**Informatique - mercredi 11 juin 2025**
**Durée : 1 h**

- **Aucun document autorisé. Calculatrice interdite.**
- **La lisibilité des codes proposés sera sensiblement prise en considération dans l'évaluation des copies.**
- **Ce sujet comporte 4 pages.**
  **Il est constitué de 3 exercices indépendants.**

**Exercice 1.**

Pour son TIPE, Alice étudie l'impact éventuel d'un engrais sur la croissance de plants de haricots. Pour cela, elle réalise deux expériences séparées. D'un côté, elle place 10 haricots à germer dans du terreau avec engrais (groupe test, T). De l'autre, elle place 10 haricots à germer dans du terreau sans engrais (groupe contrôle, C).

Après 20 jours, elle mesure la masse en grammes de chacun des haricots. Elle note les résultats du groupe T dans une liste Python `T`, et ceux du groupe C dans une liste Python `C`. Elle souhaite savoir si ses mesures permettent ou non de dire que l'engrais améliore la croissance des haricots.

Pour cela, elle calcule les moyennes $m_T$ et $m_C$, et les écarts-type $\sigma_T$ et $\sigma_C$ des données de chaque groupe. Puis elle dessine l'histogramme ci-contre où elle représente la moyenne de chaque groupe en précisant une barre d'erreur. Alice a choisi d'utiliser une barre d'erreur simple : dans chacun des deux groupes, elle considère que son intervalle de confiance est $[m - \sigma, m + \sigma]$ où $m$ est la moyenne et $\sigma$ l'écart-type.

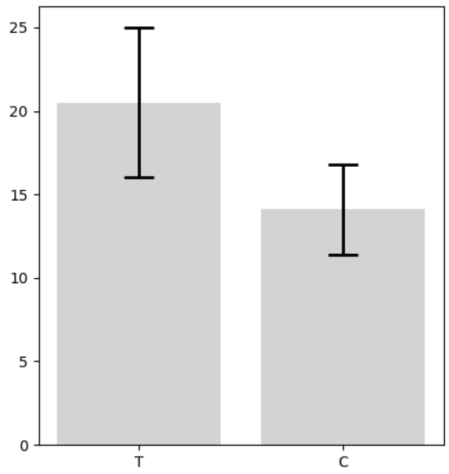


FIGURE 1 – Barres d'erreurs pour les groupes T et C.

On rappelle que pour une série statistique $(x_1, x_2, \ldots, x_n)$ la moyenne $m$ et l'écart-type $\sigma$ sont donnés par les formules :

$$m = \frac{1}{n}\sum_{k=1}^{n} x_k \quad \text{et} \quad \sigma = \sqrt{\frac{1}{n}\left(\sum_{k=1}^{n}(x_k - m)^2\right)}$$

1. Écrire une fonction `moy` et une fonction `ecart` prenant toutes les deux en argument une liste de nombres `L` et renvoyant respectivement la moyenne et l'écart-type des éléments de `L`.
2. En déduire une fonction `barres` prenant en arguments les listes `T` et `C` d'Alice et renvoyant les bornes des barres d'erreurs, c'est-à-dire les quantités suivantes, dans cet ordre, $m_T - \sigma_T, m_T + \sigma_T, m_C - \sigma_C, m_C + \sigma_C$.
3. Si on note $(a, b, c, d) = (m_T - \sigma_T, m_T + \sigma_T, m_C - \sigma_C, m_C + \sigma_C)$, à quelle condition sur $a, b, c$ et $d$, les barres d'erreurs tracées par Alice ne se recoupent pas ? En déduire une fonction `impact_engrais` prenant en arguments les listes `T` et `C` et renvoyant `True` si les valeurs mesurées par Alice permettent de conclure de manière statistiquement significative à un impact de l'engrais sur la croissance des haricots, et `False` sinon.

**Exercice 2.**
Le modèle de Lotka-Volterra décrit l'évolution de deux populations $N$ (les proies) et $P$ (les prédateurs) au cours du temps $(t)$ par le système d'équations différentielles suivant :

$$(S) \ : \ \begin{cases} \dfrac{dN}{dt} = rN - CNP \\ \dfrac{dP}{dt} = -mP + gCNP \end{cases}$$

où $r$ est le taux de croissance intrinsèque des proies, $m$ le taux de mortalité des prédateurs, $g$ le taux de croissance des prédateurs selon la densité des proies et $C$ une dernière constante. On suppose de plus qu'on connaît les popultations initales $N(t=0) = N_0$, $P(t=0) = P_0$ et qu'on cherche à décrire l'évolution des populations jusqu'à un instant $t = \texttt{T\_max}$.

