

チュートリアル資料:

<http://conf.msl.titech.ac.jp/D2MatE/2023Tutorial/tutorial2023-python-ChatGPT.html>

チュートリアル

学生と教員のための
pythonとChatGPT活用法

神谷利夫

東京工業大学

国際先駆研究機構 元素戦略MDX研究センター

チュートリアルの目的

学生・研究者視点

- 研究の解析・整理に必要な python プログラムを理解・作成できるようになる

教員・指導者視点

- 学生が提出してきた python プログラムを理解できるようになる

実際に使えるプログラム (最小二乗法) を完成させる

前提条件

- 非常に基本的なプログラミングの知識がある
 - 例えば、Fortran、Basic、Cなどの簡単なプログラムを作ったことがある
“プログラムを作れる”必要はない
 - 変数型 (整数型、実数型、配列)
制御 (if, for)、比較
関数 (引数、呼び出し、戻り値)
- 大規模言語モデル (LLM) を利用する (ChatGPT4/Copilot)
現在のプログラミングでは
検索、LLMを利用するのは必須スキル

チュートリアル構成

第1部 pythonプログラミングの基礎

1. ChatGPTを使って最小二乗法のプログラムを作る
2. ・ このプログラムをデバッグ、改善、
・ Fortran, BASIC, Cと比較しながら
pythonプログラムの文法、作り方を学ぶ。

第2部 学生が提出してきたpythonプログラムをどのように理解するか

1. “良いプログラム”への誘導: 私の数値解析講義の例
2. pythonプログラムを読みやすくする: リファクタリング

想定しているプログラミング言語とバージョン

Python: Version 3.8以降 <https://www.python.org/>
pythonはminor version up毎に機能を追加しているので注意
3.8では、型宣言、f文字列フォーマット (f'{var}')

などが使える
Major version upで互換性が維持されるかは不明

Fortran: Fortran90 WSL2/Linux、MinGW、macOS
Intel oneAPIコンパイラ
(<https://www.intel.com/content/www/us/en/developer/overview.html#gs.3bz9s2>)
FORTRAN77まで: 固定フォーマット
Fortran90以降: フリーフォーマット

C: C90/C99
C言語は後方互換性を重視しているので、大きな拡張・変更はない
C99では変数宣言位置の自由度やコメント文 (//) が拡張されている
C++は別物の言語、今回は無視する

BASIC: Visual Studio+Visual Basic <https://visualstudio.microsoft.com/ja/free-developer-offers/>
N88Basic <https://blog.oyasu.info/2022/04/14/8421/>

なぜpythonか

長所:

- 基本的に覚えることが少なく、**学習コストが低い**
- 科学計算、機械学習、可視化(グラフ化)**ライブラリが充実**
- 自分で作ったプログラムを、そのままライブラリとして**再利用**できる
- (プログラム作成 => 実行 => 修正) サイクルを回して使える状態まで**完成させる時間が短い**
- **読みやすい** (誰が書いても似たプログラムになる)
- 多言語 (C, C++, php, javascript etc) へ展開しやすい (C言語系の文法)

短所:

- 実行速度が**遅い**
- python**独特の文法**がある (インデントブロック、for文 など)

今日の私の環境

- **ChatGPT GPT3.5** (GPT4を使う設定になっていなかった)
- **Python**: python.org CPython **ver 3.11**
モジュール: numpy, scipy, openpyxl, pandas, jupyter
- **エディタ**: **cursor** (Visual Studio Code) + python拡張機能
- **その他プログラム言語**
Fortran, C: AlmaLinux9 + gcc, gfortran
Intel oneAPI compiler
Basic: N88互換Basic for Windows95
Visual Studio 2022

テキストエディタ

Visual Studio Code (VSCode): <https://code.visualstudio.com/>

現在最も広く使われている

多言語対応、マルチプラットフォーム (Win, macOS, Linux)

拡張機能 (python, ChatGPT, その他たくさん)

Cursor: VSCodeのフォーク、拡張: <https://cursor.sh/>

公式URL: <https://cursor.sh/>

参考URL: <https://weel.co.jp/media/cursor>

Python実行環境の種類

- コマンドライン

- Windowsでは **cmd.exe** を実行してコンソール (端末) ウィンドウを起動
- VSCodeでは **ターミナル** を使う

> **python** pythonスクリプト名 (‘>’ はcmd.exeのプロンプト)
で実行する

- **Pythonの対話モード:** N88Basicのようにコマンドの入力・確認をしながらプログラムを作れる
コマンドラインから

> **python**

> **ipython** # 履歴を保存するため、IPythonも利用

で対話モードに入る。抜けるときは `exit()` [ENTER]

- **Jupyter notebook:** N88Basicのようにコマンドの入力・確認をしながらプログラムを作れる
Webアプリ (python附属)、Google Colaboratory

VSCodeのJupyter拡張機能

注意: pythonにjupyterモジュール、VSCodeにpython拡張機能をインストールする必要がある

Python インタラクティブモード: pythonの場合

参考URL: http://conf.msl.titech.ac.jp/D2MatE/python/python-interactive_mode.html

N88Basicのように、コマンドの入力=>結果確認 を繰り返しながらプログラムの作製を進められる

注意: ファイルに保存できない

pythonを実行するとインタラクティブモードに入る

> python

Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>> a=3

>>> b=a2**

>>> print(b)

9

>>> b

9

>>>

Python インタラクティブモード: IPythonの場合

N88Basicのように、コマンドの入力⇒結果確認 を繰り返しながらプログラムの作成を進められる

IPythonを実行するとインタラクティブモードに入る

> ipython

In [1]: a=3

In [2]: b=a**2

In [3]: print(b)

9

In [4]: %save a.py # 上のやり取りをファイル a.py に保存

The following commands were written to file `a.py`:

a=3

b=3**2

print(b)

In [5]:

Python実行環境: VSCode+Jupyter

参考URL: <https://atmarkit.itmedia.co.jp/ait/articles/2108/06/news030.html>

- **Pythonにjupyterモジュールをインストール**

> pip install jupyter

- **Visual Code Studioでjupyter拡張機能をインストール**

1. VSCodeを起動

% code

2. [拡張機能]ビューから“python”を検索

python拡張機能をインストールすると、jupyter拡張機能も使える

- **Visual Code Studioでjupyter notebook (.ipynb) を作成する**

3. [エクスプローラー]ビューをで、適当なフォルダーを開く

4. [新規作成(New File)]メニューを選び、[File type]で “jupyter notebook”を選ぶ

*** Jupyter notebookファイル (.ipynb) をpythonプログラム (.py) に変換する**

> **jupyter nbconvert --to python hogehoge.ipynb**

最初に注意: ChatGPTは気を付けながら使いましょう

特徴

- 堂々と間違いを言う
- 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**

第1部 pythonプログラミングの基礎

1. ChatGPTを使って最小二乗法のプログラムを作る

- 2. ・ このプログラムをデバッグ、改善、
・ Fortran, BASIC, Cと比較しながら
pythonプログラムの文法、作り方を学ぶ。

第2部 学生が提出してきたpythonプログラムをどのように理解するか

- 1. “良いプログラム”への誘導: 私の数値解析講義の例
- 2. pythonプログラムを読みやすくする: リファクタリング

さて、問題です

Excelファイルinput.xlsx からデータを読み込んで、グラフを描くpythonプログラムを作ってください。

input.xlsxには、以下のデータが入っています

- ・ 2行目にはラベルとして'T', 'sigma', 'carrier density', 'mobility' が入っています
- ・ 2行目以降には、温度T、電気伝導度sigma、キャリア濃度carrier density, 移動度mobilityの数値データが入っています

プログラムは以下の動作をしてください

1. input.xlsxを読み込み、リスト変数に T, carrier densityの値をリスト変数 T, Nに読み込みます
2. Tの逆数を計算してリスト変数 Tinv に入れます
3. Carrier densityの自然対数を計算してリスト変数logNに入れます
4. T, N, Tinv, logNを画面に出力してください
5. TinvとlogNを一次式 $\log N = a + b * Tinv$ で最小二乗フィッティングをして、係数aとbを求めます
6. aとbを画面に出力してください
7. a, b, Tinvを使って、logNの計算値logNcalを計算してください
8. T, N, Tinv, logN, logNcalをExcelファイル fitting.xlsxに保存してください
9. Tinvに対してlogNとlogNcalをグラフにプロットしてください

入力Excelファイル

input.xlsx

T	sigma	carrier density	mobility
10	1	1	1
20	2	1.414214	4
30	3	1.732051	9
40	4	2	16
50	5	2.236068	25
60	6	2.44949	36
70	7	2.645751	49
80	8	2.828427	64
90	9	3	81
100	10	3.162278	100
110	11	3.316625	121
120	12	3.464102	144
130	13	3.605551	169
140	14	3.741657	196
150	15	3.872983	225
160	16	4	256

<= ラベル行

データ行

ChatGPTに聞いてみます

Excelファイルinput.xlsx からデータを読み込んで、グラフを描くpythonプログラムを作ってください。

Input.xlsxには、以下のデータが入っています

- ・ 2行目にはラベルとして'T', 'sigma', 'carrier density', 'mobility' が入っています
- ・ 2行目以降には、温度T、電気伝導度sigma、キャリア濃度carrier density, 移動度mobilityの数値データが入っています

プログラムは以下の動作をしてください

1. input.xlsxを読み込み、リスト変数に T, carrier densityの値をリスト変数 T, Nに読み込みます
2. Tの逆数を計算してリスト変数 Tinv に入れます
3. Carrier densityの自然対数を計算してリスト変数logNに入れます
4. T, N, Tinv, logNを画面に出力してください
5. TinvとlogNを一次式 $\log N = a + b * Tinv$ で最小二乗フィッティングをして、係数aとbを求めます
6. aとbを画面に出力してください
7. a, b, Tinvを使って、logNの計算値logNcalを計算してください
8. T, N, Tinv, logN, logNcalをExcelファイル fitting.xlsxに保存してください
9. Tinvに対してlogNとlogNcalをグラフにプロットしてください

このままChatGPTのプロンプトにいれます

ChatGPT4: 作成結果 (01a_fitting_gpt.py)

以降 行末までは
コメントとして
無視される

```
import openpyxl
import numpy as np
import matplotlib.pyplot as plt
```

1. Excelファイルからデータを読み込む

```
file_path = 'input.xlsx'
workbook = openpyxl.load_workbook(file_path)
sheet = workbook.active
```

```
T = []
N = []
```

```
for row in sheet.iter_rows(min_row=2, values_only=True):
    T.append(row[0])
    N.append(row[2])
```

2. Tの逆数を計算

```
Tinv = [1 / t for t in T]
```

3. Carrier densityの自然対数を計算

```
logN = [np.log(n) for n in N]
```

4. T, N, Tinv, logNを画面に出力

```
print("T:", T)
print("N:", N)
print("Tinv:", Tinv)
print("logN:", logN)
```

5. 最小二乗フィッティングで係数aとbを求める

```
a, b = np.polyfit(Tinv, logN, 1)
```

6. aとbを画面に出力

```
print("a:", a)
print("b:", b)
```

7. a, b, Tinvを使用してlogNcalを計算

```
logNcal = [a + b * t_inv for t_inv in Tinv]
```

8. 結果を新しいExcelファイルに保存

```
output_workbook = openpyxl.Workbook()
output_sheet = output_workbook.active
output_sheet.append(["T", "N", "Tinv", "logN", "logNcal"])
```

```
for t, n, t_inv, log_n, log_n_cal in zip(T, N, Tinv, logN, logNcal):
    output_sheet.append([t, n, t_inv, log_n, log_n_cal])
```

```
output_workbook.save("fitting.xlsx")
```

9. グラフをプロット

```
plt.scatter(Tinv, logN, label="Data")
plt.plot(Tinv, logNcal, color='red', label="Fitted Curve")
plt.xlabel("1/T")
plt.ylabel("log(N)")
plt.legend()
plt.show()
```

ChatGPTのプロンプトのコツ

- やってほしいことを明確に書く
- 前提や定義を明確に書く
- 条件などは箇条書きにする
箇条書きの項目が複数にわたる場合、
異なる行頭文字で区別する

その他の例:

- 質問者と回答者の立場(学生、先生、知識レベルなど)を定義し、
質疑形式で回答してもらう

第1部 pythonプログラミングの基礎

1. ChatGPTを使って最小二乗法のプログラムを作る
2. ・このプログラムをデバッグ、改善、
・ Fortran, BASIC, Cと比較しながら
pythonプログラムの文法、作り方を学ぶ。

第2部 学生が提出してきたpythonプログラムをどのように理解するか

1. “良いプログラム”への誘導: 私の数値解析講義の例
2. pythonプログラムを読みやすくする: リファクタリング

ChatGPTが作ったプログラムを見直す (01a_fitting_gpt.py)

```
import openpyxl
import numpy as np
import matplotlib.pyplot as plt
```

1. Excelファイルからデータを読み込む

```
file_path = 'input.xlsx'
workbook = openpyxl.load_workbook(file_path)
sheet = workbook.active
```

```
T = []
N = []
```

```
for row in sheet.iter_rows(min_row=2, values_only=True):
    T.append(row[0])
    N.append(row[2])
```

2. Tの逆数を計算

```
Tinv = [1 / t for t in T]
```

3. Carrier densityの自然対数を計算

```
logN = [np.log(n) for n in N]
```

4. T, N, Tinv, logNを画面に出力

```
print("T:", T)
print("N:", N)
print("Tinv:", Tinv)
print("logN:", logN)
```

5. 最小二乗フィッティングで係数aとbを求める

```
a, b = np.polyfit(Tinv, logN, 1)
```

6. aとbを画面に出力

```
print("a:", a)
print("b:", b)
```

7. a, b, Tinvを使用してlogNcalを計算

```
logNcal = [a + b * t_inv for t_inv in Tinv]
```

8. 結果を新しいExcelファイルに保存

```
output_workbook = openpyxl.Workbook()
output_sheet = output_workbook.active
output_sheet.append(["T", "N", "Tinv", "logN", "logNcal"])
```

```
for t, n, t_inv, log_n, log_n_cal in zip(T, N, Tinv, logN, logNcal):
    output_sheet.append([t, n, t_inv, log_n, log_n_cal])
```

```
output_workbook.save("fitting.xlsx")
```

9. グラフをプロット

```
plt.scatter(Tinv, logN, label="Data")
plt.plot(Tinv, logNcal, color='red', label="Fitted Curve")
plt.xlabel("1/T")
plt.ylabel("log(N)")
plt.legend()
plt.show()
```

条件1つに対して
数行のプログラムを
つなげているだけ

どうしたらプログラムが作れるか？

？？「プログラミングには適性がある」？？

- 設計が悪いプログラムならそうかもしれない
フローチャートを書いた時に分岐ループが絡み合うスパゲッティ
- 箇条書きで要件を書ければ、誰にでもプログラムを作れる

1. すべき機能を箇条書きで書く
2. ほとんどの機能は30行以下のプログラムで書ける
3. それぞれの機能を 変数でつなぐ
ことで、ほとんどのプログラムは作れる

必要な機能を 30 行以下のプログラムで作ってみる

何が必要か？

1. パラメータ指定: キーボード入力 `input()`
コマンドライン引数 `sys.argv`
2. 画面出力: 確認 `print()`
3. ファイルの読み書き (Excelファイル、テキストファイル、CSVファイル):
`openpyxl, pandas, csv, open()`
4. 計算: 算術演算子、数学関数 `math/numpy`
5. 制御: 繰り返し構文、条件分岐構文 + 比較構文 `for, if ... elif ... else`
6. 複雑な解析: ライブラリに頼る 最小二乗法 `numpy.polyfit()`
7. グラフ描画 : `matplotlib`

pythonの特徴: インタプリタ型言語

- **インタプリタ型言語** (python, perl, php, javascript など)

プログラムを書いたら即実行できる:

実行時には*real time* でプログラムが解釈されて**機械語**に翻訳されて実行される
多くの場合、最初にプログラム全体を“**中間言語**”に変換して実行される

長所: 開発が楽・高速

Basicのように、実行を確認しながらプログラムを作成できる
(インタラクティブモード、Jupyter notebook)

短所: 実行速度が遅い

参考: **コンパイル型言語** (Fortran、C、C++、C#、Rust、Goなど)

プログラムを書いたら、

- (i) コンパイルして (機械語に翻訳して) 実行可能ファイルを作成し、
- (ii) 実行する

短所: 開発に手間、時間がかかる

実行が早い

プログラム言語の比較

	python	C	Fortran	Basic
実行順序	<ul style="list-style-type: none">•ソースコード (メインファイル) の先頭から順番に実行•関数はどこにでも書けるが、関数定義はスキップされる•ソースコードの最後で終了	<ul style="list-style-type: none">•global変数はコンパイル時に初期化される•関数の集まりとしてプログラムを書く•main()関数 (特別扱い) の先頭から実行•main()関数から抜けたら終了	<ul style="list-style-type: none">•Program文から実行•End program文で終了•関数、サブルーチンはProgramブロックの外で定義	<ul style="list-style-type: none">•ソースコードの先頭から順番に実行•ソースコードの最後で終了•関数、サブルーチンはProgramブロックの外で定義
大文字・小文字	区別する	区別する	区別しない	区別しない
ブロック	行末の: で新しいブロックを開始 インデントの数によってブロックを区別	{ } で囲う	endで終わる	endで終わる
関数、サブルーチン	<ul style="list-style-type: none">•全て関数•def func_name(args) ... インデント位置を変えて関数を開始、 インデント位置が戻ったところで関数終了	<ul style="list-style-type: none">•全て関数•戻り値型宣言 func_name(args) { ... return ret_val }	<ul style="list-style-type: none">•サブルーチン subroutine sub_name(args) ... end subroutine•関数 戻り値型宣言 function func_name(args) ... return ret_val end function	<ul style="list-style-type: none">•サブルーチン sub sub_name(args) ... end sub•関数 Function func_name(args) As 戻り値型 ... End Function

