Centre d'intérêt 6

Cahier de révisions

PSI - MP : Lycée Rabelais

Méthodes numériques

Centre d'intérêt 6

Dérivation numérique

PSI - MP : Lycée Rabelais

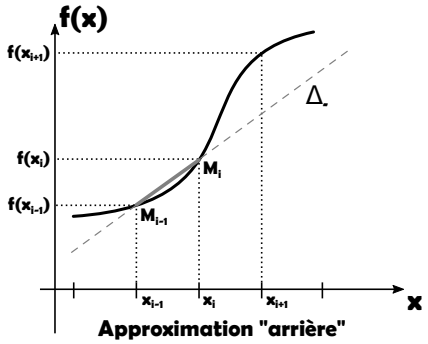
On cherche à dériver numériquement une fonction f. Cela revient calculer, de manière approximée, la pente de la courbe représentative de la fonction.

Il existe plusieurs méthodes d'approximation dites "arrière" ou "avant". On discrétise l'axe des abscisses (variable x) et on note dx un "petit" incrément sur l'axe des abscisses de telle sorte que $x_{i+1} - dx = x_i = x_{i-1} + dx$. On note également M, un point de la courbe étudiée ayant pour coordonnées $(x, f(x))$.

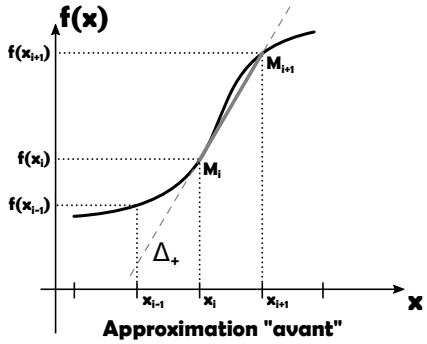
Dériver la fonction f numériquement revient à calculer, pour chaque x_i, la pente de la droite :

- Δ_- entre les points M_{i-1} et M_i (approximation "arrière") ;
- Δ_+ entre les points M_{i+1} et M_i (approximation "avant").

On a donc :

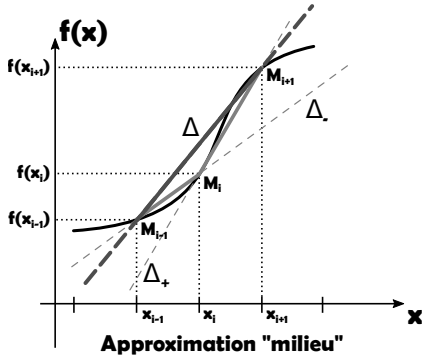


$$\frac{df}{dx}(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{dx}$$



$$\frac{df}{dx}(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{dx}$$

Une autre méthode consiste à calculer une pente moyenne entre M_{i-1} et M_{i+1}. Dans ce cas, on aura :



$$\begin{aligned}\frac{df}{dx}(x_i) &\approx \frac{1}{2}\left[\frac{f(x_i) - f(x_{i-1})}{dx} + \frac{f(x_{i+1}) - f(x_i)}{dx}\right] \\ &\approx \frac{f(x_{i+1}) - f(x_{i-1})}{2 \cdot dx}\end{aligned}$$

Exercice d'application

On définit la fonction f telle que : $f(x) = \cos(x)$. On veut calculer et tracer la dérivée de la fonction f sur l'intervalle $[0, \frac{\pi}{2}]$.

Question 1. Définir la fonction `f(x)`.

Question 2. Représenter cette fonction pour sur l'intervalle $[0, \frac{\pi}{2}]$. On pourra s'aider des instructions ci-dessous :

```
import numpy as np
import matplotlib.pyplot as plt

a = 0
b = np.pi/2
n = 100 ## nombre de subdivisions

def f(x):
    ....

lx =  [i*dx for i in range(0, ..... )]   ## lx est la liste des xi

lf = [..... for xi in lx]

plt.plot(.... , ....)
```

Question 3. Écrire les instructions permettant de calculer et tracer la dérivée de la fonction f par :

- la méthode dite "arrière" ;
- la méthode dite "avant" ;
- la méthode dite "milieu".

