

データ表現

《学修項目》

- ◎コンピュータで扱うデータ（数値、文章、画像、音声、動画など）
- 構造化データ、非構造化データ
- 情報量の単位（ビット、バイト）、二進数、文字コード
- 配列、木構造（ツリー）、グラフ
- 画像の符号化、画素（ピクセル）、色の3要素（RGB）
- 音声の符号化、周波数、標本化、量子化

《キーワード》

データの種類、数と文字列、ビットとバイト、数の表現方法、構造化されたデータ、非構造化データ、配列とデータ、線形探索、木構造、二分探索、テキストデータ、画像データ、音声データ、動画データ

《参考文献、参考書籍》

- [1] 東京大学MIセンター公開教材 「2-2 データ表現」 《利用条件CC BY-NC-SA》
- [2] 教養としてのデータサイエンス（講談社 データサイエンス入門シリーズ）
- [3] Pythonで学ぶアルゴリズムとデータ構造（講談社 データサイエンス入門シリーズ）
- [4] テキスト・画像・音声データ分析（講談社 データサイエンス入門シリーズ）

1. 構造化データ、非構造化データ

1.1 構造化データと非構造化データのあらまし [1]

構造化データ

- コンピュータで扱うデータと聞いてどの様なものを想像しますか？ 実はデータには様々な形式のものが存在します。
 - 例えば、何らかの製品を販売している会社が扱っているデータを考えてみましょう。この場合、社内の各部署のコストをまとめたデータや、いつどこにどれだけ製品を出荷したのかを記録したデータが考えられます。部署や「どこ」に通し番号を振っておけば、これらは数値で表すことができるデータと考えられます。
 - この様な数値によるデータは表（ひょう）形式で管理することができ、構造化データと呼ばれます。

非構造化データ

- 一方、近年、ネットワーク技術の進歩、ネットワーク上のサービス（Twitterなどのソーシャルサービス、メールやストレージのクラウドサービスなど）の発展、携帯端末の普及などにより様々なデータが大量に作成され、扱われる様になりました。
- 例えば、新聞・Webなどの文章データ、写真などの画像データ、電話などの音声データ、テレビなどの動画データなどを挙げることができます。こういった単純には数値では表すことができない様なデータを非構造化データと言います。
 - 例えば、何らかの製品を販売している会社では、業務日誌・議事録などの文章データ、通話記録などの音声データ、TV会議などの画像データや動画データなどの非構造化データが発生することが考えられます。

1.2 構造化データと非構造化データは何か違うのか [2]

1.2.1 構造化データ [2]

データの処理を事後的にやりやすくするため、ある特定の目的のために収集されたデータは、あらかじめデータの並べ方を厳格に定めている。このようなデータのタイプを構造化データ(structured data)と呼ぶ。

わかりやすい例は、表計算用のデータである。データ全体を目的毎のテーブルに分割し、さらにテーブルに記録される項目をフィールドとして定義し、そして各データはレコード

という単位で記録・管理する。このように構造化されたデータが**データベース(database)**である。

計算機に構造化データの処理作業を指示するときには、データの並べ方のルールに従わなくてはならない。逆にルールに従っていれば、データの並び方の構造を上手に利用した、高速かつ大規模な**データに対する検索や処理**が可能になる。

- OBトコ | DBの勉強
- SAMURAI Engineer Blog | SQLとクエリの違いは？基礎知識・サンプルコードも徹底解説！

1.2.2 非構造化データ [2]

データの並べ方に「ルール」がないデータのことを総称して非構造化データ(unstructured data)と呼ぶ。文書、画像、音声、音楽、動画などは、それぞれ格納されるデータの「単位」は同一であっても、それが**表現される形式はバラバラ**である。

例えば文書（テキストデータ）については何の言語で書かれたどの文字コードで表現されたものかが特定されていたとしても、それは非構造的である。日本語であれば**形態素解析**を行わなければ単語に分割されないし、単語間の連携は文法の解析を行わなければならない。文法の解析が終われば、構文木として初めて構造化されたデータとして手に入ることになる。

非構造化データを取り扱うためには、**メタデータ**（対象データに付随して説明する何らかの情報）が重要である。

- Wikipedia | ファイルフォーマット一覧
- udemyメディア | 形態素解析とは？おすすめの5大解析ツールや実際の応用例を紹介
- Google検索セントラル | 構造化データの仕組みについて

1.3 非構造化データの解析（認識）の難しさ [2]

例えば画像データそれ自体は非構造化データである。元の画像情報は、たとえばラスター・スキャンの方向でピクセル(pixel)と呼ぶ最小単位で構成されている。各ピクセルは色や輝度を有していることが普通であるから、その情報を各色空間でエンコードしてデジタル化する。

ここで対象とする画像データを解析、あるいは「その画像は、何の画像なのか」を認識することを考える。機械的な（計算した結果としての）画像認識とは、画像中の物体（オブジェクト）は何なのかを「言葉」で答える問題である。これは**物体認識**と呼ばれ、

- 特定物体認識
- 一般物体認識

に大別される。

- MathWorks | 物体認識 これだけは知りたい3つのこと

1.3.1 特定物体認識 [2]

交通標識のように、 標識マークの数が限られている状況を考えてみよう。 特定物体認識とは、 画像から何の交通標識か、 つまり画像に対応した言葉を認識する課題である。 各国で交通標識は法令で定められており、 矢印の形や色、 赤い丸の中に数字が入っているなど形式が統一されているから、 処理能力が低いエッジ向け推論フレームワークなどでも実装が容易である。

- [Medium.com | TrafficSignDetection : 道路標識を検出できる機械学習モデル](#)

1.3.2 一般物体認識 [2]

一般物体認識では、 ありとあらゆる言葉（対象物、 意味）に対応した画像を取り扱う。 当然、 一般物体認識のほうが格段に難しい。 一般物体認識でも、 画像の中に1つ物体が入っている問題と、 複数物体が入っている問題とはまた格段に難易度が異なる。

一般物体認識の問題は、 認識結果を最終的にどのような形式で出力するかによって、 (a)分類(b)物体検出(c)セマンティックセグメンテーションの3つに大別され、 この順に難易度は上がっていく（[参考サイト](#)）。

(a) 画像の分類とは、 画像に写っている物体のカテゴリ（クラスと呼ばれる）を認識結果として出力する問題である。

(b) 画像から物体を認識するためには、 画像の中から物体の境界を切り出す、 セグメンテーション(segmentation)という作業が必要になる。 一般物体認識で実現されるAI機能として、 物体らしき部分を特定する作業がセグメンテーションである。

(c)セグメンテーションの後、 囲われたものの名前（一般に理解可能な名称）を特定する。 特定した物体の様相（例：車輪が2つ付いていてフレームで結合されたものとしてのモーターサイクル）に応じた領域を推定する。

- [Laboro.AI | ディープラーニングによる一般物体認識とビジネス応用<下>物体検出](#)

2. データの表現方法(1) 数値・数値計算

2.1 データの表現方法：数の場合、 2進数など

データの表現方法　数の場合

- コンピュータ上でデータを表現するにはどうしたら良いでしょうか？この教材にも数値や文章など様々なデータ（値）が含まれていますが、これらの値をコンピュータは2を基底とする数、すなわち、0と1の2つの数から構成される**2進数**を用いて表しています。
- 我々が普段使っている数は0から9の数を用いる10進数です。10進数では10を基底とする数（基底と言います）で数が表現されています。
- 例えば、794という数は、各桁毎に右（1の位）から順に見ていくと、1（10の0乗）が4個、10（10の1乗）が9個、100（10の2乗）が7個からなる数として構成されていると見なせます。

2進数

- 2進数は2を基底としています。つまり、2の0乗が●個、2の1乗が◎個、2の2乗が△個、…という様な形で数を表すことになります。
 - 例えば、2進数で1101と記述すると、10進数と同様に各桁毎に右から順に、1（2の0乗）が1個、2（2の1乗）が0個、4（2の2乗）が1個、8（2の3乗）が1個からなる数として構成されています。つまり、2進数の1101は、10進数で表すと $1 \times 1 + 2 \times 0 + 4 \times 1 + 8 \times 1 = 13$ となります。
- 10進数の1101と2進数の1101がこのスライドの様に一緒に書かれていると非常に紛らわしいですね。そこでこれらを区別するために、基底を添え字として明記する記法が存在します。例えば、10進数の場合は $1101_{(10)}$ 、2進数の場合は $1101_{(2)}$ となります。この記法を用いた場合、 $1101_{(2)}$ は10進数では $13_{(10)}$ と表すということです。

2.2 Pythonによる数値表現の実装例

Pythonによる実装については、クラウド環境であれば Google colab. が即時使える。一ヵ月のPC上の環境であれば、Windows, macOS, Linuxなどで各自別途インストール作業が必要となる。詳細は [こちらの参考書籍付属の公開資料](#) を参照されたい[3]

- [Pythonで学ぶアルゴリズムとデータ構造（講談社） 公開資料](#)
- [Pythonチュートリアル | 3. 形式ばらない Python の紹介](#)

2.2.1 整数と小数

Pythonでは整数と小数は別のデータ型である。データの型は、組み込み関数typeで調べる。

```
In [21]: type(2) # 整数
```

```
Out[21]: int
```

```
In [22]: type(2.0) # 小数
```

```
Out[22]: float
```

整数同士の演算は整数となり、整数と小数の演算結果は小数になる。

```
In [66]: 123 + 456 # 整数同士の加算
```

```
Out[66]: 579
```

```
In [24]: 1024 / 256 # 除算の結果は float となる
```

```
Out[24]: 4.0
```

```
In [25]: 10 // 3 # 結果を整数で得る除算
```

```
Out[25]: 3
```

```
In [26]: 65535 % 16 # 16の剰余を取る
```

```
Out[26]: 15
```

2.2.2 2進数と整数、小数

プレフィックス 0b, 0o, 0x によって、整数型intの数値をそれぞれ2進数、8進数、16進数で記述する。

なお、print()での出力は10進数になる。

```
In [27]: binary_num = 0b1010 # Base2
octal_num = 0o2040 # Base8
hexadecimal_num = 0xFFFF # Base16

print(binary_num)
print(octal_num)
print(hexadecimal_num)
```

```
10
1056
65535
```

```
In [28]: binary_num + octal_num + hexadecimal_num # 数値はすべて整数型intであるので、普通に演算可能
```

```
Out[28]: 66601
```

```
In [29]: print(0b1111_1111_1111_1111 == 0xFFFF) # 2進数はアンダースコア _ で区切ることができる。
```

```
True
```

```
In [30]: bin(123) # 組込関数binは、整数の2進数表現を返す（結果は文字列型）
```

```
Out[30]: '0b1111011'
```

```
In [31]: bin(123*2) # 2進数はBase2であるので、2倍すればビットが左シフトする
123 << 1 # ビットシフト演算子を使っても同じ結果である
```

```
Out[31]: 246
```

```
In [32]: import sys
print(sys.float_info.max) # 浮動小数点で表現できる最大値
print(sys.float_info.min) # 浮動小数点で表現できる最小値

1.7976931348623157e+308
2.2250738585072014e-308
```

```
In [33]: print('{:.3e}'.format(12345)) ## 小数点以下の桁数を指定して指数にしたい場合
1.234e+04
```

2.3 Pythonによる数値計算の実装例（関数、微分積分）

数学の定数などは math ライブラリをインポートして使う。円周率は math.pi で定義されている。

float型をそのまま print で表示すると桁が多すぎるので、適宜 format 指示子を用いて表示を制御する。

```
In [34]: import math # mathライブラリをインポート

diag = 2.0 # 変数 diag に半径を代入
area = diag**2 * math.pi # 円の面積を求める
print('{0:.4f}'.format(area)) # 小数点以下4桁でfloat表示する

12.5664
```

数式処理ライブラリ『SymPy』を使って、初等関数に関する種々の操作・演算・微分積分などを行ってみる。

- [SymPy Tutorial 日本語ノート](#)

```
In [67]: !pip install sympy
```

```
Requirement already satisfied: sympy in /usr/local/lib/python3.10/site-packages (1.11.1)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/site-packages (from sympy) (1.2.1)
```

