、ワークフロー（処理の流れ）によって少しずつ異なる。本稿で主に取り扱っている日本語の PDF ワークフローでは `dvipdfmx` が中心になるが、ここでは全てに共通のことを説明する。

グラフィックを扱うためには、プリアンブルで次のように `graphicx` パッケージを読み込み、本文中は `\includegraphics` コマンドを用いて図を挿入する。

```
\documentclass[ドライバ名]{jsarticle}
\usepackage[hiresbb]{graphicx}
\begin{document}
  (……本文……)
\includegraphics[オプション]{ファイル名}
  (……本文……)
\end{document}
```

この**ドライバ名**のところに dvipdfmx、pdftex、xetex、luatex などが入る。**オプション**には [width=5cm] や [height=3cm] のような挿入枠の大きさの指定などが入る。

以下では、ドライバ毎に更に詳述する。

7.2.1 dvipdfmx の場合

dvipdfmx は pTeX、upTeX で標準的に使われるドライバである。PDF、PNG、JPEG 形式の図が扱えるほか、Ghostscript の力を借りて EPS 形式の図も扱うことができる。

```
\documentclass[dvipdfmx]{jsarticle}
\usepackage[hiresbb]{graphicx}
\begin{document}
\includegraphics[width=5cm]{sample.pdf}
\end{document}
```

これを pL^AT_EX と dvipdfmx で処理して PDF を作成する。

7.2.2 dvips の場合

PDF ではなく PostScript 出力が必要な場合に使うドライバである。扱える図は EPS 形式のみである。EPS 形式の図をそのまま取り込むだけなので、Ghostscript などを必要としない。

dvips を使う場合は \documentclass[dvips]{jsarticle} とする。

7.2.3 pdfTeX、X_YTeX、LuaTeX の場合

これらのエンジンは PDF を自前で出力することができる。グラフィックオプションはそれぞれ pdftex、xetex、luatex である。PDF、JPEG、PNG、JBIG2 の他、METAPOST の出力する単純な EPS ファイル（MPS ファイル）にも対応している。一般の EPS ファイルは Ghostscript で PDF に変換してから取り込む。

7.3 graphicx パッケージの詳細

ここまで何度か \usepackage[**オプション**]{graphicx} のような書き方が出てきたので、ここで少しまとめてみる。オプションの部分に記述することができる主なものには、以下のものがある。

draft	図を表示しない（枠とファイル名のみ表示する）。
final	図を表示する（デフォルト）。
hiresbb	EPS または xbb ファイルに含まれる高解像度バウンディングボックス（HiResBoundingBox）情報を利用する。
noresetpagesize	PDF ファイルのページサイズを自動設定しない（T _E X Live 2016 以降）。

draft と hiresbb は個々の図に対しても指定することができる。通常は final オプションは不要だが、次のような場合に便利である。

```
\documentclass[dvipdfmx,draft]{jsarticle} % 全体はドラフトモードで
\usepackage[final]{graphicx}              % 図はちゃんと表示する
```

graphicx パッケージを使えば、図を取り込むための \includegraphics 以外にも、図や文字を回転・拡大・縮小する命令が使用可能となる（後述）。

7.4 \includegraphics の詳細

例えば、

```
\includegraphics[width=3cm]{tiger.pdf}
```

とすると、tiger.pdf という画像を 3 cm 幅で取り込んで、その場所に出力（表示）する。画像は 1 つの大きな文字として扱われるので、必要に応じて center 環境などに入れておく。

画像ファイルは通常、文書ファイルと同じフォルダに置いておくが、例えばサブフォルダ sub1、sub2 の中の図も探させたい場合は、

```
\graphicspath{{sub1/}{sub2/}}
```

という命令を書いておく。その他、 \LaTeX の入力ファイルを見つけられるところなら、どこに図を置いても構わない。次のようにパスを指定することもできる。

```
\includegraphics[width=5cm]{C:/picture/flowers.jpg}
```

パスの区切りは Windows 環境でも \ や ¥ ではなく / を用いる。

\includegraphics は次のオプションを理解する。

- width は幅を指定する。上の例では画像の幅を 5 cm にスケール（拡大または縮小）して出力している。
- height は高さを指定する。
 $\text{\includegraphics[height=3cm]{...}}$
totalheight で「高さ+深さ」を指定することができる。図を回転した場合はこちらの方が便利である。
- width と height を同時に指定すると縦横比が変わってしまう。keepaspectratio を指定すると、縦横比を変えずに指定した幅と高さに収まるように拡大縮小する。
 $\text{\includegraphics[width=3cm,height=3cm,keepaspectratio]{...}}$
- scale=0.8 で画像のサイズが 0.8 倍になる。元の画像に 10pt で書いていた文字が footnotesize（8pt）になるように取り込みたいといった場合に便利である。
- hiresbb をつけると、小数点以下を含むバウンディングボックス情報を用いて画像を取り込む。
- clip はクリッピングする。すなわち、描画領域（バウンディングボックス）の外側を描かないという指定である。描画領域の外側に余分なものが描かれている EPS ファイルなどを取り込むと、周囲の文書が侵蝕されるので clip は指定しておく方が安全である。
- trim= $x_1 y_1 x_2 y_2$ で、左 x_1 、下 y_1 、右 x_2 、上 y_2 単位だけの図を切り詰める。単位は 1/72 インチである。例えば、図の下部 1 インチを切り詰めたいときは trim=0 72 0 0 とする。
- viewport= $x_1 y_1 x_2 y_2$ で、元の図の左下隅を原点とする左下隅 (x_1, y_1)、右上隅 (x_2, y_2) の長方形の領域を出力する。単位は 1/72 インチである。例えば、図の左下隅 1 インチ角だけ使いたいなら viewport=0 0 72 72 とする。
- angle=30 で画像が 30° 回転する。origin オプションで回転の中心を指定する。詳細は \rotatebox （後述）の解説を参照のこと。90° 回転して幅 5 cm に収めたい場合は、高さ（height）を 5 cm に指定しなければならない。
- draft で画像の枠とファイル名だけを表示する。

7.5 主な画像ファイル形式

\LaTeX でよく使われる画像形式をまとめておく。選択のポイントは、ベクトル画像かビットマップ（ピクセル・ラスター）画像か、ビットマップの場合には非可逆圧縮かどうかである。スクリーンショットやセル画などノイズが目立ちやすいピクセル画像の場合、可逆圧縮が推奨である。更に、印刷所でカラー印刷してもらうためには、CMYK 対応かどうかとも決め手となる。

- | | |
|-----|---|
| EPS | EPS（Encapsulated PostScript）は PostScript 形式の一種である。ベクトル・ビットマップ、RGB、CMYK の全てに対応している。PostScript と EPS については歴史的に重要なので、後に詳述する。 |
| PDF | PDF（Portable Document Format）は PostScript 形式が変わって、広く用いられているファイル形式である。ベクトル・ビットマップ、RGB、CMYK の全てに対応している。これについても後述する。 |
| PNG | ピングと読む。可逆圧縮であるため JPEG のようなノイズが入らず、スクリーンショットなどの保存に最適である。RGB やグレイスケールに対応しているが、残念ながら CMYK には対応していない*1。本格的なカラー印刷用には、Photoshop など CMYK に変換してから色を調整し、PDF など保存する必要がある。機械的な方法でよければ第 11 節で方法を説明する。 |


*1 このため印刷業界では PNG は普及しておらず、TIFF や Photoshop 形式（PSD）の方が一般的である。

- SVG** SVG (Scalable Vector Graphics) は比較的新しいベクトル形式の画像フォーマットである。最新のブラウザは SVG 画像に対応している。
- JPEG** ジェイペグと読む。拡張子は `jpg` または `jpeg` である。写真などのフル階調のカラーまたはグレースケールのピクセル画像用である。非可逆圧縮であるため、スクリーンショットなどではノイズが目立つ。Photoshop などで作成することができる CMYK 形式の JPEG については `dvipdfmx` に対応しているが、対応していないソフトも多いため、一般には PDF に変換しておく方が安全である。

7.6 PostScript とは？

コンピュータで扱う画像はベクトル形式とビットマップ形式（ラスター形式）に大別される。後者はピクセルという正方形の集まりで構成した画像で、非常にわかりやすいものだが、前者は「滑らかな画像」「数式で表した画像」などと説明されるため、なかなか理解しづらい。よって、ここではベクトル形式の代表格にあたる ポストスクリプト PostScript 形式について説明する。

PostScript 言語^{*2} は業界標準のページ記述言語（ページ上の文字や図形の配置を記述するための一種のプログラミング言語）であり、この言語の命令を書き込んだファイルが PostScript ファイルである。以下では、PostScript を略して PS と表記することにする。

以下に簡単な PS ファイルの例を示す。これは  のような図形を記述したものである。

```
%!PS          % PS ファイルはこの 4 文字で始まる。
10 10 moveto   % 点 (10,10) に移動する。
30 10 lineto   % 点 (30,10) まで線分を引く。
20 20 10 0 180 arc % 中心 (20,20)、半径 10、角度 0 ~ 180 度の円弧を引く。
closepath     % パスを閉じる。
stroke        % 実際に線を描く。
showpage      % ページ全体を出力する。
```

このように、PS ファイルの基本はテキストファイルである。簡単な命令をいくつか覚えれば手で記述することもできる。長さの単位は 1/72 である。

このテキストファイルを、例えば `test.ps` という名前で保存し、Ghostscript などの PS（互換）ソフトで開けば、先程のような図形を画面で確認することができる。また、PS（互換）プリンタに送れば、紙に出力される。

7.6.1 Ghostscript

ゴーストスクリプト Ghostscript はオープンソースの PS 言語インタプリタ、すなわち PS 言語で書かれた図形を画面表示したり、一般のプリンタに出力したりするソフトウェアである。別の言い方をすれば、Ghostscript とは PostScript 互換のソフトウェア リップ RIP（ラスターライザ）である。RIP とは Raster Image Processor の略で、PostScript データをラスター画像（ビットマップ）に変換する装置またはソフトウェアである。Ghostscript は PDF のラスターライズや、PostScript から PDF への変換もできる。`pdfTeX` や `dvipdfmx` などは、EPS 形式の図を出力するために Ghostscript を利用している。

7.7 EPS とは？

EPS (Encapsulated Postscript : カプセル化されたポストスクリプト) とは、1 つの図のみを含む限定された PostScript 形式のことである。EPS ファイルのことを EPSF と書く。EPS ファイルは次のような行で始まる。

```
%!PS-Adobe-3.0 EPSF-3.0
```

これ以降は、若干のコメント（% で始まる行）と PS 言語による図形の表現が続く。

^{*2} PostScript は Adobe Systems Incorporated の登録商標である。正しくは名詞ではなく形容詞なので、PostScript 言語、PostScript プリンタなどのように用いなければならない。

通常の PS ファイルは複数ページの図を含むが、EPS ファイルにはページという概念が存在しない。また、EPS ファイルの先頭付近には「バウンディングボックス」情報が必ず付加されている。バウンディングボックスとは図の外枠のことで、EPS ファイルの先頭付近に、

```
%%BoundingBox: 12 202 571 776
```

のような形式で記述されている。これは図の左下隅の座標が (12,202)、右上隅の座標が (571,776) であることを意味している (単位は 1/72 インチ)。

EPS ファイルによっては、通常のバウンディングボックス情報以外に、

```
%%HiResBoundingBox: 12.3456 201.789 570.6895 776.1234
```

のような小数点以下を含むバウンディングボックス情報を保持しているものもある。通常、 \LaTeX はこれらの情報を無視するが、これを読むようにするには、

```
\usepackage[hiresbb]{graphicx}
```

あるいは、個々の図を読み込むところで、

```
\includegraphics[hiresbb,width=5cm]{sample.eps}
```

のように hiresbb オプションを付加する。

\LaTeX は EPS ファイルの中の図を解釈することはない。単にバウンディングボックス情報だけを見て、確保する大きさを決定する。

7.8 PDF とは？


PDF (Portable Document Format) とは、PostScript と同じ Adobe Systems が開発したオープンな文書フォーマットである。PostScript の進化形とも言われるので、インターネットでの情報交換から印刷所への入稿まで、広く用いられている。Adobe から無償で配布されている Reader³をはじめ、多くの PDF 閲覧ソフトが存在する。

PDF にすればどんな環境でも同じ出力ができるというのが理想だが、現実にはフォント環境が異なれば出力も異なってしまう。これを避けるために、PDF には使用した文字のフォントデータを埋め込むことができる。

7.9 SVG とは？

SVG (Scalable Vector Graphics) は XML ベースの新しいベクトル形式の画像フォーマットである。最近のブラウザは SVG 画像に対応している。Adobe Illustrator やオープンソースの Inkscape などで作成することができる他、テキストファイルなので手で書くこともできる。例えば、