Centre d'intérêt 6

Intégration numérique

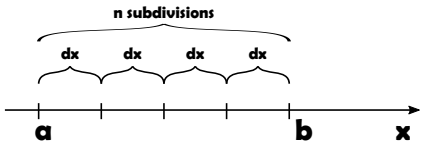
PSI - MP : Lycée Rabelais

On cherche à intégrer numériquement une fonction f sur un intervalle $[a, b]$. Cela revient donc à calculer l'aire sous la courbe représentant la fonction. On cherche donc I telle que :

$$I = \int_a^b f(x).dx$$

Il existe plusieurs méthodes d'approximation. On notera dans la suite n, le nombre de subdivision de l'intervalle $[a, b]$ et dx le "petit" segment résultant de cette subdivision. On a donc :

$$dx = \frac{b-a}{n}$$

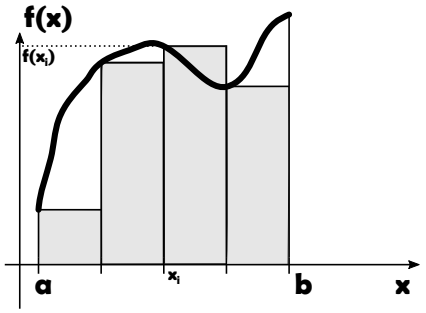


1 Méthode des rectangles

1.1 Méthode des rectangles "à gauche" ou "arrière"

Elle revient à réaliser l'approximation suivante :

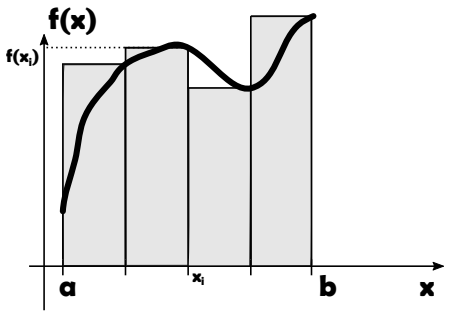
$$I \approx I_g \approx \sum_{i=0}^{n-1} f(x_i).dx \quad \text{avec } x_i = a + i.dx$$



1.2 Méthode des rectangles "à droite" ou "avant"

Elle revient à réaliser l'approximation suivante :

$$I \approx I_d \approx \sum_{i=1}^{n} f(x_i).dx \quad \text{avec } x_i = a + i.dx$$

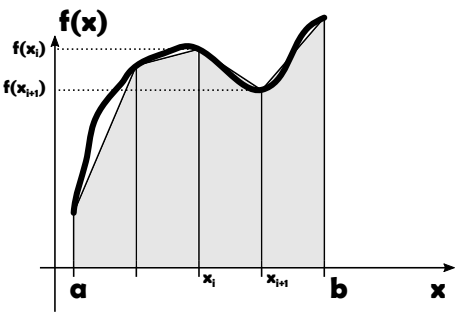


2 Méthode des trapèzes

Dans cette approximation, on a :

$$I \approx I_t \approx \sum_{i=0}^{n-1} \frac{f(x_i) + f(x_{i+1})}{2} \cdot dx \quad \text{avec } x_i = a + i.dx$$

On pourra remarquer que $I_t = \frac{I_g + I_d}{2}$



Exercice d'application

On définit la fonction f telle que : $f(x) = (\cos(x))^x$. On veut calculer l'intégrale I telle que :

$$I = \int_0^{\frac{\pi}{2}} f(x).dx$$

Question 1. Définir la fonction `f(x)`.

Question 2. Représenter cette fonction pour sur l'intervalle $[0, \frac{\pi}{2}]$. On pourra s'aider des instructions ci-dessous :

```
import numpy as np
import matplotlib.pyplot as plt

a = 0
b = np.pi/2
n = 100 ## nombre de subdivisions

def f(x):
    ....

lx =  [i*dx for i in range(0, ..... )]    ## lx est la liste des xi

lf = [..... for xi in lx]

plt.plot(.... , ....)
```

Question 3. Calculer l'intégrale I par :

- la méthode des rectangles "arrières" ;
- la méthode des rectangles "avants" ;
- la méthode des trapèzes.