Pythonのコメント文

	python	C	Fortran	Basic
インライン コメント (行末まで)	# 以降	C99以降 // 以降	Fortran90以降 ! 以降 (Fortran77までの固定フォーマットでは、行頭にCがあると1行コメント)	Visual Basic , 以降
複数行 コメント	docstringを流用 """ あるいは """" """"	/* */		

pythonの特徴: 変数名、関数名、モジュール名、予約語

大文字と小文字を区別する

- 変数 `a` と変数 `A` は別の変数
- `print("Hello")` は正しいが、
`PRINT("Hello")` は関数 `PRINT()` が見つからずエラーになる

pythonスクリプトの実行順序

コマンド python に続けて pythonスクリプトファイルパス (メインファイル) を与えて実行
> **python test.py**

- メインファイル **test.py** の先頭から順番に実行
- メインファイル **test.py** の最後で終了
exit() 関数を呼び出すと途中で終了できる
- 関数定義はどこにでも書ける
(関数内、for/if などのブロック内でも関数を書ける)
実行時には関数定義は読み飛ばされる
- 関数呼び出しを見つけたら関数内部へジャンプ、実行
return をみつける or 関数ブロック終了時に呼び出し元の次の行を実行

import文

- プログラム中に **import文** `import lib_name` を見つけたら、
lib_name.py を探して実行
ライブラリの検索パスは
 - 環境変数 PYTHONPATH
 - `sys.path` (参考URL: <https://note.nkmk.me/python-import-module-search-path/>)
で設定できる
- **lib_name.py** は **lib_name オブジェクト**としてアクセスできる
`lib_nama.py`の関数 `f()`、グローバル変数 `var` は
`lib_nama.f()`, `lib_nama.var` として使える

注意: ライブラリ名には ‘-’ は使えない

今回配布しているファイル名は ‘_’ を使っている

pythonの特徴: 基本機能は最小限 必要な機能はライブラリを使う

- 基本的な機能 (組み込み構文、組み込み関数) は最小限

組み込み変数型 : int, float, str, list (その他 tuple, dict, set, map, zipなど)
オブジェクト定義 : class
定数 : None, True, False
変数型変換 : int(), float(), str()
制御構文 : if, for, while
比較構文 : ==, !=, <, <=, >, >=, is, not
演算子 : = (代入演算子), +, -, *, /, **** (べき乗)**,
演算子 (整数) : % (剰余), **// (整数同士を割り算した整数部分)**
その他 : next(), isinstance(), setattr(), getattr()

注意: べき乗 a^b を a^b とする言語もある (MS-Excel) が、

pythonでは **$a^{**}b$**

**pythonでは、/ による計算では整数型変数も実数型変数に変換される
整数同士の割り算をしたい場合は、// を使う**

Pythonに特有の規則: インデントによるブロック定義

- **ブロックはインデント (行頭の空白文字数) の数で決まる**

注意: TABを使わないように注意する

TABキーを一定数の空白文字に置き換えるエディタを使う

以下のコードで ‘.’ はスペースの意味です

```
for i in [1, 2, 3, 4, 5]:
```

0文字インデントから開始

```
....x = 2.0 + i * 0.1
```

4文字インデント。Forブロック開始

空行は無視される

```
....print(f'i={i} x={x}')
```

空行は無視される

```
for s in ["a", "b", "c", "d", "e"]:
```

0文字インデントに戻る。Forブロック終了

```
..line = 'string is ' + s
```

2文字インデント。Forブロック開始

```
..print(f'line={line}')
```

```
exit()
```

0文字インデントに戻る。Forブロック終了

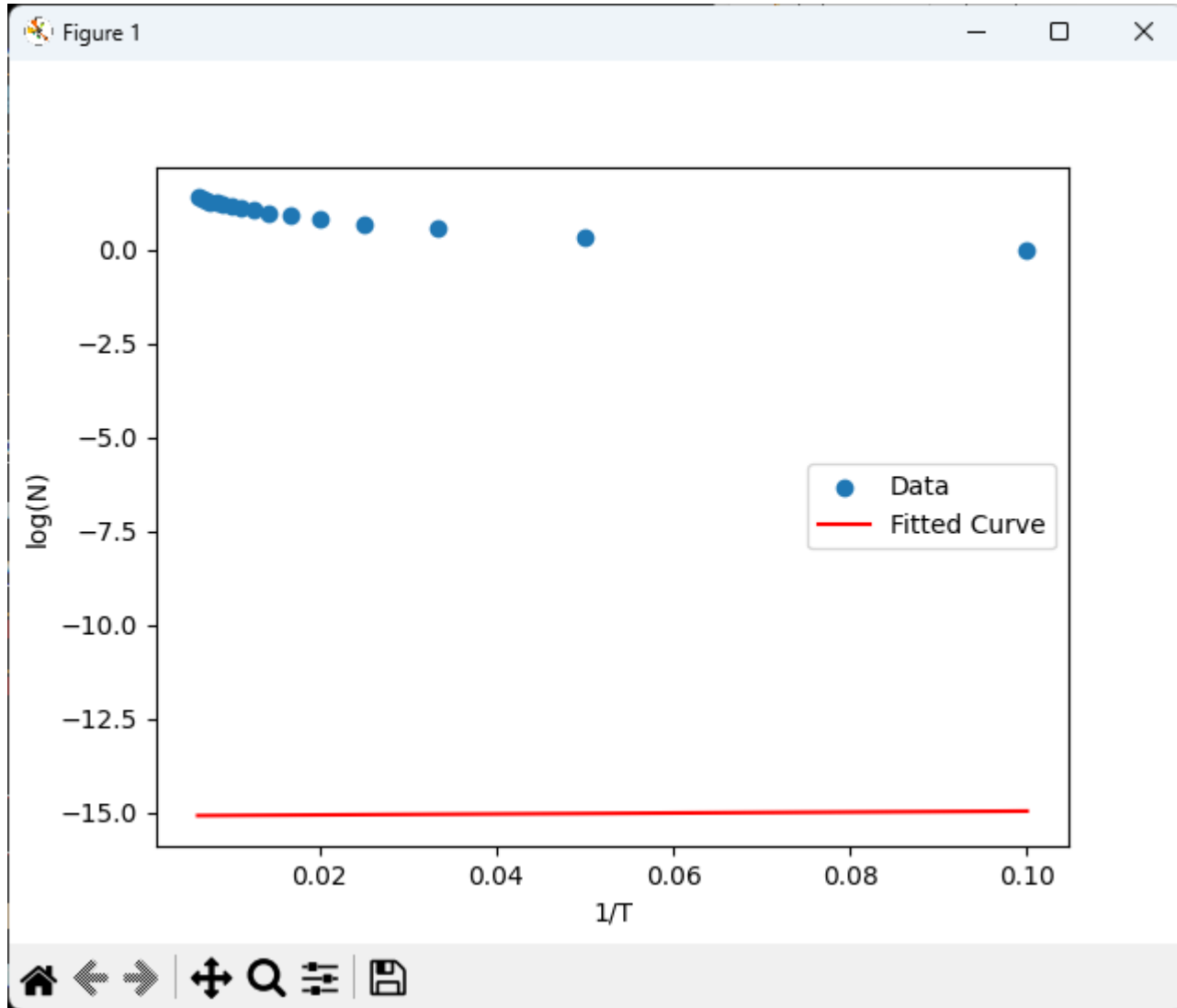
第1部 pythonプログラミングの基礎

1. ChatGPTを使って最小二乗法のプログラムを作る
2. ・ このプログラムをデバッグ、改善、
 - ・ Fortran, BASIC, Cと比較しながらpythonプログラムの文法、作り方を学ぶ

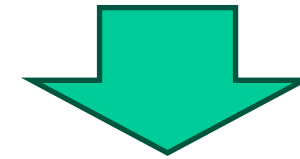
ChatGPTが作ったプログラムのバグを調べる

- ・ 入力データは正しいか
入力データは必ず画面に**リポート出力**
- ・ 計算結果は正しいか
計算結果も **画面出力**
- ・ 解析 (最小二乗法) 結果は正しいか
- ・ グラフ描画は正しいか
ファイル (特にExcelファイル) に**出力**すると確認しやすい

ChatGPT4: 実行結果 (01a_fitting_gpt.py)



入力データ Data と、
フィッティング結果 Fitted Curve が
全然合っていない



この後のパターン

- ChatGPTと対話しながら修正
- 自力で修正

いずれにしても、ある程度の
pythonに関する知識がないといけない

入力データの確認 (01a_fitting_gpt.py)

画面出力: 入力データが正しいかどうかを確認する

T: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160]

N: [1, 1.4142135623730951, 1.7320508075688772, 2, 2.23606797749979, 2.449489742783178, 2.6457513110645907, 2.8284271247461903, 3, 3.1622776601683795, 3.3166247903554, 3.4641016151377544, 3.605551275463989, 3.7416573867739413, 3.872983346207417, 4]

Tinv: [0.1, 0.05, 0.03333333333333333, 0.025, 0.02, 0.016666666666666666, 0.014285714285714285, 0.0125, 0.011111111111111112, 0.01, 0.00909090909090909, 0.008333333333333333, 0.007692307692307693, 0.007142857142857143, 0.006666666666666667, 0.00625]

logN: [0.0, 0.3465735902799727, 0.5493061443340548, 0.6931471805599453, 0.8047189562170503, 0.8958797346140274, 0.9729550745276567, 1.039720770839918, 1.0986122886681098, 1.151292546497023, 1.1989476363991853, 1.2424533248940002, 1.2824746787307684, 1.3195286648076292, 1.354025100551105, 1.3862943611198906]

a: -15.079956524131369

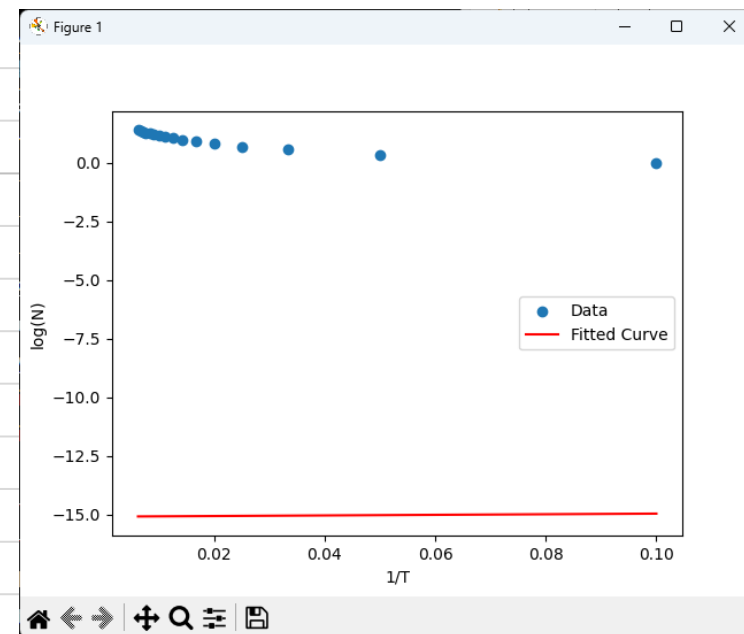
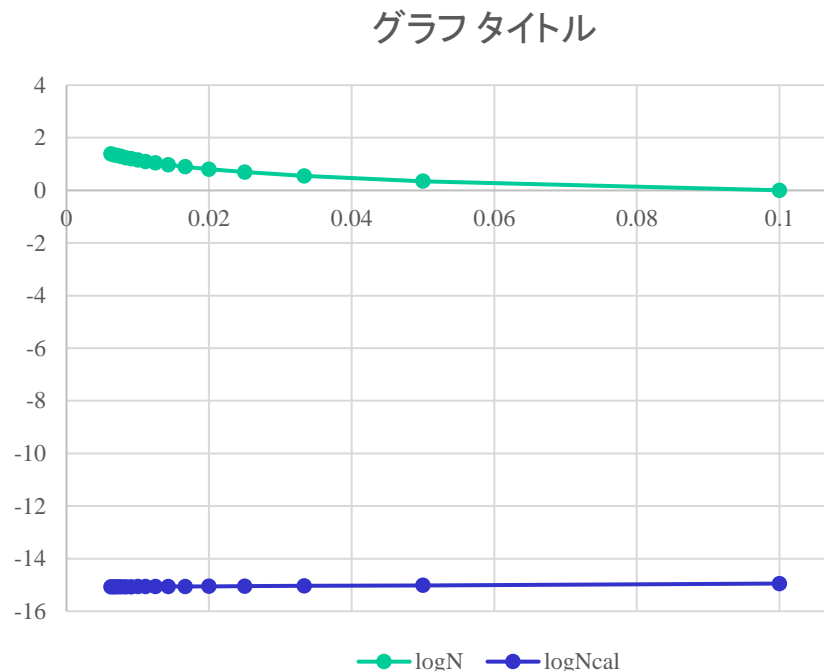
b: 1.277128417301044

T	sigma	carrier density	mobility
10	1	1	1
20	2	1.414214	4
30	3	1.732051	9
40	4	2	16
50	5	2.236068	25
60	6	2.44949	36
70	7	2.645751	49
80	8	2.828427	64
90	9	3	81
100	10	3.162278	100
110	11	3.316625	121
120	12	3.464102	144
130	13	3.605551	169
140	14	3.741657	196
150	15	3.872983	225
160	16	4	256

計算結果の確認 (01a_fitting_gpt.py)

フィッティング結果の出力ファイル fitting.xls: 計算 ($1/T$, $\log(N)$) は問題ないかを確認

T	N	Tinv	logN	logNcal
10	1	0.1	0	-14.9522
20	1.414214	0.05	0.346574	-15.0161
30	1.732051	0.033333	0.549306	-15.0374
40	2	0.025	0.693147	-15.048
50	2.236068	0.02	0.804719	-15.0544
60	2.44949	0.016667	0.89588	-15.0587
70	2.645751	0.014286	0.972955	-15.0617
80	2.828427	0.0125	1.039721	-15.064
90	3	0.011111	1.098612	-15.0658
100	3.162278	0.01	1.151293	-15.0672
110	3.316625	0.009091	1.198948	-15.0683
120	3.464102	0.008333	1.242453	-15.0693
130	3.605551	0.007692	1.282475	-15.0701
140	3.741657	0.007143	1.319529	-15.0708
150	3.872983	0.006667	1.354025	-15.0714
160	4	0.00625	1.386294	-15.072



$1/T$ 、 $\log(N)$ の計算はOK

グラフもpythonプログラムの通り

問題は最小二乗法の実行部分にある

最小二乗法の部分 (01a_fitting_gpt.py)

```
import openpyxl
import numpy as np
import matplotlib.pyplot as plt

# 1. Excelファイルからデータを読み込む
file_path = 'input.xlsx'
workbook = openpyxl.load_workbook(file_path)
sheet = workbook.active

T = []
N = []

for row in sheet.iter_rows(min_row=2, values_only=True):
    T.append(row[0])
    N.append(row[2])

# 2. Tの逆数を計算
Tinv = [1 / t for t in T]

# 3. Carrier densityの自然対数を計算
logN = [np.log(n) for n in N]

# 4. T, N, Tinv, logNを画面に出力
print("T:", T)
print("N:", N)
print("Tinv:", Tinv)
print("logN:", logN)
```

5. 最小二乗フィッティングで係数aとbを求める
a, b = np.polyfit(Tinv, logN, 1)

6. aとbを画面に出力

```
print("a:", a)
```

```
print("b:", b)
```

7. a, b, Tinvを使用してlogNcalを計算

```
logNcal = [a + b * t_inv for t_inv in Tinv]
```

8. 結果を新しいExcelファイルに保存

```
output_workbook = openpyxl.Workbook()
```

```
output_sheet = output_workbook.active
```

```
output_sheet.append(["T", "N", "Tinv", "logN", "logNcal"])
```

for t, n, t_inv, log_n, log_n_cal in zip(T, N, Tinv, logN, logNcal):

```
    output_sheet.append([t, n, t_inv, log_n, log_n_cal])
```

```
output_workbook.save("fitting.xlsx")
```

9. グラフをプロット

```
plt.scatter(Tinv, logN, label="Data")
```

```
plt.plot(Tinv, logNcal, color='red', label="Fitted Curve")
```

```
plt.xlabel("1/T")
```

```
plt.ylabel("log(N)")
```

```
plt.legend()
```

```
plt.show()
```

pythonの特徴: ライブラリ

- 必要な機能は ライブラリを読み込んで利用する

長所: 複雑なプログラムを自分で作る必要がない

“車輪の再発明はしない”

Cなどで書かれたライブラリが多く、高速

典型例: numpyはC/Fortran互換の配列 ndarray() を使い、
LAPACK、Intel MKLなどを直接呼び出す

短所: いちいち **import**文 でライブラリを読み込む必要がある

欲しい機能がどのライブラリにあるかわからない、
どう書いたらいいかわからない

とにかく 検索 や ChatGPT で調べる

ライブラリ、モジュール、パッケージ

ライブラリ: 関数、クラス、モジュール、パッケージなど、
他のプログラムから呼び出されるものの総称

モジュール: 関数やクラスをまとめたもの
拡張子が .py のpythonファイルのこと

パッケージ: 複数のモジュールをまとめたフォルダ

本チュートリアルでは明確な区別をしない

ライブラリの使い方とオブジェクト

- pythonはオブジェクト指向言語

オブジェクト: 変数に、関連する変数 (メンバ変数: attribute/property) と
関数 (メンバ関数: attribute/method) を結び付けたもの

class: オブジェクトの定義。class構文を使って定義する

インスタンス: classを変数として実体化させたもの

例: 猫クラスの定義

```
class Cat():  
    def __init__(self, x0, y0):  
        self.x = x0  
        self.y = y0  
  
    def move(self, dx, dy):  
        self.x += dx  
        self.y += dy  
  
    def where(self):  
        print(f'current position: ({self.x}, {self.y})')
```

