

TEX/LATEX

ことはじめ

cuttlefish_math



目次

第 1 章	はじめに	3
第 2 章	コンピュータとマルチメディア	4
2.1	文字コード	4
2.2	ラスタ形式とベクタ形式	5
2.3	可逆圧縮と非可逆圧縮	6
2.4	補遺	7
2.4.1	文字化けの仕組み	7
2.4.2	ラスタ形式と拡大・縮小	7
第 3 章	T_EX/L^AT_EX とは	9
3.1	T _E X と L ^A T _E X	9
3.2	T _E X と和文	9
3.3	補遺	9
3.3.1	SAT _Y SF _I について	9
3.3.2	ε-T _E X について	9
3.3.3	他の命令の引数にできない命令	10
第 4 章	グラフの作成	11
4.1	datavisualization	11
4.2	GNUPlot による方法	11
4.3	その他のグラフ	13
4.4	補遺	13
4.4.1	グラフ描画の精度	13
第 5 章	フォント	14
第 6 章	補遺	15
6.1	パッケージの相性問題	15

第 1 章 はじめに

TeX/LaTeX は理工系で広く用いられているシステムであり、特に数学に関わる分野では、文書作成のデファクトスタンダードと言える。しかしながら、その全容について把握するのは容易ではない。

たとえば、TeX/LaTeX により次の数式を記述したいとする。

$$f(t) \sim \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos nt + b_n \sin nt)$$

あなたは「シグマ TeX」などと検索することで、概ね次のようなコードを書けばよいことが分かるだろう。

```
\[
f(t) \sim \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos nt + b_n \sin nt)
\]
```

しかし、こうして課題のレポートをなんとか作れるようになったとしても、「TeX/LaTeX とは」と問われると答えに詰まるのではないだろうか。

本稿は「TeX/LaTeX を理解して、自由に文書を作成するためのチュートリアル」となるべく書かれている。特に、理工系学生がよく利用するであろう「表作成」と「グラフ作成」に重きをおき、サンプルコードを多く載せることで、レポートを作るときに役立つようになっている。本稿が TeX/LaTeX について、ブラックボックスとしてではない理解を得る一助になれば幸いである。

なお、本書の最新版は GitHub (<https://github.com/calamari-dev/texlatex>) にある。また、ライセンスは CC BY 4.0 にしたがう。

第2章 コンピュータとマルチメディア

2.1 文字コード

コンピュータにおいて、テキストを記録するには、テキストを数値の列に変換しなければならない。この変換方式のことを**文字コード**という。文字コードは複数あり、それぞれ利用できる文字や変換方式などが異なる。



図 2.1: Windows 10 のメモ帳

たとえば、最新の Windows 10 にあるメモ帳でテキストを保存すると、UTF-8 という方式で文字列が数値の列に変換される。この他にも、ソフトウェアに応じて ASCII, Shift_JIS など、さまざまな文字コードが利用されている。

現在では **Unicode** という文字コードの規格が主流になりつつある。先述の UTF-8 も、この Unicode において定義されている。この節では文字コードの代表例として、Unicode について解説する。なお、以下に示す用語は文字コードによっては異なる名前と呼ばれることもあるし、対応する概念が無いこともある。

文字コードが扱える文字の全体集合を**文字集合**という。文字集合の各要素には固有の非負整数が割り当てられており、その対応関係を Unicode では**符号化文字集合**と呼ぶ¹⁾。

符号化文字集合（の像）は、集合 $S = \{0_{(16)}, \dots, 10FFFF_{(16)}\}$ の真部分集合である。 S を**符号空間**といい、 S の要素を**符号位置**という。符号位置は、十六進数の先頭に「U+」を付けて「U+304B」のようにして表される。

符号化文字集合は、数値を並べたときに区切れ目がただ 1 通りに定まるかどうかを考慮していない。そこで、あとから区切れ目が分かるように、符号化文字集合（の像）に属する数値を整数の組²⁾（**符号単位**）へと変換する規則が必要になる。この規則を**文字符号化形式**といい、UTF-8, UTF-16, UTF-32 の 3 つが定義されている。

