I. 静止センサのノイズとバイアスに関する参考分析例

※）カレントフォルダ（このNotebookと同じフォルダ）にデータCSVファイルがあるものとする。

1. ノイズ

1-1. データフレームと要約統計量

■ pandas , scipy.stats , matplotlib.pyplot モジュールをそれぞれ， pd , stats , plt という名前でインポートする：

In [1]:
```
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
```

■ pandas モジュールの read_csv 関数を用いて，センサデータCSVファイル 'memdata_TSND121-06120699_2019-04-12_140934.329.csv' を読み込み，変数 df に格納する（p.244-245）：

In [2]:
```
df = pd.read_csv('memdata_TSND121-13121423_2019-04-16_144415.670.csv', header=None,
                 names=['item', 'time', 'acc_x', 'acc_y', 'acc_z', 'omega_x', 'omega_y', 'omega_z'])
```


センサデータCSVファイルには項目名がないため，オプション引数

```
header=None
```

と

```
names=['item', 'time', 'acc_x', 'acc_y', 'acc_z', 'omega_x', 'omega_y', 'omega_z']
```

を指定して，データフレーム df に項目名を設定している。

■ df の最上部と最下部を表示してみる(p.224)：

head(n) あるいは tail(n) の n を省略すると n は5に取られる：

In [3]:
```
df.head()
```

Out[3]:

	item	time	acc_x	acc_y	acc_z	omega_x	omega_y	omega_z
0	ags	53055784	-7	-29	10206	-70	-44	-6
1	ags	53055785	25	-26	10216	-70	-38	-6
2	ags	53055786	78	-72	10255	-83	-31	-24
3	ags	53055787	64	-77	10206	-89	-38	-24
4	ags	53055788	56	-79	10299	-70	-44	-12

In [4]:
```
df.tail()
```

Out[4]:

	item	time	acc_x	acc_y	acc_z	omega_x	omega_y	omega_z
303148	ags	53358932	105	32	10140	-76	-62	-36
303149	ags	53358933	44	-24	10145	-76	-44	-30
303150	ags	53358934	27	-47	10113	-70	-38	-6
303151	ags	53358935	30	-77	10157	-83	-31	-12
303152	ags	53358936	27	-21	10262	-89	-44	-24

■ データの最初と最後の部分は怪しい（撹乱が入る場合がある）ので，`df` の10000~35000行目までを残し，後はカットすることにした(p.226)：

In [5]:
```
df = df.iloc[10000:290001]
```

□カットしたデータフレームdfの最上部と最下部を表示してみる：

In [6]:
```
df.head()
```

Out[6]:

	item	time	acc_x	acc_y	acc_z	omega_x	omega_y	omega_z
10000	ags	53065784	66	-69	10250	-58	-44	-12
10001	ags	53065785	71	-62	10257	-58	-44	-6
10002	ags	53065786	47	-34	10196	-70	-38	-6
10003	ags	53065787	47	-55	10201	-76	-25	-12
10004	ags	53065788	15	-50	10218	-70	-31	-6

In [7]:
```
df.tail()
```

Out[7]:

	item	time	acc_x	acc_y	acc_z	omega_x	omega_y	omega_z
289996	ags	53345780	42	-38	10252	-64	-50	-18
289997	ags	53345781	37	-31	10277	-76	-62	-36
289998	ags	53345782	39	-38	10235	-76	-50	-24
289999	ags	53345783	30	-77	10177	-76	-38	-12
290000	ags	53345784	-12	-101	10167	-76	-31	-6

■ データフレームオブジェクトのメソッド `describe()` で，データの要約統計量を見てみる：

データフレームオブジェクト.describe()
`Return :` データフレームオブジェクト の要約統計量データフレームを返す
`Parameters` ：なし

