Fiches – numéro 4

Survivre en Python

Sommaire

Capacités exigibles

- ☐ Utiliser les fonctions de base de la bibliothèque `matplotlib` pour représenter un nuage de points, tracer la courbe représentative d'une fonction ou une courbe plane paramétrée.
- ☐ Utiliser les fonctions de base des bibliothèques `random` et/ou `numpy` (leurs spécifications étant fournies) pour réaliser des tirages d'une variable aléatoire.
- ☐ Déterminer la moyenne et l'écart-type d'un ensemble de tirages d'une variable aléatoire.
- ☐ Utiliser la fonction `hist` de `matplotlib.pyplot` pour représenter les résultats d'un ensemble de tirages d'une variable aléatoire.
- ☐ Utiliser la fonction `polyfit` de la bibliothèque `numpy` (sa spécification étant fournie) pour exploiter des données.

I Les bases

I/A Bonnes pratiques

À retenir

1) On n'**importera pas** `math` en physique, et on n'**importera jamais avec** `*` (`from numpy import *`) [1].
2) Séparer les cellules dans les Notebooks : imports, données, fonctions, calculs, tracés...
3) Choisir des noms de variables cohérents et logiques.
4) Écrire les unités des valeurs et tableaux en fin de ligne avec un #.
5) Comme sur les copies, **pas de calcul numérique** (`0.1/35.5` par exemple) [2], que du littéral.
6) Les `2.7*10**(-3)` sont à **absolument** proscrire. Il existe `2.7e-3` pour cela.
7) Ne convertissez pas des valeurs individuelles dans un tableau : convertissez le tableau entier (`valeurs_mV = valeurs_V * 1e-3`).
8) Attention à l'ordre des paramètres dans les fonctions ; variables locales ≠ variables globales (cf. TP00).

1. Il faut connaître l'origine des fonctions et écrire explicitement `np.pi` si on veut utiliser π
2. Vous pouvez bien sûr en faire au « brouillon » dans des cellules vides par exemple, mais le risque est de se tromper dans les unités.

I/B Calcul élémentaire

```python
# Ce qui est après un # est un commentaire, et non traité dans le code

a = 2        # affecte la valeur 2 à la variable globale a
b = a+2      # b vaut 2+2 = 4. Si on change la valeur de a, on devra recalculer b
c = 3*a      # c vaut 3*2 = 6. Si on change la valeur de a, on devra recalculer b
d = a**3     # d vaut a^3 = 8 : ** indique une puissance
e = 5.5e-3   # eNBRE est un raccourci pour 10^(NBRE). Ici, d = 0.0055.
```

I/C Fonctions basiques

```python
print(d)                # affiche la valeur de d
print(f'd = {d}')       # affiche "d = 0.0055"
print(f'd = {d:.2f}')   # affiche "d = 0.01" : décimal 2 chiffres après ','
print(f'd = {d:.2e}')   # affiche "d = 5.50e-03" : scientifique 2 décimales

type(a)   # donne le type d'objet de la variable a : int (entier)
type(d)   # donne le type d'objet de la variable d : float (décimal)

abs(-3)          # donne la valeur absolue d'un nombre : 3

len([1, 2, 3])  # donne la longueur d'une liste : 3
min([1, 2, 3])  # donne la valeur minimale d'une liste : 1
max([1, 2, 3])  # donne la valeur maximale d'une liste : 3
```

II Gestion de données

II/A Listes

```python
L = [1, 2, 3, 4] # créé la liste L contenant les valeurs 1, 2, 3 et 4
print(L[0])      # extrait la première valeur de L : 1
print(L[1])      # extrait la deuxième valeur de L : 2
print(L[1:3])    # extrait les valeurs de l'indice 1 à l'indice 3-1 de L : [2, 3]
print(L[:2])     # extrait les deux premières valeurs de L : [1, 2]
print(L[1:])     # extrait toutes les valeurs à partir de la deuxième : [2, 3, 4]
print(L[-1])     # extrait la dernière valeur de L : 4
print(L[-2:])    # extrait toutes les valeurs à partir de l'avant-dernière : [3, 4]

L.append(42)     # ajoute l'élément 42 à la fin de la liste
L2 = L + [5, 6]  # concatène la liste L et la liste [5, 6] dans une nouvelle
print(L2)        # [1, 2, 3, 42, 5, 6]
```

