理工系大学生のための Python 入門

幸谷智紀

https://na-inet.jp/python/

https://github.com/tkouya/intro_python

2025 年 1 月 17 日

初めに

本書は，大学理工系学部・学科で学ぶ人に向けて書き下ろした，Python(パイソン) というプログラミング言語の使い方を概説した入門書です。大型書店に行けばコンピュータ関係コーナーには分厚い本がいっぱい並んでいますし，ちょっと検索してみれば優れた Python の解説ページが沢山見つかります。既に本や解説ページで Python を勉強した人や，小・中・高校の授業・クラブ活動の一環で Python だけでなく Scikit-learn, TensorFlow，Pytorch といった機械学習，AI モジュールを使ったことのある人もいらっしゃると思います。こういう人は「Python 上級者」と呼ぶことにしましょう。一方，実は Scratch 以外の，いちいちテキストを打ち込んで一行一行実行するためのプログラム（スクリプト）を作っていくという体験をしたことのない人（「Python 未経験者」）も，まだいらっしゃるはずです。そこまで極端でなくても，ある種の作業のために自分で自作のプログラムを自主的に作ったことのない人（「Python 未習熟者」）は未だ大部分を占めるという状況である今（2023 年現在），本書はそのような「Python 未習熟者」をメインターゲットに，Python やそれを取り巻くエコシステム (モジュール群) を体験する目的で執筆されたのものです。

類書が多数ある中で，あえて本書を書き下ろしたのは，「理工系学部生向け」の，特に本書で取り上げる NumPy, SciPy, matplotlib, Flask といった，基盤的モジュール群を一通り体験するものが見当たらなかった，ということが一番の理由です。この中でも NumPy と SciPy については，「Python 数値計算プログラミング」（講談社）で，数値計算アルゴリズムと共に解説してありますが，本書はその手前の段階の解説がメインです。具体的に言うと

1. Python プログラムの作り方と実行方法
2. Python の文法
3. NumPy の使用例
4. SciPy の概要と使用例
5. matplotlib の概要と使用例
6. Pandas の概要と使用例
7. Flask の概要と使用事例

を 12〜14 回の講義（「授業」という言い方を当方は好みません）で無理やりでも理解してもらう，ということになります。Pytho 上級者にとっては見知った内容なので簡単かもしれませんが，そういう方は是非とも詰込み的な内容で苦労している未経験者や未習熟者の方にアドバイスをして上げて下さい。本書はそのような，一定数の上級者のヘルプを前提に，未経験者を習熟手前まで，未習熟者を上級者に引き上げるためのテキストなのです。

本書では「理工系」学生を前提に，高校数学＋大学教養課程（微分積分と線形代数）の知識を持っていること前提とした内容を幾分含んでいますので，その点も一般の人向けの入門書とは趣が異なります。今後勉強す

る専門的な内容も，本書を通じて学んだ Python の知識を利用し，どんどんプログラミングしながらモノにしていっていただければ，著者としてこれに勝る幸せはありません。その結果，「受講生が Python 使って楽してレポート書いている」という先生方からの悪評が聞こえてきやしないかと，今から期待して止みません。

2025 年 1 月 17 日
遠州茶畑のど真ん中にて
幸谷 智紀
https://na-inet.jp/python/

目次

第 1 章

Python と Python エコシステム

Python(パイソン) は数多あるコンピュータ言語 (computer language) の一つですが，特徴の一つとして，様々な機能がモジュール (module) という形でユーザに提供されているということが挙げられます。これらの多様なモジュールを含めた Python 環境を Python エコシステム (Python ecosystem) と呼びます。現在では Python 以外の言語，例えば Java, Node.js(JavaScript), Julia, Rust，R, MATLAB でも，エコシステムを提供していますので，用途と開発環境に合ったものを選んで使うことになりますが，現状，基本無料で人工知能関係の最新のモジュールをすぐに使えるエコシステムとしては，Python を凌駕するものは現れていません。本章では Python の実行方法と，エコシステムの概要を解説します。

1.1 Python の出所

Python についての最新情報は Python ソフトウェア財団 (The Python Software Foundation, PSF と略記) が主催している Web サイト

`https://python.org/`

に掲載されています。Python は数あるコンピュータ言語の一つで，最新の言語使用を備えたインタープリタ (interpreter) に，テキストとして記述されたプログラム，すなわち Python スクリプトを解釈させて実行するものです。この Python インタプリタは Windows, macOS, Linux 等，さまざまな OS 環境上で動作するものが無料で利用でき，PSF のサイトからダウンロードできます。

また，Python インタプリタと共に使用できる拡張機能は，Python モジュールと呼ばれ，PSF からリンクのある PyPI

`https://pypi.org/`

で公開されており，後述するように pip コマンド (pip モジュール) で自動的にネット経由で PyPI からダウンロードされてインストールすることができます。

なお，本書で対象とする理工系基本モジュールである NumPy, SciPy, Matplotlib は CONDA いう別のパッケージとしてまとめられており，そちらを使っての開発も可能ですが，本書では pip を用いたモジュールの追加を推奨しています。後述する Web サイト開発用の Flask, Tkl.TK をベースとした GUI 開発用の Tkinter も pip コマンドで同様にインストールできますし，今後使用するであろうモジュールも，基本 pip でインストールされることを前提としていることが多いからです。

また，クラウド型の開発環境 Google Colabo や，Jupyter Notebook 等，Web ブラウザ上で Python を実行できる便利な環境も魅力的です。とはいえ，なるべく一つの完成したスクリプトを，必要に応じて作ることが出来る能力を身に着けて頂きたいというのが本書の狙いでもありますので，以下の解説は，恐らくは一番使用するユーザ数の多いと思われる Windows 11 上にインストールされた Python 環境を前提に解説を行っていくことにします。

1.2 Python ことはじめ

まずは使用する Python 開発環境を，自分の Note PC や Desktop PC に整えて下さい。インストール方法についてはバージョンや OS に依存しますので，別資料として提示することにして，以下，インストール作業が終わって Python スクリプトが動く環境はできているものとします。

では，適切なテキストエディタ，できれば Python の文法を理解し，キーワードを色分けして表示してくれるもの（Visual Studio Editor など）を使って，下記のスクリプトを作り，適切なフォルダ（ディレクトリ）に「hellow.py」というファイル名で保存します。

ソースコード 1.1 hellow.py

```
1 # hellow.py: 最初のPython スクリプト
2 print('Hellow,␣Python!')
```

保存したフォルダをカレントとして Python インタプリタ（「python」というコマンド名）を起動し，下記のように正常に実行されることを確認して下さい。

```
$ python hellow.py ← Python インタプリタで「hellow.py」スクリプトをを実行する
Hellow, Python! ← 2 行目の「print('hellow, Python!')」が実行される。
$ ←プロンプトが戻る
```

もし，スクリプトにエラー (error, 間違い) があれば

```
$ python hellow.py
    File "hellow.py", line 2 ← 「hellow.py」スクリプト 2 行目に
      print('Hellow, Python!') ← 2 行目の内容
  IndentationError: unexpected indent ←インデントエラー（IndentationError）「意図しないインデント」（unexpected indent）が発生
$ ←プロンプトが戻る
```

のように，インタプリタがエラー内容を表示してくれているはずです。エラー内容はミスに応じて変わりますので，今後はエラーメッセージをよく読んで対応できるようになって下さい。

ここで新しく学んだ Python の内容は下記の 3 つです。

1. コメント文: #(シャープ) 以降は Python スクリプトは処理しない「コメント文」として扱う
2. print 関数: カッコ内の内容を画面（コンソール）に表示する
3. 文字列：'(シングルクォテーション) で囲まれた部分は「文字列」というデータ型として処理される

コメント文は処理内容と関係のないメモを書いておくためのものです。これから沢山のスクリプトを作っていきますので，忘れないようにコメント文にメモを書いて置く習慣をつけておきましょう。読み返した時に処理

内容を思い出しやすくなります。

では次の例題に取り組んでみましょう。

例題 1.1

コンソール画面に，ローマ字表記の自分の名前を表示するスクリプト「print_name.py」を作って下さい。

ソースコード 1.2 print_name.py

```
1 # print_name.py: 自分の名前を表示
2 print('Tomonori␣Kouya')
```

この結果,

print_name.py の実行結果

```
$ python print_name.py
Tomonori Kouya
```

となれば成功です。

問題 1.1

次の文字列をコンソール画面に表示するスクリプト「print_number.py」を作って下さい。

```
$ python print_number.py
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

1.3 対話モードによる実行

Python は,「python」という実行ファイルが，指定されたスクリプトを読み込み，スクリプトの記述に従ってコンピュータに実行できる形で命令を発行するコンピュータ言語です。人間が直接記述できるテキスト形式で記述されたスクリプト（＝プログラム）を読み込みつつ実行するこの実行ファイルのことをインタプリタと呼びます。

Python インタプリタは一行ずつ命令を読み込みながら実行する「対話モード」の機能があります。例えば前の例に示した hellow.py は

```
$ ptyhon ←引数なしでインタプリタを起動
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr  4 2023, 23:49:59) [MSC v.1934 64 bit (AMD64)]
 on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> ← Python インタプリタのプロンプトが出て，入力待ち状態となる
```

としてインタプリタを起動し，Python インタプリタのプロンプトが出たところで

```
>>> print('Hellow, Python!')  ←ここを打ち込んで「Enter キー」で実行
Hellow, Python! ←コンソール画面に文字列表示
>>> ← Python インタプリタの入力待ち状態
```

と打ち込んで実行できます。

対話モードは，短い命令を試したい時に有効です。例えば，二つの数を入力して変数 (variable)a と b に格納し，その加減乗除の結果を表示させたい時には

```
>>> a = 10 ← 変数 a に 10 を代入
>>> b = 3  ← 変数 b に 3 を代入
>>> print(a + b, a - b, a * b, a / b) ← a と b の和，差，積，商を表示
13 7 30 3.3333333333333335
>>>
```

のように対話モードで実行できます。対話モードを抜ける時には「quit()」という命令を入力するか，キーボードから [CTRL]-Z を打って下さい。

これをスクリプトで記述するときには，例えば「basic_calc.py」という名前で

ソースコード 1.3 basic_calc.py

```
1 # simple_calc.py: 加減乗除結果を表示
2 a = 10
3 b = 3
4 print(a + b, a - b, a * b, a / b)
```

というスクリプトを作り，

```
$ python simple_calc.py
13 7 30 3.3333333333333335
```

として実行できます。

以後，基本はスクリプトを作って実行しますが，短い命令を確認したい時には対話モードを使うようにします。

ここで新しく学んだ Python の内容は下記の 3 つです。

1. Python はインタプリタで実行するタイプのコンピュータ言語で，対話モードで 1 行ずつ実行することもできる
2. 変数とは名前が付いたデータを格納するための箱のようなものである
3. 加算は +，減算は −，乗算は ∗(アスタリスク)，除算は /（スラッシュ）で表現する

問題 1.2

変数 c と変数 d に文字列'abc' と'efg' を入力し，c + d を実行して下さい。対話モードだと次のようになります。

```
$ python
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr  4 2023, 23:49:59) [MSC v.1934 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> c = 'abc'
>>> d = 'def'
>>> print(c + d)
abcdef
```

```
>>>
```

また，これと同じことを実行できる Python スクリプト「str_add.py」を作成し，実行結果が上記と同じものになることを確認して下さい。

1.4 Python エコシステム

冒頭に述べたように，Python の強みは，実用的に必要とされる機能がいつでも簡単に入手できるところにあります。print 関数のように，最初から Python に備わっている標準的なモジュールやパッケージのほかにも，今後使用する予定の NumPy, SciPy, Matplotlib，Pandas 等，大規模なパッケージの多くは，PyPl 等のモジュール提供サイトを通じて自分の環境にインストールして使用できます。

例えば，前に作成した simple_calc.py において，二つの変数を受け取り，四則演算の結果を表示する機能を，モジュール化してみましょう。この機能を calc2 という関数にまとめたスクリプト simple_calc2.py は

ソースコード 1.4 simple_calc2.py

```
1  # simple_calc2.py: 加減乗除結果を表示（関数化）
2
3  # calc 関数の定義
4  def calc(val1, val2):
5      print('a␣=␣', val1) # a の値を表示
6      print('b␣=␣', val2) # b の値を表示
7      print('a␣+␣b,␣a␣-␣b,␣a␣*␣b,␣a␣/␣b␣=␣', val1 + val2, val1 - val2, val1 * val2, val1
           / val2)
8
9  # メイン関数部分
10 calc(10, 3) # a = 10, b = 3
11 calc(3, 4) # a = 3, b = 4
```

のようになります。ここで重要なことは，関数の定義にあたる 4 行目から 7 行目まで，字下げ（インデント）されていることです。関数の定義は

def 関数名（引数名 1，引数名 2，...，引数名 m）: ←関数定義ブロック開始
（字下げ）命令 1
（字下げ）命令 2
.....
（字下げ）命令 n

のように **def 関数名（**から始まり，最後の**）:** から以降は関数の処理内容が記述されます。引数名 1, 引数名 2, ..., 引数名 m は関数の処理に使用でき，処理に必要なデータを引き渡すために使用されます。この最後のコロン (:) から字下げによるブロックが始まり，処理が完了するまで同じサイズの半角スペースによる字下げが行われなければなりません。このように，Python は一連の処理単位をインデントによって定義する慣習がありますので気を付けて下さい。

このように定義されたユーザ作成の関数は様々な所で実行できるようになります。この場合は，メイン関数（いの一番に実行される個所）10 行目と 11 行目で 2 回実行されていますので

simple_calc2.py の実行結果

```
a =  10
b =  3
a + b, a - b, a * b, a / b =  13 7 30 3.3333333333333335
a =  3
b =  4
a + b, a - b, a * b, a / b =  7 -1 12 0.75
```

という結果が出るはずです。

さて，このようにユーザが必要とする機能を関数として定義しておくと，他のスクリプトでも使用したくなるものです。そこで，この関数の定義を別のスクリプト＝モジュールとして保存しておき，必要な時に呼び出せるようにしておきましょう。

まず，calc 関数の定義を抜き出した「mytool.py」を，simple_calc2.py からコピペして下記のように作成して下さい。

ソースコード 1.5　mytool.py

```
1 # mytool.py: 独自定義の関数
2 # calc 関数の定義
3 def calc(val1, val2):
4     print('a␣=␣', val1) # a の値を表示
5     print('b␣=␣', val2) # b の値を表示
6     print('a␣+␣b,␣a␣-␣b,␣a␣*␣b,␣a␣/␣b␣=␣', val1 + val2, val1 - val2, val1 * val2, val1
          / val2)
```

次に，simple_calc2.py を simple_calc3.py として別名保存し，最初の部分を

ソースコード 1.6　simple_calc3.py

```
1 # simple_calc3.py: 加減乗除結果を表示（関数化）
2 from mytool import calc # mytool モジュールから calc 関数を使用可能にする
3
4 # メイン関数部分
5 calc(10, 3) # a = 10, b = 3
6 calc(3, 4) # a = 3, b = 4
```

のように書き換えて実行し，同じ結果が出ることを確認して下さい。

このように，simple_calc3.py2 行目で mytool(.py) モジュールを読み込み，そこで定義されている calc 関数を使えるようにすることで，他のスクリプトでも同様な機能の使い回しができます。

ちなみに，単純な関数名は他のモジュールと被ることが多いので，名前空間 (namespace) を別にしておくことで同じ関数名の衝突を防ぐことができます。例えば simple_calc3.py は

ソースコード 1.7　simple_calc3_np.py

```
1 # simple_calc3_np.py: 加減乗除結果を表示（関数化と名前空間使用）
2 import mytool # mytool モジュールから calc 関数を使用可能にする
3
4 # メイン関数部分
5 mytool.calc(10, 3) # a = 10, b = 3
6 mytool.calc(3, 4) # a = 3, b = 4
```

のように書くことができ，mytool という名前空間において calc 関数が定義されていることで，他のモジュールで calc 関数が定義されていても区別して使用することができるようになります。

なお，import 文に as 節を加えて

`import` **モジュール名（**`.py` **は略す）** `as` **名前空間名**

のように自由に名前空間名を命名することができます。概ね，長いモジュール名の略称が使われることが多く，例えば NumPy, SciPy, Matplotlib の pyplot は

```
import numpy as np
import scipy as sc
from matplotlib import pyplot as plt
```

のように，それぞれ np, sc, plt という略称を使うことが多いようです。

例題 1.2 (値を返す関数)

calc 関数を，計算した値を返してくれる関数に変えましょう。関数の中で `retun` 文を使います。`calc` 関数の下に，下記の `ret_calc` 関数を追加しておきましょう。

```
1 # ret_calc 関数の定義
2 def ret_calc(val1, val2):
3     print('a␣=␣', val1) # a の値を表示
4     print('b␣=␣', val2) # b の値を表示
5     print('a␣+␣b,␣a␣-␣b,␣a␣*␣b,␣a␣/␣b␣=␣', val1 + val2, val1 - val2, val1 * val2, val1
          / val2)
6     return val1 + val2, val1 - val2, val1 * val2, val1 / val2
```

最後の `return` 文に並べた値が，`ret_calc` 関数実行後に戻され，例えば呼び出したところで

```
ans1, ans2, ans3, ans4 = ret_calc(a, b)
```

と書いておくことで，`ans1`〜`ans4` に，加減乗除の結果が代入されることになります。

他のコンピュータ言語では，まとめて複数の値を返す機構がないものが多いのですが，Python の場合，このように特別なデータ型を使用することなく，複数の結果を返すことができるようになっています。

問題 1.3

1. 1 次方程式 $ax + b = 0$ の解 $x = -b/a$ を求めるスクリプト `solve_1ji.py` を作りたい。最終的には係数 a, b を受け取って解 x を表示する関数 `print_1ji` を `mytool.py` に追加して動作するようにせよ。
2. x, a, b を受け取って左辺が 0 になっていることを確認できる関数 `check_1ji` 関数を作り，`mytool.py` に追加して実行せよ。また下記の例を使って検算を行え。
   (1) $3x + 4 = 0$, $x = -4/3$
   (2) $-5x + (-6) = 0$, $x = -6/5$

演習問題

1. Python 以外で，次の条件を満足するプログラミング言語を 3 つ以上取り上げ，それぞれの特徴を述べよ。
   (a) 無料で自分の Note PC(Windows もしくは macOS) にインストールし，プログラム開発ができる

(b) インタプリタから直接ソースコードを指定して実行する（コンパイルして実行ファイルを作らない）

(c) 開発開始から 10 年以上の歴史があり，世界中で多くのユーザに利用されている

2. 角 BCA が直角である直角三角形 ABC に対し，直角をはさむ 2 辺の長さを $a = \mathrm{BC}$, $b = \mathrm{CA}$ とする。このとき，下記の手順で直角三角形 ABC の面積 S を求める Python スクリプト `triangle.py` を作れ。

(a) 三平方の定理 $c^2 = a^2 + b^2$ を用いて c^2 の値を計算して表示する。

(b) $S := (a + b)/2$ を計算して表示する。

第 2 章

Python の文法 (1/2): 変数，オブジェクト，条件分岐

今まで見てきたように，変数とはデータを格納しておくための名前付きの箱のようなもので，名前が重複しない限り，スクリプト中のどこでも使うことができます。大変便利なものですが，その中身，つまりどのようなデータが入っているか，ということについては人間の方で把握しておく必要があります。コンピュータが扱えるデータは例外なく 2 進数として表現される bit 列の塊ですが，それを文字として扱うのか，数として扱うのか，それともデータのありかを示す番地として扱うのか・・・つまり，どのように解釈すべきものかということは最低限，スクリプト実行時には確定していなくてはいけません。本章では「変数」の中身について，スクリプト例に基づいて解説していきます。

2.1 変数とデータ型

通常，コンピュータで扱うことのできる基本的なデータ型 (datatype) は，整数型 (integer) と実数型（real number, 浮動小数点型, floating-point) に 2 分されます。

整数型と同一視できるものとして，ブール型 (boolian, true(1) か false(0) のみ)，文字型 (charactor) があります。単語や文章のように文字が結合されているものを文字列型 (string) と呼びます。

実数型の拡張としては，複素数型 (complex number) があり，二つの実数をそれぞれ実部 (real part) と虚部 (imaginary part) とし，実部と虚部のセットとして表現します。幾何学的には 2 次元座標と同一視できますので，実部を横軸，虚部を縦軸としたガウス平面として表現することもあります。

Python に限らず，主としてインタプリタを使ってプログラム（スクリプト）を動かすコンピュータ言語では，初めて使う変数の中身に相応しいデータ型自動的に割り当てられます。Python の場合は `type` 関数を使うことで，その変数のデータ型を知ることができます。

例えば次の `variable.py` を動作させてみましょう。変数 `a` から変数 `f` まで値を代入し，それにふさわしい s 変数型になっている稼働を可を確認することができます。

ソースコード 2.1 `variable.py`

```
1 # variable.py: 変数とデータ型
2 a = 1 # 整数 (integer)
3 b = True # bool 値（True か False)
4 c = -3.0 # 実数 (float)
5 d = -2 + 5 * 1j # 複素数 (complex)
```

```
e = 'abcdefg' # 文字列 (ASCII 文字)
f = '吾輩は猫である' # 文字列 (UNICODE UTF-8)

print('a, type(a) = ', a, type(a))
print('b, type(b) = ', b, type(b))
print('c, type(c) = ', c, type(c))
print('d, type(d) = ', d, type(d))
print('e, type(e) = ', e, type(e))
print('f, type(e) = ', f, type(f))
```

この結果を下記に示します。それぞれ変数の値と共に,「<class ' データ型'>」という表記がなされていることが分かります。ここで class(クラス) とは，データ型と共に，そのデータに対する操作（関数や演算）を規定したものです。

variable.py の実行結果

```
a, type(a) =  1 <class 'int'>
b, type(b) =  True <class 'bool'>
c, type(c) =  -3.0 <class 'float'>
d, type(d) =  (-2+5j) <class 'complex'>
e, type(e) =  abcdefg <class 'str'>
f, type(e) =  吾輩は猫である <class 'str'>
```

それぞれ変数の値と共に,「<class ' データ型'>」という表記がなされていることが分かります。ここで class(クラス) とは，データ型と共に，そのデータに対する操作（関数や演算）を規定したものです。

例えば，整数型 (int)，実数型 (float)，複素数型 (complex) には，四則演算が定義されており，下記のスクリプトに示すように，相互に利用することができます。この場合，$a = 1, b = 2$ とし，整数型 (int_a, int_b)，実数型 (float_a, float_b)，複素数型 (complex_a, complex_b) にそれぞれ同じ値を代入して計算を行っています。

ソースコード 2.2 number_arithmetic.py

