Exercice 2

2.1 Librairie `numpy`

2.1.1 Création de matrices

- Créer les matrices suivantes en respectant le type et en 3 opérations maximum

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 \\ 1 & 6 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0. & 0. & 0. & 0. & 0. \\ 2. & 0. & 0. & 0. & 0. \\ 0. & 3. & 0. & 0. & 0. \\ 0. & 0. & 4. & 0. & 0. \\ 0. & 0. & 0. & 5. & 0. \\ 0. & 0. & 0. & 0. & 6. \end{bmatrix}$$

  Dans le second cas, on pourra s'aider de la méthode `diag`

- En utilisant la méthode `tile`, reproduire la matrice suivante à l'aide d'une seule commande

$$\begin{bmatrix} 4 & 3 & 4 & 3 & 4 & 3 \\ 2 & 1 & 2 & 1 & 2 & 1 \\ 4 & 3 & 4 & 3 & 4 & 3 \\ 2 & 1 & 2 & 1 & 2 & 1 \end{bmatrix}$$

correction

- ```
  In [1]: M = np.ones((4,4))
  In [2]: M[2,3] = 2
  In [3]: M[3,1] = 6

  In [1]: M = np.diag([2, 3, 4, 5, 6], k=-1)
  In [2]: M = M[:, :5]
  ```

- ```
  In [1]: np.tile([[4,3], [2, 1]], (2, 3))
  ```

2.1.2 Fonctions universelles

- Créer un tableau à une dimension contenant 1 million de valeurs aléatoires comprises entre 1 et 100
- Créer une fonction `invert` qui retournera un second tableau résultat de l'opération d'inversion du premier tableau
- À l'aide de la fonction intégrée `%timeit` de l'interpréteur `ipython`, estimer le temps moyen nécessaire à l'exécution de la fonction `inverse`
- Estimer ce même temps d'exécution, en utilisant l'opérateur division

correction

```
In [1]: def inverse(value):
   ...:     output = np.empty(len(values))
   ...:     for i in range(len(values)):
   ...:         output[i] = 1./values[i]
   ...:     return output
   ...:

In [2]: v = np.random.randint(1, 100, 1000000)

In [3]: %timeit inverse(v)
1 loop, best of 3: 2.46 s per loop

In [4]: %timeit 1./v
100 loops, best of 3: 6.45 ms per loop
```

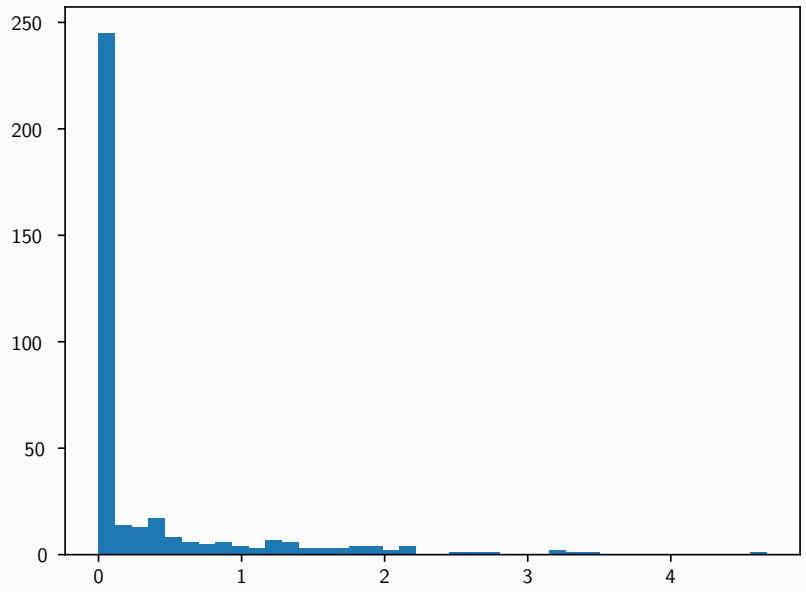
2.1.3 Sale temps sur Seattle

- Télécharger le fichier `seattle2014.csv` qui contient pour chaque jour de l'année 2014 (colonne 1), la hauteur des précipitations exprimé en dixième de millimètres (colonne 2) ainsi que les températures minimale (colonne 3) et maximale (colonne 4), exprimées en dixième de degrés Celsius, à Seattle.
- Charger l'ensemble des données dans un tableau `numpy` en prenant bien garde au caractère délimitant chaque champ puis, après avoir converti la hauteur des précipitations en centimètres et les températures en degré Celsius, calculer les valeurs suivantes sur chacune des données du fichier (hauteur des précipitations, T_{min} et T_{max}) :
  1. moyenne, médiane et écart type
  2. valeurs minimale et maximale
  3. les quantiles à 25% et 75%
- Afficher les valeurs ci-dessus pour la période estivale
- Calculer la hauteur totale d'eau tombée à Seattle en 2014
- Dénombrer le nombre total de jours dans l'année pendant lesquels il a plu à Seattle et déterminer combien de ces jours étaient pairs
- Représenter la distribution de la hauteur des précipitations à l'aide de la méthode `hist` de `matplotlib.pyplot`