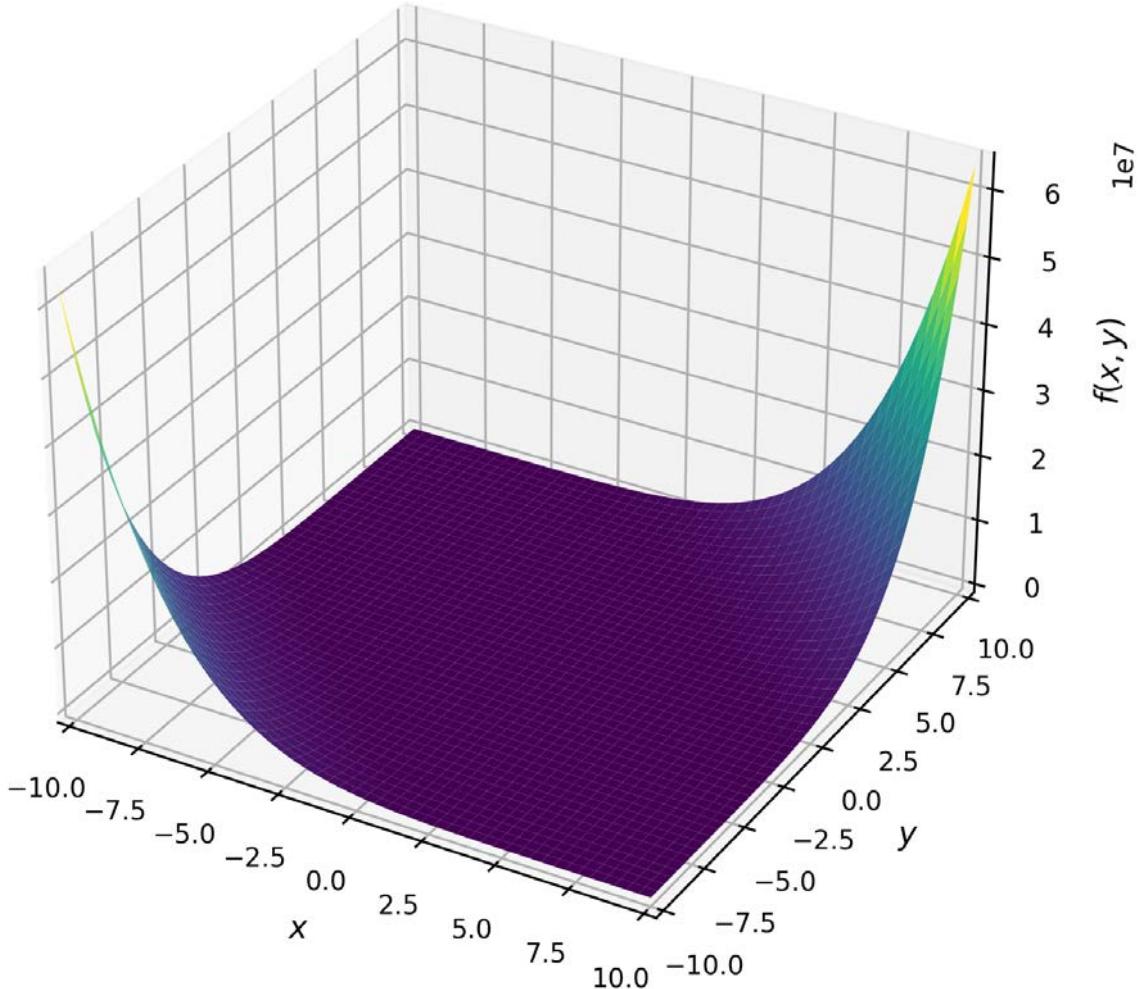
```
In [37]: import sympy as sym # SymPy ライブラリをインポート
from sympy.plotting import plot3d # Mathplot 経由で SymPy の結果を描画する設定
sym.init_printing(use_unicode=True)
%matplotlib inline

a, b, c, x, y = sym.symbols("a b c x y") # 変数 a,b,c,x,y を定義しておく
```

```
In [38]: expr = (x+y)**6 # 数式を定義
sym.expand(expr) # 多項式を展開する
```

```
Out[38]: x6 + 6x5y + 15x4y2 + 20x3y3 + 15x2y4 + 6xy5 + y6
```

In [39]: `plot3d(expr, (x, -10, 10), (y, -10, 10)) # 指定した区間でグラフを描画`



Out[39]: <sympy.plotting.plot.Plot at 0x153f42c80>

In [40]: `sym.simplify(1/(x-1)**2+1/((x-1)*(x-2))) # 式を簡単化する`

$$\frac{2x - 3}{(x - 2)(x - 1)^2}$$

In [41]: `sym.solve(a*x**2+b*x+c, x) # sympy.solve(eq, var)を使って,2次方程式を求解する`

$$\left[\frac{-b - \sqrt{-4ac + b^2}}{2a}, \frac{-b + \sqrt{-4ac + b^2}}{2a} \right]$$

1変数実関数の微分積分

In [42]: `def f(x): return sym.sin(4*x)**2 # 関数 f(x) を定義する (三角関数の例)
#def f(x): return -sym.log(1-x)/x # 関数 f(x) を定義する (対数関数の例)
sym.simplify(f(x))`

$$\sin^2(4x)$$

In [43]: `sym.diff(f(x), x, 1) # 変数xで f(x)を微分した結果を得る`

Out[43]: $8 \sin(4x) \cos(4x)$

In [44]: `sym.integrate(f(x), x) # 変数xで f(x)を積分した結果を得る`

Out[44]: $\frac{x}{2} - \frac{\sin(4x) \cos(4x)}{8}$

3. データ構造（配列、木構造、グラフ）

3.1 データの表現方法：配列

配列

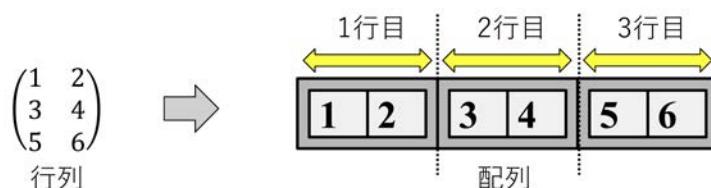
- 配列は値を順序付けて格納するデータ構造です。
- 「順序付けて格納する」という表現はやや複雑ですが、要するに配列に格納されているデータ（値）は「〇番目のデータ（値）」と呼ぶことが出来る形式で保管されているということです。
 - 配列は数学のベクトルを表すのに適したデータ構造と言えます。例えば、 $(1,2)$ と $(2, 1)$ というベクトルは全く異なる値ですよね。このベクトルの様に値が順序付けられている場合、配列を用いると良いでしょう。
 - この教材では配列は下の様な図で配列を表すことにします。この例の配列では1番目の値は8で、4番目の値は0が格納されています。

8	6	10	0	5
---	---	----	---	---

- 配列は探索や並べ替えなど様々なアルゴリズムにおいて用いられます。

配列

- 配列には数値だけではなく、様々なデータを格納することができると言えるのが一般的です。その際、配列の中に配列を格納している場合、**多重配列（多次元配列）**と呼びます。
 - 数学の行列は多重配列を用いて表すことが出来ます。行列の各行の値を1つの配列に格納します。そして、それらの配列を1つの配列に順番に格納します。
 - 例えば、以下の例では 3×2 の行列に対して、1つの行毎に配列に格納し、更にその3つの配列を1つの配列に順に格納しています。



3.2 Pythonによる配列表現の実装例

- Pythonチュートリアル | 3.1.3. リスト型 (list)

3.2.1 リスト構造（1次元配列）

```
In [45]: my_array = [10, 20, 30, 40, 50] # 5つの整数が格納された配列データ(一番最初のインデックス)
```

```
Out[45]: [10, 20, 30, 40, 50]
```

```
In [46]: my_array[2] # インデックス#2 (3番目) の要素を参照する
```

```
Out[46]: 30
```

```
In [47]: my_array.insert(2, 100) # インデックス#2の位置に 100 を挿入する
```

```
Out[47]: [10, 20, 100, 30, 40, 50]
```

```
In [48]: my_array.append(300) # 一番後ろに 300 を挿入する
```

```
Out[48]: [10, 20, 100, 30, 40, 50, 300]
```

3.2.2 多重配列（標準ライブラリ）

```
In [49]: # 2次元配列を定義する.
          # 外側の角カッコが外側(1次元目)の配列で, 中側の角カッコが内側(2次元目)の配列になる.
my_array2 = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
]
my_array2
# この場合, 外側の配列の要素数は3つで, 中側の配列の要素数もそれぞれ3つずつになる.
# この状態の二次元配列を行列(正方行列)と表現することもある
```

```
Out[49]: [[0, 1, 2], [3, 4, 5], [6, 7, 8]]
```

```
In [50]: my_array2[1][2] # 要素(1,2)を指定する(結果はスカラー)
```

```
Out[50]: 5
```

```
In [51]: my_array2[1] # 2行目全体を指定する(結果は横ベクトル)
```

```
Out[51]: [3, 4, 5]
```

```
In [52]: my_array2.append([9, 10, 11]) # 4行目を追加する(サイズが合っている必要がある)
my_array2
```

```
Out[52]: [[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]]
```

3.2.3 numpyライブラリ

```
In [53]: import numpy as np # numpyライブラリをインポート

# 2次元配列を定義する(標準ライブラリと同じ)
np.array2 = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
]
np.array2
```

Out[53]: [[0, 1, 2], [3, 4, 5], [6, 7, 8]]

```
In [54]: id3_array = np.eye(3) # 3x3の単位行列を生成
print(id3_array)

[[1. ... 0.]
 ...
 [0. ... 1.]]
```

```
In [55]: np.array2 * id3_array # そのまま * を使うとアダマール積になる
```

Out[55]: array([[0., ..., 0.],
 ...
 [0., ..., 8.]])

```
In [56]: np.dot(np.array2, id3_array) # 通常の内積は dot を使う
```

Out[56]: array([[0., ..., 2.],
 ...
 [6., ..., 8.]])

```
In [57]: # numpy高速計算の例(おまけ)
huge_array = np.random.randn(10000000) # 1,000万サイズの1次元配列(乱数を用いて生成)
huge_array
```

Out[57]: array([1.23951637, ..., -0.40586375])

