第一回 研究プログラミング WINGS-ABC チュートリアル

登壇者：片岡麻輝、関森祐樹、堀口修平

2021年10月21日（木）17：00-19：00

登壇者紹介

片岡麻輝　総合文化研究科広域科学専攻、修士一年。WINGS-ABC三期生。大泉研究室にて、計算神経科学および理論神経科学の研究を行っている。現在はシミュレーションベースでの研究テーマに取り組んでいる。株式会社ACESで機械学習アルゴリズムの開発にも従事。

関森祐樹　海洋技術環境学専攻、修士二年。WINGS-ABC二期生。生産技術研究所2部、巻研究室で、複数の海中ロボットシステムの研究をしている。複数の海中ロボットシステムを可能とする、基礎的なシステム設計や情報処理アルゴリズムを検証するために、ソフトウエアを作成している。株式会社FullDepthで水中ロボットの制御研究開発にも従事。

堀口修平　情報理工学系研究科数理情報学専攻、博士一年。WINGS-ABC一期生。生産技術研究所の小林徹也研究室で、免疫系の理論研究をしている。数理モデルのシミュレーションのほか、Webアプリ・ロボット・生物画像処理や機械学習アルゴリズムの開発でプログラミングを行う。

研究プログラミングチュートリアル　概要

研究現場のプログラミングで起こりうる問題をスキット（寸劇）形式で紹介し、それぞれについて選択問題と解説を行う形式で進行します。扱うテーマは以下の通りです。最後に、皆さんと研究プログラミングについて一緒に考える、ディスカッションの時間を設けてます。

第一回（情報収集と共有）

- コードの共有
- 環境構築
- パラメータ管理
- バージョン管理

第二回（信頼性と再現性の担保）

- コードの読みやすさ
- Linterと型チェック
- テスト
- 再現性

用語集

コード:コンピュータに解釈や実行させること目的として命令やデータ

プログラム:コンピュータへの命令や処理が記載されたもの

スクリプト:ソースコードで即座に実行できるもの

ソフトウエア:コンピュータの物理的な部分(ハード)の総称

アプリケーションソフトウエア(アプリケーション):オペレーティングシステム(OS)上で動作するソフトウエア

リポジトリ：アプリケーション開発で、システムを構成するデータやプログラムの情報を収納したデータベース

ライブラリ：ある特定の昨日を持ったプログラムを他のプログラムから引用しできるように部品化し、それらを集めてファイルに収納したもの

モジュール：ある機能を実現する、ひとまとまりのプログラム機能や要素

クラス：オブジェクトを生成するための設計図や雛形

シンタックス：プログラミング言語の構文や文法

ブレークポイント：デバッグ作業でプログラムを意図的に一時停止させる箇所

フレームワーク：アプリケーションなどの実装に必要な機能や定型コードをライブラリとして事前に用意したもの

プラットフォーム：アプリケーションでは、ミドルウエア、ライブラリ、言語処理系（ランタイム）等のこと

バージョン：同名プログラムの新旧を区別する、番号や符号

参考文献

Boswell, D., Foucher, T., & Kado, M. (2012). リーダブルコード より良いコードを書くためのシンプルで実践的なテクニック. (角 征典, Trans.). オライリージャパン.

Cormen, T. H., & Leiserson, C. E. (2009). Introduction to algorithms (3rd ed.). The MIT Press.

Dutta, S., Legunsen, O., Huang, Z., & Misailovic, S. (2018). Testing probabilistic programming systems. Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. https://doi.org/10.1145/3236024.3236057

【git】研究室メンバに向けた git / gitlab の説明資料. 溶けかけてるうさぎ - BLOG. (n.d.). Retrieved November 13, 2021, from https://meltingrabbit.com/blog/article/2020052701/.

Gorelick, M., & Ozsvald, I. (2015). ハイパフォーマンスPython. (相川 愛三, Trans.). オライリージャパン.

