機械学習入門

第1章

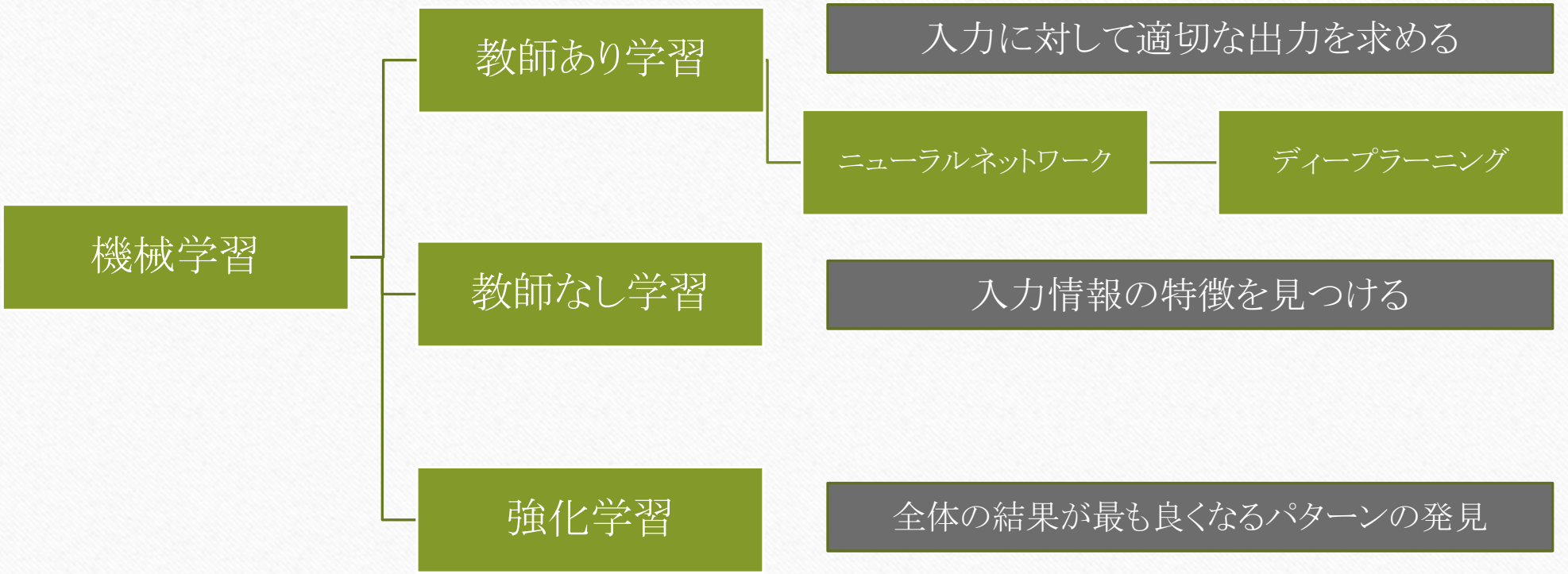
数学の基本と可視化の方法を身につけるの巻

機械学習とは

- データから法則性を抽出する統計的手法の一つ

→ 現在は**様々なモデル(アルゴリズム)**が提案されている

機械学習の分野体系


機械学習
教師あり学習
入力に対して適切な出力を求める
ニューラルネットワーク
ディープラーニング
教師なし学習
入力情報の特徴を見つける
強化学習
全体の結果が最も良くなるパターンの発見

使用環境

プログラミング言語　　：Python 3.6.4

Python ライブラリ　　：NumPy（数学、計算）

matplotlib（プロット、可視化）

Pythonの基本

- 四則演算：+ - * /
- 累乗：**
- 型の調査：type(<var, num, str…>)
- 出力：print(x), print(“something {0} something {1:f2}…”.format(x, y))
- 多次元配列：[[1, 2, 3], [4, 5, 6]]
- 配列の長さ：len(x)
- N個の連続データ：range(n+1), range(init, n+1)※要素の書き換え不可
- N個の配列：list(range(n)))※要素の書き換え可能
- タプル：x = (1, 2, 3))※要素の書き換え不可
- 条件分岐：if <condition>: … else: …
- 繰り返し：for i in range(len(x)):
- ライブラリの導入：import numpy as np
- ヘルプ：help(***functionname***)
- スライシング：x[0:9:1]　　要素番号0から9までを1つおきに抽出

NumPyの基本

np = numpy
x = np.array([[1,2,3], [4,5,6]])

ndarray型: the N dimensional array

- ベクトルの定義（1次元配列）：np**.array(**[1, 2, 3]**)**
- 行列の定義（2次元配列）：np.array(**[**[1, 2, 3], [4, 5, 6]**]**)
- 数列の生成：np.arange(0, 9, 1, dtype = None)
- 全要素が0の配列：np.zeros(m, n)
- 全要素が1の配列：np.ones(n)
- 全要素が不定の配列：np.empty()
- 要素の参照：x[2, 3]
- Rank（行列の階数）：x**.ndim**
- 行数・列数：x**.shape**
- 配列内の全要素数： x**.size**
- データの大きさ(単位：bytes): x**.itemsize**
- データ型：x**.dtype**
- 代入：y = x.**copy()**
- 行列の形の変更：x.reshape(m, n)

0から9までの公差1の等差数列⇒連番

m行n列の行列

ndim: the number of dimensions

dtype: Data Type Object

NumPyの基本2

np = numpy
x = np.array([[1,2,3], [4,5,6]])

ndarray型: the N dimensional array

- 指数：np.exp(x), np.exp2(x), np.exp10(x)
- 平方根：np.sqrt(x)
- 対数：np.log(x)
- 四捨五入：np.round(x, ***有効桁数***)
- 平均：np.mean(x)
- 標準偏差：np.std(x)

それぞれの要素の値が変化する

- 最大値：np.max(x)
- 最小値：np.min(x)

すべての要素から1つの値を求める

- 四則演算：**+ - * /**
- スカラー倍：x * 10
- 行列積：x1.dot(x2)