例: 猫クラスを使用して、猫を移動して位置を表示する

```
mycat = Cat(3.0, 5.0) # Catクラスのインスタンスを作る  
mycat.move(0.5, -1.0) # メンバ関数.move()の呼び出し  
mycat.where()
```

出力結果:

current position: (3.5, 4.0)

オブジェクトの変数や関数を呼び出す方法:

・ でつなげる

インスタンス変数名.attribute名

pythonのすべての変数、関数はオブジェクト

- **文字列変数** (ここではsとする) の代表的な method

```
s = s.strip()      # 文字列 s の前後の空白文字を削除して返す  
s_list = s.split() # 文字列 s をスペース ' ' で分割した結果をリストで返す
```

- ファイルの操作: open()関数の**返り値**もオブジェクト

```
fp = open('input.txt', 'r')      # ファイル input.txtをオープンし、  
                                # ファイルハンドルを返す  
  
lines = fp.readlines()          # ファイルを、行ごとにリスト変数として返す  
fp.close()                      # ファイルハンドルを閉じてファイル処理を終了
```

- ファイルの存在確認: **osモジュールの os.path モジュール**のisfile()関数を使う

```
import os  
print("Does file exist? ", os.path.isfile('input.txt'))
```

- **Pythonスクリプト** (ライブラリ) 自体がオブジェクト

```
mylib.pyというpythonスクリプトを import mylib で読み込む  
=> mylib.py の関数 f(x) は mylib.f(x) で呼び出せる
```


よく使うpythonライブラリ

システム関係など

- **sys**
sys.argvで起動時引数を取得
- **csv**
CSVファイルの読み書き
- **openpyxl**
Excelファイルの読み書き
- **Pandas**
Excelファイルの読み書き
scikit-learnと組み合わせて使われる

グラフ関係

- **matplotlib**
グラフの描画

科学計算関係

- **math**
基本的な数学関数。sin, cos, tan, asin, log, exp など
- **numpy**
配列、行列を含む数値計算などにはほぼ標準 (高速)。
numpy.ndarray からリストへは、ほとんど場合に自動的に型変換してくれる
線形代数関数 (逆行列、一次連立方程式の解など)
- **scipy**
numpyの機能に加え、信号処理 (FFT) など多様な数学関数がある
- **Scikit-learn**
機械学習関係

多項式線形最小二乗法: numpy ライブラリの polyfit() 関数

ChatGPTへの質問: $\log N = a + b * T_{\text{inv}}$ にフィッティング

```
# numpyモジュールを 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モジュールから polyfit を import
```

```
from numpy import polyfit
```

```
# polyfit() だけで呼び出せる
```

```
a, b = polyfit(Tinv, logN, 1)
```

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

修正 (01b_fitting_gpt_rev.py)

```
import openpyxl
import numpy as np
import matplotlib.pyplot as plt

# 1. Excelファイルからデータを読み込む
file_path = 'input.xlsx'
workbook = openpyxl.load_workbook(file_path)
sheet = workbook.active

T = []
N = []

for row in sheet.iter_rows(min_row=2, values_only=True):
    T.append(row[0])
    N.append(row[2])

# 2. Tの逆数を計算
Tinv = [1 / t for t in T]

# 3. Carrier densityの自然対数を計算
logN = [np.log(n) for n in N]

# 4. T, N, Tinv, logNを画面に出力
print("T:", T)
print("N:", N)
print("Tinv:", Tinv)
print("logN:", logN)
```

5. 最小二乗フィッティングで係数aとbを求める
b, a = np.polyfit(Tinv, logN, 1)

6. aとbを画面に出力

```
print("a:", a)
```

```
print("b:", b)
```

7. a, b, Tinvを使用してlogNcalを計算

```
logNcal = [a + b * t_inv for t_inv in Tinv]
```

8. 結果を新しいExcelファイルに保存

```
output_workbook = openpyxl.Workbook()
```

```
output_sheet = output_workbook.active
```

```
output_sheet.append(["T", "N", "Tinv", "logN", "logNcal"])
```

for t, n, t_inv, log_n, log_n_cal in zip(T, N, Tinv, logN, logNcal):

```
    output_sheet.append([t, n, t_inv, log_n, log_n_cal])
```

```
output_workbook.save("fitting.xlsx")
```

9. グラフをプロット

```
plt.scatter(Tinv, logN, label="Data")
```

```
plt.plot(Tinv, logNcal, color='red', label="Fitted Curve")
```

```
plt.xlabel("1/T")
```

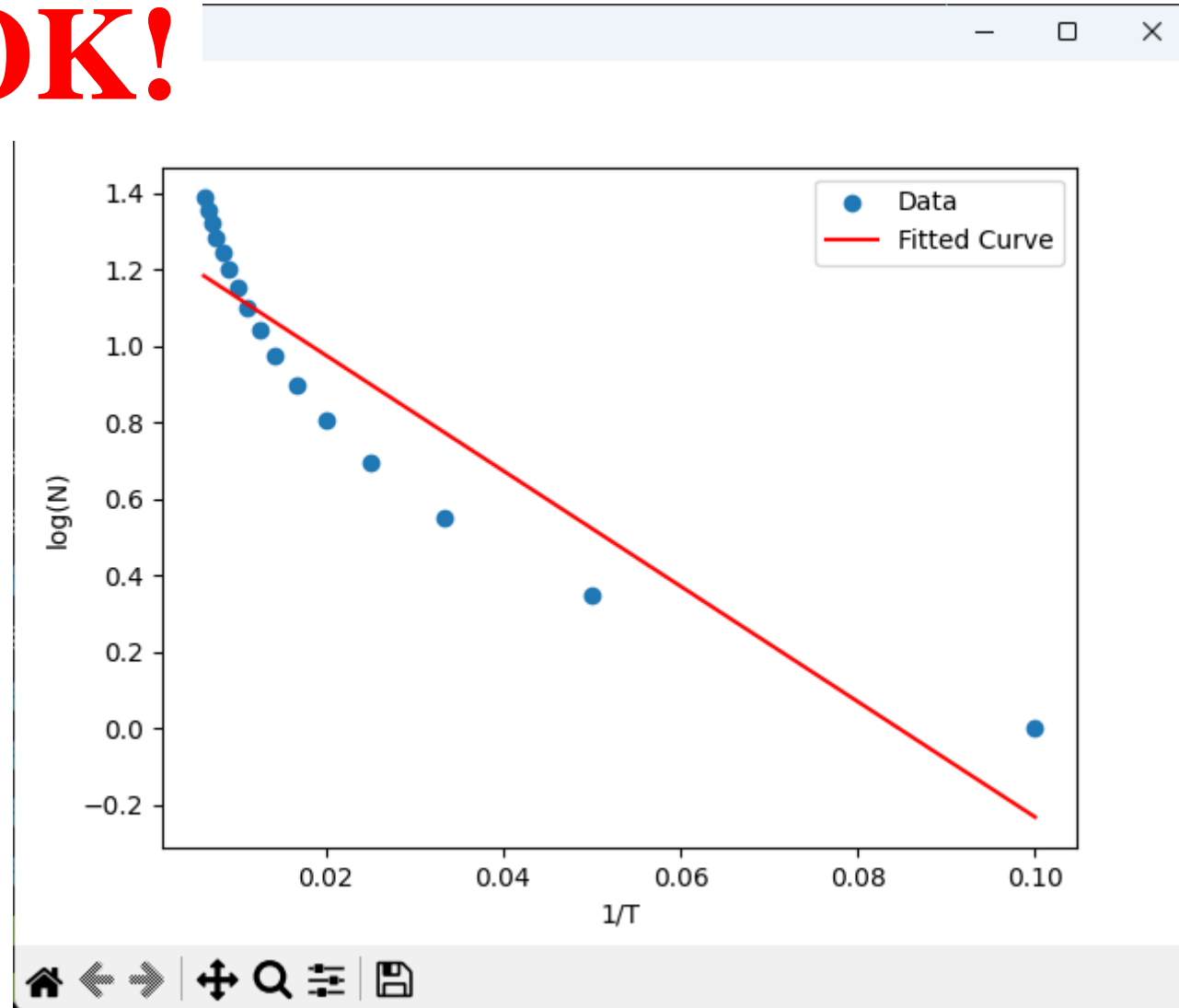
```
plt.ylabel("log(N)")
```

```
plt.legend()
```

```
plt.show()
```

ChatGPT4: 修正版実行結果 (01b_fitting_gpt_rev.py)

OK!



第1部 pythonプログラミングの基礎

1. ChatGPTを使って最小二乗法のプログラムを作る
2. ・このプログラムをデバッグ、改善、
・Fortran, BASIC, Cと比較しながら
pythonプログラムの文法、作り方を学ぶ。

第2部 学生が提出してきたpythonプログラムをどのように理解するか

1. “良いプログラム”への誘導: 私の数値解析講義の例
2. pythonプログラムを読みやすくする: リファクタリング

Pythonの文法: 変数作成

- Pythonに**型定義はない** (python3.5以降は type hint として型宣言できる)
- **変数に値を代入すると、変数を作成したことになる**
- **代入した値により変数型が決まる**

a = 3	aは整数型 (int)、 3で初期化される
a = 3.0	aは実数型 (float)、3.0で初期化される
a = "3"	aは文字列型 (str)、"3"で初期化される
a = [3, 3.0, "3"]	aは配列 (list型)、 要素として 3 (int), 3.0 (float) "3" (str) をもつ

Pythonでは自然な形で整数型と実数型を比較できる場合が多いが、**実数型として使う変数は、整数になる場合でも実数型で代入する方がトラブルが少ない**

変数作成 (01c_fitting_gpt_rev2.py)

- プログラム全体で使う変数は、
プログラム先頭で宣言したほうがいい (グローバル変数)
変数の修正が一か所で済む
どのような条件でプログラムを実行するかが一目でわかる

```
input_path = 'input.xlsx'
```

- 捨て変数 (ループ変数、一次変数) は、
使用直前に宣言したほうがいい
関数内で宣言 (ローカル変数) し、関数が終了したら破棄されるようにする
想定していない変数をスペルミスなどで使ってしまう危険を少しでも減らせる

グローバル (大域) 変数とローカル (局所) 変数

参考URL: <http://conf.msl.titech.ac.jp/Lecture/python/python-tips.html>

- 関数外で代入するとグローバル変数 : 同じスクリプト内で参照できる
- 関数内で代入するとローカル変数 : 関数を抜けたら破棄される

大きなプログラムでは、大域変数は極力使わない方がいい

多くの大域変数を使う場合、class宣言をしてオブジェクトの attribute として変数を宣言する

【注意】関数内でグローバル変数と同じ名前の変数に代入すると、その名前のローカル変数が生成される。

関数内でグローバル変数へ代入する場合は、**global 宣言**を行う

global1.py

a = 1

def f():

a = 2 局所変数を生成、大域変数を変えない

 print("in func:", a)

print("outer func1:", a)

f()

print("outer func2:", a)

実行結果

outer func1: 1

大域変数

in func: 2

局所変数

outer func2: 1

大域変数

global2.py

a = 1

def f():

global a

a = 2

大域変数に代入

 print("in func:", a)

print("outer func1:", a)

f()

print("outer func2:", a)

実行結果

outer func1: 1

大域変数

in func: 2

大域変数

outer func2: 2

大域変数

配列の使い方

2種類の配列型がある

- ・ リスト型 (list) : 要素の書き換えが可能
- ・ タプル型 (tuple): 要素の書き換えはできない
- ・ 要素には異なる変数型を入れられる
- ・ 要素には配列を入れられる => 多次元配列を作れる

配列 (リスト) を作る

```
a = []
```

```
a.append(1.0)
```

```
print(a[0])
```

```
len(a)
```

list型変数は **[]** でくる

要素を追加

1番目の要素には a[0], i番目の要素は a[i-1]で参照できる

配列の要素数を返す

```
a = (1.0, [1.0, 3.0], "a")    # list型変数は () でくる
```

tupleの要素は変更できないので、初期化時にすべての要素を与える

tupleの要素は変更できないので、.append()は使えない

C, Fortranと同じように配列を扱う

numpy.ndarray型を使います。

注意: 要素の型は同じでなければいけません

- 要素の初期化をしない場合: `: .empty()`
- 値 0.0 で初期化する場合: `: .zeros()`
- list/tuple を変換する場合: `: .array()`

Python:

```
import numpy as np
```

```
a = np.empty([3,3])
```

```
for i in range(3):
```

```
    for j in range(3):
```

```
        a[i][j] = i * j
```

C:

```
int a[3][3];
```

```
int i, j;
```

```
for(i = 0; i < 3; i++) {
```

```
    for(j = 0; j < 3; j++) {
```

```
        a[i][j] = i * j;
```

```
    }
```

```
}
```

Fortran:

```
real, dimension(3,3) :: a
```

```
integer :: i, j
```

```
do i = 1, 3
```

```
    do j = 1, 3
```

```
        a(i,j) = (i-1)*(j-1)
```

```
    end do
```

```
end do
```

よく変更する変数の変更: ハードコーディングは避ける

(01a_fitting_gpt.py)

```
input_path = 'input.xlsx'
```

ハードコーディング:

プログラム中に定数以外のパラメータを書き込む

読み込むファイルを変えるたびにプログラムを書きなおす必要がある

よく変更する変数を変更する一般的な方法:

- コマンドライン引数
- キーボード入力
- 設定ファイルを読み込む

よく変更する変数の変更: コマンドライン引数 `sys.argv` (01b_fitting_gpt-rev2.py)

pythonプログラムを実行するとき、コマンドライン引数を渡せる

> Python `hoge hoge.py arg1 arg2`

`arg1, arg2`は、`sys`モジュールの `sys.argv`リスト変数 を使って取り出せる

注意: `sys.argv[0]` にはpythonスクリプトのパスが入る
コマンドライン引数は `sys.argv[1]` 以降に入っている

コマンドライン引数を与えられていれば、`len(sys.argv)` は2以上

```
import sys
```

```
input_path = 'input.xlsx'    # デフォルト値を設定する
```

```
if len(sys.argv) > 1:
```

```
    input_path = sys.argv[1] # 1つ目の引数を sys.argv[1]で受け取り、input_pathを置き換える
```

条件分岐 (if ... else if ... else) と比較構文

	python	C	Fortran	Basic
If文	if 比較構文1: ... elif 比較構文2: ... else: ...	if(比較構文1) { ... } else if(比較構文2) { ... } else { ... }	if(条件構文1) then ... else if(条件構文2) then ... else ... end if	If 条件構文1 then ... elseif 条件構文2 then ... else ... end if
数値の比較	== , !=, <, <=, >, >=, a < b < c	== , !=, <, <=, >, >=	== (.eq.), != (.ne.) < (.lt.), <= (.le.), > (.gt.), >= (.le.)	=, !=, <, <=, >, >=,
文字列の比較	== , !=, <, <=, >, >=, a < b < c	strcmp() などの関数を使う		=, !=, <, <=, >, >=,
定数、変数が 同一であるか (値の比較ではなく、 実体の比較)	is, is not	ポインタを比較する *a == *b, *a != *b		

よく変更する変数の変更: キーボード入力 input()

```
ret = input("Input>>")
```

Input>>” を画面に表示し、キーボード入力を待つ
ENTERを押したら、入力した値を返す

retには 改行文字 (¥n) を含めた文字列が変える

=> 改行文字を削除するため、.strip() を使う

```
ret = ret.strip()    # retから前後の空白、改行文字を削除
```

```
input_path = 'input.xlsx'          # デフォルト値を設定する
```

```
str_key = input("Input file name [default: input.xlsx]>>")
```

```
str_key = str_key.strip()
```

```
if str_key != '':
```

```
    input_path = str_key # 入力されたのが 空文字 でなければ input_pathを置き換える  
                        # 空文字の場合は、input_pathはデフォルト値のまま
```


Excelファイルの読み込み openpyxlモジュール

```
import openpyxl
```

```
# 1. Excelファイルからデータを読み込む
```

```
workbook = openpyxl.load_workbook(input_path) # Excelファイル input_path を読み込む
```

```
sheet = workbook.active # 現在のワークシートを変数 sheet に代入
```

```
T = [] # Excelファイル内の値を保存するリスト変数を
```

```
N = [] # 空リスト [] で初期化
```

```
for row in sheet.iter_rows(min_row=2, values_only=True):
```

```
    # 現在シートのmin_row行以降の値を、リスト変数 row で受け取る
```

```
    T.append(row[0]) #リスト変数への追加は .append()関数を使う
```

```
    N.append(row[2])
```

```
# 現在シートの行を読み込み終わったら、.iter_rows() は None を返して for文から抜ける: iterator
```

繰り返し構文 (ループ)

- ループ変数は整数型や実数型
- ループ開始時にループ変数を初期化
- ループブロック終了時に変数を更新して条件判断をする
条件が False になったらループを抜ける

Fortranの場合: i を 1 ずつ増やしながら、

i が 10 以下の場合にdoブロックを実行

Cの場合: for文内で定義

for(初期化 ; ループブロック続行条件 ; ループ変数更新)

Fortranの繰り返し

```
do I = 1, 10
  print *, 'I=', I
end do
```

Cの繰り返し

<pre>for(i = 0 ; i < 10 ; i++) { printf("i=%d¥n", i) }</pre>	<pre>for(x = 0.0 ; i < 10.0 ; x += 2.0) { printf("i=%d x=%lf¥n", I, x) }</pre>
---	---

Pythonに特有の文法: 繰り返しはiterator (反復子) で回す

01c_fitting_gpt_rev2.py のExcelファイル読み込み部分:

```
for row in sheet.iter_rows(min_row=2, values_only=True):
```

```
    T.append(row[0])
```

```
    N.append(row[2])
```