On pourra s'aider des instructions suivantes :

```
Ig = 0

for i in range(......):
    ....
print('Ig =',Ig)
```

Centre d'intérêt 6

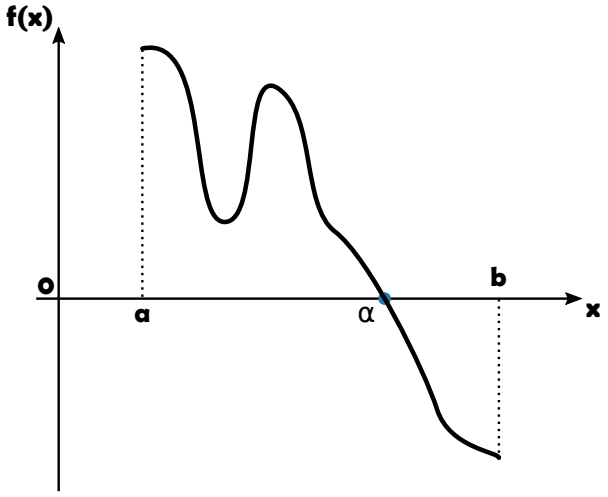
Résolution d'une équation $f(x) = 0$

PSI - MP : Lycée Rabelais

On considère une fonction f continue sur un intervalle $[a, b]$. On cherche $\alpha \in [a, b]$ tel que $f(\alpha) = 0$. Cette solution existe si $f(a) \times f(b) < 0$. Les méthodes pour approcher la valeur de α consistent à construire une suite $(x_n)_{n\geq 0}$ telle que $\lim\limits_{n\to+\infty} x_n = \alpha$.

On fixe usuellement une tolérance ε (valeur fixée). On peut utiliser plusieurs critères d'arrêt :

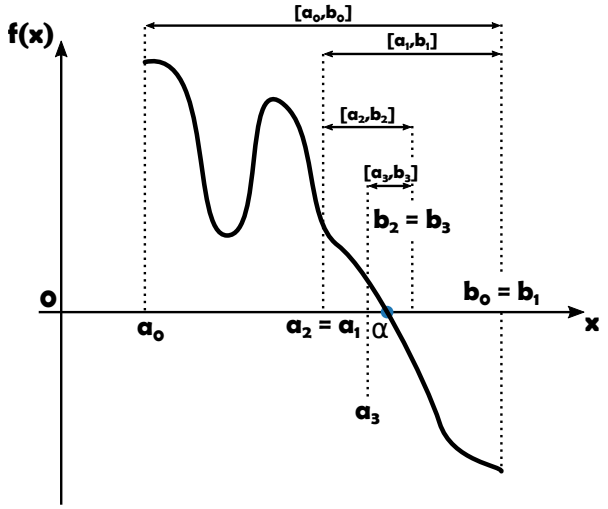
- Critère absolu : $|x_{n+1} - x_n| < \varepsilon$
- Critère relatif : $\left|\dfrac{x_{n+1} - x_n}{x_{n+1}}\right| < \varepsilon$
- Critère résiduel : $|f(x_n)| < \varepsilon$



1 Méthode de dichotomie

Les étapes de l'algorithme sont les suivantes :

1. Calculer le point milieu m de l'intervalle $[a, b]$.
2. Évaluer le signe de $f(a) \times f(m)$.
3. En déduire le sous-intervalle $[a, m]$ ou $[m, b]$ dans lequel chercher la solution.
4. Repartir à l'étape **1** tant que le critère d'arrêt n'est pas vérifié.

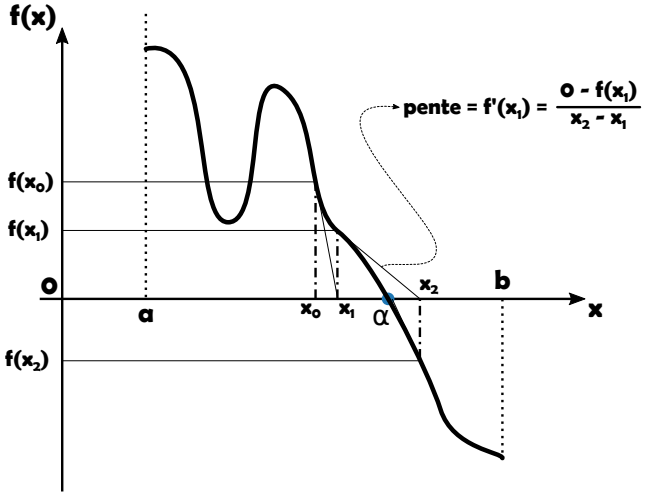


2 Méthode de Newton

Pour cette méthode, la fonction f doit être dérivable sur l'intervalle $[a, b]$.
Pour résoudre l'équation $f(\alpha) = 0$, il faut suivre les étapes suivantes :

