

3. matplotlib の基本

3.1 matplotlib の動かし方

matplotlib を動かすには、IPython が必要です。既にインストールが済んでいるので使ってみましょう。端末で `ipython3 --pylab` と入力すると IPython が `pylab` モードで立ち上がります。この状態で IPython は matplotlib GUI バックエンドが有効になります。

matplotlib を動かすのに必要な Matplotlib API 関数は、`matplotlib.pyplot` モジュールに含まれています。次のようにインポートしておくと、以降は `plt` として参照できるようになります。

```
In [1]: import matplotlib.pyplot as plt
```

新たな図を作成するために、`plt.figure` を使います。

```
In [2]: fig = plt.figure()
```

空のウィンドウが現れます。この状態ではまだプロットできないので、`add_subplot` を使いプロット領域を作成する必要があります。

```
In [3]: ax = fig.add_subplot(1, 1, 1)
```

図 3-1-1 のような空の 1 つのサブプロットを持ったウィンドウが作成されます。`plt.savefig(ファイル名)` を使って、プロットを保存することもできます。現在、`eps`、`jpeg`、`jpg`、`pdf`、`pgf`、`png`、`ps`、`raw`、`rgba`、`svg`、`svgz`、`tif`、`tiff` に対応しており、ファイル名の拡張子を判断して自動で変換されます。

```
In [4]: plt.savefig("subplot.png")
```

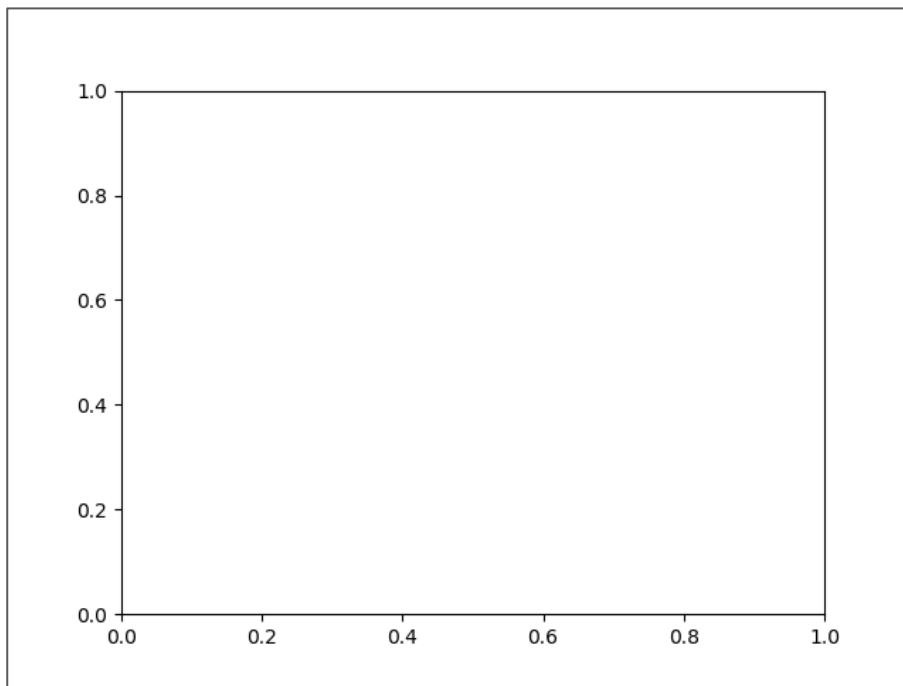


図 3 - 1 - 1 matplotlib のウィンドウに空のサブプロットを追加

In [5]: quit()

で IPython を終了します。

plt.savefig には表 3 - 1 - 1 のようなオプションがあり、例えば解像度を 300 dpi に変えられます。

```
plt.savefig("subplot.png", dpi=300)
```

また、図 3 - 1 - 1 のような周囲の余白を少なくするには、オプションとして `bbox_inches='tight'` を指定します。

表 3 - 1 - 1 plt.savefig の主要オプション

オプション	説明
fname	ファイル名、必ず必要
dpi	解像度、dots per inch、デフォルト 値：100
bbox_inches	余白を少なくしたい場合は'tight'を指定

なお ipython が pylab モードでない場合 (ipython3 のみで起動した場合)、

plt.show()を行うまでは図が表示されません。そのため、先ほどの手続きが次のように変わります。

```
In [1]: import matplotlib.pyplot as plt
```

```
In [2]: fig = plt.figure()
```

```
In [3]: ax = fig.add_subplot(1, 1, 1)
```

```
In [4]: plt.show()
```

```
In [5]: quit()
```

3.2 関数グラフの作成

まだグラフを描いていないので、次は簡単なグラフを作ってみます。もう一度 `ipython3 --pylab` で IPython を起動してみます。

```
In [1]: import matplotlib.pyplot as plt
```

```
In [2]: fig, ax = plt.subplots()
```

先ほどとは違う書式ですが、ウィンドウを生成し空のサブプロットを生成する所までを同時に行ってくれます。関数などのグラフを作る時に便利なのが NumPy です。NumPy は Python において数値計算を効率的に行うための拡張モジュールで、関数や配列などを扱うことができ多くの数学関数、統計関数が用意されています。NumPy を利用する場合、次のように import します。

```
In [3]: import numpy as np
```

それでは、NumPy を使って $\cos(x)$ を作図してみましよう。 `np.linspace` は、線形に等間隔な数列を生成する関数です。円周率を返す `np.pi` と組み合わせて `plt.plot` を使い $[-\pi, \pi]$ で作図します(図 3-2-1)。

```
In [4]: x = np.linspace(-np.pi, np.pi)
```

```
In [5]: plt.plot(x, np.cos(x), color='k', ls='-', label='cos(x)')
```

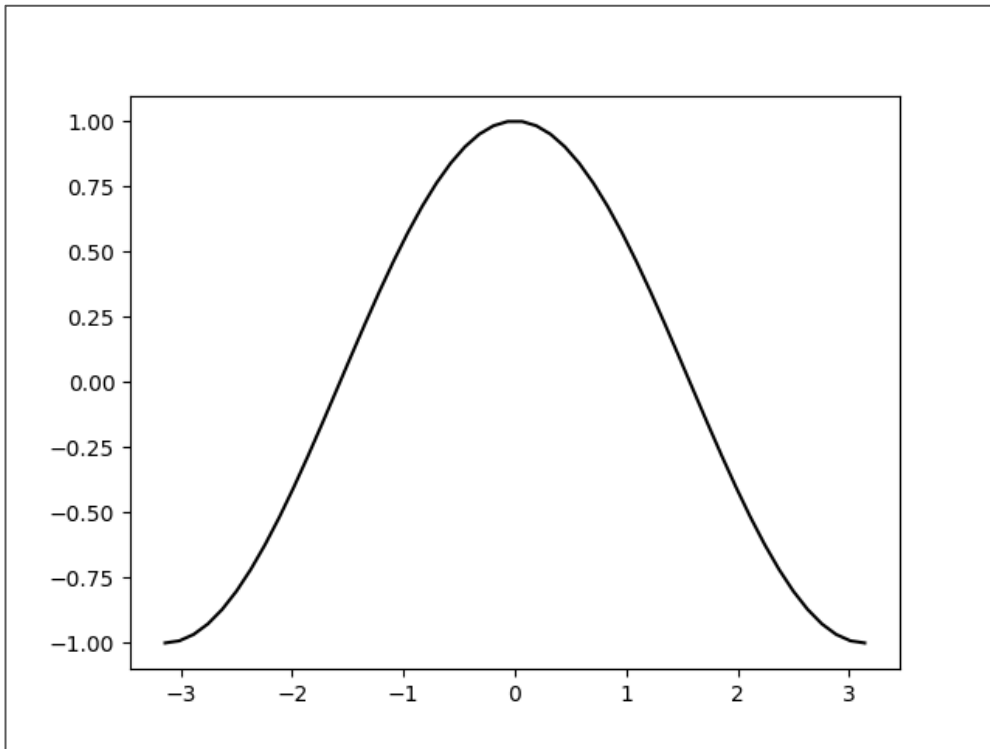


図 3-2-1 サブプロットに $\cos(x)$ をプロットしたもの

`plt.plot(x, np.cos(x))`でも動きますが、後ろに付けた `color='k'`, `ls='-'`がオプションで、それぞれ色 (黒)、線種 (実線) を意味しています。最後の `label='cos(x)'` は凡例のラベルで、後ほど説明します。`color` には図 3-2-2 のような色指定が可能で、`color='k'` と `c='k'`、`color='red'`、`c='red'` のいずれも同じです。指定可能な色の一覧を図 3-2-3 に載せておきます。どうしても自分で色を作りたい場合は、`color='#d62728'` のように、`'#rrggbb'` の書式で RGB の 0 ~ 255 の値を 2 桁の 16 進数で入力します。

