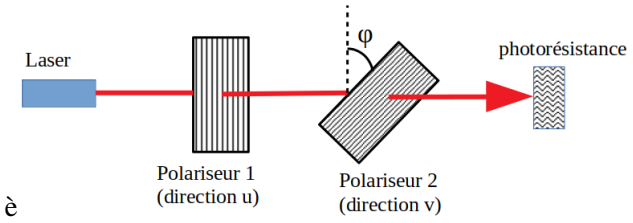
— Votre compte-rendu devra être déposé sur Moodle (cours "Physique numérique") au plus tard le 15 février 2021.
https://moodlesciences.univ-brest.fr/moodle/course/view.php?id=367
— Regrouper l'ensemble de vos documents (compte-rendu, graphes, scripts PYTHON (fichiers `.py`) dans une archive (format `.zip` par exemple) pour ne déposer **qu'un seul** fichier.
Votre compte rendu devra être un fichier au format `.pdf` (les formats `doc` ou `docx` ne sont pas autorisés) dans lequel vous répondrez aux différentes questions en y incluant vos résultats (données numériques, graphes, ...) et commentaires. N'oubliez pas de commenter largement vos programmes en précisant aussi les noms et n° complets des exercices auxquels ils se rapportent.
— Avant de quitter la salle, n'oubliez pas de sauvegarder votre travail sur une clé USB ou sur l'espace de stockage de votre ENT.

1 Lecture de fichiers, tracé et ajustement. Exemple : loi de Malus



è

On a utilisé une photorésistance (cellule photoconductrice) pour mesurer l'intensité lumineuse d'une onde initialement polarisée linéairement selon la direction u et qui a traversé un polariseur linéaire selon la direction v. On a fait varier l'angle φ entre le polariseur et la direction de polarisation initiale ($\varphi = \widehat{u,v}$). L'intensité de la lumière éclairant la photorésistance utilisée, de résistance R, est proportionnelle à $1/R^{1,6}$.

1-1. Le fichier de mesure, `exp_loi_Malus1py.txt`, que vous téléchargerez sur Moodle, est un fichier ASCII à deux colonnes : la première colonne contient les angles φ en degrés et la deuxième colonne les résistances mesurées R (en kOhms) de la cellule comme indiqué dans les commentaires incorporés au fichier.

Dans un premier temps on souhaite tracer l'intensité lumineuse en fonction de l'angle.

Utiliser la fonction `loadtxt()` du module `numpy` pour lire le fichier `exp_loi_Malus1py.txt`.
— Extraire la colonne 1 (angle) et la colonne 2 (résistance).
— Calculer l'intensité et la présenter graphiquement en utilisant des cercles rouges (tracer I en fonction de φ).

1-2. On souhaite à présent vérifier la loi de Malus : $I = I_0 \cos^2(\varphi)$.
Expérimentalement, un décalage (I_1, `shift`) de l'intensité est possible, ainsi qu'un déphasage angulaire (δ, `phase`) la loi de Malus est donc modifiée en : $I = I_1 + I_0 \cos^2(\varphi + \delta)$. Vous définirez donc la fonction `malus(angle, amplitude,phase,shift)` qui prend en compte ces 2 paramètres supplémentaires. Attention l'argument de `np.cos()` doit être en radians !
On souhaite déterminer les paramètres d'ajustement à l'expérience : (`amplitude, phase, shift`). Vous utiliserez la fonction `curve_fit()` du module scipy.optimize :

```
from scipy.optimize import curve_fit
pfit,pcov = curve_fit(malus, angle, Intens)
```

Utiliser les paramètres trouvés par `curve_fit()` et la fonction `malus()` pour tracer la courbe théorique ajustée. Ajoutez une légende indiquant les valeurs de ces paramètres et les titres des axes.

1-3. En utilisant l'aide de `curve_fit()` préciser la signification de la matrice `pcov`.

2 Résolution d'équations différentielles

On rencontre fréquemment de telles équations à résoudre dans de nombreux domaines d'applications (mécanique, chimie, biologie, ...)
PYTHON permet de résoudre numériquement toute équation (ou système d'équations) différentielle du type $y' = f(y,t)$ complétée par des conditions initiales $y(t_0) = y_0$ (sous réserve d'existence et d'unicité de la solution). L'instruction correspondante à importer depuis `scipy` (`from scipy.integrate import odeint`), s'appelle `odeint` et nécessite la rentrée de trois arguments : la fonction f (toujours à deux arguments y et t), la donnée initiale y_0, le temps initial t_0 et les instants de calcul de la solution t.
Grâce à ce formalisme, il est possible de résoudre avec PYTHON toute équation d'ordre supérieur à 1, en récrivant celle ci comme un système d'équations d'ordre 1.