Hiuchi. (2018). *生命科学研究のための GitHubの使い方 基本編*. togotv. Retrieved November 13, 2021, from https://togotv.dbcls.jp/en/20180621.html.

Lubanovic, B., Suzuki, H., & Nagao, T. (2015). 入門 Python 3. オライリージャパン .

Microsoft. (2016, April 14). Developing inside a container using Visual studio code remote development. RSS. Retrieved October 19, 2021, from https://code.visualstudio.com/docs/remote/containers.

Mukai, N. (2016). *Githubの基本的な使い方*. mLAB. Retrieved November 13, 2021, from https://mukai-lab.info/pages/tech/github/github-usage/.

Ohkouchi, K. (2020). ログ設計指針. Qiita. Retrieved November 13, 2021, from https://qiita.com/nanasess/items/350e59b29cceb2f122b3.

Pressman, R. S. (2010). Software engineering: A practitioner's approach (7th ed.). McGraw-Hill Higher Education.

Scott, C., & Straub, B. (2014). *Pro Git book*. Git. Retrieved November 13, 2021, from https://git-scm.com/book/ja/v2.

SHIFT. コンポーネント（単体、ユニット）テストとは？手法などを例で紹介 . *ソフトウェアテストの SHIFT*. Retrieved October 20, 2021, from https://service.shiftinc.jp/column/3636/.

Software Freedom Conservancy. (n.d.). git. Git. Retrieved October 19, 2021, from https://git-scm.com/.

van Rossum, G., Warsaw, B., & Coghlan, N. (2014). Python コードのスタイルガイド. pep8-ja. Retrieved October 19, 2021, from https://pep8-ja.readthedocs.io/ja/latest/.

呉 珍詰. (2020). 初心者のためのコンテナ入門教室 Think IT（シンクイット）. Retrieved October 19, 2021, from https://thinkit.co.jp/series/9490.

湊川 あい, & DQNEO. (2017). わかばちゃんと学ぶ Git使い方入門. シーアンドアール研究所.

金子 邦彦. (2021). バージョン管理, プロジェクト管理ソフトウエアのインストール . Retrieved November 13, 2021, from https://www.kkaneko.jp/tools/redmine/index.html.

河野 晋策 . あなたの生産性を向上させる Jupyter Notebook Tips. リクルート メンバーズブログ . Retrieved October 20, 2021, from https://blog.recruit.co.jp/rtc/2018/10/16/jupyter_notebook_tips/.

杉本靖博,. (2017). Gitを用いたバージョン管理のすすめ ., システム/制御/情報、Vol. 61, No. 10, pp.394-399, from https://www.jstage.jst.go.jp/article/isciesci/61/10/61_394/_pdf

長野 透. (2021, August 30).【python入門】loggingモジュールで処理の記録を残してみよう！ . 侍エンジニアブログ. Retrieved November 13, 2021, from https://www.sejuku.net/blog/23149.

チュートリアル用Github

WINGS-ABC-programming-tutorial/

- mlflow-demo
- sample_unittest

第一回
研究用ソフトの情報収集と共有

登場人物

片岡：主人公のM1大学院生

堀口：先輩

関森：教授

（この物語はフィクションです。実際の人物・団体とは一切関係ありません。）

スキット1:「コードの共有」

あらすじ

M1片岡は、神経科学理論に関する研究室に配属された。教授とのディスカッションの末、修士研究テーマは、堀口先輩の研究してきたモデルを改良して、ダイナミカルな特性を同定する研究に決まった。片岡は、先輩のモデルを使って、早速シミュレーションの実装を始めたい。

