

# データサイエンス特論

## Data Science

# データの取得と操作

1. データの格納
2. 収集と加工(操作)
  - 折れ線グラフ, ヒストグラム, 散布図, パイチャート
3. 付録



# データの格納

2

1. データが格納される次の3クラスの説明
  - `numpy.ndarray`
  - `pandas.Series`
  - `pandas.DataFrame`

本講義で、データは、NumPyとpandasの両方のみで扱うものとする。  
他の方法は扱わない。

両者のフォーマット、基本的な文法を例題を通して説明する。  
更に詳しい内容は、各自で調べること。



## □ numpyの配列, np.ndarray クラス

- <http://www.numpy.org/>
- <http://www.kamishima.net/mlmpyja/nbayes1/ndarray.html>
- N次元配列を扱うためのクラス, 1次元の場合はベクトルに, 2次元の場合は行列に, そして3次元以上の場合はテンソルに相当。
- 高速数値計算を行うため, 次の制約がある。
  - 配列内要素の型は全て同じ
  - 配列長は固定 (固定長配列)
  - 配列の各次元の要素数は同じ

In [12]:

```
1 x = np.arange(10)
2 print(x)
3 print(type(x))
```

```
[0 1 2 3 4 5 6 7 8 9]
<class 'numpy.ndarray'>
```

## □ リスト

- 本講義では使わないが, numpyと混同しないために触れる。
- シーケンス型の一つと言われる。この意味は, 要素が順序(シーケンス)立てて格納されていることである。配列の一つと考えても良い。
- 要素の型は一致しなくても良く, 数値, 文字が混在してもよい。
- 高速演算には向かない

```
: 1 #list
2 val = [10, 11, 'test']
3 val[2]
```

```
: 'test'
```

```
: 1 val = [
2 [1,3,5,7,9],
3 [2,4,6,8,10]
4 ]
5 print(val[1][4])
```

10

## □ テキストファイルの読み込み

- delimiter: データの区切り記号, skiprows: 1行目を読み飛ばす
- data = np.loadtxt("foo.csv", delimiter=",", skiprows=1)



# DataFrameにcsvファイルを読み込む

## indexとcolumns

data\_pandas\_introduction.csv  
(フォルダValuable\_Kit内にある)

見てみよう

	A	B	C	D	E
1	id	Class	Sex	Age	Height
2	A001	3	male	22	182
3	A002	1	female	38	158
4	A003	2	female	26	155
5	A004	1	female	35	162
6	A005	3	male	35	178

### 注意:

columnsは横に並んでいる複数のラベルを総称しているの  
であって, column指定は列(縦方向)を意味する。例えば,  
columnの“Age”を指定すると, この列(縦方向)が指定さ  
れる。indexも同様である。

index, columnsのそれぞれの文字または数字をラベル  
(label)と称する。

```
1 df = pd.read_csv("data_pandas_introduction.csv")
2 df
```

### index

ユーザが何も指定  
していないと, 自  
動的に0番から順  
序数が割り振られ  
る

id Class Sex Age Height

columns

0	A001	3	male	22	182
1	A002	1	female	38	158
2	A003	2	female	26	155
3	A004	1	female	35	162
4	A005	3	male	35	178

このNotebook  
を自作してみよう

```
1 print(df.columns)
2 print(df.index)
```

```
Index(['id', 'Class', 'Sex', 'Age', 'Height'], dtype='object')
RangeIndex(start=0, stop=5, step=1)
```



# DataFrameにindexを指定する

## □ 方法1:新たにindexとなる列を追加

- この例では, indexに数字と文字を混在させている
  - `df.index=[10, 11, 12, 13, 'end']`
  - `df`

	id	Class	Sex	Age	Height
10	A001	3	male	22	182
11	A002	1	female	38	158
12	A003	2	female	26	155
13	A004	1	female	35	162
end	A005	3	male	35	178

## □ 方法2:もともと, csvにindexがある場合

- この例では, ラベルidの列をindexに指定して読み込むため, `index_col =` を指定
  - `df = pd.read_csv("data_pandas_introduction.csv", index_col='id')`
  - `df`