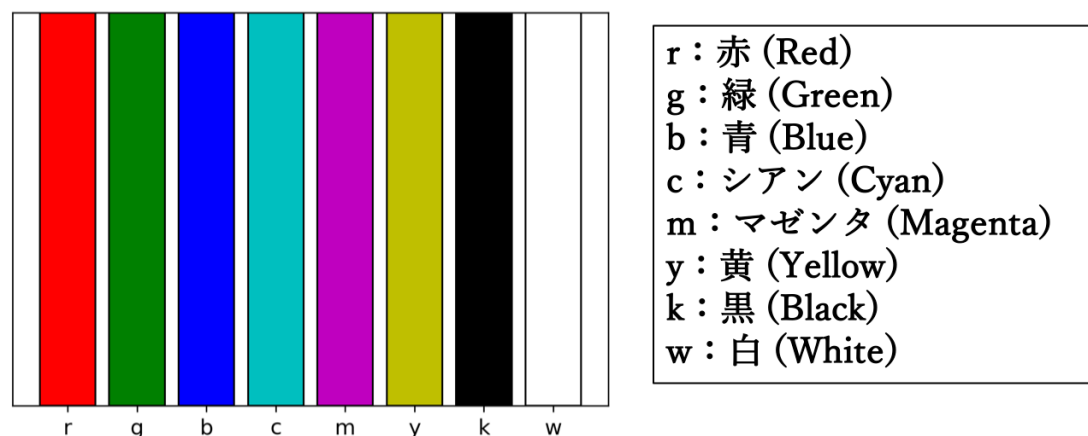


図 3-2-2 matplotlib の色指定。赤なら color="r"、または c="r"のように指定する

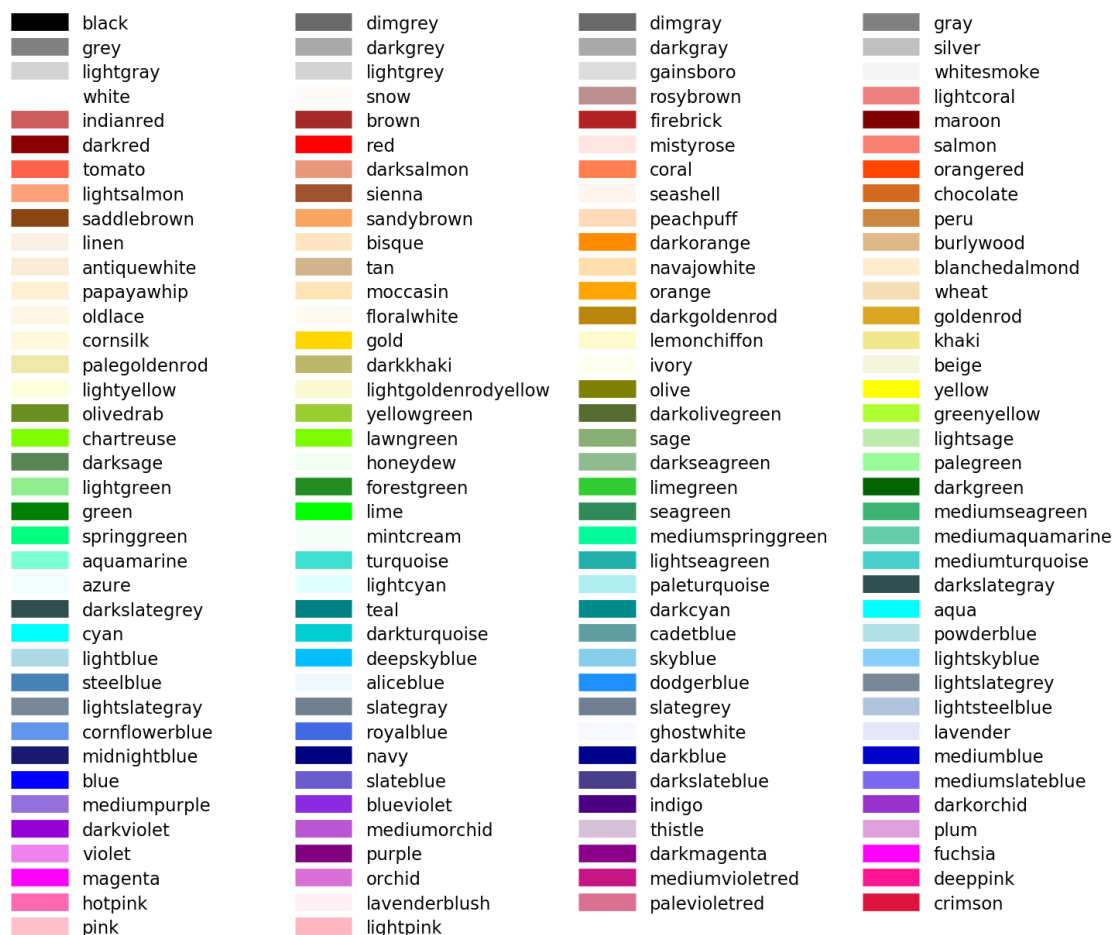


図 3-2-3 matplotlib で指定可能な色の一覧

<https://pythondatascience.plavox.info/wp-content/uploads/2016/06/colorpalette.png>





今の図に $\sin(x)$ も重ねてみます。

```
In [6]: plt.plot(x, np.sin(x), color='r', ls='--', label='sin(x)')
```

$\sin(x)$ を描く際に、`ls='--'` で線種を破線に変えています。線種として使用可能なものを表 3-2-1 にまとめました。線を区別するために `plt.legend` で凡例も追加します (図 3-2-4)。

```
In [7]: plt.legend(loc='best')
```

表 3-2-1 matplotlib の `linestyle` で指定可能な線種一覧

<code>linestyle='-'</code> or <code>'solid'</code>	: 実線	
<code>linestyle='--'</code> or <code>'dashed'</code>	: 破線	
<code>linestyle=':'</code> or <code>'dotted'</code>	: 点線	
<code>linestyle='-.'</code> or <code>'dashdot'</code>	: 一点鎖線	

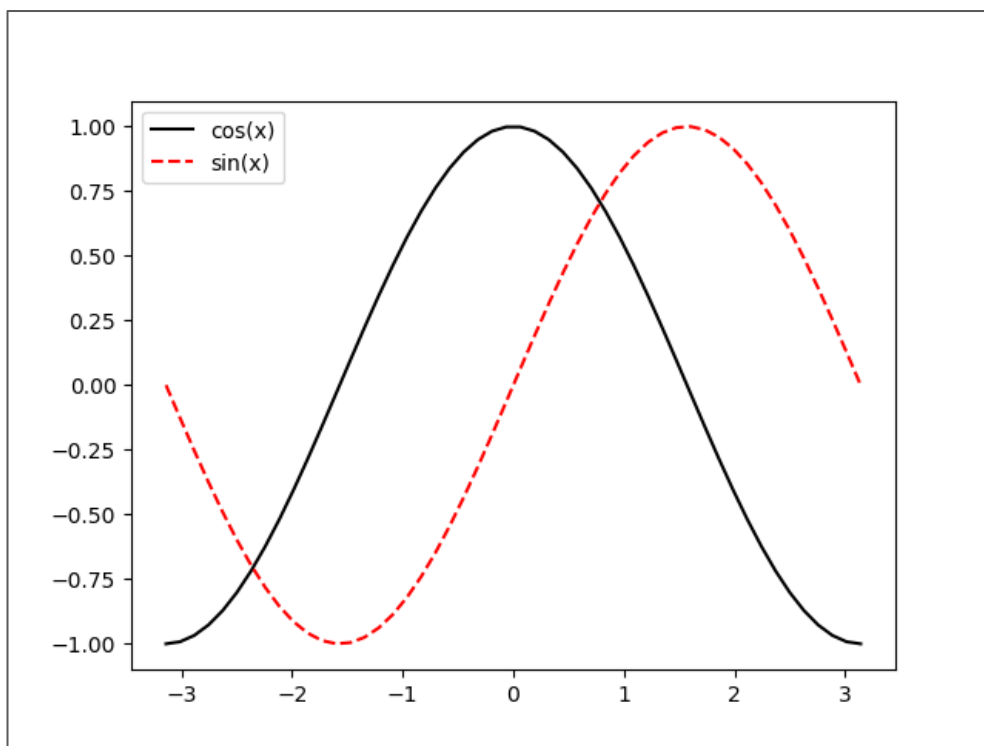


図 3-2-4 さらに $\sin(x)$ もプロットし、凡例も追加したもの

凡例の場所は `loc='best'` では自動指定ですが、`loc='upper left'` のように強制的に位置を指定することも可能です（図 3-2-5）。