（この物語はフィクションです。実際の人物・団体とは一切関係ありません。）

問1:「コードの共有」

研究課題でモデリングのプログラムをつくる必要があります。まずどの行動を取るとよいでしょうか。

1. 自力でモデルを実装する。
2. 研究分野内で同様なモデルのプログラムが無いか探す。
3. 研究室に同様なモデルのプログラムが無いか聞く。

解答と解説1:「コードの共有」

1. 自力でモデルを実装する
2. 研究分野内で同様なモデルのプログラムが無いか探す
3. 研究室に同様なモデルのプログラムが無いか聞く

まずは、研究室で作成された既存プログラムの活用をお勧めします。なぜなら、既存のプログラムは、作成者によってある程度検証されているからです。

2は他の研究者に聞いて回ってから、それぞれの人が参考にしたものを参考にすると効率が良いです。

1のように、初めから自力で実装しないでください。間違いの元です。もし計算式などが誤っていても、見落としてしまうことがよくあります。他人の協力なしで一人で取り組むのは危険です。但し、勉強のために自力で実装することは良いかもしれません。

（この物語はフィクションです。実際の人物・団体とは一切関係ありません。）

スキット2:「環境構築」

あらすじ

堀口先輩のシミュレーションプログラムを受け取った片岡は、これを試しに実行しようとするが、エラーメッセージばかり出てきてしまう。片岡は、自分のパソコンに関連ライブラリを揃える必要があるが、これは一筋縄では無いようであることに気づく。

（この物語はフィクションです。実際の人物・団体とは一切関係ありません。）

問2:「環境構築」

どうすれば同じ環境を再現できるでしょうか。

1. 同じPCを使う
2. どんなソフトウェアをインストールしたか、バージョンを含めて全て記録し、その記録に従う
3. HDDやSSDの中身を丸ごとバックアップしておき、そのバックアップからPCを起動する

解答と解説2:「環境構築」

1. 同じPCを使う
2. どんなソフトウェアをインストールしたか、バージョンを含めて全て記録し、その記録に従う
3. HDDやSSDの中身を丸ごとバックアップしておき、そのバックアップから PCを起動する

1は同じPCを使っていても、別のプロジェクトのためにソフトウェアを新しくインストールするなどして上手く動いていた当時の環境とは異なるでしょう。

3はやっても良いが、バックアップの容量を多く消費する上に、同じ環境を再現するときにPC全体の中身を取り替えなければならないので手間がかかります。

2のように、ソフトウェアのバージョンも含めて記録しておくと、必要十分なことが多いです。requirements.txtやDockerなどの仮想環境を使うと更に便利になります。

requirements.txt

どのパッケージ(ライブラリ)を使っているかを記述しておくファイル。

pipコマンドで簡単にインストールできる

requirements.txtからインストール

```
$ pip install -r requirements.txt
```

今の環境からrequirements.txtを作成

```
$ pip freeze > requirements.txt
```

例:

```
1  Babel==2.5.3
2  docopt==0.6.2
3  ipywidgets==7.4.2
4  joblib==0.11
5  nibabel==2.3.1
6  numpy==1.14.5
7  pandas==0.22.0
8  scikit-image==0.13.1
9  scikit-learn==0.20.0
10 scipy==1.0.0
11 seaborn==0.9.0
12 tifffile==2018.10.18
```

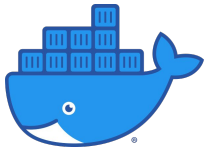
仮想環境 poetry, pyenv

プロジェクトごとに仮想環境をつくり、別々のrequirements.txtを対応させると良い。

例えば、この記事（Qiita Poetry - Pipenv の代替 by propella）の方法では

- pyenvでpython自体のバージョンを選択
- poetryでプロジェクトごとにライブラリのバージョンを管理

Docker



Pythonのパッケージ以外にインストールしたソフトウェアなども含めてOS丸ごと再現する方法もある。特にDockerは高速で非常に使い勝手が良い

Visual Studio CodeのDevcontainerという機能を使うと、非常に簡単にできる。

くわしくは:

