Informatique - TD 5 | Méthode d'Euler d'approximation d'une équ. diff.

But

On s'intéresse à des équations différentielles d'ordre 1 *(non nécessairement linéaires) sur des intervalles* I*, c'est-à-dire des équations qui relient des quantités* $y'(t), t$ *et* $y(t)$ *où :*

- $t \in I \subset \mathbb{R}$
- y *une fonction* $\mathcal{C}^1(I)$

et où on cherche à déterminer l'allure du graphique.

On sait (théoriquement) d'après le cours, résoudre des équations simples du type

$$y'(x) = u(x)y(x) + v(x) \qquad \text{avec } x \in I \text{ et } u, v \in \mathcal{C}^0(I). \tag{1}$$

Nos limites de résolution sont toutefois très rapidement atteintes si par exemple la formule de la primitive à chercher dans la MVC s'avère compliquée à trouver. De plus, qu'advient-il si la situation se complique et que l'équation n'est plus linéaire ?
Deux exemples au hasard :

$$y'(x) = \sqrt{y(x)} - x\ln(y(x)) \quad \text{avec } I = \mathbb{R}^+, \quad \text{ou} \quad y'(x) = e^{-2y(x)} - \sqrt{x} \text{ avec } I = \mathbb{R}^+$$

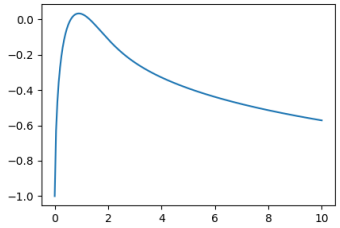
Il existe de nombreuses équations différentielles de ce type quand on essaie de décrire les évolutions de population. On évoquera d'ici la fin du TP trois modèles qui sont à votre programme de mathématiques dans le cadre des "équations autonomes" (à voir un peu plus tard) : le modèle malthusien, le modèle logistique de Verhulst, et le modèle de Gompertz.

Méthode

De manière générale et sous de bonnes conditions que nous n'évoqueront pas ici, il existe un théorème (le théorème de Cauchy-Lipschitz) qui nous assure qu'étant donné une condition initiale $y(x_0) = y_0$, il **va exister une solution à l'équation** et que de surcroit, celle-ci sera **unique.** À défaut de pouvoir la calculer, on souhaite donc pouvoir en faire une approximation numérique et graphique. Par exemple, pour l'équation

$$y'(x) = e^{-2y(x)} - \sqrt{x}$$

de solution initiale $y(0) = -1$, on obtiendrait le graphique ci-dessous :



Une des méthodes les plus simples est pour ceci la méthode dite "d'Euler", qui est à votre programme, et dont l'idée globale consiste à approcher une dérivée par son taux d'accroissement. Voyons le principe d'une telle approximation ci-dessous :

On sait que si une fonction g est dérivable en $a \in I$, alors

$$\lim_{x \to a} \frac{g(x) - g(a)}{x - a} = g'(a)$$

Supposons donc donné x très proche de a. On aurait alors

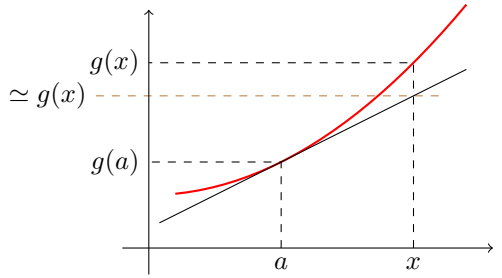
$$\frac{g(x) - g(a)}{x - a} \simeq g'(a) \tag{2}$$

(i.e. le taux d'accroissement en a *est une approximation de la dérivée.)*

et de manière équivalente

$$g(x) \simeq g'(a)(x - a) + g(a) \tag{3}$$

Ce qui se traduit graphiquement de la manière suivante :



On observe alors que, pour que l'approximation soit bonne, il vaut mieux que x soit le plus proche possible de a...

1. Comprendre la méthode sur un exemple

Les calculs dans cette question se font "à la main" et non à l'aide de Python. Ils servent à décrire et assimiler la méthode d'Euler sur un exemple. Il est vivement conseillé de les faire afin de bien comprendre la problématique...)

Soit par exemple y une solution de l'équation

$$y'(x) = \sqrt{y(x)} - x\ln(y(x)), \qquad \text{avec } I = \mathbb{R}^+ \text{ et condition initiale } y(0) = 1$$

Soit $h > 0$ très petit.

a) On note pour commencer $a = 0$ et $x_1 = a + h$. Vérifier qu'avec la méthode précédente, une approximation y_1 de $y(x_1)$ est

$$y_1 = h + 1.$$

b) On note maintenant $x_2 = x_1 + h$. *(Le "nouveau a" vaut ici x_1.)* Vérifier alors que $y_2 = h(1 + \sqrt{1+h}) + 1 - h^2\ln(1+h) \simeq y(x_2)$.