	Class	Sex	Age	Height
id				
A001	3	male	22	182
A002	1	female	38	158
A003	2	female	26	155
A004	1	female	35	162
A005	3	male	35	178



# pandas.read\_csv

## □ パラメータの説明

```
df = pd.read_csv('foo.csv', index_col='Date', parse_dates=[0], encoding='SHIFT-JIS', names=('Date', 'Day', 'Item', 'Content', 'Expense'))
```

- `index_col`: インデックスとする列
- `parse_dates`: 日付として扱う列を指定するパラメータ。日付の表記は、2017/09/15, 15-Sep-2017, Setp-15-17 などと各種ある。そのため、DataFrameに格納するとき、これを解析(`parse`)して、ある統一した年月日表記にすることが必要です。これを行うのがパラメータ `parse_dates` です。 `= [0]` は0列目を指しており、この例では `index_col` と同じになる。
- `encoding`: データのエンコーディングを指定
  - デフォルトは `uft-8`
  - Excelで作成したcsvファイルはShift JISで保存されているので、`encoding='SHIFT-JIS'` の設定となる。日本語を含まないときは、これと異なることがある。
  - Python encoding表は、`codecs — Codec registry and base classes`
  - <https://docs.python.org/3/library/codecs.html>
- `names: columns` の位置に (通常は1行目) ラベルが記述されていないとき、ラベル名を与える。



# pandas, indexとcolumns

## □ indexの追加 ⇒


- 数字と文字を混在できる
- columnsも同様の操作ができる

```
1 df.index=[10, 11, 12, 13, 'end']
2 df
```

	Survived	Pclass	Sex	Age	Fare
10	0	3	male	22	7.2500
11	1	1	female	38	71.2833
12	1	3	female	26	7.9250
13	1	1	female	35	53.1000
end	0	3	male	35	8.0500

## □ あるcolumnの名前変更

- indexも同様の操作ができる



```
1 df.rename(columns={'Pclass': 'Class'})
```

	Survived	Class	Sex	Age	Fare
10	0	3	male	22	7.2500
11	1	1	female	38	71.2833
12	1	3	female	26	7.9250
13	1	1	female	35	53.1000
end	0	3	male	35	8.0500



## □ 行, 列の指定

- `at`, `iat` : 単独要素の指定
- `loc`, `iloc`: 複数要素を選択、取得・変更  
接頭に”i“が付くと0以上の自然数指定,  
付かないとラベル名(文字)指定
- 右の例のように  
列を指定し, 全ての行を含む場合は  
“:”を指定する。

備考: **ix**は古く, **非推奨**

(Indexing and Selecting Data)

[http://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html](http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html)

## □ DataFrame 他の操作

- 追加, 削除, 値の入替などは, 他の成書  
または, Webサイトを参照

```
In [30]: 1 df.loc[1:2]
Out [30]:
```

	Survived	Pclass	Sex	Age	Fare
1	1	1	female	38	71.2833
2	1	3	female	26	7.9250

```
In [39]: 1 df.loc[:, 'Age'] #全ての行は ':' を指定
Out [39]: 0    22
          1    38
          2    26
          3    35
          4    35
          Name: Age, dtype: int64
```





# pandas series

## □ 時系列分析でよく用いる

- 1次元配列

## □ pandas.date\_range

- `pd.date_range('yyyymmdd', period=num, freq='Offset')`
- `yyyymmdd`: ex. 1/15/2001, 2001-1-15
- `period`: タイムスタンプを何回与えるか(回数)
- `freq`: 時間間隔, 例, 'H':1時間, 'D':日, 'W':週, 'M':月, 'Q':四半期(4か月), 'Y':年, 他

<http://pandas.pydata.org/pandas-docs/stable/timeseries.html>

タイムスタンプ(timestamp):これがなぜ必要か?