In [8]:
```
df.describe()
```

Out[8]:

	time	acc_x	acc_y	acc_z	omega_x	omega_y	omega_z
count	2.800010e+05	280001.000000	280001.000000	280001.000000	280001.000000	280001.000000	280001.000000
mean	5.320578e+07	36.918793	-51.503087	10187.744408	-77.517487	-45.746362	-16.654248
std	8.082947e+04	32.906436	30.996179	43.780445	8.857339	9.304337	8.396877
min	5.306578e+07	-124.000000	-199.000000	9972.000000	-125.000000	-92.000000	-60.000000
25%	5.313578e+07	15.000000	-72.000000	10160.000000	-83.000000	-50.000000	-24.000000
50%	5.320578e+07	37.000000	-52.000000	10186.000000	-76.000000	-44.000000	-18.000000
75%	5.327578e+07	59.000000	-31.000000	10218.000000	-70.000000	-38.000000	-12.000000
max	5.334578e+07	188.000000	93.000000	10401.000000	-28.000000	17.000000	24.000000

ここで，

要約名称	意味
count	個数
mean	平均値
std	標準偏差
min	最小値
25%	第１四分位数
50%	第２四分位数(中央値)
75%	第３四分位数
max	最大値

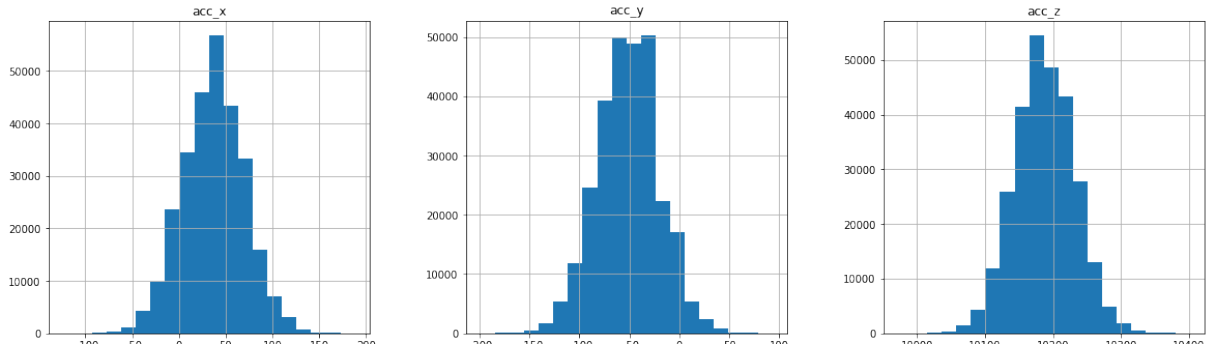
■ データフレームオブジェクトの `hist(...)` メソッドを用いて， `df` のすべての項目についてヒストグラムを描く。
□その前にデータフレームの全てのデータについてのヒストグラムを描く関数の公式を書いておこう：

