Graphe simple de sinus et cosinus

On montre en détail comment réaliser une représentation graphique simple des fonctions sinus et cosinus. Au départ le graphique utilisera les réglages par défaut et la figure sera ensuite améliorée pas à pas en commentant les instructions matplotlib utilisées.

Source : Matplotlib: plotting, par Nicolas Rougier, Mike Müller, Gaël Varoquaux

la licence **CC-by** Creative Commons Attribution 3.0 United States License s'applique à la traduction partielle de cette source reprise sur cette page du wiki !

Graphe avec les réglages par défaut

- Documentation à consulter :
  - Tutoriel pyplot
  - Documentation pyplot

Matplotlib considère un ensemble de paramètres pour la personnalisation des propriétés d'une représentation graphique. Vous pouvez cependant contrôler ces paramètres par défaut pour tout : taille de la figure et sa résolution en dpi, largeur de la ligne, la couleur et le style des axes, marques et propriétés de la grille, des textes (police, taille,...), ...

Voici donc la représentation par défaut des fonctions sinus et cosinus :

01-plotting_with_default_settings.py

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
"""
Matplotib : Plotting with default settings
Source :
http://scipy-lectures.github.io/intro/matplotlib/matplotlib.html
"""

import matplotlib.pyplot as plt  # directive d'importation standard de
Matplotlib
import numpy as np               # directive d'importation standard de
numpy

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)

plt.plot(X, C)
plt.plot(X, S)
```

```
plt.show()  # vue interactive de la figure
```

X est un tableau numpy de 256 valeurs comprises entre -π et +π (inclus). C est le cosinus (256 valeurs) et S est le sinus (256 valeurs).

Instantiation des réglages par défaut

- Documentation à consulter :
  - Customizing matplotlib

Dans le programme ci-dessous, nous avons instancié (et commenté) tous les paramètres du graphique qui influencent son apparence. Les paramètres ont été définis explicitement à leurs valeurs par défaut, mais à partir de là, vous pourrez interagir avec ces valeurs afin d'explorer leur effet (voir propriétés de la ligne et styles de ligne ci-dessous).

02-instantiating_defaults.py

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
"""
Matplotib : instantiating defaults
Source :
http://scipy-lectures.github.io/intro/matplotlib/matplotlib.html
"""
import matplotlib.pyplot as plt  # directive d'importation standard de
Matplotlib
import numpy as np               # directive d'importation standard de
numpy

# Create a figure of size 8x6 inches, 80 dots per inch
plt.figure(figsize=(8, 6), dpi=80)

# Create a new subplot from a grid of 1x1
plt.subplot(1, 1, 1)

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)

# Plot cosine with a blue continuous line of width 1 (pixels)
plt.plot(X, C, color="blue", linewidth=1.0, linestyle="-")

# Plot sine with a green continuous line of width 1 (pixels)
plt.plot(X, S, color="green", linewidth=1.0, linestyle="-")

# Set x limits
plt.xlim(-4.0, 4.0)
```

```
# Set x ticks
plt.xticks(np.linspace(-4, 4, 9, endpoint=True))

# Set y limits
plt.ylim(-1.0, 1.0)

# Set y ticks
plt.yticks(np.linspace(-1, 1, 5, endpoint=True))

# Save figure using 72 dots per inch
# savefig("exercice_2.png", dpi=72)

# Show result on screen
plt.show()
```

Changer la couleur et l'épaisseur des lignes

- Documentation à consulter :
  - Controlling line properties
  - Line API

Première étape, nous voulons avoir le cosinus en bleu et le sinus en rouge, et une ligne un peu plus épaisse pour chacun d'eux. Nous allons également changer légèrement la taille du graphique pour le rendre plus étendu horizontalement. Voici les lignes modifiées :

03-changing_colors_and_line_widths.py

```
...
plt.figure(figsize=(10, 6), dpi=80)
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-")
...
```

Fixer les limites

- Documentation à consulter :
  - xlim() command
  - ylim() command

Les limites par défaut de la figure sont un peu juste verticalement et on souhaite un peu d'espacement pour plus de clareté. On les définit en fonction des extrema en utilisant une constante multiplicative légèrement supérieure à l'unité :

04-setting_limits.py

```
...
plt.xlim(X.min() * 1.1, X.max() * 1.1)
plt.ylim(C.min() * 1.1, C.max() * 1.1)
...
```

Fixer les graduations

- Documentation à consulter :
  - xticks() command
  - yticks() command
  - Tick container
  - Tick locating and formatting

Les graduations ne sont pas idéales car elles ne montrent pas les valeurs intéressantes (+/-π,+/-π/2) pour le sinus et le cosinus. On peut modifier les graduations de manière à ne voir que ces valeurs intéressantes :

05-setting_ticks.py

```
...
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
plt.yticks([-1, 0, +1])
...
```

