卒業論文

Jupyter notebook のための CLI ツール nb_util の開発

関西学院大学理工学部

情報科学科 西谷研究室

27014561 小脇　雅健

2018 年 3 月

目 次

表 目 次

図 目 次

第1章　序論

1.1　背景

プログラム開発では，統合開発環境がいくつも用意されているが，多くの場合 terminal 上での開発が一般的である．しかし，簡単なプログラムを作成する際や，物理計算，データを解析する者にとって素早く，容易にコードに触れることが大変重要であるが，以下のような手順を繰り返す必要があり，非常に煩わしい作業になってしまう．

1. ソースコードファイルを開く
2. コードを変更する
3. 変更された内容を保存
4. OS のコンソール画面でファイルを実行
5. 実行結果を確認して 1. に戻る

上記のような煩わしさを解決する Jupyter notebook という GUI(Graphical User Interface) ベースで，ブラウザ上でコードを実行するためのインタラクティブな環境が開発・提供されている．データ解析や，物理計算，簡単なプログラムを実行する際には非常に有用なツールであり，データサイエンティストに広く利用されている．その例として，機械学習者向けに執筆されている,『Python ではじめる機械学習』や『Ipython データサイエンスブック』などの本も，Jupyter notebook で書かれている [1, 2]． さらに，Jupyter notebook はさまざまなプログラミング言語をサポートしており，尚且つプログラムを実行できるだけでなくテキストや画像を取り込むことが簡単にできる．

1.2 目的

そこで，西谷研究室に在籍している学生は，Ruby のプログラムを作成する時や物理計算をする際，Jupyter notebook を使用し日々研究に励んでいる．しかし Jupyter notebook だけでは，作業を進める上で事足りないことが数多く見受けられた．

- Jupyter notebook で作成したファイルを一つのファイルにまとめる際に，わざわざ Jupyter notebook を開いて，コピーアンドペーストをしなければならない．
- Jupyter notebook で作成したプログラムは，ファイル操作するプログラムや複雑なプログラムになると Jupyter notebook では実行できないので，terminal 上での開発に移行する．しかしその際，ソースコードとしてファイルを書き出す機能が Jupyter notebook には備わっていないので，コピーアンドペーストをしなければならない．
- 何を書いたか中身を確認する際に，Jupyter notebook を起動して中身を確認する．しかし，さっと中身を確認したいだけなのに，かなりの手間暇がかかってしまう．そこで，Linux に標準搭載されているコマンド `cat` を使用して中身を確認しようと試みるが，Jupyter notebook を構成する全ての情報が表示される為かなり見にくく使い勝手が悪い．
- 作成したものを，LaTeX に変換する際に，Jupyter notebook に付随しているもので変換することは可能だが，日本語は文字化けして変換れず，さらにフォーマットは醜いものに変換されてしまう．

そこで，本研究では Jupyter notebook を使う上で，上記の問題点を解決し，より使い勝手を良くする「nb_util」というアプリケーションを作成する．さらに，作成したアプリケーションをパッケージ化し，世界中に配信することで今後の継続的な発展に繋がると考える．

第2章　手法

本章では，本研究で nb_util を作成する際に使用した開発手法，周辺ツールについて記述する．

2.1　アジャイル開発手法

今回 nb_util を開発する手法の方針として，アジャイル開発 (Agile Development) を採用した． アジャイル (Agile) とは，直訳すると「素早い」「機敏な」と言う意味を持っている．期間を短く区切って優先度の高い機能から，実装とテストを始め，徐々に機能を増やしながら，随時実装とテストを行っていく．これにより「最後にならないと全体の仕様が見えない」というリスクを軽減できる（図 2.1）．さらに，ユーザーや顧客のフィードバックを受け入れながら開発を進めることもできる [3]．そのため，不具合の発生や仕様変更などにも迅速かつ柔軟に対応できる．

アジャイル開発手法の共通の見解として，アジャイルソフトウェア開発宣言がある．その具体的な例としては，

- プロセスやツールよりも，個人と対話
- 包括的なドキュメントよりも，動くソフトウェア
- 契約交渉よりも，顧客との協調
- 計画に従うことよりも，変化への対応

が挙げられる．これの項目について，左記のことがらに価値があることを認めながらも，右記のことがらにより価値をおくことが，アジャイル開発の考えである [4]．

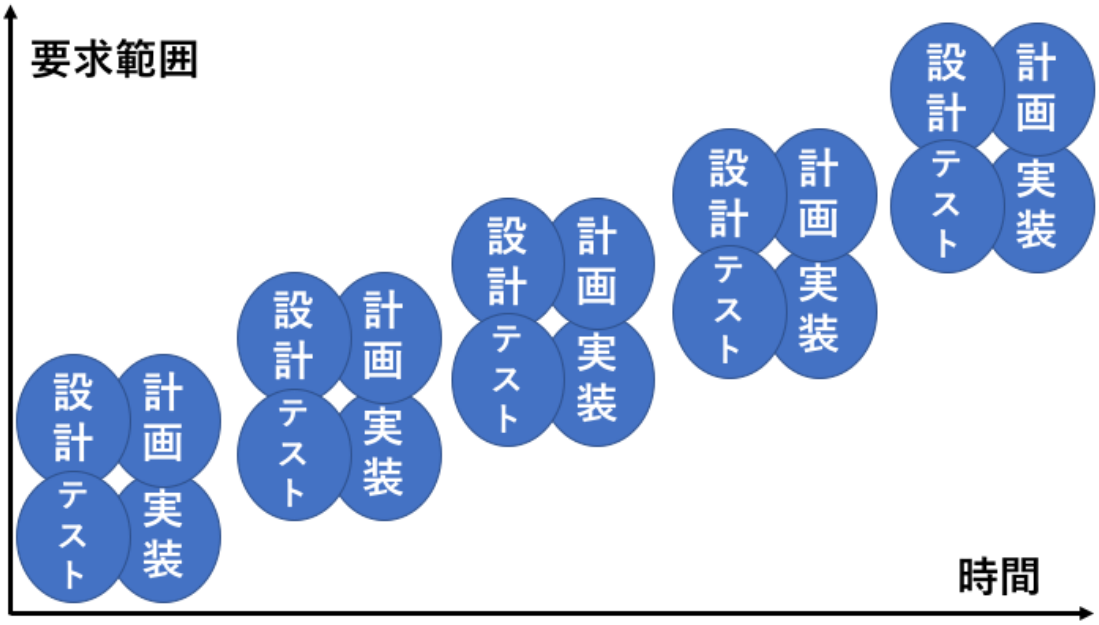

図 2.1: アジャイル開発のプロセス.

今回 nb_util を開発するにあたり，アジャイル開発手法に準じて，ある程度実用的な機能が実装され尚且つ，動作するプログラムを研究室の学生に配布し，実際の卒業論文作成に使用してもらう． さらに，Github，RubyGems.org に公開し世界中の使用者から不満や不具合についてのフィードバックをもとに改善をする．この流れを何度も繰り返すことによりより，ユーザーにとってより使いやすいアプリケーションの提供を試みる．

2.2 Jupyter notebook

Jupyter notebook 従来の機能は，Ipython Notebook の機能を継承したものであり，Web ブラウザ上で Python を実行できることや，グラフ描画パッケージとの相性が良いことから人気を集めている [5].