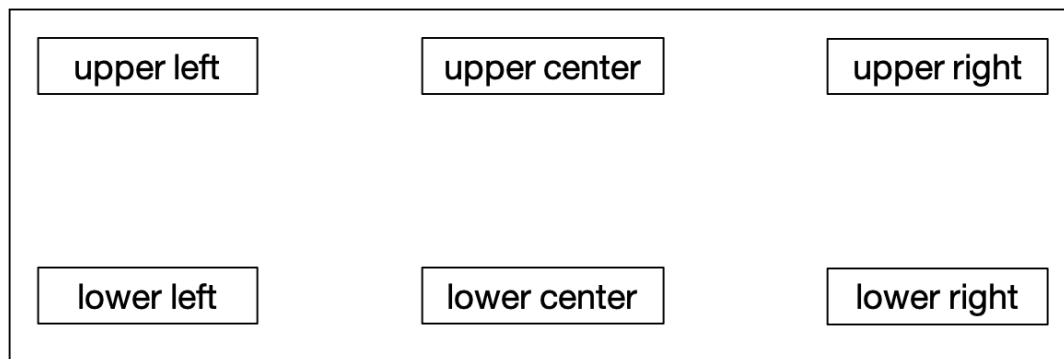


図 3-2-5 `plt.legend` で指定可能な位置。他に `best` で自動指定も可能

3.3 グラフにマーカーを追加する

これだけでは物足りないので、グラフにマーカーを追加する方法を考えていきます。まずは、これまでと同じように `python3 --pylab` を起動してサブプロットを作成し $[-\pi, \pi]$ の範囲を作成します。

```
In [1]: import matplotlib.pyplot as plt
```

```
In [2]: fig, ax = plt.subplots()
```

```
In [3]: import numpy as np
```

```
In [4]: x = np.linspace(-np.pi, np.pi)
```

グラフにマーカーを追加してみます (図 3-3-1)。`np.tanh(x)` で $\tanh(x)$ を作図します。`marker='x'` で×印のマーカーが付きます。

```
In [5]: plt.plot(x, np.tanh(x), color='k', ls='-', label='tanh(x)', marker='x')
```

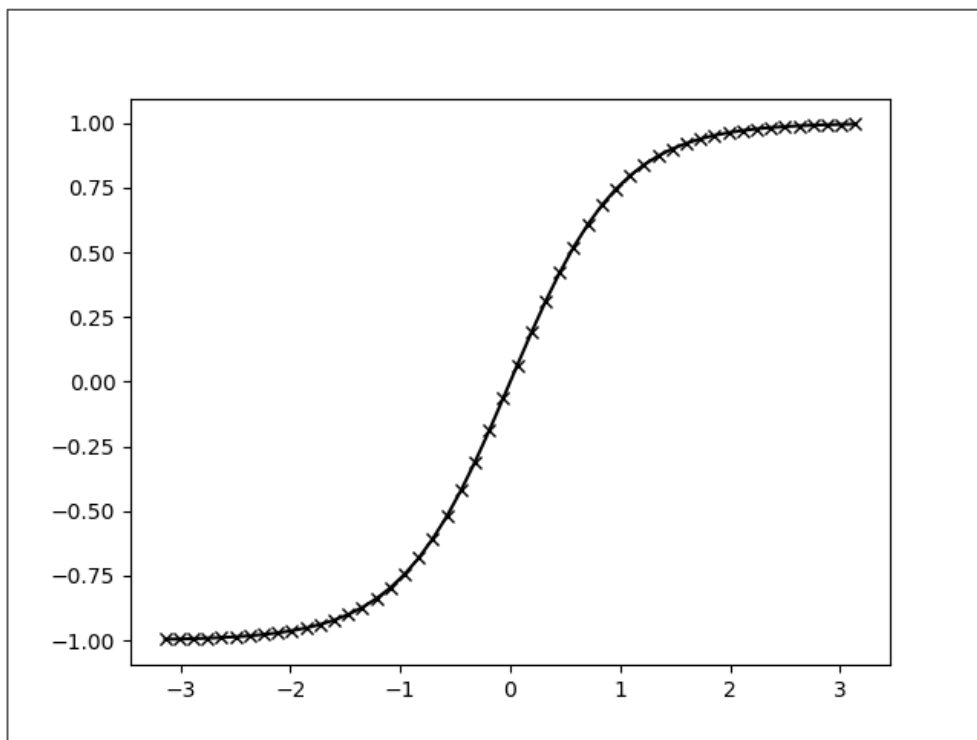


図 3-3-1 $\tanh(x)$ のグラフに×印のマーカーを追加

さらに $\tan^{-1}(x)$ を青色で重ねてみます (図 3-3-2)。marker='o', fillstyle='none'で open circle になります。

In [6]: plt.plot(x, np.arctan(x), color='b', ls='-', label='arctan(x)', marker='o', fillstyle='none')

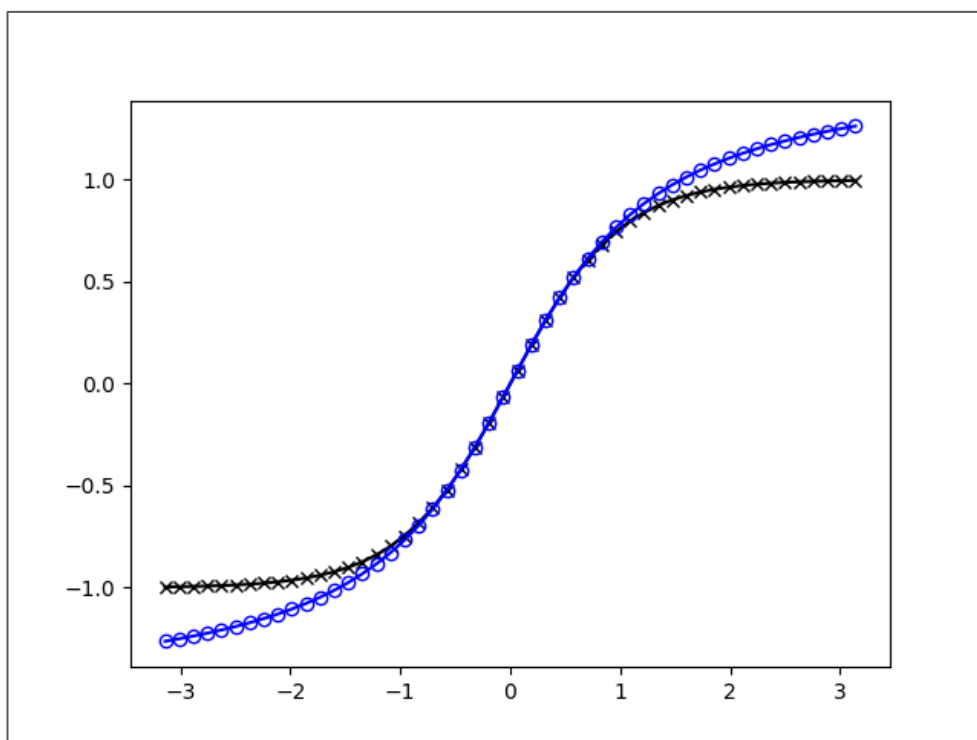


図 3-3-2 図 3-3-1 にさらに青色で $\tan^{-1}(x)$ を重ねたもの

matplotlib では数多くのマーカーが用意されています (表 3-3-1)。

表 3 - 3 - 1 matplotlib で指定可能なマーカーの一覧

marker	description	marker	description
"."	point	TICKLEFT	tickleft
"."	pixel	TICKRIGHT	tickright
"o"	circle	TICKUP	tickup
"v"	triangle_down	TICKDOWN	tickdown
"^"	triangle_up	CARETLEFT	caretleft (centered at tip)
"<"	triangle_left	CARETRIGHT	caretright (centered at tip)
">"	triangle_right	CARETUP	caretup (centered at tip)
"1"	tri_down	CARETDOWN	caretdown (centered at tip)
"2"	tri_up	CARETLEFTBASE	caretleft (centered at base)
"3"	tri_left	CARETRIGHTBASE	caretright (centered at base)
"4"	tri_right	CARETUPBASE	caretup (centered at base)
"8"	octagon	"None", "" or ""	nothing
"s"	square	'\$...\$'	render the string using mathtext.
"p"	pentagon	verts	a list of (x, y) pairs used for Path vertices. The center of the marker is located at (0,0) and the size is normalized.
"P"	plus (filled)	path (numsides, style, angle)	a Path instance. numsides: the number of sides style: the style of the regular symbol: 0: a regular polygon 1: a star-like symbol 2: an asterisk 3: a circle (numsides and angle is ignored) angle: the angle of rotation of the symbol
"*"	star		
"h"	hexagon1		
"H"	hexagon2		
"+"	plus		
"x"	x		
"X"	x (filled)		
"D"	diamond		
"d"	thin_diamond		
" "	vline		
"_"	hline		

marker='o'では塗りつぶしを変えることもできるので試してみましょう。まず marker='o', fillstyle='full'で closed circle にしてみます。

```
In [7]: plt.plot(x, np.arccosh(x), color='r', ls='-', label='arccosh(x)', marker='o', fillstyle='full')
```

ちょっと変わったマーカーを付けることもでき、marker='o', fillstyle='left'とすれば、左側だけ塗り潰せます（図 3 - 3 - 3）。

```
In [8]: plt.plot(x, np.arcsinh(x), color='c', ls='-', label='arcsinh(x)', marker='o', fillstyle='left')
```

```
In [9]: plt.legend(loc='best')
```