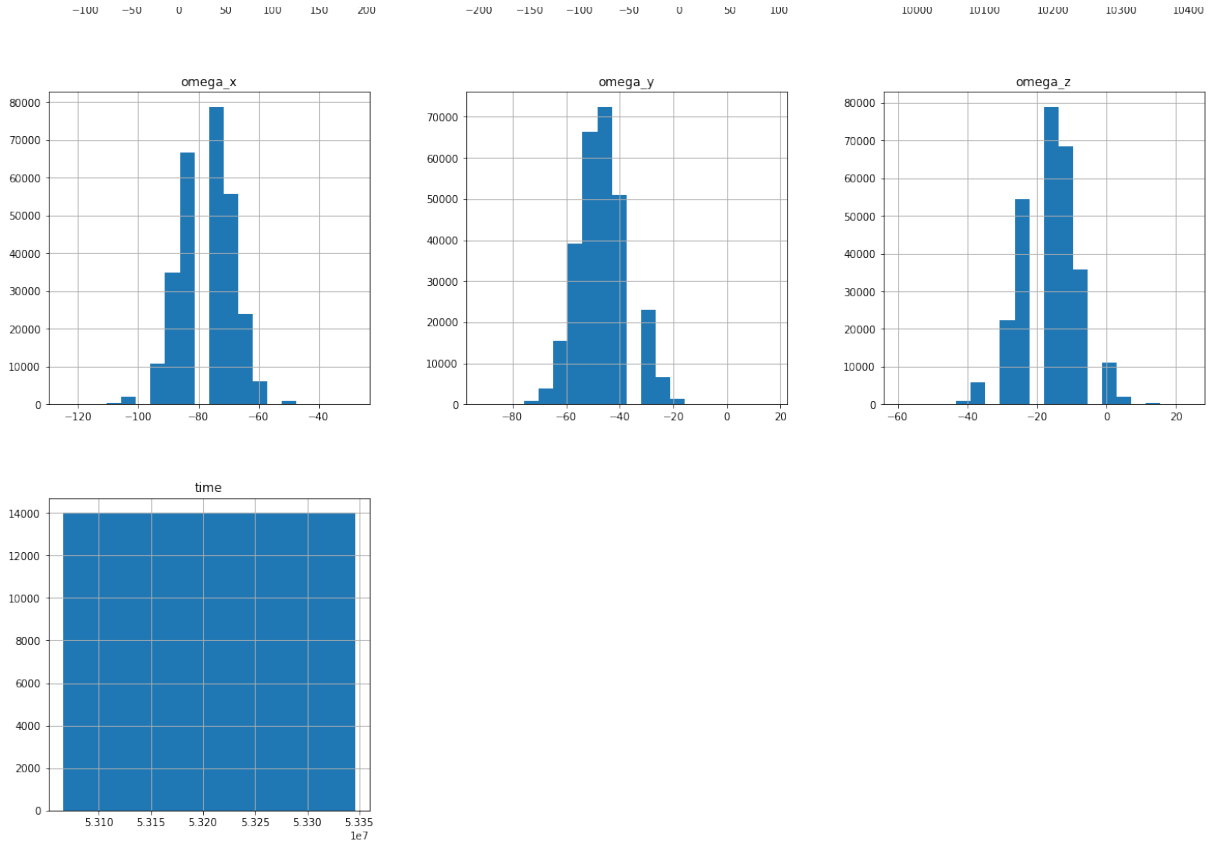
データフレームオブジェクト.hist(figsize=(数値1, 数値2), bins=数値3)
`Return :` データフレームオブジェクト の全ての項目についてのヒストグラムを返す

- `figsize=(数値1, 数値2)` ：グラフの横縦の大きさを 数値1:数値2 に設定する
- `bins=数値3` ：ヒストグラムの棒の区切り数を 数値3 にする

□上の公式を使って，dfの各種データについてのヒストグラムを描こう：

In [9]:
```
df.hist(figsize=(20, 20), bins=20)
plt.show()
```





1-2. 分析

■ ヒストグラム，正規Q-Qプロットおよび箱ひげ図を横に並べて描く(p.290)。
□その前に，必要なデータフレーム関するメソッドと， `matplotlib.pyplot` モジュールにあるグラフ描画のための関数の公式を書いておこう：

データフレームオブジェクト.columns.values
`Return :` データフレーム の行番号またはそれに当たる項目名のリストを返す

データフレームオブジェクト['項目名']
`Return :` データフレームの指定された **'項目名'** に対応する項目の**データシリーズ**を取り出す。

取り出したデータは `pandas` の `Series` （正確には `pandas.core.series.Series` ）というオブジェクトで

```
0      8
1      8
2      2
3      2
4      2
5      2
6     -4
7    -16
8    -16
9    -16
10     2
11   -10
12   -10
13   -16
```

という**データシリーズ**を返す。ここで上記右列の `0,1,2,...,13` は単なるレコード番号，左列の `8,8,2,...-16` はデータである。

データフレームオブジェクト.iloc[:, 列のindex]
Return : データフレームの指定された **列のindex (0,1,2,...)** に対応する項目の**データシリーズ**を取り出す。

データフレームオブジェクト.iloc[行のindex, :]
Return : データフレームの指定された **行のindex (0,1,2,...)** に対応する項目行の**データフレーム**を取り出す。

データフレームオブジェクト.iloc[:, 列のindex1:列のindex2+1]
Return : データフレームの指定された **列のindex1 〜 列のindex2** に対応する連続した複数の項目列の**データシリーズ**を取り出す。

データフレームオブジェクト.iloc[行のindex1:行のindex2+1, :]
Return : データフレームの指定された **行のindex1 〜 行のindex2** に対応する連続した複数の項目行の**データフレーム**を取り出す。

