◆matplotlibの練習◆

と思ったら、正規分布の勉強もしてた。

1　緒言

matplotlib は Python における基本的な可視化ライブラリである。seaborn というライブラリもあり[*1]、これは見た目が美しいことから、いずれは使ってみたいと考えているのだが、まずは基本中の基本である matplotlib を押さえることにした。

また、実行環境として Google Colaboratory を使用することにした。一応ローカルにも Anaconda を入れてあるので、Jupyter notebook を使えるのだが、今後みんなで Python を勉強していくにあたって、環境構築が不要な Google Colab を使用したほうが何かと都合がいいだろう。

2　1変数の基本的な関数

まずはデータではなく単純な関数をプロットしてみたい。変数が x の一つしかない関数を matplotlib を用いてプロットする。

とりあえず使うライブラリを import。

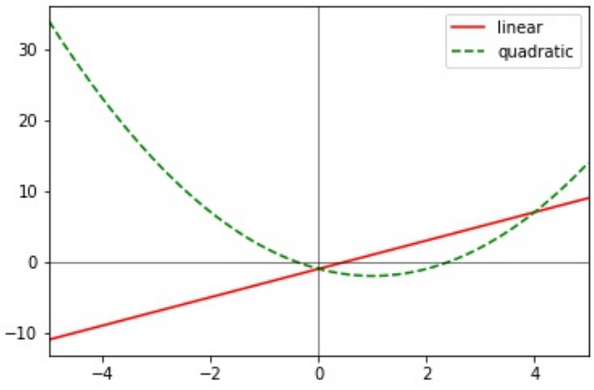
```
1 import numpy as np
2 from matplotlib import pyplot as plt
```

LaTeX だから無駄に数式を書いてみる。次の一次関数と二次関数を matplotlib にプロットさせよう。

$$y = 2x - 1 \tag{1}$$
$$y = x^2 - 2x - 1 \tag{2}$$

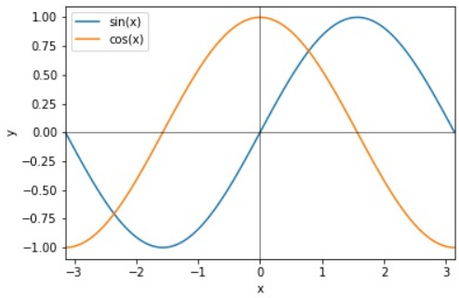
```
1 def linear(x):
2   return 2*x - 1
3
4 def quadratic(x):
5   return x**2 - 2*x -1
6
7 x = np.linspace(-5, 5, 100)  # x を numpy の linspace で list で与える。
8 y1, y2 = linear(x), quadratic(x)
9
10 plt.plot(x, y1, 'r', label='linear') # plot(x,y)を使う。'r'はマーカーで赤の実線。
       label は凡例に使う。
11 plt.plot(x, y2, 'g--', label='quadratic') # 'g--'は緑の破線
12 plt.xlim(-5, 5) # x の範囲を絞る（デフォルトでは x の範囲外まで描画される)。
13 plt.axhline(y=0, linewidth=0.5, color='k') # x 軸
14 plt.axvline(x=0, linewidth=0.5, color='k') # y 軸
15 plt.legend() # 凡例を追加
16 plt.savefig('linear_and_quadratic.jpg') # 画像を保存。
       eps 形式だとなんか TEX にうまく貼り付けられなかった。
```