La méthode d'Euler consiste à continuer sur cette lancée et à effectuer ces approximations de proche en proche afin d'approcher y sur tout un intervalle $I = [\alpha, \beta]$, comme expliqué ci-dessous de manière générale *(application sur un exemple à suivre)* :

On se donne une équation différentielle du type

$$y'(x) = f(x, y(x))$$

sur un intervalle $[\alpha, \beta]$ de condition initiale $y(\alpha) = y_0$ et $n \in \mathbb{N}^*$.

Afin de pouvoir faire un graphique approximatif de y, on se donne :

- *une valeur $h > 0$ petite appelée le "pas".*
- *une subdivision régulière $[x_0, x_1, \ldots, x_n]$ de $[\alpha, \beta]$ de pas h, (i.e. $x_0 = \alpha$, $x_n = \beta$, et $x_{i+1} - x_i = h\ \forall i$)*
- *une liste $[y_0, y_1, \ldots, y_n]$, où, pour $i > 0$, y_i serait une valeur approchée de $y(x_i)$ à déterminer, construite de la façon suivante :*

pour tout $i < n$, en admettant que l'on connait y_i, on trouve y_{i+1} en utilisant l'approximation (3) pour $g = y$, $a = x_i$ et $x = x_{i+1}(= x_i + h)$, ce qui donne

$$y(x_{i+1}) \simeq y'(x_i)(x_{i+1} - x_i) + y(x_i)$$

puis on remplace $y'(x_i)$ par sa valeur dans l'équation différentielle :

$$y'(x_i) = f\big(x_i, y(x_i)\big)$$

et pour finir, on assimile $y(x_i)$ à y_i, ce qui se résume par

$$y_{i+1} = f\big(x_i, y_i\big)h + y_i$$

Par exemple, pour l'équation différentielle avec condition initiale sur l'intervalle $[\alpha, \beta]$ suivante :

$$y'(x) = \sqrt{y(x)} - x\ln(y(x)), \qquad y(0) = 1, \quad \alpha = 0, \quad \beta = 5, \quad n = 10$$

on aurait $h = \frac{5}{10}$ et donc

$$[x_0, x_1, \ldots, x_n] = \big[0\ ,\ \tfrac{5}{10}\ ,\ 2\cdot\tfrac{5}{10}\ ,\ \ldots\ ,\ 10\cdot\tfrac{5}{10}\big] \qquad \text{avec } h = \frac{5}{10}$$

on a

$$y(x_{i+1}) \simeq y'(x_i)\underbrace{(x_{i+1} - x_i)}_{h} + y(x_i)$$

et

$$y'(x_i) = \sqrt{y(x_i)} - x_i\ln(y(x_i))$$

que l'on réinjecte dans la ligne du dessus, ce qui donne, en assimilant y_i à $y(x_i)$:

$$y_{i+1} = (\sqrt{y_i} - x_i\ln(y_i))\,h + y_i$$

De proche en proche, on obtient alors *(calcul d'approximations qui peuvent !)*

$$[y_0, y_1, \ldots, y_n] \qquad (\text{avec } y_i \simeq f(x_i) \text{ à déterminer pour } i = 1, \ldots, n.)$$

ici calculé dans la question 1 :

$$[y_0, y_1, \ldots, y_n] = [1, 1+h, h(1+\sqrt{1+h}) + 1 - h^2\ln(1+h), ...] \simeq [1, 1.5, 2.01, ...]$$

Avec n grand (ce qui n'est pas le cas dans la description ci-dessus ...!) les x_i successifs sont proches les uns des autres. Ainsi, de proche en proche (de i en $i+1$), on a donc bien $y_i \simeq y(x_i)$ et ceci valide l'utilisation de la méthode. On verra néamoins graphiquement les phénomènes observés dans la partie qui suit quand n varie.

Pour résumer

On découpe l'intervalle $[\alpha, \beta]$ en n subdivisions de largeur

$$h = \frac{\beta - \alpha}{n} > 0 \quad (\text{appelé } \textit{"pas"})$$

en posant

$$\boxed{x_k = \alpha + kh = \alpha + k\cdot\frac{\beta - \alpha}{n} \qquad \forall k = 0, \ldots, n} \tag{4}$$

et on estime que

$$\boxed{y_i \simeq y(x_i) \quad \textit{et} \quad y_{i+1} = y_i + f(x_i, y_i)\cdot h \quad \forall i = 0, \ldots, n-1} \tag{5}$$

Technique totale sur un exemple

Vérifions la pertinence de ce modèle sur un exemple simple que nous savons résoudre afin de comparer l'approximation numérique avec le modèle d'Euler. Considérons l'équation

$$y' + y = \sin x + \cos x \qquad \text{sur } I = \mathbb{R}. \tag{6}$$

2. **a)** On se place sur un intervalle $[\alpha, \beta]$ muni de la subdivision $[x_0, \dots, x_n]$ *(cf (4))*. Avec les notations de la méthode, déterminer une relation de récurrence sur les y_i adaptée à notre équation différentielle (6).

b) En déduire une fonction Python `Approx(alpha,beta,n,y0)` rendant :
- $[x_0, \dots, x_n]$;
- et $[y_0, \dots, y_n]$;

pour l'équation différentielle (6) sur un intervalle $[\alpha, \beta]$ de condition initiale $y(\alpha) =$ `y0`.
(Testez votre fonction `Approx` avec les valeurs que vous souhaitez et vérifiez que vous obtenez bien `X=[alpha,...,beta]` et `Y=[y0,...]` avec `X` et `Y` de même taille $n+1$ avant de continuer.)