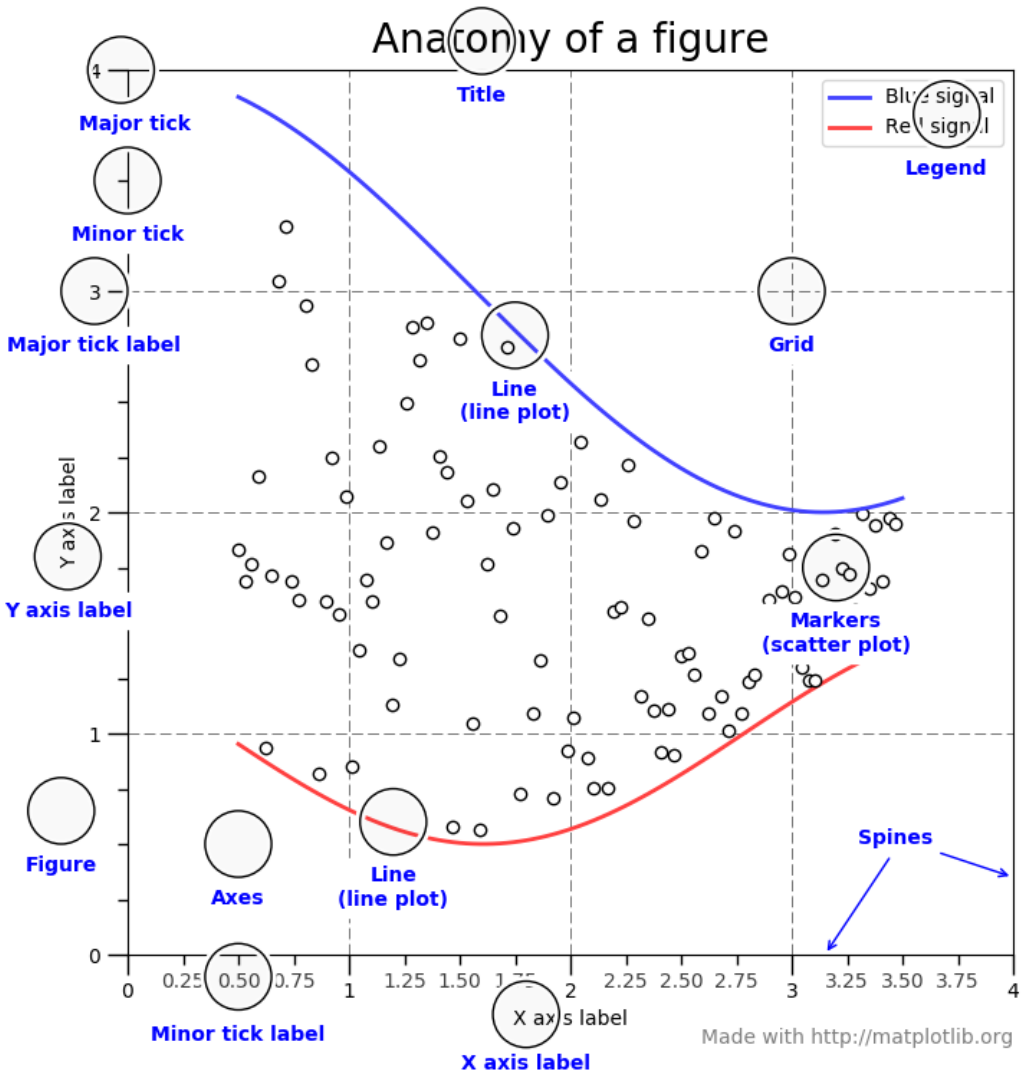
x1 * x2

- 保存：np.save('filename.npy', x)
- 読み込み：np.load ('filename.npy')
- 格納された変数のリスト：outfile.files

読み込んだ変数はoutfileに格納される

matplotlib

線グラフ：plt.plot(x, y)
表示：plt.show()

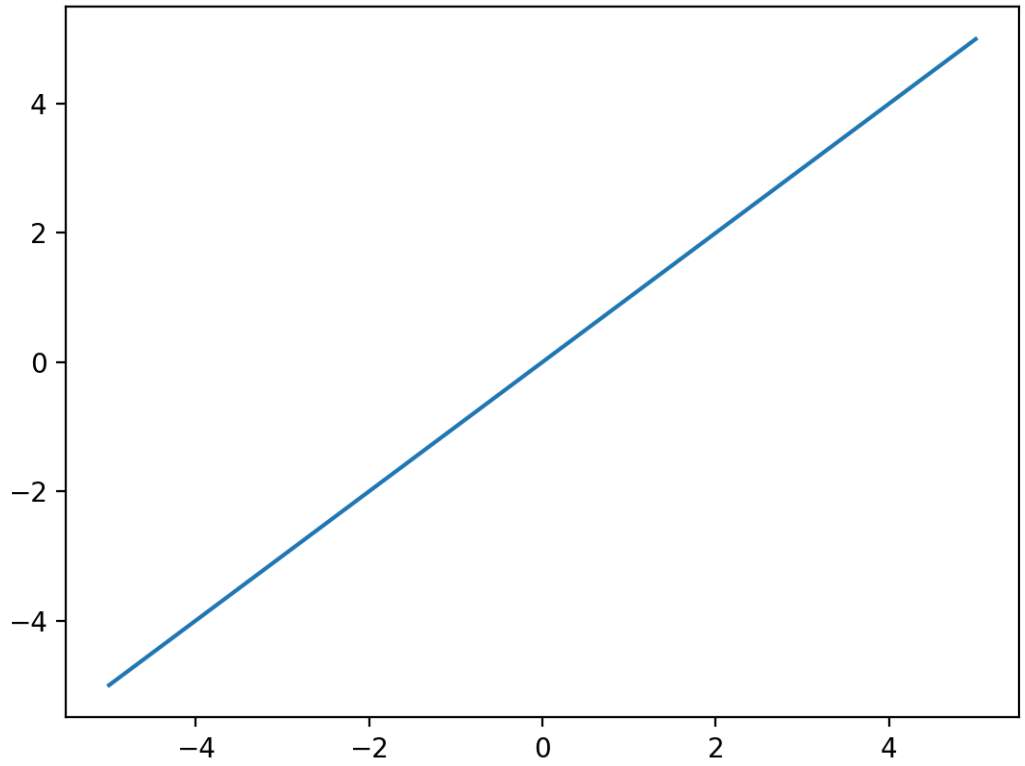


とにかくやってみよう！

$$y = ax$$

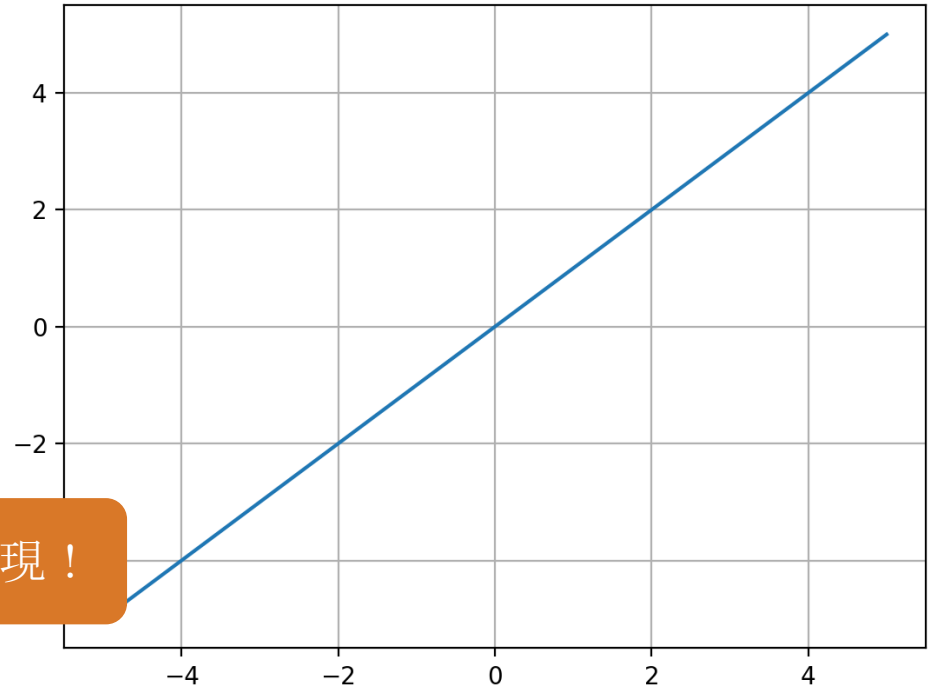
直線の方程式

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-5, 5, 10) /
#-5から5まで10に等分した数値を配列で格納