correction

```python
import numpy as np

def print_report(prcp, Tmin, Tmax):

    print("Hauteur des précipitations:")
    print("  valeur moyenne = {} cm".format(np.mean(prcp)))
    print("  valeur médiane = {} cm".format(np.median(prcp)))
    print("      écart type = {} cm".format(np.std(prcp)))
    print("     valeur min. = {} cm".format(np.min(prcp)))
    print("     valeur max. = {} cm".format(np.max(prcp)))
    print("  quantile à 25% = {} cm".format(np.percentile(prcp, 25)))
    print("  quantile à 75% = {} cm".format(np.percentile(prcp, 75)))
    print("\n")

    print("Température minimale:")
    print("  valeur moyenne = {} °C".format(np.mean(Tmin)))
    print("  valeur médiane = {} °C".format(np.median(Tmin)))
    print("      écart type = {} °C".format(np.std(Tmin)))
    print("     valeur min. = {} °C".format(np.min(Tmin)))
    print("     valeur max. = {} °C".format(np.max(Tmin)))
    print("  quantile à 25% = {} °C".format(np.percentile(Tmin, 25)))
    print("  quantile à 75% = {} °C".format(np.percentile(Tmin, 75)))
    print("\n")

    print("Température maximale:")
    print("  valeur moyenne = {} °C".format(np.mean(Tmax)))
    print("  valeur médiane = {} °C".format(np.median(Tmax)))
    print("      écart type = {} °C".format(np.std(Tmax)))
    print("     valeur min. = {} °C".format(np.min(Tmax)))
    print("     valeur max. = {} °C".format(np.max(Tmax)))
    print("  quantile à 25% = {} °C".format(np.percentile(Tmax, 25)))
    print("  quantile à 75% = {} °C".format(np.percentile(Tmax, 75)))
    print("\n")

data = np.loadtxt("./data/seattle2014.csv", delimiter=",")

day = data[:,0]
prcp = data[:,1]/100 # cm
Tmax = data[:,2]/10  # °C
Tmin = data[:,3]/10  # °C

print("* Valeurs annuelles")
print_report(prcp, Tmin, Tmax)

print("* Valeurs estivales")
summer = (day > 20140401) & (day < 20140930)
print_report(prcp[summer], Tmin[summer], Tmax[summer])

print("Hauteur totale d'eau en 2014 : {} cm".format(np.sum(prcp)))
print("Nombre de jours avec pluie : {}".format(np.sum(prcp > 0)))
print("Nombre de jours pairs avec pluie : {}".format(np.sum((prcp > 0) & (day % 2 == 0))))

import matplotlib.pyplot as plt
plt.hist(prcp, 40)

plt.show()
```

correction



2.2 Librairie `matplotlib`

2.2.1 Sale temps sur Seattle (suite)

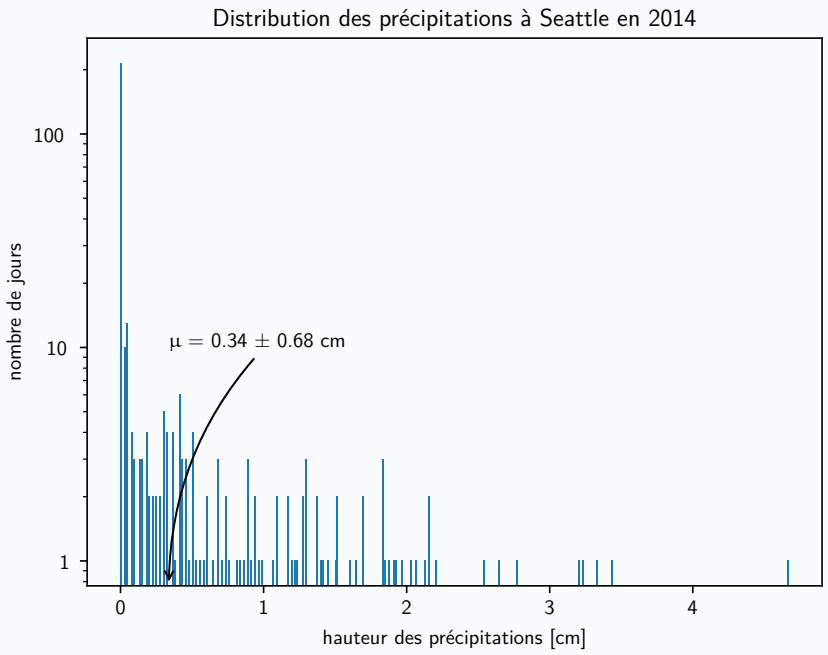
- À partir de l'histogramme représentant la distribution des hauteurs de précipitations à Seattle (*cf.* précédent TD), préciser les axes de la figure ainsi que le titre de l'histogramme
- Changer l'axe y pour passer en échelle logarithmique
- Annoter la figure pour faire apparaître la valeur moyenne et l'écart-type des précipitations
- Représenter sur une seconde figure, la variation temporelle des températures minimale et maximale. On s'aidera du code suivant pour convertir les jours extraits du fichier `csv` en une grandeur exploitable par `numpy`

```
import pandas as pd
day = pd.to_datetime(day, format="%Y%m%d")
```

