

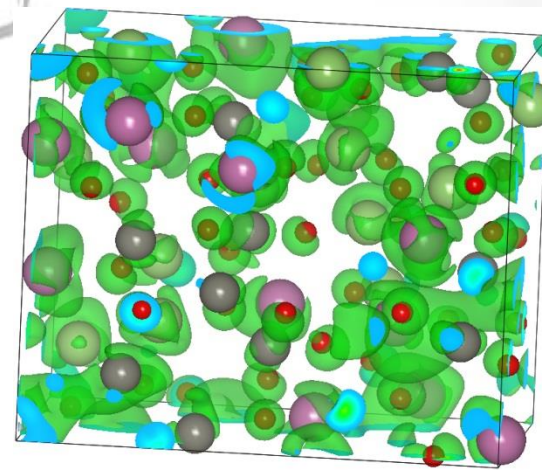
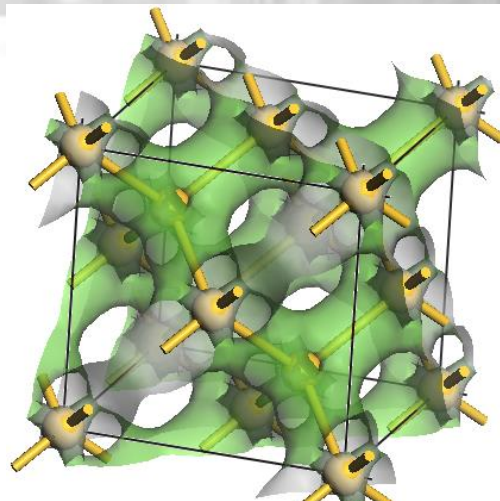
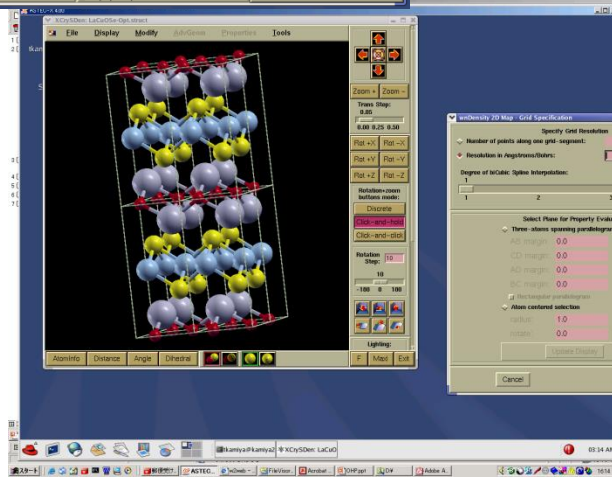
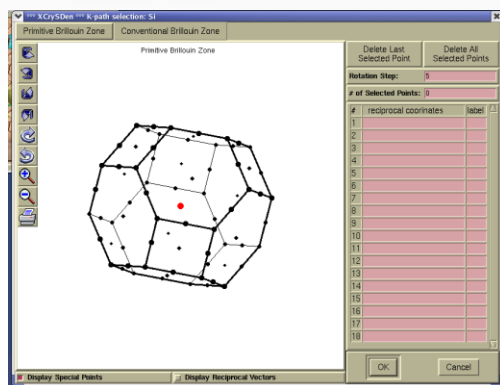
講義web: <http://d2mate.mdxes.iir.isct.ac.jp/Lecture/FundamentalsCMS/>

材料計算科学基礎

物質理工学院材料系
総合研究院 元素戦略MDX研究センター

神谷利夫

TA: 樋口龍生
南島悠人



授業日程

#09 2025/1/10 (金) 神谷担当

- ・ プログラミングの基礎 (python)
- ・ 例題: 回帰
 - ・ 線形最小二乗法 (多項式)
 - ・ 生成AI (ChatGPT、Microsoft 365 copilot, github copilot) を使ったデータ作成、プログラミング

#10 2025/1/14(火)

- ・ 生成AIの使い方 復習
- ・ python文法
- ・ 例題: 回帰
 - ・ 線形最小二乗法 (一般関数)

#11 2025/1/21(火)

- ・ 非線形最小二乗法 (numpy.curve_fit())

#12 1/24～#14 1/31 講義

評価

講義毎の課題で評価

注: 原則として、提出物の内容による評価は行わないが、

- ・ 講義で配布された以外の $+a$ があれば若干加点
- ・ 一言感想だけのように「さすがにこれはないだろう」くらいに脱力感があつたら若干減点

提出方法: T2SCHOLARで提出

提出期限: 1/16(木) 23:59

課題: 次のいずれかを選択して、成果物等を提出

- ・ 成果物の例:
 - ・ 授業で作成したプログラム (.py、.ipynb)
 - ・ 授業に関連して作成したプログラム
 - ・ 授業に関連して生成AIに質問し、その結果理解したこと
質問のプロンプトと回答、回答に関しての自分の感想・理解など
- ・ 講義に関する質問、要望。特に、第3回で行う演習の希望

質問

Q: Pythonを研究でどのように使えるのか。

A: pythonでは、研究だけでなく、いろいろな応用が見つかるはず。

- 実験データの解析、可視化 (グラフ作成)
- 実験データの蓄積 (実際にはExcelを使うか？)、データ解析
- シミュレーションプログラムの作成
- シミュレーション (量子計算など) の入力データ作成支援、計算結果の解析

pythonでできないこと・難しいこと

- 高速な計算 (C++、C#、Goなどを使う)
- ハードウェア、システム (OS、デバイスドライバ) に関するプログラムの作成

質問

Q: Pythonの参考書籍

A: 基本的に、本講義受講生に pythonの本を買って勉強をすることは勧めない

- 本講義で、「動くpythonプログラム」の作り方は理解できると期待している
それでも入門書が欲しければ、「**安くて薄い入門書**」を一通り読み、
紹介されている**プログラム**を打ち込んで動かしてみましよう

お薦めWeb: とほほのPython入門: <https://www.tohoho-web.com/python/>

- pythonでは、やりたいことのほとんどのことができるはず。
でも、それらをカバーしている本はないので、**本を集めてもきりが無い**
- プログラミングの上達に必要なこと
 - 動機: なにか**作りたいプログラム**があること
 - 検索: 作りたいプログラムの**作り方を検索する (生成AI含む)**
 - 実行とプログラムの読み込み:
良いプログラムをたくさん読むことが一番効率が良い上達法の作成

質問

Q: Pythonライブラリにはほかにどのようなものがあるのか

A: 「XXができるプログラムを作りたい」と思ったら、ほとんどのことを
実現するライブラリが見つかる。

まずは検索したりMS365 Copilotに聞いてみよう

- 科学計算: numpy (行列計算、フィッティング)、scipy (スペクトル解析、最適化)
scikit-learn (機械学習)
- グラフ: matplotlib
- MS-Office, PDFファイルの読み書き: openpyxl, pandas, python-docx, python-pptx
pymupdf
- 量子計算データの処理、材料データの検索等: pymatgen
- 画像処理, 画像認識: opencv, pyocr (OCR)
- GUI: tkinter
- Webアプリ: Flask, Dash, Django

質問

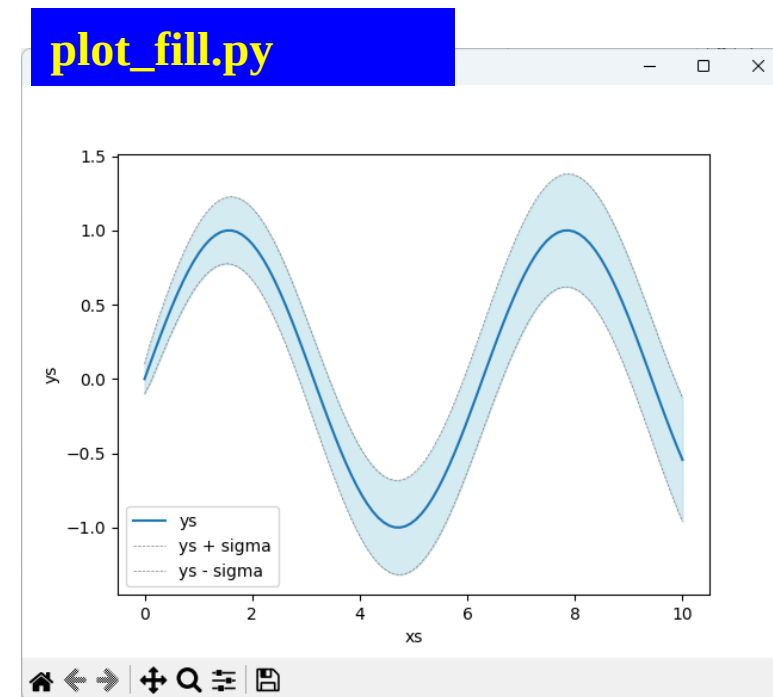
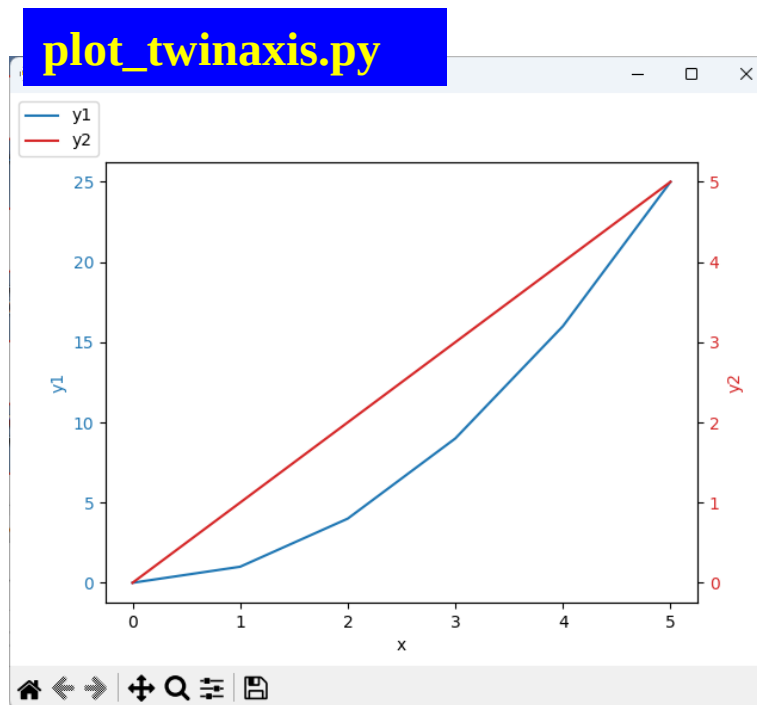
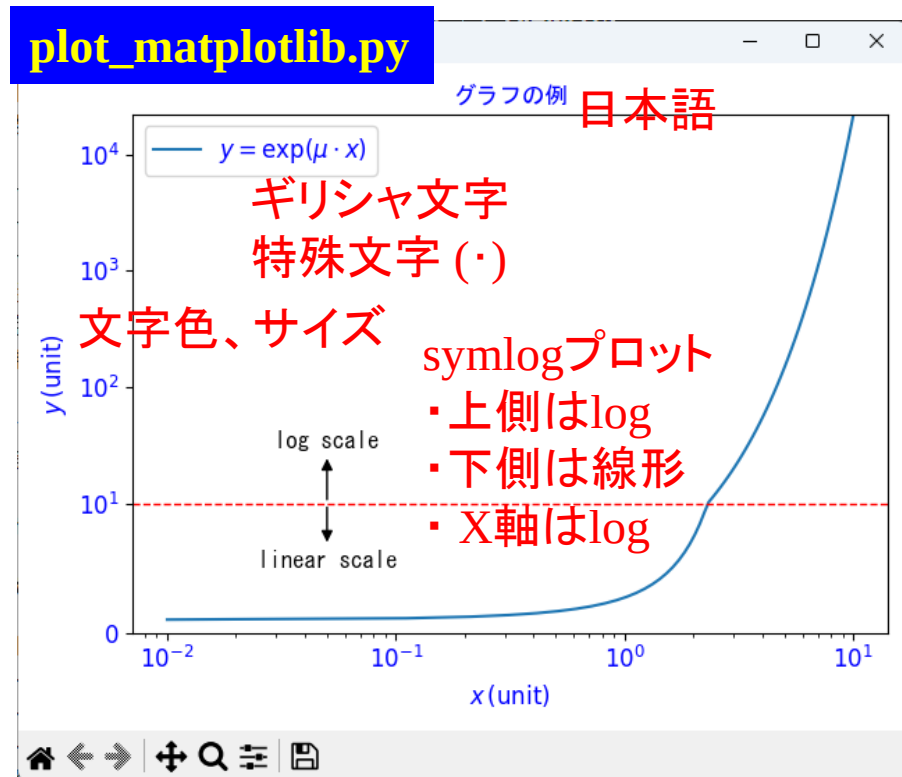
感想: matplotlibできれいな図が描けるので、慣れたらExcelから切り替えたい。