Ref.:

- <https://ja.wikipedia.org/wiki/タイムスタンプ>
- サイト <https://www.soumu.go.jp/> 内で “タイムスタンプ”を検索

Ref. is short for Reference



# pandas series

## □ 作成例

➤ データを作成してから、タイムスタンプを与える例

```
• nobs = 10 # the number of observation
• np.random.seed(123)
• y0 = np.random.normal(loc=1.5, scale=2.0, size=nobs) # loc is mean, scale is standard
  deviation
• index = pd.date_range('1/1/2000', periods=nobs, freq='Y')
• y = pd.Series(y0, index=index)
```

nobs: the number of observation

## □ タイムスタンプをindexに指定

- 時系列データでは、タイムスタンプをindexに指定することが多い。
- 上記の例では、pd.Series (indx = index)
- csvファイルを読み込むときは read\_csv(index\_col = 'ラベル名') , ラベル名はタイムスタンプの列を表すものである。



# numpyとDataFrame

## □ DataFrame → nd.ndarray

```
val = df['Age'].values
print(type(val))
<class 'numpy.ndarray'>
```

## □ nd.ndarray をDataFrameに追加

```
val = np.array([14,15,16])
df['D'] = val # 'D'は新たなcolumnを追加

nd_val = np.random.normal(size=10) # type: numpy.ndarray
series = pd.Series(nd_val) # type: pandas.core.series.Series
df = pd.DataFrame(nd_val) # type: pandas.core.frame.DataFrame
```



# numpy.ndarray <-> pandas.DataFrame 相互変換

numpy.ndarray ⇒ pandas.DataFrame に相互変換

この変換は、単純に次の表のように行えばよい。

変数の作成	type
<code>x = np.random.normal(size=10)</code>	<code>numpy.ndarray</code>
<code>y = pd.Series(x)</code>	<code>pandas.core.series.Series</code>
<code>df = pd.DataFrame(x)</code>	<code>pandas.core.frame.DataFrame</code>

pandas.DataFrame ⇒ numpy.ndarray に変換

この変換は、メソッド `values` を用いて、次のように行えばよい。

```
val = df[ 'Height' ].values
print (type (val))
```

```
<class 'numpy.ndarray'>
```



# 収集と加工（操作）

13

1. データの種類
2. データを収集
3. データを加工
  - クロス集計とヒートマップ



# データの種類

## □ 人工データ(artificial data)

- 各自が関数などを用いて作成

## □ オープンデータ(open data)

- 二次利用が可能な利用ルールで公開されたデータ
- オープンデータとは(総務省): [http://www.soumu.go.jp/menu\\_seisaku/ictseisaku/ictriyou/opendata/index.html](http://www.soumu.go.jp/menu_seisaku/ictseisaku/ictriyou/opendata/index.html)

## □ オープンデータの種類

- 次にまとめてある
- オープンデータセット(Open Data Sets) (橋本): <https://sites.google.com/site/datasciencehiro/datasets>
- Kaggle: 世界最大のデータサイエンティストコミュニティを形成し、データ分析やモデル開発のコンペティションを行うサイト
- UC Irvine Machine Learning Repository: カリフォルニア大学アーバイン校(University of California, Irvine)が運営, 機械学習やデータマイニングに関するデータの配布サイト
- scikit-learn Datasets Package: scikit-learnが提供するデータセット
- StatsModels Datasets Package: StatsModelsが提供するデータセット
- 政府e-Stat: 日本の統計が閲覧できる政府統計ポータルサイト
- 気象庁国土交通省: 気象データを取得できる
- 富山県人口移動調査: 富山県の人口移動に関するデータを取得できる
- 東京電力: 過去の電力使用量データを取得できる



# 演習： データの収集と加工

- 実際にデータを収集
- それを使えるように加工(修正)
- グラフで表現



# 例：地域別完全失業率

## □ 労働力調査長期時系列データ(総務省統計局)

- <http://www.stat.go.jp/data/roudou/longtime/03roudou.html>
- 長期時系列データ(基本集計)→表8(このページの下の方)→地域別完全失業率(エクセル: 100KB)(1983年～)
- これをクリックすると“**lt08-02.xls**”がダウンロードされる(フォルダValuable\_Kit内にあり)
- これを用いて、次のデータを作成する(動画説明)