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
  width="50" height="50" style="font-size:16px">
  <circle cx="25" cy="25" r="22" fill="#CCCCCC"
    stroke="black" stroke-width="2" />
  <text x="25" y="30" text-anchor="middle">まる</text>
</svg>
```

と記述したテキストファイルを maru.svg という名前で保存し、ブラウザで開けば  のような画像が表示される。

\LaTeX で使うには、予め Inkscape などのツールで PDF に変換するのが簡単である。Inkscape には「PDF+ \LaTeX : Omit text in PDF, and create \LaTeX file」という保存方法が存在し、SVG からテキストを除いたものを maru.pdf、テキスト部分を \LaTeX 形式で maru.pdf_tex というファイル名で出力する。

³ 最初は Acrobat Reader という名前だったが、Adobe Reader に改称され、現在は Adobe Acrobat Reader DC という名前になった。より高機能な Adobe Acrobat とは別物である。

これらを統合して出力するには、プリアンブルで `svg` パッケージを読み込んでおき、図を出力したいところに `\includesvg{maru}` と記述する。Inkscape が `inkscape` コマンドで呼び出せるように設定された環境では、`LATEX` を `--shell-escape` オプション付きで起動すれば、変換まで自動で行ってくれる。

7.10 文字列の変形

`graphicx` パッケージを使うと、`\includegraphics` 以外にもいろいろなコマンドが使えるようになる。特に便利な文字列変形のコマンド群を挙げておく。

7.10.1 `\rotatebox`[オプション]{角度}{文字列}

文字列を回転する。

文字列を `\rotatebox{45}{傾けて}` 書くことができる。 → 文字列を ^{傾け} 書くことができる。

オプション `[origin=c]` を指定すると、文字列の中心が回転の中心となる。`[origin=tr]` なら、文字列の右上隅が回転の中心となる。`origin` 指定で可以使用なのは `lrctbB`（それぞれ左、右、中、上、下、ベースライン）とその 2 文字の組み合わせである。回転の中心は `[x=3mm,y=2mm]` のように座標で指定することもできる。無指定では回転の単位は度だが、`[units=6.2832]` でラジアンに変更することができる。

7.10.2 `\scalebox`{倍率}[縦の倍率]{文字列}

文字を拡大縮小する。オプションの縦の倍率を指定しなければ、縦も横も同じ倍率で拡大縮小される。

半角カナは `\scalebox{0.5}[1]{カナ}` のように書くことができる。倍角ダッシュ — は `\scalebox{2}[1]{—}` のように書くことができる。

7.10.3 `\reflectbox`{文字列}

文字列を垂直対称にする。`\scalebox{-1}[1]{文字列}` と同じである。

7.10.4 `\resizebox`{幅}{高さ}{文字列}

文字列を指定の幅・高さに拡大縮小する。`\resizebox*` のように `*` を付けると「高さ」が「高さ+深さ」になる。幅と高さを同じ倍率で拡大縮小するには、一方の長さだけを指定して、もう一方を `!` とする。元々の幅は `\width`、高さは `\height` と書く。

長い数式をページに押し込みたい場合にも使うことができる。次の例では、数式の幅を行長 (`\columnwidth`) の 0.9 倍にしている。

```
\resizebox{0.9\columnwidth}{!}{
  {\$ \displaystyle \kappa = \kappa_{1} + \kappa_{2} + \frac{\dots}{D}$}
```

7.11 色空間とその変換

パソコンの画面の色は光の 3 原色の赤・緑・青 (Red、Green、Blue : 合わせて RGB と呼ぶ) で作られている。これに対して、印刷で使うプロセスカラーはシアン、マゼンタ、イエロー、ブラック (Cyan、Magenta、Yellow、black : 合わせて CMYK と呼ぶ) を用いる。

原理的には CMY のみでいいはずなのだが、この 3 色を混ぜて完全な黒を表すのは難しいだけでなく、黒は文字色に使われ、CMY の黒では版が少しでもずれると文字が読みにくくなるため、黒だけは特別扱いしている。図版では「より黒い黒」を表現するために K に CMY を被せることもある。これ以外にも、特定の色を正確に表したい場合には、特色 (スポットカラー) を用いることがある。

RGB と言っても、広く用いられている sRGB 以外に、より広い色域の Adobe RGB など数種類の色空間が存在する。デジタルのカラー印刷用データには Adobe RGB を用いるべきだが、それでも CMYK の色域とは完全に一致せず、変換は単純ではない。

RGB データをそのまま入稿することができる仕組みが整い始めているが、今のところ印刷所に入稿する際には CMYK 対応ソフトで色合いを調整して CMYK 形式（モノクロならグレースケール形式）で保存の方が安全である。

CMYK を扱えるソフトには Adobe 社の Photoshop（ラスター画像用）や Illustrator（ベクトル画像用）、オープンソースの KOffice の Krita（ラスター画像用）などが存在する。GIMP は `separate+` プラグインにより CMYK が扱えるようになる。

機械的な変換で構わなければ、ImageMagick の画像変換コマンド `magick` が便利である。PNG 画像をグレースケールや CMYK に変換するためのいくつかの例を以下に示す。

```
$ magick foo.png -colorspace Gray foo-gray.png
$ magick foo.png -colorspace Gray EPDF:foo-gray.pdf
$ magick foo.png -colorspace CMYK EPDF:foo-cmyk.pdf
```

PDF 変換時の EPDF: という指定は、いわゆる Encapsulated PDF（MediaBox が用紙全体ではなく画像だけを指す PDF）にするためのものである。ImageMagick で扱えるのはラスター画像のみで、ベクトル画像の色変換は Adobe Illustrator などを用いることになる。

7.12 色の指定

L^AT_EX で色を使うには、古くは `color` パッケージが使われていたが、ここではより強力な `xcolor` パッケージを紹介する。例えば、`dvipdfmx` で `graphics` と `xcolor` を使うには、次のようにする。

```
\documentclass[dvipdfmx]{jsarticle}
\usepackage{graphicx,xcolor}
```

色の指定は次のように行う。

- グレースケールの指定は、
`\color[gray]{0.5} 文字` 又は `\textcolor[gray]{0.5}{文字}`
のようにする。数値は 0～1 で、0 が黒、1 が白である。
- カラーの印刷物なら CMYK で指定する。例えば、シアン 0.75、マゼンタ 0、Yellow 0.65、Black 0 で作る色（CUD 推奨配色の緑色）で文字を書く場合は、
`\color[cmk]{0.75,0,0.65,0} 文字` 又は `\textcolor[cmk]{0.75,0,0.65,0}{文字}`
のようにする。
- ディスプレイに表示する色は RGB 値で、
`\color[rgb]{0.2,0.6,0.4} 文字` 又は `\textcolor[rgb]{0.2,0.6,0.4}{文字}`
のように指定する。あるいは、HTML にならった記法
`\color[HTML]{35A16B} 文字` 又は `\textcolor[HTML]{35A16B}{文字}`
のように指定することもできる。

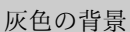
成分の割合で指定するのは面倒なので、いくつかの色名が定義されてる。次の色はどんなドライバでも使用可能である。

RGB 系 red (1,0,0)、green (0,1,0)、blue (0,0,1)、brown (.75,.5,.25)、lime (.75,1,0)、orange (1,.5,0)、pink (1,.75,.75)
purple (.75,0,.25)、teal (0,.5,.5)、violet (.5,0,.5)
CMYK 系 cyan (1,0,0,0)、magenta (0,1,0,0)、yellow (0,0,1,0)、olive (0.0,1,.5)
GRAY 系 black (0)、darkgray (.25)、gray (.5)、lightgray (.75)、white (1)

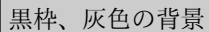
更に、`black!20` で黒 20%（薄い灰色）、`red!30!yellow` で赤 30%・黄 70%（赤みがかった黄）といった指定ができる。

`\pagecolor{色の名前}` でページの背景色を変えることができる。`\color` 命令同様、グレイレベルや RGB、CMYK の数値でも指定可能である。別の色を指定するまではページ色は変わったままなので、元の白に戻すには `\pagecolor{white}` とする。

`\colorbox{色名}{文字}` で文字の背景に色を付けることができる。色は数値でも指定可能である。

`\colorbox[gray]{0.8}{灰色の背景}` → 

`\fcolorbox{色名 1}{色名 2}{文字}` で色名 1 の枠、色名 2 の背景で文字を書くことができる。色は数値でも指定可能である。

`\fcolorbox[gray]{0}{0.8}{黒枠、灰色の背景}` → 

7.13 枠組み

枠で囲む方法はさまざま存在するが、ここでは最新・最強の `tcolorbox` パッケージを紹介する。このパッケージの特徴は、枠組みの途中で改ページすることができることと、多数のオプションにより多様な枠が描けることである。内部で `TikZ` を呼び出して使用する。

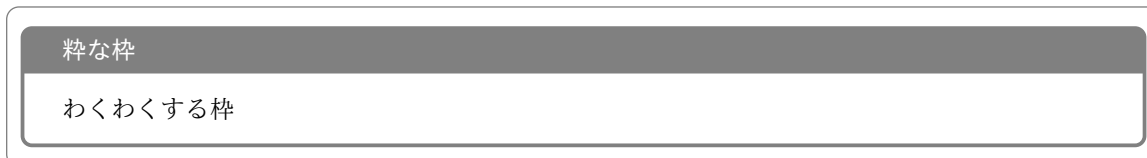
`tcolorbox` パッケージを使うには、プリアンブルに次のように記述しておく。

```
\documentclass[dvipdfmx]{jsarticle}
\usepackage{tcolorbox}
```

枠組みは `\begin{tcolorbox} ... \end{tcolorbox}` で行う。次の例のように、さまざまなオプションを指定することができる。

```
\begin{tcolorbox}[colframe=black!50,colback=white,colbacktitle=black!50,
                  coltitle=white,fonttitle=\bfseries\sffamily,title=粋な枠]
  わくわくする枠
\end{tcolorbox}
```

出力は次のようになる。



第 8 章

表組み

本章では、 \LaTeX 標準の `tabular` 環境と、それを改良するための `array`、`tabularx`、`booktabs` パッケージ、ページをまたぐ表を作成する `longtable` パッケージについて説明する。

8.1 表組みの基本

\LaTeX には、通常のテキスト内で作表する `tabular` 環境、数式内で作表する `array` 環境が用意されている。まずは、`tabular` 環境の基本として、罫線のない表を書いてみる。

品名	単価 (円)	個数
りんご	100	5
みかん	50	10

このように出力するには、次のように入力する。

```
\documentclass{jsarticle}
\begin{document}
\begin{center}
\begin{tabular}{lrr}
品名 & 単価 (円) & 個数 \\
りんご & 100 & 5 \\
みかん & 50 & 10
\end{tabular}
\end{center}
\end{document}
```

`center` 環境は表を左右中央に置くためで、表組みとの直接的な関係はない。その内側の `tabular` 環境が表をそのものを出力する環境である。この命令は、

```
\begin{tabular}{列指定}
表本体
\end{tabular}
```

の形で用いる。列指定は、

- `l` : 左寄せ (`left`)
- `c` : 中央 (`center`)
- `r` : 右寄せ (`right`)

を列の数だけ並べる。先程の `\begin{tabular}{lrr}` では、列指定は `lrr` だったので、1 列目は左寄せ、2 列目と 3 列目は右寄せとなっている。

表本体では、列の区切りは `&`、行の区切りは `\\` である。表の最下行の終わりには `\\` は付けない（後述の `\bottomrule` や `\hline` が付く場合は例外である）。

上の例では表を `center` 環境に入れたが、`flushleft`（左寄せ）や `flushright`（右寄せ）にすると、`tabular` 環境の両側に

ほんの少し余分なスペースが入っているのがわかる。この余分なスペースを消すには、例えば列指定が `{lrr}` なら `@{llr@{}}` のように、列指定の両側に `@{}` という命令を入れる。`@{何か}` は表の両側や列間に `何か` を挿入するためのものだが、`@{}` のように中身を空にすることで、本来入るはずの表の両側のスペースを消している。

8.2 `\booktabs` による罫線

先ほどの例に罫線を引いてみる。`LATEX` 標準の罫線は後述するが、ここではまず最近よく用いられる `booktabs` パッケージを説明する。日本人は格子状の罫線を好むが、横書きの文化圏では次のように横罫線だけを使うのが原則である。

品名	単価 (円)	個数
りんご	100	5
みかん	50	10

このように出力するには、`booktabs` パッケージを用いて次のようにする。

```
\documentclass{jsarticle}
\usepackage{booktabs}
\begin{document}
\begin{center}
\begin{tabular}{lrr}
\toprule
品名 & 単価 (円) & 個数 \\
\midrule
りんご & 100 & 5 \\
みかん & 50 & 10 \\
\bottomrule
\end{tabular}
\end{center}
```

`booktabs` パッケージで仕える命令は、次の通りである。

- `\toprule` 最初の罫線
- `\midrule` 中央の罫線
- `\bottomrule` 最後の罫線

8.3 `LATEX` 標準の罫線

`LATEX` 標準の `\hline` という命令で罫線を引くと次のようになる。

