

Survivre en Python

Sommaire

I Les bases	1
I/A Bonnes pratiques	1
I/B Calcul élémentaire	2
I/C Fonctions basiques	2
II Gestion de données	2
II/A Listes	2
II/B <code>numpy</code> et tableaux	2
III Automatisation	3
III/A Fonctions personnelles	3
III/B Boucles <code>for</code>	3
IV Tracé de graphiques	3
IV/A Minimal (Figure F6.1)	3
IV/B Complexe (Figure F6.2)	3
IV/C Histogramme	4
V Régressions, avec ou sans incertitudes et MONTE-CARLO	5
V/A Fonctions <code>numpy</code> utiles	5
V/B Régressions	5

Capacités exigibles

- | | |
|--|---|
| <ul style="list-style-type: none"> ☐ Utiliser les fonctions de base de la bibliothèque <code>matplotlib</code> 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 <code>random</code> et/ou <code>numpy</code> (leurs spécifications étant fournies) pour réaliser des tirages d'une variable aléatoire. | <ul style="list-style-type: none"> ☐ Déterminer la moyenne et l'écart-type d'un ensemble de tirages d'une variable aléatoire. ☐ Utiliser la fonction <code>hist</code> de <code>matplotlib.pyplot</code> pour représenter les résultats d'un ensemble de tirages d'une variable aléatoire. ☐ Utiliser la fonction <code>polyfit</code> de la bibliothèque <code>numpy</code> (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 $\neq$ 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

```

1  # Ce qui est après un # est un commentaire, et non traité dans le code
2
3  a = 2          # affecte la valeur 2 à la variable globale a
4  b = a+2        # b vaut 2+2 = 4. Si on change la valeur de a, on devra recalculer b
5  c = 3*a        # c vaut 3*2 = 6. Si on change la valeur de a, on devra recalculer b
6  d = a**3       # d vaut a^3 = 8 : ** indique une puissance
7  e = 5.5e-3     # eNBRE est un raccourci pour 10^(NBRE). Ici, d = 0.0055.

```

I/C Fonctions basiques

```

1  print(d)          # affiche la valeur de d
2  print(f'd = {d}') # affiche "d = 0.0055"
3  print(f'd = {d:.2f}') # affiche "d = 0.01" : décimal 2 chiffres après ','
4  print(f'd = {d:.2e}') # affiche "d = 5.50e-03" : scientifique 2 décimales
5
6  type(a)           # donne le type d'objet de la variable a : int (entier)
7  type(d)           # donne le type d'objet de la variable d : float (décimal)
8
9  abs(-3)           # donne la valeur absolue d'un nombre : 3
10
11 len([1, 2, 3])    # donne la longueur d'une liste : 3
12 min([1, 2, 3])    # donne la valeur minimale d'une liste : 1
13 max([1, 2, 3])    # donne la valeur maximale d'une liste : 3

```

II Gestion de données

II/A Listes

```

1  L = [1, 2, 3, 4] # créé la liste L contenant les valeurs 1, 2, 3 et 4
2  print(L[0])      # extrait la première valeur de L : 1
3  print(L[1])      # extrait la deuxième valeur de L : 2
4  print(L[1:3])    # extrait les valeurs de l'indice 1 à l'indice 3-1 de L : [2, 3]
5  print(L[:2])     # extrait les deux premières valeurs de L : [1, 2]
6  print(L[1:])     # extrait toutes les valeurs à partir de la deuxième : [2, 3, 4]
7  print(L[-1])     # extrait la dernière valeur de L : 4
8  print(L[-2:])    # extrait toutes les valeurs à partir de l'avant-dernière : [3, 4]
9
10 L.append(42)      # ajoute l'élément 42 à la fin de la liste
11 L2 = L + [5, 6]   # concatène la liste L et la liste [5, 6] dans une nouvelle
12 print(L2)         # [1, 2, 3, 42, 5, 6]