II/B numpy et tableaux

```python
import numpy as np

tab = np.array([1, 2, 3])  # créé le tableau [1, 2, 3]
tabplus1 = tab+1           # ajoute 1 à toutes les valeurs de tab
tabpetit = tab*2e-3        # multiplie toutes les valeurs de tab par 0.002
tabracin = np.sqrt(tab)    # applique racine carré à tous les éléments de tab
tabexpon = np.exp(tab)     # exponentielle
tablogne = np.log(tab)     # logarithme NÉPÉRIEN (ln français)
tablog10 = np.log10(tab)   # logarithme décimal (log français)

tabqueD1 = np.ones(taille)      # créé un tableau de taille "taille" rempli de 1
tabqueD2 = np.ones(taille)*2e-3 # créé un tableau de taille "taille" rempli de 2e-3
tabmoyen = np.mean(tab)         # donne la valeur moyenne de tab
tabecart = np.std(tab, ddof=1)  # donne l'écart-type de tab
```

III Automatisation

III/A Fonctions personnelles

```python
def puissance(arg1, arg2): # définit la fonction puissance de 2 arguments
    resultat = arg1**arg2  # variable locale de calcul
    return(resultat)       # fin de la fonction, résultat final

print(puissance(2, 3))     # donne le résultat du calcul : 8

def comparaison(x,y):
    if x > y:                                    # condition d'exécution
        print("argument 1 supérieur au second") # si oui, exécute
    elif x < y:                                  # sinon, autre condition
        print("argument 1 inférieur au second") # si oui, exécute
    else:                                        # pour tous les autres cas,
        print("argument 1 égal au second")       # exécute

print(comparaison(1,2))    # "argument 1 inférieur au second"
```

III/B Boucles `for`

```python
for i in range(10):   # créé i qui commence à 0, terminera à 9, et augmente
                      # de 1 à chaque réalisation des lignes en-dessous
    print(i)          # affichera 0, puis 1, puis 2, et jusqu'à 9

L = []                # liste vide
for i in range(3):    # exécute la suite 3 fois : i=0, puis 1, puis 2
    L.append(3*i)     # ajoute 3*i à la fin de la liste
print(L)              # [0, 3, 6]

for i, k in enumerate(L):  # i compte à partir de 0, k prend les valeurs de L
    print(f'L[{i}] = {k}') # L[0] = 0, L[1] = 3, L[2] = 6

L2 = [3*i for i in range(3)] # créé L d'une manière plus compacte
print(L2)                    # même résultat
```

IV Tracé de graphiques

IV/A Minimal (Figure F4.1)

```python
import matplotlib.pyplot as plt

abscisse = np.linspace(0, 10, 8) # définit les abscisses qu'on tracera
ordonnee = abscisse**2           # et les ordonnées

plt.plot(abscisse, ordonnee)     # place les points, les relie par des segments
plt.xlabel('t (s)')              # nomme l'abscisse
plt.ylabel('x (m)')              # nomme l'ordonnée

plt.show()                       # obligé pour afficher
```