Jupyter notebook は，開発作業領域が「ノートブック」と呼ばれる 1 つの画面で完結でき，図 2.2のようにセル（プログラムが記入する枠）の下に実行結果が表示される．

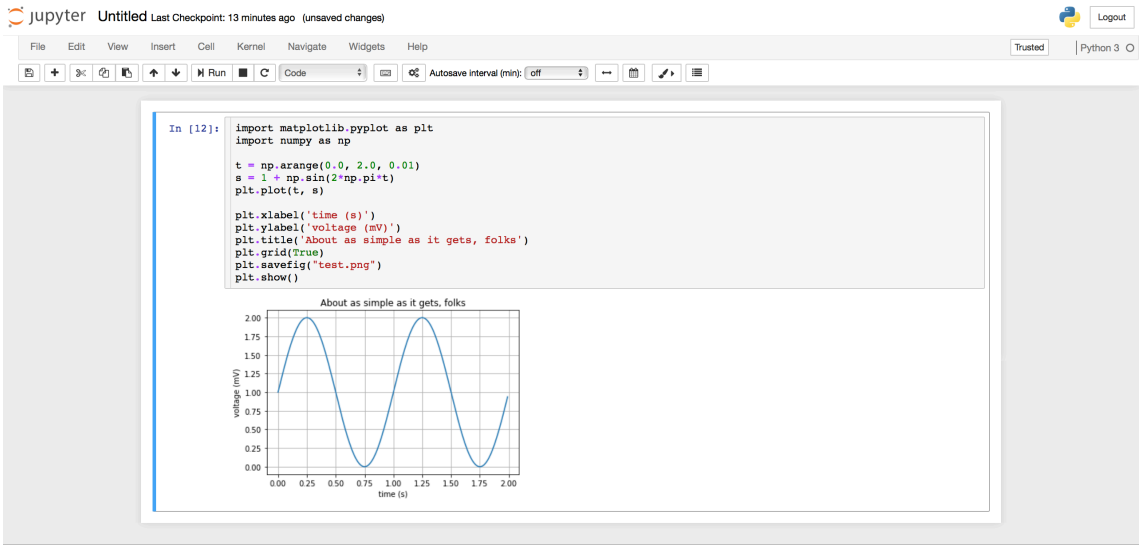

図 2.2: Jupyter notebook のメイン画面.

この機能によって，少しずつプログラムの実行結果を確認しながら作業をすることができる．また，実行結果をセル単位で記憶しているため，途中からやり直すことも容易にできる．さらに，プログラムを書きながら，違うセルで Markdown 形式で文章を書くことも可能なので，そのまま論文としてまとめることができる． しかし，Jupyter notebook に搭載されている機能には様々な問題があり，本研究で改善した具体的な内容は 3 章に記す.

2.3 CLI(Command Line Interface) 作成ライブラリ

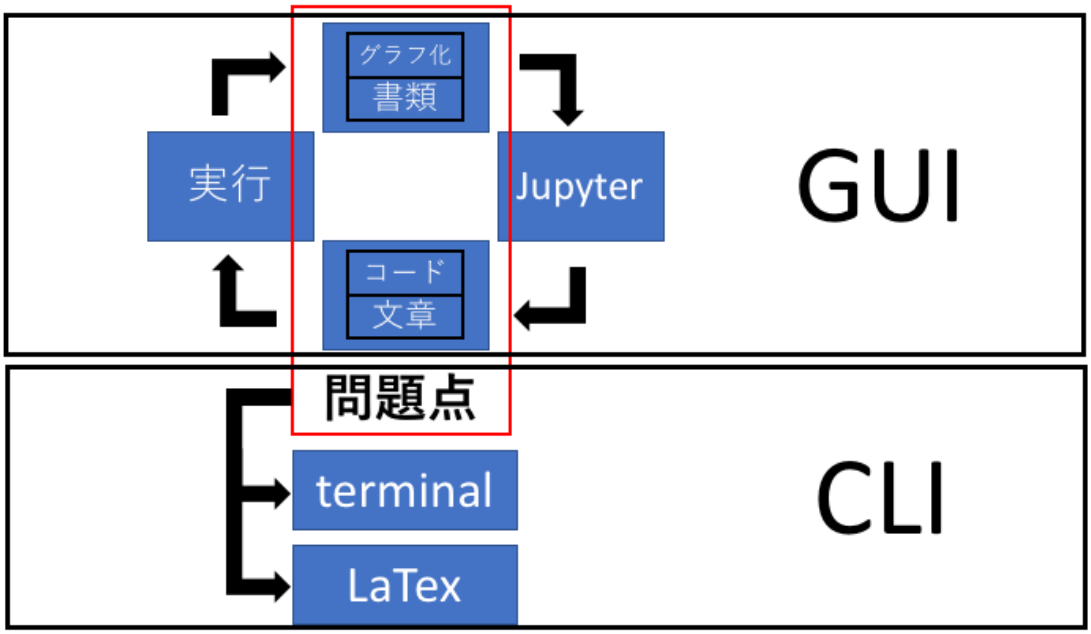

図 2.3: Jupyter_notebook を使用する際の問題点の解決方法.

Jupyter notebook を使用する場合その実行環境で以下の様なサイクルを行う.

1. Jupyter notebook を起動
2. 文章，プログラムを作成する
3. Jupyter notebook 上で実行する
4. 2 の内容が生成される

しかし，2.2 節で述べた通り，Jupyter notebook を使用する上で数多くの問題点がある．それを解決する場合，terminal や LaTeX の CLI 環境に移行し作業を行う． そこで，本研究で開発した nb_util は，作業の利便性を考え CLI 環境で提供する． CLI 環境を構築する中で，Thor または Optparse という有名なコマンド解析ライブラリが存在する．Optparse のオプションの省略記法は，覚えにくく，分かりにくいという問題があるが，Thor はフルワードを使った自然言語に近い記述法である．さらに，Optparse でのコマンド定義は Thor より複雑で記述が長くなる．従って，本研究での nb_util 開発に，Thor を用いる．

2.4 nb_utilのパッケージ化，パッケージの公開

ライブラリとは便利なパーツのようなものであり，Ruby には最初から備わっている公式のライブラリと，インストールすることで簡単に利用することができるライブラリがある．それらは，プログラム中で宣言することで容易に利用する事ができる．さらに，自ら作成しパッケージ化することが可能なので，本研究で作成した nb_util をパッケージ化し，gem.org に公開した．開発の際パッケージの管理，ソースコードのバックアップにそれぞれ， Bundler と Github を利用した．

2.4.1 RubyGems

RubyGems は通称，gem と呼ばれており，Ruby 用のパッケージ管理ツールである． gem 自体はプログラミング言語 Ruby のファイルに付属されていて，無料で誰でも簡単に使うことができる．さらに Ruby 用ライブラリのインストール，アンインストール，バージョン管理を簡単にすることができる．また，同じようなパッケージ管理ツールとして，JavaScript 用の npm，iOS アプリ用の swift-package-manager などがある．

Gemfile

Bundler はアプリケーションで利用する gem パッケージを，Gemfile というファイルに列挙し， `$ bundle install` コマンドを実行すると，gem パッケージの依存関係を解決し，依存するパッケージを全てインストールすることができる． さらに，Gemfile は特殊なフォーマットで構成されているファイルでなく，Ruby プログラムである [6].

