TD 3 : quelques bibliothèques

Les sujets de TD ne sont ni relevés, ni notés. Ils sont volontairement conçus pour ne pas pouvoir être traités en une seule séance ; n'hésitez cependant pas, si vous le souhaitez, à tenter de les finir chez vous.

Rappel : respectez les conventions de style (PEP 8 et *Zen of Python*), écrivez des tests, documentez votre code.

Intégration d'équations différentielles

Si la résolution d'équations différentielles linéaires au premier ordre est chose facile, seule une toute petite partie des équations différentielles est en fait résoluble. Dès que les coefficients d'une équation différentielle linéaire ne sont plus constants voire, pire, que l'équation différentielle n'est plus linéaire, il devient rapidement impossible d'espérer obtenir des solutions exactes.

Au contraire, essayer d'approximer les solutions d'une telle équation par ordinateur est souvent beaucoup plus aisé. Programmer des algorithmes d'approximation est chose assez aisée — c'est le cas par exemple de la méthode d'Euler —, mais les méthodes donnant rapidement des résultats intéressants sont plus complexes. Aussi, nous aurons recours à des fonctions de bibliothèques extérieures.

1. Importer le module `numpy` ainsi que la fonction `scipy.integrate.odeint`. Chercher en ligne la documentation de cette dernière, puis résoudre et tracer la solution de l'équation différentielle $f' = f$ sur l'intervalle $[0, 1]$, et en déduire une valeur approchée de e.
2. Considérons un corps de masse P en chute libre dans un champ de gravité g, partant d'une vitesse nulle. L'on considère l'évolution de sa vitesse verticale au cours du temps : elle vérifie l'équation différentielle $\dot{v} = g - \frac{\alpha}{m}v^2$ où α désigne un coefficient de frottements quadratiques. L'on fixe pour la suite les valeurs $g = 9,81\ \mathrm{m \cdot s^{-2}}$ et $m = 100$ kg.
   (a) Pour α valant $0,01$, $0,1$, 1, 10 et $100\ \mathrm{kg \cdot m^{-1}}$, représenter l'évolution de v au cours du temps.
   (b) L'on admet que v admet une limite en $+\infty$. La calculer.
   (c) Écrire une fonction détermimant le temps $\tau_{0,01}$ au bout duquel l'objet est à 99 % de sa vitesse limite.
   (d) Tracer la courbe de $\tau_{0,01}$ en fonction de α.
3. Le *modèle de Lotka-Volterra* est un grand classique de la dynamique des populations. Dans ce modèle : l'on s'intéresse à l'évolution au cours du temps de deux populations x de lièvres des neiges et y de lynx (en millions d'individus), vérifiant :
   — Le taux de natalité des lièvres α est constant.
   — Le taux de mortalité des lièvres est proportionnel au nombre de lynx d'un facteur β.
   — Le taux de natalité des lynx est proportionnel au nombre de lièvres d'un facteur δ.
   — Le taux de mortalité des lynx γ est constant.
   (a) Écrire le système d'équations différentielles croisées vérifiées par le vecteur (x, y).
   (b) L'on fixe $\alpha = \frac{2}{3}$, $\beta = \frac{4}{3}$ et $\gamma = \delta = 1$. Représenter sur le même graphe l'évolution au cours du temps des populations pour une population initiale d'un million de lynx et trois millions de lapins.
   (c) Représenter maintenant l'évolution de système au cours du temps dans le plan (x, y) (courbe des phases). Que constate-t-on ?
   (d) Représenter les courbes des phases pour différentes populations initiales, puis à population initiale fixée en faisant varier le taux de natalité des lièvres.

4. L'on s'intéresse maintenant à la résolution d'équations différentielles d'ordre supérieur. En fait, une équation différentielle linéaire d'ordre n portant sur des fonctions à valeur réelle peut être transformée en une équation différentielle d'ordre 1 portant sur des fonctions à valeur dans $\mathbf{R}^n$.

   Considérons en effet l'équation :

$$y^{(n)} = f(y, y\prime, y\prime\prime, \ldots, y^{(n-1)}, t)$$

   En posant $z_j = y^{(j)}$ pour tout $j \in [\![0, d-1]\!]$, l'équation précédente équivaut au système :

$$\begin{cases} z_0' &= z_1 \\ z_1' &= z_2 \\ &\vdots \\ z_{n-2}' &= z_{n-1} \\ z_{n-1}' &= f(z_0, z_1, \ldots, z_{n-2}, z_{n-1}, t) \end{cases}$$

   Qui est un système du premier ordre dans $\mathbf{R}^n$, immédiatement résoluble par le module `odeint`.