- Sur une troisième figure, représenter la distribution des valeurs minimale et maximale des températures. Faire en sorte que les distributions puissent être visible y compris au niveau des zones de recouvrement

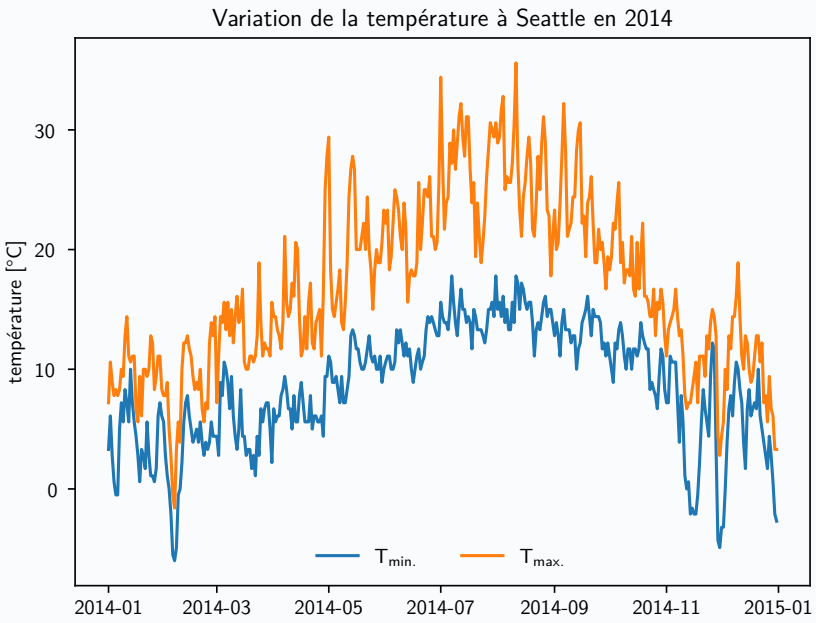
correction

```python
import numpy as np
data = np.loadtxt("./data/seattle2014.csv", delimiter=",")

day  = data[:,0]
prcp = data[:,1]/100 # cm
Tmax = data[:,2]/10  # °C
Tmin = data[:,3]/10  # °C

import matplotlib.pyplot as plt
plt.hist(prcp, 365)

plt.title("Distribution des précipitations à Seattle en 2014")
plt.xlabel("hauteur des précipitations [cm]")
plt.ylabel("nombre de jours")

plt.yscale("log")
# Remove exponential notation
from matplotlib.ticker import ScalarFormatter
plt.gca().yaxis.set_major_formatter(ScalarFormatter())

# Affiche valeur moyenne et écart type
mean = np.mean(prcp)
std  = np.std(prcp)

ymin, ymax = plt.gca().get_ylim()
plt.annotate(r"$\mu$ = {0:.2f} $\pm$ {1:.2f} cm".format(mean, std),
             xy=(mean, ymin), xytext=(mean, 10),
             arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
```