Bundler

本研究で nb_util を開発する上で，ライブラリのバージョンの違いで動かなくなる事を避けるためバージョンの管理などの作業で利用した．Bundler はアプリケーション単位で利用する gem パッケージを定義し，依存関係を解決する為のアプリケーションである．アプリケーションごとに gem パッケージの依存関係を束ねて管理することで，特定のアプリケーションだけが利用する gem パッケージを簡単に管理することができる [6].

2.4.2 Github

本研究では，ソースコードなどのバックアップ作業でGithub を利用した．Github はソフトウェア開発のためのソースコード管理サービスである．複数人が携わるソフトウェア開発において，ソースコードの共有や，バージョン管理といった作業は必要不可欠である．

2.5 Pandoc

Jupyter notebook の変換機能に深く関わっているものであり，本研究でも使用している．Pandoc はマークアップ言語で書かれた文書を別の形式に変換するツールで，入力にMarkdown, HTML, LaTeX, など， 出力に左記の他プレーンテキスト，Word docx, PDF などをサポートしている [7].

2.6 LaTeX

LaTeX は TeX の上に構築されたフリーの文書処理システムで，一つのまとまったソフトウェアではなく，様々な働きをするソフトウェアで，Leslie Lamport により開発された文章作成ソフトである． TeX は組版のために開発された言語であり，そのままでは使いにくい点があるので，LaTeX では一般的な文書作成に便利な機能拡張を施したものである．

第3章　アプリケーション概要

本章では，本研究で開発した CLA(Command Line Application) のインストール方法，使用方法，各コマンドの動作概要を記述する．

nb_util は,「Bundler」と呼ばれる gem パッケージを利用して開発し，Jupyter notebook を使用する際に，より使いやすくなる為の支援ソフトである．主に以下の機能を持つアプリケーションである．

1. my_help（json 形式）から，Jupyter notebook（ipynb 形式）に変換する．
2. 複数の Jupyter notebook ファイルを，一つのファイルにまとめる．
3. Jupyter notebook で作成したプログラムを，そのファイル拡張子にしてファイルに書き出す．
4. 中身を確認する際に，Jupyter notebook を起動せずに，コマンドラインで確認する．
5. Jupyter notebook で作成した文章を，卒業論文の形式の LaTeX に変換する．

3.1　インストール方法

3.1.1　gem を使用してインストールする場合

ruby でパッケージやライブラリを使用するために必ず必要なインストール方法である．他の開発者が完成させたパッケージを,「gem intsall」の後にパッケージ名をつけるだけで，素早く簡単にインストールすることができる．

インストール方法

`$ gem install nb_util` で,「nb_util」 をインストールする．

3.1.2 bundle を使用してインストールする場合

gem はバージョン更新が頻繁に行われており，gem とアプリケーションとの互換性が無くなり動かない場合がある．その問題を解決する方法として，Bundler という gem 管理ツールがある．Bundler を利用することで，システム側には gem はインストールされない為，ディレクトリ単位でアプリケーションをインストールし管理するという方法である．

インストール方法

Github から fork して clone することで nb_util を使うことができるようになる．

```
$ git clone git@github.com:EAGLE12/nb_util.git
$ cd nb_util
```

後の作業は，bundle を使って行う．

`$ bundle install` で，nb_util.gemspec に記述されている必要な gem がインストールされる．さらに，`$ rake install` をすることで，自分自身の環境にインストールされ，CLI(Command Line Interface) としてディレクトリの場所を問わず実行可能になる．

3.2 アンインストール方法

3.2.1 gem を使用してアンインストールする場合

インストールされた「nb_util」の gem を消去する場合は，インストール時とほとんど同じ作業でアンインストールすることが可能である．

アンインストール方法

`$ gem uninstall nb_util` でアンインストールできる．しかし複数のバージョンがインストールされている場合，アンインストールするバージョンを以下のように指定する．

```
Select gem to uninstall:
 1. nb_util-0.3.4
 2. nb_util-0.3.8
 3. nb_util-0.4.0
 4. nb_util-0.4.2
 5. nb_util-0.4.8
 6. All versions
> 6
Successfully uninstalled nb_util-0.3.4
Successfully uninstalled nb_util-0.3.8
Successfully uninstalled nb_util-0.4.0
Successfully uninstalled nb_util-0.4.2
Remove executables:
    nb_util

in addition to the gem? [Yn]  y
Removing nb_util
Successfully uninstalled nb_util-0.4.8
```

3.2.2 bundleを使用してアンインストールする場合

bundle を使用してる場合，システム側には gem はインストールされてない為 Github から clone したフォルダを消去するだけでアンインストールされる．もし，`$ rake install` している場合は，3.2.1 節の方法でアンインストール可能である．

3.3 nb_utilが用意しているタスク

起動方法は，全て各自のターミナルから実行する．3.1 節の方法で問題なくインストールできた場合，`$ nb_util` と入力すると各種使用可能なコマンドが図 3.1のように表示さ

れる．

本研究で開発したアプリケーションは，オプションを付与することによって以下の動作を行う．

1. `nb_util combine` （Jupyter notebook を一つに結合する）
2. `nb_util getcode` （Jupyter notebook からソースコードのみ抽出する）
3. `nb_util help` （nb_util が用意しているコマンドのヘルプを表示する）
4. `nb_util iputs`（Jupyter notebook の中身を表示する）
5. `nb_util ipynb2tex`（Jupyter notebook で書いた文章を LaTeX に変換する）
6. `nb_util yaml2ipynb`（yaml から Jupyter notebook に変換する）

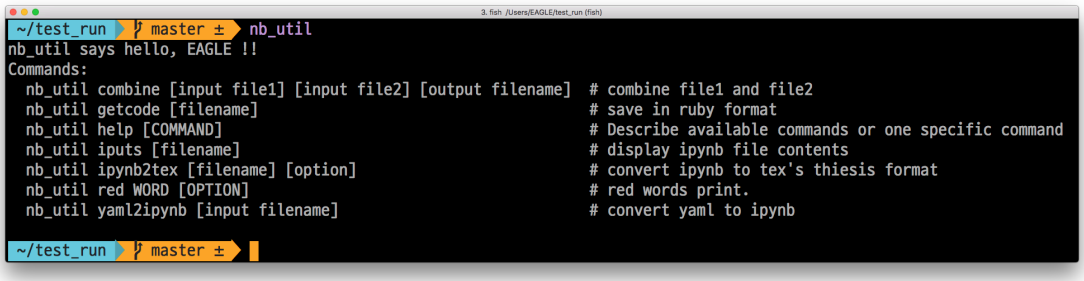

図 3.1: nb_util スタート画面．

3.3.1 nb_util combine [input file1] [input file2] [output filename]

Jupyter notebook で作成した，任意の二つのファイルをコマンドライン上から一つのファイルにまとめる．

コマンドの大まかな動作を，図 3.2に示す．

図 3.2: nb_util の combine コマンド解説.

動作方法

1. コマンドの第一引数，第二引数に任意の Jupyter notebook ファイルを選択する.