   (a) Écrire une fonction effectuant la transformation précédente et appelant `odeint` pour résoudre une équation différentielle d'ordre n sur l'ensemble des fonctions à valeurs réelles. Cette fonction prendra pour paramètres `f` une fonction de $n+1$ paramètres réels, `n`, `y_0` une liste des conditions initiales représentées sous la forme $[y(t_0), y'(t_0), \ldots, y^{(n-1)}(t_0)]$ et `t` la liste des temps.

   (b) L'on rappelle qu'une équation différentielle linéaire d'ordre 2 peut se mettre sous la forme $y'' + \frac{\omega_0}{Q} y' + \omega_0^2 y = 0$. L'on fixe arbitrairement w_0 à 2π radians par seconde, $y(t_0)$ réel strictement positif quelconque et $y'(t_0)$ nul. Tracer sur un même graphe y et y' en fonction de t pour des valeurs de Q plus grandes, égales ou plus petites que $\frac{1}{2}$.

   (c) Tracer les courbes de phases (évolution du système dans le plan (y, y')) pour différentes valeurs de Q.

   (d) On considère un pendule double dont les deux axes sont de longueur respective ℓ_1 et ℓ_2 et les deux mobiles de masse respective m_1 et m_2. L'on note θ_1 et θ_2 les positions angulaires des mobiles. Des calculs fastidieux permettent de montrer que l'équation de ce pendule double est donnée par :

$$\begin{cases} \ell_1 \ddot{\theta_1} \cos(\theta_1 - \theta_2) + \ell_2 \ddot{\theta_2} - \ell_1 \dot{\theta_1}^2 \sin(\theta_1 - \theta_2) + g \sin\theta_2 &= 0 \\ (m_1 + m_2)\ell_1 \ddot{\theta_1} + m_2 \ell_2 \ddot{\theta_2} \cos(\theta_1 - \theta_2) + m_2 \ell_2 \dot{\theta_2}^2 \sin(\theta_1 - \theta_2) + (m_1 + m_2) g \sin(\theta_1) &= 0 \end{cases}$$

   Pour des valeurs de $g = 9,81\ \mathrm{m \cdot s^{-2}}$, $\ell_1 = \ell_2 = 0,5$ m, $m_1 = 0,5$ kg et $m_2 = 1$ kg avec une position initiale $\dot{\theta_1} = \dot{\theta_2} = 0$, $\theta_1 = \frac{2\pi}{3}$ et $\theta_2 = \frac{5\pi}{6}$, tracer l'évolution au cours du temps de θ_1, θ_2, $\dot{\theta_1}$ et $\dot{\theta_2}$. Tracer également la trajectoire du pendule dans le plan (x, y), c'est-à-dire les deux courbes définies par :

$$(x_1, y_1) = (\ell_1 \sin(\theta_1), -\ell_1 \cos(\theta_1)\ ; \quad (x_2, y_2) = (\ell_1 \sin(\theta_1) + \ell_2 \sin(\theta_2), -\ell_1 \cos(\theta_1) - \ell_2 \cos(\theta_2))$$

   (e) Constater qu'un changement du pas dans les temps change fortement la trajectoire observée. Ceci est dû à l'aspect chaotique du pendule double : une erreur mineure à un instant entraîne une évolution radicalement différente du système.

Traitement et compression d'images

La capacité à traiter des images fait partie des enjeux majeurs de la recherche en informatique en ce début de XXI$^{\text{e}}$ siècle. Les usages potentiels sont multiples : de nombreux scientifiques collectent des données en très grande quantité sous forme graphique et cherchent à les interpréter automatiquement. Citons par exemple l'observation de sous-sols par tomographie, de cellules météorologiques, du fond diffus