```
\documentclass{jsarticle}
\usepackage{array}
\begin{document}
\begin{center}
\begin{tabular}{lrr}
\hline
品名 & 単価 (円) & 個数 \\
\hline
りんご & 100 & 5 \\
みかん & 50 & 10 \\
\hline
\end{tabular}
\end{center}
```

品名	単価 (円)	個数
りんご	100	5
みかん	50	10

このように、標準では線の太さが全て同じになる。`LATEX` 標準のものでは、指定列の文字列（この場合 `lrr`）の中に縦棒 (`|`) を入れると、次のように縦罫線を引くことができる。

```
\begin{tabular}{|l|r|r|} \hline
品名 & 単価 (円) & 個数 \\ \hline
りんご & 100 & 5 \\
みかん & 50 & 10 \\ \hline
\end{tabular}
```

出力は次のようになる。

品名	単価 (円)	個数
りんご	100	5
みかん	50	10

`\hline\hline` と続けて書くと 2 重の横罫線になる。また、列指定の中で `||` と書くと 2 重の縦罫線になる。

次のように、`\cline{欄番号-欄番号}` で部分的に罫線を引くことができる。

```
\begin{tabular}{|ccc|} \hline
こ & れ & は \\ \cline{2-3}
迷 & 路 & で \\ \cline{1-1} \cline{3-3}
し & よ & う \\ \hline
\end{tabular}
```

出力は次のようになる。

こ	れ	は
迷	路	で
し	よ	う

8.4 表の細かい制御

表の行送りや上下の罫線との距離は次のようにして自由に制御することができる。

まず、各行の最後の `\\` の後に `[長さ]` を付けると、その長さだけ行送りが増える。負の長さなら行送りが減る。但し、`\hline` のある行の行送りを減らすと横罫線が変な位置に来てしまう。

```
\begin{tabular}{|l|r|r|} \hline
品名 & 単価 (円) & 個数 \\ \hline
りんご & 100 & 5 \\[-5pt]
みかん & 50 & 10 \\ \hline
\end{tabular}
```

出力は次のようになる。

品名	単価 (円)	個数
りんご	100	5
みかん	50	10

全体の行送りを一定の割合で変えたい場合は、`\arraystretch` というマクロを再定義する。例えば、

```
\renewcommand{\arraystretch}{0.8}
```

で行送りが 0.8 倍になる。array パッケージで追加された `\extrarowheight` という長さを設定することで、行の高さを一律に増やすことができる。例えば、

```
\setlength{\extrarowheight}{2pt}
```

とすれば、行送りが一律 2pt 増える。

更に、特定の行だけ上下の空きを調節するには、第 3 章で説明した `\rule` を幅 0 にして挿入し、上下の罫線を押し上げ・下げるための支柱とする。例えば、

```
\rule[-1zw]{0zw}{3zw}りんご & 100 & 5 \\ \hline
```

のようにすれば、上下の罫線との間に全角の隙間が入る（値は微調整する必要がある）。

8.5 列割りの一時変更

一時的にいくつかの列をまとめて 1 列のように扱う命令は、

```
\multicolumn{まとめる列数}{列の指定}{中身}
```

である。例えば、

請求書		
品名	数量	金額
基礎からわかる情報リテラシー	1	1480 円
R で楽しむ統計	1	2500 円

のように出力するには、

```
\begin{center}
\begin{tabular}{lcr}
\multicolumn{3}{c}{\textgt{請求書}} \\
\multicolumn{1}{c}{品名} & 数量 & \multicolumn{1}{c}{金額} \\
基礎からわかる情報リテラシー & 1 & 1480 円 \\
R で楽しむ統計 & 1 & 2500 円
\end{tabular}
\end{center}
```

と入力する。ここで、

```
\multicolumn{3}{c}{\textgt{請求書}}
```

は 3 列分をまとめて中央揃え、ゴシック体で「請求書」と出力する。

```
\multicolumn{1}{c}{品名}
```

は単に「品名」を中央揃えに直しただけである。これに罫線を引いてみる。

請求書		
品名	数量	金額
基礎からわかる情報リテラシー	1	1480 円
R で楽しむ統計	1	2500 円

これを出力するには、

```
\begin{center}
\begin{tabular}{|l|c|r|} \hline
\multicolumn{3}{|c|}{\textgt{請求書}} \\ \hline
\multicolumn{1}{|c|}{品名} & 数量 & \multicolumn{1}{|c|}{金額} \\ \hline
基礎からわかる情報リテラシー & 1 & 1480 円 \\
R で楽しむ統計 & 1 & 2500 円 \\ \hline
\end{tabular}
\end{center}
```

と入力する。

8.6 横幅の指定

ある列の幅を例えば 5 cm に固定して左揃えにするには、1 の代わりに `p{5cm}` と指定する。中央揃えなら `c` の代わりに `>\centering p{5cm}`、右揃えなら `r` の代わりに `>\raggedleft p{5cm}` とする。

全体の横幅の定まった表は `tabular` の代わりに `tabularx` パッケージの `tablarx` を用いる。使い方は、プリアンブルに、

```
\usepackage{tabularx}
```

と書いておき、表を出力したい場所に、

```
\begin{tabularx}{幅}{列指定}
  ⋮
\end{tabularx}
```

と書く。幅を自由に変えて構わない列は `X` と指定する。例えば、

請求書		
品名	数量	金額
R で楽しむ統計	1	2500 円

のように横幅を 65 mm の幅にするには、

```
\begin{center}
\begin{tabularx}{65mm}{|X|r|r|} \hline
\multicolumn{3}{|c|}{\textgt{請求書}} \\ \hline
品名 & 数量 & 金額 \\ \hline
R で楽しむ統計 & 1 & 2500 円 \\ \hline
\end{tabularx}
\end{center}
```

とする。最初の列は `X` と指定されているので、幅は可変である。次の 2 つの列は `r` と指定されているので、右寄せとなる。`X` と指定された列は、中身が長くなっても適当に改行される。

8.7 色のついた表

`colortbl` パッケージを利用すれば、行ごと、列ごと、あるいは特定のセルに色を付けることができる。そこで、前述した `xcolor` パッケージを利用する場合は、

```
\usepackage[table]{xcolor}
```

のようにオプション `table` を付けて呼び出すと、`xcolor` 内部から `colortbl` パッケージが読み込まれる。

ここでは白に近いグレイ (`\color{gray}{0.8}`) を使って説明する。この 0.8 という数値は 0 (黒) と 1 (白) の間で選択する。まず、行全体の背景色の指定は次のようにして行う。

```
\begin{tabular}{|c|} \hline
\rowcolor{gray}{0.8} 第 1 の行 \\ \hline
第 2 の行 \\ \hline
\end{tabular}
```

列全体の背景色の指定は次のようにする。

```
\begin{tabular}{|>\columncolor{gray}{0.8}c|c|} \hline
最初の列 & 次の列 \\ \hline
最初の列 & 次の列 \\ \hline
\end{tabular}
```

行と列の指定がかち合うときは `\rowcolor` が勝つ。特定のセルだけに色をつけるには `\multicolumn` を用いる。

```
\begin{tabular}{|c|c|} \hline
\multicolumn{1}{|>{\columncolor[gray]{0.8}}c|}{左上} & 右上 \\ \hline
左下 & \multicolumn{1}{|>{\columncolor[gray]{0.8}}c|}{右下} \\ \hline
\end{tabular}
```

`\usepackage[table]{xcolor}` を用いると `\rowcolors` という命令も利用可能となる。例えば、3 列目以降は奇数行で黒 20%、偶数行で無色とするには `\rowcolors{3}{black!20}{}` とする。

8.8 ページをまたぐ表

`tabular` 環境は 1 つの大きな文字と同等に扱われるため、ページをまたぐことができない。ページをまたぐ表を作表するには `longtable` パッケージを用いる。

```
\begin{longtable}{|l|l|}
\hline 名前 & 住所 \\ \hline \endhead
\hline \endfoot
技評太郎 & 東京都新宿区市谷左内町 21-13 \\
..... & ..... \\
..... & .....
\end{longtable}
```

`\endhead` までの部分は各ページの表の頭に出力する。ここでは横線 (`\hline`)、名前、住所、改行 (`\\`) を出力している。その次から `\endfoot` までの部分は各ページの表の最後に出力するものである。ここでは横線だけになっている。それ以外は、通常の `tabular` 環境と同様である。

これを L^AT_EX で処理すると最初は、

```
Package longtable Warning: Table widths have changed. Return LaTeX.
```

というメッセージが画面に出力される。このメッセージが出なくなるまで繰り返し L^AT_EX を実行する。

8.9 表組みのテクニック

表の列間隔を変えるには `\setlength` を用いて `\tabcolsep` という変数の値を変更する。例えば、

```
この{\setlength{\tabcolsep}{3pt}\footnotesize
\begin{tabular}{|c|c|c|} \hline
2 & 9 & 4 \\ \hline
7 & 5 & 3 \\ \hline
6 & 1 & 8 \\ \hline
\end{tabular}} を 3 次の魔法陣という。
```

とすると、

この

2	9	4
7	5	3
6	1	8

 を 3 次の魔法陣という。

のように出力される。列間隔は `\tabcolsep` に設定した値 (上の例では 3 pt) の 2 倍 (6 pt) になる。元々の `\tabcolsep` の値は 6 pt (列間隔は 12 pt) である。

このように一時的に変数の値を変更する場合は、変更の命令と `tabular` 環境全体を中括弧 `{ }` で囲んでおく必要がある。もし、`tabular` 環境全体が `center` 環境などの中にあるなら、変数の値の変更はその環境の外に及ばないので中括弧で囲む必要はない。

なお、上の例では表は上下中央揃えになったが、

```
\begin{tabular}[b]{...}
```

とすると、表の下端が前後の文のベースラインに一致し、

この

2	9	4
7	5	3
6	1	8

 を 3 次の魔法陣という。

のように出力される。逆に、

```
\begin{tabular}[t]{...}
```

とすると、表の上端が前後の文のベースラインと一致する。

tabular 環境の前に例えば、

```
\setlength{\arrayrulewidth}{0.8pt}
```

と書いておくと、罫線の太さが 0.8 pt になる（元の値は 0.4 pt）。

```
\setlength{\doublerulesep}{0pt}
```

とすると 2 重罫線の間隔が 0 pt になる（元の値は 2 pt）。2 重罫線の間隔を 0 pt にすると `\hline\hline` や `{|c||}` のように罫線を重複指定することで罫線の太さを 2 倍にすることができる（array パッケージの場合）。

次の表のように小数点で桁揃えしたい場合や、微妙な文字間・行間の調整をしたい場合がある。

T (deg)	t (sec)	X_n
10^{12}	0	0.496
3×10^{11}	0.00129	0.488*
1.3×10^9	98*	0.15

このような場合は、`` と書くと「何々」と同サイズの空白が出力されることを用いるのが簡単である。また、`\rlap{何々}` とすれば右に向かって「何々」と出力してから、その幅だけ左に戻るなので、あたかも「何々」を出力しなかったような列揃えになる。

次の入力例は、`` と入力する代わりに `~` で数字の幅の空白が出力できるように `~` を `\renewcommand` で再定義している。center 環境内での再定義なので、center 環境を抜ければ `~` の定義は元に戻る。

```
\begin{center}
\renewcommand{~}{\phantom{0}}
\begin{tabular}{rlr} \toprule
\multicolumn{1}{c}{ $T$  (deg)} & & \\
\multicolumn{1}{c}{ $t$  (sec)} & & \\
\multicolumn{1}{c}{ $X_n$ } & & \\
$ 10^{12}$ & ~0 & 0.496 \\
$ 3 \times 10^{11}$ & ~0.00129 & 0.488\rlap{*} \\
$ 1.3 \times 10^9$ & 98* & 0.15~ \\
\end{tabular}
\end{center}
```

ここで `\[-4pt]` は、その行間を標準より 4 pt 狭くする命令である。欧文用のクラスファイルを使う場合は行間を狭くする必要はほとんどないが、和文用のクラスファイルでは行間が広く設定してあるので、このような数表を組むと行間が広くなりすぎる。3 ~ 4 pt 狭くするとよいだろう。

第 9 章

図・表の配置

L^AT_EX には自動で図・表を配置する `figure` 環境、`table` 環境が用意されている。この機能を強化した `float` パッケージを使うことにより、より柔軟な図・表の配置ができると共に、図・表に似た「プログラムリスト」などの新しい環境を簡単に作ることができる。

9.1 図の自動配置

図を自動配置するには `figure` 環境を用いる。例えば、