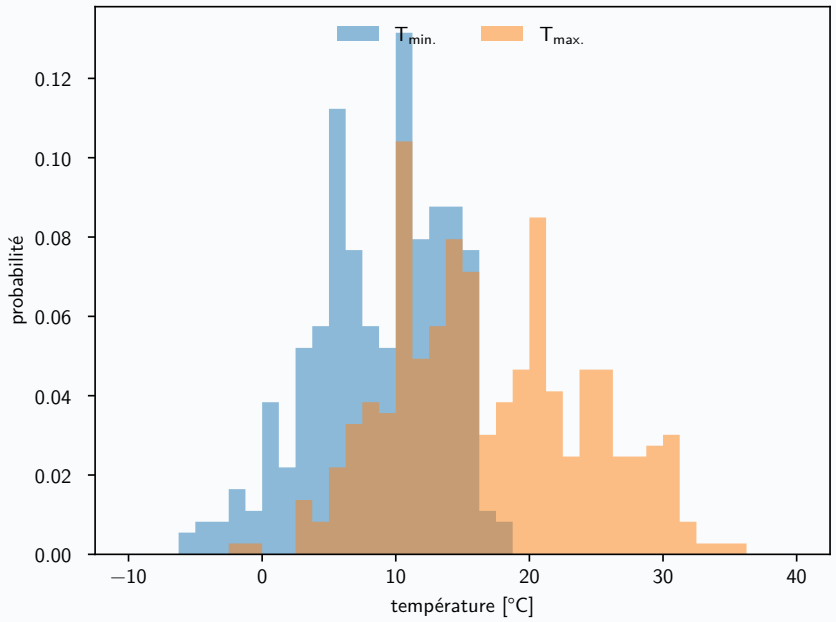



```python
# Variation des températures minimale et maximale au cours de l'année 2014
import pandas as pd
day = pd.to_datetime(day, format="%Y%m%d")

plt.figure()
plt.plot(day, Tmin, label=r"T$_\mathrm{min.}$")
plt.plot(day, Tmax, label=r"T$_\mathrm{max.}$")
plt.title("Variation de la température à Seattle en 2014")
plt.ylabel("température [°C]")
plt.legend(loc="lower center", ncol=2)
```

correction




```python
# Distributions normalisées des températures minimale et maximale
plt.figure()
kwargs = dict(histtype="stepfilled", alpha=0.5, bins=40, range=(-10, 40))
plt.hist(Tmin, label=r"T$_\mathrm{min.}$", **kwargs, weights=np.ones_like(Tmin)/len(Tmin))
plt.hist(Tmax, label=r"T$_\mathrm{max.}$", **kwargs, weights=np.ones_like(Tmax)/len(Tmax))
plt.xlabel("température [°C]")
plt.ylabel("probabilité")
plt.legend(loc="upper center", ncol=2)

# Show everything !
plt.show()
```



2.2.2 Fonctions discontinues[a]

- Représenter la fonction d'Heaviside $\Theta(x)$ définie

$$\begin{cases} \Theta(x) & = 1 \quad \text{si } x \geq 0 \\ & = 0 \quad \text{sinon} \end{cases}$$

- Le noyau radioactif de ^{11}C est un émetteur β^+ utilisé lors de tomographie par émission de positrons. La réaction permettant la production de cet élement est la suivante

$$p + {}^{14}_{7}\mathrm{N} \rightarrow {}^{11}_{6}\mathrm{C} + \alpha$$

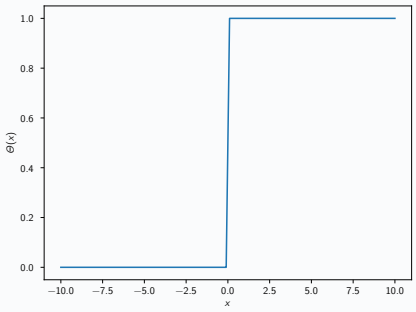
En tenant compte du taux de production de ^{11}C par irradiation et du nombre de noyaux se désintégrant, on peut montrer que le nombre de noyaux de ^{11}C au cours du temps s'exprime de la façon suivante

$$\begin{cases} n(t) & = \frac{n_i}{\lambda}\left(1 - e^{-\lambda t}\right) \quad \text{si } t \leq t_0 \\ & = n(t_0)\, e^{-\lambda(t-t_0)} \quad \text{si } t > t_0 \end{cases}$$

où $\lambda = \frac{\ln 2}{T_{1/2}}$ et $T_{1/2} = 20.36$ minutes. n_i correspond au taux d'irradition et est égal à $3\ 10^8$ noyaux/s. Représenter $n(t)$ pour $t_0 = 3$ heures.