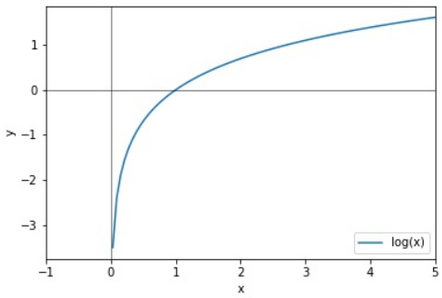
[*1] ただし、これは matplotlib のラッパーである。



いい感じだ。三角関数 $y = \sin x,\ y = \cos x$ や対数関数 $y = \log x$ もプロットしてみよう。

```
import math
pi = math.pi
x = np.linspace(-pi, pi, 100)

plt.plot(x, np.sin(x), label='sin(x)') # 三角関数は
    numpy のを使わないと怒られた。x が list なので。
plt.plot(x, np.cos(x), label='cos(x)')
plt.xlim(-pi, pi)
plt.xlabel('x') # ちゃんとラベルも追加した
plt.ylabel('y')
plt.axhline(y=0, linewidth=0.5, color='k')
plt.axvline(x=0, linewidth=0.5, color='k')
plt.legend()
plt.savefig('trigonometric.jpg')

x = np.linspace(-1, 5, 100)
plt.plot(x, np.log(x), label='log(x)')
plt.xlim(-1, 5)
plt.xlabel('x') # ちゃんとラベルも追加した
plt.ylabel('y')
plt.axhline(y=0, linewidth=0.5, color='k')
plt.axvline(x=0, linewidth=0.5, color='k')
plt.legend()
plt.savefig('logarithmic.jpg')
```





だいたい流れは分かった。x として、描きたい範囲の list を numpy の linspace(start, end, slice) で作り、matplotlib の plot に x と y の list を渡せばいいらしい。ただ、凡例やラベルや軸を書く書き方を忘れてしまいそうだ。

3　正規分布

2 変数の関数のプロットをしたいと思い、二変数の正規分布を描いてみたいと思ったのだが、その前にちょっと休憩。そもそも僕は、正規分布をよく知らない。

このページ[*2]を見て、正規分布の式を 1 次元から多次元に拡張するところを勉強しようと思う。

3.1　平均

確率変数 x の平均とは x の期待値である。確率密度関数を $p(x)$ として、x の平均 μ_x は、

$$\mu_x = \int_x xp(x)dx \tag{3}$$

で表される。うん、これはまあ言われてみれば分かる。

3.2　分散

確率変数 x の分散とは、平均 μ_x からの x の二乗誤差の期待値である。x の分散 σ_x^2 は、

$$\sigma_x^2 = \int_x (x-\mu_x)^2 p(x)dx \tag{4}$$

式 3 の x に $x-\mu_x$ を代入した感じ？

3.3　共分散

確率変数 x と確率変数 y の共分散は、同時確率密度関数[*3]を $p(x,y)$ として次の式で表される。

$$\sigma_{xy} = \int_x \int_y (x-\mu_x)(y-\mu_y)p(x,y)dydx \tag{5}$$

……分からない。なんとなく式 4 に似ているのは分かる。x, y は、例えば数学のテストの点数と国語のテストの点数、だとか、身長と体重だとか、まあそういうのを思い浮かべるといいのだと思う。

[*2] 一次元の正規分布から多次元正規分布へ - HELLO CYBERNETICS　https://www.hellocybernetics.tech/entry/2016/10/06/111153

[*3] 何ですかそれは。

この式のポイントは、x が大きくなると、y も大きくなるような関係がある場合、共分散は正の値、x が大きくなると y の値が小さくなるような関係がある場合、共分散は負の値を取ることである。いやあ、式を見るだけじゃよく分からんな……[*4]。

3.4 相関係数

確率変数 x と、確率変数 y の相関係数 ρ_{xy} は、次の式で表される。

$$\sigma_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y} \tag{6}$$

共分散を正規化したものである。共分散は元の値の大小に引きずられてしまうので分かりにくい。この相関係数は [-1,1] の範囲に収まるので分かりやすい。相関係数はよく見るやつである。