Pythonの繰り返し構文

```
for 変数 in iterator: # Pythonの繰り返し変数は iterator
```

```
    forブロック開始 # iterator: 呼び出されるたびに次の値を 変数 に返し、  
                    # 値がなくなったら None を返す
```

上の例では、sheet.iter_rows() が iterator を返す

Pythonに特有の文法: 繰り返しはiteratorで回す

例1: 配列変数を一つずつ実行

```
i_list = [1, 2, 3, 4, 5]
```

```
for i in i_list:  
    print(f'i={i}')
```

例2:

```
str_list = ['a', 'b', 'c', 'd', 'e']
```

```
for s in s_list:  
    print(f's={s}')
```

例3: 指定した範囲の整数を一つずつ実行

```
for i in range(0, 10):  
    print(f'i={i}')
```

例5: 指定した範囲の実数を一つずつ実行

```
for s in numpy.arange(0.0, 10.0, 2.0):  
    print(f's={s}')
```

例6: ファイルを1行ずつ読み込む

```
fp = open('input.txt', 'r')  
for line in fp:  
    print(f's={s}')
```

Pythonに特有の文法: 繰り返しはiteratorで回す

iteratorとして使える変数: list, tuple, dict.keys()/values()/items()
numpy.ndarray()
ファイルハンドル (open()やcsv.reader()の戻り値)

iteratorを返す関数: range() (range型)
numpy.arange() (numpy.ndarray型)

iteratorの使い方

- **for var in iterator** 文で使う
- **next(iterator): iterator** から値を取り出し、要素位置を一つ進める

Excelファイルの読み込み: pandasの方が簡単

(01b_fitting_gpt_rev2.py, 05-read_excel_pandas.py)

- Pandas:**
- Excelファイルの読み書きに便利
 - 機械学習ライブラリ scikit-learn と組み合わせて使われる
 - 配列は **DataFrame** 型。ラベル (columns) と index (indexes) と一緒に管理できる
 - 配列は “**行優先**” なので注意

```
import pandas as pd
```

```
df = pd.read_excel(input_path)
```

```
labels = df.columns.to_numpy()
```

```
data_list = df.to_numpy().T
```

```
T = data_list[0]
```

```
N = data_list[1]
```

```
# input.xlsxから DataFrame型変数 df に読み込む
```

```
# dfのラベル numpy.ndarray型に変換する
```

```
# dfのデータを 2次元numpy.ndarray型に変換する
```

```
# このままでは行優先なので、
```

```
# data_list[0] は Excelファイルの2行目のデータになっている
```

```
# T,N配列を取り出すため、.Tで転置行列を取る
```

```
# data_listを リスト型変数 T, N に代入する。
```

```
# numpy.ndarrayはlist型の替わりとして使えるので、list型に変換しなくても大丈夫
```

テキストファイルの読み込み

(05-read_text_file.py)

- 最も基本的なファイル処理:
open()を使って受け取ったファイルハンドルオブジェクトで操作する

```
fp = open(input_path, 'r')           # open()関数でファイルを開き、ファイルハンドルオブジェクト fp を取得

data_list = []                       # データを格納するためのリストを作成
labels = fp.readline().split()       # 1行目のラベルを取得してリストに追加。strip()によりスペースで分割
for line in fp:                      # 1行ずつ読み込んで line に返す
    data = line.split()              # line文字列をスペースで分割して返す
    data = [float(d) for d in data]   # dataの要素 (文字列型) を実数型 (float) に変換してリストを作る
    data_list.append(data)           # 1次元リスト data_list の要素として、1次元リスト dataを追加し、2次元リストにする

T = data_list[0]                     # data_listを リスト型変数 T, N に代入する。
N = data_list[1]
```

型変換 (05-read_text_file.py)

- ・ テキストファイルからは、基本的には **文字列型** になる
例外: openpyxl, pandas: Excel内での型に依存
numpy: 数値型しか読み込めない
- ・ **文字列型 (str) を実数型 (float) に変換しないと、計算に使えない**

型変換関数:

文字列型 $s \Rightarrow$ 実数型 v : **$v = \text{float}(s)$**

文字列型 $s \Rightarrow$ 整数型 i : **$i = \text{int}(s)$**

実数型、整数型 $v \Rightarrow$ 文字列型 s : **$s = \text{str}(v)$**
 $s = f'\{v\}'$

テキストファイルの読み込み: with文

(05-read_text_file_with.py)

```
with open(input_path, 'r') as file:
```

```
# with ブロックで open() すると、withブロックから抜けるときに  
# 自動的に .close() してくれる
```

```
    data_list = []
```

```
    labels = file.readline().split()
```

```
    for line in file:
```

```
        data = line.split()
```

```
# line文字列を スペース で分割して返す
```

```
        data = [float(d) for d in data]
```

```
# dataの要素 (文字列型) を実数型 (float) に変換してリストを作る
```

```
        data_list.append(data)
```

```
# 1次元リスト data_list の要素として、1次元リスト dataを追加し、2次元リストにする
```

```
T = data_list[0]
```

```
# data_listを リスト型変数 T, N に代入する。
```

```
N = data_list[1]
```

テキストファイルの読み込み: `numpy.loadtxt()`

(05-read_text_file_loadtxt.py)

- 2次元数値データを読み込むなら1番簡単
- **実数型の `numpy.ndarray` 型 2次元配列**を返す
- 数値データしか読み込めない

`data_list = np.loadtxt(input_path, skiprows=1)` # 最初の 1 行をスキップし、2 行目以降を 2 次元配列に読み込んで返す

`T = data_list[0]`

`data_list` を リスト型変数 `T, N` に代入する。

`N = data_list[1]`

CSVファイルの読み込み: csvライブラリ

(05-read_csv_file.py)

- CSV (Comma Separated Values) ファイルの読み書き
データ中に , があったり、” ” でクオートされているデータも正常に読み込める

```
with open(input_path, '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)
```

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

Python特有の文法: リスト内包表記

(01c_fitting_gpt_rev2.py)

```
data = [float(d) for d in data]
```

リスト内包表記

繰り返し構文 (ループ)

+

配列 (list) 作成

を同時に行える

Python特有の文法: リスト内包表記とfor文 (01c_fitting_gpt_rev2.py)

```
data = [float(d) for d in data]
```

以下の繰り返し構文と同じ:

```
data = []  
for d in data:  
    data.append(float(d)) # 配列 (list) への追加は .append() を使う
```

長所: プログラムが短くなる。場合によっては可読性が上がる

forループよりも早い (ほとんどの場合無視できる違い)

短所: わかりにくい

個人の意見: 非常に単純なテンプレパターン以外では使わないことを推奨

リスト内包表記: ChatGPTで解決

Q: `data = [float(d) for d in data]` をfor文に展開してください

A:

```
data = []
```

```
for d in data:
```

```
    value = float(d)
```

```
    Tinv.append(value)
```

Q: 次の2重for文をリスト内表記に変換してください

```
l = []
```

```
for i in range(10):
```

```
    for j in range(10):
```

```
        l.append([i, j, i*j])
```

A: `l = [[i, j, i*j] for i in range(10) for j in range(10)]`

数学関数: math/numpy/scipy モジュールを使う

- Pythonの標準 (組み込み関数) には数学関数はない。
- math、numpy、scipy モジュールなどに含まれている関数を使う

01c_fitting_gpt_rev2.pyの例:	別の書き方:	mathモジュールを使う例:
<code>import numpy as np</code>	<code>from numpy import log</code>	<code>from math import log</code>
<code>logN = []</code>	<code>logN = []</code>	<code>logN = []</code>
<code>for n in N:</code>	<code>for n in N:</code>	<code>for n in N:</code>
<code> logN.append(np.log(n))</code>	<code> logN.append(log(n))</code>	<code> logN.append(log(n))</code>

その他の数学関数: `exp()`, `log()`, `sin()`, `cos()`, `tan()`, `sinh()`, `cosh()`, `tanh()`
`arcsin()`, `arccos()`, `arccot()` など

入力データの確認: 画面出力 print()

基本的な書き方: `print(arg1, arg2, arg3, ...)`

`arg1, arg2, arg3`を文字列に変換して順番に出力

最後に改行する (改行したくないときは、引数の後に `end = ''` を入れる: 例: `print("x=", x, end = '')`)

リスト変数 `T N` を画面に出力

```
print("T:", T)
```

```
print("N:", N)
```

実行結果: 読みにくい

```
T: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160]
```

```
N: [1, 1.4142135623730951, 1.7320508075688772, 2, 2.23606797749979,  
2.449489742783178, 2.6457513110645907, 2.8284271247461903, 3, 3.1622776601683795,  
3.3166247903554, 3.4641016151377544, 3.605551275463989, 3.7416573867739413,  
3.872983346207417, 4]
```


出力文字列の整形: f-string (f文字列、フォーマット済み文字列リテラル)

- **print() には整形機能はない**
- **整形は、文字列変数自身が持っている機能**
.format() 関数、C形式の書式指定 など使えが、
f-string を使うべき

例: f“T: {T}“

- **文字列の前に ‘f’ を書く**
- **{ }の中に変数名を書くと、変数の内容が展開される**

T = 297.14 の時、**f“T: {T}“**は

“T: 297.14”

と出力される

出力文字列の整形: f-string

変数の値を桁数を指定して出力

参考URL: <https://note.nkmk.me/python-f-strings/>

```
print(f'T: {T:.2f}')      # 実数変数Tを有効数字2桁で表示
print(f'T: {N:.4e}')      # 実数変数Nを e 形式 (1.0e2 など)、小数点以下4桁で表示
print(f'Tinv: {Tinv}')    # 実数変数Tinvをフォーマット指定なしで表示
print(f'logN: {logN:.6g}') # 実数変数logNを有効数字6桁で表示

print(f'nAtom: {nAtom:6d}') # 整数変数 nAtom を 整数6桁で表示
print(f'name: {name:10}')  # 文字列変数 name を 10桁で表示

print(f'name: {1000/T}')   # {}内にはpython文を書ける
                           # ここでは 1000 / T を計算した結果を出力
```

数値データを横に並べて見やすく (01c_fitting_gpt_rev2.py)

文字列中の ¥t はTAB文字に置き換えて出力される

```
print('T¥tN¥t1/T¥tlog(N)')
```

zip()を使うと、複数のリスト変数を1つずつ取り出してくれる

```
for t, n, t_inv, log_n in zip(T, N, Tinv, logN):
```

```
    print(f'{t:.2f}¥t{n:.4e} ¥t{t_inv:.4f}¥t{log_n:.6g}')
```

numpy.polyfit()関数呼び出し (01c_fitting_gpt_rev2.py)

多項式線形最小二乗法は、numpy モジュールの .polyfit() 関数で実行できる

numpyモジュールのimport。numpyだと長いので、npという別名をつける

```
import numpy as np
```

x軸の値リストに Tinv、y軸の値リストに logN を渡し、

次数 1 の多項式で最小二乗法を行う。

フィッティング関数は $y = b * x + a$ <= 直観的な係数の順序と逆なので注意！

```
b, a = np.polyfit(Tinv, logN, 1)
```

CやFortran: 関数は1つの値しか返せない

Cの関数は1つの値しか返せない

```
double f(x) {  
    double x;  
    return x * x;  
}  
  
int main(void) {  
    double x2;  
    x2 = f(3.0);  
}
```

Cプログラムポインタ変数を使う例

```
void f(x, px, px2) {  
    double x;  
    double *px, *px2;  
    *px = x;  
    *px2 = x * x;  
}  
  
int main(void) {  
    double ret, x2ret;  
    f(3.0, &xret, &x2ret);  
}
```

Cプログラムで配列に返す例

```
void f(x, ret) {  
    double x;  
    double ret[];  
    ret[0] = x;  
    ret[1] = x * x;  
}  
  
int main(void) {  
    double ret[2];  
    f(3.0, ret);  
}
```

Cプログラムでグローバル変数を使う例

```
double xret, x2ret;  
void f(x) {  
    double x;  
    double ret[];  
    ret[0] = x;  
    ret[1] = x * x;  
}  
  
int main(void) {  
    f(3.0);  
    /* xret, x2ret を参照する */  
}
```

古い言語では、関数が返せる値は一つ (1つの変数) だけ

- ・ ポインタ変数を引数に渡す: C, C++, Fortran2003以降
- ・ 配列、構造体、オブジェクトに値を入れて返す: C, C++
- ・ その他の場合、グローバル変数を使う

面倒くさい、わかりにくい、危険

Python特有の文法: 関数は複数の値を返せる

例:

```
def f(x):  
    return x, x*x
```

戻り値の数がわかっている場合: 変数を並べて受け取る

```
a1, a2 = f(3.0)          # 戻り値2つを 2つの変数 a1, a2 で受け取る  
print(a1, a2)           # 3.0, 9.0 が表示される
```

注意: 関数の戻り値の数と 受け取る変数の数が一致しないとエラーになる

戻り値の数が不明な場合／変わる場合: 配列 (tuple) で受け取る

```
a = f(3.0)               # 戻り値2つを 配列変数 (tuple) a で受け取る  
print(a)                 # (3.0, 9.0) が表示される  
a1 = a[0]                # 配列の第1要素を a1 に入れる  
a2 = a[1]                # 配列の第2要素を a2 に入れる
```

最小二乗の結果の画面出力: f-string (fitting-gpt-rev2.py)

ChatGPTのプログラム (fitting-gpt.py) :

```
print("a:", a)  
print("b:", b)
```

こっちの方がわかりやすい (fitting-gpt-rev2.py)

```
print()  
print(f'fitting result: 1/T = {a:.6f} + {b:.6f} * log(N)')
```

出力結果:

[空行]

fitting result: 1/T = 1.277128 + -15.079957 * log(N)

fitting結果をExcelファイルへ保存 (01a_fitting_gpt.py)

```
import openpyxl
```

```
# プログラム最初のグローバル変数に、出力ファイル名の変数を定義
```

```
output_path = 'fitting.xlsx'
```

```
output_workbook = openpyxl.Workbook()
```

```
# 新規にExcel workbookを作る
```

```
output_sheet = output_workbook.active
```

```
# 現在のworksheetを受け取る
```

```
output_sheet.append(["T", "N", "Tinv", "logN", "logNcal"]) # 1行目にラベル文字列を書込む
```

```
# zip()を使うと、複数のリストの要素を一つずつ取り出してくれる
```

```
for t, n, t_inv, log_n, log_n_cal in zip(T, N, Tinv, logN, logNcal):
```

```
    output_sheet.append([t, n, t_inv, log_n, log_n_cal])
```

```
# 新しい行として、データを追加
```

```
print()
```

```
print(f'Save data to [{output_path}]')
```

```
# ファイル保存のメッセージを表示
```

```
output_workbook.save(output_path)
```

```
# Excelファイルに保存
```


Excelファイルへ保存: pandasを使うほうが簡単

(01c_fitting_gpt_rev2.py)

```
import numpy as np
import pandas
```

```
# 保存するデータを2次元配列にする
```

```
# pandas.DataFrame型に変換するには転置を取る必要があるので、
```

```
# numpy.ndarray()型に変換して、.Tで転置を取る
```

```
save_list = np.array([T, N, Tinv, logN, logNcal]).T
```

```
# pandas.DataFrame型に変換。columns= でラベルの配列を設定できる
```

```
df = pd.DataFrame(save_list, columns=["T", "N", "1/T", "log(N)", "log(Ncal)"])
```

```
# Excelファイルに保存。index=Trueとすると、1列目に連番が入る
```

```
df.to_excel(output_path, index=False)
```

グラフを描く: matplotlib (01c_fitting_gpt_rev2.py)

```
# matplotlib モジュールの pyplotクラスを plt という別名でimport  
import matplotlib.pyplot as plt
```

```
# xデータにリスト変数 Tinv、yデータに logN を指定して散布図を描く  
# 凡例に表示する文字列を label="Data"で与えている
```

```
plt.scatter(Tinv, logN, label="Data")
```

```
# xデータにリスト変数 Tinv、yデータに logNcal を指定して折れ線グラフを描く  
# 線の色は red
```

```
# 凡例に表示する文字列を label="Fitted Curve"で与えている
```

```
plt.plot(Tinv, logNcal, color='red', label="Fitted Curve")
```

```
plt.xlabel("1/T")          # x軸の文字列を 1/T
```

```
plt.ylabel("log(N)")       # y軸の文字列を log(N)
```

```
plt.legend()               # .scatter()、.plot()で与えた label を凡例として表示する
```

```
plt.show()                 # グラフを表示する
```

```
# グラフウィンドウを閉じるまで、プログラムは停止
```

実行結果 (01c_fitting_gpt_rev2.py)

