Matplotlib : Fonctionnalités avancées

Maîtriser Matplotlib : Un guide complet pour la visualisation de données · https://moncoachdata.com/tutos/matplotlib-fonctionnalites-avancees/

Ce tuto présente les **fonctionnalités avancées de Matplotlib** : annotations et texte, remplissage entre des courbes, séries temporelles, graphiques 3D, interactivité et animations. Tous les exemples supposent ces imports :

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

Python

Ajouter des annotations et du texte

Les annotations et le texte améliorent la communication des idées et mettent en évidence les points clés de tes graphiques. C'est très utile en présentation. Matplotlib propose plusieurs méthodes :

- `plt.text(x_pos, y_pos, texte, fontsize=taille)` ajoute un texte personnalisé à des coordonnées précises ;
- `plt.annotate(texte, xy=position_flèche, xytext=position_texte)` ajoute un texte avec une flèche ou un marqueur qui pointe vers un endroit précis du graphique ;
- **Étiqueter des points de données :** dans un nuage de points, tu peux étiqueter chaque point avec `text()` ou `annotate()`.

Dans cet exemple, on met en évidence le pic de ventes de juillet :

```python
# Chiffre d'affaires mensuel de deux magasins
months = ['Jan', 'Fév', 'Mar', 'Avr', 'Mai', 'Juin', 'Juil', 'Août', 'Sept', 'Oct', 'Nov', 'Déc
```

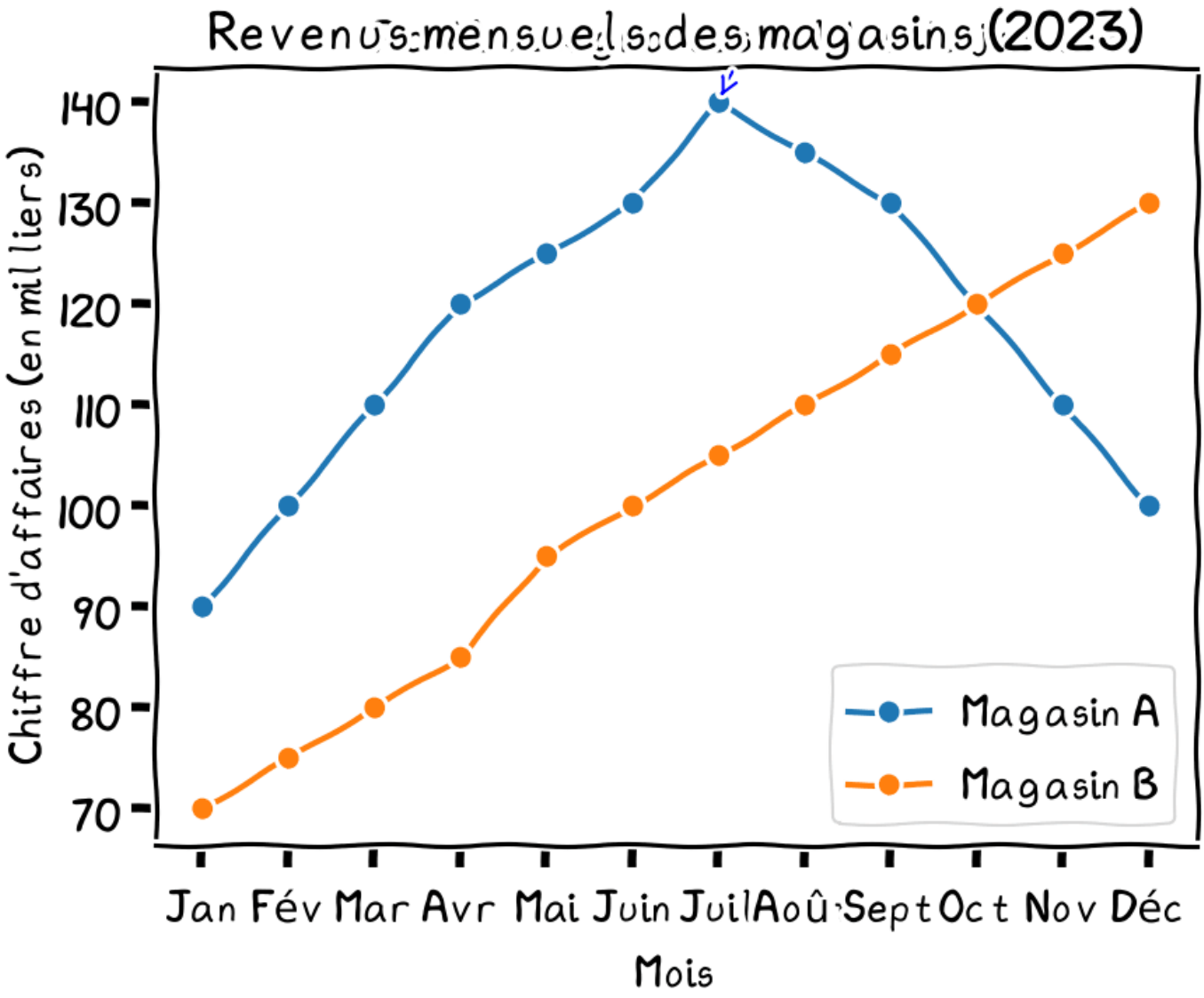
```
']
store_locations = ['Magasin A', 'Magasin B']
revenue_data = np.array([[90, 100, 110, 120, 125, 130, 140, 135, 130, 120, 110, 100],
                         [70, 75, 80, 85, 95, 100, 105, 110, 115, 120, 125, 130]])

# Style « bande dessinée » appliqué uniquement dans ce bloc with
with plt.xkcd():
    plt.plot(months, revenue_data[0], marker='o', label=store_locations[0])
    plt.plot(months, revenue_data[1], marker='o', label=store_locations[1])

    # Mettre en évidence le pic de juillet
    plt.annotate("Pic dû aux grosses soldes de juillet", xy=('Juil', 140), xytext=('Mar', 145),
                 arrowprops=dict(arrowstyle='->', color='blue'))

    plt.title('Revenus mensuels des magasins (2023)')
    plt.ylim(65, 155)
    plt.xlabel('Mois')
    plt.ylabel("Chiffre d'affaires (en milliers)")
    plt.grid(True)
    plt.legend()
    plt.show()
```

