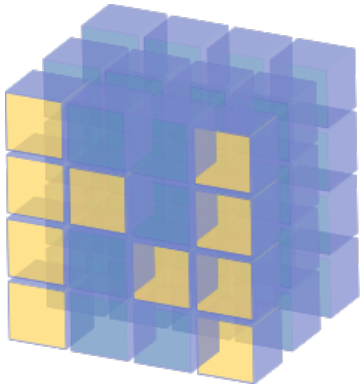
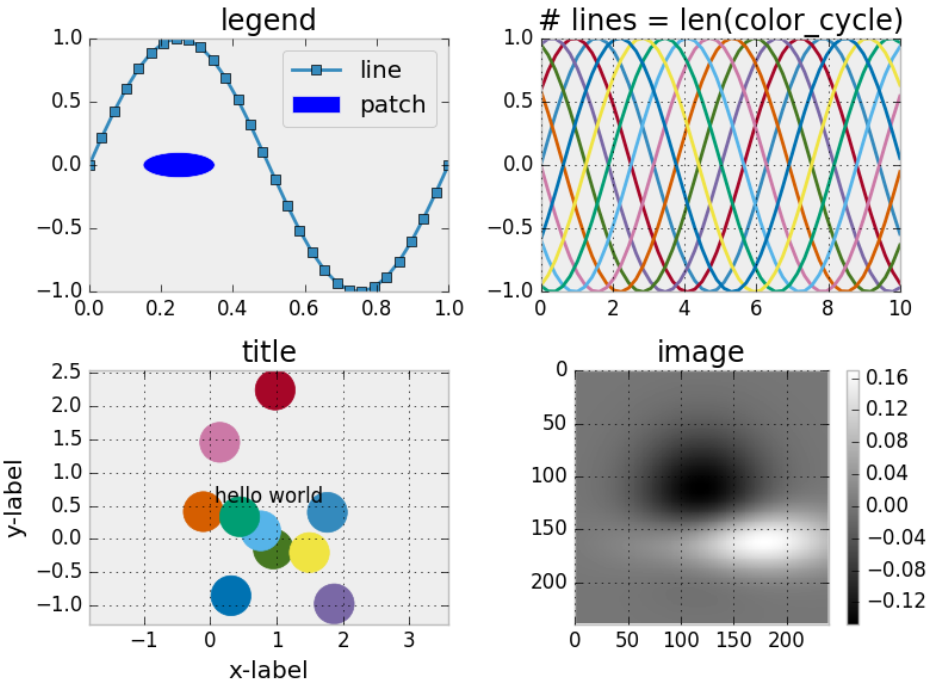
Introduction à NUMPY et MATPLOTLIB



NumPy

⚠

Ce document est un TP/cours et il est à connaître parfaitement : vous devez savoir manipuler des tableaux NUMPYet faire des tracés de courbes avec MATPLOTLIB.



Dans ce TP/cours, nous allons découvrir et utiliser la bibliothèque de calcul scientifique `numpy` et la bibliothèque `matplotlib.pyplot` qui permet de tracer des graphiques. Commençons donc tout d'abord par importer ces deux bibliothèques. Comme nous allons en faire usage constant, on définit une abréviation (un alias), qui est conventionnellement utilisée :

PYTHON

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

1 À la découverte de NUMPY

NUMPY est une bibliothèque de calcul scientifique en PYTHON. Elle introduit des tableaux multidimensionnels et des fonctions efficaces pour les manipuler, implémentées directement en C. Le calcul vectoriel permet de réaliser les opérations vectorielles et matricielles très rapidement, sans utiliser (directement) de boucles `for` ou `while`.

1.1 Tableaux NUMPY

NUMPYest construit autour d'une brique de base qui est le tableau [1], au vrai sens informatique du terme : données de type homogène, accès et modification des éléments en temps constant, taille fixe et représentation efficace en mémoire.

Le nom de ces tableau est `array` (dont le type est plus précisément `np.ndarray`) et ils ont tous le même type, appelé `dtype`. Le constructeur `np.array` prend en argument un objet à convertir en tableau (un itérable comme une liste ou un tuple) et créé le tableau NUMPY correspondant.