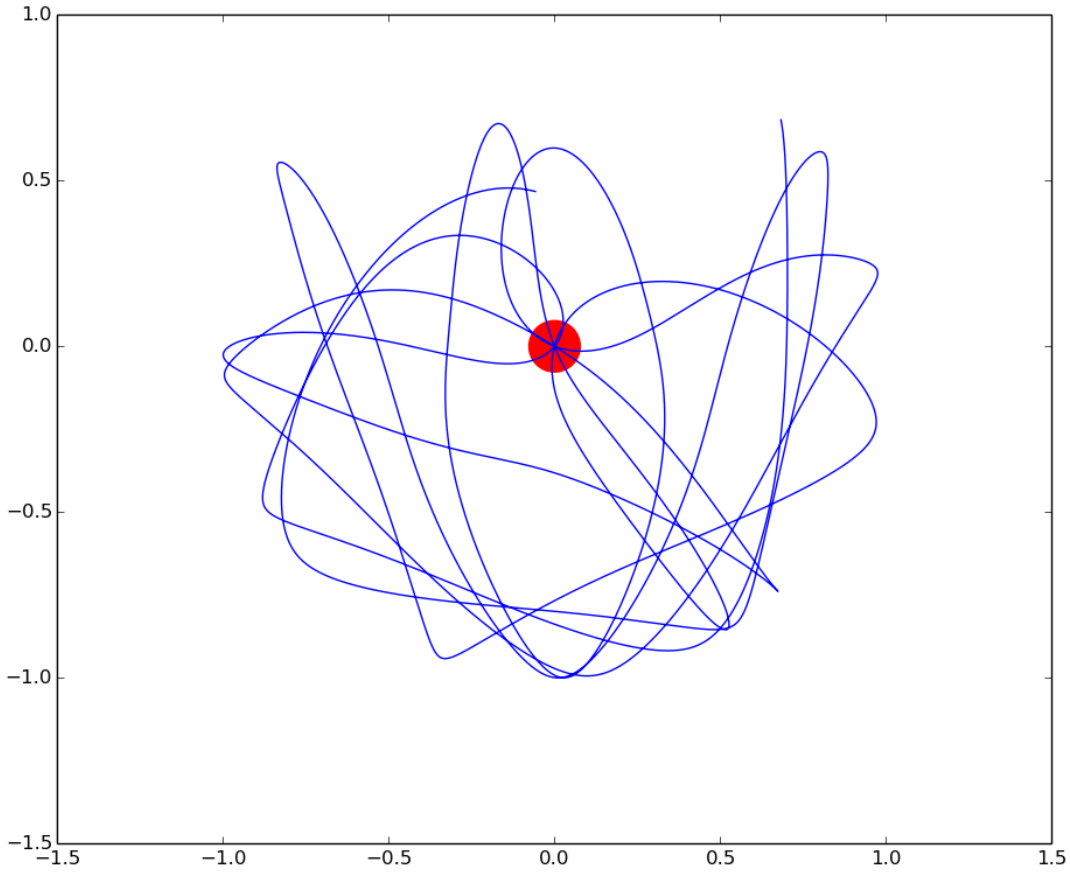

FIGURE 1 – Représentation de l'évolution du pendule double au cours du temps.

cosmologique en astrophysique, d'imperfections sur des matériaux, de la chaleur émise par l'activité humaine, des textes sur des archives papier détériorées, etc.

Dans la majorité des cas, les images sont des images *matricielles* ou *raster*. Cela signifie que l'image est découpée en une grille de *pixels* (*picture elements*), et qu'à chaque pixel est associé une information discrétisée. En d'autres termes, une image est une fonction de $[\![0, n-1]\!] \times [\![0, p-1]\!]$ dans M où $n \times p$ est la *définition* et M est un ensemble fini, généralement inclus dans $\mathbf{R}^d$. d sera appelé par la suite *dimension du champ* de l'image (vocabulaire non standard !).

Dans le cas d'une image en noir et blanc, M est simplement l'ensemble {noir, blanc} (identifié à $\{0, 1\}$). Pour une image par niveaux de gris, l'on choisit pour M généralement $[\![0, 255]\!]$, c'est-à-dire que chaque pixel peut prendre un niveau de gris parmi les 255 existants. Pour une image en couleurs, l'on choisit une trichromie : M vaut $[\![0, 255]\!]^3$, c'est-à-dire que chaque pixel porte un niveau de rouge, un niveau de vert et un niveau de bleu.

Une image est donc issue d'un échantillonnage dans l'espace (découpage en pixels) et d'intensité (les valeurs prises sont dans un ensemble fini). Cette perte d'informations permet toutefois de travailler avec un nombre fini d'images, chacune d'entre elles étant étant donc codée par une représentation binaire de taille finie. Une fois l'échantillonnage fait, l'image ne peut plus être altérée.