1. Calculer une valeur $x_0 \in [a, b]$.
2. Construire la valeur suivante $x_{n+1} = x_n - \dfrac{f(x_n}{f'(x_n)}$.
3. Repartir à l'étape **2** tant que le critère d'arrêt n'est pas vérifié.

Pour retrouver la relation de récurrence, il faut écrire la pente $f'(x_i)$ en un point $(x_i, f(x_i))$.



Exercice d'application

On considère la fonction $f(x) = \sin(\sin(x)) - \sin(\sin(2x))$. On veut trouver la solution $\alpha \in [1,2]$ de l'équation $f(\alpha) = 0$.

Question 1. Définir la fonction `f`.

Question 2. Tracer la courbe représentative de la fonction sur l'intervalle souhaité en utilisant le script ci-dessous :

```
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return ................

Lx = np.linspace(...................) ## documentation numpy fournie ci-dessous

Lf = f(Lx) ## fonctionne car tableaux numpy !

plt.plot(Lx,Lf)
```

Documentation `numpy`

Le module `numpy` permet de gérer notamment des vecteurs et des tableaux. La plupart des opérations sur les listes peuvent s'appliquer sur les tableaux `numpy`. Quelques opérations élémentaires sont données ci-dessous :

- `A = numpy.zeros((a,b))` crée un tableau de `a` lignes et `b` colonnes.
- numpy.shape(A) donne le tuple associé à la taille de `A` (ici `(a,b)`).
- `A[i]` permet d'accéder à la ligne `i` au complet.
- `A[i,j]` permet d'accéder à la valeurs stockée à l'indice `i` de la ligne et `j` de la colonne (cela est équivalent à `A[i][j]`). Les `slices` du type `A[i:j,:k]` permettent de parcourir une portion du tableau (mais ce n'est pas équivalent à `A[i:j][:k]`).
- La fonction `append` ne fonctionne pas pour des tableaux `numpy` !

`numpy.linspace(start , stop , num)` renvoie un vecteur `numpy` de `num` valeurs régulièrement réparties sur l'intervalle `[start , stop]`.

Question 3. Postuler sur les solutions à trouver.

Question 4. Déterminer la solution $\alpha \in]a,b[$ par la méthode de dichotomie

Question 5. Calculer puis définir la fonction `fprim(x)` qui renvoie la dérivée de la fonction f.

Question 6. En déduire la solution $\alpha \in]a,b[$ par la méthode de Newton.

Centre d'intérêt 6

Filtrage numérique

PSI - MP : Lycée Rabelais

On s'intéresse ici aux signaux obtenus par des capteurs puis à leur filtrage. Le vocabulaire suivant, propre à la mesure, est à connaitre :

- **Incertitudes** : elles caractérisent la dispersion des valeurs attribuées à une mesure ;
- **Résolution** : plus petite mesure que l'on peut réaliser avec le capteur ;
- **Quantification** : plus petite grandeur que l'on peut quantifier avec un codage numérique donné ;
- **Échantillonnage** : action qui consiste à prélever les valeurs d'un signal à intervalles définis, généralement réguliers ;
- **Justesse** : aptitude d'un capteur à mesurer, en moyenne, la valeur attendue ;
- **Fidélité** : aptitude d'un capteur à mesurer la même valeur pour une même grandeur (mais pas nécessairement la bonne) ;
- **Linéarité** : capacité d'un capteur à fournir un signal proportionnel à la grandeur mesurée ;
- **Sensibilité** : paramètre exprimant la variation du signal de sortie d'un appareil de mesure en fonction de la variation du signal d'entrée.