## □ データの作成

- タブ「原数値」を開く
  - (季節調整値 <http://www.stat.go.jp/data/roudou/pdf/point04.pdf> は用いない)
- 対象:
- 地域:北海道, 南関東, 北陸, 近畿
- ラベル名(英語)をそのまま:Hokkaido, Southern-Kanto, Hokuriku, Kinki
- 平成24年1～3月から平成31年10～12月の期間
- これを, “**Data.csv**”として保存(各自で作成するものです)。

## □ グラフ化(プロット)

- 折れ線グラフ, ヒストグラム, 散布図, パイチャート
- pandasとmatplotlibの両方で試す
- このグラフを見て, 何が言えるか? さらに, 仮説を考える。

このデータとプログラムを出力したHTMLファイル“[test\\_Plot\\_UnemploymentRate.html](#)”を渡します(フォルダ内にもあり)。





# 動画 : It08-02.xls⇒Data.csvへの編集

It08-02.xls (全ページで紹介、Valuable\_Kit内にあり)の編集の様子

It08-02.xls [互換モード] - Excel

ファイル タッチ ホーム 挿入 ページレイアウト 数式 データ 校閲 表示 ヘルプ 実行したい作業を入力してださい

元に戻す フォント(F): Times New 1 フォント サイズ(F): 10 文字拡大 文字縮小 フォントの色

元に戻す フォント 手がき 編集 基本

R2C3

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	[基本集計]			長期時系列表 8 (3) 地域別完全失業率												
2	[Basic Tabulation]			Historical data 8 (3) Unemployment rate - Regions												
3																
4				(%)	(percent)											
5				季節調整値 Seasonally adjusted series												
6		四半期		北海道	東北	南関東	北関東・甲信	北陸	東海	近畿	中国・四国	九州・沖縄				
7		Quarter		Hokkaido	Tohoku	Southern-Kanto	Northern-Kanto, Koshin	Hokuriku	Tokai	Kinki	Chugoku, Shikoku	Kyushu, Okinawa				
8	昭和 58 年	1~3月	Jan.-Mar.	4.7	2.7	2.7	1.6	2.0	1.8	3.0	2.5	3.4				
9	1983	4~6月	Apr.-Jun.	4.0	2.8	2.6	1.6	1.6	1.8	3.0	2.7	3.8				
10		7~9月	Jul.-Sep.	4.5	3.0	2.6	1.7	2.1	1.9	3.0	2.6	3.5				
11		10~12月	Oct.-Dec.	3.8	2.9	2.5	1.8	2.3	1.7	2.8	2.9	3.5				
12	昭和 59 年	1~3月	Jan.-Mar.	4.6	3.1	2.4	1.6	2.1	1.8	3.3	3.0	3.5				
13	1984	4~6月	Apr.-Jun.	4.8	2.8	2.6	1.8	1.9	2.0	2.9	2.8	3.7				
14		7~9月	Jul.-Sep.	3.9	2.7	2.6	1.7	1.6	2.1	3.0	2.7	3.9				
15		10~12月	Oct.-Dec.	4.6	2.7	2.5	1.5	1.9	2.0	2.8	2.8	3.6				
16	昭和 60 年	1~3月	Jan.-Mar.	4.4	2.7	2.4	1.6	1.8	2.0	2.6	2.8	3.6				
17	1985	4~6月	Apr.-Jun.	4.8	2.8	2.4	1.6	1.9	1.8	2.7	2.7	3.3				
18		7~9月	Jul.-Sep.	4.3	2.7	2.5	1.6	1.6	1.8	2.7	2.6	3.5				
19		10~12月	Oct.-Dec.	4.6	2.9	2.6	1.8	1.6	2.2	3.1	2.8	3.6				
20	昭和 61 年	1~3月	Jan.-Mar.	4.4	2.8	2.6	1.8	1.8	2.0	2.9	2.7	3.8				
21	1986	4~6月	Apr.-Jun.	4.2	2.7	2.6	1.7	2.3	2.0	3.2	2.7	3.9				
22		7~9月	Jul.-Sep.	4.6	2.8	2.7	1.8	2.4	2.0	3.3	3.0	3.9				
23		10~12月	Oct.-Dec.	4.6	2.6	2.6	1.8	2.3	2.0	3.2	2.8	4.0				
24	昭和 62 年	1~3月	Jan.-Mar.	4.1	2.8	2.8	1.9	2.4	2.1	3.4	2.9	4.1				
25	1987	4~6月	Apr.-Jun.	4.6	2.9	2.9	1.9	2.3	2.1	3.5	3.1	4.1				
26		7~9月	Jul.-Sep.	4.3	2.7	2.5	1.8	2.3	2.0	3.1	3.0	3.9				
27		10~12月	Oct.-Dec.	4.2	2.6	2.5	1.7	1.9	1.8	3.2	2.6	3.8				
28	昭和 63 年	1~3月	Jan.-Mar.	4.0	2.5	2.5	1.7	2.0	1.8	3.2	2.9	3.6				
29	1988	4~6月	Apr.-Jun.	3.4	2.4	2.3	1.8	1.7	1.9	2.8	2.5	3.5				
30		7~9月	Jul.-Sep.	4.2	2.5	2.4	1.5	1.9	1.8	2.8	2.7	3.5				
31		10~12月	Oct.-Dec.	3.4	2.3	2.4	1.5	1.9	1.7	2.8	2.4	3.1				
32	平成 元年	1~3月	Jan.-Mar.	3.3	2.2	2.3	1.5	1.7	1.4	2.9	2.2	3.4				
33	1989	4~6月	Apr.-Jun.	3.4	2.2	2.3	1.4	1.8	1.6	2.7	2.5	3.1				
34		7~9月	Jul.-Sep.	3.0	1.8	2.2	1.5	1.5	1.7	2.9	2.0	2.8				

