1 LaTeX で講義ノートを取ろう!

ashiato45　(http://ashiato45.github.io/)

1. これは何？

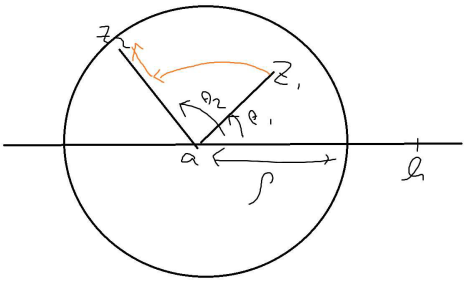
2014 年夏学期の間「解析学 4(ルベーグ積分)」「複素解析学 2」「代数学 1」の講義をリアルタイムで LaTeX でとった記録です．正直 LaTeX はただのライトユーザなのですが，そんなものでもなんとかなるものかと笑っていただければ幸いです．

1.1　想定する読者

LaTeX にある程度興味があり，基本的な操作ができることは仮定します．奥村晴彦先生の「LaTeX2e 美文書入門」を流し読みしたことがあれば十二分です．いくらかプログラミングの知恵的なものも出てきますが，プログラミングの知識は仮定しません．

1.2　実際どんな感じなの

こんなです↓



よって，

$$^{\exists}\, M > 0 : {}^{\forall}\, z \in B \cap H : \implies |g(z)|\,|z-a|^{1/2} < M \tag{59}$$

$z_1, z_2 \in B \cap H \implies$

$$|f(z_2) - f(z_1)| = \left| \int_{z_1}^{z_2} g(z) dz \right| \tag{60}$$

$$\overset{\text{黄色い積分路}}{\leq} \left| \int_{\theta_1}^{\theta_2} \frac{M}{r_1^{1/2}} r_1 d\theta \right| + \left| \int_{r_1}^{r_2} \frac{M}{r^{1/2} dr} \right| \tag{61}$$

$$= M r_1^{1/2} \left| \theta_2 - \theta_1 \right| + M \left[2 r^{1/2} \right]_{r_1}^{r?2} \tag{62}$$

$z_0 \in \overline{H}$ を固定する．

$$w = f(z) = \int_{z_0}^{z} \frac{dz}{(z-a)(z-b)(z-c)(z-d)} \quad (z \in \overline{H}) \tag{63}$$

よって，

$$^{\forall}\, \epsilon > 0 : {}^{\exists}\, \rho > 0 : {}^{\forall}\, z_1, z_2 \in B(a, \rho) \cap H : |f(z_2) - f(z_1)| < \epsilon \tag{64}$$

よって，コーシーの収束条件より，$\lim_{H \ni \to a} f(z)$ は存在し，$f(z)$ は $z = a$ で連続になる．$z = b, c, d$ でも同様.

主な理由は，

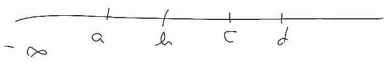
$$\int_a^{a+1} \frac{dx}{(x-a)^{1/2}} = \left[2(x-a)^{1/2} \right]_a^{a+1} < \infty \tag{65}$$

$z = \infty$ での f の連続性は，$z = \infty$ の近くで $g(z) \simeq 1/z^2$ で，

$$\int_1^{\infty} \frac{dx}{x^2} = \left[-x^{-1} \right]_1^{\infty} = 1 < \infty \tag{66}$$

となる．

(2) z_0 を $z_0' \in \overline{H}$ にかえると $f(z)$ は $\pm \int_{z_0}^{z_0'} g(z) dz$ だけかわる．*(*全体として平行移動する*)*．$z_0 = -\infty$ と取る．



2. 環境

2.1 LaTeX 環境

普通に TeXLive 2014 を導入しました．パスも自動で通るので楽ですね．

2.2 エディタ

Emacs 24.3 の Windows 用ビルドを使っています．エディタは TeX 系の場合は LyX や TeXworks が主流かと思いますが，Emacs は Vim と並んで古くからプログラマやハッカーに親しまれていたエディタで，高い拡張性で自分好みにすることができます．リアルタイムでノートを取りたいという状況で速い入力が求められていますから，このカスタマイズしやすいという性質はとても助けになります．ただ，癖のあるエディタですので慣れるには少し時間がかかるかもしれません．入門にはまず「Emacs チュートリアル」が良いとされていますので興味のある方は調べてみてください．