Q: Excelで学术论文に使えるグラフに整えるのは大変。

定型のデータ (Excelやテキストデータファイルなど) を読みこんで、
定型のグラフを作るのであれば、matplotlibを使ったプログラムを使うほうが早い

matplotlibでどのようなグラフが描けるか:

http://conf.mdxes.iir.isct.ac.jp/D2MatE/python/python_syntax.html の「グラフ」参照



質問

Q: .ipynb, .pyと.拡張子の違いをあまり理解できなかった

A: WindowsやLinux (BSD Unix@macOS) では**ファイルの拡張子**によって何のためのファイルかが**判断される**

.ipynb: Jupyter notebookの (python)プログラム

- 実行方法: Jupyter notebookで読み込んで実行
 - 一応、コマンドからも実行できる: **jupyter nbconvert --to notebook --execute YourNotebook.ipynb**
- 長所: 対話形式でプログラムを実行、作成できる
- 短所: プログラムの実行が面倒くさい。
ファイルがJSON形式で読みにくい
- 使い道: プログラムの開発

.py: python プログラム

- 実行方法: python YourProgramName.py
- 長所: pythonコマンドで簡単に実行できる。ファイルを見るとプログラムを読める
- 短所: 対話式での実行は難しい (Debuggerを使うことはできる)
- 使い道: 実用にする場合は.pyファイルにして、バッチファイルなどから起動

質問

Q: 自分のパソコン(Mac)ではうまく動かなかった。

- ・ pythonはversion 3.9.6しかない。

A: pythonのversionごとの機能は

http://conf.mdxes.iir.isct.ac.jp/D2MatE/python/python_history.html 参考のこと。

- ・ 講義で使っているプログラムは、**version 3.6以降**で動くはず
(自己参照 f文字列 f"{x=}" を使う場合は3.8以降)
- ・ LinuxやmacOSのコンソール (BSD Unixベース) では、**パッケージマネージャでは最新版を扱えないことも多い**。新しいpythonを使いたい場合は自分で入れる。

CPython (python.orgの標準python): ただし、パッケージマネージャと**競合問題**が生じる可能性

<https://www.python.org/downloads/>

<https://www.python.org/downloads/release/python-3131/>

Anaconda/miniconda: 最近、商用利用では無料ではなくなった。

パッケージマネージャ conda と、pipを両方使うとパッケージの競合が発生する

<https://www.anaconda.com/download>

質問: ライブラリエラー

Q: 自分のパソコンでは No module named "numpy/matplotlib" エラーが出た。

ChatGPTに聞いたら、**versionが異なると答えた**

A: 基本的にこのエラーは

numpy、matplotlibライブラリがインストールされていないため

ChatGPTの回答は間違っているか、
あるいは、「numpyとmatplotlibはインストールしたはずなのに」のような追加情報を入れて質問したために
「versionが異なる」との回答があったものと推測

ライブラリがインストールできない場合の対応法

1. pipを最新版にし、wheelライブラリの最新版をインストールする
 - > `pip install --upgrade pip`
 - > `pip install wheel`
2. パッケージをインストール、あるいは最新版にする
 - > `pip install numpy matplotlib`
 - > `pip install --upgrade numpy matplotlib`
3. それでも No module エラーが出るようであれば、アンインストールしてやりなおし
 - > `pip uninstall numpy matplotlib`
 - > `pip install numpy matplotlib`

それでもだめな場合 (仮想環境を作って全ライブラリを再インストールする方が早い)

1. インストールされているライブラリを確認する
インストール済みパッケージの一覧: > `pip list`
特定のパッケージのバージョン表示: > `pip show パッケージ名`
2. 場合によっては、使えるバージョンを調べる
 - > `pip install パッケージ名==バージョン指定でインストール`
 - > `pip install パッケージ名==バージョン番号`
 - バージョン競合などが発生した場合に、強制的にインストール
 - > `pip install --force-reinstall パッケージ名==バージョン番号`

pip 簡易マニュアル

1. パッケージのインストール
 > pip install パッケージ名
2. 最新版にする
 > pip install --upgrade パッケージ名
3. バージョン指定でインストール
 > pip install パッケージ名==バージョン番号
4. アンインストール
 > pip uninstall パッケージ名
5. インストール済みパッケージの一覧
 > pip list
6. 特定のパッケージのバージョン表示
 > pip show パッケージ名
7. パッケージを指定して、使えるバージョンを調べる
 > pip install パッケージ名==
8. バージョン競合などが発生した場合に、強制的にインストール
 > pip install --force-reinstall パッケージ名==バージョン番号
9. ユーザーのホームディレクトリにパッケージをインストール
 > pip install --user パッケージ名

Pythonの仮想環境

- ・ pythonでは多くのライブラリを使うため、わりと高い頻度で、ライブラリごとのバージョン依存性の問題 (競合) が発生する
- ・ そのため、特定のライブラリだけをインストールした「仮想環境」を作って、場合により**仮想環境を使い分ける**のが一般的

仮想環境の作り方

> python -m venv 仮想環境名 (例えばlecture)

**現在のディレクトリの下に仮想環境名のディレクトリを作り、
仮想環境用のファイルをコピーする**

仮想環境を使う (activate)

Windows:

> 仮想環境名¥Scripts¥activate

macOS、Linux:

> source 仮想環境名/bin/activate

仮想環境から抜ける (ディアクティベート)

> deactivate

その他

- Jupyter notebookでは、プログラムコード以外にもコメントブロックを入れられる: Markdown書式でも書ける
- 熱電対データの最小二乗回帰
最小二乗法では、もとのデータを通らない (誤差が発生する) 可能性がある
元のデータを必ず通るようにするためには、「**補間 (interpolate)**」を使う
例: `from scipy.interpolate.interp1d`
http://conf.mdxes.iir.isct.ac.jp/D2MatE/python/python_syntax.html の「補間」
- ファイル名について: `data_ax^4+bx^2+c.csv`
OSの特殊文字の可能性のある文字は極力使わないほうがいい
OS/ターミナルアプリの禁止文字: **/\ : *? <> |**
避けたほうがいい例: **空白, 引用符 “ ” ` , @#()[]{}\$~ => _** **を使う**
先頭文字が **.**: 隠し属性ファイルになるケースが多い
pythonのモジュール名としては **-** は使えない: **_** **を使う**

