Python3モジュールブック

各種モジュールの基本的な使用方法

OpenCV / Pillow / pygame / NumPy / matplotlib / SymPy /

hashlib / Cython / Numba / ctypes

Copyright © 2017-2018, Katsunori Nakamura

中村勝則

2018年2月12日

目 次

1 画像の入出力と処理 1

1.1 OpenCV . . . 1
1.1.1 動画像の入力 . . . 2
1.1.2 ユーザインターフェース . . . 2
1.1.3 フレームのファイルへの保存 . . . 3
1.1.4 静止画像の読込み . . . 3
1.1.5 色の分解と合成 . . . 4
1.1.6 画像の類似性の検査 . . . 5
1.1.6.1 AKAZE 特徴量の算出 . . . 5
1.1.6.2 特徴量データの照合 . . . 6
1.1.6.3 サンプルプログラム . . . 6
1.2 Pillow . . . 8
1.2.1 画像ファイルの読込みと保存 . . . 8
1.2.2 Image オブジェクトの新規作成 . . . 9
1.2.3 画像閲覧 . . . 9
1.2.4 画像編集 . . . 10
1.2.4.1 画像の拡大と縮小 . . . 10
1.2.4.2 画像の部分の取り出し . . . 10
1.2.4.3 画像の複製 . . . 10
1.2.4.4 画像の貼り付け . . . 11
1.2.4.5 画像の回転 . . . 11
1.2.5 画像処理 . . . 11
1.2.5.1 色の分解と合成 . . . 11
1.2.6 描画 . . . 12
1.2.7 アニメーション GIF の作成 . . . 13

2 GUI とマルチメディア 14

2.1 pygame . . . 14
2.1.1 基礎事項 . . . 14
2.1.1.1 Surface オブジェクト . . . 14
2.1.1.2 アプリケーションの実行ループ . . . 14
2.1.2 描画機能 . . . 16
2.1.2.1 描画のサンプルプログラム . . . 19
2.1.2.2 回転，拡縮のサンプルプログラム . . . 20
2.1.3 キーボードとマウスのハンドリング . . . 22
2.1.4 音声の再生 . . . 23
2.1.5 スプライトの利用 . . . 24

3 科学技術系 29

3.1 数値計算と可視化のためのパッケージ：NumPy / matplotlib . . . 29
3.1.1 配列オブジェクトの生成 . . . 29
3.1.1.1 多次元配列の生成 . . . 31
3.1.2 データ列に対する演算：1 次元から 1 次元 . . . 31
3.1.3 データの可視化：基本 . . . 32
3.1.3.1 2 次元のプロット：折れ線グラフ . . . 33

3.1.3.2 複数のグラフの作成 35
3.1.4 乱数の生成 40
3.1.5 データの可視化：ヒストグラム，散布図 40
3.1.6 データ列に対する演算：n 次元から 1 次元 42
3.1.7 データの可視化：3 次元プロット 43
3.1.7.1 ワイヤフレーム 43
3.1.7.2 面プロット（surface plot） 43
3.1.8 高速フーリエ変換（FFT） 45
3.1.8.1 時間領域から周波数領域への変換：フーリエ変換 45
3.1.8.2 周波数領域から時間領域への変換：フーリエ変逆換 46
3.1.9 データの可視化：棒グラフ，グラフのアスペクトについて 49
3.1.9.1 棒グラフのプロット 49
3.1.9.2 プロットのアスペクト比の指定 49
3.1.10 フーリエ変換を使用する際の注意 49
3.1.11 行列の計算 49
3.1.11.1 行列の和と積 49
3.1.11.2 単位行列，ゼロ行列，他 50
3.1.11.3 行列の要素の編集 50
3.1.11.4 行列式と逆行列 51
3.1.11.5 固有値と固有ベクトル 51
3.1.11.6 その他 52
3.1.12 入出力 53
3.1.12.1 配列オブジェクトのファイル I/O 53
3.1.13 行列の比較 55
3.1.14 配列を処理するユーザ定義関数の実装について 55
3.2 SymPy 58
3.2.1 モジュールの読込みに関する注意 58
3.2.2 基礎事項 58
3.2.3 基本的な数式処理機能 61
3.2.4 解析学的処理 63
3.2.4.1 極限 63
3.2.4.2 導関数 63
3.2.4.3 原始関数 64
3.2.4.4 級数展開 65
3.2.5 各種方程式の求解 65
3.2.5.1 代数方程式の求解 65
3.2.5.2 微分方程式の求解 66
3.2.5.3 階差方程式の求解 66
3.2.6 線形代数 67
3.2.7 総和 68
3.2.8 数値近似 69
3.2.9 書式の変換出力 69
3.2.9.1 LaTeX 69
3.2.10 グラフのプロット 69

4 セキュリティ関連 **71**
4.1 hashlib . . . 71
4.1.1 基本的な使用方法 . . . 71

5 プログラムの高速化 **72**
5.1 Cython . . . 72
5.1.1 使用例 . . . 72
5.1.2 高速化のための調整 . . . 74
5.2 Numba . . . 75
5.2.1 基本的な使用方法 . . . 75
5.2.2 型指定による高速化 . . . 76
5.3 ctypes . . . 76
5.3.1 C 言語による共有ライブラリ作成の例 . . . 77
5.3.2 共有ライブラリ内の関数を呼び出す例 . . . 77
5.3.3 引数と戻り値の扱いについて . . . 78
5.3.3.1 配列データの受け渡し . . . 80

6 免責事項 **83**

1　画像の入出力と処理

1.1　OpenCV

OpenCV ライブラリは米インテル社によって開発され，現在は BSD ライセンスで配布されるオープンソースソフトウェアである．このライブラリは静止画像，動画像の入出力から画像処理，画像認識，機械学習のための機能を提供する．また，クロスプラットフォームのライブラリであり，インターネットサイト http://opencv.org/ から関連の情報を入手することができる．

OpenCV は独自の UI を実現する機能を備えており，画像の表示や，キーボード，マウスからの入力を受け付けるための簡便な機能も提供する．本書では OpenCV に関して導入的な内容について説明するので，更に詳しい事柄に関しては上記の情報サイトなどを参照すること．

このライブラリの使用に先立って，必要なソフトウェアを Python 処理系に予めインストールしておく必要がある．実際に使用する場合は，次のようにして Python 処理系にライブラリを読み込む．

```
import cv2
```

この後，cv2 クラスの各種メソッドが使用できる．

Python における OpenCV モジュールの利用は，別のモジュール NumPy を前提としており，画像のフレームは NumPy の ndarray クラス（多次元配列）のオブジェクトとして扱われる．

【サンプルプログラム】

サンプルプログラム opencv01.py を次に示す．

プログラム：opencv01.py

```
1  # coding: utf-8
2  # モジュールのインポート
3  import cv2
4
5  # 動画像入力の開始
6  cap0 = cv2.VideoCapture(0)
7  # フレームレートの取得
8  fps = cap0.get(cv2.CAP_PROP_FPS)
9  # フレームサイズの取得
10 w = cap0.get(cv2.CAP_PROP_FRAME_WIDTH)
11 h = cap0.get(cv2.CAP_PROP_FRAME_HEIGHT)
12 print(fps,'(fps)'); print(w,'*',h)
13
14 # キャプチャと表示
15 while True:
16     # フレームの読取り
17     (ret,frame) = cap0.read()
18     if ret:
19         # フレームの表示
20         cv2.imshow('Camera: 0',frame)
21         # キーボードの読取り
22         k = cv2.waitKey(1)
23         if k == 27:     # ESCなら終了
24             break
25         elif k == 67 or k == 99:    # 'C' 'c' なら静止画保存
26             # JPEGのQualityは 0 - 100 : 大きいほど高画質
27             # cv2.imwrite('capture.jpg',frame,[cv2.IMWRITE_JPEG_QUALITY,100])
28             # PNGのQualityは 0 - 9 : 小さい程高画質
29             cv2.imwrite('capture.png',frame,[cv2.IMWRITE_PNG_COMPRESSION,3])
30     else:
31         print('Camera is not ready.')
32         break
33
34 # 終了処理
35 cap0.release()           # 動画像入力の開放
36 cv2.destroyAllWindows() # ウィンドウの消去
```

このプログラムは，カメラから画像フレームを取得し，それをウィンドウに表示する処理の繰り返しで動画像をリアルタイムに表示している．また，毎回の繰り返し処理の中でキーボードの値を取り込み，エスケープボタンが押されたら繰り返し処理を中断する形となっている．

1.1.1 動画像の入力

動画像の入力源はシステムに接続されたカメラもしくは動画ファイルであり，画像フレームは VideoCapture クラスのオブジェクトを介して取得する．動画像の入力処理に先立って，入力元を指定して VideoCapture オブジェクトを生成しておく．

＜ VideoCapture のコンストラクタ＞

1) VideoCapture(カメラ番号)

カメラ番号は 0 から開始する整数であり，システムで最初に認識されるカメラは 0 である．

2) VideoCapture(動画ファイルのパス)

引数には動画ファイルのパスを文字列として与える．

VideoCapture オブジェクトに対して get メソッドを使用して各種の情報を取得することができる．書き方は

VideoCapture オブジェクト.get(属性番号)

属性番号は cv2 のプロパティとして表 1 のように定義されている．

表 1: VideoCapture オブジェクトから得られる値（一部）

属性番号	値
`cv2.CAP_PROP_FPS`	フレームレート (fps)
`cv2.CAP_PROP_FRAME_WIDTH`	フレーム幅
`cv2.CAP_PROP_FRAME_HEIGHT`	フレーム高さ

動画像の入力処理を終了する場合は，VideoCapture オブジェクトに対して release メソッドを使用する．

動画像の入力は VideoCapture オブジェクトから read メソッドを使用して 1 フレームずつ取り出す．

＜フレームのキャプチャ＞

書き方： VideoCapture オブジェクト.read()

メソッド実行後は **(実行結果, フレーム)** のタプルが返される．実行結果は真理値であり，キャプチャ成功の場合は True，失敗の場合は False となる．得られるフレームは NumPy の ndarray オブジェクトである．

1.1.2 ユーザインターフェース

cv2 のメソッド imshow を使用して，read メソッドで読み取った画像フレームを表示することができる．

＜ imshow メソッドによるフレームの表示＞

書き方： cv2.imshow(ウィンドウタイトル, フレーム)

cv2 のメソッド waitKey を使用して，その時点でのキーボードの値を取得することができる．

< waitKey メソッドによるキーボードの値の取得>
書き方：　cv2.waitKey(待ち時間)
待ち時間の単位は ms である．その時点で押されているキーの値（コード）が返される．何も押されていない場合は 255 が返される．

ユーザインターフェースの使用を終了する場合は，cv2 の destroyAllWindows メソッドを使用する．

1.1.3　フレームのファイルへの保存

cv2 の imwrite メソッドを使用することでフレームを静止画としてファイルに保存することができる．

< imwrite メソッドによるフレームのファイル保存>
書き方：　cv2.imwrite(ファイル名, フレーム, 圧縮指定)
ファイル名は拡張子を付けた文字列で指定する．特に拡張子が `.jpg`，`.png` の場合は，それぞれ JPEG,PNG フォーマットとして圧縮保存ができる．圧縮指定は次の通り．
JPEG:　`[cv2.IMWRITE_JPEG_QUALITY`, 値`]`
値は 0〜100 までの整数値で，値が大きい方が高画質（ファイルサイズ大）である．
圧縮指定を省略すると 95 が暗黙値となる．
PNG:　`[cv2.IMWRITE_PNG_COMPRESSION`, 値`]`
値は 0〜9 までの整数値で，値が小さい方が高画質（ファイルサイズ大）である．
圧縮指定を省略すると 3 が暗黙値となる．

1.1.4　静止画像の読込み

cv2 の imread メソッドを使用することで，画像ファイルを読み込むことができる．

< imread メソッドによる画像ファイルの読込み>
書き方：　cv2.imread(ファイル名, 読込みフラグ)
ファイル名は拡張子を付けた文字列で指定する．読み込みフラグの意味は次の通り．

`cv2.IMREAD_UNCHANGED`　：　画像データをそのまま読み込む（変更なし）
`cv2.IMREAD_COLOR`　：　アルファチャンネル（不透明度の指定）を無視する（デフォルト）
`cv2.IMREAD_GRAYSCALE`　：　グレースケールに変換して読み込む

読み取った画像をフレームオブジェクト (ndarray) として返す．

静止画像を読み込んで表示するプログラム opencv02.py を次に示す．

プログラム：opencv02.py

```
1  # coding: utf-8
2
3  # モジュールのインポート
4  import cv2
5
6  # 画像ファイルの読込み
7  frame = cv2.imread('Earth.jpg',cv2.IMREAD_UNCHANGED)     # そのまま
8  #frame = cv2.imread('Earth.jpg',cv2.IMREAD_COLOR )       # αチャンネル無視
9  #frame = cv2.imread('Earth.jpg',cv2.IMREAD_GRAYSCALE )   # グレースケールに変換
10
11 # リサイズ
```

```
12 fr2 = cv2.resize(frame,(640,640))
13
14 # 表示
15 cv2.imshow('Camera: 0',fr2)
16
17 # 待ち
18 while True:
19         # キーボードの読取り
20         k = cv2.waitKey(1)
21         if k == 27:     # ESCなら終了
22             break
23
24 # 終了処理
25 cv2.destroyAllWindows() # ウィンドウの消去
```

このプログラムの 12 行目にあるように，cv2 の resize メソッドを使用することで画像サイズの拡縮ができる．

> **＜ resize メソッドによる画像の拡縮＞**
> **書き方：　cv2.resize(フレーム, (幅, 高さ))**
> 　引数に与えたフレームを（幅, 高さ）にリサイズしたフレームを返す．

■ フレームのサイズの取得

フレームオブジェクトのプロパティ shape の第 0 番目の要素にはフレームの高さが，第 1 番目の要素にはフレームの横幅が格納されている．

例. フレーム im のサイズの取得

```
>>> im.shape[1], im.shape[0]   Enter   ←サイズの取得
(199, 67)    ← 199 × 67 のサイズが得られた
```

1.1.5　色の分解と合成

フレームの色の分解と合成には cv2 のメソッド split, merge を使用する．

> **＜色の分解と合成＞**
> **分解：**　cv2.split(**フレーム**)
> 　引数に与えたフレームの色を分解し，**(赤フレーム, 緑フレーム, 青フレーム)** のタプルを返す．
> **合成：**　cv2.merge(**3 色のフレームから成るタプル**)
> 　合成されたフレームを返す．

画像を読み込んで，分解と再合成をするプログラム opencv03.py を次に示す．

プログラム：opencv03.py

```
1  # coding: utf-8
2
3  # モジュールのインポート
4  import cv2
5
6  # 画像ファイルの読込み
7  frame = cv2.imread('Earth.jpg',cv2.IMREAD_UNCHANGED)
8
9  # リサイズ
10 f = cv2.resize(frame,(320,320))
11
12 # 色分解
13 (f_r, f_g, f_b) = cv2.split(f)
```

```
14
15 # 表示
16 cv2.imshow('Red',f_r)        # 赤の成分
17 cv2.imshow('Green',f_g)      # 緑の成分
18 cv2.imshow('Blue',f_b)       # 青の成分
19
20 # 再度合成
21 f_rgb = cv2.merge( (f_r, f_g, f_b) )
22 # 表示
23 cv2.imshow("All",f_rgb)
24
25 # 待ち
26 while True:
27         # キーボードの読取り
28         k = cv2.waitKey(1)
29         if k == 27:      # ESCなら終了
30             break
31
32 # 終了処理
33 cv2.destroyAllWindows() # ウィンドウの消去
```

1.1.6 画像の類似性の検査

OpenCV は画像認識のための各種の機能を提供している．ここでは，2 つの画像の類似の度合いを算出する方法の 1 つを紹介する．与えられた 2 つの画像を比較するには，それら画像を予め**特徴データ**に変換し，それぞれの特徴データを比較するという手法を取る．ここでは，画像から **AKAZE 特徴量**[1] を生成して，それらの距離を比較する方法を取り上げて説明する．

【2 つの画像の類似度を算出する手順】

1. 与えられた 2 つの画像それぞれの AKAZE 特徴量を算出する．これにより得られた特徴データは**特徴点**のデータの集合を含む．
2. 上で得られた特徴データから，2 つの画像の特徴点を**総当りマッチング**（Brute-Force Matching）の手法で照合してそれぞれの特徴点の**距離**を求める．
3. 上で得られた特徴点の距離の平均値を求めて，それが小さい程 2 つの画像の類似度は高いと判断する．

1.1.6.1 AKAZE 特徴量の算出

AKAZE 特徴量を算出するには，cv2 の AKAZE_create メソッドを使用してある種の「特徴検出器」（detector）を生成し，それを用いて画像の特徴量を算出する．

＜ detector の生成＞
書き方： cv2.AKAZE_create()
この結果 detector が生成され，それが返される．
実行例： detector = cv2.AKAZE_create()
この結果，検出器 detector が生成される．

得られた detector に対して detectAndCompute メソッドを使用することで画像の特徴量を算出する．

[1]AKAZE 特徴量の算出手法は，KAZE 特徴量のアルゴリズムを更に改善したものである．詳しくは研究者のサイト http://www.robesafe.com/personal/pablo.alcantarilla/を参照のこと．

＜画像の特徴量の算出＞

書き方：　**検出器**.detectAndCompute(**フレーム**, None)

　この結果，**(特徴点データ, 特徴量データ)** のタプルが返される．

実行例：　(kp, des) = detector.detectAndCompute(f1, None)

　この結果，フレーム f1 から特徴点データ kp と特徴量データ des が得られる．

1.1.6.2　特徴量データの照合

得られた特徴量データから 2 つの画像の「距離」のデータを算出するには，2 つの特徴量データを照合するある種の照合器（matcher）が必要となる．照合器を生成するには cv2 のメソッド BFMatcher を使用する．

＜ matcher の生成＞

書き方：　cv2.BFMatcher(cv2.NORM_HAMMING)

　この結果，照合器が生成され返される．

実行例：　bfm = cv2.BFMatcher(cv2.NORM_HAMMING)

得られた照合器に対して match メソッドを使用すると，与えた 2 つの画像の特徴量データから，各特徴点の比較結果のリストが生成される．このリストの要素の distance プロパティに特徴点の距離が保持されている．

1.1.6.3　サンプルプログラム

用意したテンプレート画像（パターン画像）と他の画像との類似の度合いを算出するサンプルプログラムを挙げ，実行結果について考える．テンプレート画像として用意したのは図 1 の画像 (a),(b) である．

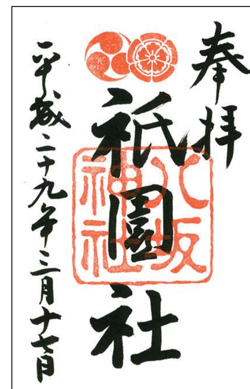
(a) ptn_yasaka1.jpg

(b) ptn_yasaka2.jpg

図 1: パターン画像

これと，図 2 の画像 (c),(d) [2] を比較する処理を実現する．

(c) Earth_small.jpg



(d) yasaka.jpg

図 2: 認識に使用する画像

[2]サンプルに用いた画像は，京都の八坂神社（http://www.yasaka-jinja.or.jp/）の御朱印と地球の画像である．

サンプルプログラム opencv04.py を次に示す.

プログラム：opencv04.py

```
# coding: utf-8
# AKAZEアルゴリズムによる類似度の算出

# モジュールのインポート
import cv2

# 画像ファイルの読込み(1)
f1 = cv2.imread('ptn_yasaka1.jpg',cv2.IMREAD_GRAYSCALE)
#f1 = cv2.imread('ptn_yasaka2.jpg',cv2.IMREAD_GRAYSCALE)

# 画像ファイルの読込み(2)
frame = cv2.imread('yasaka.jpg',cv2.IMREAD_COLOR)
#frame = cv2.imread('Earth_small.jpg',cv2.IMREAD_COLOR)
# 色分解
(f2, f2_g, f2_b) = cv2.split(frame)

# 特徴検出器の準備(1)
detector = cv2.AKAZE_create()
(target_kp, target_des) = detector.detectAndCompute(f1, None)

# 特徴検出器の準備(2)
(comp_kp, comp_des) = detector.detectAndCompute(f2, None)

# Brute-Force matcherの準備
bfm = cv2.BFMatcher(cv2.NORM_HAMMING)

# 特徴がマッチしたポイントのリスト
match = bfm.match(target_des, comp_des)
# 特徴点の距離の平均
dist = [m.distance for m in match]
result = sum(dist)/len(dist)
print( 'Result:',result )

# 表示
kpim = cv2.drawKeypoints(f1, target_kp, None)
cv2.imshow("f2",kpim)

# 待ち
while True:
        # キーボードの読取り
        k = cv2.waitKey(1)
        if k == 27:     # ESCなら終了
            break

# 終了処理
cv2.destroyAllWindows() # ウィンドウの消去
```

使用する画像を変えて実行した結果を表 2 に示す.

表 2: 実行結果の類似度（平均値）の比較

比較の組み合わせ	類似度（平均値）
(a) と (c) を比較	128.4059829059829
(b) と (c) を比較	139.23362445414847
(a) と (d) を比較	112.66239316239316
(b) と (d) を比較	112.69868995633188

この結果から，地球の画像 (c) よりも御朱印の画像 (d) の方がテンプレート画像により近い（類似している）と結論できる.

1.2 Pillow

Pillow は Python で静止画像を処理するためのモジュールである．本書では Pillow に関する導入的な内容について説明する．更に詳しい事柄に関しては，インターネットサイト https://pillow.readthedocs.io/ をはじめとする情報源を参照すること．

このモジュールの使用に先立って，必要なソフトウェアを Python 処理系にインストールしておく必要がある．実際に使用する場合は，次のようにして必要なサブモジュールを Python 処理系に読み込む．

`from PIL import` **Pillow のサブモジュール**

1.2.1 画像ファイルの読込みと保存

画像ファイルを読み込むには，Image モジュールのメソッド open を使用する．

例. 画像ファイル test01.jpg の読込み．

```
from PIL import Image                  ← Image モジュールの読込み
im = Image.open('test01.jpt','r')      ←画像ファイルの読み込み
```

＜画像ファイルの読込み＞
書き方： `Image.open(`**ファイル名，モード**`)`
モードの指定は省略可能であるが，指定するばあいは 'r' を与える．この結果，画像データが Image オブジェクトとして返される．ファイル名（パス名）は拡張子を伴う文字列で与える．
Pillow のバージョンが 4.1 の場合に読み書きがサポートされている画像フォーマットは次の通りである．

BMP	EPS	GIF
ICNS	ICO	IM
JPEG	JPEG 2000	MSP
PCX	PNG	PPM
SGI	SPIDER	TIFF
WebP	XBM	

Image オブジェクトに対して各種の編集や処理を行うことができる．ファイルから読み込んだ Image オブジェクトには size，format，mode， といったプロパティがあり，それぞれ画素サイズ（横縦各成分のタプル），画像フォーマット，画像モードの情報が保持されている．また info プロパティには辞書型オブジェクトとして各種情報が保持されている．

画像の各種プロパティの調査

```
>>> from PIL import Image  Enter          ←パッケージの読込み
>>> im = Image.open('Earth.jpg')  Enter   ←画像ファイルの読込み
>>> im.size  Enter                        ←画素サイズの調査
(2048, 2048)                              ←画素サイズ
>>> im.format  Enter                      ←画像フォーマットの調査
'JPEG'                                    ←画像フォーマット
>>> im.mode  Enter                        ←画像モードの調査
'RGB'                                     ←画像モード
>>> im.info['dpi']  Enter                 ←画像解像度の調査
(300.0, 300.0)                            ←画像解像度
```

画像モードは色成分の構成を意味するもので，表 3 のような種類がある．

表 3: 画像モード

画像モード	色成分の構成	画像モード	色成分の構成
'1'	白黒 2 値	'L'	8 ビットグレースケール
'RGB'	24 ビットカラー	'RGBA'	24 ビットカラー＋ α 値（8 ビット）
'CMYK'	減色混合カラー（32 ビット）	'HSV'	HSV 色空間表現によるカラー（24 ビット）

※ 各色の成分のことを**バンド**（band）と呼ぶ

info プロパティの中には 'dpi' というキーワードがあり，対象画像の解像度の値が保持されている．

Image オブジェクトは save メソッドを使用することでファイルに保存することができる．

＜画像のファイルへの保存＞
書き方：　Image オブジェクト.save(**ファイル名, オプション**)
この結果 Image オブジェクトがファイルに保存される．ファイル名（パス名）は拡張子を伴う文字列で与える．

本書では特に圧縮形式のフォーマットとして JPEG, PNG の利用頻度が高いと考え，これらのフォーマットで保存する場合のオプションについて説明する．

表 4: 画像ファイルを保存する際のオプション（一部）

フォーマット	キーワード引数	意味
JPEG	quality=**整数値**	画質の指定．1〜95 の値で値が大きいほど高画質．デフォルトは 75
	dpi=**(x,y)**	横方向，縦方向それぞれの解像度
PNG	compress_level=**整数値**	圧縮率の指定．0〜9 の値で値が小さいほど高画質．デフォルトは 6
	dpi=**整数値**	画像の解像度

1.2.2　Image オブジェクトの新規作成

Image モジュールの new メソッドを使用することで，Image オブジェクトを新規に作成することができる．

＜ Image オブジェクトの作成＞
書き方：　Image.new(**モード, サイズ, 初期ピクセル値**)
初期ピクセル値：　生成した直後に全ての画素に与える初期値
モードが '1' の場合は 0（黒）か 1（白），'L' の場合は 0（黒）〜255（白），その他のモードでは各成分 0〜255 のタプル．

新規に作成した Image オブジェクト上に描画したり，他の Image オブジェクトを配置（複写）することができる．

1.2.3　画像閲覧

Image オブジェクトに対して show メソッドを使用すると，OS に設定されている画像ビューワが起動して当該オブジェクトの内容を表示することができる．

例. Image オブジェクト im の表示

```
im.show()
```

この結果，画像ビューワが起動して im の内容が表示される．画像ビューワが開いている間は show メソッド実行部分で当該スレッドがブロックし，画像ビューワが終了するのを待つ．

このメソッドを使用するには，Python 処理系を実行する OS において，使用する画像ビューワの設定をしておく必要がある．

1.2.4 画像編集

ここでは，Image オブジェクトを加工する方法について説明する．

1.2.4.1 画像の拡大と縮小

resize メソッドを使用することで Image オブジェクトの画素サイズを変更することができる．

例. Image オブジェクトの画素サイズ変更

```
im2 = im1.resize( (300,300) )
```

これは Image オブジェクト im1 を 300 × 300 の画素サイズにして，それを im2 としている例である．

画像の画素サイズを変更すると，画素が乱れて画質が劣化することが多いが resize メソッドを実行する際に，画質の劣化を軽減するための各種の**フィルタ**（表 5）を指定することができる．

リサイズにおけるフィルタ指定の例

```
im2 = im1.resize( (300,300), resample=Image.LANCZOS )
```

表 5: 各種フィルタ

フィルタ	効果
Image.NEAREST	画素の補間に近傍の画素を使用する（デフォルト）
Image.BOX	画素の補間に**ボックス近似**を使用する
Image.BILINEAR	画素の補間に**線形近似**を使用する
Image.HAMMING	線形近似より若干鮮明な補間処理
Image.BICUBIC	画素の補間に **3 次補間**を使用する
Image.LANCZOS	Lanczos フィルタによる補間処理

フィルタ選択の判断は扱う画像によって異なるので実際に実行して目視確認するのが良い．

参考）画素サイズ変更には thumbnail メソッドも使用できる．thumbnail メソッドは対象オブジェクトそのものを変更する．

1.2.4.2 画像の部分の取り出し

crop メソッドを使用すると，Image オブジェクトから矩形領域を取り出すことができる．Image オブジェクト im から部分を取り出す場合，取り出したい矩形領域の**左上の座標**を (U_x, U_y)，**右下の座標**を (L_x, L_y) とするとき次のようにする．

```
im2 = im.crop( (Ux,Uy,Lx,Ly) )
```

この結果，指定した矩形領域が Image オブジェクト im2 として得られる．

1.2.4.3 画像の複製

Image オブジェクトの複製を作成するには copy メソッドを使用する．

複製の例. Image オブジェクト im の複製 im2 を作成する

```
im2 = im.copy()
```

1.2.4.4 画像の貼り付け

Image オブジェクトの上に別の Image オブジェクトを貼り付ける（複写する）には paste メソッドを使用する．貼り付けられる Image オブジェクトを im1, 貼り付ける Image オブジェクトを im2 とする場合，im1 上の貼り付ける位置を (P_x, P_y) とすると，次のように記述して実行する．

書き方： `im1.paste(im2,(Px,Py))`

この結果，im1 上の (P_x, P_y) の位置に im2 の内容が貼り付けられる．(im1 自体が変更される）

1.2.4.5 画像の回転

Image オブジェクトを回転させたものを得るには rotate メソッドを使用する．

書き方： **Image オブジェクト**.rotate(**角度**)