そこに TeX 編集用のモード [*1]YaTeX[*2] を導入しています．他に試したわけではないのですが，`\section` などの入力支援はもちろん，すでに書いてしまったところをコマンドで囲みたい，`\end{XXX}`に対応する `\begin{XXX}`にジャンプしたい，`\label` コマンドの名前を適当に埋めてほしいなど細かい機能が詰まっていて便利です．

また，Emacs にはかな漢字変換システム [*3] である SKK を入れています．SKK の 1 番の特徴は文節区切りを全て手動で指定することで，例えば「保つ」と入力したいときには「TamoTsu」のように漢字入力の始めで Shift を押しながら入力し，送り仮名のはじまりでまた Shift を押しながら入力する，という風にします．一見小指が忙しそうに見えて，まあ実際そうなのですが，こちらの意図しないところで文節が区切られて区切り直しにイライラすることがないですし，こちらの入力した通りになるので黒板を見たまま画面に目を落としたくないというときにも便利です．また，あまり指摘されないのですが辞書登録が非常に簡単で，登録されていない単語を変換しようとして候補が尽きてしまったときにはその場でどう書くべきなのかを教えることができます．そこで教えた書き方はそのまま文章の中に入力されるので，普段の IME で「あー，登録されてないんだ，仕方がないから全部消して一個一個漢字を打って後で登録しなきゃ」となるところが，一個一個漢字を打って単語を入力することそのものが辞書登録になり，非常に便利です．気楽に「広義一様収束」のような単語も登録でき，変換ミスはすごく減ります．

2.3 マクロ

全体的に略記のためのものですので，どれも簡単なものです．どの講義でもそこそこ便利に使えたコマンドは，

- `\newcommand{\tatev}[1]{\left(\begin{matrix}#1\end{matrix}\right)}`
  ちょっとインラインで縦ベクトルを打ちたいというときにわざわざ matrix 環境を作るのが面倒だと思って作ったコマンドなのですが，結局軽く行列を打ちたいときなどにも使っています．あまり込

[*1] Emacs では編集する文章の種類によって「モード」を設定し，その文章の特性にあうように操作や表示を変えることができます．

[*2] 広瀬雄二氏の作．`http://www.yatex.org/` から入手できる．関係ないが，「やさしい Emacs-Lisp 講座」は楽しく読ませていただいた．

[*3] MS-IME や ATOK のような，かなで打って漢字に変換するためのソフトウェア．

みいっていない数値のみの行列などだと便利に使えます.

- `\newcommand{\Forall}[1]{{}^{\forall}\hiderel{#1}\mathrm{\colon\;}}`
  講義だと略記としてよく全称記号 $\forall$ を使うわけですが，コマンドを使って変数とその変数の満たすべき条件とを書いて，成り立つ性質はそのあとに続けるという風に書くためのコマンドです．あと，全称記号が大きくなるのは気に入らないので上付き文字として表示するようにしています．次で例を示します.
- `\newcommand{\Exists}[1]{{}^{\exists}\hiderel{#1}\mathrm{\colon\;}}`
  存在記号 $\exists$ について同様のコマンドです．例えば,

```
\Forall{x\in I}
\Forall{\epsilon > 0}
\Exists{\delta > 0}
\Forall{y \in (x-\delta, x+\delta)\cap I}
 f(y)\in (f(x)-\epsilon, f(x)+\epsilon)
```

$$\forall x \in I\colon\ \forall \epsilon > 0\colon\ \exists \delta > 0\colon\ \forall y \in (x-\delta, x+\delta) \cap I\colon\ f(y) \in (f(x)-\epsilon, f(x)+\epsilon) \tag{1.1}$$

のようになります.