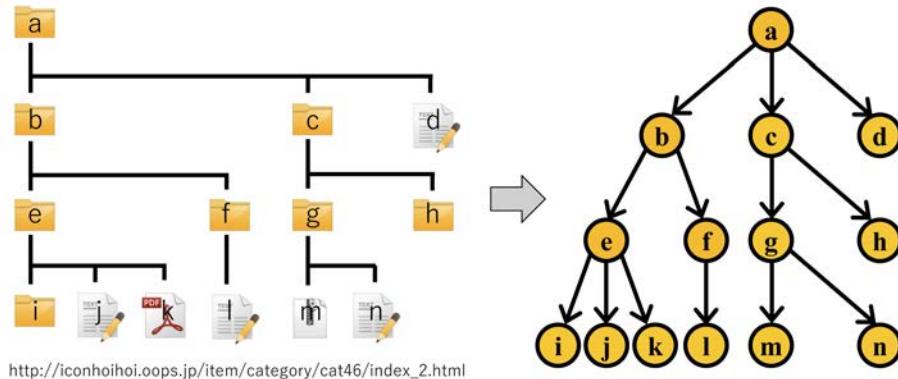
```
In [58]: %timeit np.sum(huge_array) # numpyのsumメソッド(高速)を使って,巨大なリストの和を求める計
1.95 ms ± 21.9 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
In [59]: %timeit sum(huge_array) # 標準ライブラリのsum関数(低速)を使って,巨大なリストの和を求める計
695 ms ± 9.54 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

3.3 データの表現方法：木構造

木構造

- 木構造（ツリー）は樹形図状にデータを格納するデータ構造です。組織図や家系図、コンピュータ上のファイル・フォルダの階層構造などは樹形図状に表現できますが、こういった様々な分野で用いるデータ構造です。
- 例えば、以下はファルダ類の階層構造の例ですが、この教材では右の図の様な丸と矢印からなる図で木構造で表すことになります。

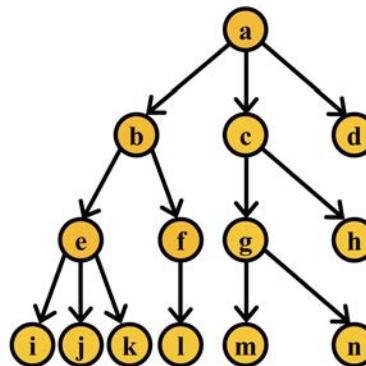


東京大学 数理・情報教育研究センター 小林浩二 2021 CC BY-NC-SA

24

木構造について

- 木構造において用いる簡単な用語を説明します。
- 図の丸 で表す部分を **点** と呼び、矢印を **枝**（もしくは **辺**）と呼びます。矢印が1本も入って来ない点を **根** と呼びます。下の例では点aが根です。ある点から出た矢印の先の点をその点の **子** と呼びます。下の例では点aの子は点b,c,dの3つです。



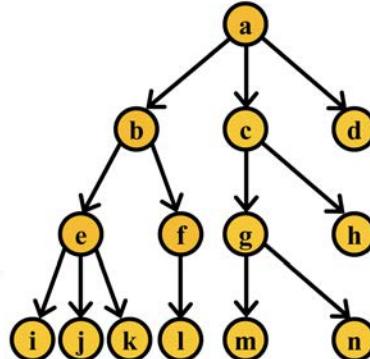
東京大学 数理・情報教育研究センター 小林浩二 2021 CC BY-NC-SA

25

木構造の代表的な表現方法は (1)隣接リスト, (2)隣接行列 の2つです。いずれも多重配列を用いて実現することができます[1]。

隣接リスト

- 隣接リストは、各点毎に1つの配列を用意し、その点の子を全て格納します。格納する順番は自由ですが、利便性から何らかの順序（例えば、昇順・降順など）に従って格納することが多いです。
- 例えば、右の木構造は下の様な多重配列による隣接リストで表します。左端が点aの子（の名前）を格納した配列を格納しています。その後、b,c,d…と順に各点の子を格納しています。
- dは子がありませんが、それでも4番目は何も格納しないd用の領域になっています。



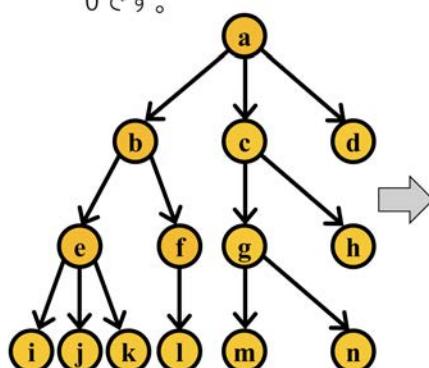
a	b	c	d	e	f	g	h	i	j	k	l	m	n
b	c	d	e	f	g	h		i	j	k	l	m	n

東京大学 数理・情報教育研究センター 小林浩二 2021 CC BY-NC-SA

27

隣接行列

- 隣接行列は、その名前の通り行列を用いて木構造を表します。既に述べた通り、行列は多重配列で表すことができます。この行列ではx行y列の値が1だと、点xから点yへの枝がある、すなわち、点xの子がyであることを意味します。
- 例えば、aの行はb,c,d列だけが1であり、それ以外は全て0です。



	a	b	c	d	e	f	g	h	i	j	k	l	m	n
a	0	1	1	1	0	0	0	0	0	0	0	0	0	0
b	0	0	0	0	1	1	0	0	0	0	0	0	0	0
c	0	0	0	0	0	0	1	1	0	0	0	0	0	0
d	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	0	0	0	0	0	1	1	1	0	0	0
f	0	0	0	0	0	0	0	0	0	0	0	1	0	0
g	0	0	0	0	0	0	0	0	0	0	0	0	1	1
h	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	0	0	0	0	0	0
j	0	0	0	0	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0	0	0	0	0
l	0	0	0	0	0	0	0	0	0	0	0	0	0	0
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	0	0	0	0	0	0	0	0	0	0	0	0	0	0

行列

28

3.4 Pythonによる木構造の実装例

Pythonでゼロから木構造を実装するには、最初に単一のノードを表す Node クラスから作成する必要がある。しかもクラスにはノードの追加や削除、トラバース（横断検索のイテレータ）などをメソッドとして実装する必要があり非常に面倒であるし、バグの温床にもなりかねない。

- Python で木データ構造を実装する

したがって、Pythonで木構造を簡単かつ信頼性高く実装するため、anytreeライブラリを使用する。

- [anytree documentation | Tree Rendering](#)

```
In [63]: # 最後に anytreeライブラリ を pipでインストールする
!pip install anytree --user

# 実行後,最後の行で Successfully installed anytree-*.**.* (*.*.*はバージョン番号) と表示される
```

Requirement already satisfied: anytree in /usr/local/lib/python3.10/site-packages (2.8.0)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.10/site-packages (from anytree) (1.16.0)

```
In [64]: from anytree import Node, RenderTree, AsciiStyle  # anytreeから必要な機能をインポート

# 各ノードを構成していく.rootノードはそのまま,childノードはparentノードを指定する.
f = Node("f") # rootノードは "f" で,これが木構造のインスタンス名となる
b = Node("b", parent=f)
a = Node("a", parent=b)
d = Node("d", parent=b)
c = Node("c", parent=d)
e = Node("e", parent=d)
g = Node("g", parent=f)
i = Node("i", parent=g)
h = Node("h", parent=i)

print(RenderTree(f, style=AsciiStyle()).by_attr())  # RenderTreeを使って木構造を表示
```

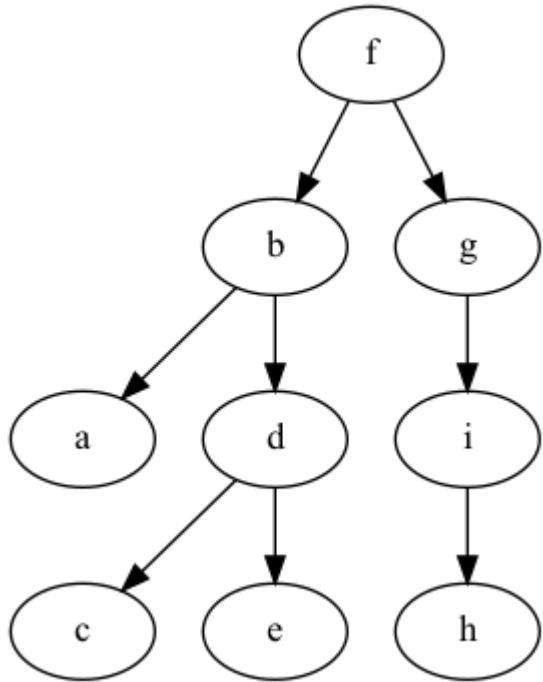
```
f
|-- b
|   |-- a
|   +- d
|       |-- c
|       +- e
+- g
    +- i
        +- h
```

*以下の例は、実行環境に Graphviz (dotコマンド) がインストールされている必要がある。

```
In [68]: from anytree.exporter import DotExporter  # GraphVizを使って木構造を可視化する
DotExporter(f).to_picture("tree.png")  # カレントディレクトリに tree.png という名前の画像を作成

from IPython.display import Image  # IPython.display モジュールを使って、対話型ノートブック上に画像を表示
Image('tree.png')  # カレントディレクトリの tree.png 画像ファイルを読み込んでノートブック上に表示
```

Out[68]:

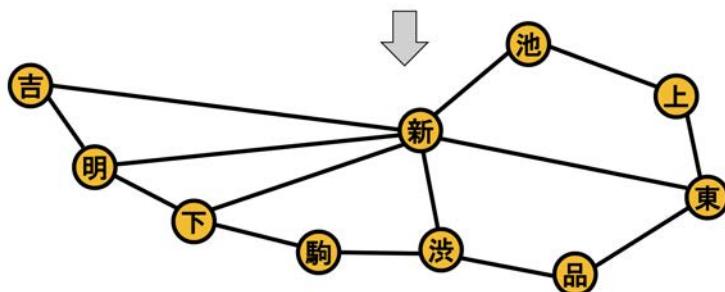
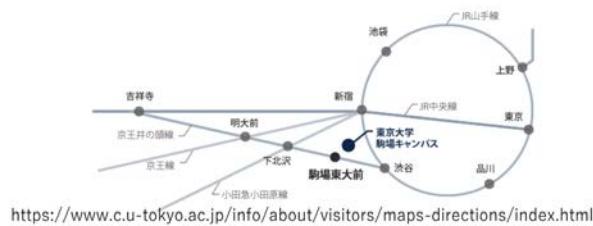


3.5 データの表現方法：グラフ

グラフは、ネットワーク状にデータを格納するデータ構造です。交通網やSNSの接続関係などもグラフで表現することができます。

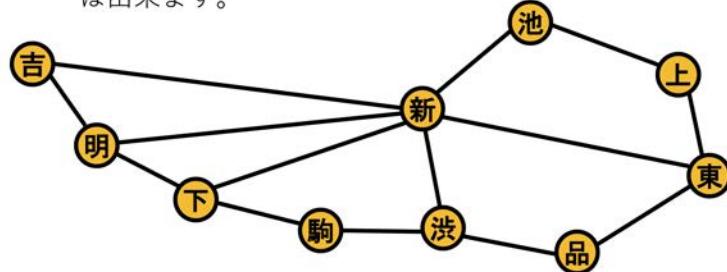
グラフ

- 例えば、下の図は東大周辺の電車の路線図ですが、駅を点、線路を枝と見なすことでグラフで表すことができます。



グラフ

- このグラフの例が木構造と似ていることに気が付いたでしょうか？ 実はグラフは木構造を一般化したものです。
- 木構造と同様に、図の丸で表す部分を点と呼び、矢印を枝（もしくは辺）と呼びます。
- 木構造は幾つかの異なる点をたどって同じ点に戻ってくることは出来ない様な構造になっていましたが、その様なことが可能である構造の場合、グラフと呼ぶことになります。
 - その様なことが出来ない場合（木構造）でもグラフと呼ぶことは出来ます。

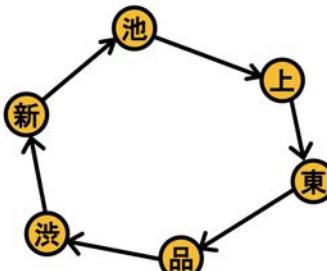


東京大学 数理・情報教育研究センター 小林浩二 2021 CC BY-NC-SA

31

グラフ

- 先に紹介した木構造との違いは、枝の書き方にも表れています。このグラフの例では、枝は矢印ではなく単なる線になっています。この様な枝をもつグラフを無向グラフと呼びます。実はグラフの枝を矢印にすることも可能で、枝が矢印になっている（枝に向きがある）グラフを有向グラフと呼びます。
- 有向グラフは例えば一方通行を含む交通網や、SNSのフォロー関係（ユーザーを点、フォローを枝とみなす）などを表現するときに用いられます。また、先程の例では、山手線の外回り（時計回り。勿論、内回りでも構いません）に着目した場合、一部を有向グラフで表すことが出来ます。



東京大学 数理・情報教育研究センター 小林浩二 2021 CC BY-NC-SA

32

3.6 Pythonによるグラフ構造の実装例

Pythonでグラフ構造を簡単かつ信頼性高く実装するため、 NetworkXライブラリを使用する[3].

- Software for Complex Networks | NetworkX

```
In [72]: # 最初に NetworkXライブラリ を pipでインストールする
!pip install networkx
```

```
# 実行後、最後の行で Successfully installed network-**.* (**.*はバージョン番号) と表示さ
```

```
Collecting networkx
```

```
  Downloading networkx-2.8.6-py3-none-any.whl (2.0 MB)
```

```
    2.0/2.0 MB 257.6 kB/s eta 0:0
```

```
0:00 [36m0:00:01m eta 0:00:01
```

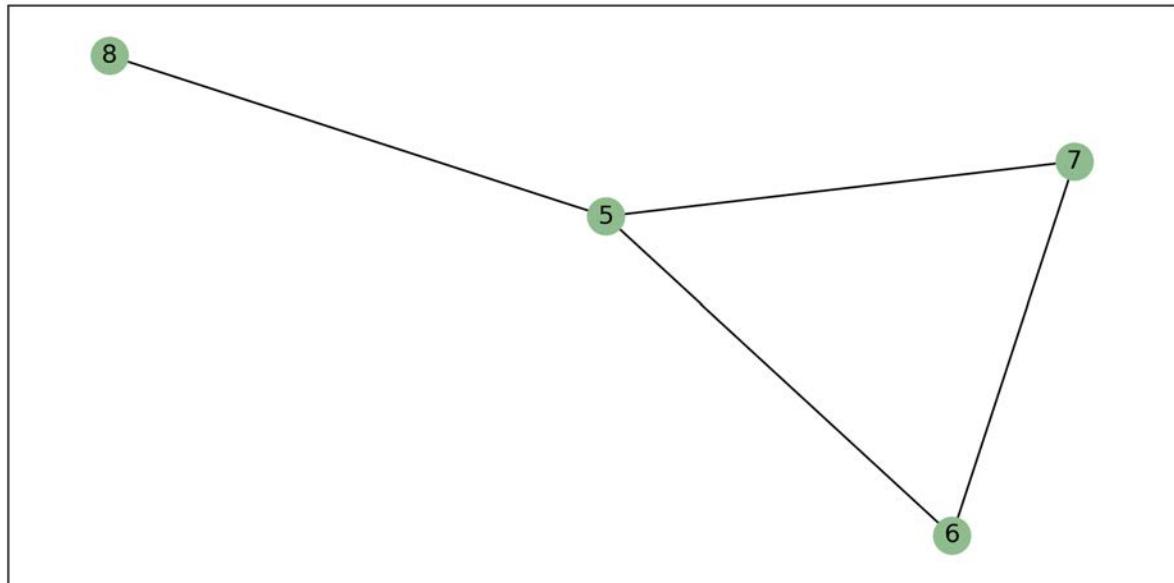
```
Installing collected packages: networkx
```