3.5 独立性

確率変数 x と確率変数 y が独立であるとは、

$$p(x, y) = p(x)p(y) \tag{7}$$

が成り立つことである。一方の試行が他方の試行に影響を与えないことで、サイコロをふることなどが例として挙げられる。これが成り立てば、相関係数は 0 になる。

3.6 一次元の正規分布

一次元の正規分布は次の式で表される。

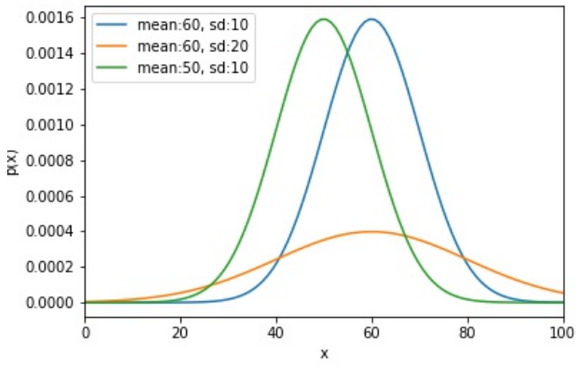
$$f(x) = \frac{1}{\sqrt{2\pi\sigma_x^2}} \exp\left(-\frac{(x-\mu_x)^2}{2\sigma_x^2}\right) \tag{8}$$

へえ。σ_x(分散のルート、つまり標準偏差) と μ_x^2（平均）が分かれば書けるのか[*5]。matplotlib で書いてみるか。

```
x = np.linspace(0, 100, 100)

def gaussian(x, mu, sigma):
  e = np.exp(-(x-mu)**2/ (2*sigma**2))
  return 1/(2*pi*sigma**2) * e # ルートを付けるの忘れてた

plt.plot(x, gaussian(x,60,10), label='mean:60, sd:10')
plt.plot(x, gaussian(x,60,20), label='mean:60, sd:20')
plt.plot(x, gaussian(x,50,10), label='mean:50, sd:10')
plt.xlim(0, 100)
plt.xlabel('x')
plt.ylabel('p(x)')
plt.legend()
```

[*4] 今回 x, y が確率変数だから余計に分からないのかも。

[*5] そらそうやな、という感じだが、改めて見ると「へえ」と思ってしまう。



標準偏差の値を 10 から 20 に変えるだけで、えらくグラフが平べったくなった[*6]。
ん？　これは確率密度関数ということは全区間で積分すると 1 になる？

```
# gaussian にちゃんとルートを付け直した。
from scipy import integrate
integrate.quad(lambda x:gaussian(x, 60, 10), 0, 100)

# (0.9999683277715794, 3.6566249334013534e-09)
```

だいたい 1 になった。
$\mu_x = 0, \sigma_x^2 = 1$ であるような正規分布を標準正規分布という。

$$f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) \tag{9}$$

3.7　特別な多次元標準正規分布

独立な確率変数 $x_i (i = 1, ..., N)$ を考える。これをベクトルとして表す[*7]。

$$\boldsymbol{x} = (x_i, ..., x_N)^{\mathrm{T}} \tag{10}$$

同時分布 $p(x_1, ..., x_N) = p(\boldsymbol{x})$ は、次のように個々の確率密度変数の積になる。

$$p(\boldsymbol{x}) = p(x_1)...p(x_N) \tag{11}$$

なので、多変数の標準正規分布は、

$$f(\boldsymbol{x}) = \frac{1}{(\sqrt{2\pi})^N} \exp\left(-\sum_{i=1}^{N} \frac{x_i^2}{2}\right) \tag{12}$$

となる。指数の積の肩は和になるんだよね。
ここで！！　ベクトルの内積を使うとシグマを使わなくて済むのだ。すごい。

$$f(\boldsymbol{x}) = \frac{1}{(\sqrt{2\pi})^N} \exp\left(-\frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}}{2}\right) \tag{13}$$

[*6] 上のグラフは正規分布の式の分母に$\sqrt{}$ をつけるのを忘れていたので、縦軸の値が間違っていますね……。
[*7] 複数の変数をベクトル化して扱うのは Cousera の機械学習でだいぶ慣れた。