a = 1 #傾き
b = 0 #切片

y = a * x + b

plt.plot(x,y) #線グラフを生成・プロット

plt.show() #グラフ全体を生成・表示
```



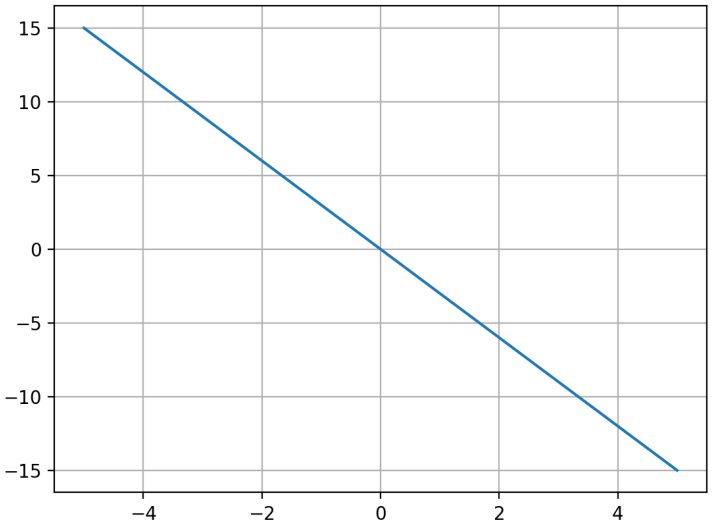
少し判りづらいので・・・

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-5, 5, 10) /
#-5から5まで10等分した数値を配列で格納

a = 1 #傾き
b = 0 #切片

y = a * x + b

plt.grid(True)

plt.plot(x,y) #線グラフを生成・プロット

plt.show() #グラフ全体を生成・表示
```

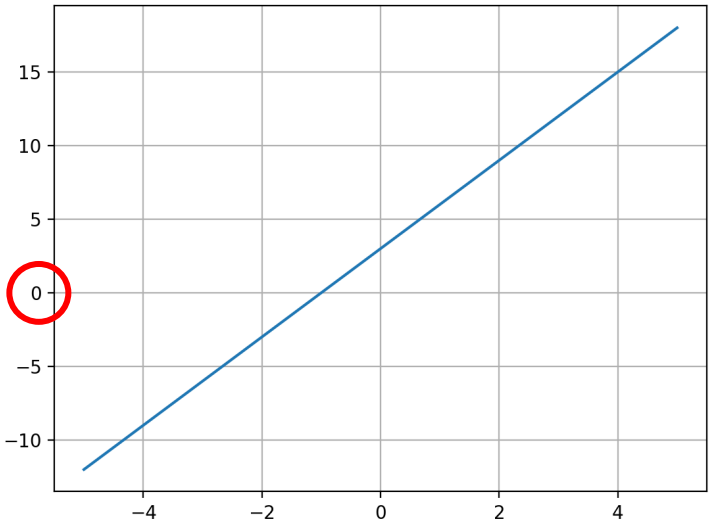


グリッド線が出現！

a = −3



a = 3, b = 5



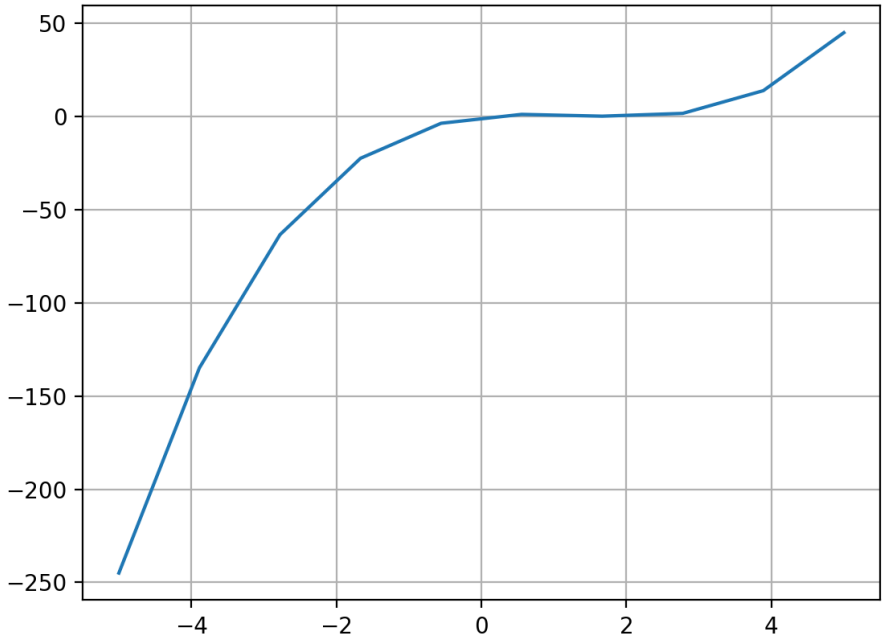
$$
\begin{aligned}
f(x) &= (x-2)x(x+2) \\
&= (x^2-4)x \\
&= x^3+4x
\end{aligned}
$$

matplotlibで可視化しよう

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-5, 5, 10)

def f(x):
  return (x-2) * x * (x-2)

plt.plot(x, f(x))

plt.grid(True)