データフレームオブジェクト.iloc[行のindex1:行のindex2+1, : 列のindex1:列のindex2+1]
Return : データフレームの指定された **行のindex1 〜 行のindex2** , **列のindex1 〜 列のindex2** に対応する連続した複数の項目行の**データフレーム**を取り出す。

plt.hist(データシリーズ, bins=正の整数)
Return : データシリーズ のヒストグラムオブジェクト

- bins=正の整数 は階級数の指定（省略した場合は階級数10）

stats.probplot(データシリーズ, dist="norm", plot=plt)
Return : データシリーズ の正規Q-Qプロットオブジェクト

plt.boxplot(データシリーズ, labels=リスト)
Return : データシリーズ の箱ひげ図オブジェクト

- labels=リスト は複数の箱ひげ図に項目名を表示する場合に使用する（省略した場合は, 1,2,... と表示されるので, これが不必要な場合は labels=[''] とすればよい）

□まずはデータフレームdfから，項目リストを取得する：

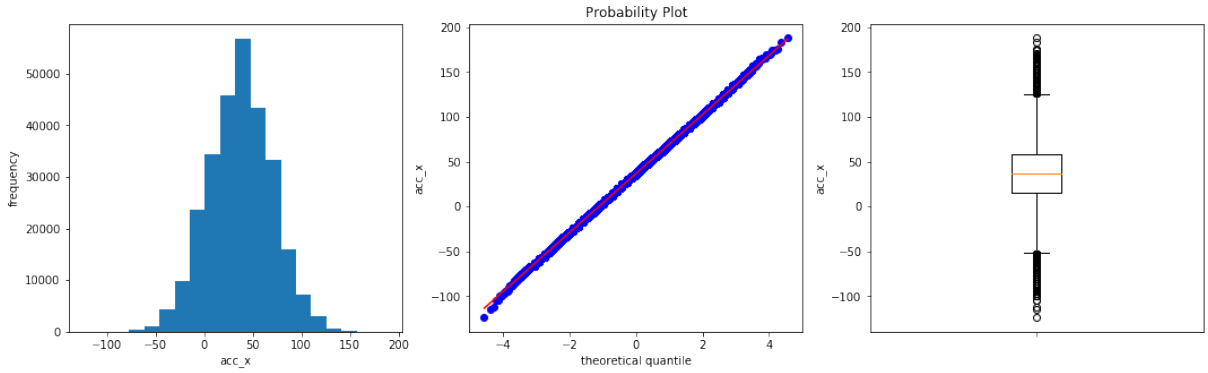
In [10]:
```
items = df.columns.values
print(items)
```

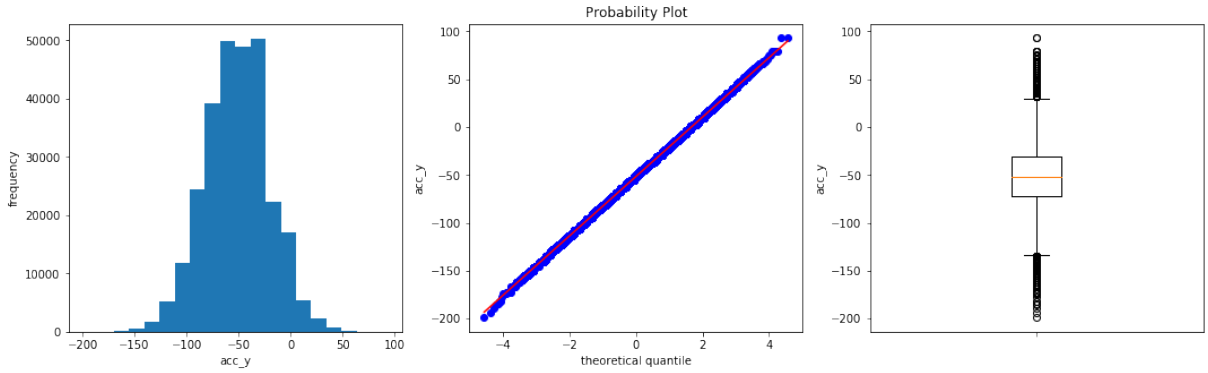
```
['item' 'time' 'acc_x' 'acc_y' 'acc_z' 'omega_x' 'omega_y' 'omega_z']
```