Python



Pour étiqueter chaque point, parcours les noms et appelle `annotate()` pour chacun. Tu n'as alors pas besoin de `arrowprops`, mais tu dois utiliser `textcoords='offset points'` : la position de l'étiquette (`xytext`) est alors un décalage en points par rapport au point de données.

```
# Chiffre d'affaires de quatre magasins sur un mois donné
store_names = ['Magasin A', 'Magasin B', 'Magasin C', 'Magasin D']
revenue_data = np.array([120, 150, 155, 130])    # en milliers
colors = ['red', 'green', 'blue', 'orange']

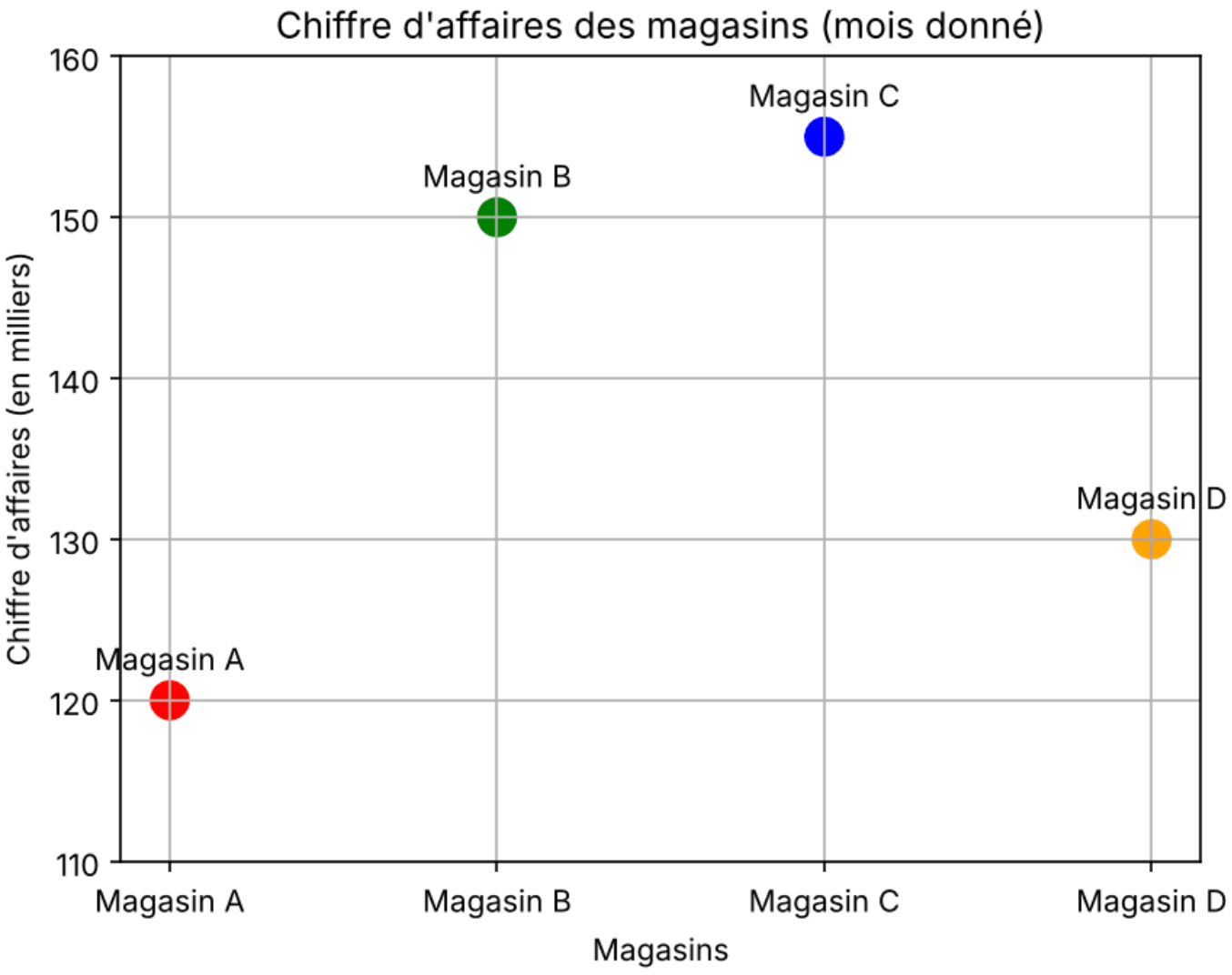
plt.scatter(store_names, revenue_data, s=150, c=colors, marker='o', label='Chiffre d\'affaires'
```

```python
)

# Étiqueter chaque magasin, 10 points au-dessus du point de données
for i, store in enumerate(store_names):
    plt.annotate(store, (store, revenue_data[i]), textcoords='offset points',
                 xytext=(0, 10), ha='center')
    # ou : plt.text(store, revenue_data[i] + 1, store)

plt.title("Chiffre d'affaires des magasins (mois donné)")
plt.xlabel('Magasins')
plt.ylabel("Chiffre d'affaires (en milliers)")
plt.ylim(110, 160)
plt.grid(True)
plt.show()
```

Python



Remplir la zone entre des courbes

Il est parfois utile de mettre en évidence les régions entre deux courbes, pour aider à voir où l'une dépasse l'autre. C'est le rôle de `fill_between`. Le paramètre `alpha` contrôle l'intensité de la couleur de remplissage.