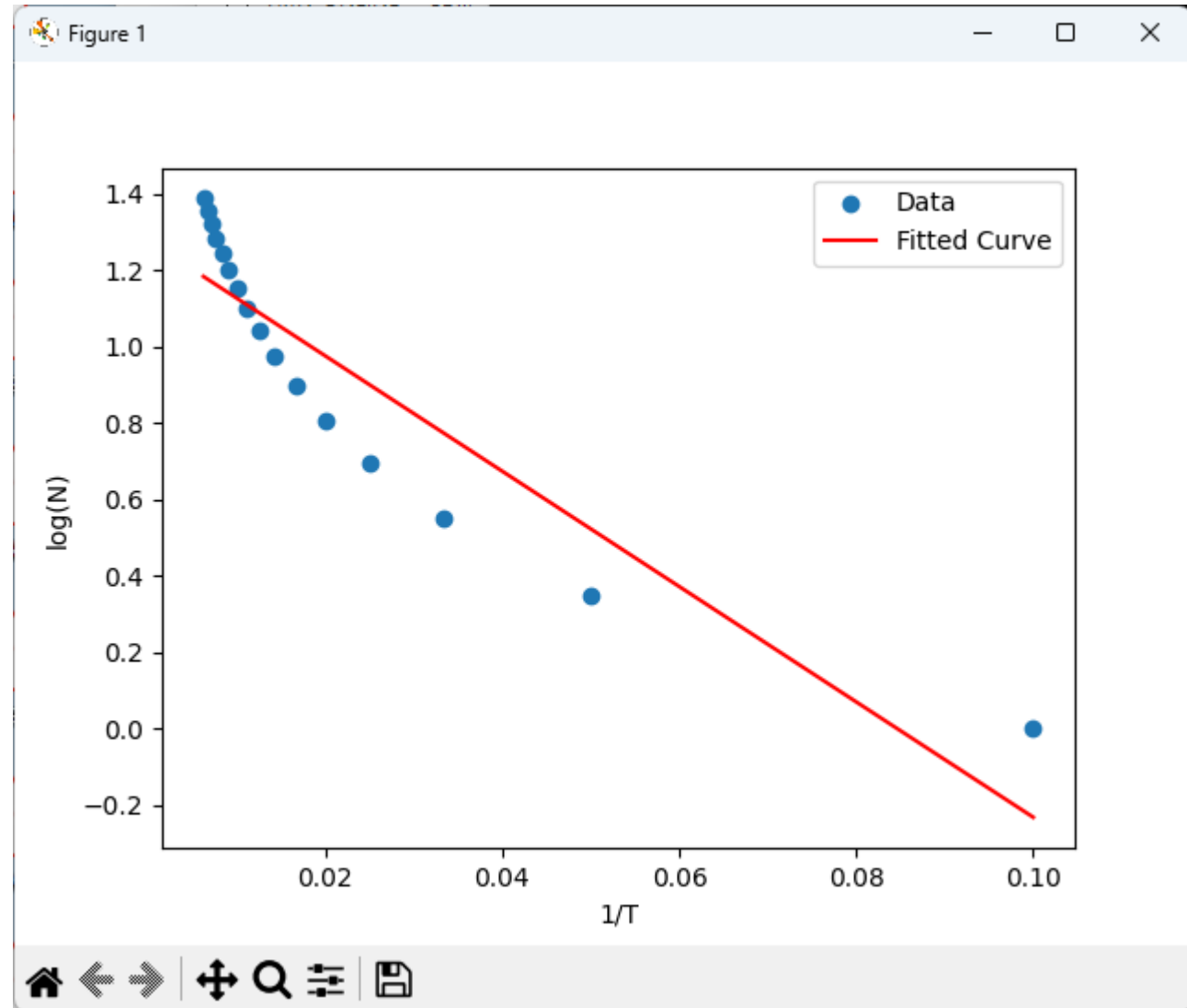
> Python fitting-gpt-rev2.py

Read data from [input.xlsx]

T	N	1/T	log(N)
10.00	1.0000e+00	0.1000	0
20.00	1.4142e+00	0.0500	0.346574
30.00	1.7321e+00	0.0333	0.549306
40.00	2.0000e+00	0.0250	0.693147
50.00	2.2361e+00	0.0200	0.804719
-- cut --			
120.00	3.4641e+00	0.0083	1.24245
130.00	3.6056e+00	0.0077	1.28247
140.00	3.7417e+00	0.0071	1.31953
150.00	3.8730e+00	0.0067	1.35403
160.00	4.0000e+00	0.0063	1.38629

fitting result: $1/T = 1.277128 + -15.079957 * \log(N)$

Save fitting data to [fitting.xlsx]



第1部 pythonプログラミングの基礎

1. ChatGPTを使って最小二乗法のプログラムを作る
2. ・ このプログラムをデバッグ、改善、
・ Fortran, BASIC, Cと比較しながら
pythonプログラムの文法、作り方を学ぶ。

第2部 学生が提出してきたpythonプログラムをどのように理解するか

1. “良いプログラム”への誘導: 私の数値解析講義の例
2. pythonプログラムを読みやすくする: リファクタリング

“良いプログラム”への誘導: 講義(数値解析) での工夫

計算材料科学特論

<http://conf.msl.titech.ac.jp/Lecture/ComputationalMaterialsScience/index-numericalanalysis.html>

初回資料: June 12, 9:07 Lecture materials on June 13 have been uploaded. (20230613ComputerAndErrorSources.zip)

- 比較的簡単なプログラムを配布している
- 授業の課題として、選択問題を出している
 - 問1 10進数、16進数などの変換
 - 問2 配布したプログラムが何をやっているのかを説明する
のどちらかを選び、回答する

効果:

- ・ プログラミングに興味のある学生が回答する (選択)
- ・ 模範プログラムを読む機会ができる
- ・ プログラミングの上達には、いろいろなプログラムを「読む」「動かす」「書く」こと

配布プログラム: 浮動小数点の誤差

Representation of floating point in computer:

$$-1. \mathbf{011101}_2 \times 2^{-\mathbf{015}_{10}} \quad (\text{in binary})$$

Errors arise from converting Base 10 to Base 2.

- Some values do not have errors between Base 10 and Base 2 if fraction equals to 2^n

$$1.0 = (1.0)_2 \times 2^0$$

$$0.5 = (1.0)_2 \times 2^{-1}$$

$$0.125 = (1.0)_2 \times 2^{-3}$$

$$0.0390625 = 1.25 \times 2^{-5} = (1.01)_2 \times 2^{-5}$$

$$1.75 = 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = (1.11)_2$$

$$0.65625 = 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} = (0.10101)_2$$

$$100.0 = 1.5625 \times 64 = (1 + 2^{-1} + 2^{-4}) \times 2^6 = (1.1001)_2 \times 2^4$$

- Other values have errors

even if it is represented by a simple figure in Base 10:

$$\mathbf{0.1 = (1.1001100110011001 \cdots)_2 \times 2^{-3}}$$

プログラム (roundoff error): sum_error.py

Usage: python sum_error.py *h n iPrintStep*

Summing up *h* for *n* times with different precision interger types. Output every *iPrintStep* steps.

python sum_error.py 0.1 100 20

exact:	sum16 (error)	sum32 (error)	sum64 (error)
0.1000:	0.099975585937500000 (+2.44e-05)	0.100000001490116119 (-1.49e-09)	0.100000000000000006 (+0.00e+00)
2.1000:	2.095703125000000000 (+4.30e-03)	2.100000143051147461 (-1.43e-07)	2.1000000000000000533 (-4.44e-16)
4.1000:	4.089843750000000000 (+1.02e-02)	4.099998474121093750 (+1.53e-06)	4.1000000000000001421 (-8.88e-16)
6.1000:	6.121093750000000000 (-2.11e-02)	6.099996566772460938 (+3.43e-06)	6.099999999999994316 (+6.22e-15)
8.1000:	8.148437500000000000 (-4.84e-02)	8.099994659423828125 (+5.34e-06)	8.099999999999987210 (+1.24e-14)

python sum_error.py 0.125 100 20

exact:	sum16 (error)	sum32 (error)	sum64 (error)
0.1250:	0.125000000000000000 (+0.00e+00)	0.125000000000000000 (+0.00e+00)	0.125000000000000000 (+0.00e+00)
2.6250:	2.625000000000000000 (+0.00e+00)	2.625000000000000000 (+0.00e+00)	2.625000000000000000 (+0.00e+00)
5.1250:	5.125000000000000000 (+0.00e+00)	5.125000000000000000 (+0.00e+00)	5.125000000000000000 (+0.00e+00)
7.6250:	7.625000000000000000 (+0.00e+00)	7.625000000000000000 (+0.00e+00)	7.625000000000000000 (+0.00e+00)
10.125:	10.125000000000000000 (+0.00e+00)	10.125000000000000000 (+0.00e+00)	10.125000000000000000 (+0.00e+00)

python sum_error.py 0.0390625 100 20

exact:	sum16 (error)	sum32 (error)	sum64 (error)
0.0391:	0.039062500000000000 (+0.00e+00)	0.039062500000000000 (+0.00e+00)	0.039062500000000000 (+0.00e+00)
0.8203:	0.820312500000000000 (+0.00e+00)	0.820312500000000000 (+0.00e+00)	0.820312500000000000 (+0.00e+00)
1.6016:	1.601562500000000000 (+0.00e+00)	1.601562500000000000 (+0.00e+00)	1.601562500000000000 (+0.00e+00)
2.3828:	2.382812500000000000 (+0.00e+00)	2.382812500000000000 (+0.00e+00)	2.382812500000000000 (+0.00e+00)
3.1641:	3.164062500000000000 (+0.00e+00)	3.164062500000000000 (+0.00e+00)	3.164062500000000000 (+0.00e+00)

配布プログラム: 数の表現と基数

Base 10 $1975 = 1 \times 1000 + 9 \times 100 + 7 \times 10 + 5 \times 1$
(decimal) $= 1 \times 10^3 + 9 \times 10^2 + 7 \times 10^1 + 5 \times 10^0$
(10進数) the 1000's place
 (1000の位)

All data in computer are represented by **0 or 1** (binary) : **bit (b)**

Base 2 $(11011)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
(binary) $= 1 \times (16)_{10} + 1 \times (8)_{10} + 0 \times (4)_{10} + 1 \times (2)_{10} + 1 \times (1)_{10}$
(2進数) $= (27)_{10}$

Base r $N = a_n r^n + a_{n-1} r^{n-1} + \cdots + a_3 r^3 + a_2 r^2 + a_1 r^1 + a_0 r^0$
(r 進数) $= (a_n a_{n-1} \cdots a_3 a_2 a_1 a_0)_r$

配布プログラム: 数の表現と基数

Base 8 (octal) (8進数)

(01234567)

2 digits: $0 \sim 8^2 - 1 = 63$

$$\mathbf{00: } 0 \times 8^1 + 0 \times 8^0 = 0$$

$$\mathbf{53: } 5 \times 8^1 + 3 \times 8^0 = 43$$

$$\mathbf{77: } 7 \times 8^1 + 7 \times 8^0 = 63$$

Base 16 (hexadecimal) (16進数)

(0123456789ABCDEF) = (0 ~ 15)

2 digits: $0 \sim 16^2 - 1 = 255$

$$\mathbf{00: } 0 \times 16^1 + 0 \times 16^0 = 0$$

$$\mathbf{9F: } 9 \times 16^1 + 15 \times 16^0 = 159$$

$$\mathbf{FF: } 15 \times 16^1 + 15 \times 16^0 = 255$$

(ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz

0123456789+/-) = (0 ~ 63)

配布プログラム: 基数の変換

Base r to Base 10

$$N_r = (a_n a_{n-1} \cdots a_3 a_2 a_1 a_0)_r$$

$$N_{10} = a_0 r^0 + a_1 r^1 + a_2 r^2 + a_3 r^3 + \cdots + a_{n-1} r^{n-1} + a_n r^n$$

$$\text{Ex. } 1101_2 = 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 = 13_{10}$$

Base 10 to Base r

$$N_{10} = (b_n b_{n-1} \cdots b_3 b_2 b_1 b_0)_{10} = (c_n c_{n-1} \cdots c_2 c_1 c_0)_r$$

$$= c_0 r^0 + c_1 r^1 + c_2 r^2 + \cdots + c_{n-1} r^{n-1} + c_n r^n$$

$$= c_0 + r(c_1 + c_2 r^1 + \cdots + c_{n-1} r^{n-2} + c_n r^{n-1})$$

$$= c_0 + r(c_1 + r(c_2 + c_3 r^1 + \cdots + c_{n-1} r^{n-3} + c_n r^{n-2}))$$

$$(1) N_{10}^{(0)} = N_{10} = N_{10}^{(1)} * r + c_0 \quad \text{where } 0 \leq c_0 < r$$

$$(2) N_{10}^{(1)} = N_{10}^{(2)} * r + c_1 \quad \text{where } 0 \leq c_1 < r$$

... repeat until $N_{10}^{(n+1)} = 0$

$$\Rightarrow N_r = (c_n c_{n-1} \cdots c_2 c_1 c_0)_r$$

Ex. Base 10 to Base 8

$$302_{10} = 8 \times 37 + 6$$

$$37_{10} = 8 \times 4 + 5$$

$$4_{10} = 8 \times 0 + 4$$

$$300_{10} = 456_8$$

配布プログラム: base.py

Program: base.py

Usage: python base.py value base_source base_target

Ex.

COMMAND:

python base.py FA 16 8

Convert FA in base 16 to base 8

OUTPUT:

Convert FA in base 16 to base 10

1st digit = 10: $+ 10 * 16^0 \Rightarrow + 10_{10} \Rightarrow 10_{10}$

2nd digit = 15: $+ 15 * 16^1 \Rightarrow + 240_{10} \Rightarrow 250_{10}$

Convert 250 in base 10 to base 8

$250_{10} = 31 * 8 + 2$: base_8 $\Rightarrow 2$

$31_{10} = 3 * 8 + 7$: base_8 $\Rightarrow 72$

$3_{10} = 0 * 8 + 3$: base_8 $\Rightarrow 372_8$ result

python推奨書式: PEP8

PEP8公式: <https://pep8-jp.readthedocs.io/>

- ・ 教育的には、PEP8準拠のプログラムコードを使ったほうがいいかもしれない

個人的にはPEP8には従っていません。

上記のPEP8公式ドキュメントにも注意が書いてあります

- ・ 一貫性にこだわりすぎるのは、狭い心の現れである
- ・ 一貫性を崩すべき場合があることも知っておいてください
疑問に思ったときは、あなたの判断を優先してください。
他の例を調べ、一番良さそうなものを決めて下さい。
- ・ 特に、このPEPに準拠するためにコードの後方互換性を壊すようなことは絶対にしないで下さい！

PEP8に変換するのも簡単

- ・ ChatGPTを使う
- ・ formatterを使う (flake8, pylint)

変数の命名法の例

スネークケース (snake case): 例 atom_site_position

キャメルケース (camel case): 例 atomSitePosition)

個人ルール:

- ・ ループ変数: i (整数)、s (文字列)、path/f/filename (ファイル名)、(ix, iy, iz) (多重ループ)
- ・ 数を表す変数: **n**atom ・ 順序を表す変数: **i**atom ・ 文字列: atom_**n**ame
- ・ 配列を表す変数: atom**s**, atom_**list**, atom_**array**

PEP8:

- ・ 定数: 全て大文字、スネークケース。ファイルのトップレベルで定義
- ・ 関数、変数: 小文字
- ・ 型変数: キャメルケースで短く
- ・ モジュール名: 全て小文字、スネークケース
- ・ クラス名: キャメルケース

ハンガリアン記法: 変数名の最初の数文字で変数型・意味を区別する

現在は推奨されていないが、わかりやすいこともある

- ・ 整数: 最初に i
- ・ 文字列: 最初に str
- ・ ハンドル (ファイルハンドル、メモリハンドルなど): 最初に h

pythonを使う上での基本方針

- ・ モジュールごとに、似た変数型がある
配列変数の例: **list**型、numpy.**ndarray**型、pandas.**DataFrame**型

=> 都合の良いものを利用し、相互に変換して使う

短所: 使用メモリが大きい、実行オーバーヘッドが大きい

現代のプログラミングの考え方:

無駄なチューニングに時間は使わない

- ・ メモリが足りなければ増やす
- ・ 実行速度が遅ければ早いPCを使う
- ・ それでも足りなければ Cなどで書き直す

学生へのメッセージ: わかりやすいことが一番

- **最優先: プログラムの可読性**
デバッグ、修正、メンテナンスがしやすくなる
- わかりやすい・読みやすい変数名 (意味不明な省略語を使わない)
- わかりにくい構文 (リスト内包表記) を使わない
- 機能単位ごとに空行を入れて見やすく
- 必要に応じてコメント文を入れる (冗長なコメントは入れない)
- プログラム、関数に `''' ~ '''` コメント (docstring) を入れる **
** すいません。私はあまり使ってないです

読みにくいプログラム: 03a_dirty.py

```
import openpyxl
import numpy as np
import matplotlib.pyplot as plt
workbook = openpyxl.load_workbook('input.xlsx')
sheet = workbook.active
c = []; d = []
for row in sheet.iter_rows(min_row=3,
values_only=True):
    c.append(row[0])
    d.append(row[2])
e = [1 / g for g in c]
f = [np.log(h) for h in d]
output_workbook = openpyxl.Workbook()
output_sheet = output_workbook.active
output_sheet.append(["T", "N", "1/T", "log(N)"])
for g, h, i, j in zip(c, d, e, f):
    output_sheet.append([g, h, i, j])
output_workbook.save("fitting.xlsx")
plt.scatter(e, f)
plt.show()
```

- 複数の命令が1行に書かれている
=> 複数行に分解する
- 変数名を見ても、何の変数からわからない
=> 意味のある変数に置き換える
- 全てのコードが詰まっていて、機能ごとの区別がつきにくい
=> 機能ごとの単位で空行を入れて分割する
関数化する
- コメント文がない
=> コメント文の追加、関数のdocstringの追加
- グラフのプロットデータ、X軸、Y軸が何かわからない
=> 凡例、xラベル、yラベルを追加

ChatGPTに書き直してもらう Ver4

次のプログラムは、input.xlsxの1列目から温度 T 、2列目からキャリア濃度 N を読み込み、 $1/T$ と $\log(N)$ に変換し、その結果をfitting.xlsxに保存し、グラフを描くプログラムです。以下のように読みやすく書き直してください

- ・ 全体のリファクタリングをしてください
- ・ 変数名をわかりやすく置き換えてください
- ・ 入力ファイル名、出力ファイル名を変数に入れてください
- ・ 可能な範囲で関数化し、入出力をdocstringで追加してください
- ・ T - N 図を左のグラフ枠に追加し、右のグラフ枠に $1/T$ - $\log(N)$ 図を描いてください
- ・ グラフに凡例、 x ラベル、 y ラベルを追加してください

```
import openpyxl
import numpy as np
import matplotlib.pyplot as plt
workbook = openpyxl.load_workbook('input.xlsx')
sheet = workbook.active
c = [];d = []
for row in sheet.iter_rows(min_row=3, values_only=True):
    c.append(row[0])
    d.append(row[2])
e = [1 / g for g in c]
f = [np.log(h) for h in d]
output_workbook = openpyxl.Workbook()
```