Le système d'équations différentielles $(S)$ étant impossible à résoudre de manière exacte, on propose d'utiliser la méthode d'Euler pour calculer une approximation de la solution. Le principe de cette méthode est de calculer une approximation de $N(t_i)$ et $P(t_i)$ en différents points $t_i$ régulièrement espacés entre 0 et `T_max`. On choisit d'écrire $t_i = i \times h$ où $h = \dfrac{\texttt{T\_max}}{\texttt{Nb\_iter}}$ est le pas, et où `Nb_iter` est le nombre d'itérations de la méthode d'Euler.

On utilise alors que pour $f$ une fonction $\mathcal{C}^1$, on a, si $h$ est suffisamment petit :

$$f(t_i + h) \simeq f(t_i) + h \times f'(t_i)$$

Dans le cas des fonctions $N$ et $P$, on remplace alors $N'(t_i) = \dfrac{dN}{dt}(t_i)$ et $P'(t_i) = \dfrac{dP}{dt}(t_i)$ par les expressions données dans le système $(S)$. On obtient alors des suites $(N_i)_{i \in [\![0, \texttt{Nb\_iter}]\!]}$ et $(P_i)_{i \in [\![0, \texttt{Nb\_iter}]\!]}$ définies par les valeurs initiales $N_0$ et $P_0$ et les relations de récurrence suivantes :

$$\forall i \in [\![0, \texttt{Nb\_iter} - 1]\!], \ \begin{cases} N_{i+1} = N_i + h \times (rN_i - CN_iP_i) \\ P_{i+1} = P_i + h \times (-mP_i + gCN_iP_i) \end{cases}$$

1. Définissez la liste `L_t` contenant les valeurs des $t_i$ pour $i \in [\![0, \texttt{Nb\_iter}]\!]$.
2. Compléter le code Python suivant pour qu'il calcule les listes `L_N` et `L_P` contenant respectivement les valeurs des $N_i$ et $P_i$ pour $i \in [\![0, \texttt{Nb\_iter}]\!]$.
   *On recopiera uniquement les lignes 14, 22 et 23 sur la copie.*

```
1  ## constantes du modele
2  r = 1
3  m = 0.75
4  C = 0.1
5  g = 0.75
6  Tmax = 40
7
8  ## valeurs initiales
9  N0 = 5
10 P0 = 5
11
12 ## choix du pas
13 Nb_iter = 1000
14 h = ## LIGNE A COMPLETER
15
```

```
## calcul des listes L_N et L_P :
L_N = [N0]
L_P = [P0]
for k in range(Nb_iter):
    N = L_N[-1]
    P = L_P[-1]
    N_next = ## LIGNE A COMPLETER
    P_next = ## LIGNE A COMPLETER
    L_N.append(N_next)
    L_P.append(P_next)
```

3. Voici les graphes obtenus lorsqu'on trace $N(t)$ et $P(t)$ en fonction de $t$ (figure 1), et $P(t)$ et fonction de $N(t)$ (figure 2).

