I Capacité numérique :

Simuler, à l'aide d'un langage de programmation ou d'un tableur,
un processus aléatoire permettant de caractériser la variabilité
de la valeur d'une grandeur composée

II Outils

On rappelle quelques bases fondamentales pour l'utilisation de python en sciences physiques. En préambule, rappelons que `python` utilise des règles d'**indentation** strictes.

II.1 Modules

On aura besoin de charger des bibliothèques nommées `modules` pour accéder à certaines fonctions.

```
import math
```

Les fonctions qu'elle apporte seront appelées par `math.nom_de_la_fonction`

II.2 Variables et affectation

On peut manipuler des entiers, des flottants, sur lesquels on peut utiliser les opérations usuelles et qu'on affecte à des variables par =. La structure avec `f'` permet par exemple de former une chaîne de caractères utilisant les valeurs des variables.

```
a = 2
b,c = 3,5
d = a + b
print(d, a + c)
print(f'la valeur de d est: {d}')
print(f'le quotient par / de deux entiers est un flottant "d/2": {d/2}')
print(f'la division euclidienne est obtenue par "d//2": {d//2}')
print(f'son reste est obtenu par "d%2": {d%2}')
print(f'on élève à une puissance en utilisant "pow" ou "**": {pow(a,4)} ou {a**5}')

print(f'on peut préciser l\'approximation décimale des flottants: ln(12) =
↪  {math.log(12):.3f}')
print(f'on utilise une notation scientifique avec ".2e" = {math.log(12):.2e}')
```

Sur les chaînes de caractères, on utilisera par exemples les opérations suivantes :

```
a = "bonjour"
b = " "
c = "la HX2"
print(f'"+" désigne la concaténation: {a+b+c}')
print(f'"*" n désigne la répétition n fois: {a*5}')
```

II.3 Listes et slicing

On manipulera souvent des listes de variables, de types variables. Comme tous les objets en `python`, elles possèdent des `méthodes` accessibles par la notation `objet.methode` :

```
a = "bonjour"
b = 3.5
c = 2
liste = [a,b]
liste.append(c)
print(f'on accède à un élément de position donnée par liste[0]: "{liste[0]}')
print(f'liste[-1] désigne le dernier élément: "{liste[2]}')
print(f'les opérateurs + et * n joignent ou dupliquent les éléments d\'une
↪  liste:{liste+liste, 3*liste}')
print(f'le nombre d\'éléments est donné par "len": {len(liste)}')
```

On peut former des listes en utilisant des boucles `for`, des conditions `if`.

```
autre_liste = [3*i for i in range(10) if 2*i<19]
print(autre_liste)
```

On peut «trancher» les listes (slice en anglais) pour n'en prendre que certains éléments en choisissant le premier, le dernier, et l'incrément.

```
print(f'un élément sur deux entre le deuxième et le septième: {autre_liste[1:6:2]}')
print(f'un élément sur trois entre le premier et l\'avant dernier:
↪  {autre_liste[:-2:3]}')
print(f'tous les éléments supérieurs à 13: {[ i for i in autre_liste if i > 13]}')
```

II.4 Numpy

Pour manipuler des données numériques, tracer des courbes, on utilisera un type de liste particulier, les `numpy.array` facilitant les manipulations d'un grand nombre de données en parallèle. On en fait ici un alias `np` plus court à écrire.

```
import numpy as np

abscisses = np.array(range(10))
ordonnees = abscisses + 2
ordonnees2 = abscisses *2
ordonnees3 = abscisses * abscisses
ordonnees4 = np.sqrt(abscisses)
print(f'les opérateurs usuels comme "+,*" s\'appliquent à chacun des éléments de la
↪  liste: {ordonnees}' )
print(ordonnees4[-1],ordonnees3[2],ordonnees2[0])
```

II.5 Fonctions et boucles

On définit des fonctions avec `def`, le corps de la fonction doit ensuite être indenté de 4 espaces, tout comme les éléments des boucles et structures conditionnelles.

```
def ma_fonction(ma_variable):
    seuil = 8
    if ma_variable > seuil:
        return ma_variable
    else:
        print("trop petit")
        return seuil
print(ma_fonction(12))
print(ma_fonction(5))
```