```
Successfully installed networkx-2.8.6
```

```
In [73]: %matplotlib inline
import matplotlib.pyplot as plt
import networkx as nx # NetworkX ライブラリをインポート
```

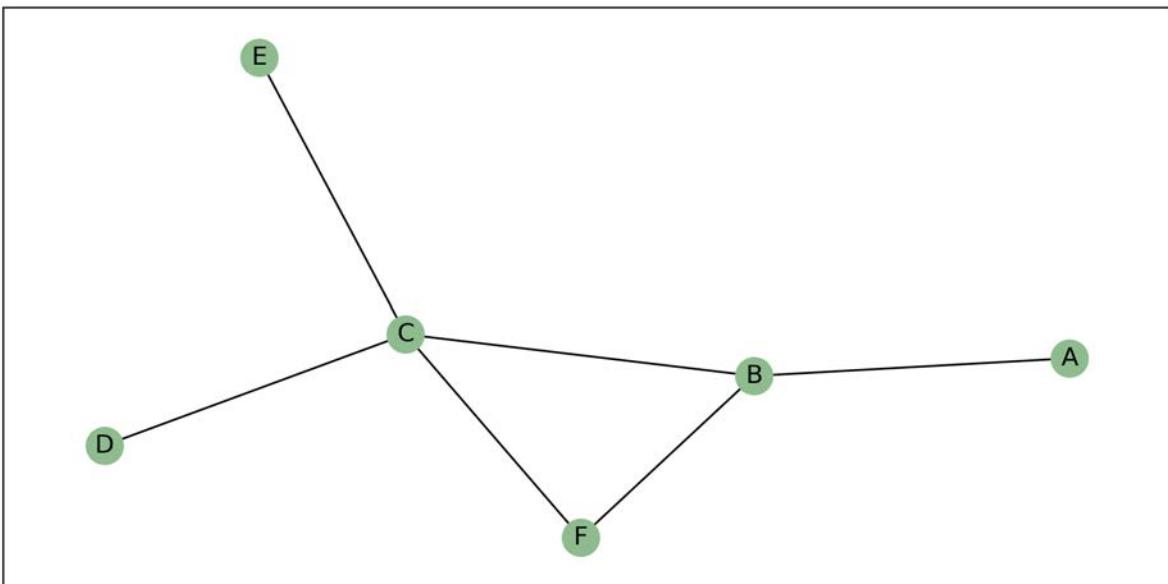
```
In [74]: # 簡単なグラフを生成してみる
graph = nx.Graph() # 無向グラフ
graph.add_edge(5,6) # ノード間の接続関係を定義
graph.add_edge(6,7)
graph.add_edge(5,8)
graph.add_edge(5,7)

nx.draw_networkx(graph, node_color='darkseagreen') # matplotlib経由でグラフを可視化
```



```
In [75]: graph2 = nx.Graph() # 無向グラフ
graph2.add_edges_from([('A', "B"), ("B", "C"), ("B", "F"), ("C", "D"), ("C", "E"), ("D", "E")])

nx.draw_networkx(graph2, node_color='darkseagreen') # matplotlib経由でグラフを可視化
```



グラフをランダム生成して表示してみる

```
In [76]: import random

# グラフ生成する関数を定義する
def generate_graph(n, m):
    """ n個の頂点とm個の辺をもつグラフを作る """
    graph_data = [[0] * n for i in range(n)]
    # 同じ辺が同一視されるように set を用意する
    edge_set = set()
    while len(edge_set) < m:
        i, j = random.sample(range(n), 2)
        if i > j: i, j = j, i
        edge_set.add((i, j))
        graph_data[i][j] = graph_data[j][i] = 1
    return graph_data, edge_set
```

```
In [77]: # 試しに、16頂点に対して20の辺を生成したランダムグラフを生成してみる  
random.seed(6)  
node_num = 16  
edge_num = 20  
my_graph, edge_set = generate_graph(node_num, edge_num)  
  
edge_set # 接続関係のsetを表示
```

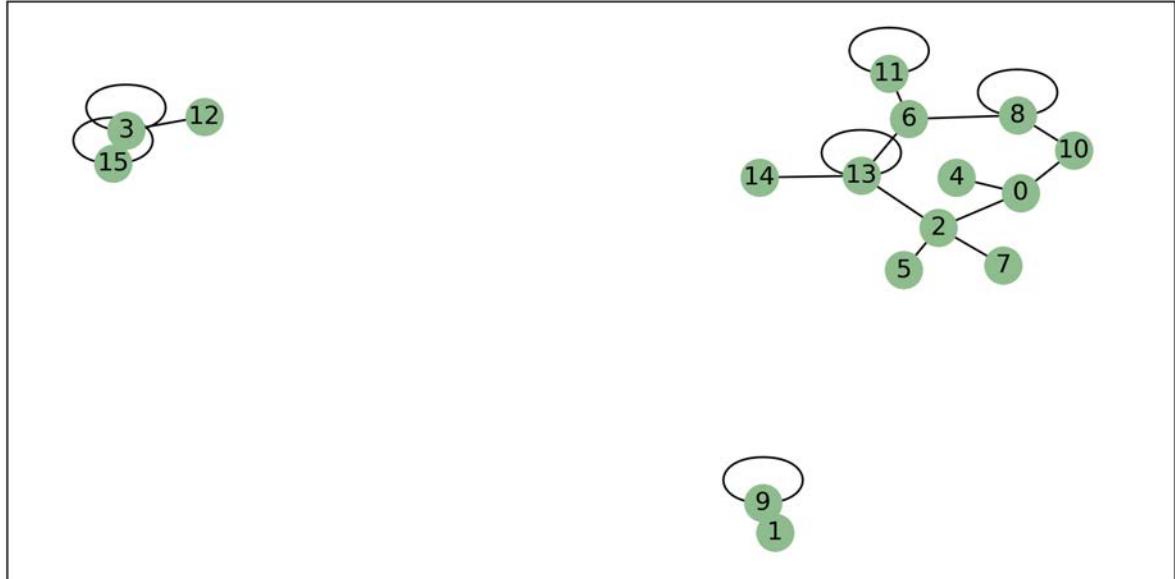
```
Out[77]: {(0, 2), (0, 4), (0, 10), (1, 9), (2, 5), (2, 7), (2, 13), (3, 3), (3, 12), (3, 15), (6, 8)}
```

```
In [78]: my_graph # 隣接行列表現
```

```
Out[78]: [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

```
In [79]: graph3 = nx.Graph() # 無向グラフ
graph3.add_edges_from(edge_set) # 接続関係のsetを利用してまとめて定義

pos3 = nx.spring_layout(graph3)
nx.draw_networkx(graph3, pos3, node_color='darkseagreen') # matplotlib経由でグラフを表示
```



4. データの表現方法(2) テキストデータ

4.1 データの表現方法：文章，テキストデータ，文字コード，情報の量と単位（バイト）

データの表現方法 文章（文字）の場合

- ここまで話で、コンピュータは2進数を使っており、我々が普段用いている10進数の数も2進数で表すことが出来ることが分かりました。
- では、数字ではなく文章、つまり文字はコンピュータはどのように表すのでしょうか？ 実はやはり2進数を用いて表しており、数字と文字を一对一で対応付けて考えています。
 - 例えば、Aは $0_{(10)}$ 、Bは $1_{(10)}$ 、Cは $2_{(10)}$ 、…、Zは $25_{(10)}$ と対応付ければ数で文字を表すことが出来ることが分かりますね。この対応付けでは、アルファベットのMは $13_{(10)}$ ですので、コンピュータ上ではMを表すのに $1101_{(2)}$ という数を用いれば良いことになります。

文字コード

- 実際にコンピュータが用いている数字と文字の対応付けを**文字コード**と言いますが、7桁の2進数でアルファベットや記号などを表す**ASCII**と呼ばれる文字コードがあります。
 - ASCIIでは、例えば $65_{(10)}$ とA、 $90_{(10)}$ とZ、 $97_{(10)}$ とa、 $122_{(10)}$ とzが対応付けられており、記号では、 $33_{(10)}$ と!、 $63_{(10)}$ と?、 $61_{(10)}$ と=などの様に対応付けられています。
 - 余程のことがない限り、どの文字にどの様な数が対応付けられているかは覚える必要ありません。
- ASCIIの対応付けには日本語は含まれていません。日本語を含む文字コードとしては、**UTF-8**、**JIS**、**Shift_JIS**（シフトJIS）などを挙げることができます。

情報の単位と量

- 2進数も10進数と同様に数字ですので、「 $1101_{(2)}$ は4桁の値」と表現できます。ですが、コンピュータ上で表す場合には特にこの1桁分を**ビット (bit)**という単位で数えます。つまり、「 $1101_{(2)}$ は4ビットの値」であると言えます。先程の文字コードはASCIIは7桁でしたので、7ビットで文字や記号を表していることになります。
- また8ビットを**1バイト (Byte, Bと略記)**と言います。
 - スマホなどの携帯端末のデータ通信の量を表すのにメガやギガという単位が出て来ますが、1キロバイト (KB) = 1024バイト (B)、1メガバイト (MB) = 1024KB、1ギガバイト (GB) = 1024MB、1テラバイト (TB) = 1024GB、…という具合に大きくなっています。
- コンピュータ上でデータをやり取りする際に、そのデータにどれだけの情報（**情報量**）が含まれるかについて議論する場合、ビットやバイトという単位を用いて議論するのが一般的です。

4.2 テキスト分析/基礎技術

自然言語で書かれた文（一般的に句点で区切られる単位）や文章（複数の文からなり、段落など、何らかの一つのまとまりをなすもの）、文書（レポートや新聞記事など一つで完結しているもの）を総称してテキストと呼ぶ[4]。

テキスト分析は、一般に**自然言語処理 (natural language processing)**と呼ばれる技術に含まれる。

自然言語処理は大きく2つの研究領域にわけることができる。1つは基礎技術であり、何らかのアプリケーションそのものというよりは、何らかのアプリケーションを開発する際

に要素技術として利用される技術群である。基礎技術には、形態素解析、構文解析、意味解析、文脈解析などがある。

もう1つは応用技術であり、具体的なアプリケーションそのものである。応用技術には情報抽出や情報検索、評議分析、機械翻訳などが含まれる。

- [Alsmiley | 自然言語処理とは！？できることをまとめたNLP入門書](#)

4.2.1 形態素解析

形態素解析は、文を形態素と呼ばれる単位に分割する処理である。ここでいう文とは、通常日本語においては句点で区切られる単位である。

形態素解析は、大きく2つの作業にわけられる。まず、形態素辞書を利用して、文からラティスと呼ばれるデータ構造を作成する作業を行う。次に、そのラティス上で、最もらしい形態素列を探索する。

日本語の文は分かち書きされていないため、解析のためにはまず文頭から一文字ずつ形態素辞書をひきながら、辞書に記述されている単語を探していく。その結果、形態素ラティスと呼ばれるグラフが構築される。

複数の解析候補が存在する場合、どの解析候補が最も適切かを判断するため、隣接する形態素の確率を考える。確率は文書カテゴリごとに様相が異なるので、単語境界の情報が付与されたコーパスに基づいて計算する（ビタビ・アルゴリズムを使用する）。

- [Alsmiley | 自然言語処理に欠かせない「形態素解析」とは？代表的なツールを紹介](#)

4.2.2 構文解析

構文解析とは文の構文構造を明らかにする処理である。日本語では、伝統的に係り受けと呼ばれる構造を解析する係り受け解析として研究が行われてきた。文節の境界を明らかにすることを、[チャンキング \(chunking\)](#) と呼ぶことがある。これは、文節の境界の同定を、文を文節という塊 (chunk) に分ける処理と考えることができるためである。

文節の境界を明らかにする手法としては、規則に基づく手法と機械学習に基づく手法がある。

規則に基づく手法では、形態素解析の結果に、人手で作成した規則を適用し、文節の境界を定める。

機械学習に基づく手法としては、あらかじめ文節区切りが付与されたコーパスから、文節の境界を機械学習に基づいて推定する。よく用いられる手法として、BIO法がある。

- [Pythonによる日本語自然言語処理 | 文節チャンキング](#)

