明石工業高等専門学校専攻科

専攻科特別講義レポート課題

情報可視化レポート

ME2208 髙橋 尚太郎
(機械・電子システム工学専攻１年)

目 次

1　仕様と実現方法

1. たまにある単純作業の繰り返しの作業は何か
   光学式測距センサ (LiDAR) で得た距離データ (センサデータ) のばらつきを統計的に処理したい。ロボットの自律移動を行うために、ロボットの自己位置・速度の推定を行う必要がある。一例として、LiDAR を用いてランドマークとの距離を計測し、自己位置を得る手法が提案されている。ロボットに搭載するセンサデータは、センサ内部の状態や周囲の環境等、さまざまな影響を受けて出力される。

2. どのようなデータを可視化したいか
   センサデータに含まれる誤差 (ノイズ) の傾向を可視化したい。

3. どのような機能があればそれが実装できそうか
   統計処理で扱う関数やモジュールと、それを可視化するためのツールがあれば実装できる。センサデータを解析するために、python において、以下に示したツールを用いることが想定される。

   - 距離データが格納されたファイルの読み込み
   - 配列等への格納
   - 確率分布、平均、分散、標準偏差、正規分布を扱うためのモジュール
   - 数値の可視化 (グラフ化ツール)

   順にこれらを実装することで、測距データに含まれるノイズの傾向が明らかになる。

2　実装と解説

前章の機能を用いて、以下の手順で機能を実装する。

2.1　センサデータの読み込みとヒストグラムの描画

センサデータを用意する。(sensor_data_200.txt) ソースコード 1[1] を実行することで、date、time、ir、lidar の列要素からなる配列を得る。実行結果を図 1 に示す。

ソースコード 1: センサデータの読み込み

```
1 import pandas as pd # pandasファイル読み込み・統計処理モジュールのインポート:
2 data = pd.read_csv("sensor_data_200.txt", delimiter=" ",
3 header=None, names = ("date","time","ir","lidar"))
4 # read_csvテキストデータを読み込む関数。引数に列の属性を指定する。:
5 data
```

	date	time	ir	lidar
0	20180122	95819	305	214
1	20180122	95822	299	211
2	20180122	95826	292	199
3	20180122	95829	321	208
4	20180122	95832	298	212
...	...	...	...	...
58983	20180124	120023	313	208
58984	20180124	120026	297	200
58985	20180124	120030	323	204
58986	20180124	120033	326	207
58987	20180124	120036	321	208

58988 rows × 4 columns

図 1: 実行結果

例えば、"lidar" 列の 0 ～ 5 番目の要素を抜き出して表示する場合は、ソースコード 2[1] を実行する。実行結果を図 2 に示す。

ソースコード 2: 要素の抜き出し表示

```
1 print(data["lidar"][0:5])
```

```
0    214
1    211
2    199
3    208
4    212
Name: lidar, dtype: int64
```

図 2: 実行結果

ヒストグラムの描画はソースコード 3[1] を実行する。実行結果を図 3 に示す。

ソースコード 3: ヒストグラムの描画

```
1 import matplotlib.pyplot as plt # 描画モジュールのインポート
2 data["lidar"].hist(bins = max(data["lidar"]) - min(data["lidar"]),align='left')
3 # ヒストグラムのビン数をセンサデータの最大値と最小値の差に設定
4 plt.show() # 描画
```

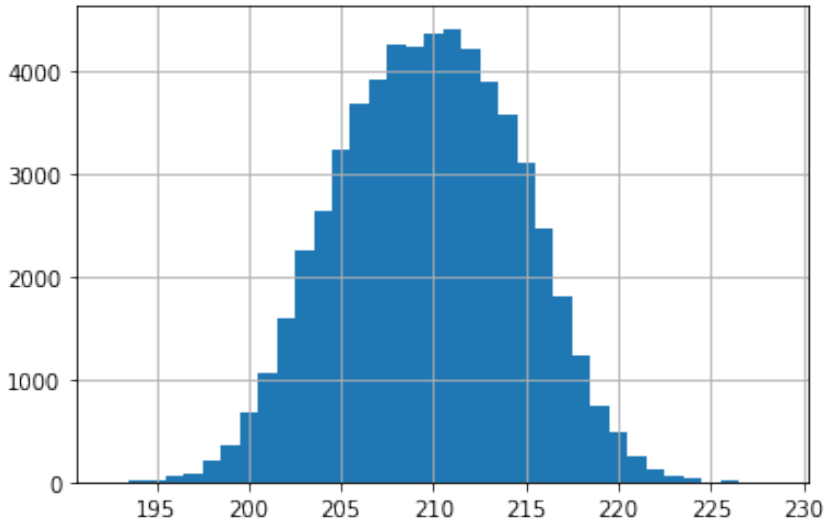

図 3: 実行結果

このヒストグラムから、以下の 3 点を読み取ることができる。

- 194〜226 [mm] までの範囲の値が得られた。
- 210 [mm] 付近の頻度が高い。
- 高頻度の部分からセンサ値が左右に離れるほど頻度が低くなる。