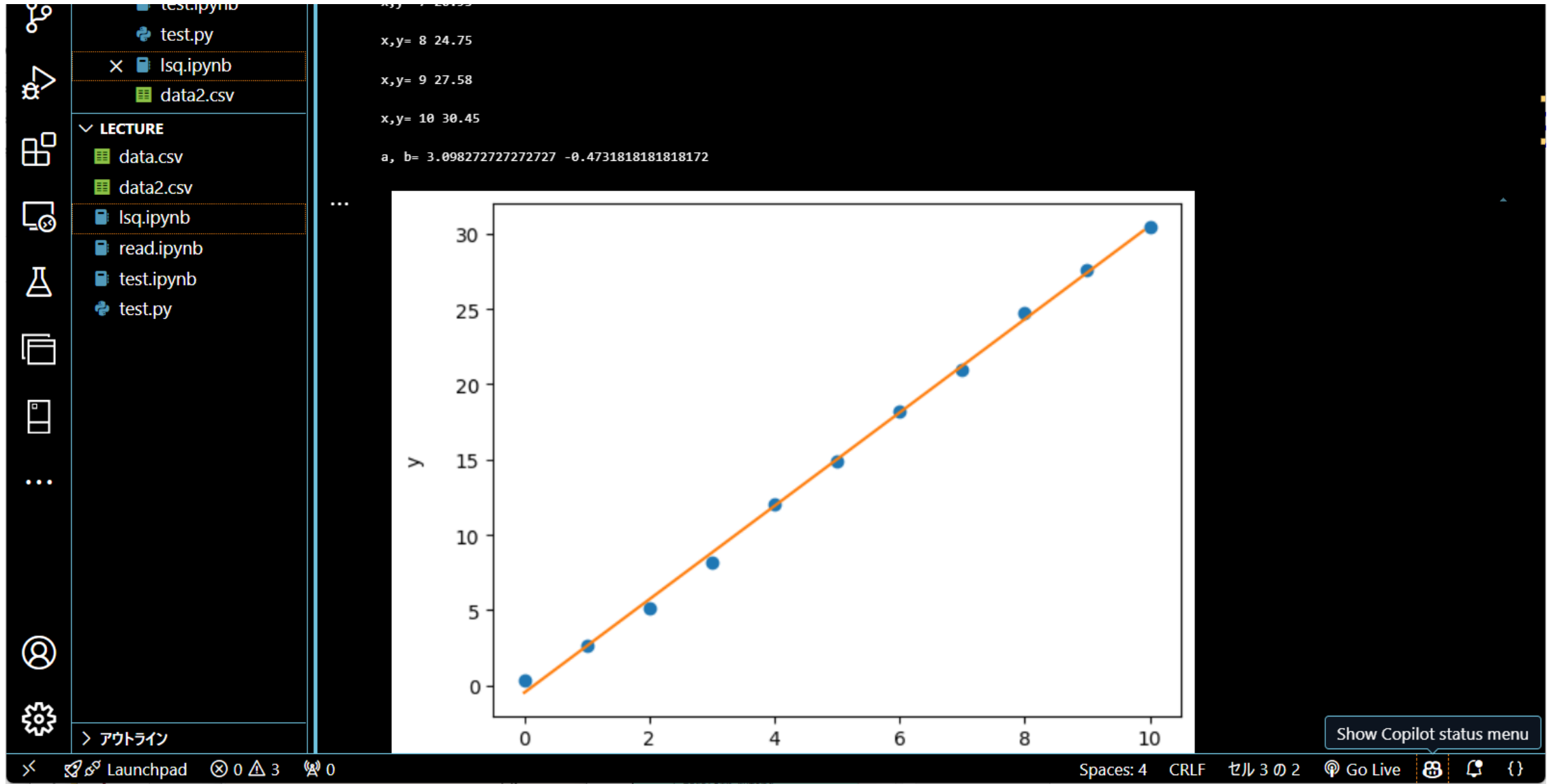
前回の復習

- Visual Studio Codeの使い方 (拡張機能)
- pythonを触ってみる (Jupyter notebook拡張機能)
- 生成AI (MS365 copilot) を使ってCSV形式のデータを作る
- 生成AIにプログラムの使い方を説明させる
- 一次多項式の最小二乗法のpythonプログラムを作成
(`numpy.polyfit()`)

一次多項式最小二乗法の実行結果

天下りの的に `polyfit()` 関数を使ったプログラムが用意されていた

↑ どうやって探すのか？



生成AI

どうやってpolyfit()関数を見つけるか

ChatGPTへの質問: “pythonで $\log N = a + b * T_{inv}$ にフィッティングするプログラム”

```
# numpyライブラリを 別名 np でimport
import numpy as np

# numpy.polyfit() の呼び出し。
# numpyは 別名 np でimportしているので、np.polyfit() で呼び出す
a, b = np.polyfit(Tinv, logN, 1)
```

```
# numpyライブラリから polyfit を import
from numpy import polyfit

# polyfit() だけで呼び出せる
a, b = polyfit(Tinv, logN, 1)
```

でも、この時の回答のプログラムは間違っていた

numpy polyfitを検索してみる

Bingで“numpy polyfit”を検索



NumPy

<https://numpy.org> › [stable](#) › [generated](#) · [このページを訳す](#) ⋮

[numpy.polyfit](#) — NumPy v1.26 Manual

公式のAPIリファレンス

Relative condition number of the fit. Singular values smaller than this relative to the largest singular value will be ignored. The default value is $\text{len}(x) \cdot \text{eps}$, ...

[Numpy.poly1d](#) · [Numpy.polyval](#) · [Numpy.polyder](#) · [Convenience Classes](#)



Qiita

<https://qiita.com> › [Python](#) ⋮

知恵袋系: Qiita, Quoraの質が高い

[Numpy.polyfit](#) を使ったカーブフィッティング #Python

2019/05/10 — 様々な補間法と最小2乗法をPythonで理解する のうち、「[Numpy.polyfit](#) を使ったカーブフィッティング」を、実データっぽい模擬データを解析するように ...

numpy polyfitを検索してみる

Bingで“numpy polyfit”を検索



よちよちpython

<https://chayarokurokuro.hatenablog.com> › Android

Numpyだけで回帰分析その4。polyfit()について。

2020/01/22 — polyfit() 多項式係数生成マシン. 各点(x,y)を結ぶ線に近似する次数degまでの多項式の係数を計算し出力する。... この点を結ぶ線に近似する、近からず ...

<https://chayarokurokuro.hatenablog.com/entry/2020/01/22/190133>

python

```
# 近似線の係数を算出
k = np.polyfit(x, y, 1)

# 変数に置き換えとこ
a, b = k

# 適当にx軸の両端の値
x_reg = np.array([0, 25])

# 直線の式を作って配列xを代入
y_reg = a * x_reg + b
```

polyfitで返ってくる係数は、
 $y = a * x + b$
の順番

ChatGPTは、 $y = a + b * x$ の係数を
 $a, b = np.polyfit(Tinv, logN, 1)$
で受け取っていたので、a と b がひっくり返っている

注意: ChatGPTは気を付けながら使いましょう

- 堂々と間違いを言う
- 計算は大丈夫?: MS365 copilotに質問したら、GPT3以降は結構計算できると答えた。でも、信用しないこと
- 役に立つことも多い
- 受け答えは、学習されて将来の回答に使われる可能性がある。
秘密情報は絶対に入れない

注意

- 回答の真偽を見分けること
異なる表現で複数回聞いてみる
他の情報ソースとクロスチェックする

**AIを有効利用するには
人間にも相応の「知識」が必要**

生成AI (大規模言語モデル LLM) の利用

VSCodeの**ChatGPT**拡張機能を使うときは、**OpenAI API Key** (課金) が必要
ChatGPTで**GPT4o**などの最新のモデルを使うにも 課金が必要。

1. <https://platform.openai.com/docs/overview>
2. ただし、無料の**Microsoft 365 Copilot** でも Chat GPT4を使っているので、それ以上のGPTモデルを使う必要が無ければ、API Keyの課金は勧めない

Github copilotの利用

VSCodeの**github copilot**拡張機能を使うと、コード内でコメントを書いてコード生成ができる

1. 科学大の学生は無料で使える
<https://docs.github.com/ja/education>

課題の回答から: Numpy.polynomial polyfit

Qiita : <https://qiita.com/sitar-harmonics/items/37f482d7fa2dcb9ddc91>

NumPy API Ref : <https://numpy.org/doc/stable/reference/routines.polynomials.html>

numpy.polynomial.polynomial.polyfit()

注: `np.polyfit()`と異なり、**戻り値は次数が小さい係数から大きい係数の順**
(`np.polyfit()`では、**次数が大きい係数から小さい係数の順**)

```
import numpy as np
```

```
ai = np.polynomial.polynomial.polyfit(xs,ys,3)
```

```
np.polynomial.polynomial.polyval()
```

#計算値の計算もpolyval()だけで実行できる

```
xcal = np.arange(0.0, 10.0, 0.1)
```

```
ycal = np.polynomial.polynomial.polyval(xcal, ai)
```

#importを工夫してコードを簡単にする:

```
from numpy.polynomial.polynomial import polyval
```

```
ai = polyval(xs, ys, 3)
```

```
xcal = np.arange(0.0, 10.0, 0.1)
```

```
ycal = polyval(xcal, ai)
```

polyfit()のプログラム例

http://conf.mdxes.iir.isct.ac.jp/D2MatE/python/python_syntax.html

最小二乗法

- numpy.polyfit()を用いたn次多項式最小二乗 (numpy.polyfit(), forループで計算値を計算)
- numpy.polyfit()を用いたn次多項式最小二乗 (numpy.polyfit()+np.poly1d())
- numpy.polynomial.polynomial.polyfit()を用いたn次多項式最小二乗 (polyfit()+polyval())

Pythonの文法

講義Web: pythonの文法とプログラム例

<http://d2mate.mdxes.iir.isct.ac.jp/Lecture/FundamentalsCMS>

[D²MatE Top](#)

[original HTML](#)

[display HTML](#)

2024年度Q4 材料計
算科学基礎

python

プログラム例と文法

講義予定

#01 ~ #08: ・機械学習の基
礎・演習

#09: ・Visual Studio Code
による環境構築

・pythonプログラム実習:
polyfit()を用いた多項式最小
二乗プログラムの作成

#10:

・より複雑な最小二乗法
線形最小二乗法
非線形最小二乗法

#11:

・生成AIを利用したプログラ
ム作成

#12~#14:講義

[original HTML](#)

[display HTML](#)

簡単なプログラム例

- ・ ▶ キーボード入力とコンソール出力 ([download/show](#))
- ・ ▶ list型とforループ (基本型) ([download/show](#))
- ・ ▶ list型とforループ (list変数をiteratorとして使う) ([download/show](#))
- ・ ▶ list型とforループ (enumerate) ([download/show](#))
- ・ ▶ リスト内包表記を使ったforループ ([download/show](#))
- ・ ▶ if ~ elif ~ else ([download/show](#))
- ・ ▶ CSVファイルの読み込み (基本形) ([download/show](#))
- ・ ▶ CSVファイルの読み込み (with構文) ([download/show](#))
- ・ ▼ CSVファイルの読み込み (CSVライブラリ) ([download/show](#))

Clipboardへコピー

copy

Exampleプログラムの内容

```
import csv

infile = "data.csv"

with open(infile, "r") as fp:
    csv_reader = csv.reader(fp)
    labels = next(csv_reader)
    data_list = []

    for row in csv_reader:
        row = [float(d) for d in row]
        data_list.append(row)

print("labels:", labels)
print("data_list:", data_list)
```

input.csvファイルを開いてファイルハンドルを取得
CSVファイルを読み込むため、csv.reader() でreaderオブジェクトを取得
labelを読み込む
1行ずつ読み込み、list型配列をrowに入れる
rowの要素 (文字列型) を実数型 (float) に変換してリストを作る

エラー・バグ報告の仕方

```
isigma: 1
is : 2
Read data from [.%test.csv]
Traceback (most recent call last):
  File "%weighted_mobility.py", line 392, in <module>
    main()
  File "%weighted_mobility.py", line 384, in main
    exec_constT()
  File "%weighted_mobility.py", line 244, in exec_constT
    labels, datalist = read_csv(infile)
  File "%weighted_mobility.py", line 178, in read_csv
    labels = next(fin)
UnicodeDecodeError: 'cp932' codec can't decode byte 0xef in position 0: illegal multibyte sequence
(base) PS F:%data-UE%weighted_mobility>
```