plt.show()
```

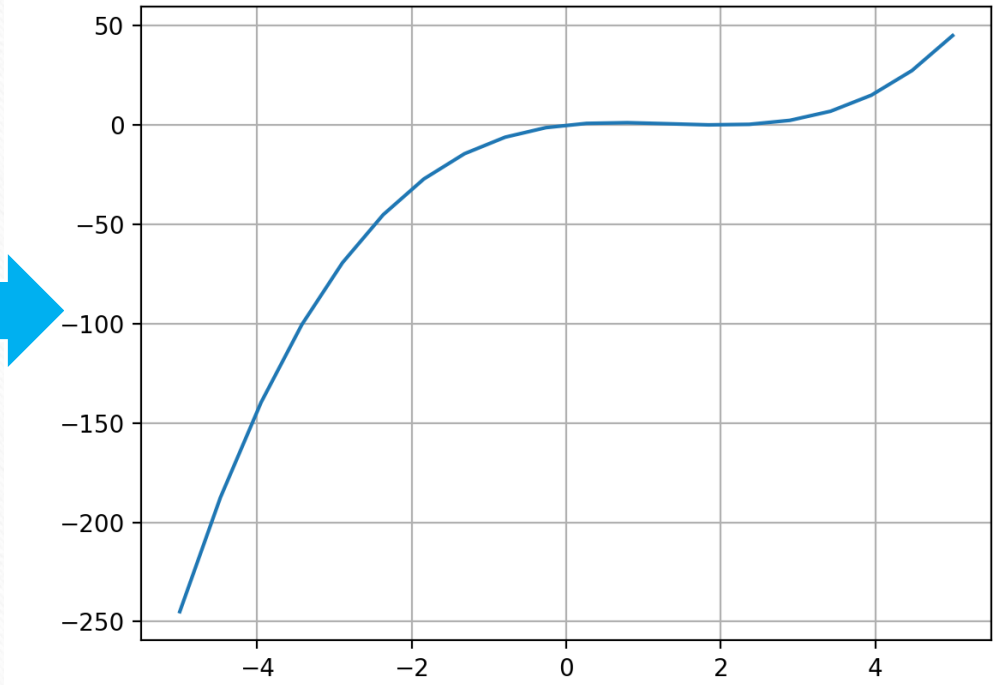
関数式を関数として定義

ここがスッキリ



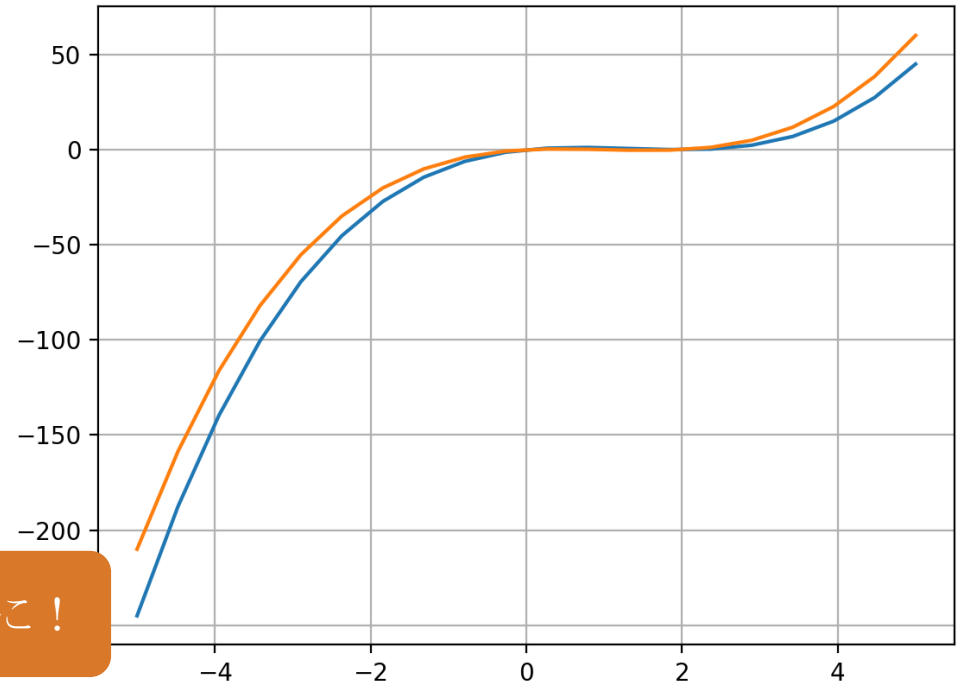
matplotlibで可視化しよう

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-5, 5, 20)

def f(x):
  return (x-2) * x * (x-2)

plt.plot(x, f(x))

plt.grid(True)

plt.show()
```



グラフがより滑らかなものへと変化した

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-5, 5, 20)

def f_2(x,w):
return (x-w) * x * (x-2)

plt.plot(x, f_2(x, 2)) # w = 2
plt.plot(x, f_2(x, 1)) # w = 1

plt.grid(True)