```

II/B numpy et tableaux

```

1  import numpy as np
2
3  tab = np.array([1, 2, 3]) # créé le tableau [1, 2, 3]
4  tabplus1 = tab+1         # ajoute 1 à toutes les valeurs de tab
5  tabpetit = tab*2e-3      # multiplie toutes les valeurs de tab par 0.002
6  tabracin = np.sqrt(tab)  # applique racine carrée à tous les éléments de tab
7  tabexpon = np.exp(tab)   # exponentielle
8  tablogne = np.log(tab)   # logarithme NÉPÉRIEN (ln français)
9  tablog10 = np.log10(tab) # logarithme décimal (log français)
10
11 tabqueD1 = np.ones(taille) # créé un tableau de taille "taille" rempli de 1
12 tabqueD2 = np.ones(taille)*2e-3 # créé un tableau de taille "taille" rempli de 2e-3
13 tabmoyen = np.mean(tab)    # donne la valeur moyenne de tab
14 tabcart = np.std(tab, ddof=1) # donne l'écart-type de tab

```

III Automatisation

III/A Fonctions personnelles



```

1  def puissance(arg1, arg2): # définit la fonction puissance de 2 arguments
2      resultat = arg1**arg2 # variable locale de calcul
3      return(resultat)      # fin de la fonction, résultat final
4
5  print(puissance(2, 3))     # donne le résultat du calcul : 8
6
7  def comparaison(x,y):
8      if x > y:               # condition d'exécution
9          print("argument 1 supérieur au second") # si oui, exécute
10         elif x < y:         # sinon, autre condition
11             print("argument 1 inférieur au second") # si oui, exécute
12         else:               # pour tous les autres cas,
13             print("argument 1 égal au second")      # exécute
14
15  print(comparaison(1,2))   # "argument 1 inférieur au second"

```

III/B Boucles for



```

1  for i in range(10):        # créé i qui commence à 0, terminera à 9, et augmente
2                              # de 1 à chaque réalisation des lignes en-dessous
3      print(i)               # affichera 0, puis 1, puis 2, et jusqu'à 9
4
5  L = []                     # liste vide
6  for i in range(3):         # exécute la suite 3 fois : i=0, puis 1, puis 2
7      L.append(3*i)          # ajoute 3*i à la fin de la liste
8  print(L)                   # [0, 3, 6]
9
10 for i, k in enumerate(L):  # i compte à partir de 0, k prend les valeurs de L
11     print(f'L[{i}] = {k}') # L[0] = 0, L[1] = 3, L[2] = 6
12
13 L2 = [3*i for i in range(3)] # créé L d'une manière plus compacte
14 print(L2)                   # même résultat

```

IV Tracé de graphiques

IV/A Minimal (Figure F6.1)



```

1  import matplotlib.pyplot as plt
2
3  abscisse = np.linspace(0, 10, 8) # définit les abscisses qu'on tracera
4  ordonnee = abscisse**2           # et les ordonnées
5
6  plt.plot(abscisse, ordonnee)     # place les points, les relie par des segments
7  plt.xlabel('t (s)')              # nomme l'abscisse
8  plt.ylabel('x (m)')              # nomme l'ordonnée
9
10 plt.show()                       # obligé pour afficher

```

IV/B Complexe (Figure F6.2)



```

1  X = np.linspace(0, 10, 8)
2  Y = X**2
3
4  plt.figure(figsize=(8, 6))       # dimension horizontale, verticale
5  plt.grid()                       # affiche un quadrillage de lecture
6  plt.xticks(fontsize=20)          # affiche les nombres de l'axe x plus grand
7  plt.yticks(fontsize=20)          # affiche les nombres de l'axe y plus grand