Pythonなどのinterpreter型言語は、エラーを起こしたプログラム行数、内容、原因などを表示してくれる。

バグレポート: 以下の内容をまとめて報告すること

1. OS (Windows10, OS Xなど) とプログラム環境 (python、Jupyterなど) とバージョン
2. 実行したプログラムのバージョン (不明な場合はファイルのタイムスタンプ)。

あるいはプログラム自身を添付

3. 入力ファイル、出力ファイルを添付
4. エラーメッセージのスクリーンショット、
あるいはコンソール出力のファイル

コンソール出力の取り方 (リダイレクト > を使ってファイルに落とす)

c:> 実行コマンドライン > out.txt

とすると、”実行コマンドライン” のコンソール出力が out.txtに保存されます。

一次多項式の線形最小二乗法

機械学習回帰のチュートリアルコース 2022年

<http://conf.mdxes.iir.isct.ac.jp/D2MatE/2022Tutorial/tutorial2022.html>

材料計算科学・データ解析チュートリアルコース 2022年度

2022年度チュートリアルコースは終了いたしました。

チュートリアル資料の更新は、下記の4/2 14:04更新と録画公開で終了しました。

配布プログラムの更新は、[Topページ](#)にて行います。

-
- 2023/4/2 14:04 第5回チュートリアル (3月22日) 講義資料 最終更新版: [20230322Tutorial.zip](#)
2023/4/1 注: Arrhenius plot機能について、活性化エネルギーの計算に間違いがあったものを修正しました。
 - 2023/3/22 第1～5回チュートリアル録画 2023年3月いっぱいので予定で公開: [2022年度チュートリアル録画](#)

チュートリアル資料 再構成版: チュートリアルの資料をテーマごとにまとめなおしました。

ファイル名が日本語になっています。ダウンロードができない場合、tkamiya@msl.titech.ac.jp までご連絡ください。

- [01-注意.pdf](#)
- [02-Launcher.pdf](#)
- [02-python-tips.pdf](#)
- [03-強化学習プログラムbaves gp qui.pyの使い方.pdf](#)
- [04-線形最適化・最小二乗法\(回帰\).pdf](#)
- [05-機械学習\(回帰\).pdf](#)
- [06-非線形最適化・最小二乗法.pdf](#)

最小二乗法: 二乗誤差を最小化

問題: あるデータの組 $(x_1, y_1), \dots, (x_n, y_n)$ が理論式 $f(x) = a + bx$ に従うことがわかっている。未知変数 a と b を求めよ。
ただし、データ (y_i) には誤差 ε_i が含まれている: $y_i = f(x_i) + \varepsilon_i$

直観的に、以下の最小化問題を解けばいいと見当がつく

- $\max_{x_i} |f(x_i) - y_i|$ を最小化: ミニマックス近似
- L1ノルム $S = \sum |f(x_i) - y_i|$ を最小化
プログラムが比較的面倒
- L2ノルム (ユークリッド距離) の二乗 $S = \sum (f(x_i) - y_i)^2$ を最小化: 最小二乗法
(線形最適化では) プログラムが非常に簡単

$$L_p \text{ノルム: } [\sum |f(x_i) - y_i|^p]^{1/p}$$

$f(x)$: モデル

線形最小二乗法 (線形回帰)

線形問題: $f(x_i) = a + bx_i$ のように、 $f(x_i)$ が未知変数 a, b の線形関数になっている
 x_i の線形関数である必要はない

$$f(x_i) = a + bx_i$$

目的関数 $S = \sum (a + bx_i - y_i)^2$ を最小化

$$dS/da = 2\sum (a + bx_i - y_i) = 2an + 2b\sum x_i - 2\sum y_i = 0$$

$$dS/db = 2\sum x_i(a + bx_i - y_i) = 2a\sum x_i + 2b\sum x_i^2 - 2\sum x_i y_i = 0$$

$$\times 2n \cdot a + 2\sum x_i \cdot b = 2\sum y_i$$

$$2\sum x_i \cdot a + 2\sum x_i^2 \cdot b = 2\sum x_i y_i$$

の連立方程式を解けばよい

行列で表したほうが見通しがいい

$$\begin{pmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum y_i \\ \sum x_i y_i \end{pmatrix}$$

2x2の行列方程式を解けば、 (a, b) が求まる

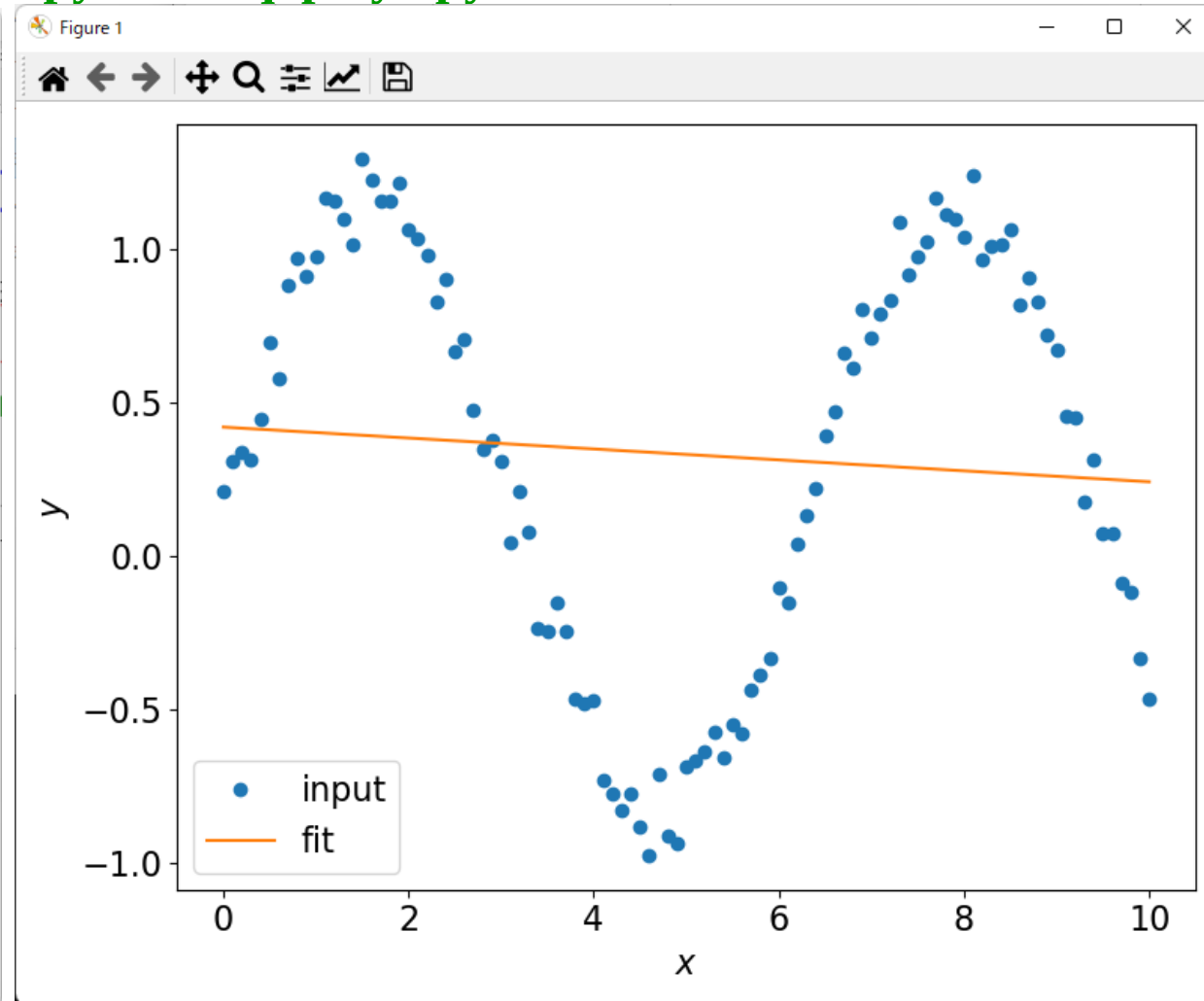
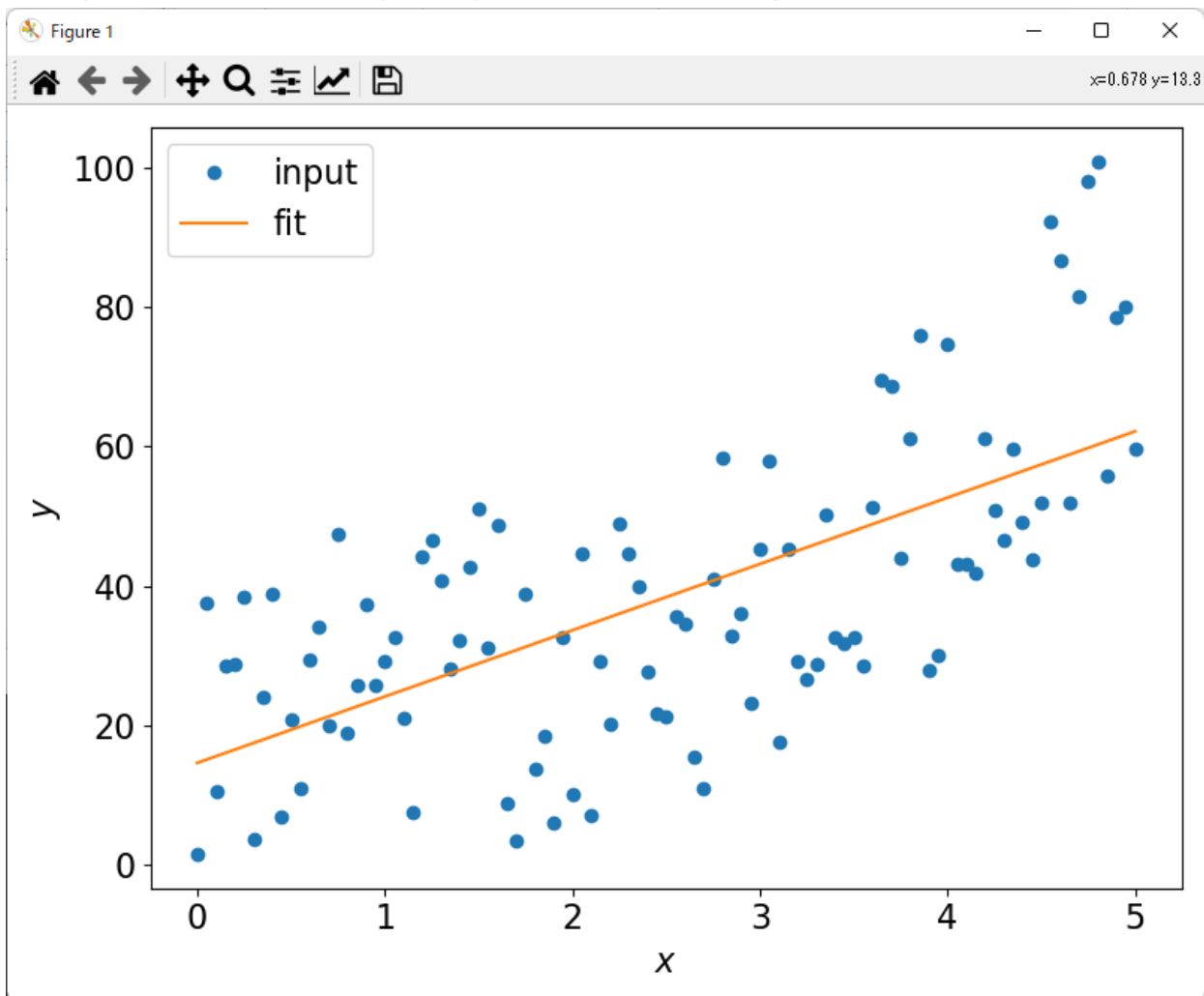
Program: lsq_poly1.py