On présente ici plusieurs méthodes de filtrage de signaux. On considère un signal d'entrée $\mathscr{E}$ dont on dispose des valeurs échantillonnées $e_i = \mathscr{E}(t_i)$ pour tous les temps t_i. On note également $s_i = \mathscr{S}(t_i)$, le signal de sortie à l'instant t_i.

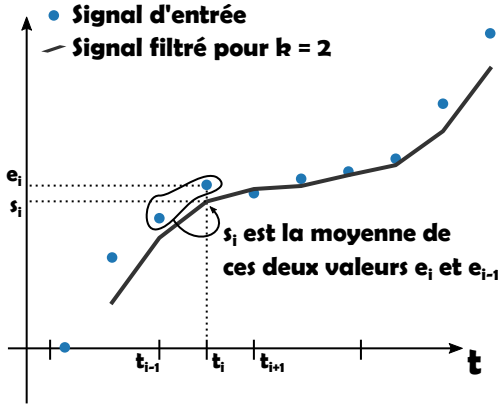
1 Filtre à moyenne glissante

Un filtre à moyenne glissante sur k valeurs consiste à renvoyer, en sortie du filtre et au temps t_i, la moyenne des k échantillons de $\mathscr{S}$ précédant le temps t_i.

On aura donc :

$$\mathscr{S}(t_i) = s_i = \frac{1}{k} \cdot \sum_{j=i-k+1}^{i} e_j = \frac{1}{k} \cdot \sum_{j=i-k+1}^{i} \mathscr{E}(t_i)$$

Bien entendu, avec un tel filtre, on perd les k premières valeurs du signal d'entrée.



2 Filtre avec fonction de transfert

On note $F(p) = \dfrac{S(p)}{E(p)}$ la fonction de transfert du filtre reliant la sortie $\mathscr{S}$ à l'entrée $\mathscr{E}$. Pour obtenir, l'équation de récurrence entre la sortie et l'entrée, il faut réécrire l'équation $S(p) = F(p) \cdot E(p)$ dans le domaine temporel.

2.1 Filtre passe-bas du premier ordre : $F(p) = \dfrac{S(p)}{E(p)} = \dfrac{1}{1+\tau \cdot p}$

On a ici $\tau \cdot \dfrac{\mathrm{d}\mathscr{S}}{\mathrm{d}t}(t) + \mathscr{S}(t) = \mathscr{E}(t)$ ce qui donne $\tau \cdot \dfrac{s_i - s_{i-1}}{\mathrm{Te}} + s_i = e_i$ avec Te la période d'échantillonnage. Ce qui permet donc d'écrire :

$$s_i = \frac{\mathrm{Te}}{\mathrm{Te}+\tau} \cdot e_i + \frac{\tau}{\mathrm{Te}+\tau} \cdot s_{i-1}$$

2.2 Filtre passe-bas du deuxième ordre : $F(p) = \dfrac{S(p)}{E(p)} = \dfrac{1}{1 + \dfrac{2\xi}{\omega_0} \cdot p + \dfrac{1}{\omega_0^2} \cdot p^2}$

Ici, il faut suivre la même méthode en remarquant que $\dfrac{\mathrm{d}^2\mathscr{S}}{\mathrm{d}t^2}(t_i) \approx \dfrac{\dfrac{\mathrm{d}\mathscr{S}}{\mathrm{d}t}(t_i) - \dfrac{\mathrm{d}\mathscr{S}}{\mathrm{d}t}(t_{i-1})}{\mathrm{Te}}$.

Exercice d'application

On définit un "faux" signal mesuré prenant les valeurs $e_i = \mathscr{E}(t_i)$ rangées dans le tableau `numpy` `Ei` associées au temps t_i du tableau `Ti`.