引数に与える角度の単位は「度」である．この処理によって，回転された結果の Image オブジェクトが返される．

回転の結果，元の画像が Image オブジェクトの画素サイズに収まらずに切れてしまうことがある．その場合は rotate メソッドの引数にキーワード引数 expand=True を与えると，回転後に画像が収まるように結果の Image オブジェクトの画素サイズが拡大される．また，回転処理によって画素が乱れることがあるが，キーワード引数 resample=**フィルタ** を与えることで乱れを軽減することができる．フィルタは基本的には表 5 に挙げたものが指定できる．(注：使用できないものもある）

rotate メソッドの他に，Image オブジェクトの 90 度単位の回転や，上下左右の反転を実行する transpose メソッドがある．

書き方： **Image オブジェクト**.transpose(**手法**)

引数に指定した手法に従って，Image オブジェクトを回転もしくは反転したオブジェクトを返す．指定できる手法（method）を表 6 に示す．

表 6: 回転・反転の手法

method	処理
`Image.FLIP_LEFT_RIGHT`	左右反転
`Image.FLIP_TOP_BOTTOM`	上下反転
`Image.ROTATE_90`	反時計回り 90 度回転
`Image.ROTATE_180`	180 度回転
`Image.ROTATE_270`	時計回り 90 度回転

1.2.5 画像処理

ここでは，Image オブジェクトのピクセル値を処理（画像補正をはじめとする処理）する方法について説明する．

1.2.5.1 色の分解と合成

Image オブジェクトの色成分（バンド）を分解して，別々の Image オブジェクトとして取り出すには split メソッドを使用する．

書き方： **Image オブジェクト**.split()

これにより，各色成分に分けられた Image オブジェクトのタプルが返される．例えばモードが 'RGB' である Image

オブジェクト im があるとき，

```
(r,g,b) = im.split()
```

とすると，r，g，b にそれぞれ赤，緑，青に分解された Image オブジェクトが得られる．これら Image オブジェクトの画像モードはグレースケールすなわち 'L' である．

split メソッドとは逆に，グレースケールの Image オブジェクトからカラーの Image オブジェクトを合成するには merge メソッドを使用する．

書き方： Image.merge(**画像モード, 画像のタプル**)

画像モードは表 3 のものを指定する．画像のタプルは画像モードの各色成分（バンド）とするグレースケールの Image オブジェクトを並べたのもである．

例. グレースケールの Image オブジェクト r, g, b の合成

```
im2 = Image.merge( 'RGB', (r,g,b) )
```

これにより，カラーのイメージオブジェクト im2 が得られる．

1.2.6 描画

Pillow の ImageDraw サブパッケージを使用すると，Image オブジェクトの上に図形や文字を描画することができる．描画機能を使用するには次のようにしてパッケージを読み込む．

```
from PIL import ImageDraw
```

【描画の考え方】

Image オブジェクトに描画するには，対象の Image オブジェクトから取得した Draw オブジェクトに対して各種の描画メソッドを実行する．

例. Image オブジェクト im から Draw オブジェクト drw を取得する

```
from PIL import ImageDraw
drw = ImageDraw.Draw(im)
```

以後，この drw オブジェクトに対して各種描画メソッドを実行すると im 上に描画される．

【描画処理の例】直線の描画

新規に作成した Image オブジェクトに直線を描画するプログラム pillow01.py を示す．

プログラム：pillow01.py

```
 1  # coding: utf-8
 2  #モジュールの読込み
 3  from PIL import Image
 4  from PIL import ImageDraw
 5
 6  # 白いImageオブジェクトの生成
 7  im = Image.new( 'RGB', (640,480), (255,255,255) )
 8  # Drawオブジェクトの取得
 9  drw = ImageDraw.Draw(im)
10
11  # 楕円の描画
12  drw.ellipse( [100,100,539,379], fill=(0,0,255) )
13  # 直線の描画
14  drw.line( [0,0,639,479], fill=(255,0,0), width=32 )
15
16  im.show()   # ビューワによる表示
```

解説

7 行目で，白に初期化された Image オブジェクト im を作成し，9 行目で im から Draw オブジェクト drw を取得している．12 行目で drw に対して楕円を，14 行目で直線を描画している．楕円の描画には ellipse メソッド，直線の描画には line メソッドを使用しており，引数に座標リストと，各種のキーワード引数を与えている．キーワード引数の 'fill=' には色の成分をタプルで与える．また 'width=' には線の太さをピクセル数で与える．

これにより，im 上に楕円と直線が描かれる．このプログラムを実行すると図 3 のような画像が表示される．

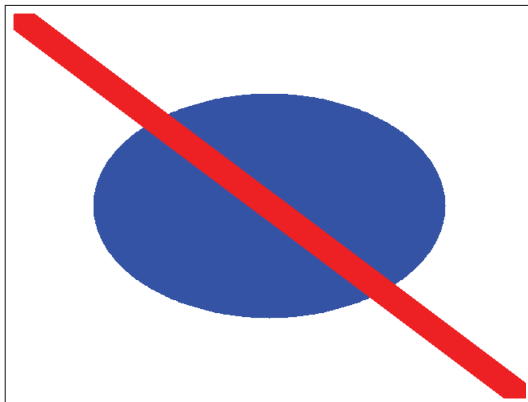


図 3: 実行結果

【描画メソッド】

Pillow で使用できる描画メソッドの一部を紹介（表 7）する．

表 7: Draw オブジェクトに対する描画メソッド（一部）

メソッド	働き
point(座標リスト, fill=色)	点（複数）を描画する
line(座標リスト, fill=色, width=太さ)	折れ線を描画する
ellipse(座標リスト, fill=色)	楕円を描画する
arc(座標リスト, 開始角, 終了角, fill=色)	円弧を描画する
pieslice(座標リスト, 開始角, 終了角, fill=色)	パイの形を描画する
rectangle(座標リスト, fill=色)	長方形を描画する
polygon(座標リスト, fill=色)	ポリゴン（多角形）を描画する

1.2.7 アニメーション GIF の作成

Image オブジェクトに対する save メソッドに，アニメーション GIF として保存する機能がある．save メソッドの第 1 引数にファイル名を指定する際，拡張子として '.gif' を付けると GIF 形式で保存される．更に save メソッドのキーワード引数として

save_all=True, append_images=**Image オブジェクトのリスト**, duration=**フレームの表示時間**

といったものを指定するとアニメーション GIF として保存される．

例． Image オブジェクト im0, im1, im2, ⋯ , imN をアニメーション GIf として保存する手順

```
imglist = [ im1, im2, ⋯ , imN ]      ←Image オブジェクト群 im1, im2, ⋯ , imN のリストを作成
im0.save('anim.gif', save_all=True, append_images=imglist, duration=1000 )   ←保存
```

これで，アニメーション GIF が 'anim.gif' として保存される．**フレームの表示時間**にはミリ秒単位の整数値を指定する．繰り返し表示するアニメーションを作成するには，save メソッドにキーワード引数 loop=True を指定する．

2　GUI とマルチメディア

2.1　pygame

pygame はウィンドウ上の描画機能や UI デバイス（キーボード，ゲーム用コントロールパッドなど）からの入力をハンドリングする機能，マルチメディア再生の機能を提供するモジュールであり，SDL [3] を用いて構築されたライブラリである．pygame はゲームプログラムを構築するために便利な機能を提供するが，ウィンドウ描画と UI デバイスのハンドリングを実現するための高速でコンパクトな汎用ライブラリと見るべきである．本書では pygame の基本的な使用方法について解説する．pygame に関する情報はインターネットサイト https://www.pygame.org/ を参照のこと．

pygame の使用に先立って，次のようにして必要なモジュールを読み込んでおく必要がある．

```
import pygame
```

また，pygame 配下の各種モジュールを読み込むことが必要になることがある．

例. 終了イベントを意味する定数 QUIT を読み込む

```
from pygame.locals import QUIT
```

2.1.1　基礎事項

pygame の機能を使用するには，最初に init 関数を呼び出して初期化処理をする必要がある．

初期化の例.

```
pygame.init()
```

2.1.1.1　Surface オブジェクト

pygame では，画像データなどのピットマップを **Surface オブジェクト**として扱う．アプリケーションウィンドウも Surface オブジェクトとして生成し，その上に図形や文字を描いたり，画像などの Surface オブジェクトを貼り込む形で描画を実現する．

アプリケーションウィンドウの Surface オブジェクトは次のようにして生成する．

例.

```
sf = pygame.display.set_mode( (400,300) )
```

この例では横 400 ドット，縦 300 ドットのサイズのウィンドウが生成され，Surface オブジェクト sf として扱われる．この後，この sf 上に描画したい別の Surface オブジェクトを貼り付けたり，各種の描画メソッドで図形や文字を描く．

pygame の座標系は，多くの GUI ライブラリと共通である．すなわち，左上を原点として，横方向を X 軸（右に行くほど座標値は大），縦方向を Y 軸（下に行くほど座標値は大）とする．

2.1.1.2　アプリケーションの実行ループ

pygame を用いたアプリケーションの基本的な動作は，

1)　描画処理
アプリケーションウィンドウの Surface オブジェクトに対する描画処理である．

2)　イベントハンドリング
イベントキュー [4] からイベントを取り出し，対応する処理を実行する．

3)　ディスプレイの更新

を繰り返すループである．

---

[3]SDL (Simple DirectMedia Layer) はクロスプラットフォームのマルチメディア用 API であり，グラフィックの描画やサウンドの再生などの機能を提供する．SDL は Windows(Microsoft), OSX(Apple), Linux, iOS(Apple), Android(Google) といった OS で利用できる．

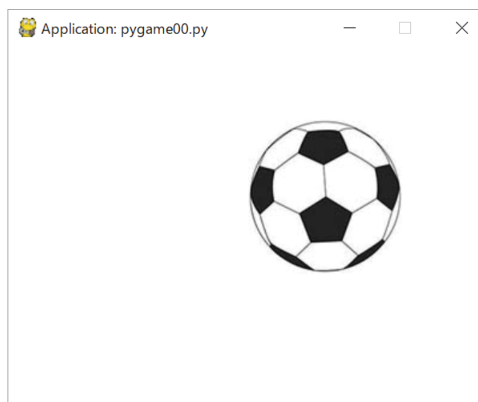
[4]イベントは短時間に多くのものが発生する．アプリケーションが受け取ったイベントは**イベントキュー**に蓄積される．

pygame は SDL を用いて構築されているため，描画とイベント処理の実行速度が大きい．このため，上記ループにおいてアプリケーション実行のタイミングを適切に制御しなければ，システムの CPU タイムの大きな部分を（不必要に）占有することになる．pygame には，アプリケーション実行のタイミングを制御するための Clock クラスが用意されており，このクラスのオブジェクトを用いて，アプリケーションウィンドウのフレームレートを制御して CPU タイムの不必要な要求を緩和することができる．具体的には次のようにして，フレームレート制御用のオブジェクトを生成する．

```
fps = pygame.time.Clock()
```

この例では fps に Clock オブジェクトが得られており，これに対してフレームレートの設定を行う．

サンプルプログラム pygame00.py を示しながら，pygame のアプリケーションの基本的な動作について説明する．このプログラムは，ボールの画像をウィンドウに表示して，一定のフレームレートでボールを移動するものである．ボールはウィンドウの端に衝突すると反射（バウンド）する．（図 4）



ボールがウィンドウ内でバウンドする．

図 4: pygame00.py を実行したところ

プログラム：pygame00.py

```
1  # coding: utf-8
2
3  # モジュールの読み込み
4  import sys
5  import pygame
6  from pygame.locals import QUIT
7
8  # pygameの初期化
9  pygame.init()
10
11 w = 400; h = 300
12 sf = pygame.display.set_mode( (w,h) )   # アプリケーションウィンドウ
13 pygame.display.set_caption('Application: pygame00.py')
14
15 fps = pygame.time.Clock()   # フレームレート制御のための Clock オブジェクト
16
17 # 画像の読み込み
18 im1 = pygame.image.load('ball01.jpg')
19 im1_w = im1.get_width()     # 画像の横幅の取得
20 im1_h = im1.get_height()    # 画像の高さの取得
21
22 # メインループ
23 x = 0; y = 0    # ボールの位置
24 dx = 3; dy = 2  # 移動量
25 while True:
26     sf.fill( (255,255,255) )    # 背景の色
27     sf.blit(im1, (x,y) )        # ボールの描画
28     # イベントキューを処理するループ
```

```
29        for ev in pygame.event.get():
30            if ev.type == QUIT:      # 「終了」イベント
31                pygame.quit()
32                print('quitting...')
33                sys.exit()
34        # ディスプレイの更新
35        pygame.display.update()
36        # フレームレートの設定
37        fps.tick(30)    # 30FPSに設定
38        # ボール移動（位置変更）の処理
39        x += dx; y += dy     # 移動
40        if x + im1_w > w:    # ボールが右端に衝突した場合の処理
41            x = w - im1_w - 1
42            dx *= -1
43        elif x < 0:          # ボールが左端に衝突した場合の処理
44            x = 0
45            dx *= -1
46        if y + im1_h > h:    # ボールが床に衝突した場合の処理
47            y = h - im1_h - 1
48            dy *= -1
49        elif y < 0:          # ボールが天井に衝突した場合の処理
50            y = 0
51            dy *= -1
```

プログラムの 18〜20 行目でボールの画像 ball01.jpg を load メソッドで読み込んで，その幅と高さを取得（get_width メソッド，get_height メソッド）している．

26〜27 行目でボールの画像をアプリケーションウィンドウに表示し，29〜33 行目でイベントハンドリングを行っている．ここでは QUIT イベント（アプリケーションウィンドウの閉じるボタンをクリックしたときに発生）を検出してアプリケーションの終了処理をハンドリングするのみとしている．アプリケーションには短時間に複数のイベントが発生し，それらは**イベントキュー**と呼ばれる待ち行列に蓄積される．29 行目の for 文は，イベントキューからイベントを 1 つずつ取り出し，それを順番にハンドリングするループとなっている．イベントキューのイベント列を取り出すには pygame.event.get() を実行する．イベントキューから取り出された個々のイベントはイベント種別をはじめとする各種のプロパティが保持されている．イベント種別は pygame.locals の中に定数として定義されており，プログラムの 6 行目にあるような形で読み込んで使用するのが一般的である．

35 行目では update メソッドを用いてウィンドウの更新を行い，37 行目では tick メソッドを用いてアプリケーションウィンドウのフレームレートを設定している．(引数にフレームレートを与える）

画像の描画は 27 行目にあるように blit メソッドを用いる．(詳しくは後述)

13 行目の set_caption メソッドはウィンドウのタイトルを設定する．

30 行目にあるように，QUIT イベントを受けてこのアプリケーションは終了する．このイベントはアプリケーションウィンドウの閉じるボタンをクリックしたときに発生する．この際の pygame の終了処理として quit メソッドを実行する．この後，sys.exit() を呼び出してアプリケーションを終了させている．

2.1.2 描画機能

Surface オブジェクトに各種の図形や文字などを表示する方法を説明する．

■ 四角形

塗り潰し： pygame.draw.rect(Sf, Color, Rect)

外周　　： pygame.draw.rect(Sf, Color, Rect, Width)

Surface オブジェクト Sf に対して四角形を描画する．引数に与えるものは次の通り．

Color - (R,G,B) のタプル．値の範囲は 0〜255 の整数値
Rect - 描画位置 (X,Y) とサイズ W × H の値から成るタプル (X,Y,W,H)
Width - 外周を描く場合の線の太さ

■ **楕円**

塗り潰し： pygame.draw.ellipse(Sf, Color, Rect)

外周　　： pygame.draw.ellipse(Sf, Color, Rect, Width)

Surface オブジェクト Sf に対して楕円を描画する．楕円の位置とサイズは，その楕円に外接する四角形を元に考える．引数に与えるものは次の通り．

Color - (R,G,B) のタプル．値の範囲は 0〜255 の整数値
Rect - 描画位置 (X,Y) とサイズ W × H の値から成るタプル (X,Y,W,H)
Width - 外周を描く場合の線の太さ

■ **円**

塗り潰し： pygame.draw.circle(Sf, Color, (X,Y), R)

外周　　： pygame.draw.circle(Sf, Color, (X,Y), R, Width)

Surface オブジェクト Sf に対して円を描画する．引数に与えるものは次の通り．

Color - (R,G,B) のタプル．値の範囲は 0〜255 の整数値
(X,Y) - 円の中心の座標
R - 円の半径
Width - 外周を描く場合の線の太さ

■ **線分**

書き方： pygame.draw.line(Sf, Color, (X1,Y1),(X2,Y2), Width)

Surface オブジェクト Sf に対して線分を描画する．引数に与えるものは次の通り．

Color - (R,G,B) のタプル．値の範囲は 0〜255 の整数値
(X1,Y1),(X2.Y2) - 始点と終点の座標
Width - 線の太さ

■ **折れ線**

書き方： pygame.draw.lines(Sf, Color, Closing, Plist, Width)

Surface オブジェクト Sf に対して折れ線を描画する．引数に与えるものは次の通り．

Color - (R,G,B) のタプル．値の範囲は 0〜255 の整数値
Closing - 始点と終点を結ぶか（True）否か（False）
Plist - 始点から終点までの座標のリスト．要素は各座標のタプル
Width - 線の太さ

■ **多角形**

塗り潰し： pygame.draw.polygon(Sf, Color, Plist)

外周　　： pygame.draw.polygon(Sf, Color, Plist, Width)

Surface オブジェクト Sf に対して多角形を描画する．引数に与えるものは次の通り．

Color - (R,G,B) のタプル．値の範囲は 0〜255 の整数値
Plist - 頂点の座標のリスト．要素は各座標のタプル
Width - 外周を描く場合の線の太さ

■ **画像**

ファイルから読込み　： pygame.image.load(Fname)

画像ファイルをパス Fname から読み込んで Surface オブジェクトとして返す．

ファイルに保存　　　： pygame.image.save(S, Fname)

Surface オブジェクト S を画像ファイル Fname（パス名）に保存する．

Surface オブジェクトを別の Surface オブジェクトに貼り付けるには blit メソッドを使用する．

例. Surface オブジェクト s1 を別の Surface オブジェクト s2 に貼り付ける

```
s2.blit(s1,(20,10))
```

この例では，Surface オブジェクト s1 を，s2 上の (20,10) の位置に貼り付けている．
アプリケーションウィンドウも Surface オブジェクトであり，画像を表示するには同様の方法を用いる．

■ 文字列

pygame では文字列は Surface オブジェクトとして表示する．この際，フォントとサイズ，スタイルなどの情報を保持する Font オブジェクトを生成し，これを用いて文字列データを Surface オブジェクトに変換する．Font オブジェクトを生成するには SysFont メソッドを使用する．

Font オブジェクトの生成： pygame.font.SysFont(**フォント名, サイズ**)

注） フォント名に None を指定すると自動的にデフォルトのフォントが採用されるが，その場合は日本語が使用できないことが多い．

Font オブジェクトを用いて文字列を Surface オブジェクトに変換するには render メソッドを用いる．

文字列から Surface を生成： Font **オブジェクト**.render(**文字列, アンチエイリアス指定, 色**)

「アンチエイリアス指定」は True か False で,「色」は RGB 値のタプルで与える．

例. 文字列から Surface オブジェクトを生成

```
fnt = pygame.font.SysFont('ipa ゴシック',32)
txt = fnt.render('日本語のメッセージ', True, (255,255,255) )
```

この結果，文字列をビットマップとして保持する Surface オブジェクト txt が生成される．

フォント名の取得：

pygame で利用できるフォント名は get_fonts メソッドを使用することで調べることができる．このメソッドを実行すると利用可能なフォント名のリストが返される．次に示すプログラム pygame03.py を実行すると，利用可能なフォント名の一覧が表示される．

プログラム：pygame03.py

```
1  # coding: utf-8
2
3  # モジュールの読み込み
4  import pygame
5
6  # pygameの初期化
7  pygame.init()
8
9  # フォントリストの取得と表示
10 fl = pygame.font.get_fonts()
11 for m in fl:
12     print(m)
```

このプログラムを実行すると，次の例のようにフォント名が表示される．

```
arial
arialblack
calibri
    ⋮
（途中省略）
    ⋮
ipap 明朝
ipaex ゴシック
ipaex 明朝
```

2.1.2.1 描画のサンプルプログラム

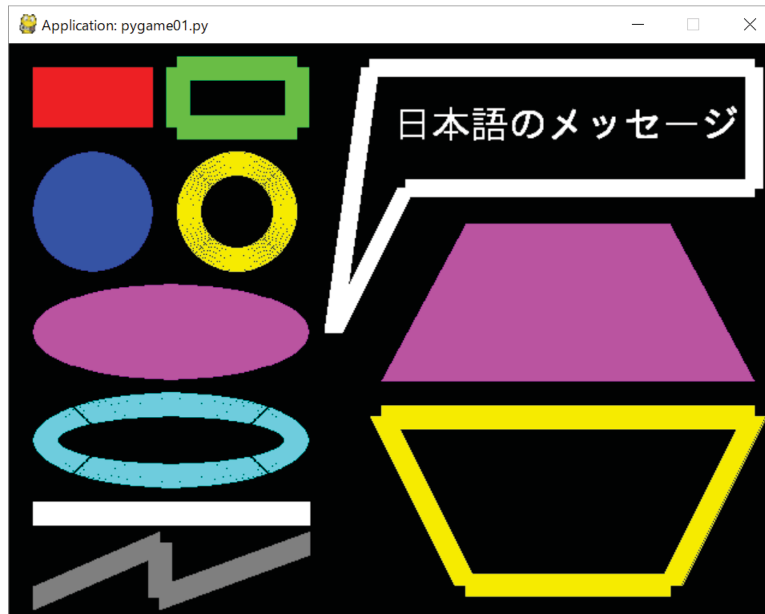
先に解説した描画機能を使用したサンプルプログラム pygame01.py を示す.

プログラム：pygame01.py

```
 1  # coding: utf-8
 2
 3  # モジュールの読み込み
 4  import sys
 5  import pygame
 6  from pygame.locals import QUIT
 7
 8  # pygameの初期化
 9  pygame.init()
10
11  sf = pygame.display.set_mode( (640,480) )   # アプリケーションウィンドウ
12  pygame.display.set_caption('Application: pygame01.py')
13
14  fps = pygame.time.Clock()   # フレームレート制御のための Clock オブジェクト
15
16  # メインループ
17  while True:
18      # 四角形（塗りつぶし）の描画
19      pygame.draw.rect(sf, (255,0,0), (20,20,100,50))
20      # 四角形（枠）の描画
21      pygame.draw.rect(sf, (0,255,0), (140,20,100,50), 20 )
22
23      # 円（塗りつぶし）の描画
24      pygame.draw.circle(sf, (0,0,255), (70,140), 50 )
25      # 円（線による円周）の描画
26      pygame.draw.circle(sf, (255,255,0), (190,140), 50, 20 )
27
28      # 楕円（塗りつぶし）の描画
29      pygame.draw.ellipse(sf, (255,0,255), (20,200,230,80) )
30      # 楕円（線による周）の描画
31      pygame.draw.ellipse(sf, (0,255,255), (20,290,230,80), 20 )
32
33      # 線分の描画
34      pygame.draw.line(sf, (255,255,255), (20,390),(250,390), 20 )
35
36      # 折れ線の描画（開放端）
37      plst = [(20,460), (125,415), (125,460), (250,415)]
38      pygame.draw.lines(sf, (127,127,127), False, plst, 20 )
39      # 折れ線の描画（始点と終点を結ぶ）
40      plst = [(270,240), (300,20), (620,20), (620,120), (330,120)]
41      pygame.draw.lines(sf, (255,255,255), True, plst, 15 )
42
43      # 文字の描画（システムフォント）
44      fnt = pygame.font.SysFont('ipaゴシック',32)     # システムフォント
45      txt = fnt.render('日本語のメッセージ', True, (255,255,255) )
46      txt_rct = txt.get_rect()        # テキストオブジェクトの領域サイズ
47      sf.blit(txt,(320,50))
48
49      # ポリゴンの描画（塗りつぶし）
50      plst = [(310,280), (380,150), (550,150), (620,280)]
51      pygame.draw.polygon(sf, (255,0,255), plst )
52
53      # ポリゴンの描画（線による周）
54      buf = pygame.Surface( (330,160) )   # バッファ
55      plst = [(10,10), (80,150), (250,150), (320,10)] # バッファ上の座標
56      pygame.draw.polygon(buf, (255,255,0), plst, 20 )    # バッファに描画
57      sf.blit( buf, (300,300) )   # バッファをウィンドウへ
58
59      # イベントキューを処理するループ
60      for ev in pygame.event.get():
61          if ev.type == QUIT:     # 「終了」イベント
62              pygame.quit()
63              print('quitting...')
64              sys.exit()
```

```
65
66      # ディスプレイの更新
67      pygame.display.update()
68      # フレームレートの設定
69      fps.tick(4)  # 4FPSに設定
```

このプログラムを実行すると図 5 のようなウィンドウが表示される．



各種の描画機能を実行したサンプル

図 5: pygame01.py を実行したところ

■ 回転，拡縮など

画像に対して回転，拡大，縮小をするには，元の画像（Surface オブジェクト）に対してそれらの処理を施したものを生成して，それを別の Surface オブジェクトに貼り付けるという手順を踏む．

回転，拡縮のためのメソッド

回転　　　：　pygame.transform.rotate(Sur,Angle)
サイズ変更：　pygame.transform.scale(Sur,NewSize)

これらメソッドは Surface オブジェクト Sur を回転，拡縮し，その結果として得られる Surface オブジェクトを返す．元の Sur は変更されない．回転処理において Angle には回転角を反時計回りで指定する．単位は 360 進法の「度」である．サイズ変更において，NewSize には幅と高さのタプル (W,H) で与える．要素は整数で与えること．

2.1.2.2　回転，拡縮のサンプルプログラム

画像の回転，拡縮を応用したアニメーションを表示するサンプルプログラムを示す．図 6 に示す画像を回転するアニメーションを表示するプログラムを pygame02.py に，拡縮するアニメーションを表示するプログラムを pygame04.py に示す．

pygame_logo.png

図 6: この画像を回転，拡縮する

プログラム：pygame02.py

```
1  # coding: utf-8
2
3  # モジュールの読み込み
4  import sys
5  import pygame
6  from pygame.locals import QUIT
7
8  # pygameの初期化
9  pygame.init()
10
11 sf = pygame.display.set_mode( (320,240) )   # アプリケーションウィンドウ
12 pygame.display.set_caption('Application: pygame02.py')
13
14 fps = pygame.time.Clock()   # フレームレート制御のための Clock オブジェクト
15
16 # 画像の読込み
17 im1 = pygame.image.load('pygame_logo.png')
18 agl = 0.0   # 初期角度
19
20 # メインループ
21 while True:
22     sf.fill( (0,0,0) )  # 毎フレームクリアする
23     # 画像の回転と表示
24     im2 = pygame.transform.rotate(im1,agl%360)  # 回転処理
25     sf.blit(im2, (50,10) )  # ウィンドウに転送
26     agl += 2.4  # 次回の角度
27
28     # イベントキューを処理するループ
29     for ev in pygame.event.get():
30         if ev.type == QUIT:     # 「終了」イベント
31             pygame.quit()
32             print('quitting...')
33             sys.exit()
34
35     # ディスプレイの更新
36     pygame.display.update()
37     # フレームレートの設定
38     fps.tick(30)    # 30FPSに設定
```

プログラム：pygame04.py

```
1  # coding: utf-8
2
3  # モジュールの読み込み
4  import sys
5  import pygame
6  from pygame.locals import QUIT
7
8  # pygameの初期化
9  pygame.init()
10
11 sf = pygame.display.set_mode( (320,120) )   # アプリケーションウィンドウ
12 pygame.display.set_caption('Application: pygame04.py')
13
14 fps = pygame.time.Clock()   # フレームレート制御のための Clock オブジェクト
15
16 # 画像の読込み
17 im1 = pygame.image.load('pygame_logo.png')
18 im1_w = im1.get_width()
19 im1_h = im1.get_height()
20 rat = 0.0   # 初期比率
21 dr = 0.02
22
23 # メインループ
24 while True:
25     sf.fill( (0,0,0) )  # 毎フレームクリアする
26     # 画像の回転と表示
```

```
27      im2 = pygame.transform.scale(im1,(int(rat*im1_w),int(rat*im1_h)) )  # 拡縮処理
28      sf.blit(im2, (10,10) )  # ウィンドウに転送
29      rat += dr   # 次回の比率
30      if rat > 1.5:
31          rat = 1.5
32          dr *= -1
33      elif rat < 0.0:
34          rat = 0.0
35          dr *= -1
36
37      # イベントキューを処理するループ
38      for ev in pygame.event.get():
39          if ev.type == QUIT:     # 「終了」イベント
40              pygame.quit()
41              print('quitting...')
42              sys.exit()
43
44      # ディスプレイの更新
45      pygame.display.update()
46      # フレームレートの設定
47      fps.tick(30)    # 30FPSに設定
```

■ **Surface オブジェクトのサイズ**

Surface オブジェクトに対して get_width，get_height メソッドを引数なしで使用することで幅と高さが得られる．

2.1.3　キーボードとマウスのハンドリング

キーボードやマウスのイベントとして代表的なもの（イベント種別）を表 8 に挙げる．これらイベントは pygame.locals の中に定数として定義されており，それらを読み込んで使用することができる．