    1. `$ nb_util combine sample1.ipynb sample2.ipynb sample1_sample2_combine`

コマンドの第三引数に，任意の名前を入力し書き出しが行われる．書き出されるファイルは，コマンドを実行しているカレントディレクトリに保存される.

combine コマンドにより生成されるディレクトリー構成

```
.
├── sample1.ipynb
├── sample1_sample2_combine.ipynb
└── sample2.ipynb
```

3.3.2 nb_util getcode [filename]

Jupyter notebook でプログラムを作成したものをファイルとして書き出す際，使用していた kernel の言語を自動分析し，最適な拡張子でファイルを保存する．

コマンドの大まかな動作を，図 3.3に示す．

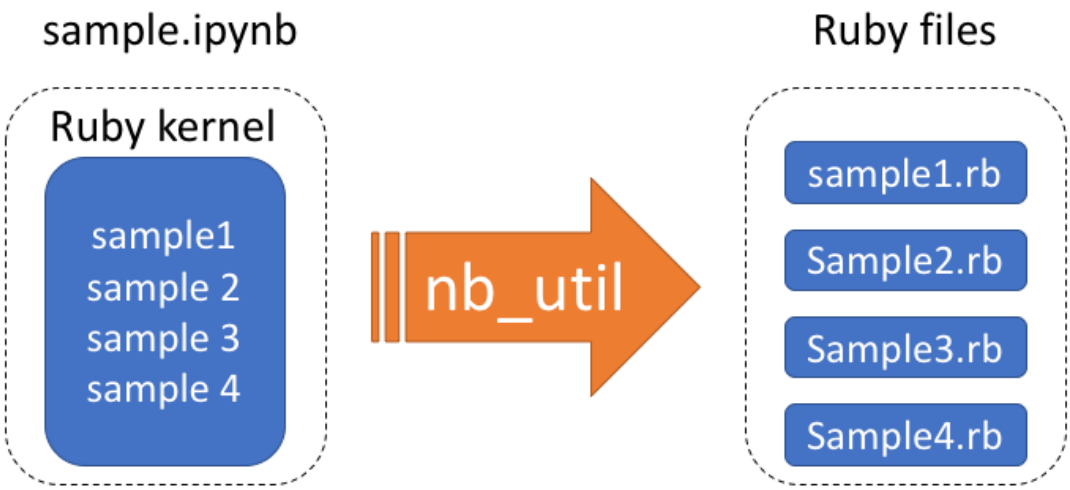

図 3.3: nb_util の getcode コマンド解説．

動作方法

1. コマンドの第一引数に任意の Jupyter notebook ファイルを選択する．

    1. `$ nb_util getcode sample.ipynb`

書き出されるファイルは，コマンドを実行しているカレントディレクトリに保存される．

getcode コマンドにより生成されるディレクトリー構成

```
├── sample.ipynb
└── sample_code1.rb
```

3.3.3 nb_util iputs [filename]

通常 Jupyter notebook の中身を確認する際，以下の作業を踏むのが一般的である．

1. terminal を起動する．
2. Jupyter notebook を起動する．
3. 目的のファイルを開く．

しかし，中身を確認するだけの為に上記の作業を繰り返すのは手間が掛かってしまう．そこで，Linux に標準搭載されている，`$ cat` コマンドを使用する．しかし，この方法も問題点があり，Jupyter notebook のファイルは図 3.4の様な構成になっている為，不要な情報も表示されてしまう．

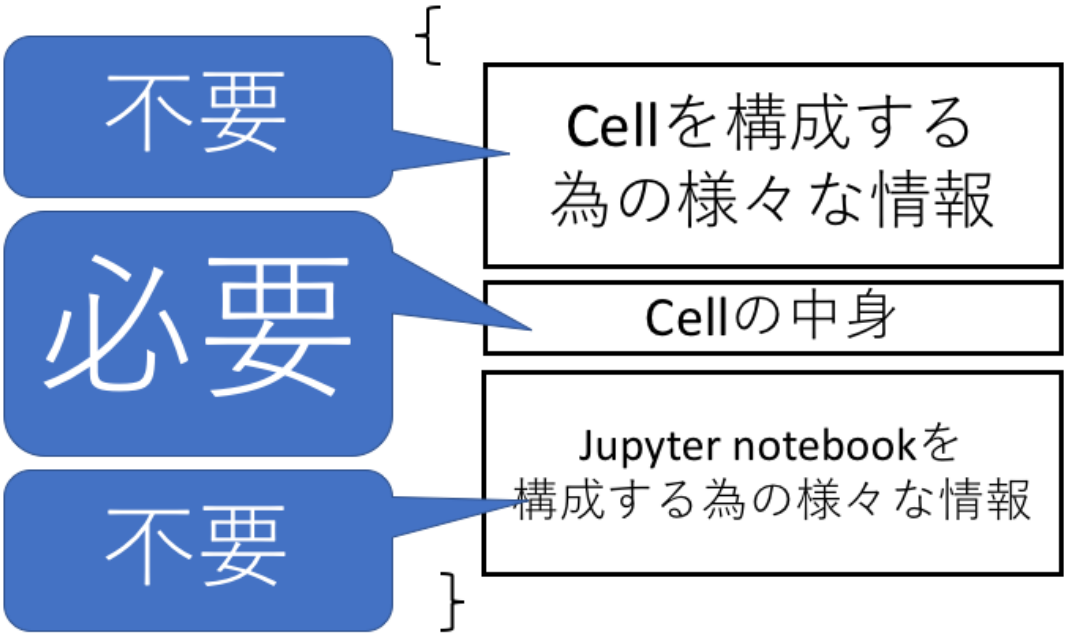

図 3.4: Jupyter notebook のファイル (ipynb) 構成．

上記二点の問題点を解決するコマンドがこのコマンドである．

動作方法

1. コマンドの第一引数に任意の Jupyter notebook ファイルを選択する．
    1. `$ nb_util iputs sample.ipynb`

Jupyter notebook を起動せずにターミナル上で必要な部分のみ表示される．

3.3.4 nb_util ipynb2tex [filename] [option]

Jupyter notebook で作成したものを，卒業論文のフォーマットに対応した LaTeX ファイルに変換する．コマンドの大まかな動作を，図 3.5に示す．

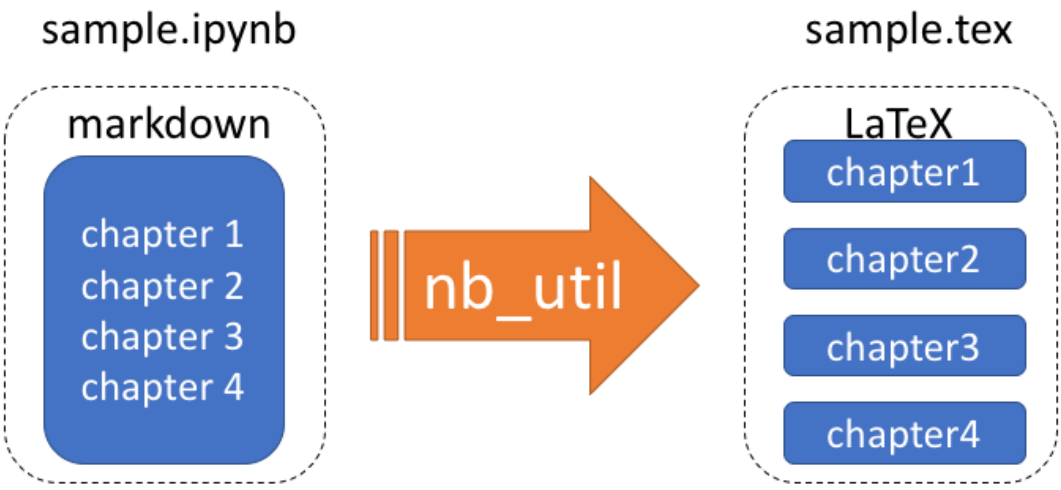

図 3.5: nb_util の ipynb2tex のコマンド解説．

Markdown 形式で書かれた文章を，#(章) ごとにファイルを分割し，さらにそれを LaTeX に変換して「split_files」に保存する．ファイルを分割することにより，文章を書き進めていくうちに文量が膨大になり変更したい場所を探しづらくなるのを防止する為だ．

具体的な変換は，表 3.2をもとに Markdown の命令をそれぞれ対応した LaTeX 命令に出力する．

表 3.1: Markdown と LaTeX の命令対照表 1-1.