1. Tester directement en console et comprendre les résultats :

PYTHON
```
>>> t1 = np.array([1, 2, 3, 4, 5, 6])
>>> t1
>>> t1.dtype
>>> t2 = np.array([1, 2, 3.5])
>>> t2
>>> t2.dtype
>>> t3 = np.array([1, 2, 3], dtype=np.float64)
>>> t3
>>> t4 = np.array([1, 2, 3], dtype=np.float16)
>>> t4
>>> np.who()
>>> help(np.array) # Lire rapidement, sans passer trop de temps
```

2. Que représentent les types `np.float16`, `np.float32` et `np.float64` ?
3. Essayer et commenter :

PYTHON
```
>>> L = [-12, 10 ** 6]
>>> t5 = np.array(L, np.int64)
>>> t5
>>> t6 = np.array(L, np.int8)
>>> t6
>>> t7 = np.array(L, np.float32)
>>> t7
>>> t8 = np.array(L, np.float16)
>>> t8
```

4. Définir un tableau `Numpy` de type `np.int32` contenant dans cet ordre les entiers 4, 2, 6.
5. La taille d'un tableau s'obtient avec `len` comme pour les listes. Vérifiez.

1. Multidimensionnel, mais cela, c'est pour plus tard.

1.2 Accès aux éléments et tranches

On accède aux éléments d'un tableau, et on les modifie comme pour les listes. On peut aussi faire des tranches sur les tableaux comme pour les listes.

6. Prévoir *puis* vérifier :

PYTHON
```
>>> t = np.array([1, 2, 3, 4])
>>> t[1]
>>> t[1] = 0
>>> t
>>> t[1:3]
>>> t[1:]
>>> t[::2]
>>> t[1::2]
```

⚠ Contrairement aux listes, aux tuples ou aux chaînes de caractères, les tranches pour les tableaux NUMPY *ne* créent *pas* un *nouveau* tableau, mais une *vue* sur le tableau initiale. Une modification de la tranche entraîne donc une modification du tableau initial.

7. Vérifier :

PYTHON
```
>>> vue = t[1:3]
>>> vue
>>> vue[0] = 42
>>> vue
>>> t
```

Contrairement aux listes, on peut également extraire un sous-tableau en donnant un liste (ou un tableau NUMPY) contenant les indices à extraire. En revanche il y a création d'un nouveau tableau, ce n'est plus une vue.

8. Rappeler, pour une liste `liste`, ce que renvoie `liste[a:b:c]`. Que représentent alors les entiers `a`, `b` et `c` ?
9. Essayer et expliquer :

PYTHON
```
>>> liste = [1, 2, 3, 4]
>>> liste[[1, 3]]
>>> tableau = np.array(liste)
>>> tableau[[1, 3]]
```

Nous verrons que ceci est très utile pour la réalisation de *masques*.

1.3 Constructeurs de tableaux

1.3.1 `arange`

On peut créer des tableaux contenant des suites arithmétiques avec `np.arange` qui fonctionne comme `range` mais qui renvoie un `array`.

10. Prévoir et vérifier :

PYTHON
```
>>> np.arange(9)
>>> np.arange(2, 7)
>>> np.arange(2, 7, 2)
```

11. Créer le tableau NUMPY des nombres pairs entre 0 et 10 inclus, de type `np.int32`, en utilisant `np.array` et `range` d'un part et directement `np.arange` d'autre part.

1.3.2 `linspace`

La commande `np.linspace(a, b, n)` renvoie le tableau contenant la subdivision régulière de $[a, b]$ en n termes, c'est-à-dire n nombres régulièrement espacés dont le premier vaut a et le dernier b.