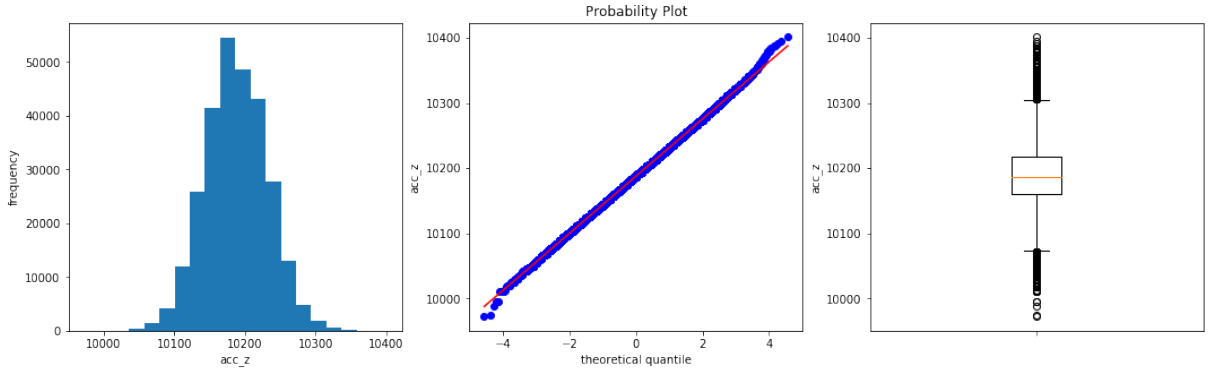
リスト変数 items に df の項目名が，確かに取得されている。

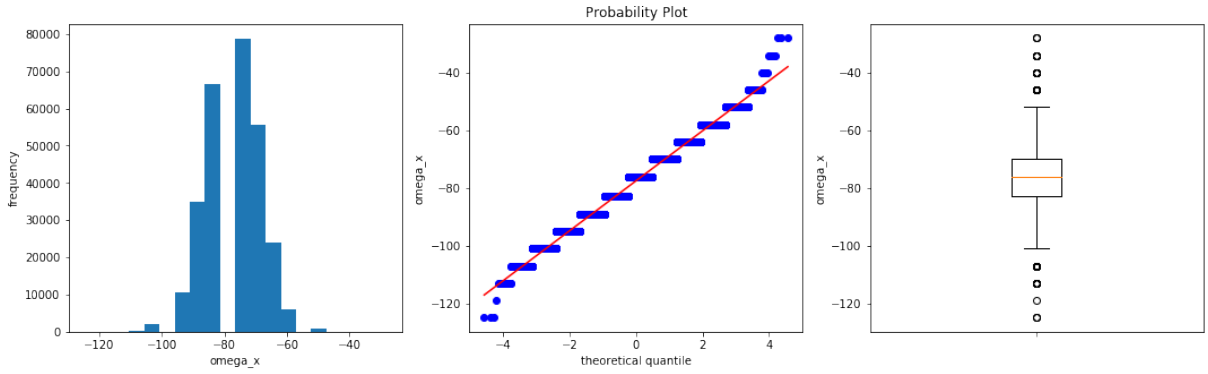
□ for 文を使って， 'acc_x', 'acc_y', 'acc_z', 'omega_x', 'omega_y', 'omega_z' のヒストグラム，Q-Qプロット，箱ひげ図を順に描画していく：

