

6.1 Sale temps sur Seattle (suite)

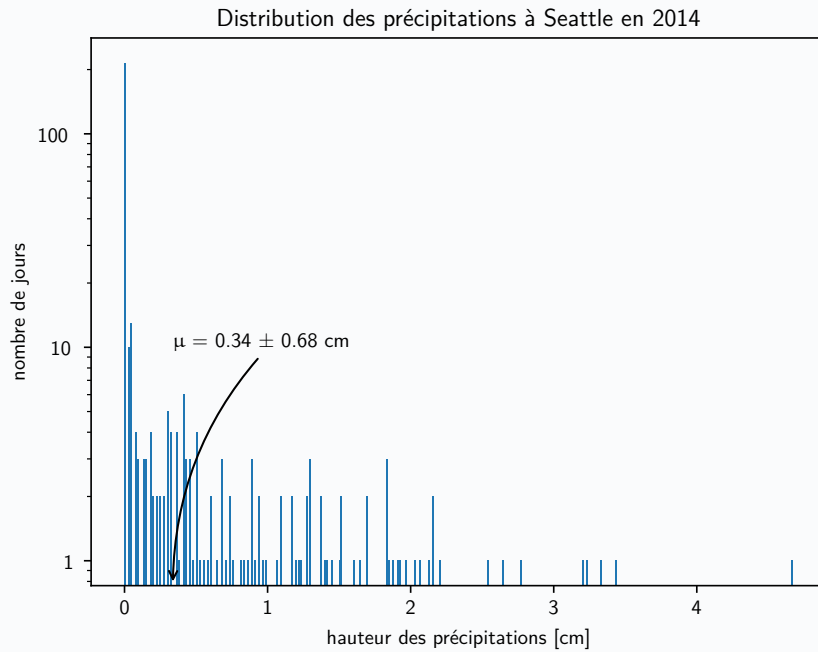
- À partir de l'histogramme représentant la distribution des hauteurs de précipitations à Seattle (cf. précédent TD), préciser les axes de la figure ainsi que le titre de l'histogramme
- Changer l'axe y pour passer en échelle logarithmique
- Annoter la figure pour faire apparaître la valeur moyenne et 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

```
1 import numpy as np
2 data = np.loadtxt("./data/seattle2014.csv", delimiter=",")
3
4 day = data[:,0]
5 prcp = data[:,1]/100 # cm
6 Tmax = data[:,2]/10 # °C
7 Tmin = data[:,3]/10 # °C
8
9 import matplotlib.pyplot as plt
10 plt.hist(prcp, 365)
11
12 plt.title("Distribution des précipitations à Seattle en 2014")
13 plt.xlabel("hauteur des précipitations [cm]")
14 plt.ylabel("nombre de jours")
15
16 plt.yscale("log")
17 # Remove exponential notation
18 from matplotlib.ticker import ScalarFormatter
19 plt.gca().yaxis.set_major_formatter(ScalarFormatter())
20
21 # Affiche valeur moyenne et écart type
22 mean = np.mean(prcp)
23 std = np.std(prcp)
24
25 ymin, ymax = plt.gca().get_ylim()
26 plt.annotate(r"$\mu$ = {0:.2f} $\pm$ {1:.2f} cm".format(mean, std),
27             xy=(mean, ymin), xytext=(mean, 10),
28             arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
```