IV/B Complexe (Figure F4.2)

```python
X = np.linspace(0, 10, 8)
Y = X**2

plt.figure(figsize=(8, 6))     # dimension horizontale, verticale
plt.grid()                     # affiche un quadrillage de lecture
plt.xticks(fontsize=20)        # affiche les nombres de l'axe x plus grand
plt.yticks(fontsize=20)        # affiche les nombres de l'axe y plus grand
```

```python
plt.xlabel('$x$ en cm',          # Donne le nom de l'axe x, $ pour le mode math
           fontsize=20)          # en grand
plt.ylabel('$y$ en cm',          # Donne le nom de l'axe y, $ pour le mode math
           fontsize=20)          # en grand

plt.scatter(X, Y,                # nuage de points X en abscisse et Y en ordonnée
            marker='x', s=100,   # possibilité de customiser le tracé
            color='blue',        # pour la couleur
            label='Données')     # pour la légende

plt.errorbar(X+1, Y+1,           # nuage de points X abscisse et Y ordonnée
             xerr=.5,            # incertitude en x
             yerr=5,             # incertitude en y
             capsize=3,          # indique la limite des erreurs
             color='orange',     # pour la couleur
             label='Avec incertitudes')

plt.plot(X-1, Y-1,               # graphique relié
         color="g",              # couleur
         marker="^",             # marker
         markersize=10,          # taille marker
         linestyle="dotted",     # type de ligne
         linewidth=2,            # épaisseur
         label="Courbe verte diamant pointillés")

plt.title('Présentation des options de tracé',
          fontsize=20)
plt.legend(fontsize=15)

plt.tight_layout()               # évite les débordements ou rognages
plt.xlim(min(X)-.1, max(X)+2)    # pour les limites d'affichage en abscisse
plt.ylim(min(Y)-6, max(Y)+10)    # pour les limites d'affichage en ordonnée
plt.show()
```

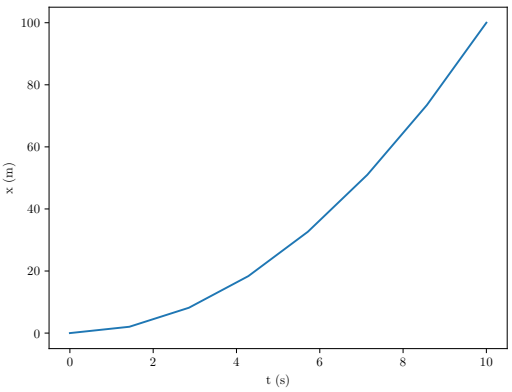


Figure F4.1 – Figure minimale.

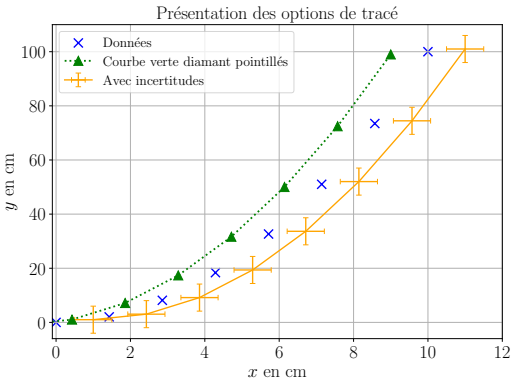


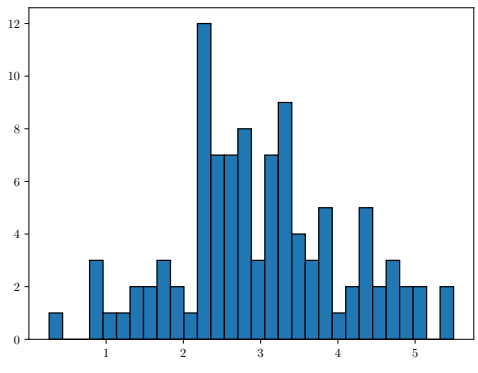
Figure F4.2 – Figure complexe.

IV/C Histogramme

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

data = np.random.normal(loc=3, scale=1, size=100)

# trace l'histogramme de l'échantillon
plt.hist(data, bins=30, edgecolor="black")
plt.show()
```



V Régressions, avec ou sans incertitudes et Monte-Carlo

V/A Fonctions numpy utiles

```python
a, b = np.polyfit(X, Y, 1) # donne les coefficients a et b de Y = a*X+b

tabdecou = np.linspace(min, max, nbre) # découpe [min, max] en nbre parties égales

# loi normale autour de "moyenne" avec un écart-type "ecart"
x_simu = np.random.normal(moyenne, ecart)
# loi uniforme dans l'intervalle [x-Delta_x, x+Delta_x]
X_simu = np.random.uniform(x-Delta_x, x+Delta_x)
```

V/B Régressions