Question 1. Compléter le code fourni ci-dessous pour définir le signal filtré en utilisant une moyenne glissante sur $k = 2$ valeurs.

```
import random
import numpy as np
import matplotlib.pyplot as plt

def f(t):
    return t**3-5*t**2+random.random()

Ti = np.linspace(-2,2,100) ## instants ti pour mesure du signal
Ei = [f(ti) for ti in Ti]  ## "faux" signal mesuré ei = E(ti)
plt.scatter(Ti,Ei,s=10,label='mesure') ## tracé

## Filtrage avec moyenne glissante et k = 2
Tfiltre = []
Ffiltre = []

k = 2
for j in range(.........):
    ..........
    ...


    ... à compléter


    ...

plt.plot(Tfiltre,Ffiltre,color='red',label='signal filtre par moyenne glissante') ## tracé

## Affichage de la légende et de la grille
plt.legend()
plt.grid(True)
```

Question 2. Mettre en place un filtre passe-bas d'ordre 1. Analyser l'influence de la constante de temps de ce filtre.

Question 3. Mettre en place un filtre passe-bas d'ordre 2. Analyser l'influence de ses paramètres.

Centre d'intérêt 6

Résolution d'équations différentielles

PSI - MP : Lycée Rabelais

1 Contexte mathématique

On peut montrer que toute équation différentielle sur un intervalle $I \in \mathbb{R}$ peut se mettre sous la forme suivante :

$$Y'(t) = \mathscr{F}(Y(t), t)$$

Avec :

- Y la fonction inconnue de I dans $\mathbb{R}^n$. Dans le cas général, Y est bien un vecteur. Dans le cas d'une équation différentielle "ordinaire" d'ordre 1, Y est un scalaire.
- et $\mathscr{F}$, une fonction de $\mathbb{R}^n \times \mathbb{R}$ qui à $Y(t)$ et t associe une valeur dans $\mathbb{R}^n$.

Pour une résolution numérique, on discrétise l'intervalle I en n subdivisions de longueur dt (le **pas** de la discrétisation). On notera t_k la k-ième discrétisation de t de telle sorte que $t_k = k \cdot \mathrm{dt} + t_0$ où t_0 est le temps "initial".

On notera également Y_k l'approximation de $Y(t_k)$. Y_0 sera le vecteur contenant les conditions initiales de telle sorte que $Y(t_0) = Y_0$.

En faisant l'approximation d'Euler : $Y'(t_k) \approx \dfrac{Y(t_{k+1}) - Y(t_k)}{\mathrm{dt}}$, on montre la relation de récurrente ci-dessous qui permet de calculer tous les Y_k à partir de Y_0 :

$$Y_{k+1} = Y_k + \mathrm{dt} \cdot F(Y(t_k), t_k)$$

2 Mise en œuvre sur `Python`

Sur `Python`, de nombreuses bibliothèques permettent de résoudre ce type de problème. Nous utiliserons ici le module `odeint` de la bibliothèque `scipy.integrate`. Cette bibliothèque utilise une approximation plus fine que celle d'Euler mais le concept est le même.
Pour résoudre une équation différentielle avec ce module, il faudra donc écrire `tab_sol = odeint(F,Y0,tab_t)` où :

- `tab_sol` est le tableau contenant la solution (tous les Y_k) ;
- `F` est la fonction $\mathscr{F}$ définie précédemment qui prendra en argument `Y` (correspondant à Y_k) et `t` (correspondant au temps t_k) ;
- `Y0` est le vecteur conditions initiales ;
- `tab_t` est le tableau des temps (t_k) issus de la discrétisation (s'obtient facilement avec `np.linspace(debut,fin,nb de points)`).

Le code fourni ci-dessous à titre d'exemple permet de résoudre l'équation différentielle suivante sur l'intervalle $I = [t_0, t_f] = [0, 10]$:

$$y'(t) = -2 \cdot t \cdot y(t) + \cos(t) \text{ et } y(0) = 1$$

Ce qui équivaut à :

$$y'(t) = \mathscr{F}(y(t), t) \text{ et } y(0) = 1$$
$$\text{avec } \mathscr{F}(y(t), t) = -2 \cdot t \cdot y(t) + \cos(t)$$