- `\newcommand{\openright}[1]{\left[#1\right)}` 半開区間のためのものです．特に普通のエディタだと括弧は自動的に対応するものが補完されてしまうので，左右異なる括弧を使うことになるこのような場面で便利です.
- `\newcommand{\banme}[2]{\stackrel{#1}{\stackrel{\vee}{#2}}}` 列ベクトルで何番目かを指定したいときなどに使います．例えば

$$e_k = (0, \dots, \overset{k}{\overset{\vee}{1}}, \dots, 0) \tag{1.2}$$

- ```
  \newcommand{\easypicture}[1]{
   \begin{center}
   \includegraphics[width=\textwidth/2]{#1}
   \end{center}}
  ```
  講義中にさくっと書いた絵を何も考えずにとりあえず入れこみたいときのためのものです.
- `\newcommand{\desceq}[1]{\overset{\text{\fbox{\tiny #1}}}{=}}` イコールに注釈を入れることは講義でよくあると思いますが，そのためのものです．同様に不等号や同値にも定義しています．例えば,

$$\lim_{n\to\infty} \int f_n(x)dx \overset{\text{一様収束性}}{=} \int \lim_{n\to\infty} f_n(x)dx \tag{1.3}$$

のように使います.

- `\def\MARU#1{\textcircled{\scriptsize #1}}` 丸囲みのコマンドです．こちら `http://icepotato.cocolog-nifty.com/blog/2013/07/tex-5ee7.html` を使わせていただいています．奥村先生の本にも`\MARU` は定義されているのですが，後述の `enumerate` と相性が悪かったのでこちらを使っています.
  ```

他にも講義中によく使うと思った記号はその場で Emacs の画面を [C-x 2] で 2 分割して，一方でコマンドを定義してその後の入力を楽にしたりしていました．例えば，「σ-加法族」は数式モードに入ったり日本語に入ったりでキーボード入力と相性が悪いので，これだけのためにコマンドを`\sfield` を定義したりしました．また，`\underbrace` を

$$|f(x)-f(y)| \leq \underbrace{|f(x)-f_n(x)|}_{A_1\text{とする}} + \underbrace{|f_n(x)-f_n(y)|}_{A_2\text{とする}} + \underbrace{|f_n(y)-f(y)|}_{A_3\text{とする}} \tag{1.4}$$

などのようによく使うのですが，いかんせん長いので (入力補完を使ってもいいのですが)`\ub` のようにしていました．

2.4 導入したパッケージ

amsmath,amsfonts,amssymb 数学系の文章を書くなら外せないでしょう．数式モードの`\align` は常用するはずです．今更の豆知識ですが，デフォルトで使える`\eqnarray` よりも`\align` を使うべきだということで，「Avoid eqnarray!」という記事 `http://tug.org/pracjourn/2006-4/madsen/madsen.pdf` があるのでよかったらどうぞ．

enumitem `enumerate` 環境を強化する環境で，番号の振り方を`\begin{enumerate}[label=(\arabic*)` のようにすれば (1)(2)…のように番号が振られ，`\begin{enumerate}[label=\MARU{\arabic*}]` のようにすれば①②…のように番号が振られ，`\begin{enumerate}[label=Step \roman*]` のようにすれば Step i, Step ii, …のように番号が振られるという風に柔軟に番号が振れるようになります．黒板に合わせるのに便利です．

breqn 長い数式のときの改行というのは面倒なものですが，breqn の`\dmath` は数式中の演算子を見つけて適当に区切りのいいところで改行してくれます．ただ，集合の中の条件のところで改行されてしまったりと何も考えずに使うと余計に面倒なことになるので単純だけど長い式 (やたら多い足し算とか) のようなときに主に便利です．ちなみに，改行は`\hiderel` で止めることができます．

ntheorem 定理環境を作るためのパッケージです． 番号の振り方など機能が豊富なので調べてみてください．

(mathbbold) パッケージではないのですが，