```
# number_arithmetic.py: 整数,実数,複素数の四則演算
int_a, int_b = 1, 2
float_a, float_b = 1.0, 2.0
complex_a, complex_b = 1.0 + 0 * 1j, complex(2.0, 0)

print('int_a, int_b =          ', int_a, int_b)
print('float_a, float_b =      ', float_a, float_b)
print('complex_a, complex_b = ', complex_a, complex_b)

# 同じデータ型どうしの四則演算
print('a + b = ', int_a + int_b, float_a + float_b, complex_a + complex_b)
print('a - b = ', int_a - int_b, float_a - float_b, complex_a - complex_b)
print('a * b = ', int_a * int_b, float_a * float_b, complex_a * complex_b)
print('a / b = ', int_a / int_b, float_a / float_b, complex_a / complex_b)

# データ型混合四則演算
print('a + b = ', int_a + float_b, float_a + complex_b, complex_a + int_b)
print('a - b = ', int_a - float_b, float_a - complex_b, complex_a - int_b)
print('a * b = ', int_a * float_b, float_a * complex_b, complex_a * int_b)
```

```
20 print('a␣/␣b␣=␣', int_a / float_b, float_a / complex_b, complex_a / int_b)
```

同じデータ型どうしはもちろん，異なるデータ型に対しても正確に計算されていることが分かります。もちろん，そのような組み合わせに対応できる仕組みが Python 側に備わっているから可能なのであって，想定されていないデータ型どうしの演算は実行できないこともままあります。

問題 2.1

$a = 20240403$, $b = 20250401$, $c = 20340331$ とする時，次の式の値を計算するスクリプト `eval_long_formula.py` を作り，その結果を確認して下さい。なおべき乗は`**`を使用し，例えば a^2 は `a**2` と記述します。

1. $\frac{a+b}{c+b}$ [ヒント: (a + b) / (c + b)]
2. ab^5c
3. $(a-b)^2 + (b-c)^3 + (c-a)^4$

2.2 複素演算の復習

Python における複素数 c, d は，実部と虚部はそれぞれ `real` と `imag` という変数に格納されています。従って例えば $c = 3 + 4\mathrm{i}$, $d = -2 - 5\mathrm{i}$ を代入し，その実部 ($\mathrm{Re}\ c = 3$, $\mathrm{Re}\ d = -2$) と虚部 ($\mathrm{Im}\ c = 4$, $\mathrm{Im}\ d = -5$) を取り出すには，下記のように `c.real`, `c.imag` と指定します。

```
>>> c = complex(3, 4) ← c = 3 + 4i を代入
>>> c.real ← c の実部を取り出す
3.0
>>> c.imag ← c の虚部を取り出す
4.0
>>> d = -2 - 5 * 1j ← d= -2 - 5i を代入
>>> d.real ← d の実部を取り出す
-2.0
>>> d.imag ← d の虚部を取り出す
-5.0
```

よく使用される複素演算を復習しておきましょう。虚数単位 $\mathrm{i} = \sqrt{-1}$ は $\mathrm{i}^2 = -1$ であることに注意すると，

$$
\begin{aligned}
\text{[加減算]}\ & c \pm d = (\mathrm{Re}\ c \pm \mathrm{Re}\ d) + (\mathrm{Im}\ c \pm \mathrm{Im}\ d)\mathrm{i} \\
\text{[乗算]}\ & cd = \{(\mathrm{Re}\ c)\cdot(\mathrm{Re}\ d) - (\mathrm{Im}\ c)\cdot(\mathrm{Im}\ d)\} + \{(\mathrm{Re}\ c)\cdot(\mathrm{Im}\ d) - a(\mathrm{Im}\ c)\cdot(\mathrm{Re}\ d)\}\,\mathrm{i} \\
\text{[共役]}\ & \overline{c} = \mathrm{Re}\ c + (-\mathrm{Im}\ c)\mathrm{i} \\
\text{[絶対値]}\ & |d| = \sqrt{(\mathrm{Re}\ d)^2 + (\mathrm{Im}\ d)^2} = \sqrt{d\cdot\overline{d}} \\
\text{[逆数]}\ & d^{-1} = \frac{1}{d} = \frac{d\overline{d}}{|d|^2} \\
\text{[除算]}\ & c/d = c\cdot d^{-1}
\end{aligned}
$$

となります。Python ではそれぞれ次のように計算できます。

```
>>> print(c, d) ← c, d の値の確認
(3+4j) (-2-5j)
>>> c + d ←加算
(1-1j)
>>> c - d ←減算
(5+9j)
>>> c * d ←乗算
(14-23j)
>>> c.conjugate() ← 共役
(3-4j)
>>> abs(c) ← 絶対値
5.0
>>> d ** (-1) ←逆数 (1)
(-0.06896551724137931+0.1724137931034483j)
>>> 1 / d ←逆数 (2)
(-0.06896551724137931+0.1724137931034483j)
>>> c / d ←除算
(-0.896551724137931+0.24137931034482757j)
```

問題 2.2

$c = -3-4\mathrm{i}$, $d = 2+5\mathrm{i}$ として, $c+d$, $c-d$, cd, c/d の値を求める Python スクリプト `complex_arithmetic.py` を作り，その結果を手計算と比較して確認して下さい。

2.3 文字列演算と標準入力

コンピュータはキーボード，マウス，ファイル等，データを入力する装置と，ディスプレイ，ファイル等，データを書き出す装置がついています。これらを合わせて入出力 (I/O, Input and output) 装置と呼びます。このうち，よく使用するものを標準入出力 (standard I/O) と呼び，キーボードは標準入力 (standard input)，ディスプレイを標準出力 (stardard output) もしくはコンソール (console) と呼び，Python では `print` 関数が標準出力関数となります。

では，Python の標準入力関数 `input` を使ってみましょう。キーボードから入力された値は全て文字列型になります。

ソースコード 2.3 `input_string.py`

```
1 # input_string.py: 文字列の入力
2 str_a = input('str_a␣=?␣') # str_a の入力
3 str_b = input('str_b␣=?␣') # str_b の入力
4
5 # str_a, str_b の確認
6 print('str_a,␣str_b␣=␣', str_a, str_b)
7
8 # 文字列の長さ
9 print('len(str_a),␣len(str_b)␣=␣', len(str_a), len(str_b))
```

```
10
11 # 文字列の連結
12 print('str_a␣+␣str_b␣=␣', str_a + str_b)
13
14 # 文字列の最初の一文字目と最後の文字を表示
15 print('First␣and␣last␣char␣of␣str_a␣=␣', str_a[0], str_a[len(str_a) - 1])
16 print('First␣and␣last␣char␣of␣str_b␣=␣', str_b[0], str_b[len(str_b) - 1])
```

これを実行すると，次のようになります。

input_string.py の実行結果:ASCII 文字

```
str_a =? Tomonori ←入力
str_b =? Kouya    ←入力
str_a, str_b =  Tomonori Kouya
len(str_a), len(str_b) =  8 5
str_a + str_b =  TomonoriKouya
First and last char of str_a =  T i
First and last char of str_b =  K a
```

日本語も処理できます。

input_string.py の実行結果: 日本語入力

```
str_a =? 幸谷 ←入力
str_b =? 智紀 ←入力
str_a, str_b =  幸谷 智紀
len(str_a), len(str_b) =  2 2
str_a + str_b =  幸谷智紀
First and last char of str_a =  幸 谷
First and last char of str_b =  智 紀
```

このように，標準入力関数 input は受け取ったデータを全て文字列として解釈します。そのため，整数型，実数型，複素数型として数値データを受け取った時には型キャスト関数 int, float, complex インスタンスを使う必要があります。

ソースコード 2.4　typecast.py

```
1  # typecast.py: 型キャスト例
2  str_a = input('a␣=?␣')
3  str_b = input('b␣=?␣')
4
5  # 型キャスト
6  int_a, int_b = int(str_a), int(str_b) # 整数型へ
7  float_a, float_b = float(str_a), float(str_b) # 実数型へ
8  complex_a, complex_b = complex(str_a), complex(str_b) # 複素数型へ
9
10 # データとデータ型の確認
11 print('int_a,␣type(int_a)␣=␣', int_a, type(int_a))
12 print('float_a,␣type(float_a)␣=␣', float_a, type(float_a))
13 print('complex_a,␣type(fcomplex_a)␣=␣', complex_a, type(complex_a))
```

typecast.py の実行結果

```
a =? 5 ← '5' を入力
b =? 4 ← '4' を入力 k
int_a, type(int_a) =  5 <class 'int'>
float_a, type(float_a) =  5.0 <class 'float'>
complex_a, type(fcomplex_a) =  (5+0j) <class 'complex'>
```

問題 2.3

複素数の入力を次のように行うスクリプト input_complex.py を作って下さい。

1. a を実部として入力
2. b を虚部として入力
3. $c := a + b \times \mathrm{i}$ としてまとめる
4. print 文で c を出力

2.4 書式付き出力

内部データを標準出力（ディスプレイ等）に文字列として表示する際に，形式を整えたいことがあります。例えば下記のような指定をしたい時に活躍するのが書式付き出力 (formatting output) です。

- 表示文字数
- 左寄せ，右寄せ，中央寄せ
- 小数点以下の桁数
- 指数表示

これらの指定を施した出力を行うため，print 文に対して 3 つの方法が使用できます。

1. 書式化演算子 % を利用
2. format メソッド
3. f 文字列指定

以下のスクリプトで，文字列，整数，浮動小数点数（実数）の出力事例を見ていくことにしましょう。

ソースコード 2.5 print_format.py

```
1  # print_format.py: 書式付き出力
2
3  str_a = input('a␣=␣')
4  str_b = input('b␣=␣')
5
6  # 文字列の表示
7  print('str_a,␣str_b␣=␣', str_a, str_b) # 書式指定なし
8  print('str_a,␣str_b␣=␣%10s,␣%s' % (str_a, str_b)) # 書式化演算子 %
9  print('str_a,␣str_b␣=␣{:10s},␣{}'.format(str_a, str_b)) # format メソッド
10 print(f'str_a,␣str_b␣=␣{str_a:10s},␣{str_b:s}') # f 文字列
11
12 # 整数の表示
```

```
13 int_a = int(str_a) # 文字列→整数
14 int_b = int(str_b)
15 print('int_a,␣int_b␣=␣', int_a, int_b) # 書式指定なし
16 print('int_a,␣int_b␣=␣%10d,␣%d' % (int_a, int_b)) # 書式化演算子 %
17 print('int_a,␣int_b␣=␣{:10d},␣{:d}'.format(int_a, int_b)) # format メソッド
18 print(f'int_a,␣int_b␣=␣{int_a:10d},␣{int_b:d}') # f 文字列，10進表示
19 print(f'int_a,␣int_b␣=␣{int_a:b},␣{int_b:b}') # f 文字列，2進表示
20 print(f'int_a,␣int_b␣=␣{int_a:o},␣{int_b:o}') # f 文字列，8進表示
21 print(f'int_a,␣int_b␣=␣{int_a:#o},␣{int_b:#o}') # f 文字列，8進表示，0o-prefix
22 print(f'int_a,␣int_b␣=␣{int_a:x},␣{int_b:x}') # f 文字列，16進表示
23 print(f'int_a,␣int_b␣=␣{int_a:#x},␣{int_b:#x}') # f 文字列，16進表示，0x-prefix
24
25 # 浮動小数点数（実数）の表示
26 float_a = float(str_a)
27 float_b = float(str_b)
28 print('float_a,␣float_b␣=␣', float_a, float_b) # 書式指定なし
29 print('float_a,␣float_b␣=␣%15g,␣%g' % (float_a, float_b)) # 書式化演算子 %
30 print('float_a,␣float_b␣=␣{:15g},␣{:g}'.format(float_a, float_b)) # format メソッド
31 print(f'float_a,␣float_b␣=␣{float_a:15g},␣{int_b:g}') # f 文字列
32 print(f'float_a,␣float_b␣=␣{float_a:15.2f},␣{int_b:f}') # f 文字列，小数表示
33 print(f'float_a,␣float_b␣=␣{float_a:20.15e},␣{int_b:e}') # f 文字列，指数表示
```

print_format.py の表示例

```
a = 254 ←入力値 str_a
b = 511 ←入力値 str_b
str_a, str_b =  254 511
str_a, str_b =        254, 511
str_a, str_b = 254       , 511
str_a, str_b = 254       , 511
int_a, int_b =  254 511
int_a, int_b =        254, 511
int_a, int_b =        254, 511
int_a, int_b =        254, 511
int_a, int_b = 11111110, 111111111
int_a, int_b = 376, 777
int_a, int_b = 0o376, 0o777
int_a, int_b = fe, 1ff
int_a, int_b = 0xfe, 0x1ff
float_a, float_b =  254.0 511.0
float_a, float_b =             254, 511
float_a, float_b =             254, 511
float_a, float_b =             254, 511
float_a, float_b =          254.00, 511.000000
float_a, float_b = 2.540000000000000e+02, 5.110000e+02
```

表 2.1　主なデータ型

数値	整数型	int
	論理型	bool
	浮動小数点型	float
	複素数型	complex
シーケンス	文字列型	str
その他	データ型なし	NoneType

問題 2.4

複素数 c を入力し，書式付き出力するスクリプト `output_complex.py` を作れ。[ヒント: 実部 (`c.real`) と虚部 (`c.imag`) をそれぞれ float 型の書式付き出力を行って複素数の標準形として出力する。]

2.5　変数とオブジェクト

ここまでに登場してきた変数のデータ型を含む一覧表を表**??**に示します。論理型 (Boolean type) は，True(真, 1) と False(偽, 0) の 2 値しか取らないデータ型で，後述する条件分岐で使用されます。「シーケンス」とは，文字列型のように複数の文字（データ）の連なりとして表現されるデータ型です。後述するリスト (list)，タプル (tuple)，集合 (set) もシーケンスの一種です。

Python では，変数は全てこれらのデータ型で規定されたオブジェクト (object) への参照 (reference) になっています。例えば `a = 2024` と Python で記述すると，正確には個別の ID が付加された整数型のオブジェクトが生成され（メモリ上の領域が作られ）て 2024 という整数がこのオブジェクトに格納されます。「`a`」という変数は，この整数型のオブジェクトへの参照（矢印）が格納されており，代入 (`=`) はオブジェクトへの参照を変数に設定する処理に他なりません。

この動きを具体的に見るために，次の Python スクリプトを動かしてみましょう。変数が参照しているオブジェクトの ID を見るために `id` 関数を使用しています。

ソースコード 2.6　`object.py`

```
1  # object.py: 変数とオブジェクト
2  a = 2024 # (1)
3  print('a が参照しているオブジェクトのデータ型:␣', type(a))
4  print('a が参照しているオブジェクトの ID␣␣␣␣␣:␣', id(a))
5  b = a # (2)
6  print('b が参照しているオブジェクトのデータ型:␣', type(b))
7  print('b が参照しているオブジェクトの ID␣␣␣␣␣:␣', id(b))
8  print('a␣=␣', a)
9  print('b␣=␣', b)
10
11 a = 'This␣is␣a␣pen.' # (3)
12 print('a が参照しているオブジェクトのデータ型:␣', type(a))
13 print('a が参照しているオブジェクトの ID␣␣␣␣␣:␣', id(a))
14 print('b が参照しているオブジェクトのデータ型:␣', type(b))
15 print('b が参照しているオブジェクトの ID␣␣␣␣␣:␣', id(b))
16 print('a␣=␣', a)
17 print('b␣=␣', b)
```

これを動かすと下記のような実行結果が得られます。オブジェクトの ID は動かすたびに異なりますので，実際の ID 値は各自確認して下さい。

object.py の表示例

```
a が参照しているオブジェクトの ID      :  2228462526640
b が参照しているオブジェクトのデータ型:  <class 'int'>
b が参照しているオブジェクトの ID      :  2228462526640
a =  2024
b =  2024
a が参照しているオブジェクトのデータ型:  <class 'str'>
a が参照しているオブジェクトの ID      :  2228465708272
b が参照しているオブジェクトのデータ型:  <class 'int'>
b が参照しているオブジェクトの ID      :  2228462526640
a =  This is a pen.
b =  2024
```

代入を行っている (1), (2), (3) において実際に行われていることを図示したものが図 2.1 になります。(1) では整数型のオブジェクトへの参照が a に作られ，(2) でこの参照が b と共有されます。(3) では a の参照が文字列型へのオブジェクトに変更され，結果として a と b は異なるオブジェクトへの参照を保持することになります。

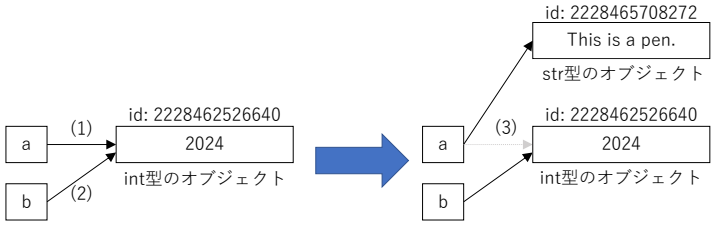


図 2.1　変数とオブジェクトの関係

Python では全てのデータはオブジェクトとして扱い，変数はデータの実態ではなく，オブジェクトへの参照であると覚えておきましょう。ちなみに，変数からの参照を失ったオブジェクトは自動的に消去されます。このようなメモリ管理をガベージコレクション (gabage collection,「ゴミ集め」の意味) と呼び，不要なメモリ消費を極力抑えることができます。

2.6　数学関数の計算

実用上頻繁に使用される数学関数として，四則演算の他に，べき乗 (x^y)，平方根 $\sqrt{x} = x^{1/2}$, 指数関数 $\exp(x) = e^x$，自然対数関数 $\log_e x = \log x$，常用対数関数 $\log_{10} x$，三角関数 $\sin x$, $\cos x$, $\tan x$，逆三角関数 $\sin^{-1} x = \arcsin x$, $\cos^{-1}(x) = \arccos x$, $\tan^{-1} x = \arctan x$ 等があります。べき乗以外は，実数関数は標準の math モジュール，複素関数は cmath モジュールが必要です。後述する NumPy では両方使えます。

2.6.1 math モジュール

ソースコード 2.7 float_calc.py

```
1  # float_calc.py: 実数型の計算
2  import math # 数学関数
3
4  a = input('a␣=␣')
5  b = input('b␣=␣')
6  a, b = float(a), float(b)
7  # 四則演算
8  print('a␣+␣b␣=␣', a + b)
9  print('a␣-␣b␣=␣', a - b)
10 print('a␣*␣b␣=␣', a * b)
11 print('a␣/␣b␣=␣', a / b)
12
13 # 平方根: math.sqrt
14 print('sqrt(a)␣=␣', math.sqrt(a))
15
16 # べき乗
17 c = math.sqrt(b)
18 print('(sqrt(b))^2␣=␣', c ** 2)
19
20 # 指数関数,三角関数,対数関数
21 print('exp(a)␣=␣', math.exp(a))
22 print('sin(a)␣=␣', math.sin(a))
23 print('cos(a)␣=␣', math.cos(a))
24 print('tan(a)␣=␣', math.tan(a))
25 print('log(a)␣=␣', math.log(a))
26 print('log10(a)␣=␣', math.log10(a))
```

2.6.2 cmath モジュール

複素数を引数とする数学関数を使うには，cmath モジュールを使います。

ソースコード 2.8 complex_func.py

```
1  # complex_func.py: 複素関数の計算
2  import cmath # 複素関数
3
4  a = 1.2 + 3.4j # 1.2 + 3.4 i
5  print('a␣=␣', a)
6
7  c = cmath.sqrt(a)
8  print('sqrt(a)␣=␣', cmath.sqrt(a))
9  print('sqrt(a)^2␣=␣', c ** 2)
10
11 # 指数関数,三角関数,対数関数
12 print('exp(a)␣=␣', cmath.exp(a))
13 print('sin(a)␣=␣', cmath.sin(a))
14 print('log(a)␣=␣', cmath.log(a))
15 print('log10(a)␣=␣', cmath.log10(a))
```

問題 2.5

次の計算を実行するスクリプト calc_formula.py を作って下さい。複素数として求める必要のあるものもありますので，エラーが出ない形で計算結果を出力するようにして下さい。

1. $\exp((-4/3)^3) = e^{(-4/3)^3}$
2. $\sqrt{-5}$
3. $\exp(\log -10)$
4. $\mathrm{i}^{\mathrm{i}} = (\sqrt{-1})^{\sqrt{-1}}$

2.7 2 次方程式の解法と条件分岐

ではここで 2 次方程式 $ax^2 + bx + c = 0$ を解くプログラム quadratic_eq.py を作ってみましょう。

ソースコード 2.9 quadratic_eq.py

```
1  # quadratic_eq.py: 2次方程式を解く
2  import math # sqrt 関数
3
4  # 係数入力
5  a = input('a␣=?␣')
6  b = input('b␣=?␣')
7  c = input('c␣=?␣')
8  a, b, c = float(a), float(b), float(c)
9  print(f'␣␣{a:25.17g}␣*␣x^2')
10 print(f'+␣{b:25.17g}␣*␣x')
11 print(f'+␣{c:25.17g}␣=␣0')
12
13 # 判別式
14 d = b ** 2 - 4 * a * c
15
16 x1 = (-b + math.sqrt(d)) / (2 * a)
17 x2 = (-b - math.sqrt(d)) / (2 * a)
18 # 出力
19 print(f'x1␣=␣{x1:25.17e}')
20 print(f'x2␣=␣{x2:25.17e}')
```

判別式 $d = b^2 - 4ac$ の値に応じてそれぞれ計算式を変えてみましょう。16 行目以降で if 分を使い，$d \geq 0$ の時は従来の計算式，$d < 0$ の時は実部と虚部の計算を行って複素数型として解を求めます。

ソースコード 2.10 quadratic_eq_mod.py

```
16 # 判別式
17 d = b ** 2 - 4 * a * c
18
19 if d >= 0: # 実数解の場合
20     print('Real␣solutions:␣\n')
21     x1 = (-b + math.sqrt(d)) / (2 * a)
22     x2 = (-b - math.sqrt(d)) / (2 * a)
23     # 出力
24     print(f'x1␣=␣{x1:25.17e}')
25     print(f'x2␣=␣{x2:25.17e}')
26 else: # 複素数解の場合
27     print('complex␣solutions:␣\n')
```

```
28      x1 = complex(-b / (2 * a), math.sqrt(-d) / (2 * a))
29      x2 = complex(-b / (2 * a), -math.sqrt(-d)) / (2 * a)
30      # 出力
31      print(f'x1␣=␣{x1:25.17e}')
32      print(f'x2␣=␣{x2:25.17e}')
```

問題 2.6

2 次方程式を解くスクリプトに次の改良を加えよ。

1. $a = 0$ の時，1 次方程式 $bx + c = 0$ を解けるようにせよ。
2. 複素数を係数とする 2 次方程式を解くプログラムを，cmath モジュールの sqrt 関数を使って作れ。この場合，判別式による場合分けは必要なくなる。

演習問題

1. 三角形 ABC の 3 辺の長さを $a = \mathrm{BC}$, $b = \mathrm{CA}$ とし，角 ACB を α とする。a, b, α をキーボードから入力し，次の手順で c, s, S を計算して出力する Python スクリプト `triangle.py` を作れ。
   (a) 入力された a, b, α から $c = \mathrm{AB}$ を余弦定理
   $$a^2 = b^2 + c^2 - 2bc\cos\alpha$$
   を用いて求める（式変形は自力で行うこと）。
   (b) $s := (a + b + c)/2$ を求める。
   (c) ヘロンの公式 $S := \sqrt{s(s-a)(s-b)(s-c)}$ で三角形の面積を求める。
   余力のある人は，角度を度 (°) で入力できるようにスクリプトに一工夫加えること。
2. 3 つの異なる複素数 c_1, c_2, c_3 を入力し，この 3 点が描く複素平面上の三角形の面積を求める Python スクリプト `complex_triangle.py` を作れ。

第 3 章

Python の文法 (2/2) リスト，ループ，辞書，集合，ファイル入出力

「退屈なことは Python にやらせよう」(Automate the boring stuff with Python) という本が出版されているぐらい，コンピュータは機械的てつまらない作業をさせるにはうってつけです。一番の強みは，何万回，何億回と繰り返すことも厭わないことで，プログラムではループ (loop) を使うことで実現できます。本章では代表的な for ループと，沢山のデータを格納するためのデータ構造（リスト，辞書，集合）について学んでいきます。

3.1 リストとループ

最初の章で使った `print_number.py`(p.3) は，for ループを使って次のように簡単に書くことができます。print 関数に `end=' '` と指定することで，改行の代わりに半角スペースが入り，数字を横並びにすることができるようになります。

ソースコード 3.1 `print_number_for.py`

```
1  # print_number_for.py: for ループを使って 0 から 20 まで表示する
2  for i in range(21):
3    print(i, end='␣') # 改行の代わりにスペースを入れる
4  print() # 最後に改行
```

この for 文は，複数のデータを格納できるリスト (list) の生成にも使用できます。

ソースコード 3.2 `print_number_for.py`

```
6  # int_array リストに 0 から 20 を入れる
7  int_array = [i for i in range(21)]
8  print(int_array) # [, ]付きで出力
9
10 # int_array の中身をすべて表示
11 for num in int_array:
12   print(num, end='␣')
13 print() # 最後に改行
```

リストはかなり自由な使い方ができます。例えば逆順にしたい時には `reversed` 関数を使います。

ソースコード 3.3 `print_number_for.py`

```
15 # 逆順で表示
```

```
16 for num in reversed(int_array):
17   print(num, end='␣')
18 print()
```

while ループを使う逆順表示も可能です。

ソースコード 3.4 print_number_for.py

```
19 # while 文で逆順に表示
20 max_num = 20
21 while max_num >= 0:
22   print(max_num, end='␣')
23   max_num -= 1
24 print()
```

問題 3.1

n を与えた時，1 から n までの自然数の和と積を計算するスクリプト sum_product.py を作って下さい。リストを使っても使わなくても実装は可能です。

3.2 エラトステネスの篩：素数判定法

素数 (prime number) とは，1 と自分自身以外の約数を持たない自然数で，1 は除きます。小さい順に並べると, 2, 3, 5, 7, 11, 13, 17, 19, 23, ... となり，感覚的には，次第に間隔が開きながらも，無数に存在していることが分かります。素数を知ることで，整数の素因数分解 (prime factorization) が可能となり，最大公約数や最小公倍数を自動的に求めることも可能となります。ここでは，一番原始的な素数判定法であるエラトステネスの篩 (Sieve of Eratosthenes) をリストを使って実装していきます。

エラトステネスの篩は，2 以上 n 以下の素数を次のようにして割り出していきます。

1. 3 以上 n 以下の自然数のうち，2 で割り切れるものを取り除く
2. 残った自然数のうち，3 で割り切れるものを除く
3. 以下同様

下記のスクリプトでは，整数の剰余を求める演算子 %(パーセント）を使い，残った素数のうち小さいものを prime_number リストに追加しています。

ソースコード 3.5 e_sieve.py

```
1  # e_sieve.py: エラトステネスの篩,素数を取り出し
2  input_str = input('Input␣integer␣=?␣')
3  max_num = int(input_str)
4
5  # 素数を格納する
6  prime_number = [2] # 2以上を素数判定する
7
8  # メインループ
9  for i in range(3, max_num + 1):
10     flag_prime = 0 # 0のままだと素数
11     for pnum in prime_number:
12         if i % pnum == 0:
13             flag_prime = 1 # 素数でない
```

```
14              break
15
16      # 既存の素数にはないもの
17      if flag_prime == 0:
18          prime_number.append(i) # i を素数に追加
19
20  # 素数をリストアップ
21  print('Prime␣number␣(<', max_num, '):', prime_number)
```

このスクリプトを基に，n を入力値とし，2 以上 n 以下の素数リストを返す prime_list 関数を実装して上記スクリプトと同じ結果が返ってくることを確認して下さい。

問題 3.2

指定された整数を素因数分解するスクリプト prime_factorization.py を作れ。また，これに基づいて与えられた整数の素因数分解リストを返す prime_fact 関数を実装してみて下さい。

なお，素因数分解のアルゴリズムは次のようになります。

1. 指定された整数までの素数リストを取得 [ヒント:prime_list 関数を使用]
2. 小さい素数から与えられた整数が割り切れるかどうかを確認し，割り切れれば素因数リスト factlist に追加。これを割り切れなくなるまで続ける

3.3 辞書と集合

要素の順序が不定でも構わないデータの塊を扱いたい時には，辞書 (dictionary) や集合 (set) が便利です。どちらも中カッコ ({ }) を用いて表記される点は共通ですが，辞書にはデータ一つ一つに索引 (index) が付加されますが，集合は索引がありません。

3.3.1 辞書の例

例えば，「英単語: 日本語」の対を集合で定義すると次のように表記されます。データの取り出しはキー (key) を使って行います。

ソースコード 3.6 word_dict.py

```
1  # word_dict.py: 辞書の例
2  mammal_dict = {'cat':'猫', 'dog':'犬', 'horse':'馬', 'cow':'牛', 'squerrel':'リス'}
3
4  # key を使って値を取り出し
5  print('cat␣=>', mammal_dict['cat'])
```

辞書は，添え字として文字列が指定できるリストと考えると良いでしょう。

word_dictpy の実行結果 (1/2)

```
cat => 猫
```

ループで回すときには items() 関数を用いることで，キーと値を同時に取り出すことができ案す。

ソースコード 3.7 word_dict.py

```
6  # key と値を同時に取り出し
```

```
7 for key, val in mammal_dict.items():
8     print(key, '->', val)
```

これを実行すると，全ての辞書データがキーと値のセットとして表示されます。

word_dict.py の実行結果 (2/2)

```
cat -> 猫
dog -> 犬
horse -> 馬
cow -> 牛
squerrel -> リス
```

問題 3.3

word_dict.py を次の動作が可能なように改良して下さい。

1.「cat」を入力すると「猫」が表示される
2.「猫」と入力すると「cat」が表示される

3.3.2 集合の例

先に述べたように，キーなしで値だけをまとめたものが集合になります。A, B が集合とすると，数学では

和集合 $A \cup B = \{c \mid c \in A \text{ または } c \in B\}$
積集合 $A \cap B = \{c \mid c \in A \text{ かつ } c \in B\}$
差集合 $A - B = \{c \mid c \in A \text{ かつ } c \notin B\}$

という定義がありますが，これと全く同じように集合を作ることができます。

ソースコード 3.8 set.py

```
1  # set.py: 集合の使い方
2  input_str = input('Input␣maxmum␣integer␣n␣=?␣')
3  n = int(input_str)
4
5  # A = { n以下の偶数の集合 }
6  # B = { n以下の 3の倍数の集合 }
7  A = set() # 空集合
8  B = set() # 空集合
9  for i in range(1, n + 1):
10     if i % 2 == 0: A.add(i) # 要素iをAに追加
11     if i % 3 == 0: B.add(i) # 要素iをBに追加
12
13 print('A␣=␣', A)
14 print('B␣=␣', B)
15
16 # 和集合
17 print('A␣|␣B␣=␣', A | B)
18 print('A.union(B)␣=␣', A.union(B))
19
20 # 積集合
```

```
21 print('A␣&␣B␣=␣', A & B)
22 print('A␣intersection␣B␣=␣', A.intersection(B))
23
24 # 差集合
25 print('A␣-␣B␣=␣', A - B)
26 print(A.difference(B))
```

set.py の実行結果 (1/2)

```
Input maxmum integer n =? 10
A =  {2, 4, 6, 8, 10}
B =  {9, 3, 6}
A | B =  {2, 3, 4, 6, 8, 9, 10}
A.union(B) =  {2, 3, 4, 6, 8, 9, 10}
A & B =  {6}
A intersection B =  {6}
A - B =  {8, 2, 10, 4}
A.differernce(B) =  {8, 2, 10, 4}
```

要素の追加は add 関数，削除は discard 関数（エラーなし）か remove 関数（指定要素が見つからないとエラーになる）で実行できます。

ソースコード 3.9 set.py

```
28 # 要素の削除
29 print('A.discard(2)␣=␣', A.discard(2))
30 print('A.remove(2)␣␣=␣', A.remove(2)) # エラーになる
```

set.py の実行結果 (2/2)

```
A.discard(2) =  None
Traceback (most recent call last):
(略)
KeyError: 2
```

問題 3.4

上記のスクリプトで定義した集合 A と B を用いて対称差集合 $(A - (A \cap B)) \cup (B - (A \cap B))$ を求める部分を追記せよ。

3.4 ファイル入出力

ファイル (file) は，永続的にデータを補完するための入れ物です。コンピュータが扱えるデータは全て 2 進数 (0 と 1 の組）として表現できるデジタルデータですが，全てのデジタルデータが文字として認識される範囲であれば文字列となり，文字列だけからなるファイルのことをテキストファイルと呼びます。テキストファイル以外は全てバイナリファイルと呼びます。例えば，今まで作ってきた Python スクリプト（*.y) は全てテキストファイルですし，Word ファイル，画像ファイル (PNG, JPEG 等）や動画ファイル (MPEG 等）は

バイナリファイルです。

Python でもファイルの入出力が可能です。ここではテキストファイルを扱う事例を見ていくことにします。

3.4.1 文字コードとテキストファイルの入出力

文字コード (character code) は，扱える文字集合（character set) と文字に割り当てる 0 以上の整数，すなわちエンコーディングスキーム (encoding scheme) がセットになったものの総称です。

コンピュータがまだ非力だったころは，欧米で使用するアルファベット・数字・記号のみを 7bit で表現できる ASCII コードで事足りていましたが，日本人向けには当然ひらがな・カタカナ・漢字が使用できないと困りますし，日本以上に膨大な感じを必要とする中国に至っては 16bit でも足りず，現在では全世界で標準的に使用される，絵文字も含めた文字集合としてユニコード (unicode) が標準となっています。このエンコードには 32bit が必要で，ASCII コードとの互換性を考慮して，これを 8bit ずつ区切って表現する形式が UTF-8 と呼ばれているものになります。以後，今まで作ってきた Python スクリプトにはコメントに日本語を使っていますが，これも UTF-8 で表現されたものになります。異なるエンコーディングスキームで読み書きを行うと，意図した文字が表示できないことになりますが，これが文字化けと呼ばれる現象の一因です。

ということで，下記のような UTF-8 で記述された日本語ファイル `student_grade_name.csv` を読み込み，1 行ごとに行番号を付加して表示する Python スクリプトを作ってみましょう。

```
学籍番号・氏名表, ,
, ,
学籍番号, 氏名, よみがな
A0001, 安藤流星, あんどうりゅうせい
A0002, 伊藤薫, いとうかおる
A0003, 井坂瑠々子, いさかるるこ
A0004, 梅坂武, うめさかたけし
```

図 3.1 `student_grade_name.csv` の冒頭部分

ファイルを扱う際には

1. ファイルを開く（open 関数）
2. ファイルのデータを読み書きする
3. ファイルを閉じる（close 関数）

というように，指定したファイルの開閉が必ず必要になります。Python の場合，open・close 関数でこの動作を行います。

ソースコード 3.10 `read_write_file.py`

```
1 # read_write_file.py: テキストファイルを読み書きする
2
3 # ファイル名の指定
4 input_filename = 'student_grade_name.csv' # 読み込み用
5 output_filename = 'linenum_student_grade_name.txt' # 書き込み用
6
7 # ファイルを開く
8 with open(input_filename, 'r', encoding='utf-8') as f_in: # 読み込み用
```

```
    with open(output_filename, 'w', encoding='utf-8') as f_out: # 書き込み用
        # 1行ずつ読み込み
        num_line = 1
        while line := f_in.readline():
            # 標準出力
            print(f'{num_line:5d}:{line}', end='␣')
            # ファイル出力
            f_out.write(f'{num_line:5d}:{line}')
            num_line += 1

        # ファイルを閉じる
        f_out.close()
f_in.close()
```

このスクリプトを動作させた結果，出来上がるのが linenum_student_grade_name.txt でここに行番号が付加されたデータが書き込まれます。

問題 3.5

read_write_file.py を改良し，読み取ったテキストファイルの文字数をカウントして表示する機能を付加して下さい。

演習問題

1. prime_factorization.py(P.23) の出力を Python のべき乗形式にせよ。例えば２ ２ ３ ３ ３ ５という出力を得た時には，は 2**2 * 3**3 * 5**1 と表示する。
2. set.py を参照し，自然数 n を入力し，次の集合を求めるスクリプトを作れ。
   (a) $A = \{a \mid a \text{ は } 5 \text{ の倍数 }\}$
   (b) $B = \{b \mid b \text{ は } 6 \text{ の倍数 }\}$
   (c) $C = \{c \mid c \text{ は } 7 \text{ の倍数 }\}$
   (d) $D = A \cup B \cup C$
   (e) $E = A \cap B \cap C$

第 4 章

基盤モジュール: NumPy

大量の実数演算や複素演算を行う科学技術計算においては，高速な計算を行うための工夫を行ったライブラリの利用が必須です。既に見てきたように，Pytnon はネイティブな機能として有限桁の浮動小数点演算をベースにこれらの処理が可能ですが，現代のコンピュータの特性を生かした高速化については別途，C/C++ 等のコンパイラ言語を用いての実装が不可欠で，その有力なモジュールが NumPy，そして NumPy をベースとした SciPy です。本章ではまず NumPy について，これらの機能や高速性を，Python スクリプトを用いた事例を通じて学んでいくことにします。

4.1 NumPy の初等関数

NumPy は math モジュール，cmath モジュールで使用できる基本的な初等関数が使用できます。NumPy には”np”というエイリアスを使用することが多いので，以下，そのように読み込むようにしています。

ソースコード 4.1 np_calc.py

```
1  # np_calc.py: NumPy の初等関数機能
2  import numpy as np # NumPy
3  import math # math モジュール
4  import cmath # cmath モジュール
5
6  a = -3.0
7  z = -1.0 + 2.0j
8
9  print('a␣=␣', a)
10 print('z␣=␣', z)
11
12 # 平方根: NumPy
13 print('np.sqrt(a)␣=␣', np.sqrt(a))
14 print('np.sqrt(z)␣=␣', np.sqrt(z))
15
16 # 平方根: math, cmath
17 # print('math.sqrt(a) = ', math.sqrt(a)) # エラーになる
18 print('cmath.sqrt(z)␣=␣', cmath.sqrt(z))
19
20 # べき乗: NumPy
21 c = np.sqrt(a)
22 w = np.sqrt(z)
23 print('(np.sqrt(a))^2␣=␣', c ** 2)
```

```
print('(np.sqrt(z))^2␣=␣', w ** 2)

# べき乗: math
# c = math.sqrt(a) #エラーになる
w = cmath.sqrt(z)
# print('(math.sqrt(a))^2 = ', c ** 2)
print('(cmath.sqrt(z))^2␣=␣', w ** 2)

# 指数関数,三角関数,対数関数: NumPy
print('np.exp(a)␣␣␣=␣', np.exp(a))
print('np.sin(a)␣␣␣=␣', np.sin(a))
print('np.log(a)␣␣␣=␣', np.log(a))
print('np.log10(a)␣=␣', np.log10(a))
print('np.exp(z)␣␣␣=␣', np.exp(z))
print('np.sin(z)␣␣␣=␣', np.sin(z))
print('np.log(z)␣␣␣=␣', np.log(z))
print('np.log10(z)␣=␣', np.log10(z))

# 指数関数,三角関数,対数関数: math
print('math.exp(a)␣␣␣=␣', math.exp(a))
print('math.sin(a)␣␣␣=␣', math.sin(a))
# print('math.log(a) = ', math.log(a)) # エラーになる
# print('math.log10(a) = ', math.log10(a)) # エラーになる
print('cmath.exp(z)␣␣␣=␣', cmath.exp(z))
print('cmath.sin(z)␣␣␣=␣', cmath.sin(z))
print('cmath.log(z)␣␣␣=␣', cmath.log(z))
print('cmath.log10(z)␣=␣', cmath.log10(z))
```

NumPy の数学関数は引数にリストを与えることで，各要素ごとの関数値を計算することができます。例えば閉区間 $[a, b]$ を n 分割し，端点を含む分割点 $x_i = (b-a)/n$ $(i = 0, 1, ..., n)$ における $\tan x_i$ と $tan^{-1}x_i$ の値を計算するスクリプトは下記のようになります。

ソースコード 4.2　np_calc_vec.py

```
# np_calc_vec.py: リストを引数とする関数計算
import numpy as np # NumPy

# [-π/4, π/4]をn 分割
n = 5 # 分割数
a, b = -np.pi / 4.0, np.pi / 4.0 # 端点
h = (b - a) / n # 区間幅

# x = [a, a + h, ..., a * (n - 1)h = b - h, a * nh = b]
xlist = np.linspace(a, b, n) # [a + h * i for i in range(n + 1)]
print('x␣=␣', xlist)

# tan_list = tan(x), atan_list = atan(x)
tan_list, atan_list = np.tan(xlist), np.arctan(xlist)
print('␣tan(x)␣=␣', tan_list)
print('atan(x)␣=␣', atan_list)
```

NumPy の linspace 関数を用いて区間を n 等分して端点を含む値をリスト化し，それを $\tan x$ と $\tan^{-1} x$ の引数として使用しています。

np_calc_vec.py の実行結果

```
x =  [-0.78539816 -0.39269908  0.          0.39269908  0.78539816]
 tan(x) =  [-1.         -0.41421356  0.          0.41421356  1.        ]
atan(x) =  [-0.66577375 -0.37419668  0.          0.37419668  0.66577375]
```

問題 4.1

1. 適当な x に対して $\log(\exp(x)) = x$ かつ $\log_{10}(10^x) = x$ であることを確認する check_log_exp.py スクリプトを作れ。
2. 適当な x に対して $\cos(\arccos(x)) = x$ かつ $\tan(\arctan(x)) = x$ であることを確認する check_cos_acos.py スクリプトを作れ。

4.2 絶対誤差と相対誤差

正しい値を真値 (true value) と呼び，真値に近いがズレがあるかもしれない値を近似値 (approximation) と呼びます。今，真値 $x \in \mathbb{R}$ に対応する近似値 $\widetilde{x} \in \mathbb{R}$ とする時，誤差 (error) は $E(\widetilde{x})$ と表現し

$$E(\widetilde{x}) = \widetilde{x} - x \tag{4.1}$$

と定義します。真値とのずれを表現したものですが，ズレの大きさだけ見たい時は誤差の絶対値を使います。この時

$$aE(\widetilde{x}) = |E(\widetilde{x})| \tag{4.2}$$

を絶対誤差 (absolute error) と呼びます。これを使用して，近似値に含まれる誤差の大きさを真値との割合で示したものが相対誤差 (relative error) で，

$$rE(\widetilde{x}) = \begin{cases} \frac{aE(\widetilde{x})}{|x|} & (x \neq 0) \\ aE(\widetilde{x}) & (x = 0) \end{cases} \tag{4.3}$$

と定義されます。

更に，真値と一致する m 進桁数を求める際にはこの相対誤差 $rE(\widetilde{x})$ を用いて

$$\lfloor -\log_m(rE(\widetilde{x})) \rfloor \tag{4.4}$$

を求めます。これを m 進有効桁数 (significant digits) と呼びます。$\lfloor \cdot \rfloor$ は小数点以下を切り捨てて整数部のみ取り出す床関数 (floor function) です。

以下は $x = \sqrt{2}$, $\widetilde{x} = 1.4142$ とした時の，$E(\widetilde{x})$, $aE(\widetilde{x})$, $rE(\widetilde{x})$ そして 10 進有効桁数を求める Python スクリプトです。

ソースコード 4.3 errors.py

```
1 # errors.py: 誤差の計算
2 import numpy as np # NumPy
3
4 # 真値
5 x = np.sqrt(2.0)
6 print('True␣x␣=␣', x)
7
```

```
# 近似値
approx_x = 1.4142
print('Approx x = ', approx_x)

# 誤差
Err = approx_x - x
print(f'E(Approx x) = {Err:10.2e}')

# 絶対誤差
aErr = np.abs(Err)
print(f'aE(Approx x) = {aErr:10.2e}')

# 相対誤差
rErr = aErr
if x != 0.0:
    rErr = aErr / np.abs(x)
print(f'rE(Approx x) = {rErr:10.2e}')

# 10進有効桁数
print(f'Sig. digits = {np.floor(-np.log10(rErr)):10.0g}')
```

誤差は大きさを確認するためのものですので，通常 2〜3 桁程度の表示で十分です。これを実行すると下記のようになります。

errors.py の実行結果

```
True x =  1.4142135623730951
Approx x =  1.4142
E(Approx x) =  -1.36e-05
aE(Approx x) =   1.36e-05
rE(Approx x) =   9.59e-06
Sig. digits =          5
```

問題 4.2

適当な区間 $[a, b]$ を n 等分し，全ての点において $\widetilde{x} = \cos(\arccos(x))$ を求め，絶対誤差が非ゼロの点における相対誤差と 10 進有効桁数を求める Python スクリプト `relerr_list.py` を作れ。

4.3 自動微分と数値微分

微分 (differentiation) は，関数 $f(x) \in \mathbb{R}$ の導関数 (derivative) を求める操作の総称です。Python で扱える微分のための手法は，いわゆる手計算と同じ記号操作で求める自動微分 (automatic differentiation) と，導関数の定義に則って極限値の近似値を使う数値微分 (numerical differentiation) の 2 種類があります。

■自動微分　例えば $f(x) = \exp(\cos x) - x^3$ の導関数 $f'(x) = -\exp(\cos x) * \sin x - 3x^2$ を求めるには，augograd パッケージが使えます。

ソースコード 4.4　`autodiff.py`

```
# autograd_diff.py: 自動微分
```

```
import autograd.numpy as np
import autograd

# 元の関数
def func(x):
    return np.exp(np.cos(x)) - x ** 3

# 真の導関数
def true_dfunc(x):
    return -np.exp(np.cos(x)) * np.sin(x) - 3 * x ** 2

# x = [-5, 5]
x = np.linspace(-5, 5, 100)

# 自動微分による導関数
dfunc = autograd.elementwise_grad(func)

# 相対誤差チェック
reldiff = np.abs((dfunc(x) - true_dfunc(x)) / true_dfunc(x))
print('reldiff␣=␣', reldiff)
```

■数値微分　Python の数式で記述できる関数の導関数は自動微分で計算できますが，例えば条件分岐を伴うような，一つの数式としては表現できない関数に対しては，そのまま適用できません。そういう時には微分の定義に立ち戻り

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{4.5}$$

という極限値の右辺の値の近似値，すなわち，適度に小さい h を与えて $f'(x) \approx (f(x+h) - f(x)))/h$ と見立てます。これを数値微分と呼びます。導関数の定義から，(4.5) 式の右辺の極限値は $h \to +0$ (数直線の右から 0 に近づく) であっても，$h \to -0$ (数直線の左から 0 に近づく) のどちらでも同じ値に収束する場合のみ考えますので，数値微分としては，小さい正の h に対して

$$f'(x) \approx \begin{cases} \frac{f(x+h)-f(x)}{h} & \text{(前進差分商)} \\ \frac{f(x+h)-f(x-h)}{2h} & \text{(中心差分商)} \\ \frac{f(x)-f(x-h)}{h} & \text{(後退差分商)} \end{cases} \tag{4.6}$$

の 3 種類の近似が考えられます。これを Python スクリプトで計算したものが `num_diff.py` です。

ソースコード 4.5　`num_diff.py`

```
# num_diff.py: 数値微分
import numpy as np # NumPy

# 元の関数
def func(x):
  return np.exp(np.cos(x)) - x ** 3

# 真の導関数
def true_dfunc(x):
  return -np.exp(np.cos(x)) * np.sin(x) - 3 * x ** 2

# 前進差分商: (f(x + h) - f(x)) / h
def forward_diff(x, h, f):
```

```
14   return (f(x + h) - f(x)) / h
15
16 # 中心差分商: (f(x + h) - f(x - h)) / 2h
17 def central_diff(x, h, f):
18   return (f(x + h) - f(x - h)) / (2.0 * h)
19
20 # 後退差分商: (f(x) - f(x - h)) / h
21 def backward_diff(x, h, f):
22   return (f(x) - f(x - h)) / h
23
24 # x = [-5, 5]
25 x = np.linspace(-5, 5, 4)
26 h = 10.0**(-5)
27
28 # 前進差分商,中心差分商,後退差分商
29 fdiff = forward_diff(x, h, func)
30 cdiff = central_diff(x, h, func)
31 bdiff = backward_diff(x, h, func)
32
33 # 相対誤差チェック
34 reldiff = np.abs((fdiff - true_dfunc(x)) / true_dfunc(x))
35 print('Forward␣diff␣relerr␣=␣', reldiff)
36 reldiff = np.abs((cdiff - true_dfunc(x)) / true_dfunc(x))
37 print('Central␣diff␣relerr␣=␣', reldiff)
38 reldiff = np.abs((bdiff - true_dfunc(x)) / true_dfunc(x))
39 print('Backword␣diff␣relerr=␣', reldiff)
```

これを実行した結果を下記に示します。明らかに中心差分商の精度が良いことが分かります。

num_diff.py の実行結果

```
Forward diff relerr =  [2.02200099e-06 7.39511504e-06 4.87809771e-06 1.97723151e-06]
Central diff relerr =  [4.04379069e-11 1.82339348e-11 1.63852760e-11 4.48259351e-11]
Backword diff relerr=  [2.02192012e-06 7.39515151e-06 4.87806494e-06 1.97732116e-06]
```

問題 4.3

$p(x) = \sin(\exp(x)) = \sin e^x$ の数値微分に基づく導関数を次の方法で求めるスクリプト diff_poly.py を作り，正しい導関数 $p'(x)$ が導出できていることを，下記の方法で確認せよ。

1. 適当な区間 $[a, b]$ を設定し，これを n 等分した時の分点を $x_i \in [a, b]$ とする。
2. すべての分点 x_i において小さい値 h を用いて計算した数値微分の値 $p'(x_i)$ の相対誤差を求め，0 に近いことを確認する。

4.4 1 変数多項式と代数方程式

n 次多項式 $p_n(x) = \sum_{i=0}^{n} a_i x^i$ を左辺として持つ非線形方程式 $p_n(x) = 0$ を代数方程式 (algebraic equation) と呼びます。既に見てきた 1 次方程式，2 次方程式は $n = 1, 2$ の代数方程式です。まずは NumPy の polynomial モジュールの使い方から見ていくことにします。

1 変数多項式については，NumPy では poly1d モジュールが伝統的に使われてきましたが，NumPy 1.4 からは polynomial パッケージを使うことが推奨されています。以下は多項式の定義（係数のみ），値の計算，導関数（多項式の微分），原始関数（多項式の積分），解の計算と検算を行っています。

例えば 3 次多項式 $p_3(x) = 2 + 2x^2 + 4x^3$ に対して，

$$
\begin{aligned}
p_3(3) &= 2 + 2 \times 3^2 + 4 \times 3^3 = 128 \\
p_3'(x) &= 4x + 12x^2 \\
p_3'(3) &= 4 \times 3 + 12 \times 3^2 = 120 \\
P_3(x) &= \int p_3(x)dx = 2x + \frac{2}{3}x^3 + x^4 \\
P_3(3) &= 2 \times 3 + \frac{2}{3} \times 3^3 + 3^4 = 105
\end{aligned}
$$

の計算を行うスクリプトは下記のようになります。

ソースコード 4.6　eval_poly.py

```
1  # eval_poly.py : 多項式の計算
2  import numpy as np # NumPy
3  import numpy.polynomial.polynomial as npppoly # Polynomial モジュール
4
5  # 係数を指定した公式の定義
6  # p(x) = 2 + 0 * x + 2 * x^2 + 4 * x^3
7  p_coef = np.array([2, 0, 2, 4]) # 単なるリストも可
8  print('p(x) = ', p_coef)
9
10 # p(3)の値の計算
11 print('p(3) = ', npppoly.polyval(3, p_coef))
12
13 # p(x)の導関数p'(x)とp'(3)の計算
14 dp_coef = npppoly.polyder(p_coef) # 係数の計算
15 print('p\'(x) = ', dp_coef)
16 print('p\'(3) = ', npppoly.polyval(3, dp_coef))
17
18 # p(x)の原始関数P(x)とP(3)の計算
19 ip_coef = npppoly.polyint(p_coef) # 係数の計算
20 print('P(x) = ', ip_coef)
21 print('P(3) = ', npppoly.polyval(3, ip_coef))
22
23 # p(x) = 0の解
24 sols = npppoly.polyroots(p_coef)
25 print('roots of p(x) = ', sols)
26
27 # 検算 p(x) == 0?
28 p_vals = npppoly.polyval(sols, p_coef)
29 print('p(roots) = ', p_vals)
```

このスクリプトを実行すると，下記のように表示されます。

eval_poly.py の実行結果

```
p(x) =  [2 0 2 4]
p(3) =  128.0
p'(x) =  [ 0.  4. 12.]
p'(3) =  120.0
P(x) =  [0.         2.         0.         0.66666667 1.        ]
P(3) =  105.0
roots of p(x) =  [-1.  +0.j          0.25-0.66143783j  0.25+0.66143783j]
p(roots) =  [-3.55271368e-15+0.00000000e+00j -4.44089210e-16+1.22124533e-15j
 -4.44089210e-16-1.22124533e-15j]
```

詳細の説明は省きますが，係数 a_i が既知の代数方程式の解 (根, root) は行列固有値として求めることができます。これを利用して，下記のように根として，1, 2, ..., 20 を持つ代数方程式を生成し，真の解との相対誤差を求める Python スクリプトを下記に示します。

ソースコード 4.7　roots_poly.py

```
1  # roots_poly.py : 代数方程式を解く
2  import numpy as np # NumPy
3  import numpy.polynomial.polynomial as npppoly # Polynomial モジュール
4
5  # 次数
6  max_deg = 20
7
8  # 真の解 : true_roots = [n, n-1, ..., 1]
9  true_roots = np.array(range(max_deg, 0, -1))
10 print('true_roots␣=␣', true_roots)
11
12 # 多項式p(x) = (x - n) * ... * (x - 1) の係数を生成
13 poly_coef = npppoly.polyfromroots(true_roots)
14 print('polynomial␣=␣', poly_coef)
15
16 # 代数方程式の解 (根)を導出
17 approx_roots = npppoly.polyroots(poly_coef)
18 approx_roots.sort() # 並び替え
19 true_roots.sort() # 並び替え
20 print('approx_roots␣=␣', approx_roots)
21
22 # 相対誤差の計算と表示
23 relerr_approx_roots = np.abs((approx_roots - true_roots) / true_roots)
24 print('relerr␣␣␣␣␣␣␣=␣', relerr_approx_roots)
```

roots_poly.py の実行結果

```
true_roots =  [20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1]
polynomial =  [ 2.43290201e+18 -8.75294804e+18  1.38037598e+19 -1.28709312e+19
  8.03781182e+18 -3.59997952e+18  1.20664780e+18 -3.11333643e+17
  6.30308121e+16 -1.01422999e+16  1.30753501e+15 -1.35585183e+14
  1.13102770e+13 -7.56111184e+11  4.01717716e+10 -1.67228082e+09
  5.33279460e+07 -1.25685000e+06  2.06150000e+04 -2.10000000e+02
  1.00000000e+00]
approx_roots =  [ 1.          2.          3.          4.00000002  4.99999947  6.0000
0709
  6.99993943  8.00035997  8.99843499 10.00522433 10.98679214 12.0273193
 12.95810032 14.05250478 14.95015682 16.03436037 16.98177036 18.00624863
 18.99865001 20.00013198]
relerr       =  [1.46549439e-14 2.34479103e-13 1.15821427e-10 5.38099831e-09
 1.05630513e-07 1.18136622e-06 8.65267594e-06 4.49959094e-05
 1.73889574e-04 5.22433082e-04 1.20071417e-03 2.27660827e-03
 3.22305251e-03 3.75034119e-03 3.32287863e-03 2.14752316e-03
 1.07233198e-03 3.47145955e-04 7.10522741e-05 6.59883799e-06]
```

あまり精度が良くないことが分かります。次数が大きくなるにつれて精度が落ちる問題は，次第に悪条件 (ill-conditioned) 化していると言えます。

問題 4.4

以下の多項式 $p_n(x)$ を左辺とする代数方程式 $p_n(x) = 0$ の解を求めて下さい。また，導出した解の検算（$p_n(x) = 0$ の確認）も行って下さい。

1. 自分の学籍番号を係数とする $p_n(x)$。例えば学籍番号が 1823054 の時は $p_6(x) = x^6 + 8x^5 + 2x^4 + 3x^3 + 5x + 4$ となる。
2. $p_n(x) = \sum_{i=0}^{n} x^i (n = 1, 2, 5, 10)$

4.5 NumPy の NDarray 機能

線形代数の基本であるベクトル (vector) や行列 (matrix) の計算は，NumPy が提供する N 次元配列，すなわち NDarray(N-Dimensional array) の機能を使って行います。以下，ベクトル，行列演算の機能をざっと見ていくことにしましょう。

4.5.1 ベクトルと行列の定義と演算

ベクトルは 1 次元，行列は 2 次元の NDarray として定義します。これによって，リストでは定義されていない，配列単位での加減算やスカラー倍が可能となります。

■ベクトルの定義と演算　$\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$ として

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix},\ \mathbf{b} = \begin{bmatrix} -3 \\ -2 \\ -1 \end{bmatrix} \tag{4.7}$$

とし，$3\mathbf{a} + \mathbf{b}$ を

$$3\mathbf{a} + \mathbf{b} = 3 \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} -3 \\ -2 \\ -1 \end{bmatrix} = \begin{bmatrix} 3 \times 1 + (-3) \\ 3 \times 2 + (-2) \\ 3 \times 3 + (-1) \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \\ 8 \end{bmatrix} \tag{4.8}$$

として計算するスクリプトを下記に示す。

ソースコード 4.8　first_vector.py

```
1  # first_vector.py: 基本線型計算
2  import numpy as np # NumPy
3
4  # ベクトル
5  vec_a = np.array([1, 2, 3])
6  vec_b = np.array([-3, -2, -1])
7
8  print('vec_a␣=␣', vec_a)
9  print('vec_b␣=␣', vec_b)
10
11 # ベクトル演算ができる
12 vec_c = 3 * vec_a + vec_b
13 print('vec_c␣=␣', vec_c)
```

上記のスクリプトを実行し，下記のように，$\mathbf{a}$, $\mathbf{b}$ が (4.7) 式で定義した通りになっているか，(4.8) 式に示した演算の結果が得られているかを確認して下さい。

first_vector.py の実行結果

```
vec_a =  [1 2 3]
vec_b =  [-3 -2 -1]
vec_c =  [0 4 8]
```

問題 4.5

$\mathbf{a}, \mathbf{b} \in \mathbb{C}^3$ として

$$\mathbf{a} = \begin{bmatrix} 1+\mathrm{i} \\ 2+2\mathrm{i} \\ 3+3\mathrm{i} \end{bmatrix},\ \mathbf{b} = \begin{bmatrix} -3-3\mathrm{i} \\ -2-2\mathrm{i} \\ -1-\mathrm{i} \end{bmatrix} \tag{4.9}$$

とし，$\sqrt{2}\mathbf{a} - \sqrt{3}\mathbf{b}$ を求めるスクリプト first_cvector.py を作れ。

■行列の定義と演算　$A, B \in \mathbb{R}^{3\times3}$ として

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix},\ B = \begin{bmatrix} -3 & -2 & -1 \\ -2 & -1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \tag{4.10}$$

とし，$3A+B$ を

$$3A+B=3\begin{bmatrix} 3\times1+(-3) & 3\times2+(-2) & 3\times3+(-1) \\ 3\times2+(-2) & 3\times3+(-1) & 3\times4+0 \\ 3\times3+(-1) & 3\times4+0 & 3\times5+1 \end{bmatrix}=\begin{bmatrix} 0 & 4 & 8 \\ 4 & 8 & 12 \\ 8 & 12 & 16 \end{bmatrix} \tag{4.11}$$

として計算するスクリプトを下記に示す。

ソースコード 4.9 first_matrix.py

```
1  # first_matrix.py: 最初の行列演算
2  import numpy as np # NumPy
3
4  # 行列
5  mat_a = np.array([
6      [1, 2, 3],
7      [2, 3, 4],
8      [3, 4, 5]
9  ])
10 mat_b = np.array([
11     [-3, -2, -1],
12     [-2, -1, 0],
13     [-1, 0, 1]
14 ])
15
16 print('mat_a␣=␣\n', mat_a)
17 print('mat_b␣=␣\n', mat_b)
18
19 # 行列演算ができる
20 mat_c = 3 * mat_a + mat_b
21 print('mat_c␣=␣\n', mat_c)
```

上記のスクリプトを実行し，下記のように行列 A, B が (4.10) 式で定義した通りになっているか，(4.11) 式に示した演算の結果が得られているかを確認して下さい。

first_matrix.py の実行結果

```
mat_a =
 [[1 2 3]
 [2 3 4]
 [3 4 5]]
mat_b =
 [[-3 -2 -1]
 [-2 -1  0]
 [-1  0  1]]
mat_c =
 [[ 0  4  8]
 [ 4  8 12]
 [ 8 12 16]]
```

問題 4.6

$A, B \in \mathbb{C}^{3\times 3}$ として

$$A = \begin{bmatrix} 1+\mathrm{i} & 2+2\mathrm{i} & 3+3\mathrm{i} \\ 2+2\mathrm{i} & 3+3\mathrm{i} & 4+4\mathrm{i} \\ 3+3\mathrm{i} & 4+4\mathrm{i} & 5+5\mathrm{i} \end{bmatrix},\ B = \begin{bmatrix} -3-3\mathrm{i} & -2-2\mathrm{i} & -1-\mathrm{i} \\ -2-2\mathrm{i} & -1-\mathrm{i} & 0 \\ -1-\mathrm{i} & 0 & 1+\mathrm{i} \end{bmatrix} \tag{4.12}$$

とし，$(\sqrt{2}+\mathrm{i})A - (\sqrt{3}\mathrm{i})B$ を求めるスクリプト `first_cmatrix.py` を作れ。

■特殊なベクトル，行列の定義　全ての要素が 0 となるベクトルをゼロベクトル (zero vector), すべての要素が 0 となる行列をゼロ行列 (zero matrix) と呼び，それぞれ $\mathbf{0} \in \mathbb{C}^n$, $O \in \mathbb{C}^{n\times n}$ と書きます。また Python では

```
np.zeros(3), np.zeros((3, 3))
```

と記述します。

同様に，全ての要素が 1 となる n 次元ベクトルや n 次正方行列は

```
np.ones(3), np.ones((3, 3))
```

と記述します。

`zeros_ones.py` の実行結果

```
np.zeros(3) =  [0. 0. 0.]
np.zeros((3, 3)) =
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
np.ones(3) =  [1. 1. 1.]
np.ones((3, 3)) =
 [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

■ドット積と内積，行列・ベクトル積，行列積　ドット積 (dot product) は，二つのベクトル $\mathbf{a}, \mathbf{b} \in \mathbb{C}^n$ の各要素 $a_i, b_i \in \mathbb{C}$ の積和のことです。これはちょうどベクトルを $n\times 1$ の行列としてみた時，$\mathbf{a}^T\mathbf{b}$ という行列積に相当するものです

$$\mathbf{a}^T\mathbf{b} = \sum_{i=1}^{n} a_i b_i \tag{4.13}$$

これに対し，内積 (inner product) は $\mathbf{a}$ の共役 $\overline{\mathbf{a}} = [\overline{a}_1\ \ldots\ \overline{a}_n]^T$ と $\mathbf{b}$ のドット積で，$(\mathbf{a}, \mathbf{b})$ と記述します。

$$(\mathbf{a}, \mathbf{b}) = \overline{\mathbf{a}}^T\mathbf{b} = \sum_{i=1}^{n} \overline{a}_i b_i \tag{4.14}$$

これらを NumPy で実行するためには，NDarray クラスに備わっている dot 関数を使用します。

ソースコード 4.10　products.py(1/2)

```
1  # products.py: ベクトルのドット積,内積,行列・ベクトル積,行列積
2  import numpy as np
3
4  # ベクトル
5  vec_a = np.array([1, 2, 3])
6  vec_b = np.array([-3, -2, -1])
7
8  print('vec_a␣=␣', vec_a)
9  print('vec_b␣=␣', vec_b)
10
11 # ドット積
12 ip_ab = vec_a.dot(vec_b)
13 print('(a,␣b)␣=␣', ip_ab)
14
15 # 複素ベクトル
16 vec_c = np.array([1 + 1j, -2 + 2j])
17 vec_d = np.array([-2 + 3j, -1 - 4j])
18 print('vec_c␣=␣', vec_c)
19 print('vec_d␣=␣', vec_d)
20
21 # ドット積と内積
22 dot_cd = vec_c.dot(vec_d)
23 ip_cd = np.conj(vec_c).dot(vec_d)
24 print('␣c␣.␣d␣=␣', dot_cd)
25 print('(c,␣d)␣=␣', ip_cd)
```

行列ベクトル積，行列積もこの dot 関数で計算できます。行列乗算については@(アットマーク）が演算子代わりに使用できます。

ソースコード 4.11　products.py(2/2)

```
36 # 行列
37 mat_a = np.array([[1, 2, 3], [2, 2, 3], [3, 3, 3]])
38 mat_b = np.array([[-3, -3, -3], [-3, -2, -2], [-3, -2, -1]])
39 print('mat_a␣=␣\n', mat_a)
40 print('mat_b␣=␣\n', mat_b)
41
42 # 行列・ベクトル積
43 mat_av = mat_a.dot(vec_a)
44 print('A␣*␣a␣=␣', mat_av)
45
46 # 行列乗算
47 mat_ab = mat_a.dot(mat_b)
48 print('A␣*␣B␣=␣\n', mat_ab)
49 mat_ab_at = mat_a @ mat_b
50 print('A␣@␣B␣=␣\n', mat_ab_at)
```

products.py の実行結果

```
vec_a =  [1 2 3]
vec_b =  [-3 -2 -1]
(a, b) =  -10
vec_c =  [ 1.+1.j -2.+2.j]
vec_d =  [-2.+3.j -1.-4.j]
 c . d =  (5+7j)
(c, d) =  (-5+15j)
mat_a =
 [[1 2 3]
 [2 2 3]
 [3 3 3]]
mat_b =
 [[-3 -3 -3]
 [-3 -2 -2]
 [-3 -2 -1]]
A * a =  [14 15 18]
A * B =
 [[-18 -13 -10]
 [-21 -16 -13]
 [-27 -21 -18]]
A @ B =
 [[-18 -13 -10]
 [-21 -16 -13]
 [-27 -21 -18]]
```

■単位行列と逆行列　単位行列 I_n は，数字の 1 に相当するもので

$$I_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix} \tag{4.15}$$

という対角行列 (diagonal matrix) です。

n 次正方行列 $A \in \mathbb{C}^{n\times n}$ が $\det(A) \neq 0$ の時，逆行列 A^{-1} を持つことが知られています。これは

$$AA^{-1} = A^{-1}A = I_n \tag{4.16}$$

という性質があります。

以下に，単位行列の定義，逆行列の導出と，(4.16) 式が成立することを確認するスクリプトを示します。

ソースコード 4.12　inv_eye.py

```
1 # inv_eys.py: 単位行列と逆行列
```

```
2  import numpy as np
3
4  # 単位行列 I
5  print('Unit matrix:\n', np.eye(3)) # 正方行列
6  print('Unit matrix:\n', np.eye(3, 3)) # 行数と列数の指定
7  print('Unit matrix:\n', np.diag([1, 1, 1])) # 対角行列
8
9  # 逆行列
10
11 # 元の行列
12 mat_a = np.array([
13     [3, 4, 0],
14     [4, 3, 4],
15     [3, 4, 3]
16 ])
17
18 print('A =\n', mat_a)
19
20 # 逆行列は存在するか?
21 # det(A) != 0 ?
22 print('det(A) = ', np.linalg.det(mat_a))
23 # rank(A) == n ?
24 print('rank(A) = ', np.linalg.matrix_rank(mat_a))
25
26 # NumPy の線型代数パッケージ(linalg)
27 # で逆行列 (inv)を導出
28 inv_mat_a = np.linalg.inv(mat_a)
29 print('A^(-1) =\n', inv_mat_a)
30
31 # 行列 * 逆行列
32 print('A * A^(-1) =\n', mat_a @ inv_mat_a)
33 print('A^(-1) * A =\n', inv_mat_a @ mat_a)
```