II.6 Courbes

Le module `matplotlib` permet de tracer des ensembles de points : on utilisera le plus souvent des `np.array` pour les abscisses et les ordonnées. La méthode `np.linspace` permet d'utiliser un grand nombre de points d'abscisses, uniformément répartis dans un intervalle.

```
import matplotlib.pyplot as plt
xmin = 0
xmax = 12
Npoints = 200
AbscissesII = np.linspace(xmin,xmax,Npoints)
OrdonneesII = np.power(AbscissesII,3)
OrdonneesIII = np.power(AbscissesII,.5)

fig,(axeII,axeIII) = plt.subplots(2,1) # pour organiser plusieurs graphes sur unemême
↪  figure
axeII.plot(AbscissesII,OrdonneesII,label="cube")
axeII.legend()
axeIII.plot(AbscissesII,OrdonneesIII,label="racine")
axeIII.legend()
fig.suptitle("Ceci est un titre")
axeII.set_xlabel("abscisses")
axeII.set_xlabel(None)
axeIII.set_xlabel("abscisses")
axeII.set_ylabel("ordonnees")
axeIII.set_ylabel("ordonnees")
fig.show()
```

On pourra consulter `https://matplotlib.org/cheatsheets/` pour plus de précisions.

III Chute libre

III.1 Dispositif expérimental

On cherche à mesurer l'accélération de la pesanteur g en étudiant la chute libre d'un corps dans le vide. Le dispositif consiste en :

- un objet de masse m en chute libre dans le vide ;
- est lâché sans vitesse initiale d'une altitude h_1 ;
- sa vitesse v est mesurée par un capteur spécifique quand il passe en un point d'altitude $h_2 < h_1$.

III.2 Modèle

Montrer qu'on a :

$$v^2 = 2g(h_1 - h_2)$$

III.3 Incertitudes

On cherche à observer l'effet sur l'incertitude sur la mesure de g des sources d'incertitude suivantes :

- incertitude sur les lectures des altitudes h_1 et h_2 sur une règle ;
- incertitude sur la mesure de la vitesse par un capteur.

On **simule** ici numériquement les répartitions des valeurs mesurées pour ces grandeurs si on reproduisait un grand nombre de fois la manipulation, en supposant connue la loi de répartition des erreurs.

1. Lecture des altitudes sur une règle

   On suppose une répartition uniforme d'erreurs entre deux graduations de la règle distantes de $2\Delta h$.

   On rappelle que l'incertitude-type vaut alors :

   $$\Delta h/(\sqrt{3})$$

   On utilisera la fonction `random.random_sample` pour tirer un nombre flottant aléatoire correspondant à une altitude lue entre deux graduations de la règle.

2. Mesure de la vitesse

   On suppose une répartition d'erreurs autour d'une vitesse v donnée par une loi normale d'incertitude-type Δv.

   On utilisera la fonction `random.normal` pour tirer un nombre flottant aléatoire correspondant cette loi normale.

IV Code

IV.1 Fonctions permettant de réaliser les tirages aléatoires

On utilise les fonctions offertes par `numpy.random`

```
# tirage aléatoire pour une loi normale de moyenne X[0] et d'incertitude-type X[1]
def tirage_normal(X):
  return X[0] + X[1]*np.random.normal()

# tirage aléatoire pour une répartition uniforme entre X[0] et X[1]
def tirage_uniforme(X):
  return X[0] + (X[1]-X[0])*np.random.random_sample()
```

IV.2 Paramètres

On suppose pour cette simulation connue les valeurs vraies de g, h_1 et h_2. On en déduit celle de v.

```
# Accélération de la pesanteur
g = 9.80665 #en m/s, valeur normale de la CGPM

# Hauteur de chute
h1 = 3 # en m
h2 = 2 # en m
h0 = h1-h2
Deltah = 5e-3 # en m, demi-largeur de la lecture de h

# Vitesse atteinte
v0 = np.sqrt(2* g * h0) # en m/s
Deltav = 1e-2 # en m/s, incertitude-type sur la mesure de v
```