Usage: `python lsq_poly1.py input_path`

▶ 実行アイコンで起動時引数を指定するのは大変なので、ターミナルから打ち込む

`python lsq_poly1.py random-poly.xlsx`

`python lsq_poly1.py random-sin.xlsx`



Program: lsq_poly1.py

```
import sys
import numpy as np
from numpy import sin, cos, tan, pi
from pprint import pprint
import csv
import openpyxl
import pandas as pd
import matplotlib.pyplot as plt
```

#起動時引数を受け取る

#配列を見やすく表示
#CSVファイルの読み書き
#Excel (.xlsx)ファイルの読み書き
#Excel (.xlsx)ファイルの読み書き

```
infile = 'random-poly.xlsx'
outfile = 'lsq-poly1.csv'
fontsize = 16
```

#デフォルトの入力ファイル名
#デフォルトの出力ファイル名
#グラフのフォントサイズ

```
argv = sys.argv
narg = len(argv)
if narg >= 2:
    infile = argv[1]
```

起動時引数のリスト変数をargvに代入
#argvの要素数をnargに代入
#**argv[0]には起動スクリプトのパスが入る (nargは必ず1以上)**
#起動時引数を与えられていたら (narg>=2)
#第一起動引数 argv[1] を infile に置き換える

Program: lsq_poly1.py

最小二乗計算を実行する関数 lsq を定義する

def lsq(x, y, iPrint = 0):

 n = len(x)

numpyの配列 ndarray を使うと、行列計算が簡単にできる

ただし、行列計算をする場合には、ベクトルも 2x1の 行列としないといけない

 Si = np.empty([2, 1])

 # 2x1の行列 (ベクトル) を作る。要素は初期化しない

 Sij = np.empty([2, 2])

 # 2x2の行列 を作る。要素は初期化しない

 x_sum = 0.0

 x2_sum = 0.0

 y_sum = 0.0

 xy_sum = 0.0

 for i in range(0, n):

 x_sum += x[i]

 # $x_sum = \sum x_i$

 x2_sum += x[i]**2

 # $x2_sum = \sum x_i^2$

 y_sum += y[i]

 # $y_sum = \sum y_i$

 xy_sum += x[i] * y[i]

 # $xy_sum = \sum x_i y_i$

Program: lsq_poly1.py

```
def lsq(xs, ys, iPrint = 0):
```

```
--cut--
```

```
    for i in range(0, n):
```

```
        x_sum += xs[i]           #  $x\_sum = \sum x_i$   
        x2_sum += xs[i]**2       #  $x2\_sum = \sum x_i^2$   
        y_sum += ys[i]          #  $y\_sum = \sum y_i$   
        xy_sum += xs[i] * ys[i] #  $xy\_sum = \sum x_i y_i$ 
```

```
    Sij[0, 0], Sij[1, 0] = n, x_sum      # Sij行列要素に代入  
    Sij[0, 1], Sij[1, 1] = x_sum, x2_sum # Sij行列要素に代入  
    Si[0], Si[1] = y_sum, xy_sum        # Siベクトル要素に代入
```

```
    if iPrint == 1:                    # lsq()関数に iPrint = 1 を与えられた場合のみ、repeat出力  
        print("Vector and Matrix:")  
        print("Si=")  
        pprint(Si)  
        print("Sij=")  
        pprint(Sij)  
        print("")
```

Program: lsq_poly1.py

```
def lsq(x, y, iPrint = 0):
```

```
--cut--
```

```
# numpy.ndarray同士に @ 演算子 を使うと、行列の積を計算する
```

```
# np.linalg.inv(Sij) で、Sijの逆行列を計算
```

```
    ci = np.linalg.inv(Sij) @ Si    # 2x2行列と2x1行列の積なので、返り値の ci は2x1行列
```

```
# 注: ci は2x1の2次元リストなので、ciの転置の第0成分が係数ベクトル
```

```
# ciの転置を取って (ci.transpose()) リスト変数に変換 (.tolist())
```

```
    ci = ci.transpose().tolist()
```

```
# ci[0] が係数ベクトル
```

```
return ci[0]
```

Program: lsq_poly1.py

```
# よくつかわれる書き方: __name__ 変数で動作を変える
# lsq-poly1.pyがpythonコマンドの引数で与えられたら (起動するクリプトであれば)、
#   __name__ = “__main__” が入っている
# lsq-poly1.pyがライブラリとしてimportされたら、
#   __name__ にはモジュール名 (ライブラリスクリプトのファイル名、lsq-poly1) が入っている
```

```
# 以下の書き方をすることで、起動スクリプトで呼ばれたときには main() 関数を呼び出し、
# importされたときには main() 関数を呼び出さずに、純粹にライブラリとして使える
```

```
def main():
```

```
--cut--
```

```
if __name__ == "__main__":
    main()
```

注: ライブラリ名に “-” は使えないので、“_” を使うようにする

Program: lsq_poly1.py

def main():

print("Least-squares method for linear function")

print(f"norder=1")

計算条件の表示

print(f"infile={infile}")

入力パラメータのrepeat

print(f"outfile={outfile}")

print("")

print(f"Read [{infile}]")

何をしようとしているかを表示すると、デバッグが楽

df = pd.read_excel(infile, engine = 'openpyxl')

pandasとopenpyxlを使ってExcelファイルを読み込む

labels = df.columns.to_list()

1行目のラベルをリスト変数labelsに入れる

xs = df[labels[0]]

1列目のデータを変数 xs に入れる (変数型は DataFrame型)

ys = df[labels[1]]

2列目のデータを変数 ys に入れる (変数型は DataFrame型)

ndata = len(xs)

print("")

print(f"Execute linear least-squares method")

ci = lsq(xs, ys, iPrint = 1)

関数 lsq()を呼び出して最小二乗法を実行

Program: lsq_poly1.py

```
def main():
```

```
--cut--
```

```
    print("""
```

```
    print(f"Execute linear least-squares method")
```

```
    ci = lsq(x, y, iPrint = 1)          # 関数 lsq()を呼び出して最小二乗法を実行
```

```
    print("LSQ function")
```

```
    print(f"f(x) = {ci[0]} + {ci[1]} * x")    # 係数を関数形で表示すると見やすい
```

```
    print("""
```

```
    ycal = []
```

```
    fp = open(outfile, 'w')
```

```
    fout = csv.writer(fp, lineterminator='¥n') # CSVモジュールを使ってCSVファイルに書き込む
```

```
    fout.writerow(['x', 'y', 'y(LSQ)'])
```

```
    for i in range(0, ndata):
```

```
        yl = ci[0] + ci[1] * xs[i]
```

```
        ycal.append(yl)
```

```
        fout.writerow([xs[i], ys[i], yl])
```

```
    fp.close()
```

Program: lsq_poly1.py

```
def main():
```

```
--cut--
```

```
fig, axes = plt.subplots(1, 1, figsize = (8, 6))
```

```
axes.plot(xs, ys, label = 'input', linestyle = '', marker = 'o')
```

```
axes.plot(xs, ycal, label = 'fit', linestyle = '-')
```

```
axes.tick_params(labelsize = fontsize)
```

```
axes.set_xlabel('$x$', fontsize = fontsize)
```

文字列を \$~\$ で囲んだ部分はLaTeXが使える

```
axes.set_ylabel('$y$', fontsize = fontsize)
```

```
axes.legend(fontsize = fontsize)
```

```
plt.tight_layout()
```

```
plt.pause(0.1)
```

pause()を使うと、グラフを描画してプログラムを継続実行

```
print("")
```

```
print("Press ENTER to terminate")
```

```
input()
```

pause()を使う場合、input()でキー入力待ちにする

ENTERを押すと、グラフを閉じてプログラム終了

m次多項式の線形最小二乗法

線形最小二乗法: 多項式

$$f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_px^p = \sum_{k=0}^p a_k x^k$$

これも、 $f(x)$ は a_i に関する線形関数

$f(x)$ は x に関する非線形関数 (p 次多項式)

目的関数 $S = \sum_{j=1}^n (\sum_{k=0}^p a_k x_j^k - y_j)^2$ を最小化

$$\frac{dS}{da_{k'}} = 2 \sum_{j=1}^n x_j^{k'} (\sum_{k=0}^p a_k x_j^k - y_j) = 0 \quad k'=0, 1, \cdots, p$$

$$\sum_{j=1}^n (\sum_{k=0}^p a_k x_j^{k+k'} - y_j x_j^{k'}) = 0$$

$$\text{※ } \sum_{k=0}^p a_k \sum_{j=1}^n x_j^{k+k'} = \sum_{j=1}^n y_j x_j^{k'}$$