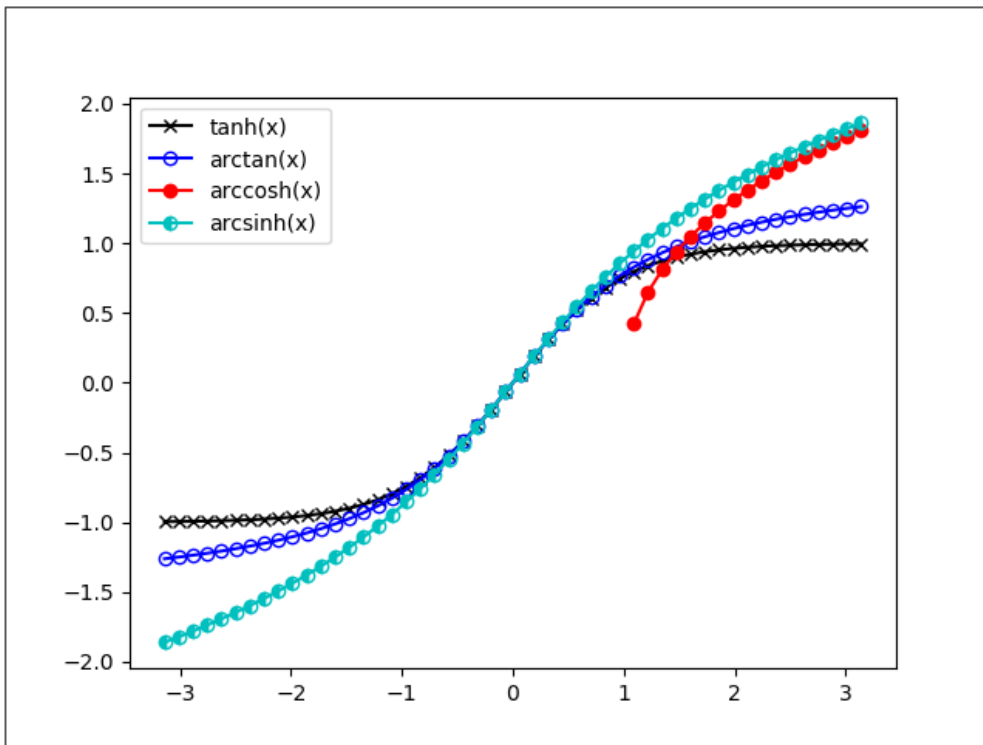


図 3-3-3 図 3-3-2 に $\cosh^{-1}(x)$ と $\sinh^{-1}(x)$ を重ね、凡例も付けた

他にも様々な塗りつぶしオプションがあるので一覧にしました（表 3-3-2）。

表 3-3-2 matplotlib の fillstyle で指定可能な塗り潰しオプション一覧

fillstyle : マークの塗り潰し		
fillstyle='full'	: 全部●.....
fillstyle='none'	: なし○.....
fillstyle='left'	: 左側◐.....
fillstyle='right'	: 右側◑.....
fillstyle='top'	: 上側◕.....
fillstyle='bottom'	: 下側◖.....

3.4 複数のグラフを並べる

これまでは、ウィンドウに1つのサブプロットのみでしたが、グラフを並べて表示したいこともあるでしょう。サブプロットは複数配置可能ですので、試してみましょう。fig.add_subplot を使い縦に2つの図を並べてみます。

```
In [1]: import matplotlib.pyplot as plt
```

```
In [2]: fig = plt.figure()
```

```
In [3]: ax1 = fig.add_subplot(2, 1, 1)
```

```
In [4]: ax2 = fig.add_subplot(2, 1, 2)
```

fig.add_subplot の最初の引数が縦に並べる数、2つ目が横に並べる数、3つ目がサブプロットのうちの何番目に当たるかを表します。ax1 が上のサブプロット、ax2 が下のサブプロットに対応します。それぞれのサブプロットに図を描くには、ここで定義した ax1、ax2 を使います。

これまでのように $[-\pi, \pi]$ の範囲を作成し、上のサブプロット ax1 に $\sinh(x)$ を描き凡例を追加してみます (図 3-4-1)。サブプロットを指定して描く場合は、先ほどの plt.plot の代わりに ax1.plot のように指定します。凡例についても同様に、plt.legend の代わりに ax1.legend を使います。

```
In [5]: import numpy as np
```

```
In [6]: x = np.linspace(-np.pi, np.pi)
```

```
In [7]: ax1.plot(x, np.sinh(x), color='b', ls='-', lw=6, label='sinh(x)')
```

```
In [8]: ax1.legend(loc='best')
```

線がこれまでより太いのが分かります。lw=6 と指定したためです (デフォルトは lw=1)。lw=6 の代わりに linewidth=6 としても同じです。下のサブ

プロット ax2 に $\cosh(x)$ を描き凡例を追加してみます (図 3-4-2)。ls=""とすれば、線が消えてマーカーのみになります。marker='x', ms='3'で×印で大きさ3のマーカーを追加します (デフォルトは ms=6)。markersize=3 としても同じです。

```
In [9]: ax2.plot(x, np.cosh(x), color='r', ls="", marker='x', ms='3',  
label='cosh(x)')
```

```
In [10]: ax2.legend(loc='best')
```

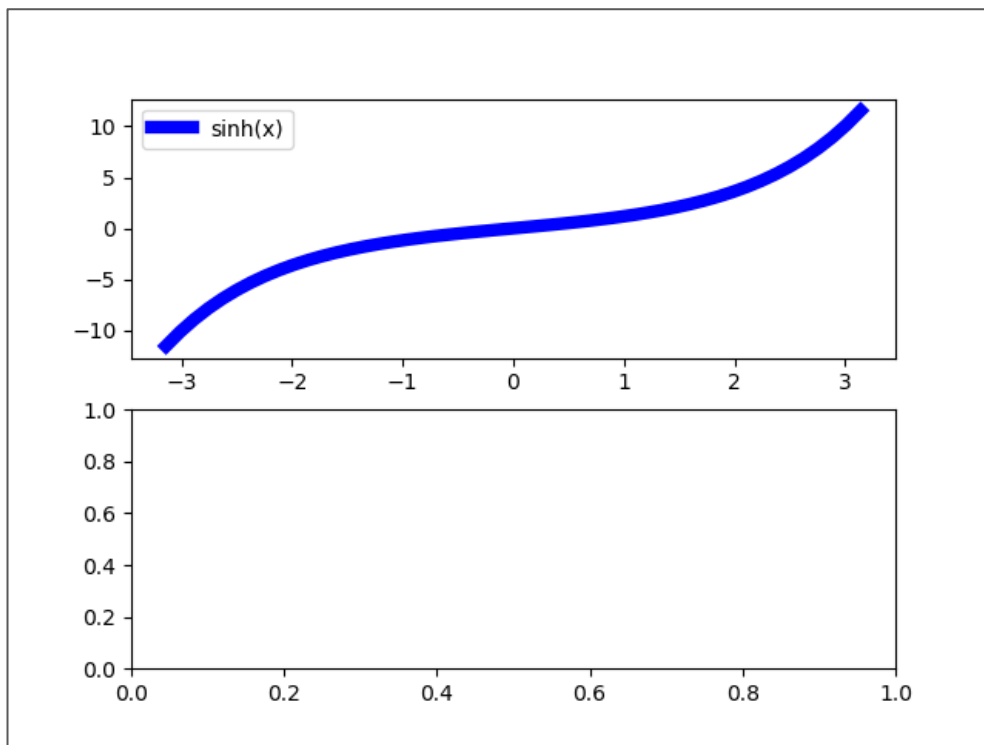


図 3-4-1 matplotlib のウィンドウに2つのサブプロットを追加し、上のサブプロットに $\sinh(x)$ を描き凡例を追加した

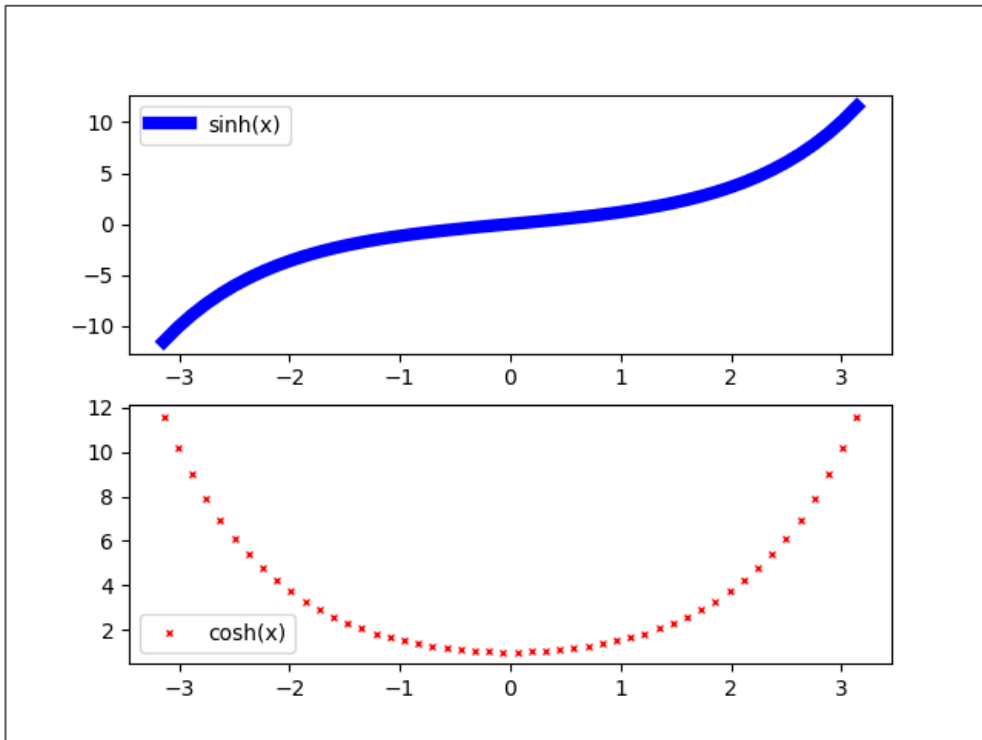


図 3-4-2 下のサブプロットに $\cosh(x)$ と凡例を追加

1つのウィンドウに4つのサブプロットを追加することもできます。

```
In [1]: import matplotlib.pyplot as plt
```

```
In [2]: fig = plt.figure(figsize=(9, 6))
```