```
\DeclareSymbolFont{bbold}{U}{bbold}{m}{n}
\DeclareSymbolFontAlphabet{\mathbbold}{bbold}
```

として`\bbold` で小文字の黒板太字を打てるようにしています．雪江明彦先生の代数学の本に影響されました．

kmathmacro Twitter で知りあったあおえういさんの作られて LaTeX マクロで，痒いところに手が届くような機能が入っています．`https://bitbucket.org/keno1728/kmathmacro` から入手できます．全機能見たわけではないのですが，内部の括弧を自動的にサイズ調整してくれる`\paren`，集合の条件の側を数式，テキスト，複数行テキストと 3 通りに書かせてくれる便利な`\set`，自然数`\N`，実数`\R` など一通りが入っています．

2.5 図の挿入

Windows にデフォルトでついている「ペイント」で黒板の図をさっと書いて先の`\easypicture`で挿入しています．別にそのつもりで買ったわけではないのですが，手持ちのノート PC にタッチパネルがついているので図を描くには便利です．

2.6 コンパイル環境

ワードプロセッサ系のソフトから TEX 系のソフトに移ってきてはじめに戸惑うのが WYSIWYG でないこと，コンパイルが必要なことですが，このコンパイル，pdf 生成，pdf 表示は自動化しておくことができます．そのために，OMake と SumatraPDF を使います．

OMake はビルドツールの一種で，あるファイルを作るために必要なコマンドを必要なだけ実行することができます．例えば，「file.aaa」を作るのに「file.bbb」を使って「commandX file.bbb」を実行する必要があり，「file.bbb」を作るのに「file.ccc」と「file.ddd」を使って「commandY file.ccc file.ddd」を実行する必要があるときにはその旨を OMakefile として書いておいて「omake」として OMake を実行することで，「file.bbb」「file.ccc」「file.ddd」のうち変更されたものだけを検知してそれに応じたコマンドを実行して，それで変更が起きればさらに連鎖的にコマンドを実行して最終的に「file.aaa」を作ってくれます．これを利用して，

$$\text{main.tex} \xrightarrow{\text{platex}} \text{main.dvi} \xrightarrow{\text{dvipdfmx}} \text{main.pdf} \tag{1.5}$$

という連鎖を OMakefile で指示しています．また，images フォルダに画像があった場合には extractbb コマンドを実行してそれに対応する boundingbox ファイル (*.xbb) を自動で生成するようになっています．と言うと大仰ですが，実のところこちら `http://qiita.com/nojima/items/eebbea5574a8ff42f8b2` をいじっただけです．「main.tex」というファイルをメインに使っているので，「TARGET = main」に，「–kanji」は環境に合わせて直しましょう．また，Bounding Box 生成のコマンドとして「extractbb」を使いたかたので，「EBB = extractbb」としました．TEXLive2014 のデフォルトは ebb のはずなので，別に ebb のままでもいい気もします．

OMake は変更を加える度に手動で実行してもいいのですが，「omake -P」と監視モードで実行すると，対象にしているファイルの変更を自動的に検知して OMakefile の指示通りにコマンドを処理してくれます．

OMake は `http://omake.metaprl.org/index.html` から入手できます．Windows ユーザーは Ohter binaries から.msi ファイルをダウンロードしてインストールするといいでしょう．パスは確か自動では通らなかったと思うので，「omake.exe」のあるディレクトリにパスを通しましょう [*4]．

SumatraPDF は，PDF ビューアです．PDF ビューアとしては Adobe Acrobat Reader が普通かと思いますが，Acrobat Reader はファイルを開いている間ファイルをロックしてしまうので，ファイルを開いている間に変更を加えることができません．それに比べ，SumatraPDF は開いているファイルに変更が加えられた場合にそのファイルを再読み込みするという非常に便利な機能がついています．先の OMake の監視モードと組み合わせると，エディタで「main.tex」を変更すると OMake が自動的に変更を検知して platex,dvipdfmx を実行して「main.pdf」が生成され，その生成 (更新) を SumatraPDF が検知して再読み込みするという非常に快適な流れができます．Sumatra PDF は `http://blog.kowalczyk.info/software/`

[*4] よく分からない人は「パスを通す」でぐぐりましょう．デフォルトのインストール先だと「C:/Program Files(x86)/Omake/bin」になると思います．

sumatrapdf/download-free-pdf-viewer-ja.html から入手できます．

まとめると，編集しているディレクトリ構成は基本的に，

（作業フォルダ）
├ main.tex（**これを編集**）
├ main.dvi（main.tex **から生成**）
├ main.pdf（main.dvi **から生成，**SumatraPDF **で監視**）
├ OMakefile（OMake **に指示を出す**）
└（images）
　├（**画像**）
　├（Bounding Box）
　├…
　└（**画像**）

のようになっています．

また，作業がすぐ開始できるように，