- Remplir entre l'axe x et la courbe : `plt.fill_between(x, y)` ;
- Remplir entre deux courbes : `plt.fill_between(x, y1, y2)` ;
- Remplir entre deux courbes sous une condition : `plt.fill_between(x, y1, y2, where=condition)` ;
- Remplir plusieurs régions avec des conditions et des couleurs différentes, en appelant `fill_between` plusieurs fois.

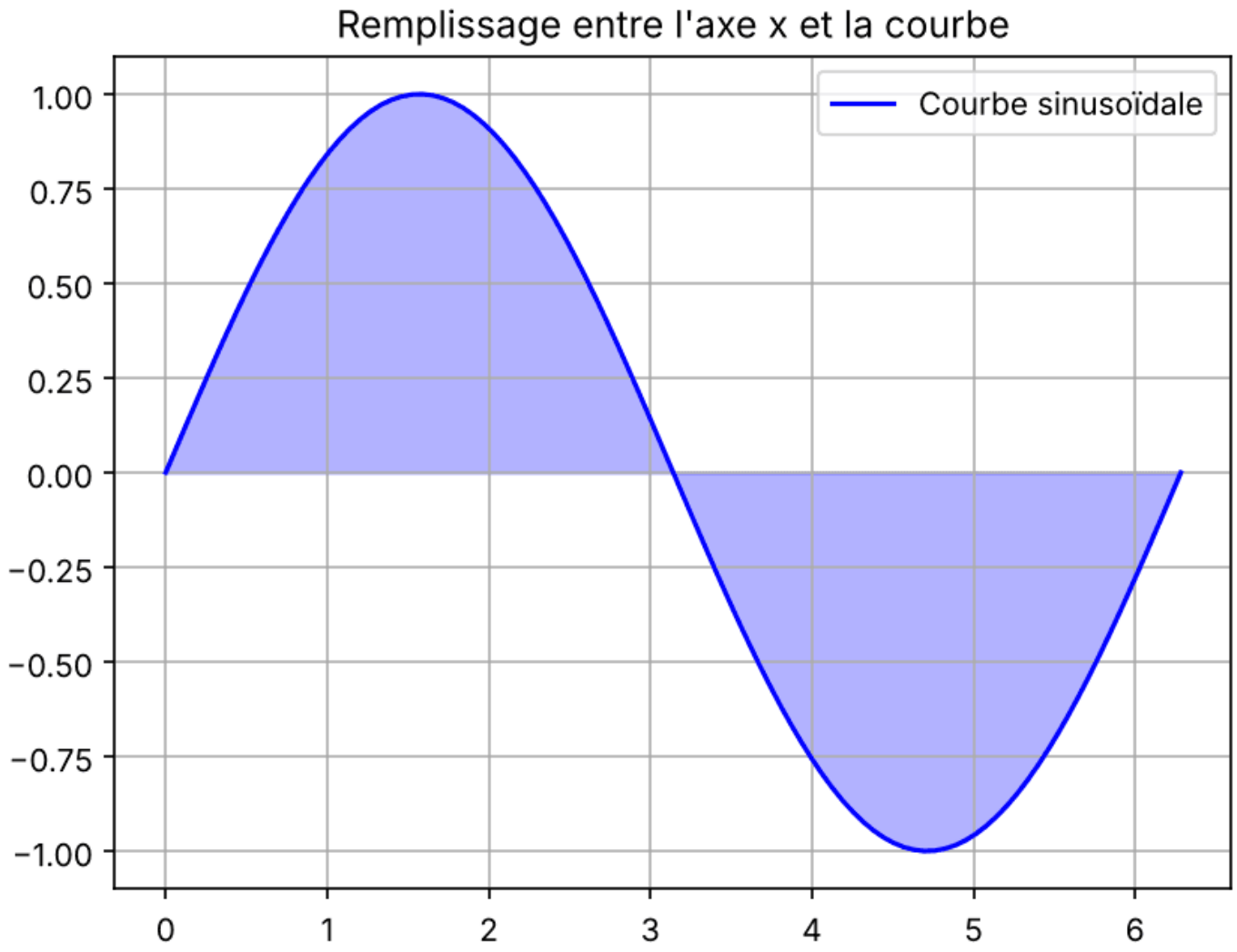
```python
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

plt.plot(x, y, label='Courbe sinusoïdale', color='blue')

# Remplir entre l'axe x et la courbe
plt.fill_between(x, 0, y, alpha=0.3, color='blue')

plt.title("Remplissage entre l'axe x et la courbe")
plt.grid(True)
plt.legend()
plt.show()
```