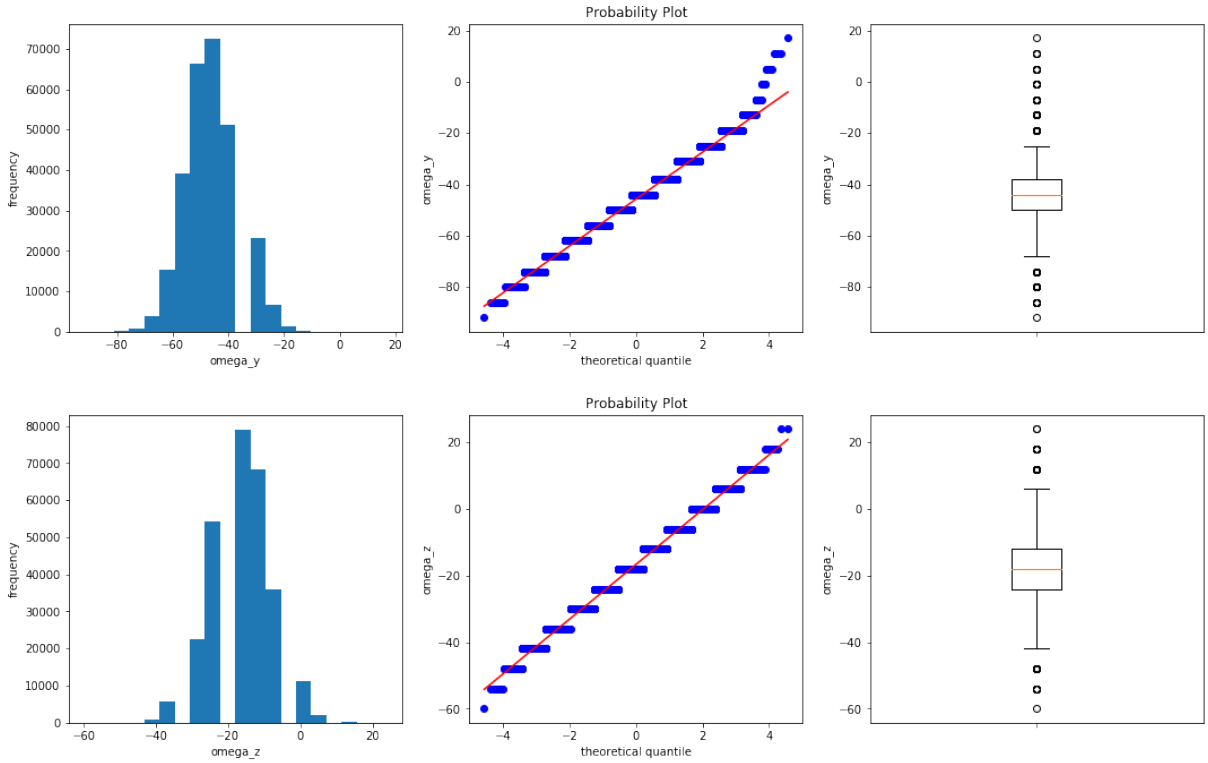
In [11]:
```
for i in range(2,8):            #iについて2,3,...,7を逐次代入して以下のブロックを繰り返す

    plt.figure(figsize=(18,5)) #1描画パレットをサイズを18:5に設定

    ###### ヒストグラム #####
    plt.subplot(1,3,1)                  #パレットを1行3列に分割し，最左列に以下のグラフを描画
    plt.hist(df.iloc[:, i], bins=20)    #dfのi列目の項目データについて100区切りのヒストグラムを描画
    plt.xlabel(f'{items[i]}')            #x軸ラベルをitemリストのi番目の要素に設定
    plt.ylabel('frequency')             #y軸ラベルを'frequency'に設定

    ##### Q-Qプロット #####
    plt.subplot(1,3,2)                                      #パレットを1行3列に分割し，中間列に以下のグラフを描画
    stats.probplot(df.iloc[:, i], dist="norm", plot=plt) #dfのi列目の項目データのQ-Qプロットを描画
    plt.xlabel('theoretical quantile')                      #x軸ラベルを'theoretical quantile'に設定
    plt.ylabel(f'{items[i]}')                                #y軸ラベルをitemリストのi番目の要素に設定

    ##### 箱ひげ図 #####
    plt.subplot(1,3,3)                          #パレットを1行3列に分割し，最右列に以下のグラフを描画
    plt.boxplot(df.iloc[:, i], labels=['']) #dfのi列目の項目データの箱ひげ図を描画(x軸メモリに何も表示しない)
    plt.ylabel(f'{items[i]}')                    #y軸ラベルをitemリストのi番目の要素に設定

    plt.show()  #各種グラフ描画関数の戻り値を表示しない
```


Probability Plot
frequency
acc_x
theoretical quantile
acc_x


Probability Plot
frequency
acc_y
theoretical quantile
acc_y


Probability Plot
frequency
acc_z
theoretical quantile
acc_z


Probability Plot
frequency
omega_x
theoretical quantile
omega_x



角速度についてのヒストグラムが**歯抜け**していることは気にするな。これは，階級数の取り方で異なってくる。重要なのは，正規Q-Qプロットで，全て**理論直線上**に乗る。これでデータの度数分布が**正規分布**していることが分かる。