- Think IT 初心者のためのコンテナ入門教室
- Visual Studio Code Developing inside a Container

ちなみに、Google Colaboratoryも仮想環境を使っている

スキット3:「パラメータ管理」

あらすじ

片岡は、やっとのことでシミュレーション環境を整えられたので、遂に研究に取り掛かる。ただ、まだ理論の知識が薄く、シミュレーション改善の方針が立てられない。そこで、まずは堀口先輩の研究で着目されて無かったパラメータを変数として、モデル検証を始めようとする。

（この物語はフィクションです。実際の人物・団体とは一切関係ありません。）

問3:「パラメータ管理」

パラメータを変えながら何回もシミュレーションを行って、結果がどう変わるかを調べたいとします。どのように結果を出力させればよいでしょうか。

1. パラメータごとの結果をテキストファイルに出力し、最後に結果をまとめてグラフを作成する
2. 全てのシミュレーションを一度に実行して、結果をテキストファイルにまとめて出力し、その後、グラフを作成する
3. 全てのシミュレーションを一度に実行して、まとめた結果をグラフとして出力する

解答と解説3:「パラメータ管理」

1. パラメータごとの結果をテキストファイルに出力し、最後に結果をまとめてグラフを作成する
2. 全てのシミュレーションを一度に実行して、結果をテキストファイルにまとめて出力し、その後、グラフを作成する
3. 全てのシミュレーションを一度に実行して、まとめた結果をグラフとして出力する

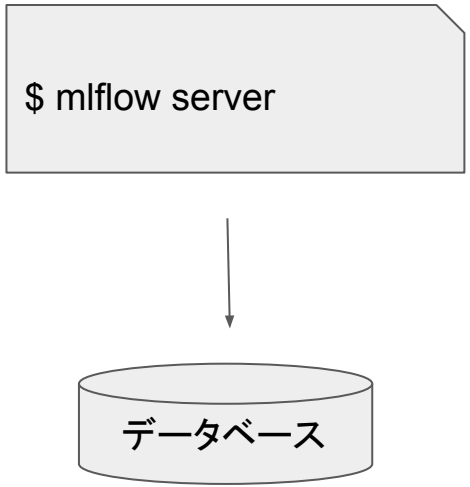
2や**3**の場合、例えばある条件のときだけ生じるバグを見つけて修正したとします。このとき全てのパラメータについてのシミュレーションをし直さないと結果のグラフを得られませんが、修正した条件以外の場合はすでに全く同じ計算をしていたはずなので、以前の計算時間が無駄になってしまいます。