これまでとは異なり、`plt.figure` でウィンドウを作成する際に `figsize=(9, 6)` で大きさを指定しました。このようにすれば図の（横、縦）のサイズを指定することが可能です。ウィンドウの中にサブプロットを作成します。

```
In [3]: ax1 = fig.add_subplot(2, 2, 1)
```

ウィンドウの左上に図が出てきます。3つ目の引数の1番目が左上、2番目が右上、3番目が左下、4番目が右下に対応します。サブプロットの(0.4, 0.4)の位置に `plt.text` でテキストを書いてみます。一番目の引数が横の位置、二番目の引数が縦の位置、3番目の引数がテキストで、`fontsize=20` で文字の大きさを

指定できます。

```
In [4]: plt.text(0.4, 0.4, "(2,2,1)", fontsize=20, color='k')
```

```
Out[4]: Text(0.4,0.4,'(2,2,1)')
```

右上のサブプロットを追加し、rotation=30 で 30 度回転させてみます。

```
In [5]: ax2 = fig.add_subplot(2, 2, 2)
```

```
In [6]: plt.text(0.4, 0.4, "(2,2,2)", rotation=30, fontsize=20, color='k')
```

```
Out[6]: Text(0.4,0.4,'(2,2,2)')
```

左下にサブプロットを追加し、横にちょっとずらして (0.2、0.4) に配置します。新たに出てきた alpha=0.7 は不透明度を指定する引数で、0 が透明、1 が不透明です（デフォルトは 1）。

```
In [7]: ax3 = fig.add_subplot(2, 2, 3)
```

```
In [8]: plt.text(0.2, 0.4, "(2,2,3)", fontsize=20, color='k', alpha=0.7)
```

```
Out[8]: Text(0.2,0.4,'(2,2,3)')
```

右下にもサブプロットを追加します（図 3-4-3）。

```
In [9]: ax4 = fig.add_subplot(2, 2, 4)
```

```
In [10]: plt.text(0.2, 0.4, "(2,2,4)", fontsize=20, color='k')
```

```
Out[10]: Text(0.2,0.4,'(2,2,4)')
```

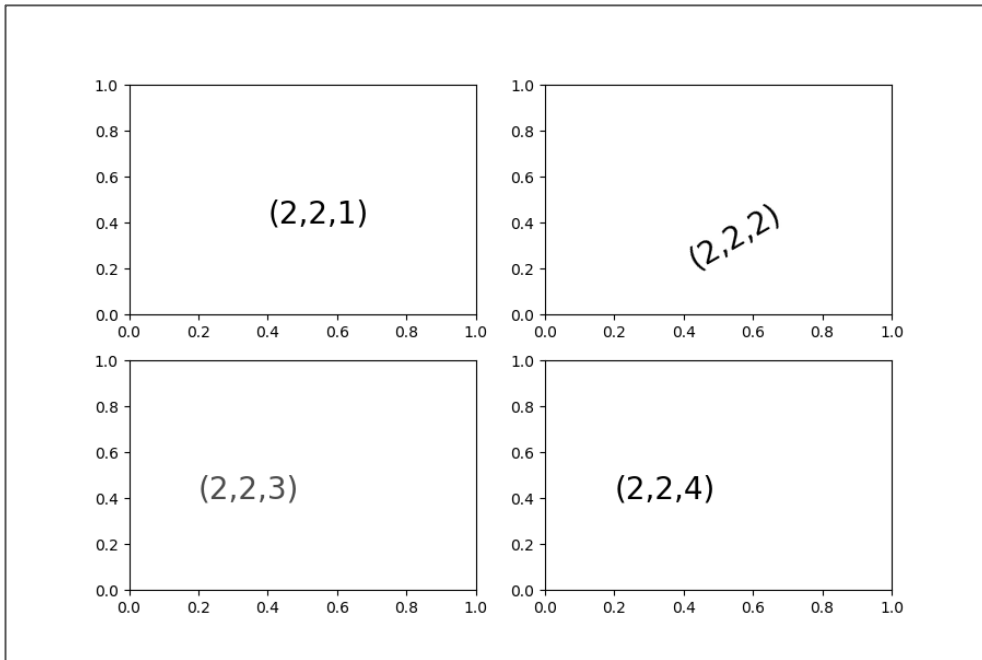



図 3-4-3 ウィンドウに4つのサブプロットを追加した

図に $x * \sin(x)$ のグラフを追加してみます。オプションとして追加した `drawstyle='default'` は、これまでも使用してきたデフォルト設定の折れ線グラフです。

```
In [11]: import numpy as np
```

```
In [12]: x = np.linspace(-np.pi, np.pi)
```

```
In [13]: ax1.plot(x, x * np.sin(x), color='r', ls='--', drawstyle='default')
```

```
Out[13]: [<matplotlib.lines.Line2D at 0x117cf0160>]
```

他に3種類の設定が使えるので、`ax2~ax4` に並べてみます (図 3-4-4)。

`ax2` : `drawstyle='steps-post'` : 階段状 (後側)

`ax3` : `drawstyle='steps-pre'` または `'steps'` : 前側

`ax4` : `drawstyle='steps-mid'` : 中央

```
In [14]: ax2.plot(x, x * np.sin(x), color='r', ls='-', drawstyle='steps-post')
```

Out[14]: [<matplotlib.lines.Line2D at 0x118141048>]

In [15]: ax3.plot(x, x * np.sin(x), color='r', ls='-', drawstyle='steps-pre')

Out[15]: [<matplotlib.lines.Line2D at 0x1181509b0>]

In [16]: ax4.plot(x, x * np.sin(x), color='r', ls='-', drawstyle='steps-mid')

Out[16]: [<matplotlib.lines.Line2D at 0x1183b07b8>]

3種類の階段状のグラフは、よく見ないと違いが分かりませんが、グラフの両端をみると、前側、後側、両方、のいずれに飛び出しているかが違ってきます。

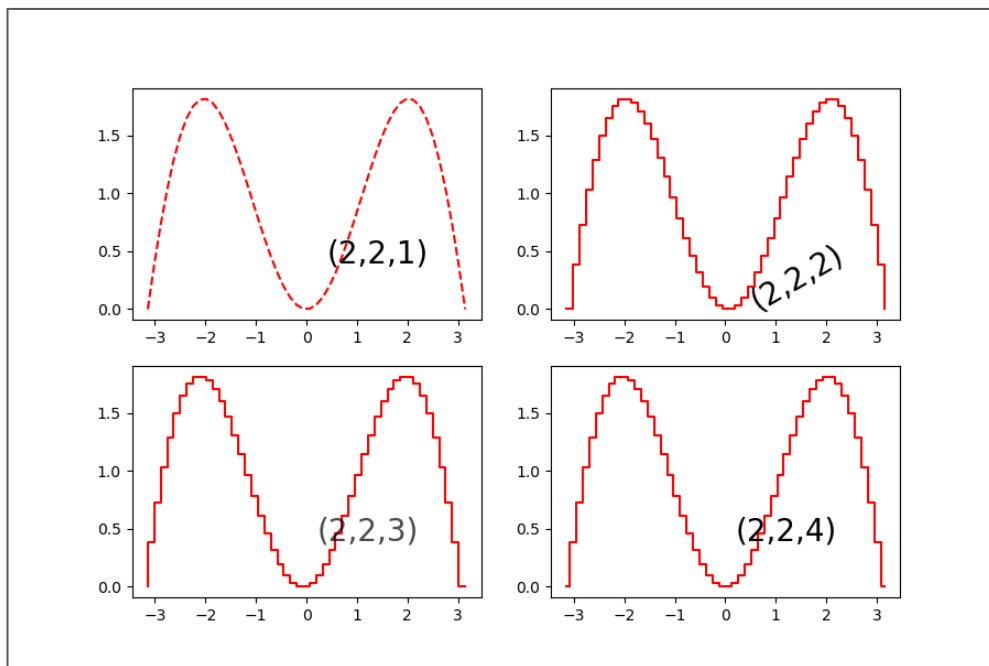


図3-4-4 $x \sin(x)$ のグラフ。drawstyleを変えて並べた

最後に折れ線グラフで利用できるオプション一覧を表3-4-1にまとめておきます。

表 3 - 4 - 1 matplotlib の pyplot.plot のオプション一覧

オプション	説明
x, y	x軸上の座標、y軸上の値
color or c	線やマーカーの色
linestyle or ls	線の種類、デフォルト値：'-'
linewidth or lw	線の太さ、デフォルト値：1
drawstyle	線を描く時のスタイル、デフォルト値：'default'
marker	マーカーの種類、デフォルト値：'None'
markeredgecolor or mec	マーカーの淵の色
markeredgewidth or mew	マーカーの淵の幅、デフォルト値：1
markerfacecolor or mfc	マーカーの内部の色
markersize or ms	マーカーの大きさ、デフォルト値：6
fillstyle	マーカーの塗り潰しスタイル、デフォルト値：'full'
alpha	不透明度、デフォルト値：1.0
label	凡例を付ける場合