IV.3 Simulation des mesures

On crée des listes `numpy array` de mesures de h_1, h_2 et v pour faciliter leur manipulation.

```
# Mesures
N = 10000 # nombre de points de mesure
g_calculee = np.array([])

h1min = h1-Deltah
h1max = h1+Deltah
h2min = h2-Deltah
h2max = h2+Deltah

MesuresV = np.array([tirage_normal([v0,Deltav]) for i in range(N)])
MesuresH1 = np.array([tirage_uniforme([h1min,h1max]) for i in range(N)])
MesuresH2 = np.array([tirage_uniforme([h2min,h2max]) for i in range(N)])
```

IV.4 Étude de g

On calcule les valeurs de g, leur valeur moyenne et l'écart-type de leur distribution.

```
# Valeurs de $h = h1-h2
MesuresH = MesuresH1 - MesuresH2
# On pourrait faire la même chose sans numpy array avec:
# MesuresH = [MesuresH1[i] - MesuresH2[i] for  i in range(N)]
# ou
# for i in range(N)
#    MesuresH[i] = MesuresH1[i] - MesuresH2[i]

# Valeurs calculées de g = v^2/(2h)
```

```
g_calculees = MesuresV**2/(2*MesuresH)
g_moyenne = np.average(g_calculees)
Stdevg = np.std(g_calculees,ddof=1)
print(f'moyenne des valeurs calculées de g: {g_moyenne:.3e}')
print(f'incertitude-type sur les valeurs calculées de g: {Stdevg:.3e}')
```

L'argument `ddof=1` passé à la méthode `std` de `numpy` est nécessaire pour utiliser le terme en $\sqrt{N-1}$ dans la définition de l'écart-type corrigé :

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(m_i - \overline{m})^2},$$

On affiche un histogramme de ces valeurs.

```
histo,(axehisto)=plt.subplots(1,1)
axehisto.hist(g_calculees, bins = 50, color = 'blue', edgecolor = 'black')
axehisto.set_xlabel('g (m/s^2)')
axehisto.set_ylabel('effectif')
histo.suptitle(f'Pour {N} iterations')
histo.show()
```

V Questions

V.1 Manipulations élémentaires

1. Créer une liste nommée `liste1` contenant les entiers pairs entre $4 et 20$.
2. Créer une liste `liste2` contenant leurs produits par 3.
3. Transformer ces listes en `numpy.array` et en déduire leurs produits termes à terme de leurs éléments.
4. Créer une fonction `somme(n)` qui renvoie la somme des $1/p^2$ pour $p \in [1;n]$. On peut montrer que `somme(n)` tend vers $\pi^2/6$ quand $n \to \infty$.

   Calculer la différence relative pour quelques grandeurs valeurs de n, et l'afficher en notation scientifique. La valeur de π est accessible par `np.pi`.
5. Tracer la courbe représentative de $x \mapsto e^{(-x/2)}\cos(2\pi x)$ pour $x \in [0;5]$. On utilisera les fonctions `np.exp` et `np.cos` (dont les arguments sont des radians).

V.2 Incertitudes composées

1. Afficher l'écart-type relative des mesures de g.
2. Afficher les incertitudes-types et les incertitudes-types relatives sur v, $h1$, $h2$.
3. Afficher l'écart-type des valeurs mesurées de h et comparer à l'incertitude-type composée à partir des incertitudes-types de h_1 et h_2. Calculer l'incertitude-type relative sur h.
4. Déterminer et afficher l'incertitude-type relative composée sur g en fonction des incertitudes-types relatives précédentes et comparer à l'écart-type relatif sur g.

   On a $g = v^2/(2h)$, soit $\Delta g/g = \sqrt{4(\Delta v/v)^2 + (\Delta h/h)^2}$, qu'on compare à l'écart-type des valeurs calculées de g.
5. Augmenter d'un facteur 10 la précision sur la vitesse ? Quel est l'effet sur la précision de la détermination de g. Commenter.

V.3 Effet du nombre de mesures

1. Changer le nombre d'itérations N d'un facteur 100 dans un sens ou dans l'autre. Cela change-t-il les résultats précédents ?

   On passe de $N = 1e4$ à $Np = 100$:
2. Pour un jeu de $N = 1e4$ mesures, effectuer des moyennes de $m = 100$ et étudier l'incertitude-type des $N/m = 100$ mesures obtenues. Vérifier qu'il est réduit d'un rapport $\sqrt{m}$.