TD 10 : Tracé de Courbes et Modules

Exercice 1 (Module math)

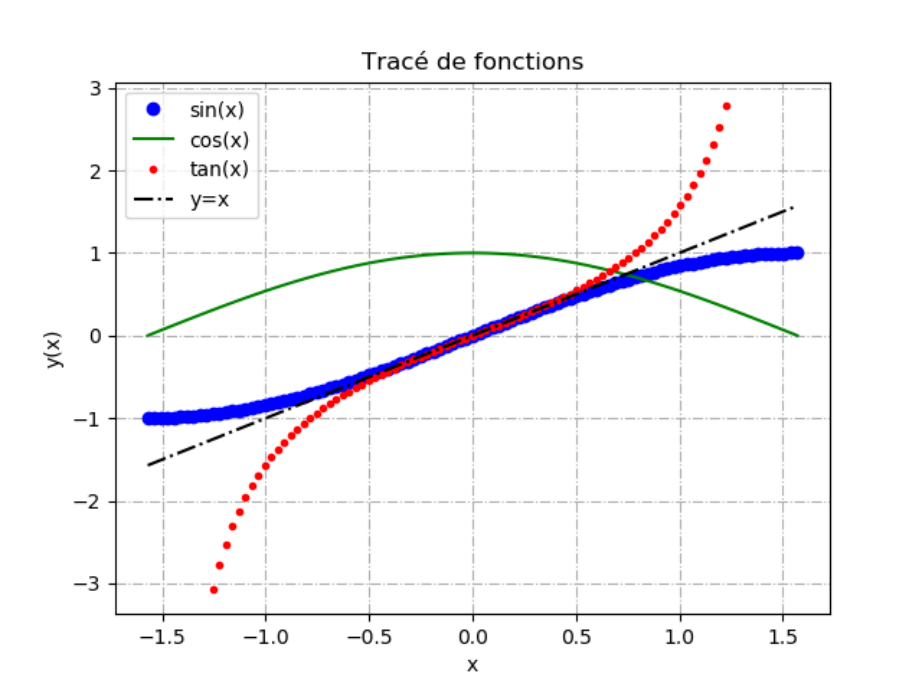
Le module `math` contient essentiellement les fonctions mathématiques usuelles pour le type `float`. En tapant `help(math)` dans l'interpréteur python, répondre aux questions suivantes :

1. Vérifiez ce que renvoie la fonction `help` appliquée aux fonctions `floor` et `ceil`.
2. Quelle fonction du module `math` convertit directement un angle exprimé en degré, en radian ? Quelle est la fonction réciproque ? Vérifiez que vous savez les reprogrammer.
3. Dans la section `DATA` de la documentation du module `math`, quelles sont les deux données dont il est bon de connaître l'existence ? Sinon, comment les calcule-t-on ?

Rappel 1 : il n'existe pas qu'une seule manière d'associer un entier à un réel ! Le **constructeur** standard `int` ne doit pas être utilisé pour convertir un flottant en entier : on utilise les **fonctions** `floor`, `ceil` ou `trunc` du module `math` selon l'opération que l'on souhaite réaliser, ou la fonction standard `round`.
Rappel 2 : le module `cmath` est l'équivalent du module `math` pour les nombres complexes.

Exercice 2 (Module matplotlib.pyplot : tracé de courbes)

On importe traditionnellement ce module par `import matplotlib.pyplot as plt`. Tous les éléments de ce module ont donc pour préfixe `plt`.
Tracer les courbes représentatives des fonctions sin, cos, tan sur $[-\pi/2, \pi/2]$ si elles sont définies (avec `N=100` points régulièrement répartis) en utilisant le type `list` de Python.



1. À l'aide du mémento fourni, reproduire la figure précédente.
2. Conservez un fichier `template_matplotlib.py` contenant la partie du code qui vous a permis de tracer la figure.