季節調整値 原数値 季節指数 地域区分

100%

# 動画 : It08-02.xls⇒Data.csvへの編集

Data.csvに、余計なデータが混入していると、スクリプトエラーが生じる。  
余計なデータの除去: 動画のように、データの周辺の複数の行や列を削除。慣れている人は、適当なエディタで開けばどこに余計なデータがあるか眼で見て分かるので、手作業でそれを削除。

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Hokkaido	Southern-IHokuriku	Kinki											
2		5.3	4.5	3.9	5.3									
3		5.3	4.4	3.4	5.2									
4		5.4	4.4	3.5	4.8									
5		4.8	4.3	3.2	4.8									
6		4.8	4.4	3.4	4.9									
7		4.8	4	3.5	4.3									
8		4	4	3.4	4.1									
9		4.5	3.9	3.2	4.2									
10		4.3	3.6	3.1	4.2									
11		4	3.6	3.2	4.2									
12		3.6	3.5	3.2	4.1									
13		4.4	3.4	3	3.9									
14		3.6	3.4	2.7	3.9									
15		3.4	3.4	2.8	3.7									
16		3.5	3.3	2.5	4.1									
17		3.4	3.1	2.8	3.6									
18		3.6	3.3	3	3.5									
19		3.5	3.1	2.6	3.8									
20		4	3.1	2.7	3.5									
21		3.4	3.2	2.5	3.2									
22		3.6	3	2.5	3.1									
23		3.4	3	2.7	3									
24		3.3	2.9	2.7	2.8									
25		3	2.8	2.3	2.9									
26		2.9	2.6	2.1	2.8									
27		3	2.5	1.9	2.7									
28		2.8	2.5	1.8	2.8									
29		3	2.3	2.2	3.1									
30														
31														
32														
33														
34														