correction

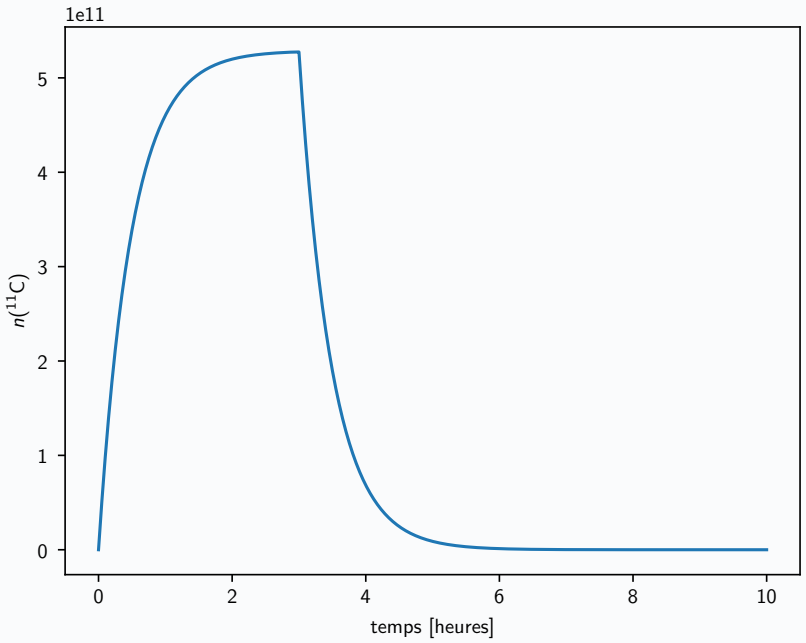
```python
def heaviside(x):
    return 0.5 * (np.sign(x) + 1)

import numpy as np
x = np.linspace(-10, 10, 100)

import matplotlib.pyplot as plt
plt.plot(x, heaviside(x))
plt.xlabel(r"$x$")
plt.ylabel(r"$\Theta(x)$")

plt.show()
```




```python
import numpy as np

# Définition des constantes du problème
ni  = 3e8*3600  # noyaux/h
T12 = 20.36/60  # hours
l   = np.log(2)/T12

def carbon11(t, t0):
    conds = [t <= t0, t > t0]
    funcs = [lambda t: ni/l*(1-np.exp(-l*t)),
             lambda t: ni/l*(1-np.exp(-l*t0))*np.exp(-l*(t-t0))]
    return np.piecewise(t, conds, funcs)

t0 = 3 # hours
t  = np.linspace(0, 10, 1000)
n  = carbon11(t, t0)

import matplotlib.pyplot as plt
plt.plot(t, n)
plt.xlabel("temps [heures]")
plt.ylabel(r"$n(^{11}\mathrm{C})$")

plt.show()
```

[a] on pourra s'aider ou pas de la fonction `piecewise` de `numpy`

correction



À titre de comparaison, la fonction suivante

```
def carbon11bis(t, t0):
    import math
    y = np.empty(len(t))
    for i, val in enumerate(t):
        if t[i] <= t0:
            y[i] = ni/l*(1-math.exp(-l*val))
        else:
            y[i] = ni/l*(1-math.exp(-l*t0))*math.exp(-l*(val-t0))
    return y
```

qui réalise le même calcul que `carbon11` défini plus haut mais sans user de la fonction `piecewise` de `numpy` est, peut-être plus naturelle à écrire mais n'en demeure pas moins beaucoup plus longue à exécuter

```
In [1]: t = np.linspace(0, 10, 1000000)

In [2]: %timeit carbon11(t, t0)
1 loop, best of 3: 332 ms per loop

In [3]: %timeit carbon11bis(t, t0)
1 loop, best of 3: 6.54 s per loop
```

2.2.3 *Iris setosa, Iris virginica et Iris versicolor*

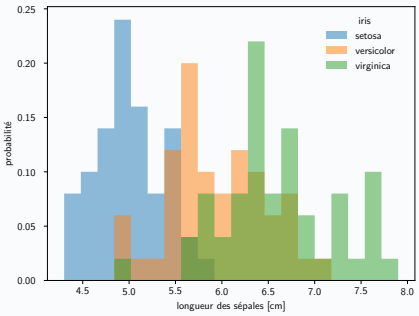
Le jeu de données *Iris* contient les propriétés morphologiques de 3 espèces de fleur d'iris collectées par Edgar Anderson. Ce jeu de données est surtout reputé par l'utilisation faite en 1936 par Ronald Fisher pour démontrer la puissance de son algorithme d'analyse discriminante linéaire à même de séparer les 3 espèces de fleur d'iris. Ces données sont devenues depuis un cas typique pour de nombreuses techniques de classification automatique en *machine learning*.