Python



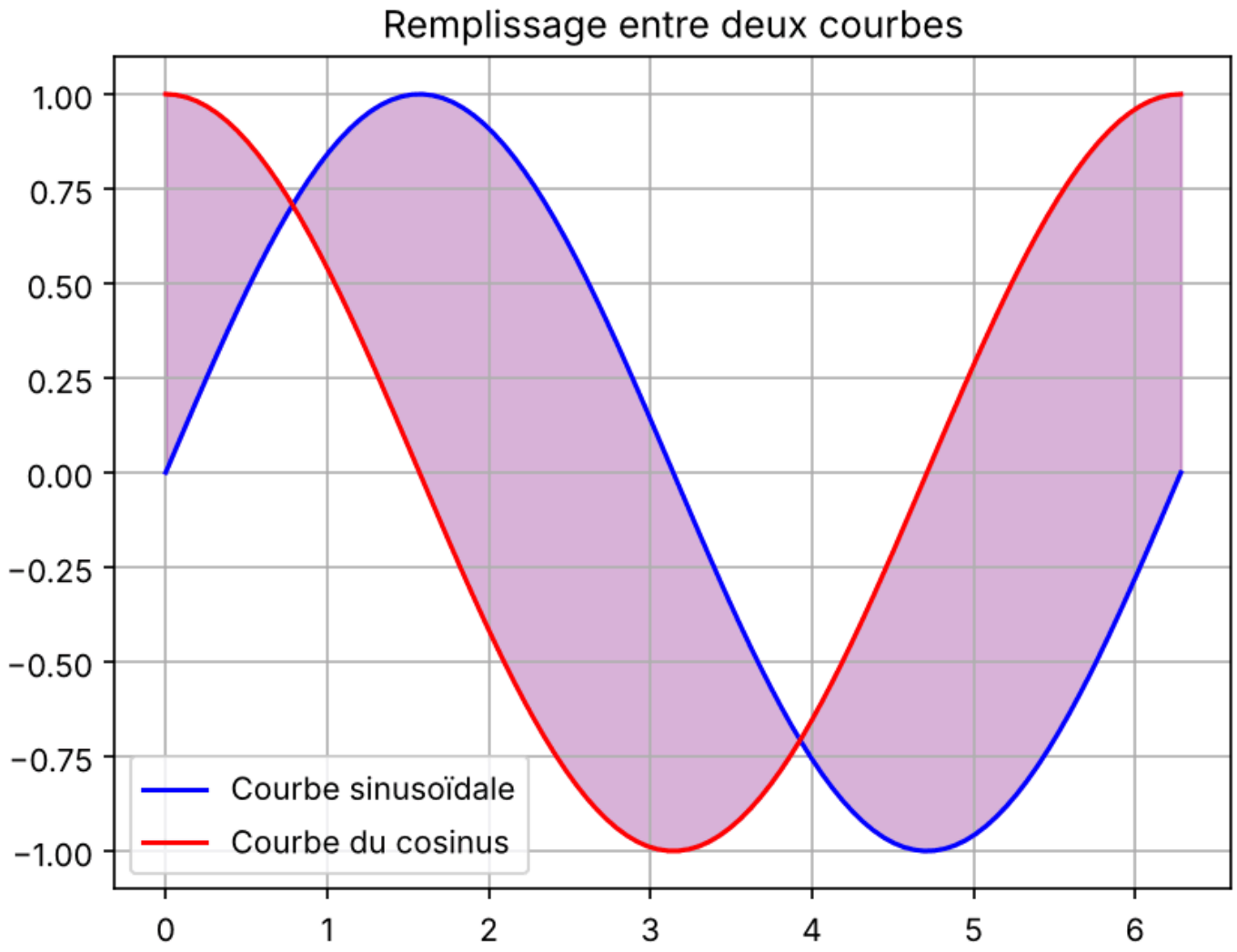

```python
y1 = np.sin(x)
y2 = np.cos(x)

plt.plot(x, y1, label='Courbe sinusoïdale', color='blue')
plt.plot(x, y2, label='Courbe du cosinus', color='red')

# Remplir la zone entre les deux courbes
plt.fill_between(x, y1, y2, alpha=0.3, color='purple')

plt.title('Remplissage entre deux courbes')
plt.grid(True)
plt.legend()
plt.show()
```

Python



Avec le paramètre `where`, tu remplis uniquement les zones où une condition est vraie, ici là où le sinus dépasse le cosinus. Tu peux aussi y ajouter une annotation :

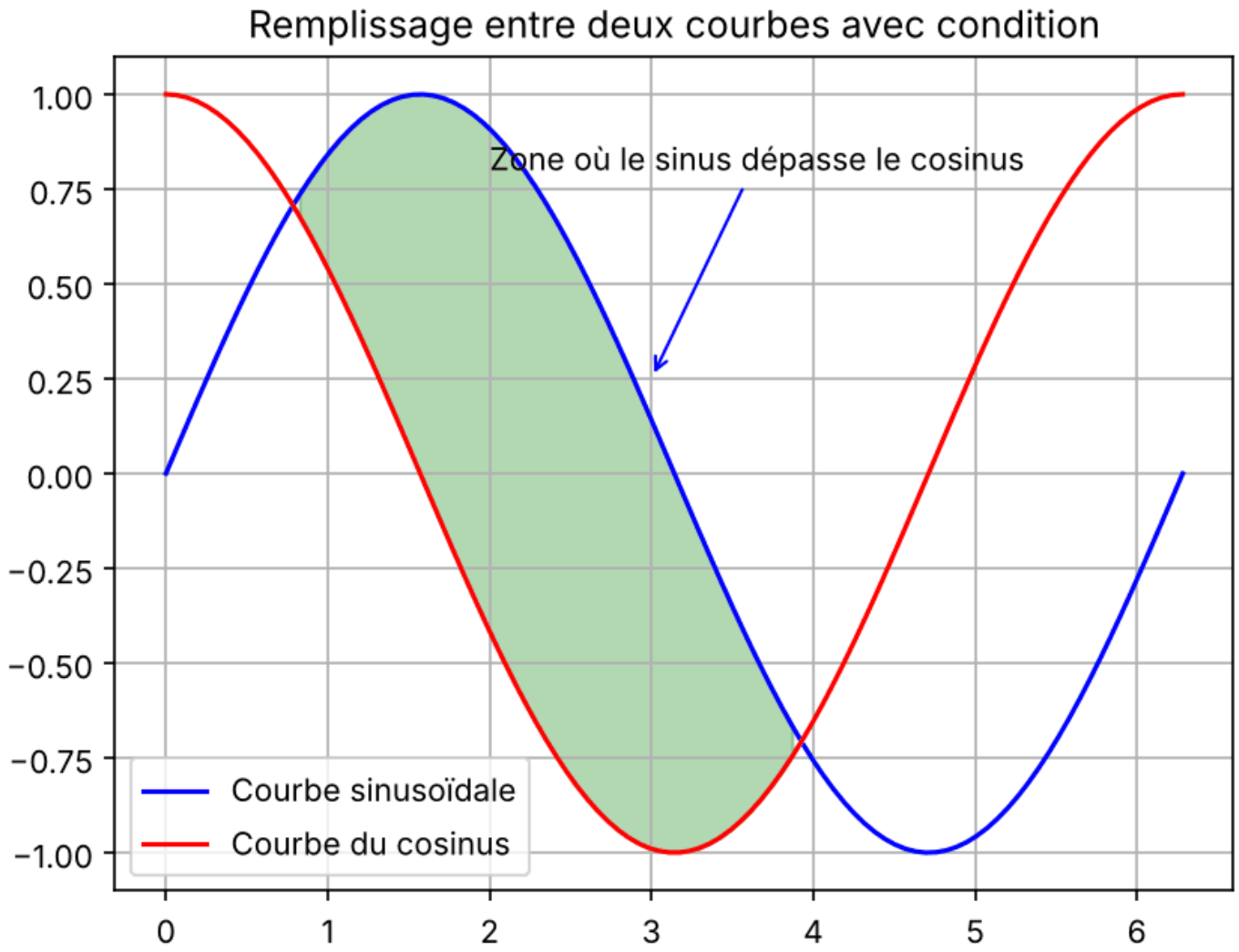
```python
plt.plot(x, y1, label='Courbe sinusoïdale', color='blue')
plt.plot(x, y2, label='Courbe du cosinus', color='red')

# Remplir la région entre les deux courbes là où y1 > y2
plt.fill_between(x, y1, y2, where=(y1 > y2), alpha=0.3, color='green')

# Annoter la zone remplie
plt.annotate('Zone où le sinus dépasse le cosinus', xy=(3, 0.25), xytext=(2, 0.80),
             arrowprops=dict(arrowstyle='->', color='blue'))

plt.title('Remplissage entre deux courbes avec condition')
plt.grid(True)
plt.legend()
plt.show()
```