ChatGPTの回答 Ver4

以下のコードは、
リファクタリングされ、
変数名がわかりやすく置き換えられ、
関数化され、
入出力がドキュメントストリングで説明され、
 $1/T\text{-}\log(N)$ 図と $T\text{-}N$ 図のグラフが追加され、
グラフに凡例と軸ラベルが追加されました。

-- ここに修正したコード --

このコードでは、
 $1/T\text{-}\log(N)$ 図と $T\text{-}N$ 図の両方のグラフが表示され、
グラフに凡例と軸ラベルが追加されました。

ChatGPTの修正プログラム Ver4: 03e_dirty_GPT_rev4.py

```
import openpyxl
import numpy as np
import matplotlib.pyplot as plt

def load_data(input_filename):
    """
    Excelファイルから温度とキャリア濃度データを読み込む

    Args:
    input_filename (str): 入力ファイルの名前

    Returns:
    list, list: 温度データのリストとキャリア濃度データのリスト
    """
    workbook = openpyxl.load_workbook(input_filename)
    sheet = workbook.active

    temperatures = []
    concentrations = []
```

```
    for row in sheet.iter_rows(min_row=3,
                               values_only=True):
        temperatures.append(row[0])
        concentrations.append(row[1])

    return temperatures, concentrations

def transform_data(temperatures, concentrations):
    """
    温度とキャリア濃度データを1/Tとlog(N)に変換する
    Args:
    temperatures (list): 温度データのリスト
    concentrations (list): キャリア濃度データのリスト
    Returns:
    list, list: 1/Tデータのリストとlog(N)データのリスト
    """
    inv_temperatures = [1 / T for T in temperatures]
    log_concentrations = [np.log(N) for N in concentrations]

    return inv_temperatures, log_concentrations
```

ChatGPTの修正プログラム Ver4: 03e_dirty_GPT_rev4.py

```
def save_to_excel(temperatures, concentrations,
inv_temperatures, log_concentrations, output_filename):
    """
    温度、キャリア濃度、1/T、log(N)のデータをExcelファイル
    に保存する
```

Args:

temperatures (list): 温度データのリスト

concentrations (list): キャリア濃度データのリスト

inv_temperatures (list): 1/Tデータのリスト

log_concentrations (list): log(N)データのリスト

output_filename (str): 出力ファイルの名前

"""

output_workbook = openpyxl.Workbook()

output_sheet = output_workbook.active

output_sheet.append(["T", "N", "1/T", "log(N)"])

```
for T, N, inv_T, log_N in zip(temperatures, concentrations,
inv_temperatures, log_concentrations):
    output_sheet.append([T, N, inv_T, log_N])
```

output_workbook.save(output_filename)

```
def plot_TN_1T_logN_graph(temperatures, concentrations,
inv_temperatures, log_concentrations):
    """
```

温度とキャリア濃度データからT-N図と1/T-log(N)図の散布図を作成し、表示する

Args:

temperatures (list): 温度データのリスト

concentrations (list): キャリア濃度データのリスト

inv_temperatures (list): 1/Tデータのリスト

log_concentrations (list): log(N)データのリスト

"""

ChatGPTの修正プログラム Ver4: 03e_dirty_GPT_rev4.py

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

ax1.scatter(temperatures, concentrations, label='T-N Data')
ax1.set_xlabel('T')
ax1.set_ylabel('N')
ax1.legend()

ax2.scatter(inv_temperatures, log_concentrations,
label='1/T-log(N) Data')
ax2.set_xlabel('1/T')
ax2.set_ylabel('log(N)')
ax2.legend()

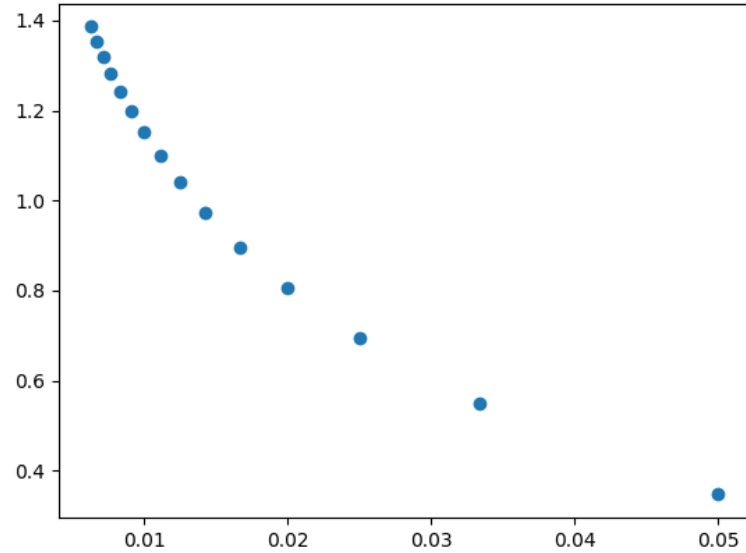
plt.show()
```

```
if __name__ == "__main__":
    input_filename = 'input.xlsx'
    output_filename = 'fitting.xlsx'

    temperatures, concentrations = load_data(input_filename)
    inv_temperatures, log_concentrations =
transform_data(temperatures, concentrations)
    save_to_excel(temperatures, concentrations,
inv_temperatures, log_concentrations, output_filename)
    plot_TN_1T_logN_graph(temperatures, concentrations,
inv_temperatures, log_concentrations)
```

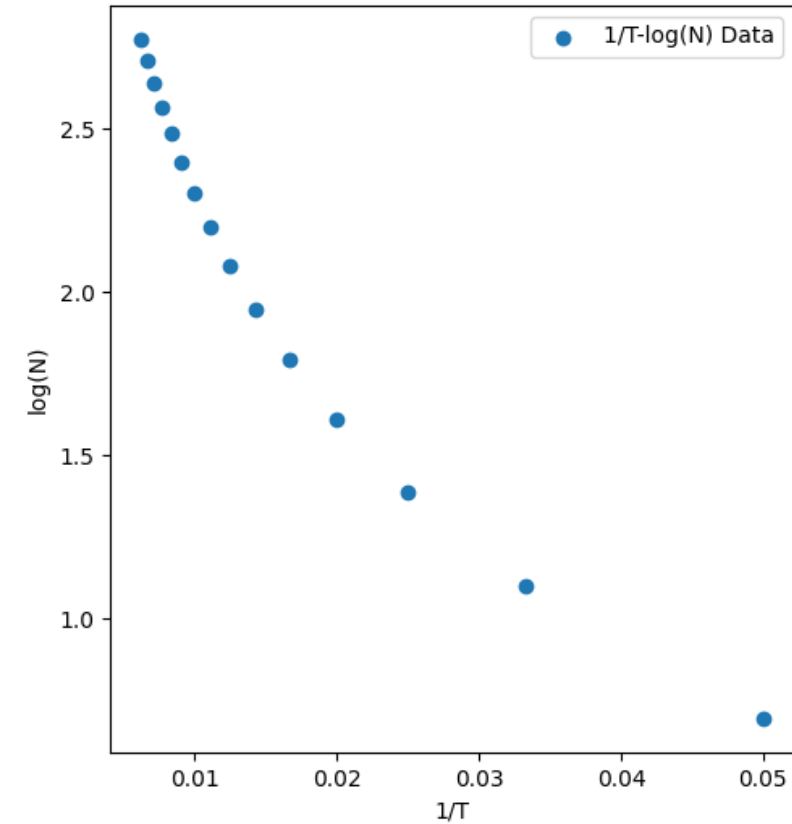
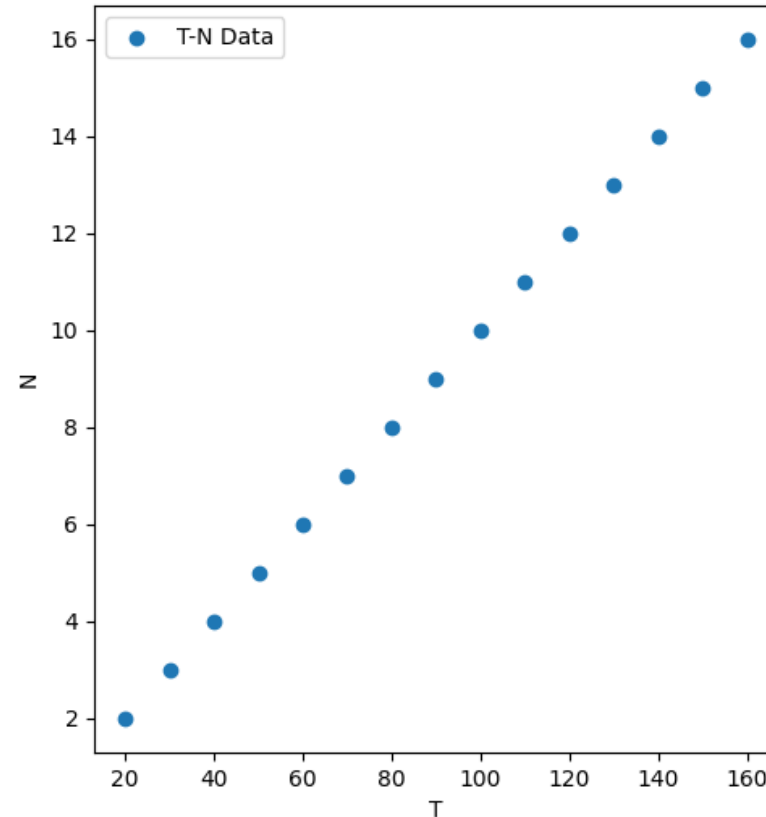
修正プログラム Ver4の実行結果: 03e_dirty_GPT_rev4.py

修正前 03a_dirty.py



修正後 03e_dirty_GPT_rev4.py

Figure 1



docstring (ドキュメント文字列) によるヘルプ支援機能

参考例: <https://qiita.com/flcn-x/items/393c6f1f1e1e5abec906>

```
def load_data(input_filename):
```

```
    """
```

Excelファイルから温度とキャリア濃度データを読み込む

Args:

input_filename (str): 入力ファイルの名前

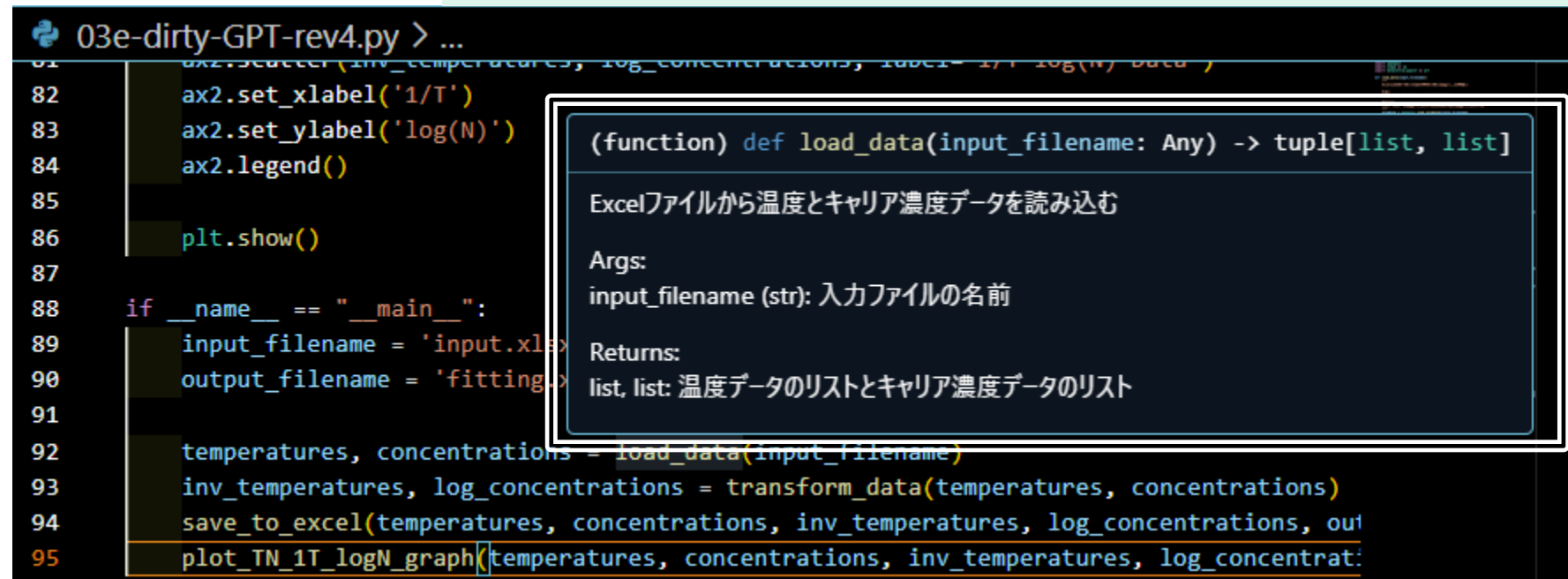
Returns:

list, list: 温度データのリストとキャリア濃度データのリスト

```
    """
```

プログラムの先頭(importの直後)、関数やclass定義の最初に docstring をいれると、helpとして使える

- ・ `.__doc__` というメンバ変数で参照できる
- ・ VSCodeでは、関数呼び出しや関数定義にカーソルを置いていくと、関数情報が表示される