PYTHON
```
>>> np.linspace(0, 1, 5)
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

12. Comment obtenir `np.linspace(a, b, n)` avec `np.arange` ? Vérifier.
13. Comment obtenir `np.arange(a, b, pas)` avec `np.linspace` ? Vérifier.
14. Dans quels cas les bornes sont-elles incluses ou exclues ?

✔ On utilise plutôt `np.linspace` si l'on cherche à subdiviser un intervalle en connaissant le nombre de subdivisions. On utilise plutôt `np.arange` si on connaît le pas.

1.3.3 Quelques tableaux particuliers

- `np.zeros(n)` permet de créer un tableau de taille n initialisé avec la valeur 0..
- `np.ones(n)` permet de créer un tableau de taille n initialisé avec la valeur 1..
- `np.random.rand(n)` permet de créer tableau de taille n à coefficients aléatoires uniformes sur $[0, 1[$.

15. Tester.
16. Quel est le type des tableaux créés ?

1.4 Calculs mathématiques et fonctions vectorialisées

1.4.1 Constantes

Les constantes mathématiques usuelles sont accessibles avec NUMPY :

PYTHON
```
>>> np.e
>>> np.pi
```

1.4.2 Fonctions

Un des atouts majeurs des tableaux NUMPY est de pouvoir leur appliquer directement des fonctions dites *vectorialisées* qui vont s'effectuer sur chaque élément du tableau, et ce, de façon optimisée. Pas besoin de boucle !

✔ Appliquée à un tableau, les fonctions sont appliquées à tous les cœfficients.

Les fonctions mathématiques de la bibliothèque NUMPY sont vectorialisées et donc à privilégier. N'hésitez pas à utiliser l'aide si vous avez un doute sur ce que réalise une de ces fonctions.

`np.sqrt` ($\sqrt{\cdot}$)	`np.sin`	`np.cos`	`np.tan`	`np.arcsin`
`np.arccos`	`np.arctan`	`np.sinh`	`np.cosh`	`np.exp`
`np.log`	`np.log10`	`np.floor` ($\lfloor\cdot\rfloor$)	`np.ceil` ($\lceil\cdot\rceil$)	etc.

Les opérations de base `+`, `*`, `/`, `//`, `**`, `%` sont également vectorialisées.

17. Tester :

PYTHON
```
>>> t = np.array([1, 2, 3])
>>> t
>>> t + t
>>> t * t
>>> t ** 3
>>> 1 / t
>>> np.sin(t)
>>> np.log(t)
>>> np.exp(t)
>>> t % 2
```

18. Créer un tableau avec le cosinus des 10 angles régulièrement espacées entre 0 et 2π.

1.4.3 Vectorialisation

19. Essayer et commenter

PYTHON
```
>>> t = np.array([1, 2, 3])
>>> import math
>>> math.sin(t)
```