■ 以下のメソッドでデータの度数分布の**尖度**と**歪度**を調べれば，データの正規分布性を数値的に確かめられる。

データシリーズ.kurt()
`Return :` データシリーズの度数分布の尖度を算出し，値が0に近ければ正規分布に近い(完全な正規分布は0)

- 尖度は度数分布の正規分布からの尖り具合の指標で，正なら完全正規分布から尖っており，負ならへしゃげている

データシリーズ.skew()
Return : データシリーズの度数分布の歪度を算出し，値が0に近ければ正規分布に近い(完全な正規分布は0)

- 歪度は度数分布の正規分布からの左右の偏り具合の指標で，正なら完全正規分布から右に偏り，負なら左に偏っている

□まず data_items というリスト変数に，データフレームの物理量の全ての項目名を順に並べておく：

In [12]:
```
data_items = items[2:]
print(data_items)
```

```
['acc_x' 'acc_y' 'acc_z' 'omega_x' 'omega_y' 'omega_z']
```

□ data_items のそれぞれの要素の項目について，尖度と歪度を求める：

In [13]:
```
for i in data_items:
    print(f'{i}:  kurt={df[i].kurt()}  skew={df[i].skew()}')
```

```
acc_x:  kurt=0.022680681379089407  skew=-0.008121965097061451
acc_y:  kurt=0.03729947152233848  skew=-0.015823897984537944
acc_z:  kurt=0.03153895542050922  skew=0.010085299970719984
omega_x:  kurt=-0.1599044840761379  skew=-0.0309501272548534
omega_y:  kurt=0.1350136958984276  skew=0.06501324388996166
omega_z:  kurt=0.010642898099706422  skew=-0.008275750648545572
```


全ての物理量の度数分布の尖度と歪度は0に非常に近い。したがって，それらの度数分布は全て正規分布しているといえるだろう。

1-3. 結論

■ 床(机上)に静止させたセンサの計測データより，物理データにはノイズが存在することが分かった。これらのノイズは正規分布する。ガウシアンノイズと呼ばれるこれらのノイズは，計測機器に普通に現れる機械的誤差の一部であり，実地の計測に影響を及ぼす可能性がある。特に微小な運動を計測する場合には，これらのノイズを考慮して慎重に実験計画を立ててそれを実行しなければならない。また，データ分析を行う際，これらのノイズを取り除く方法は，各種のフィルタリングという数学的方法をコンピュータに実装して行われるが完全なものではない。したがって，データを分析して結果を出す際には，前述のことを踏まえて，どの程度の誤差が生じているのかを見積もり，それを明らかにしておく必要がある。なお，ノイズが正規分布していない場合，つまりガウシアンノイズでない場合，計測機器の純粋なノイズ以外に別種のノイズが混ざっている可能性があるので，原因を調査し，その要因を極力取り除かなければならない。

1-4. 注意

■ ここで用いたデータは，坂本先生が新たに，静寂な研究室で慎重に計測したものであるので，（同じ機器でも）君達のデータとは異なった結果が出ていると思う。なぜか？　どうすればよいか？　考えて考察に述べよ。

ヒストグラムの両端のロングテール部分に現れる比較的少数の小さな検出値は，センサ固有のものではない。この部分(**外れ値**)を取り除く関数を用意したので，そのモジュールをDLしてして使ってみよ。外れ値は，　平均 ± 4×標準偏差 の範囲から外れるデータとしている。

```
import remove_outlieis as ro
ro.remove_outlieis(データフレームオブジェクト, 項目名)
```

で実行できる。詳しくは，

```
ro.remove_outlieis?
```

でマニュアルが表示されるので，それを読め。

2. バイアス

■ データフレームの統計的要約の平均(mean)と標準偏差(std)の部分をみてみよう：

In [14]:
```
df.describe().iloc[1:3]
```

Out[14]:

	time	acc_x	acc_y	acc_z	omega_x	omega_y	omega_z
mean	5.320578e+07	36.918793	-51.503087	10187.744408	-77.517487	-45.746362	-16.654248
std	8.082947e+04	32.906436	30.996179	43.780445	8.857339	9.304337	8.396877