3. **a)** En déduire une fonction Python `DessineSol(alpha,beta,n,y0)` qui dessine l'approximation ainsi obtenue de la solution à l'équation (6) de condition initiale $y(\alpha) = y0$ sur la subdivision $[x_0, \dots, x_n]$. (Rajouter éventuellement une légende permettant d'identifier le "n" de la courbe.)

b) Appliquer à l'équation

$$y' + y = \sin x + \cos x \tag{7}$$

de solution initiale $y(0) = 0$ sur l'intervalle $[0, 10]$ avec $n = 100$.

Confrontation d'un exemple à des solutions exactes

Sur l'exemple précédent, nous allons maintenant confronter l'approximation d'Euler avec la solution explicitement calculée afin d'évaluer graphiquement la précision de la méthode.

On considère toujours l'équation (6)

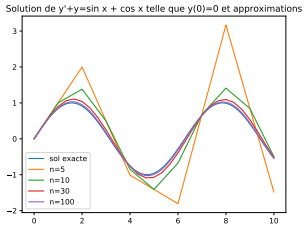
4. Les solutions de cette équation (n'hésitez pas à le calculer vous même !) sont

$$S = \left\{ y \in \mathbb{R} \mapsto x \mapsto \lambda e^{-x} + \sin x \mid \lambda \in \mathbb{R} \right\}$$

Déterminer en fonction de y_0 la formule de la solution φ de condition initiale $\varphi(\alpha) = y_0$.

5. Soit `L` une liste d'entiers n non nuls. En s'inspirant de ce qui été fait dans la partie précédente, réaliser un programme Python de paramètres `y0,alpha,beta,L` permettant de mettre sur un même graphique
- la solution φ de (6) sur $[\alpha, \beta]$ de condition initiale $\varphi(\alpha) = y_0$
- Pour chaque $n \in$ `L`, les solutions approchées sur l'intervalle $[\alpha, \beta]$ de conditions initiales $y(\alpha) = y_0$, pour une subdivision $[x_0, \dots, x_n]$.

6. Tester avec `L=[5,10,30,100]`, $[\alpha, \beta] = [0, 10]$ et $y(0) = 0$. On doit obtenir ce type de graphique :



Qu'en conclure quant au n ?

À retenir : Vous devez être capable de réitérer cette méthode sur un exemple si elle vous est demandée. La technique devra toutefois être rappelée (au moins brièvement) dans l'énoncé.

7. Vitesse d'exécution

On souhaite ici étudier l'impact de l'augmentation de n sur le temps d'exécution de la fonction de résolution `"Approx"`. Posons pour l'instant :
- $N = 50$ (à augmenter ou diminuer suivant la vitesse d'exécution de votre ordinateur...)
- une liste de n valant `Liste_n=[0,10,20,..., 490,500]`.

On rappelle que dans le module `time` se trouve la fonction `time()` qui rend le temps écoulé depuis un moment fixe précis. Utiliser cette fonction pour créer le graphique des points reliés (n, "temps de N exécutions de la fonction `Approx` pour n") avec $n \in$ `Liste_n` et des α, β, y_0 évidemment communs.
On conidère ici par convention que le temps mis pour $n = 0$ est 0.
Semble-t-il y avoir un rapport entre n et le temps d'éxécution ?

Confrontation à des solutions obtenues par un modèle de Python

Dans le module `scipy.integrate`, il existe deux fonctions (suivant l'ancienneté de votre version) permettant de donner des approximations de solutions d'équations différentielles, chacune basée sur la donnée du "f", de $[\alpha, \beta]$, de la condition intiale, et du n (ou du pas.)
La première fonction s'appelle `solve_ivp` *(la plus récente)* et la deuxième `odeint`.
Voyons comment elles fonctionnent sur l'exemple précédemment utilisé dans les conditions de l'équation (7) :

$$y' + y = \sin x + \cos x \qquad \text{sur } [\alpha, \beta] = [0, 10], \text{ avec } y(0) = 0.$$

où, suivant les notations utilisées dans le TP, on a

$$f(x, y) = \sin x + \cos x - y$$

car $y' = \sin x + \cos x - y$. (f est la partie de droite de l'équation différentielle $y' = \dots$)

Tout d'abord, point commun des 2 méthodes : les données de $[\alpha, \beta]$ et de y_0 :

```
alpha=0
beta=10
y0=0
```

Ensuite, la donnée du "f", qui est pratiquement identique, mais **attention**, il y a une inversion des arguments de f dans le cas `odeint` par rapport au cas`solve_ivp`!!

```
# Pour solve_ivp :
def f(x,y):
    return(-y+np.cos(x)+np.sin(x))
```

```
# Pour odeint :
def f(y,x):
    return(-y+np.cos(x)+np.sin(x))
```