```
\ref{fig:2ji}は関数  $y = x^2$  のグラフである。
\begin{figure}
  \centering
  \includegraphics[width=5cm]{2ji.pdf}
  \caption{関数  $y = x^2$  のグラフ}
  \label{fig:2ji}
\end{figure}%
このグラフは下に凸である。
```

と書くと、L^AT_EX は `\begin{figure} ... \end{figure}%` の部分を取りあえず無視して、

図??は関数 $y = x^2$ のグラフである。このグラフは下に凸である。

と出力する（図の番号が ?? になっている）。そして、そのページの上か下の余ったところに図を出力し、図のすぐ下に、

図 1 関数 $y = x^2$ のグラフ

のように見出しを出力する。もし、そのページに収まらないようなら、次のページ以降に回される。

`\label{...}` と `\ref{...}` の中身は、単なる符丁^{ラベル}なので、何でも構わないが、両方に同じ文字列を書いておく必要がある。

「図??は……」のように、本文中で図の番号が ?? になってしまうが、これは L^AT_EX をもう一度実行すると、

図 1 は関数 $y = x^2$ のグラフである。このグラフは下に凸である。

のように正しい番号に置き換わる。`\label`、`ref` 及び L^AT_EX を 2 回実行することに意味については第 10 章を参照のこと。

`\begin{figure}[htbp]` のように `\begin{figure}` の直後に [] で囲んだ文字を追記することで、図の出力の可能な位置を指定することができる。これらの文字の意味は次の通りである。

- t ページ上端 (top) に図を出力する。
- b ページ下端 (bottom) に図を出力する。
- p 単独ページ (page) に図を出力する。
- h できればその位置 (here) に図を出力する。
- H 必ずその位置 (Here) に図を出力する (要 `float` パッケージ)。

何も指定しなければ [tbp]、すなわちページ上端、ページ下端、単独ページに出力できることになる。**htbp** を並べる順序には意味はない。**[b!]** のように ! を付けると、より強い指定となる。**H** は **float** パッケージを利用していないと使うことができない。また、**H** を指定すると、他のオプションは指定することができなくなる。

`\caption{...}` は図の説明を出力する命令だが、これを用いて図目次を自動的に作成することもできる。図目次を作成するには、図目次を出力したい場所に `\listoffigures` 命令を記述する。

図目次を作成する場合には、

```
\caption[短い説明]{長い説明}
```

のように 2 通りの説明を付けることができる。長い説明は図の下に、短い説明は図目次に使われる。短い説明がない場合は、長い説明が図目次にも使われる。目次については第 10 章の第 2 節も参照のこと。

9.2 表の自動配置

表の自動配置には `table` 環境を用いる。`figure` 環境と同様に自動的に適当な位置に配置され、「表 1」「表 2」... といった番号が付く。

`table` 環境の使い方は `figure` 環境の場合と全く同じである。但し、表の場合はキャプションを上を書くというルールがあるので、次のように `\caption` は上に配置する。

魔法陣

```
\begin{table}
  \caption[3 次の魔法陣]{3 次の魔法陣の例。縦・横・斜めの和がいずれも 15 である。}
  \label{mahou}
  \begin{center}
    \setlength{\tabcolsep}{3pt}\footnotesize
    \begin{tabular}{|c|c|c|} \hline
      2 & 9 & 4 \\ \hline
      7 & 5 & 3 \\ \hline
      6 & 1 & 8 \\ \hline
    \end{tabular}
  \end{center}
\end{table}%
では、縦・横・斜めの和が等しい。
```

9.3 左右に並べる配置

独立な図を左右に並べて配置するには、次のように `minipage` 環境を用いるのが簡単である。ここで `\columnwidth` は版面の幅（段組の場合は段の幅）である。版面の幅の 0.4 倍の小さなページを作って左右に並べる。`minipage` の中では `\columnwidth` が `minipage` の幅になる。

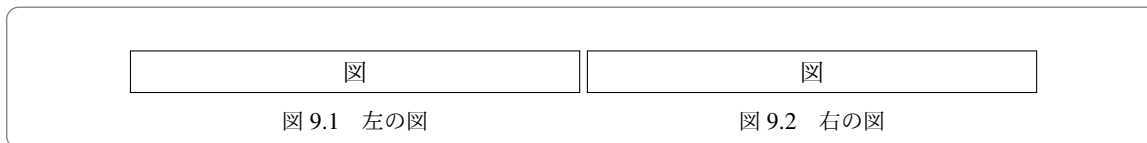
```
\begin{figure}
  \centering
  \begin{minipage}{0.4\columnwidth}
    \centering
    \includegraphics[width=\columnwidth]{1.pdf}
    \caption{左の図}\label{fig:左}
  \end{minipage}
  \begin{minipage}{0.4\columnwidth}
    \centering
```

```

\includegraphics[width=\columnwidth]{r.pdf}
\caption{右の図}\label{fig:右}
\end{minipage}
\end{figure}

```

次のように出力される。



関連した複数の図を並べるには subcaption パッケージを用いるのが便利である。プリアンプルには、

```
\usepackage{subcaption}
```

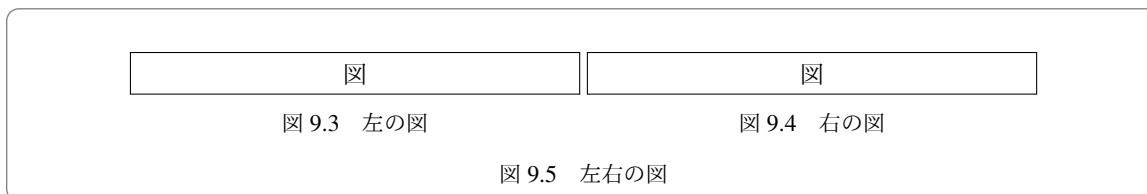
と書いておき、

```

\begin{figure}
\centering
\begin{subfigure}{0.4\columnwidth}
\centering
\includegraphics[width=\columnwidth]{l.pdf}
\caption{左の図}\label{fig:l}
\end{subfigure}
\begin{subfigure}{0.4\columnwidth}
\centering
\includegraphics[width=\columnwidth]{r.pdf}
\caption{右の図}\label{fig:r}
\end{subfigure}
\caption{左右の図}
\label{fig:rl}
\end{figure}

```

とすれば、次のように出力される。



この場合、`\ref{fig:l}`、`\ref{fig:r}`、`\ref{fig:rl}` で出力されるものはそれぞれ 9.3、9.4、9.5 のようになる。

見ての通り、デフォルトでは図が隣接してしまうので、これがまずい場合は `\subfigure` 間に例えば `\hspace{+5mm}` のように適当な水平方向のスペースを入れる。

9.4 図・表が思い通りの位置に出力されない場合

昔の `jarticle` などと比べて、今日の `jsarticle` などは図・表が入りやすい設定になっている。図・表がうまく配置できない場合、`LATEX` は「Too many unprocessed floats」（未処理の図や表が多すぎる）というエラーを出すことがある。この場合は、`figure` や `table` に `H` オプションを付けて出力する位置を明示的に指定する。

9.5 回り込み

図や表のまわりに文章を回り込ませるためのパッケージはいくつか存在するが、ここでは `wrapfig` パッケージについて説明する。

`\begin{wrapfigure}[行数]{l または r}{幅}` で図を配置する（表の場合には `wrapfigure` の代わりに `wraptable` 環境が用意されている）。`l` で左、`r` で右に図が配置される。

```
\begin{wrapfigure}{r}{8zw}
  \vspace*{-\intextsep}
  \includegraphics[width=8zw]{tiger.pdf}
\end{wrapfigure}
```

とすると …

このように右に虎の絵が出て、文章が回り込む。行数は指定しなくても自動で計算される。文章と図の水平距離は `\columnsep`（段組の場合の段間の空き）と同じになる。`jsarticle`、`jsbook` では段間の空きは全角の整数倍になっているので、図の幅も全角の整数倍にしておく方が、本文の余計な伸び縮みが起きないのでよいだろう。また、垂直方向には `\intextsep`（図と本文の垂直方向の空き、標準では $12\text{pt} \pm 2\text{pt}$ ）だけ空きが入るので、段落の切れ目に置いた場合、先程の例のように `\intextsep` だけ戻るとほぼ段落の上端と一致する。



回り込みの位置でちょうどページが分割される場合や、箇条書きなどの環境と重なるとうまく処理されないので注意が必要である。また、`float` パッケージと併用する場合は、`float` の後に `wrapfig` を読み込む必要がある。

第 10 章

相互参照・目次・索引・リンク

L^AT_EX で作成した文書には自動的に目次を付けることができる。また、`\label`、`\ref`、`\pageref` といった命令を用いると、章・節・図・表などの番号・ページを参照することができる。更に、索引に載せたい語句に `\index` という命令を付加しておけば、MakeIndex（または mendex、upmendex）というソフトを併用することにより索引を自動的に作成することができる。

10.1 相互参照

相互参照とは「5.3 節を参照されたい」とか「結果は 123 ページの図 10.5 のようになった」のように、ページ・章・節・図・表・数式などの番号を入れることである。

L^AT_EX を使えば、この相互参照が簡単に行える。まず、参照したい番号出力する命令の直後に、次のようにしてラベル（名札）を貼っておく。

入力	<code>\section{文書処理とコンピュータ}</code> <code>\label{bunsho}</code> <code>\subsection{\LaTeX による文書処理}</code> <code>\label{labun}</code>
出力	5 文書処理とコンピュータ 5.1 L ^A T _E X による文書処理

見ての通り `\label{...}` は出力には直接影響しない。しかし、これで「文書処理とコンピュータ」というセクションには `bunsho` というラベルが貼られ、「L^AT_EX による文書処理」というサブセクションには `labun` というラベルが貼られた。

ラベルは `label{bunsho}` のような半角文字でも `label{文書}` のような全角文字でも構わない。但し、ラベル中で半角の `\{ }` の 3 文字は使用することができない。また、同じラベルを 2 箇所に貼ることはできない。大文字と小文字は区別される^{*1}。

ラベル名としては、章のラベルには `ch:bunsho`、節のラベルには `sec:LaTeX`、図のラベルには `fig:zu`、式のラベルには `eq:Eular` のように系統的に命名するとよいだろう。

先程の例では「L^AT_EX による文書処理」という節に `labun` というラベルを貼ったが、その前でも後でも、この「L^AT_EX による文書処理」という節を参照したいところがあれば、次のようにする。

入力	この件については <code>\ref{labun}</code> 節（ <code>\pageref{labun}</code> ページ）を参照されたい。
出力	この件については 5.1 節（123 ページ）を参照されたい。

すなわち、`labun` というラベルを貼った節番号を出力したい場所には `\ref{labun}` と記述し、ページ番号を出力したい場所には `\pageref{labun}` と記述する。ところが `\ref` や `\pageref` を用いた際は、L^AT_EX を 1 回実行しただけでは正しい出力が得られない。

^{*1} 実際には、大文字と小文字だけ異なるラベル名などは、紛らわしいので使わないことを推奨する。

L^AT_EX の 1 回目の実行では、次のような警告メッセージが表示される。

```
LaTeX Warning: Reference 'labun' on page 1 undefined on input line 8.  
LaTeX Warning: There were undefined references.  
LaTeX Warning: Label(s) may have changed. Rerun to get cross-references right.
```

これは「相互参照を正しくするためにもう一度実行して下さい」ということである。もし、この警告を無視して dvi ファイルを出力すると、節番号・ページの部分が伏せ字 (“??”) になってしまう。

通常は 2 回目の L^AT_EX の実行で警告が出なくなる。しかし、2 回目の実行で正しい番号を埋めた際、肝心のページ番号がずれてしまうことがあり得る。

また、1 回目の実行の後で文書ファイルに手を加えた際も、ページ番号が合わなくなる。これらの場合は伏せ字にはならないが、完全に辻褄が合うまで、

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

という警告メッセージが出る。この警告メッセージが出なくなるまで実行を繰り返す必要がある。

また、2 回目以降の実行でも、

```
LaTeX Warning: There were undefined references.
```

の警告が出る際は、`\ref` や `\pageref` に対応する `\label` がない場合である。

10.2 目次

L^AT_EX で目次を出力するのはとても簡単である。目次を出力したい場所（例えば、序文の後）に `\tableofcontents` と記述するだけである。同様に、`\listoffigures`、`\listoftables` という命令で図目次、表目次を出力することができる。

但し、これらの目次を出力するには、前節で述べた事と同様の理由で、文書ファイルを L^AT_EX で少なくとも 2 回処理しなければならない。

目次にどのレベルまでの見出しを出力するかはドキュメントクラスによって決まっているが、変更は容易である。出力するレベルを `\section` までにするなら、

```
\setcounter{tocdepth}{1}
```

`\subsection` までにするなら、

```
\setcounter{tocdepth}{2}
```

とプリアンブルに記述しておく。

10.3 索引と MakeIndex、mendex、upmendex

索引を用意するには、昔はまず本文から重要な語句を拾い出してカードに書き、それを五十音順に並べ替えていた。これは大変な作業で、よく間違いが生じた。

MakeIndex というソフトを L^AT_EX を組み合わせて用いることで、文書ファイル中で索引に載せたい語に `\index` という命令を付けておくだけで、自動的に索引が作成される^{*2}。

MakeIndex を日本語化したものが mendex、upmendex である。以下では、mendex を用いた索引の作り方を説明するが、upmendex でも全く同様である。

^{*2} MakeIndex 以外に xindy というソフトが存在する。

10.4 索引の作り方

例えば、次のような文章を考える。

ピッツィカートすべき個所の指定は、楽譜の上では pizz と書かれ、またもとどおりに弓でひく箇所
に、イタリア語で acro（弓）と書くことになっています。

この中で、

ピッツィカート pizz 弓 acro

の 4 つの語を索引に載せたいとする。それには `\index{...}` という命令を用いる。この命令は、半角アルファベット
や平仮名・片仮名だけの索引語は、