これを実行すると，下記のように単位行列，逆行列が導出でき，(4.16) 式が近似的に成立することも見て取れます。

inv_eye.py の実行結果

```
Unit matrix:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
Unit matrix:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
Unit matrix:
 [[1 0 0]
 [0 1 0]
 [0 0 1]]
A =
 [[3 4 0]
 [4 3 4]
 [3 4 3]]
det(A) =  -21.0
rank(A) =  3
A^(-1) =
 [[ 0.33333333  0.57142857 -0.76190476]
 [ 0.         -0.42857143  0.57142857]
 [-0.33333333  0.          0.33333333]]
A * A^(-1) =
 [[1.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 1.00000000e+00 0.00000000e+00]
 [5.55111512e-17 0.00000000e+00 1.00000000e+00]]
A^(-1) * A =
 [[ 1.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  1.00000000e+00  0.00000000e+00]
 [-5.55111512e-17  0.00000000e+00  1.00000000e+00]]
```

■ベクトル，行列のノルム　ノルム (norm) とは，ベクトル $\mathbf{v} \in \mathbb{C}^n$ や行列 $A \in \mathbb{C}^{n\times n}$ から正の実数への関数の一種で，$\|\mathbf{v}\|_p$，$\|A\|_p (\geq 0)$ と書き，絶対値が持つ次の性質を持っているものの呼称です。

正値性　全てのベクトル $\mathbf{v} \in \mathbb{C}^n$ や行列 $A \in \mathbb{C}^{n\times n}$ に対し，必ず $\|\mathbf{v}\|_p$, $\|A\|_p \geq 0$。また，$\mathbf{v} = 0$, $A = O$ の時にのみ $\|\mathbf{v}\|_p$, $\|A\|_p = 0$ となる。

スカラー倍　定数 $\alpha \in \mathbb{C}$ に対し，$\|\alpha\mathbf{v}\|_p = |\alpha| \cdot \|\mathbf{v}\|_p$, $\|\alpha A\|_p = |\alpha| \cdot \|A\|_p$。

三角方程式　$\mathbf{v}, \mathbf{w} \in \mathbb{C}^n$, $A, B \in \mathbb{C}^{n\times n}$ に対し

$$\begin{aligned} \|\mathbf{v}+\mathbf{w}\|_p &\leq \|\mathbf{v}\|_p + \|\mathbf{w}\|_p \\ \|A+B\|_p &\leq \|A\|_p + \|B\|_p \end{aligned} \tag{4.17}$$

ベクトルにしろ行列にしろ，サイズが大きくなると比較が難しくなります。そこで活躍するのがこのノルムです。よく使用するのは，$p = 1, 2, \infty$ で，ベクトルノルムでは

$$\begin{aligned} 1\text{ノルム}\quad \|\mathbf{a}\|_1 &= \sum_{i=1}^{n} |a_i| \\ \text{ユークリッドノルム}\quad \|\mathbf{a}\|_2 &= \sqrt{\sum_{i=1}^{n} |a_i|^2} \quad = \sqrt{(\mathbf{a},\mathbf{a})} \\ \text{無限大ノルム}\quad \|\mathbf{a}\|_\infty &= \max_i |a_i| \end{aligned}$$

となり，行列ノルムでは

$$\begin{aligned} 1\text{ノルム}\quad \|A\|_1 &= \max_{\mathbf{x}\neq 0} \frac{\|A\mathbf{x}\|_1}{\|\mathbf{x}\|_1} = \max_j \sum_{i=1}^{n} |a_{ij}| \\ \text{ユークリッドノルム}\quad \|A\|_2 &= \max_{\mathbf{x}\neq 0} \frac{\|A\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \sqrt{\max_i \lambda_i(A^*A)} \quad (\text{ここで } \lambda_i(A) \text{ は } A \text{ の固有値}) \\ \text{無限大ノルム}\quad \|A\|_\infty &= \max_{\mathbf{x}\neq 0} \frac{\|A\mathbf{x}\|_\infty}{\|\mathbf{x}\|_\infty} = \max_i \sum_{j=1}^{n} |a_{ij}| \end{aligned}$$

となります。また行列ノルムではフロベニウスノルムというものも使用されます。これは行列要素を縦に並べてベクトル化した時のユークリッドノルムに相当します。

$$\|A\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}|^2}$$

これらのノルムを計算する NumPy の機能は全て線形代数モジュール (linalg) にまとめられています。

ソースコード 4.13　`norm.py`

```
1  # norm.py: ベクトルと行列のノルム
2  import numpy as np
3
4  # ベクトル
5  vec_a = np.array([1, 2, 3])
6  print('vec_a␣=␣', vec_a)
7
8  # 行列
9  mat_a = np.array([[1, 2, 3], [2, 2, 3], [3, 3, 3]])
10 print('mat_a␣=␣\n', mat_a)
11
12 # ユークリッドノルム
13 print('||vec_a||_2␣=␣', np.linalg.norm(vec_a))
14 print('||mat_a||_2␣=␣', np.linalg.norm(mat_a))
15
16 # 1ノルム
17 print('||vec_a||_1␣=␣', np.linalg.norm(vec_a, 1))
```

```
18 print('||mat_a||_1␣=␣', np.linalg.norm(mat_a, 1))
19
20 # 無限大ノルム
21 print('||vec_a||_inf␣=␣', np.linalg.norm(vec_a, np.inf))
22 print('||mat_a||_inf␣=␣', np.linalg.norm(mat_a, np.inf))
23
24 # フロベニウスノルム(行列のみ)
25 print('||mat_a||_F␣=␣', np.linalg.norm(mat_a, 'fro'))
```

手計算では確認が難しい行列のユークリッドノルムも含め，全てのノルムが計算できていることが確認できます。

norm.py の実行結果

```
vec_a =  [1 2 3]
mat_a =
 [[1 2 3]
 [2 2 3]
 [3 3 3]]
||vec_a||_2 =  3.7416573867739413
||mat_a||_2 =  7.615773105863909
||vec_a||_1 =  6.0
||mat_a||_1 =  9.0
||vec_a||_inf =  3.0
||mat_a||_inf =  9.0
||mat_a||_F =  7.615773105863909
```

問題 4.7

1. $\|\mathbf{a}\|_2 = \sqrt{(\mathbf{a}, \mathbf{a})}$ であることを確認する Python スクリプトを作れ。
2. ベクトル $\mathbf{v}$, $\mathbf{w}$ と行列 A, B が与えられているとき，三角不等式が成立することを確認する Python スクリプトを作れ。

4.6 乱数列の生成，統計関数，行列乗算ベンチマーク

深層学習は，適当な非ゼロの値から出発し，学習を重ねながら正解を導くようにニューロンの重みづけを変更していきます。この時使用される「適当な非ゼロの値」は乱数 (random number) として数学的に生成されたものです。まずは NumPy から乱数を ndarray としてベクトルや行列として生成し，その平均 (average), 分散 (variant), 標準偏差 (standard deviation) を計算するスクリプトを動かしてみましょう。

ソースコード 4.14　random_test.py

```
1 # random_test.py: 乱数と統計関数
2 import numpy as np # NumPy
3
4 # 乱数seed の指定
5 rng = np.random.default_rng(seed=20240405)
6
```

```
# 乱数列生成
rand_vec = rng.random(3)
print('rand_vec, type(rand_vec) = ', rand_vec, type(rand_vec))
# 平均,分散,標準偏差
print('ave, var, std = %7.3g, %7.3g, %7.3g' % (np.average(rand_vec), np.var(rand_vec),
    np.std(rand_vec)))

# 乱数行列生成
rand_mat = rng.random((3, 3))
print('rand_mat = \n', rand_mat)
print('ave, var, std = %7.3g, %7.3g, %7.3g' % (np.average(rand_mat), np.var(rand_mat),
    np.std(rand_mat)))
```

これを実行すると下記のようになります。乱数ですので，環境によって数字は異なります。

norm.py の実行結果

```
rand_vec, type(rand_vec) =  [0.04816281 0.77801654 0.9651537 ] <class 'numpy.ndarray'>
ave, var, std =   0.597,   0.157,   0.396
rand_mat =
 [[0.57626548 0.94308848 0.17285614]
 [0.3705871  0.11832061 0.01597963]
 [0.13692318 0.25497593 0.93845524]]
ave, var, std =   0.392,   0.109,   0.331
```

本章の最後に，行列積の計算時間を計測するスクリプトを作ってみましょう。時間計測は time パッケージの time 関数を使っています。

ソースコード 4.15　matmul_bench.py

```
# matmul_bench.py: 行列乗算ベンチマーク
import numpy as np # NumPy
from time import time # 時間計測

# 乱数seedの指定
rng = np.random.default_rng(seed=20240405)

# 行列サイズ入力
str_dim = input('Input dimension = ')
dim = int(str_dim)

# 乱数行列生成
mat_a = rng.random((dim, dim))
mat_b = rng.random((dim, dim))

# 行列乗算 1
stime = time()
mat_c_dot = mat_a.dot(mat_b)
dot_time = time() - stime

# 行列乗算 2
stime = time()
mat_c_at = mat_a @ mat_b
```

```
at_time = time() - stime

# 出力
print('||C_dot||_F␣=␣%25.17e' % np.linalg.norm(mat_c_dot, 'fro'))
print('||C_at␣||_F␣=␣%25.17e' % np.linalg.norm(mat_c_at , 'fro'))
print('s(C_dot),␣s(C_at)␣=␣%7.3f,␣%7.3f' % (dot_time, at_time))
```

問題 4.8

matmul_bench.py を自分の PC 上で実行し，$\dim = 2000, 5000, 10000$ の時の計算時間を計測し，その変化について考察せよ。その際，計算時間は 10 回反復した平均値を使用すること。また標準偏差も導出すること。

第 5 章

基盤モジュール: SciPy

NumPy が提供する NDarray や数学関数をはじめとする基本機能をベースに，より複雑な統計計算，科学技術計算を行うためのパッケージが SciPy です。ここでは，高度な線形代数問題，積分，常微分方程式の初期値問題を例に，その機能の一部を紹介していきます。

5.1 連立一次方程式の求解

n 次正方行列 A と n 次元ベクトル $\mathbf{b}$ が与えられた時

$$A\mathbf{x} = \mathbf{b} \tag{5.1}$$

を満足する未知の n 次元ベクトル $\mathbf{x}$ を求める問題が連立一次方程式 (linear system of equations) の求解問題です。もし行列 A が逆行列 A^{-1} を持てば，左右両辺にこれを左から乗じて

$$\mathbf{x} = A^{-1}\mathbf{b}$$

が解となりますが，このやり方では計算量が多く，次元数 n が大きくなると時間を要します。計算量の少ない方法がありますので，Python では SciPy の線形代数パッケージ (linalg) の `solve` 関数を用いて解いてみましょう。この中では解いた結果の検証をノルム相対誤差

$$rE_p(\widetilde{\mathbf{x}}) = \begin{cases} \frac{\|\widetilde{\mathbf{x}}-\mathbf{x}\|_p}{\|\mathbf{x}\|_p} & (\mathbf{x} \neq 0) \\ \|\widetilde{\mathbf{x}}-\mathbf{x}\|_p & (\mathbf{x} = 0) \end{cases} \tag{5.2}$$

で行っています。p は使用したノルム $(1, 2, \infty)$ が入ります。

ソースコード 5.1 `linsolve.py`

```
1  # linsolve.py: 連立一次方程式を高速に解く
2  import numpy as np # NumPy
3  import scipy.linalg as sclinalg # Scipy.linalg
4  import time # time 関数
5
6  # 乱数seed の指定
7  rng = np.random.default_rng(seed=20240521)
8
9  # 正方行列の次数
10 input_str = input('Input␣size␣of␣square␣matrix␣>␣')
11 n = int(input_str)
```

```
12
13 # 乱数行列生成
14 mat_a = rng.random((n, n)) # n x n 行列
15 print('||mat_a||_2␣=␣', np.linalg.norm(mat_a))
16 # 解を生成
17 true_x = rng.random((n, 1)) # n 次元ベクトル
18 print('||x||_2␣=␣', np.linalg.norm(true_x))
19
20 # 定数ベクトルb を生成
21 b = mat_a @ true_x
22
23 # (1)逆行列を求めて連立一次方程式を解く
24 start_time = time.time()
25 inv_x = sclinalg.inv(mat_a) @ b
26 end_time_inv = time.time() - start_time
27 relerr_inv = np.linalg.norm(inv_x - true_x) / np.linalg.norm(true_x)
28
29 # (2) solve 関数を用いて連立一次方程式を解く
30 start_time = time.time()
31 solve_x = sclinalg.solve(mat_a, b)
32 end_time_solve = time.time() - start_time
33 relerr_solve = np.linalg.norm(solve_x - true_x) / np.linalg.norm(true_x)
34
35 # ノルム相対誤差と計算時間の表示
36 print( '␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣inv␣␣␣␣␣␣␣␣␣␣␣solve')
37 print(f'Computational␣time␣(s):␣{end_time_inv:10.2g},␣{end_time_solve:10.2g}')
38 print(f'Norm2␣Relative␣error␣␣:␣{relerr_inv:10.2e},␣{relerr_solve:10.2e}')
```

これを実行してみると，次元数が大きくなると inv 関数を用いるより，solve 関数を用いた方が高速に解けることが分かります。

linsolve.py の実行結果

```
Input size of square matrix > 10000
||mat_a||_2 =  5773.377361270599
||x||_2 =  57.722332431809136
                                inv       solve
Computational time (s):          32,          13
Norm2 Relative error  :   2.44e-11,   2.93e-12
```

問題 5.1

linsolve.py を改良し, A, $\mathbf{b}$ をそれぞれ n 次複素正方行列, n 次元複素ベクトルとして与える clinsolve.py を作成し,

1. inv 関数を使用して $\mathbf{x} := A^{-1}\mathbf{b}$ を計算
2. solve 関数を使用して $\mathbf{x}$ を導出

の計算時間とノルム相対誤差をそれぞれ求められるようにして下さい。

5.2 正方行列の固有値と固有ベクトル

n 次正方行列 $A \in \mathbb{C}^{n\times n}$ に対して，定数 $\lambda \in \mathbb{C}$ と非ゼロ n 次ベクトル $\mathbf{v} \in \mathbb{C}^n$ が

$$A\mathbf{v} = \lambda\mathbf{v} \tag{5.3}$$

という関係を満足するとき，λ を A の右固有値 (right eigenvalue)，$\mathbf{v}$ を λ に対応する A の右固有ベクト (right eigenvector) と呼びます。また，定数 $\varpi \in \mathbb{C}$ と非ゼロ n 次ベクトル $\mathbf{w} \in \mathbb{C}^n$ に対して

$$\overline{A}^T\mathbf{w} = \varpi\mathbf{w} \tag{5.4}$$

という関係を満足するとき，ω を A の左固有値 (left eigenvalue)，$\mathbf{w}$ を ω に対応する A の左固有ベクトルと呼びます。定義式 (5.4) は

$$\overline{\mathbf{w}^T}A = \omega\overline{\mathbf{w}^T} \tag{5.5}$$

とも定義されます。

単に固有値，固有ベクトルという時には一般には右固有値，右固有ベクトルを意味します。

NumPy にも固有値・固有ベクトルを求める `np.linalg.eig` 関数はありますが，SciPy の `scipy.linalg.eig` 関数の方がより多くのバリエーションに対応していますので，特に支障ない場合はこちらの利用をお勧めします。

ソースコード 5.2 `eig.py`

```
1  # eig.py: 正方行列の固有値・固有ベクトル
2  import numpy as np # NumPy
3  import scipy.linalg as sclinalg # SciPy.linalg
4
5  # 乱数seedの指定
6  rng = np.random.default_rng(seed=20240521)
7
8  # 正方行列の次数
9  input_str = input('Input␣size␣of␣square␣matrix␣>␣')
10 n = int(input_str)
11
12 # 乱数行列生成
13 mat_a = rng.random((n, n))
14 print('mat_a␣=␣\n', mat_a)
15
16 # 全ての固有値
17 eigen_values = sclinalg.eig(mat_a)
18
19 # 固有値，(右)固有ベクトル
20 eigen_values, Vr = sclinalg.eig(mat_a, right=True)
21 print('Eigenvalues␣=␣', eigen_values)
22 print('Right␣eigenvectors␣=␣', Vr)
23
24 # 固有値,左固有ベクトル，(右)固有ベクトル
25 eigen_values, Vl, Vr = sclinalg.eig(mat_a, left=True, right=True)
26 print('Eigenvalues␣=␣', eigen_values)
27 print('Left␣eigenvectors␣=␣\n', Vl)
28 print('Right␣eigenvectors␣=␣\n', Vr)
```

```
29
30 print('index␣||(A␣-␣eig␣*␣I)␣Vr␣||␣/␣||Vr||␣␣||(A␣-␣eig␣*␣I)␣Vl||␣/␣||Vl||')
31 for index in range(n):
32   # A * Vr == Labmda * Vr ?
33   print(f'{index:5d}␣{np.linalg.norm((mat_a␣-␣eigen_values[index]␣*␣np.eye(n))␣@␣Vr[:,
         ␣index])␣/␣np.linalg.norm(Vr[:,␣index]):30.17e}␣{np.linalg.norm((mat_a.T␣-␣
         eigen_values[index].conj()␣*␣np.eye(n))␣@␣Vl[:,␣index])␣/␣np.linalg.norm(Vl[:,␣
         index]):30.17e}')
```

乱数行列の左右固有値，固有ベクトルを求め，検算として定義式 (5.3), (5.4) を満足しているか，ノルム相対誤差を求めて確認しています。例えば $n = 2$ の場合はという結果を得ることができます。

eig.py の実行結果

```
Input size of square matrix > 2
mat_a =
 [[0.7763316  0.54878817]
 [0.97934203 0.65359049]]
Eigenvalues =  [ 1.45063602+0.j -0.02071393+0.j]
Right eigenvectors =  [[ 0.63122694 -0.56710364]
 [ 0.77559819  0.82364644]]
Eigenvalues =  [ 1.45063602+0.j -0.02071393+0.j]
Left eigenvectors =
 [[ 0.82364644 -0.77559819]
 [ 0.56710364  0.63122694]]
Right eigenvectors =
 [[ 0.63122694 -0.56710364]
 [ 0.77559819  0.82364644]]
index ||(A - eig * I) Vr || / ||Vr||  ||(A - eig * I) Vl|| / ||Vl||
 0        2.2204604604925031308e-16        2.4825341532472731 2e-16
 1        1.1102230246251565 4e-16        2.2204604604925031308e-16
```

問題 5.2

eig.py を改良し，A を n 次複素正方行列として与える ceig.py を作成し，左右固有値，固有ベクトルを導出し，検算として定義式 (5.3), (5.4) を満足しているか，ノルム相対誤差を求めて確認できるようにして下さい。

5.3 定積分

例えば

$$\int_2^3 x \exp(\sin(x^2))dx \tag{5.6}$$

という定積分を計算するためには，SciPy の integrate モジュールを使って次のように計算します。

ソースコード 5.3 integration.py

```
# integration.py: 数値積分による定積分 +問題 12.4
import numpy as np
import scipy.integrate as scint # 積分パッケージ

# 被積分関数
def func1(x):
    return x * np.exp(np.sin(x ** 2))
# 定積分
a, b = 2, 3
ret = scint.quad(func1, a, b)

print('integral[', a, ',␣', b, ']␣:␣', ret[0])
print('ret␣=␣', ret)
print('aE(ans)␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣=␣', ret[1])
print('rE(ans)␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣=␣', np.abs(ret[1] / ret[0]))
```

integration.py の実行結果

```
integral[ 2 ,  3 ] :  3.4199262740710528
ret =  (3.4199262740710528, 7.773062132546322e-12)
aE(ans)               =  7.773062132546322e-12
rE(ans)               =  2.2728741819610433e-12
```

問題 5.3

上記のスクリプトを使用して下記の定積分の値を求めよ。

1. $\int_0^{\pi} \exp(\sin x)\ dx$
2. $\int_0^3 \cos x^2 dx$

5.4 常微分方程式

ここで使用する常微分方程式は

$$\frac{d^r \mathbf{y}}{dx^r} = \tilde{\phi}\left(x, \mathbf{y}, \frac{d\mathbf{y}}{dx}, \frac{d^2\mathbf{y}}{dx^2}, ..., \frac{d^{r-1}\mathbf{y}}{dx^{r-1}}\right) \tag{5.7}$$

という形式で表現できるもののみ扱うことにします。解はベクトル関数 $\mathbf{y}(x) = [y_1(x)\ ...\ y_n(x)]^T$ となります。

特に基本となるのは一階の一次元常微分方程式

$$\frac{dy}{dx} = f(x, y) \tag{5.8}$$

で，2 階以上の常微分方程式については

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_r \end{bmatrix} = \begin{bmatrix} y \\ dy/dx \\ \vdots \\ d^{r-1}y/dx^{r-1} \end{bmatrix}$$

と置き換えることによって，r 階常微分方程式 (5.7) を 1 階の常微分方程式

$$\frac{d}{dx}\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{r-1} \\ y_r \end{bmatrix} = \begin{bmatrix} y_2 \\ y_3 \\ \vdots \\ y_r \\ \widetilde{\phi}(x, y_1, y_2, ..., y_{r-1}) \end{bmatrix} \tag{5.9}$$

と同一視できます。したがって，以降は n 次元 1 階常微分方程式

$$\frac{d\mathbf{y}}{dx} = \mathbf{f}(x, \mathbf{y}) \tag{5.10}$$

のみ考えることにします。

常微分方程式の解は無数に存在することは，最も簡単な次の例題で分かります。

例題 5.1

1 次元の常微分方程式

$$\frac{dy}{dx} = y$$

の解は右辺を左辺に移項して両辺を x について積分することによって得られる。この時，積分定数 $c \in \mathbb{R}$ が残り，結局，解 $y(x)$ は

$$y(x) = c\exp(x)$$

となる。すなわち，この解は無数に存在する。

解を一意に定めるためには，常微分方程式に対して条件を設定する必要があります。そのうち，変数 x が $x = x_0$ と固定された地点での $\mathbf{y}(x_0) = \mathbf{y}_0$ を与えられた問題を，「初期値問題」(initial value problem, IVP) と呼び，この $\mathbf{y}_0$ を初期値もしくは初期条件 (initial condition) と呼びます。したがって，常微分方程式の初期値問題は

$$\left\{ \begin{array}{rcl} \frac{d\mathbf{y}}{dx} & = & \mathbf{f}(x, \mathbf{y}) \\ \mathbf{y}(x_0) & = & \mathbf{y}_0 \end{array} \right. \tag{5.11}$$