線形最小二乘法：多項式

$$\sum_{k=0}^p a_k \sum_{j=0}^n x_j^{k+k'} = \sum_{j=0}^n y_j x_j^{k'}$$

$$\begin{pmatrix} n & \sum x_i & \sum x_i^2 & \cdots & \sum x_i^p \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & & \sum x_i^{p+1} \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & & \sum x_i^{p+2} \\ \vdots & & & \ddots & \\ \sum x_i^p & \sum x_i^{p+1} & \sum x_i^{p+2} & & \sum x_i^{2p} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_p \end{pmatrix} = \begin{pmatrix} \sum y_i \\ \sum y_i x_i \\ \sum y_i x_i^2 \\ \vdots \\ \sum y_i x_i^p \end{pmatrix}$$

$$X_{k,k'} = \sum_{j=1}^n x_j^{k+k'}$$

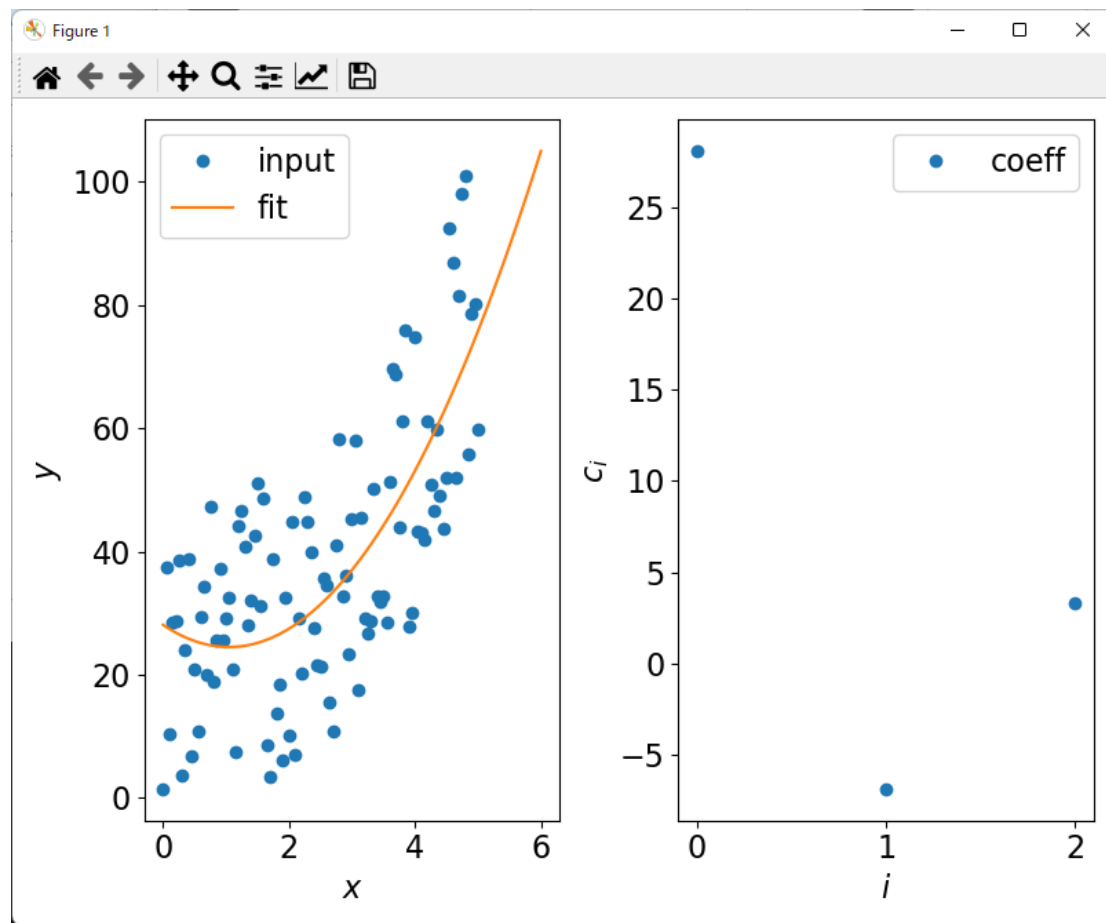
$$A_k = a_k \quad Y_k = \sum_{j=1}^n y_j x_j^{k'}$$

$$XA=Y \quad A=X^{-1}Y$$

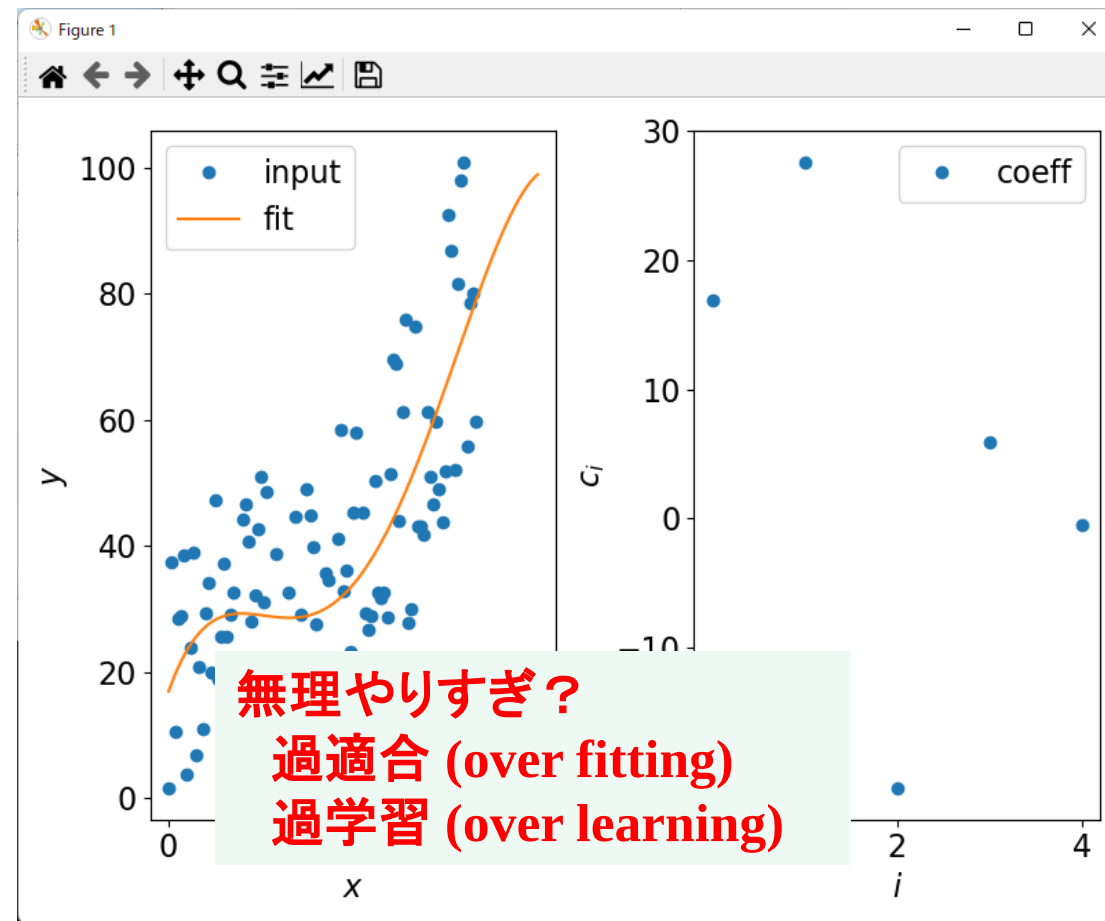
Program: lsq_polynomial.py

Usage: `python lsq_polynomial.py input_path norder`

`python lsq_polynomial.py random-poly.xlsx 2`



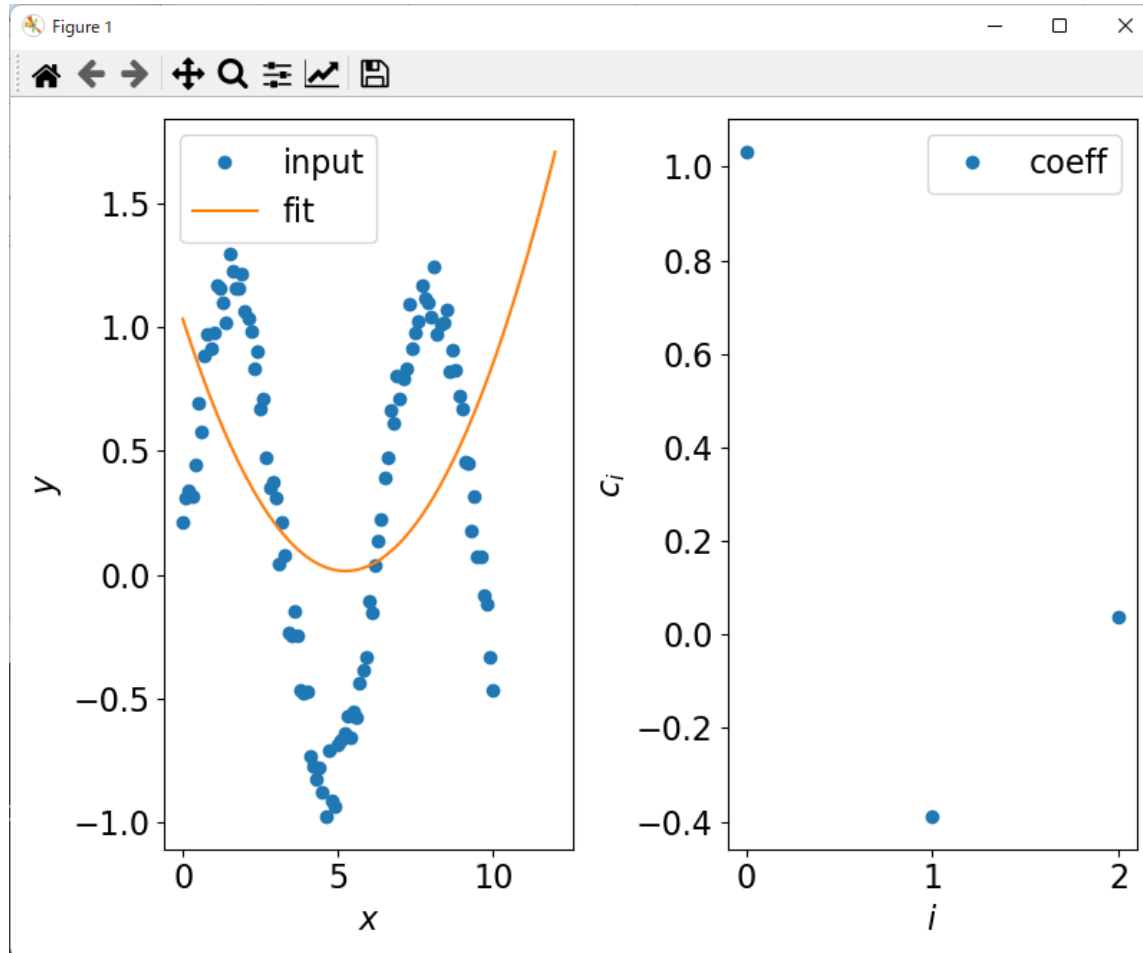
`python lsq_polynomial.py random-poly.xlsx 4`