Python



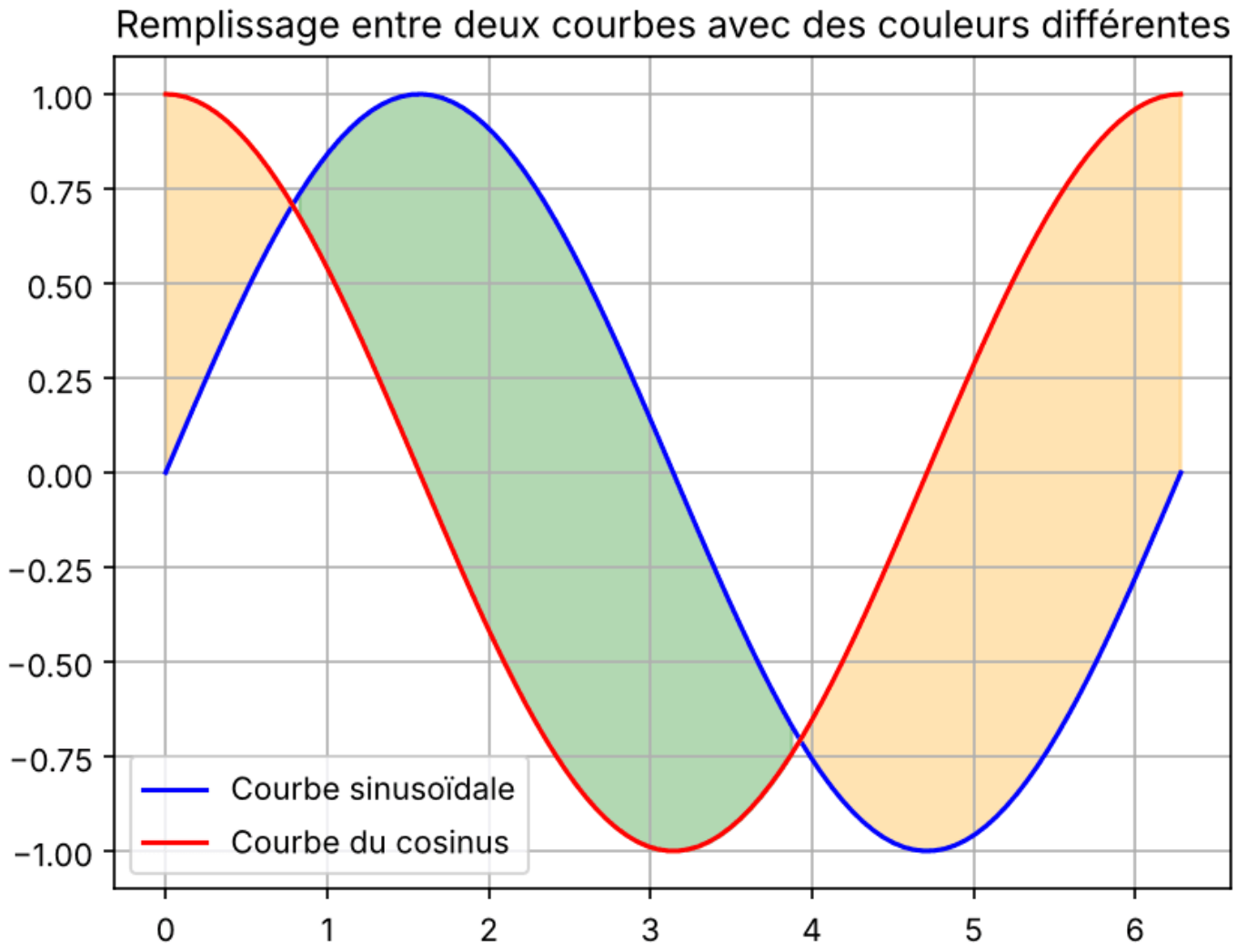
Enfin, tu peux remplir plusieurs régions avec des couleurs différentes :

```python
plt.plot(x, y1, label='Courbe sinusoïdale', color='blue')
plt.plot(x, y2, label='Courbe du cosinus', color='red')

# Deux régions, deux couleurs
plt.fill_between(x, y1, y2, where=(y1 > y2), alpha=0.3, color='green')
plt.fill_between(x, y1, y2, where=(y1 <= y2), alpha=0.3, color='orange')

plt.title('Remplissage entre deux courbes avec des couleurs différentes')
plt.grid(True)
plt.legend()
plt.show()
```