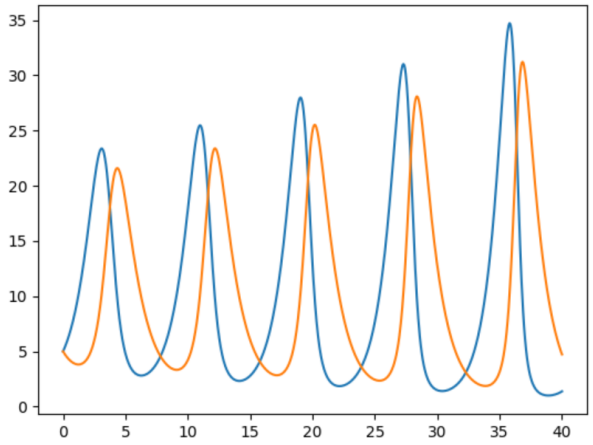


FIGURE 1 – $N(t)$ et $P(t)$ en fonction de $t$

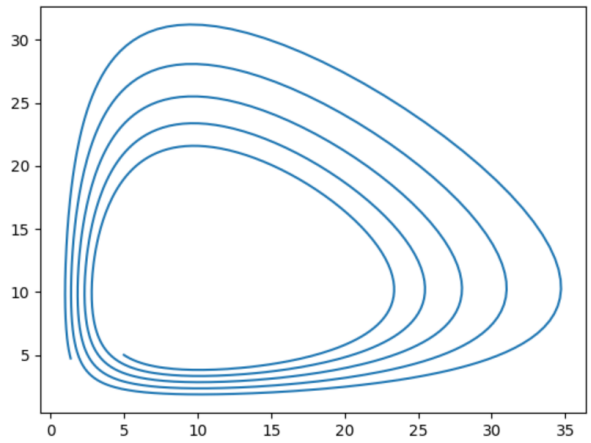


FIGURE 2 – $P(t)$ en fonction de $N(t)$

Écrire le code permettant d'obtenir ces graphes en Python.

4. Sur les graphes précédents, on constate que les valeurs maximales des populations de proies et de prédateurs augmentent à chaque oscillation. En réalité, on peut montrer que les solutions exactes du système $(S)$ sont périodiques, de sorte que les graphes obtenus devraient plutôt être les suivants :

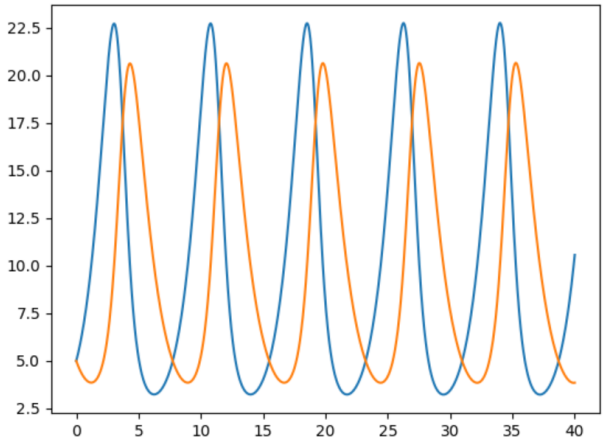


FIGURE 3 – $N(t)$ et $P(t)$ en fonction de $t$

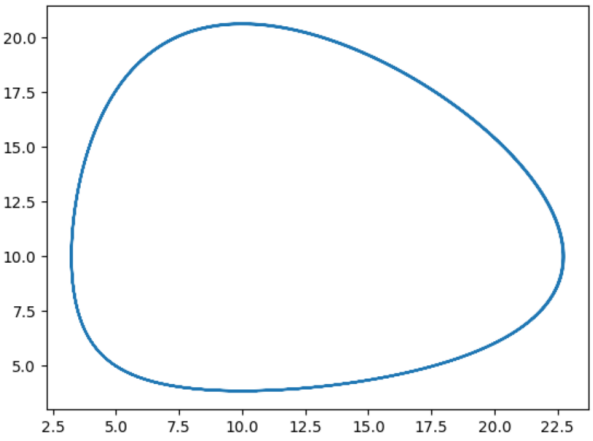


FIGURE 4 – $P(t)$ en fonction de $N(t)$

Que peut-on modifier dans le script Python pour obtenir une meilleure approximation de la solution exacte ?

**Exercice 3.**

1. Écrire une fonction `binom` prenant en arguments $n$ et $p$ et simulant une variable aléatoire de loi binomiale $\mathcal{B}(n,p)$.
2. Écrire une fonction `maximum` prenant en argument une liste de nombres `L` et renvoyant son plus grand élément. *On ne pourra pas utiliser la fonction `max` de Python.*

Dans la suite de l'exercice, on fixe un entier $N \in \mathbb{N}^*$ et on considère $N$ variables aléatoires $X_1, X_2, \ldots, X_N$ mutuellement indépendantes et de loi binomiale $\mathcal{B}(n,p)$. On s'intéresse à la variable aléatoire $Y = \max(X_1, X_2, \ldots, X_N)$.

3. Écrire une fonction `maxi_alea` prenant en arguments $N$, $n$ et $p$ et simulant $Y$.
4. Écrire un programme Python permettant de déterminer la valeur de $k$ pour laquelle la probabilité $\mathbb{P}(Y = k)$ est la plus grande possible.