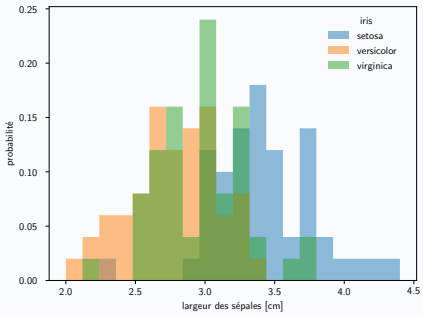
- Télécharger le fichier `iris.csv` qui contient la longueur et la largeur des sépales en cm (colonne 1 et 2), la longueur et la largeur des pétales en cm (colonne 3 et 4) ainsi qu'une dernière colonne dont la valeur, 0, 1 ou 2, est relative à l'espèce de la fleur d'iris (0 = *iris setosa*, 1 = *iris versicolor*, 2 = *iris virginica*). Charger ce fichier dans un tableau `numpy`
- Représenter les distributions normalisées de longueur et de largeur des sépales et des pétales pour les 3 espèces
- Représenter dans un diagramme (largeur des sépales *vs.* longueur des sépales), la largeur des pétales ainsi que l'espèce de fleur d'iris considérée

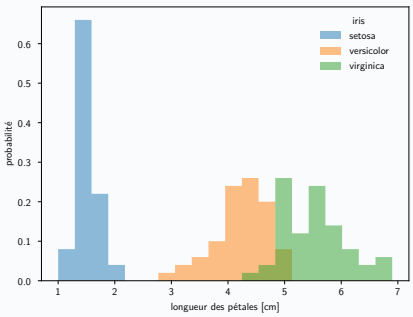
- Représenter l'ensemble des combinaisons possibles de données (largeur des sépales *vs.* longueur des sépales, largeur des sépales *vs.* largeur des pétales…), les figures situées dans la diagonale devant correspondre aux distributions normalisées des différentes grandeurs.

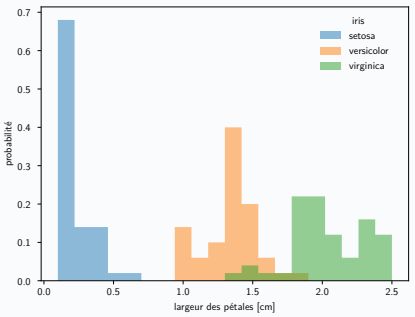
correction

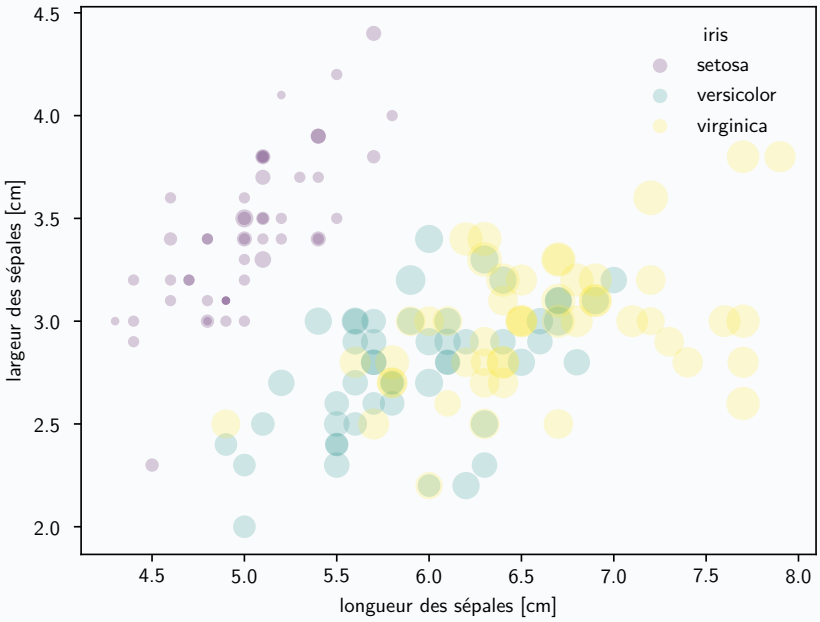
```python
import numpy as np

data = np.loadtxt("./data/iris.csv", delimiter=",")

sepal_length = data[:, 0]
sepal_width  = data[:, 1]
petal_length = data[:, 2]
petal_width  = data[:, 3]
species      = data[:, 4]

# Distributions des longueurs
import matplotlib.pyplot as plt

style = dict(histtype="stepfilled", alpha=0.5, bins=20)

iris = {0 : "iris setosa", 1 : "iris versicolor", 2 : "iris virginica"}
labels = {"longueur des sépales [cm]" : sepal_length,
          "largeur des sépales [cm]"  : sepal_width,
          "longueur des pétales [cm]" : petal_length,
          "largeur des pétales [cm]"  : petal_width}

for xlabel, data in labels.items():
    # Determine best range and bin probability
    r=(np.min(data), np.max(data))
    plt.figure()
    for key, name in iris.items():
        d = data[species == key]
        w = np.ones_like(d)/len(d)
        plt.hist(d, **style, label=name, range=r, weights=w)
    plt.xlabel(xlabel)
    plt.ylabel("probabilité")
    plt.legend()

# Diagrammes longueur vs. largeur sépales
plt.figure()
plt.scatter(sepal_length, sepal_width, s=100*petal_width,
            c=species, cmap="viridis", alpha=0.2)
plt.xlabel("longueur des sépales [cm]")
plt.ylabel("largeur des sépales [cm]")

# Création d'une légende à partir d'un scatter plot vide
color = plt.cm.get_cmap("viridis")
for key, name in iris.items():
    rgba = color(key/2)
    plt.scatter([], [],c=rgba, alpha=0.2, label=name)
plt.legend()

plt.show()
```