項目	Markdown	LaTeX
見出し（章）	# 見出し	¥chapter 見出し
見出し（節）	## 見出し	¥section 見出し
見出し（小節）	### 見出し	¥subsection 見出し
見出し（小々節）	#### 見出し	¥subsubsection 見出し
箇条書き（番号付き）	1. その 1 1. その 2 1. その 3	¥begin{itemize} ¥tightlist ¥item その 1 ¥item その 2 ¥item その 3 ¥end{itemize}
箇条書き（印付き）	* その 1 * その 2 * その 3	¥begin{enumerate} ¥def¥labelenumi{¥arabic{enumi}.} ¥tightlist ¥item その 1 ¥item その 2 ¥item その 3 ¥end{enumerate}
引用	>引用文	¥begin{quote} 引用文 ¥end{quote}
ソースコード	‘ソースコード‘	¥texttt{ ソースコード }

表 3.2: Markdown と LaTeX の命令対照表 1-2.

項目	Markdown	LaTeX
画像	![キャプション．￥label{label 名 }](写真のパス)	￥begin{figure}[H] ￥centering ￥begin{center} ￥includegraphics[width=150mm]{ 写真のパス } ￥end{center} ￥caption{ キャプション．￥label{label 名 }} ￥end{figure}

動作方法

1. コマンドの第一引数に任意の Jupyter notebook ファイルを選択する．
    1. `$ nb_util ipynb2tex sample.ipynb`
2. コマンドライン上に表示される指示に従い内容を入力する．入力された情報は，論文の表紙になり，`./mk_latex/split_files/informations` に保存される．
    1. 論文タイトル
        1. `thesis title: type in your thesis title`
    2. 学籍番号
        1. `student number(eight-digit): 12345678`
    3. 氏名
        1. `your name: 小脇　雅健`
    4. 上記の内容の確認
        1. `Are you ok with it?: y`
3. 書き出されるファイルは，コマンドを実行しているカレントディレクトリに保存される．
4. 変換が正常に終了すると，`./mk_latex/thesis/thesis.tex` と，`./mk_latex` が自動で立ち上がる．

5. Jupyter notebook から，LaTeX に変換する際，1 章毎に分割して `mk_latex/split_files` に chapter1 から順番に保存している．その為，LaTeX で実行する際は `mk_latex/thesis/thesis.tex` をメインとし，そこから各 chapter を読み込んでいる．つまり，PDF 化する際は，`mk_latex/thesis/thesis.tex` を実行する．

ipynb2tex コマンドにより生成されるディレクトリー構成

```
.
├── mk_latex
│   ├── latex
│   │   └── sample.tex
│   ├── split_files
│   │   ├── chapter1
│   │   │   └── chapter1.tex
│   │   ├── chapter2
│   │   │   └── chapter2.tex
│   │   ├── informations
│   │   │   └── informations.tex
│   │   └── tmp
│   │       └── tmp.tex
│   ├── thesis
│   │   ├── thesis.log
│   │   └── thesis.tex
│   └── thesis_pieces
│       ├── form00_style.tex
│       ├── tightlist_setting.tex
│       └── usepackage.tex
└── sample.ipynb
```

さらに，2回目以降実行すると，過去に変換されたファイルを全て実行した時刻単位でバックアップファイルとして，old フォルダに作成するシステムにしている．そのため，変更をしたが過去の内容の方が良い場合など，簡単にアクセスできるつくりにしている．

バックアップが生成されるディレクトリー構成

```
.
├── old
│   └── thesis
│       └── 2018212354
│           └── mk_latex
│               │   └── sample.tex
│               ├── split_files
│               │   ├── chapter1
│               │   │   └── chapter1.tex
│               │   ├── chapter2
│               │   │   └── chapter2.tex
│               │   ├── informations
│               │   │   └── informations.tex
│               │   └── tmp
│               │       └── tmp.tex
│               ├── thesis
│               │   ├── thesis.log
│               │   └── thesis.tex
│               └── thesis_pieces
│                   ├── form00_style.tex
│                   ├── tightlist_setting.tex
│                   └── usepackage.tex
└── sample.ipynb
```

さらに，コマンドの第二引数にオプションが指定できる．

オプション一覧

1. $ nb_util ipynb2tex [filename] -h

このオプションは，handout 用の形式に変換するコマンド.

1. `$ nb_util ipynb2tex [filename] -d`

変換を重ねていくと，バックアップファイルが増え自身の環境を圧迫してしまう．それを解決する為に，ipynb2tex によって生成されたファイルを全て消去するコマンドである.

3.3.5 nb_util yaml2ipynb [input filename]

西谷研で使われている，my_help アプリケーションのコンテンツを，Jupyter notebook に変換する.

コマンドの大まかな動作を，図 3.6に示す.

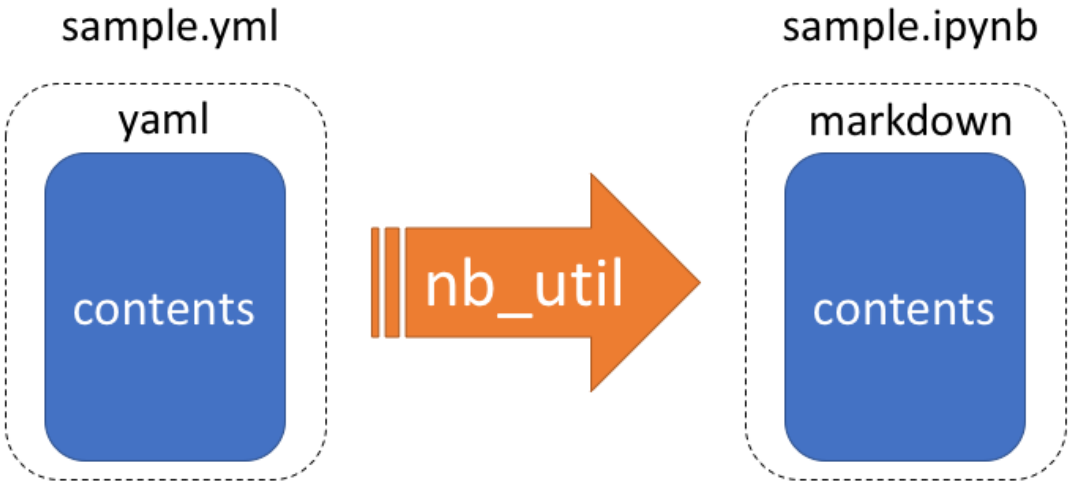

図 3.6: nb_util の yaml2ipynb のコマンド解説.