plt.show()
```



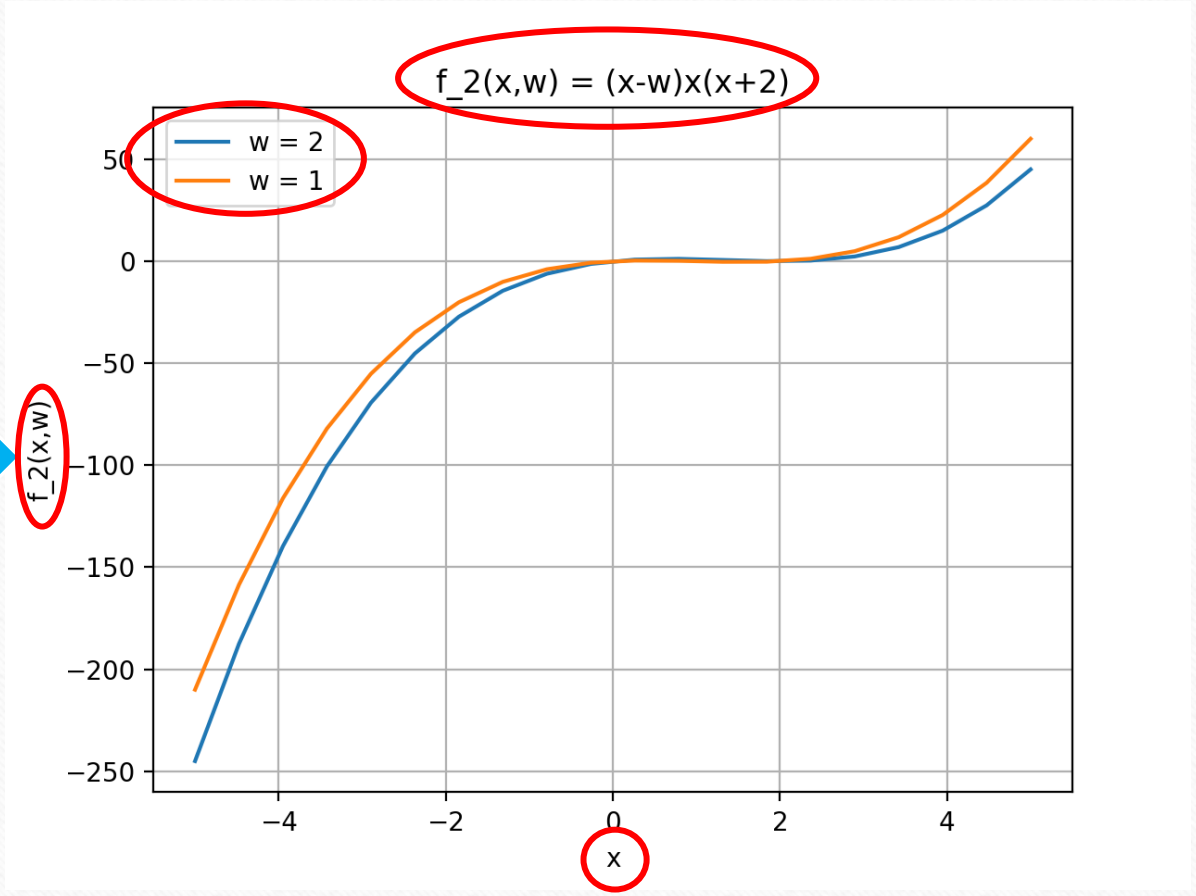
グラフ線が2つに！

$f(x) = (x - w)x(x + 2)$ へ関数式を変更

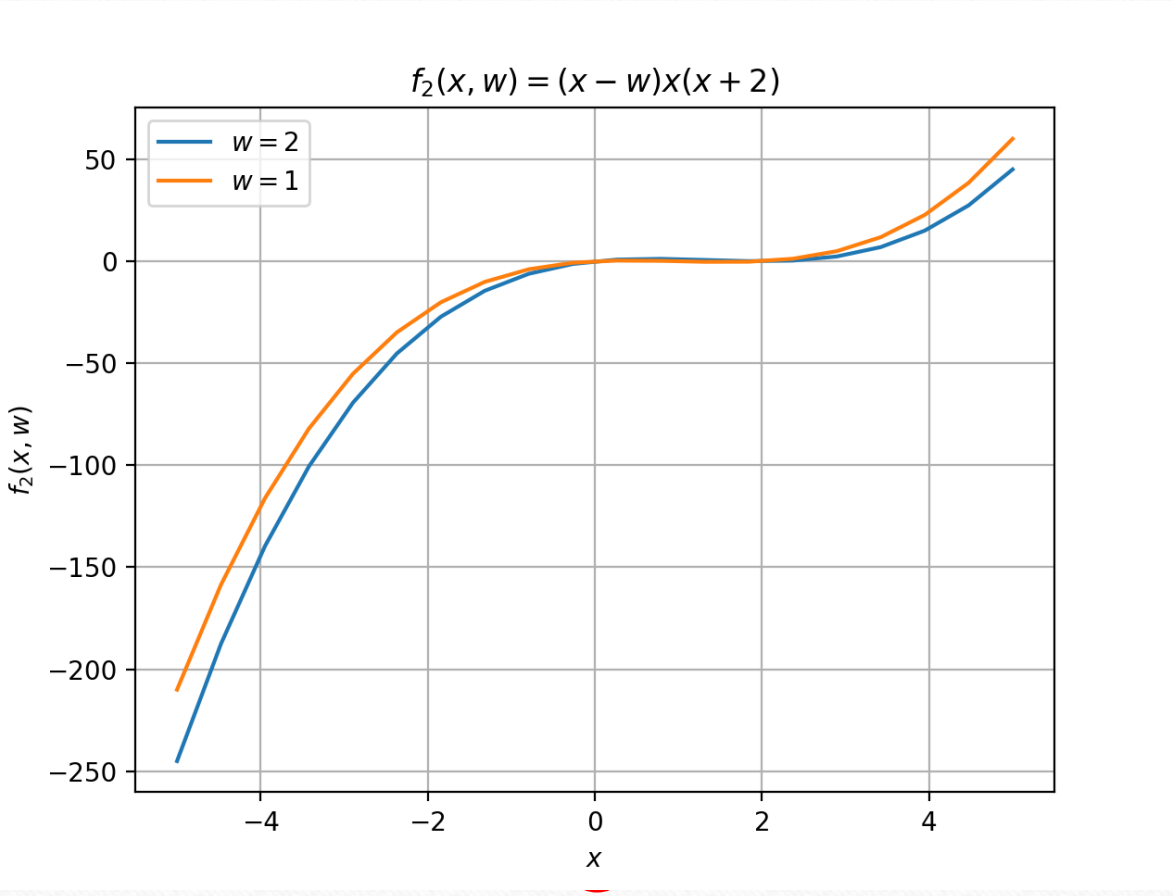
- 線グラフ2つ出てきたけどどっちが何なのかよくわからない
- そもそもどの関数式を描いたものだっけ？
- 縦軸と横軸が示すものは？

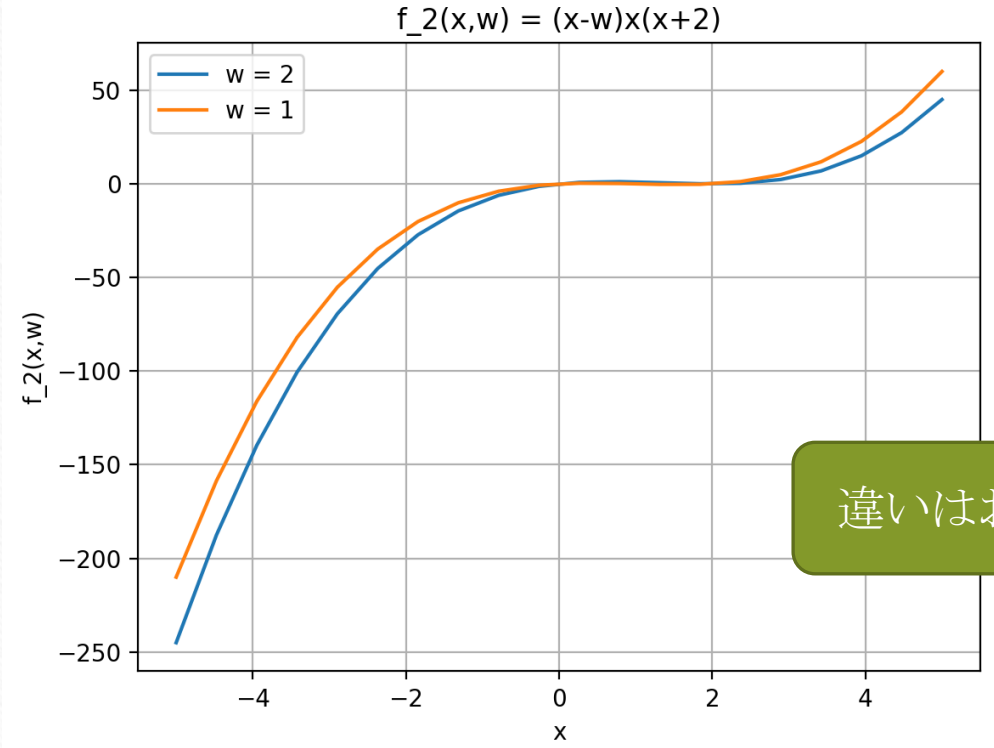
全て解決できます！