使用方法：plt.plot(x, y)、plt.plot(x, y, オプション)

3.5 グラフの体裁を整える

グラフを作図した際にサイズを調整したり、目盛り線を変更したり、タイトルを付けたりするなど、体裁を整えたいこともあると思います。いくつかの例を紹介しておきます。まずは $\sinh(x)$ を描いてみます。ウィンドウを作成する際に、`plt.figure(figsize=(6, 3))` としています。

```
In [1]: import matplotlib.pyplot as plt
```

```
In [2]: fig = plt.figure(figsize=(6, 3))
```

```
In [3]: ax = fig.add_subplot(1, 1, 1)
```

```
In [4]: import numpy as np
```

```
In [5]: x = np.linspace(-np.pi, np.pi)
```

```
In [6]: ax.plot(x, np.sinh(x), color='k')
```

```
Out[6]: [<matplotlib.lines.Line2D at 0x1116b0908>]
```

図に `plt.title` でタイトルを付けてみます (図 3-5-1)。`fontsize=24` で文字の大きさを 24 ポイントに変えました。

```
In [7]: plt.title("sinh(x)", fontsize=24)
```

```
Out[7]: Text(0.5,1,'sinh(x)')
```

ちなみに、`fontsize` は整数の他に文字列で指定することもでき、`"xx-small"`、`"x-small"`、`"small"`、`"medium"`、`"large"`、`"x-large"`、`"xx-large"` を指定可能です。`fontweight` というオプションで文字の太さを変えることもでき、0 ~ 1000 の整数か、`"ultralight"`、`"light"`、`"normal"`、`"regular"`、`"book"`、`"medium"`、`"roman"`、`"semibold"`、`"demibold"`、`"demi"`、`"bold"`、`"heavy"`、`"extra bold"`、`"black"` の文字列を指定可能です。例えば太字にするには、`fontweight="bold"` です。`fontsize` オプションは見た目に反映されますが、`fontweight` オプションは細かく設定し

でも変わらないようです（表 3-5-1）。

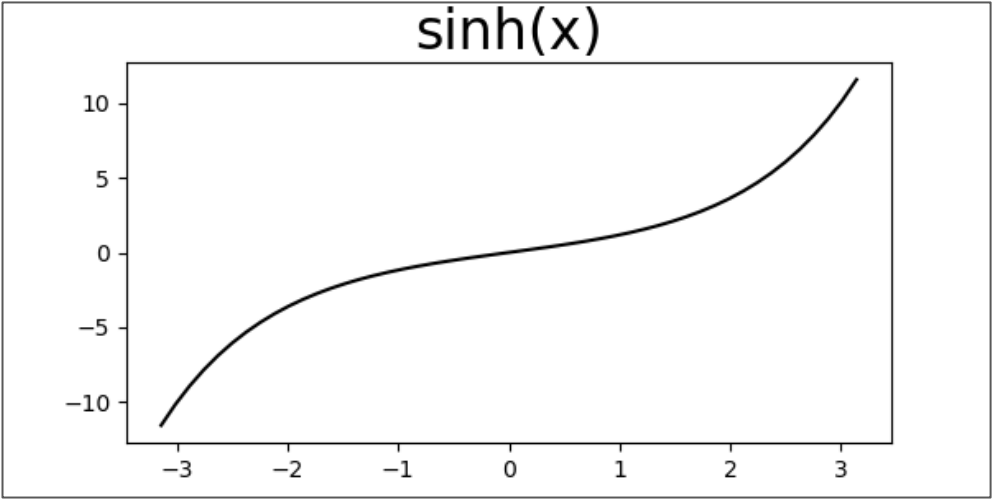


図 3-5-1 $\sinh(x)$ をプロットし、タイトルを付けた

表 3-5-1 文字列の大きさと太さを指定するオプション一覧

<code>fontsize=None</code> <code>fontsize='xx-small'</code> <code>fontsize='x-small'</code> <code>fontsize='small'</code> <code>fontsize='medium'</code> <code>fontsize='large'</code> <code>fontsize='x-large'</code> <code>fontsize='xx-large'</code>	<code>fontweight=None</code> <code>fontweight='ultralight'</code> <code>fontweight='light'</code> <code>fontweight='normal'</code> <code>fontweight='regular'</code> <code>fontweight='book'</code> <code>fontweight='medium'</code> <code>fontweight='roman'</code> <code>fontweight='semibold'</code> <code>fontweight='demibold'</code> <code>fontweight='demi'</code> <code>fontweight='bold'</code> <code>fontweight='heavy'</code> <code>fontweight='extra bold'</code> <code>fontweight='black'</code>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

軸の大目盛りの間隔を変更し、ラベルの付いていない小目盛りも追加します。目盛り線の設定には `matplotlib.ticker` を使います。`ticker.AutoLocator()` が大目盛り、`ticker.AutoMinorLocator()` が小目盛りの自動設定を返し、`axis.set_major_locator` や、`axis.set_minor_locator` の値を置き換えます。x 軸に小目盛りが追加されたでしょうか（図 3-5-2）。大目盛りの方は `ticker.AutoLocator()` の値が 1 毎だったので、変更はありません。

```
In [8]: import matplotlib.ticker as ticker
```

```
In [9]: ax.xaxis.set_major_locator(ticker.AutoLocator())
```

```
In [10]: ax.xaxis.set_minor_locator(ticker.AutoMinorLocator())
```

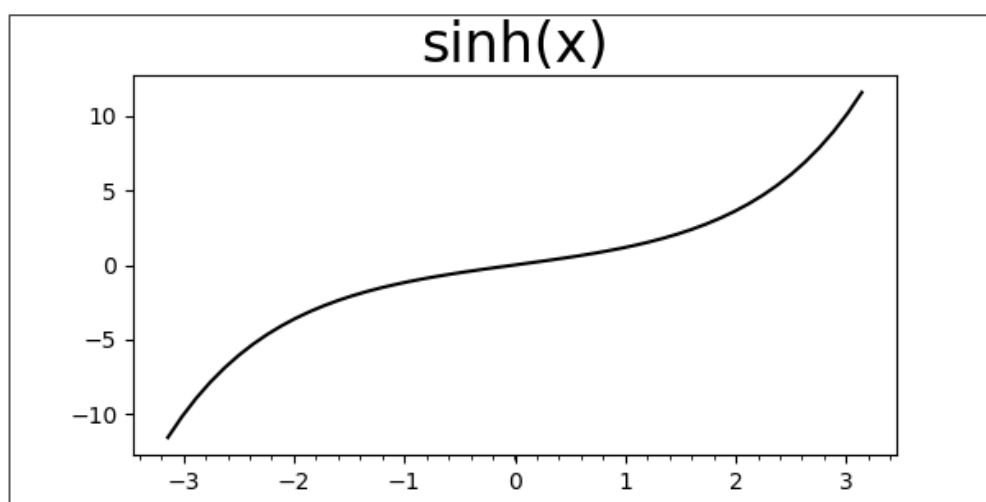


図 3-5-2 x 軸に小目盛りを追加した

y 軸についても変更します。今度は `ticker.MultipleLocator(値)` を使い、大目盛りを 10 毎、小目盛りを 2 毎に手動設定してみます（図 3-5-3）。`MultipleLocator` は指定した値の間隔で目盛りの設定を返すものです。

```
In [11]: ax.yaxis.set_major_locator(ticker.MultipleLocator(10.00))
```

```
In [12]: ax.yaxis.set_minor_locator(ticker.MultipleLocator(2.00))
```