表 8: マウスとキーボードの代表的なイベント

イベント定数	イベントの種類	利用できるイベント属性（プロパティ）の一部
MOUSEBUTTONDOWN	マウスのボタンが押された	pos,button：位置とボタン
MOUSEBUTTONUP	マウスのボタンが放された	pos,button：位置とボタン
MOUSEMOTION	マウスが動いた	pos,buttons：位置とボタンタプル
KEYDOWN	キーボードが押された	key,mod,unicode：キー番号とモディファイア，文字コード
KEYUP	キーボードが放された	key,mod：キー番号とモディファイア

キーボードとマウスのイベントをハンドリングするサンプルプログラム pygame05.py を示す．

プログラム：pygame05.py

```
1   # coding: utf-8
2
3   # モジュールの読み込み
4   import sys
5   import pygame
6   from pygame.locals import QUIT, MOUSEBUTTONDOWN, MOUSEBUTTONUP, \
7                            MOUSEMOTION, KEYDOWN, KEYUP
8
9   # pygameの初期化
10  pygame.init()
11
12  sf = pygame.display.set_mode( (320,240) )   # アプリケーションウィンドウ
13  pygame.display.set_caption('Application: pygame05.py')
14
15  fps = pygame.time.Clock()   # フレームレート制御のための Clock オブジェクト
16
17  # メインループ
18  while True:
19      # イベントキューを処理するループ
```

```
20        for ev in pygame.event.get():
21            if ev.type == QUIT:      # 「終了」イベント
22                pygame.quit()
23                print('quitting...')
24                sys.exit()
25            elif ev.type == MOUSEBUTTONDOWN:
26                print('Mouse button was pressed:\t',ev.pos,ev.button)
27            elif ev.type == MOUSEBUTTONUP:
28                print('Mouse button was released:\t',ev.pos,ev.button)
29            elif ev.type == MOUSEMOTION:
30                print('Mouse is moving:\t\t',ev.pos,ev.buttons)
31            elif ev.type == KEYDOWN:
32                print('A key was pressed:\t\t',ev.key,ev.mod,ev.unicode)
33            elif ev.type == KEYUP:
34                print('The key was released:\t\t',ev.key,ev.mod)
35
36        # ディスプレイの更新
37        pygame.display.update()
38        # フレームレートの設定
39        fps.tick(12)    # 12FPSに設定
```

このプログラムを実行すると小さなウィンドウが表示され，キーボード，マウスからのイベントを受けて，それらイベントの各種プロパティの値が表示されることが確認できる．

2.1.4 音声の再生

pygame は WAV 形式音声データに加えて，MP3 [5] や Ogg Vorbis [6] といったフォーマットの音声データを再生することができる．pygame の音声再生機能は pygame.mixer.music パッケージにあり，例えば音声データ 'dat1.mp3' を読込んで再生するには次のように記述する．

```
pygame.mixer.music.load('dat1.mp3')
pygame.mixer.music.play()
```

pygame.mixer.music パッケージの主要な機能を表 9 に挙げる．

表 9: サウンド再生のための主な機能

機能（関数）	説明
load(Fname)	Fname のパスから音声データを読み込む
play( )	読み込まれた音声データを再生する
stop( )	再生中の音声を 停止 する
pause( )	再生中の音声を 一時停止 する
unpause( )	一時停止した音声の再生を再開する
rewind( )	再生中の音声を巻き戻す．（先頭に戻す）
fadeout(t)	再生中の音声をフェードアウトして 停止 する フェードアウトタイムは t で指定する．（単位：ミリ秒）
get_pos( )	再生中の位置（時刻）を返す．（単位：ミリ秒）
set_pos(t)	再生中の位置（時刻）t を指定する．（単位：ミリ秒）
get_volume( )	音量の設定値（0〜1.0）を取得する
set_volume(v)	音量を設定（$0.0 \leq v \leq 1.0$）する
get_busy( )	音声が再生中であれば True を，そうでなければ False を返す

注）再生位置の制御に関しては，扱う音声フォーマットによって扱いが異なる．

音声データを再生するプログラムの例 pygame06.py をに示す．

[5] **MP3**（MPEG-1 Audio Layer-3）：最も広く普及している音声圧縮フォーマットである．厳密には，特許に関連する問題のため，完全に自由に使える技術ではないので，商用の利用には注意が必要である．

[6] **Ogg Vorbis**：Vorbis コーデックで圧縮伸張し，Ogg コンテナにサウンドを格納する音声フォーマットである．オープンソースであり自由に利用できる．

プログラム：pygame06.py

```
# coding: utf-8

# モジュールの読み込み
import sys
import pygame
from pygame.locals import QUIT

# pygameの初期化
pygame.init()

sf = pygame.display.set_mode( (320,240) )   # アプリケーションウィンドウ
pygame.display.set_caption('Application: pygame06.py')

fps = pygame.time.Clock()   # フレームレート制御のための Clock オブジェクト

# サウンドデータの読込みと再生
pygame.mixer.music.load('sound02.ogg')
pygame.mixer.music.play()

# メインループ
while True:
    # イベントキューを処理するループ
    for ev in pygame.event.get():
        if ev.type == QUIT:     # 「終了」イベント
            pygame.quit()
            print('quitting...')
            sys.exit()

    # 再生が終了したときの処理
    if pygame.mixer.music.get_busy():
        pass
    else:
        print('music has finished.')
        pygame.quit()
        sys.exit()

    print('Playing:',pygame.mixer.music.get_pos(),'msec' )

    # ディスプレイの更新
    pygame.display.update()
    # フレームレートの設定
    fps.tick(2) # 2FPSに設定
```

このプログラムを実行すると，小さなウィンドウを表示した後，音声ファイル'sound02.ogg' を読み込んで再生を開始する．再生中は再生の経過時間が標準出力にミリ秒単位で表示される．

2.1.5 スプライトの利用

先に挙げたプログラム pygame00.py では，1つのボールがウィンドウ内で移動する様子を示した．プログラムの基本的な流れとしては概ね，

1. ウィンドウ内の消去
2. ゲームフィールド1場面の状態（画像の位置など）の生成
3. ゲームフィールド1場面の表示とウィンドウの更新

というものであり，これらを全て「メインループ」内で記述している．

実際のゲームでは，ゲームフィールド内で複数のキャラクタ（画像部品など）がそれぞれ独自の動きをすることが多く，全ての画像部品の動きを表現するための変数などを個別に用意して，メインループ内で全ての変数の変化を記述するのは非常に煩雑な作業となる．

ここで紹介する**スプライト**を使用すると，ゲームフィールド内にある画像部品を別々のオブジェクトとして管理でき，ゲーム開発が大幅に簡素化できる．具体的には，ゲームフィールドに登場するそれぞれのキャラクタを個別のオブジェクト（Sprite クラス）として生成して扱い，個々のキャラクタの動きを，そのオブジェクトに対するメソッドとして実装するというスタイルを取る．

【Sprite クラス】

Sprite クラスは，ゲームキャラクタの画像を image 属性として，その位置とサイズを rect 属性として保持する．ゲームキャラクタはゲーム展開の個々のフレームで位置を始めとする状態を更新することで動きを実現する．Sprite オブジェクトの状態を変化させる（更新する）ためのメソッドとして update，それを Surface に描画するためのメソッドとして draw があり，Sprite クラスの拡張クラスの定義でそれらをオーバーライドして，各種スプライトに独自の動きを実現する．Sprite クラスは pygame.sprite.Sprite としてアクセスできる．

■ サンプルプログラム 1

次に示すサンプルプログラム pygame_sprt1.py は，先に挙げたプログラム pygame00.py と類似の動作（ボールがバウンドするアニメーション）をするものである．これに沿ってスプライトの最も基本的な取り扱いについて説明する．

プログラム：pygame_sprt1.py

```
1  # coding: utf-8
2  # モジュールの読込み
3  import sys
4  import pygame
5  from pygame.locals import QUIT      # 終了イベント
6  from pygame.locals import Rect      # 矩形クラス
7
8  pygame.init()   # pygameの初期化
9
10 #####################################################
11 # スプライト用クラスの定義                          #
12 # pygame.sprite.Sprite を継承して拡張する          #
13 # 基本プロパティ：                                  #
14 #   image   -   スプライト用画像                    #
15 #   rect    -   スプライトの位置とサイズ（矩形）    #
16 #   vx, vy  -   移動時の差分（移動量）              #
17 #####################################################
18 class MySprite( pygame.sprite.Sprite ):
19     def __init__(self,imgFname,x,y,vx,vy):
20         pygame.sprite.Sprite.__init__(self)
21         self.image = pygame.image.load(imgFname).convert_alpha()
22         width = self.image.get_width()
23         height = self.image.get_height()
24         self.rect = Rect(x,y,width,height)
25         self.vx = vx
26         self.vy = vy
27     def update(self):
28         global w, h     # ウィンドウサイズ（大域変数）
29         if self.rect.left < 0:
30             self.vx = -self.vx
31             self.rect.left = 0
32         if self.rect.right > w:
33             self.vx = -self.vx
34             self.rect.right = w
35         if self.rect.top < 0:
36             self.vy = -self.vy
37             self.rect.top = 0
38         if self.rect.bottom > h:
39             self.vy = -self.vy
40             self.rect.bottom = h
41         self.rect.move_ip(self.vx,self.vy)
42     def draw(self,screen):
43         screen.blit(self.image,self.rect)
44
45 #####################################################
```

```
46 # ゲームフィールドの準備                                          #
47 ###################################################
48 w = 400;    h = 300     # ウィンドウサイズ
49 
50 # ウィンドウ（Surface）の生成
51 sf = pygame.display.set_mode( (w,h) )
52 pygame.display.set_caption('Sprite Test (1)')
53 
54 # Clockオブジェクトの生成（フレームレート制御関連）
55 fps = pygame.time.Clock()
56 
57 ###################################################
58 # スプライトの生成                                              #
59 ###################################################
60 spr1 = MySprite('ball.png', 0,0, 5,3 )
61 
62 ###################################################
63 # メインループ                                                  #
64 ###################################################
65 while True:
66     # フレームレート設定（毎回）
67     fps.tick(30)
68     # ウィンドウ消去（毎回）
69     sf.fill( (0,0,0) )      # 毎回，真っ暗にする
70     # スプライト描画
71     spr1.update()       # スプライトの状態の更新
72     spr1.draw(sf)       # スプライトの表示
73     # イベント検出部
74     for ev in pygame.event.get():
75         if ev.type == QUIT:     # 終了処理
76             pygame.quit()
77             print('終了します．')
78             sys.exit()
79     # ウィンドウの更新
80     pygame.display.update()
```

このプログラムでは，Sprite クラスの拡張クラスとして MySprite クラスを定義して，このクラスのインスタンスとしてゲームキャラクタを取り扱う．MySprite クラスはゲームキャラクタ用の画像（image 属性）と，その位置とサイズの情報（rect 属性）を保持するだけでなく，フレーム更新時の位置の変化量も vx, vy プロパティとして保持するように実装されている．MySprite クラスのインスタンスを生成する際，コンストラクタの引数に，画像のファイル名，それに初期の位置と変化量を与える．

メインループ内では，ウィンドウの消去とスプライトの状態の更新，表示を繰り返し行なっている．ボールの絵として表示されるスプライト spr1 は，フレームの更新の度に，

- 移動量プロパティ vx, vy に基づく位置の変更
- ウィンドウの端に衝突したかどうかの判定と，それに基づく移動方向の変更

を update メソッドで行っている．実際の表示は draw メソッドで行っている．

rect プロパティは配下に top, bottom, left, right プロパティを持っており，それぞれ上下左右の座標値に対応している．これらプロパティの値を参照して，ウィンドウの端に衝突したことを判定している．

このプログラムを実行した様子を図 7 に示す．

■ サンプルプログラム 2

複数のスプライトを保持する Group クラスを使用すると，複数のゲームキャラクタを同時に扱うことが容易になる．Group クラスは pygame.sprite.Group としてアクセスできる．

先のプログラム pygame_sprt1.py を変更して，複数のスプライトを同時に扱う形にしたプログラム pygame_sprt2.py を次に示す．

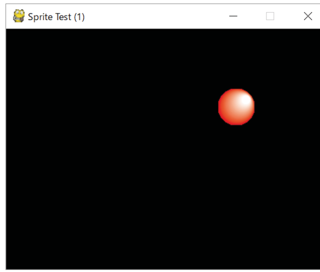


図 7: 1 個のボールがバウンドするアニメーション

プログラム： pygame_sprt2.py

```
1  # coding: utf-8
2  # モジュールの読込み
3  import sys
4  import pygame
5  from pygame.locals import QUIT      # 終了イベント
6  from pygame.locals import Rect      # 矩形クラス
7
8  pygame.init()   # pygameの初期化
9
10 #####################################################
11 # スプライト用クラスの定義                                  #
12 # pygame.sprite.Sprite を継承して拡張する                  #
13 # 基本プロパティ：                                         #
14 #   image   -   スプライト用画像                           #
15 #   rect    -   スプライトの位置とサイズ（矩形）           #
16 #   vx, vy  -   移動時の差分（移動量）                     #
17 #####################################################
18 class MySprite( pygame.sprite.Sprite ):
19     def __init__(self,imgFname,x,y,vx,vy):
20         pygame.sprite.Sprite.__init__(self)
21         self.image = pygame.image.load(imgFname).convert_alpha()
22         width = self.image.get_width()
23         height = self.image.get_height()
24         self.rect = Rect(x,y,width,height)
25         self.vx = vx
26         self.vy = vy
27     def update(self):
28         global w, h     # ウィンドウサイズ（大域変数）
29         if self.rect.left < 0:
30             self.vx = -self.vx
31             self.rect.left = 0
32         if self.rect.right > w:
33             self.vx = -self.vx
34             self.rect.right = w
35         if self.rect.top < 0:
36             self.vy = -self.vy
37             self.rect.top = 0
38         if self.rect.bottom > h:
39             self.vy = -self.vy
40             self.rect.bottom = h
41         self.rect.move_ip(self.vx,self.vy)
42     def draw(self,screen):
43         screen.blit(self.image,self.rect)
44
45 #####################################################
46 # ゲームフィールドの準備                                    #
47 #####################################################
48 w = 400;    h = 300     # ウィンドウサイズ
49
50 # ウィンドウ（Surface）の生成
51 sf = pygame.display.set_mode( (w,h) )
52 pygame.display.set_caption('Sprite Test (2)')
53
54 # Clockオブジェクトの生成（フレームレート制御関連）
```

```
55 fps = pygame.time.Clock()
56
57 ######################################################
58 # スプライトの生成                                     #
59 ######################################################
60 sg = pygame.sprite.Group()      # スプライトグループの生成
61 for i in range(5):
62     spr = MySprite('ball.png', i*80,i*60, 5,3 )
63     sg.add(spr)
64
65 ######################################################
66 # メインループ                                         #
67 ######################################################
68 while True:
69     # フレームレート設定（毎回）
70     fps.tick(30)
71     # ウィンドウ消去（毎回）
72     sf.fill( (0,0,0) )       # 毎回，真っ暗にする
73     # スプライト描画
74     sg.update()
75     sg.draw(sf)
76     # イベント検出部
77     for ev in pygame.event.get():
78         if ev.type == QUIT:      # 終了処理
79             pygame.quit()
80             print('終了します．')
81             sys.exit()
82     # ウィンドウの更新
83     pygame.display.update()
```

このプログラムの前半部は pygame_sprt1.py と同じであるが，複数のスプライトをまとめる Group である sg を生成し，そこに 5 個の同じスプライトを初期位置を変えて登録するものである．メインループ内でも，スプライトグループである sg に対して update し，draw している．

このプログラムを実行した様子を図 8 に示す．

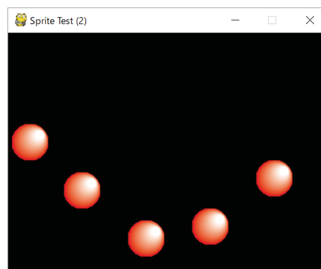


図 8: 5 個のボールがバウンドするアニメーション

3 科学技術系

3.1 数値計算と可視化のためのパッケージ：NumPy / matplotlib

ここでは，数値計算のためのパッケージである NumPy と データを可視化するためのパッケージである matplotlib に関して導入的に解説する．

NumPy は LAPACK（http://www.netlib.org/lapack/）や BLAS （http://www.openblas.net/）といった数値演算ライブラリを使用して構築されており，大規模な配列データを処理するための機能を提供する．NumPy はオープンソースのソフトウェアであり，ソフトウェア本体やドキュメントなどがインターネットサイト http://www.numpy.org/ で公開されている．

NumPy を使用するには次のようにしてモジュールを読み込む必要がある．

```
import numpy
```

あるいは，次のようにしてパッケージ名の別名を指定して読み込むことも慣例となっている．

```
import numpy as np
```

こうすることで，NumPy の各種関数やクラス，プロパティを numpy.〜 として記述する代わりに np.〜 として記述することができる．(以後の説明ではこの慣例に従う）

Python は動的な型付けの言語処理系であり，リストをはじめとするデータ構造の要素の型に制限はない．しかし NumPy の処理は C 言語や FORTRAN などで実装された演算機能を含んでおり，扱うデータ列（配列）の要素は表 10 に挙げるような特定の数値型に限定される．

表 10: NumPy の代表的な数値型

タイプ	意味	タイプ	意味
int8	符号付き 8 ビット整数	uint8	符号無し 8 ビット整数
int16	符号付き 16 ビット整数	uint16	符号無し 16 ビット整数
int32	符号付き 32 ビット整数	uint32	符号無し 32 ビット整数
int64	符号付き 64 ビット整数	uint64	符号無し 64 ビット整数
float16	16 ビット浮動小数点数	complex64	64 ビット複素数
float32	32 ビット浮動小数点数	complex128	128 ビット複素数
float64	64 ビット浮動小数点数	complex192	192 ビット複素数
float96	96 ビット浮動小数点数	complex256	256 ビット複素数
float128	128 ビット浮動小数点数		

※ 処理系によっては使えない型もあるので確認すること

3.1.1 配列オブジェクトの生成

NumPy では配列データ（データ列，行列）は array オブジェクト（データ型は ndarray）として扱う．array オブジェクトのコンストラクタの引数に，配列データをリストで与えることができる．

例. リストから NumPy の配列を生成する例

```
>>> import numpy as np  Enter    ←パッケージを'np' として読み込んでいる
>>> lst = [1,2,3,4,5]  Enter    ←通常のリストの形でデータ列を生成
>>> ar = np.array(lst)  Enter    ← NumPy の配列に変換
>>> ar  Enter    ←内容の確認
array([1, 2, 3, 4, 5])    ← NumPy の配列になっている
```

■ 配列の要素の型について

array オブジェクトのプロパティ dtype には，当該 array オブジェクトの要素の型が保持されている．

例.（続き）array の要素の型

```
>>> ar.dtype  Enter      ←型の調査
dtype('int32')     ←要素の型は 'int32' であることがわかる
```

array オブジェクトを生成する際，コンストラクタにキーワード引数 dtype=**型** を与えることで要素の型を指定することができる．このときの型は表 10 のもの（パッケージ名. を先頭に付けたもの）を指定する．

例. 型を指定した配列の生成

```
>>> lst = [1,2,3,4,5]  Enter      ←通常のリストの形でデータ列を生成
>>> ar = np.array(lst,dtype=np.float64)  Enter      ← NumPy の配列に変換（float64）
>>> ar  Enter      ←内容の確認
array([ 1., 2., 3., 4., 5.])     ←浮動小数点数（float64）の要素を持つ NumPy の配列になっている
>>> ar.dtype  Enter      ←型の調査
dtype('float64')     ←要素の型は 'float64' であることがわかる
```

■ データ列の生成（数列の生成）

range 関数に似た方法でデータ列を生成する関数として arange がある．

例. 数列の生成

```
>>> a = np.arange(0.0, 2.0, 0.1)  Enter      ← 0 以上 2 未満の範囲の数列を 0.1 刻みで生成
>>> a  Enter      ←内容の確認
array([ 0.  , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.  ,
        1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9])          ←得られた数列
```

これとは別に，指定した区間を等分して n 個の要素から成る数列を得るには linspace 関数を使用する．

例. 区間 $[n_1, n_2]$（n_1 以上 n_2 以下）を等分して 10 個のデータを得る

```
>>> np.linspace( 0, 1.0, 10 )  Enter      ← 0〜1 までを等分（最後の 1 を含む）
array([ 0.  , 0.11111111, 0.22222222, 0.33333333, 0.44444444,     ←得られた数列
        0.55555556, 0.66666667, 0.77777778, 0.88888889, 1.  ])
```

このように最後の 1 を含んで 10 個のデータを得るため，結果的に 9 等分したものとなる．linspace 関数にキーワード引数 'endpoint=False' を与えると最後の 1 を含まず，結果として 10 等分したデータを得ることができる．

例. 10 等分した形でデータを得る

```
>>> np.linspace( 0, 1.0, 10, endpoint=False )  Enter      ← 0〜1 までを 10 等分
array([ 0.  , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])     ←得られた数列
```

arange や linspace を使用して生成したデータ列は，NumPy の各種関数の定義域として与えることができ，値域のデータ列を得る目的に使用することができる．対数スケールで関数の定義域のデータ列を生成するには logspace を使用るのが良い．

例. 2^0〜2^{10} の数列を対数スケールで 6 個の要素として生成

```
>>> np.logspace( 0, 10, 6, base=2 )  Enter      ←対数スケール列生成
array([ 1.00000000e+00, 4.00000000e+00, 1.60000000e+01,     ←得られた数列
        6.40000000e+01, 2.56000000e+02, 1.02400000e+03])
```

キーワード引数 'base=' を省略すると 10 が基数となる．

例. 10^0〜10^{10} の数列を対数スケールで 6 個の要素として生成

```
>>> np.logspace( 0, 10, 6 )  Enter     ←対数スケール列生成
array([ 1.00000000e+00, 1.00000000e+02, 1.00000000e+04,     ←得られた数列
        1.00000000e+06, 1.00000000e+08, 1.00000000e+10])
```

3.1.1.1 多次元配列の生成

多次元の配列もリストから生成することができる．

例. 2 次元配列の生成

```
>>> a2 = np.array([[1,2,3,4],[5,6,7,8]])  Enter     ← 2 次元配列（行列）の生成
>>> a2  Enter     ←内容確認
array([[1, 2, 3, 4],      ←生成結果
       [5, 6, 7, 8]])
```

2 行 4 列の配列が得られている．

例. 3 次元配列の生成

```
>>> a3 = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])  Enter     ← 3 次元配列の生成
>>> a3  Enter          ←内容確認
array([[[1, 2],       ←生成結果
        [3, 4]],

       [[5, 6],
        [7, 8]]])
```

■ 配列のサイズの検査

配列オブジェクトの shape プロパティに配列の各次元のサイズがタプルとして保持されている．

例. （先の続き）配列のサイズ

```
>>> a2.shape  Enter     ← a2 のサイズを求める
(2, 4)     ← 2 行 4 列である
>>> a3.shape  Enter     ← a3 のサイズを求める
(2, 2, 2)     ← W × D × H は 2 × 2 × 2 である
```

1 次元のデータ列からも shape は取得できる．

例. 1 次元のデータ列のサイズ

```
>>> a1 = np.array([1,2,3])  Enter     ←データ列の生成
>>> a1.shape  Enter     ← a1 のサイズを求める
(3,)     ←タプルの形で得られる
>>> a1.shape[-1]  Enter     ←タプルの最終要素としてサイズを求める
3        ←長さが得られる（len(a1) としてもも同様）
```

3.1.2 データ列に対する演算：1 次元から 1 次元

NumPy に用意されている数学関数は，データ列からデータ列を生成することができる．また，これを応用すると関数のプロット（2 次元）が実現できる．

例. 正弦関数の列の生成

```
>>> import numpy as np  Enter     ←パッケージを'np' として読み込んでいる
>>> lx = np.arange(0.0,6.28,0.01)  Enter     ←データ列（定義域）の生成
>>> ly = np.sin(lx)  Enter     ←上記データ列の各要素に対する正弦関数の列の生成
```

これで，定義域 lx に対する正弦関数の値域 ly が生成された．これらデータ列を matplotlib パッケージでプロットする例を次に示す．

例. （つづき）正弦関数の列のプロット

```
>>> import matplotlib.pyplot as plt  Enter    ← matplotlib パッケージを'plt' として読み込んでいる
>>> plt.plot(lx,ly)  Enter    ←プロットオブジェクトの生成
[<matplotlib.lines.Line2D object at 0x0000026B4AF4B828>]    ←生成結果
>>> plt.show()  Enter    ←プロットを表示
```

この結果，図 9 に示すようなプロットが表示される．

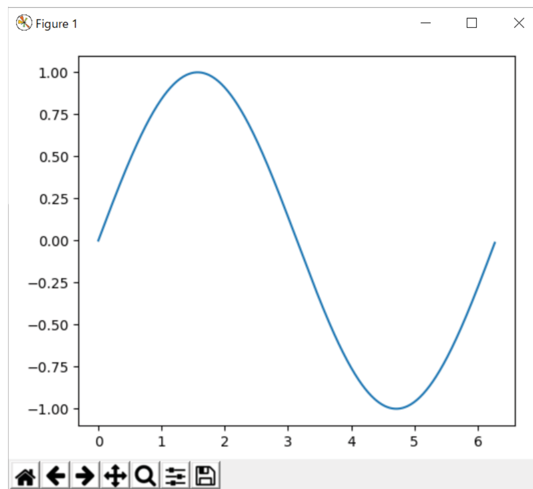


図 9: プロットの表示

あるいは，plt.bar(lx,ly) とすることで**棒グラフ**を描画することもできる．matplotlib に関しては「3.1.3 データの可視化」で説明する．

ここで紹介した正弦関数 sin は Numpy が提供する関数群の中の 1 つであり，この他にもたくさんの関数が提供されている．使用頻度が高いと思われるものを表 11 に示すが，詳しくは Nympy の公式サイトを参照のこと．

表 11: Numpy が提供する関数の一部

関数	説明	関数	説明
sum	配列要素の合計を求める	mean	配列要素の平均を求める
var	配列要素の分散を求める	std	配列要素の標準偏差を求める
max	配列要素の最大値を求める	min	配列要素の最小値を求める

3.1.3 データの可視化：基本

matplotlib はデータを可視化するためのオープンソースのパッケージであり，関連情報がインターネットサイト http://matplotlib.org/ で公開されている．matplotlib に含まれるデータの可視化の機能は matplotlib.pyplot モジュールにあり，これを読み込むには次のようにする．

```
import matplotlib.pyplot as plt
```

こうすることで，パッケージの別名として plt を指定することができ，可視化のためのクラス，プロパティを plt.〜 として記述することができる．（以後の説明ではこの慣例に従う）

3.1.3.1　2 次元のプロット：折れ線グラフ

正弦関数と余弦関数をプロットするプログラムを例に挙げて，2 次元プロットの基本的な方法について説明する．まずプログラム例 nplot01.py を示す．

プログラム：nplot01.py

```
1  # coding: utf-8
2  # モジュールの読み込み
3  import numpy as np
4  import matplotlib.pyplot as plt
5
6  # データ列の生成
7  lx = np.arange(0.0,6.28,0.01)     # 定義域の生成
8  ly1 = np.sin(lx)                  # 正弦関数の列
9  ly2 = np.cos(lx)                  # 余弦関数の列
10
11 # データ列のプロット
12 plt.plot(lx,ly1, label='sin(x)') # プロット(1)
13 plt.plot(lx,ly2, label='cos(x)') # プロット(2)
14 plt.xlabel('x')      # 横軸ラベル
15 plt.ylabel('y')      # 縦軸ラベル
16 plt.legend()         # 凡例の表示
17 plt.title('trigonometric functions: sin, cos')  # タイトルの表示
18 plt.show()       # プロットの表示
```

このプログラムを実行すると図 10 のようなプロットが表示される．

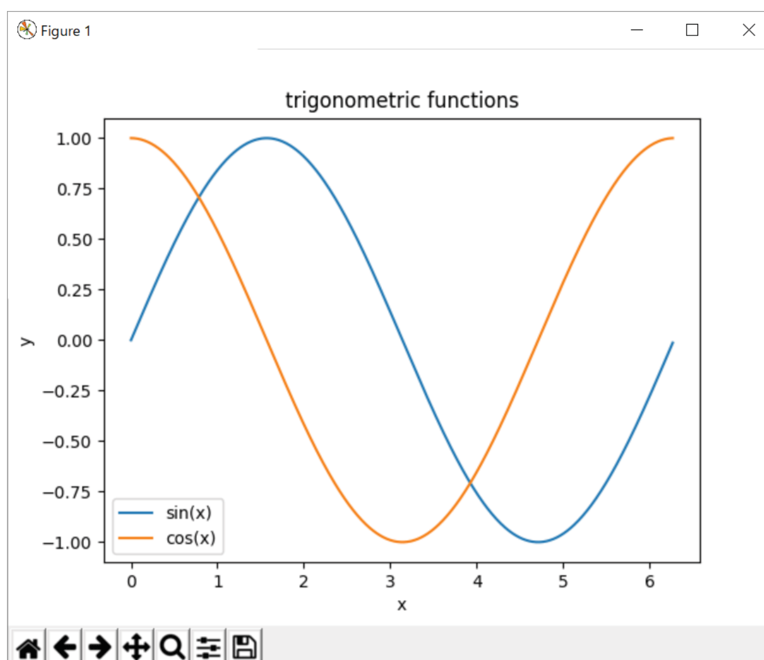


図 10: プロットの表示

プログラムの説明：

7～9 行目で定義域の集合 lx とそれに対する，正弦関数，余弦関数の値域の集合 ly1, ly2 を生成している．それらをプロットしているのが 12,13 行目であり，plot 関数を使用している．plot 関数の第 1，第 2 の引数に横軸データと縦軸データをそれぞれ与え，キーワード引数 'labe=' にそのデータ列の**ラベル**を与える．これはプロットを表示する際の**凡例**となる．