2.2　ノイズの数値化

前項より、センサデータの特徴が明らかになった。ここでは、センサデータのノイズの傾向を数値化する。具体的には、ノイズの平均値、分散、標準偏差を求める。`data["lidar"]` の平均値を、式 (1) に表す。

$$z_{\mathrm{LiDAR}} = \{z_i | i = 0, 1, 2, ..., N-1\} \tag{1}$$

ここで、z_{LiDAR} の平均値 μ は、式 (2) で与えられる。

$$\mu = \frac{1}{N} \sum_{i=0}^{N-1} z_i \tag{2}$$

これをソースコード 4 のように実装する。なお、`mean1` の計算は、平均値の定義に基づいており、`mean2` の計算は、`pandas` の平均を求める `mean` メソッドを利用している。ソースコード 4[1] に実装を示す。実行結果を図 4 に示す。

ソースコード 4: 平均値の計算

```
1 mean1 = sum(data["lidar"].values)/len(data["lidar"].values)
2 mean2 = data["lidar"].mean()
```

```
3 print(mean1,mean2)
```

```
209.73713297619855 209.73713297619855
```

図 4: 実行結果

ソースコード 5[1] のように実装することで、ヒストグラム上に平均値を出力する。実行結果を図 5 に示す。

ソースコード 5: 平均値の出力

```
1 data["lidar"].hist(bins = max(data["lidar"]) - min(data["lidar"]),color="orange",align
      ='left') ###avgplot###
2 plt.vlines(mean1,ymin=0,ymax=5000,color="red")
3 plt.show()
```

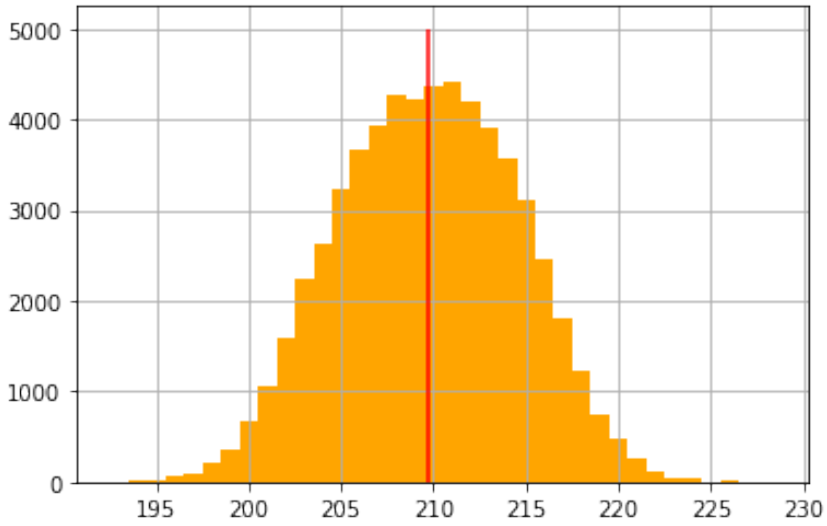

図 5: 実行結果

2.2.1　分散

次に分散を求める。標本分散は、式 (3) で定義される。

$$\sigma^2 = \frac{1}{N}\sum_{i=0}^{N-1}(z_i - \mu)^2 (N > 0) \tag{3}$$

各値と平均値の差の二乗値は、各値と平均値が離れているほど大きくなるため、分散は、リストの値が互いに大きく異なる程大きくなる。この問題を解決するために、不偏分散が式 (4) で定義される。

$$\sigma^2 = \frac{1}{N-1}\sum_{i=0}^{N-1}(z_i - \mu)^2 (N > 1) \tag{4}$$

これらの分散をソースコード 6[1] のように実装する。分散の計算モジュールとして、python には、pandas

と numpy が挙げられる。それぞれ、実装の考え方が異なり、pandas は不偏分散を、numpy は標本分散を元に実装されている。実行結果を図 6 に示す。

ソースコード 6: 分散の計算

```
# 定義から計算  ### calcvar
zs = data["lidar"].values
mean = sum(zs)/len(zs)
diff_square = [ (z - mean)**2 for z in zs]

sampling_var = sum(diff_square)/(len(zs)) # 標本分散
unbiased_var = sum(diff_square)/(len(zs)-1) # 不偏分散

print(sampling_var)
print(unbiased_var)

# を使用 Pandas
pandas_sampling_var = data["lidar"].var(ddof=False) # 標本分散
pandas_default_var = data["lidar"].var() # デフォルト（不偏分散）

print(pandas_sampling_var)
print(pandas_default_var)

# を使用 NumPy
import numpy as np

numpy_default_var = np.var(data["lidar"]) # デフォルト（標本分散）
numpy_unbiased_var = np.var(data["lidar"], ddof=1) # 不偏分散

print(numpy_default_var)
print(numpy_unbiased_var)
```

```
23.407709770274106
23.40810659855441
23.4077097702742
23.408106598554504
23.4077097702742
23.408106598554504
```