```python
# ============================================================================ #
#                                    Imports                                   #
# ============================================================================ #

import numpy as np  # pour les tableaux et la gestion des données
import matplotlib.pyplot as plt  # pour les graphiques, l'affichage des données

# ============================================================================ #
#                            Données sans incertitude                          #
# ============================================================================ #

X = np.array([0, 2, 4, 6, 8, 10])  # À modifier + écrire unité
Y = np.array([0.5, 7.9, 11, 17.5, 26, 31.8])  # À modifier + écrire unité

# ============================================================================ #
#                                  Régression                                  #
# ============================================================================ #

# Donne a le coefficient directeur et b l'ordonnée à l'origine
a, b = np.polyfit(X, Y, 1)
# Affiche a et b. .3f pour 3 valeurs après la virgule
print(f"a = {a:.3f}, b = {b:.2f}")

# ============================================================================ #
#                                  Utilisation                                 #
# ============================================================================ #

X_reg = np.linspace(min(X), max(X), 100)  # découpe l'intervalle [min(X), max(X)]
Y_reg = a * X_reg + b  # calcule les valeurs de Y par régression linéaire

# ============================================================================ #
#                             Tracé sans incertitude                           #
# ============================================================================ #

plt.figure(figsize=(8, 6))
plt.grid()
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.xlabel("$grandeur$ en UNITÉ", fontsize=20)
plt.ylabel("$grandeur$ en UNITÉ", fontsize=20)

plt.scatter(X, Y, marker="x", s=100, color="b", label="Mesures")
plt.plot(X_reg, Y_reg, "r", label="Régression linéaire")

plt.title("Titre efficace et descriptif", fontsize=20)
plt.legend(fontsize=15)
plt.show()

```

```python
# ========================================================================== #
#                                Avec incertitude                            #
# ========================================================================== #

Delta_X = 0.1 * np.ones(len(X))  # À modifier + écrire unité
uX = Delta_X / np.sqrt(3)  # incertitude type B
Delta_Y = 0.3 * np.ones(len(Y))  # À modifier + écrire unité
uY = Delta_Y / np.sqrt(3)  # incertitude type B

# ========================================================================== #
#                             Tracé avec incertitude                         #
# ========================================================================== #

plt.close()  # il faut fermer la figure précédente

plt.figure(figsize=(8, 6))
plt.grid()
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.xlabel("$grandeur$ en UNITÉ", fontsize=20)
plt.ylabel("$grandeur$ en UNITÉ", fontsize=20)

plt.errorbar(
    X, Y, xerr=uX, yerr=uY, linestyle="None", capsize=3, color="b", label="Mesures"
)
plt.plot(X_reg, Y_reg, "r", label="Régression linéaire")

plt.title("Titre efficace et descriptif", fontsize=20)
plt.legend(fontsize=15)
plt.show()

# ========================================================================== #
#                             Monte-Carlo sur a, b                           #
# ========================================================================== #

N = 10000  # nombre de régressions à effectuer

a_liste, b_liste = [], []  # création des listes vides pour stocker les valeurs
for i in range(N):
    X_simu = np.random.uniform(X - Delta_X, X + Delta_X)  # simule X dans [X-Delta_X, X+Delta_X]
    Y_simu = np.random.uniform(Y - Delta_Y, Y + Delta_Y)  # Pareil pour Y

    a_simu, b_simu = np.polyfit(X_simu, Y_simu, 1)  # régression simulée

    a_liste.append(a_simu)
    b_liste.append(b_simu)

a_moy, b_moy = np.mean(a_liste), np.mean(b_liste)  # moyennes des listes
ua, ub = np.std(a_liste, ddof=1), np.std(b_liste, ddof=1)  # écarts-types des listes

# Affichage, à modifier pour l'écriture scientifique
print(f"Coef.directeur = {a_moy:.3e} ± {ua:.3e} unité")
print(f"Ordonnée à l'origine = {b_moy:.3e} ± {ub:.3e} unité")
```

Monte-Carlo scalaire

Revoir le cours N3 pour les incertitudes Monte-Carlo.