5. Récupérer l'image mise à disposition dans les documents partagés pour cette séance. Importer la bibliothèque `matplotlib.img` (sous l'alias `mpimg`), et utiliser la fonction `mpimg.imread` pour travailler sur l'image avec Python. Remarquez que la valeur de la couleur de chaque pixel n'est pas comprise dans

l'ensemble $[\![0, 255]\!]^3$, mais prend dans un sous-ensemble fini de $[0, 1]^3$, ce qui revient au même. Matplotlib peut afficher directement une image dont la dimension du champ est 3 : utiliser pour cela la commande `plt.imshow`.

6. (a) L'on rappelle que la composante rouge d'un pixel de coordonnées i, j est stockée à l'adresse $image[i][j][0]$. En une seule ligne, récupérer un tableau à deux dimensions contenant la composante rouge de chaque pixel. La commande `plt.imshow` permet également de représenter une image dont la dimension du champ est égale à 1. Pour ce faire, matplotlib utilise une *carte de pseudocouleurs*, c'est-à-dire qu'il associe à chaque valeur une unique couleur pour faciliter la visualisation. Par défaut, cette carte est une carte appelée *viridis*, adaptée pour représenter une image dont les pixels représentent des températures mesurées. Renseigner le paramètre `cmap` de la fonction aux valeurs `"hot"`, `"Greys"`, `"copper"`. Chercher dans la documentation d'autres cartes de couleurs existantes.

   (b) L'on souhaite maintenant visualiser seulement le niveau de rouge. Pour cela, reprendre l'image initiale avec une dimension de champ de 3, et fixer la deuxième et la troisième composante à 0 pour chaque pixel. Afficher l'image : quelle est la différence avec le rendu de la question précédente ?

   (c) L'on souhaite aplanir l'image en niveaux de gris. L'on utilise pour cela la formule suivante : $Y' = 0,299r + 0,585g + 0,114b$ où r, g et b sont respectivement les composantes rouge, verte et bleue d'un pixel. Construire à partir de l'image de départ une image à champ d'une dimension représentant la luminance, et l'afficher avec la carte de couleurs `"greys"`. L'inverser en appliquant à chaque pixel l'opération $x \longmapsto 1 - x$.

7. L'on définit également pour un pixel sa chrominance bleue $C_b = -0,1687 * r - 0,3313g + 0,5b + 0,5$ et sa chrominance rouge $C_r = 0,5r - 0,4187g - 0,0813b + 0,5$. Ces deux grandeurs sont liées à la perception des couleurs par l'œil.

   (a) Écrire une fonction calculant la luminance, la chrominance bleue et la chrominance rouge d'un pixel connu sous sa forme rouge, vert, bleu. En utilisant la fonction `np.linalg.solve`, écrire également sa fonction réciproque. Ces deux fonctions permettent de représenter différemment le même espace de couleurs.

   (b) Définir ensuite une fonction qui, à partir d'un `ndarray` à trois dimensions représentant les teintes rouge, verte et bleue, renvoie un `ndarray` à trois dimensions représentant la luminance, la chrominance bleue et la chrominance rouge chaque pixel de l'image. Écrire également la fonction réciproque. *Attention : l'on évitera d'écrire des boucles à plusieurs millions de tours...*

   (c) Cette photo est légèrement sous-éclairée. Lui appliquer la correction $Y' \longmapsto \sqrt{Y'}$. *Malheureusement, cette transformation un peu brutale fait apparaître des* artefacts.

8. Il est bien connu que l'œil est beaucoup plus sensible à la luminance qu'à la chrominance, du fait de la présence de bâtonnets en nombre largement supérieur aux cônes dans la rétine. Nous allons donc *sous-échantillonner la chrominance* de notre image.

   (a) L'on se donne un paramètre $k \in \mathbf{N}$. Écrire une fonction qui, à partir d'un `ndarray` de taille $n \times p$, renvoie un `ndarray` de taille $\left\lfloor \frac{n}{k} \right\rfloor \times \left\lfloor \frac{p}{k} \right\rfloor$ dont l'élément d'indice `i, j` est la moyenne des valeurs des éléments d'indice `(k*i, k*j)` à `(k*i + k - 1, k*j + k - 1)`.

   (b) Écrire une fonction qui, à partir d'un `ndarray` représentant la luminance et de deux `ndarray` représentant les chrominances sous-échantillonnées d'un facteur k, affiche une image reconstituée. Pour une valeur $k = 2$, la qualité de l'image semble-t-elle vraiment avoir été détériorée ? Même question pour les valeurs $k = 5$, $k = 10$, $k = 20$, $k = 50$ et $k = 100$.

   (c) Quelle est la quantité de mémoire économisée en effectuant cette compression ?