4.3 Pythonによるテキスト分析の実装例

4.3.1 形態素解析 MeCab

日本語形態素解析システムとして広く用いられている [MeCab](#) をトークナイザとして用いてコーパスを読み込む方法を紹介する。

- Pythonによる日本語自然言語処理 | MeCabを使う

```
In [80]: !pip install mecab-python3 #MeCabライブラリをインストール
!pip install unidic-lite #辞書のインストール(国立国語研究所 https://clrd.ninjal.ac.jp/unidic/)

Collecting mecab-python3
  Downloading mecab_python3-1.0.5-cp310-cp310-macosx_10_15_x86_64.whl (282 kB)
    ━━━━━━━━━━━━━━━━ 282.8/282.8 kB 2.6 MB/s eta 0:00:00 [36m0:00:01[36m0:00:01
Installing collected packages: mecab-python3
Successfully installed mecab-python3-1.0.5
Collecting unidic-lite
  Downloading unidic-lite-1.0.8.tar.gz (47.4 MB)
    ━━━━━━━━━━━━━━ 47.4/47.4 kB 2.3 MB/s eta 0:00:01[36m0:00:01
Preparing metadata (setup.py) ... done
Building wheels for collected packages: unidic-lite
  Building wheel for unidic-lite (setup.py) ... done
  Created wheel for unidic-lite: filename=unidic_lite-1.0.8-py3-none-any.whl size=47658836 sha256=550b98f957732b679b92391fb1d744858b0904c21f0b8a1ae39230a70127c0fa
  Stored in directory: /Users/wasaki/Library/Caches/pip/wheels/89/e8/68/f9ac36b8cc6c8b3c96888cd57434abed96595d444f42243853
Successfully built unidic-lite
Installing collected packages: unidic-lite
Successfully installed unidic-lite-1.0.8
```

簡単な日本語の文章を MeCab (wakati) 解析によって分かち書きしてみる

```
In [81]: import MeCab
wakati = MeCab.Tagger("-Owakati")
wakati.parse("彼は森の中を歩いた。").split()
#wakati.parse("確率の計算には、形態素解析と同様に条件付き確率場などのモデルを利用することができます")
#wakati.parse("とうきょうとつきょきょかきょく").split()
#wakati.parse("すもももももももものうち").split()
```

```
Out[81]: ['彼', 'は', '森', 'の', '中', 'を', '歩い', 'た', '.']
```

```
In [82]: tagger = MeCab.Tagger()
print(tagger.parse("彼は森の中を歩いた。"))
#print(tagger.parse("すももももももものうち"))
```

彼	カレ	カレ	彼	代名詞	1
は	ワ	ハ	は	助詞-係助詞	
森	モリ	モリ	森	名詞-普通名詞-一般	0
の	ノ	ノ	の	助詞-格助詞	
中	ナカ	ナカ	中	名詞-普通名詞-副詞可能	1
を	オ	ヲ	を	助詞-格助詞	
歩い	アルイ	アルク	歩く	動詞-一般	五段-力行 連用形-イ音便
た	タ	タ	た	助動詞	助動詞-タ 終止形-一般
.		.		補助記号	句点
EOS					

単語を出現数順に抽出してみる。

- パーソルプロセス&テクノロジー SMKT事業部 | 形態素解析エンジンMeCabをPythonでやってみた

In [83]: # Wikipedia「形態素解析」の序文を切り出してきたものを text に代入

```
text = '''形態素解析とは、文法的な情報の注記の無い自然言語のテキストデータ(文)から、\n対象言語の文法や、辞書と呼ばれる単語の品詞等の情報にもとづき、形態素(おおまかにいえば、\n言語で意味を持つ最小単位)の列に分割し、それぞれの形態素の品詞等を判別する作業である。\\n\n自然言語処理の分野における主要なテーマのひとつであり、機械翻訳やかな漢字変換など応用も多い\\n\n(もちろん、かな漢字変換の場合は入力が通常の文と異なり全てひらがなであり、その先に続く文章も\\n\nその時点では存在しないなどの理由で、内容は機械翻訳の場合とは異なったものになる)。\\n\nもっぱら言語学的な観点を主として言語学で研究されている文法にもとづく解析もあれば、\\n\nコンピュータ上の自然言語処理としてコンピュータでの扱いやすさに主眼を置いた解析もある。\\n\n以下は後者のためのツールを用いた例で、「お待ちしております」という文を形態素解析した例である\\n\n（茶筌を使用した）。'''
```

In [84]: # MeCabで形態素を parseToNode で抽出してきた後、品詞が[名詞]に合致するものだけを検索し、
それを noun_count[word] ハッシュに格納していく。すでに出現している名詞であれば出現数を+1する。

```
mecabTagger = MeCab.Tagger()
noun_count = {}

node = mecabTagger.parseToNode(text)
while node:
    word = node.surface
    hinshi = node.feature.split(",")[-1]
    if word in noun_count.keys() and hinshi == "名詞":
        noun_freq = noun_count[word]
        noun_count[word] = noun_freq + 1
    elif hinshi == "名詞":
        noun_count[word] = 1
    else:
        pass
    node = node.next

# 最後に出現順に降順ソートしたものを生成する。
noun_count = sorted(noun_count.items(), key=lambda x:x[1], reverse=True)
print(noun_count)
```

```
[('言語', 7), ('形態', 4), ('解析', 4), ('文法', 3), ('自然', 3), ('文', 3), ('情報', 2), ('品詞', 2), ('処理', 2), ('機械', 2), ('翻訳', 2), ('かな', 2), ('漢字', 2), ('変換', 2), ('場合', 2), ('コンピュータ', 2), ('例', 2), ('注記', 1), ('テキスト', 1), ('データ', 1), ('対象', 1), ('辞書', 1), ('単語', 1), ('意味', 1), ('最小', 1), ('単位', 1), ('列', 1), ('分割', 1), ('それぞれ', 1), ('判別', 1), ('作業', 1), ('分野', 1), ('テーマ', 1), ('ひとつ', 1), ('応用', 1), ('入力', 1), ('通常', 1), ('全て', 1), ('ひらがな', 1), ('先', 1), ('文章', 1), ('時点', 1), ('存在', 1), ('理由', 1), ('内容', 1), ('もの', 1), ('観点', 1), ('主', 1), ('研究', 1), ('主眼', 1), ('以下', 1), ('後者', 1), ('ため', 1), ('ツール', 1), ('茶筌', 1), ('使用', 1)]
```

4.3.2 構文解析 CaboCha

CaboCha は、日本語係り受け解析器である。係り受けとは、文節間の意味的な修飾関係のことであり、格関係など意味的な関係を捉える上で重要である。CaboCha は、サポートベクターマシンと Cascaded Chunking Model と呼ばれる上昇型決定アルゴリズムに基づいており、固有表現解析などの豊富な機能も持つ。

Python実装の詳細は、以下の参考サイトを参照のこと。

- Python による日本語自然言語処理 | CaboChaを使う
- Google Colab で MeCab と CaboCha を使う最強の方法
- Google Colab | Python自然言語処理 101本ノックその2

以下は Python 自然言語処理 101 本ノックその 2 の最初のコードの抜粋である (Google Colab 環境のみ)。

```
In [ ]: # (引用元での動作)確認日時(2020/7/19)

!apt install -y curl file git libmecab-dev make mecab mecab-ipadic-utf8 swig
!pip install mecab-python3
import os

filename_crfpp = 'crfpp.tar.gz'
!wget "https://drive.google.com/uc?export=download&id=0B4y35FiV1wh7QVR6VXJ5d
        -O $filename_crfpp
!tar zxvf $filename_crfpp
%cd CRF++-0.58
!./configure
!make
!make install
%cd ..

os.environ['LD_LIBRARY_PATH'] += ':/usr/local/lib'
FILE_ID = "0B4y35FiV1wh7SDd1Q1dUQkZQaUU"
FILE_NAME = "cabocha.tar.bz2"
!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=dow
$(wget --quiet --save-cookies /tmp/cookies.txt \
    --keep-session-cookies --no-check-certificate 'https://docs.google.com/uc?
    | sed -rn 's/.*confirm=([0-9A-Za-z_]+).*\/1\n/p'&id=$FILE_ID" -O $FILE_NA
!tar -xvf cabocha.tar.bz2
%cd cabocha-0.69
!./configure --with-mecab-config=`which mecab-config` --with-charset=UTF8
!make
!make check
!make install
%cd ..

%cd cabocha-0.69/python
!python setup.py build_ext
!python setup.py install
!ldconfig
%cd ../
%cd cabocha-0.69
!make
!make check
!make install
%cd ../
```

```
In [ ]: # インストールチェック
```

```
!mecab --version
!cabocha --version
```

```
In [ ]: # CaboChaを使って構文解析を実行する
```

```
import CaboCha
cbc = CaboCha.Parser()
target = "彼は森の中を歩いた"
#target = "私は永遠に駆け出しエンジニアスピリットで学び続けるんだ。そしてプロになるためにまず100本"

# ラティス形式で出力
print(cbc.parse(target).toString(CaboCha.FORMAT_LATTICE))
```

```
# 構文ツリー形式で出力
print(cbc.parse(target).toString(CaboCha.FORMAT_TREE))
```

5. データの表現方法(3) 画像データ

5.1 データの表現方法：画像，動画，標本化，量子化，RGB，色空間

データの表現方法 画像・動画の場合

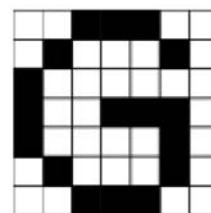
- ここまで話の流れで想像がつくと思いますが、画像や動画の場合もコンピュータは2進数を用いて数値で表しています。どの様に数値で表すのかと言うと、音声の場合と同様に、標本化と量子化を行います。やはり、この2つの操作をまとめて画像の符号化と呼びます。
- 動画はパラパラ漫画の様に画像を大量に使えば実現できると考えて、この教材では画像の場合のみ説明します。

標本化 画像の場合

- 音声の場合は波があり、標本化ではその波の値を取得する間隔を決めていました。つまり、波を一定間隔で区切っていました。これと同じ様に、画像でも画像を縦と横に一定間隔で区切って値を取得する位置を決めて標本化を行います。
- この区切った四角形の部分を**画素（ピクセル）**と言います。画素1つにつき1つの色を表す様にします。その為、画素の数（解像度と言います）が多い程、元の画像の情報を保存できるわけです。
 - 例えば以下の例では、アルファベットの「G」を 7×7 個の画素で構成する様に標本化しています。



標本化
→



量子化 画像の場合

- 音声の場合、量子化するのは波の大きさでした。画像の場合は、画素を構成する数値は色になります。
- 簡単のため、まず白黒の画像を考えましょう。色は白と黒の2通りしかありませんので、画素をなるべく小さく取ってしまえば、画素は白か黒しかありません。そこで白い部分を0、黒い部分を1で表せば、画像の各画素を1ビットで量子化することができます。
 - 例えば以下の例では、各画素をGの字が存在する画素を1、存在しない画素を0で表しています。

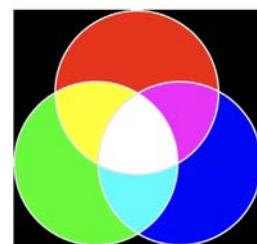


東京大学 数理・情報教育研究センター 小林浩二 2021 CC BY-NC-SA

19

RGB

- 白黒画像ではない場合に様々な色を表す為の代表的な方法にRGB方式があります。Rは赤 (Red)、Gは緑 (Green)、Bは青 (Blue) を表しており、コンピュータは光の三原色である赤・緑・青の3色を混ぜあわせてこれらの割合を調整することで様々な色を作成しています。
 - 例えば、赤：緑：青 = 1 : 1 : 0 の場合は黄色になり、赤：緑：青 = 1 : 1 : 1 の場合は白になります。全てを0にすると黒になります。
 - 赤・緑・青の量をそれぞれ256段階で量子化した場合、1つの画素の色は8ビット × 3 = 24ビットで表すことが出来ます。



東京大学 数理・情報教育研究センター 小林浩二 2021 CC BY-NC-SA

20

5.2 Pythonによる画像解析の実装例

ここでは画像解析の最初の一歩として、プログラミング言語であるPythonと画像処理ライブラリであるOpenCVを使って、ファイルからの画像の読み込み、グレイスケール変換、HSVヒストグラムの確認、ガンマ補正、そして処理後の画像ファイルの書き出しを体験してみる[4].