```
start sumatrapdf main.pdf
runemacs main.tex
omake -P
```

というバッチファイルを「takenote.bat」という名前で用意してパスを通しています．

2.7 ノートの管理

ノートの管理には Git と Github を使っています．Git はバージョン管理システムの 1 つで，それが何かを説明するのは一言だと難しいのですが，あるプロジェクトでのファイルが製作の過程でどのように追加・変更・削除されていったかという過程を記録して，必要なときにその過程を利用できるようにするためのソフトウェアです．例えば，ファイルを変更していってある時点のファイルまで変更を巻き戻したいというときに利用できます．本来は複数人でプロジェクトを進行するためのものなので，複数人がファイルに手を加えたときにその変更を統合して執筆者・開発者たちにその変更を伝える機能もあります．Git が管理するプロジェクトを「リポジトリ」とよびます．リポジトリが管理しているファイルを変更してキリが良くなったところで，そのファイルの変更を Git の「変更を登録する候補の置き場」[*5] に登録し，変更をリポジトリに記録する [*6] というのが基本的な使い方です．コミットしたとき，その時点のリポジトリの管理しているファイルの状態は全て保存され，その後どれだけファイルを変更しても必要なときにはその記録を呼び出して復元することができます．

Github は，Git リポジトリのオンラインの置き場で，無料で利用することができます．ローカルのリポジトリと Github のリポジトリは同期することができますから，手元のプロジェクトの変更の記録を全て Github に置いておくことができますし，何かあったとき，あるいは別の PC で作業したいときは Git に置いてあるリポジトリと同じものをローカルに持ってくることができます．

特にバージョン管理まで持ち出して変更を記録する必要まではなかったのですが，いわゆる「何もしてないのに壊れた」という状況はたまにありますのでやってもいいかなぐらいの気分でした．Git を使った

[*5] ステージといいます．
[*6] 「コミットする」といいます．

一番の理由は Github で，LaTeX でとったノートをローカルに置いておいたら PC が故障したときにノートを全て失いますから，オンラインストレージが欲しかったというのが一番の理由です．先に述べた理由で Git は Github との同期が簡単にできますから，ノートを取ったらコマンド一発叩いてすぐに同期できるのが非常に便利です．

Git の使い方はなんとなく検索すれば沢山分かりやすい資料が出てきますが，一応「入門 git」は持っています．

先に Git は本来複数人でプロジェクトを進めるためのものだということを述べましたが，この Github に置いてあるリポジトリを数学科の院生の方と共有したところ (オンラインなのでこういうことができます．) ノートの誤字や誤りを修正していただけるという思わぬ効果が得られました．これを推し進めれば，講義ノートを複数人で取るという未来っぽいことも可能になるのかもしれません．

3. 夏学期やってみて

3.1 情報と表示の分離は意識しないほうがいい

コンピュータを扱う人のなかでは「ある文章がどう表示されるか」と「その文章がどのような情報を含んでいるか」は分離するのが一般に良いとされていて，HTML と CSS はその代表例だと思います．例えば，「この文章のここを赤字にする」という表示の指定を「この文章のここが重要である」という「情報」と，「重要なところは赤字で表示する」という「表示」の 2 つに分けたほうが良いということです．このように分離しておくと何が便利かというと，例えば文章を書きすすめていったときに「重要なところを赤字ではなく太字にしたい」などという変更が生まれたときに，「重要なところは赤字で表示する」という表示の指示を「重要なところは太字で表示する」と変更するだけで重要なところが全部太字になった文章が得られて便利だからです．また，「この文章から『重要である』ところだけをプログラムを使って抜き出したい」というときに分離をしておけば「重要である」と書いたところを抜きだすプログラムが簡単に書けるようになります．しかし，このような分離はやってみれば分かるのですがどこまでが表示の領分でどこからが情報の領分なのかは結構曖昧になってしまい，そのためにもたつくことになりがちです．特に今回は講義ノートを手早く取りたいということですし，そのために悩んでいると私なんかは途中で飽きてしまうので，そのあたりの分離は考えず，そのまま「この文章のここを赤字にする」といった風にノートを取っていったほうが良いのではないかと思いました．