```

```

8 plt.xlabel('$x$ en cm',          # Donne le nom de l'axe x, $ pour le mode math
9           fontsize=20)          # en grand
10 plt.ylabel('$y$ en cm',         # Donne le nom de l'axe y, $ pour le mode math
11          fontsize=20)          # en grand
12
13 plt.scatter(X, Y,                # nuage de points X en abscisse et Y en ordonnée
14            marker='x', s=100,    # possibilité de customiser le tracé
15            color='blue',         # pour la couleur
16            label='Données')     # pour la légende
17
18 plt.errorbar(X+1, Y+1,           # nuage de points X abscisse et Y ordonnée
19             xerr=.5,             # incertitude en x
20             yerr=5,              # incertitude en y
21             capsize=3,           # indique la limite des erreurs
22             color='orange',      # pour la couleur
23             label='Avec incertitudes')
24
25 plt.plot(X-1, Y-1,              # graphique relié
26          color="g",              # couleur
27          marker="^",             # marker
28          markersize=10,          # taille marker
29          linestyle="dotted",     # type de ligne
30          linewidth=2,            # épaisseur
31          label="Courbe verte diamant pointillés")
32
33 plt.title('Présentation des options de tracé',
34           fontsize=20)
35 plt.legend(fontsize=15)
36
37 plt.tight_layout()              # évite les débordements ou rognages
38 plt.xlim(min(X)-.1, max(X)+2)  # pour les limites d'affichage en abscisse
39 plt.ylim(min(Y)-6, max(Y)+10)  # pour les limites d'affichage en ordonnée
40 plt.show()

```

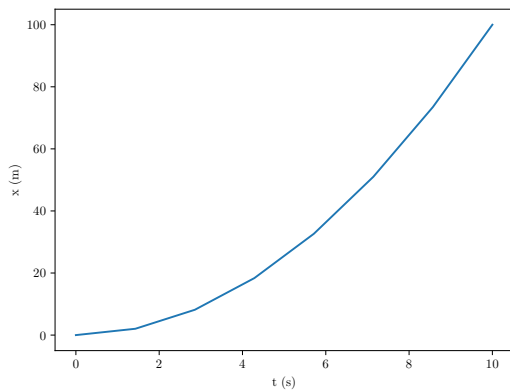


FIGURE F6.1 – Figure minimale.

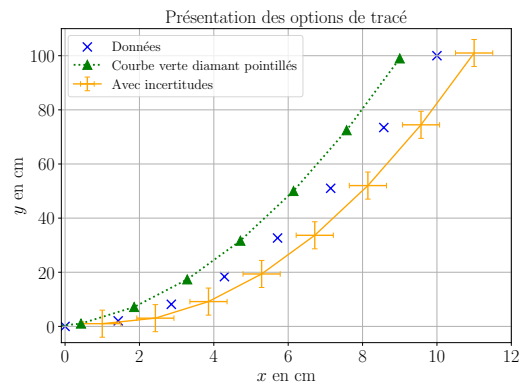


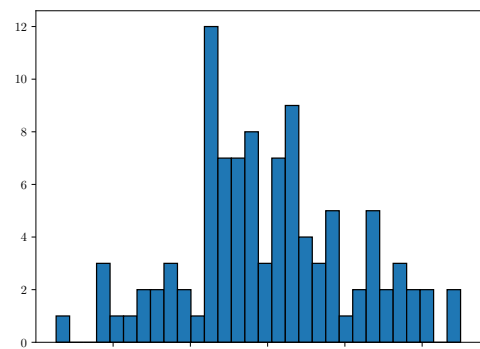
FIGURE F6.2 – Figure complexe.

IV/C Histogramme

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 data = np.random.normal(loc=3, scale=1, size=100)
5
6 # trace l'histogramme de l'échantillon
7 plt.hist(data, bins=30, edgecolor="black")
8 plt.show()

```



V Régressions, avec ou sans incertitudes et MONTE-CARLO

V/A Fonctions numpy utiles

```

1 a, b = np.polyfit(X, Y, 1) # donne les coefficients a et b de Y = a*X+b
2
3 tabdecou = np.linspace(min, max, nbre) # découpe [min, max] en nbre parties égales
4
5 # loi normale autour de "moyenne" avec un écart-type "ecart"
6 x_simu = np.random.normal(moyenne, ecart)
7 # loi uniforme dans l'intervalle [x-Delta_x, x+Delta_x]
8 X_simu = np.random.uniform(x-Delta_x, x+Delta_x)

```

V/B Régressions