La donnée des x_i est également différente : Pour `solve_ivp`, on donne seulement la taille du pas, la fonction s'occupera du reste :

```
pas=0.1 # rappel : c'est le "h"
```

Pour `odeint`, on donne directement la liste des x_i (pour obtenir le même résultat que précédemment :

```
nb_points=101
X=np.linspace(alpha,beta,nb_points)
# ou même résultat
n= 100
pas=(beta-alpha)/n # pour obtenir ici un pas de 0.1
X=[alpha + k*h for k in range(n+1)]
```

et pour finir, on demande la liste des Y solutions approximatives :

```
# Pour solve_ivp :
solution = solve_ivp(f, [alpha,beta], [y0],
    max_step=pas)

X=solution.t    # donne la liste [x0,...,xn]
Y=solution.y[0] # dpnne la liste [y0,...,yn]
```

```
# Pour odeint :
y = odeint(f, y0, X) # donne le tableau des [[y0
    ,...,yn]]
```

8. Après avoir obervé le résultat d'une (ou des deux) méthodes précédente, choisir une de ces méthodes et effectuer le même graphique que dans la question), c'est-à-dire un graphique comparatif des approximations pour `L=[5,10,30,100]` avec la solution exacte. Est-ce ça semble mieux ou moins bien que la méthode d'Euler ?

Application : Des modèles de croissance de population à une variable

Dans cette partie, on étudie divers modèles de croissance de population à l'aide de la méthode d'Euler ou de toute autre méthode de votre choix ici. On notera $N(t)$ le nombre d'individus présents dans une population à un instant $t \geq 0$ donné. Le but est de donner ici des courbes correspondant aux trois modèles donnés.

Modèle de Malthus

Dans ce modèle attribué à Thomas Malthus, on considère une population isolée et sans contrainte, au sens suivant : espace et nourriture illimités, absence de prédateur, resistance aux maladies, etc.... On note r le "taux de croissance" de l'espèce (différence entre le taux de natalité et le taux de mortalité).

Dans ce modèle, on estime que

$$N'(t) = rN(t) \quad \forall t \geq 0$$

(phénomène décrit : la vitesse d'augmentation de la population est proportionnelle à la quantité d'individus.)
On obtiendra un graphique du type suivant :

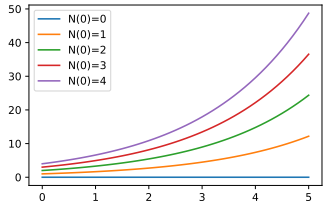
9. Construire un algorithme qui aboutit à une fonction `Malthus(r,beta,pas,N0)` ou `Malthus(r,beta,n,N0)` où

- `r` est le taux de croissance du modèle
- $\alpha = 0$ est le moment initial commun
- `N0` est une liste de $N(0)$ initiaux

de manière à ce que la fonction dessine l'ensemble des courbes approchant les solutions N sur $[0, \beta]$ de conditions initiales $N(0) = N[i]$.

Visualisez le phénomène de croissance exponentielle en appliquant la fonction à $r = 0.5, \beta = 5$ avec un pas de $0,1$ ou $n = 50$, pour `N0=[0,1,...,4]`.

On obtient :



Pour rappel : Les solutions de cette équation différentielle sont simples à trouver d'après le cours. Il s'agit de l'ensemble des fonctions

$$N : t \geq 0 \mapsto N(0)e^{rt}$$

Le modèle logistique

Dans ce modèle attribué à Verhlust en 1836, on considère une population isolée avec une contrainte environnementale, au sens suivant : nourriture illimités, absence de prédateur, resistance aux maladies, etc...mais espace limite. On note r le "taux de croissance" de l'espèce et K la capacité d'accueil du milieu.
Dans ce modèle, on estime que

$$N'(t) = r\left(1 - \frac{N(t)}{K}\right)N(t) \quad \forall t \geq 0$$

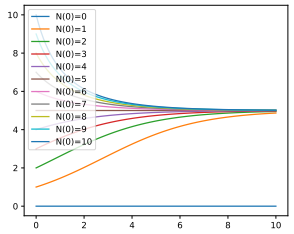
une partie du phénomène décrit : $1 - \frac{N(t)}{K}$ est proche de 0 quand $N(t)$ se rapproche de la capacité d'accueil. Ainsi, $N'(t)$ est proche de 0, c'est-à-dire que la population a tendance à stagner.

10. De même que pour le modèle de Logistique, construire un algorithme qui aboutit à une fonction `Logistique(r,K,beta,pas,NO)` ou `Logistique(r,K,beta,n,NO)` où

- `r` est le taux de croissance du modèle
- K est la capacité d'accueil du milieu
- $\alpha = 0$ est le moment initial commun
- `NO` est une liste de $N(0)$ initiaux

de manière à ce que la fonction dessine l'ensemble des courbes approchant les solutions N sur $[0, \beta]$ de conditions initiales $N(0) = N[i]$.
Visualisez que le phénomène de limite de capacité du milieu est bien modélisé ici. en prenant par exemple $r = 0.5$, $K = 5, \beta = 10$ avec un pas de $0,1$ ou $n = 50$, pour `NO=[0,1,...,10]`.