という形で与えられます。これを Python で扱うには，SciPy の integrate パッケージを使用します。

■Python スクリプト例　常微分方程式のソルバーは SciPy.integrate パッケージにあります。初期値問題の場合は `solve_ivp` 関数をソルバーとして使うのが標準で，独立変数 x は `t` としてソルバーでは使用されます。

以下のスクリプトでは，ソルバーに積分区間内の評価点を決めさせる方法（デフォルト）と，ユーザが必要となる評価点を指定する方法で，それぞれ数値解を求め，そのグラフを描いています。後者のようにしておくと，積分区間におけるグラフを滑らかに描くことができるようになります。

ソースコード 5.4　`ode_ivp.py`

```
1 # ode_ivp.py: 常微分方程式の初期値問題
2 import numpy as np
3 import scipy.integrate as scint # ODE ソルバー
4 import matplotlib.pyplot as plt # グラフ描画
5
6
7 # 陽的形式の右辺
8 # y' = func(t, y) = y
```

```
def func(t, y):
  return y


# 初期値
# y(0) = 1
y0 = [1.0]

# t = [0, 1]
t_interval = [0.0, 1.0]
print(t_interval)

# 常微分方程式を解く
ret = scint.solve_ivp(func, t_interval, y0) # 評価点t が可変になる
ret_fix = scint.solve_ivp(
  func, t_interval, y0,
  t_eval=np.linspace(t_interval[0], t_interval[1], 10)
) # 評価点t が固定化される

# 結果を表示
print(ret)

# y を表示
print(ret.y)
```

ode_ivp.py の実行結果

```
  [0.0, 1.0]
  message: The solver successfully reached the end of the integration interval.
  success: True
   status: 0
     t: [ 0.000e+00  1.000e-01  1.000e+00]
     y: [[ 1.000e+00  1.105e+00  2.718e+00]]
   sol: None
    t_events: None
    y_events: None
     nfev: 14
     njev: 0
   nlu: 0
   [[1.         1.10519301 2.718327  ]]
```

問題 5.4

上記のスクリプトを使用して下記の常微分方程式の初期値問題の解を指定区間において求めよ。

1. $y' = -y, y(0) = 1, t \in [0, 1]$
2. $y' = -xy, y(0) = 1, t \in [0, 1]$

第 6 章

基盤モジュール: Matplotlib

「百聞は一見に如かず」という諺がある通り，物事を伝える際には文章や言葉の説明より，写真やイラストを見せる方が，理解してもらいやすいものです。特に，現代のように，コンピュータが大量のデータを瞬時に処理でき時代においては，数字の羅列を目で追うことは不可能で，リアルタイムにグラフ化し，データの変化を目で見ることで理解することが重要です。Python では多種多様なグラフを生成することのできる Matplotlib パッケージが用意されていますので，データサイエンスでは欠かせない可視化ツールを Python で簡単に作ることが出来ます。この章では，膨大な Matplotlib の機能のほんの一部を使ってみることにします。

6.1 最初の関数グラフ

まず，$x \in [a, b]$ における関数 $y = \sin(x)$ のグラフを描いてみましょう。描き方としては

1. $[a, b]$ を n 分割した時の分割幅 $x_h := (b - a)/n$ を計算する
2. $x_i := a + ih (i = 0, 1, ..., n)$ を計算し，リスト $x = [x_0, x_1, ..., x_n]$ を作る
3. $y := \sin(x) = [\sin(x_0), \sin(x_1), ..., \sin(x_n)]$ を作る
4. ax.plot(x, y) を実行し，横軸が x，縦軸が y の値となる 2 次元折れ線グラフを描く

となります。

ソースコード 6.1 plot_sine.py

```
1  # plot_sine.py: サインカーブを描く
2  import matplotlib.pyplot as plt # Matplotlib
3  import numpy as np # NumPy
4
5  # x の範囲[a, b]を指定
6  a = 0
7  b = 4 * np.pi
8
9  # x 区間の分割数を指定
10 n = 10
11
12 # x の配列を生成
13 x_h = (b - a) / n # 小区間幅
14 x = [ a + i * x_h for i in range(n + 1)]
15
16 # x ごとに y の値を計算
```

```
17 y = np.sin(x)
18
19 print(x)
20 print(y)
21
22 # 2次元グラフを描画
23 # (1) Figure オブジェクト(fig)と
24 # その上にAxes オブジェクト(ax)を生成
25 fig, ax = plt.subplots()
26
27 # (2) 折れ線グラフをax 上に描画
28 ax.plot(x, y)
29
30 # (3) グラフタイトル,x 軸タイトル，y 軸タイトルを指定
31 ax.set_title('y␣=␣sin(x)')
32 ax.set_xlabel('x')
33 ax.set_ylabel('y')
34
35 # (4) グリッドを描画
36 ax.grid()
37
38 # (5) グラフを表示
39 plt.show()
```

これを実行すると，図 6.1 のように，Windows 上にグラフが描画されます。プログラムを終了したい時にはこのグラフウィンドウを閉じて下さい。

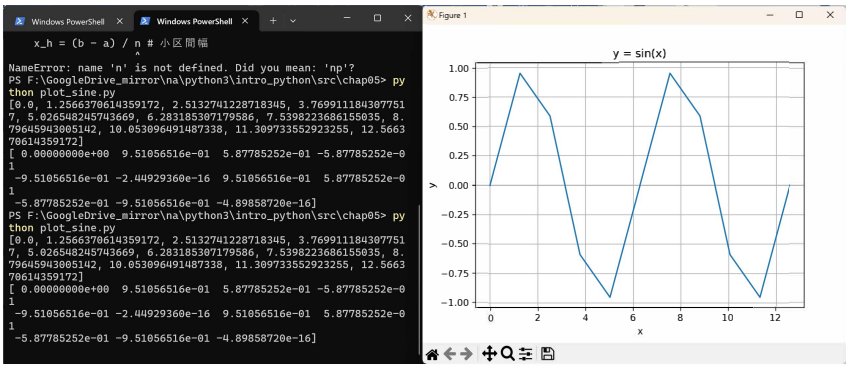


図 6.1　plot_sine.py 実行画面

問題 6.1

plot_sine.py に基づいて次の変更を行った plot_cosine.py を作成せよ。

1. なめらかな曲線に見えるよう，分割数 n を大きくする
2. $y = \cos(x)$ のグラフを描画する
3. グラフタイトルを関数式に変更する

6.2 複数グラフを並べる

Matplotlib は相当複雑なグラフを自在に描くことができます。例えば，複数のグラフをまとめて表示することも朝飯前です。

下記の例は，左側に三角関数，右側に指数関数と対数関数のグラフを描いたものになります。

ソースコード 6.2 plots_functions.py

```
1  # plot_functions.py: 複数のグラフを並べる
2  import matplotlib.pyplot as plt # Matplotlib
3  import numpy as np # NumPy
4
5  # x 区間の分割数を指定
6  n = 100
7
8  # Figure オブジェクト(fig)と
9  # その上にAxes オブジェクト(ax, ax2)を
10 # 1行 2列に生成し,サイズを 10x5 とする
11 fig, (ax, ax2) = plt.subplots(1, 2, figsize=(10, 5))
12
13 # 三角関数のグラフを描く
14 # x の範囲[a, b]を指定
15 a = 0
16 b = 4 * np.pi
17 x_h = (b - a) / n # 小区間幅
18 x = [ a + i * x_h for i in range(n + 1)]
19
20 # x ごとに y の値を計算
21 y_sin = np.sin(x)
22 y_cos = np.cos(x)
23
24 # 三角関数グラフを描画
25 ax.plot(x, y_sin, label='sin(x)')
26 ax.plot(x, y_cos, label='cos(x)')
27 ax.set_title('y␣=␣sin(x),␣cos(x)')
28 ax.set_xlabel('x')
29 ax.set_ylabel('y')
30 ax.set_ybound(lower=-3, upper=3)
31 ax.legend()
32 ax.grid()
33
34 # 指数関数を対数関数をfig 上に描画
35 # x の配列を生成し，exp(x)とlog(x)を計算
36 a = -3
37 b = 3
38 x_h = (b - a) / n # 小区間幅
39 x = [ a + i * x_h for i in range(n + 1)]
40 y_exp = np.exp(x)
41 y_log = np.log(x) # x < の時はNaN を含む
42
43 # 折れ線グラフをax2 上に描画
44 ax2.plot(x, y_exp, label='exp(x)')
45 ax2.plot(x, y_log, label='log(x)')
```

```
ax2.set_title('y␣=␣exp(x),␣log(x)')
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.set_ybound(lower=-3, upper=4)
ax2.legend()
ax2.grid()

# グラフを表示
plt.show()
```

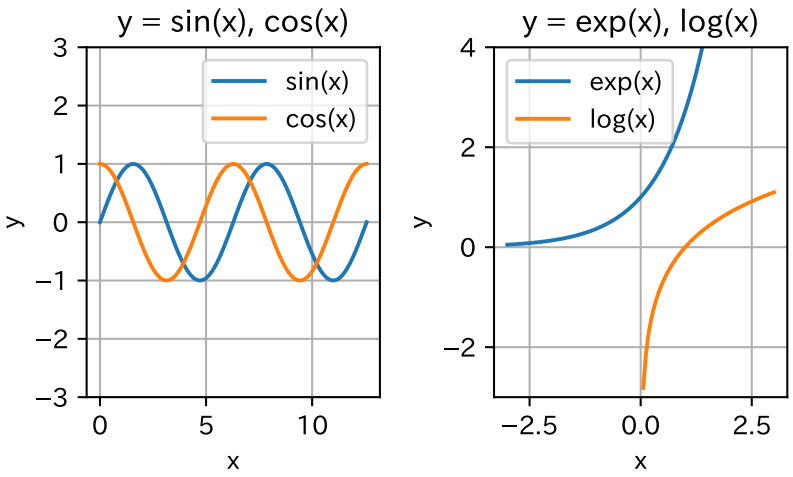


図 6.2　複数のグラフを描画する

問題 6.2

plot_functions.py の右のグラフに $y = \tan(x)$ のグラフを描画し，結果について考察せよ。

6.3　2 つの y 軸を持つグラフを描画する

一枚のグラフに，2 つの異なるスケールを持つ値をプロットしたい時があります。例えば，$f(x)$ の関数値と共に，数値微分の相対誤差を表示したい時，同じ x の値に対して，左縦軸は $f(x)$ のスケールで，右縦軸は log スケールにしておき，適切なスケールで値を描画すると，関数の値と誤差の変動がどのように連動しているかが一目で分かります。

このグラフを描画するスクリプトを下記に示します。

ソースコード 6.3　plot_num_diff.py

```
# plot_num_diff.py: 数値微分と相対誤差の描画
import numpy as np # NumPy
import matplotlib.pyplot as plt # Matplotlib

# 元の関数
def func(x):
    return np.exp(np.cos(x)) - x ** 3
```

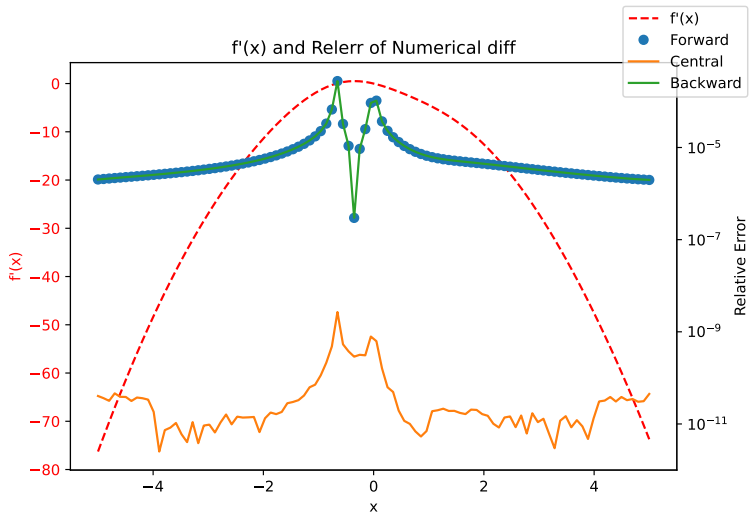


図 6.3　関数グラフ (左軸) と数値微分の相対誤差 (右軸)

```
8
9  # 真の導関数
10 def true_dfunc(x):
11     return -np.exp(np.cos(x)) * np.sin(x) - 3 * x ** 2
12
13 # 前進差分商: (f(x + h) - f(x)) / h
14 def forward_diff(x, h, f):
15     return (f(x + h) - f(x)) / h
16
17 # 中心差分商: (f(x + h) - f(x - h)) / 2h
18 def central_diff(x, h, f):
19     return (f(x + h) - f(x - h)) / (2.0 * h)
20
21 # 後退差分商: (f(x) - f(x - h)) / h
22 def backward_diff(x, h, f):
23     return (f(x) - f(x - h)) / h
24
25 # x = [-5, 5]
26 x = np.linspace(-5, 5, 100)
27 h = 10.0**(-5)
28
29 # 前進差分商,中心差分商,後退差分商
30 fdiff = forward_diff(x, h, func)
31 cdiff = central_diff(x, h, func)
32 bdiff = backward_diff(x, h, func)
33
34 # 相対誤差チェック
35 reldiff_f = np.abs((fdiff - true_dfunc(x)) / true_dfunc(x))
36 print('Forward diff relerr =', reldiff_f)
37 reldiff_c = np.abs((cdiff - true_dfunc(x)) / true_dfunc(x))
38 print('Central diff relerr =', reldiff_c)
39 reldiff_b = np.abs((bdiff - true_dfunc(x)) / true_dfunc(x))
```

```
print('Backword␣diff␣relerr=␣', reldiff_b)

# 導関数のグラフ描画
true_dfunc_y = true_dfunc(x)
print('x␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣=', x)
print('True␣dfunc␣␣␣␣␣␣␣␣␣=', true_dfunc_y)

fig, ax1 = plt.subplots()

# 関数グラフのy 軸（左）を設定
ax1_color = 'red' # 赤
ax1.set_title('f\'(x) and Relerr of Numerical diff')
ax1.set_xlabel('x')
ax1.set_ylabel('f\'(x)', color=ax1_color)
ax1.plot(x, true_dfunc_y, '--', label='f\'(x)',␣color=ax1_color)␣#␣赤の点線
ax1.tick_params(axis='y',␣labelcolor=ax1_color)␣#␣y 軸に目盛を入れる

#␣2番目のy 軸（右）を設定
ax2␣=␣ax1.twinx()␣#␣x 軸は共通
ax2.set_ylabel('Relative Error')
ax2.set_yscale('log')␣#␣log スケールに変更
ax2.plot(x,␣reldiff_f,␣'o',␣label='Forward')␣#␣○プロット
ax2.plot(x,␣reldiff_c,␣label='Central')
ax2.plot(x,␣reldiff_b,␣label='Backward')
ax2.tick_params(axis='y')

#␣凡例
fig.legend()

#␣グラフ描画
plt.show()
```

問題 6.3

問題 4.3 に導関数と数値微分の相対誤差のグラフを描画する機能を付加した `plot_diff_poly.py` を作れ。

6.4 常微分方程式の解の描画

常微分方程式のソルバーは SciPy.integrate パッケージにあります。初期値問題の場合は `solve_ivp` 関数をソルバーとして使うのが標準で，独立変数 x は `t` としてソルバーでは使用されます。

以下のスクリプトでは，ソルバーに積分区間内の評価点を決めさせる方法（デフォルト）と，ユーザが必要となる評価点を指定する方法で，それぞれ数値解を求め，そのグラフを描いています。後者のようにしておくと，積分区間におけるグラフを滑らかに描くことができるようになります。

```
# plot_ode_ivp.py: 常微分方程式の初期値問題と解プロット
import numpy as np
import scipy.integrate as scint # ODE ソルバー
import matplotlib.pyplot as plt # グラフ描画

# 陽的形式の右辺
# y' = func(t, y) = y
def func(t, y):
```

```
  return y

# 初期値
# y(0) = 1
y0 = [1.0]

# t = [0, 1]
t_interval = [0.0, 1.0]
print(t_interval)

# 常微分方程式を解く
ret = scint.solve_ivp(func, t_interval, y0) # 評価点tが可変になる
ret_fix = scint.solve_ivp(
  func, t_interval, y0,
  t_eval=np.linspace(t_interval[0], t_interval[1], 10)
) # 評価点tが固定化される

# 結果を表示
print(ret)

# yを表示
print(ret.y)

# t-yグラフを描画
# グラフ初期化
figure, axis = plt.subplots()

# 値をセット
axis.plot(ret.t, ret.y[0, :], label='Automatic')
axis.plot(ret_fix.t, ret_fix.y[0, :], label='t_eval')

# x軸，y軸，グラフタイトルをセット
axis.set(xlabel='t', ylabel='y', title='y\' = y')

#␣グリッドを描画
axis.grid()

#␣凡例
axis.legend()

#␣グラフ保存ファイル名
figure.savefig('ode.png')

#␣グラフを画面に描画
plt.show()
```

問題 6.4

1. 上記のスクリプトに問題 5.4 のグラフを描く機能を追加して実行せよ。
2. 上記のスクリプトに真の解を計算する関数を追加し，数値解の相対誤差を求めてグラフとして描画せよ。

6.5　ワードクラウドを作る

ワードクラウド (word cloud) は，テキストファイルに含まれる単語全ての出現回数をカウントし，大きい順に大きく表示し，雲のようにまとめて表示したものです。下記に青空文庫 (https://www.aozora.gr.jp/) に登録されている 2 作品のワードクラウドを表示したものを示します。

図 6.4　ワードクラウドの例:「本はどのように消えていくのか」(左) と「吾輩は猫である」(右)

ソースコード 6.4　wordcloud.py

```
# wordcloud.py: WordCloud の生成
# https://github.com/WorksApplications/sudachi.rs/tree/develop/python
# WordCloud: https://amueller.github.io/word_cloud/
# Requests: https://requests.readthedocs.io/en/latest/
import sudachipy as sdc
import requests # Requests
import re # Regular expression
import string # string.split
import numpy as np # wordcloud では必須
import wordcloud
import matplotlib.pyplot as plt # グラフ描画

# 青空文庫にアクセス
#富田倫生「青空のリスタート」
#url = 'https://www.aozora.gr.jp/cards/000055/files/698_51256.html'
# 夏目漱石「吾輩は猫である」
url = 'https://www.aozora.gr.jp/cards/000148/files/789_14547.html'
#url = 'https://www.aozora.gr.jp/'
r = requests.get(url)
if r.status_code != 200: # 正常アクセス時のみ
    print(url +'cannot␣be␣read...')
    exit # 終了

#r.encoding = 'utf-8' #強制的にUTF-8 encoding
r.encoding = 'shift_jis' # Shift-jis encoding
print('encoding␣=␣', r.encoding)
#print('r.text = ', r.text)

# 正規表現ですべてのタグ<*>を消去
# https://stackoverflow.com/questions/9662346/python-code-to-remove-html-tags-from-a-
    string
```

```
plaintext = re.sub(re.compile(r'<[^>]+>'), '', r.text)
print('plaintext␣=␣', plaintext)

# 一行ごとに分割
plaintext_lines = plaintext.split('\n')

# SudachiPy による分かち書きと品詞同定
tokenizer_obj = sdc.Dictionary().create()
#morphemes = tokenizer_obj.tokenize("吾輩は猫である。名前はまだない。")
#morphemes = tokenizer_obj.tokenize(plaintext)

# "普通名詞"のみ取り出し
noun = []

for line in plaintext_lines:
    morphemes = tokenizer_obj.tokenize(line)

    for m in morphemes:
        #print(
        # 'm = ',
        # m.surface(), # 単語
        # m.reading_form(), # 単語読み(カタカナ)
        # m.part_of_speech() # 品詞情報
        #)

        info = m.part_of_speech()
        #print('info = ', info[0], info[1], info[2])
        if info[0] == "名詞" and info[2] == "一般":
            noun.append(m.surface()) # 単語追加

# 名詞&一般のみ表示
#print('noun = ', noun)

# 辞書型
noun_count = {}
for word in noun:
    # 初見の単語数はゼロリセット
    if word not in noun_count:
        noun_count[word] = 0

    noun_count[word] += 1

# 単語数の多い順に逆順ソート
sorted_noun_count = sorted(noun_count.items(), reverse=True, key=lambda x:x[1])
print(sorted_noun_count)

# 名詞を半角スペースで接続
max_word_num = 100 # 100単語まで
word_list = ""
word_count = 0
for item in sorted_noun_count:
    word_list += item[0] + "␣"
    word_count += 1
    if word_count >= max_word_num:
```

```
85          break
86
87  print(word_list)
88
89  # Word cloud生成
90  # https://moji.or.jp/ipafont/ipaex00401/
91  ttf_path = './ipaexg.ttf'
92  #ttf_path = 'C:/windows/fonts/gothic.ttf'
93
94  word_list_cloud = wordcloud.WordCloud(font_path=ttf_path,background_color="white").
        generate(word_list)
95  plt.imshow(word_list_cloud, interpolation='bilinear')
96  plt.axis('off') # 軸表示なし
97
98  # WordCloud 表示
99  plt.show()
```

問題 6.5

青空文庫から好きな作品を一つ選び，ワードクラウドを作れ。

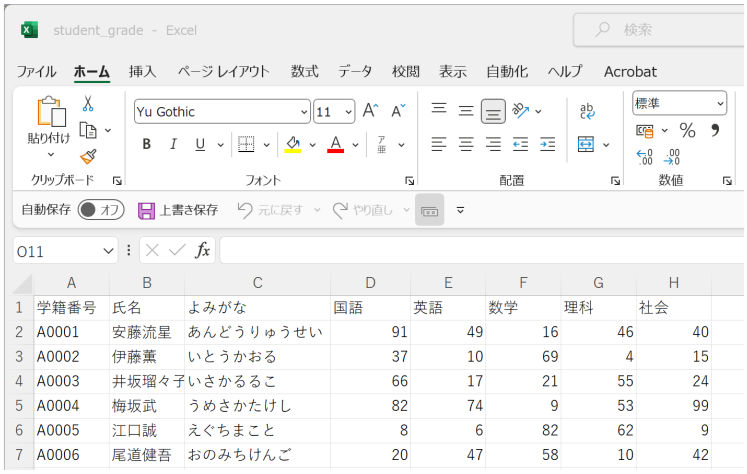
第 7 章

基盤モジュール: Pandas

表計算 (table calculation) ソフトウェアは，その名の通り，数値や文字列を含むデータを表にまとめて整理するためのデータ処理ツールです。Microsoft 社が開発した Excel（エクセル）は代表的な表計算ソフトウェアで，長い歴史を誇ります。その他，無料で使用することのできる Google Sheets や，Apache 財団の OpenOffice Calc などがあり，いずれも表を用いたデータ処理やグラフく作成機能を備えています。このように電子化された表計算ソフトウェアのデータは，あらかじめ整理されている分，Python にとって扱いやすいものと言えます。本章では主として Pandas パッケージを用いた Excel ファイルの扱い方を学び，簡単な統計ツールを作っていきます。