図 6: 実行結果

2.2.2 標準偏差

標準偏差を求める。標準偏差は、分散の平方根であるため、ソースコード 7[1] のようにして実装できる。pandas では、std メソッドを使ってデータから標準偏差を求めることができる。実行結果を図 7 に示す。

ソースコード 7: 標準偏差の計算

```
import math ### calcstddev
```

```
2
3 # 定義から計算
4 stddev1 = math.sqrt(sampling_var)
5 stddev2 = math.sqrt(unbiased_var)
6
7 # Pandas を使用
8 pandas_stddev = data["lidar"].std()
9
10 print(stddev1)
11 print(stddev2)
12 print(pandas_stddev)
```

```
4.838151482774605
4.83819249292072
4.838192492920729
```

図 7: 実行結果

センサデータの誤差の傾向は、標準偏差を用いて説明することができる。今回の場合、平均値が 209.7 [mm] で、 標準偏差が ±4.8 [mm] で誤差が生じることになる。

2.2.3　確率分布

続いて、センサデータのリスト $z_{\mathrm{LiDAR}} = \{z_i | i = 0, 1, 2, ..., N-1\}$ から、N 回目以降に取得されるセンサ値 $z_N, z_{N+1}...$ を予想する問題を考える。直観的には、ヒストグラムの頻度の大きい値が出やすいと予測できる。これを数値化したものを仮に確率として定義する。つまり、0 回目から $N-1$ 回目まで記録されたセンサデータにあるデータ z が m 個含まれていたら、その値の出る確率を $P(z) = m/N$ と考えることを指す。まず、各センサデータの頻度を集計する。ソースコード 8[1] に実装を示す。実行結果を図 8 に示す。

ソースコード 8: センサデータの頻度の集計

```
1 freqs = pd.DataFrame(data["lidar"].value_counts()) # 関数の DataFramevalue_counts()メ
      ソッドを用いて、出現頻度の大きい順に配列データに格納する。
2 freqs.transpose() # 横向きに出力
```

	211	210	208	209	212	207	213	206	214	205	...	197	196	223	224	226	195	194	193	227	229
lidar	4409	4355	4261	4228	4201	3920	3897	3674	3572	3225	...	84	59	55	32	15	13	10	4	3	1

1 rows × 35 columns

図 8: 実行結果

その後、確率の列を追加する。ソースコード 9[1] に実装を示す。

ソースコード 9: 確率の列の追加

```
freqs["probs"] = freqs["lidar"]/len(data["lidar"]) # 確率: "lidar" 列に入っているそれぞれの頻度をの要素数で割る。data
freqs.transpose()
```

	211	210	208	209	212	207	213	206	214	205	...	197	196	223	224	226	195
lidar	4409.000000	4355.000000	4261.000000	4228.000000	4201.000000	3920.000000	3897.000000	3674.000000	3572.000000	3225.000000	...	84.000000	59.000	55.000000	32.000000	15.000000	13.00000
probs	0.074744	0.073829	0.072235	0.071676	0.071218	0.066454	0.066064	0.062284	0.060555	0.054672	...	0.001424	0.001	0.000932	0.000542	0.000254	0.00022

2 rows × 35 columns

図 9: 実行結果

出力から、最頻出のセンサデータは、211 で、0.075 程度の確率で発生と分かる。さらに、出力結果をセンサデータで並び変えて、横軸にセンサ軸、縦軸に確率を描くことで確率質量関数を得る。各変数に対して、確率がどのように分布するかを表す P を確率分布と呼ぶ。確率質量関数の実装をソースコード 10[1] に示す。実行結果を図 10 に示す。

ソースコード 10: センサデータの並び替え

```
freqs["probs"].sort_index().plot.bar() # 確率を距離の小さい順に並び替え
plt.show()
```