On doit obtenir :



Remarque : les solutions de cette équation différentielle peuvent être trouvée (cf futur cours en maths). Il s'agit de l'ensemble des fonctions

$$N : t \geq 0 \mapsto \frac{K}{1 - \left(1 - \frac{K}{N(0)}\right)e^{-rt}}$$

Le modèle de Gompertz

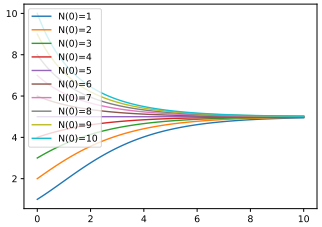
Gompertz introduit en 1825 la notion d'évolution du taux de mortalité avec l'âge avançant, avec capacité du milieu limitée. Avec les mêmes notations que le modèle logistique, Il introduit l'équation :

$$N'(t) = r \ln\left(\frac{K}{N(t)}\right)N(t) \quad \forall t \geq 0$$

11. De même que pur les deux autres modèles, construire un algorithme qui aboutit à une fonction `Gompertz(r,K,beta,pas,NO)` ou `Malthus(r,beta,n,NO)` où

- `r` est le taux de croissance du modèle
- K est la capacité d'accueil du milieu
- $\alpha = 0$ est le moment initial commun
- `NO` est une liste de $N(0)$ initiaux

de manière à ce que la fonction dessine l'ensemble des courbes approchant les solutions N sur $[0, \beta]$ de conditions initiales $N(0) = N[i]$.
Visualisez là aussi que le phénomène de limite de capacité du milieu est bien modélisé ici. en prenant par exemple $r = 0.5$, $K = 5, \beta = 10$ avec un pas de $0,1$ ou $n = 50$, pour `NO=[0,1,...,10]`.
On doit obtenir :



Application à des systèmes et à un modèle proies/prédateurs

Cette partie est à faire par les 5/2 ou par les plus à l'aise uniquement

On pose $I = [\alpha, \beta]$ un intervalle. On considère maintenant f, g deux fonctions connues $\mathcal{C}^1$ sur $\mathbb{R}^2$ ainsi que deux fonctions $u, v \in \mathcal{C}^1(I, \mathbb{R})$ que l'on cherche à déterminer, qui vérifient le système d'équation :

$$\mathcal{S} : \begin{cases} u'(t) = f(u(t), v(t)) \\ v'(t) = g(u(t), v(t)) \end{cases}$$

12. Avec des conditions initiales $u(\alpha) = u_0$ et $v(\alpha) = v_0$, en vous inspirant des techniques précédentes, proposer une méthode pour représenter les approximations des courbes représentatives des fonctions u, v pout $t \in [\alpha, \beta]$.

13. On considère maintenant le système de Lotka-Volterra qui modélise de manière très simplifiée un modèle "Proie-Prédateur" dans lequel une proie (par exemple des moutons) et un prédateur (par exemple un loup) interagissent indépendamment de toute action extérieure :

$$\mathcal{S} : \begin{cases} u'(t) &= a_1 u(t) - b_1\, u(t)v(t) \\ v'(t) &= -a_2 v(t) + b_2\, u(t)v(t) \end{cases}$$

où :
- t est le temps,
- $x(t), y(t)$ les effectifs respectifs des proies et des prédateurs au temps t
- $a_1 > 0$ et $-a_2 < 0$ sont les taux d'évolution naturelle respectivement des proies et des prédateurs
- $b_1, b_2 > 0$ des coefficients d'interaction entre les proies et les prédateurs.

a) Proposer une fonction Python permettant de faire le graphique des courbes d'effectif des proies et des prédateurs sur un intervalle de temps $[0, \beta]$.

b) Tester ce programme avec $u_0 = v_0 = 1$, $a_1 = a_2 = 0.5$, $b_1 = b_2 = 1$ et $\beta \geqslant 30$, puis faire varier ces paramètres. phénomène d'alternance cyclique entre l'augmentation des proies / diminution des prédateurs et réciproquement.

FE 5 - Méthode d'Euler d'approximation d'une équ. diff. Solutions

1. **a)** D'après l'approximation (3), on a

$$\begin{aligned} y(x_1) &\simeq y'(a)(x_1-a)+y(a) \\ &\simeq \left(\sqrt{y(a)}-a\ln(y(a))\right)(x_1-a)+y(a) \\ &\simeq \left(\sqrt{1}-0\ln(1)\right)h+1 \\ &\simeq h+1 \end{aligned}$$

D'où

$$\boxed{y_1 = h+1}$$

b) D'après l'approximation (3), on a

$$\begin{aligned} y(x_2) &\simeq y'(x_1)(x_2-x_1)+y(x_1) \\ &\simeq \left(\sqrt{y(x_1)}-x_1\ln(y(x_1))\right)h+y(x_1) \\ &\simeq (\sqrt{y_1}-x_1\ln(y_1))\,h+y_1 \\ &\simeq \left(\sqrt{1+h}-h\ln(1+h)\right)h+1+h \\ &\simeq h(1+\sqrt{1+h})+1-h^2\ln(1+h) \end{aligned}$$