On peut vectorialiser une fonction pour qu'elle s'applique terme à terme avec `np.vectorize`.

20. Définir une fonction `fact` qui calcule la factorielle d'un entier $n \geq 0$.
21. Essayer :

PYTHON
```
>>> t = np.arange(10)
>>> fact(t)
>>> afact = np.vectorize(fact)
>>> afact(t)
```

1.4.4 Fonctions spécifiques aux tableaux

22. Tester et comprendre :

PYTHON
```
>>> t = np.linspace(2, 1, 5)
>>> t
>>> np.sum(t)
>>> t.sum()
>>> np.prod(t)
>>> t.prod()
>>> np.mean(t)
>>> t.mean()
>>> np.min(t)
>>> t.min()
>>> np.max(t)
>>> t.max()
>>> np.sort(t)
>>> t
>>> t.sort()
>>> t
```

2 Tracés de graphes

Les facilités sur la manipulation des tableaux NUMPY permettent de tracer tout aussi facilement des graphes.

MATPLOTLIB offre de nombreuses possibilités pour faire des tracés graphiques. On présente ici une infime partie des possibilités, il faut prendre l'habitude de consulter l'aide (en ligne, c'est plus confortable) pour personnaliser ses tracés.

La plupart des fonctions possèdent des arguments avec des valeurs par défaut. On n'a pas besoin de préciser ces arguments si on ne veut pas les changer, et l'ordre dans lequel on les appelle n'importe pas si on les désigne par leur nom : `fonction(arg1=val1, arg2=val2, ...)`.

Quelques commandes de base :

- `plt.figure(titre)` crée un figure (fenêtre) avec le titre précisé. Si on n'utilise pas cette fonction, la figure est créée automatiquement ;
- `plt.plot(x, y, label=<chaine>)`, où `x` et `y` sont des tableaux, calcule la courbe correspondante, `label` permet de préciser une légende (qui peut contenir du LaTeX). Il faut

appeler `plt.legend()` pour afficher la légende, l'argument optionnel `loc='best'` permet de laisser python choisir l'emplacement le moins gênant ;

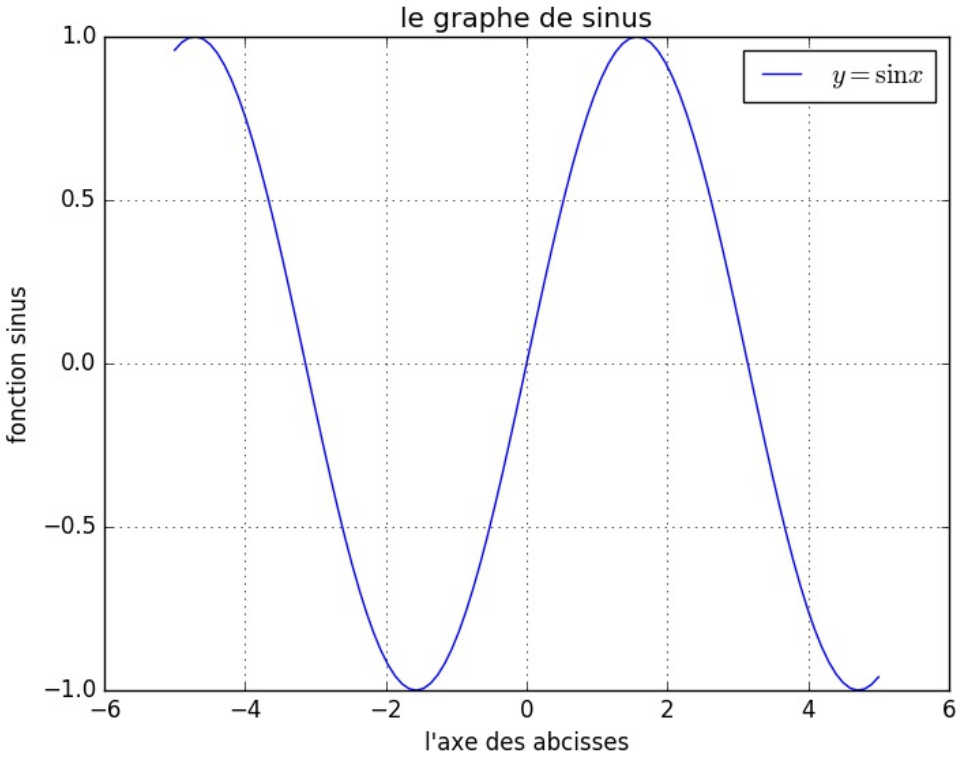
- `plt.xlabel(<chaine>)` et `plt.ylabel(<chaine>)` permettent de préciser un titre pour chaque axe ;
- `plt.title(<chaine>)` permet de préciser un titre ;
- `plt.show()` permet d'afficher le graphe ;
- `plt.savefig(<nom_fichier>)` permet d'enregistrer le graphe dans un fichier image ;
- `plt.grid()` permet de voir une grille ;
- `plt.axis([xmin, xmax, ymin, ymax])` ou `plt.xlim(xmin, xmax)` ainsi que `plt.ylim(ymin, ymax)` permettent de préciser l'échelle des axes.

23. Essayer :

PYTHON