```
\index{pizz}
```

のように `\index{索引語}` の形で用いる。漢字を含む索引後は、

```
\index{ゆみ@弓}
```

のように `\index{よみかた@索引語}` の形で用いる。更に、文書ファイルのプリアンブルに

```
\usepackage{makeidx}  
\makeindex
```

と記述しておく。また、索引を出力したい場所（大抵は文書の最後）に `\printindex` と記述しておく。

先程の例に索引語の指定などを書き加えると次のようになる。

```
\documentclass{jsarticle}  
\usepackage{makeidx}  
\makeindex  
\begin{document}  
ピッツィカート\index{ピッツィカート}すべき個所の指定は、  
楽譜の上では pizz\index{pizz} と書かれ、  
またもとどおりに弓\index{ゆみ@弓}でひく箇所に、  
イタリア語で acro\index{acro}（弓）と書くことになっています。  
  
\printindex  
\end{document}
```

この文書ファイルを `ongaku.tex` という名前で保存し、まず \LaTeX で通常通りに処理する。すると、`ongaku.tex` と同じ
ディレクトリに `ongaku.idx` というファイルが作成される（これは `\makeindex` の仕業であり、`\usepackage{makeidx}`
や `\printindex` はこの段階では特に意味を持たない）。

次に `mendex` でこの `ongaku.idx` ファイルを処理する。コマンドなら、

```
$ mendex ongaku.idx
```

のように打ち込むことになる。これで `mendex` は `ongaku.idx` ファイルをアルファベット・50 音順に並べ替え、`ongaku.ind`
というファイルに出力する。最後に、もう一度 \LaTeX で処理すると `\printindex` コマンドが `ongaku.ind` を読み込んで、
その場所に索引が挿入される。

10.5 ハイパーリンク

米国 Los Alamos National Laboratory 発祥の有名な物理学のプレプリントサーバ e-Print archive^{*3} では \TeX を用いたプ
レプリントを蓄積しているが、その相互参照を容易にするために \HyperTeX という仕組みが開発された^{*4}。

^{*3} <http://arxiv.org>、現在は Cornell University 内

^{*4} <http://arxiv.org/hypertext/>

これは当時 World Wide Web で使われ始めたハイパーリンクの仕組みを TeX に持ち込んだものである。現在 HyperTeX の仕組みを利用するために広く用いられているのは hyperref パッケージである。

プリアンブルには、できるだけ後の方に⁵

```
\usepackage[ドライバ名]{hyperref}
```

と記述しておく。ドライバ名には dvipdfmx、pdfTeX、dvips、tex4ht、xetex、hypertex などが指定できる。最後の hypertex は HyperTeX 拡張をサポートする一般のドライバ用である (dviout はこれを指定する)。LuaTeX は pdfTeX でも大丈夫だが、葉が化けるのを防ぐために unicode=true というオプションも付加する。

これで、リンクを貼りたい箇所に、例えば、

```
HyperTeX は \href{http://arxiv.org/}{arXiv} で開発された。
```

のように記述すれば、その部分がリンクとなる。

hyperref パッケージで使える主な命令をまとめておく。

- `\href{URL}{テキスト}` は HTML の `テキスト` に相当し、リンクを作成する。
- `\hyperbseurl{URL}` は `<base href="URL">` に相当し、相対 URL の基準を指定する。
- `\hyperimage{URL}` は `<image src="URL">` に相当し、画像を挿入する。
- `\hyperlink{名前}{テキスト}` は `テキスト` に相当し、文書内へのリンクを作成する。
- `\hypertarget{名前}{テキスト}` は `テキスト` に相当し、文書内からのリンクの飛び先となる。

hyperref では、

```
\url{http://www.gihyo.co.jp/}
```

のように書いた URL もリンクとなる。この url パッケージは単独で用いても URL の表記に便利である。`\url` コマンドは `\verb` に似ているが、長い URL は途中で改行することができる。長いファイル名を書く際にも便利である。

⁵ いろいろなコマンドを書き換えるので、できるだけ後で指定しないと他のパッケージの影響を受けてしまう。

第 11 章

文献の参照と文献データベース

本などで読んだことについて書く際は、原文を引用する・しないに関わらず、出典を明記するのが読者へのサービスであると同時に著者への礼儀である。これを怠ると、法的にも道義的にも責任を問われかねない。文献の参照・引用の仕方を学ぶことは、非常に大切なことで、大学で習うレポート・論文の書き方の大きな部分を占めている。

ここでは、文献の参照法から BibTeX、pBibTeX などを用いた文献データベースの構築法までを解説する。

11.1 文献の参照

文献の参照法にはいろいろな流儀が存在するが、基本的には書籍であれば著者名・書名・出版者（出版社名）・出版年を挙げる。横書きの文書での文献の参照の仕方は、木下是雄『理科系の作文技術』^{*1}の 9.4 節に簡潔にまとめられている。また、欧文文献の参照については van Leunen の *A Handbook for Scholars*^{*2} に非常に詳しく書かれている。LaTeX による参考文献の扱いは、主にこの書籍に依っている。因みに、van Leunen は数学にも詳しい英語学者で TeX の作者である Knuth 教授の講義録『クヌース先生のドキュメント纂法』^{*3}にもゲスト講師として登場している。

Van Leunen が推奨する方式では、参考文献リストは文書の最後に、通し番号を付けて並べる。そして、本文中では、

Van Leunen の *A Handbook for Scholars* [3] によれば……
Van Leunen [3] によれば……
……である [3–5, 7].

などのように、参照すべき文献の番号を [] で囲んで付加する（最後の例は文献番号 3, 4, 5, 7 の文献を参照すべきことを表している）。この [3] などは括弧書きに過ぎず、

[3] によれば……

のように名詞として用いるのは正しくない（実際にはあまりまもられていないが）。

文書の最後に付加する文献リストは、本文で出現する順に並べることもよくあるが、Van Leunen の流儀では、第 1 著者の姓のアルファベット順に並べる。第 1 著者の姓が同じ場合は、名のアルファベット順に並べる。第 1 著者が同じなら、第 2 著者の姓、名、……、と比較していき、著者が全く同じなら文書のタイトルのアルファベット順に並べる。アルファベット順とは、アポストロフィを無視し、ハイフンをスペースと見なし、スペース、A, B, ..., Z の順に並べることである。文書のタイトルの最初の冠詞（a, the）は無視する。

また、有名な *The Chicago Manual of Style*^{*4} の流儀では、本文中には (Knuth 1991) のように著者名と出版年を並べる。同じ年のものがいくつも存在する場合は、1983a, 1983b のなどとする。この流儀の利点は、文献の加除があっても参照番号を振り直す必要が（殆ど）ないことだったが、現在では LaTeX などのシステムが自動的に参照番号を振ってくれるので、有り難みが薄れた。番号だけより情報量が多く、文献が推測しやすいという利点はあるが、参照文献が多いと逆にうるさく感じる。変形として [Knu 91] のような短い形もよく用いられる。

^{*1} 木下是雄『理科系の作文技術』中公新書 624（中央公論社、1981）

^{*2} Mary-Claire van Leunen, *A Handbook for Scholars*. Alfred A. Knopf, 1978; Oxford University Press, 1992.

^{*3} Donald E. Knuth, Tracy Larrabee, and Paul M. Roberts. *Mathematical Writing*. MAA Notes No.14 The Mathematical Association of America, 1989; 有澤誠訳『クヌース先生のドキュメント纂法』（共立出版、1989）

^{*4} *The Chicago Manual of Style*, 16th edition, University of Chicago Press, 2010. オンライン版も存在する。

分野によっては、参考文献は脚注に記載する。この場合、同じ文献を続けて参照する際は *ibid.*（「同所」の意味のラテン語）と記述する。日本では ^{シスト}SIST（科学技術情報流通技術基準、<https://jipsti.jst.go.jp/sist/>）で参考文献の書き方が提案されている^{*5}。実際には、論文を投稿する学会誌ごとに文献の参照の仕方が決まっているので、それに従わなければならない。

さて、 \LaTeX では、参考文献は次の 3 通りの扱い方がある。

- 文献リストも参照番号付けも人間が行う方法
- 文献リストは人間が作成し、参照番号をコンピュータに付けさせる方法
- 文献データベースに基づいて全てをコンピュータ化する方法

各方法を以下の節で順に解説する。

11.2 全て人間が行う方法

\LaTeX で文献リストを出力するには `thebibliography` 環境というものをを用いる。例えば、

参考文献

- [1] 木下是雄『理科系の作文技術』中公新書 624（中央公論社、1981）
[2] Mary-Claire van Leunen. *A Handbook for Scholars*. Alfred A. Knopf, 1978.

のような文献リストを出力するには、

```
\begin{thebibliography}{9}  
\item  
  木下是雄『理科系の作文技術』  
  中公新書 624（中央公論社、1981）  
\item  
  Mary-Claire van Leunen.  
  \emph{A Handbook for Scholars.}  
  Alfred A. Knopf, 1978.  
\end{thebibliography}
```

のように入力する。

上の例で `\begin{thebibliography}{9}` の “9” は参考文献に付ける番号が 1 桁以内であることを表す。2 桁以内なら “99”、3 桁以内なら “999” などとする。

もし、番号を [1], [2], [2a], [3], ... のように付けたい場合は、

```
\begin{thebibliography}{9a}  
\item ...  
\item ...  
\item [{[2a]}] ...  
\item ...  
\end{thebibliography}
```

のようにする。通常、この参考文献リストは文書の最後（または各章の最後）に付加する。本文中で参照する際は、この方法では、

Knuth~ [2] によれば……であることが知られている~ [3--5, 7]

のように自分で番号を付加する（行分割を行わない空白 ~ を用いる）。

^{*5} JST の SIST 事業は 2011 年度末に終了した。

11.3 半分人間が行う方法

参考文献リストは人間が作成し、参照番号はコンピュータに付けさせる方法である。まず、先程と同様な文献リストを作成するのだが、ここでは `\item` の代わりに `\bibitem` という命令を用いる。`\bibitem` の直後には `{ }` で囲んで適当な参照名を付けておく。

```
\begin{thebibliography}{9}  
  \bibitem{木是}  
    木下是雄『理科系の作文技術』  
    中公新書 624 (中央公論社, 1981)  
  \bibitem{leu}  
    Mary-Claire van Leunen.  
    \emph{A Handbook for Scholars.}  
    Alfred A. Knopf, 1978.  
\end{thebibliography}
```

上の例では、木下是雄氏の本には“木是”という参照名を、van Leunen の本には“leu”という参照名を付けた。これは覚えやすいものなら何でも構わない。例えば、“leu-handbook”や“木下：作技”や“木下 81”のような付け方も考えられる。参照名の中に空白やコンマを含めることはできない。大文字・小文字は区別されるので、leu と Leu は異なる本のことになってしまう（しかし、後述の BibTeX は参照名の大文字・小文字を無視するので、leu と Leu のような紛らわしい名前は付けない方がよいだろ）。

こうして文献リストを作成しておき、本文中で文献を参照する際には番号ではなく参照名を用いる。例えば、

木下 [1] や van Leunen [2] は……

と出力するには、

木下~\cite{木是} や van Leunen~\cite{leu} は……

とする（行分割をしない空白 ~ を用いる）。こうすれば、参考文献が加除されたり、順序が変わったりすると参照番号も自動的に更新される。

但し、このように `\cite` と `\bibitem` を用いた相互参照のある文章を処理するには、`LaTeX` を少なくとも 2 回実行しなければならない。最初に `LaTeX` で処理すると、次のような警告（warning）が出力される。

```
LaTeX Warning: Citation '木是' on page 1 undefined on input line 8.  
LaTeX Warning: Citation 'leu' on page 1 undefined on input line 8.  
LaTeX Warning: Label(s) may have changed. Rerun to get cross-references right.
```

すなわち「参照番号を正しくするために再実行しろ」というわけである。そこで、もう一度このファイルを `LaTeX` で処理すると、今度は警告メッセージが出力されないし、正しく参照番号が出力される。

次回からは、文献と番号の対応が変化していなければ、文書ファイルを編集しても `LaTeX` での処理は 1 回で構わない。

尚、例えば、

……であると言われている [1, 2].

のように複数の文献を参照するには、

……であると言われている~\cite{木是,leu}

のように半角コンマで区切る。また、例えば、

木下 [1, 161–167 ページ] や van Leunen [2, pp.9–44] によると……

のようにページ数などの補助情報を付加するには、

木下~\cite[161--167 ページ]{木下} や van Leunen~\cite[2, pp.9-44]{leu} によると…
...

のように \cite[補助情報]{参照名} の要領で記述する。

文献リストは通し番号以外に好きな「番号」を付けることができる。例えば、

参考文献

[leu78] Mary-Claire van Leunen. *A Handbook for Scholars*. Alfred A. Knopf, 1978.

[木下 81] 木下是雄『理科系の作文技術』中公新書 624（中央公論社, 1981）

のように出力するには、