D'où

$$\boxed{y_2 = h(1+\sqrt{1+h})+1-h^2\ln(1+h)}$$

2. **a)** On a $x_{i+1}-x_i = \frac{\beta-\alpha}{n}$ par hypothèse sur la subdivision. Ainsi, en réinjectant dans les relations (5), on obtient

$$y_{i+1} \simeq y_i + f'(x_i)\frac{\beta-\alpha}{n}$$

D'où, en réinjectant l'équation différentielle pour $x = x_i$:

$$y'(x_i) = -y(x_i)+\sin x_i+\cos x_i = \cos x_i - y_i$$

avec l'approximation $y_i = y(x_i)$, on a

$$\boxed{y_{i+1} \simeq y_i + (\sin x_i + \cos x_i - y_i)\frac{\beta-\alpha}{n} \quad \forall i = 0,\dots n-1}$$

b) On peut proposer

```
def Approx(alpha,beta,n,y0):
    '''Rend la liste des n approximations des
       abscisses et ordonnées des solutions
       obtenues par la méthode d'Euler de l'
       équation y'+y=cos(x)+sin(x) sur l'intervalle
        [alpha;beta] pour la condition initiale y
       (0)=y0'''
    h=(beta-alpha)/n
    X=[alpha+k*h for k in range(n+1)]
    Y=[y0]
    for k in range(n):
        y=Y[k]+h*(np.cos(X[k])+np.sin(X[k])-Y[k])
        Y+=[y]
    return(X,Y)
```

ou "plus joli" :

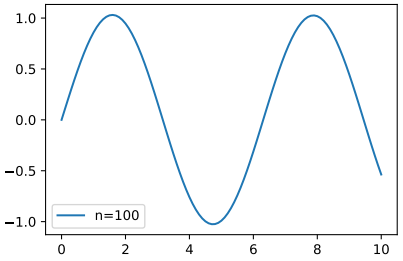
```
def Approx2(alpha,beta,n,y0):
    X=np.linspace(alpha,beta,n+1)
    Y=[y0]
    h=(beta-alpha)/n
    for x in X[:-1]:# sinon, il y a un "y" en trop
       à la fin.
        Y+=[Y[-1]+h*(-Y[-1] + np.cos(x) + np.sin(x)
           )]
    return(X,Y)
```

3. **a)** Une solution serait de faire

```
def DessineSol(alpha,beta,n,y0):
    '''Dessine l'approximation de la solution de l'
       équation y'+y=cos(x)+sin(x) sur l'intervalle
        [a;b] pour la condition initiale y(0)=y0
       obtenues par la méthode d'Euler pour un pas
       de h=(b-a)/n '''
    plt.close('all')
    [X,Y]=Approx(alpha,beta,n,y0)
    plt.plot(X,Y,label='n='+str(n))
    plt.legend()
    plt.show()
```

```
alpha=0
beta=10
y0=0
n=100
DessineSol(alpha,beta,n,y0)
```

b) On obtient :



4. Soit $y \in \mathcal{S}$. Alors il existe $\lambda \in \mathbb{R}$ tel que

$$y(x) = \lambda e^{-x} + \sin x$$

$$\begin{aligned} y(\alpha) = y_0 &\Leftrightarrow \lambda e^{-\alpha} + \sin\alpha = y_0 \\ &\Leftrightarrow \lambda = (y_0 - \sin\alpha)\, e^{\alpha} \end{aligned}$$

La solution de condition initiale $y(0) = y_0$ est donc

$$y(x) = (y_0 - \sin\alpha)\, e^{\alpha} e^{-x} + \sin x$$

i.e.

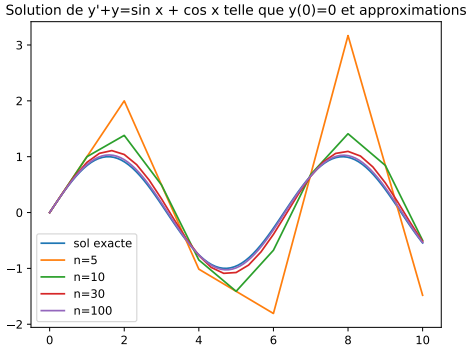
$$\boxed{y(x) = (y_0 - \sin\alpha)\, e^{a-x} + \sin x}$$

5. On propose la suite de programmes suivants :

```
def phi(x,y0,a):
    return((y0-np.sin(a))*np.exp(a-x) + np.sin(x))

def Compare_exacte_approx(y0,alpha,beta,L):
    plt.close('all')

    # graphe de la solution exacte :
    X=np.linspace(alpha,beta,100)
    plt.plot(X,phi(X,y0,alpha),label='sol exacte')

    # Les approximations :
    for n in L:
        [X,Y]=Approx(alpha,beta,n,y0)
        plt.plot(X,Y,label='n='+str(n))

    # dessiner les graphes obtenus :
    plt.legend(loc='best')
    plt.title("Solution de y'+y=sin x + cos x telle
        que y("+str(alpha)+")="+str(y0)+" et
       approximations")
    plt.show()
```

6. Avec

```
alpha=0
beta=10
y0=0

L=[5,10,30,100]
Compare_exacte_approx(y0,alpha,beta,L)
```



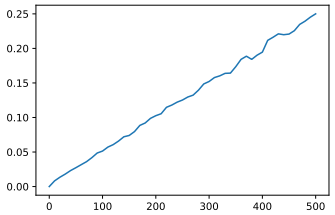
le graphique est le suivant : On constate (logiquement !) que plus le pas diminue, plus la précision est importante.