```

1 # ===== #
2 # Imports #
3 # ===== #
4
5 import numpy as np # pour les tableaux et la gestion des données
6 import matplotlib.pyplot as plt # pour les graphiques, l'affichage des données
7
8 # ===== #
9 # Données sans incertitude #
10 # ===== #
11
12 X = np.array([0, 2, 4, 6, 8, 10]) # À modifier + écrire unité
13 Y = np.array([0.5, 7.9, 11, 17.5, 26, 31.8]) # À modifier + écrire unité
14
15 # ===== #
16 # Régression #
17 # ===== #
18
19 # Donne a le coefficient directeur et b l'ordonnée à l'origine
20 a, b = np.polyfit(X, Y, 1)
21 # Affiche a et b. .3f pour 3 valeurs après la virgule
22 print(f"a = {a:.3f}, b = {b:.2f}")
23
24 # ===== #
25 # Utilisation #
26 # ===== #
27
28 X_reg = np.linspace(min(X), max(X), 100) # découpe l'intervalle [min(X), max(X)]
29 Y_reg = a * X_reg + b # calcule les valeurs de Y par régression linéaire
30
31 # ===== #
32 # Tracé sans incertitude #
33 # ===== #
34
35 plt.figure(figsize=(8, 6))
36 plt.grid()
37 plt.xticks(fontsize=10)
38 plt.yticks(fontsize=10)
39 plt.xlabel("$grandeur$ en UNITÉ", fontsize=20)
40 plt.ylabel("$grandeur$ en UNITÉ", fontsize=20)
41
42 plt.scatter(X, Y, marker="x", s=100, color="b", label="Mesures")
43 plt.plot(X_reg, Y_reg, "r", label="Régression linéaire")
44
45 plt.title("Titre efficace et descriptif", fontsize=20)
46 plt.legend(fontsize=15)
47 plt.show()
48

```

```

49 # ===== #
50 #                                     Avec incertitude                                     #
51 # ===== #
52
53 Delta_X = 0.1 * np.ones(len(X)) # À modifier + écrire unité
54 uX = Delta_X / np.sqrt(3) # incertitude type B
55 Delta_Y = 0.3 * np.ones(len(Y)) # À modifier + écrire unité
56 uY = Delta_Y / np.sqrt(3) # incertitude type B
57
58 # ===== #
59 #                                     Tracé avec incertitude                                     #
60 # ===== #
61
62 plt.close() # il faut fermer la figure précédente
63
64 plt.figure(figsize=(8, 6))
65 plt.grid()
66 plt.xticks(fontsize=10)
67 plt.yticks(fontsize=10)
68 plt.xlabel("$grandeur$ en UNITÉ", fontsize=20)
69 plt.ylabel("$grandeur$ en UNITÉ", fontsize=20)
70
71 plt.errorbar(
72     X, Y, xerr=uX, yerr=uY, linestyle="None", capsize=3, color="b", label="Mesures"
73 )
74 plt.plot(X_reg, Y_reg, "r", label="Régression linéaire")
75
76 plt.title("Titre efficace et descriptif", fontsize=20)
77 plt.legend(fontsize=15)
78 plt.show()
79
80 # ===== #
81 #                                     Monte-Carlo sur a, b                                     #
82 # ===== #
83
84 N = 10000 # nombre de régressions à effectuer
85
86 a_liste, b_liste = [], [] # création des listes vides pour stocker les valeurs
87 for i in range(N):
88     X_simu = np.random.uniform(X - Delta_X, X + Delta_X) # simule X dans [X-Delta_X, X+Delta_X]
89     Y_simu = np.random.uniform(Y - Delta_Y, Y + Delta_Y) # Pareil pour Y
90
91     a_simu, b_simu = np.polyfit(X_simu, Y_simu, 1) # régression simulée
92
93     a_liste.append(a_simu)
94     b_liste.append(b_simu)
95
96 a_moy, b_moy = np.mean(a_liste), np.mean(b_liste) # moyennes des listes
97 ua, ub = np.std(a_liste, ddof=1), np.std(b_liste, ddof=1) # écarts-types des listes
98
99 # Affichage, à modifier pour l'écriture scientifique
100 print(f"Coef.directeur = {a_moy:.3e} ± {ua:.3e} unité")
101 print(f"Ordonnée à l'origine = {b_moy:.3e} ± {ub:.3e} unité")

```



MONTÉ-CARLO scalaire

Revoir le cours N2 pour les incertitudes MONTE-CARLO.