さらに**3**では、結果がグラフしかないので、グラフの細かい見た目を調整するためには全てのシミュレーションをもう一度実行する必要があります。

mlflow Tracking

膨大なパラメータやシミュレーション結果を管理するのは大変です。

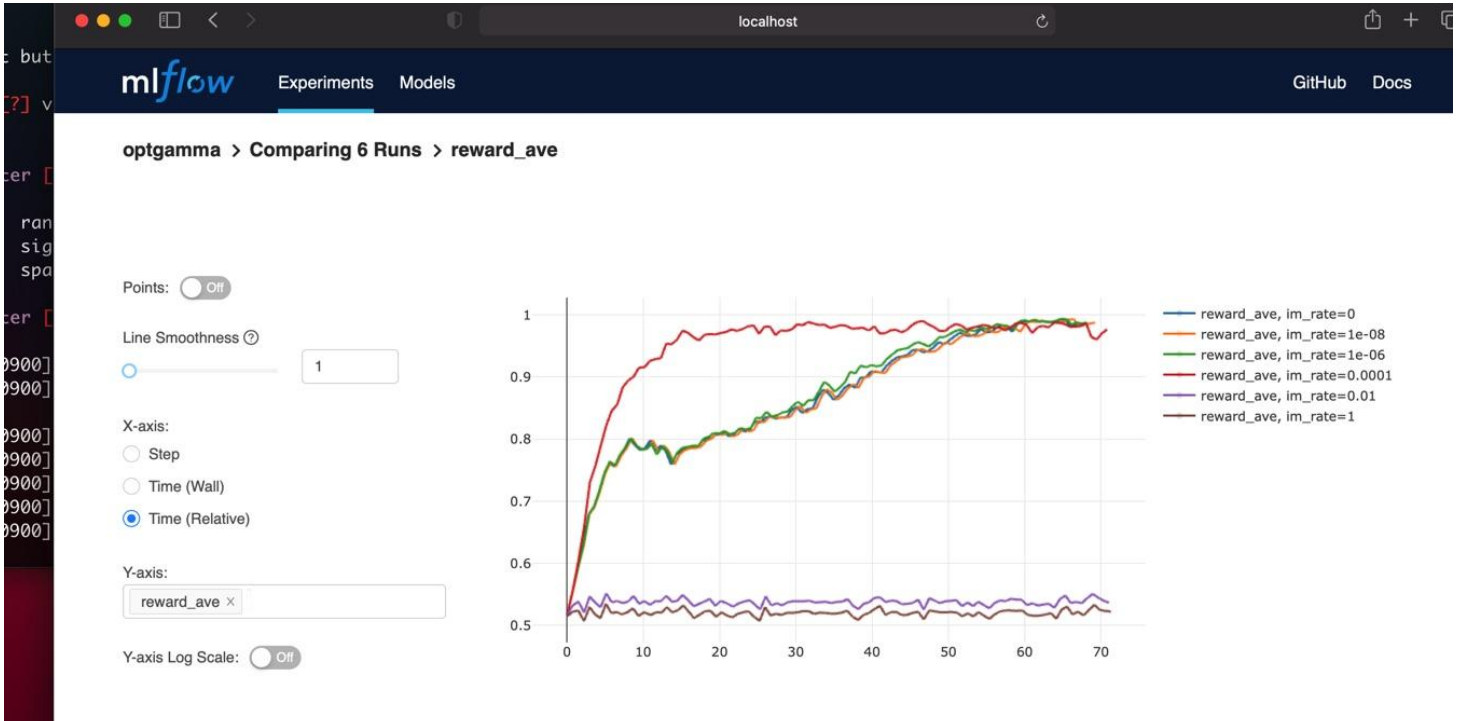
mlflow Trackingはそれらを管理するための仕組みを提供しています。




```
with mlflow.start_run():
    mlflow.log_params({
        “param1”:param1,
        “param2”:param2,
    })
    # なにか計算をする
    quality = ....
    mlflow.log_metrics({
        "quality":quality,
    })
```

mlflow Tracking 可視化

Webブラウザを用いた簡単な可視化用のインターフェースも提供



スキット4:「バージョン管理」

あらすじ

片岡は、パラメータを変えていく過程で、理論とシミュレーションの知見が得られてきた。ここからは、シミュレーションを使って、モデルの改善を進めていくことになる。将来的には、後輩たちがこの一連の研究テーマを更に進めたり、派生させた別の研究に繋げられるようにしたい、と教授は話されている。つまり、自分のモデルを実装したシミュレーションを研究室内で共有していくことになる。研究グループでプログラムをどのように管理すれば良いだろうか、と片岡は考える。

（この物語はフィクションです。実際の人物・団体とは一切関係ありません。）

問4:「バージョン管理」

一つの研究プロジェクトに複数人が関わっているので、研究室内で共通プログラムを実装しています。正しい管理方法はどれでしょうか。

1. 安定した最新版プログラムのみ共有し、開発中のものは自己管理する
2. まだ動作確認していない作業中のプログラム以外は共有し、これまで作成した安定版プログラムの記録も残す
3. 開発作業中のプログラムを含め、今まで作成したすべてのプログラムを共有する

解答と解説4:「バージョン管理」

1. 安定した最新版プログラムのみ共有し、開発中のものは自己管理する
2. まだ動作確認していない作業中のプログラム以外は共有し、これまで作成した安定版プログラムの記録も残す
3. 開発作業中のプログラムを含め、今まで作成したすべてのプログラムを共有する