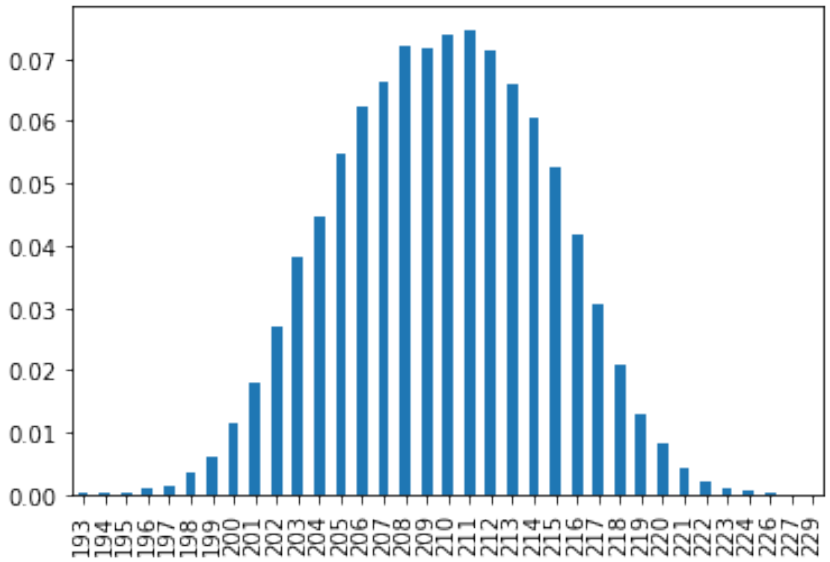

図 10: 実行結果

確率分布が求まったため、ソフトウェアでセンサデータの発生をシミュレーションすることができる。以下のように実装する。sample メソッドを使うことで、確率分布から値を選ぶことができる。ソースコード 11[1] に実装を示す。実行結果を図 11 に示す。

ソースコード 11: センサデータのシミュレーション

```
def drawing(): 関数として定義# ###one_sampling###
  return freqs.sample(n=1, weights="probs").index[0]
drawing() # 実行
```

図 11: 実行結果

2.2.4　正規分布

以前の結果より、センサデータのばらつきが正規分布に従うことが分かる。センサデータは、整数の値のみの離散的な値しかとらないが、連続的な値を取得できるものと仮定する。センサデータ z が、a 以上 b 未満に入る確率を式 (5) で定義する。正規分布を式 (6) に示す。

$$P(a \leq z < b) = \int_a^b p(z)dz \tag{5}$$

$$p(z) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(z-\mu)^2}{2\sigma^2}\right\} \tag{6}$$

σ^2 は分散、μ は平均値である。先ほど求めたセンサデータの平均値 $\mu = 209.7\,[\mathrm{mm}]$、分散 $\sigma^2 = 23.4$ を代入して式 (6) を描画する。ソースコード 12[1] に実装を示す。実行結果を図 12 に示す。

ソースコード 12: センサデータの正規分布の描画

```
1 def p(z, mu=209.7, dev=23.4): ###pdf_from_def###
2   return math.exp(-(z - mu)**2/(2*dev))/math.sqrt(2*math.pi*dev) #math 関数による正規
      分布の実装
3 zs = range(190,230) ###pdf_plot_from_def###
4 ys = [p(z) for z in zs]
5
6 plt.plot(zs,ys)
7 plt.show()
```

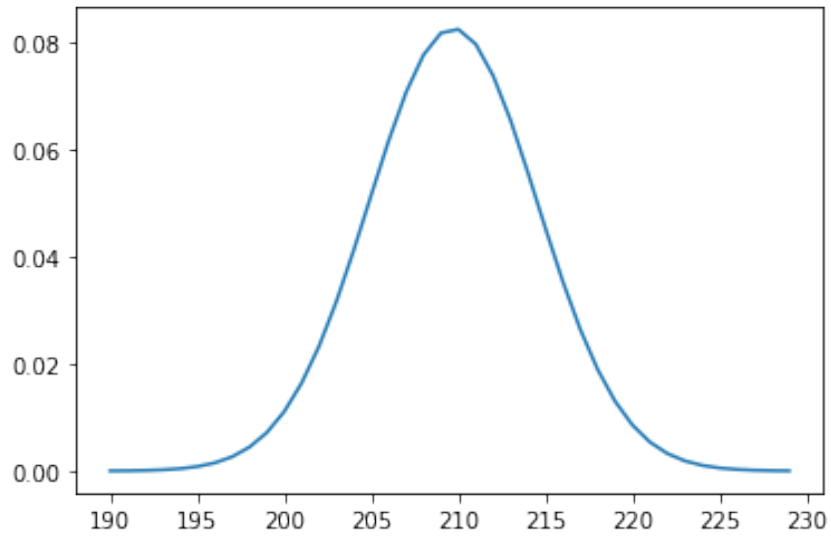

図 12: 実行結果

参考文献

[1] 上田隆一, 詳解確率ロボティクス python による基礎アルゴリズムの実装. 講談社, 2022.