7.1 Excel と表

今回使用する Excel ファイルは図 7.1 に示すような，個人ごとに 5 科目（国語，英語，数学，理科，社会）の得点が記されているものとします。



	A	B	C	D	E	F	G	H
1	学籍番号	氏名	よみがな	国語	英語	数学	理科	社会
2	A0001	安藤流星	あんどうりゅうせい	91	49	16	46	40
3	A0002	伊藤薫	いとうかおる	37	10	69	4	15
4	A0003	井坂瑠々子	いさかるるこ	66	17	21	55	24
5	A0004	梅坂武	うめさかたけし	82	74	9	53	99
6	A0005	江口誠	えぐちまこと	8	6	82	62	9
7	A0006	尾道健吾	おのみちけんご	20	47	58	10	42

図 7.1　今回使用する Excel ファイルの内容

まず，Excel で簡単な集計を行ってみましょう。追加するのは

① 個人ごとの平均値 (Excel の average 関数を使用)，中央値 (median 関数)，標準偏差 (stdev.p 関数)
② 科目ごとの平均値，中央値，標準偏差をグラフ化する
③ 科目ごとに上位 5 名のリストと得点（棒グラフも含む）

とします。元のデータは別シートを作成し，そちらにすべてのデータを移して作業してみて下さい。完成版は図 7.2 のようになります。

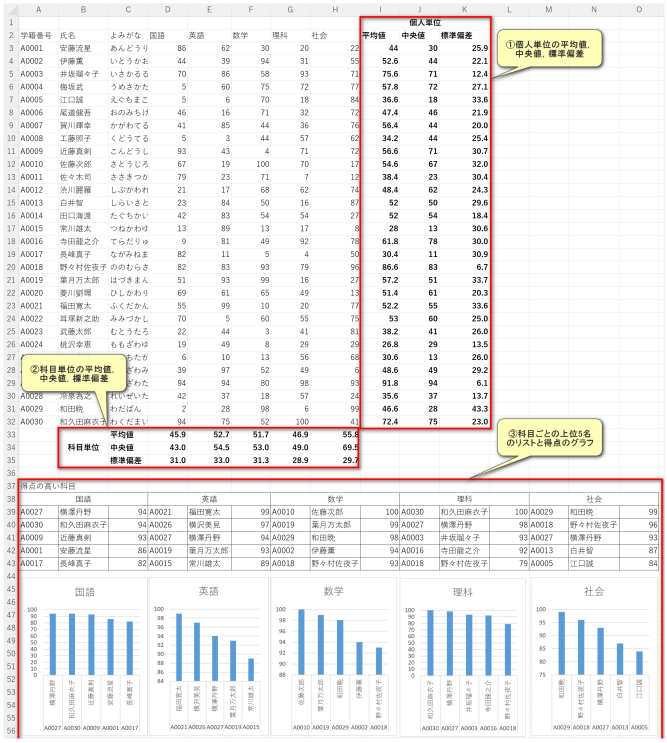


図 7.2　統計処理とグラフを付加した Excel シート

必ずしものようなレイアウトで作る必要はありませんので，自分で創意工夫したレイアウトで作成してみて下さい。ただし，次の点は気を付けて下さい。

1. 統計値はせいぜい小数点 1 桁で表示すること。
2. 得点のグラフは必ず最大値を 100 点に統一すること。

Excel だけでもこれらの処理は可能ですが，例えばこれに個人単位の得点グラフを付けろとか，個人ごとにレポート形式にせよと言われると，手作業では手間がかかります。

ということで，ここから先は Python で作業を肩代わりしてみましょう。

問題 7.1

図 7.2 を作成し，次のデータとグラフを追加せよ。

1. 科目別の最小点と最大点
2. 科目別の平均点のグラフ

7.2 Pandas を用いた Excel ファイルの読み出し

では図 7.1 に示した Excel ファイル student_grade.xlsx を Python から読み出し，図 7.2 に示すような科目別と個人別の平均値を導出してみましょう。ここで作成する Python スクリプトは「student_grade.py」とします。

7.2.1 Excel ファイルの読み出しと科目ごとの統計値算出・グラフ描画

まず pandas と japanize_matplotlib モジュールをインストールしておき，下記の部分を作成して「student_graphde.xlsx」が読み出せるか，確認して下さい。

ソースコード 7.1 student_grade.py(1/3)

```
1  # student_grade.py: 生徒の成績統計計算
2  import numpy as np # NumPy
3  import pandas as pd # Pandas
4  import matplotlib.pyplot as plt # Matplotlib
5  import japanize_matplotlib # Matplotlib 日本語対応
6
7  # Excel ファイル名
8  filename = 'student_grade.xlsx'
9  # Excel シート名
10 sheetname = '学籍番号・氏名・タイトルなし'
11 # 教科名
12 subjects = ['国語', '英語', '数学', '理科', '社会']
13
14 # Excel ファイル読み込み確認
15 try:
16     pd.ExcelFile(filename)
17 except:
18     print(filename, 'が開けませんでした。')
19
20 # Excel ファイル読み込み時のみ動作
21 with pd.ExcelFile(filename) as xls:
22     sheet = pd.read_excel(xls, sheetname)
23     print(sheet)
24
25     # 科目ごとの平均点,標準偏差,最高点,最低点
26     for i in range(len(subjects)):
27         print('{:s}の平均点,標準偏差,最高点,最低点:␣{:3.1f},␣{:5.3g},␣{:4d},␣{:4d}'.format
               (
28             subjects[i],
29             np.average(sheet[subjects[i]]),
30             np.std(sheet[subjects[i]]),
31             np.amax(sheet[subjects[i]]),
32             np.amin(sheet[subjects[i]])
33         ))
```

この部分の動作確認ができれば，科目ごとの平均点を棒グラフ化します。

ソースコード 7.2　student_grade.py(2)

```
55    # 平均点のグラフ化
56    x = subjects # ['国語', '英語', '数学', '理科', '社会']
57    y = [np.average(sheet[x[i]]) for i in range(5)]
58    fig, ax = plt.subplots()
59    # 棒グラフ
60    ax.set_title('科目別平均点')
61    ax.set_ylabel('得点')
62    ax.bar(x, y)
63    # グラフ表示
64    plt.show()
```

グラフは図 7.3 のように表示されます。

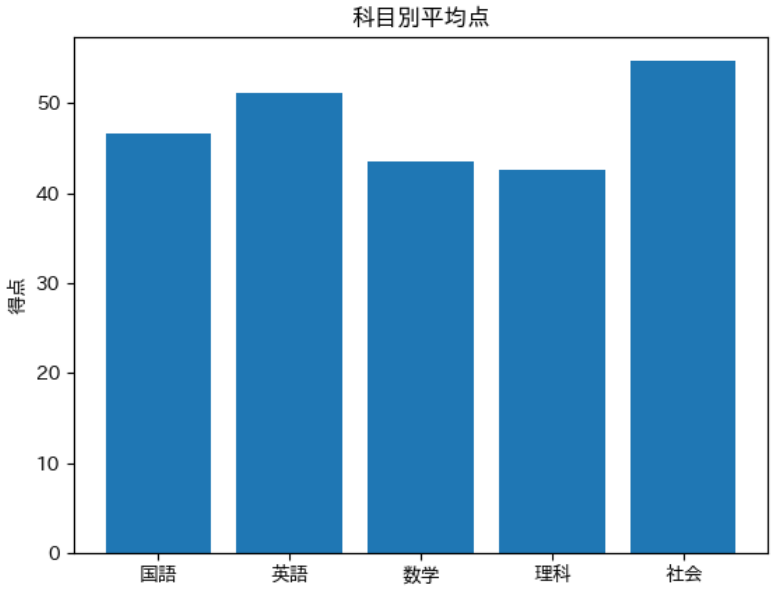


図 7.3　科目別平均点のグラフ

問題 7.2

中央値，標準偏差も導出しグラフ化せよ。

7.2.2　生徒ごとの統計値の導出

次に，生徒ごとの統計値の導出と，グラフ描画を行います。

ソースコード 7.3　student_grade.py(2)

```
63    # 生徒ごとの統計値
64    student_name = input('生徒の氏名：')
65    find_row = sheet.loc[sheet['氏名'] == student_name]
66
67    # 生徒が見つかったときのみ動作
```

```
68    if(find_row.empty == False):
69        print(student_name, '␣->␣\n', find_row)
70        #print(subjects[0], '::', find_row[subjects[0]].to_numpy()[0])
71        #print(student_name, ' -> \n', find_row.to_numpy())
72        #for i in range(len(subjects)):
73        # print(subjects[i], ' -> ', find_row[subjects[i]])
74
75        #find_row_scores = find_row.loc[student_name, subjects[0]]
76        find_row_scores = [find_row[subjects[i]].to_numpy()[0] for i in range(len(
              subjects))]
77        print(find_row_scores)
78
79        # 個人成績のグラフ化
80        x = subjects # ['国語', '英語', '数学', '理科', '社会']
81        y = find_row_scores # [find_row[x[i]] for i in range(5)]
82        #print(y)
83        fig, ax = plt.subplots()
84        # 棒グラフ
85        ax.set_title(student_name + 'の得点')
86        ax.set_ylabel('得点')
87        ax.bar(x, y)
88        # グラフ表示
89        plt.show()
90
91        print(student_name, 'の平均点,␣標準偏差,最高点,最低点:␣',
92            np.average(find_row_scores),
93            np.std(find_row_scores),
94            np.amax(find_row_scores),
95            np.amin(find_row_scores)
96        )
97
98    else:
99        print('名前:␣', student_name, '␣が見つかりません。')
```

個人別の得点グラフは図 7.4 にようになります。

問題 7.3

次の処理を追加せよ。

1. 科目ごとの上位 5 名のリストアップと棒グラフ化
2. 科目ごとの下位 5 名のリストアップと棒グラフ化

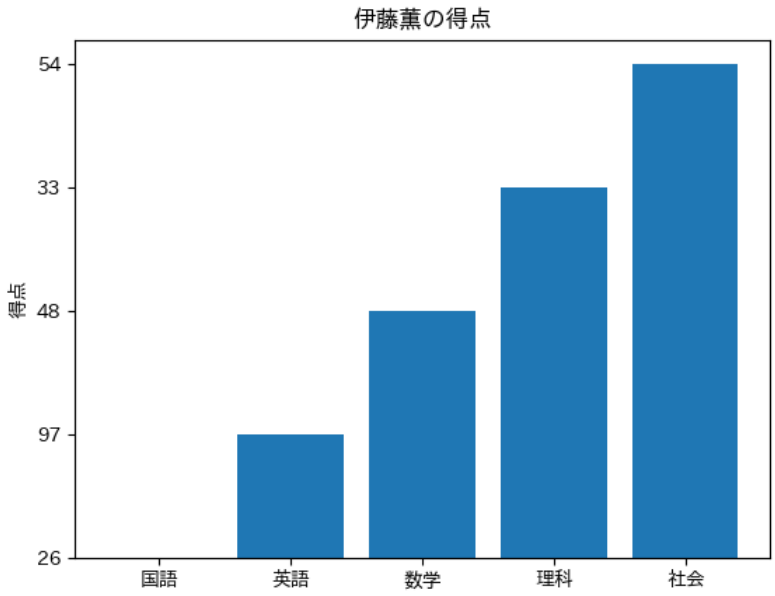


図 7.4　個人別得点グラフ

第 8 章

Flask による Web アプリの構築

いわゆる「ホームページ」や「Web サイト」「Web ページ」と呼ばれているものは，ネット内でいつでもアクセスできる Web サーバ (Server) 上で提供される一種のアプリケーションソフトウェア（アプリ）を意味します。Web アプリを作成するための開発環境は様々なものがあり，日々，新しいものが提供されています。本章ではその中でも Python をベースとした「Flask」というモジュールの使い方を簡単に紹介します。

8.1 Web アプリケーションの概要

Web(「うぇぶ」と発音，蜘蛛の巣の意味) とは，インターネット上で公開されている Web サーバ (Web server) にある情報を (Web) ブラウザ (Web browser) で閲覧するシステムを意味します。Web サーバもブラウザも，基本は一台のマシン (スマートフォン，パソコン等の機器) 上で動作するネイティブアプリケーション (native application) ですが (8.1 の左図)，インターネットを介して Web サーバとブラウザが強調して動作する Web システム全体がアプリケーションの一種と言えますので，これを Web アプリケーション (8.1 の右図) とも呼称します。

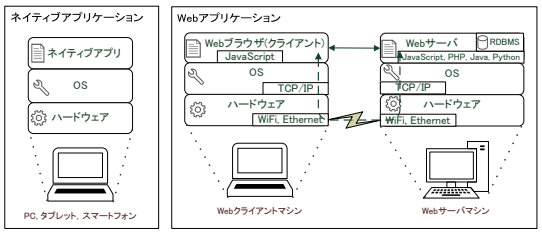


図 8.1　ネイティブアプリケーション（左）と Web アプリケーション

Web アプリケーションは，ユーザ側にブラウザさえあれば，いつでもアクセスできる Web サーバ上にある機構を使って様々なサービスを提供できるため，小さな組み込み機器 (IoT, Internet of Things) からスマートフォン，タブレット，パソコン等に至るまで，ハードウェアを選ばずに使用することができます。Web アプリケーションを作るためのソフトウェアも多様な組み合わせで開発でき，Python 環境でも，ここで取り上げる Flask 以外に，Django，FastAPI 等，様々なモジュールが Web アプリケーション開発用として提供されています。

Flask はその中でも簡単な機能を作る事に向いており，Web サーバの側で動作する Web アプリケーショ

ンを手軽に作成することができます。実際にインターネット上で公開するにはサーバ環境を用意する必要がありますが，ここでは自分で作って自分で遊べるごく簡単な Web アプリケーションを作ってみることにしましょう。

8.2 HTML と CSS

まず，ブラウザで閲覧できる Web ページを HTML(Hyper Text Markup Language) ファイルとして作ってみましょう。今まで Python ソースコードを作ってきたテキストエディタで，あ，下記の内容を打ち込み，適切なフォルダに「first.html」という名前で保存して下さい。Python とは異なり，インデントはしてもしなくても問題ありませんが，構造を見やすくするためにあえて行っています。「自分の学籍番号」「Student Number」は自分の学籍番号に，「氏名」「Name」は自分の氏名に置き換えて記述して下さい。

ソースコード 8.1 first.html

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="UTF-8" />
5          <title>The first HTML file</title>
6      </head>
7      <body>
8          <h1>最初のHTML ファイル</h1>
9          <h2>自分の学籍番号　氏名</h2>
10         <hr />
11         ここに本文を書く。
12         <hr />
13         <address>Copyright (c) 20xx Student Number, Name</address>
14     </body>
15 </html>
```

一読して分かることは，HTML ファイルの特徴として，タグ (tag) と呼ばれる**<タグ>**...**</タグ>**というブロックで文書が構造化されていることです。開始タグ (**<タグ>**) と終了タグ (**</タグ>**) がセットになっているものは

1. <html>...</html> HTML の中身 (省略可)
2. <head>...</head> Web ページのヘッダ情報 (ブラウザ本体には表示されない)
3. <title>...</title> Web ページタイトル (ブラウザのタブに表示される)
4. <body>...</body> Web ページの本文（ブラウザ本体に表示される）
5. <h1>...</h1> トップの見出し
6. <h2>...</h2> 2 番目の見出し
7. <address>...</address> アドレス (著作権表示などを書く)

です。

また，<!DOCTYPE html>(HTML5 の宣言)，<meta>（Web ページの情報），<hr>（水平線）は終了タグ (**</タグ>**) のない単独タグとなっています。このように文書の構造を表現するタグはたくさんありますので，興味のある方は調べてみて下さい。

では，この first.html を，Chrome, Edge, Safari といったブラウザで直接開いてみましょう。図 8.2 の

ような表示がされるはずです。

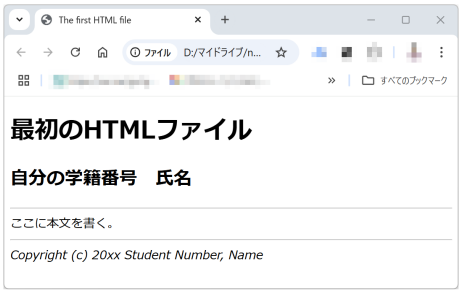


図 8.2　first.html を Chrome（ブラウザ）で閲覧したもの

これはあくまで「デフォルト」の表示ですので，ブラウザによっては異なる見栄えになるかもしれません。もう少しデザイン的に凝りたければ，CSS(Cascading Style Sheet) という機能を使って，フォントの書式や色，背景色や配置を自在に変更することができますので，興味のある方は調べてみて下さい。

以降ではこの HTML ファイルの構造を基本にして，Web サーバ側で動作する Web アプリケーションを作っていきます。

問題 8.1

本文に箇条書きを行う機能として`<ol>...</ol>`(Ordered List, 順序付き箇条書き)と`<ul>...</ul>`(Unordered List, 番号なし箇条書き）がある。どちらも項目は`<li>...</li>`(LIst, 箇条書きの項目) をこれらのタグの中に挿入して形成する。この 2 種類の箇条書きを使って図 8.3 という本文表示ができるよう，`first.html` を修正して `second.html` を作れ。

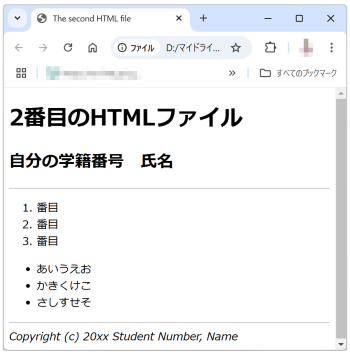


図 8.3　second.html を Chrome（ブラウザ）で閲覧したもの

8.3 Flask で Hellow, Python!

Python 最初のスクリプト (1.1) がそうであったように,「ようこそ，Flask の世界へ！」という一文のみをコンソール（コマンドライン画面）とブラウザに表示する「hellow」アプリを作ってみましょう。先に，pip コマンド等を用いて Flask モジュールをインストールした Python 環境を用意しておきます。

次に，適当なフォルダ位置に「flask」フォルダを作成して下さい，以降の Web アプリはこの中にサブフォルダ単位で作成していくことにします。今回は「flask」フォルダ内に「hellow」フォルダを作成し，flask\yen hellow フォルダで作業を行っていきます。

ソースコード 8.2 flask¥hellow¥app.py

```
1  # flask\hellow\app.py: 最初のFlaskアプリ
2  from flask import Flask # flaskモジュールからFlaskクラスをインポートする
3
4  app = Flask(__name__) # Flaskのインスタンスを appに生成する
5
6  # トップページアクセス時の処理
7  @app.route('/')
8  def hellow_world():
9      mojiretsu = 'ようこそ，Flaskの世界へ！'
10     print(mojiretsu) # コンソールに表示
11     return mojiretsu # ブラウザに出力
```

1. app.py をテキストエディタで作成し，flask¥hellow フォルダに保存する。
2. 次のコマンドで Flask を起動する。

   ```
   $ export FLASK_ENV='development' ←開発用モード（ソースコードの自動更新あり）の師弟
   ```

3. ローカルマシンを Web サーバ（ポート番号は 5000 がデフォルト）としてアプリケーションを起動

   ```
   $ flask run
   ```

   上記が動かない場合は python -m flask run と指定
4. 起動したら，適当なブラウザ (Google Chrome か Edge を推奨) で

   ```
   http://127.0.0.1:5000/
   ```

   にアクセスし，図 8.4 のように retun で返される文字列 (mojiretsu 変数の中身) が表示されていることを確認する。

また，この際，コマンドライン画面（コンソール）にも文字列が表示されていることも確認して下さい。これは print 文の処理によるものです。

問題 8.2

ブラウザのタブを複数開いて http://127.0.0.1:5000/にアクセスできることを確認せよ。

図 8.4　最初の Flask アプリ実行画面（ブラウザ）

8.4　HTML テンプレートの使用

とりあえずブラウザに文字を表示させることはできましたが，これでは HTML ファイルに比べて情報が少なすぎて物足りません。そこで，テンプレート (template) 機能を用いて，必要な個所のみ書き換えを行って HTML ファイルを表示できるよう，|app.py—を書き換えてみましょう。変更箇所は下記の通りです。

ソースコード 8.3　flask¥hellow¥app.py

```
1  # app.py: 最初のFlaskアプリ(HTMLテンプレートの使用版)
2  from flask import Flask
3  from flask import render_template # 追加
4
5  app = Flask(__name__)
6
7  # トップ
8  @app.route('/')
9  def hellow_world():
10     mojiretsu = 'ようこそ，Flaskの世界へ！'
11     # print(mojiretsu) # コメントに
12     # return mojiretsu # コメントに
13     return render_template('index.html', mystr = mojiretsu) # 追加
```

flask モジュールから，render_template 関数をインポートし，これを用いて return で返す文字列を HTML ファイルに変更します。

次に，「hellow」フォルダ内に「templates」フォルダを生成し，この中に下記の template.html を作ります。前に作成した first.html をコピペして必要なところのみ書き換えて下さい。

ソースコード 8.4　flask¥hellow¥templates¥template.html

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="UTF-8" />
5          <title>First Flask Application by 学籍番号,自分の名前</title>
6      </head>
7      <body>
8          <h1>最初のFlask アプリケーション</h1>
9          <h2>自分の学籍番号　氏名</h2>
10         <hr />
11         {% block main_content %}
12         {% endblock %}
```

```
13          <hr />
14          <address>Copyright (c) 20xx 学籍番号，氏名</address>
15      </body>
16  </html>
```

特徴的なのは，本文に{% block ブロック名 %}{% endblock %}という記述があることです。このブロック部分のみ，アプリケーションの動作に応じて書き換えることができるようになります。「templates」フォルダ内に index.html を下記のように記述すると，{{ mystr }}のところが，アプリケーションから渡される変数 mystr で書き換えられます。

ソースコード 8.5　flask¥hellow¥templates¥index.html

```
1  {# index.html #}
2  {% extends 'template.html' %}
3  {% block main_content %}
4  <!-- 文字列を挿入 -->
5  {{ mystr }}
6  {% endblock %}
```