3.8 matplotlib で二変数正規分布をプロット

このあと、一般の正規分布について議論していたが、まずこの標準正規分布で 2 変数の場合について、matplotlib で実装してみよう。

まず、2 変数の正規分布を返す関数と、x, y を表す ndarray[*8] を作った。

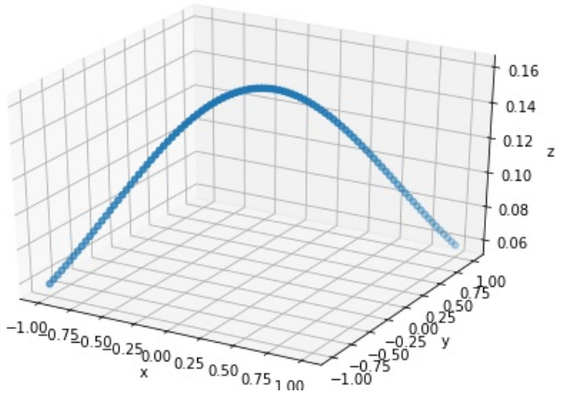
```
import math
import numpy as np
pi = math.pi

def gaussian_3d(x,y):
  return 1/np.sqrt(2 * pi)**2 * np.exp(-(x**2 + y**2) / 2)

x = np.linspace(-1,1,100)
y = np.linspace(-1,1,100)
```

3 次元プロットをするためには、Axes3D とかいうライブラリを使うらしい。散布図の描画には scatter というメソッドを呼ぶ。

とりあえず、x, y を gaussian_3d に渡して z を作り、scatter に突っ込んでみた。

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

z = gaussian_3d(x,y)

fig = plt.figure()
ax = Axes3D(fig)

ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")

ax.scatter(x, y, z)

plt.show()
```

[*8] 上で list とか言ってたやつも本当はこれ。



んんん……、線じゃなくて連続的な面として描いてほしいんだけどな。。。

いろいろググっているうちに、numpy の meshgrid というメソッドがあることが分かった。これは格子を作ってくれるメソッドらしい。もっぱら三次元空間に何かをプロットしたいときに使うメソッドのようだ。

```
p = np.linspace(0,5,6)
q = np.linspace(0,5,6)

P, Q = np.meshgrid(p, q)

'''
P
[[0. 1. 2. 3. 4. 5.]
 [0. 1. 2. 3. 4. 5.]
 [0. 1. 2. 3. 4. 5.]
 [0. 1. 2. 3. 4. 5.]
 [0. 1. 2. 3. 4. 5.]
 [0. 1. 2. 3. 4. 5.]]

Q
[[0. 0. 0. 0. 0. 0.]
 [1. 1. 1. 1. 1. 1.]
 [2. 2. 2. 2. 2. 2.]
 [3. 3. 3. 3. 3. 3.]
 [4. 4. 4. 4. 4. 4.]
 [5. 5. 5. 5. 5. 5.]]
'''
```

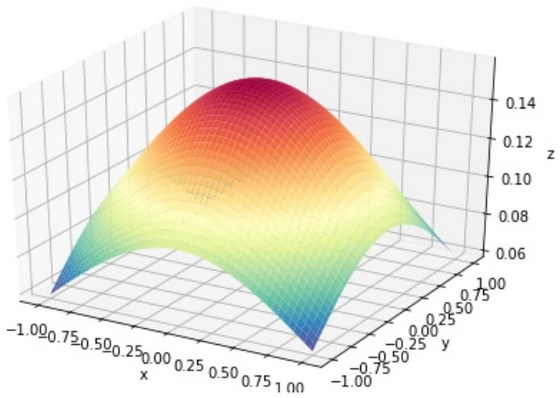
なんか分からんけど、P と Q が行列になった。こいつを gaussian_3d に渡すと、それぞれの要素同士の計算をしてくれるようである。なので当然戻り値も行列になる。要するに、P も Q も gaussian_3d(P,Q) も面になったのだ。