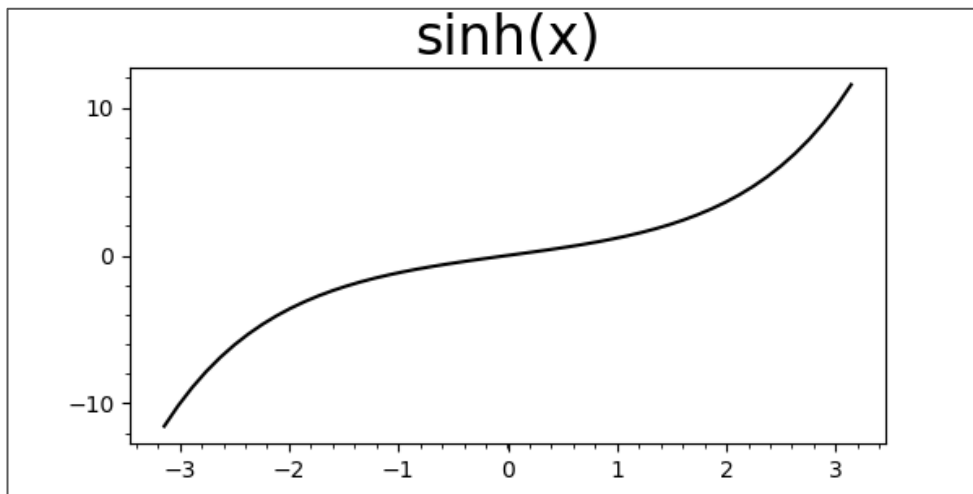


図 3-5-3 y 軸の大目盛りを 10 毎、小目盛りを 2 毎にした

x 軸と y 軸にラベルを付けてみます。ax.set_xlabel、ax.set_ylabel を使います。文字サイズや色を同時に指定可能です。黒色で x-axis、赤色で y-axis が表示されたでしょうか。サブプロットを複数設定した場合は、ax の番号を変えて、それぞれのサブプロットに別のラベルを設定可能です。

```
In [13]: ax.set_xlabel("x-axis", fontsize=20)
```

```
Out[13]: Text(0.5,61.3222,'x-axis')
```

```
In [14]: ax.set_ylabel("y-axis", fontsize=20, color='r')
```

```
Out[14]: Text(30.4722,0.5,'y-axis')
```

x-axis が図からはみ出しているので、plt.subplots_adjust で調整してみます。hspace=0.8 で水平を 8 割の大きさに、bottom=0.2 で下に 2 割の空きを付けます (図 3-5-4)。

```
In [15]: plt.subplots_adjust(hspace=0.8, bottom=0.2)
```

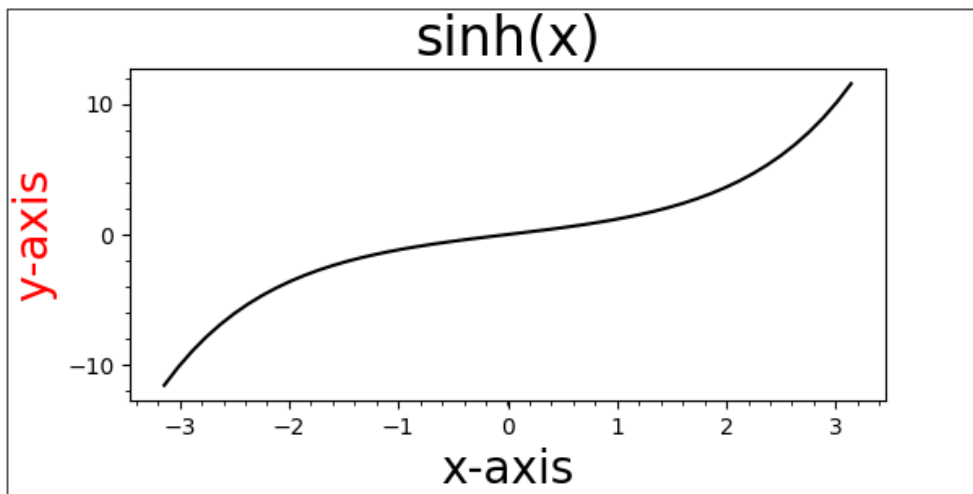


図3-5-4 x軸、y軸のラベルを追加し、プロット範囲も調整

さらに、`plt.grid` を使い大目盛りの位置にグリッド線を引きます。折れ線グラフ同様、色や線種、線の太さなども指定できます。灰色の点線に設定してみます(図3-5-5)。

```
In [16]: plt.grid(color='gray', ls=':')
```

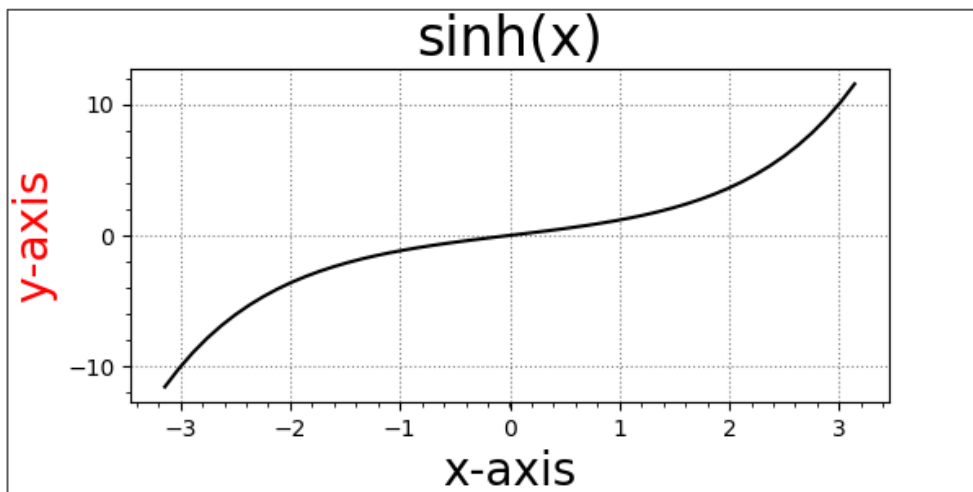


図3-5-5 グリッド線を追加した

$x=0$ 、 $y=0$ などの線を追加することもできます。`plt.axhline` で水平方向、`plt.axvline` で鉛直方向の線を引きます。色や線種が指定できるので、黒の破線

に設定してみます (図 3-5-6)。

```
In [17]: plt.axhline(y=0, color='k', ls='--')
```

```
Out[17]: <matplotlib.lines.Line2D at 0x111707828>
```

```
In [18]: plt.axvline(x=0, color='k', ls='--')
```

```
Out[18]: <matplotlib.lines.Line2D at 0x111708518>
```

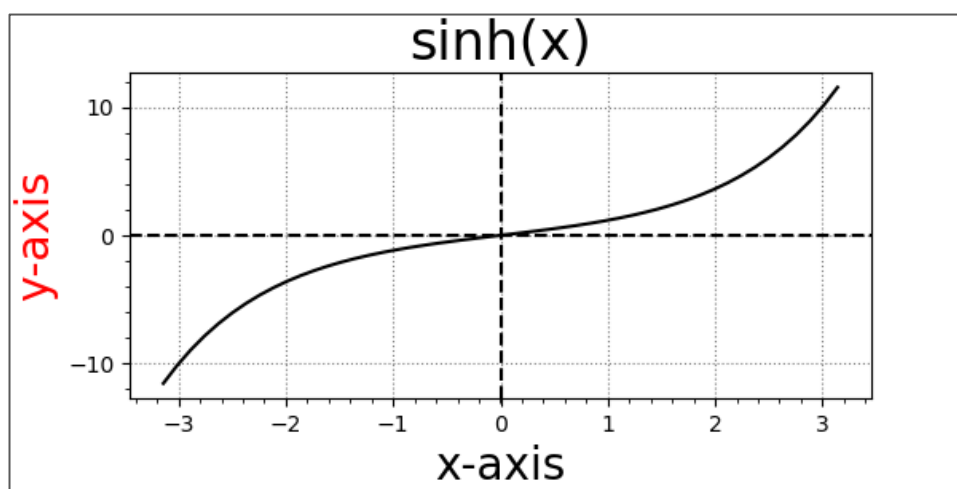


図 3-5-6 $x=0$ 、 $y=0$ の線を引いた

折れ線グラフと一緒に使える機能をもう 1 つ紹介しておきます。まずは点線で $\cosh(x)$ を重ねます。 $\sinh(x)$ より $\cosh(x)$ の値が大きいのので、`plt.fill_between` (`ax.fill_between` も同じ) を使い、その間を灰色で塗り潰してみます (図 3-5-7)。1 番目の引数が x 軸、2 番目の引数が y 軸を塗り潰す下限値、3 番目の引数が y 軸を塗り潰す上限値を表しています。 `color='gray'`, `alpha=0.4` で灰色の半透明な色で塗り潰します。 `plt.title` でタイトルも変えておきます。先ほどの `plt.title` をもう一度呼び出すと、これまでのタイトルが置き換えられます。

```
In [19]: ax.plot(x, np.cosh(x), color='k', ls='--')
```

```
Out[19]: [<matplotlib.lines.Line2D at 0x10c9d6828>]
```

```
In [20]: ax.fill_between(x, np.sinh(x), np.cosh(x), color='gray', alpha=0.4)
```

Out[20]: <matplotlib.collections.PolyCollection at 0x10c9ec278>

In [21]: plt.title("sinh(x) & cosh(x)", fontsize=24)

Out[21]: Text(0.5,1,'sinh(x) & cosh(x)')