```
03e-dirty-GPT-rev4.py > ...
81 ax2.scatter(inv_temperatures, log_concentrations, label='1/T log(N) data')
82 ax2.set_xlabel('1/T')
83 ax2.set_ylabel('log(N)')
84 ax2.legend()
85
86 plt.show()
87
88 if __name__ == "__main__":
89     input_filename = 'input.xlsx'
90     output_filename = 'fitting.xlsx'
91
92     temperatures, concentrations = load_data(input_filename)
93     inv_temperatures, log_concentrations = transform_data(temperatures, concentrations)
94     save_to_excel(temperatures, concentrations, inv_temperatures, log_concentrations, output_filename)
95     plot_TN_1T_logN_graph(temperatures, concentrations, inv_temperatures, log_concentrations)
```

(function) def load_data(input_filename: Any) -> tuple[list, list]

Excelファイルから温度とキャリア濃度データを読み込む

Args:

input_filename (str): 入力ファイルの名前

Returns:

list, list: 温度データのリストとキャリア濃度データのリスト

その他

- エラー処理 (読み込むファイルが存在しない など)
- 関数: デフォルト引数、キーワード引数
- **辞書型**
- 複素数 : **complex型** $c = 3.0 + 2.0j$
- 行列計算: 一次連立方程式、固有値: **numpy**
- 数値解析: 最適化、スペクトル解析、フーリエ変換: **scipy**
- 機械学習: **pandas, scikit-learn**
- GUI : **tkinter**, pygtk
- 装置制御 (GPIB): **PyVISA**
- 通信: ネットワーク、Webアプリ: **Flask, FastAPI**

参考: その他のプログラム、情報

D2MatE (智慧とデータが拓くエレクトロニクス新材料開発拠点) 公開・非公開プログラム情報

http://conf.msl.titech.ac.jp/D2MatE/D2MatE_programs.html

- [Top page](#)
- [共通ファイル・単位など](#)
[神谷・片瀬研共通単位表](#)
- [標準ディレクトリ構成](#)
- [Launcherプログラミング](#)
- ▶ [tkProg python ライブラリ: tklib](#)
- [データ読み込みplugin仕様](#)
[plug-inリスト](#)
[tklib.tkfilter](#)
- [plotly-Dash-Flaskメモ](#)
- [pythonでGPIB制御メモ](#)
- [SILVACO ATLAS tips](#)
- ▶ [python Tips](#)
- ▶ [python 高度な内容](#)
- ▶ [プログラミングメモ](#)
- ▶ [Linuxサーバ設定](#)
- ▶ [神谷・片瀬研メモ](#)
- ▶ [講義資料](#)
- [D2MatE拠点限定ページ](#)
- [更新履歴](#)

智慧とデータが拓くエレクトロニクス新材料開発拠点 公開・非公開プログラム情報 (Data Driven Materials Research Institute for Electronics)

質問、要望、バグ報告などの連絡先: 神谷 利夫 tkamiya@msl.tech.ac.jp
東京工業大学 [国際先駆研究センター](#) [元素戦略MDX研究センター](#) 教授

News (ユーザ)

- **New!** 2023/12/15 11:06 1月24日に [チュートリアル: 学生と教員のためのpythonとChatGPT活用法](#) を開催します
- 2023/9/26 12:02 一般向けパッケージを更新しました。
 - ・ GPIB、filterの追加

News (開発者)

更新

"External programs/外部プログラム" メニューの "Install modules" ボタンをクリックすると、必要なモジュールをすべてインストールします。インストールするモジュールは以下の通りです。

numpy scipy matplotlib seaborn scikit-learn chardet openpyxl pandas pymatgen
physbo
pygments msoffcryptography python-pptx

Python関係

参考: 数値解析プログラムを作りたい人へ

計算材料科学特論 講義資料 (数値解析)

<http://conf.msl.titech.ac.jp/Lecture/ComputationalMaterialsScience/index-numericalanalysis.html>

2023年度Q2 計算材料科学特論 (資料: 英語 + 日本語版) Computational Materials Science 2023 Q2

数値解析に関する講義資料・pythonプログラム (神谷担当分)
Lecture materials on numerical analysis (by Kamiya)

講義で使うプレゼン資料は、Other related programsの下にあります
Lecture presentation slides will be found after the python tips section.

Update News:

- All slides (re-organized): [AllSlides2023E.pdf](#)
Assignments: [AllAssignments.pdf](#)
- July 3, 11:45 Lecture materials on July 4 have been uploaded. ([20230703Summary.zip](#))
- June 30, 12:43 Lecture materials on June 30 have been updated. ([20230630Optimization2.zip](#))
- June 30, 8:45 Lecture materials on June 30 have been uploaded.
- June 24, 11:27 Lecture materials on June 27 have been uploaded. ([20230627LSQEquationOptimize.zip](#))
- June 22, 12:17 Lecture materials on June 23 have been uploaded. ([20230622InterpolateSmoothing.zip](#))
- June 18, 7:57 Lecture materials on June 20 have been uploaded. ([20230618Diffeq.zip](#))
- June 15, 12:47 Lecture materials on June 16 have been uploaded. ([20230616DifferentialIntegration.zip](#))
- June 12, 9:07 Lecture materials on June 13 have been uploaded. ([20230613ComputerAndErrorSources.zip](#))

python ノート

- [起動と対話モード](#)
- [基本変数型](#)
- [変数型\(上級者向\)](#)

June 16, 2023	数値微分 Numerical differentiation 数値積分 Numerical integration	20230616DifferentialIntegration.zip	June 15, 12:48 uploaded
<p>python programs: Numerical differentiation</p> <p>Level *: Effect of x mesh h on numerical differentiation using two-point forward difference method:</p> <p>It compares the differentiation errors for different x mesh h for two-point forward difference method, $df / dx \sim (f(x+h) - f(x)) / h$.</p> <p>Program: download diff_h.py (Source code)</p> <p>Algorithms / Python modules: none</p> <p>Usage: <code>python diff_h.py</code></p> <p>Output: to display and diff_h.csv.</p> <p>What you learn: function definition</p> <p>Level *: Effect of different approximations on numerical differentiation:</p> <p>It compares the differentiation errors for two- to seven-point difference approximations.</p> <p>Program: Download diff_order.py (Source code)</p> <p>Algorithms / Python modules: none</p> <p>Usage: <code>python diff_order.py</code></p> <p>Output: to display, diff_order.csv, and graph</p> <p>Visualization: implemented in the python script</p> <p>What you learn: how to plot graph using matplotlib</p> <p>Level *: Effect of different approximations on numerical differentiation with random noise data:</p> <p>It compares the differentiation errors for two- to seven-point difference approximations.</p> <p>Program: download diff_order_noise.py (Source code)</p> <p>Algorithms / Python modules: two- to seven-point difference approximations</p> <p>Usage: <code>python diff_order_noise.py</code></p> <p>Output: to display, diff_order_noise.csv, and graph</p> <p>What you learn: random functions in random module</p> <p>Level ***: Richardson extrapolation differentiation:</p> <p>Program: download diff_richardson.py (Source code)</p> <p>Algorithms / Python modules: Richardson extrapolation</p>			

参考：結晶工学スクール関連プログラム

<http://conf.msl.titech.ac.jp/Lecture/jsap-crystal/index.html>

2023年度 応用物理学会 結晶工学分科会

結晶工学スクール 関連資料

東京工業大学 科学技術創成研究院 神谷利夫

★注意★ 以下の資料、python プログラムは教育目的で作成したもので、間違いやバグ、計算精度の不足、プログラムの最適化などの保証は全くありません。自己責任でお使いください。

2023年度結晶工学スクール参加者以外への拡散・共有もご遠慮ください。

- pythonは、anaconda 3x を推奨します（インストール方法はこのページ最後をご覧ください）。その他のdistributionでは、python 3.x を選択し、必要なモジュール (numpy, scipyなど) をインストールしてください
- 下記の「レベル」の★の数は、プログラムの難易度を表しています
- *.py のリンクをクリックすると、pythonプログラムのダウンロードができます。うまくいかない場合は、リンクをマウスで右クリックし、ファイルに保存してください。
- (プログラムコード) あるいは (プログラムコード・実行結果) をクリックすると、web browser上でプログラムコードを表示します。
コメントが色付けされているので、プログラムを理解する場合はこちらをご覧ください。

2023年度 結晶工学スクール「バンド構造を用いた材料開発（実践編）」資料

- 神谷講義スライド [Kamiya-Slides2023-1.pdf](#) (2023/7/25 7:15更新)
- 講義資料本文 [20220702Text.pdf](#) (2022/7/28 15:18更新)
付録 [20210805Appendix.pdf](#)

バンド理論

- レベル★ 二次元波動関数の描画 [wavefunction2D.py](#) (プログラムコード・実行結果)

説明：自由電子モデル、箱型ポテンシャル量子ドット、水素原子用モデルの2次元波動関数を描画

使用しているアルゴリズム：なし

Usage1: `python wavefunction2D.py pw kx0 ky0 kz0 kx ky kz`

自由電子。

波数ベクトル(量子数) (kx0, ky0, kz0) (任意実数) をもち、

Bloch波数 (kx, ky, kz) の波動関数 (平面波) を描画

Usage2: `python wavefunction2D.py qbox nx ny nz kx ky kz`

無限大ポテンシャルに閉じ込められた量子ドット。

量子数 (nx, ny, nz) (= 1, 2, 3...)をもち、

Bloch波数 (kx, ky, kz) の□ 波動関数 (正弦・余弦関数) を描画

Usage3: `python wavefunction2D.py H n l m kx ky kz`

水素原子様モデル。

量子数 (n, l, m) (n = 1, 2, ..., l = 0, 1, ..., n-1, m = -l, -l+1, ..., l-1, l) をもち、

Bloch波数 (kx, ky, kz) の波動関数 を描画。見やすいように、動径関数のサイズは適当に変えている

実行方法： `python wavefunction2D.py pw 0.1 0.0 0.0 0.0 0.0 0.0`

波数ベクトル (0.1, 0, 0), Bloch波数 (0, 0, 0) の平面波波動関数を描画

実行方法： `python wavefunction2D.py pw 0.1 0.0 0.0 0.5 0.0 0.0`

波数ベクトル (0.1, 0, 0), Bloch波数 (1/2, 0, 0) の平面波波動関数を描画

実行方法： `python wavefunction2D.py pw 0.1 0.0 0.0 1.0 0.0 0.0`

波数ベクトル (0.1, 0, 0), Bloch波数 (1, 0, 0) の平面波波動関数を描画

実行方法： `python wavefunction2D.py qdot 1 1 1 0.0 0.0 0.0`

量子数 (1, 1, 1) の量子ドットの波動関数を描画

実行方法： `python wavefunction2D.py H 1 0 0 0.0 0.0 0.0`

H 1s (n = 1, l = 0, m = 0)の水素用原子の波動関数を描画

実行方法： `python wavefunction2D.py H 2 1 1 0.0 0.0 0.0`

H 2p_x (n = 2, l = 1, m = 1)の水素用原子の波動関数を描画

- レベル★★ 三次元自由電子バンド [free_electron_band.py](#) (プログラムコード・実行結果)

説明：自由電子モデル (ゼロポテンシャル) による三次元バンド構造

使用しているアルゴリズム：なし

実行方法： `python free_electron_band.py`

資料： [pw.pdf](#)

- レベル★★★ 平面波基底による一次元バンド計算 [pw1d.py](#) (プログラムコード・実行結果)

説明：平面波基底と井戸型ポテンシャルによる一次元バンド構造

使用しているアルゴリズム：フーリエ変換 (numpy.fft.fft()), エルミート行列の対角化 (numpy.linalg.eig())

Usage: `python pw1d.py mode (args)`

まとめに替えて: 質問やリクエスト受け付けます

- ・ 本チュートリアルに関連した質問等、気楽にご相談ください。
- ・ 本チュートリアルコースは、D2MatEの活動の一環として来年度以降も継続して行っています。

質問、チュートリアル・講演会などのリクエストがあれば、
kamiya.t.aa@m.titech.ac.jp
宛にお送りください。

カバー可能なテーマ:

計算科学、データ科学、プログラミング
材料科学、半導体工学

その他

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-00E%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 tkamiya@ms.titech.ac.jp へ報告をお願いします

1. OS (Windows10, OS Xなど) およびバージョン
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。

関数で複数の値を返す、他の方法: 推奨しない

実は、引数がリスト変数やオブジェクトの場合、その要素・メンバーにいて値を返せる:
どのような変数型であれば使えるかは、mutable, immutable について調べてください

```
def f(x, ret_list):  
    ret_list.append(x)  
    ret_list.append(x * x)  
  
ret_list = []  
f(3.0, ret_list)  
print("return values:", ret_list)
```

引数 **ret_list** が関数内で (予期せずに) 変更されるので、トラブルの元
副作用 はなるべく発生させないほうがいい

強く推奨: 関数の入力と戻り値を明確に分離 副作用を作らない

良いプログラムの構造:

- 入力と出力が明確に区別されている
- 入力変数は変更しない
- 関数の結果は戻り値で受け取る
- 副作用がない: 関数の結果は、入力の値だけで決まる
(関数内に定数のフラグなどを作らない)

この規則を守ると、プログラムを 入力 => 出力のチェーンで組み立てられる

```
output1 = func1(input1)
```

```
output2 = func2(input1, output1)
```

```
output3 = func3(input1, output2)
```


関数のチェーンとしてプログラムを書きなおす

01d_fitting_gpt_rev2_functions.py

```
def main():
```

```
    input_path, output_path = initialize()
```

```
    T, N = read_file(input_path)
```

```
    Tinv, logN = convert(T, N)
```

```
    print_input_data(T, N, Tinv, logN)
```

```
    a, b = lsq(Tinv, logN, norder = 1)
```

```
    print_lsq_result(a, b)
```

```
    logNcal = cal_fitting_curve(Tinv, a, b)
```

```
    save_result(output_path, T, N, Tinv, logN, logNcal)
```

```
    plot_result(Tinv, logN, logNcal)
```

```
#=====
```

```
# main()関数を実行
```

```
#=====
```

```
main()
```

main()関数から実行するトリック

main()

を呼び出してもいい

(呼び出す関数を定義した後で呼び出さないと、関数名が未定義エラーになる)

豆知識:

- ・ pythonスクリプトは、そのままライブラリとして使える
- ・ ライブラリとして呼び出された場合も、スクリプトファイルの最初から最後まで実行する
=> 上の例だと、main()関数も呼び出してしまう

__name__: ライブラリとして呼び出す場合に
不要なコードを実行しないようにするテクニック

fitting-gpt-rev2-functions.py:

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

Pythonのお約束: `__name__` 変数

Pythonでは、スクリプトファイルが実行されるとき、`__name__` 変数が設定される。

- ・ スクリプトファイルがモジュールとして読み込まれる場合: モジュール名
- ・ スクリプトファイルが直接実行される場合: `'__main__'`

`__name__` 変数の内容により、
スクリプトが直接実行された場合とモジュールとして読み込まれた場合で、動作を変えることができる。

```
def main():  
    print("executed directly")
```

```
If __name__ == '__main__':  
    main()
```

とすると、モジュールとして読み込まれたときには `main()` は実行されないが
直接実行された場合は `main()` が実行される。

モジュールを開発している際に、モジュール単体を実行して動作確認をする場合などに使う。

Pythonのエラー処理: try ~ except

Pythonのエラー処理はかなり厳しい

以下のように、頻繁に発生するエラーのすべてでプログラムの実行が終了する

- ・ `int(str)` で文字列を整数型に変換しようとしたときに、文字列の書式が整数でなかった場合
- ・ リストの要素を `a[i]` で取得しようとしたときに、`i` が `a[]` の要素範囲外だった時
- ・ 未定義値の変数 (`None`) や変換できない型を含む演算
(`str => float`などの変換は自動的に実行されない)

参考: *perl* は、程度の低いエラーには寛容

- ・ `int(str)` での文字列書式エラー: できる限り整数型に変換する。だめな時は 0 を返す
- ・ リストの範囲外の要素にアクセスした場合: `undef` (未定義値) を返す
- ・ `undef` や未定義の変数を含む演算: 未定義値は 0 として演算を実行
- ・ 可能な限り、型の変換は自動的に行われる (“3.0” + 1 は 4.0になる)

Pythonでエラーが起こったときにプログラムを停止しないためには、try ~ except 構文を使う

まず try ブロックを実行し、エラー (例外、exception) が発生したら、exceptブロックを実行する

```
def pfloat(str):  
    try:  
        # float(str) を実行してみる  
        # str が浮動小数点として変換できる文字列であれば、変換した値を返す  
        # str が浮動小数点として変換できない場合、例外が発生して except ブロックを実行する  
        return float(str)  
    except:  
        # str が浮動小数点として変換できない場合、None (未定義値) を返す  
        return None
```

getarg() 関数

Pythonの問題: 起動時引数を取得する sys.argv リスト変数は、範囲外の index を渡すとエラーになってプログラムが終了する

Perlの場合: リストに範囲外のindexを渡しても、undef が返ってくる。実行は継続。

引数で初期値を更新する関数 getarg() では、どのようなことが想定されるか

1. 何番目の引数か、**index (position) を渡し**、その値を return で返す
2. 引数を与えられない場合 => 初期値をそのまま使う
引数のindexの他、**初期値も渡す必要**がある
3. 引数を与えられた場合 => 初期値を引数で置き換える
4. 初期値を与えない場合は None を返す
初期値を与えないで getarg(position) を呼び出せるようにし、この際には初期値を Noneにする。
“デフォルト引数” 機能を使う

getarg()の引数としては、position と defval を渡す

defval はデフォルト引数となっているので、

getarg(position) と呼び出されたら、defvalには Noneが代入される

```
def getarg(position, defval = None):
```

```
    try:
```

sys.argv[position] が存在したら sys.argv[position] を返す

```
        return sys.argv[position]
```

```
    except:
```

sys.argv[position] が存在しなかったら、defval を返す

```
        return defval
```

その他: ChatGPT

ChatGPTの話題 (2023/4現在)

2023/3/29

「GPT-4より強力なAIの開発を直ちに停止せよ」——公開書簡にマスク氏やウォズニアック氏が署名

<https://www.itmedia.co.jp/news/articles/2303/29/news180.html>

2023/4/1

ChatGPTの「開発停止」を求めた専門家たちからの“警告”の意味

<https://wired.jp/article/chatgpt-pause-ai-experiments-open-letter/>

2023/4/1

「ChatGPT」イタリアで一時使用禁止に データ収集で違反の疑い

<https://www3.nhk.or.jp/news/html/20230401/k10014026391000.html>

「チャットGPT」使用巡り科学誌の対応割れる… サイエンスは禁止、ネイチャーは明記求める

<https://www.yomiuri.co.jp/science/20230406-OYT1T50193/>

「チャットGPTの利用は、極めて慎重に考えるべきだ。ルールを緩めるのは厳しくするよりもはるかに容易だ」。サイエンス誌のホールデン・ソープ編集長は、3月にワシントンで開かれた米国科学振興協会の年次総会で見解を述べた。

同誌は1月、チャットGPTや他のAIによる論文の執筆を禁じる方針を示した。同誌は、論文が著者による独自の成果物であることを求めており、**AIが生成した文章は「AIからの盗用で、不正に該当する」**と説明している。

一方、英科学誌ネイチャーは1月、利用禁止までは踏み込まなかったが、**AIを共著者として認めず、使用した場合は論文中に明記**することを求めた。米科学誌セルも、3月8日に出した声明で、「論文の読みやすさなどを改善するツールは認めている」とし、ネイチャー誌と同様の対応をとることを明らかにした。（ワシントン支局 富山優介）

SamsungのエンジニアがChatGPTに社外秘のソースコードを貼り付けるセキュリティ事案が発生

2023/4/10 <https://gigazine.net/news/20230410-samsung-chatgpt-security-leak/>

- Samsungデバイスソリューション(DS)部門事業署内では、ChatGPTの使用が許可されていたとのこと。
- Samsung側はChatGPTの使用を許可するに際して、「社内情報セキュリティに注意し、私的な内容を入力してはいけない」という告知を出していました。
- それにもかかわらず、従業員は半導体の設備計測用プログラムのソースコードをChatGPTに入力し、コードの修正を依頼してしまったとのこと。
- 別の従業員は歩留まり計算用プログラムのソースコードをすべてChatGPTに入力し、コードの最適化を図った。

無料のChatGPT (GPT3.5系モデル)

2021年までのデータで学習しているということだったが、更新されている情報もある様子
**以前は情報がない場合でも、もっともらしい理由と一緒に
堂々とうそをついていたが、改善されているかもしれない**

質問例: 現在の日本の首相は誰で、どのような政策をとっていますか

回答: 2023年4月7日時点での日本の首相は、情報が更新されている場合を除き、菅義偉(すが・よしひで)首相です。菅首相は、2020年9月に安倍晋三(あべしんぞう)前首相の後任として就任しました。

質問例: ドイツの首相は誰ですか？

回答: 2023年4月7日時点でのドイツの首相は、オラフ・ショルツ(Olaf Scholz)です。ショルツは、2021年12月8日に行われたドイツ連邦議会選挙で、所属するドイツ社会民主党(SPD)が勝利したことにより、同年12月15日に首相に就任しました。ショルツ首相は、前首相のアンゲラ・メルケル(Angela Merkel)氏に代わり、ドイツの21世紀第1の首相となります。

Chat-GPT: 2023/3/22には大嘘を回答

質問: tkinterでhtmlを表示する方法を教えてください

2023/3/22の回答: TkinterのTextウィジェットは、HTMLを表示することができます。この例では、ScrolledTextウィジェットを作成し、htmlというHTMLコードをTextウィジェットに挿入しています。configureメソッドを使用して、テキストのフォントを設定しています。HTMLを挿入することで、TextウィジェットでHTMLのタグが解釈され、適切なスタイルが適用されます。

自信をもって「できます」と答えていますが、大嘘です

```
import tkinter as tk
import tkinter.scrolledtext as scrolledtext
import tkinter.font as tkfont
import tkinter.constants as tkc

root = tk.Tk()

