Cours 5 : Corrigés

« Déjà essayé. Déjà raté. N'importe. Essayer encore. Rater encore. Rater mieux. »
Cap au pire, 1982, Samuel Beckett.

1 Méthode d'Euler

Réponse 1.

Pour une fonction f définie dans un intervalle réel I, dérivable en a jusqu'à l'ordre n :

$$f(a+h) = f(a) + \frac{h}{1!}\cdot f'(a) + \frac{h^2}{2!}\cdot f^{(2)}(a) + \cdots + \frac{h^n}{n!}\cdot f^{(n)}(a) + o(h^n)$$

Réponse 2.

Le schéma numérique explicite d'Euler se déduit de cette formule à partir de l'équation différentielle issue de la modélisation physique du problème qui donne l'expression de f' en fonction de f et de la variable temporelle t qui remplace a.

— $y_i = f(a) = f(t)$,
— $pas = h$,
— $y_{i+1} = f(a+h) = f(a) + h\cdot f'(a) = f(t+pas)+$,
— $eqdf : f, t \mapsto f'(t)$.

Alors le schéma numérique explicite peut s'écrire sous la forme générale suivante :

$$y_{i+1} = y_i + pas\cdot eqdf(y_i, t)$$

Réponse 3.

Pour le schéma numérique implicite, la dérivée est approchée par sa valeur à droite $f'(a+h)$. C'est à dire :

$$y_{i+1} = y_i + pas\cdot eqdf(y_{i+1}, t)$$

Cette expression fait intervenir y_{i+1} de chaque côté de l'égalité. Ce schéma nécessite donc la résolution de l'équation d'inconnue y_{i+1} par la méthode de Newton par exemple puisque y_i est déjà une bonne approximation de la solution pour démarrer l'algorithme qui va donc converger rapidement.

2 Étude d'un pendule

2.1 Lancement de l'escarpin

2.1.1 Balancement du siège

Réponse 4.

$$\ddot{\theta}_1 = -\frac{g}{l}\cdot\sin\theta_1 - \delta(\dot{\theta}_1)\cdot\frac{k\cdot l}{m}\cdot\dot{\theta}_1^2$$

où $\delta(\dot{\theta}_1)$ est une fonction qui renvoie -1, +1 ou 0 selon le signe de $\dot{\theta}_1$.

Le système est soumis à l'action de trois forces, le poids $\vec{P}$, la traînée $\vec{T}$ et l'action des cordes $\vec{R}$.

$$\text{PFD} \implies \vec{P} + \vec{T} + \vec{R} = m\cdot\vec{a}(B/R_0)$$

Avec $\vec{P} = -m\cdot g\cdot\vec{y}_0 = -m\cdot g\cdot\sin\theta_1\cdot\vec{x}_1 - m\cdot g\cdot\cos\theta_1\cdot\vec{y}_1$, $\vec{T} = -k\cdot\delta(\dot{\theta}_1)\cdot l^2\cdot\ddot{\theta_1}^2\cdot\vec{x}_1$, $\vec{R} = R\cdot\vec{y}_1$, $\vec{V}(B/R_0) = \left[\frac{d\vec{AB}}{dt}\right]_{R_0} = l\cdot\dot{\theta}_1\cdot\vec{x}_1$ et $\vec{a}(B/R_0) = \left[\frac{d\vec{V}(B/R_0)}{dt}\right]_{R_0} = l\cdot\ddot{\theta}_1\cdot\vec{x}_1 + l\cdot\dot{\theta_1}^2\cdot\vec{y}_1$.

L'application du PFD aboutit donc à ce système d'équations différentielles.

$$\begin{cases} -m\cdot g\cdot\sin\theta_1 - k\cdot\delta(\dot{\theta}_1)\cdot l^2\cdot\ddot{\theta_1}^2 &= m\cdot l\cdot\ddot{\theta}_1 \\ -m\cdot g\cdot\cos\theta_1 + R &= m\cdot l\cdot\dot{\theta_1}^2 \end{cases}$$

La première équation de ce système permet d'écrire l'équation différentielle demandée.