3.2 がんがん改行しよう

TeX では，改行の指示をしない限りは得られる出力は改行されません．そこで，編集上便利なところで改行しながら書いていくと楽です．例えば，長く連なった等式では $=$ が出る度に改行しておくと編集が楽になります．

3.3 まめにコンパイルしよう

LaTeX のエラーメッセージは結構アテにならないことが多く，エラーが出ている行が必ずしも書き間違えた行だとは限りません．何かエラーが出たときに「ここまでは大丈夫なはずだからミスはここからここの間だ」という推論をしやすくするために，まめにコンパイルすると楽です．

3.4 コメントアウトを使えるようになろう

とはいっても，やっぱりどこがエラーを出しているのか分からないということはあります．そのようなときに変更した箇所を一度コメントにしてしまってコンパイルして，どこを消せばコンパイルが通るのかを調べてエラーが出ているところを特定するという方法があります．文章のある部分をコメントにしてしまうことを「コメントアウト」とよくよびます．

また，どこがおかしいかを探っている間に講義がすごく進んでしまい，これ以上エラー探しに時間が割けないという場合は仕方がないので，その部分を全てコメントアウトしてしまい，コンパイルが通るようにしてその箇所を飛ばしてノート取りを続けることができます．家に帰ったらどこが文法的におかしいのかを落ち着いて特定して，その部分のノートを復元しましょう．うまく動かなかたとはいえコメントとして LaTeX コードが残っていますから，割合楽に復元できるはずです．

コメントアウトはそこそこ高度なテキストエディタには搭載されているはずで，Emacs+YaTeX の場合には「M-x comment-region」でコメントアウトが利用できます．

3.5 LaTeX にあうように適当に変える

講義の黒板そのままを LaTeX で再現しようとするととんでもなく高度なことをする羽目になることは多々あります．そのようなときは，その情報を LaTeX での表現にあうように適当に変えて取りましょう．例えば，あるところに矢印がはってあって「ここは XXX」と書いてあった場合，LaTeX だと大変なのでその箇所の☆を打っておいて，「☆は XXX」と後ろに書くなどが考えられます．どうしても気になるならば，家に帰ってからその箇所を修正すればいいでしょう．

4. 反省・今後の目標

4.1 太字とかの特殊書体が楽

手書きだと黒板太字などの文字がぐちゃぐちゃになるものですが，それが綺麗に出るのはストレスが減って良かったと思います．後で読むときもずっと楽でした．

4.2 可換図式

冬学期になって可換図式を書くことが増えたのでいい加減 LaTeX で可換図式を書く方法を覚えなければならないと思いました．

4.3 画像の追加

さきの OMakefile だと，画像の変更は自動的に検知するのですが画像の追加は検知してくれません．OMake の使い方をちゃんと調べて，あの OMakefile が何をしているのかを正確に知らなければなあと思います．

4.4 行列

行列の入力がいまの腕ではとても遅く，行列のある行や列に「第 i 行」などと注釈をさしたいなどということがとても手軽にはできない状況なので何か考えないといけません．入力支援として例えば表計算ソフトなどを使うことも考えたほうがいいのかもしれません．

5. FAQ

5.1 MS Word の数式入力使ったほうが良いのでは

最近は結構いいらしいですね．しかし Word 持ってないんです．

5.2 成績は良くなりましたか

(^ー^)？

5.3 タイプスピードはどのぐらいですか

たまに Twitter のタイプスピードチェックで調べるのですが，調子が良くて 6key/sec ぐらいです．講義を取っているときはそんなに調子が良くないので 3key/sec ぐらいではないでしょうか．