か $\xrightarrow{\text{符号位置に変換}} \text{U+304B} \xrightarrow{\text{符号単位に変換}} \text{E3 81 8B}$

図 2.2: 文字の符号単位への変換

表 2.1 に Unicode の符号化文字集合を一部抜粋して示す。

例として「M 系列」という文字列について考えよう。「M 系列」の各文字を Unicode の符号位置に置き換えると次のようになる。

1) 対応関係なのに集合？と思うのももっともだが、公式資料には確かに「a mapping from an abstract character repertoire to a set of nonnegative integers」[2] とある。

2) 正確には 1 バイトの組である。

表 2.1: 符号化文字集合の一部

U+	0	1	2	3	4	5	6
0020		!	"	#	\$	%	&
0030	0	1	2	3	4	5	6
0040	@	A	B	C	D	E	F

U+004D U+7CFB U+5217

そして、これらを UTF-8 で符号単位の列に変換すると次のようになる³⁾。「4D」が U+004D, 「E7 B3 BB」が U+7CFB, 「E5 88 97」が U+5217 にそれぞれ対応する。

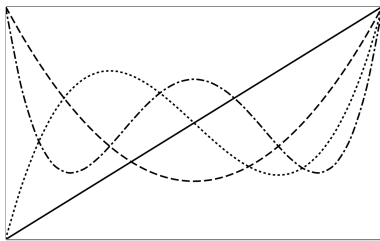
4D E7 B3 BB E5 88 97

2.2 ラスタ形式とベクタ形式

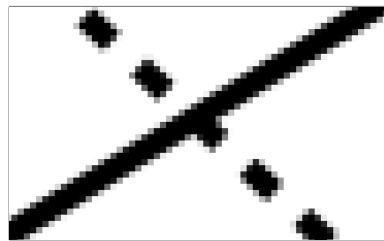
コンピュータで画像を記録する方法は、**ラスタ形式**と**ベクタ形式**の2つに大別される。ラスタ形式は、画像を単色の要素（**画素**）を集めたものとして記録する方法である。これに対し、ベクタ形式は画像を図形の集まりとして記録する方法である。

例を挙げると、**JPEG**、**PNG** はラスタ形式である。また、ベクタ形式である **SVG** は、Web ページのデザインにしばしば利用されている。

ベクタ形式に比べ、ラスタ形式は拡大・縮小の影響を受けやすい。ラスタ形式の画像を拡大したときの様子を図 2.3 に示す（ただし、ラスタ形式の画像を拡大・縮小した結果は利用したソフトウェアに依存する。詳しくは小節 2.4.2 を参照のこと）。



(a) 画像全体



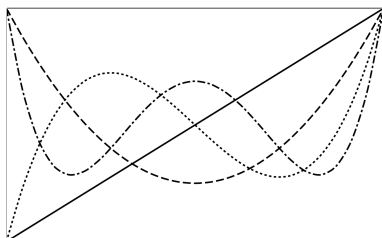
(b) 中央付近を拡大

図 2.3: ラスタ形式の画像を拡大したときの様子

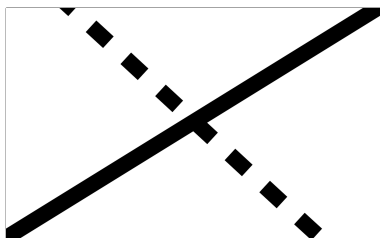
これに対し、ベクタ形式の画像は拡大しても図 2.3 のようにならない。これは、ベクタ形式では拡大・縮小に応じて図形を描画しなおせるからである。ベクタ形式の画像を拡大したときの様子を

3) UTF-16 と UTF-32 では符号単位の列をバイト列に変換する方法（**文字符号化スキーム**）も問題になる。UTF-8 では、符号単位を並べたものをそのままバイト列とする。