1はうまくいきません。研究室で作成するソフトのように、頻繁に3更新するものは、どうしても複数のバージョンができてしまいます。

2過去のプログラムは古いバージョンのOSを使っている人々に親切です。新しいプログラムには後方互換性が無い場合があります。

3プログラムのバージョン管理は大事ですが、書き途中のプログラムを共有しても作成者以外は恐らく使えません。

Git

Gitは分散バージョン管理をする無料でオープンソースなツールです。

安定版プログラム、自分が編集しているプログラム、他の人が編集したプログラムなどを切り替えたり、取り入れたり、組み合わせたりできます。更に、過去に作成したバージョンにも簡単に遡れたり、取り組み中のプログラムを一時的に残したりもできます。ブランチ機能を使って、他のプロジェクトメンバーと作業の割り振り、バグの対応等も、整理整頓しやすいです。

是非利用することをお勧めします！

Gitウェブサイト　https://git-scm.com/

Scott, C., & Straub, B. (2014). *Pro Git book*. Git. Retrieved November 13, 2021, from https://git-scm.com/book/ja/v2.

湊川 あい, & DQNEO. わかばちゃんと学ぶ Git使い方入門

他にもたくさんネット資料もあります！

【補足】研究室でGitを使う

- 以下のような研究場面でGitが有用
  - シミュレーション・実験・解析のソースコード
  - LaTEX論文
- 個人作業にもバージョン管理が有効！
- 研究でのGitの使いかた
  - 杉本靖博,. (2017). Gitを用いたバージョン管理のすすめ., システム/制御/情報、Vol. 61, No. 10, pp.394-399, from https://www.jstage.jst.go.jp/article/isciesci/61/10/61_394/_pdf
  - *【git】研究室メンバに向けた git / gitlab の説明資料*. 溶けかけてるうさぎ - BLOG. (n.d.). Retrieved November 13, 2021, from https://meltingrabbit.com/blog/article/2020052701/.
  - Hiuchi. (2018). *生命科学研究のためのGitHubの使い方 基本編*. togotv. Retrieved November 13, 2021, from https://togotv.dbcls.jp/en/20180621.html.
- ユーザインターフェースの使いかた
  - Mukai, N. (2016). *Githubの基本的な使い方*. mLAB. Retrieved November 13, 2021, from https://mukai-lab.info/pages/tech/github/github-usage/.
  - 金子 邦彦. (2021). バージョン管理，プロジェクト管理ソフトウエアのインストール . Retrieved November 13, 2021, from https://www.kkaneko.jp/tools/redmine/index.html.
- Git 以外にもMercurialやSubversionなどのソフトもある

ディスカッション

今までの研究プログラムに関する情報収集や共有方法を振り返り、チュートリアルで紹介された知識/テクニックと照らし合わせて、改善できそうなポイントを考えてみましょう。

第一回のまとめ

1. コードの共有
   - まずは、研究室内で同様のプログラムを探し、できる限り再利用する
   - 自分でプログラムを書くのはリスクが伴うので、最後の手段に残す
2. 環境構築
   - ソフトウェアのバージョンも含めて関連ライブラリ情報を記録する
   - できれば、requirements.txtやDockerなどの便利な仮想環境を使う
3. パラメータ管理
   - パラメータごとの結果をテキストファイルに出力する
   - 最後に結果をまとめてグラフを作成する
4. バージョン管理
   - 動作確認済みのプログラムは適宜共有し、過去のバージョンもバックアップする
   - 研究室全体でGitを活用できると尚よい

予告 第二回 研究プログラミング チュートリアル

日時：2021年10月23日（土）17：00-19：00

テーマ：信頼性と再現性の担保

- コードの読みやすさ
- Linterと型チェック
- テスト
- 再現性

実践的なプログラミングテクニックを例題と伴に紹介します。

皆さんご参加を！