```
\begin{thebibliography}{木下 99}
  \bibitem[leu78]{leu}
    Mary-Claire van Leunen.
    \emph{A Handbook for Scholars.}
    Alfred A. Knopf, 1978.
  \bibitem[木下 81]{kino}
    木下是雄『理科系の作文技術』
    中公新書 624（中央公論社, 1981）
\end{thebibliography}
```

とする。ここで \begin{thebibliography}{木下 99} の“木下 99”は最も長い文献の「番号」である（長すぎても構わない）。

11.4 cite と overcite

連続した文献を引用すると [1, 2, 3] のようになってしまう。これを [1–3] のようにするには cite というパッケージを用いる。すなわち、文書ファイルのプリアンブルに、

```
\usepackage{cite}
```

と書いておく。約物は標準で欧文用となるので、和文用にするには例えば次のように再定義する。 \inhibitglue は全角文字間の余分なグルー（スペース）を抑制する命令である。

```
\usepackage{cite}
\renewcommand{\citeleft}{\inhibitglue []} % ← 左括弧を全角 [ に
\renewcommand{\citeright}{\inhibitglue ]} % ← 右括弧を全角 ] に
\renewcommand{\citemid}{\inhibitglue {, }} % ← 引用番号と補助情報の区切りを全角コンマに
\renewcommand{\citepunct}{\inhibitglue ,} % ← 引用番号間の区切りを全角コンマに
```

本文中の引用番号を、梅棹³ のように上付きにするには、cite の代わりに overcite パッケージを用いる。

11.5 文献処理の全自動化

L^AT_EX と組み合わせて文献データベースから自動的に文献リストを作成するための Bib_TE_X というツール（Oren Patashnik 作、コマンド名 bibtex）が存在する。これを松井正一氏が日本語化したのが jBib_TE_X である。現在では p_TE_X 用のものが pBib_TE_X という名前で配布されている（コマンド名 pbibtex）。Unicode 版の upBib_TE_X（コマンド名 upbibtex）も開発されている。

この章の残りでは ((u)pBib_TE_X) の使い方と文献データベースの作り方を説明する。詳しくは順を追って説明するが、処理の流れは、

- ① 文書ファイルと文献データベースを用意する。
- ② L^AT_EX を実行する（参照情報を aux ファイルに書き出す）
- ③ B_IB_TE_X を実行する（bbl ファイルを作成する）
- ④ L^AT_EX を実行する（bbl ファイルを取り込む）
- ⑤ L^AT_EX を実行する（相互参照を解決する）

ようになる。原稿を手直しする度に、これだけの回数実行しなければならないわけではない。これ以降は、文献リストに加除がないなら L^AT_EX を 1 回だけ実行すれば十分である。

11.6 文献データベース概論

昔は文献カードを作成するのが研究者の仕事の 1 つであった。カード作りの考え方については梅棹忠夫氏『知的生産の技術』⁶が古典的な名著である。

しかし、今やノートパソコンを図書館に持ち込んでノートをとる時代、更に進んで論文はほとんど「e ジャーナル」になってパソコンで読める時代である。参考にしたい文献を見つけたら、その場で自分用の文献データベースに登録するようにするとよい。

文献専用のデータベース・ソフトもいろいろ作成されているが、テキストエディタでテキストファイルに書き込む程度で十分である。単純な検索にはエディタの検索機能が使えるし、Ruby、Python などの軽量言語を用いれば複雑な加工もできる。それに、テキストファイルならコンピュータ（環境）に依存しないので安心である。

文献データベースファイル（bib ファイル）のファイル名は拡張子を bib にする。これは文献（bibliography）の綴りから取った名前である。例えば、butsuri.bib、suugaku.bib などというテキストファイルに、文献カードに書くような内容を書き込む。具体的には、次のような流儀で @Book の中に参照名、著者名、書名、出版社名、出版年などを書いておくのが B_IB_TE_X の流儀である。欄を並べる順番は自由である。

```
@book{leunen, % ← leunen が参照名
  author = "Mary-Claire van Leunen",
  title  = "A Handbook for Scholars",
  publisher = "Alfred A. Knopf",
  year   = "1978",
  memo   = "何でもメモっておける"}
@book{木下：作技,
  yomi   = "Koreo Kinoshita", % 読みは名・性の順で統一
  author = "木下 是雄",
  title  = "理科系の作文技術",
  series = "中公新書 624",
  publisher = "中央公論社",
  year   = "1981"}
```

⁶ 梅棹忠夫『知的生産の技術』岩波新書青版 722（岩波書店、1969 年）

第 12 章

美しい文書を書くために

ちょっとした書き方の違いで、素人と玄人の違いが出てしまう。また、`LaTeX` は自動的に殆どのことをやってくれるが、最終工程で視覚的な調整を行うと見栄えが格段と変わる。本章では、そのような工夫をまとめてみた。

12.1 全角か半角か？

全角文字とその半分の幅の半角文字しかなかった時代は終わり、今では「半角」「全角」文字と言っても欧文のプロポーション文字を指すことが多くなった。

全角文字	WindowsでWordを使う。
昔の半角文字	WindowsでWordを使う。
今の「半角」文字	WindowsでWordを使う。

全角・半角という呼び方が相応しくなくなったため、和文文字・欧文文字、あるいは2バイト文字・1バイト文字という呼び方もされるようになったが、Unicode にするとバイト数も変わってしまう。

呼び方はともかく、少なくとも `TeX` では、欧文や数字には欧文（半角）文字を使うのが鉄則である。意見が分かれるのは「C 言語」「朝 8 時」のような和文の文脈で使われる 1 文字だけの英数字で、この場合は全角を使いたくなるかもしれないが、一般には欧文文字で統一する方がよいだろう。

半角	午後 5 時 55 分	BASIC から C 言語へ
全角	午後 5 時 5 5 分	B A S I C から C 言語へ
1 文字だけ全角	午後 5 時 55 分	BASIC から C 言語へ

12.2 句読点・括弧類

句読点や括弧の類は、和文では和文用（全角）を使うのが一般的である。欧文用を使うと、全角を基本とする文字の並びが乱れてしまう。

全角読点・全角句点	地球は、青かった。
全角コンマ・全角句点	地球は，青かった。
全角コンマ・全角ピリオド	地球は，青かった．
半角コンマ・半角ピリオド	地球は, 青かった.

特に括弧では、欧文用のものは `g` などの下の部分（descender）をカバーするために下に伸びているため、和文で使うと下にずれて見える。

全角括弧	括弧（かっこ）だ。
半角括弧	括弧(かっこ)だ。

12.3 引用符

和文用の引用符には、かぎ括弧「」、二重かぎ括弧『』などがある。欧文用の“ダブルクォート”に相当するものとして、和文用（全角）の“ダブル引用符”もよく用いられる*¹。

*¹ 昔の min10 などのフォントメトリックでは、バグのため全角の“引用符”の位置がおかしくなっていた。

12.4 疑問符・感嘆符

疑問符「？」や感嘆符「！」は、もともと日本語にはないもので、その扱いにも揺れがあるが、全角ものを用い、縦書きの場合には直後に全角（1zw）の空きを入れるのが標準的な組み方である。横書きの場合は、特にルールはないが半角（0.5zw）程度の空きが適当なようである。

現在広く用いられている縦組用和文フォントメトリックの tmin10 や、otf パッケージの縦組用和文フォントメトリックでは「？」「！」の後の全角空きが自動的に挿入される。横組では、otf パッケージのフォントメトリックでは半角の空きが挿入される。後続の文字が句読点・終わり括弧（閉じ括弧類）・リーダー類（一、…、…）の場合は空きは挿入されない。

横書きの jis フォントメトリックでは空が入らないので「？」「！」の直後には半角スペースを入れるなどの工夫が必要である。

jis フォントメトリック	あら！ ほんと？ ウッソー！！
jis フォントメトリック + 半角スペース	あら！ ほんと？ ウッソー！！
otf フォントメトリック	あら！ ほんと？ ウッソー！！

otf パッケージのフォントメトリックで「？」「！」の直後に空きを入れないようにするには \<（または \inhibitglue）という命令を入れる。

あっ！ \<と驚く。 → あっ！と驚く。

12.5 自動挿入されるスペース

前述したように、和文のフォントメトリックによっては「？」「！」の直後に自動的にグルー（glue：伸縮するスペース）が入る。グルー以外にも、和文のフォントメトリックによってカーン（kern：伸縮も行分割もしないスペース）が挿入される場合がある。

和文フォントメトリックからのグルー・カーンの挿入がない和文文字間には、\kanjiskip というグルーが自動挿入される。この \kanjiskip の量はドキュメントクラスなどで定義されているが、段落ごとに自由に変更することができる。例えば、