プログラムの 14,15 行目はグラフの横軸と縦軸のラベルを関数 xlabel，ylabel で与えている．16 行目では関数 legend によってグラフに凡例を付与している．17 行目では関数 title によってグラフにタイトルを付与している．

最後に関数 show によって実際にプロットを表示している．

【plot 関数のキーワード引数】

plot 関数のキーワード引数には先に説明した 'label=' 以外にも様々なもの（表 12）がある．

表 12: plot 関数のキーワード引数（一部）

キーワード引数	説明
label=**凡例**	「凡例」を文字列で与える．False を与えると凡例を表示しない．
color=**色**	描画色を指定する．次の文字列が指定できる． 'red', 'green', 'blue', 'cyan', 'magenta', 'yellow', 'black' この他にも**カラーマップ**を用いた方法もある．
lw=**太さ**	描画の線の太さを与える．単位はポイント
ls=**スタイル**	線のスタイルを次のような文字列で与える． '-'（実線：デフォルト），'--'（破線），':'（点線），'-.'（一点鎖線）（他にもあり）
marker= **マーカーの種類**	マーカーの種類を次のような文字列で与える． '.'（ドット），'o'（丸），'s'（■），'*'（星），'+'（十字），'x'（×）， '^'（三角形），'v'（逆三角形），'>'（右向き三角形），'<'（左向き三角形） （その他多数）
markersize= **マーカーのサイズ**	マーカーのサイズを数値で与える．単位はポイント
alpha=**α 値**	透明度を $0 \leq \alpha \leq 1$ の範囲で指定する．大きいほど濃い．

線のスタイル，色，太さを設定する例として，次のプログラム nplot01.py を示す．これは，不連続な関数（正接関数：$\tan(x)$）を 3 つの定義域

$$-\frac{\pi}{2} \leq x \leq \frac{\pi}{2}, \quad \frac{\pi}{2} \leq x \leq \frac{3\pi}{2}, \quad \frac{3\pi}{2} \leq x \leq \frac{5\pi}{2}$$

に分けてプロットする例であり，それぞれの定義域で線の設定を異なるものにしている．

プログラム：nplot02.py

```
1  # coding: utf-8
2  # モジュールの読み込み
3  import numpy as np
4  import matplotlib.pyplot as plt
5
6  # データ列の生成
7  p1 = -1.0 * np.pi / 2.0
8  p2 = np.pi / 2.0
9  p3 = 3.0 * np.pi / 2.0
10 p4 = 5.0 * np.pi / 2.0
11
12 lx1 = np.arange(p1+0.01, p2, 0.01)  # 定義域の生成(1)
13 ly1 = np.tan(lx1)                   # 正接関数の列(1)
14 lx2 = np.arange(p2+0.01, p3, 0.01)  # 定義域の生成(2)
15 ly2 = np.tan(lx2)                   # 正接関数の列(2)
16 lx3 = np.arange(p3+0.01, p4, 0.01)  # 定義域の生成(3)
17 ly3 = np.tan(lx3)                   # 正接関数の列(3)
18
19 # データ列のプロット
20 plt.plot(lx1,ly1, color='red',  lw=3, ls=':'  ) # プロット(1)
21 plt.plot(lx2,ly2, color='black',lw=2, ls='--' ) # プロット(2)
22 plt.plot(lx3,ly3, color='blue', lw=1, ls='-.' ) # プロット(3)
23 plt.grid(ls='--',lw=0.8,alpha=1.0)              # グリッド表示
24 plt.ylim(-100,100)
25 plt.xlabel('x')     # 横軸ラベル
26 plt.ylabel('y')     # 縦軸ラベル
27 plt.title('trigonometric functions: tan(x)')  # タイトルの表示
28 plt.show()      # プロットの表示
```

このプログラムを実行すると図 11 のようなプロットが表示される．

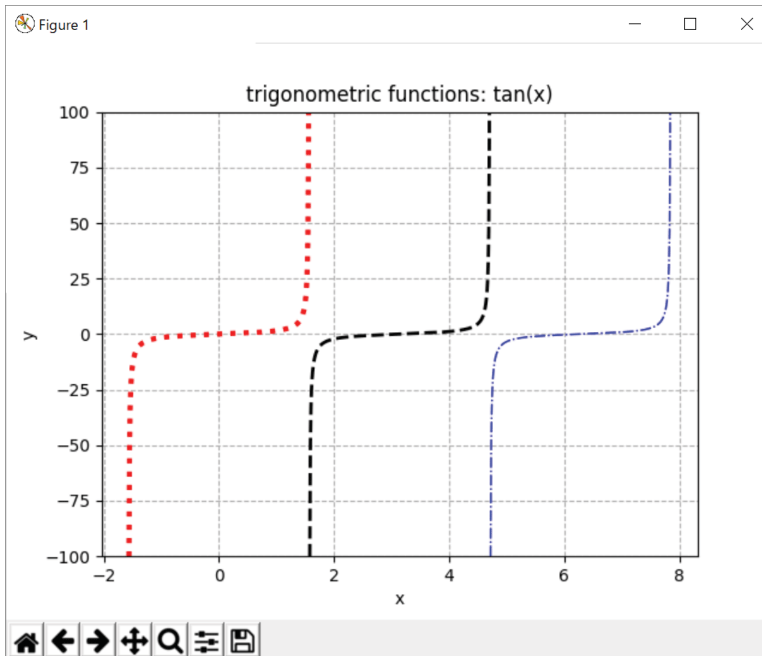


図 11: プロットの表示

プロットに**グリッド線**を描くには grid 関数を実行する．この関数の引数には plot 関数のキーワード引数と共通するものがある．

3.1.3.2　複数のグラフの作成

1 枚の図に複数のグラフを作成するには subplots メソッドを使用する．subplots を呼び出す際にグラフの縦横の並び（行，列の数）を指定すると，それに対応する作図用オブジェクトが生成される．実際にプログラムの例を示して説明する．

■ 2 つのグラフを作成する例

正弦関数と余弦関数の 2 つを別のグラフとして作成するプログラム nplot02-2.py を次に示す．

プログラム：nplot02-2.py

```
1  # coding: utf-8
2
3  # モジュールの読込み
4  import numpy as np
5  import matplotlib.pyplot as plt
6
7  # データ列の生成
8  x  = np.arange(-6.3, 6.3, 0.01) # 定義域の生成
9  y1 = np.sin(x)                  # 値域の生成(1)
10 y2 = np.cos(x)                  # 値域の生成(2)
11
12 # matplotlibによるプロット
13 (fig, ax) = plt.subplots(2, 1, figsize=(5,3))
14 plt.subplots_adjust(hspace=1.0)
15
16 ax[0].plot(x, y1, linewidth=1, color='red')
17 ax[0].set_title('sin(x)')
18 ax[0].set_ylabel('y')
19 ax[0].set_xlabel('x')
20 ax[0].grid(True)
21
22 ax[1].plot(x, y2, linewidth=1, color='green')
```

```
23 ax[1].set_title('cos(x)')
24 ax[1].set_ylabel('y')
25 ax[1].set_xlabel('x')
26 ax[1].grid(True)
27
28 plt.show()
```

このプログラムを実行してグラフを表示した例を図 12 に示す.

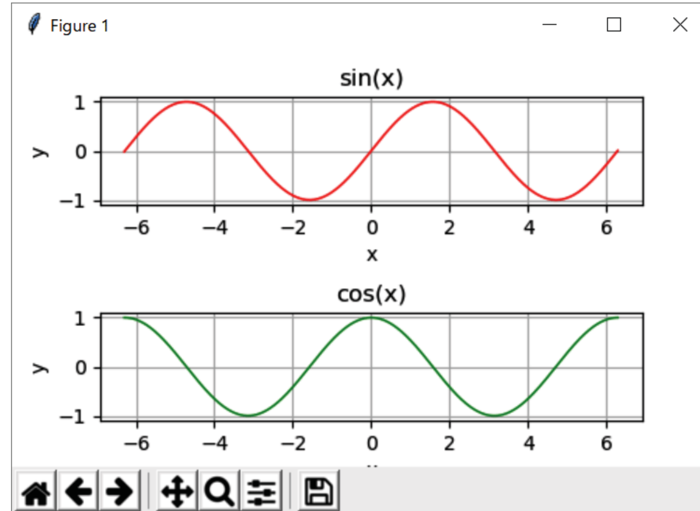


図 12: 2 つのグラフを表示した例(縦に 2 つ)

プログラムの解説:

nplot02-2.py の 8〜10 行目で定義域と値域(正弦関数,余弦関数)を生成している.13 行目で 2 行 1 列の並びでグラフを表示する形で subplots を呼び出している.

書き方:

subplots(**行数, 列数**)

また,このときキーワード引数 figsize=(**横のサイズ, 縦のサイズ**) を与えると,グラフ全体のサイズを指定することができる.subplots の実行後,matplotlib.figure.Figure オブジェクトと matplotlib.axes.Axes オブジェクトのタプルが返される.今回のプログラムでは,これらを (fig, ax) に受け取っており,ax[インデックス] に対して描画処理を行っている.

14 行目にあるように subplots_adjust メソッドを呼び出すと,描画するグラフの間隔を設定することができる.縦に並ぶグラフの上下の間隔はキーワード引数 hspace に,横に並ぶグラフの左右の間隔はキーワード引数 wspace に指定する.

プログラム nplot02-2.py の 13〜14 行目を,

```
(fig, ax) = plt.subplots(1, 2, figsize=(7,3))
plt.subplots_adjust(wspace=0.4)
```

と書き換えると,左右にグラフを並べる形の表示となる.(図 13 参照)

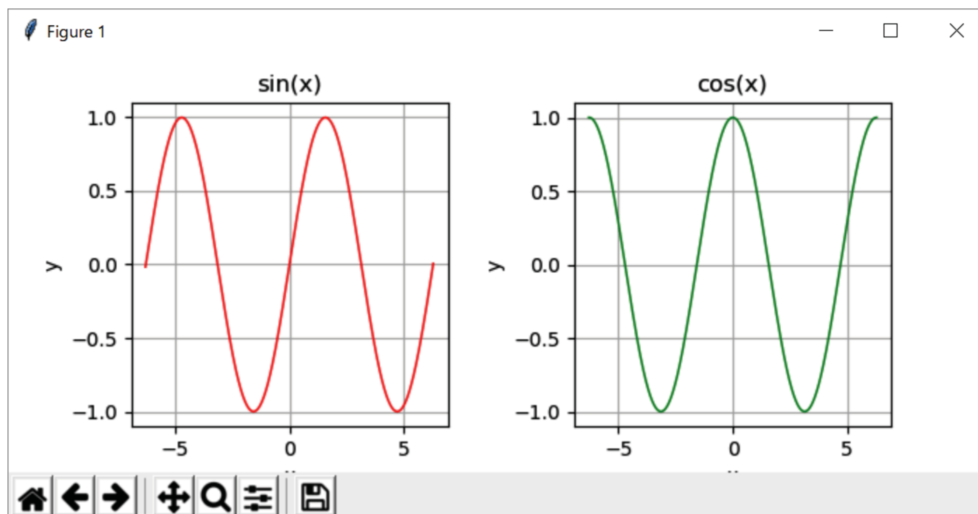


図 13: 2 つのグラフを表示した例（横に 2 つ）

■ 縦横にグラフを表示する例

正弦関数，余弦関数，指数関数，対数関数の 4 つを別のグラフとして作成するプログラム nplot02-4.py を次に示す．

プログラム：nplot02-4.py

```
1  # coding: utf-8
2
3  # モジュールの読込み
4  import numpy as np
5  import matplotlib.pyplot as plt
6
7  # データ列の生成
8  x1 = np.arange(-6.3, 6.3, 0.01) # 定義域の生成(1)
9  y1 = np.sin(x1)                 # 値域の生成(1)
10 y2 = np.cos(x1)                 # 値域の生成(2)
11 y3 = np.exp(x1)                 # 値域の生成(3)
12
13 x2 = np.arange(0.01, 10, 0.01)  # 定義域の生成(2)
14 y4 = np.log(x2)                 # 値域の生成(4)
15
16 # matplotlibによるプロット
17 (fig, ax) = plt.subplots(2, 2, figsize=(8,4))
18 plt.subplots_adjust(wspace=0.3,hspace=0.7)
19
20 ax[0,0].plot(x1, y1, linewidth=1, color='red')
21 ax[0,0].set_title('sin(x)')
22 ax[0,0].set_ylabel('y')
23 ax[0,0].set_xlabel('x')
24 ax[0,0].grid(True)
25
26 ax[1,0].plot(x1, y2, linewidth=1, color='green')
27 ax[1,0].set_title('cos(x)')
28 ax[1,0].set_ylabel('y')
29 ax[1,0].set_xlabel('x')
30 ax[1,0].grid(True)
31
32 ax[0,1].plot(x1, y3, linewidth=1, color='blue')
33 ax[0,1].set_title('exp(x)')
34 ax[0,1].set_ylabel('y')
35 ax[0,1].set_xlabel('x')
36 ax[0,1].grid(True)
37
38 ax[1,1].plot(x2, y4, linewidth=1, color='black')
39 ax[1,1].set_title('log(x)')
40 ax[1,1].set_ylabel('y')
41 ax[1,1].set_xlabel('x')
```

```
42 ax[1,1].grid(True)
43
44 plt.show()
```

プログラムの解説：

nplot02-4.py の 8〜14 行目でデータ列（定義域，正弦関数，余弦関数，指数関数，対数関数）を生成している．17 行目で 2 行 2 列の並びでグラフを表示する形で subplots を呼び出している．この結果として生成されたオブジェクト ax[行インデックス, 列インデックス] に対して描画処理を行っている．

このプログラムを実行してグラフを表示した例を図 14 に示す．

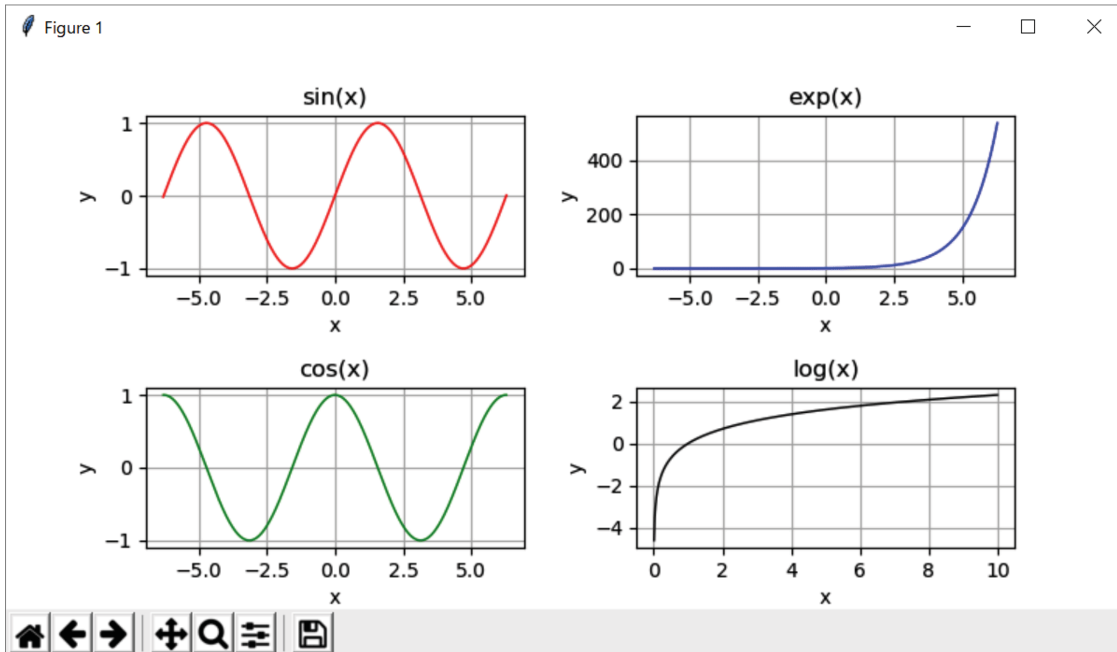


図 14: 縦横に 4 つのグラフを表示した例

■ 日本語の見出し・ラベルの表示

グラフ中の見出しやラベルに日本語フォントを使用するには matplotlib の FontProperties クラスを利用する．このクラスは matplotlib.font_manager パッケージにある．

例. 日本語フォントの読み込み（Windows 環境）

```
from matplotlib.font_manager import FontProperties
fp = FontProperties(fname=r'C:¥WINDOWS¥Fonts¥msgothic.ttc', size=11)
```

これは Windows の環境で標準的に利用できる「MS ゴシック（標準）」を読み込んで，11 ポイントのサイズのフォントとして FontProperties クラスの fp オブジェクトを生成している例である．先の例のプログラム nplot01.py を変更してタイトル，軸ラベル，凡例を日本語で表示する形にしたプログラム nplot01j.py を示す．

プログラム：nplot01j.py

```
1 # coding: utf-8
2 # モジュールの読み込み
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from matplotlib.font_manager import FontProperties
6
7 # グラフで使用する日本語フォント
8 fp = FontProperties(fname=r'C:\WINDOWS\Fonts\msgothic.ttc', size=11)
9
```

```
10 # データ列の生成
11 lx = np.arange(-3.15,3.15,0.01) # 定義域の生成
12 ly1 = np.sin(lx)                # 正弦関数の列
13 ly2 = np.cos(lx)                # 余弦関数の列
14
15 # データ列のプロット
16 plt.plot(lx,ly1, label='正弦関数') # プロット(1)
17 plt.plot(lx,ly2, label='余弦関数') # プロット(2)
18 plt.xlabel('定義域',fontproperties=fp)    # 横軸ラベル
19 plt.ylabel('値域',fontproperties=fp)      # 縦軸ラベル
20 plt.legend(prop=fp)                       # 凡例の表示
21 # タイトルの表示
22 plt.title('正弦関数, 余弦関数のプロット',fontproperties=fp)
23 plt.grid(True)
24 plt.show()      # プロットの表示
```

5 行目で FontProperties クラスを読み込み，8 行目でフォントを読み込んで fp に与えている．18,19 行目にあるように，軸ラベル設定時にキーワード引数 fontproperties=fp を与えると指定したフォントが有効になる．凡例にフォントを設定する場合は 20 行目にあるように legend メソッドのキーワード引数に prop=fp と指定する．

このプログラム実行して作成したグラフの例を図 15 に示す．

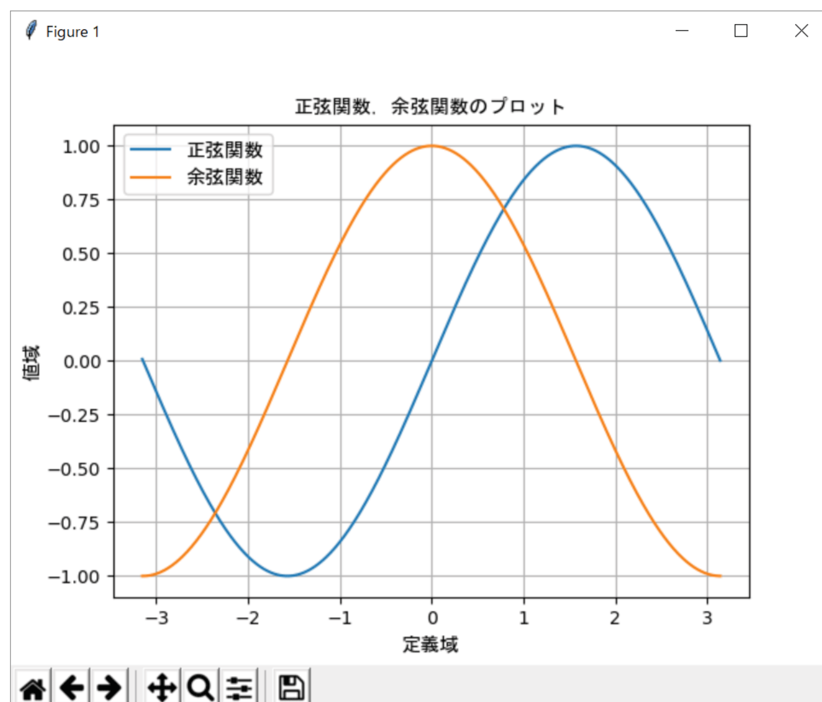


図 15: 見出しとラベルを日本語にした例

3.1.4　乱数の生成

■ 一様乱数の生成（0 以上 1 未満の範囲）

書き方：

1) np.random.rand()　　乱数を 1 つ生成する．
2) np.random.rand(**個数**)　　指定した個数の乱数を配列として生成する．
3) np.random.rand(n,m)　　n 行 m 列の乱数の配列を生成する．

例.

```
>>> np.random.rand()  Enter      ←乱数を 1 つ生成
0.9002721968823484               ←成績結果
>>> np.random.rand(5)  Enter     ←乱数を 5 個生成
array([ 0.32644595, 0.20630809, 0.98017323, 0.09793674, 0.39467418])   ←成績結果
>>> np.random.rand(5,3)  Enter   ←5 行 3 列の乱数配列を生成
array([[ 0.41218582, 0.36665746, 0.14565054],    ←成績結果
       [ 0.2018859 , 0.88586831, 0.88083754],
       [ 0.73630688, 0.78485615, 0.98865664],
       [ 0.58109305, 0.75149191, 0.26337745],
       [ 0.67083326, 0.91048167, 0.9451317 ]])
```

■ 整数の乱数の生成

書き方：　np.random.randint(L, H, **個数**)

整数 L 以上 H 未満の範囲で乱数を，指定した個数生成する．

例.

```
>>> np.random.randint(0,10,20)  Enter   ←0 以上 10 未満の乱数を 20 個生成
array([2, 1, 7, 4, 0, 1, 4, 3, 9, 2, 7, 6, 0, 8, 1, 4, 8, 9, 1, 7])   ←成績結果
```

■ 正規分布となる乱数の生成

書き方：　np.random.normal(**平均, 標準偏差, 生成個数**)

生成個数に (n, m) のタプルを与えると n 行 m 列の配列として生成する．

例.

```
>>> np.random.normal(0,1,10)  Enter   ←平均 0, 標準偏差 1 の正規分布データを 10 個生成
array([ 0.01706595, -0.44630863, 0.64802007, -0.86528213, 0.11454459,   ←成績結果
        1.55471322, 0.24815903, 1.16232311, 0.16387414, -0.52767615])
>>> np.random.normal(0,1,(5,3))  Enter   ←5 行 3 列の正規分布データを生成
array([[ 0.79680422, -1.20956605, 0.58653553],
       [-0.49538379, -0.70013524, -1.68294362],
       [-2.03277246, 0.7823404 , -1.4837754 ],
       [ 1.46255008, 0.4963593 , -0.01242249],
       [ 0.49222816, -0.03615764, 1.31829024]])
```

3.1.5　データの可視化：ヒストグラム，散布図

ここでは正規分布に従う乱数データを使用してヒストグラムと散布図の作成方法について説明する．

【ヒストグラムの作成例】

ヒストグラムのプロットには hist 関数を使用する．サンプルプログラムを nplot03.py に示す．

プログラム：nplot03.py

```
1 # coding: utf-8
2 # モジュールの読み込み
3 import numpy as np
4 import matplotlib.pyplot as plt
```

```
5
6  # データ列の生成
7  dat = np.random.normal(50,10,100000)
8
9  # ヒストグラムの表示
10 plt.hist(dat,bins=40)
11 plt.xlabel('x')
12 plt.ylabel('n')
13 plt.grid(ls='--',lw=0.8,alpha=1.0)
14 plt.title('mean:50, SD:10, total:100000')
15 plt.show()
```

これは平均が 50（$\mu = 50$），標準偏差が 10（$\sigma = 10$）となる 100,000 件のデータのヒストグラムを表示する例である．またヒストグラムの階級の数は 40 に設定されている．

このプログラムを実行すると図 16 のようなプロットが表示される．

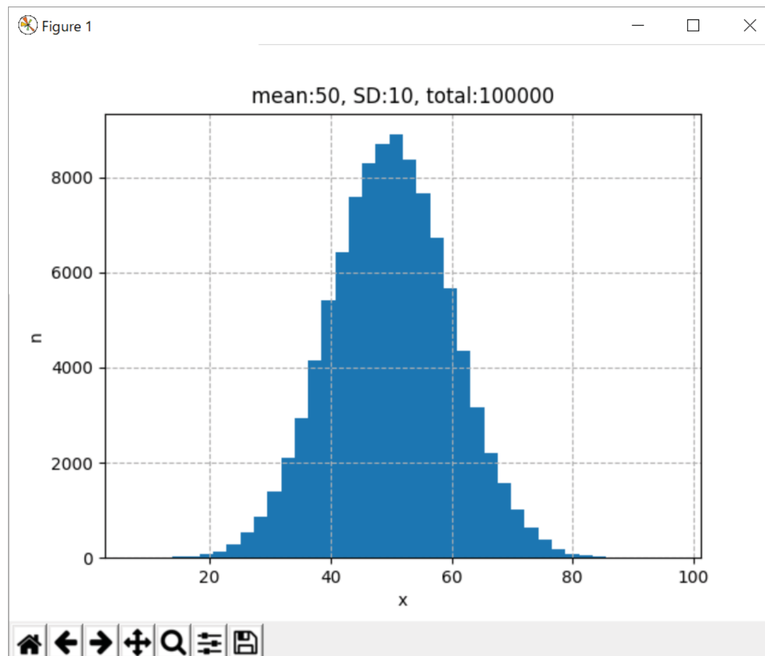


図 16: プロットの表示

【散布図の作成例】

散布図のプロットには scatter 関数を使用する．サンプルプログラムを nplot04.py に示す．

プログラム：nplot04.py

```
1  # coding: utf-8
2  # モジュールの読み込み
3  import numpy as np
4  import matplotlib.pyplot as plt
5
6  # データ列の生成
7  datx = np.random.normal(50,10,4000)
8  daty = np.random.normal(50,10,4000)
9
10 # ヒストグラムの表示
11 plt.scatter(datx,daty,alpha=0.2)
12 plt.xlabel('x')
13 plt.ylabel('y')
14 plt.grid(ls='--',lw=0.8,alpha=1.0)
15 plt.title('mean:50, SD:10, total:4000')
16 plt.show()
```

これは 4,000 件の 2 次元のデータ列の散布図を作成する例で，それぞれの軸の平均が 50 （$\mu = 50$），標準偏差が 10 （$\sigma = 10$）となる場合の例である．

このプログラムを実行すると図 17 のようなプロットが表示される．

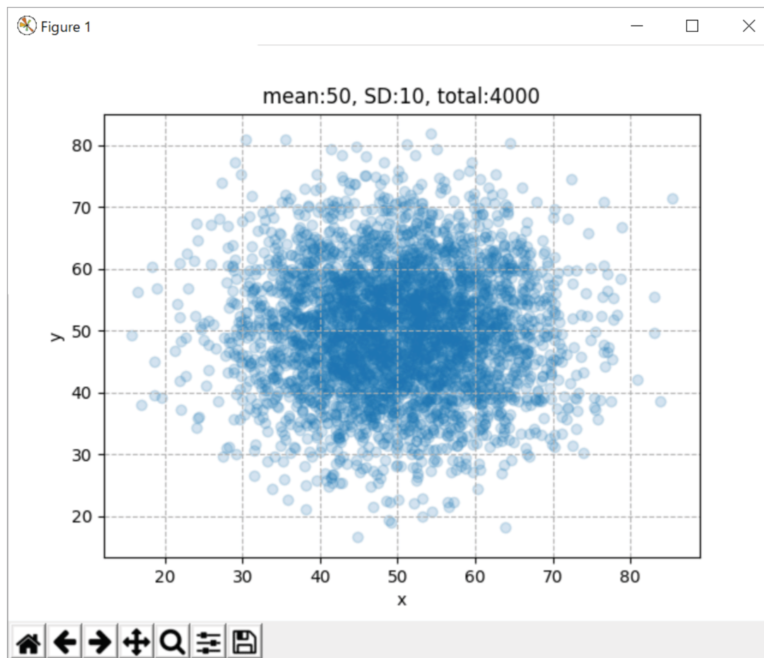


図 17: プロットの表示

3.1.6　データ列に対する演算：n 次元から 1 次元

ここではプログラム例を示しながら，高次元の関数の計算方法について説明する．高次元の関数 $\mathbb{R}^n \xrightarrow{f} \mathbb{R}$ の例として次のような関数について考える．

$$z = \cos(\sqrt{x^2 + y^2})$$

この関数の概形を図 18 に示す．

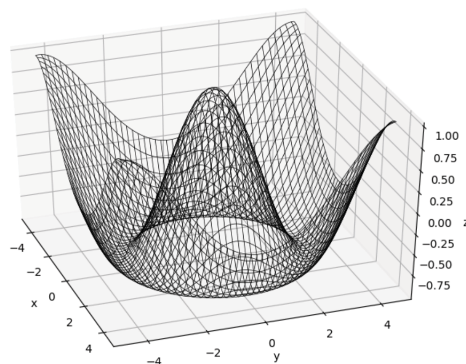


図 18: $z = \cos(\sqrt{x^2 + y^2})$ の概形

n 個の独立変数から 1 つの従属変数への関数を求めるには meshgrid を用いて n 次元の**定義域の格子**を生成し，それに対する関数の値（値域）を算出する方法がある．定義域の格子を生成するには，それを構成する各軸の 1 次元配列を meshgrid の引数に与える．