Un exemple d'équation différentielle est celle qui régit la décharge d'un condensateur de capacité ($C = 1$ μF) à travers une résistance ($R = 1$ kΩ) : $RC\frac{du}{dt} = -u$. Voilà comment la résoudre :

```
R,C=1e3,1e-6 # définition des paramètres électriques (Résistance, capacité)
def condensateur(u,t):
    """Définition de la fonction qui contient l'équation
    RCdu/dt = -u (u= tension) """ # Docstring de la fonction
    dudt=- u/(R*C)
    return dudt
u0=1 #tension initiale
t = np.linspace(0, 3, 21)*1e-3 # temps en s
sol = odeint(condensateur, u0, t) # solution de l'équa. diff. avec ode
#------tracé du résultat de odeint
plt.plot(t, sol[:, 0], 'bo', label='U(t) de odeint')
#------tracé résultat analytique
plt.plot(t, u0*np.exp(-t/(R*C)), 'r', label='U(t) analytique')
```

Même si l'équation ne comporte pas la variable t au second membre, il est important de noter que la fonction correspondante devra cependant avoir t pour second argument.
Un autre exemple, la série d'instructions :

```
def fct(y,t):
  """ Système de 2 équations différentielles liées"""
  dy0dt=np.cos(t)*y[1]
  dy1dt=y[0]-y[1]
  return (dy0dt,dy1dt)
yini=[10.,2.] #cond. initiales y_0(0) et y_1(0)
t = np.linspace(0,30,301)
sol = odeint(fct,yini,t) # matrice des solutions de l'équa. diff
```

permet de calculer et de stocker dans la variable *sol* les solutions du système de 2 équations différentielles couplées :

$$\begin{aligned} y_0'(t) &= \cos(t)\, y_1(t) \qquad (y_0(0) = 10) \\ y_1'(t) &= y_0(t) - y_1(t) \qquad (y_1(0) = 2) \end{aligned}$$

aux instants $t = 0, 0.1, 0.2, \cdots$ jusqu'à $t = 30$. Par exemple, pour tracer la solution en fonction de t : $y_0(t)$, on pourra alors taper `y0=sol[:,0]` puis `plt.plot(t,y0)`.

2-1. Exercice Résoudre pour $t \in [0, 10]$: $y' = -y + ty^2$ avec $y(0) = 1$ (Bernoulli d'ordre 2)
et tracer la solution obtenue y en fonction de t. Comparer à la solution analytique $y(t) = \dfrac{1}{1+t}$.

2-2. Exercice Résoudre l'équation de la balistique à une dimension $\frac{d^2z}{dt^2} = -g$ avec pour conditions initiales $z(0) = 0$; $v_z(0) = 10$ m/s; on prendra $g = 9,8$ m/s^2 (accélération de la pesanteur à la surface de la Terre). A l'aide de la commande `subplot` représenter la position du projectile d'une part et sa vitesse d'autre part, en fonction du temps entre 0 et 2 secondes.

2-3. Modèle épidémiologique SIR

Références :

— Billet de Blog de D. Louapre :
https://scienceetonnante.com/2020/03/12/epidemie-nuage-radioactif-et-distanciation-sociale/

— Image des mathématiques (CNRS) :
https://images.math.cnrs.fr/Modelisation-d-une-epidemie-partie-1.html

Ce modèle épidémiologique est un modèle simple dans lequel la population est compartimentée : personnes saines (S), infectées (I) et rétablies (R). On modélise deux phénomènes simples : les personnes infectées vont contaminer les personnes saines et les personnes infectées vont progressivement guérir.

Schématiquement on a :

$$\boxed{S} \xrightarrow{\beta} \boxed{I} \xrightarrow{\gamma} \boxed{R}$$

β est le taux de transmission, γ est le taux de guérison.

Les paramètres β et γ sont liés à la durée d de la maladie, aux nombres moyens de contacts quotidiens C entre personnes et à la probabilité P qu'un contact entre une personne infectée et une personne saine conduise à la transmission de la maladie.

Un paramètre très souvent utilisé, appelé « R_0 » est le nombre moyen de personnes qu'un individu infecté va contaminer, il vaut $R_0 = C \times P \times d$.