# スクロール可能なテキストウィジェットを作成
text = scrolledtext.ScrolledText(root, wrap=tkc.WORD)
text.pack(fill=tkc.BOTH, expand=True)

# HTMLを設定
html = """<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>HTML in Tkinter Text Widget</title>
</head>
<body>
<h1>Heading 1</h1>
<p>This is a paragraph.</p>
<ul>
<li>List item 1</li>
<li>List item 2</li>
<li>List item 3</li>
</ul>
</body>
</html>

```

実行結果

```
tk
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>HTML in Tkinter Text Widget</title>
</head>
<body>
<h1>Heading 1</h1>
<p>This is a paragraph.</p>
<ul>
<li>List item 1</li>
<li>List item 2</li>
<li>List item 3</li>
</ul>
</body>
</html>
```

Chat-GPT: 2023/4/6にはある程度改善

質問: tkinterでhtmlを表示する方法を教えてください

2023/4/6の回答: tkinterでは、HTMLを表示するためのネイティブなウィジェットはありませんが、Webviewを使ってHTMLを表示することができます。Webviewは、Tkinterウィンドウ内に埋め込まれたWebブラウザです。

以下は、TkinterでWebviewを使ってHTMLを表示するための基本的なコード例です。

```
import tkinter as tk
import tkinter.ttk as ttk
import webview
```

```
root = tk.Tk()
```

```
# Webviewウィジェットを作成します
web = ttk.Frame(root, width= 600, height=400)
web.pack()
```

```
# WebviewウィジェットにHTMLをロードします
webview.create_window("My WebView", html="<h1>Hello, World!</h1>", js_api=None, width=600, height=400)
```

```
root.mainloop()
```

間違い。webview.start(gui="qt") を呼び出す

3/22の回答よりはましになったが、まだ動くコードは回答できない。

「webview」を使えばいいというヒントが得られたので、GoogleやQuoraなどの知恵袋系で検索すればもっとヒントが得られる

検索結果例: <https://self-development.info/%E3%80%90python%E3%80%91web%E3%82%B9%E3%82%AD%E3%83%AB%E3%81%A7gui%E3%82%A2%E3%83%97%E3%83%AA%E3%82%92%E9%96%8B%E7%99%BA%E3%81%A7%E3%81%8D%E3%82%8Bpywebview/>

Chat-GPT

コメントのないプログラム (peakfit-tutorial.py) を放り込んでみる

```
import numpy as np
from numpy import exp
import pandas as pd
from scipy.optimize import minimize

method = "nelder-mead"

#=====
# Source parameters to be fitted
#=====
infile = 'peak.xlsx'
nx = None

# optimization parameters
func = '2.0 * (x[0] - 3.0)**2 + (x[1] - 1.0)**4 + 2.0'
x0s = [1.3, 0.6, 0.1]

maxiter = 100
tol = 1.0e-5

#=====
# functions
#=====
def cal_y(x, params):
    I0 = params[0]
    x0 = params[1]
    w = params[2]
    a = 0.832554611 / w
    X = a * (x - x0)
    return I0 * exp(-X * X)

def ycal_list(xd, params):
    y = []
```

回答

このプログラムは、与えられた関数を最小化するパラメータを見つけるために、**シンプレックス法**を使用しています。具体的には、**Excelファイルからデータを読み取り、与えられた関数によって予測される値と実際の値の差を最小化するようなパラメータ**を見つけています。

プログラムの流れは以下の通りです。

モジュールのインポート:

パラメータの設定:

関数の定義: **パラメータを与えられた関数に代入して、予測される値を計算する関数を定義**します。

最小化する関数の定義: **与えられたパラメータに対して、予測されるyの値と実際のydの値の差を計算し、それらの二乗和を求める関数を定義**します。

メインルーチンの定義: Excelファイルからデータを読み取り、最小化するパラメータを見つけるために、**scipy.optimizeのminimize関数を使用してシンプレックス法を実行**します。最適化の結果を出力します。

このプログラムは、Excelファイルから読み込んだデータに対して、与えられた関数を最小化するパラメータを見つける例です。**実際の応用においては、関数を適切に定義し、最小化すべきパラメータを設定する必要があります。また、データの前処理や後処理も必要になる場合があります。**

Chat-GPT

Launcher.pyのスクリプト (AIが絶対に知らないプログラム言語) を放り込んでみる

```
[Button1.Spectrum Analysis]  
Caption=Deconvolution
```

```
join_path spectrum_dir $(tkprog_X_path) spectrum  
join_path script $(spectrum_dir) deconvolution.py
```

```
:check_exist $(script) "$[ $(script) ] が見つかりません。インストールしてください"
```

```
read_ini $i "spectrum" "work_dir" last_dir "  
chdir $(last_dir)
```

```
set_if_blank mode mode=gs  
set_if_blank x_label x_label=0  
set_if_blank y_label y_label=1  
set_if_blank xmin xmin=-1e100  
set_if_blank xmax xmax=1e100  
set_if_blank func_type func_type=gauss  
set_if_blank fwhm fwhm=0.12  
set_if_blank zeroc zeroc=0  
set_if_blank dump dump=1.0  
set_if_blank nsmoothiter nsmoothiter=5  
set_if_blank nmaxiter nmaxiter=100  
set_if_blank eps eps=1.0e-4
```

```
get_open_file_name "*.xlsx" "." Choose PES Excel(.xlsx) file  
set input_path=$o  
get_directory $o fdir  
get_filebody $o fbody  
get_filename $o fname  
get_last_directory $o last_dir
```

```
read_labels $(input_path) labels
```

```
set_title Deconvolution:$(last_dir)
```

```
new_dialog
```

回答

It looks like this is a script written in a programming language, possibly Python, that performs deconvolution of spectra. The script first sets up some variables related to the input data, such as the file path and labels. It then prompts the user to choose a Python executable and a Python script file to use for the deconvolution. Finally, it creates a dialog box for the user to enter additional parameters for the deconvolution, such as the type of function to use and the number of iterations to perform.

プログラムの解釈の結果では・・・

入力したプログラムがきちんと self-explanatory になっていれば、かなりの精度で、理解を助ける回答が得られる

- ・ 質問の仕方
- ・ 与える情報

について、質の高い情報を与えることで、的確な回答が得られる可能性は高まる

- ・ ChatGPTでも理解できるプログラムを書く
- ・ 英訳の場合、英訳しやすい日本語文に書き換える。
など

この点でも、使う人間の「知識」、「語学力」、「スキル」が問われる

Chat-GPT

質問: $df(t)/dt = 1 / (1 + t)$ を数値積分で解くpythonプログラムを教えて

回答されたコード: ちゃんと動きます

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# 常微分方程式を定義する
def model(y, t):
    dydt = 1 / (1 + t)
    return dydt

# 初期値を設定する
y0 = 0

# 時間範囲を設定する
t = np.linspace(0, 10, 101)

# 常微分方程式を解く
y = odeint(model, y0, t)

# 結果をプロットする
plt.plot(t, y)
plt.xlabel('t')
plt.ylabel('y')
plt.show()
```


Chat-GPT

質問: python tkinterでフローティングチップスを表示する方法を教えてください

回答されたコード: ちゃんと動きます

```
import tkinter as tk
```

```
class Tooltip:
```

```
    def __init__(self, widget, text):
```

```
        self.widget = widget
```

```
        self.text = text
```

```
    # チップウィンドウを作成
```

```
    self.tipwindow = tk.Toplevel(widget)
```

```
    self.tipwindow.wm_overrideredirect(True)
```

```
    self.tipwindow.wm_geometry('150x30+1+1')
```

```
    # ラベルを作成
```

```
    self.label = tk.Label(self.tipwindow, text=self.text, justify=tk.LEFT, background='#ffffe0', relief=tk.SOLID,  
borderwidth=1)
```

```
    self.label.pack(ipadx=1)
```

```
    # トリガーとなるイベントにコールバック関数を設定
```

```
    self.widget.bind('<Enter>', self.enter)
```

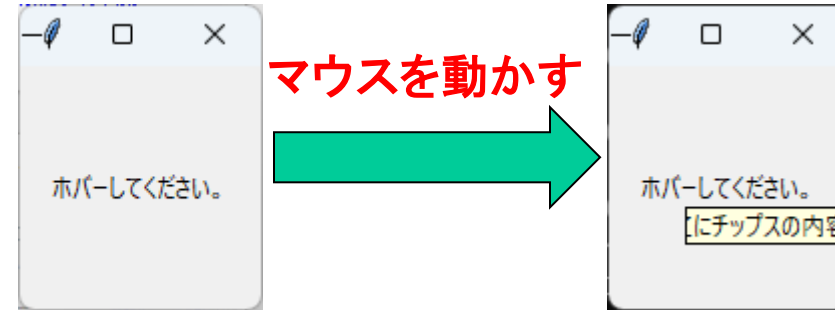
```
    self.widget.bind('<Leave>', self.leave)
```

```
    def enter(self, event):
```

```
        # チップウィンドウの位置を設定
```

```
        x = y = 0
```

```
        x, y, cx, cy = self.widget.bbox('insert')
```



リファクタリング途中結果

ChatGPTに書き直してもらおう

以下のプログラムを読みやすく書き直してください

```
import openpyxl
import numpy as np
import matplotlib.pyplot as plt
workbook = openpyxl.load_workbook('input.xlsx')
sheet = workbook.active
c = []; d = []
for row in sheet.iter_rows(min_row=3, values_only=True):
    c.append(row[0])
    d.append(row[2])
e = [1 / g for g in c]
f = [np.log(h) for h in d]
output_workbook = openpyxl.Workbook()
output_sheet = output_workbook.active
output_sheet.append(["T", "N", "1/T", "log(N)"])
for g, h, i, j in zip(c, d, e, f):
    output_sheet.append([g, h, i, j])
output_workbook.save("fitting.xlsx")
plt.scatter(e, f)
plt.show()
```

ChatGPTの回答

以下のコードを読みやすくリファクタリングしました。コード内にコメントも追加しています。

-- ここに修正したコード --

このコードでは、変数名を具体的でわかりやすいものに変更し、コメントを追加して可読性を向上させました。また、出力ファイルを保存する際にファイル名を `fitting.xlsx` に指定し、最後にデータを散布図としてプロットしました。

ChatGPTの修正プログラム: **dirty-GPT-rev1.py**

```
import openpyxl
import numpy as np
import matplotlib.pyplot as plt

# 入力ファイルを読み込む
workbook = openpyxl.load_workbook('input.xlsx')
sheet = workbook.active

# データを格納するためのリストを初期化
c_values = []
d_values = []

# シートからデータを取得し、リストに追加
for row in sheet.iter_rows(min_row=3, values_only=True):
    c_values.append(row[0])
    d_values.append(row[2])
```

```
# データの変換と計算
e_values = [1 / c for c in c_values]
f_values = [np.log(d) for d in d_values]

# 出力用のワークブックとシートを作成
output_workbook = openpyxl.Workbook()
output_sheet = output_workbook.active

# ヘッダーを追加
output_sheet.append(["T", "N", "1/T", "log(N)"])

# データを出力シートに追加
for g, h, i, j in zip(c_values, d_values, e_values, f_values):
    output_sheet.append([g, h, i, j])

# 出力用のファイルを保存
output_workbook.save("fitting.xlsx")

# データを散布図としてプロット
plt.scatter(e_values, f_values)
plt.show()
```

ChatGPTに書き直してもらう Ver2

次のプログラムは、`input.xlsx`の1列目から温度 T 、2列目からキャリア濃度 N を読み込み、 $1/T$ と $\log(N)$ に変換し、その結果を`fitting.xlsx`に保存し、グラフを描くプログラムです。以下のように読みやすく書き直してください

- ・ 全体のリファクタリングをしてください
- ・ 変数名をわかりやすく置き換えてください
- ・ 入力ファイル名、出力ファイル名を変数に入れてください

```
import openpyxl
import numpy as np
import matplotlib.pyplot as plt
workbook = openpyxl.load_workbook('input.xlsx')
sheet = workbook.active
c = []
d = []
for row in sheet.iter_rows(min_row=3, values_only=True):
    c.append(row[0])
    d.append(row[2])
e = [1 / g for g in c]
f = [np.log(h) for h in d]
output_workbook = openpyxl.Workbook()
output_sheet = output_workbook.active
output_sheet.append(["T", "N", "1/T", "log(N)"])
for g, h, i, j in zip(c, d, e, f):
```

ChatGPTの回答 Ver2

以下は、プログラムのリファクタリングと変数名のわかりやすい置き換え、ファイル名の変数化を行ったものです。また、コードを関数に分割して可読性を向上させました。

-- ここに修正したコード --

このリファクタリングにより、コードがより読みやすくなり、変数名もわかりやすくなりました。また、処理が関数に分割され、各機能が独立して理解しやすくなりました。

ChatGPTの修正プログラム Ver2: **dirty-GPT-rev2.py**

```
import openpyxl
import numpy as np
import matplotlib.pyplot as plt

# 入力ファイルと出力ファイルのファイル名を変数に格納
input_filename = 'input.xlsx'
output_filename = 'fitting.xlsx'

def read_data(filename):
    workbook = openpyxl.load_workbook(filename)
    sheet = workbook.active

    temperatures = []
    concentrations = []

    for row in sheet.iter_rows(min_row=3, values_only=True):
        temperatures.append(row[0])
        concentrations.append(row[1])

    return temperatures, concentrations
```

```
def transform_data(temperatures, concentrations):
    inv_temperatures = [1 / T for T in temperatures]
    log_concentrations = [np.log(N) for N in concentrations]

    return inv_temperatures, log_concentrations

def save_to_excel(temperatures, concentrations,
inv_temperatures, log_concentrations, output_filename):
    output_workbook = openpyxl.Workbook()
    output_sheet = output_workbook.active

    output_sheet.append(["T", "N", "1/T", "log(N)"])

    for T, N, inv_T, log_N in zip(temperatures,
concentrations, inv_temperatures, log_concentrations):
        output_sheet.append([T, N, inv_T, log_N])

    output_workbook.save(output_filename)
```


ChatGPTの修正プログラム Ver2: **dirty-GPT-rev2.py**

```
def plot_data(inv_temperatures, log_concentrations):  
    plt.scatter(inv_temperatures, log_concentrations)  
    plt.show()  
  
# データ読み込み  
temperatures, concentrations = read_data(input_filename)  
  
# データ変換  
inv_temperatures, log_concentrations = transform_data(temperatures, concentrations)  
  
# データ保存  
save_to_excel(temperatures, concentrations, inv_temperatures, log_concentrations, output_filename)  
  
# データプロット  
plot_data(inv_temperatures, log_concentrations)
```

ChatGPTに書き直してもらう Ver3

次のプログラムは、input.xlsxの1列目から温度 T 、2列目からキャリア濃度 N を読み込み、 $1/T$ と $\log(N)$ に変換し、その結果をfitting.xlsxに保存し、グラフを描くプログラムです。以下のように読みやすく書き直してください

- ・ 全体のリファクタリングをしてください
- ・ 変数名をわかりやすく置き換えてください
- ・ 入力ファイル名、出力ファイル名を変数に入れてください
- ・ グラフに凡例、xラベル、yラベルを追加してください

```
import openpyxl
import numpy as np
import matplotlib.pyplot as plt
workbook = openpyxl.load_workbook('input.xlsx')
sheet = workbook.active
c = []
d = []
for row in sheet.iter_rows(min_row=3, values_only=True):
    c.append(row[0])
    d.append(row[2])
e = [1 / g for g in c]
f = [np.log(h) for h in d]
output_workbook = openpyxl.Workbook()
output_sheet = output_workbook.active
output_sheet.append(["T", "N", "1/T", "log(N)"])
```

ChatGPTの回答 Ver3

以下のコードは、リファクタリングされ、変数名がわかりやすく置き換えられ、入力ファイル名と出力ファイル名が変数に格納され、グラフに凡例と軸ラベルが追加されました。

-- ここに修正したコード --

このコードでは、可読性が向上し、変数名がわかりやすくなり、凡例と軸ラベルがグラフに追加されました。

ChatGPTの修正プログラム Ver3: **dirty-GPT-rev3.py**

```
import openpyxl
import numpy as np
import matplotlib.pyplot as plt

# 入力ファイル名と出力ファイル名を変数に格納
input_filename = 'input.xlsx'
output_filename = 'fitting.xlsx'

def read_data(filename):
    workbook = openpyxl.load_workbook(filename)
    sheet = workbook.active

    temperatures = []
    concentrations = []

    for row in sheet.iter_rows(min_row=3, values_only=True):
        temperatures.append(row[0])
        concentrations.append(row[1])

    return temperatures, concentrations
```

```
def transform_data(temperatures, concentrations):
    inv_temperatures = [1 / T for T in temperatures]
    log_concentrations = [np.log(N) for N in concentrations]

    return inv_temperatures, log_concentrations

def save_to_excel(temperatures, concentrations,
inv_temperatures, log_concentrations, output_filename):
    output_workbook = openpyxl.Workbook()
    output_sheet = output_workbook.active

    output_sheet.append(["T", "N", "1/T", "log(N)"])

    for T, N, inv_T, log_N in zip(temperatures,
concentrations, inv_temperatures, log_concentrations):
        output_sheet.append([T, N, inv_T, log_N])

    output_workbook.save(output_filename)
```

ChatGPTの修正プログラム Ver3: **dirty-GPT-rev3.py**

```
def plot_data(inv_temperatures, log_concentrations):  
    plt.scatter(inv_temperatures, log_concentrations, label='Data Points')  
    plt.xlabel('1/T')  
    plt.ylabel('log(N)')  
    plt.legend()  
    plt.show()  
  
# データ読み込み  
temperatures, concentrations = read_data(input_filename)  
  
# データ変換  
inv_temperatures, log_concentrations = transform_data(temperatures, concentrations)  
  
# データ保存  
save_to_excel(temperatures, concentrations, inv_temperatures, log_concentrations, output_filename)  
  
# データプロット  
plot_data(inv_temperatures, log_concentrations)
```