以下では、基本的に Google colab.環境下でのPythonとOpenCVライブラリを前提として実装されている。また、サンプル画像のダウンロードには wgetコマンドを使用しているが、これはGoogle colab.あるいは Linux環境などで標準的に入っているが、Windows版の jupyter notebookでは別途インストールと設定作業が必要なので注意されたい。

入力画像は、各自がスマートフォンで撮影した画像など、任意の画像を"sample.jpg"という名前でソースコードと同じフォルダ（Google colab.の場合は、各セッションに割当てられる一時作業領域）に保存してあるものとして説明する。

5.2.1 OpenCV2ライブラリの準備と画像ファイルの配置

```
In [88]: !pip install opencv-python
```

```
Collecting opencv-python
  Downloading opencv_python-4.6.0.66-cp36abi3-macosx_10_15_x86_64.whl (46.4 MB)
    ━━━━━━━━━━━━━━━━ 46.4/46.4 MB 1.5 MB/s eta 0:00:01
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.10/site-packages (from opencv-python) (1.23.2)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.6.0.66
```

```
In [89]: %matplotlib inline
import cv2
from matplotlib import pyplot as plt
import numpy as np
# import cv2 # OpenCV2ライブラリのインポート
# import matplotlib.pyplot as plt
print(cv2.__version__) # バージョン表示
```

```
4.6.0
```

```
In [90]: # JPEG画像データを カレントディレクトリ直下のフォルダ(一時作業領域)へダウンロードする。
!wget -nc https://raw.githubusercontent.com/MDASH-shinshu/MDASH-T-DE/main/2/
# wgetしなくても,Google colab.の左メニュー [ファイル] アイコンをクリックして,ブラウザへファイルを
# JPEG画像ファイル(例えば sample.jpg) がダウンロード・配置できたことを確認する
!ls -al .
```

```
--2022-09-03 14:41:08-- https://raw.githubusercontent.com/MDASH-shinshu/MDASH-T-DE/main/2/resources/sample.jpg
raw.githubusercontent.com (raw.githubusercontent.com) をDNSに問い合わせています...
185.199.108.133, 185.199.110.133, 185.199.111.133, ...
raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443
に接続しています... 接続しました。
HTTP による接続要求を送信しました、応答を待っています... 200 OK
長さ: 789195 (771K) [image/jpeg]
`sample.jpg' に保存中

sample.jpg          100%[=====] 770.70K 1.96MB/s 時間 0.4s

2022-09-03 14:41:10 (1.96 MB/s) - `sample.jpg' へ保存完了 [789195/789195]

total 288
drwx-----@ 1 wasaki  staff   16384  9  3 14:41 .
drwx-----@ 1 wasaki  staff   16384  9  3 14:00 ..
-rwx-----@ 1 wasaki  staff   10244  9  3 14:01 .DS_Store
drwx-----@ 1 wasaki  staff   16384  9  3 14:00 1
drwx-----@ 1 wasaki  staff   16384  9  3 14:39 2
drwx-----@ 1 wasaki  staff   16384  8 28 13:31 3
drwx-----@ 1 wasaki  staff   16384  9  3 14:37 CRF++-0.58
drwx-----@ 1 wasaki  staff   16384  8 10 11:53 _backups
drwx-----@ 1 wasaki  staff   16384  9  3 14:03 _resources
-rwx-----@ 1 wasaki  staff  790570  9  3 14:35 crfpp.tar.gz
drwx-----@ 1 wasaki  staff   16384  8 10 11:48 ex0
-rwx-----@ 1 wasaki  staff   789195  9  3 14:41 sample.jpg
```

5.2.2 画像の表示, グレースケール変換

```
In [91]: # ファイル名(sample.jpg)の部分は、自分がアップロードした画像ファイル名に変えてください
img = cv2.imread("./sample.jpg") # 画像の読み込み

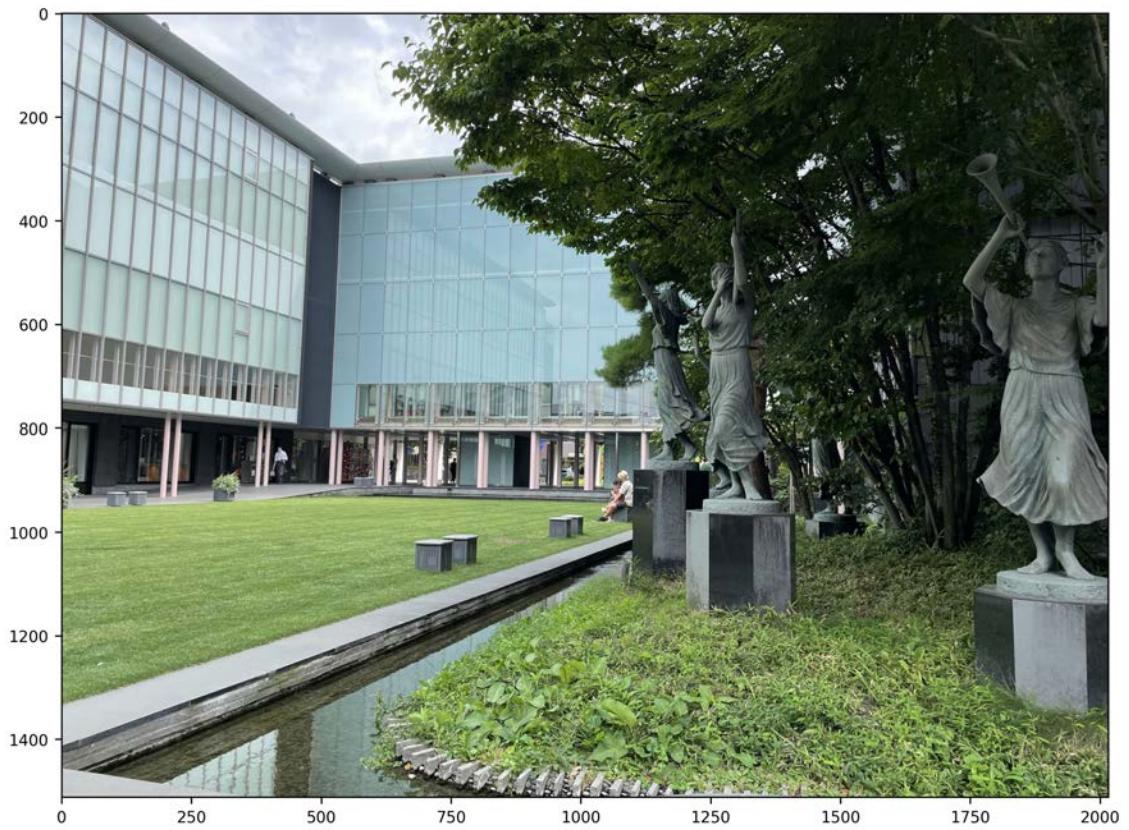
print(img.shape) # 画像サイズ,色数

print(img.dtype) # 画素のビット幅

plt.figure(figsize=(10, 8))
im_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # 色空間をBGR から RGB に変換する
plt.imshow(im_rgb) # matplotlib.imshowメソッドを用いて読み込んだ画像を表示

(1512, 2016, 3)
uint8

Out[91]: <matplotlib.image.AxesImage at 0x1551b30a0>
```



```
In [92]: plt.figure(figsize=(10, 8))
im_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # BGR を GRAYスケール に変換する()
plt.imshow(im_gray, cmap="gray") # matplotlib.imshowメソッドを用いて読み込んだ画像を表示
```

```
Out[92]: <matplotlib.image.AxesImage at 0x191518340>
```



5.2.3 HSVヒストグラム解析とガンマ補正

In [93]: ## HSV色座標系に変換してヒストグラムを表示する関数 (引用:udemy | 【Pythonで学ぶ】OpenCVでの)

```

def get_hsv_report(rgb_image, plot_show = False, statistics_show=False):
    """HSV色座標系に変換してヒストグラムを表示する。さらにパーセントタイルを利用して全体的に明るい
    Args:
        rgb_image(obj): rgbイメージ画像
        plot_show(bool): hsv票色系のグラフをプロットするか
        statistics_show(bool):標準偏差を出力するか(処理が重い)
    Returns:
        hsv票色系のパーセントタイルの値
    """
    hsv = cv2.cvtColor(rgb_image, cv2.COLOR_BGR2HSV) # hsv票色系に変換
    h,s,v = cv2.split(hsv) # 各成分に分割

    def get_percentile_list(k, datas):
        #ここでは、パーセントタイルを 5,50,95パーセントとする。
        #パーセントタイルを利用して、輝度より明るい、暗い画像を判定などに利用する
        percentile = [5,50,95] #パーセントタイルの設定。ここは、必要に応じて変更する
        out_datas = {}
        for i in percentile:
            value = np.percentile(np.array(datas), i)
            s = k + "_" + str(i)
            out_datas[s] = value
        return out_datas

    out_dict = {}
    plt.figure(figsize=(10, 8))

    #色相
    if(plot_show == True):
        plt.hist(h.ravel(),256,[0,256], color="red", alpha=0.7, histtype="st")
        data = get_percentile_list("h_per",h.ravel())
        out_dict.update(data)

    if(statistics_show == True):
        out_dict['h_pstdev'] = statistics.pstdev(h.ravel()/255)

    #彩度
    if(plot_show == True):
        plt.hist(s.ravel(),256,[0,256], color="green", alpha=0.7, histtype="st")
        data = get_percentile_list("s_per",h.ravel())
        out_dict.update(data)
    if(statistics_show == True):
        out_dict['s_pstdev'] = statistics.pstdev(s.ravel()/255)

    #輝度
    if(plot_show == True):
        plt.hist(v.ravel(),256,[0,256], color="blue", alpha=0.7, histtype="st")
        plt.legend(bbox_to_anchor=(1, 1), loc='upper right', borderaxespad=0)
        plt.show()

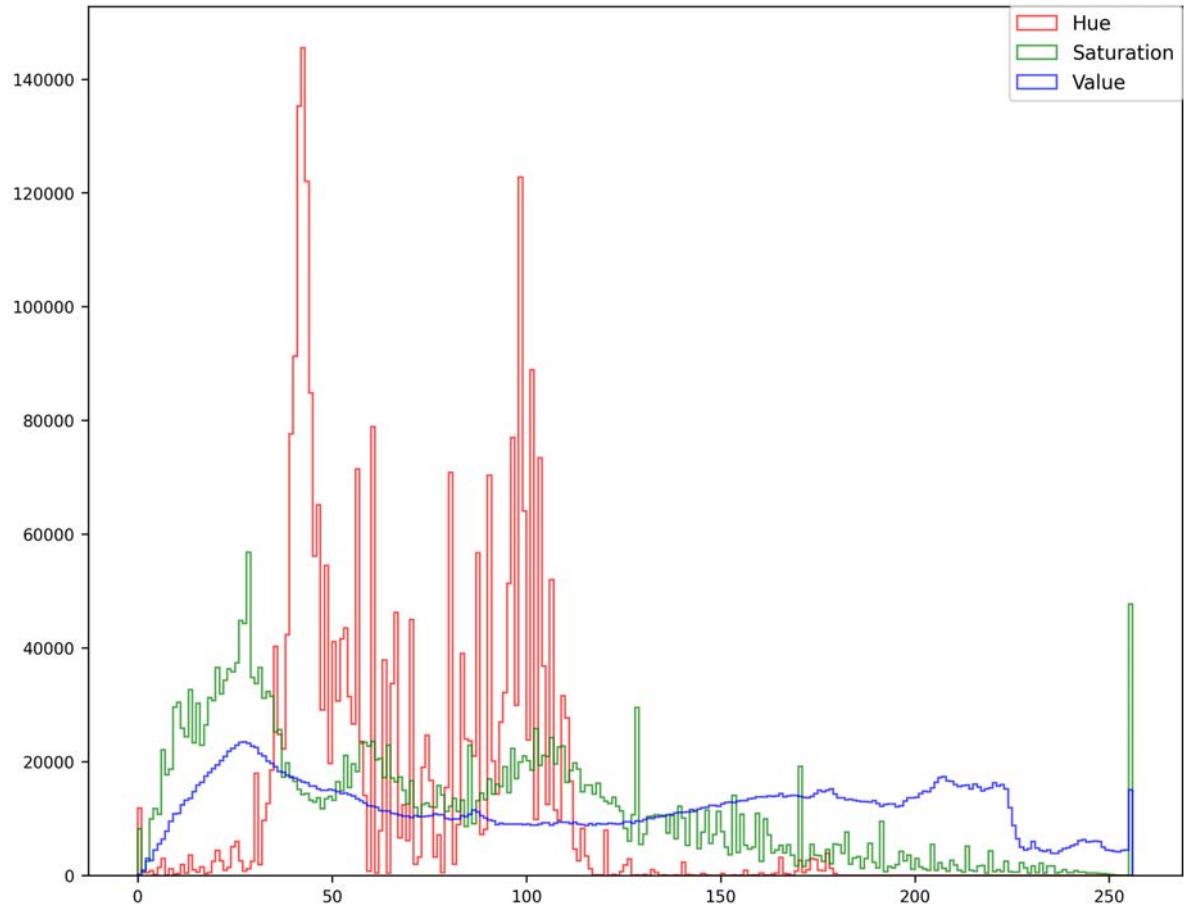
        data = get_percentile_list("v_per",v.ravel())
        out_dict.update(data)
    if(statistics_show == True):
        out_dict['v_pstdev'] = statistics.pstdev(v.ravel()/255)

    return out_dict

```

In [94]: hsv_report = get_hsv_report(img,True) # オリジナル画像 img に対して 色相(Hue), 彩
#print(hsv_report)