```
x = np.linspace(-5, 5, 100)
plt.plot(x, np.sin(x), label='$y=\sin x$')
plt.ylabel('fonction sinus')
plt.xlabel("l'axe des abcisses")
plt.title('le graphe de sinus')
plt.legend(loc='best')
plt.grid()
plt.show()
```

Vous devriez obtenir :



24. Tracer le graphe de la fonction $f : x \mapsto \dfrac{1}{1+25x^2}$ sur $[-1,1]$. Faire apparaître une légende, et une grille. Enregistrer le graphe dans un fichier.

25. Tracer la fonction $x \mapsto e^{e^x}$ sur $[0,5]$. S'arranger pour qu'on voie quelque chose !

L'aide sur `plt.plot` donne les diverses options possibles pour le tracer.

Par exemple, un troisième paramètre à `'r'` donnera une courbe rouge, `'b'` donnera une courbe bleue Avec `color='\#24e5cf'` on peut préciser une couleur en RGB (hexadécimal).

On peut marquer les points avec l'option `marker=` : `'o'` pour des ronds, `'+'` pour des plus, `'*'` pour des étoiles, etc.

Le style de ligne peut aussi être précisé avec `linestyle=` ou `ls=` : `'-'` pour continu, `'--'` pour tirets, `':'` pour pointillés, `'-.'`, etc.

Enfin, on peut utiliser un argument condensé : par exemple `'bo'` ou `'g-v'` ou `'--'`, etc.

26. Essayer :

PYTHON