例. $x \in (-4.5, 4.5]$，$y \in (-4.5, 4.5]$ の定義域の格子を生成する

`>>> x = np.arange(-4.5, 4.5, 0.1)` Enter ←x 軸の範囲の列を生成

`>>> y = np.arange(-4.5, 4.5, 0.1)` Enter ←y 軸の範囲の列を生成

`>>> (X,Y) = np.meshgrid(x,y)` Enter ←x 定義域の格子を生成

`>>> Z = np.cos(np.sqrt(X**2+Y**2))` Enter ←関数の値域の生成

これにより，X,Y,Z に関数の定義域と値域のデータ配列ができる．

3.1.7 データの可視化：3 次元プロット

matplotlib で 3 次元のプロットを実現するには Axes3D クラスを使用する．このクラスを使用するには，次のようにして必要なモジュールを読み込む．

```
from mpl_toolkits.mplot3d import Axes3D
```

Axes3D オブジェクトを生成するには，コンストラクタの引数に figure オブジェクトを与える．

例. ax = Axes3D(plt.figure())

以後，この ax オブジェクトに対して 3 次元描画のメソッドを実行する．

3.1.7.1 ワイヤフレーム

先に挙げた関数 $z = \cos(\sqrt{x^2 + y^2})$ の**ワイヤフレーム**プロットを描画するプログラムを nplot05.py に示す．

プログラム：nplot05.py

```
1  # coding: utf-8
2  # モジュールの読み込み
3  import numpy as np
4  import matplotlib.pyplot as plt
5  from mpl_toolkits.mplot3d import Axes3D
6  
7  # meshgridの作成
8  x = np.arange(-4.5, 4.5, 0.1)
9  y = np.arange(-4.5, 4.5, 0.1)
10 (X,Y) = np.meshgrid(x,y)
11 
12 # 関数の算出
13 Z = np.cos(np.sqrt(X**2+Y**2))
14 
15 # 関数のプロット
16 ax = Axes3D(plt.figure())
17 ax.plot_wireframe(X,Y,Z, lw=0.5, color='black')
18 ax.set_xlabel('x')
19 ax.set_ylabel('y')
20 ax.set_zlabel('z')
21 plt.show()
```

17 行目にある plot_wireframe メソッドでワイヤフレームを生成している．このメソッドの第 1～第 3 引数に x, y, z それぞれの軸のデータを格納する配列を与える．キーワード引数には表 12 に挙げるようなものが指定できる．

軸のラベルを表示するには，18～20 行目にあるように set_xlabel，set_ylabel，set_zlabel メソッドを使用する．

このプログラムを実行すると図 19 のようなプロットが表示される．

3.1.7.2 面プロット（surface plot）

先のワイヤフレーム描画で使用した plot_wireframe メソッドの代わりに plot_surface メソッドを使用すると**面プロット**（surface plot）ができる．この際，matplotlib の**カラーマップ**モジュールを使用することで面にカラーマップを施すことができる．このためには必要なモジュールを次のようにして読み込んでおく．

```
from matplotlib import cm
```

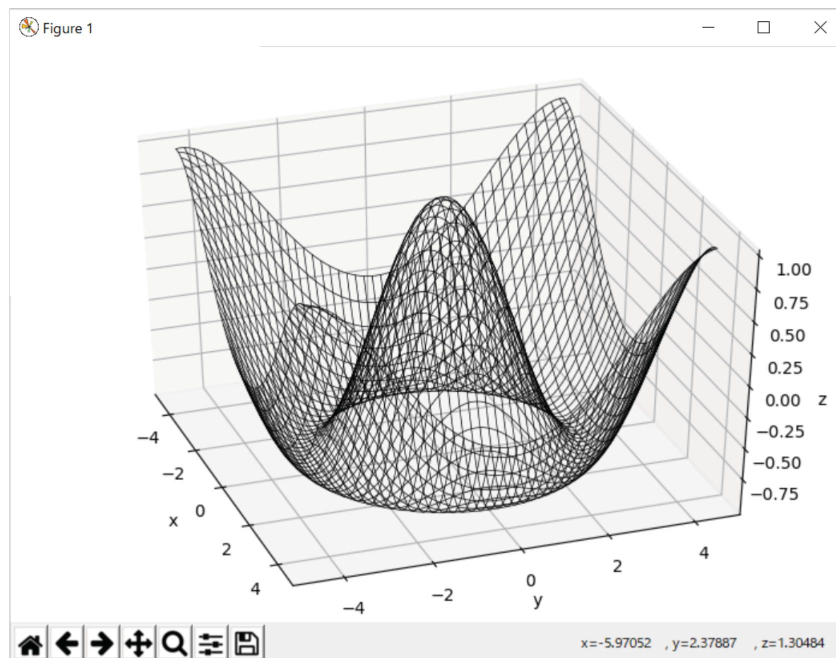


図 19: プロットの表示

サンプルプログラム nplot05-2.py の実行を例にして面プロットの実行結果を見る．

プログラム：nplot05-2.py

```
# coding: utf-8
# モジュールの読み込み
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

# meshgridの作成
x = np.arange(-4.5, 4.5, 0.1)
y = np.arange(-4.5, 4.5, 0.1)
(X,Y) = np.meshgrid(x,y)

# 関数の算出
Z = np.cos(np.sqrt(X**2+Y**2))

# 関数のプロット
ax = Axes3D(plt.figure())

#ax.plot_surface(X, Y, Z, cmap=cm.gray, shade=True)
#ax.plot_surface(X, Y, Z, cmap=cm.hot, shade=True)
#ax.plot_surface(X, Y, Z, cmap=cm.cool, shade=True)
#ax.plot_surface(X, Y, Z, cmap=cm.bwr, shade=True)
ax.plot_surface(X, Y, Z, cmap=cm.seismic, shade=True)

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.show()
```

プログラムの 19〜23 行目が面プロットを実行する部分であり，コメントを切り替えてこの内の 1 つを実行することでカラーマップの様子を見ることができる．カラーマップは plot_surface のキーワード引数 cmap= に与える．

このプログラムを実行して表示されるグラフのバリエーションを図 20 に示す．

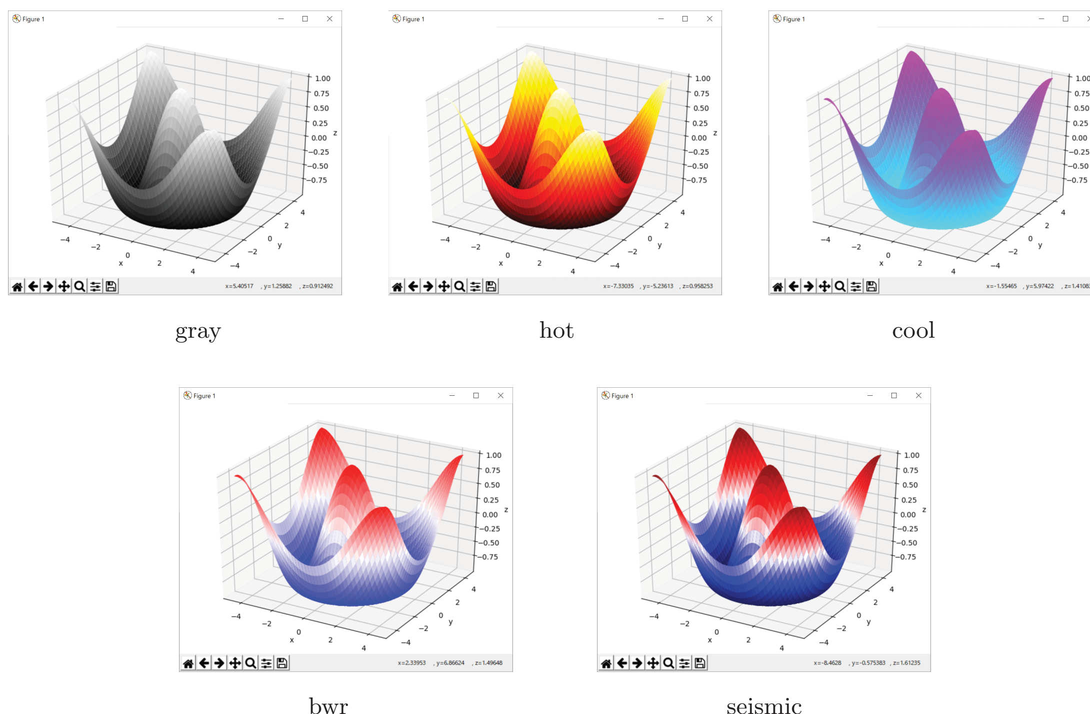


図 20: いくつかのカラーマップの例

3.1.8 高速フーリエ変換（FFT）

フーリエ変換は，時間 t の関数 $h(t)$ を別の変数 ω の関数 $H(\omega)$ に変換する次のような操作である．

$$H(\omega) = \int_{-\infty}^{\infty} h(t)\ \exp(-i\omega t)\, dt$$

また $\omega = 2\pi f$ と解釈すると，時間の関数から周波数の関数への変換であると見ることができ，この変換を応用すると，時間軸で解釈される振動（波形）を周波数成分に展開することができる．

関数 $H(\omega)$ は次のような**フーリエ逆変換**で元の関数 $h(t)$ に戻る．

$$h(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} H(\omega)\ \exp(i\omega t)\, d\omega$$

これらフーリエ変換と逆変換は信号解析をはじめとする工学的応用に不可欠な処理である．ここでは NumPy が提供するフーリエ変換と逆変換の機能の使用方法について導入的に解説する．

3.1.8.1 時間領域から周波数領域への変換：フーリエ変換

NumPy の fft パッケージに含まれる fft 関数を使用することで時間の関数を周波数の関数に変換することができる．

書き方： np.fft.fft(**データ列**)

これにより，与えたデータ列（時間領域）をフーリエ変換して周波数領域に変換したデータ列を返す．フーリエ変換が対象とするデータは**複素数**であり，変換によって得られる周波数領域のデータ列も複素数である．通常の信号解析では，扱う波形データは実数で構成されることが一般的であるが，フーリエ変換の結果得られるデータは複素数である．

フーリエ変換により得られたデータ列の横軸のスケール（周波数）を求めるには fftfreq 関数を用いる．

書き方： np.fft.fftfreq(**データ個数**, d=**時間領域の最大値**)

データ個数は fft 関数に与えたデータ列の長さであり，そのデータの最終要素の時刻を**時間領域の最大値**として与える．fftfreq 関数は周波数スケールのデータ列を返す．この値を横軸として，fft 関数で得られた周波数成分のデータ列を縦軸としてプロットすると，周波数領域のプロットができる．ただし，フーリエ変換の結果として得られるデータ

は複素数であることに注意すること．

3.1.8.2 周波数領域から時間領域への変換：フーリエ変逆換

周波数領域のデータ列を時間領域のデータ列に変換（**フーリエ変逆換**）するには ifft 関数を使用する．

書き方： np.fft.ifft(**周波数領域のデータ列**)

フーリエ変換，フーリエ逆変換の処理を行うサンプルプログラムを npfft01.py に示す．またこのプログラムに従って各種のプロットの機能についても解説する．

プログラム：npfft01.py

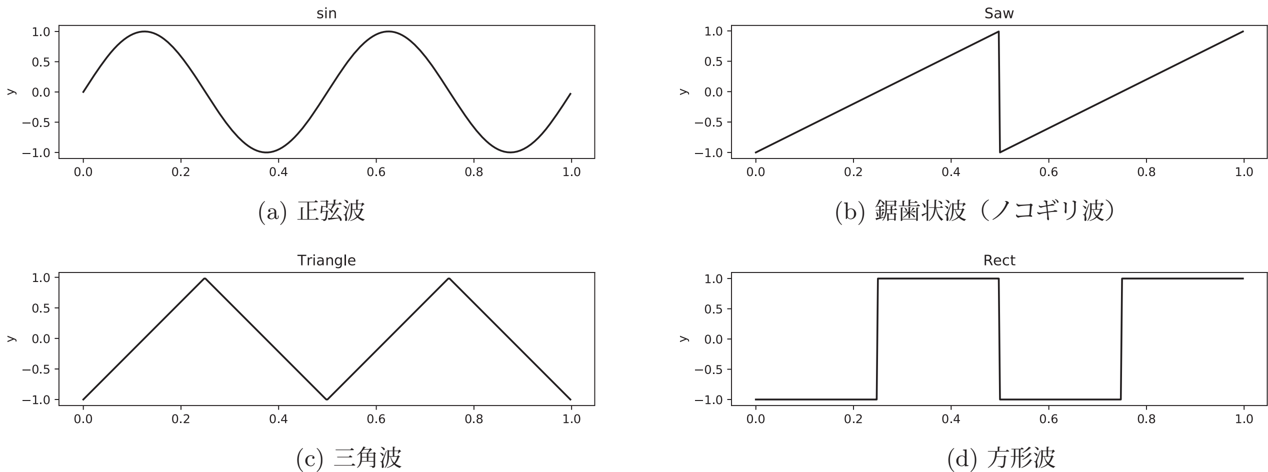
```
1  # coding: utf-8
2  # モジュールの読み込み
3  import numpy as np
4  import matplotlib.pyplot as plt
5  import pylab
6
7  ##########################################################
8  #   波形データの生成                                     #
9  ##########################################################
10 #----- 時間軸データ -----
11 d_x = np.array( [i/400.0 for i in range(400)] )
12
13 #----- 正弦波 -----
14 d_sin = np.sin( 4.0 * np.pi * d_x )
15
16 # 波形のプロット
17 pylab.figure(figsize=(8,2))
18 plt.plot(d_x, d_sin, lw=1.5, color='black')
19 plt.xlabel('time')  # 横軸ラベル
20 plt.ylabel('y')     # 縦軸ラベル
21 plt.title('sin')
22 plt.show()
23
24 #----- 鋸歯状波（ノコギリ波）-----
25 d_saw = np.array( [i/100.0-1.0 for i in range(200)]*2 )
26
27 # 波形のプロット
28 pylab.figure(figsize=(8,2))
29 plt.plot(d_x, d_saw, lw=1.5, color='black')
30 plt.xlabel('time')  # 横軸ラベル
31 plt.ylabel('y')     # 縦軸ラベル
32 plt.title('Saw')
33 plt.show()
34
35 #----- 三角波 -----
36 tmp = [i/50.0-1.0 for i in range(100)]
37 d_tri = np.array( (tmp+tmp[::-1])*2 )
38
39 # 波形のプロット
40 pylab.figure(figsize=(8,2))
41 plt.plot(d_x, d_tri, lw=1.5, color='black')
42 plt.xlabel('time')  # 横軸ラベル
43 plt.ylabel('y')     # 縦軸ラベル
44 plt.title('Triangle')
45 plt.show()
46
47 #----- 方形波 -----
48 d_rct = np.array( ([-1.0]*100+[1.0]*100)*2 )
49
50 # 波形のプロット
51 pylab.figure(figsize=(8,2))
52 plt.plot(d_x, d_rct, lw=1.5, color='black')
53 plt.xlabel('time')  # 横軸ラベル
54 plt.ylabel('y')     # 縦軸ラベル
55 plt.title('Rect')
```

```
56  plt.show()
57
58  #########################################################
59  #   フーリエ変換                                          #
60  #########################################################
61  # 周波数軸データ
62  n = len(d_x)             # データ長
63  frq = n * np.fft.fftfreq(n,d=1.0)
64
65  #----- 正弦波の解析 -----
66  f_sin = np.fft.fft(d_sin)
67  f_sin_n = np.sqrt( f_sin.real**2 + f_sin.imag**2 )
68
69  # パワースペクトルのプロット
70  pylab.figure(figsize=(8,2))
71  plt.bar(frq, f_sin_n, color='black')
72  plt.xlim(-8,8)
73  plt.xlabel('Frequency (Hz)')    # 横軸ラベル
74  plt.ylabel('Power')     # 縦軸ラベル
75  plt.title('Power spectrum of sin')
76  plt.show()
77
78  #----- 鋸歯状波(ノコギリ波)の解析 -----
79  f_saw = np.fft.fft(d_saw)
80  f_saw_n = np.sqrt( f_saw.real**2 + f_saw.imag**2 )
81
82  # パワースペクトルのプロット
83  pylab.figure(figsize=(8,2))
84  plt.bar(frq, f_saw_n, color='black')
85  plt.xlim(-60,60)
86  plt.xlabel('Frequency (Hz)')    # 横軸ラベル
87  plt.ylabel('Power')     # 縦軸ラベル
88  plt.title('Power spectrum of Saw')
89  plt.show()
90
91  #----- 三角波の解析 -----
92  f_tri = np.fft.fft(d_tri)
93  f_tri_n = np.sqrt( f_tri.real**2 + f_tri.imag**2 )
94
95  # パワースペクトルのプロット
96  pylab.figure(figsize=(8,2))
97  plt.bar(frq, f_tri_n, color='black')
98  plt.xlim(-30,30)
99  plt.xlabel('Frequency (Hz)')    # 横軸ラベル
100 plt.ylabel('Power')     # 縦軸ラベル
101 plt.title('Power spectrum of Triangle')
102 plt.show()
103
104 #----- 方形波の解析 -----
105 f_rct = np.fft.fft(d_rct)
106 f_rct_n = np.sqrt( f_rct.real**2 + f_rct.imag**2 )
107
108 # パワースペクトルのプロット
109 pylab.figure(figsize=(8,2))
110 plt.bar(frq, f_rct_n, color='black')
111 plt.xlim(-60,60)
112 plt.xlabel('Frequency (Hz)')    # 横軸ラベル
113 plt.ylabel('Power')     # 縦軸ラベル
114 plt.title('Power spectrum of Rect')
115 plt.show()
116
117 #########################################################
118 #   フーリエ逆変換                                        #
119 #########################################################
120 #----- 方形波の逆変換 -----
121 i_rct = np.fft.ifft(f_rct)
122
123 # 波形のプロット
124 pylab.figure(figsize=(8,2))
```

```
125 plt.plot(d_x, i_rct, lw=1.5, color='black')
126 plt.xlabel('time')  # 横軸ラベル
127 plt.ylabel('y')     # 縦軸ラベル
128 plt.title('Rect (Fourier inverse transform)')
129 plt.show()
```

解説：

11〜56 行目で，各種の波形データ（正弦波，鋸歯状波，三角波，方形波）を生成してそれらをプロットしている．この部分により表示されるグラフを図 21 に示す．

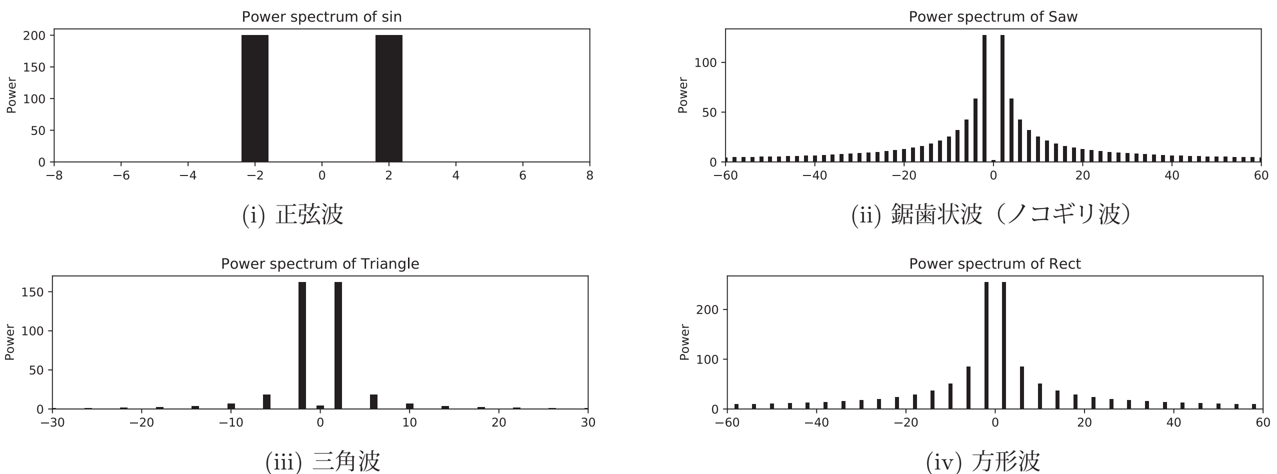


(a) 正弦波　(b) 鋸歯状波（ノコギリ波）

(c) 三角波　(d) 方形波

図 21: 波形データ（時間の関数）

これらの波形データは振幅 ±1.0 で周波数は 2Hz である．すなわち，最大の時刻は 1 で，その間に 2 回のサイクルを繰り返すものである．これらの波形データをフーリエ変換してプロットしているのが 62〜115 行目の部分である．プロットに必要となる横軸（周波数スケール）のデータは 62〜63 行目で生成している．この部分の実行により得られる周波数領域のプロット（パワースペクトル[7]）を図 22 に示す．



(i) 正弦波　(ii) 鋸歯状波（ノコギリ波）

(iii) 三角波　(iv) 方形波

図 22: パワースペクトル（周波数の関数）

この結果，各波形とも周波数は 2Hz であるので，主たる成分である 2Hz の成分が最も強く表示されている．当然であるが正弦波はそれ自身が 1 つの周波数成分であるので 2Hz のみの成分から成ることがわかる．フーリエ変換の結果として得られる周波数の成分は，正負の両方の周波数領域にわたる形となる．

[7]各周波数成分の「強度」を知るために，複素数である周波数成分のノルムを使って表示したもの．

この処理で得られた方形波の周波数領域のデータをフーリエ逆変換により再度時間領域のデータ列に戻している．（121 行目）　それをプロットしているのが 124〜129 行目の部分であり，プロット結果を図 23 に示す．

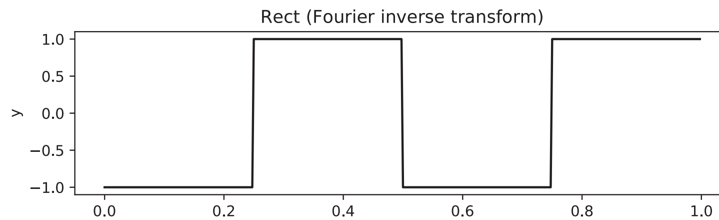


図 23: フーリエ逆変換で再構成した方形波

3.1.9　データの可視化：棒グラフ，グラフのアスペクトについて

3.1.9.1　棒グラフのプロット

棒グラフをプロットするには matplotlib の bar 関数を使用する．（プログラム npfft01.py の 71,84,97,110 行目）

3.1.9.2　プロットのアスペクト比の指定

matplotlib によるプロットにおいて，作図時のアスペクト比（縦横比）を指定するには pylab モジュールを読み込んで（プログラム npfft01.py の 5 行目）関数 figure を使用する．

書き方：　pylab.figure(figsize=(**横**, **縦**))

3.1.10　フーリエ変換を使用する際の注意

フーリエ変換で時間軸上の波形を周波数成分に変換する際には注意しなければならないことがある．離散的で有限な長さのデータをフーリエ変換すると，**与えられたデータ列が繰り返されるものと見做した上での周波数成分**が得られる．先のサンプルプログラムは，2Hz の単純な波形がそのまま繰り返されるという前提の信号解析であり，実際の信号解析ではそのようなことはあり得ない．すなわち，波形データを無思慮に切り出してフーリエ変換を実行すると，元の波形データには含まれない周波数成分が現れてしまう．これは，データ列の時間軸の両端を繋いだ形の波形が繰り返されていると見做した結果である．このような問題を解決するには，波形データを適切に切り出すための**窓関数**[8]を**推定**しながら，それを適用する形でフーリエ変換を実行しなければならない．

信号解析のための具体的な工夫に関しては本書の範囲を超えるため割愛する．

3.1.11　行列の計算

NumPy は，数値（複素数）から成る行列とベクトルを扱うための各種の機能を提供している．

3.1.11.1　行列の和と積

行列の和を求めるには通常の加算演算子 '+' が使用できる．

例. 行列の和

```
>>> a1 = np.array([[1,2],[3,4]])  Enter    ←行列 a1 を生成
>>> a2 = np.array([[5,6],[7,8]])  Enter    ←行列 a2 を生成
>>> a1 + a2  Enter          ←加算の実行
 array([[ 6, 8],            ←処理結果
        [10, 12]])
```

通常の乗算演算子 '*' を用いて行列同士を計算すると，各要素毎の積を要素とする行列が得られる．行列同士の積を

[8]実際の周波数成分のみを適切に取り出すために，適切な時間領域のデータを切り出す関数．

求めるには dot 関数を使用する．

例． 行列の積

```
>>> a1 * a2  Enter          ←'*' の実行
array([[ 5, 12],            ←処理結果
       [21, 32]])
>>> np.dot(a1,a2)  Enter     ←行列の積
array([[19, 22],            ←処理結果
       [43, 50]])
```

3.1.11.2 単位行列，ゼロ行列，他

全ての要素がゼロや 1 であるような行列や，単位行列を生成する例を示す．

例． 全てゼロの配列の生成： zeros 関数

```
>>> np.zeros( 3 )  Enter         ←ゼロが 3 つの配列
array([ 0., 0., 0.])            ←処理結果
>>> np.zeros( (2,3) )  Enter     ← 2 行 3 列のゼロ行列
array([[ 0., 0., 0.],           ←処理結果
       [ 0., 0., 0.]])
```

例． 全て 1 の配列の生成： ones 関数

```
>>> np.ones( 3 )  Enter          ← 1 が 3 つの配列
array([ 1., 1., 1.])            ←処理結果
>>> np.ones( (2,3) )  Enter      ← 2 行 3 列の 1 ばかりの行列
array([[ 1., 1., 1.],           ←処理結果
       [ 1., 1., 1.]])
```

例． 単位行列の生成： identity 関数

```
>>> np.identity( 5 )  Enter      ← 5 × 5 の単位行列
array([[ 1., 0., 0., 0., 0.],    ←処理結果
       [ 0., 1., 0., 0., 0.],
       [ 0., 0., 1., 0., 0.],
       [ 0., 0., 0., 1., 0.],
       [ 0., 0., 0., 0., 1.]])
```

3.1.11.3 行列の要素の編集

既存の行列（配列）の要素を編集する方法について説明する．

例． 指定した値で既存の配列の要素を全て置き換える： fill 関数

```
>>> a = np.ones( (3,4) )  Enter      ← 3 行 4 列の 1 ばかりの行列を生成
>>> a  Enter                         ←内容の確認
array([[ 1., 1., 1., 1.],            ←結果表示
       [ 1., 1., 1., 1.],
       [ 1., 1., 1., 1.]])
>>> a.fill( 3.1416 )  Enter          全ての要素を 3.1416 で満たす
>>> a  Enter                         ←内容の確認
array([[ 3.1416, 3.1416, 3.1416, 3.1416],     ←結果表示」
       [ 3.1416, 3.1416, 3.1416, 3.1416],
       [ 3.1416, 3.1416, 3.1416, 3.1416]])
```

通常のリストの編集と同じように，添字を指定して個別の要素の値を設定することが可能である．例えば疎な行列

（スパース行列：sparse matrix）[9] を生成するには，予め 0 ばかりの要素の行列を生成しておき，添字を指定して要素の値を設定するという方法が良い．

行列を転置するには transpose 関数を使用する．

例. 行列の転置

```
>>> a = np.array([[1,0,-2],[5,2,-3],[2,-1,3]])  Enter     ←3 行 3 列の行列を生成
>>> a  Enter                        ←内容の確認
array([[ 1,  0, -2],                ←結果表示
       [ 5,  2, -3],
       [ 2, -1,  3]])
>>> np.transpose(a)  Enter          ←転置の実行 (1)
array([[ 1,  5,  2],                ←処理結果
       [ 0,  2, -1],
       [-2, -3,  3]])
>>> a.T  Enter                      ←転置の実行 (2)
array([[ 1,  5,  2],                ←処理結果
       [ 0,  2, -1],
       [-2, -3,  3]])
```

この例にあるように，T プロパティからも転置行列を取り出すことができる．

3.1.11.4 行列式と逆行列

例. 行列式を求める（先のつづき）： det 関数

```
>>> np.linalg.det(a)  Enter     ←行列式を求める
20.999999999999989              ←結果表示
```

例. 逆行列を求める（つづき）： inv 関数

```
>>> np.linalg.inv(a)  Enter                             ←逆行列を求める
array([[ 0.14285714, 0.0952381 , 0.19047619],          ←結果表示
       [-1.        , 0.33333333, -0.33333333],
       [-0.42857143, 0.04761905, 0.0952381 ]])
```