7. Voici un exemple obtenu avec

```
from time import time

N=50
Liste_n=[10*k for k in range(51)] # va de 10 en 10
    jusqu'à 500

alpha=0
beta=10
y0=0
```

```
    Temps=[0]

    for n in Liste_n[1:]:
        # pour chaque n on exécute N fois Approx
        temps_initial=time()
        for _ in range(N):
            Approx(alpha,beta,n,y0)
        temps_final=time()
        temps_mis=temps_final-temps_initial
        Temps+=[temps_mis]
    print(Temps)

    #

    # Graphique :
    plt.close('all')
    plt.plot(Liste_n,Temps)
    plt.show()
```

On obtient :

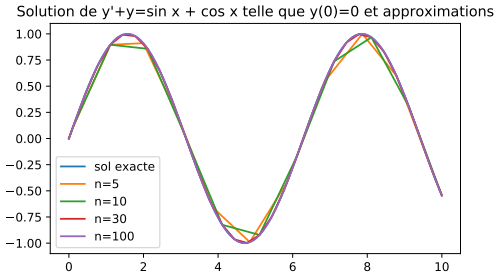


Il semblerait que le temps mis soit proportinnel à n.

8.

```
def phi(x,y0,a):
    return((y0-np.sin(a))*np.exp(a-x) + np.sin(x))

def f(x,y):
    return(-y+np.sin(x)+np.cos(x))


def Compare_exacte_solve(y0,alpha,beta,L):
    plt.close('all')

    # graphe de la solution exacte :
    X=np.linspace(alpha,beta,100)
    plt.plot(X,phi(X,y0,alpha),label='sol exacte')

    # Les approximations :
    for n in L:
        pas=(beta-alpha)/n
        solution = solve_ivp(f, [alpha,beta], [y0],
            max_step=pas)

        # Récupération des solutions :
        X=solution.t
        Y=solution.y[0]
        plt.plot(X,Y,label='n='+str(n))

    # dessiner les graphes obtenus :
    plt.legend(loc='best')
    plt.title("Solution de y'+y=sin x + cos x telle
        que y("+str(alpha)+")="+str(y0)+" et
       approximations par module")
    plt.show()

alpha=0
beta=10
y0=0

L=[5,10,30,100]

Compare_exacte_solve(y0,alpha,beta,L)
```

Le résultat est le suivant :



ça a l'air bien plus précis que la méthode d'Euler !

9.

```
def fMalthus(x,y):
    return(r*y)

def Y_Malthus(r,y0,beta,pas):
```

```
    '''Rend les [X,Y] du modèle de Malthus pour le
       taux de croissance r et la condition
       initiale y0.'''
    # résolution :
    solution = solve_ivp(fMalthus, [0,beta], [y0],
       max_step=pas)
    X=solution.t
    Y=solution.y[0]
    return(X,Y)

def Malthus(r,beta,pas,N0):
    '''Dessine un ensemble de courbes de conditions
        initiales dans Y0'''
    plt.close('all')
    # les courbes une par une :
    for y0 in N0:
        [X,Y]=Y_Malthus(r,y0,beta,pas)
        plt.plot(X,Y,label='N(0)='+str(y0))
    plt.legend(loc='best')
    plt.show()

##
r=0.5
beta=5
pas=0.1
N0=[i for i in range(5)]
Malthus(r,beta,pas,N0)
```

10. On peut faire un copié collé de ce qu'il y a écrit pour Malthus en remplaçant "Malthus" par "Logistique" puis "Gompertz" et en utilisant successivement :

```
def fLogistique(x,y):
    return(r*(1-y/K)*y)

def fGompertz(x,y):
    return(r*np.log(K/y)*y)
```

mais on peut aussi, plutôt que de faire des copiés-collés, établir une fonction générale, qui fait tout ceci quelquesoit l'équation de départ, définie par une fonction f :

```
def Y_Approche(y0,f,beta,pas):
    '''Rend les [X,Y] approchés d'une équation y'=f
       (x,y).'''
    # résolution :
    solution = solve_ivp(f, [0,beta], [y0],
       max_step=pas)
    X=solution.t
    Y=solution.y[0]
    return(X,Y)

def Plusieurs_cdt_initiales(f,beta,pas,N0):
    '''Dessine un ensemble de courbes de conditions
        initiales dans Y0'''
    plt.close('all')
    # les courbes une par une :
    for y0 in N0:
        [X,Y]=Y_Approche(y0,f,beta,pas)
        plt.plot(X,Y,label='N(0)='+str(y0))
    plt.legend(loc='best')
    plt.show()
```

puis avec le modèle logistique :

```
K=5
beta=10
N0=[i for i in range(11)]

Plusieurs_cdt_initiales(fLogistique,beta,pas,N0)
```