```
R = gaussian_3d(P,Q)
np.round(R, 3)  # 見やすいように桁数を制限

'''
array([[0.159, 0.097, 0.022, 0.002, 0. , 0. ],
```

```
        [0.097, 0.059, 0.013, 0.001, 0. , 0. ],
        [0.022, 0.013, 0.003, 0. , 0. , 0. ],
        [0.002, 0.001, 0. , 0. , 0. , 0. ],
        [0. , 0. , 0. , 0. , 0. , 0. ],
        [0. , 0. , 0. , 0. , 0. , 0. ]])
'''
```

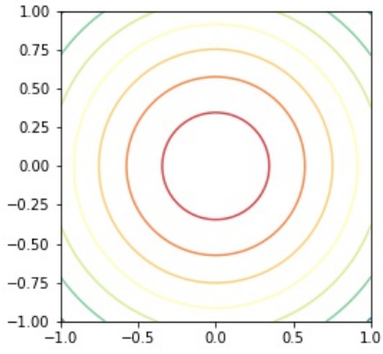
x, y を meshgrid にして、それを gaussian_3d に渡して、できた行列を Axes3D に渡してみた。plot_surface というメソッドがあったので、そっちを使ってみることにした。

```
X, Y = np.meshgrid(x,y)
Z = gaussian_3d(X,Y)
ax.plot_surface(X, Y, Z, cmap='Spectral_r')
```



やったー。が、正直言うと 3 次元プロットは見た目はかっこいいが、向こう側が見えないという致命的欠陥がある。そこでよく使われるのが等高線図である。contour[*9]というメソッドを使う。

```
plt.contour(X, Y, z,cmap='Spectral_r')
plt.gca().set_aspect('equal') # アス比を正方形に
```



[*9] 輪郭という意味の英語

3.9　一般の多次元正規分布

閑話休題。上の二変数標準正規分布を一般の多次元正規分布に拡張する方法を学ぶ。ベクトルや行列を使って表記するといいらしい。

$\boldsymbol{x}$ は N 個の確率変数を並べたベクトルである。よって確率変数 x_i の平均 μ_i を同様にして並べたものは、次のように定義できる。

$$\boldsymbol{\mu} = (\mu_1, ..., \mu_N)^{\mathrm{T}} \tag{14}$$

次に、分散を表現したいところだが、その前に x_i と x_j の共分散の式は次のように表される。ここで、E は期待値を表す。分散/共分散とは、要は平均からの偏差の期待値なのである。

$$\sigma_{x_i x_j} = E\left((x_i - \mu_{x_i})(x_j - \mu_{x_j})\right) \tag{15}$$

分散とは $i = j$ の場合の特別な共分散と考えられる。つまり、分散と共分散は同じ土俵にしてしまったほうが扱いが楽である。

N 個の確率変数があるとき、2 つの i, j の組み合わせを考えると、全部で N^2 個の組み合わせがあると考えられる。なので、分散は $N \times N$ の行列で表したほうが楽なのだ。

次のような分散を表す行列 Σ を考えてしまう[*10]。

$$\Sigma = \begin{pmatrix} \sigma_{x_1}^2 & \sigma_{x_1 x_2} & \cdots & \sigma_{x_1 x_N} \\ \sigma_{x_2 x_1} & \sigma_{x_2}^2 & \cdots & \sigma_{x_2 x_N} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{x_N x_1} & \sigma_{x_N x_2} & \cdots & \sigma_{x_N}^2 \end{pmatrix} \tag{16}$$

行列の対角成分が分散、そのほかは共分散になっている感じ。これを分散共分散行列と呼ぶ。

Σ の i, j 成分は式 15 で表されるので、Σ をベクトルを使って書き直すと、

$$\Sigma = E\left((\boldsymbol{x} - \boldsymbol{\mu}_x)(\boldsymbol{x} - \boldsymbol{\mu}_x)^{\mathrm{T}}\right) \tag{17}$$

になる。これ、縦ベクトル×横ベクトル=行列をうまく利用しているようなのだが、うーん、E が入っているからいまいちパッと分からない。式 15 を Σ に代入しちゃえ。

$$\boldsymbol{x} - \boldsymbol{\mu}_x = (x_1 - \mu_{x_1},\ x_2 - \mu_{x_2},\ ...,\ x_N - \mu_{x_N})^{\mathrm{T}} \tag{18}$$

$$\Sigma = E\begin{pmatrix} (x_1 - \mu_{x_1})(x_1 - \mu_{x_1}) & (x_1 - \mu_{x_1})(x_2 - \mu_{x_2}) & \cdots & (x_1 - \mu_{x_1})(x_N - \mu_{x_N}) \\ (x_2 - \mu_{x_2})(x_1 - \mu_{x_1}) & (x_2 - \mu_{x_2})(x_2 - \mu_{x_2}) & \cdots & (x_2 - \mu_{x_2})(x_N - \mu_{x_N}) \\ \vdots & \vdots & \ddots & \vdots \\ (x_N - \mu_{x_N})(x_1 - \mu_{x_1}) & (x_N - \mu_{x_N})(x_2 - \mu_{x_2}) & \cdots & (x_N - \mu_{x_N})(x_N - \mu_{x_N}) \end{pmatrix} \tag{19}$$

確かに式 17 書くことができそうだ。

これらを使って、一般の多次元正規分布は次のように書けるそうだ。

$$f(\boldsymbol{x}) = \frac{1}{(\sqrt{2\pi})^m \sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^{\mathrm{T}} \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu})\right) \tag{20}$$

ただし、$|\Sigma|$ は Σ の行列式。

[*10] 行列は TeX で書くのがとてもめんどくさい。

この式の導出については、本当にただひたすら式変形の話になってしまうので、天下り的に受け入れることにしよう。

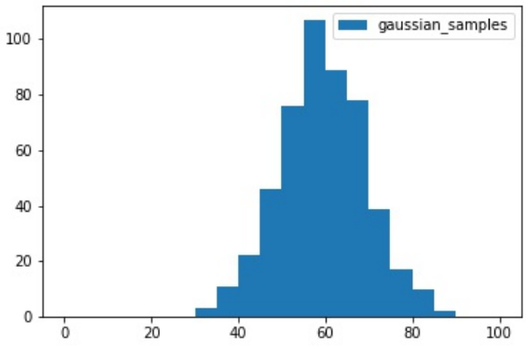
3.10　正規分布を返すメソッド

と、ここまで人力で実装してきてあれなのだが、scipy には正規分布を返すメソッドが用意されている。norm は正規分布。pdf は確率密度関数（Probability Density Function）。出力されるグラフは当然同じなので省略する。