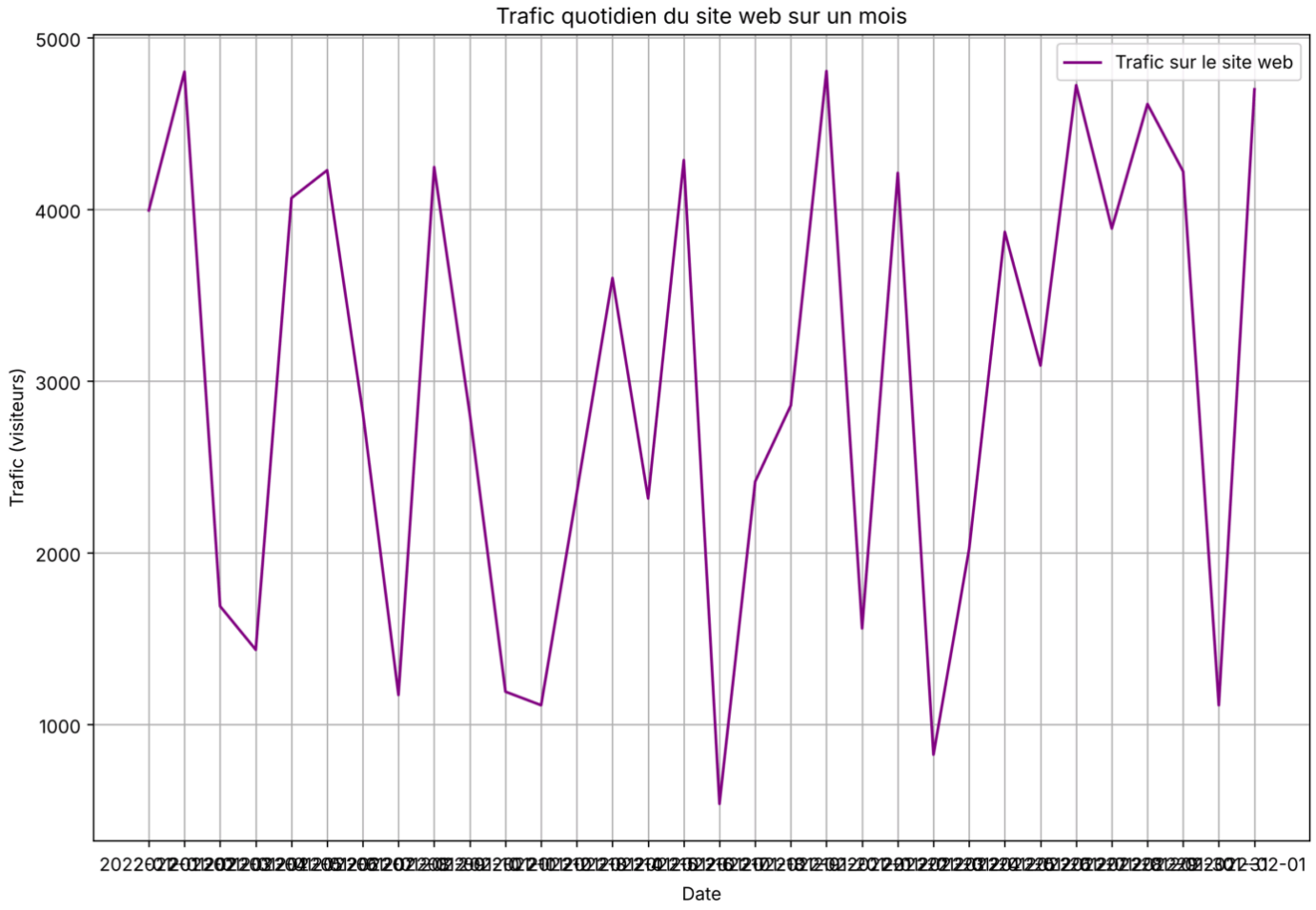
Python



Tracer des séries temporelles

Les séries temporelles sont très courantes en finance, en climatologie ou en analyse commerciale, et souvent volumineuses : on ne tire pas le meilleur des données avec quelques agrégations. Matplotlib permet de les interpréter facilement.

Imagine que tu veuilles tracer le trafic d'un site web sur un mois. Avec un tracé linéaire brut, l'axe x est illisible : les dates se chevauchent.



Le sous-module `matplotlib.dates` fournit des outils pour formater et personnaliser les dates et heures de l'axe x :

- `DateFormatter` personnalise le format des étiquettes de date (ordre de l'année, du mois et du jour, séparateurs) ;
- `DayLocator`, `MonthLocator`, `YearLocator`... fixent la fréquence des repères sur l'axe ;
- `AutoDateLocator` et `AutoDateFormatter` déterminent automatiquement l'emplacement et le format selon la plage de données ;
- `plt.gcf()` (*get current figure*) et `plt.gca()` (*get current axes*) donnent accès à la figure et aux axes courants. On crée un format de date, puis on le passe à `plt.gca().xaxis.set_major_formatter(...)`.

Note : la fonction `plot_date` est dépréciée. Avec des valeurs de type `datetime` ou `Timestamp`, `plt.plot()` gère déjà l'axe des dates.