```
x = np.linspace(-5, 5, 100)
plt.plot(x, np.sin(x), marker='o',
         label='Sinus')
plt.plot(x, np.cos(x), 'rv:',
         label='Cosinus')
plt.title("Fonctions trigo")
plt.legend(loc='best')
plt.show()
```

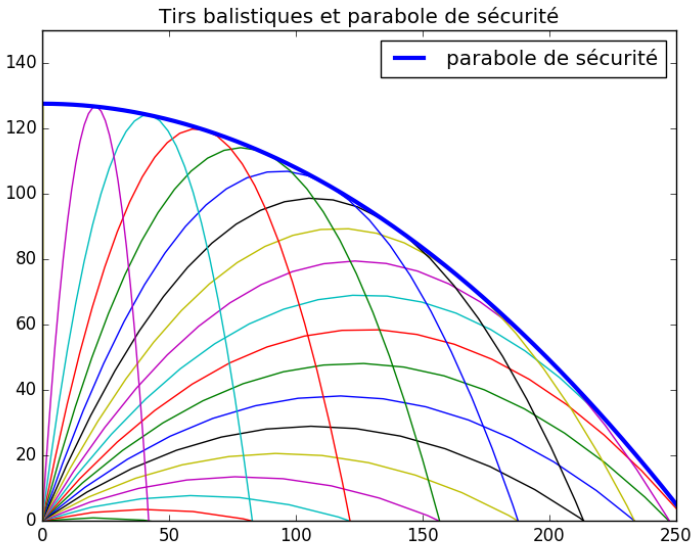
27. **Tirs balistiques** : on jette un projectile M à partir du point O (origine) avec une vitesse v_0 constante, mais faisant un angle variable θ avec l'horizontale. Si le projectile est soumis seulement à la gravitation, la trajectoire est connue : ce sera une parabole. On démontre même de façon classique que l'ensemble des trajectoires est « enveloppée » par une parabole, dite de sécurité.

Le principe fondamental de la dynamique donne, avec une masse $m = 1$, $x'' = 0$ et $y'' = -g$ soit $x(t) = (v_0 \cos\theta)t$ et $y(t) = (v_0 \sin\theta)t - \frac{g}{2}t^2$.

Superposer sur un même graphe les trajectoires pour une vingtaine d'angles entre 0 et $\frac{\pi}{2}$ (Ne garder que $y \geqslant 0$!), avec $g = 9{,}8$, $v_0 = 50$ (USI).

Tracer également sur le même graphe la parabole de sécurité d'équation $y = \dfrac{v_0^2}{2g} - \dfrac{g}{2v_0^2}x^2$.

Vous devriez obtenir quelque chose comme :



On peut aussi séparer le grapheur en plusieurs sous-figures avec `plt.subplot(xyz)` ou `plt.subplot(x,y,z)` où `x` est le nombre de lignes, `y` le nombre de colonnes, `z` le numéro de la figure.

28. Essayer :

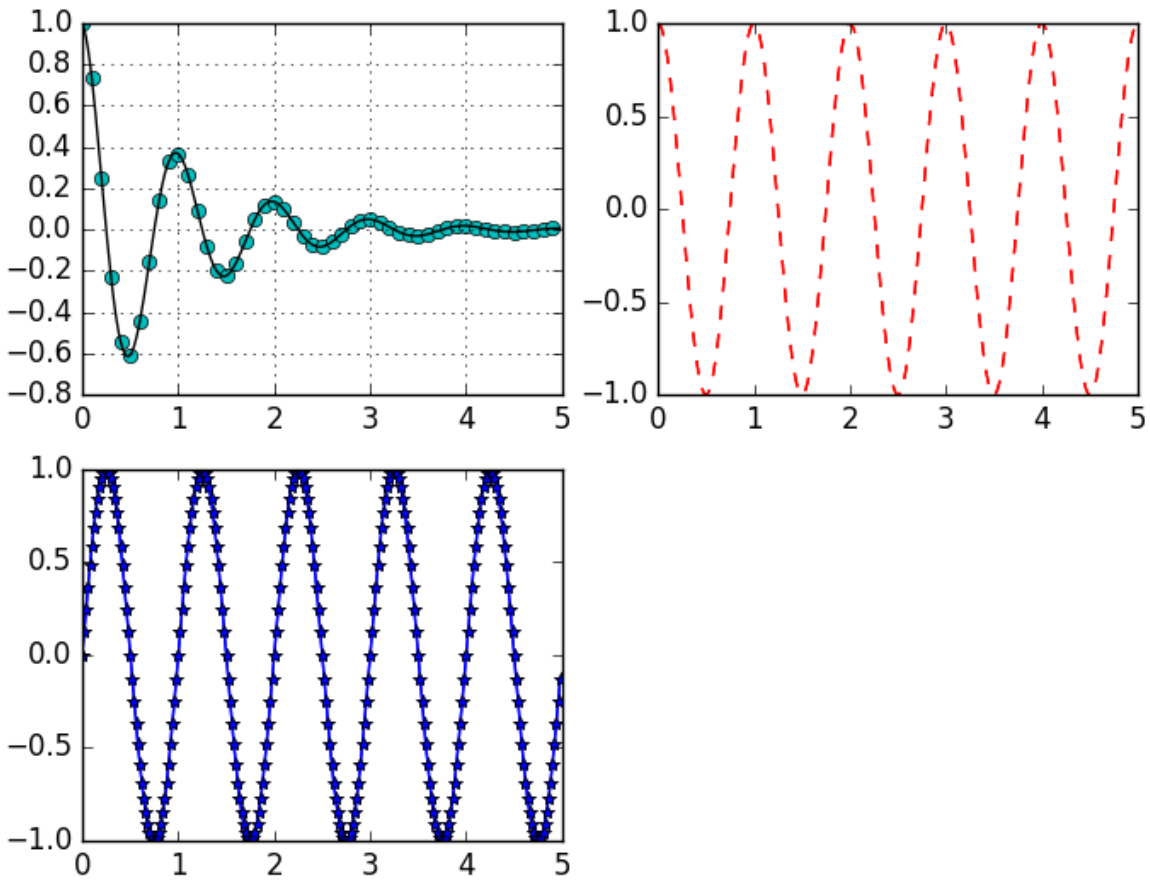
PYTHON