図 2.4 に示す。



(a) 画像全体



(b) 中央付近を拡大

図 2.4: ベクタ形式の画像を拡大したときの様子

2.3 可逆圧縮と非可逆圧縮

この節では、ラスタ形式の画像について扱う。多くの場合、ラスタ形式の画像ファイルは、データサイズを削減するために圧縮されている。圧縮した後のデータから元のデータを復元できる圧縮方式を**可逆圧縮**、復元できない圧縮方式を**非可逆圧縮**という。

たとえば、PNG は可逆圧縮、JPEG は非可逆圧縮である⁴⁾。非可逆圧縮のほうがデータサイズが小さくなる傾向にあるが、画像の性質にもよるので一概には言えない。大雑把に言うと、PNG は線画、JPEG は写真に強い傾向にある。

非可逆圧縮を利用するときは、圧縮するたびに画像が劣化することに注意しなければならない。また、データサイズを削減するために品質をあまりに低く設定すると、画像が非常に粗くなることがある。図 2.5 に、JPEG の品質 (0 以上 100 以下の整数を設定できる) を 10 に設定して圧縮したときの様子を示す。



(a) 元画像



(b) 圧縮した画像

図 2.5: 画像の過剰な圧縮に伴う劣化

4) JPEG の規格は可逆圧縮にも対応している。しかし、多くの場合 JPEG は非可逆圧縮に利用されるので、本書では JPEG は非可逆圧縮であると見なす。

2.4 補遺

2.4.1 文字化けの仕組み

多くの場合、文字化けはテキストの文字コードをソフトウェアが誤判定したときに発生する。たとえば、図 2.6 では UTF-8 で作成したテキストを Shift_JIS で開いてしまっている。

文字化けを直すには、適切な文字コードを指定してテキストを開きなおせばよい。Visual Studio Code⁵⁾などのテキストエディタは、ユーザが文字コードを選択しなおす機能を備えている。

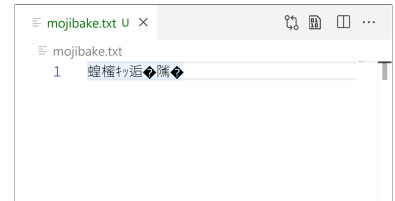


図 2.6: 文字化けしたテキスト

また、テキストを UTF-8 で符号化したにもかかわらず、テキストエディタが文字コードを ASCII と判定する場合がある。これは、UTF-8 と ASCII の仕様には共通する部分があるので、たまたまテキスト中の文字がすべて ASCII の文字集合にも属していた場合、文字コードをどちらとも解釈できることがあるからである。

2.4.2 ラスタ形式と拡大・縮小

ラスタ形式の画像に対する拡大・縮小は、元の画像から新たな画像を作成する操作と言える。そのため、拡大・縮小の手法は複数存在する。ニアレストネイバー法、バイリニア法、バイキュービック法は、よく知られた拡大・縮小の手法である。各手法で画像を拡大したときの様子を図 2.7 に示す。ただし、図 2.7 の画像編集はすべて GIMP⁶⁾で行った。また、各手法の特徴が分かりやすくなるように、(b) から (d) は (a) の画像を幅が 100px になるまでニアレストネイバー法で縮小してから、幅が 1600px になるまで拡大した。

図 2.7 において生クリームの境界線に注目すると、拡大結果がバイキュービック法、バイリニア法、ニアレストネイバー法の順になめらかであることが分かる。

図 2.7 で挙げた以外にも、画像が粗くなるのを回避するために機械学習を応用した手法が提案されていたりと [1], 様々な手法が研究されている。

5) <https://code.visualstudio.com>

6) <https://www.gimp.org>



(a) 元画像



(b) ニアレストネイバー法



(c) バイリニア法



(d) バイキュービック法

図 2.7: 画像の拡大手法の比較

第 3 章 $\text{T}_{\text{E}}\text{X}/\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ とは

3.1 $\text{T}_{\text{E}}\text{X}$ と $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$

$\text{T}_{\text{E}}\text{X}$ とは, Donald Knuth により開発された組版システムである. $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ とは, Leslie Lamport により開発された $\text{T}_{\text{E}}\text{X}$ のマクロである.

```
Hello, plain \TeX !  
\bye
```

この他にも, Con $\text{T}_{\text{E}}\text{X}$ t というマクロパッケージがあったりする.