動作方法

コマンドの第一引数に任意のファイルを選択する．書き出されるファイルは，コマンドを実行しているカレントディレクトリに保存される.

```
$ nb_util yaml2ipynb sample.yml
```

第4章　プログラム解説

本章では，本研究で作成した nb_util のさらなる継続的な発展が可能なように，各コマンドのプログラム解説を記述する．

4.1　Notebook 形式 (.ipynb) の内部構造

Jupyter notebook が生成するファイル形式は，ipynb という拡張子で構築されている．しかし，その中身は json 形式であり，以下の様な内部構造になっている [8].

最上位の要素は以下 4 つで構成されている．

- cells (list)
- metadata (dict)
- nbformat (int)
- nbformat_minor (int)

1. cells には，各 cell の情報が記入されている．
    1. 各 cell で重要な要素は，cell_type, metadata, sources の 3 つである．
        1. cell_type には，各 cell の型，markdown, code, raw の 3 つの型で構成されている．
        2. sources には，cell に入力した内容が，行ごとに配列に格納されている．
2. metadata, nbformat, nbformat_minor は Jupyter を構成する言語情報などが記述されている．

4.1 節の内容を図 2.1 に示す．

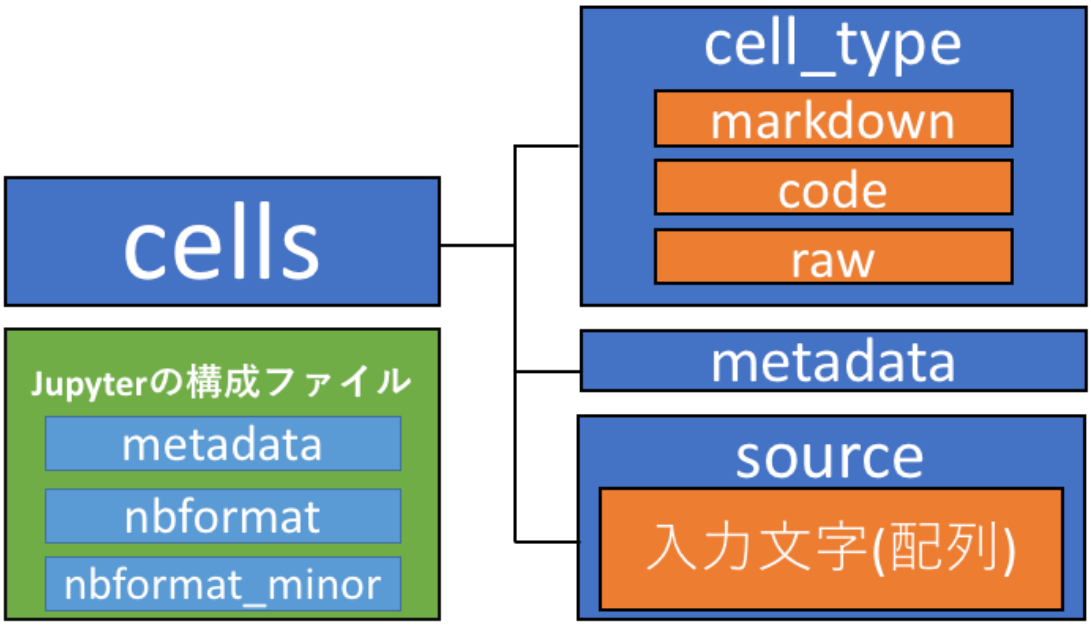

図 4.1: ipynb の内部構造の模式図.

4.2 nb_util combine

Jupyter notebook の二つのファイルを一つに結合するコマンドである．使用方法については，3 章を参照.

4.2.1 ファイルの結合

ipynb のファイルを結合する関数 combine は以下のように実装した.

```
def combine(argv0, argv1, argv2)
  ipynb0 = JSON.load(File.read(ARGV[1]))
  ipynb1 = JSON.load(File.read(ARGV[2]))
  output_filename = ARGV[3]
  p output_filename
  ipynb0["cells"].each do |cell|
    pp cell
    ipynb1["cells"] << cell
  end
  File.open(output_filename + ".ipynb", 'w') do |target|
    JSON.dump(ipynb1,target)
  end
end
```

処理の流れ

変数 ipynb0, ipynb1 はそれぞれ ipynb のファイルを読み込むための変数であり，コマンドラインからの入力ファイル argv0, argv1 に対応している．さらに，output_filename は結合したファイルを出力する変数である．各ファイル（ipynb0, ipynb1）の要素の cells からファイルの中身をそれぞれ読み込み， ipynb1 のファイルの末尾に，ipynb0 のファイルを追記する． その ipynb1 の内容を，output_filename にファイルとして書き込み保存される．

4.3 nb_util getcode

Jupyter notebook で作成したプログラムをファイルとして書き出すコマンドである．使用方法については，3 章を参照.

4.3.1 ソースコードの書き出し

ipynb のファイルから，ソースコードを抽出する関数 getcode は以下のように実装した.

```
def getcode(argv0)
//////ファイル操作のコードは省略//////
  for i in 0..j - 1 do
    eval("if @hash#{i}[\"cell_type\"] != \"code\" then flag = 1 end")
    if flag == 0 then
      eval("puts @getcode = @hash#{i}[\"source\"]")
      source_count = source_count + 1
      output_filename = ipynb_filename + source_count.to_s
      + ipynb["metadata"]["language_info"]["file_extension"]
      File.open(output_filename, 'w+') do |f|
        f.puts(@getcode)
      end
    end
    flag = 0
  end
end
```

処理の流れ

cells の中の cell_type が code の場合だけ，source をファイルに書き出す様に実装している． ファイルの拡張子の判定は，Jupyter notebook を構成するファイルの，metadata の language_info の部分から判定する様にしている．その判定内容をもとに，ソースコードの内容を，output_filename にファイルとして書き込み保存される．