前回同様，app.py を起動させ，ブラウザで閲覧すると 8.5 のように，HTML ファイルとして評させることができるようになります。

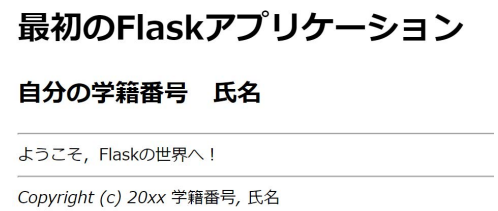


図 8.5　HTML テンプレート使用時

問題 8.3

8.3 と同じ表記を行えるようにテンプレートを使用した「second」アプリを作成し，ブラウザを動作させて実行結果を確認せよ。

8.5　フォームを用いた数式の計算

次に，1 変数関数 $f(x)$ を x を用いた式としてユーザが与え，指定した x の値を用いて $f(x)$ の値を計算する Web アプリ「func_calc」を作ってみましょう。そのためには数式を文字列として与え，それを計算式として評価 (evaluate) する式パーサ (formula parser) の機能が必要になります。

インタプリタ言語の特徴の一つに，スクリプトの中でスクリプト文を動作する事ができるということが挙げられます。Python の場合は eval 関数というものがあり，例えば下記のように，x**2 + 3 という文字列を str_formula 変数に与えて eval 関数の引数として渡して実行すると，実行時における変数 x の値を評価して計算してくれます。

```
>>> str_formula = 'x**2 + 3'
```

```
>>> x = 3
>>> eval(str_formula)
12
```

使いようによっては大変便利な関数ですが，Web プログラミングでは，ユーザからの入力は危険なものも含まれているという前提でスクリプトを組む必要があり，eval 関数の使用は望ましくありません。

そこで，NumPy と組み合わせての動作も可能な式パーサを Pytnon で使用できるようにした NumExpr[**?**] を使用することとします。pip コマンドなどで NumExpr をインストールし，上記の計算が可能であることを確認しましょう。

```
>>> import numexpr as ne
>>> str_formula = 'x**2 + 3'
>>> x = 3
>>> float(ne.evaluate(str_formula))
12.0
```

eval 関数とは異なり，形指定をして値を取り出す必要があります。

この式パーサを使って, ユーザから与えられた数式を計算するフォーム (form) を作ってみましょう。フォームとは，HTML のタグの一つで，下記のようにブラウザ経由でユーザからの入力を受け付ける機構を提供してくれます。まずは最小限の機能を使って数式計算用のフォームを作ってみましょう。前節で作ったテンプレート (template.html) を使い index.html(図 8.6) を作ります。

ソースコード 8.6 flask¥func_calc¥template¥index.html

```
1  {# index.html: 関数計算 #}
2  {% extends 'template.html' %}
3  {% block main_content %}
4      <!-- 入力フォーム -->
5      <h2>関数入力フォーム</h2>
6      <form action="/calc" method="POST">
7      <ul>
8          <li>関数式（「x」を変数として指定）: <input type="text" name="input_formula" size
               ="30" /></li>
9          <li>x の値：<input type="text" name="input_x" size="10" /></li>
10     </ul>
11     <p><input type="submit" value="計算実行！"> <input type="reset" value="リセット
           "></p>
12     </form>
13     <hr />
14     <!-- 計算値出力 -->
15     <h2>計算結果</h2>
16     <p>x = {{ str_x }}</p>
17     <p>f(x) = {{ str_formula }} = {{ str_func_val }} </p>
18 {% endblock %}
```

フォームは<form>...</form>で囲まれている部分で，この中の<input>タグでユーザからの入力を受け付けます。ここで使用している入力タイプは下記の 3 つです。

text ・・・テキストボックス (文字列)

submit　・・・サブミットボタン (入力データの送信)
reset　・・・フォーム内入力のリセット

サブミットボタンがクリックされる際に入力されていたテキストボックスなどの入力値に，それぞれ name 属性で指定された文字列がセットされ，<form>の action 属性で指定された URL に，method で指定された手法 (この場合は POST メソッド) で入力データが転送されます。

例えば $x = 3$ の時，$x^2 + 3 = 3^2 + 3 = 12$ を計算したい場合は，図 8.6 のように実行します。

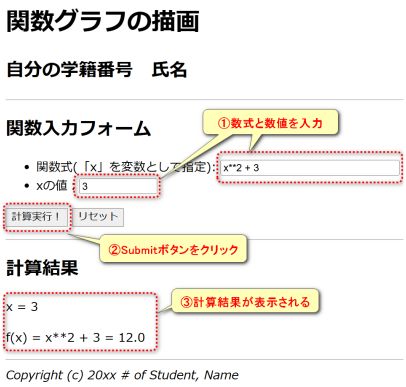


図 8.6　関数値計算アプリのフォーム

この環数値計算アプリの動作を実現するためには，フォームからの数式と x の値を受け取って処理する必要があります。「/calc」にアクセスされた際にはフォームの値が入力されているという前提で処理を行うようにすると，下記のように次のようになるでしょう。

ソースコード 8.7　flask¥func_calc¥app.py

```
1  # app.py : 関数式の計算
2  from flask import Flask
3  from flask import render_template
4  from flask import request # フォーム入力を受け取る
5  from numpy import * # NumPy 関数を直接呼出し
6  import numexpr as ne # NumExpr: 式パーサ
7
8  app = Flask(__name__)
9
10 # 関数フォームを開く
11 @app.route('/')
12 def index():
13     return render_template('index.html')
14
15 # 関数値の評価
16 @app.route('/calc', methods = ['POST']) # POST でのみ受付
17 def calc():
18     str_formula = request.form['input_formula']
19     str_x = request.form['input_x']
20     # 計算実行
21     x = float(str_x) # x を数値に変換
22     func_val = float(ne.evaluate(str_formula))
```

```
23      # 計算結果を戻す
24      return render_template('index.html', str_x = str_x, str_formula = str_formula,
            str_func_val=str(func_val))
```

フォームから渡されるデータは `request.form` リストに保存されており，`<input>`タグの `name` 属性値をインデックスとして参照できます。

問題 8.4

x の値と関数 f が次のように与えられるときの $f(x)$ の値を func_calc アプリを使って求め，実行時のブラウザ画面のスナップショットを画像として保存せよ。

1. $x = -5,\ f(x) = (x-3)(x+4)(x-1)$
2. $x = -3,\ f(x) = \exp(\cos(x^2)) = e^{\cos(x^2)}$

8.6　数式に基づくグラフ描画アプリ

関数値計算アプリを拡張し，指定区間 $x \in [a, b]$ における関数 $f(x)$ のグラフを描画する「graph」アプリを作ってみましょう。図 8.7 のようなフォームを `index.html` として作り，関数グラフを PNG 画像としてブラウザ上に表示するようにします。

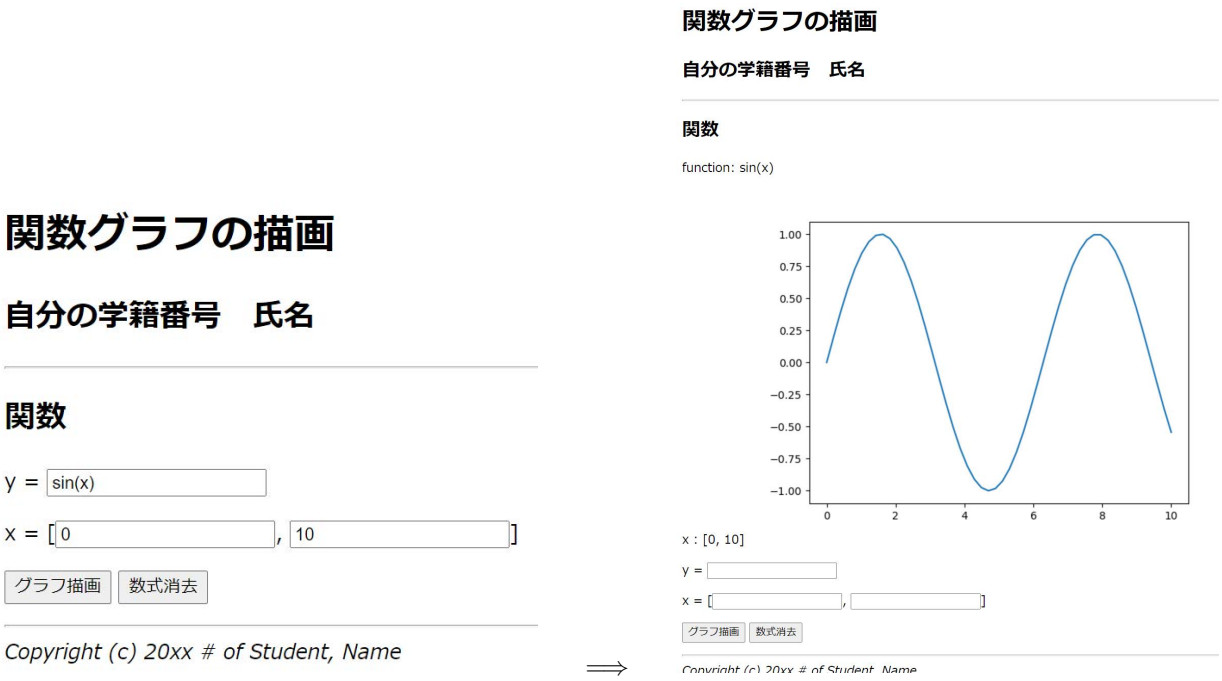


図 8.7　Flask による関数グラフ描画ツールの実行画面

`flask¥graph¥templates¥template.html` は省略し，下記の `index.html` を入力フォームとして使えるようにしておいて下さい。

ソースコード 8.8　`flask¥graph¥templateshtml`

```
{# index.html: 数式入力フォーム＆グラフ描画画面 #}
{% extends 'template.html' %}
{% block main_content %}
    <h2>関数</h2>
        {% if formula %}
        <div name="graph">
            function: {{ formula }}<br />
            x : [{{ x_min }}, {{ x_max }}]
            <!-- y = {{ y_vals }} -->
            <img src="{{␣img_name␣}}" alt="{{␣formula␣}}" />
        </div>
        {% endif %}
        <form action="/draw" method="POST">
            <p>y = <input type="text" name="formula" length="20" /></p>
            <p>x = [<input type="text" name="x_min" length="5" />, <input type="text"
                name="x_max" length="5" />]</p>
            <p><input type="submit" value="グラフ描画" /> <input type="reset" value="数式
                消去" /></p>
        </form>
{% endblock %}
```

描画した関数は PNG ファイルとして保存され，ファイルの位置が `img_name` として渡される，という前提で`<img>`タグの指定が行われています。

この入力フォームからの区間の端点を a =`x_min`，b =`x_max` として与え，関数式は `formula` として与えます。

グラフを描画するためには区間を分割し，各分割点で関数値を計算する必要があります。ここでは `div` で分割数を指定しています。

ソースコード 8.9　flask¥graph¥app.py

```
# graph/app.py : 関数グラフの描画
from flask import Flask
from flask import render_template
from flask import request # methods as argument
from flask import redirect
from flask import Response
from numpy import * # NumPy の関数を使用する
import numexpr as ne # 式パーサ
import matplotlib.pyplot as plt # グラフ描画ライブラリ

app = Flask(__name__)

# 固定ディレクトリ
PNG_DIR = 'static/'

# Cache コントロール用
# https://stackoverflow.com/questions/45583828/python-flask-not-updating-images
app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0

#@app.after_request
def add_header(response):
    response.headers['Cache-Control'] = 'no-store,␣no-cache,␣must-revalidate,␣post-
        check=0,␣pre-check=0,␣public,␣max-age=0'
    response.headers['Pragma'] = 'no-cache'
```

```
24      response.headers['Expires'] = '0'
25      return response
26
27
28  @app.route('/')
29  @app.route('/index')
30  def index():
31      return render_template('index.html')
32
33
34  @app.route('/draw', methods = ['GET', 'POST'])
35  def draw():
36      formula_text = request.form['formula']
37      x_min = request.form['x_min']
38      x_max = request.form['x_max']
39      div = 50
40      graph_img_name = PNG_DIR + 'graph.png'
41
42      # 計算とグラフ描画
43      x_array = linspace(float(x_min), float(x_max), div)
44      y_array = [ne.evaluate(formula_text) for x in x_array]
45      fig, ax = plt.subplots()
46      ax.plot(x_array, y_array)
47      fig.savefig(graph_img_name)
48      return render_template('index.html', formula = formula_text, x_min = x_min, x_max
            = x_max, img_name = graph_img_name)
```

まずは図 8.7 の実行例のように動作するかどうか，$f(x) = \sin(x),\ x \in [0, 10]$ を入力して動作を確認してみましょう。

問題 8.5

次の関数グラフを描画せよ。なお，四則演算 (+, -, *, /) は省略せずに書くこと。例) ：例 $2x/3 + \cos(x^2)$ → `2 * x / 3 + cos(x * x)`

1. $(x-3)(x+4)(x-1),\ x \in [-5, 5]$
2. $\exp(\cos(x^2)),\ x \in [-3, 3]$

[できそうならチャレンジ！問題] 現状では x 区間の分割数を 50(`div=50`) に固定している。この分割数もユーザが入力して変更できるようにせよ。

8.7 統計ツールを Web アプリに

本章の最後に，前の章で作成した Excel ファイルを読み込み，各学生ごとの成績データをグラフ化する「statistic」アプリを作ってみましょう。実行時には図 8.8 のように，成績表示する学生を選べるようにフォームを使います。

テンプレートファイルは省略します。また，`index.html` で表示する学生選択のためのフォームは，学生数が多いため，Python スクリプトで自動生成しますので，ここでフォームは定義しないようにします。

ソースコード 8.10 `flask¥statistics¥templates¥index.html`

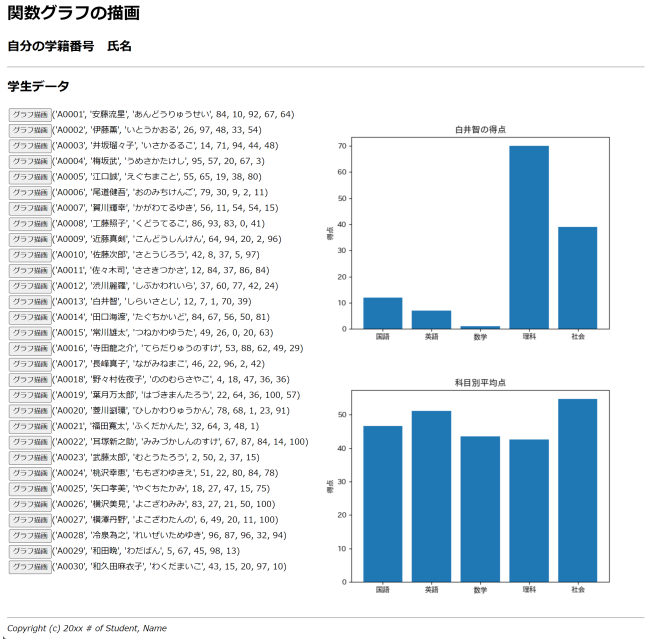


図 8.8　Flask による簡易統計ツール実行時の画面

```
{# index.html: 学生データ表示＆グラフ描画画面 #}
{% extends 'template.html' %}
{% block main_content %}
    <h2>学生データ</h2>
    <div name="table" style="float:left">
        {{ list|safe }}
    </div>
    <div name="graph">
        <img src="{{ graph1 }}" alt="{{ graph1_comment }}" />
        <img src="{{ graph2 }}" alt="{{ graph2_comment }}" />
    </div>
{% endblock %}
```

Python スクリプトは下記のようになります。

ソースコード 8.11　flask¥statistics¥app.py

```
# flask\statistics\app.py : CSV ファイルの描画
from flask import Flask
from flask import render_template
from flask import request # methods as argument
from flask import redirect
from flask import Response
import matplotlib # グラフ描画ライブラリ
matplotlib.use('Agg') # 再プロット時のエラー防止 (GUI 禁止)
import matplotlib.pyplot as plt # グラフ描画ライブラリ
import japanize_matplotlib # Matplotlib 日本語対応
# pandas
```

```
12 import numpy as np # NumPy
13 import pandas as pd # Pandas
14
15 app = Flask(__name__)
16
17 # 固定ディレクトリ
18 PNG_DIR = 'static/'
19
20 # Cache コントロール用
21 # https://stackoverflow.com/questions/45583828/python-flask-not-updating-images
22 app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0
23
24 # 選択した生徒の成績
25 find_row_student = {}
26 find_row_scores = []
27
28 # Excel ファイル名
29 filename = 'student_grade.xlsx'
30 # Excel シート名
31 sheetname = '学籍番号・氏名・タイトルなし'
32 # 教科名
33 subjects = ['国語', '英語', '数学', '理科', '社会']
34 # フィールド数（カラム数）
35 # 学籍番号,氏名,氏名読み仮名,教科名
36 max_num_column = 3 + len(subjects)
37 # 科目平均点グラフファイル名
38 subject_graph_name = 'subject_graph.png'
39 # 生徒別グラフファイル名
40 student_graph_name = 'student_graph.png'
41 # 選択した生徒の成績
42 #find_row_student = {}
43 # canvas_right, left_flag
44 canvas_right_flag = 0 # 存在なし
45 canvas_left_flag = 0 #
46
47 # 関数フォームを開く
48 @app.route('/')
49 @app.route('/index')
50 def index():
51     # Excel ファイル読み込み確認
52     try:
53         pd.ExcelFile(filename)
54     except:
55         print(filename, 'が開けませんでした。')
56
57     # Excel ファイル読み込み時のみ動作
58     with pd.ExcelFile(filename) as xls:
59         sheet = pd.read_excel(xls, sheetname)
60         #print(sheet)
61
62         # 科目ごとの平均点,標準偏差,最高点,最低点
63         for i in range(len(subjects)):
64             print('{:s}の平均点,標準偏差,最高点,最低点:␣{:3.1f},␣{:5.3g},␣{:4d},␣{:4d}'.
                    format(
```

```
65                  subjects[i],
66                  np.average(sheet[subjects[i]]),
67                  np.std(sheet[subjects[i]]),
68                  np.amax(sheet[subjects[i]]),
69                  np.amin(sheet[subjects[i]])
70              ))
71
72          # 平均点のグラフ化: subject_graph.png
73          x = subjects # ['国語', '英語', '数学', '理科', '社会']
74          y = [np.average(sheet[x[i]]) for i in range(5)]
75          fig, ax = plt.subplots()
76          # 棒グラフ
77          ax.set_title('科目別平均点')
78          ax.set_ylabel('得点')
79          ax.bar(x, y)
80          # グラフ表示
81          #plt.show()
82          # ファイルに書き出し
83          plt.savefig(PNG_DIR + subject_graph_name)
84
85          # 一行ごとに読み込み
86          table = '<table>'
87          print('sheet.shape:␣', sheet.shape)
88          for i in range(sheet.shape[0]):
89              # table += '<tr><td>' + str(tuple(sheet.iloc[i, :])) + '</td></tr>'
90              table += '<tr><td>'
91              table += '<form␣action="/draw"␣method="POST">'
92              table += '<input␣type="submit"␣value="グラフ描画">'
93              table += str(tuple(sheet.iloc[i, :]))
94              table += '<input␣type="hidden"␣name="student_id"␣value="' + sheet.iloc[i,
                    0] + '">'
95              table += '</form>'
96              table += '</td></tr>'
97              print(tuple(sheet.iloc[i, :]))
98
99          table += '</table>'
100
101     return render_template('index.html', list = table)
102
103 # 関数グラフ描画
104 @app.route('/draw', methods = ['GET', 'POST'])
105 def draw():
106
107     # 学生id取得
108     student_id = request.form['student_id']
109
110     # Excelファイル読み込み確認
111     try:
112         pd.ExcelFile(filename)
113     except:
114         print(filename, 'が開けませんでした。')
115
116     # Excelファイル読み込み時のみ動作
117     with pd.ExcelFile(filename) as xls:
```

```
118         sheet = pd.read_excel(xls, sheetname)
119         #print(sheet)
120
121         # 科目ごとの平均点,標準偏差,最高点,最低点
122         for i in range(len(subjects)):
123             print('{:s}の平均点,標準偏差,最高点,最低点:␣{:3.1f},␣{:5.3g},␣{:4d},␣{:4d}'.
                    format(
124                 subjects[i],
125                 np.average(sheet[subjects[i]]),
126                 np.std(sheet[subjects[i]]),
127                 np.amax(sheet[subjects[i]]),
128                 np.amin(sheet[subjects[i]])
129             ))
130
131         # 平均点のグラフ化: subject_graph.png
132         x = subjects # ['国語', '英語', '数学', '理科', '社会']
133         y = [np.average(sheet[x[i]]) for i in range(5)]
134         fig, ax = plt.subplots()
135         # 棒グラフ
136         ax.set_title('科目別平均点')
137         ax.set_ylabel('得点')
138         ax.bar(x, y)
139         # グラフ表示
140         #plt.show()
141         # ファイルに書き出し
142         plt.savefig(PNG_DIR + subject_graph_name)
143
144         # 一行ごとに読み込み
145         table = '<table>'
146         print('sheet.shape:␣', sheet.shape)
147         for i in range(sheet.shape[0]):
148             # table += '<tr><td>' + str(tuple(sheet.iloc[i, :])) + '</td></tr>'
149             table += '<tr><td>'
150             table += '<form␣action="/draw"␣method="POST">'
151             table += '<input␣type="submit"␣value="グラフ描画">'
152             table += str(tuple(sheet.iloc[i, :]))
153             table += '<input␣type="hidden"␣name="student_id"␣value="' + sheet.iloc[i,
                        0] + '">'
154             table += '</form>'
155             table += '</td></tr>'
156             print(tuple(sheet.iloc[i, :]))
157             # 学生id の発見
158             if sheet.iloc[i, 0] == student_id:
159                 find_row_student = sheet.iloc[i, 1]
160                 find_row_scores = [sheet.iloc[i, j] for j in range(3, 8)]
161
162         table += '</table>'
163
164         # 学生別グラフの描画
165         #find_row_scores = []
166         student_name = ''
167         if len(find_row_student) > 0 :
168             student_name = find_row_student
169             #find_row_scores = [find_row_student[subjects[i]] for i in range(len(
```

```
                subjects))]
170             print('find_row_scores␣=␣', find_row_scores)
171
172         # 個人成績のグラフ化
173         x = subjects # ['国語', '英語', '数学', '理科', '社会']
174         y = find_row_scores # [find_row[x[i]] for i in range(5)]
175         #print(y)
176         fig_right, ax = plt.subplots()
177         # 棒グラフ
178         ax.set_title(student_name + 'の得点')
179         ax.set_ylabel('得点')
180         if len(student_name) > 0:
181             ax.bar(x, y)
182         # グラフ表示
183         #plt.show()
184         plt.savefig(PNG_DIR + student_graph_name)
185
186     print('graph1,␣graph2␣=␣', student_graph_name, subject_graph_name)
187     return render_template('index.html', list = table, graph1 = PNG_DIR +
            student_graph_name, graph1_comment = '学生データ
            ', graph2 = PNG_DIR + subject_graph_name, graph2_comment = '教科別')
```