$\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$ の話も.

3.2 $\text{T}_{\text{E}}\text{X}$ と和文

$\text{pT}_{\text{E}}\text{X}$ は JIS などの文字集合に対応した $\text{T}_{\text{E}}\text{X}$ 処理系である. $\text{upT}_{\text{E}}\text{X}$ は内部コードを Unicode にした $\text{pT}_{\text{E}}\text{X}$ の拡張である. $\text{LuaT}_{\text{E}}\text{X}$ は $\text{T}_{\text{E}}\text{X}$ 処理系を汎用プログラミング言語である **Lua** によって拡張できるようにした $\text{T}_{\text{E}}\text{X}$ 処理系である.

3.3 補遺

3.3.1 $\text{SAT}_{\text{Y}}\text{SF}_{\text{I}}$ について

$\text{SAT}_{\text{Y}}\text{SF}_{\text{I}}$ は, 和文に対応した非 $\text{T}_{\text{E}}\text{X}$ 系の組版システムである. $\text{SAT}_{\text{Y}}\text{SF}_{\text{I}}$ を利用するという選択肢もある.

3.3.2 $\epsilon\text{-T}_{\text{E}}\text{X}$ について

実は, $\text{upL}^{\text{A}}\text{T}_{\text{E}}\text{X}$ を起動したときに実行されるのは, 元々の $\text{upT}_{\text{E}}\text{X}$ とは少し異なる $\text{T}_{\text{E}}\text{X}$ 処理系である.

コマンドラインで `uplatex` を実行すると, 次のメッセージが出力される.

```
This is e-upTeX, Version 3.141592653-p3.9.0-u1.27-210218-2.6 (utf8.uptex) (TeX  
Live 2021/W32TeX) (preloaded format=uplatex)  
restricted \write18 enabled.
```

★★

本書では以降、 ε -TeX 拡張の有無についてはいちいち断らないことにする。すなわち、upTeX と言ったら ε -upTeX を指すものとする。

3.3.3 他の命令の引数にできない命令

TeX の仕様により、`\verb` など一部の命令は、他の命令の引数にできない。たとえば、次の L^AT_EX ソースは `\verb` が `\footnote` の引数になっているので不正である。

```
\documentclass[uplatex,dvipdfmx]{jsarticle}
\begin{document}
\TeX のロゴ (\emph{\verb!\TeX!で出力できる) は、Donald Knuthがこう表記するよう
に求めている。
\end{document}
```

実際、`example.tex` を上記の内容で作成し、次のコマンドを実行すると「LaTeX Error: `\verb` illegal in command argument.」というエラーが出力される。

```
> uplatex -kanji=utf8 -no-guess-input-enc example
```

この問題を解決する手っ取り早い方法は、`\verb` を使わないことである。すなわち、`\` を `\textbackslash` に置き換えればよい。

```
\documentclass[uplatex,dvipdfmx]{jsarticle}
\begin{document}
\TeX のロゴ (\emph{\texttt{\textbackslash TeX}で出力できる) は、Donald Knuthが
こう表記するように求めている。
\end{document}
```

第 4 章 グラフの作成

本稿では、次の 3 つの方法を紹介する.

1. PGFPlots による方法
2. GNUPlot による方法
3. TikZ の datavisualization ライブラリによる方法

4.1 datavisualization

TikZ

4.2 GNUPlot による方法

あああ¹⁾.