iris
setosa
versicolor
virginica
probabilité
longueur des sépales [cm]


iris
setosa
versicolor
virginica
probabilité
largeur des sépales [cm]

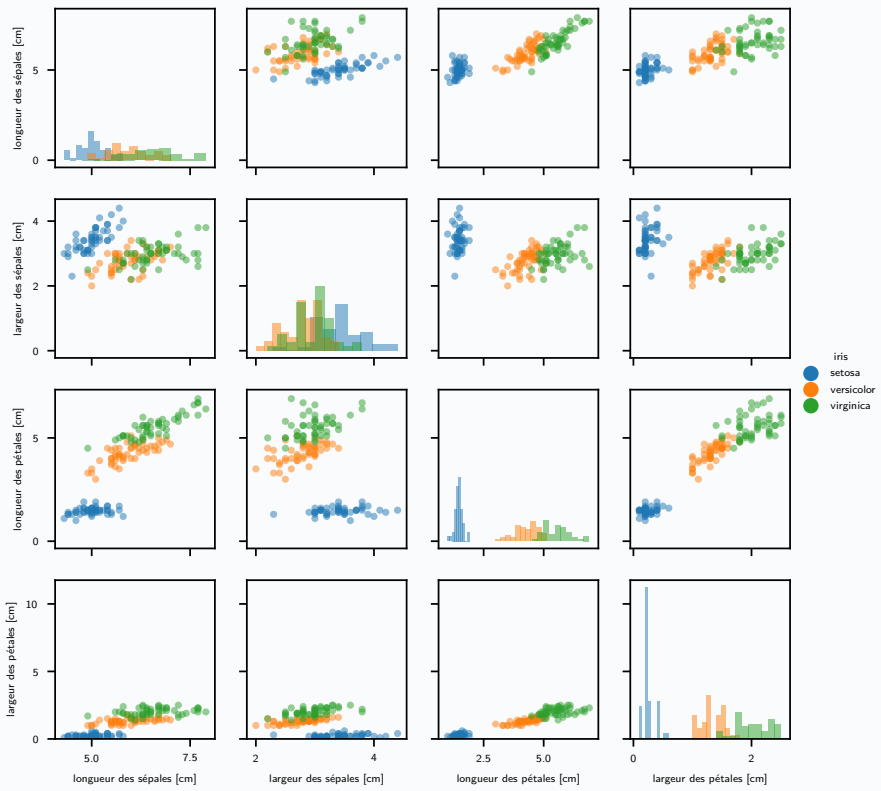

iris
setosa
versicolor
virginica
probabilité
longueur des pétales [cm]


iris
setosa
versicolor
virginica
probabilité
largeur des pétales [cm]


iris
setosa
versicolor
virginica
largeur des sépales [cm]
longueur des sépales [cm]

correction

```python
# Changement de taille de police uniquement pour cette figure
with plt.rc_context({"font.size": 5}):

    # Définition d'une grille de sous-figures
    fig, ax = plt.subplots(len(labels), len(labels),
                           sharex="col", sharey="row",
                           figsize=(1.5*len(labels), 1.5*len(labels)))

    for l1, d1 in labels.items():
        i1 = list(labels.keys()).index(l1)
        for l2, d2 in labels.items():
            i2 = list(labels.keys()).index(l2)
            for key, name in iris.items():
                sc = (species == key)
                if l1 == l2:
                    ax[i1, i2].hist(d1[sc], alpha=0.5, bins=10, normed=True)
                else:
                    ax[i1, i2].scatter(d2[sc], d1[sc], s=5, alpha=0.5)
        ax[-1, i1].set_xlabel(l1)
        ax[i1, 0].set_ylabel(l1)

    # Création d'une légende à partir d'un scatter plot vide
    for key, name in iris.items():
        plt.scatter([], [], label=name)
        plt.legend(title="iris", bbox_to_anchor=(1, len(iris)/2+1), loc="upper left")
    fig.subplots_adjust(right=0.9)


plt.show()
```