```
def f(t) :
  return np.exp(-t) * np.cos(2*np.pi * t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.subplot(221)
plt.plot(t1, f(t1), 'co')
plt.plot(t2, f(t2), 'k')
plt.grid()
plt.subplot(222)
plt.plot(t2, np.cos(2*np.pi * t2), 'r--')
plt.subplot(223)
plt.plot(t2, np.sin(2*np.pi * t2), 'b*-')
plt.show()
```

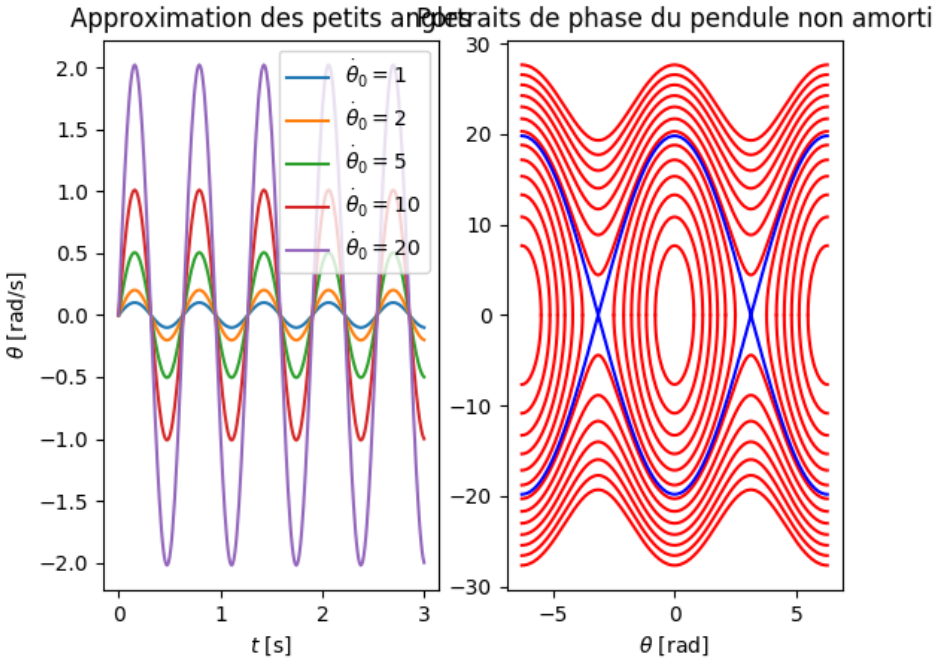
Vous devriez obtenir :

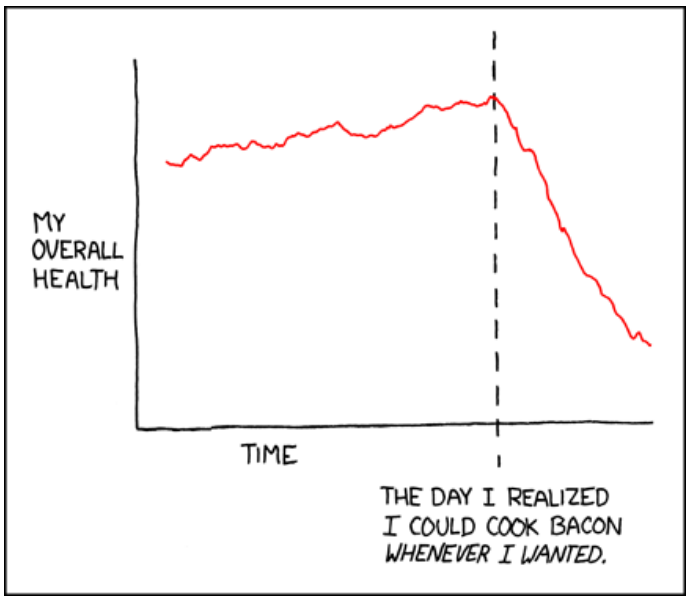


La conservation de l'énergie mécanique pour un pendule simple permet d'obtenir l'équation : $E = \frac{1}{2}mL^2\dot{\theta}^2 + mgL(1-\cos\theta)$ d'où on peut tirer $\dot{\theta}$ en fonction de θ pour tracer un portrait de phase $\dot{\theta} = \pm\omega_0\sqrt{2(C-1+\cos\theta)}$, avec $\omega_0 = \sqrt{\frac{g}{L}}$. On prendra $L = 0.1$ m.

29. Superposez quelques portraits de phase ($\dot{\theta}$ en fonction de θ) en faisant varier $C \geqslant 0$. Observer le cas limite $C = 2$ que vous devez avoir étudié en sciences physiques.
30. Avec l'approximation des petits angles, on sait résoudre l'équation différentielle, et on trouve, pour $\dot{\theta}(0) = 0$, $\theta(t) = \theta_0\cos(\omega_0 t)$. Tracer, dans des subplots d'une même figure, quelques θ en fonction de t et les portraits de phase de la question précédente.

Vous devriez obtenir :





https://xkcd.com/418/