3.1.11.5 固有値と固有ベクトル

行列を転置するには linalg パッケージの eig 関数を使用する．

例. 行列の固有値と固有ベクトルを求める（つづき）： eig 関数

```
>>> (e,ev) = np.linalg.eig(a)  Enter     ←固有値と固有ベクトルを求める
>>> e  Enter     ←固有値の配列の内容を確認
array([ 0.82433266+2.03631557j, 0.82433266-2.03631557j, 4.35133469+0.j ])   ←結果表示
>>> ev  Enter     ←固有ベクトルの配列の内容を確認
array([[-0.01760985-0.35786298j, -0.01760985+0.35786298j, -0.21323532+0.j ],   ←結果表示
       [-0.85880917+0.j        , -0.85880917-0.j        , -0.90931800+0.j ],
       [-0.36590772-0.01350281j, -0.36590772+0.01350281j, 0.35731146+0.j ]])
```

固有値は複素数のスカラーであり，与えられた行列が固有値を持つ場合はそれらを配列にして返す．個々の固有値にはそれぞれ対応する固有ベクトルがあり，それを配列にしたものを返す．eig 関数は固有値と固有ベクトルを次のような形式のタプルで返す．

(固有値の列, 固有ベクトルの配列)

注)

eig 関数が返したタプルにおいて，固有値と固有ベクトルは順番に対応する．すなわち，**固有値の列**の第 0 番目の

[9] 大部分の要素が 0 であるような行列．

固有値に対する固有ベクトルは，**固有ベクトルの配列**の第 0 番目の列である．従って，指定した固有値に対応する固有ベクトルを取り出すには，**固有ベクトルの配列**を転置した後に添え字を指定して取り出す必要がある．

3.1.11.6 その他

■ 行列のランク

行列のランクを求めるには linalg パッケージの matrix_rank 関数を使用する．

例． 行列のランクを求める： matrix_rank 関数

```
>>> a1 = np.array([[2,-1,5],[-3,0,7],[9,4,-6]])  Enter   ←行列 a1 を生成
>>> a1  Enter                ←内容の確認
array([[ 2, -1, 5],          ←結果表示
       [-3, 0, 7],
       [ 9, 4, -6]])
>>> np.linalg.matrix_rank( a1 )  Enter     ←ランクの算出
3        ←結果表示
>>> a2 = np.array([[2,-1,5],[-3,0,7],[-4,2,-10]])  Enter   ←行列 a2 を生成
>>> a2  Enter                ←内容の確認
array([[ 2, -1, 5],          ←結果表示
       [ -3, 0, 7],
       [ -4, 2, -10]])
>>> np.linalg.matrix_rank( a2 )  Enter     ←ランクの算出
2        ←結果表示
```

■ 複素共役行列

複素共役行列を求めるには conjugate 関数を使用する．

例． 複素共役行列を求める： conjugate 関数

```
>>> ca = np.array([[1-2j,0],[0,-3+1j]])  Enter    ←行列 ca を生成
>>> ca  Enter                ←内容の確認
array([[ 1.-2.j, 0.+0.j],          ←結果表示
       [ 0.+0.j, -3.+1.j]])
>>> np.conjugate(ca)  Enter              ←複素共役行列を求める
array([[ 1.+2.j, 0.-0.j],          ←結果表示
       [ 0.-0.j, -3.-1.j]])
```

■ エルミート共役行列

エルミート共役行列を求めるには conjugate 関数と転置を組み合わせる．

例． エルミート共役行列を求める

```
>>> ca = np.array([[1-2j,-5+7j],[4-6j,-3+1j]])  Enter   ←行列 ca を生成
>>> ca  Enter                ←内容の確認
array([[ 1.-2.j, -5.+7.j],          ←結果表示
       [ 4.-6.j, -3.+1.j]])
>>> np.conjugate( ca.T )  Enter    ←エルミート共役行列を求める
array([[ 1.+2.j, 4.+6.j],          ←結果表示
       [-5.-7.j, -3.-1.j]])
```

■ ベクトルのノルム

ベクトルのノルムを求めるには linalg パッケージの norm 関数を使用する．

例. ベクトルのノルムを求める： norm 関数

```
>>> v = np.array([1,1])   Enter      ←ベクトル v を生成
>>> np.linalg.norm(v)   Enter      ←ノルム（長さ）を求める
1.4142135623730951          ←結果表示
```

3.1.12 入出力

実際のデータ解析においては扱うデータのサイズが大きい場合が多く，適宜ファイルに出力保存したり，それを読み込んで処理をするという流れになる．ここでは，NumPy で扱うデータをファイルに保存する，あるいはファイルから読み込む方法について説明する．

3.1.12.1 配列オブジェクトのファイル I/O

配列オブジェクトのファイル I/O において 3 種類の形式を選ぶことができる．1 つはテキスト形式であり，CSV や TSV などの形式に沿った入出力により，他の処理系とのデータ連携が容易になる．

■ テキストファイルへの保存

random パッケージの rand 関数を使用して大きな乱数表を作成して，それを CSV 形式[10] のテキストファイルとして保存する例を示す．

例. 100 行 10 列の一様乱数を CSV データとして保存する

```
>>> rnd = np.random.rand( 100, 10 )   Enter          ←乱数表を生成
>>> np.savetxt('random.csv',rnd,delimiter=',')   Enter     ← CSV データとして保存
>>>
```

この例では savetxt 関数を使用して配列オブジェクトをファイルに保存している．この関数の使用方法は次の通りである．

書き方： np.savetxt(**出力先ファイル名, 配列オブジェクト，**delimiter=**区切り文字**)

この方法で保存したテキストファイル 'random.csv' を Microsofot 社の表計算ソフト Excel で開いた例を図 24 に示す．

	A	B	C	D	E	F	G	
1	7.0586177E-01	9.8161076E-01	9.6601103E-01	8.9160988E-02	1.7624552E-01	6.6354880E-01	4.2147287E-01	6.5005
2	7.0944301E-01	4.8800316E-01	4.3856332E-01	5.9453042E-01	1.1948329E-01	8.5148735E-01	4.1622966E-01	5.8108
3	7.5425629E-01	1.9923005E-01	8.0172316E-01	2.4649451E-01	3.6265127E-01	1.1033137E-01	7.9105902E-01	7.8114
4	5.9106786E-02	4.0207431E-01	7.8047742E-01	1.9629748E-01	4.0758277E-01	9.6285520E-01	3.7309372E-01	9.8884
5	3.9081352E-01	6.8698652E-01	4.5855603E-01	1.1644451E-01	8.6609573E-01	2.1652586E-01	3.8099760E-01	9.3932
6	1.6489224E-01	6.9020775E-01	4.6385445E-01	9.7552768E-01	3.8880095E-02	2.9494056E-03	3.2352230E-02	4.6105
7	7.2729904E-01	8.4540898E-01	6.3104417E-01	4.0700006E-01	2.9498669E-01	3.2818641E-01	2.6988482E-01	2.6890
8	5.9819136E-01	7.6903394E-01	4.3795073E-01	5.2329515E-01	8.2189522E-01	9.7185035E-01	6.5961091E-01	2.4032
9	4.2947333E-01	6.7136186E-01	3.5284120E-01	6.2424114E-02	7.8685739E-01	9.8900315E-01	4.0814013E-02	2.2256
10	8.2624744E-01	2.6035833E-01	4.2931582E-01	3.6102405E-01	9.2454230E-01	2.5976500E-01	7.5668125E-01	3.0273
11	7.4204853E-01	5.3672794E-01	4.1130967E-01	1.3676242E-03	2.0832216E-01	3.8416228E-01	3.8210064E-01	3.9788
12	9.5327874E-01	9.6415344E-01	4.4619234E-01	3.9782717E-01	8.6448164E-01	9.2265489E-01	9.8670874E-01	5.3431
13	7.8811375E-01	4.8073127E-01	5.0499849E-01	4.8085178E-01	5.0479560E-01	9.9246350E-01	7.6266953E-01	7.3231
14	2.9278840E-01	5.1439589E-01	3.2649005E-01	1.7486252E-01	3.1071240E-01	2.6462944E-01	9.1237733E-01	7.0149
15	5.7602735E-01	4.4285908E-01	6.0423864E-01	1.3168516E-01	2.0861528E-02	9.3103036E-01	7.1858372E-01	7.3882
16	2.9736532E-01	2.9200560E-01	1.2145600E-01	2.3980909E-01	2.1420448E-01	2.4229163E-01	3.0452012E-01	9.8704

図 24: Microsoft 社の表計算ソフト Excel でファイルを開いたところ

■ テキストファイルからの読み込み

関数 loadtxt を使用するとテキストファイルからデータを読み込むことができる．

[10] コンマ ',' で区切られたテキスト形式の表データである．詳しくは IETF が定める技術仕様 RFC 4180 を参照のこと．

例. CSV 形式のデータを読み込む（つづき）

```
>>> rnd2 = np.loadtxt('random.csv',delimiter=',')   Enter   ← CSV データの読み込み
>>> rnd == rnd2   Enter   ←元の配列との比較を試みる
 array([[ True, True, True, True, True, True, True, True, True, True],  ←比較結果
        [ True, True, True, True, True, True, True, True, True, True],
                                 ⋮
                             （途中省略）
                                 ⋮
        [ True, True, True, True, True, True, True, True, True, True]], dtype=bool)
```

この実行結果から，保存したデータを再度読み込んだものが，元の配列と同じ内容であることが検証できた.（行列同士の比較に関しては「3.1.13 行列の比較」を参照のこと）

loadtxt 関数の使用方法は次の通りである.

書き方 (1)： np.loadtxt(**入力元ファイル名**，delimiter=**区切り文字**)

この関数は，読み込んだデータを配列オブジェクトとして返す.

※ 見出し行のある CSV 形式データを読み込む際の注意

CSV 形式データは 1 行目が見出し行となっていることがあり，そのようなデータを先の方法で読み込むとエラーが発生する．見出し行には NumPy の配列として扱えない型（文字列など）のデータが使用されることが多いというだけでなく，そもそも計算の対象とするデータではないので，見出し行は読み込みの際に無視する必要がある．次に示すように，loadtxt 関数のキーワード引数 skiprows= に整数値を指定すると，入力ファイルの先頭から指定した行数の読み込みを無視することができる.

書き方 (2)： np.loadtxt(**入力元ファイル名**，delimiter=**区切り文字**, skiprows=**スキップする行数**)

入力データの先頭 1 行が見出し行となっている場合は skiprows=1 を指定することで，正しく入力処理が行われる.

■ バイナリファイルへの I/O

関数 save, load を使用することでバイナリファイルに対する I/O ができる.

例. バイナリファイルに対する I/O（つづき）

```
>>> np.save('random.npy',rnd)   Enter   ←配列 rnd をファイル 'random.npy' に保存
>>> rnd2 = np.load('random.npy')   Enter   ←ファイル 'random.npy' から読み込み
>>> rnd == rnd2   Enter   ←元の配列との比較を試みる
 array([[ True, True, True, True, True, True, True, True, True, True],  ←比較結果
        [ True, True, True, True, True, True, True, True, True, True],
                                 ⋮
                             （途中省略）
                                 ⋮
        [ True, True, True, True, True, True, True, True, True, True]], dtype=bool)
```

この例でも保存と読み込みが正常に行われたことがわかる.（行列同士の比較に関しては「3.1.13 行列の比較」を参照のこと）

save と load の使用方法：

np.save(**出力先ファイル名, 配列オブジェクト**)
np.load(**入力元ファイル名**)

load 関数は読み込んだデータを配列オブジェクトとして返す.

■ データの圧縮保存と読み込み

サイズの大きな配列オブジェクトを複数取り扱う場合，それら配列をまとめて圧縮処理 [11] して，1 つのファイルとして保存することができる.

[11] ZIP フォーマットで圧縮している．複数のデータをまとめて管理する**アーカイブ**の機能も備えている.

例. 配列オブジェクトの圧縮保存と読み込み（つづき）

```
>>> np.savez('random.npz',name1=rnd)  Enter   ←配列 rnd をファイル 'random.npz' に圧縮保存
>>> arc = np.load('random.npz')  Enter   ←ファイル 'random.npy' から読み込み
>>> arc['name1']==rnd  Enter   ←元の配列との比較を試みる
array([[ True, True, True, True, True, True, True, True, True, True],   ←比較結果
       [ True, True, True, True, True, True, True, True, True, True],
                          :
                      （途中省略）
                          :
       [ True, True, True, True, True, True, True, True, True, True]], dtype=bool)
```

この例でも保存と読み込みが正常に行われたことがわかる.（行列同士の比較に関しては「3.1.13 行列の比較」を参照のこと）

圧縮保存の使用方法：

np.savez(**出力先ファイル名, データ名=配列オブジェクト, ⋯**)
np.load(**入力元ファイル名**)

圧縮データを load 関数で読み込むと NpzFile オブジェクトが返される．この NpzFile オブジェクトに添え字として [**データ名**] を付けることで，配列オブジェクトを参照することができる．先の例ではデータ名として 'name1' というものを与えて配列 rnd を 'random.npz' というファイルに保存している．それを読み込んで arc という NpzFile オブジェクトを得た後，データ名を付加して arc['name1'] として配列オブジェクトを参照している．このような方法で，複数の配列オブジェクトに異なるデータ名を付けて圧縮保存することができる．

3.1.13 行列の比較

先に示した例のように，2 つの行列（配列）の要素が全て等しいかどうかを調べる場合，比較演算子 '==' を用いると，各要素の比較結果である真理値の行列が得られる．この比較結果の行列（真理値の行列）に対して all メソッドを使用すると全て True かどうかを更に判定することができる．

例. 行列の要素が全て等しいかどうかの判定（つづき）

```
>>> (rnd == rnd2).all()  Enter   ←全ての要素が True か判定
True        全ての要素が True である
```

all の他にも,「少なくとも 1 つの要素が True である」ことを検査する any メソッドもある.

例. any メソッドを使用した「少なくとも 1 つの要素が True である」判定

```
>>> a1 = np.array([[1,2],[3,4]])  Enter   ←配列 a1 を生成
>>> a2 = np.array([[1,2],[5,4]])  Enter   ←配列 a21 を生成
>>> a3 = np.array([[-1,-2],[-5,-4]])  Enter   ←配列 a3 を生成
>>> (a1==a2).all()  Enter   ← a1 と a2 の要素が全て等しいか検査
False   ←異なるものがあることがわかる
>>> (a1==a2).any()  Enter   ← a1 と a2 の要素で互いに等しいものが 1 つでもあるかを検査
True   ←等しい要素があることがわかる
>>> (a2==a3).any()  Enter   ← a2 と a3 の要素で互いに等しいものが 1 つでもあるかを検査
False   ←等しい要素がないことがわかる
```

3.1.14 配列を処理するユーザ定義関数の実装について

NumPy は配列に対する処理を実行する便利な関数やメソッドを提供しているが，ユーザ独自の関数の実装が求められることも多い．素朴な方法としては，NumPy の配列の要素 1 つ 1 つに対して施す処理を for などの文で繰り返し実行するというものが挙げられる．ただし for 文による繰り返し処理は実行時間が大きくなることも多く，またプ

ログラムの可読性が低下する原因にもなる．ここでは，ユーザ定義関数の実装において実行時間と可読性の問題を小さくするための方法について紹介する．

■ for, map, vectorize の比較

次のような関数を考える．

$$\sin(x)+\frac{1}{2}\sin(2x)+\frac{1}{3}\sin(3x)+\frac{1}{4}\sin(4x)+\frac{1}{5}\sin(5x)+\frac{1}{6}\sin(6x)+\frac{1}{7}\sin(7x)+\frac{1}{8}\sin(8x)$$

これはノコギリ波（鋸歯状波）の波形を近似的に表現した関数である．この関数を定義域となる配列データに対して実行することを考える．次に示すサンプルプログラム npfunctest01.py は $[-20, 20)$ の範囲の 0.0004 刻みのデータ列 x に対して，上に示した関数を fun として定義して適用するものである．

プログラム：npfunctest01.py

```
1  # coding: utf-8
2  # モジュールの読み込み
3  import numpy as np
4  import matplotlib.pyplot as plt
5  import time
6
7  x = np.arange(-20,20,0.0004)     # 定義域データの配列
8  n = len(x)                       # 要素の数
9  print('要素数：',n)
10
11 ##############################################################
12 #   sin(x) + 1/2*sin(2*x) + 1/3*sin(3*x) 求める              #
13 ##############################################################
14
15 # 要素に対する計算を実行する関数
16 def fun(x):
17     return np.sin(x) + 1.0/2.0*np.sin(x*2.0) + \
18             1.0/3.0*np.sin(x*3.0) + 1.0/4.0*np.sin(x*4.0) + \
19             1.0/5.0*np.sin(x*5.0) + 1.0/6.0*np.sin(x*6.0) + \
20             1.0/7.0*np.sin(x*7.0) + 1.0/8.0*np.sin(x*8.0)
21
22 #-----  実行時間テスト(1) ----------------------
23 print('方法1：forによる繰り返し')
24 t1 = time.time()
25 ly1 = []
26 for i in range(n):
27     ly1.append( fun(x[i]) )
28 y1 = np.array(ly1)
29 t = time.time() - t1
30 print(t,'秒\n')
31
32 plt.plot(x,y1)
33 plt.show()
34
35 #-----  実行時間テスト(2) ----------------------
36 print('方法2：map関数による方法')
37 t1 = time.time()
38 ly2 = map(fun,x)
39 y2 = np.array(list(ly2))     # この時に計算が実行される
40 t = time.time() - t1
41 print(t,'秒\n')
42
43 plt.plot(x,y2)
44 plt.show()
45
46 #-----  実行時間テスト(3) ----------------------
47 print('方法3：np.vectorizeによる方法')
48 t1 = time.time()
49 vfun = np.vectorize(fun)     # 関数が「ベクトル化」される
50 y3 = vfun(x)                 # 計算実行
51 t = time.time() - t1
52 print(t,'秒')
53
```

```
54 plt.plot(x,y3)
55 plt.show()
```

解説：

7行目で定義域のデータを生成している．16〜20行目で関数 fun を定義しており，この関数を後の行で定義域の全要素に対して実行する．

25〜28行目では for 文を用いて計算を行い，同様の計算を 38〜39行目では map 関数を使用して実行している．49行目では NumPy の vectorize 関数を使用して，関数 fun を vfun に変換している．この結果得られた vfun は NumPy の配列の全要素に対して一度に処理を施し，結果の値を要素として持つ配列を返す．

このプログラムを実行した結果図 25 のようなグラフが表示される．(3 回表示される)

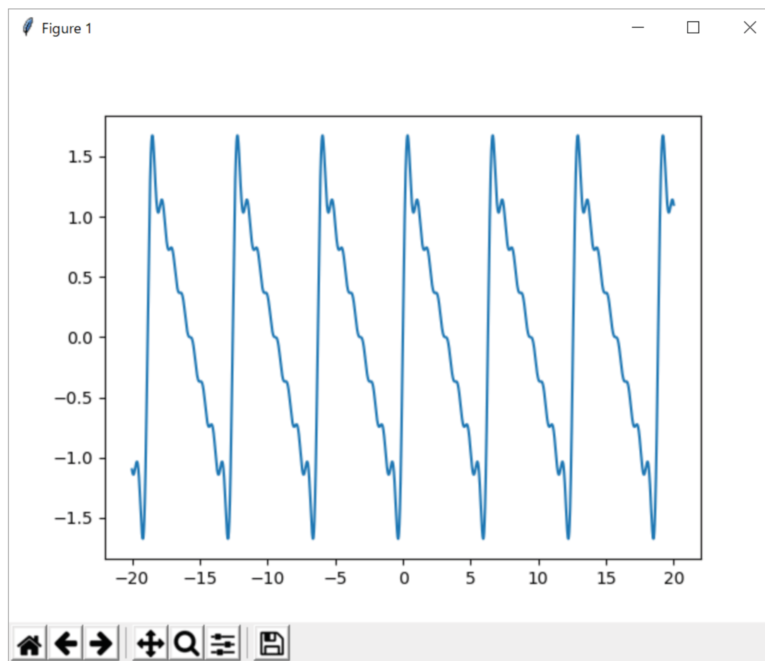


図 25: グラフの表示（3 回表示される）

プログラムの実行に伴って次のように標準出力に出力され，for による方法，map による方法，vectorize による各方法での実行時間がわかる．

例. 実行結果の出力例

要素数： `100000`

方法 1：`for` による繰り返し

`2.2479827404022217` 秒

方法 2：`map` 関数による方法

`1.8539953231811523` 秒

方法 3：`np.vectorize` による方法

`1.2290003299713135` 秒

for による実行が最も時間がかかり，map 関数による実行はそれよりも若干早いことがわかる．NumPy の vectorize 関数で**ベクトル化**された関数による実行が最も早い（for と比較して約 2 倍の速度である）ことがわかる．ただし，計算対象のデータの要素の数や型，実行する関数の定義，さらには計算機環境によって実行時間は変わるので注意が必要である．

3.2 SymPy

SymPy は Python に数式処理機能を提供するパッケージ [12] である．**数式処理システム**（CAS: Computer Algebra System）とは，数式を記号的に処理するシステムであり，代数的な計算を記号のまま実行する．例えば，$a + a$ という式を評価すると $2a$ という形（あるいは $2 * a$ という形）で結果が得られる．

SymPy は，数式やそれを構成する記号を独特のオブジェクトとして扱うため，一般的に知られる数式処理システム [13] と比較すると，記号の扱いに違いがある．本書では他の数式処理システムとの違いを意識しながら，SymPy の使用方法ついて導入的なレベルで説明する．SymPy に関するより詳しい情報についてはインターネットサイト http://www.sympy.org/ を参照のこと．

3.2.1 モジュールの読込みに関する注意

SymPy は多くの関数やメソッドを提供している．そのため，クラス名や関数名などが他のパッケージと重複する可能性が大きくなるので，Python に読み込む際には注意が必要である．例えば，

```
from sympy import *
```

などとしてパッケージを読み込むと，オブジェクトの生成や関数呼び出しにおいてパッケージ名の指定を省略することができて操作が簡便である反面，他のモジュールを読み込んで併用する場合にクラスや関数の名前が衝突するという問題が起こる．Python に SymPy のみを読み込んで，数式処理ツールとして利用する場合は上記の形でモジュールを読み込んでも特に問題はないが，数式処理機能を応用プログラムに組み込んで利用するということを目的とする場合は，名前の衝突に関しては注意を払う必要がある．すなわち，SymPy を読み込む際に，

```
from sympy import 関数名,…
```

などとして，使用する関数名やクラス名を明に指定する方がよい．あるいは，

```
import sympy
```

としてパッケージを読み込んで，関数を呼び出す際に，

```
sympy.関数名(引数)
```

としてパッケージ名を明に指定するのが良い．また，多くの機能が関数としてだけでなく，各クラスのメソッドとしても実装されているので，極力メソッドの形式で数式処理機能を使用するのが安全である．

3.2.2 基礎事項

SymPy による数式の計算には通常の算術記号（+, -, *, /）が使える．また冪乗は '**' である．ただし，他の数式処理システムと違う点として，Python の変数と，数式を構成する記号は全く別のものであるということがある．次の実行例について考える．

誤った操作の例

```
>>> from sympy import *  [Enter]   ←SymPy を読み込み各種名前を大域化する（注：好ましくはない）
>>> x + x  [Enter]                 ←数式の簡単化を試みる
Traceback (most recent call last):        エラーメッセージが表示される
File "<stdin>", line 1, in <module>
NameError:  name 'x' is not defined]
```

エラーメッセージが表示されているが，原因は，未定義の変数同士を加算しようとしたことにある．SymPy で記号的計算を実行するには，計算対象の記号を予め生成しておく必要がある．次の例について考える．

[12] 文献参照："Open source computer algebra systems: SymPy", David Joyner, Ondej ertk, et.al., *ACM Communications in Computer Algebra archive* Volume 45 Issue 3/4, Sep./Dec. 2011, pp.225-234,ACM New York, USA

[13] ウルフラム・リサーチ社の *Mathematica*，ウォータールー大学（カナダ）で開発された Maple，フリーソフトウェア（GNU GPL）の Maxima などが有名である．

正しい操作の例

```
>>> x = Symbol('x')  Enter    ←'x' という記号を生成している
>>> x + x  Enter              ←数式の簡単化を試みる
2*x                           ←正しい結果が得られている
```

この例では，SymPy の記号オブジェクト'x' を生成して，それを Python の変数 x に与えている．すなわち

「変数 x に記号 'x' が代入されている」

と考えると良い．他の数式処理システムでは，変数と代数記号の区別がないが，SymPy の利用においては，このことを常に念頭に置く必要がある．

■ 数式処理のためのオブジェクト

sympy パッケージの Symbol 関数を用いることで数式処理のための記号オブジェクトを生成することができる．

書き方： sympy.Symbol(**文字列**)

文字列として記述されたものを数式記号オブジェクトとして返す．また関数 symbols を使用（先頭が小文字であることに注意）すると，複数の記号を空白で（あるいはコンマで）区切った形の文字列を引数に与えて，複数の数式記号を同時に生成することができる．この場合は戻り値は生成した記号のタプルである．

書き方： sympy.symbols(**文字列**)

数式記号を生成する例

```
>>> import sympy  Enter                  ←パッケージの読み込み
>>> w = sympy.Symbol('w')  Enter         ←数式記号の生成（1 個）
>>> (x,y,z) = sympy.symbols('x y z')  Enter   ←数式記号の生成（3 個）
>>> w + x + x  Enter                     ←記号的計算
w + 2*x                                  ←計算結果
```

■ 数式の簡単化（評価）

数式記号を生成すると，それらを算術記号（+, -, *, /）でつなげることで計算（簡単化，評価）されるが，もっと単純に，文字列として記述した数式を関数 simplify の引数に与えることでも計算結果が得られる．

書き方： sympy.simplify(**文字列**)

例. 数式の簡単化

```
>>> import sympy  Enter                     ←パッケージの読み込み
>>> sympy.simplify('a + b + a + c')  Enter  ←計算
2*a + b + c                                 ←計算結果
```

この例のような処理では，数式記号を明に生成しなくても良い．すなわち，simplify 関数を使用すると，より簡単に（文字列型の式から）数式オブジェクトを生成することができる．

※ 文字列型の式を SymPy の式に変換する，より基本的な関数 sympify が存在する．詳しくは SymPy のインターネットサイトを参照のこと．

数式からのオブジェクトの取り出し

数式オブジェクトに対して atoms メソッドを使用すると，その数式に含まれるオブジェクトを集合の形で取得することができる．

例. 数式から各種オブジェクトを取り出す

```
>>> import sympy  Enter          ←パッケージの読み込み
>>> s = sympy.simplify('a+b+a+2*b+c+pi+f(x)+g(y)')  Enter  ←計算
>>> s  Enter          ←内容の確認
2*a + 3*b + c + f(x) + g(y) + pi       ←計算結果が保持されている
>>> s.atoms(sympy.Symbol)  Enter  ←記号の取り出し
{y, a, b, x, c}          ←代数記号の集合（セット）が得られている
>>> s.atoms(sympy.Number)  Enter  ←数値の取り出し
{2, 3}          ←数値の集合（セット）が得られている
>>> s.atoms(sympy.Function)  Enter  ←関数の取り出し
{g(y), f(x)}          ←関数の集合（セット）が得られている
```

このように atoms メソッドの引数に sympy.Symbol, sympy.Number,sympy.Function を指定することで，各種オブジェクトの集合（セット）が得られるので，これを list 関数でリストに変換すると，数式を構成するオブジェクトのリストとして得ることができる．

■ 式 f(x,y,...) の構造 （頭部と引数列の取り出し）

f(x,y,...) の形をした式の**頭部**と**引数列**はそれぞれ，func, args プロパティから得られる．

例.

```
>>> s = sympy.simplify('f(x,y,z)')  Enter    ←式の生成
>>> s.func  Enter          ←頭部の取り出し
f                          ←頭部
>>> s.args  Enter          ←引数列の取り出し
(x, y, z)                  ←引数の列（タプル）
```

■ 定数

数式の中で使用する定数は以下の通り．

E ： ネイピア数

pi ： 円周率

I ： 虚数単位 （Python の元来の虚数単位 j と混同しないこと）

oo ： 正の無限大 （厳密な意味では数ではない）

例. 定数の使用

```
>>> sympy.simplify('sin(pi)')  Enter  ← sin(π) の計算
0        ←計算結果
>>> sympy.simplify('I * I')  Enter  ← i * i の計算
-1        ←計算結果
```

定数は sympy パッケージのオブジェクトとしても使用できる．すなわち，

sympy.E，sympy.pi，sympy.I，sympy.oo

として参照することができる．

本書では以降，

```
import sympy
```

として SymPy を読み込んで使用することを前提とする．

3.2.3 基本的な数式処理機能

先に説明した simplify に加えて，次に挙げるような基本的な数式処理機能が使用できる．

■ 式の展開

expand メソッドを使用すると数式を展開することができる．

例.

```
>>> s = sympy.simplify('(a+b)**10')  [Enter]   ←数式の生成
>>> s2 = s.expand()  [Enter]     ←展開の処理
>>> s2  [Enter]          ←結果の確認
a**10 + 10*a**9*b + 45*a**8*b**2 + 120*a**7*b**3 + 210*a**6*b**4 + 252*a**5*b**5 +
210*a**4*b**6 + 120*a**3*b**7 + 45*a**2*b**8 + 10*a*b**9 + b**10     ←処理結果
```

同様の処理を　`sympy.expand(s)`　として関数の形式で実行することもできる．

■ 因数分解

factor メソッドを使用すると数式の因数分解ができる．

例. （先の例の続き）

```
>>> s2.factor()  [Enter]      ←因数分解の処理
(a + b)**10     ←処理結果
```

同様の処理を　`sympy.factor(s2)`　として関数の形式で実行することもできる．

■ 指定した記号による整理

collect メソッドを使用すると，指定した記号で整理することができる．

例.

```
>>> s = sympy.simplify('(a+b+x)**2').expand()  [Enter]  ←数式の生成：(a+b+x)^2 の展開
>>> s  [Enter]         ←結果の確認
a**2 + 2*a*b + 2*a*x + b**2 + 2*b*x + x**2     ←処理結果
>>> s.collect('x')  [Enter]       ←記号 x で整理
a**2 + 2*a*b + b**2 + x**2 + x*(2*a + 2*b)     ←処理結果
```

同様の処理を　`sympy.collect(s,'x')`　として関数の形式で実行することもできる．

■ 約分：分数の簡単化 (1)

cancel メソッドを使用すると，分数を約分することができる．複雑な分数

$$\frac{ax^2+2axy+ay^2+bx^2+2bxy+by^2}{acx+acy+adx+ady+bcx+bcy+bdx+bdy}$$

が約分される様子を例示する．

例.

```
>>> s1 = sympy.simplify('a*x**2 + 2*a*x*y + a*y**2 +
            b*x**2 + 2*b*x*y + b*y**2')  [Enter]      ←数式の生成 (分子)
>>> s2 = sympy.simplify('a*c*x + a*c*y + a*d*x + a*d*y +
            b*c*x + b*c*y + b*d*x + b*d*y')  [Enter]      ←数式の生成 (分母)
>>> s = s1 / s2  [Enter]            ←複雑な分数の作成
>>> s  [Enter]          ←結果の確認
(a*x**2 + 2*a*x*y + a*y**2 + b*x**2 + 2*b*x*y + b*y**2)/
  (a*c*x + a*c*y + a*d*x + a*d*y + b*c*x + b*c*y + b*d*x + b*d*y)   ←処理結果 (複雑な分数)
>>> s.cancel()  [Enter]  ←約分の実行
(x + y)/(c + d)     ←処理結果
```

約分した結果が

$$\frac{x+y}{c+d}$$

として得られている．

■ 部分分数

apart メソッドを使用すると，分数を部分分数にすることができる．（ただし，複数の記号オブジェクトからなる分数は処理できない） 分数

$$\frac{5x^3+6x^2+x+4}{x^4+4x^3+4x^2+4x+3}$$

が部分分数に変形される様子を例示する．

例.

```
>>> s1 = sympy.simplify('5*x**3 + 6*x**2 + x + 4')  Enter  ←数式の生成 (分子)
>>> s2 = sympy.simplify('x**4 + 4*x**3 + 4*x**2 + 4*x + 3')  Enter  ←数式の生成 (分母)
>>> s = s1 / s2  Enter          ←1つの長い分数の作成
>>> s  Enter                    ←結果の確認
(5*x**3 + 6*x**2 + x + 4)/(x**4 + 4*x**3 + 4*x**2 + 4*x + 3)  ←処理結果 (1つの長い分数)
>>> s3 = s.apart()  Enter                ←部分分数への変形
>>> s3  Enter                   ←結果の確認
-1/(x**2 + 1) + 4/(x + 3) + 1/(x + 1)   ←処理結果
```

部分分数

$$-\frac{1}{x^2+1}+\frac{4}{x+3}+\frac{1}{x+1}$$

に変換されていることがわかる．

同様の処理を `sympy.apart(s)` として関数の形式で実行することもできる．

■ 分数の簡単化 (2)

ratsimp メソッドを使用すると，分数をまとめることができる．（通分の処理を含む） 先の例（部分分数分解）で得られた結果の s3 に対して ratsimp を適用した例を示す．

例. （先の例の続き）

```
>>> s3.ratsimp()  Enter                ←簡単化の処理
(5*x**3 + 6*x**2 + x + 4)/(x**4 + 4*x**3 + 4*x**2 + 4*x + 3)   ←処理結果
```

元の式に戻り，1つの分数としてまとめられていることがわかる．

同様の処理を `sympy.ratsimp(s3)` として関数の形式で実行することもできる．

■ 代入（記号の置換）

subs メソッドを使用すると，記号（Symbol）を別の記号に置き換えることができる．

例.

```
>>> (a,b,x) = sympy.symbols('a b x')  Enter  ←記号の生成
>>> s = 2*a + 3*b  Enter    ←式の生成
>>> s.subs(a,x)  Enter    ←記号 a を記号 x に置き換える
3*b + 2*x           ←処理結果
>>> s.subs(a+b,x)  Enter    ←式を別のものに置き換えることは…
2*a + 3*b                   ←できない.
```

この例の様に，式を別のものに置き換えることはできない．複数の記号の置換処理には，置換規則を辞書オブジェク

トにして与える.

例. （先の例の続き）

```
>>> s.subs( {a:x,b:1} )  [Enter]    ←記号 a を記号 x に，b を 1 に置き換える
2*x + 3                              ←処理結果
```

■ 数学関数

SymPy では，Python の数値演算で使用できる各種の数学関数と同じ名前のものが概ね使用できるが，対数関数は log の表記を基本とする．ln の表記でも入力できるが，それは log として扱われる．

例. 対数関数

```
>>> sympy.simplify('ln(1)')  [Enter]    ← ln の表記も使用できる
0                                        ←計算結果
>>> sympy.simplify('ln(x)')  [Enter]    ← ln を代数的に記述すると
log(x)                                   ← log の表記に統一される
```

3.2.4 解析学的処理

ここでは，微分と積分を基本とする解析学的な処理機能について説明する．

3.2.4.1 極限

与えられた式の極限を求めるには limit メソッドを使用する．

書き方： 式.limit(対象の変数, 向かう極限)

次に，

$$\lim_{x \to 1} \frac{1}{x-1}$$

を求める例を示す．

例.

```
>>> x = sympy.symbols('x')  [Enter]    ←記号 x の生成
>>> s = 1 / (x - 1)  [Enter]           ←式 1/(x-1) の生成
>>> s.limit(x,1)  [Enter]              ← x → 1 の極限を求める
oo                                      ←処理結果：∞
```

ただしこの例の式では，x の数直線上における「右から左」の極限であり,「左から右」の極限では計算結果が異なる．極限に向かう方向を指定して厳密に計算するには，limit メソッドに 3 番目の引数として '+' もしくは '-' を与える．（次の例を参照）

例. （先の例の続き）

```
>>> s.limit(x,1,'+')  [Enter]    ←「右から左」の極限
oo                                ←処理結果：∞
>>> s.limit(x,1,'-')  [Enter]    ←「左から右」の極限
-oo                               ←処理結果：−∞
```

この例において同様の処理を `sympy.limit(s,x,1,'-')` として関数の形式で実行することもできる．

3.2.4.2 導関数

与えられた式を，**ある変数を定義域として値域を与える関数**として見た場合，diff メソッドを使用して，その変数についての導関数を求める（偏微分する）ことができる．

例.

```
>>> x = sympy.symbols('x')  Enter        ←記号 x の生成
>>> s = sympy.simplify('sin(x)')  Enter  ←式の生成
>>> s.diff(x)  Enter                     ← x についての導関数を求める
cos(x)                                   ←処理結果
```

これは,

$$\frac{d}{dx}\sin(x)$$

を求めた例である.

この例において同様の処理を　`sympy.diff(s,x)`　として関数の形式で実行することもできる.

■ 微分操作の遅延実行

Derivative クラスを使用すると，導関数を求める処理（微分操作）を**遅延実行**することができる.

例. 微分操作の遅延実行

```
>>> s = sympy.simplify('Derivative(sin(x),x)')  Enter  ←式の生成
>>> s  Enter                 ←結果の確認
Derivative(sin(x), x)        ←元のままの式が得られている
>>> s.doit()  Enter          ←式の「実行」
cos(x)                       ←処理結果：導関数が得られている
```

この例のように doit メソッドを使用すると微分操作が実行される．Derivative を用いた導関数の表現は，微分演算の抽象的な表現を数式として記述可能 [14] にする．これにより，微分方程式を記述することが可能となるだけでなく，後の「3.2.9.1 LaTeX」のところで説明する書式変換などにおいても有効な表現手段を与える.

3.2.4.3　原始関数

先の導関数の算出と逆の処理をするには integrate メソッドを使用する.

例.

```
>>> x = sympy.symbols('x')  Enter        ←記号 x の生成
>>> s = sympy.simplify('cos(x)')  Enter  ←式の生成
>>> s.integrate(x)  Enter                ← diff(x) と逆の処理
sin(x)                                   ←処理結果
```

得られた値に定数項が付いていないので，厳密な意味ではこの処理では原始関数を求めたことにはならない．厳密な意味での原始関数を求めるには「3.2.5 各種方程式の求解」で説明する**微分方程式の求解**の方法を参照のこと.

この例において同様の処理を　`sympy.integrate(s,x)`　として関数の形式で実行することもできる.

■ integrate の遅延実行

Integral クラスを使用すると，integrate による処理を**遅延実行**することができる.

例. integrate の遅延実行

```
>>> s = sympy.simplify('Integral(cos(x),x)')  Enter  ←式の生成
>>> s  Enter                 ←結果の確認
Integral(cos(x), x)          ←元のままの式が得られている
>>> s.doit()  Enter          ←式の「実行」
sin(x)                       ←処理結果
```

[14] 関数に対する操作であるので，Derivative は広義の**汎関数**である.

■ **定積分**

integrate を使用して定積分を求めることができる．

例. $\int_1^\infty \frac{1}{x^2}\,dx$ を求める

```
>>> x = sympy.symbols('x')  Enter            ←記号 x の生成
>>> inf = sympy.simplify('oo')  Enter        ←無限大記号の生成
>>> s = sympy.simplify('1/x**2')  Enter      ←関数の生成
>>> s.integrate( (x,1,inf) )  Enter          ←定積分の実行
1                                             ←積分結果
```

遅延実行の形で同様の処理をおこなうこともできる．(次の例参照)

```
>>> s = sympy.simplify('Integral(1/x**2,(x,1,oo))')  Enter  ←遅延実行のための式の生成
>>> s.doit()  Enter                          ←処理の実行
1                                             ←積分結果
```

3.2.4.4 級数展開

与えられた式の級数展開（テイラー展開／マクローリン展開）を得るには series メソッドを使用する．

例. $\exp(x)$ の展開

```
>>> x = sympy.symbols('x')  Enter            ←記号 x の生成
>>> s = sympy.simplify('exp(x)')  Enter      ←式の生成
>>> s.series(x)  Enter                       ←級数展開
1 + x + x**2/2 + x**3/6 + x**4/24 + x**5/120 + O(x**6)   ← 6 番目の項まで計算される
>>> s.series(x,0,8)  Enter                   ←級数展開：0 を起点に 8 番目まで
1 + x + x**2/2 + x**3/6 + x**4/24 + x**5/120 + x**6/720 + x**7/5040 + O(x**8)   ←処理結果
>>> s.series(x,1,6)  Enter                   ←級数展開：1 を起点に 6 番目まで
E + E*(x - 1) + E*(x - 1)**2/2 + E*(x - 1)**3/6 + E*(x - 1)**4/24 +
          E*(x - 1)**5/120 + O((x - 1)**6, (x, 1))      ←処理結果
```

この例において同様の処理を `sympy.series(s,x)` のようにして関数の形式で実行することもできる．

3.2.5 各種方程式の求解

ここでは，各種の方程式の求解のためのメソッドを紹介する．

3.2.5.1 代数方程式の求解

代数方程式の解を求めるには solve 関数を使用する．

1) $f(x) = 0$ の x についての求解

例. $x + 1 = 0$ を x について解く

```
>>> x = sympy.symbols('x')  Enter            ←記号 x の生成
>>> s = sympy.simplify('x+1')  Enter         ←式の生成
>>> sympy.solve(s,x)  Enter                  ← s = 0 を満たす x の求解
[-1]                                          ←解は x = -1 の 1 個
```

2) $f_1(x) = f_2(x)$ の x についての求解

等式を表現するには Eq クラスを使用する．

例. $2x+1=3x-5$ を x について解く

```
>>> x = sympy.symbols('x')  [Enter]          ←記号 x の生成
>>> s = sympy.simplify('Eq(2*x+1,3*x-5)')  [Enter]      ←方程式の生成
>>> sympy.solve(s,x)  [Enter]          ← s を満たす x の求解
[6]                    ←解は x = 6 の 1 個
```

solve 関数は 4 次の代数方程式まで一般的に求解する.

例. 2 次の代数方程式 $ax^2+bx+c=0$ の求解

```
>>> x = sympy.symbols('x')  [Enter]          ←記号 x の生成
>>> s = sympy.simplify('a*x**2 + b*x + c')  [Enter]      ←方程式の生成
>>> sympy.solve(s,x)  [Enter]          ← s = 0 を満たす x の求解
[(-b + sqrt(-4*a*c + b**2))/(2*a), -(b + sqrt(-4*a*c + b**2))/(2*a)]   ←解は 2 個
```

連立方程式の求解

solve 関数に与える方程式と変数をリストにして与えることで連立方程式を解くことができる.

例.

```
>>> eq = sympy.simplify('[a*x + b*y - e, c*x + d*y - f]')  [Enter]   ←方程式の生成
>>> v = sympy.simplify('[x,y]')  [Enter]       ←変数リストの生成
>>> sympy.solve(eq,v)  [Enter]          ←求解
{x:  (-b*f + d*e)/(a*d - b*c), y:  (a*f - c*e)/(a*d - b*c)}     ←解が得られている
```

このように，解は辞書オブジェクトの形で得られる.

3.2.5.2 微分方程式の求解

代数方程式の解を求めるには dsolve 関数を使用する．この関数には，解くべき方程式と，求めるべき解の関数を引数に指定する.

例. $\frac{d}{dx}f(x) - \frac{1}{\sin(x)} = 0$ の $f(x)$ についての求解

```
>>> f = sympy.simplify('f(x)')  [Enter]          ←求めるべき関数 f(x) の生成
>>> eq = sympy.simplify('Derivative(f(x),x)-1/sin(x)')  [Enter]   ←微分方程式の生成
>>> sol = sympy.dsolve(eq,f)  [Enter]        ←求解
>>> sol  [Enter]                ←解の確認
Eq(f(x), C1 + log(cos(x) - 1)/2 - log(cos(x) + 1)/2)     ←解
```

解として，

$$f(x) = C_1 + \frac{1}{2}\log(\cos(x) - 1) - \frac{1}{2}\log(\cos(x) + 1)$$

が得られている.（C_1 は定数項）

dsolve 関数の引数に与える方程式は `Eq(...)` の形でも良い.

不定積分によって原始関数を求める場合は dsolve 関数を使用する.

3.2.5.3 階差方程式の求解

（差分方程式，漸化式）rsolve 関数を使用すると，階差方程式（差分方程式）を解くことができる．これは漸化式の一般化も含む.

例. $f(n+1) - rf(n) = 0$ の $f(n)$ についての求解

```
>>> f = sympy.simplify('f(n)')  Enter        ←求めるべき関数 f(n) の生成
>>> s = sympy.simplify('f(n+1)-r*f(n)')  Enter  ←階差方程式の生成
>>> sympy.rsolve(s,f)  Enter          ←求解
C0*r**n                       ←解
```

解が $C_0 \cdot r^n$ として得られている．

初期値を辞書オブジェクトの形で与えることもできる．

例. （先の続き）初期値 $f(0) = a$ を与える

```
>>> ini = sympy.simplify('{f(0):a}')  Enter  ←初期値の生成
>>> sympy.rsolve(s,f,ini))  Enter        ←求解
a*r**n                       ←解
```

解が $a \cdot r^n$ として得られている．

3.2.6 線形代数

Matrix クラスを使用すると行列が表現できる．例えば，

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

を SymPy のオブジェクトとして生成する例を次に示す．

```
>>> (a,b,c,d) = sympy.symbols('a b c d')  Enter  ←記号の生成
>>> m = sympy.Matrix([[a,b],[c,d]])  Enter  ←行列の生成
>>> sympy.pprint(m)  Enter  ←整形表示
```

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$ ←表示結果

この例の様に pprint 関数を使用すると，行列を整形表示する．この関数は行列以外の数式にも使用することができる．

Matrix オブジェクトの和，差，積には通常の算術記号が使用できる．

■ 行列式

行列オブジェクトに対して det メソッドを使用すると行列式を得ることができる．

例. （先の例の続き）行列式

```
>>> m.det()  Enter  ←行列式を求める
a*d - b*c        ←処理結果
```

■ 逆行列

正則な行列オブジェクトに対して inv メソッドを使用すると逆行列を得ることができる．

例. （先の例の続き）逆行列

>>> im = m.inv() [Enter] ←行列式を求める

>>> sympy.pprint(im) [Enter] ←整形表示

$$\begin{bmatrix} \frac{1}{a} + \frac{bc}{a^2\left(d - \frac{bc}{a}\right)} & -\frac{b}{a\left(d - \frac{bc}{a}\right)} \\ -\frac{c}{a\left(d - \frac{bc}{a}\right)} & \frac{1}{d - \frac{bc}{a}} \end{bmatrix}$$

←表示結果

■ 固有値，固有ベクトル

行列オブジェクトに対して eigenvals メソッドを使用すると，固有値を求めることができる．固有ベクトルも共に求める場合は eigenvects メソッドを使用する．

例. 固有値，固有ベクトルの算出

>>> m = sympy.Matrix([[3,1],[2,4]]) [Enter] ←行列式を求める

>>> sympy.pprint(m) [Enter] ←整形表示

$$\begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix}$$

←表示結果

>>> m.eigenvals() [Enter] ←固有値を求める

{5: 1, 2: 1} ←固有値と**代数的重複度**[15] の辞書オブジェクトが得られる

>>> ev = m.eigenvects() [Enter] ←固有値と固有ベクトルを求める

>>> sympy.pprint(ev) [Enter] ←整形表示

$$\left[\left(2,\quad 1,\quad \left[\begin{bmatrix} -1 \\ 1 \end{bmatrix}\right]\right),\quad \left(5,\quad 1,\quad \left[\begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix}\right]\right)\right]$$

←表示結果

3.2.7 総和

総和を表す式として Sum がある．これは総和を意味する式であり，doit メソッドにより評価される．

書き方： Sum(**式,(変数, 初期値, 終了値**))

例えば Sum(f(k), (k, k0, n)) という式は，

$$\sum_{k=k_0}^{n} f(k)$$

を意味する．

例. 初項 a_1，公差 d の等差数列 $a_1, a_2, \cdots, a_n$ の n 番目までの総和

>>> s = sympy.simplify('Sum(a1+(k-1)*d, (k,1,n))') [Enter] ←一般項 $a_1 + (k-1)d$ の形で与える

>>> sympy.simplify(s.doit()) [Enter] ←評価の実行

n*(2*a1 + d*n - d)/2 ←評価結果

この例でもわかるように，Sum は遅延実行される式であり，doit により実際に評価される．

[15] algebraic multiplicity

3.2.8 数値近似

数式の数値近似値を求めるには evalf メソッドを使用する．

例. 円周率を 70 桁の精度で求める

```
>>> sympy.pi.evalf(70)  Enter  ←数値近似を求める
3.141592653589793238462643383279502884197169399375105820974944592307816
```

当然ではあるが，数値化できない式には無意味である．(次の例参照)

例. 数値近似が得られないもの

```
>>> s = sympy.simplify('a+b')  Enter  ←記号のみからなる数式
>>> s.evalf(70)  Enter  ←数値近似を求めようとしても…
a + b      ←できない．
```

3.2.9 書式の変換出力

SymPy の式を別の言語（MathML,LaTeX など）の表現に変換する方法が用意されている．

3.2.9.1 LaTeX

sympy のオブジェクトとして表現した部分積分の公式は，

```
Eq(Integral(u,v),v*u-Integral(v,u))
```

であるが，これを LaTeX の式に変換する例を次に示す．

例.

```
>>> s = sympy.simplify('Eq(Integral(u,v),v*u-Integral(v,u))')  Enter  ←部分積分の公式
>>> s  Enter  ←内容確認
Eq(Integral(u, v), u*v - Integral(v, u))    ←内容表示
>>> print(sympy.latex(s))  Enter  ← LaTeX の形式に変換して表示
\int u\, dv = u v - \int v\, du    ←内容表示
```

これを LaTeX で処理すると，

$$\int u\,dv = uv - \int v\,du$$

と表示される．このように latex 関数を使用することで LaTeX の表現を生成できる．

3.2.10 グラフのプロット

SymPy は関数のグラフを描く機能（グラフのプロット）を提供する．1 変数の関数のプロット（2 次元のグラフ）を作成するには plot 関数を使用する．

書き方：

plot(**式**, (**変数**, **最小値**, **最大値**))

例. 正弦関数のプロット

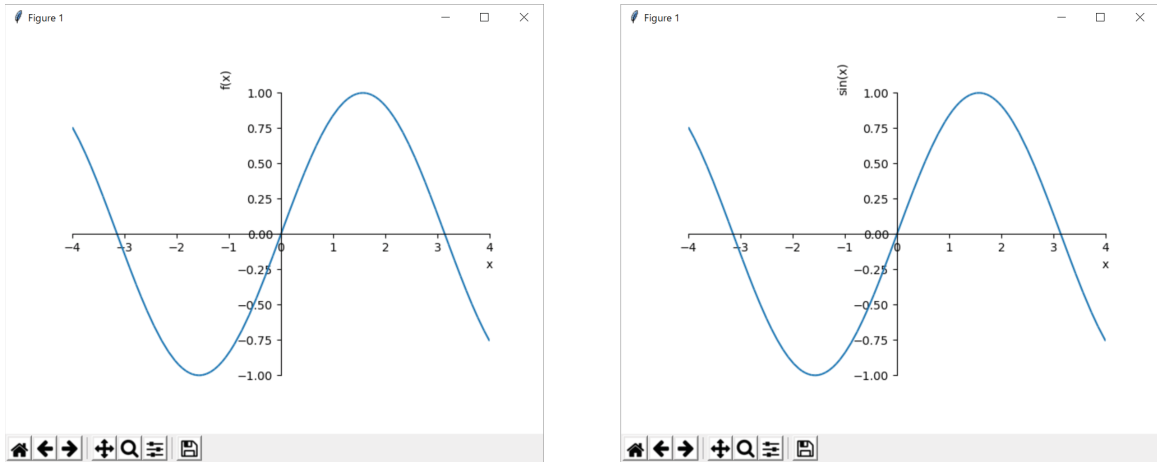
```
>>> f = sympy.sympify('sin(x)')  Enter  ← f に正弦関数の式を与える
>>> sympy.plot(f,('x',-4,4))  Enter  ←プロット開始
<sympy.plotting.plot.Plot object at 0x000002C53BBF1E10>  ←戻り値
```

この処理の結果，図 26 の (a) のようなグラフが表示される．plot 関数にはキーワード引数 xlabel, ylabel を指定することができ，次の例のように実行すると軸のラベルを表示する（図 26 の (b)）ことができる．

例. 正弦関数のプロット（軸ラベル付き）

```
>>> sympy.plot(f,('x',-4,4),xlabel='x',ylabel='sin(x)')
```
Enter ←プロット開始

```
<sympy.plotting.plot.Plot object at 0x000002C53FE00080>
```
←戻り値



(a) 軸ラベル指定なし　　(b) 軸ラベル指定あり

図 26: プロットの表示

2 変数の関数のプロット（3 次元のグラフ）を作成するには plot3d 関数を使用する．この関数は，モジュール sympy.plotting にあるため，使用に際してはこのモジュールをインポートしておく必要がある．

書き方：

plot3d(**式, (変数 1, 最小値, 最大値), (変数 2, 最小値, 最大値)**)

例. $\cos\left(\sqrt{x^2+y^2}\right)$ のプロット

```
>>> from sympy.plotting import plot3d
```
Enter ←モジュールのインポート

```
>>> f = sympy.sympify('cos((x**2+y**2)**(1/2))')
```
Enter ←f に関数の式を与える

```
>>> plot3d(f,('x',-4.5,4.5),('y',-4.5,4.5),xlabel='x',ylabel='y',title='z')
```
Enter

↑プロット開始

```
<sympy.plotting.plot.Plot object at 0x000002C53DAF6828>
```
←戻り値

この処理の結果図 27 のようなグラフが表示される．

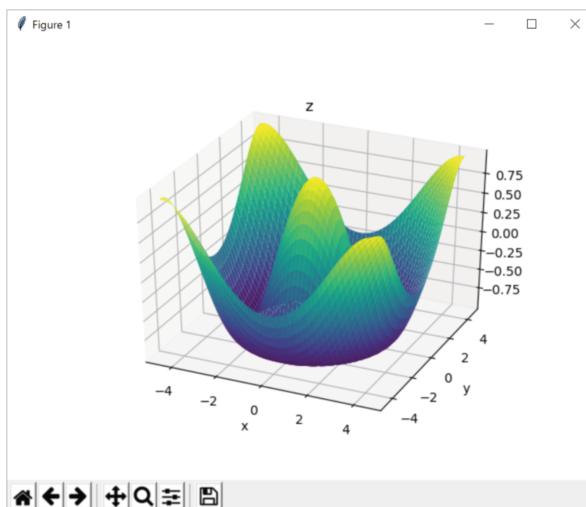


図 27: 3 次元のプロット

4 セキュリティ関連

4.1 hashlib

hashlib は暗号学的な**メッセージダイジェスト**を生成するためのモジュールであり，Python に標準的に提供されている．hashlib が提供するダイジェスト作成アルゴリズムは，MD5, SHA1, SHA224 , SHA256, SHA384, SHA512 である．

メッセージダイジェストを生成する機能は，**パスワード文字列の秘匿化**や，文書の**デジタル署名**の生成に必要となる．このモジュールは使用に先立って，次のようにして必要なモジュールを読み込んでおく必要がある．

```
import hashlib
```

4.1.1 基本的な使用方法

ここでは，パスワード文字列を秘匿化する処理を例に挙げて hashlib の基本的な使用方法について説明する．

例. パスワード文字列 'MyPassword' の秘匿化（MD5 による）

```
>>> m = hashlib.md5(b'MyPassword')  Enter    ←ダイジェスト生成用オブジェクトの生成（MD5）
>>> m.digest()  Enter    ←ダイジェスト生成
b'HP=¥xfdXr¥x0b¥xd5¥xff5¥xc1¥x02¥x06ZR¥xd7'    ←得られたダイジェスト（バイト列）
>>> m.hexdigest()  Enter    ← 16 進数表現でダイジェスト生成
'48503dfd58720bd5ff35c102065a52d7'    ←得られたダイジェスト（文字列）
```

このように，ハッシュ化のアルゴリズム（この例では md5）の名前のコンストラクタの引数に秘匿化したい（ハッシュ化したい）文字列をバイト列形式で与えた後，digest メソッドでダイジェスト（秘匿化されたデータ）を生成する．あるいは hexdigest メソッドを使用すると 16 進数表現の文字列としてダイジェストを得ることができる．

5　プログラムの高速化

Python インタプリタによるプログラムの実行時間は，同様のアルゴリズムを実装した C 言語のプログラムと比べて数十倍～百数十倍程度大きい．このことは計算量の多い処理を行う際に大きな問題となる．ここでは，プログラムの実行時間を小さくするための方法についていくつか紹介する．

5.1　Cython

Python のプログラムの実行時間を短縮するための方法の 1 つに **Cython 処理系** の利用がある．Cython 処理系は公式のインターネットサイト http://cython.org/ から入手できるが，必要となる C 言語処理系 [16] も Cython の導入に先立って準備しておくこと．本書では Cython 処理系について導入的に解説する．Cython の詳細に関しては公式サイトをはじめとするドキュメント [17] を参照のこと．

Cython 処理系は Python の言語仕様を拡張した **Cython 言語**を扱う．Cython 言語で記述されたソースプログラムは一旦 C 言語のソースプログラムに翻訳され，更にそれが C 言語処理系によって実行形式のプログラムに翻訳される．最終的に Cython のプログラムは Python 処理系のためのモジュールとなり，Python のプログラムから呼び出すことができる．Cython 処理系を用いて作成されたモジュールプログラムの実行速度は，通常の Python プログラムの実行に比べて数倍からそれ以上となる．

5.1.1　使用例

サンプルプログラムを挙げ，Cython を用いることでプログラムの実行時間が短縮されることを例示する．次に示す fib.py はフィボナッチ数列を表示するプログラム [18] である．

プログラム：fib.py

```
1  # coding: utf-8
2
3  import time
4
5  # フィボナッチ数列の生成
6  def fib1(n):
7      if n == 0 or n == 1:
8          return( 1 )
9      else:
10         f = fib1(n-1) + fib1(n-2)
11         return( f )
12
13 def fib(n):
14     t1 = time.time()
15     for i in range(n):
16         print( fib1(i) )
17     t = time.time() - t1
18     print(t,'秒')
```

このプログラムをモジュールとして Python 処理系に読み込んで実行した例を次に示す．

[16] 例：Windows 環境では Visual Studio，Apple Macintosh の場合は Xcode.
[17] Cython の日本語ドキュメントサイト：http://omake.accense.com/static/doc-ja/cython/
[18] ここに示すプログラムは，フィボナッチ数の再帰的な関数定義をそのまま実装したものである．フィボナッチ数の生成は**動的計画法**などのアルゴリズムを採用すると大幅に高速化できるが，ここでは大きな実行時間を要する例として，敢えてこの形のプログラムを示す．

```
>>> import fib  Enter    ←モジュールとして読込み
>>> fib.fib(35)  Enter    ←フィボナチ数列表示の開始
1
1
2
3
:
（途中省略）
:
5702887
9227465
8.10300588607788 秒     ←要した時間
```

次に，このプログラムを Cython によって高速化する手順を示す．

【手順】

1． Cython のプログラムとして用意する

先のプログラム fib.py の拡張子を '.pyx' にしたものを用意する．プログラム自体は変更しない．

今回は Cython のプログラム fibC.py として用意する．

2． 翻訳用スクリプトを作成して実行する

Cython プログラムを翻訳するための次のような Python プログラム setup.py を用意する．

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext

setup(
    cmdclass = 'build_ext': build_ext,
    ext_modules = [Extension('fibC', ['fibC.pyx'])]      # ←プログラム名を指定する
)
```

下から 2 行目にあるように，翻訳対象のプログラムの名前を指定する．

この翻訳用スクリプトを，build_ext –inplace という引数とオプションを付けて実行する．

例． 翻訳処理

```
py setup.py build_ext --inplace  Enter    ←翻訳処理の開始
running build_ext
cythoning fibC.pyx to fibC.c
building 'fibC' extension
C:¥Program Files (x86)¥Microsoft Visual Studio 14.0¥VC¥BIN¥x86_amd64¥cl.exe /c
    :
  （途中省略）
    :
fibC.obj : warning LNK4197: エクスポート 'PyInit_fibC' が複数回指定されています。
一番最初の指定を適用します。
    ライブラリ build¥temp.win-amd64-3.6¥Release¥fibC.cp36-win_amd64.lib とオブジェクト
build¥temp.win-amd64-3.6¥Release¥fibC.cp36-win_amd64.exp を作成中
コード生成しています。
コード生成が終了しました。
```

この処理の結果，モジュール fibC が生成される．

3． Python 処理系で実行する

Cython で作成したモジュール fibC を Python 処理系に読み込んで実行する例を次に示す．

```
>>> import fibC [Enter]     ←モジュールとして読込み
>>> fibC.fib(35) [Enter]    ←フィボナチ数列表示の開始
1
1
2
3
:
（途中省略）
:
5702887
9227465
(2.273406505584717, '秒')        ←要した時間
```

先に示した fib.py を Python 処理系で実行する場合と比べて，実行速度が約 3.56 倍になっていることがわかる．

5.1.2 高速化のための調整

Cython は Python のプログラムを C 言語のプログラム変換して翻訳する．このため，Cython のプログラムを記述する際に変数や関数の型を明に宣言することにより，より効率的に C 言語のプログラムに変換されることがある．先の fib.py を元に，変数や関数の型を明に宣言する形にしたプログラム fibC2.pyx を次に示す．

プログラム：fibC2.pyx

```
1  # coding: utf-8
2  
3  import time
4  
5  # フィボナッチ数列の生成
6  cdef int fib1( int n ):      # 型を指定した関数の定義
7      if n == 0 or n == 1:
8          return( 1 )
9      else:
10         f = fib1(n-1) + fib1(n-2)
11         return( f )
12 
13 def fib( int n ):            # 引数の型の指定
14     cdef int    i            # 変数の型の指定
15     t1 = time.time()
16     for i in range(n):
17         print( fib1(i) )
18     t = time.time() - t1
19     print(t,'秒')
```

このプログラムでは，変数や関数の仮引数の型を明に宣言し，外部から呼び出されない関数の型を指定している．型の指定には 'cdef' を用いる．このプログラムを翻訳して実行した結果の例を次に示す．

```
>>> import fibC2 [Enter]     ←モジュールとして読込み
>>> fibC2.fib(35) [Enter]    ←フィボナチ数列表示の開始
1
1
2
3
:
（途中省略）
:
5702887
9227465
(0.1199653148651123, '秒')        ←要した時間
```

はじめに示した fib.py を Python 処理系で実行する場合と比べて，実行速度が約 77.17 倍になっていることがわかる．

5.2 Numba

Numba は LLVM [19] を用いて Python のプログラムを実行するためのモジュールであり，関連の情報はインターネットの公式サイト http://numba.pydata.org/ から入手できる．本書では Numba について導入的に解説する．Numba に関する詳しいことは公式サイトを参照のこと．

5.2.1 基本的な使用方法

Numba は **JIT コンパイラ** [20] を使用して Python のプログラムを実行する．具体的には，Python のソースプログラム中に JIT コンパイラに対する指示を**デコレータ**として記述するという方法を取る．この方法によると，元々Python で記述したプログラムをあまり変更することなく実行時間の短縮が望める．ここでは先に挙げたフィボナッチ数を計算するプログラム fib.py を Numba によって高速化する過程を例示する．

fib.py を Numba で実行するために改訂したものが次に示す fibN1.py である．

プログラム：fibN1.py

```
1  # coding: utf-8
2
3  from numba import jit
4  import time
5
6  # フィボナッチ数列の生成
7  @jit
8  def fib1(n):
9      if n == 0 or n == 1:
10         return( 1 )
11     else:
12         f = fib1(n-1) + fib1(n-2)
13         return( f )
14
15 @jit
16 def fib(n):
17     t1 = time.time()
18     for i in range(n):
19         print( fib1(i) )
20     t = time.time() - t1
21     print(t,'秒')
```

解説：

プログラムの 3 行目で Numba のパッケージを読み込んでいる．7 行目と 15 行目にある '@jit' は直下に記述した関数を JIT コンパイルの対象とすることを指示するデコレータである．

このプログラムを実行した結果の例を次に示す．

[19] C 言語をはじめとする各種言語のためのコンパイラ基盤である．．元々はイリノイ大学（米）で開発され，オープンソースとして公開されている．

[20] 「実行時コンパイラ」（Just-In-Time コンパイラ）．ソフトウェアの実行時にコードのコンパイルを行い実行速度の向上を図るコンパイラのこと．

```
>>> import fibN1  Enter   ←モジュールとして読込み
>>> fibN1.fib(35)  Enter   ←フィボナチ数列表示の開始
1
1
2
3
:
（途中省略）
:
5702887
9227465
0.1561622196289062 秒       ←要した時間
```

はじめに示した fib.py を実行する場合と比べて，実行速度が約 51.9 倍になっていることがわかる．

5.2.2 型指定による高速化

JIT コンパイルする対象の関数の引数や戻り値のデータ型を指定することで，プログラムの実行時間が更に短縮できる場合がある．関数の型指定をする形で先の fibN1.py を改訂したプログラムを fibN2.py に示す．

プログラム：fibN2.py

```
 1  # coding: utf-8
 2
 3  from numba import jit, i8
 4  import time
 5
 6  # フィボナッチ数列の生成
 7  @jit('i8(i8)')
 8  def fib1(n):
 9      if n == 0 or n == 1:
10          return( 1 )
11      else:
12          f = fib1(n-1) + fib1(n-2)
13          return( f )
14
15  @jit('i8(i8)')
16  def fib(n):
17      t1 = time.time()
18      for i in range(n):
19          print( fib1(i) )
20      t = time.time() - t1
21      print(t,'秒')
```

解説：

プログラムの 3 行目で Numba のパッケージを読み込み，7 行目と 15 行目にデコレータを記述している点は先の fibN1.py と共通するが，関数の引数と戻り値の型を ‘i8’ で指定している．これは「8 バイト整数型」を意味する表記である．(詳しくは公式サイトを参照のこと)

※ Numba による高速化の度合いは対象となる関数や使用する他のパッケージに大きく依存する点に留意すること．

5.3 ctypes

ctypes は標準のパッケージであり，C 言語と互換性のあるデータ型を提供し，動的リンク/共有ライブラリ内の関数呼び出しを可能にする．本書では ctypes について導入的に解説する．ctypes に関する詳しい情報は Python の公式サイトのドキュメントなどを参照すること．

5.3.1　C 言語による共有ライブラリ作成の例

C 言語で記述した関数を GNU の C コンパイラによって共有ライブラリにする手順を例示する．フィボナッチ数を求める C 言語のプログラム（関数）を fibCC.c に示す．

プログラム：fibCC.c

```
#include    <stdio.h>

long fib0( n )
long n;
{
    if ( n == 0 ) {
        return( 1 );
    } else if ( n == 1 ) {
        return( 1 );
    } else {
        return( fib0(n-1) + fib0(n-2) );
    }
}

void fib( n )
long n;
{
    long c;

    for ( c = 0; c < n; c++ ) {
        printf("%ld\n",fib0(c));
    }
}
```

このプログラムを gcc コマンドによってコンパイルし，他の言語処理系から使用できる**共有ライブラリ**を作成するには，コンパイルオプション '-shared' を指定する．

例. MinGW 環境下（Windows）での共有ライブラリの作成

```
gcc -O2 -shared -o fibCC.dll fibCC.c
```

この処理が正常に終了すると共有ライブラリ fibCC.dll が作成される．

5.3.2　共有ライブラリ内の関数を呼び出す例

ctypes モジュールの CDLL クラスを使用することで，外部の共有ライブラリを Python 処理系に読み込み，共有ライブラリ内の関数を呼び出すことができる．CDLL クラスのインスタンス（CDLL オブジェクト）を生成する際に，コンストラクタに外部の共有ライブラリを指定する．先の例で作成した共有ライブラリを読み込んで関数を呼び出す Python 側プログラムの例を fibCCpy.py に示す．

プログラム：fibCCpy.py

```
# coding: utf-8

# モジュールの読込み
import ctypes
import time

# 共有ライブラリの読込み
ex = ctypes.CDLL('fibCC.dll')

t1 = time.time()

# 共有ライブラリ中の関数の呼び出し
ex.fib(35)

t = time.time() - t1
print(t,'秒')
```

このプログラム例では，共有ライブラリ fibCC.dll を読み込んで CDLL オブジェクト ex を生成し，それに対するメソッドとして関数名を指定することで，関数 fib を呼び出している．このプログラムを実行した例を次に示す．

```
1
1
2
3
:
（途中省略）
:
5702887
9227465
0.07100534439086914 秒          ←要した時間
```

はじめに示した fib.py を実行する場合と比べて，実行速度が約 114.127 倍になっていることがわかる．

5.3.3　引数と戻り値の扱いについて

実用的な形で Python と C 言語の連携をするためには，相互にデータの受け渡しをする必要がある．ここでは，C 言語で作成した共有ライブラリと Python プログラムとの間で変数の値や配列を受け渡しするための基本的な方法を紹介する．

次に示す C 言語のプログラム ctypesTest01.c は，引数として受け取った値を処理して値を返す 3 つの関数を実装したものである．

C のプログラム：ctypesTest01.c

```
1  #include    <stdio.h>
2  #include    <string.h>
3  
4  /*-------------------------------------------------------*
5   *  整数（int）の受け渡し                                  *
6   *-------------------------------------------------------*/
7  int excgInt( a )
8  int a;
9  {
10     int r;
11 
12     r = 2 * a;
13 
14     printf("C側)\t\t与えられた整数 %d を2倍します: %d\n",a,r);
15     return( r );
16 }
17 
18 /*-------------------------------------------------------*
19  *  浮動小数点数（double）の受け渡し                        *
20  *-------------------------------------------------------*/
21 double  excgDouble( a )
22 double  a;
23 {
24     double  r;
25 
26     r = 2.0 * a;
27 
28     printf("C側)\t\t与えられた浮動小数点数 %lf を2.0倍します: %lf\n",a,r);
29     return( r );
30 }
31 
32 /*-------------------------------------------------------*
33  *  文字列（char*）の受け渡し                              *
34  *-------------------------------------------------------*/
35 char    *excgString( a )
36 char    *a;
37 {
```

```
38        static char r[256];
39
40        strcpy(r,a);
41        strcat(r," <- を返します. ");
42
43        printf("C側)\t\t与えられた文字列 \'%s\' を加工します: %s\n",a,r);
44        return( r );
45    }
```

このプログラムで実装した関数は次の3つである．

- int excgInt(int a) ： 引数 a の値を 2 倍した値を返す関数.
- double excgDouble(double a) ： 引数 a の値を 2.0 倍した値を返す関数.
- char *excgString(char *a) ： 引数 a で示す文字列を加工したもののポインタ返す関数.

これらの関数を呼び出す Python のプログラム ctypesTest01.py を次に示す.

Python 側プログラム：ctypesTest01.py

```
1  # coding: utf-8
2
3  # モジュールの読込み
4  import ctypes
5
6  # 共有ライブラリの読込み
7  ex = ctypes.CDLL('ctypesTest01.dll')
8
9  # 整数の受け渡し
10 r = ex.excgInt( 4 )          # 戻り値は暗黙で int 型
11 print('python側)\t戻り値:',r)
12
13 # 浮動小数点数の受け渡し
14 ex.excgDouble.restype = ctypes.c_double      # 戻り値を double と設定
15 a = ctypes.c_double(2.3)                     # 引数を double に変換
16 r = ex.excgDouble( a )                       # 関数呼び出し
17 print('python側)\t戻り値:',r)
18
19 # 文字列の受け渡し
20 ex.excgString.restype = ctypes.c_char_p      # 戻り値を char* と設定
21 a0 = '元の文字列'.encode('utf-8')             # 送るデータを作成
22 a = ctypes.c_char_p( a0 )                    # それを char* に変換
23 r0 = ex.excgString( a )                      # 関数呼び出し
24 r = r0.decode('utf-8')
25 print('python側)\t戻り値:',r)
```

解説：

整数（int 型）の値を受け渡しする方法は単純であり，10 行目の記述の通りである．整数以外の型の引数や戻り値を扱う場合は，関数呼び出しに先立って準備のための処理が必要となる．

【共有ライブラリの関数の戻り値の扱い】

C 言語で記述した関数の戻り値の型は restype 属性で指定する．今回のプログラム（ctypesTest01.py）の場合，excgDouble 関数は double 型の値を返すので，14 行目にあるように restype として ctypes.c_double を設定している．同様に，文字列のポインタを返す関数 excgString には，20 行目にあるように restype として ctypes.c_char_p を設定している．

● **関数からの戻り値の型の指定**

CDLL オブジェクト. 共有ライブラリの関数名.restype = ctypes. **型指定**

C 言語の型を意味する ctypes の表現（一部）を表 13 に示すが，更に詳しい情報については Python の公式サイトを参照すること．

表 13: ctypes における C 言語のための型の指定：

ctypes での型	C 言語における型	ctypes での型	C 言語における型
c_int	int 型	c_long	long 型
c_uint	unsigned int 型	c_ulong	unsigned long 型
c_short	short int 型	c_ushort	unsigned short int 型
c_float	float 型	c_double	double 型
c_char, c_byte	char 型	c_ubyte	unsigned char 型
c_char_p	char のポインタ型	c_void_p	void のポインタ型

先のプログラム ctypesTest01.py を実行した例を次に示す.

```
C 側)       与えられた整数 4 を 2 倍します: 8
python 側)  戻り値: 8
C 側)       与えられた浮動小数点数 2.300000 を 2.0 倍します: 4.600000
python 側)  戻り値: 4.6
C 側)       与えられた文字列 '元の文字列' を加工します: 元の文字列 <- を返します.
python 側)  戻り値: 元の文字列 <- を返します.
```

【記憶の管理について】

先のプログラム例 ctypesTest01.c，ctypesTest01.py では，文字列の処理結果は関数 excgString 内の static の配列に格納されている．この形の実装では記憶の管理（配列の管理）が C 言語側に委ねられていることになるが，データ構造の管理を全て Python 側で行うには，Python 側のデータ構造のポインタのみを C の関数に渡すという形で実装する必要がある．また，C のプログラム側で malloc 関数などで記憶域を確保するというのも記憶域の解放といった管理を C 側に委ねなければならないので，Python との連携を安全な形で実現するには余り好ましくない.

次に，Python 側のリストを C 言語の配列のポインタとして受け渡す方法について説明する.

5.3.3.1 配列データの受け渡し

Python 側のリストなど，多数の要素を持ったデータ構造を C 言語の関数に渡すには，C 言語のポインタの形に変換する必要がある．また，C 言語の関数で処理した配列を Python 側で受け取るには，やはり配列のポインタとして Python プログラムが受け取り，それをリストなどのデータ構造に変換する必要がある.

ここでは，double 型の配列に格納された値から正弦関数の値を算出して，同じく double 型の配列として作成するプログラムを例に挙げ，C 言語の関数との間で配列を受け渡しする方法を紹介する．定義域の値の配列から正弦関数の値の配列を作成する C 言語のプログラム ctypesTest02.c を次に示す.

C のプログラム：ctypesTest02.c

```
1  #include    <stdio.h>
2  #include    <math.h>
3
4  /*-------------------------------------------------------*
5   *  配列の受け渡し                                        *
6   *-------------------------------------------------------*/
7  int excgArray( a, n, r )
8  double  *a, *r;
9  int     n;
10 {
11     int     x;
12
13     for ( x = 0; x < n; x++ ) {
14         r[x] = sin( a[x] );
```

```
15      }
16
17      printf("C側)\t\t正弦関数の値の列を算出しました. : %d個\n",x);
18
19      return( x );
20  }
```

このプログラムは，定義域のデータを保持する配列のポインタを double *a に受け取って正弦関数の値を算出する関数 excgArray を実装したものである．計算結果も配列に格納するが，そのための配列の記憶域も呼び出し元（Python プログラム側）が用意したものを使用する．(配列のポインタを仮引数 r に受け取る）すなわち，Python 側で用意した配列のポインタを関数 excgArray に渡すので，C 言語プログラムの側では，配列の確保といった記憶の管理はしない．

C 言語の関数 excgArray を呼び出す Python プログラム ctypesTest02.py を次に示す．

Python 側プログラム：ctypesTest02.py

```
1   # coding: utf-8
2
3   # モジュールの読込み
4   import ctypes
5   import matplotlib.pyplot as plt
6
7   # 共有ライブラリの読込み
8   ex = ctypes.CDLL('ctypesTest02.dll')
9
10  # 引数と戻り値の型の設定
11  ex.excgArray.argtype =  [ ctypes.POINTER(ctypes.c_double),
12      ctypes.c_int, ctypes.POINTER(ctypes.c_double) ]     # 引数の型を設定
13  ex.excgArray.restype = ctypes.c_int                      # 戻り値を double と設定
14
15  # 配列データの生成（リスト）
16  ax = [ x/100.0 for x in range(628) ]    # 定義域用
17
18  # リストをCの配列（ポインタ）に変換
19  n = len( ax )                           # 要素の個数（長さ）
20  ar_t = ctypes.c_double * n              # 配列の型の生成
21  ax2 = ctypes.byref( ar_t( *ax ) )       # Cに渡す定義域の配列（ポインタ）
22
23  # 得られた値域のデータを保持する配列の用意
24  ay2  = ar_t()                           # Cに渡す値域の配列（ポインタ）
25
26  # Cの関数の呼び出し
27  n2 = ex.excgArray( ax2, n, ay2 )
28  ay = list(ay2)                          # 値域の配列をリストに変換
29
30  print('Python側)\t戻り値:',n2)
31
32  # 可視化
33  plt.plot(ax,ay)
34  plt.show()
```

解説：

このプログラムでは，リスト ax に $0\sim2\pi$ の範囲の数値を 0.01 刻みで作成し，それを C の関数に渡すことができるポインタ ax2 として変換している．計算結果の正弦関数の値を格納する配列のポインタを ay2 に受け取り，それを Python のリスト ay に変換している．定義域のリスト ax と値域のリスト ay から matplotlib を使って関数のグラフとして可視化している．

プログラムの実行の結果，標準出力に次のように表示される．

```
C 側）        正弦関数の値の列を算出しました．：  628 個
Python 側）   戻り値：  628
```

また，プロットしたグラフが図 28 のような形で表示される．

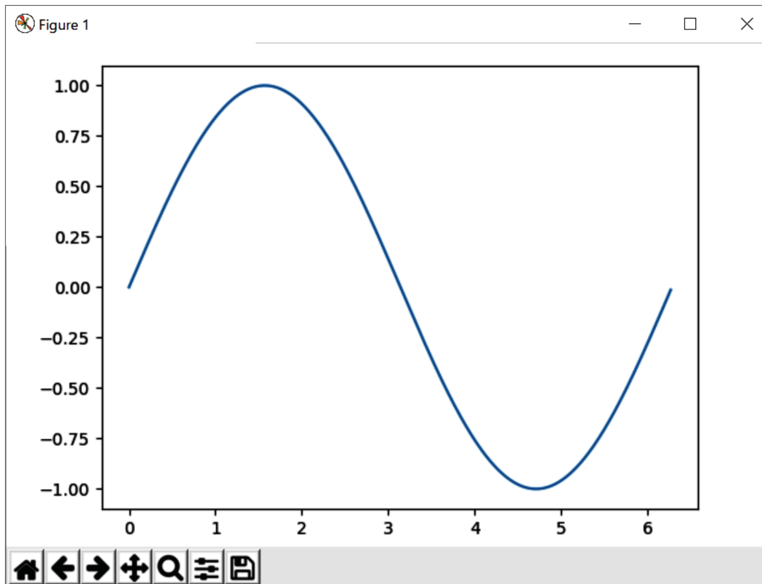


図 28: プロットの表示

【配列のポインタを C の関数の引数に与えるための処理】

呼び出す関数の引数の型を argtype 属性にリスト形式で設定する．

● **関数の引数の型の指定**

CDLL オブジェクト. 共有ライブラリの関数名.argtype ＝ **引数の型指定のリスト**

関数 excgArray は

int excgArray(double *a, int n, double *r)

のような型として定義されているので，引数の型に応じて次のように argtype 属性を設定する．

[ctypes.POINTER(ctypes.c_double), ctypes.c_int, ctypes.POINTER(ctypes.c_double)]

これを行っているのがプログラム ctypesTest02.py の 11～12 行目である．また，配列へのポインタであることを指定するために POINTER(**型指定**) という表現を用いる．

プログラムの 20 行目では，C の関数の引数に与える配列の型をサイズを含めて ar_t として定義している．

● **配列とサイズの型の定義**

型指定 * サイズ

この型を用いて 21 行目では定義域のリスト ax から C に渡す配列 ax2 を生成し，24 行目では，計算結果を格納する配列 ay2 を生成している．

● **C 言語に渡す形式への変換**

byref(**型 (*リスト) **)

27 行目では，ax2, ay2，それにデータの個数 n を引数に与えて C 言語の関数 excgArray を呼び出している．計算結果が格納されている配列 ay2 の内容を 28 行目で Python のリスト ay に変換している．

最後に 33～34 行目でグラフをプロットしている．

6　免責事項

本書に掲載したプログラムリストは全て試作品であり，実用に向けた参考的資料である．また本書の使用に伴って発生した損害の一切の責任を筆者は負わない．

索 引

AKAZE_create, 5
AKAZE 特徴量, 5
all, 55
any, 55
apart, 62
arange, 30
arc, 13
args, 60
array, 29
atoms, 59
Axes3D, 43

bar, 32, 49
BFMatcher, 6
blit, 16, 17
BMP, 8

cancel, 61
circle, 17
Clock, 15
collect, 61
complex128, 29
complex192, 29
complex256, 29
complex64, 29
conjugate, 52
copy, 10
crop, 10
CSV, 53
ctypes, 76
cv2, 1
Cython, 72

Derivative, 64
destroyAllWindows, 3
det, 51, 67
detectAndCompute, 5
diff, 63
digest, 71
distance, 6
doit, 64, 68
dot, 50
Draw, 12
dsolve, 66
dtype, 30

E, 60
eig, 51
eigenvals, 68
eigenvects, 68
ellipse, 13, 17
EPS, 8
Eq, 65
evalf, 69
expand, 11, 61

factor, 61
fadeout, 23
fft, 45
fftfreq, 45
figsize, 49
figure, 43, 49
fill, 13, 50
float128, 29
float16, 29
float32, 29
float64, 29
float96, 29
Font, 18
font_manager, 38
FontProperties, 38
format, 8
func, 60

get, 2, 16
get_busy, 23
get_fonts, 18
get_height, 16, 22
get_pos, 23
get_volume, 23
get_width, 16, 22
GIF, 8
grid, 35
Group, 26

hashlib, 71
hexdigest, 71
hist, 40

I, 60
ICNS, 8

ICO, 8
identity, 50
ifft, 46
IM, 8
image, 8
Image.BICUBIC, 10
Image.BILINEAR, 10
Image.BOX, 10
Image.FLIP_LEFT_RIGHT, 11
Image.FLIP_TOP_BOTTOM, 11
Image.HAMMING, 10
Image.LANCZOS, 10
Image.NEAREST, 10
Image.ROTATE_180, 11
Image.ROTATE_270, 11
Image.ROTATE_90, 11
ImageDraw, 12
imread, 3
imshow, 2
imwrite, 3
in32, 29
info, 8
init, 14
int16, 29
int64, 29
int8, 29
Integral, 64
integrate, 64
inv, 51, 67

JPEG, 8
JPEG2000, 8

KEYDOWN, 22
KEYUP, 22

latex, 69
legend, 33
limit, 63
linalg, 51, 52
line, 13, 17
lines, 17
linspace, 30
load, 16, 17, 23, 54
loadtxt, 53
logspace, 30

match, 6
matplotlib, 29, 32
Matrix, 67
matrix_rank, 52
MD5, 71
merge, 4, 12
meshgrid, 42
mode, 8
MOUSEBUTTONDOWN, 22
MOUSEBUTTONUP, 22
MOUSEMOTION, 22
MP3, 23
MSP, 8

ndarray, 1, 29
new, 9
norm, 52
normal, 40
NpzFile, 55
Numba, 75
NumPy, 1, 29

Ogg Vorbis, 23
ones, 50
oo, 60
open, 8
OpenCV, 1

paste, 11
pause, 23
PCX, 8
pi, 60
pieslice, 13
Pillow, 8
play, 23
plot, 33, 69
plot3d, 70
plot_surface, 43
plot_wireframe, 43
PNG, 8
point, 13
polygon, 13, 17
PPM, 8
pprint, 67
pygame, 14
pylab, 49

QUIT, 16
quit, 16

rand, 40
randint, 40
ratsimp, 62
read, 2
rect, 16
rectangle, 13
release, 2
render, 18
resize, 4, 10
rewind, 23
rotate, 11, 20
rsolve, 66

save, 9, 17, 54
savetxt, 53
scale, 20
scatter, 41
series, 65
set_caption, 16
set_pos, 23
set_volume, 23
set_xlabel, 43
set_ylabel, 43
set_zlabel, 43
SGI, 8
SHA1, 71
SHA224, 71
SHA256, 71
SHA384, 71
SHA512, 71
shape, 4, 31
show, 9, 33
simplify, 59
size, 8
solve, 65
sparse matrix, 51
SPIDER, 8
split, 4, 11
Sprite, 25
stop, 23
subplots, 35
subplots_adjust, 36
subs, 62
Sum, 68
Surface, 14
surface plot, 43
Surface オブジェクトのサイズ, 22
Symbol, 59
symbols, 59
sympify, 59
SymPy, 58
sympy.Function, 60
sympy.Number, 60
sympy.Symbol, 60
SysFont, 18

T, 51
thumbnail, 10
tich, 16
TIFF, 8
title, 33
transpose, 11, 51

uin32, 29
uint16, 29
uint64, 29
uint8, 29
unpause, 23
update, 16

vectorize, 56, 57
VideoCapture, 2

waitKey, 2
WebP, 8
width, 13

XBM, 8
xlabel, 33

ylabel, 33

zeros, 50

アスペクト比, 49
アニメーション GIF, 13
イベントキュー, 14, 16
イベント種別, 16
因数分解, 61
エルミート共役行列, 52
音声の再生, 23
階差方程式, 66
回転, 20
拡縮, 20
カラーマップ, 43
関数のグラフ, 69
画像モード, 8

共有ライブラリ, 77
逆行列, 67
行列式, 67
行列のランク, 52
グリッド線, 35
固有値, 68
固有ベクトル, 68
差分方程式, 66
散布図, 40
式の展開, 61
数式処理システム, 58
スパース行列, 51
スプライト, 25
漸化式, 66
総和, 68
代数方程式, 65
遅延実行, 64
定数, 60
デジタル署名, 71
頭部と引数列の取り出し, 60
日本語の見出し・ラベル, 38
配列のサイズ, 31
汎関数, 64
パスワード文字列の秘匿化, 71
パワースペクトル, 48
ヒストグラム, 40
微分方程式, 66
フィルタ, 10
フォント名の取得, 18
複素共役行列, 52
フレームのサイズ, 4
フーリエ逆変換, 45
フーリエ変換, 45
ベクトル化, 57
ベクトルのノルム, 52
棒グラフ, 32, 49
窓関数, 49
見出し行, 54
メッセージダイジェスト, 71
免責事項, 83
面プロット, 43
連立方程式, 66
ワイヤフレーム, 43