Exercice 3 (Module numpy)

`numpy` signifie `Numerical Python`. C'est un module fondamental pour le calcul scientifique. Alors qu'un tableau python (implémenté par le type `list`) est de taille variable et peut contenir des données de nature différentes, `numpy` propose le type `ndarray` (*numerical data array*) pour les tableaux homogènes de taille fixe, plus efficaces à l'usage. Ces tableaux sont créés à l'aide de fonctions du module `numpy` : `array`, `zeros`, `empty`, `linspace`, `arange`, `full`, etc. Le type des éléments peut être déclaré à la création par l'argument optionnel `dtype` (*data type*). Par défaut les tableaux contiennent des flottants (`numpy.float64` ou `float` ou `numpy.double`).

```
import numpy as np
import matplotlib.pyplot as plt
pi = np.pi
x = np.linspace(-pi/2, pi/2, 30)
y = np.sin(x)
plt.title('Tracé de la fonction sinus avec Numpy')
plt.plot(x,y,'bo',label='np.sin(x)')
plt.legend()
plt.show()
```

1. Exécuter ce code. Afficher `x`. À quoi sert la fonction `np.linspace` ?
2. Quel est le type de `x` ? Comment avoir un tableau python habituel ?
3. Que prend comme argument la fonction `np.sin` ? Quelle différence avec la fonction `math.sin` ? Comment traduire les termes *element-wise*, *ufunc*, *standard broadcasting* dans ce contexte ?

Exercice 4 (Représentation de fonctions)

1. Tracer sur le même graphe et sur l'intervalle $[-1, 1]$, avec des couleurs différentes, les courbes d'équations $y = x^n$ pour $n \in \{2; 3; 4; 5; 10\}$.

   Expliquer pourquoi les courbes sont plus ou moins plates autour de zéro, vérifier la parité des fonctions.
2. Tracer simultanément sur l'intervalle $[-1, 3]$ les courbes d'équations $y = 1 + 2x$ et $y = 1 + 2x + (x - 1)^n$ pour $n \in \{2; 3; 4; 5\}$.

   Donner la valeur en $x = 0$ de ces fonctions et vérifiez sur les courbes.

   Montrer que toutes les courbes passent par un même point pour $n > 1$, donner ses coordonnées.

   Montrer que toutes les courbes ont la même tangente en $x = 1$.

Mémento Matplotlib

`import matplotlib.pyplot as plt`	Importe le module, préfixe `plt.`
`plt.plot(X, Y, 'ro', label='')`	`X` et `Y` sont des tableaux de même taille contenant la liste des coordonnées x et y. `'ro'` $\rightarrow$ red, symbole $\circ$ `'b+'` $\rightarrow$ blue, symbole $+$
`plt.legend()`	affiche les différents `label` définis via `plt.plot`
`plt.arrow(X,Y,Vx,Vy,head_width=0.2,` `  head_length=0.5,color="red")`	Trace le vecteur de coordonnées $(v_x,\ v_y)$ à partir du point de coordonnées $(x,\ y)$ avec les mêmes conventions que `plt.plot`.
`plt.axis('equal')` ou si bug, `plt.gca().set_aspect('equal')`	Permet d'avoir la même représentation de l'unité en x et en y
`plt.xlim(xmin, xmax)` `plt.ylim(ymin, ymax)`	Fixe les limites souhaitées en x et en y
`plt.text(x,y,"texte", fontsize=12)`	Ecrit `texte` au point de coordonnées (`x`, `y`)
`plt.xlabel("x (unité)")` `plt.ylabel("y (unité)")`	Pour écrire les grandeurs sur les axes
`plt.grid(linestyle='-.')`	Trace la grille en arrière plan
`plt.savefig('image.png')`	Sauvegarde le graphique dans un fichier png
`plt.show()`	Pour afficher la figure. A mettre à la fin du programme.
`plt.figure()`	Pour avoir plusieurs fenêtres de graphique, à mettre au début de chaque bloc correspondant à une fenêtre

Mémento Numpy

Les fonctions fournies par le module sont **vectorielles** !

`import numpy as np`	Importe le module, préfixe `np.`
`np.linspace(start, stop, n)`	renvoie un tableau de `n` valeurs régulièrement réparties sur l'intervalle [`start`,`stop`]
`np.arange(start, stop, step)`	renvoie un tableau de valeurs comprises en `start` inclu et `stop` **exclu** par pas de `step`
`np.zeros(N)`	renvoie un tableau de `N` zéros
`np.empty(N)`	renvoie un tableau de `N` cases non initialisées

Soit `x` un tableau.

Tracé de la fonction `f` Méthode 1 – fonction `f` vectorielle	`plt.plot(x, f(x))`
Tracé de la fonction `f` Méthode 2 – fonction `f` **non** vectorielle	`y = [f(u) for u in x]` `plt.plot(x, y)`
Tracé de la fonction `f` Méthode 3 – fonction `f` **non** vectorielle	`y = list( map(f, x) )` `plt.plot(x, y)`
Tracé de la fonction `f` Méthode 4 – fonction `f` **non** vectorielle	`g = np.vectorize(f)` `plt.plot(x, g(x))`