```
\setlength{\kanjiskip}{0zw plus 0.15zw minus 0.05zw}
```

とすれば、 $0_{-0.05}^{+0.15}zw$ のグルーが挿入される。

この値は、段落（または \nbox{...} などの箱）が閉じた時点での値が、段落（または箱）全体に適用されるため、次のような使い方の場合は中括弧を閉じる前に段落の区切りとなる空行（または、それと同じ意味を持つ par という命令）を入れておく必要がある。

`{\setlength{\kanjiskip}{0.5zw}すかすかに組む\par}` → すかすかに組む

また、和文・欧文間には \xkanjiskip というグルーが挿入される。この値は伝統的には全角の 1/4 に設定され、 \LaTeX 標準では $0.25zw \pm 1pt$ に設定されているが、雑誌などではゼロ、またはそれに近い値に設定されていることがよくある。

数学 I 実力テスト → 数学 I 実力テスト

`\mbox{\setlength{\xkanjiskip}{0zw}数学 I 実力テスト}` → 数学I実力テスト

注意しなければならないことは、和文・欧文間に半角文字を入れたり入れなかったりすると、スペースの量がまちまちになるかもしれないことである。半角空白のスペース（欧文の単語間スペース）の量は欧文フォントによって異なり、10 ポイントの Computer Modern Roman フォント（cmr, lmr）では $3.33_{-1.11}^{+1.66}pt$ と広いのだが、昔からの Times や Palatino 相当フォント（ptm, ppl）なら $2.5_{-0.6}^{+1.5}pt$ 、 \TeX Gyre 版（qtm, qpl）なら $2.5_{-0.83}^{+1.25}pt$ 程度、更に新しい Nimbus15 の Times 相当フォントでは $2.5_{-1}^{+2}pt$ である。

jsarticle や jsbook では \xkanjiskip を ptm, ppl の単語間スペースと同じ $2.5_{-0.6}^{+1.5}pt$ に設定してあるので、これらを用いる際は和欧文間には空白を入れても入れなくてもスペースの量は同じだが、Computer Modern フォントを使う際は、どちらかに統一する方がよいだろう。但し、同じ Palatino でも pplj, pplx の語間は $2.91_{-0.70}^{+1.75}pt$ である。

12.6 アンダーライン

“`\underline{何々}`”と入力すれば何々のように下線（アンダーライン）を引くことができる。しかし、アンダーラインはタイプライター時代の遺物であり推奨しないというのが \TeX の作者 Knuth 教授の考え方である。そのためもあって、 \TeX の`\underline`はごく単純な作りになっており、いったん箱で囲むので伸び縮みや途中での改行ができない。強調は欧文なら *italic* 体や **boldface**、和文なら圏点や太字、ゴシック体を用いるのがよいだろう。

12.7 欧文の書き方

欧文の場合に必要な、いくつかの注意点をまとめておく。

12.7.1 スペースの入れ方

欧文は、当然ながら欧文（半角）文字で記述する。単語の区切りに半角空白を入れるのは当然だが、コンマやピリオドの後、括弧の外側にも必ず空白を入れる。但し、句読点・疑問符・感嘆詞の直前、括弧・引用符の内側には空白を入れない。

(誤) Red,green,and blue are colors(or colours).

(正) Red, green, and blue are colors (or colours)

(誤) Red, green, and blue are colors (or colours)

この単純なルールのおかげで、欧文では空白のあるところではどこでも改行することができ、特別な禁則処理（句読点や閉じ括弧が行頭に来ない処理）は不要である。

12.7.2 引用符

欧文の“Quote!”のようなダブルクォートは、左シングルクォート（バッククォート）2個と右シングルクォート2個で囲んで書く。

(誤) He said, "Hello." → He said, "Hello."

(正) He said, ‘‘Hello.’’ → He said, “Hello.”

次のような場合は、小さいスペースを出力する命令`\`、を区切りに入れる。

‘‘He said, ‘Hello.’\,’’ → “He said, ‘Hello.’”

‘He said, ‘‘Hello.’’\,’ → ‘He said, “Hello.”’

12.7.3 2種類のスペース

\TeX の流儀では、英語には単語間のスペースとセンテンス間のスペースが存在する。以下で説明することがややこしければ、文書ファイルの最初の方に`\frenchspacing`と記述しておくこと。そうすることで、単語間もセンテンス間も同じスペースになる。元に戻すコマンドは`\nonfrenchspacing`である。

\TeX 標準（`\nonfrenchspacing`）では、10ポイントのTimesやPalatino（ptm, ppl）の単語間のスペースは2.5 [pt]、センテンス間のスペースは3.1 [pt]である（さらに伸び縮みする）。 \TeX は、ピリオドや感嘆符・疑問符・コロンのセンテンスの区切りを判断するが、例外として、大文字1文字の直後のピリオドは省略のピリオドと解釈され、通常の単語間の扱いになる。

次の例を見よ。

Hello. I’m H. Okumura. → Hello. I’m H. Okumura.

Hello. の直後のスペースはセンテンス間のスペースである。一方、I’m の直後のスペースやH. の直後のスペースは単語間のスペースである。

記号類は無視される。例えば（“H”）. の直後のスペースは、大文字1文字Hの後なので単語間のスペースになる。直前のアルファベットに関わらず、文末のスペースにするには`\@`を用いる。

(誤) I watch TV. It's fun. → I watch TV. It's fun.

(旧) I watch TV\@. It's fun. → I watch TV. It's fun.

(新) I watch TV.\@ It's fun. → I watch TV. It's fun.

この最後の形は jsarticle、jsbook 専用の書き方である。2 番目の古い書き方では V とピリオドのカーニングがうまくいかない。

直前のアルファベットに関わらず単語間のスペースにするには _ とする。

(誤) FooBar Co., Ltd. is cool. → FooBar Co., Ltd. is cool.

(正) FooBar Co., Ltd.\ is cool. → FooBar Co., Ltd. is cool.

(誤) Mr. and Mrs. Okumura → Mr. and Mrs. Okumura

(正) Mr.\ and Mrs.\ Okumura → Mr. and Mrs. Okumura

(誤) Mr.~and Mrs.~Okumura → Mr. and Mrs. Okumura

最後の例のように、行分割しない空白 ~ も必ず単語間のスペースの広さになる。Mr. の直後では改行してほしくないことが多いので、この場合は ~ を使うのがよいだろう。

12.8 その他の調整

12.8.1 改行幅の調整

仕上がりを見てから調節しなければならない例として、 $\frac{f(x)}{\int_{-\infty}^{\infty} f(x) dx}$ のような大きな数式による改行の乱れがある。

この場合、すぐ下の行に何もなければ、インライン数式を `\smash{\$...\$}` のように全体を `\smash` で囲んで改行幅が乱れないようにすることができる。このような調整は必須ではないが、見苦しい場合は適用することがある。

12.8.2 図の調整

L^AT_EX は組版から図の配置までを自動で行ってくれるが、特に図がたくさん入る場合は、自動ではなかなか人間の美意識に叶う配置ができない。最終段階で視覚的な調整が必要となる。これについては第 9 章で述べた。



例えば、右のように図にテキストを回り込ませる場合、図がテキストの上端や、回り込みのないテキストの右端など、顕著な位置にぴったり揃うようにすると整った感じがする。このようなルールを破って大胆に配置することもあるが、中途半端にずれていると素人組版に見えてしまう。

付録 A

TikZ

TikZ^{*1} は TeX の中で使える強力な^{ドロー}描画コマンド群である。従来の `picture` 環境、METAPOST、Asymptote、PSTricks に置き換えて使うことができる。統計ツール R やグラフ作成ツール gnuplot と組み合わせれば、より高度なグラフを描くことができる。

A.1 PGF/TikZ とは？

TikZ は TeX の中で使える強力な^{ドロー}描画コマンド群である。作者は、スライド作成用パッケージ Beamer の作者としても有名な Till Tantau 氏である。

TikZ は見ての通り *k* だけイタリック体で記述する。名前の由来は TikZ ist *kein* Zeichenprogramm (TikZ はドローツールではない) である (再帰的な略語)。従来の L^AT_EX の `picture` 環境を強力にしたもので、機能的には METAPOST、Asymptote、PSTricks に近いのだが、バックエンドとして PGF (Portable Graphics Format) という仕組みを使っており、PostScript を介さないのでトラブルが起きにくく、日本語の問題も生じない。また、独立なプログラムを使わず TeX の中に直接記述できるのも大きな利点である。Beamer でも PGF を使っている。

TikZ を使うには、 ϵ -TeX 拡張された TeX が必要となるが、最近のものであれば大丈夫である。PGF/TikZ の詳しい PDF マニュアルは、TeX Live を入れたシステムなら、ターミナルに `texdoc tikz` と打ち込めば表示される。

A.2 TikZ の基本

TikZ を使うには、次の例のように、プリアンブルに `\usepackage{tikz}` と記述する。更に、`dvipdfmx` で PDF 化する場合 (通常の日本語文書処理の場合) は、ドキュメントクラスのオプションに `dvipdfmx` と記述する^{*2}。

簡単な例を試してみる。

```
\documentclass[dvipdfmx]{jsarticle}
\usepackage{tikz}
\begin{document}
\tikz\draw(0,0)--(0.1,0.2)--(0.2,0.0)--(0.3,0.2)--(0.4,0.0);
\end{document}
```

これで「 \wedge 」のような出力が得られる。(0,0) などは座標であり、デフォルトの単位は cm だが (12mm, 3pt) のように TeX が理解する単位を付けることもできる。pTeX 専用の `zw` などの単位は使えないが、本文フォントの `1zw` に相当する長さ `\Cwd` が日本語ドキュメントクラスの中で定義されているので、例えば `3zw` なら `3\Cwd` と書くことができる。

このように `\draw` は点を結んで折れ線を描く。`draw` 文の最後にはセミコロンが必要となる。複数の `\draw` が存在する場合は、

```
\tikz{\draw(0,0)-- (10pt,10pt); \draw(0,10pt)-- (10pt,0);}
```

のように中括弧で囲むか、あるいは、

^{*1} TikZ の読み方は特に決まっていないが「ティックジー」「ティックス」などと読まれている。

^{*2} あるいは、`dvipdfmx` オプションを付加した `graphicx` と `xcolor` を先に読み込んでおく。





```
\begin{tikzpicture}
  \draw(0,0) -- (10pt,10pt);
  \draw(0,10pt) -- (10pt,0);
\end{tikzpicture}
```

のように tikzpicture 環境を用いる。因みに、この記述では \times が描かれる。

A.3 いろいろな図形の描画

x 座標も y 座標も単位を 1 mm にして、線幅 2pt、角の丸み 8pt の矢印を描いてみる。

```
\begin{tikzpicture}[x=1mm,y=1mm]
  \draw[line width=2pt, rounded corner=8pt, ->]
    (0,0) -- (5,5) -- (10,0) -- (15,5) -- (20,0);
\end{tikzpicture}
```

因みに、この記述では  が描かれる。矢印は `->`、`<-`、`<->`、`->>` などが指定できる。また、矢印の形には TikZ 標準 ()、L^AT_EX 標準 ()、ステルス戦闘機型 () などが用意されている。

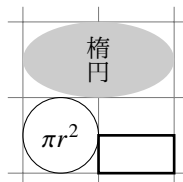
```
\begin{tikzpicture}[line width=1pt]
  \draw[->] (0,1) -- (1,1);
  \draw[-latex] (0,0.5) -- (1,0.5);
  \draw[-stealth] (0,0) -- (1,0);
\end{tikzpicture}
```

TikZ 標準は、数式で $f: A \rightarrow B$ と書く際の `\to` (`\rightarrow`) と (大きさは異なるが) 同じ形である。

`\draw` でグリッドや円、長方形、楕円も描くことができる。勿論、日本語や数式も書くことができる。`\fill` にすると、指定した色で塗りつぶす (この例では 20% の灰色)。

```
\begin{tikzpicture}
  \draw[step=1,gray] (-0.2,-0.2) grid (2.2,2.2);
  \draw (0.5,0.5) circle (0.5) node {$\pi r^2$};
  \draw[line width=1pt] (1,0) rectangle (2,0.5);
  \fill[black!20] (1,1.5) ellipse (1 and 0.5);
  \draw (1,1.5) node {\hbox{\tate 楕円}};
\end{tikzpicture}
```

出力は次のようになる



より複雑な図形は、制御点を 2 つ与えたベジエ (Bézier) 曲線で描くことができる。第 2 の制御点が第 1 のものと同じ場合は省略することができる。

```
\begin{tikzpicture}[x=1mm,y=1mm]
  \fill[gray] (0,0) circle (1)
    (10,10) circle (1)
    (20,10) circle (1)
    (20,0) circle (1);
  \draw (0,0) -- (10,10) -- (20,10) -- (20,0);
  \draw[line width=2pt] (0,0) .. controls (10,10) and (20,10) .. (20,0);
  \draw[line width=2pt,gray] (0,0) .. controls (10,10) .. (20,0);
\end{tikzpicture}
```

出力は次のようになる。



折れ線と曲線は次のように混在させることができる。

```
\begin{tikzpicture}[x=1mm,y=1mm,line width=2pt]
\draw (2,2) circle (2);
\draw (2,4) -- (6,4) -- (6,0) .. controls (6,3) and (7,4) ..
(12,4) -- (9,0) -- (12,0);
\end{tikzpicture}
```

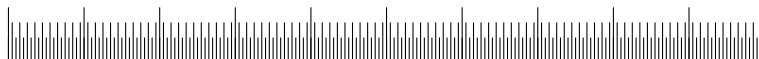
出力は次のようになる。



繰り返しも `\foreach` という強力な命令で簡単に実現できる。... は書くのを省略したのではなく、本当にこのように記述すれば TikZ が補ってくれる。

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw (0,0) -- (100,0);
\foreach \x in {0.0,...,100} \draw (\x,0) -- (\x,3);
\foreach \x in {0.5,...,100} \draw (\x,0) -- (\x,5);
\foreach \x in {0.0,10,...,100} \draw (\x,0) -- (\x,7);
\end{tikzpicture}
```

出力は次のようになる。



応用として、センター試験でよく使われる楕円の番号を作成してみる。

```
\newcommand{\egg}[1]{\raisebox{-3pt}{%
\begin{tikzpicture}[x=1pt,y=1pt,line width=1pt]
\draw (0,0) ellipse (4.5 and 6);
\draw (0,0) node {\fontfamily{phv}\fontsize{9pt}{0}\selectfont #1\;/};
\end{tikzpicture}}}
\newcommand{\eggg}[1]{\raisebox{-3pt}{%
\begin{tikzpicture}[x=1pt,y=1pt,line width=1pt]
\draw[fill=black!30] (0,0) ellipse (4.5 and 6);
\draw (0,0) node {\fontfamily{phv}\fontsize{9pt}{0}\selectfont #1\;/};
\end{tikzpicture}}}
```

これで `\egg{0}` `\egg{1}` `\egg{2}` `\eggg{0}` `\eggg{1}` `\eggg{2}` とすれば ① ② ③ ④ ⑤ ⑥ と出力される。

次は `\A` のようなキーボード記号である。

```
\renewcommand{\keytop}[2][12]{%
\begin{tikzpicture}[x=0.1em,y=0.1em]
\useasboundingbox (0,0) rectangle (#1,9);
\shadedraw[top color=black!20,rounded corners=0.2em] (0,-3)
rectangle (#1,9);
\draw[anchor=base] (#1/2,0) node {\sffamily #2};
\end{tikzpicture}
```

`\keytop{A}` と書けば `\A` と出力される。 `\Enter` のように幅の広いものは `\ketop[30]{Enter}` のように幅を 0.1em 単位で指定する。

A.4 グラフの描画 (1)

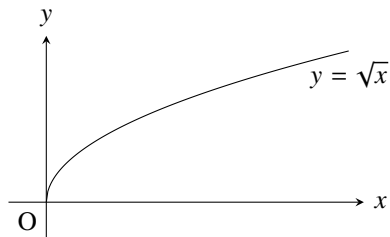
TikZ では `sin`, `cos`, `exp`, `sqrt` などの関数が利用できる。但し、TeX で実装しているので、遅く、精度の低い固定小数点数である。次のような簡単なことは可能である。

```

\begin{tikzpicture}[domain=0:4,samples=200,>=stealth]
\draw[>-] (-0.5,0) -- (4.2,0) node[right] {$x$};
\draw[>-] (0,-0.5) -- (0,2.2) node[above] {$y$};
\draw plot (\x, {\sqrt{\x}}) node[below] {$y=\sqrt{x}$};
\draw (0,0) node[below left] {0};
\end{tikzpicture}