# 付録

データサイエンス、機械学習、統計、人工知能のアプリケーションは、全て、コンピュータ上で計算を行っている。

コンピュータはCPUとメモリを有しており、数字は離散表現かつ有限長であるため、様々な問題や特性を有している。

このことを知ることともデータサイエンティストとして重要な教養である。



# 正規化と標準化

データの正規化 (normalization), 標準化(standardization), ベクトルの正規化 (vector normalization) を見る  
 正規化: 最小値0, 最大値1とする

$$x_{norm}^i = \frac{x^i - x_{min}}{x_{max} - x_{min}}$$

標準化: 平均値0, 標準偏差1の確率変数とする

$$x_{std}^i = \frac{x^i - \mu}{\sigma}$$

ベクトルxの正規化: ベクトルのノルムを1とする

$$\frac{x}{|x|}$$

この操作が必要となる場合は？

## □ 正規化

- 最小値を0, 最大値を1に変換する
- 比較が容易になる(比率として見ることができるため)

## □ 標準化

- 確率変数が対象
- 平均値0, 標準偏差を1に変換する
- 標準正規分布に従う確率変数を見たいときに用いる

## □ ベクトルの正規化

- ベクトルのノルム(大きさ)を1にする

## □ 注意点:

- むやみに, 正規化, 標準化を行って良いものではない
- 分布の中心が移動する。例えば, 街のある交差点を中心に発生する交通事故, 温度変化など, その位置に意味がある場合, 中心が移動すると, データの意味を失うことがある。



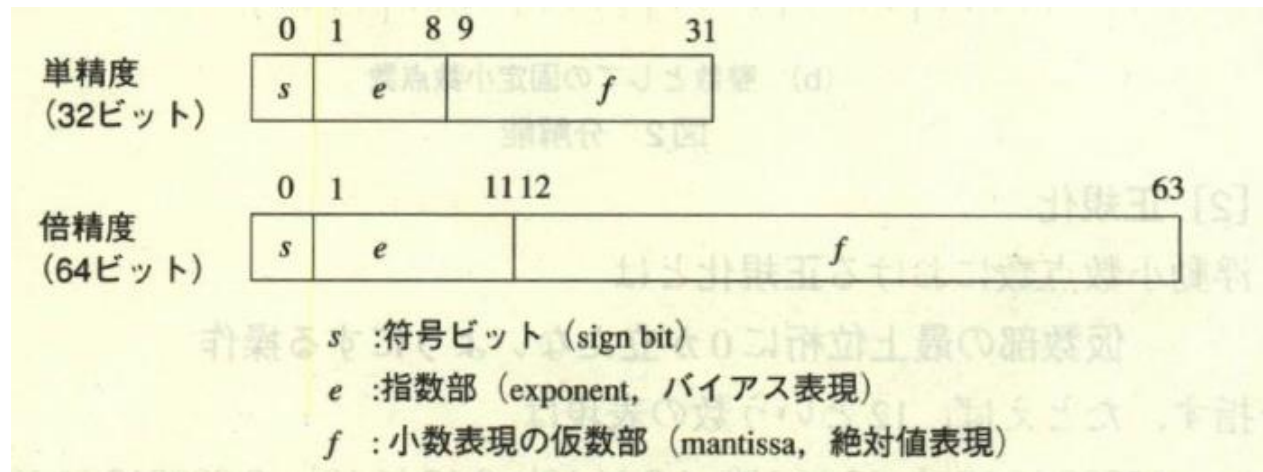
# 有限桁に起因する問題

## □ 有限桁数で打ち切り

➤ 例えば、1.234567890123456789 は表現できない

## □ 浮動小数点数 IEEE754規格

- 単精度 小数表現の有効桁数23ビット →10進数約7.2桁、 $10^{\{\pm 38\}}$
- 倍精度 小数表現の有効桁数52ビット →10進数約16.0桁、 $10^{\{-308\sim+307\}}$



## □ 10進数の0.1は、厳密に2進数に変換できない

- $0.1_{10} = 0.00011_{2}$  (赤文字で循環する)
- 参考文献: 橋本, 他: 図解コンピュータ概論, オーム社, 平成29年



# 丸め, 情報落ち, 桁落ち

## □ 丸め

- 切り上げ, 切捨て, 四捨五入(五捨六入)など, 有効桁数を超える値の処理に伴う誤差をいう
- 整数部が2桁のみならば, 小数第1位を何とかしなければならない。