4.4 nb_util ipynb2tex

Jupyter notebook で作成したファイルを，論文の提出形式の LaTeX ファイルに変換するコマンドである． 使用方法については，3 章を参照．

4.4.1 LaTeX ファイルに変換

Markdown形式のipynb ファイルを，論文形式のLaTeX ファイルに変換する関数ipynb2texは以下のように実装した．

```
def ipynb2tex_thesis(target)
 loop do
   target_parent = File.dirname(target)
   exist_info = File.join(target_parent,
   'mk_latex/split_files/informations/informations.tex')
   if File.exist?(exist_info)
     FileUtils.mkdir_p(target_parent + '/split_files/informations')
     FileUtils.cp(exist_info, target_parent)
   else
     your_informations(ARGV[1], "thesis")
   end
   print "Are you ok with it?: "
   input = STDIN.gets.to_s.chomp
   if input == 'Y' || input == 'y'
     location = Open3.capture3("gem environment gemdir")
     versions = Open3.capture3("gem list nb_util")
     latest_version = versions[0].split(",")
     cp_lib_data_thesis_gem = File.join(location[0].chomp,
     "/gems/#{latest_version[0].chomp.gsub(' (','-')
     .gsub(')','')}/lib/data/thesis")
     cp_lib_data_thesis_pieces_gem = File.join(location[0].chomp,
     "/gems/#{latest_version[0].chomp.gsub(' (','-')
     .gsub(')','')}/lib/data/thesis_pieces")
     cp_lib_data_thesis_bundle = File.join(Dir.pwd, '/lib/data/thesis')
     cp_lib_data_thesis_pieces_bundle = File.join(Dir.pwd,
     '/lib/data/thesis_pieces')
     re_fig = /(.+\.jpg)|(.+\.jpeg)|(.+\.png)/
```

```
print "\e[32minputfile: \e[0m"
target = File.expand_path(ARGV[1])
print "\e[32m#{target}\n\e[0m"
print "\e[32moutputfile: \e[0m"
tex_src = target.sub('.ipynb', '.tex')
print "\e[32m#{tex_src}\n\e[0m"
target_parent = File.dirname(target)
target_basename = File.basename(tex_src)
Open3.capture3("jupyter nbconvert --to latex #{target}")
lines = File.readlines(tex_src)
lines.each_with_index do |line, i|
  line.sub!("\documentclass[11pt]{article}",
    "\documentclass[11pt,dvipdfmx]{jsarticle}")
  print "\e[32m#{line}\n\e[0m" if line =~ re_fig
  line.sub!(line, '%' + line) if line.include?('.svg')
end
File.open(tex_src, 'w') { |file| file.print lines.join }
FileUtils.mkdir_p(target_parent + '/latex')
FileUtils.mv(tex_src, target_parent + '/latex')
replace_figs(File.join(target_parent + '/latex',
target_basename), "thesis")
revise_lines(File.join(target_parent + '/latex', target_basename))
split_files(File.join(target_parent + '/latex', target_basename),
 target, "thesis")
FileUtils.mv(target_parent + '/tmp.tex',
 target_parent +'/split_files/tmp')
FileUtils.mv(target_parent + '/informations.tex',
target_parent + '/split_files/informations')
mk_thesis_location(target, "thesis")
FileUtils.mv(target_parent + '/.splits_location.tex',
 target_parent + '/thesis')
mk_xbb(target, re_fig)
```

```
      if Dir.exist?(cp_lib_data_thesis_pieces_bundle.to_s) &&
      Dir.exist?(cp_lib_data_thesis_bundle.to_s)
        FileUtils.cp_r(cp_lib_data_thesis_pieces_bundle, target_parent)
        FileUtils.cp_r(cp_lib_data_thesis_bundle, target_parent)
      else
        FileUtils.cp_r(cp_lib_data_thesis_pieces_gem, target_parent)
        FileUtils.cp_r(cp_lib_data_thesis_gem, target_parent)
      end
      mk_latex_and_mv_to_latex(target, target_parent, "thesis")
      Open3.capture3("open #{target_parent}")
      Open3.capture3("open #{target_parent}/mk_latex/thesis/thesis.tex/")
      exit
      break
    elsif input == 'N' || input == 'n'
      target_parent = File.dirname(target)
      FileUtils.rm_r(File.join(target_parent.to_s, '/informations.tex'))
      break
    end
  end
end
```

処理の流れ

1. 表紙の作成

論文の表紙を作成する関数，your_informations 関数が実行される．

```
def your_informations(target, thesis_or_handout)
  info = Array.new(3)
  print "thesis title: "
  info[0] = STDIN.gets.to_s.chomp.gsub(/_/,'\_')
  print "student number(eight-digit): "
  info[1] = STDIN.gets.to_s.chomp.gsub(/_/,'\_')
  print "your name: "
  info[2] = STDIN.gets.to_s.chomp.gsub(/_/,'\_')
  target_parent = File.dirname(target)
  d = Date.today
  thesis_infomations = <<"EOS"
//////省略//////
EOS
  handout_infomations = <<"EOS"
//////省略//////
EOS
  FileUtils.mkdir_p(target_parent + '/split_files/informations')
  File.open(target_parent + '/informations.tex', "w") do |f|
    if thesis_or_handout == "thesis"
      f.print(thesis_infomations)
    end
    if thesis_or_handout == "handout"
      f.print(handout_infomations)
    end
  end
end
```

info 配列を用意し，info[0] に論文のタイトル，info[1] に学生番号，info[2] に氏名を格納する．その取得した情報をもとに，LaTeX 形式に，個々の情報を格納する．さらに，handout 用，thesis 用を判別し，ipynb2tex_thesis 関数又は，ipynb2tex_handout 関数に

渡す.

2. コマンドの識別

1. 実行されているコマンドが，bundle または gem からインストールされたのかを判断する.

分割された LaTeX を紐付けて実行するメインの mk_latex/thesis/thesis.tex と LaTeX で使用するパッケージ，スタイルなどを構成する mk_latex/thesis_pieces/ を使用者各々が実行したカレントディレクトリに作成する為だ．この二つのファイルは，インストール方法によって，ファイルが存在するアドレスが違うため，gem でインストールした場合，bundle を使用した場合を以下のプログラムで解析している.

```
location = Open3.capture3("gem environment gemdir")
versions = Open3.capture3("gem list nb_util")
latest_version = versions[0].split(",")
cp_lib_data_thesis_gem = File.join(location[0].chomp,
                      "/gems/#{latest_version[0]
                      .chomp.gsub(' (','-').gsub(')','')}/lib/data/thesis")
cp_lib_data_thesis_pieces_gem = File.join(location[0].chomp,
                           "/gems/#{latest_version[0]
               .chomp.gsub(' (','-').gsub(')','')}/lib/data/thesis_pieces")
cp_lib_data_thesis_bundle = File.join(Dir.pwd, '/lib/data/thesis')
cp_lib_data_thesis_pieces_bundle =
                  File.join(Dir.pwd, '/lib/data/thesis_pieces')

if Dir.exist?(cp_lib_data_thesis_pieces_bundle.to_s) &&
Dir.exist?(cp_lib_data_thesis_bundle.to_s)
  FileUtils.cp_r(cp_lib_data_thesis_pieces_bundle, target_parent)
  FileUtils.cp_r(cp_lib_data_thesis_bundle, target_parent)
else
  FileUtils.cp_r(cp_lib_data_thesis_pieces_gem, target_parent)
  FileUtils.cp_r(cp_lib_data_thesis_gem, target_parent)
end
```

3. 成形する

1. チャプター毎に分割し，論文形式に成形する．

Jupyter notebook で作成したファイルを，今後の編集を考慮し章 (#) 毎に分割していく．さらに，Jupyter notebook で用意されているツールで変換した場合 chapter を section として扱われてしまうため，chapter として正しく扱われるように処理を施している．そのほかについても，一つ下の階層を示す命令に変換されるため，同様の処理を行う．

```
 def split_files(target, input_ipynb, thesis_or_handout)
   target_parent = File.absolute_path("../..", target)
   ipynb = JSON.parse(File.read(input_ipynb))
   pickup_ipynb = ipynb["cells"].to_s.split(",")
   chapter = pickup_ipynb.grep(/"# /)
   .map{ |i| i.gsub(/.*# /, '').gsub(/".*/, '').gsub(/\\n/, '') }
   if thesis_or_handout == "thesis"
     chapter_size = chapter.size
     p chapter
     for num in 0..chapter_size-1 do
      splitters = [ ["\\section{#{chapter[num]}}"
      , target_parent + "/chapter#{num+1}.tex",
      FileUtils.mkdir_p(target_parent + "/split_files/chapter#{num+1}")],
        ["\\begin{Verbatim}", target_parent + '/tmp.tex',
         FileUtils.mkdir_p(target_parent + '/split_files/tmp')]]
      cont = File.read(target)
      splitters.reverse.each do |splitter|
        split = cont.split(splitter[0])
        split[1].to_s.gsub!(/subsection/, 'section')
        split[1].to_s.gsub!(/subsubsection/, 'subsection')
        split[1].to_s.gsub!(/paragraph/, 'subsubsection')
        cont = split[0]
//////ファイル操作のコードは省略//////
  end
end
```