On a donc : $\beta = C \times P = R_0/d$, et $\gamma = 1/d$

Le système d'équations différentielles est donc :

$$\begin{cases} \frac{\mathrm{d}S(t)}{\mathrm{d}t} &= -\beta S(t)I(t) \\ \frac{\mathrm{d}I(t)}{\mathrm{d}t} &= \beta S(t)I(t) - \gamma I(t) \\ \frac{\mathrm{d}R(t)}{\mathrm{d}t} &= \gamma I(t) \end{cases}$$

Résoudre ce système d'équations différentielles dont les solutions $S(t)$, $I(t)$ et $R(t)$ sont les proportions des différentes populations. Pour la Covid-19, prenez les valeurs typiques : $d = 10$ jours, $P = 0,5\%$. Prendre les conditions initiales suivantes : $I_i = I(t = 0) = 1000/N$ et $R_i = R(t = 0) = 0$ et bien sûr $S_i = 1 - I_i - R_i$.

Tracez l'évolution temporelle ($t_{max} = 180$ jours) des trois populations (saines, infectées, rétablies), en prenant comme population totale $N = 70$ millions (France).

Testez différentes valeurs de C (nombre de contacts quotidiens) : $C = 50, 35, 20$ pour modifier la valeur du R_0 et commentez les courbes obtenues et notamment comment on peut obtenir le fameux « aplatissement de la courbe » des malades.

3 Fonctions périodiques et séries de Fourier (Pramode C.E., Linux Gazette n°98 – Janvier 2004)

On démontre que toute fonction $g(t)$, périodique de période T (de pulsation $\omega = 2\pi/T$ et de fréquence $f = 1/T$) et satisfaisant à certaines conditions de continuité et de dérivabilité, peut se décomposer en une somme de fonctions sinusoïdales dites "SÉRIES DE FOURIER" :

$$g(t) = a_0 + \sum_{n=1}^{\infty}[a_n \cos(n\omega t) + b_n \sin(n\omega t)]$$

avec :

$$a_0 = \frac{1}{T}\int_{-T/2}^{+T/2} g(t)dt; \quad a_n = \frac{2}{T}\int_{-T/2}^{+T/2} g(t)\cos(n\omega t)\,dt; \quad b_n = \frac{2}{T}\int_{-T/2}^{+T/2} g(t)\sin(n\omega t)\,dt$$

Nous rencontrons des "signaux" partout. Le haut-parleur d'un PC génère du son en convertissant des signaux électriques en vibrations. Nous voyons des objets autour de nous parce que ces objets font rebondir des signaux lumineux vers nos yeux. Nos postes de télévision et de radio reçoivent des signaux électromagnétiques. Nous sommes immergés dans un "océan de signaux" ! L'analyse des signaux est donc d'une importance cruciale dans de nombreuses branches de la science et de l'ingénierie. La majorité des choses complexes de ce monde peuvent s'expliquer en se fondant sur d'autres, plus simples. La perception de Joseph Fourier était que des signaux variant de façon complexe dans le temps peuvent être exprimés sous la forme d'une combinaison de courbes sinus/cosinus simples de fréquence et d'amplitude variables. Nous vérifierons cette hypothèse en représentant graphiquement quelques équations simples avec l'aide de PYTHON.

Commençons par une simple somme de signaux sinusoïdaux

```
In[]:  t = np.linspace(-np.pi,np.pi,241)
In[]:  y2Fourier = np.sin(t)-(1/2)*np.sin(2*t)
In[]:  plt.plot(t, y2Fourier)
```

Il n'y a presque aucune indication ici que quelque chose d'intéressant va se produire. À présent, essayons une représentation graphique de : `y3Fourier = np.sin(t)-(1/2)*np.sin(2*t)+(1/3)*np.sin(3*t)`
Continuons à ajouter des termes à la série ($N = 5$ pour 4 termes, etc) :

```
yFourier = sum([(-1)**(n+1)*np.sin(n*t)/n for n in range(1,N)])
```

3-1. Exercice : Trouver la fonction $g1(t)$ correspondant à la série de Fourier `yFourier`. Tracer sur un même graphe, la fonction et sa série de Fourier `yFourier` pour plusieurs valeurs croissantes de N.