```

出力は次のようになる。



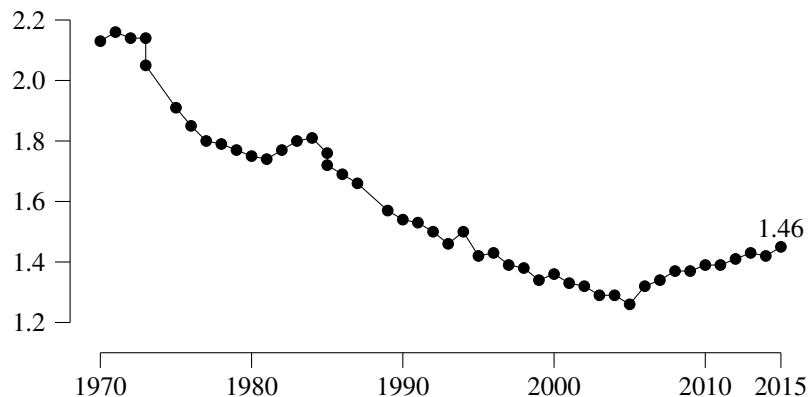
数値の表を与えてグラフをプロットすることもできる。例えば、毎年の日本の合計特殊出生率が、

```

1970 2.13
1971 2.16
1972 2.14
...
2015 1.46

```

のようなテキストファイル TFR.tbl で与えられているとする。これを、



のように描くには次のようにする。

```

\begin{tikzpicture}[x=2mm,y=40mm]
\draw (1968,1.2) -- (1968,2.2);
\foreach \x in {1.2,1.4,1.6,1.8,2.0,2.2}
\draw (1968,\x) -- (1967,\x) node[left] {\x};
\draw (1970,1.1) -- (2015,1.1);
\foreach \x in {1970,1980,...,2000,2010,2015}
\draw (\x,1.1) -- (\x,1.05) node[below] {\x};
\draw[mark=*] plot file {TFR.tbl} node[above] {1.46};
\end{tikzpicture}

```



あるいは、もっと単純に 2.13 1.46 のようなスパークライン (sparkline) で描くには次のようにする。

```

\begin{tikzpicture}[x=1pt]
\fill (1970,2.13) circle (2pt) node[left] {2.13};
\draw plot file {TFR.tbl};
\fill (2015,1.46) circle (2pt) node[right] {1.46};
\end{tikzpicture}

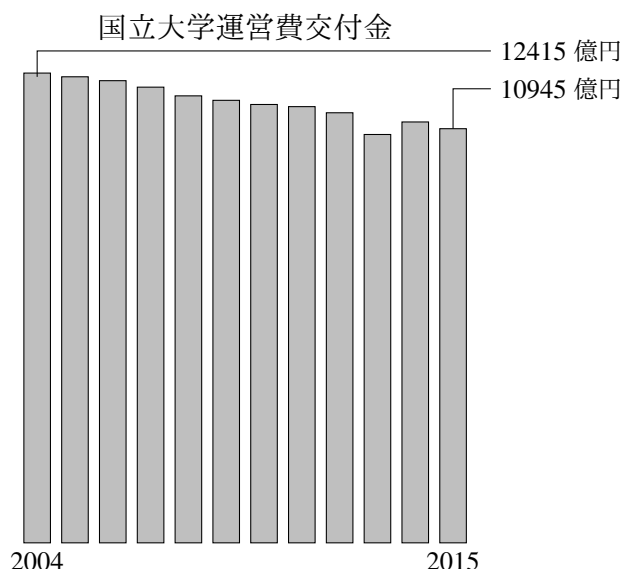
```

但し、TikZ は TeX の固定小数点数で計算するので、あまり大きな値を扱うと “ERROR: Dimension too large.” となることがある。その場合は、適当に数値を切り詰める。

次の棒グラフの例では、年から 2000 を引いた値を x 座標としている。

```
\begin{tikzpicture}[ybar,x=5mm,y=0.005mm]
\draw[fill=lightgray] plot coordinates
{( 4,12415) ( 5,12317) ( 6,12214) ( 7,12043) ( 8,11813) ( 9,11695)
(10,11585) (11,11528) (12,11366) (13,10792) (14,11123) (15,10945)};
\draw (4,0) node[below] {2004};
\draw (15,0) node[below] {2015};
\draw (4,12315)|-(16,13000) node[right] {12415 億円};
\draw (15,10945)|-(16,12000) node[right] {10945 億円};
\draw (9.5,13000) node[above] {\large 国立大学運営費交付金};
\end{tikzpicture}
```

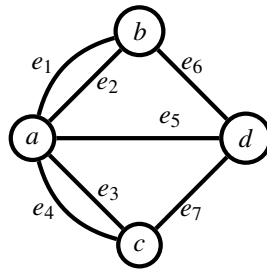
この例では、引出線を引くために `--` ではなく `|-` を使っている。 $(x_1, y_1) |-(x_2, y_2)$ は $(x_1, y_1) -- (x_1, y_2) -- (x_2, y_2)$ と同じことで、先に水平方向に、次に水平方向に線を引く。水平と垂直の順序を入れ替えた `-|` も用意されている。



A.5 グラフの描画 (2)

グラフと言えば、グラフ理論のグラフも簡単に描くことができる。ノード (点) に名前を付け、両端のノードを指定して辺を描けばよい。次のグラフは、有名な ^{ケーニヒスベルク}Königsberg の橋の問題のグラフである。

```
\begin{tikzpicture}[line width=1.6pt,node distance=2cm]
\node(a) [draw,circle]{$a$};
\node[above right of=a](b) [draw,circle]{$b$};
\node[below right of=a](c) [draw,circle]{$c$};
\node[right of=a,node distance=2.82cm](d) [draw,circle]{$d$};
\draw (a) to[bend left] node[midway,left] {$e_{1}$} (b);
\draw (a) -- node[midway,right] {$e_{2}$} (b);
\draw (a) -- node[midway,right] {$e_{3}$} (c);
\draw (a) to[bend right] node[midway,left] {$e_{4}$} (c);
\draw (a) -- node[pos=.7,above] {$e_{5}$} (d);
\draw (b) -- node[midway,above] {$e_{6}$} (d);
\draw (c) -- node[midway,below] {$e_{7}$} (d);
\end{tikzpicture}
```



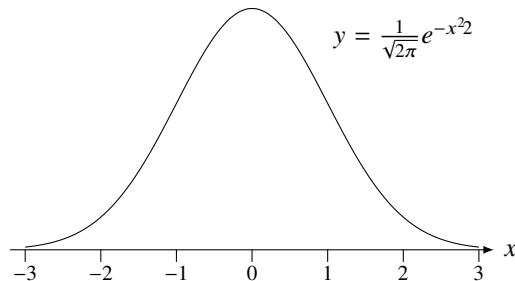
この例では、ノード a を基準とした相対的な位置で b , c , d を指定している。辺は両端点で指定し、辺の途中にラベルを付けている。bend left や bend right で湾曲した辺を描くには --ではなく to を用いる（この例では -- を全て to に置き換えても構わない）。

A.6 gnuplot との連携

ニュープロット

gnuplot³ は昔から使われている強力なプロットツールである。これと連携させて TikZ のプロットを描くことができる。例えば、標準正規分布のグラフを描いてみる。次のように `function{...}` という命令を使えば、関数の中身 `exp(-x**2/2)/sqrt(2*pi)` が gnuplot に渡され、関数値の表に変換される。この際、platex に `-shell-escape` オプションを付けて起動しなければならない。gnuplot で渡される命令と、返される表のファイル名は LaTeX 文書のファイル名と `-id=` で与えられた名前から生成される。一旦これらが生成されれば、関数を変えない限り gnuplot を呼び出すことはないので `-shell-escape` オプションも不要である。

```
\begin{tikzpicture}[domain=-3:3,samples=50,>=latex,y=8cm]
  \draw[->] (-3.2,0) -- (3.2,0) node[right] {$x$};
  \draw plot[id=dnorm,smooth] function{exp(-x**2/2)/sqrt(2*pi)};
  \draw (2,0.35) node {$y=\frac{1}{\sqrt{2\pi}}e^{-x^2/2}$};
  \foreach \x in {-3,...,3}
    \draw (\x,0) -- (\x,-0.02) node[below=-2pt] {\footnotesize $\x$};
\end{tikzpicture}
```



尚、これぐらいの関数であれば、TikZ だけで、

```
\draw plot[smooth] (\x, {exp(-0.5 * \x \x) / sqrt(2 * pi)});
```

のようにして描くこともできる。

gnuplot から返される表は、# で始まるコメント行を除けば、次のような形式である。

```
-3.00000 0.00443 i
-2.87755 0.00635 i
-2.75510 0.00897 i
...
```

最後の文字 i は範囲内、 o は範囲外、 u は未定義を表す。gnuplot に関わらず、このような表を用意しておけば、

```
\draw plot[smooth] file {ファイル名};
```

のようにしてプロットすることができる。

³ GNU プロジェクトを思わせる名前なのでグニュープロットと呼ばれることもあるが、GNU プロジェクトとは無関係である。

別の方法として gnuplot で、

```
set term tikz
```

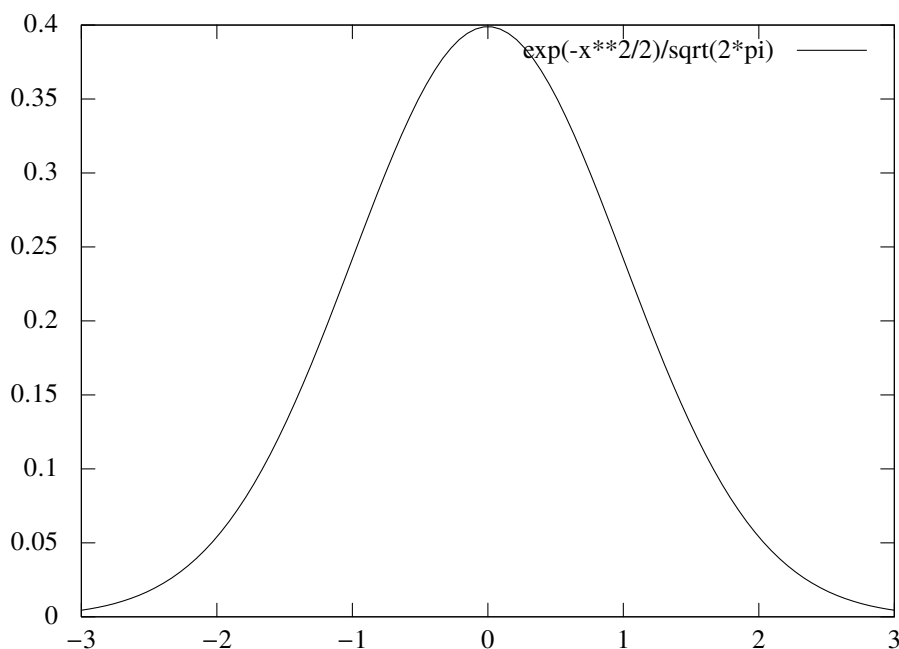
とすれば、TikZ 形式の出力が得られる。例えば、

```
set term tikz monochrome
set output "dnorm-gp.tex"
set xrange [-3:3]
plot exp(-x**2/2)/sqrt(2*pi) with lines
exit
```

とし、 \LaTeX 文書の中では、

```
\documentclass[dvipdfmx]{jsarticle}
\usepackage{gnuplot-lua-tikz}
\begin{document}
\input{dnorm-gp}
\end{document}
```

とすれば、下のような図が得られる。実際には、更に手を加えて見栄えを良くする必要がある。



A.7 他の図との重ね書き節

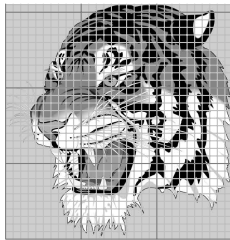
TikZ は強力な描画能力を備えているが、コマンドだけで全てを描くのは難しい。そこで、他のソフトで描いた図の上に \TeX の数式などを TikZ で重ね書きする方法を考える。例えば、Ghostscript の虎の絵に説明を加えてみる。図は、

```
\includegraphics[width=3cm]{tiger.pdf}
```

のような命令で読み込むのであった。

この任意の位置に文字や図を重ね書きするために、一時的にグラフ用紙を重ね書きしてみる。

```
\begin{tikzpicture}[inner sep=0pt]
  \node[anchor=south west] (image) at (0,0)
    {\includegraphics[width=3cm]{tiger.pdf}};
  \draw[step=0.1,lightgray] (0,0) grid (image.north east);
  \draw[step=1,gray] (0,0) grid (image.north east);
\end{tikzpicture}
```

これを見ながら、座標 (0.3,2.5) の場所を中心とする位置に文字を入れてみる。tikzpicture 環境内に追加するのは次の 2 行である。

```
\node[white] at (0.3,2.5) {\hbox{\tate {\LARGE 虎}さん}};
\node at (0.2,0.3) {がー};
```

うまくいったらグラフ用紙 (grid) を描く次の 2 行は消しておく。

```
\draw[step=0.1,lightgray] (0,0) grid (image.north east);
\draw[step=1,gray] (0,0) grid (image.north east);
```

結果は下のようになる。