2.2.4 Interface graphique

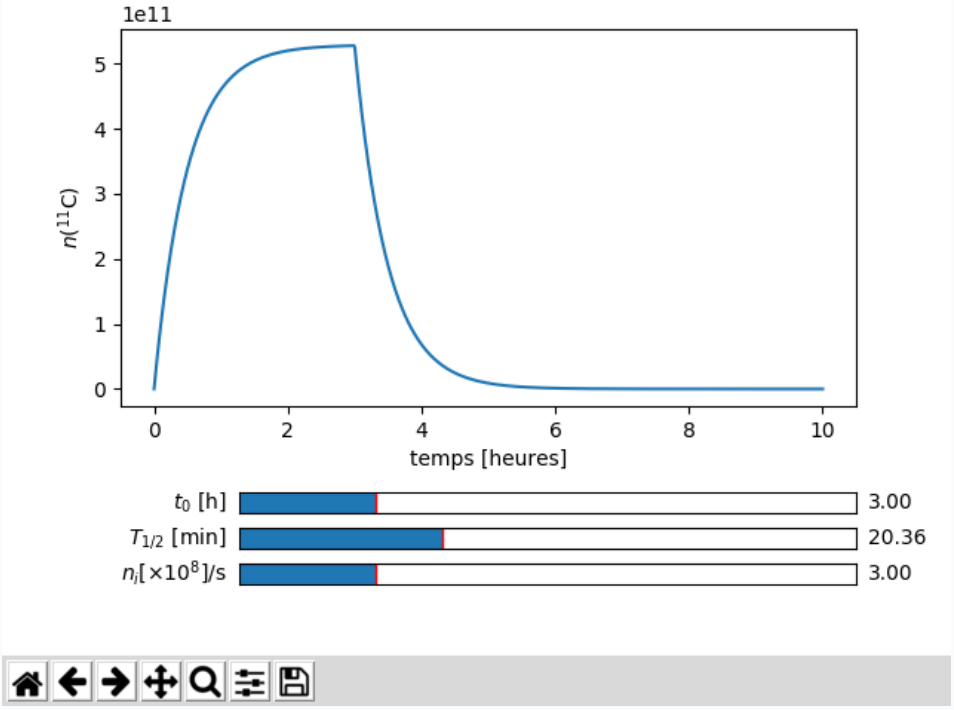
Reprendre l'exercice sur la variation du nombre de noyaux de carbone 11 en ajoutant à la représentation initiale, trois *sliders* respectivement n_i, $T_{1/2}$ et t_0 et faire en sorte que la figure se reconstruise à chaque nouvelle valeur de

ces paramètres.

correction

```python
import numpy as np

# Définition des constantes du problème
ni  = 3e8   # noyaux/s
T12 = 20.36 # min
t0  = 3     # hours

def n(t, ni=ni, t0=t0, T12=T12):
    T12 /= 60  # hours
    ni *= 3600 # noyaux/h
    l = np.log(2)/T12
    conds = [t <= t0, t > t0]
    funcs = [lambda t: ni/l*(1-np.exp(-l*t)),
             lambda t: ni/l*(1-np.exp(-l*t0))*np.exp(-l*(t-t0))]
    return np.piecewise(t, conds, funcs)

t = np.linspace(0, 10, 1000) #hours

import matplotlib.pyplot as plt
fig, ax = plt.subplots()
l, = plt.plot(t, n(t))
plt.xlabel("temps [heures]")
plt.ylabel(r"$n(^{11}\mathrm{C})$")

# Définition des sous-figures où afficher les sliders
axni  = plt.axes([0.25, 0.10, 0.65, 0.03])
axt12 = plt.axes([0.25, 0.15, 0.65, 0.03])
axt0  = plt.axes([0.25, 0.20, 0.65, 0.03])

plt.subplots_adjust(bottom=0.35)

from matplotlib.widgets import Slider
sni = Slider(axni, r"$n_i [\times10^8]\mathrm{/s}$", 1, 10, valinit=ni/1e8)
st12 = Slider(axt12, r"$T_{1/2}$ [min]", 1, 60, valinit=T12)
st0 = Slider(axt0, r"$t_{0}$ [h]", 1, 10, valinit=t0)

def update(val):
    nx = n(t, sni.val*1e8, st0.val, st12.val)
    l.set_ydata(nx)
    ax.set_ylim(ax.get_ylim()[0], 1.1*np.max(nx))
    fig.canvas.draw_idle()

sni.on_changed(update)
st12.on_changed(update)
st0.on_changed(update)

plt.show()
```

correction


1e11
5
4
3
2
1
0
$n(^{11}C)$
0
2
4
6
8
10
temps [heures]
t_0 [h]
3.00
$T_{1/2}$ [min]
20.36
n_i[×10^8]/s
3.00