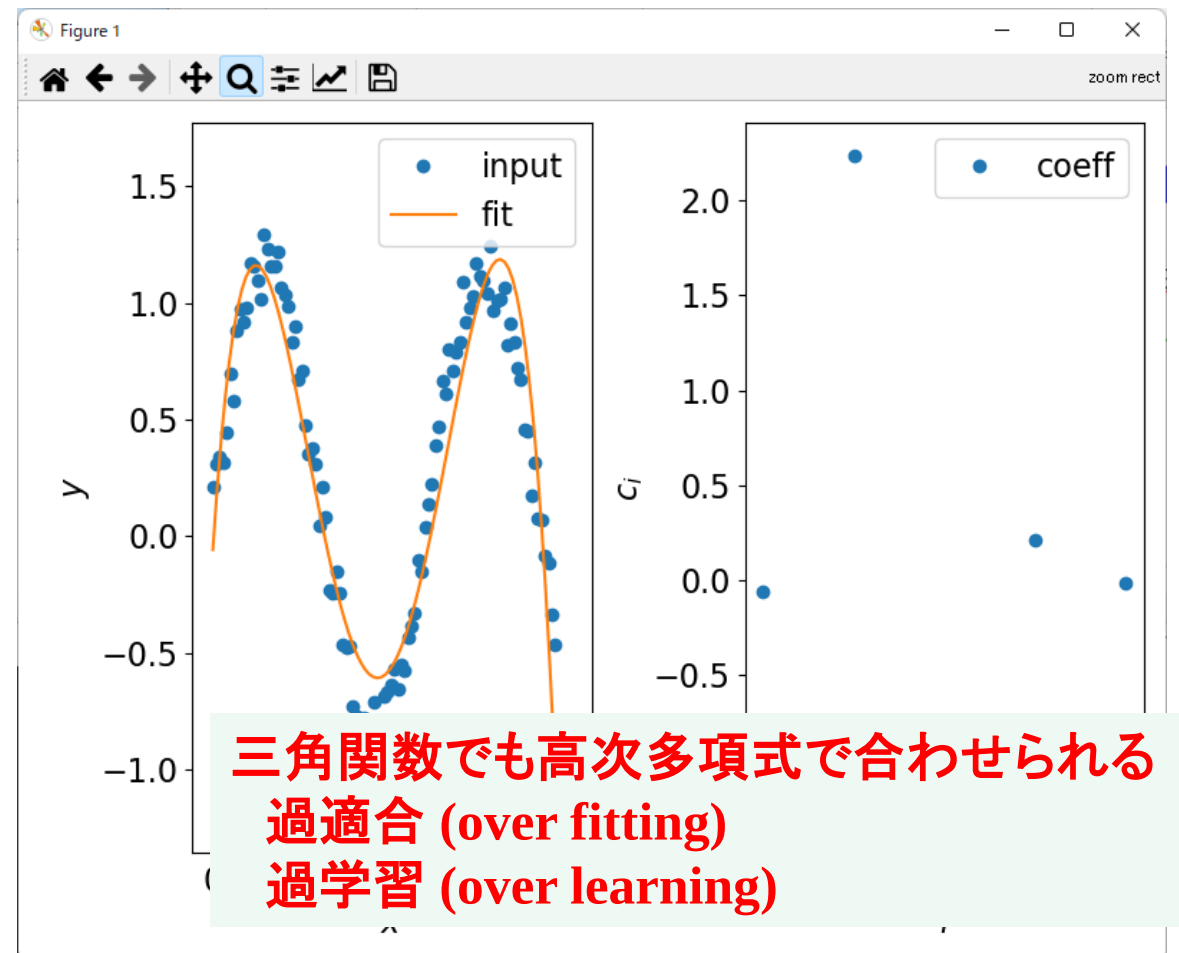
Program: lsq_polynomial.py

Usage: `python lsq_polynomial.py input_path norder`

`python lsq_polynomial.py random-sin.xlsx 2`



`python lsq_polynomial.py random-sin.xlsx 4`



Program: lsq_polynomial.py

プログラムの説明はしないので、
MS365 Copilotに聞いてね

線形最小二乗法: 任意の関数

$$f(x) = a_1 f_1(x) + a_2 f_2(x) + \cdots + a_p f_p(x) = \sum_{k=0}^p a_k f_k(x)$$

$f(x)$ は a_i に関する線形関数

$f_k(x)$ がどんなに複雑な関数でも、あるいは数値テーブルだったとしても、OK

目的関数 $S = \sum_{j=1}^n (\sum_{k=1}^p a_k f_k(x_j) - y_j)^2$ を最小化

$$\frac{dS}{da_{k'}} = 2 \sum_{j=1}^n f_{k'}(x_j) (\sum_{k=1}^p a_k f_k(x_j) - y_j) = 0 \quad k'=0, 1, \cdots, p$$

$$\sum_{j=1}^n (\sum_{k=1}^p a_k f_{k'}(x_j) f_k(x_j) - y_j f_{k'}(x_j)) = 0$$

$$\text{※ } \sum_{k=1}^p a_k \sum_{j=1}^n f_{k'}(x_j) f_k(x_j) = \sum_{j=1}^n y_j f_{k'}(x_j)$$

線形最小二乗法: 任意の関数

$$\sum_{k=1}^p a_k \sum_{j=0}^n f_{k'}(x_j) f_k(x_j) = \sum_{j=0}^n y_j f_k(x_j)$$

$$\begin{pmatrix} \sum f_1(x_i) f_1(x_i) & \sum f_1(x_i) f_2(x_i) & \sum f_1(x_i) f_3(x_i) & \cdots & \sum f_1(x_i) f_p(x_i) \\ \sum f_2(x_i) f_1(x_i) & \sum f_2(x_i) f_2(x_i) & \sum f_2(x_i) f_3(x_i) & & \sum f_2(x_i) f_p(x_i) \\ \sum f_3(x_i) f_1(x_i) & \sum f_3(x_i) f_2(x_i) & \sum f_3(x_i) f_3(x_i) & & \sum f_3(x_i) f_p(x_i) \\ \vdots & & & \ddots & \\ \sum f_p(x_i) f_1(x_i) & \sum f_p(x_i) f_2(x_i) & \sum f_p(x_i) f_3(x_i) & & \sum f_p(x_i) f_p(x_i) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_p \end{pmatrix} = \begin{pmatrix} \sum y_i f_1(x_i) \\ \sum y_i f_2(x_i) \\ \sum y_i f_3(x_i) \\ \vdots \\ \sum y_i f_p(x_i) \end{pmatrix}$$

$$X_{k,k'} = \sum_{j=1}^n f_{k'}(x_j) f_k(x_j) \quad A_k = a_k \quad Y_k = \sum_{j=1}^n y_j f_k(x_j)$$

$$XA=Y \quad \text{を解く: } A=X^{-1}Y$$

$f_k(x)$ は複数の変数 $\{x^{(m)}\}$ の関数 $f_k(\{x_j^{(m)}\})$ でもよい

Program: lsq_general.py

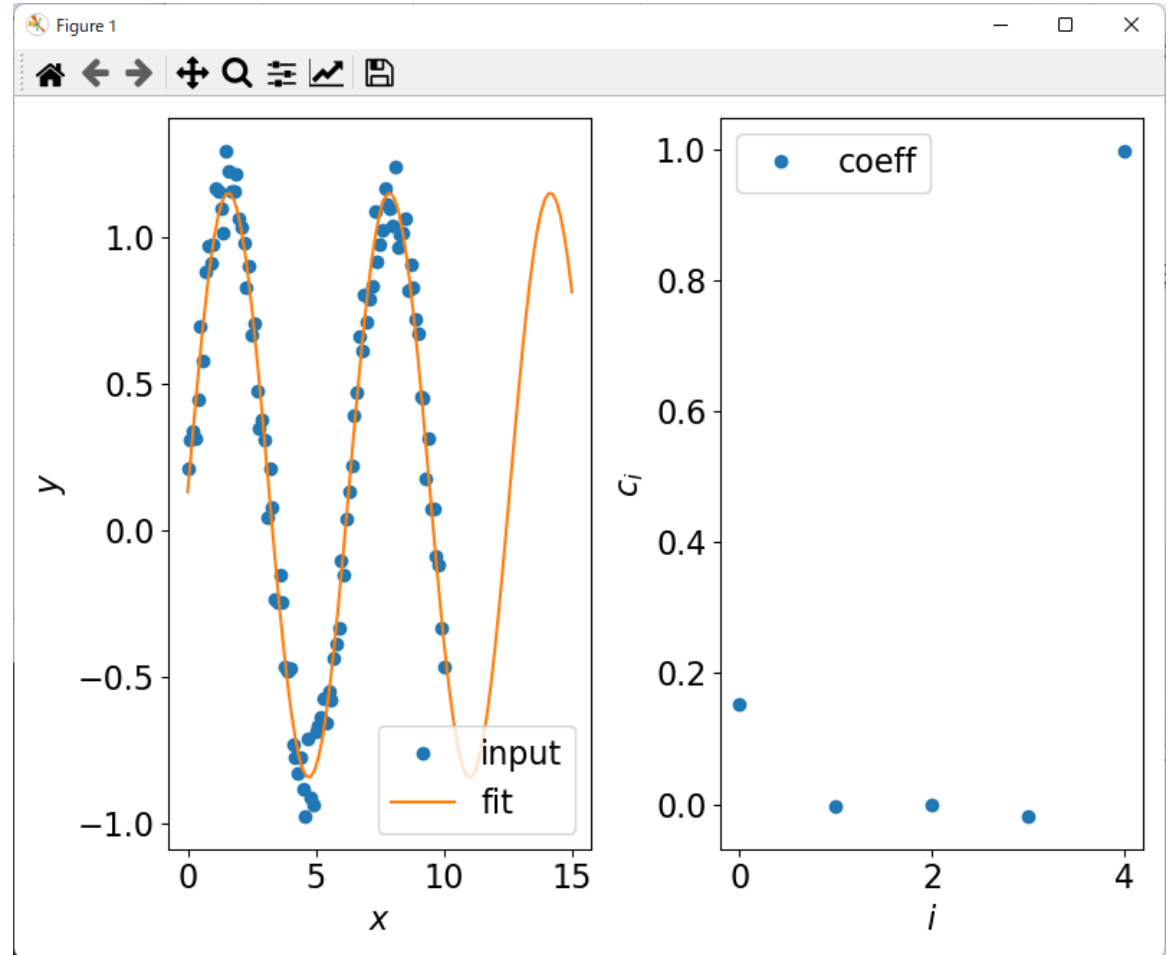
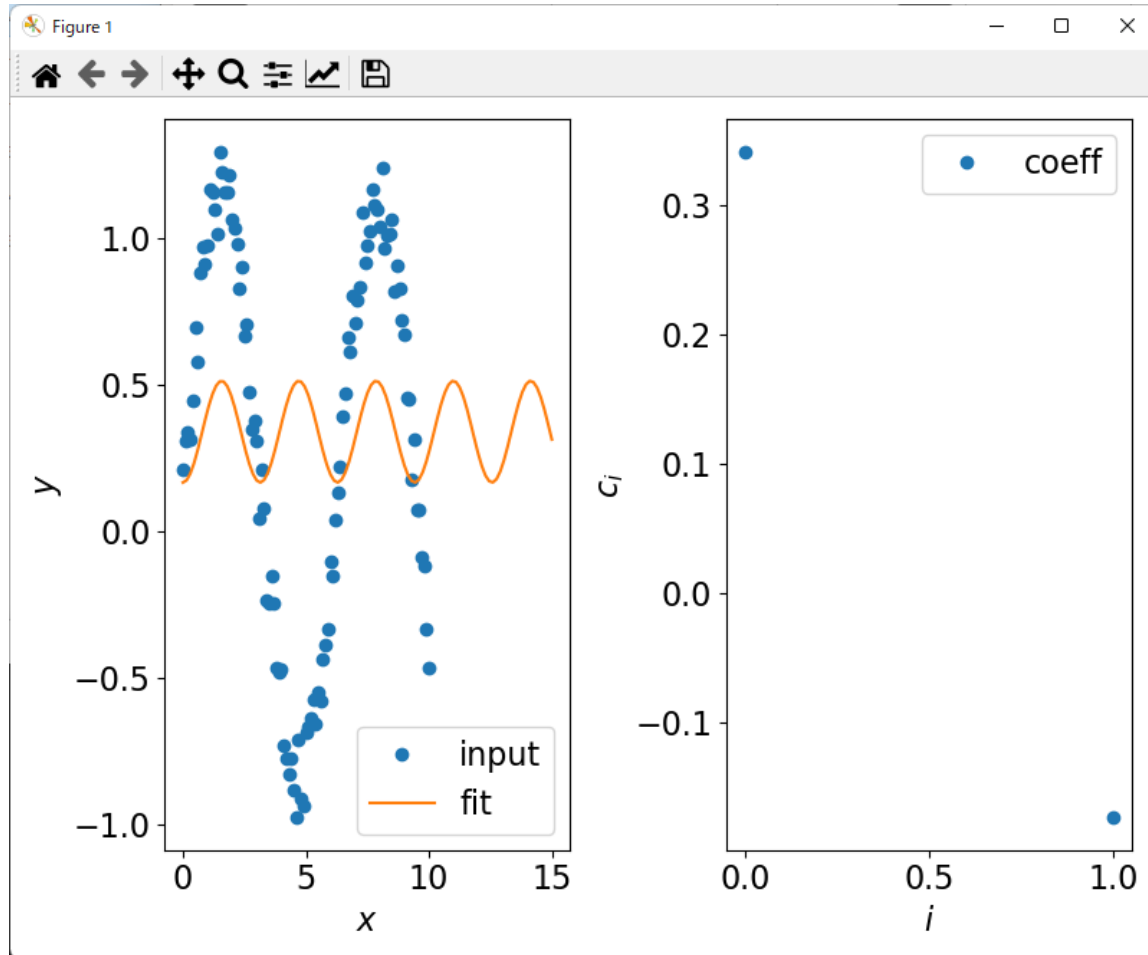
Usage: `python lsq_general.py input_path nfunc`