```
plt.plot(x, f_2(x, 2), label="w = 2") # w = 2
plt.plot(x, f_2(x, 1), label="w = 1") # w = 1

plt.grid(True)

plt.legend()

plt.title("f_2(x,w) = (x-w)x(x+2)")
plt.xlabel("x")
plt.ylabel("f_2(x,w)")

plt.show()
```

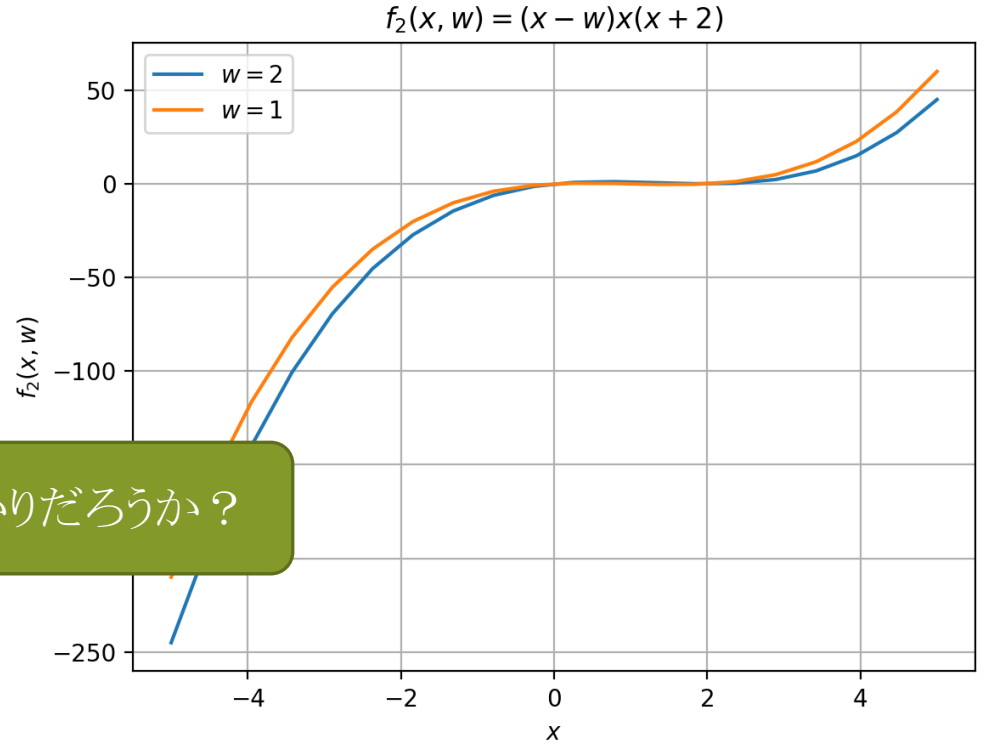


```
plt.plot(x, f_2(x, 2), label="$w = 2$") # w = 2
plt.plot(x, f_2(x, 1), label="$w = 1$") # w = 1

plt.grid(True)

plt.legend()

plt.title("$f_2(x,w) = (x-w)x(x+2)$")
plt.xlabel("$x$")
plt.ylabel("$f_2(x,w)$")

plt.show()
```





違いはお分かりだろうか？



通常

TeX形式

TeXとは？

記述はマークアップ言語形式

論文なんかでよく使われる

MS Wordなんかで書くよりずっときれいでスタイリッシュな数式やフォントに仕上がる

MS Office

$$f(x) = (x + 2)x(x - 2)$$

LATEX

$$f(x) = (x - 2)x(x + 2)$$

TeXはすばらしい！！

卒研の論文にwordなんか使わないよね？笑

閑話休題。

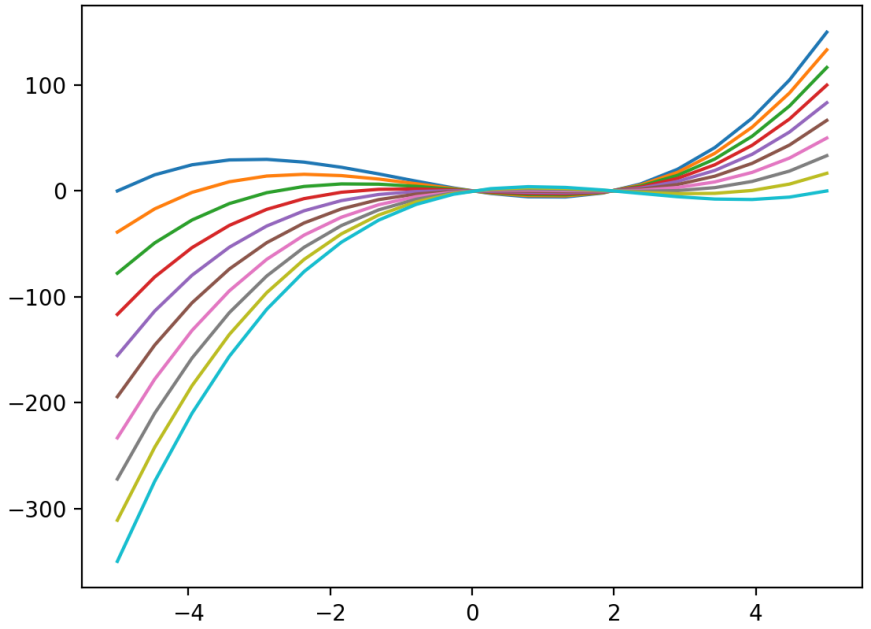
今度は

$$f(x) = (x - w)x(x + 2)$$ の w を動的な値にすると・・・？

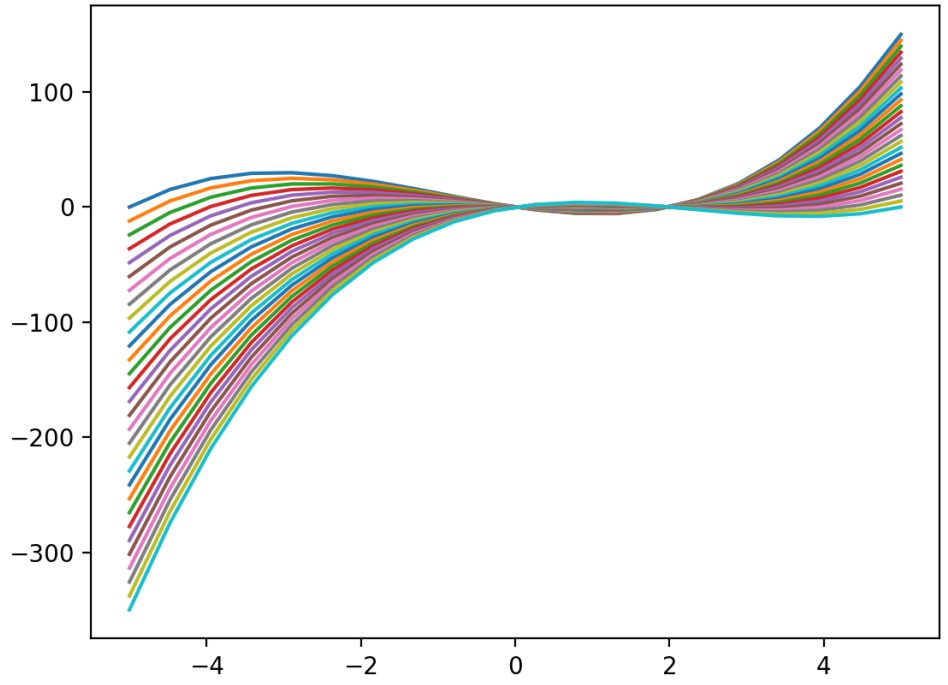
➡ つまり**2変数関数**の場合を考える