Détermination des composantes d'un signal

Nous avons vu qu'ajouter ensemble des sinus de fréquence et d'amplitude différentes nous donne des signaux qui semblent totalement différents. Maintenant, la question est : supposons une forme d'onde complexe, serons-nous en mesure d'indiquer quelle combinaison de sinus (fréquence et amplitude) a donné lieu à ce signal particulier ? Utilisons d'abord une fonction, de PYTHON `np.trapz` qui exécute une intégration numérique par la méthode des trapèzes sur notre intervalle de temps $t \in [-\pi, \pi]$.

- Essayons d'intégrer la fonction sinus simple, $\sin(t)$: `np.trapz(np.sin(t),t)` Nous constatons que l'intégrale est égale à zéro. La courbe du sinus a une surface équivalente au-dessous et au-dessus du niveau zéro.
- Essayons de représenter le graphique de `np.sin(t)*np.sin(t)`. Nous constatons que la fonction a été complètement décalée au-dessus de zéro. Elle devrait maintenant avoir une surface non nulle.
  `In[]: np.trapz(np.sin(t)*np.sin(t),t)`. On trouve π.

- Essayons à présent de représenter graphiquement `np.sin(2*t)*np.sin(t)`. Le graphe indique que l'intégrale devrait être égale à zéro. Vérifions-le.
  Nous commençons à présent à avoir une « idée » de ce que nous emploierons pour séparer les composants de notre signal complexe. Multiplier un sinus par un sinus d'une fréquence différente donne zéro – nous obtenons des résultats non nuls uniquement lorsque les fréquences correspondent. Supposons que notre signal complexe soit :
  ```
  np.sin(t) - (1/2)*np.sin(2*t) + (1/3)*np.sin(3*t) - (1/4)*np.sin(4*t)
  ```
  Multiplier ceci par `np.sin(t)` permet d'obtenir :
  ```
  np.sin(t)*np.sin(t) - (1/2)*np.sin(2*t)*np.sin(t) +
  (1/3)*np.sin(3*t)*np.sin(t) - (1/4)*np.sin(4*t)*np.sin(t)
  ```
  Le premier terme donne π, tous les autres termes deviennent nuls. Le fait d'obtenir une valeur non nulle nous indique que $\sin(t)$ est présent dans le signal. Maintenant, multiplions le signal par $\sin(2\,t)$: si nous obtenons un résultat non nul, cela signifie que $\sin(2\,t)$ est présente dans le signal. Nous répéterons ce processus autant de fois que nous le souhaitons.
- Comment faire pour obtenir l'amplitude de chaque composante ? Faisons une autre expérience :
  ```
  In[]: yFourier=sum([(-1)**(n+1)*np.sin(n*t)/n for n in [1,2,3,4]])
  In[]: [np.trapz(yFourier*np.sin(n*t),t)/np.pi for n in [1,2,3,4]]
  Out[]: [1.0, -0.5, 0.33333333333333337, -0.25]
  ```

Nous constatons que nous obtenons l'amplitude de chaque composante du signal : $(1, -\frac{1}{2}, \frac{1}{3}, -\frac{1}{4})$.

3-2. Exercice : Refaire l'expérience précédente avec la fonction $g1(t)$.

3-3. Exercice : Écrire un programme PYTHON qui trace une fonction périodique $f(t)$ donnée, puis qui calcule ses coefficients de Fourier a_n et b_n(pour n allant jusqu'à N) et qui enfin compare $f(t)$ à sa reconstruction à partir des coefficients trouvés (voir figure ci-dessous). Vous pouvez vous aider du fichier `my_functions_fourier.py` que vous trouverez sur Moodle. Les différentes fonctions de ce fichier devront être importées dans votre programme. Une aide pour l'utilisation de ces fonctions est disponible.

Les fonctions à traiter sont les suivantes :

3-3-a) $f(t) = \sin^3(t)$

3-3-b) $f(t)$= fonction créneau centrée sur 0 et valant 1 entre $-T/8$ et $T/8$ et 0 ailleurs

3-3-c) $f(t)$=fonction créneau centrée sur 0 et valant 1 entre $-T/16$ et $T/16$ et 0 ailleurs

3-3-d) $f(t)$= fonction créneau décentrée de $T/16$, de largeur $T/8$

On prendra un nombre suffisant de termes de Fourier selon les cas, typiquement N entre 8 et 32. Pour tracer le spectre, utilisez `np.bar()`. On pourra légèrement décaler la position des deux types de coefficients : a_n à droite et b_n à gauche (voir l'aide de `np.bar()` et la figure ci-après).