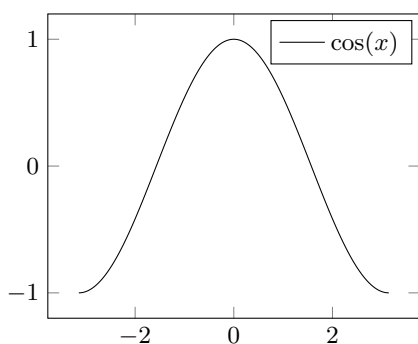
```
\begin{tikzpicture}
  \begin{axis}[width=7cm]
    \addplot [smooth,samples=100,domain=-pi:pi] {cos(deg(x))};
    \addlegendentry{$\cos(x)$};
  \end{axis}
\end{tikzpicture}
```

```
\input{gnuplot.tex}
```

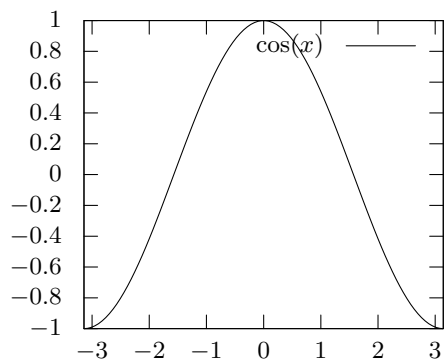
```
gnuplot> set terminal lua tikz size 7cm,5cm
gnuplot> set output "gnuplot.tex"
gnuplot> set xrange [-pi:pi]
gnuplot> plot cos(x) linetype rgb "black" title '$\cos(x)$'
gnuplot> exit
```

```
\begin{tikzpicture}
  \begin{axis}[width=7cm]
    \addplot [smooth,samples=100,mark=none,domain=-pi:pi] function [id=cos] {
      cos(x)};
  \end{axis}
\end{tikzpicture}
```

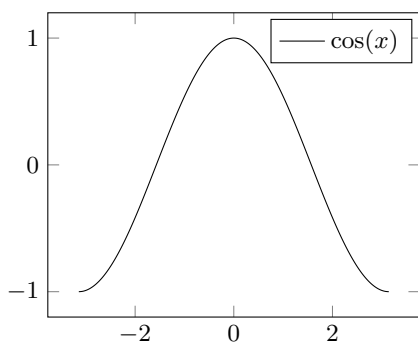
1) set term tikz としている資料もあるが、これは set terminal lua tikz の省略形である.



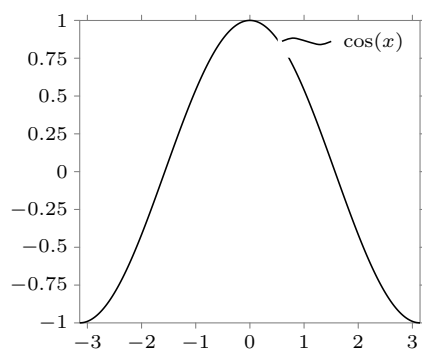
PGFPlots



gnuplot-lua-tikz



GNUPlot on PGFPlots



datavisualization

図 4.1: グラフ描画の比較

```
\addlegendentry{$\cos(x)$};
\end{axis}
\end{tikzpicture}
```

```
\begin{tikzpicture}
\datavisualization [
  scientific axes,
  visualize as smooth line/.list={cosine},
  x axis={length=5cm},
  y axis={length=4cm},
  legend=north east inside,
  cosine={label in legend={text={$\cos(x)$}}}
]
data [set=cosine, format=function] {
  var x : interval [-pi:pi] samples 100;
  func y = cos(deg(\value x));
};
\end{tikzpicture}
```

4.3 その他のグラフ

3D グラフなども同様の要領で作成できる．以下に PGFPlots を利用した例を示す．

```
\begin{tikzpicture}
  \begin{axis}[width=7cm,colormap/blackwhite]
    \addplot3 [surf,miter limit=1,samples=30,domain=-2:2] {exp(-(x^2+y^2))};
  \end{axis}
\end{tikzpicture}
```