```python
import numpy as np
import matplotlib.pyplot as plt

xn = 20
wn = 10

x = np.linspace(-5, 5, xn)
w = np.linspace(-5, 5, wn)

def f_2(x, w):
    return (x-w) * x * (x-2)


y = np.zeros((xn, wn))

for i0 in range(xn):
    for i1 in range(wn):
        y[i0, i1] = f_2(x[i0], w[i1])

plt.plot(x, y)

plt.show()
```

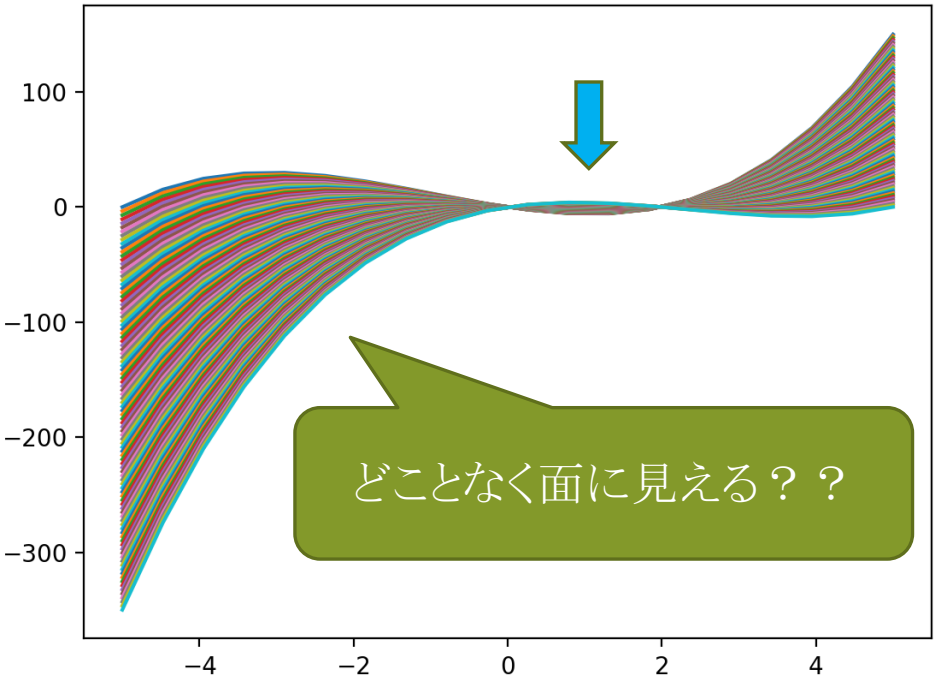
range()はリストを返す



wn = 30



wn = 100



当然、2変数関数は**平面**において表現の範囲に限界が出て来る

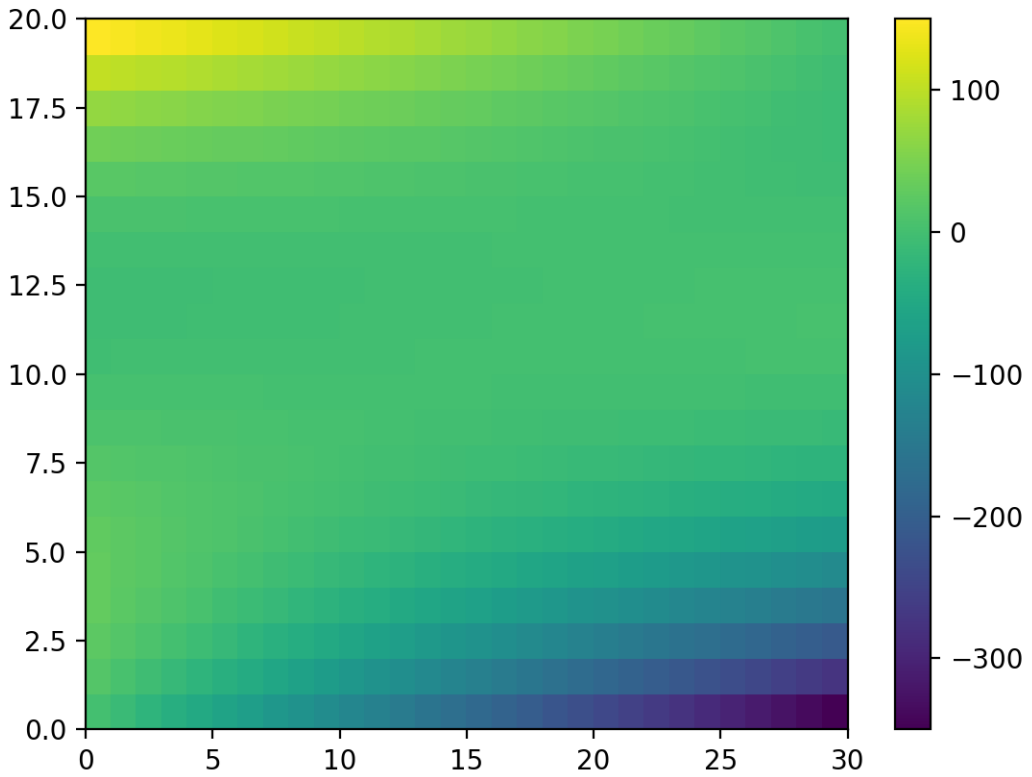
だったら**曲面**で表せばいいよね？

曲面 = 2次元配列 = 行列 を可視化する方法

- 色分け → pcolor
- 等高線 → contour
- 立体グラフ → surface (mpl_toolkits.mplot3D)

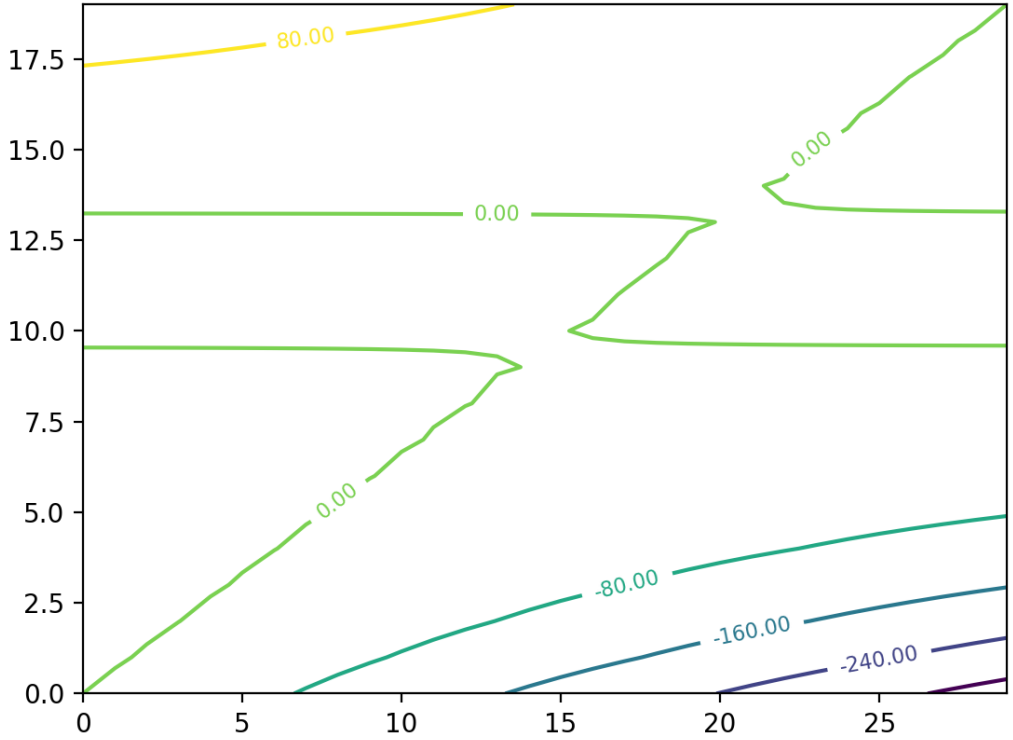
色分けで表す