`python lsq_general.py random-sin.xlsx 2`

$$f(x) = 0.341 - 0.173\cos(2x)$$

`python lsq_general.py random-sin.xlsx 5`

$$f(x) = 0.152 - 0.00252\cos(2x) - 0.000347\sin(2x) - 0.0177\cos(x) + 0.997\sin(x)$$



Program: lsq_general.py

項毎に関数形が違うので、項が区別できる文字列リスト変数 flabels を作っておくとわかりやすい

```
flabels = ['1', 'cos(2x)', 'sin(2x)', 'cos(x)', 'sin(x)', 'cos(3x)', 'sin(3x)', 'x', 'x^2', 'exp(x)']
```

i 番目の関数の値を返す関数。if文で条件分岐する

def lsqfunc(i, x):

if i == 0:

return 1.0

elif i == 1:

return cos(2.0*x)

elif i == 2:

return sin(2.0*x)

elif i == 3:

return cos(x)

elif i == 4:

return sin(x)

elif i == 5:

return cos(3.0*x)

elif i == 6:

return sin(3.0*x)

elif i == 7:

return x

elif i == 8:

return x**2

elif i == 9:

return exp(x)

生成AI: pythonらしく書き直してもらおう

プロンプト: 次のプログラムを、リスト内包表記、sum, map関数などを使って、pythonらしいプログラムに書き直してください。また、書き直した部分にはコメントを入れてください。

lsq_general.py:

```
def lsqfunc(i, x):  
    if i == 0:  
        return 1.0  
    elif i == 1:  
        return cos(2.0*x)  
    elif i == 2:  
        return sin(2.0*x)  
    elif i == 3:  
        return cos(x)  
    elif i == 4:  
        return sin(x)  
    elif i == 5:  
        return cos(3.0*x)  
    elif i == 6:  
        return sin(3.0*x)  
    elif i == 7:  
        return x  
    elif i == 8:  
        return x**2  
    elif i == 9:  
        return exp(x)
```

lsq_general_copilot.py:

```
def lsqfunc(i, x):  
    functions = [lambda x: 1.0,  
                 lambda x: cos(2.0 * x),  
                 lambda x: sin(2.0 * x),  
                 lambda x: cos(x),  
                 lambda x: sin(x),  
                 lambda x: cos(3.0 * x),  
                 lambda x: sin(3.0 * x),  
                 lambda x: x,  
                 lambda x: x ** 2,  
                 lambda x: exp(x)  
                ]  
    return functions[i](x)
```

生成AI: pythonらしく書き直してもらおう

プロンプト: 次のプログラムを、リスト内包表記、sum, map関数などを使って、pythonらしいプログラムに書き直してください。また、書き直した部分にはコメントを入れてください。

lsq_general.py:

```
def mlsq(xs, ys, m):
    n = len(xs)
    Si = np.empty([m, 1])
    Sij = np.empty([m, m])
    for l in range(0, m):
        v = 0.0
        for i in range(0, n):
            v += ys[i] * lsqfunc(l, xs[i])
        Si[l, 0] = v

    for j in range(0, m):
        for l in range(j, m):
            v = 0.0
            for i in range(0, n):
                v += lsqfunc(j, xs[i]) * lsqfunc(l, xs[i])
            Sij[j, l] = Sij[l, j] = v
```

```
print("Vector and Matrix:")
print("Si=")
print(Si)
print("Sij=")
print(Sij)
print("")
```

```
ci = np.linalg.inv(Sij) @ Si
ci = ci.transpose().tolist()
return ci[0]
```

lsq_general_copilot.py:

```
def mlsq(xs, ys, m):
    Si = np.array([[sum(ys[i] * lsqfunc(l, xs[i]) for i in range(len(xs)))) for l in range(m)])
    Sij = np.array([[sum(lsqfunc(j, xs[i]) * lsqfunc(l, xs[i]) for i in range(len(xs))) for l in range(m)] for j in range(m)])
```

```
print("Vector and Matrix:")
print("Si=")
print(Si)
print("Sij=")
print(Sij)
print("")
```

```
ci = np.linalg.inv(Sij) @ Si
return ci.flatten().tolist()
```

生成AI: pythonらしく書き直してもらおう

プロンプト: 次のプログラムを、リスト内包表記、sum, map関数などを使って、pythonらしいプログラムに書き直してください。また、書き直した部分にはコメントを入れてください。

lsq_general.py:

```
def main():
    --cut--
    df = pd.read_excel(infile, engine = 'openpyxl')
    labels = df.columns.to_list()
    x = df[labels[0]]
    y = df[labels[1]]
    ndata = len(x)
    minx = min(x)
    maxx = max(x)
    if minx < 0.0:
        xcal0 = minx * 1.5
    else:
        xcal0 = minx * 0.5
    if maxx < 0.0:
        xcal1 = maxx * 0.5
    else:
        xcal1 = maxx * 1.5
    --cut--

    xcal = [xcal0 + i * xcalstep for i in range(ncal)]
    ycal = []
    for i in range(ncal):
        _x = xcal[i]
        yl = 0.0
        for k in range(nfunc):
            yl += ci[k] * lsqfunc(k, _x)

    ycal.append(yl)
```

lsq_general_copilot.py:

```
def main():
    --cut--
    df = pd.read_excel(infile, engine='openpyxl')
    labels = df.columns.to_list()
    x, y = df[labels[0]], df[labels[1]]
    ndata = len(x)
    minx, maxx = min(x), max(x)
    xcal0 = minx * 1.5 if minx < 0.0 else minx * 0.5
    xcal1 = maxx * 0.5 if maxx < 0.0 else maxx * 1.5
    --cut--

    xcal = [xcal0 + i * xcalstep for i in range(ncal)]
    ycal = [sum(ci[k] * lsqfunc(k, xcal[i]) for k in range(nfunc)) for i in range(ncal)]
```

生成AI: pythonらしく書き直してもらおう

プロンプト: 次のプログラムを、リスト内包表記、sum, map関数などを使って、pythonらしいプログラムに書き直してください。また、書き直した部分にはコメントを入れてください。

lsq_general.py:

```
def main():
```

```
--cut--
```

```
    print("LSQ function")
```

```
    print("f(x) = {:.3g}".format(ci[0]), end = "")
```

```
    for i in range(1, nfunc):
```

```
        print(" + {:.3g} * {}".format(ci[i], flabel[i]), end = "")
```

```
    print("")
```

```
    xcal = [xcal0 + i * xcalstep for i in range(ncal)]
```

```
    ycal = []
```

```
    for i in range(ncal):
```

```
        _x = xcal[i]
```

```
        yl = 0.0
```

```
        for k in range(nfunc):
```

```
            yl += ci[k] * lsqfunc(k, _x)
```

```
    ycal.append(yl)
```

```
# Plot
```

```
--cut--
```

lsq_general_copilot.py:

```
def main():
```

```
--cut--
```

```
    print("LSQ function")
```

```
    print(f"f(x) = {ci[0]:6.3g}", end="")
```

```
    for i in range(1, nfunc):
```

```
        print(f" + {ci[i]:6.3g} * {flabel[i]}", end="")
```

```
    print("")
```

```
    xcal = [xcal0 + i * xcalstep for i in range(ncal)]
```

```
    ycal = [sum(ci[k] * lsqfunc(k, xcal[i]) for k in range(nfunc)) for i in range(ncal)]
```

```
    # Plot
```

```
--cut--
```