# Sujet Python et Mathématiques - EFI

## Partie I : Étude et calcul de suites

On considère la suite $(u_n)$ définie par

$$u_0 = 1, \quad u_{n+1} = \frac{1}{2}u_n + 3.$$

1. Calculer $u_1$, $u_2$ et $u_3$.
2. Montrer que $(u_n)$ converge et déterminer sa limite.
3. Écrire une fonction Python `suite(n)` qui calcule $u_n$ par récursion.
4. On peut montrer par récurrence que pour une suite arithmético-géométrique $u_{n+1} = au_n + b$, on a $u_n = a^n(u_0 - l) + l$ avec $l$ la limite. Démontrer ceci et déterminer $u_n$ explicitement.
5. Écrire une fonction Python `suite_explicite(n)` qui calcule $u_n$ via la formule explicite.
6. Vérifier avec Python que `suite(n)` et `suite_explicite(n)` donnent les mêmes résultats pour $n = 20$.

## Partie II : Estimation de $\pi$ par la méthode de Monte Carlo

**Exercice 6.** On souhaite estimer la valeur de $\pi$ par une méthode probabiliste appelée « méthode de Monte Carlo ». Elle repose sur une idée géométrique simple.

On considère le quart de disque de rayon 1 centré à l'origine dans le carré $[0, 1] \times [0, 1]$.

**Partie A – Modélisation géométrique**

1. Rappeler la formule de l'aire d'un disque et en déduire celle d'un quart de disque de rayon 1.
2. On choisit un point $(X, Y)$ au hasard dans le carré $[0, 1] \times [0, 1]$ (loi uniforme). Montrer que la probabilite que ce point tombe dans le quart de disque est $\frac{\pi}{4}$.

**Partie B – Modele probabiliste**

3. On simule $n$ points $(X_i, Y_i)$ independants, uniformement choisis dans $[0, 1]^2$. On definit :

$$I_i = \begin{cases} 1 & \text{si } X_i^2 + Y_i^2 \leq 1 \\ 0 & \text{sinon} \end{cases}$$

   Montrer que l'esperance de $I_i$ est $\frac{\pi}{4}$.
4. On note $M_n = \frac{1}{n}\sum_{i=1}^{n} I_i$. Donner une estimation de $\pi$ en fonction de $M_n$.

**Partie C – Simulation**

5. Ecrire une fonction Python qui simule $n$ couples $(X_i, Y_i)$ et retourne une estimation de $\pi$ par la methode de Monte Carlo.
6. Tester cette fonction pour $n = 1000$, $n = 10000$ et $n = 100000$.
7. Refaire l'experience 100 fois avec $n = 10000$ et representer les resultats sur un histogramme.
8. Expliquer pourquoi l'estimation s'ameliore quand $n$ augmente.

## Partie III : Approximation numérique de dérivées

Soit une fonction $f$ définie et dérivable sur un intervalle $[a, b]$.

1. Rappel : pour $x \in [a, b]$ et un petit $h > 0$, on approxime la dérivée par la différence finie avant :
$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h} \approx \frac{f(x+h) - f(x)}{h}.$$
   (a) Montrer que cette approximation est d'ordre 1 en $h$, c'est-à-dire que l'erreur est proportionnelle à $h$.
2. Écrire une fonction Python `derivee_avant(f,x,h)` qui calcule cette approximation, où `f` est une fonction Python, `x` un flottant, `h` un petit flottant.
3. Écrire une fonction Python `derivee_centrale(f,x,h)` qui utilise la différence centrée :
$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}.$$
   Indiquer l'ordre d'erreur de cette approximation.
4. Pour la fonction $f(x) = \sin(x)$, sur $[0, 2\pi]$, programmer un calcul des dérivées approchées en 100 points régulièrement espacés.
5. Tracer avec `matplotlib` la courbe de $f'(x) = \cos(x)$ (valeur exacte) et les approximations obtenues avec `derivee_avant` et `derivee_centrale` (sur le même graphique).
6. Étudier numériquement l'erreur maximale (en valeur absolue) entre la vraie dérivée et chaque approximation lorsque $h$ varie de $10^{-1}$ à $10^{-5}$ (par exemple, en $h = 10^{-k}$ pour $k = 1, \ldots, 5$). Tracer ces erreurs en échelle logarithmique.
7. Conclure sur l'efficacité relative des deux méthodes.

## Partie IV : Étude d'une fonction polynomiale et zéros

On considère la fonction $g$ définie par
$$g(x) = x^4 - 3x^3 + 2.$$

1. Étudier les variations de $g$ (calcul de la dérivée, tableaux de signe).
2. Déterminer les points critiques et leurs natures (minimum, maximum, point d'inflexion).
3. Écrire une fonction Python `g(x)` qui calcule $g(x)$.
4. Utiliser la méthode de dichotomie (bisection) pour approximer une racine de $g$ sur $[0, 2]$ avec une précision $10^{-6}$. On rappelle la fonction Python suivante :

```python
def dichotomie(f, a, b, epsilon):
    while b - a > epsilon:
        m = (a + b)/2
        if f(m) == 0:
            return m
        elif f(a)*f(m) < 0:
            b = m
        else:
            a = m
    return (a + b)/2
```

5. Afficher la valeur approchée trouvée et vérifier $g$ en ce point.
6. Tracer la courbe de $g$ sur $[-1, 3]$.

## Partie V : Calculs numériques avancés et suites (1h30)

1. La fonction exponentielle peut être approchée par la somme de sa série de Taylor centrée en 0 :

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}.$$

   Écrire une fonction Python `factorielle(n)` calculant $n!$.

2. Écrire une fonction Python `exp_approx(x,n)` qui calcule la somme des $n$ premiers termes pour approcher $e^x$.
3. Pour $x = 1$, calculer les approximations pour $n = 5, 10, 15$ et comparer avec la valeur de `math.exp(1)`.
4. Implémenter une fonction Python `erreur_approx(x,n)` qui retourne l'erreur absolue entre `exp_approx(x,n)` et `math.exp(x)`.
5. Tracer la courbe de l'erreur en fonction de $n$ (par exemple pour $n = 1$ à 30).
6. (Bonus) Adapter `exp_approx` pour utiliser une méthode récursive afin d'éviter de recalculer les factorielles à chaque terme.
7. Étudier numériquement la convergence de la suite définie par

$$w_0 = 1, \quad w_{n+1} = \frac{1}{2}(w_n + \frac{2}{w_n})$$

   (suite de Héron pour $\sqrt{2}$).

   (a) Écrire une fonction Python `heron_sqrt2(n)` qui calcule $w_n$.

   (b) Calculer $w_{10}$ et comparer à $\sqrt{2}$ (fonction `math.sqrt(2)`).