4. tex へ変換

1. Jupyter notebook に備わっている，`jupyter nbconvert --to latex` コマンドをシステム内で実行し，tex に変換する．
2. tex に変換されたファイルをファイルの末尾から先頭にかけて読み込む．

変換されたファイルの先頭部分は，ファイルに必要な情報，構成ファイル，パッケージが数百行に渡り記述されている．しかし，ファイル末尾は，構成ファイル等内容に関係ないものは記述されてない．その為，tex ファイルを読み込む．

4.4.2 生成されたファイルを消去

ipynb2tex コマンドで生成された，mk_latex, old フォルダを完全に消去する．

処理の流れ

完全消去コマンドなので，謝って消去をしてしまわない様に消去の際，確認の処理を用いている．作成したフォルダが存在するか否かを`Dir.exist?`関数で判断し，mk_latex または，old ファイルが存在すれば消去する．

```
def delete_folder(target)
  loop do
    print "Are you sure?: "
    input = STDIN.gets.to_s.chomp
    if input == 'Y' || input == 'y'
      mk_latex = File.join(File.absolute_path("../", target), '/mk_latex')
      old = File.join(File.absolute_path("../", target), '/old')
      if Dir.exist?(mk_latex) && Dir.exist?(old)
        print "\e[31mdelete: \e[0m"
        FileUtils.rm_r(mk_latex)
        puts "\e[31m#{mk_latex}\e[0m"
        print "\e[31mdelete: \e[0m"
        FileUtils.rm_r(old)
        puts "\e[31m#{old}\e[0m"
        break
      elsif Dir.exist?(mk_latex)
        print "\e[31mdelete: \e[0m"
        FileUtils.rm_r(mk_latex)
        puts "\e[31m#{mk_latex}\e[0m"
        break
      elsif Dir.exist?(old)
        print "\e[31mdelete: \e[0m"
        FileUtils.rm_r(old)
        puts "\e[31m#{old}\e[0m"
        break
      else
        puts "\e[31mNo such file or directory\e[0m"
        break
      end
    elsif input == 'N' || input == 'n'
      puts
      break
    end
  end
end
```

4.5 nb_util yaml2ipynb

4.5.1 jupyter botebook へ変換

my_help(本研究室西谷が開発したメモソフト) の yaml ファイルから，ipynb に変換する関数 yaml2ipynb は以下のように実装した．使用方法については，3 章を参照.

```
def yaml2ipynb(argv0)
//////ファイル操作のコードは省略//////
   contsinglehash = flatten_hash_from cont
   contsinglehashneed = contsinglehash.select do
    |k, v| k.match(/title/) || k.match(/cont/)
    end
   pp contsinglehashneed
   contdiffstring = contsinglehashneed.to_s
   contdiffstring.gsub!(/=>/,'')
     str = contsinglehashneed.keys
     contnum=str.count
     for i in 0..contnum do
       ignore = str[i].to_s
       p ignore
       contdiffstring.sub!(/#{ignore}/,'')
   end
   contdiffstring.gsub!(/:\"\"\"/,'# ').gsub!(/\[/,'').gsub!(/\]/,'')
   contdiffstring.gsub!(/{/,'').gsub!(/}/,'').gsub!(/}/,'')
   contdiffstring.gsub!(/\", :\"\"\"/,'\n').gsub!(/\"/,'').gsub!(/,/,'\n')
   #cell のデータ
//////cell の構成データは省略//////
   File.open(output_filename, 'w+') do |f|
    f.print(meta)
  end
end
```

```
def flatten_hash_from hash
  hash.each_with_object({}) do |(key, value), memo|
    next flatten_hash_from(value).each do |k, v|
      memo["#{key}.#{k}".intern] = v
    end if value.is_a? Hash
    memo[key] = value
  end
end
```

処理の流れ

my_help の構造は，title にメモの題目，cont に内容が記述されるという仕組みになっている為， title, cont の中身を抽出する． しかし，多次元配列になっており，扱うのが煩わしい為，一次元配列に変換する，flatten_hash_from という関数を実装した．この関数で配列を一次元配列にし，中身（title, cont）を抽出し，gsub 関数を使用し置換をする．成形されたファイルの内容を，output_filename にファイルとして書き込み保存される．

第5章　総括

本研究では，Jupyter notebook を使用する際，見受けられた数多くの不便を解消するアプリケーション「nb_util」を作成した．このアプリケーションによる成果を以下に記す．

1. 作成したファイルを，Jupyter notebook を起動することなく terminal 上で複数のファイルを一つにまとめることが可能になった．

2. 作成したプログラムファイルをソースコードファイルとして保存する際，Jupyter notebook を起動してコピーアンドペーストせずに，簡単に terminal からファイル化をすることが可能になった．

3. terminal 上での Jupyter notebook のファイルの可視化を行い，Jupyter notebook を起動せずに中身を確認することが可能になった． Linux に標準搭載のコマンド`$ cat` とは違い，ファイルの必要な内容部分のみ表示することが可能である．

4. 作成したファイルを，論文形式の LaTeX に変換することが可能になり，LaTeX の煩わしいルールを考える必要がなくなり，直感的に書くことが可能な Markdown 形式で文章を書くことができ，学習意欲，効率の向上に繋がると考えられる．

5. 作成したアプリケーションを「nb_util」という名前のライブラリとして保存しプログラムの解説を行うことで，プログラムの継続的な発展を可能にした．また，今後文章を作成する際に今回のプログラムをライブラリとして利用し効率よく作成する事が可能である．

謝辞

本研究を行うにあたり，終始多大なるご指導，御鞭撻をいただいた西谷滋人教授に対し，深く御礼申し上げます．また，本研究の進行に伴い，様々な助力，知識の供給を頂きました西谷研究室の同輩，先輩方に心から感謝の意を示します．本当にありがとうございました．

参考文献

[1] Sarah Guido, Andreas Mueller (2016), ”Introduction to Machine Learning with Python”. (中田 秀基 (訳)「Python ではじめる機械学習」(オライリージャパン, 2017)).

[2] Cyrille Rossant (2014), ”IPython Interactive Computing and Visualization Cookbook”. (菊池 彰 (訳)「IPython データサイエンスクックブック」(オライリージャパン, 2015)).

[3] 平鍋 健児, 野中 郁次郎, 「アジャイル開発とスクラム」(翔泳社, 2013).

[4] Ward Cunningham, ”Manifesto for Agile Software Development”, `http://agilemanifesto.org`, accessed February 5, 2018.

[5] 池内孝啓, 片柳薫子, 岩尾 エマ はるか,@driller, 「Python ユーザのための Jupyter［実践］入門」(技術評論社,2017).

[6] すがわらまさのり, 寺田玄太郎, 三村益隆, 近藤宇智朗, 橋立友宏, 関口亮一, 「パーフェクト Ruby」(技術評論社,2013).

[7] ”Pandoc a universal document converter”, `http://pandoc.org/index.html`, accessed February 1, 2018.

[8] Jupyter Development Team, ”The Jupyter Notebook Format”, `https://nbformat.readthedocs.io/en/latest/format_description.html`, accessed February 5, 2018.