```
#plt.plot(x, y)
plt.pcolor(y)

plt.colorbar()

plt.show()
```



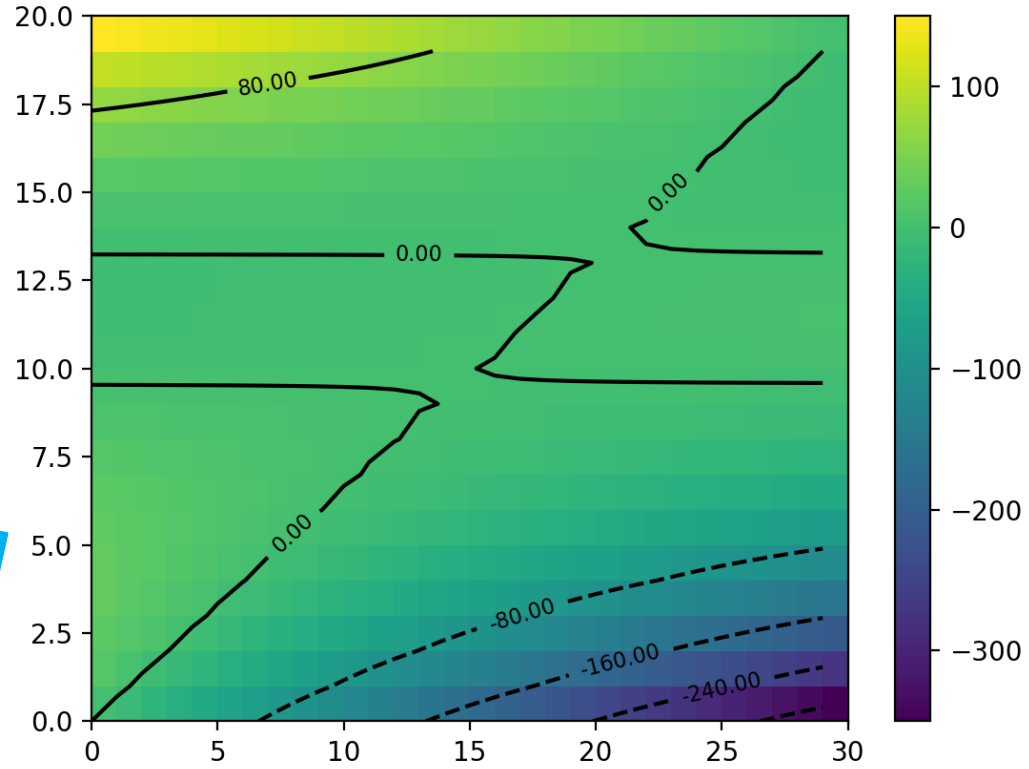
等高線で表す

```
#plt.plot(x, y)
plt.contour(y).clabel(fmt="%3.2f", fontsize=8)

plt.show()
```



色分け＋等高線で表す

```
#plt.plot(x, y)
plt.pcolor(y)

plt.colorbar()

plt.contour(y, colors="black")/
           .clabel(fmt="%3.2f", fontsize=8)

plt.show()
```



曲面で表す

```
plt.plot(x, f_2(x, 2), label="w = 2") # w = 2
plt.plot(x, f_2(x, 1), label="w = 1") # w = 1

plt.grid(True)

plt.legend()

plt.title("f_2(x,w) = (x-w)x(x+2)")
plt.xlabel("x")
plt.ylabel("f_2(x,w)")

plt.show()
```

グラフを複数表示する

```
plt.plot(x, f_2(x, 2), label="w = 2") # w = 2
plt.plot(x, f_2(x, 1), label="w = 1") # w = 1

plt.grid(True)

plt.legend()

plt.title("f_2(x,w) = (x-w)x(x+2)")
plt.xlabel("x")
plt.ylabel("f_2(x,w)")

plt.show()
```

数学をPythonで表現とどうなる？

ベクトル

行ベクトル

$$A = \begin{pmatrix} 1 & 2 \end{pmatrix}$$

```
A = np.array([1,2])
```

列ベクトル

$$A = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

```
A = np.array([[1],
              [2]])
```

ベクトル

ベクトル和

$$A + B = \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \begin{pmatrix} 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \end{pmatrix}$$

```
A = np.array([[1],[2]])
B = np.array([[3],[4]])
A + B
```

ベクトル差

$$A + B = \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \begin{pmatrix} 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \end{pmatrix}$$

```
A = np.array([[1],[2]])
B = np.array([[3],[4]])
A - B
```

行列

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

```
np.array([[1, 2, 3],
          [4, 5, 6]])
np.array([[1, 4],
          [2, 5],
          [3, 6]]).T
```

転置行列(transposed matrix)

$$^tA = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

```
np.array([[1, 4],
          [2, 5],
          [3, 6]])
np.array([[1, 2, 3],
          [4, 5, 6]]).T
```

行列

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

```
np.array([[1, 2, 3],
          [4, 5, 6]])
np.array([[1, 4],
          [2, 5],
          [3, 6]]).T
```

転置行列(transposed matrix)

$$^tA = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

```
np.array([[1, 4],
          [2, 5],
          [3, 6]])
np.array([[1, 2, 3],
          [4, 5, 6]]).T
```

微分

$$
\begin{aligned}
f(x) &= (x-2)x(x+2) \\
&= (x^2-4)x \\
&= x^3+4x
\end{aligned}
$$

偏微分

$$f(x) = (x - w)x(x + 2)$$

非線形関数

- シグモイド関数
- ソフトマックス関数
- ガウス関数

詳細は次回にて

線形と非線形

- $f(x+y)=f(x)+f(y)$ f(x+y)=f(x)+f(y)
- $f(kx)=kf(x)$ f(kx)=kf(x)　　ただし、kは実数

$$f(x+y)=f(x)+f(y)$$

To be continued…

シグモイド関数

ソフトマックス関数

ガウス関数

リファレンス

Pythonで動かして学ぶ！ あたらしい機械学習の教科書 - 伊藤　真

Python - https://www.python.org/

matplotlib - https://matplotlib.org/

matplotlib入門 - http://bicycle1885.hatenablog.com/entry/2014/02/14/023734

[Pythonによる科学・技術計算] 2次元(カラー)等高線等の描画，可視化，matplotlib
- https://qiita.com/sci_Haru/items/5b4c34d5330a545001cf