Voici le même exemple avec seulement quelques lignes de personnalisation :

```python
import matplotlib.dates as mdates

rng = np.random.default_rng(10)

# Échantillon de trafic quotidien sur un mois
date_rng = pd.date_range(start='2022-01-01', end='2022-02-01')
traffic_data = rng.integers(500, 5000, len(date_rng))

# DataFrame
traffic_df = pd.DataFrame({'Date': date_rng, 'Trafic (visiteurs)': traffic_data})

plt.figure(figsize=(12, 8))
plt.plot(traffic_df['Date'], traffic_df['Trafic (visiteurs)'],
         label='Trafic sur le site web', color='purple')

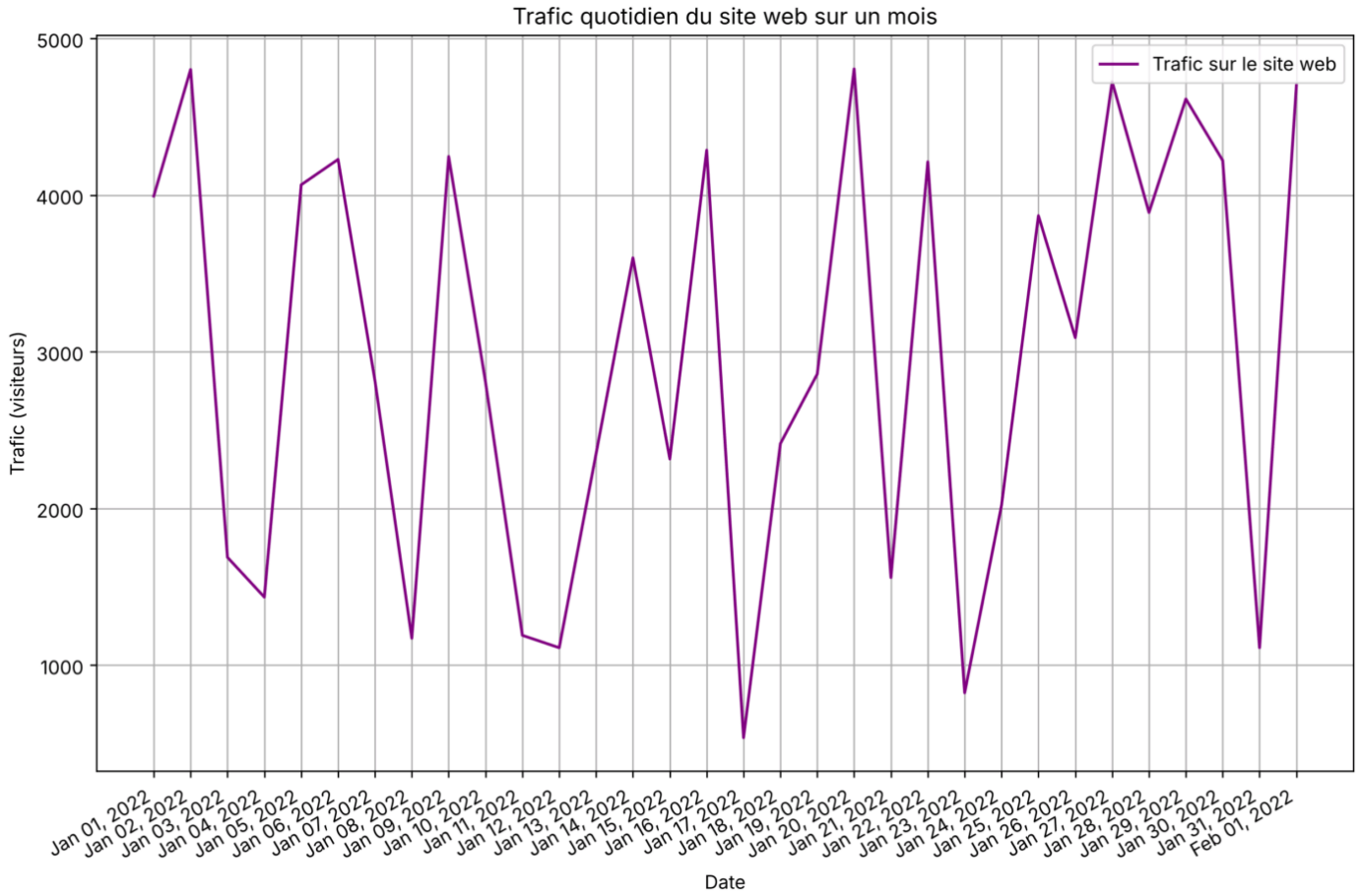
# Afficher une étiquette par jour
plt.xticks(traffic_df['Date'])

# Format de date : « jan 01, 2022 »
date_format = mdates.DateFormatter('%b %d, %Y')
plt.gca().xaxis.set_major_formatter(date_format)

# Rotation et alignement automatique des dates
plt.gcf().autofmt_xdate()

plt.xlabel('Date')
plt.ylabel('Trafic (visiteurs)')
plt.title('Trafic quotidien du site web sur un mois')
plt.grid(True)
plt.legend(loc='upper right')
plt.show()
```

Python



Grâce à `matplotlib.dates`, l'axe x est personnalisé et le graphique devient bien plus lisible.

Créer des graphiques 3D

Avec trois variables ou plus, un tracé 3D aide à mieux comprendre des données complexes : modélisation de terrain, caractéristiques géographiques, imagerie médicale 3D...

Pour créer un tracé 3D, tu crées une figure avec `plt.figure()`, puis tu ajoutes un sous-graphique 3D avec `fig.add_subplot(111, projection='3d')` (`111` signifie une ligne, une colonne, et le sous-graphique à la première et unique position). Depuis Matplotlib 3.2, il n'est plus nécessaire d'importer `Axes3D` explicitement.