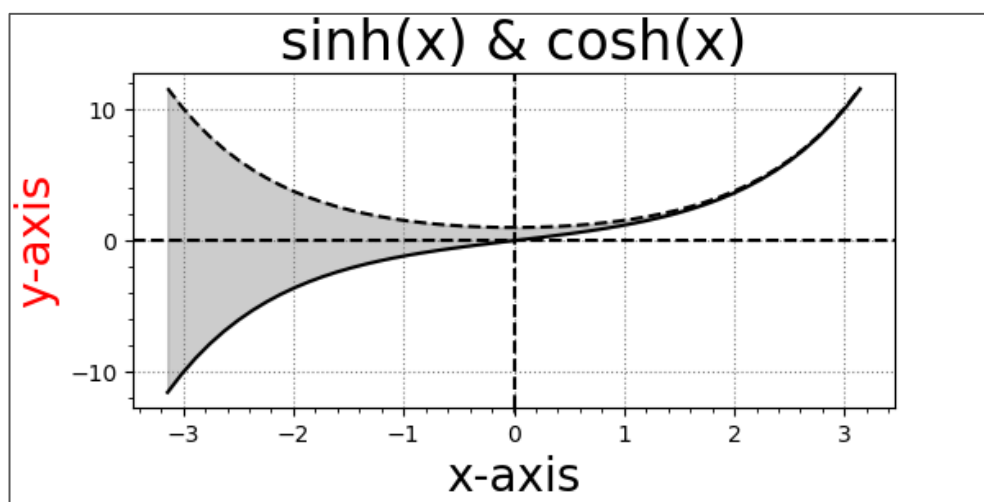


図 3-5-7 sinh(x) と cosh(x) の間を灰色で塗り潰した

最後に、グラフの体裁を整える際に用いることがある主なメソッドを表 3-5-2 にまとめておきます。

表 3-5-2 図の体裁を整える時に用いるメソッド一覧

plt.メソッド	ax.メソッド	説明
plt.title	ax.set_title	グラフのタイトルを付ける
plt.xlabel	ax.set_xlabel	x軸のラベルを付ける
plt.ylabel	ax.set_ylabel	y軸のラベルを付ける
plt.text	ax.text	文字列を表示
plt.tick_params	ax.tick_params	軸のパラメータ設定
plt.xlim	ax.set_xlim	x軸の範囲を指定
plt.ylim	ax.set_ylim	y軸の範囲を指定
plt.axvline	ax.axvline	x=0の線をプロット
plt.axhline	ax.axhline	y=0の線をプロット
plt.grid	ax.grid	グリッド線を描く
plt.legend	ax.legend	凡例を付ける
なし	ax.invert_xaxis	x軸を反転
なし	ax.invert_yaxis	y軸を反転
なし	ax.xaxis.set_major_locator	x軸の主（大）目盛線設定
なし	ax.xaxis.set_minor_locator	x軸の副（小）目盛線設定
なし	ax.yaxis.set_major_locator	y軸の主（大）目盛線設定
なし	ax.yaxis.set_minor_locator	y軸の副（小）目盛線設定
なし	ax.xaxis.tick_top	x軸の目盛を上につける
なし	ax.xaxis.tick_bottom	x軸の目盛を下につける（デフォルト）
なし	yaxis.tick_right	y軸の目盛を右につける
なし	yaxis.tick_left	y軸の目盛を左につける（デフォルト）

これらのメソッドのうち plt.text では、書式の設定で様々なオプションが出てきたので、ここで表 3-5-3 にまとめておきます。なお、最初の行の x、y を除けば、plt.text と同じオプションを plt.title、plt.xlabel、plt.ylabel でも使うことができます。

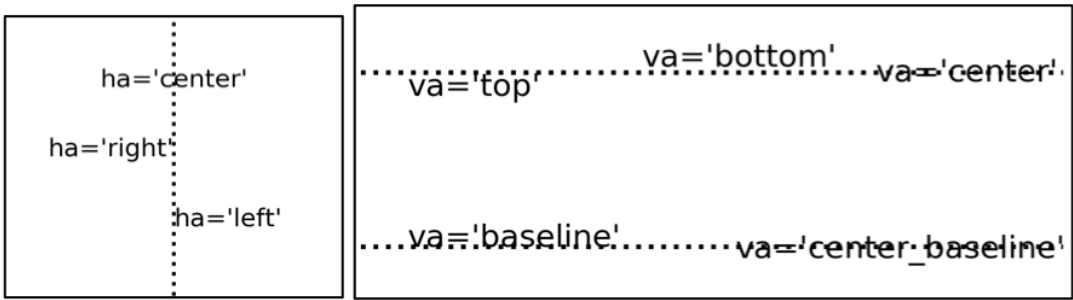
表 3-5-3 には、これまでに出てこなかったオプションとして、水平方向の位置を指定する horizontalalignment（省略形：ha）、鉛直方向の位置を指定する verticalalignment（省略形：va）があります。ha には'center'、'right'、'left'を指定可能で、va には'top'、'bottom'、'center'、'baseline'、'center_baseline'を指定可能です（表 3-5-4）。x、y で指定する中央の位置を点線で示しています。

表 3-5-3 matplotlib の pyplot.text の主要オプション一覧

オプション	説明
x, y (必須)	x軸上の座標、y軸上の値
s (必須)	表示する文字列
color	色、デフォルト：黒
backgroundcolor	背景色、デフォルト：白
fontsize	フォントサイズ、数字か次の文字列：'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'
fontweight	文字幅、0～1000か文字列
fontstyle	字体、'normal', 'italic', 'oblique'、デフォルト：'normal'
horizontalalignment or ha	水平方向の位置、'center', 'right', 'left'
verticalalignment or va	鉛直方向の位置、'top', 'bottom', 'center', 'baseline', 'center_baseline'
linespacing	文字間隔、フォントサイズへの倍率、デフォルト値：1.2
alpha	不透明度、デフォルト値：1.0
rotation	文字列を回転する角度、デフォルト値：0

使用方法：plt.text(x, y, s)、plt.text(x, y, s, オプション)

表 3-5-4 文字列の水平位置と鉛直位置を指定するオプション、点線は中央の位置



本章では、Numpy を使った関数の作成が度々出てきました。Numpy には便利な機能が多数含まれていますが、ここでは関数の作成や簡単な計算に用いることができる主要な数学関数を紹介しておきます（表 3-5-5）。

表 3-5-5 Numpy の主要な数学関数一覧

数学関数	説明
<code>np.ceil(x)</code>	x の「天井」 (x 以上の最小の整数) を返す
<code>np.floor(x)</code>	x の「床」 (x 以下の最大の整数) を返す
<code>np.sign(x)</code>	xの符号を正の場合1、負の場合-1、ゼロの場合0で返す
<code>np.round(x)</code>	x を四捨五入した値を返す
<code>np.trunc(x)</code>	x を切り捨てした値を返す
<code>np.fix(x)</code>	xを0に近い方の値に丸める
<code>np.absolute(x)</code>	x の絶対値を返す
<code>np.mod(x, y)</code>	x % yと同じ演算
<code>np.fmin(x, y)</code>	x, yを比較して小さい方を返す (NaNがあればNaNではない方)
<code>np.fmax(x, y)</code>	x, yを比較して大きい方を返す (NaNがあればNaNではない方)
<code>np.exp(x)</code>	e^x を返す
<code>np.log(x)</code>	x の自然対数: $\log_e(x)$
<code>np.log2(x)</code>	2が底のxの対数: $\log_2(x)$
<code>np.log10(x)</code>	10が底のxの対数: $\log_{10}(x)$
<code>np.power(x, y)</code>	x の y 乗 (x^y) を返す
<code>np.copysign(x, y)</code>	x の大きさ (絶対値) で y と同じ符号の浮動小数点数を返す。 例えば <code>copysign(1.0, -0.0)</code> は -1.0
<code>np.pi</code>	π を返す
<code>np.e</code>	eを返す
<code>np.radians(x)</code>	角 x を度からラジアンに変換
<code>np.deg2rad(x)</code>	角 x を度からラジアンに変換
<code>np.rad2deg(x)</code>	角 x をラジアンから度に変換
<code>np.cos(x)</code>	x ラジアンの余弦 $\cos(x)$ を返す
<code>np.sin(x)</code>	x ラジアンの正弦 $\sin(x)$ を返す
<code>np.tan(x)</code>	x ラジアンの正接 $\tan(x)$ を返す
<code>np.arccos(x)</code>	x の逆余弦をラジアンで返す
<code>np.arcsin(x)</code>	x の逆正弦をラジアンで返す
<code>np.arctan(x)</code>	x の逆正接をラジアンで返す
<code>np.cosh(x)</code>	x の双曲線余弦 $\cosh(x)$ を返す
<code>np.sinh(x)</code>	x の双曲線正弦 $\sinh(x)$ を返す
<code>np.tanh(x)</code>	x の双曲線正接 $\tanh(x)$ を返す
<code>np.acosh(x)</code>	x の逆双曲線余弦を返す
<code>np.asinh(x)</code>	x の逆双曲線正弦を返す
<code>np.atanh(x)</code>	x の逆双曲線正接を返す
<code>np.atan2(y, x)</code>	$\text{atan}(y / x)$ を、 $-\pi \sim \pi$ の間で返す 極座標平面において原点から (x, y) へのベクトルが X 軸の正の方向となす角