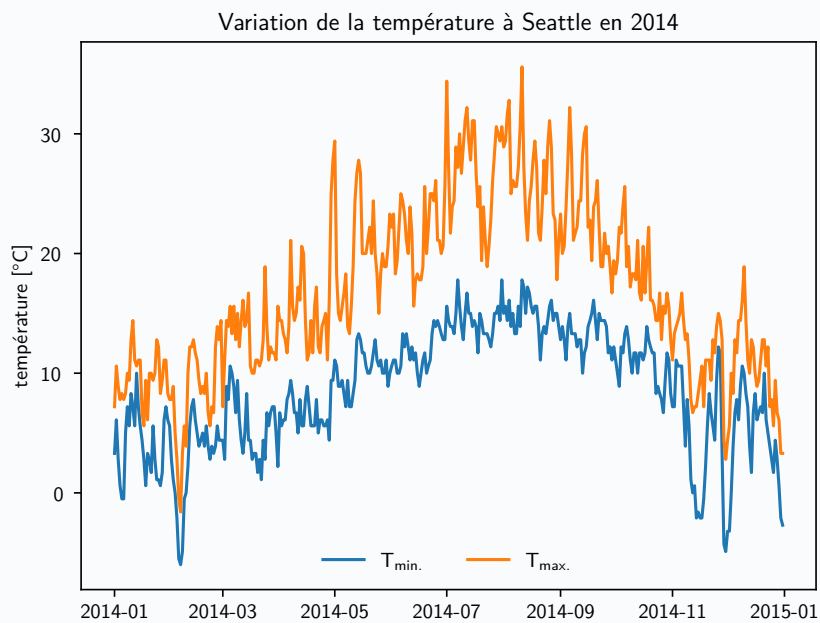
correction



```

1 # Variation des températures minimale et maximale au cours de l'année 2014
2 import pandas as pd
3 day = pd.to_datetime(day, format="%Y%m%d")
4
5 plt.figure()
6 plt.plot(day, Tmin, label=r"$T_{\mathrm{min.}}$")
7 plt.plot(day, Tmax, label=r"$T_{\mathrm{max.}}$")
8 plt.title("Variation de la température à Seattle en 2014")
9 plt.ylabel("température [°C]")
10 plt.legend(loc="lower center", ncol=2)

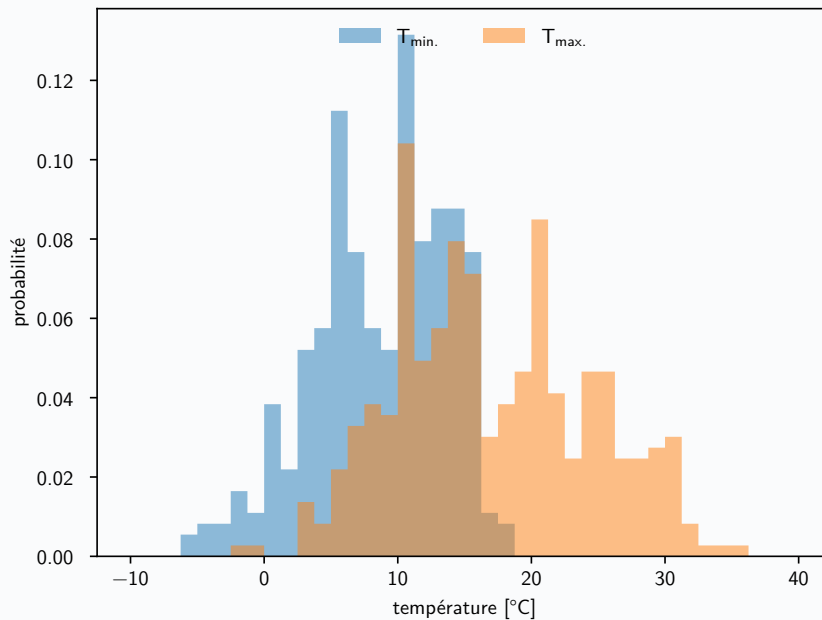
```



```

1 # Distributions normalisées des températures minimale et maximale
2 plt.figure()
3 kwargs = dict(histtype="stepfilled", alpha=0.5, bins=40, range=(-10, 40))
4 plt.hist(Tmin, label=r"$T_{\mathrm{min}}$", **kwargs, weights=np.ones_like(Tmin)/len(Tmin))
5 plt.hist(Tmax, label=r"$T_{\mathrm{max}}$", **kwargs, weights=np.ones_like(Tmax)/len(Tmax))
6 plt.xlabel("température [°C]")
7 plt.ylabel("probabilité")
8 plt.legend(loc="upper center", ncol=2)
9
10 # Show everything !
11 plt.show()