Réponse 5.

```
def delta(x, eps = 1e-14):
    '''renvoie -1., +1., 0. selon le signe de x.'''
    if x < -eps : return (-1.)
    if x > eps : return (1.)
    else : return (0.)
```

L'équation obtenue précédemment est une équation différentielle non linéaire du second ordre à une variable. Le problème à résoudre ici est donc :

$$(\mathscr{C}) : \begin{cases} \ddot{\theta}_1(t) &= -\frac{g}{l} \cdot \sin\theta_1(t) - \delta(\dot{\theta}_1(t)) \cdot \frac{k \cdot l}{m} \cdot \dot{\theta}_1(t)^2 \\ \dot{\theta}_1(0) &= 0 \\ \theta_1(0) &= \theta_{10} \end{cases}$$

On prendra les valeurs numériques suivantes pour les constantes du problème : $g = 9.81$, $l = 2$, $j = 0.5$, $m = 65$, $k = 0.8$ et $\theta_{10} = 0.5$ en unités du système international.

On suppose que le module `numpy` est importé avec la syntaxe `import numpy as np`.

Réponse 6.

```
def eqdf(t, theta, theta_p, g = 9.81, l = 2, m = 65, k = 0.4):
    '''renvoie l'accélération angulaire du pendule.'''
    return(- (g / l) * np.sin(theta) - \
           delta(theta_p) * (k * l / m) * pow(theta_p, 2))
```

Réponse 7.

$$F_1 : Y = (Y_0, Y_1) \to Y_1, -\frac{g}{l} \cdot \sin Y_0 - \delta(Y_1) \cdot \frac{k \cdot l}{m} \cdot Y_1^2$$

```
def F1(t, Y, g = 9.81, l = 2, m = 65, k = 0.4):
    '''renvoie le vecteur derive de Y.'''
    return(Y[1], eqdf(t, Y[0], Y[1], g, l, m, k))
```

Soit n un entier strictement supérieur à 1 et $J_n = [\![0, n-1]\!]$. On note ht le pas temporel tel que $\forall i \in J_n$, $t_i = t_{min} + i \cdot ht$. On pose $Y_i = [y_i, \dot{y}_i]$. Le procédé de vectorisation permet donc de définir ces deux suites $(y_i)_{i \in J_n}$ et $(\dot{y}_i)_{i \in J_n}$.

Réponse 8.

$ht = \frac{t_{max} - t_{min}}{n-1}$

```
ht = (t_max - t_min) / (n - 1)
t = np.arange(t_min, t_max + ht, ht)
##t = np.linspace(t_min, t_max, n)
##t = [t_min + k * ht for k in range(n)]

theta_10 = 0.5
Y0 = [theta_10, 0]
```

Réponse 9.

$$Y_{i+1} = Y_i + ht \cdot \dot{Y}_i = Y_i + ht \cdot F_1(t_i, Y_i)$$

Réponse 10.

```
def Euler(tmin, tmax, n, Y0, F):
    ht = (tmax - tmin) / (n - 1)
    t = np.arange(tmin, tmax + ht, ht)
    Yt = [[Y0[0]], [Y0[1]]]
    for i in range(n - 1) :
        Yt_p = F(t[i], [Yt[0][i], Yt[1][i]])
        Yt[0].append(Yt[0][i] + ht * Yt_p[0])
        Yt[1].append(Yt[1][i] + ht * Yt_p[1])
    return(t, Yt)
```

Réponse 11.

```
tE, ResE = Euler(t_min, t_max, n, Y0, F1)

plt.plot(tE, ResE[0])
plt.savefig('Balancement_Euler.png')
plt.show()
```

Réponse 12.

La balançoire a des oscillations de plus en plus amples, ce qui est en contradiction avec l'analyse physique. Il faut donc en conclure que le schéma d'Euler n'est pas suffisamment consistant.

Il existe d'autres schémas numériques permettant d'approcher la courbe représentative d'une fonction solution d'une équation différentielle.

Réponse 13.

Il en existe plusieurs, comme les méthodes de Heun, Runge Kutta d'ordre 2 ou 4 vues en cours.

Les deux dernières courbes sont plus conformes à ce que l'analyse physique peut anticiper.

Dorénavant, les résultats de la simulation numérique seront disponibles dans une liste `t` pour les instants et dans une liste `RES = [THETA, THETA_p]` contenant les listes des valeurs de θ_1 et de $\dot{\theta}_1$.

2.1.2 Balancement de la jambe

À la première oscillation, la jeune femme perd (ou lance) son escarpin.

Dans un premier temps, l'effet dynamique du balancement de sa jambe est négligé.

À l'instant $t_b = 1$ s, la jeune femme balance sa jambe avec une vitesse constante $\dot{\theta}_2 = -2 \text{ rad} \cdot \text{s}^{-1}$. On suppose alors que la chaussure quitte le pied lorsque la jambe fait un angle de 45° avec la direction des cordes de l'escarpolette.

Réponse 14.

1. $\forall t \in [0, t_b]$, $\theta_2(t) = 0$ et comme $\dot{\theta}_2(t)$ est constant, $\forall t \in [t_b, t_l]$, $\theta_2(t) = \dot{\theta}_2 \cdot (t - t_b)$.
   Donc $t_l = t_b + \frac{\theta_2(t_l)}{\dot{\theta}_2(t_l)}$, c'est à dire, $t_l = 1.4$ s.
2. $\forall t \in [0, t_b]$, $\vec{AP} = -d \cdot \vec{x}_1 - (l + j) \cdot \vec{y}_1$ et $\forall t \in [t_b, t_l]$, $\vec{AP} = -d \cdot \vec{x}_1 - l \cdot \vec{y}_1 - j \cdot \vec{y}_2$.
   Donc $\vec{AP} = (-d + j \cdot \sin\theta_2) \cdot \vec{x}_1 - (l + j \cdot \cos\theta_2) \cdot \vec{y}_1$.
3. $\forall t \in [0, t_b]$, $\vec{V}(P/R_0) = -d \cdot \dot{\theta}_1 \cdot \vec{y}_1 + (l + j) \cdot \dot{\theta}_1 \cdot \vec{x}_1$ et $\forall t \in [t_b, t_l]$, $\vec{V}(P/R_0) = -d \cdot \dot{\theta}_1 \cdot \vec{y}_1 + l \cdot \dot{\theta}_1 \cdot \vec{x}_1 + j \cdot \dot{\theta}_2 \cdot \vec{x}_2$
   Donc $\vec{V}(P/R_0) = (l \cdot \dot{\theta}_1 + j \cdot \dot{\theta}_2 \cdot \cos\theta_2) \cdot \vec{x}_1 + (-d \cdot \dot{\theta}_1 + j \cdot \dot{\theta}_2 \cdot \sin\theta_2) \cdot \vec{y}_1$.

Réponse 15.

Pour cela, il faut déjà pouvoir récupérer l'indice de `t` correspondant à $t = t_l$.

```
tb = 1
theta_2 = -np.pi/4
theta_p2 = -1.5
tl = tb + theta_2 / theta_p2

itl = int((tl - t_min) * (n - 1) / (t_max - t_min))

THETA1 = [Res[0][itl], Res[1][itl]]
```

Réponse 16.

```
def base10(V, theta1):
    '''V est écrit dans la base B1, theta1 est la valeur de theta1 à l'instant
considéré. la fonction renvoie V dans la base B0.'''
    return( [V[0] * np.cos(theta1) - V[1] * np.sin(theta1), \
             V[0] * np.sin(theta1) + V[1] * np.cos(theta1)] )
```

Réponse 17.

```
P1 = [- d + j * np.sin(theta_2), -(l + j * np.cos(theta_2))]
VP1 = [l * THETA1[1] + j * theta_p2 * np.cos(theta_2), \
       - d * THETA1[1] + j * theta_p2 * np.sin(theta_2)]

P = base10(P1, THETA1[0])
VP = base10(VP1, THETA1[0])
```

2.2 Trajectoire de l'escarpin

Réponse 18.

L'escarpin étant considéré de faible traînée, cela revient à déterminer la trajectoire d'un mobile en chute libre parfaite. La solution analytique existe donc et il est alors inutile d'utiliser des méthodes numériques.

3 Méthode de Newton

Réponse 19.

Le principe de cette méthode repose sur l'équation de la droite passant par le point d'abscisse x_0 et de coefficient directeur $f'(x_0)$. On recherche l'abscisse du point de cette droite qui coupe l'axe des abscisses. Cela revient à écrire :

$$\frac{\Delta f}{\Delta x} = f'(x_k) = \frac{0 - f(x_k)}{x_{k+1} - x_k}$$

On a alors à l'itération k : $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$. Le principe de cette méthode consiste alors à répéter cette relation de récurrence jusqu'à ce que $f(x_{k+1}) \approx 0$.

Enfin, la dérivée peut être approchée par un schéma d'Euler explicite :

$$f'(x_k) = \frac{f(x_k + h) - f(x_k)}{h}$$

```
def Newton(f, e, x0):
    '''renvoie la solution approchée de f(x) = 0 par la méthode de Newton.'''
    xk = x0
    yk = f(x0)
    h = 1e-8
    while abs( yk ) > e :
        xk -= h * yk / (f(xk + h) - yk)
        yk = f(xk)
    return( xk )
```

Réponse 20.

Cette question n'est vraiment pas triviale, même si son énoncé le laisse supposer.

On passe donc à des fonctions à deux variables avec $t = \begin{pmatrix} t1 \\ t2 \end{pmatrix}$ et $F(t) = \begin{pmatrix} F1(t) \\ F2(t) \end{pmatrix}$.

Enfin, $F_t(t) = \begin{pmatrix} \frac{\partial F1}{\partial t1}(t) & \frac{\partial F1}{\partial t2}(t) \\ \frac{\partial F2}{\partial t1}(t) & \frac{\partial F2}{\partial t2}(t) \end{pmatrix}$.

Afin de justifier le schéma proposé, on va écrire la formule de Taylor-Young appliquée aux fonctions vectorielles :

$$F(t_k + h) \approx F(t_k) + F_t(t_k) \cdot h$$

et on cherche h tel que $F(t_k + h) \approx 0$.

Alors $F(t_k) \approx -F_t(t_k) \cdot h$, c'est à dire $h \approx F_t^{-1} \cdot [-F(t_k)] = -F_t^{-1} \cdot F(t_k)$.

Or $h = t_{k+1} - t_k$, donc

$$t_{k+1} \approx t_k - F_t^{-1} \cdot F(t_k)$$

Réponse 21.

On peut admettre les approximations suivantes :

$$F_t(t) = \begin{pmatrix} \frac{\partial F1}{\partial t1}(t) & \frac{\partial F1}{\partial t2}(t) \\ \frac{\partial F2}{\partial t1}(t) & \frac{\partial F2}{\partial t2}(t) \end{pmatrix} = \begin{pmatrix} \frac{F1(t1+h,t2) - F1(t1,t2)}{h} & \frac{F1(t1,t2+h) - F1(t1,t2)}{h} \\ \frac{F2(t1+h,t2) - F2(t1,t2)}{h} & \frac{F2(t1,t2+h) - F2(t1,t2)}{h} \end{pmatrix}$$

4 Méthode d'Euler, procédé de vectorisation

Réponse 22.

Avec le module `numpy` de Python, il existe deux fonctions qui permettent de générer ce tableau. Enfin, on peut aussi choisir de définir une fonction pour le générer.

```
import numpy as np

pas = Tmax / (N - 1)
N = int(Tmax / pas) + 1

T = np.linspace(0, Tmax, N)

T = np.arange(0, Tmax + pas, pas)

def Tableau(tmax, h) :
    return( np.arange(0, tmax + h, h) )

T = Tableau(Tmax, pas)
```

Réponse 23.

On considère que `K`, `m` et `omega0` sont des variables globales définies précédemment dans le programme. Il serait bien sûr plus rigoureux de les intégrer dans les arguments de cette fonction.

```
def f1(ti, yi) :
    # Cette fonction ne dépend pas directement de ti car elle est linéaire en yi
    return( np.array([yi[1], pow(omega0, 2) * (K - yi[0]) - \
                      2 * m * omega0 * yi[1]]) )
```

J'utilise le type `array` du module `numpy` pour les tableaux.

Réponse 24.

La relation de récurrence peut être obtenue en appliquant la relation de Taylor-Young, ou le schéma d'Euler explicite.

$$Y_{i+1} = Y_i + h \cdot \frac{dY(t)}{dt} + o(h) \approx Y_i + h \cdot F(t, Y(t))$$

Réponse 25.

```
def EulerExplicite(Yini, h, Tmax, F) :
    '''renvoie le tableau de la solution de l'équation dY/dt = F(t, Y).'''
    T = Tableau(Tmax, h)
    SY = np.zeros([len(T), 2]) # tableau de type array à coefficients nuls
    SY[0] = Yini # première ligne du tableau renseignée
    for i in range(len(T) - 1) : # on s'arrête à l'avant dernier terme
        SY[i + 1] = SY[i] + h * F(T[i], SY[i])
    return( SY )
```

Réponse 26.

On peut se référer pour répondre à cette question à la définition de l'*erreur de consistance* $e(h)$.

$$e(h) = \sum_{i=0}^{N-1} \left| y_{reelle}(t_i + 1) - \left[y(t_i) + h \cdot F(t_i, y(t_i)) \right] \right|$$

La méthode d'Euler nous assure que $\lim_{h \to 0} e(h) = 0$. On sait aussi que c'est une méthode d'ordre 1, donc elle possède un majorant de la forme $K \cdot h^1$.

Si le pas de temps est divisé par un facteur 10, l'erreur de calcul va donc diminuer d'un facteur 10 également.

Par contre, le temps de calcul et la taille des tableaux vont eux aussi augmenter.

Réponse 27.

N est la taille du tableau `T`. Je note $C(N)$, la complexité en nombre d'opérations de la méthode d'Euler implantée.

— `T = Tableau(Tmax, h)` demande N affectations.
— `SY = np.zeros([len(T), 2])` demande $2 \cdot N$ affectations.
— `SY[0] = Yini` demande 2 affectations.
— `for i in range(len(T) - 1) :` implique $N-1$ passages dans la boucle.
— `SY[i + 1] = SY[i] + h * F(T[i], SY[i])` demande 2 affectations pour `SY[i + 1]`, 2 additions, 2 multiplications et l'appel de la fonction `F` demande 2 affectation et 7 opérations mathématiques.

Si l'on décide de compter comme opération élémentaire chaque comparaison, affectation ou opération arithmétique, on trouve un coût majoré par :

$$C(N) = N + 2 \cdot N + 2 + (N-1) \cdot (2 + 2 + 2 + (2 + 7)) = 18 \cdot N - 13$$

La complexité est donc en $O(N)$.

Le temps de calcul va donc être multiplié par un facteur 10 si le pas de temps est divisé par un facteur 10.

Réponse 28.

La réponse à cette question nécessite de connaître la taille mémoire nécessaire pour stocker un flottant.

Format des nombres réels écrits en virgule flottante Les nombres réels écrits en virgule flottante sont définis par la norme internationale IEEE 754 de 1985 qui spécifie deux formats de nombres en virgule flottante en base 2 selon le tableau suivant.

Précision	Encodage	Signe	Exposant	Mantisse	Valeur d'un nombre	Chiffres significatifs
Simple	32 bits	1 bit	8 bits	23 bits	$(-1)^S \times M \times 2^{(E-127)}$	environ 7
Double	64 bits	1 bit	11 bits	52 bits	$(-1)^S \times M \times 2^{(E-1023)}$	environ 16

Alors `T` nécessite $N \times 8$ octets, et `SY`, $2 \times N \times 8$. Le nombre d'octets nécessaire en mémoire pour réaliser cette simulation avec un nombre de pas de temps $N = 10000$ est donc de 240 kO.

Informatique : Convergence de l'algorithme

Réponse 29.

Il faut vérifier la propriété suivante :

$$\forall t \in [0.9 \cdot T_{max}, T_{max}], |\omega_m(t) - \omega_m(T_{max})| < |0.1 \times 10^{-3} \cdot \omega_m(T_{max})|$$

```
def TestConvergence(t, w) :
    '''Teste si la fonction semble converger vers sa valeur finale.'''
    i = len(w) - 1 # On part de la fin
    while t[i] > 0.9 * t[-1] :
        if np.abs( (w[i] - w[-1]) / w[-1] ) > 0.001 :
            return(False)
        i -= 1
    return(True)
```

Informatique : Détermination du temps de réponse à 5% du système

Réponse 30.

```
def CalculT5(t, w) :
    '''renvoie le temps de réponse à 5/100 de w.'''
    if TestConvergence(t, w) :
        i = len(w) - 1 # on part encore de la fin
        while np.abs( (w[i] - w[-1]) / w[-1] ) < 0.05 :
            i -= 1
        return(t[i])
    return(-1)
```