11. Avec les mêmes données que dans la question précédente puis

```
K=5
beta=10
N0=[i for i in range(1,11)]

Plusieurs_cdt_initiales(fGompertz,beta,pas,N0)
```

12. De la même manière que précédemment, on peut découper l'intervalle $[\alpha,\beta]$ en petits pas : $[t_0,\dots,t_n]$ où $t_k=\alpha+k\frac{\beta-\alpha}{n}$ et n grand.
De la même façon également, comme le pas $h=\frac{\beta-\alpha}{n}$ est très petit, on peut tout à fait considérer que

$$u'(t_i)\simeq\frac{u(t_{i+1})-u(t_i)}{h}\quad;\quad v'(t_i)\simeq\frac{v(t_{i+1})-v(t_i)}{h}$$

D'où

$$u(t_{i+1})\simeq u(t_i)+hu'(t_i)\quad;\quad v(t_{i+1})\simeq v(t_i)+hv'(t_i)$$

et en utilisant ensuite le système d'équation $\mathcal{S}$:

$$u(t_{i+1})\simeq u(t_i)+hf(u(t_i),v(t_i))\quad;\quad v(t_{i+1})\simeq v(t_i)+hg(u(t_i),v(t_i))$$

Ainsi, on peut proposer de placer dans $\mathbb{R}^2$ les points (t_i, x_i) et (t_i, y_i) approximations des courbes u, v en les construisant par récurrence avec la formule

$$\boxed{\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \begin{pmatrix} x_i + hf(x_i, y_i) \\ x_i + hg(x_i, y_i) \end{pmatrix} = \begin{pmatrix} x_i + (i+1)\frac{\beta-\alpha}{n} f(x_i, y_i) \\ x_i + (i+1)\frac{\beta-\alpha}{n} g(x_i, y_i) \end{pmatrix} \quad \forall i \in [\![0, n-1]\!]}$$

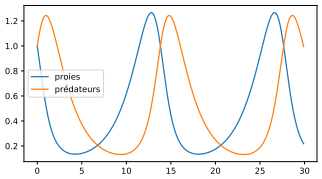
13. **a)** Dans ce cas, on a

$$f(u,v) = a_1 u - b_1\, uv, \quad g(u,v) = -a_2 v + b_2\, uv$$

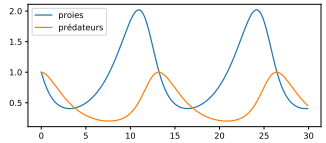
On peut donc proposer la fonctions suivante :

```
1  def LV(u0,v0,a1,a2,b1,b2,beta):
2      u=u0 # valeur des abscisses
3      v=v0 # valeur des ordonnées
4
5      U=[u] # liste des ordonnées de u
6      V=[v] # liste des ordonnées de v
7
8      n=int(beta*10) # on construit un grand n
9      h=beta/n    # création du pas
10     T=[0] # liste des abscisses
11
12     # création des listes U et V avec la formule de
           récurrence
13     for i in range(1,n):
14         u=u+h*(a1*u-b1*u*v)
15         v=v+h*(-a2*v+b2*u*v)
16         U+=[u]
17         V+=[v]
18         T+=[h*i]
19
20     # affichage du graphique avec légende
21     plt.close('all')
22     plt.figure('Lotka-Volterra')
23     plt.plot(T,U,label='proies')
24     plt.plot(T,V,label='prédateurs')
25     plt.legend()
26     plt.show()
```

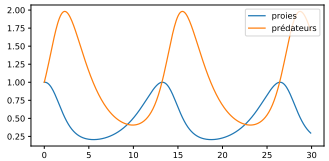
b) Voici le graphique obtenu pour les conditions demandées :



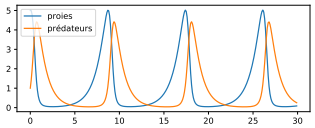
Si on modifie maintenant seulement le paramètre $b_2 = 0.5$, on obtient :



ou alors si on modifie seulement le paramètre $b_1 = 0.5$, on obtient :



Dans les trois cas, on note une périodicité du phénomène. Quand les prédateurs augmentent, les proies diminuent logiquement. Si les proies ne sont plus assez nombreuses, le prédateurs sont soumis de plus en plus à leur seule extinction naturelle et donc diminuent.
De ce fait, les proies, voyant leur prédateur disparaitre, sont de plus en plus soumis à leur natalité naturelle et donc réaugmentent et ainsi de suite. Le phénomène ne change pas si on modifie le nombre initial de proies et de prédateurs, par exemple, si on garde $b_1 = b_2 = 1$, $a_1 = a_2 = 0.5$ mais que l'on demande $u_0 = 5$, $v_0 = 1$ (5 fois plus de proies que de prédateurs au début :



ou le contraire :