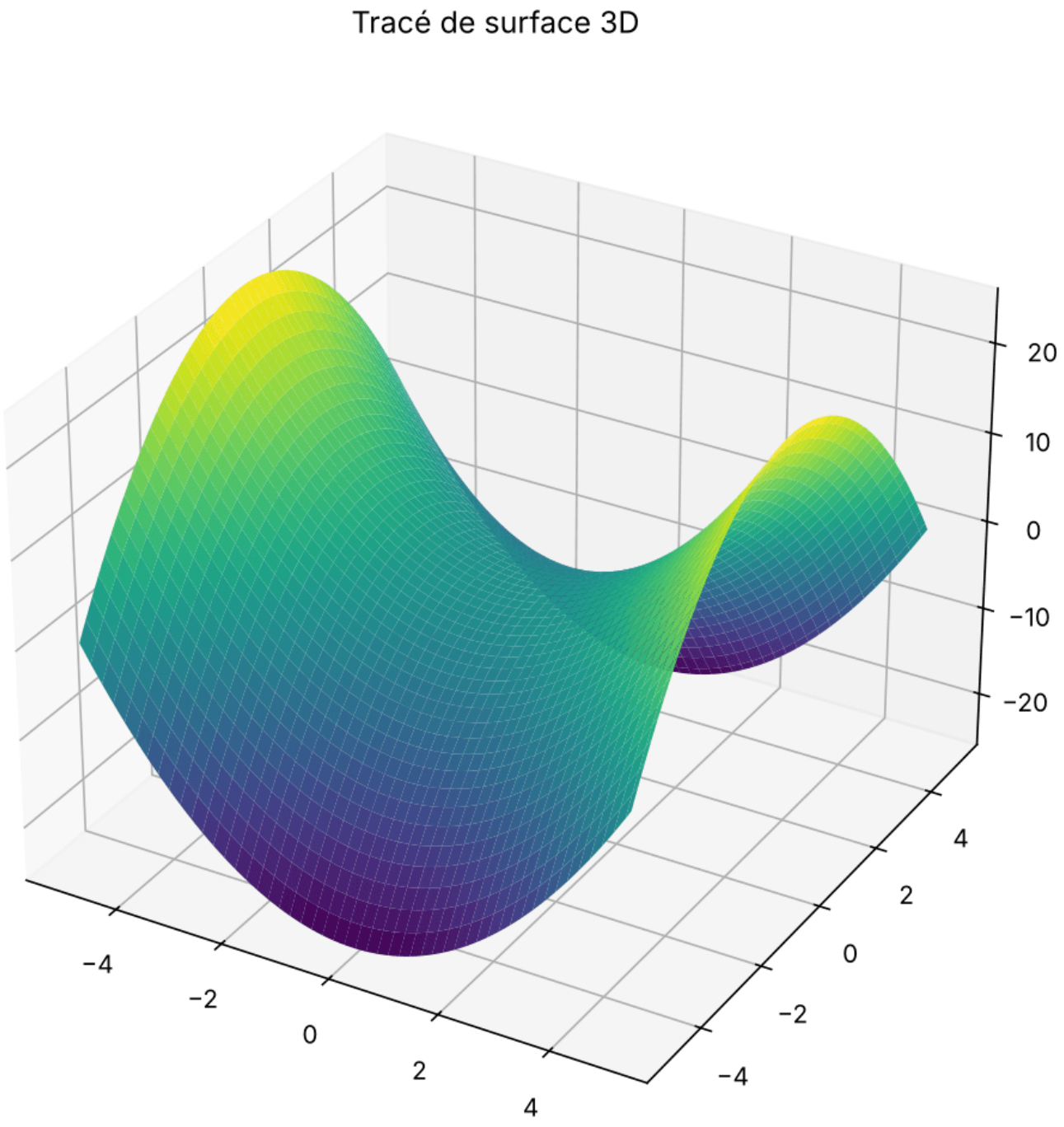
```python
# Grille de valeurs X et Y
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)

# Fonction à tracer : une forme de selle
Z = X**2 - Y**2

# Tracé de surface 3D
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis')

ax.set_title('Tracé de surface 3D')
plt.show()
```

Python



Tu peux aussi créer des lignes 3D, des nuages de points 3D, etc.

Ajouter de l'interactivité et des animations

Pouvoir tourner, zoomer et survoler le tracé 3D ci-dessus serait bien pratique. Dans un notebook Jupyter, c'est possible avec le backend interactif : installe `ipympl` puis utilise la commande magique `%matplotlib widget` (l'ancien `%matplotlib notebook` ne fonctionne plus dans les versions récentes de Jupyter). Pour revenir à un tracé statique, utilise `%matplotlib inline`.

Avec un tracé interactif, tu peux aussi créer des animations grâce à `FuncAnimation`, du module `matplotlib.animation`. Elle prend la figure, la fonction à appeler de manière répétée et l'intervalle de temps entre deux appels (en millisecondes). Le code ci-dessous crée une onde sinusoïdale animée : la fonction `animate` est appelée à intervalle régulier, et le tracé est redessiné à chaque appel.

```python
from matplotlib.animation import FuncAnimation

%matplotlib widget

fig, ax = plt.subplots()
x = np.linspace(0, 2 * np.pi, 1000)

def animate(frame):
    y = np.sin(x + 0.1 * frame)      # on fait varier la phase pour animer l'onde
    ax.clear()
    ax.plot(x, y)
    ax.set_ylim(-1.2, 1.2)
    ax.set_title('Onde sinusoïdale animée')

ani = FuncAnimation(fig, animate, interval=100)
plt.show()
```

Python

Dans la fonction `animate`, tu peux aussi appeler `pd.read_csv('live_data.csv')` sur un fichier mis à jour régulièrement : tu obtiens alors un tracé en temps réel, qui évolue avec les données.

Conclusion

Dans cette série sur Matplotlib, tu as vu les principes du tracé, la personnalisation, les subplots multiples, le remplissage entre courbes et les fonctionnalités avancées : tracés 3D, séries temporelles, annotations et texte. Matplotlib fournit les bases pour créer des visuels convaincants qui racontent l'histoire de tes données : n'hésite pas à jouer avec.

Pour des graphiques statistiques plus rapides à écrire, découvre ensuite la série Seaborn.
Documentation : guide de l'utilisateur Matplotlib.

Pour aller plus loin, consulte aussi notre guide des 9 bibliothèques Python pour la data science.


MATPLOTLIB