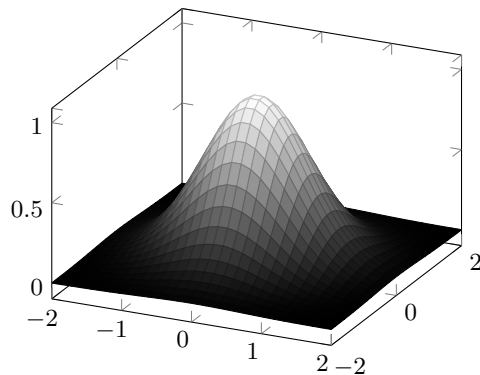


図 4.2: $e^{-(x^2+y^2)}$ のグラフ

4.4 補遺

4.4.1 グラフ描画の精度

PGFPlots の Revision 1.18 (2021/05/08) p.54 を引用する．

About the precision and number range: Starting with version 1.2, `\addplot expression` uses a floating point unit. The FPU provides the full data range of scientific computing with a relative precision between 10^{-4} and 10^{-6} . The `/pgf/fpu` key provides some more details.

Note that pgfplots makes use of lualatex's features: if you use lualatex instead of pdflatex, pgfplots will use lua's math engine which is both faster and more accurate (`compat=1.12` or higher).

つまり、処理系が Lua¹TeX であり、`compat` に 1.12 以上のバージョンが指定されていれば、PGFPlots は Lua の数学エンジンを利用して計算を行う．これは TeX 上の実装より高速かつ正確である．

第 5 章 フォント

この節では処理系を LuaT_EX に限定する。これは、LuaT_EX が日本語組版に対応しており、かつフォントを簡単に換えられる処理系だからである。

LuaT_EX において、和文フォントの変更は `luatexja-fontspec` パッケージにより行う。たとえば、この文書ではプリアンブルに次のように書くことで、ソースコードのフォントを白源¹⁾に変更している。

```
\setmonofont{HackGen Regular}[BoldFont={HackGen Bold}]
```

1) <https://github.com/yuru7/HackGen>

第 6 章 補遺

6.1 パッケージの相性問題

1. cleveref は amsmath の後に読み込まなければならない
2. hyperref は xcolor の後に読み込まれると xcolor の色指定を利用できる
3. geometry は hyperref の後に読み込まなければならない（あるいは geometry に hyperref オプションを渡す）
4. hyperref を読み込んだ場合、nameref は `\begin{document}` の直前に読み込まれる

これを見ると分かるが、特に hyperref に関するものが多い。

hyperref

The environments `equation` and `eqnarray` are not supported too well. For example, there might be spacing problems (`eqnarray` isn't recommended anyway, see CTAN:info/l2tabu/, the situation for `equation` is unclear, because nobody is interested in investigating). Consider using the environments that package `amsmath` provide, e.g. `gather` for `equation`. The environment `equation` can even be redefined to use `gather`:

```
\usepackage{amsmath}
\let\equation\gather
\let\endequation\endgather
```

参考文献

- [1] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Learning a deep convolutional network for image super-resolution. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pp. 184–199, Cham, 2014. Springer International Publishing.
- [2] Ken Whistler, Mark Davis, and Asmus Freytag. Unicode technical report #17: Unicode character encoding model. <https://www.unicode.org/reports/tr17/tr17-7.html>, 2008. Accessed: 2021-08-07.

索引

J		符号化文字集合	4
JPEG	5	符号空間	4
		符号単位	4
L			
L ^A T _E X	9	へ	
Lua	9	ベクタ形式	5
LuaT _E X	9		
luatexja-fontspec	14	も	
		文字コード	4
P		文字集合	4
PNG	5	文字符号化形式	4
pT _E X	9	文字符号化スキーム	5
S		ら	
SATySF _I	9	ラスタ形式	5
SVG	5		
T			
T _E X	9		
U			
Unicode	4		
upT _E X	9		
か			
可逆圧縮	6		
画素	5		
に			
ニアレストネイバー法	7		
は			
バイキュービック法	7		
バイリニア法	7		
ひ			
非可逆圧縮	6		
ふ			
符号位置	4		

更新履歷

2021/08/11第 1 版公開