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from scipy.integrate import odeint
4
5  def F(Y,t):
6      return - 2*t*Y + np.cos(t)
7
8  t0 = 0
9  tf = 10.
10 n = 500 ## nb de subdivisions
11 Y0 = 1
12
13 tab_t = np.linspace(t0,tf,n+1)
14 tab_sol=odeint(F,Y0,tab_t)
15 plt.plot(tab_t,tab_sol)
```

Exercices d'application

Exercice n°1

Résoudre, sur l'intervalle $[0,5]$, l'équation différentielle suivante :

$$x'(t) + 5 \cdot x(t) \cdot \sin(x(t)) = \frac{1}{x(t)} \text{ et } x(0) = 2$$

Exercice n°2

On pose $Y = \begin{pmatrix} u \\ v \end{pmatrix}$. On veut résoudre, sur l'intervalle $[0,1]$, le système d'équations différentielles suivant :

$$\begin{array}{l} u'(t) + \cos(v(t)) + \cos(t) = 0 \\ v'(t) + \sin(u(t)) + \sin(t) = 0 \end{array} \text{ avec } u(0) = 2 \text{ et } v(0) = 4$$

Q1. Déterminer la fonction $\mathscr{F}$ et Y_0 tels que le système d'équations différentielles s'écrive :

$$Y'(t) = \mathscr{F}(Y(t), t) \text{ avec } Y(0) = Y_0$$

Q2. Résoudre l'équation différentielle numériquement puis tracer la courbe paramétrée représentant les points $M(u, v)$.

Exercice n°3 : Balançoire sous oscillations forcées

On considère une balançoire de moment d'inertie $J = 90$ kg.m^2, de masse $m = 10$ kg et suspendue à un portique avec une corde de longueur $L = 3$ m. On exerce, par intermittence, un couple sur la balançoire d'intensité $C = 50$ N.m. On note θ l'angle entre la corde et la verticale. On prendra $g = 9.81$ m/s^2.

Les conditions initiales sont $\theta(0) = 0.1$ et $\dot{\theta}(0) = 0$

L'étude dynamique mène à l'équation suivante :

$$\begin{array}{l} J \cdot \ddot{\theta}(t) + m \cdot g \cdot L \cdot \sin(\theta(t)) = C \text{ si } \theta(t) \leq 0 \text{ et } \dot{\theta}(t) \geq 0 \\ J \cdot \ddot{\theta}(t) + m \cdot g \cdot L \cdot \sin(\theta(t)) = 0 \text{ sinon.} \end{array}$$

Q1. On introduit le vecteur $Y = \begin{pmatrix} \theta \\ \dot{\theta} \end{pmatrix}$. Mettre le problème sous la forme :

$$\dot{Y}(t) = \mathscr{F}(Y(t), t) \text{ avec } Y(0) = Y_0$$

Q2. Résoudre l'équation différentielle numériquement puis tracer l'évolution de θ en fonction du temps entre $t = 0$ et $t = 60$ secondes. Commenter.

Q3. Que se passe-t-il si on lance une simulation pendant 120 secondes.

Centre d'intérêt 6

Résolution d'un système linéaire

PSI - MP : Lycée Rabelais

On s'intéresse ici à la résolution d'un système linéaire de la forme $A \cdot x = b$ où A est une matrice carrée d'ordre n, b un vecteur colonne de taille n et x le vecteur inconnu. Si A est une matrice inversible (ce qui sera considéré être le cas), alors la solution de ce problème est : $x = A^{-1} \cdot b$.

Dans `Python`, le module `numpy` (par exemple), permet de résoudre un tel problème. Pour déterminer le vecteur inconnu $X = [x, y, z, v]$ à partir de ce système :

$$\begin{cases} x + 2y + 3z + 4v = 10 \\ 4x + 5y + 6z + 7v = 11 \\ 7x + 8y + 9z + 10v = 12 \\ 10x + 11y + 12z + 13v = 13 \end{cases}$$

Il suffira d'écrire :

```
import numpy
A = numpy.array([[1,2,3,4],[4,5,6,7],[7,8,9,10],[10,11,12,13]])
b = numpy.array([10,11,12,13])
x = numpy.linalg.solve(A,b)
print(x)
```

Exercice d'application

On considère le système suivant :

$$\begin{cases} x + y + 2z = 5 \\ x = y + z + 1 \\ x = 3 - z \end{cases}$$

On pose : $X = [x, y, z]$.

Question 1. Déterminer A et b tels que $A \cdot X = b$.

Question 2. Déterminer la solution X de ce système matriciel.