```
from scipy import stats

x = np.linspace(0, 100, 1000)
p = scipy.stats.norm.pdf(x, 60, 5)
plt.plot(x, p, label='mu=60, sd=5')
plt.legend()
```

さらになんと次のように、正規分布に従う乱数の ndarray を返してくれるメソッドもある。ヒストグラムにしてみた。

```
g = np.random.normal(60, 10, 500)
plt.hist(g, bins=20, range=(0,100))
```



np.random.normal の引数は、（平均, 標準偏差, サンプル数）の意味。

4　結言

特になにか難しいことをしているわけではなかったが、真面目に matplotlib を触ったのがこれが初めてな気がする。結構呪文的なところが多く、ちゃんと覚えられないので、当分はコピペでやることになりそうだ。また、matplotlib や numpy にどんなメソッドがあるのかというのはきちんと覚えたい。しばらくはこれらの可視化系ライブラリの練習になりそうだ。また、今回は乱数や連続関数しかプロットしなかったが、今度は意味のあるデータ[*11]をプロットしたり、pandas を用いて集計したりしてみたい。

また、正規分布の定義式を初めてじっくり眺めてみた。そこから多次元に拡張するのは、正直難しくてあまり

[*11] 今考えているのは気象庁のアメダス観測データ。

よく分からなかったが、多変数を扱うのにベクトルや行列で記述すると効率的であるということが分かったので、今後線形代数やこれらを使った量の表記についても慣れていく必要があるなと思った。