#輝度は青色の Value で、元画像が「暗すぎる」「明るすぎる」の傾向が解る。



In [95]: ## 画像全体の明度を変更(明るめ、暗め)するため、ガンマ補正する関数 (引用:udemy | 【Pythonで学ぶ】)

```
def gamma_correction(image, gamma):
    """ガンマ補正を利用して、画像を明るくしたり暗くしたりする
    Args:
        image(obj): イメージ画像
        gamma(float): ガンマ値 0~1までは、暗くする、1以上は明るくする
    Returns:
        ガンマ補正後のイメージ画像
    """
    # 整数型で2次元配列を作成[256,1]
    lookup_table = np.zeros((256, 1), dtype = 'uint8')
    for loop in range(256):
        # γテーブルを作成
        lookup_table[loop][0] = 255 * pow(float(loop)/255, 1.0/gamma)

    # lookup tableを用いてガンマ変換
    image_gamma = cv2.LUT(image, lookup_table)

    return image_gamma
```

In [96]: plt.figure(figsize=(10, 8))
#ガンマ補正を利用して明るくする
im_gamma = gamma_correction(im_gray, 1.6) # ガンマ補正 1.6 に設定(調整してみよ)
plt.imshow(im_gamma, cmap="gray")

Out[96]: <matplotlib.image.AxesImage at 0x191b92980>



5.2.4 ガンマ補正後のグレースケール画像の書き出し

```
In [97]: # 処理後の画像結果をカレントディレクトリに出力  
cv2.imwrite("output.jpg", im_gamma)
```

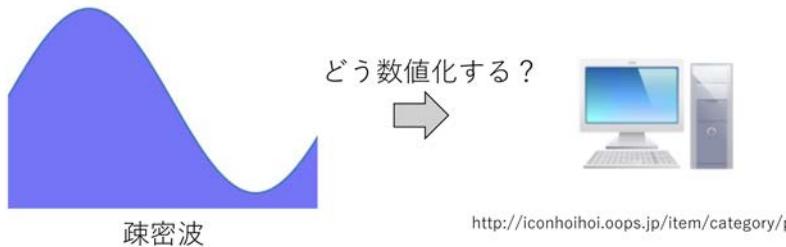
Out[97]: True

6. データの表現方法(4) 音声データ

6.1 データの表現方法：音声， アナログとデジタル， 標本化， 量子化， PCM， サンプリング

データの表現方法 音声の場合

- では、音（声）をコンピュータで扱う場合、どの様に表しているのかと言うと、やはり2進数を使って数値で表しています。しかし、例えばスピーカの音量の大小を数値で表すというのは分かりますが、音を数値で表すと言っても良く分からぬと思います。
- 実は、音は空気の密度が濃くなったり薄くなったりする波によって我々の耳に届いています。つまり、この波（疎密波と言います）がどの様な形をしているのかを数値で表してコンピュータで扱う様にすれば良いのです。



東京大学 数理・情報教育研究センター 小林浩二 2021 CC BY-NC-SA

11

アナログとデジタル

- この様な波は連続的な値で構成されています。一方でコンピュータが扱う値は離散的である必要があります。前者を **アナログ** の値と呼び、後者を **デジタル** の値と呼びます。
- 例えば、体温計には、水銀の伸びと体温計本体に刻まれた目盛りから体温を目測する水銀体温計と「 36.5°C 」の様に一定の桁数で計測した体温を表示してくれるデジタル体温計（電子体温計）がありますね。後者は名前にもデジタルと入っていますが、（例えば）小数点1桁までの離散的な値で結果を表示するデジタルの値になっています。一方前者は水銀が伸びたり縮んだりするものの、水銀が 0.1°C 刻みで結果を表示する訳ではなく、必ず連続的な値で結果を表示します。
- 同じ人がそれぞれの体温計で体温を測った場合、水銀体温計は「 $36.524\cdots^{\circ}\text{C}$ 」の様な結果となるのに対し、デジタル温度計は「 36.5°C 」の様な結果を表示します。

東京大学 数理・情報教育研究センター 小林浩二 2021 CC BY-NC-SA

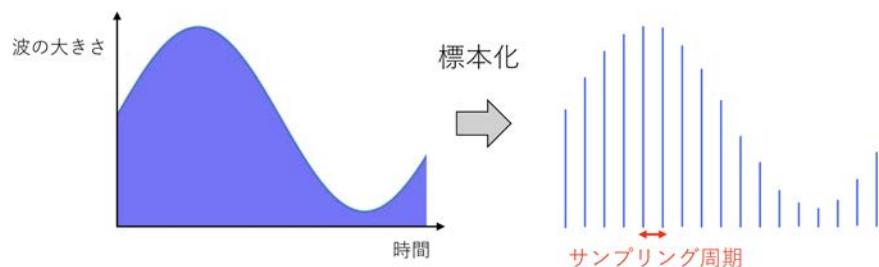
12

標本化と量子化

- つまり、コンピュータが音声（つまり、波）を扱うには、アナログの値からデジタルの値に変換してやる必要があります。その為には波に対して**標本化**と**量子化**と呼ばれる操作を行います。この2つの操作をまとめて音声の符号化と呼びます。
- 大まかに言うと、波を横方向に離散化することを標本化と言い、縦方向に離散化することを量子化と言います。

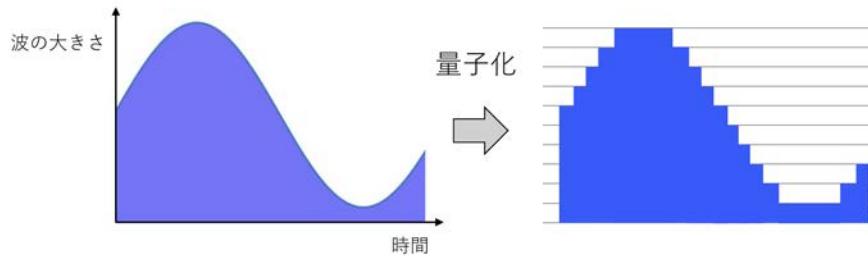
標本化

- 標本化は波の値を取得（記録）する間隔を決める操作です。例えば、30秒間にわたって発生している音（波）は、元のアナログの値ではその30秒間のどの時間（11秒目だろうと25.78…秒目だろうと）を見ても波の値を取得することができます。標本化では、例えば5秒間隔（すなわち、0秒目、5秒目、10秒目、…、25秒目、30秒目）のみ波の値を取得してコンピュータに与える（記録させる）ことにします。
- 1秒間に何回値を取得したかを表す値を**サンプリング周波数**と言います。サンプリング周波数の逆数を**サンプリング周期**と言います。



量子化

- 量子化は波の大きさ（量）の値を離散的な値に変換する操作です。例えば、元の波の各値の大きさは0以上100以下の値であった場合、波の大きさは24.5であったり、82.334…であったりするのですが、量子化ではそれを例えば0、10、20、…、90、100の11通りの値に制限してしまう操作になります（つまり、24.5→20、82.334…→80の様に変換してしまいます）。



東京大学 数理・情報教育研究センター 小林浩二 2021 CC BY-NC-SA

15

PCM

- 標本化の際の値を取得する間隔は、元の波を上手くコンピュータ上で再現可能な様な間隔にする必要があります。具体的には元の波の周波数（1秒間に繰り返される波の数）の2倍以上になる様にサンプリング周波数を設定すれば良いことが知られています。
- 代表的なデジタルへの変換方式にPCM（Pulse Code Modulation, パルス符号変調）があります。PCMはCD, DVD, mp3ファイルなど多くのメディアやファイルで利用されています。
- 例えば、一般的なCDはサンプリング周波数を44.1kHzとし、値を $2^{16}=65536$ 段階、すなわち、16ビットの値に変換して記録されています。

東京大学 数理・情報教育研究センター 小林浩二 2021 CC BY-NC-SA

16

6.2 Pythonによる音声データの特徴量解析例

ここでは音声データ解析の一歩として、Pythonと音声処理ライブラリ libROSAを使って、音声データの読み込み、再生、音声波形のグラフを表示、スペクトログラムへの変換(STFT)、逆STFTでスペクトログラムから音声を復元を体験してみる。

以下では、基本的に Google colab.環境下でのPythonとlibROSAライブラリを前提として実装されている。また、サンプル画像のダウンロードには wget コマンドを使用しているが、これはGoogle colab.あるいは Linux 環境などで標準的に入っているが、Windows 版のコマンド実行の jupyter notebook では別途インストールと設定作業が必要なので注意されたい。

入力音声データは、各自が手持ちにあるフリーの音声データや録音データなど、任意のデータをソースコードと同じフォルダ（Google colab.の場合は、各セッションに割当てられる一時作業領域）に保存してあるものとして説明する。

6.2.1 libROSAライブラリの準備とファイルの配置

```
In [98]: !pip install tqdm # 進捗状況を可視化するプログレスバー のライブラリ  
!pip install librosa # 音声処理統合ライブラリ  
!pip install matplotlib  
!pip install pandas
```

```
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/site-packages (4.64.0)
Requirement already satisfied: librosa in /usr/local/lib/python3.10/site-packages (0.9.2)
Requirement already satisfied: decorator>=4.0.10 in /usr/local/lib/python3.10/site-packages (from librosa) (5.1.1)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/site-packages (from librosa) (21.3)
Requirement already satisfied: joblib>=0.14 in /usr/local/lib/python3.10/site-packages (from librosa) (1.1.0)
Requirement already satisfied: scikit-learn>=0.19.1 in /usr/local/lib/python3.10/site-packages (from librosa) (1.1.2)
Requirement already satisfied: audioread>=2.1.9 in /usr/local/lib/python3.10/site-packages (from librosa) (3.0.0)
Requirement already satisfied: scipy>=1.2.0 in /usr/local/lib/python3.10/site-packages (from librosa) (1.9.1)
Requirement already satisfied: pooch>=1.0 in /usr/local/lib/python3.10/site-packages (from librosa) (1.6.0)
Requirement already satisfied: resampy>=0.2.2 in /usr/local/lib/python3.10/site-packages (from librosa) (0.4.0)
Requirement already satisfied: numpy>=1.17.0 in /usr/local/lib/python3.10/site-packages (from librosa) (1.23.2)
Requirement already satisfied: numba>=0.45.1 in /usr/local/lib/python3.10/site-packages (from librosa) (0.56.2)
Requirement already satisfied: soundfile>=0.10.2 in /usr/local/lib/python3.10/site-packages (from librosa) (0.10.3.post1)
Requirement already satisfied: setuptools<60 in /usr/local/lib/python3.10/site-packages (from numba>=0.45.1->librosa) (59.8.0)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /usr/local/lib/python3.10/site-packages (from numba>=0.45.1->librosa) (0.39.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.10/site-packages (from packaging>=20.0->librosa) (3.0.9)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.10/site-packages (from pooch>=1.0->librosa) (2.28.1)
Requirement already satisfied: appdirs>=1.3.0 in /usr/local/lib/python3.10/site-packages (from pooch>=1.0->librosa) (1.4.4)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/site-packages (from scikit-learn>=0.19.1->librosa) (3.1.0)
Requirement already satisfied: cffi>=1.0 in /usr/local/lib/python3.10/site-packages (from soundfile>=0.10.2->librosa) (1.15.1)
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/site-packages (from cffi>=1.0->soundfile>=0.10.2->librosa) (2.21)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/site-packages (from requests>=2.19.0->pooch>=1.0->librosa) (1.26.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/site-packages (from requests>=2.19.0->pooch>=1.0->librosa) (3.3)
Requirement already satisfied: charset-normalizer<3,>=2 in /usr/local/lib/python3.10/site-packages (from requests>=2.19.0->pooch>=1.0->librosa) (2.1.1)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/site-packages (from requests>=2.19.0->pooch>=1.0->librosa) (2022.6.15)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/site-packages (3.5.3)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/site-packages (from matplotlib) (4.37.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/site-packages (from matplotlib) (21.3)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/site-packages (from matplotlib) (9.2.0)
```

```
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/site-packages (from matplotlib) (1.23.2)
Requirement already satisfied: pyparsing>=2.2.1 in /usr/local/lib/python3.10/site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/site-packages (1.4.4)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/site-packages (from pandas) (2022.2.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/site-packages (from pandas) (2.8.2)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/site-packages (from pandas) (1.23.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/site-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
```

In [104...]

```
import os
from tqdm import tqdm # tqdmライブラリをインポート
import numpy as np
import matplotlib.pyplot as plt
import IPython.display
from IPython.display import display
import pandas as pd

import librosa
import librosa.display
import librosa.util
```

In [105...]

```
# pyplot: matplotlibパッケージ内のモジュール.欲しいプロットを作るために暗黙的かつ自動的に図形
# 以下でpyplotのデフォルト値を設定しておく
plt.rcParams.update({
    'font.size' : 10,
    'font.family' : 'Meiryo' if os.name == 'nt' else 'DejaVu Sans' # Google
    , 'figure.figsize' : [10.0, 5.0]
    , 'figure.dpi' : 300
    , 'savefig.dpi' : 300
    , 'figure.titlesize' : 'large'
    , 'legend.fontsize' : 'small'
    , 'axes.labelsize' : 'medium'
    , 'xtick.labelsize' : 'small'
    , 'ytick.labelsize' : 'small'
})
```

In [106...]

```
# 多次元配列のデータ構造 ndarray の表示設定
np.set_printoptions(threshold=0) # 可能ならndarrayを省略して表示
np.set_printoptions(edgeitems=1) # 省略時に1つの要素だけ表示
```

6.2.2 音声の読み込みとデータの確認

In [102...]

```
# MP3音声データを カレントディレクトリ直下のフォルダ(一時作業領域)へダウンロードする.
# サンプル音声素材 tam-x05.mp3 クリスマス風 ゆっくり目 (Copyright© TAM Music Factory A
!wget -nc https://raw.githubusercontent.com/MDASH-shinshu/MDASH-T-DE/main/2/
# wgetしなくてもGoogle colab.の左メニュー [ファイル] アイコンをクリックして, ブラウザへファイルを
```

```
# 音声データがダウンロード・配置できたことを確認する
!ls -al ./

--2022-09-03 14:43:04-- https://raw.githubusercontent.com/MDASH-shinshu/MDASH-T-DE/main/2/resources/tam-x05.mp3
raw.githubusercontent.com (raw.githubusercontent.com) をDNSに問い合わせています...
185.199.110.133, 185.199.111.133, 185.199.109.133, ...
raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.133|:443
に接続しています... 接続しました。
HTTP による接続要求を送信しました、応答を待っています... 200 OK
長さ: 870416 (850K) [audio/mpeg]
`tam-x05.mp3' に保存中

tam-x05.mp3          100%[=====] 850.02K 2.13MB/s 時間 0.4s

2022-09-03 14:43:07 (2.13 MB/s) - `tam-x05.mp3' へ保存完了 [870416/870416]

total 288
drwx-----@ 1 wasaki  staff   16384  9  3 14:43 .
drwx-----@ 1 wasaki  staff   16384  9  3 14:00 ..
-rwx-----@ 1 wasaki  staff   10244  9  3 14:01 .DS_Store
drwx-----@ 1 wasaki  staff   16384  9  3 14:00 1
drwx-----@ 1 wasaki  staff   16384  9  3 14:41 2
drwx-----@ 1 wasaki  staff   16384  8 28 13:31 3
drwx-----@ 1 wasaki  staff   16384  9  3 14:37 CRF++-0.58
drwx-----@ 1 wasaki  staff   16384  8 10 11:53 _backups
drwx-----@ 1 wasaki  staff   16384  9  3 14:03 _resources
-rwx-----@ 1 wasaki  staff  790570  9  3 14:35 crfpp.tar.gz
drwx-----@ 1 wasaki  staff   16384  8 10 11:48 ex0
-rwx-----@ 1 wasaki  staff  1147470  9  3 14:42 output.jpg
-rwx-----@ 1 wasaki  staff  789195  9  3 14:41 sample.jpg
-rwx-----@ 1 wasaki  staff  870416  9  3 14:43 tam-x05.mp3
```

In [108...]

```
# 上記でダウンロードした音声ファイルを設定
audio_path = 'tam-x05.mp3'; print(audio_path)

# あるいは,librosaに標準で入っているサンプル音声ファイルを読み込み (https://librosa.org/librosa.util.example_audio_file.html)
# サンプル音声素材 Kevin_MacLeod_-_Vibe_Ace.hq.ogg ポップアップ 速めのテンポ
#audio_path = librosa.util.example_audio_file(); print(audio_path)

# ターゲットとするサンプリング周波数で読み込む
y, sr = librosa.load(audio_path) # サンプリング周波数 22.05kHzで読み込む (default)
# y, sr = librosa.load(audio_path, sr=None) # オリジナルのサンプリング周波数で読み込む
# y, sr = librosa.load(audio_path, sr=4096) # 4kHzで re-samplingして読み込む

# 読み込み結果の表示
print([type(y), y.shape], [type(sr), sr]) # sr はSampling Rate(サンプリング周波数)
```

tam-x05.mp3

```
/usr/local/lib/python3.10/site-packages/librosa/util/decorators.py:88: UserWarning: PySoundFile failed. Trying audioread instead.
    return f(*args, **kwargs)
[<class 'numpy.ndarray'>, (1597760,)] [<class 'int'>, 22050]
```

6.2.3 音声をプレーヤーと波形可視化で確認

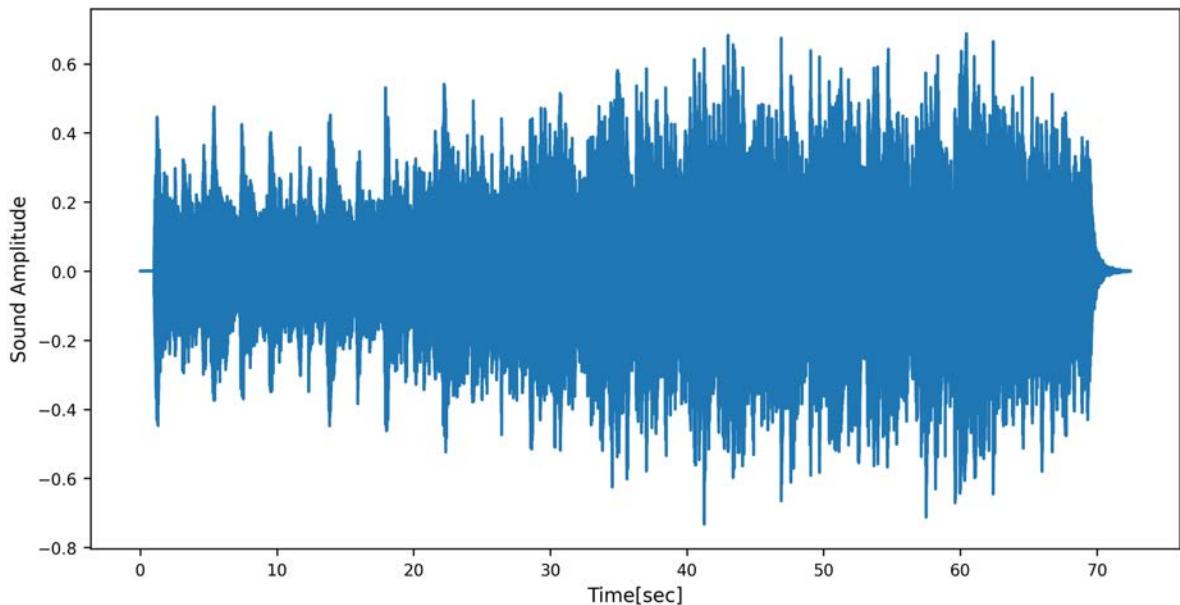
In [109...]

```
# IPython経由でブラウザの音声プレーヤを呼び出して読み込んだデータ列 y を サンプリングレート sr で再生する
# IPython.display.display() ドキュメント https://ipython.readthedocs.io/en/stable/api/generated/IPython.display.Audio.html
display(IPython.display.Audio(y, rate=sr))
```

▶ 0:00 / 1:12 ⏸ ⏹ ⋮

```
In [110]: time = np.arange(0, len(y)) / sr # 時間 = yのデータ数 / サンプリング周波数sr
plt.plot(time, y) # xにtime, yにyとしてプロット
plt.xlabel("Time[sec]") # x軸とy軸にラベルを設定(x軸は時間、y軸は振幅)
plt.ylabel("Sound Amplitude")

plt.show() # matplotlib経由でx-yグラフを表示
```

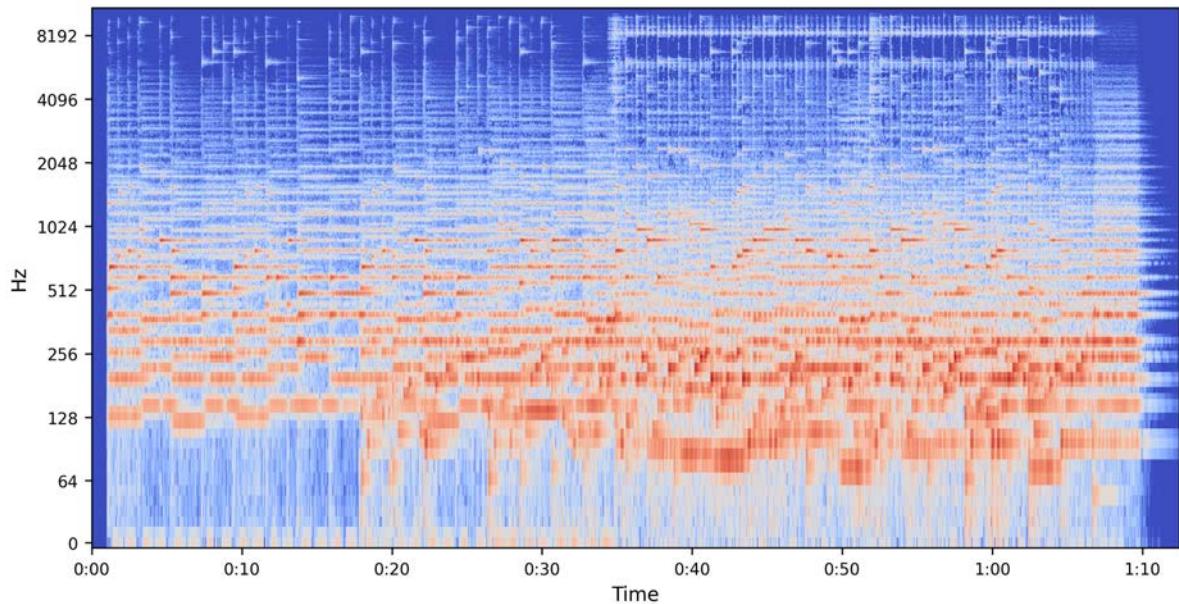


6.2.4 スペクトログラムへの変換

音声信号に対して、時間 \Rightarrow 周波数空間変換、具体的には「短時間フーリエ変換（Short-time Fourier transform : STFT）」を行うことで、スペクトログラムが得られる。スペクトログラムとは、音声を周波数スペクトルの時間経過で示したものである。

```
In [115]: D = librosa.stft(y) # Short-time Fourier transform : STFT (default FFTウィンドウ)
S, phase = librosa.magphase(D) # FFT変換後は複素数となっているので、それを 強度S と 位
Sdb = librosa.amplitude_to_db(S) # 強度S を デジベル(音圧)Sdb へ変換
librosa.display.specshow(Sdb, sr=sr, x_axis='time', y_axis='log') # スペクトログラムを表示
```

Out[115]: <matplotlib.collections.QuadMesh at 0x1929912a0>



```
In [112]: tempo, beat_frames = librosa.beat.beat_track(y=y, sr=sr) # BPM(音楽の場合, 1分間  
print(tempo)
```

86.1328125

6.2.5 ピッチのシフト、時間のストレッチなどの加工例

```
In [113...]: # ピッチシフト：1オクターブ上にしてみる  
label = "1 octave down"  
eps = 1.0  
n_fft = 2048  
y_shift = librosa.effects.pitch_shift(y, sr, n_steps=+12) # 1オクターブ上下: n_  
##plt.plot(np.log(np.abs(np.fft.fft(y_shift))**2+eps)[0:n_fft//2+1], label=l  
display(IPython.display.Audio(y_shift, rate=sr))
```

```
/var/folders/zk/s10ljgvj0ql0g5f0b61h5dm80000gn/T/ipykernel_16403/388626839  
6.py:5: FutureWarning: Pass sr=22050 as keyword args. From version 0.10 pas-  
sing these as positional arguments will result in an error  
    y_shift = librosa.effects.pitch_shift(y, sr, n_steps=+12) # 1オクターブ上下:  
n_steps=+12. 完全5度上(半音7個分): n_steps=7 など
```

0:00 / 1:12

```
In [114]: # 1/2倍にタイムストレッチ  
y_slow = librosa.effects.time_stretch(y, 1./2)  
##plt.plot(np.log(np.abs(np.fft.fft(y_slow))**2+eps)[0:n_fft//2+1], label=label)  
display(IPython.display.Audio(y_slow, rate=sr))
```

```
/var/folders/zk/s10ljgvj0ql0g5f0b61h5dm80000gn/T/ipykernel_16403/227959400  
2.py:2: FutureWarning: Pass rate=0.5 as keyword args. From version 0.10 pas-  
sing these as positional arguments will result in an error  
    y_slow = librosa.effects.time_stretch(y, 1./2)
```

memo