※）データフレーム df に **describe** メソッドを作用させると， df の要約統計量が**表形式**で作成されるが，これは実は**データフレーム**オブジェクトであるので，それにもまた **iloc** プロパティが適用できることに注意。

■ センサを静止しさせているにも関わらず，加速度と角速度が計測されている。例えば加速度のx成分は `3.6846638[mG]` , 角速度のx成分は `-0.77461077[deg/sec]` である。ただし，加速度z成分は `1018.796412[mG]` で重力加速度の大きさとほぼ同等(ただし重力加速度ときっちり同じ値ではない)の加速度を検出している。

なお，標準偏差はノイズの(平均からの)ばらつき程度である。

■ センサを静止させている場合，横方向加速度(acc_x と acc_y)の平均は 0 に，鉛直方向加速度(acc_z)の平均が 10000 に，角速度(omega_x , omega_y , omega_z)の全ての成分の平均が 0 にならなければならない(このセンサの加速度の単位は10^{-1}mG，角速度の単位は10^{-2}deg/secであることに注意)が，そうはなっていない。これは，電子機器（今の場合は圧電素子）を用いて計測値（今の場合は加速度・角速度）を正しく（電気）信号伝達するために，予め一定の電圧をかけておかなければならないことに起因する。この予めかけられらた一定電圧のことを**バイアス**という。センサは内部で信号伝達後にバイアスを取り除いて再計算し計測値を補正しているのであるが，実験は出荷時とは異なる環境（気温や気圧他）で行うので，バイアスがどうしても残ってしまう場合が多い。したがって，計測結果に現れるバイアスを予め知っておいて，計測機器（今の場合はセンサ）を調整する（センサではオフセットを設定し直す）か，データ分析で補正するか（静止データで， acc_x , acc_y , omega_x , omega_y , omega_z では平均値を差し引き， acc_z は平均値の 10000 からのズレを差し引く）のどちらかを行う必要がある。

ただし，実験によってはバイアスが無視できる場合がある。

3. 重力加速度

■ 地球上での重力加速度の大きさは，$9.806[\mathrm{m/s^2}]$で，これを単位として$1[\mathrm{G}]$と表す場合がある（重力単位系）。本加速度センサは重力単位系を用いているが，出力する単位を$10^{-1}\mathrm{mG}$としているので，$9.806[\mathrm{m/s^2}] = 10000[10^{-1}\mathrm{mG}]$と出力する。§2で掲げた表で，`acc_z` の平均値が `10187.744408` を取っているのは，(バイアスを取り除くと)地球重力加速度を検出しているからである。さて，本実験でセンサz軸が鉛直上向きになるようにセンサを水平な机上に置いて計測したことを思い出そう。一方，地球に結び付けられた直交座標系$(\mathrm{O}; \mathbf{i}, \mathbf{j}, \mathbf{k})$の基本ベクトル$\mathbf{i}, \mathbf{j}$(それぞれx, y軸に沿う)を水平面上にとり，$\mathbf{k}$(z軸に沿う)を鉛直上向きにとったとき，物体の重力は鉛直下向きにかかるので，重力加速度ベクトルは$-9.806\mathbf{k}[\mathrm{m/s^2}] = -1\mathbf{k}[\mathrm{G}]$である。明らかにセンサで計測された重力加速度は(正の値をとっているので)，地球に結び付けられた座標系における重力加速度とは，その符号が反転している。これはなぜか？

ヒント:

1. よく知られたガリレオのピサの斜塔での実験で，異なる質量の物体の重力加速度は同じである*。

   > *空気抵抗は無視する。

2. 1の事実により，自由落下する閉ざされた箱の中にいる人は重力を感じない。実際，宇宙ステーションの中が無重力状態なのは，宇宙ステーションが自由落下しているからである**。

   > **しかし，宇宙ステーションは地球表面に対してほぼ水平に高速で移動しているので地球に衝突しない！。

3. 2の逆もいえる。つまり，地球やその他の星から遠く離れた無重力の宇宙空間にある箱を重力加速度と同じ大きさで一直線に加速させたとき，箱の中にいる人は，あたかも地球上にいるのと同じ重力を感じる。
4. センサは重力のことを知らない。

In [ ]: