Physique

Capacité numérique 3 – Équation différentielle du deuxième ordre non-linéaire

Pour le lundi 11 mars 08:00

Objectifs

- 💻 À l'aide d'un langage de programmation, résoudre numériquement une équation différentielle du deuxième ordre non-linéaire et faire apparaître l'effet des termes non-linéaires.
- 💻 Utiliser la fonction `odeint` de la bibliothèque `scipy.integrate`.
- — Transformer une équation différentielle d'ordre n en un système différentiel de n équations d'ordre 1.

Première partie

Pendule simple

Cette partie étudie le modèle du pendule simple. En notant θ l'angle du pendule par rapport à la verticale descendante, cet angle est régi par l'équation différentielle

$$\ddot{\theta} + \omega_0^2 \sin(\theta) = 0\,. \tag{1}$$

1 Étude préliminaire

1. Pourquoi n'est-il pas possible d'appliquer la méthode d'Euler pour estimer l'évolution de θ ?

La méthode d'Euler s'applique à une équation différentielle d'ordre 1. Or, ici, l'équation différentielle est d'ordre 2.

Pour pallier cela, une matrice 2×1, notée X, est introduite et définie telle que

$$X(t) = \begin{pmatrix} \theta(t) \\ \dot{\theta}(t) \end{pmatrix}\,.$$

2. Déterminer fonction F de $\mathbb{R}^2 \to \mathbb{R}^2$ telle que l'équation (1) s'écrive

$$\frac{\mathrm{d}X}{\mathrm{d}t} = F(X)\,. \tag{2}$$

Par définition,

$$\frac{\mathrm{d}X}{\mathrm{d}t} = \begin{pmatrix} \dot{\theta} \\ \ddot{\theta} \end{pmatrix} = F\begin{pmatrix} \theta \\ \dot{\theta} \end{pmatrix}\,.$$

Par identification directe pour la ligne 1 et d'après l'équation différentielle pour la ligne 2, il vient

$$F : \begin{pmatrix} a \\ b \end{pmatrix} \mapsto \begin{pmatrix} b \\ -\omega_0^2 \sin(a) \end{pmatrix}\,.$$

3. En déduire, en s'inspirant de la méthode d'Euler vue pour les systèmes d'ordre 1, les relations (mathématiques) permettant d'obtenir les valeurs de $\theta(t_{i+1})$ et $\dot{\theta}(t_{i+1})$ connaissant $\theta(t_i)$ et $\dot{\theta}(t_i)$ avec $t_i = pi$, p le pas de la simulation et i l'étape d'itération.

D'après la méthode d'Euler, pour une grandeur $y(x)$ telle que $\dot{y} = F(y, x)$,

$$y(x_{i+1}) = y(x_i) + p\,F(y(x_i), x_i)\,.$$

Alors, les deux lignes de l'équation déterminée à la question précédente donnent

$$\theta(t_{i+1}) = \theta(t_i) + p\,\dot{\theta}(t_i)\,, \quad \dot{\theta}(t_{i+1}) = \dot{\theta}(t_i) + p\,\ddot{\theta}(t_i) = \dot{\theta}(t_i) + p \times \left(-\omega_0^2 \sin(\theta(t_i))\right)\,.$$

2 Mise en œuvre

1. Écrire le code définissant la fonction `F_pendule`, prenant comme paramètres la valeur `t` de t_i et la valeur `X` de $X(t_i)$ et qui rend $F(X(t_i))$ dans le cas du pendule simple étudié. Attention à bien identifier ce qu'est X. Poser $\omega_0 = 2\pi$ (c'est-à-dire $f_0 = 1\,\text{Hz}$) en début du script.

```
4  import numpy as np
5  import matplotlib.pyplot as plt
6
7  omega0 = 2*np.pi
8
9  def F_pendule(X):
10     theta = X[0]
11     omega = X[1] # omega = d theta / dt
12     # Application des formules
13     return (omega, -omega0**2*np.sin(theta))
```

2. Écrire une fonction `next_Euler` prenant comme paramètres la valeur `theta_init` de $\theta(t_i)$, la valeur `omega_init` de $\omega(t_i) = \dot{\theta}(t_i)$, la valeur `t_init` de t_i, la fonction `F` définie précédemment et le pas `p` et qui rend les valeurs de $\theta(t_{i+1})$, $\omega(t_{i+1})$ et t_{i+1}.

```
15 def next_Euler(theta_init, omega_init, t_init, F, p):
16     # se ramener aux notations de la partie théorique
17     X_init = (theta_init, omega_init)
18     FX = F(X_init)
19     # Application des formules
20     theta_next = theta_init + p * FX[0]
21     omega_next = omega_init + p * FX[1]
22     t_next = t_init + p
23     return theta_next, omega_next, t_next
```

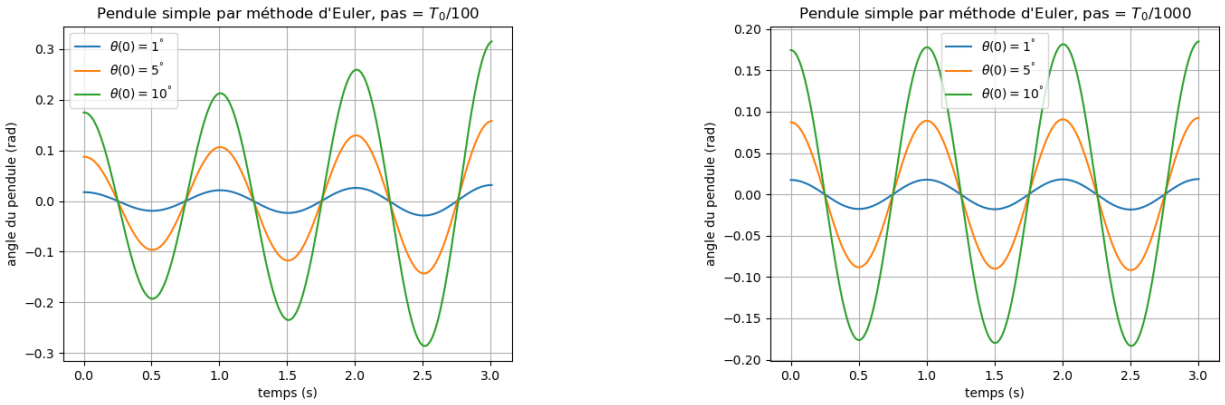
3. Soient les conditions initiales $\theta(0) = \theta_0$ et $\dot{\theta}(0) = 0$. Écrire alors un code permettant d'estimer les valeurs de $\theta(t)$ et $\dot{\theta}(t)$ de $t = 0$ à $t = 3T_0$ avec T_0 la période propre de l'oscillateur harmonique obtenue aux petites angles, avec un pas de $T_0/100$, pour θ_0 valant $1, 5, 10$ degrés et de sauvegarder les tracés de ces estimations, ensemble, dans une image « `fig1a.png` ». Commenter le résultat obtenu : périodes, amplitudes.

Le code est ci-dessous, le tracé en figure 1a.

```
25 thetas = {}
26 omegas = {}
27 time_s = {}
28 T0 = 2*np.pi/(omega0)
29 pas = T0/100
30 temps_max = 3*T0
31 for theta_0 in [1, 5, 10]:
```

```
    # conversion de l'angle en radian
    theta_0_rad = theta_0/180*np.pi
    # initialisation des listes
    theta_liste = [theta_0_rad]
    omega_liste = [0]
    temps_liste = [0]
    # application de la méthode d'Euler
    while temps_liste[-1] < temps_max:
        theta_init = theta_liste[-1]
        omega_init = omega_liste[-1]
        t_init = temps_liste[-1]
        theta_next, omega_next, t_next = next_Euler(theta_init, omega_init, t_init,
        ↪  F_pendule, pas)
        theta_liste.append(theta_next)
        omega_liste.append(omega_next)
        temps_liste.append(t_next)
    # stocker les résultats dans les dictionnaires
    thetas[theta_0] = theta_liste
    omegas[theta_0] = omega_liste
    time_s[theta_0] = temps_liste

plt.figure()
plt.title("Pendule simple par méthode d'Euler, pas = $T_0/100$")
for label in thetas:
    theta_liste = thetas[label]
    omega_liste = omegas[label]
    temps_liste = time_s[label]
    plt.plot(
        temps_liste,
        theta_liste,
        label = f"$\\theta(0) = {label}^\\degree$")
plt.legend() # afficher les labels des courbes
plt.xlabel("temps (s)")
plt.ylabel("angle du pendule (rad)")
plt.grid() # pour afficher le quadrillage
plt.savefig("fig1a.png")
```



(a) *Le tracé « `fig1a.png` ».*

(b) *Le tracé « `fig1b.png` ».*

Figure 1

Pour de petits angles, l'isochronisme des oscillations est retrouvé. En revanche, l'amplitude des oscillations n'est pas constante, il s'agit d'un artefact dû à la méthode d'Euler. Un pas plus petit permet de résoudre ce problème.

4. Reprendre la question précédente avec un pas de $T_0/1000$, dans une image « `fig1b.png` ». Le problème rencontré est-il résolu ?.

Avec un pas de $T_0/1000$, les courbes obtenue sont données en figure 1b. Les amplitudes sont désormais stables.

5. Tracer à présent les courbes pour θ_0 valant $10, 30, \ldots, 150$ degrés dans une image « `fig2.png` ». Commenter le résultat obtenu:périodes, amplitudes.

Le code est ci-dessous, le tracé en figure 2. Plus l'amplitude de l'oscillation est importante, plus la période l'est. Il n'y a plus isochronisme des oscillations.

```
109 thetas = {}
110 omegas = {}
111 time_s = {}
112 # l'angle de 170° est déjà mis pour la suite
113 for theta_0 in range(10, 171, 20):
114     # conversion de l'angle en radian
115     theta_0_rad = theta_0/180*np.pi
116     # initialisation des listes
117     theta_liste = [theta_0_rad]
118     omega_liste = [0]
119     temps_liste = [0]
120     # application de la méthode d'Euler
121     while temps_liste[-1] < temps_max:
122         theta_init = theta_liste[-1]
123         omega_init = omega_liste[-1]
124         t_init = temps_liste[-1]
125         theta_next, omega_next, t_next = next_Euler(theta_init, omega_init, t_init,
            ↪  F_pendule, pas)
126         theta_liste.append(theta_next)
127         omega_liste.append(omega_next)
128         temps_liste.append(t_next)
129     # stocker les résultats dans les dictionnaires
130     thetas[theta_0] = theta_liste
131     omegas[theta_0] = omega_liste
132     time_s[theta_0] = temps_liste
133
134 plt.figure()
135 plt.title("Pendule simple par méthode d'Euler, pas = $T_0/1000$")
136 for label in thetas:
137     if label > 150:
138         continue
139     theta_liste = thetas[label]
140     omega_liste = omegas[label]
141     temps_liste = time_s[label]
142     plt.plot(
143         temps_liste,
144         theta_liste,
145         label = f"$\\theta(0) = {label}^\\degree$")
146 plt.legend() # afficher les labels des courbes
147 plt.xlabel("temps (s)")
148 plt.ylabel("angle du pendule (rad)")
149 plt.grid() # pour afficher le quadrillage
150 plt.savefig("fig2.png")
```

6. Tracer à présent la courbe pour $\theta(0) = 170°$ dans une image « `fig3.png` ». Commenter.

Le tracé est en figure 2. Le pendule commence à faire plusieurs tours sur lui-même dans le sens négatif (anti-trigonométrique ou horaire). Ce comportement est faux : le pendule n'a pas, dans l'état initial, assez d'énergie pour aller à $|\theta| > \theta(0)$. C'est à nouveau une erreur due aux approximations de la méthode d'Euler.

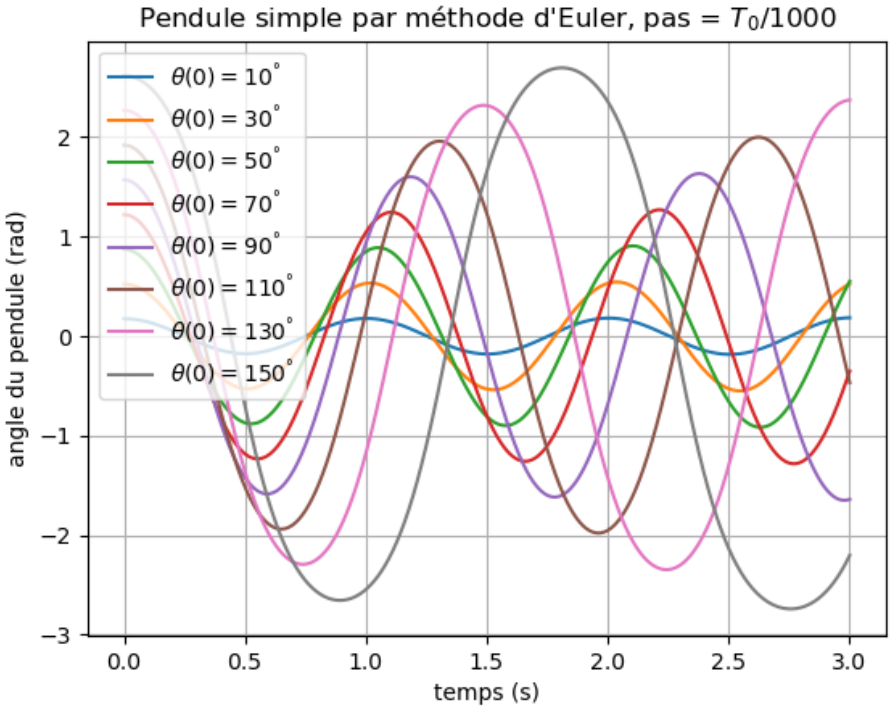


Figure 2 – *Le tracé « `fig2.png` ».*

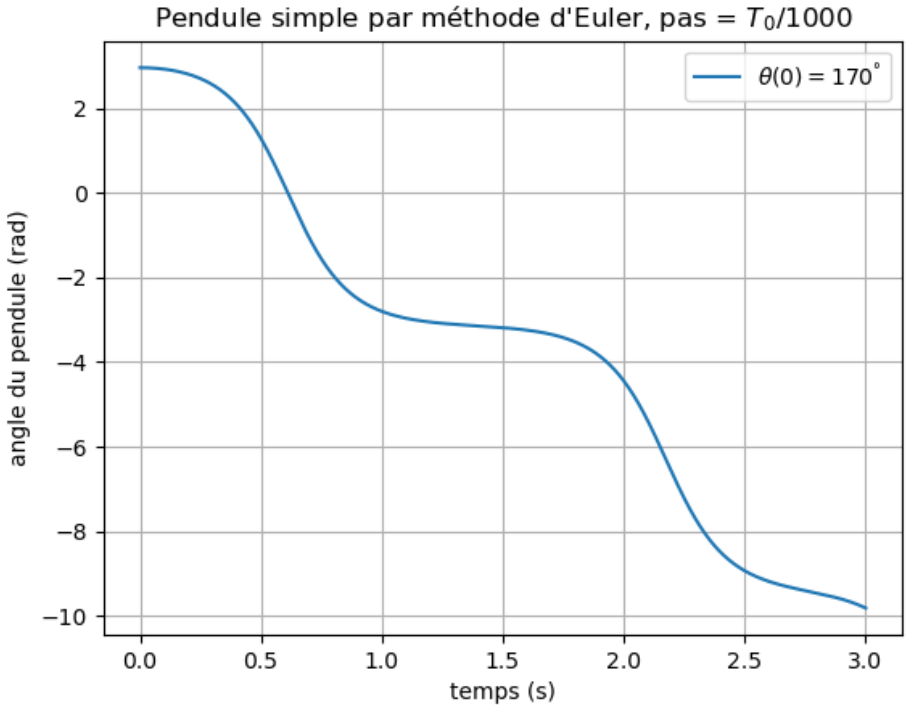


Figure 3 – *Le tracé « `fig3.png` ».*

3 Utilisation de `odeint`

La bibliothèque `scipy.integrate` comporte une fonction `odeint` dont la documentation complète est disponible en ligne[1]. Pour un système tel que $\frac{\mathrm{d}X}{\mathrm{d}t} = F(X)$ avec X pouvant être un vecteur, son utilisation peut être simplifiée, avec les lignes nécessaires en amont, en

```
from scipy.integrate import odeint # import de la fonction
X0 = (theta_init, omega_init) # le vecteur X à l'instant initial, comme avant
temps = ... # la liste des instants t auxquels obtenir X(t)
X_liste = odeint(F, X0, temps)
```

avec `F` une fonction qui n'est pas tout à fait celle précédemment définie dans ce sujet. En effet, celle-ci doit prendre deux paramètres, `X` et `temps`. Il faudra donc « mettre à jour » `F` dans le code pour utiliser `odeint`.

La sortie de `odeint` est alors un `array` de taille `len(temps),len(X0)` contenant les valeurs de X pour tous les instants t dans `temps`.

1. Écrire alors un code permettant d'obtenir les valeurs de $\theta(t)$ de $t = 0$ à $t = 3T_0$ avec T_0 la période propre de l'oscillateur harmonique obtenue aux petites angles, avec un pas de $T_0/100$, pour θ_0 valant $10, 30, \ldots, 170$ degrés et de sauvegarder les tracés de ces simulations, ensemble, dans une image « `fig4.png` ». Commenter le résultat obtenu : périodes, amplitudes.

Le code est ci-dessous.

```
# Modification de la fonction pour utiliser odeint
def F_pendule_odeint(X, t):
    return F_pendule(X)

# import de odeint
from scipy.integrate import odeint

temps = np.arange(0, 3*T0, T0/100)
thetas = {} # dictionnaire pour stocker les résultats
omegas = {}
for theta0 in range(10, 171, 20):
    theta0_rad = theta0 * np.pi/180
    X0 = (theta0_rad, 0)
    odeint_output = odeint(F_pendule_odeint, X0, temps)
    thetas[theta0] = odeint_output[:,0]
    omegas[theta0] = odeint_output[:,1]

plt.figure()
plt.title("Pendule simple par odeint, pas = $T_0/100$")
for label in thetas:
    theta_liste = thetas[label]
    plt.plot(
        temps,
        theta_liste,
        label = f"$\\theta(0) = {label}^\\degree$")
plt.legend() # afficher les labels des courbes
plt.xlabel("temps (s)")
plt.ylabel("angle du pendule (rad)")
plt.grid() # pour afficher le quadrillage
plt.savefig("fig4.png")
```

Le tracé obtenu est donné en figure 4. L'utilisation de `odeint` est plus précise que la méthode d'Euler. Les

1. https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html

amplitudes sont bien stables, ce qui est attendu avec la conservation de l'énergie potentielle, et la période des oscillations dépend bien de l'amplitude du mouvement.

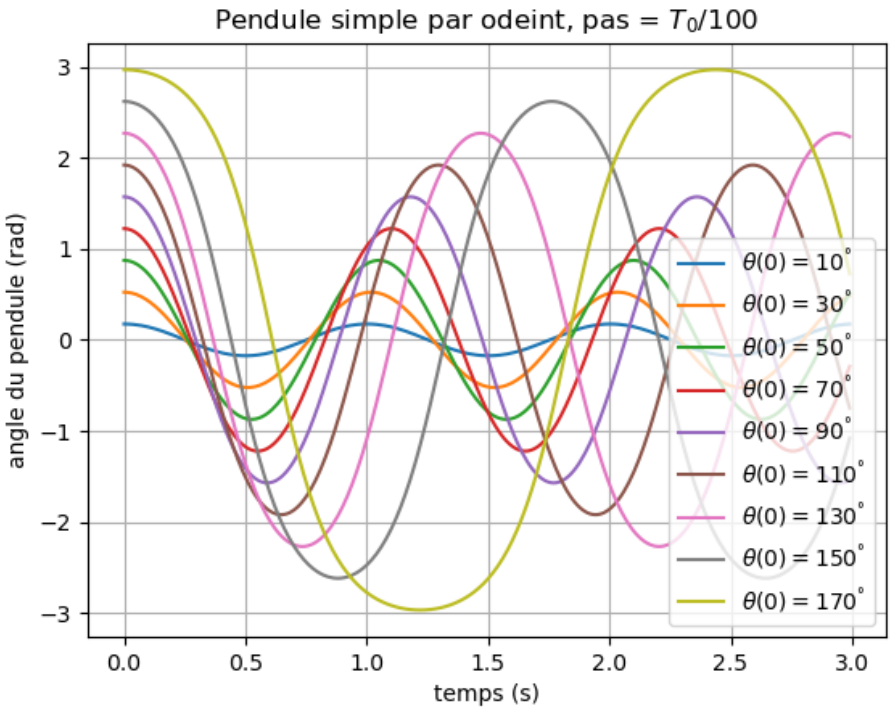


Figure 4 – *Le tracé « `fig4.png` ».*

2. Obtenir alors, à partir des courbes calculées par `odeint`, le portrait de phase de ce système, dans une image « `fig5.png` ».

Le portrait de phase est le tracé de $\dot{\theta} = f(\theta)$. Or, `odeint` a déjà fourni ces données pour les angles traités précédemment (et une vitesse initiale nulle). Alors, le portrait de phase s'obtient sans calcul supplémentaire par le code suivant.

```
199 plt.figure()
200 plt.title("Portrait de phase du pendule simple")
201 for label in thetas:
202     theta_liste = thetas[label]
203     plt.plot(
204         thetas[label],
205         omegas[label],
206         label = f"$\\theta(0) = {label}^\\degree, v_0=0$",
207     )
208 plt.legend() # afficher les labels des courbes
209 plt.xlabel("$\\theta$ (rad)")
210 plt.ylabel("$\\dot{\\theta}$ (rad/s)")
211 plt.xlim([-2*np.pi, 2*np.pi]) # ajustement des bornes de l'axe des abscisses
212 plt.ylim([-np.pi*omega0, np.pi*omega0]) # ajustement des bornes de l'axe des ordonnées
213 plt.grid() # pour afficher le quadrillage
214 plt.savefig("fig5.png")
```

Le tracé obtenu est donné en figure 5.

3. Ce portrait de phase n'est en fait pas complet, seuls les mouvements périodiques autour de $\theta = 0$ sont obtenus. Trouver un jeu de conditions initiales supplémentaires permettant de compléter le portrait de phase dans la gamme $-2\pi \leqslant \theta \leqslant 2\pi$ et $-\pi\omega_0 \leqslant \dot{\theta} \leqslant \pi\omega_0$ tout en conservant un visuel agréable (courbes ni trop espacées, ni trop rapprochées). Le sauvegarder dans une image « `fig6.png` ».

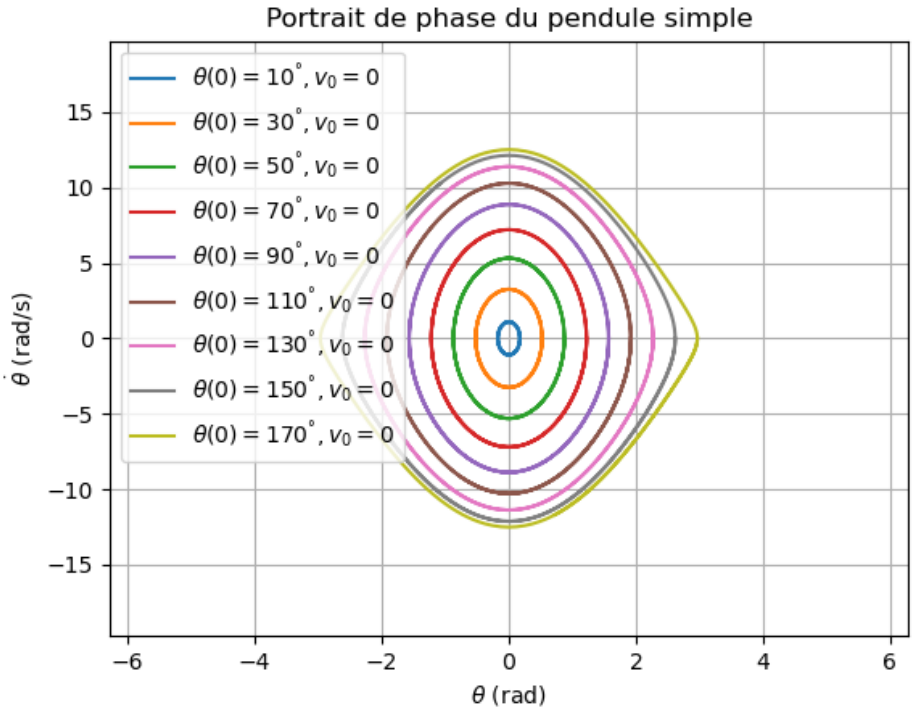


Figure 5 – *Le tracé « `fig5.png` ».*

Il faut d'une part obtenir les oscillations autour des positions d'équilibre $\theta = \pm 2\pi$; et d'autre part obtenir les courbes pour les états de diffusion (plusieurs tours dans le sens direct et dans le sens indirect).

Pour les oscillations autour des positions d'équilibre, il est possible de simplement reprendre les mêmes conditions initiales mais translatées de $\pm 2\pi$. Prendre des angles un peu moins proches permet d'obtenir un visuel plus agréable. Pour les états de diffusion, il est possible *en première intention* de remarquer qu'il suffit de prendre une vitesse initiale suffisante en valeur absolue à un angle initial nul pour obtenir le tracé correspondant pour le portrait de phase. Mais alors, il manque une partie de la courbe ($\theta < 0$ pour une vitesse positive et inversement). Alors, prendre comme valeur initiale de θ sa valeur minimale ou maximale permet de compléter le portrait de phase, d'où le choix des conditions initiales dans le code suivant.

```
conditions_initiales = [] # liste des CI pour X
for theta0 in range(10,181,30):
    theta0 *= np.pi/180 # conversion en rad
    conditions_initiales.append([theta0, 0])
    # mettre aussi les CI pour les oscillations autour des deux autres
    # positions d'équilibre à traiter
    conditions_initiales.append([theta0+2*np.pi, 0])
    conditions_initiales.append([theta0-2*np.pi, 0])
# mettre les CI pour les états de diffusion
for omega in np.arange(2.01*omega0, 4*omega0, 2):
    conditions_initiales.append([-2*np.pi, omega])
    conditions_initiales.append([2*np.pi, -omega])

thetas = {} # dictionnaire pour stocker les résultats
omegas = {}
cas = 0 # pour pouvoir retrouver chaque courbe
for CI in conditions_initiales:
    cas += 1
    X0 = CI
    odeint_output = odeint(F_pendule_odeint, X0, temps)
    thetas[cas] = odeint_output[:,0]
    omegas[cas] = odeint_output[:,1]
```

```
238
239 plt.figure()
240 plt.title("Portrait de phase du pendule simple")
241 for label in thetas:
242     theta_liste = thetas[label]
243     plt.plot(
244         thetas[label],
245         omegas[label],
246     )
247 plt.xlabel("$\\theta$ (rad)")
248 plt.ylabel("$\\dot{\\theta}$ (rad/s)")
249 plt.xlim([-2*np.pi, 2*np.pi]) # ajustement des bornes de l'axe des abscisses
250 plt.ylim([-np.pi*omega0, np.pi*omega0]) # ajustement des bornes de l'axe des ordonnées
```

Le tracé obtenu est donné en figure 6.

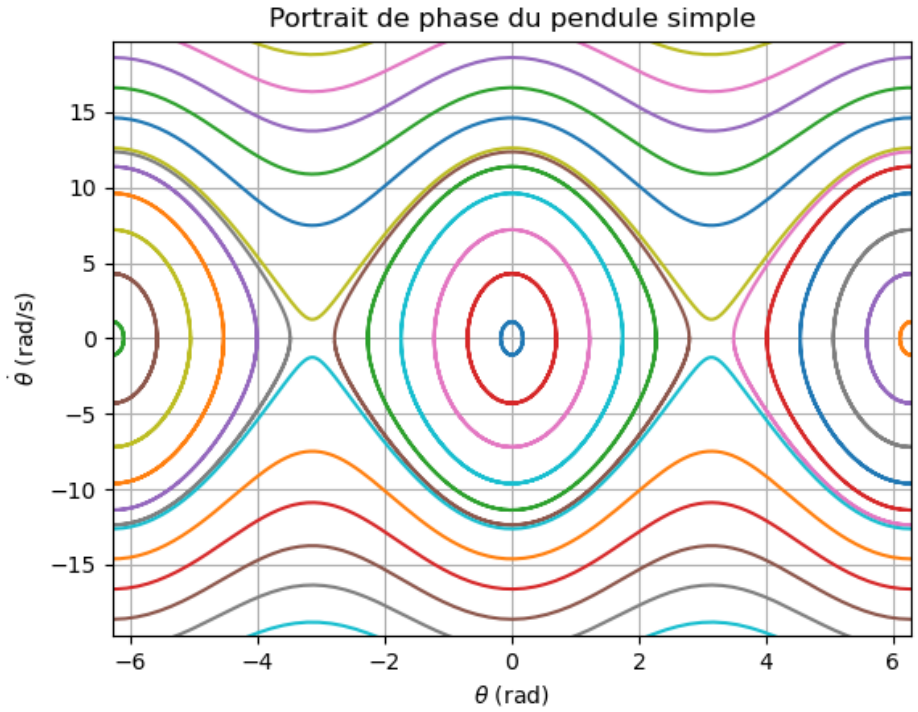


Figure 6 – *Le tracé « `fig6.png` ».*

Deuxième partie

Oscillateur de Van der Pol

L'oscillateur de Van der Pol est régi par l'équation différentielle

$$\frac{\mathrm{d}^2y}{\mathrm{d}t^2} + a\left(\frac{y^2}{b^2} - 1\right)\frac{\mathrm{d}y}{\mathrm{d}t} + \omega_0^2 y = 0\,. \tag{3}$$

Dans la suite, les paramètres seront fixés à $a = 3\,\mathrm{SI}$, $b = 1\,\mathrm{SI}$, $\omega_0 = 7\,\mathrm{rad/s}$.

4. Déterminer la fonction `F_VanderPol` utilisable avec `odeint`, analogue à `F_pendule_odeint` mais pour l'équation (3). Puis, en utilisant la méthode précédemment employée pour traiter le pendule simple avec `odeint`, obtenir $y(t)$ et $\dot{y}(t)$ pour t allant de 0 à 100 s avec 1000 points pour les conditions initiales $y(0) = 0\,\mathrm{rad}$ et $\dot{y}(0) = 0{,}1\,\mathrm{rad{\cdot}s^{-1}}$.

Tracer $y(t)$ dans une image « `fig7a.png` » et le portrait de phase avec *ces* conditions initiales dans une image « `fig7b.png` ». Commenter le portrait de phase : à terme, $y(t)$ est-elle périodique ? Sinusoïdale ? Justifier.

Commençons par déterminer F. Vu l'équation différentielle,

$$F : \begin{pmatrix} y \\ \dot{y} \end{pmatrix} \mapsto \begin{pmatrix} \dot{y} \\ -a\left(\frac{y^2}{b^2} - 1\right)\frac{\mathrm{d}y}{\mathrm{d}t} - \omega_0^2 y \end{pmatrix}.$$

Ainsi, le code définissant `F_VanderPol` est celui ci-après.

```
254 a = 3
255 b = 1
256 omega_0 = 7
257 def F_VanderPol(X, t):
258     y = X[0]
259     dy_dt = X[1] # = d y / dt
260     # Application des formules
261     return (dy_dt, -a*(y**2/b**2-1)*dy_dt - omega_0**2 * y)
```

L'obtention de $y(t)$ et $\dot{y}(t)$ avec les conditions initiales données est alors réalisée par le code ci-après.

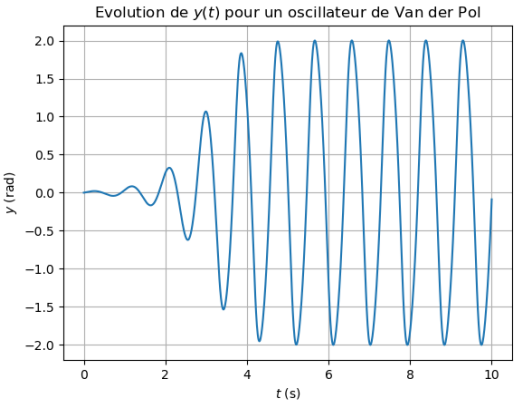
```
263 y0 = 0
264 dy_dt0 = 0.1
265 temps = np.linspace(0, 10, 1000)
266 odeint_output = odeint(F_VanderPol, (y0, dy_dt0), temps)
267 y_fct_t = odeint_output[:,0]
268 dy_dt_fct_t = odeint_output[:,1]
```

Les tracés demandés, comme précédemment, s'obtiennent avec le code suivant et sont donnés en figures 7a et 7b.

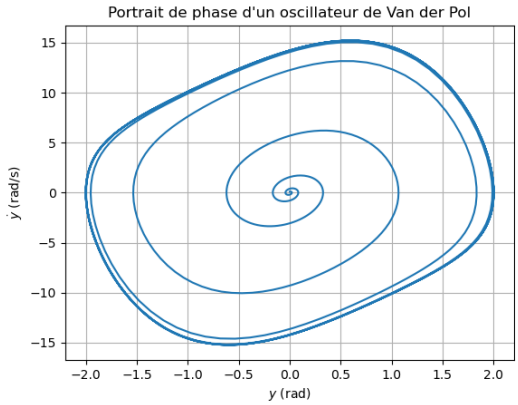
```
270 plt.figure()
271 plt.title("Evolution de $y(t)$ pour un oscillateur de Van der Pol")
272 plt.plot(
273     temps,
274     y_fct_t,
275 )
276 plt.xlabel("$t$ (s)")
277 plt.ylabel("$y$ (rad)")
278 plt.grid() # pour afficher le quadrillage
279 plt.savefig("fig7a.png")
280
281 plt.figure()
282 plt.title("Portrait de phase d'un oscillateur de Van der Pol")
283 plt.plot(
284     y_fct_t,
285     dy_dt_fct_t,
286 )
287 plt.xlabel("$y$ (rad)")
288 plt.ylabel("$\\dot{y}$ (rad/s)")
289 plt.grid() # pour afficher le quadrillage
290 plt.savefig("fig7b.png")
```

Le portrait de phase de l'oscillateur de Van der Pol est fermé. À partir d'une certaine date, le phénomène est donc périodique. Le portrait de phase du mouvement périodique n'est pas une ellipse, le régime permanent n'est donc pas sinusoïdal.

5. Faire varier les conditions initiales. Que dire du régime permanent de l'oscillateur ? Prendre par exemple $y(0) = 2\,\text{rad}$ et $\dot{y}(0) = 1\,\text{rad·s}^{-1}$ puis $y(0) = -1{,}75\,\text{rad}$ et $\dot{y}(0) = 12\,\text{rad·s}^{-1}$.



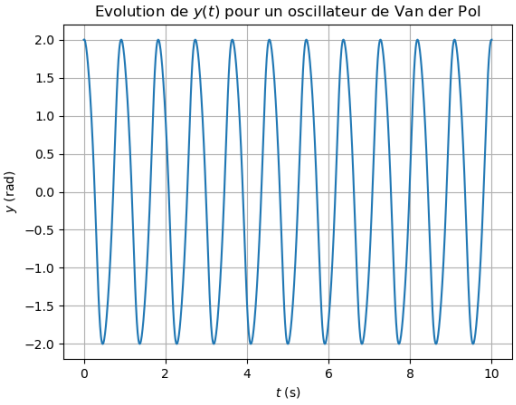
(a) *Le tracé « `fig7a.png` ».*



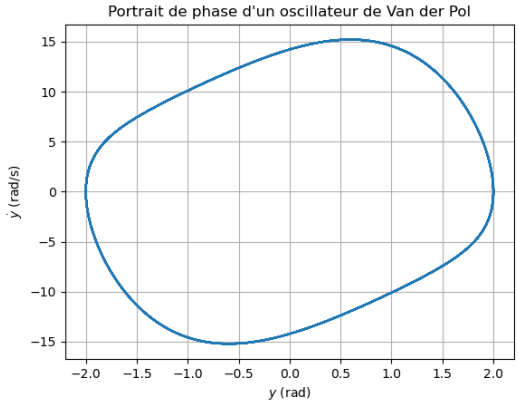
(b) *Le tracé « `fig7b.png` ».*

Figure 7

Avec un code très similaire, les tracés des figures 8a et 8b sont obtenus pour $y(0) = 2\,\text{rad}$ et $\dot{y}(0) = 1\,\text{rad}\cdot\text{s}^{-1}$ et des figures 9a et 9b pour $y(0) = -1{,}75\,\text{rad}$ et $\dot{y}(0) = 12\,\text{rad}\cdot\text{s}^{-1}$.

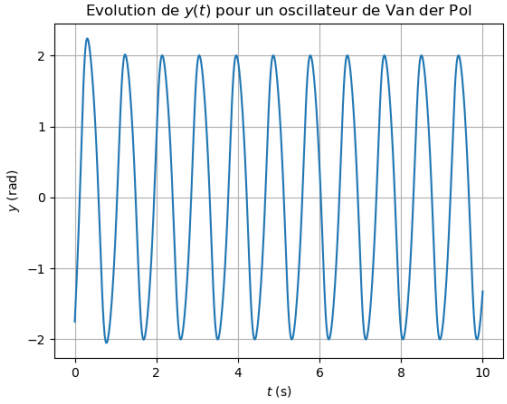


(a) *Le tracé « `fig8a.png` ».*

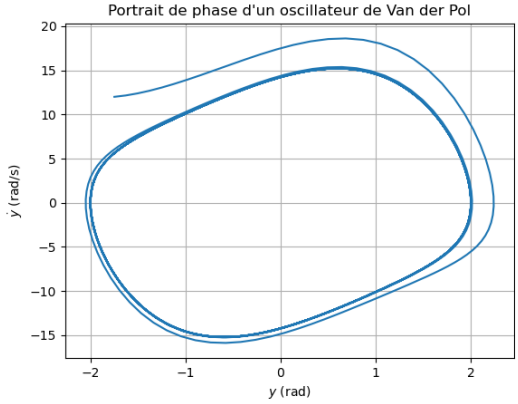


(b) *Le tracé « `fig8b.png` ».*

Figure 8



(a) *Le tracé « `fig9a.png` ».*



(b) *Le tracé « `fig9b.png` ».*

Figure 9

Le régime permanent reste le même, quelles que soient les conditions initiales.