Fixer le texte des graduations

- Documentation à consulter :
  - Working with text
  - xticks() command
  - yticks() command
  - set_xticklabels()
  - set_yticklabels()

Les graduations sont adéquates, mais leur écriture (3.142,...) n'est pas très explicite. on préférerait lire π au lieu de 3.142. La solution consiste à fournir en plus des graduations une seconde liste reprenant les textes correspondants à écrire. Notez qu'on utilise LaTeX et sa syntaxe pour une représentation de qualité :

06-setting_tick_labels.py

```
...
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
           [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
```

```
plt.yticks([-1, 0, +1],
           [r'$-1$', r'$0$', r'$+1$'])
...
```

Les chaînes de caractères incluant du code LaTeX pour les formules mathématiques doivent être préfixées par la lettre “r” afin que leur interprétation des caractères “backslach” soit utilisée comme code LaTeX et pas comme des codes d'échappement de chaînes de caractères en Python. *Cf.* la documentation.

Déplacer les lignes de délimitation (spines)

- Documentation à consulter :
  - Spines
  - Axis container
  - Transformations tutorial

Les lignes de délimitations verticales et horizontales (spines) sont les lignes qui connectent les graduations sur les axes et délimitent les limites de la surface visualisant les données du graphique. Ces lignes peuvent être placées arbitrairement et jusqu'à présent elles délimitaient une “boîte” aux limites des axes. Nous voudrions les déplacer et juste conserver deux lignes en plein milieu du graphique. Puisqu'il y avait quatre lignes, on va éliminer la visualisation des lignes supérieures et de droite en fixant la couleur à une valeur “nulle”, et on va placer les lignes “du bas” et “de gauche” aux coordonnées zéro :

07-moving_spines.py

```
...
ax = plt.gca()  # gca stands for 'get current axis'
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
...
```

Ajouter une légende

- Documentation à consulter :
  - Legend guide
  - legend() command
  - Legend API

Ajoutons une légende dans le coin supérieur gauche. Cela nécessite juste d'ajouter des couples

“clé/valeur” à la commande “plot”, pour lesquelles la clé est “label”, et la valeur corresponde la fonction (sinus ou cosinus) qui sera affichée dans le cadre de la légende :

08-adding_a_legend.py

```
...
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-",
label="cosinus")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-",
label="sinus")

plt.legend(loc='upper left')
...
```

Annoter certains points

- Documentation à consulter :
  - Annotating axis
  - annotate() command

Annotons quelques points particuliers en utilisant la commande “annotate”. On choisit l'abscisse 2π/3 et on veut annoter à la fois le cosinus et le sinus. On va dans les deux cas dessiner une ligne en pointillé, marquer le point par un symbole (rond par défaut) de taille 50 en unité de points typographiques, et en utilisant la commande “annotate” pour afficher un texte accompagné d'une flèche.

09-annotate_some_points.py

```
...
t = 2 * np.pi / 3
plt.plot([t, t], [0, np.cos(t)], color='blue', linewidth=2.5,
linestyle="--")
plt.scatter([t, ], [np.cos(t), ], 50, color='blue')
plt.annotate(r'$cos(\frac{2\pi}{3})=-\frac{1}{2}$',
            xy=(t, np.cos(t)), xycoords='data',
            xytext=(-90, -50), textcoords='offset points', fontsize=16,
            arrowprops=dict(arrowstyle="->",
connectionstyle="arc3,rad=.2"))

plt.plot([t, t],[0, np.sin(t)], color='red', linewidth=2.5,
linestyle="--")
plt.scatter([t, ],[np.sin(t), ], 50, color='red')
plt.annotate(r'$sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
            xy=(t, np.sin(t)), xycoords='data',
            xytext=(+10, +30), textcoords='offset points', fontsize=16,
            arrowprops=dict(arrowstyle="->",
```

```
connectionstyle="arc3,rad=.2"))
...
```

Le diable est dans les détails*

*Attribué à Nietzsche

- Documentation à consulter :
  - Artists
  - BBox

Les textes des graduations sont à présent difficilement lisibles à cause des lignes bleue et rouge. On peut les écrire avec une taille supérieure et on peut aussi ajuster leurs propriétés de telle sorte qu'ils soient rendus que un fond blanc semi-transparent (transparence définie par alpha). Cela nous permet de voir correctement à la fois les données et les textes des graduations.

title : 10-devil_is_in_the_details.py

```
...
for label in ax.get_xticklabels() + ax.get_yticklabels():
    label.set_fontsize(16)
    label.set_bbox(dict(facecolor='white', edgecolor='None',
alpha=0.65))
...
```

From:
https://dvillers.umons.ac.be/wiki/ - **Didier Villers, UMONS - wiki**

Permanent link:
https://dvillers.umons.ac.be/wiki/teaching:progappchim:plot_sinus_cosinus?rev=1487942044

Last update: **2017/02/24 14:14**