## □ 情報落ち

- 絶対値の大きさが著しく異なる値の加減算で生じる誤差である
- 有効桁数が4桁のメモリの場合:  $1.000 \times 10^4 + 1$ としても, 1が無視される。
- よくある例: 100万人の顧客情報の加算, 微分方程式の数値解法で0.001を100万回加算, これらは情報落ちの可能性が高まる。

## □ 桁落ち

- ほぼ同じの値の減算を行った結果, 有効数字が減少することをいう
- 4桁の有効数字  $1.000 \times 10^4 - 9.999 \times 10^3 = 1 \times 10^0$  であるが,

$$\begin{array}{r} 1000 \times 10 \\ - 9999 \times 1 \\ \hline \end{array}$$

これが4桁有効数字の引き算となり, 有効数字上位3桁は無くなり, 最下位1桁のみが有効となる。すなわち, 結果として得られる 1 は有効桁数が1桁, もし, ある計算を行って,  $1.000 \times 10^0$  が得られたとしても, 小数部の “.000” は偶然にもこの数字なのか, 桁落ちした数字なのか判別できない。



# 方程式の演算誤差

## □ 演算 (operation)

加減乗除、全ては、基本的に加算で計算する。(減算は符号反転、乗算は加算の繰返し、除算は減算の繰返し)

これまで見てきたように、演算を行うと誤差が生じる

## □ 連立一次方程式の解き方

- ▶ 誤った解き方; 数学の教科書では、そのため、7の逆数から求める。

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

$$7x = 21, \quad \text{strict sol.} = 3$$

$$x = (7^{-1})(21) \approx 0.14 \cdot 21 = 2.94$$

- ▶ 正しいとき方、両辺を同時に7で除算を行う。  
精度の点と演算量が少ない。

## □ 参考

- ▶ 連立一次方程式にはガウスの消去法が用いられることがある
- ▶ これは、行列の質(条件数で表される)が良いときで、7元(元は式の数)まででない、解の精度がよくないことが経験的に知られている。
- ▶ そのときは、反復法などを適用すべきである。



# 機械イプシロン

## □ 数字分解能の変化

- 数の表現が先のIEEE表現に基づくとき
- 数の絶対値が小さいとき、数字分解能(数における隣との距離)は小さく、絶対値が大きいときは大きい

## □ 機械イプシロン(machine epsilon)

INT\_Epsilon\_Newton

- 定義は、「1より大きい最小の数」(= $A$ とおく)と1との差である。
- 式で表すと、計算機イプシロン  $\epsilon = A - 1$ 、言い換えると 1に $\epsilon$ を足しても、計算結果が1となるくらい小さな数を言う。
- IEEE754, 倍精度(64ビット)では、 $\epsilon = 2.220446049250313e-16$ となる

## □ 収束判定

- 尤度関数などの非線形計算では、数値解の正確さを上げるために、反復計算が行われる。この際、無限に細かい精度を追い求めることはできず、計算を有限時間で留めるために、収束判定が行われる。
- 例えば、古典的で有名なニュートン法で  $f(x) = x^2 - 9$  の解を求める。初期値  $x_0 = 1.0$  から開始し、収束判定のためのイプシロン=0.01 とすると、数値解(近似解) = 3.00009155413138 を得た。厳密解3に誤差が加わる





# おわりに

## □ 有限桁数に伴う誤差

- 数値計算の過程の多くのステージにおいて、誤差が混入、発生することわかった。
- よって、Pythonパッケージが計算した値を絶対正しい、とは思うことは誤りに嵌まる危険性がある。
- データを得た背景、データが産み出されたメカニズム(またはモデル)と環境、使用したい条件などの多角的条件を念頭において、得られた数値結果を客観的かつ論理的に評価できる能力を身に付けてほしい。