```



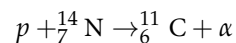
Télécharger la solution [↗](#)

6.2 Fonctions discontinues

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

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

- 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 élément est la suivante



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

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

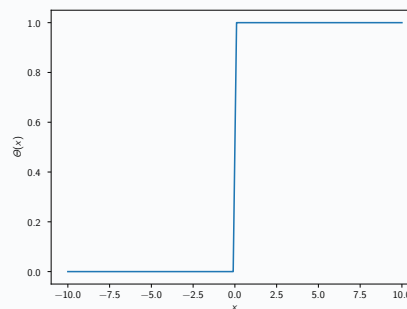
où $\lambda = \frac{\ln 2}{T_{1/2}}$ et $T_{1/2} = 20.36$ minutes. n_i correspond au taux d'irradiation et est égal à $3 \cdot 10^8$ noyaux/s. Représenter $n(t)$ pour $t_0 = 3$ heures^a.

^aon pourra s'aider ou pas de la fonction piecewise de numpy

```

1  def heaviside(x):
2      return 0.5 * (np.sign(x) + 1)
3
4  import numpy as np
5  x = np.linspace(-10, 10, 100)
6
7  import matplotlib.pyplot as plt
8  plt.plot(x, heaviside(x))
9  plt.xlabel(r"$x$")
10 plt.ylabel(r"$\Theta(x)$")
11
12 plt.show()

```



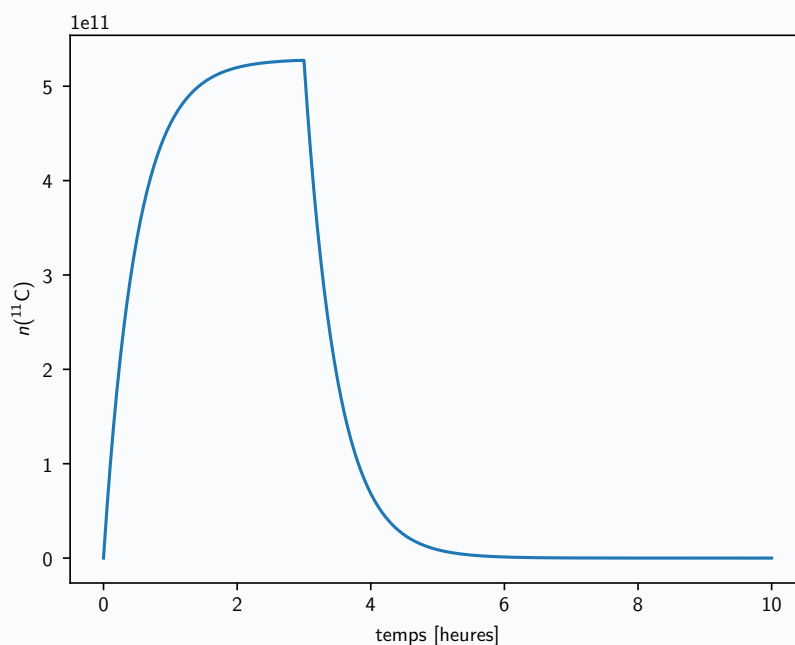
Télécharger la solution [↗](#)

```

1  import numpy as np
2
3  # Définition des constantes du problème
4  ni = 3e8*3600 # noyaux/h
5  T12 = 20.36/60 # hours
6  l = np.log(2)/T12
7
8  def carbon11(t, t0):
9      conds = [t <= t0, t > t0]
10     funcs = [lambda t: ni/l*(1-np.exp(-l*t)),
11              lambda t: ni/l*(1-np.exp(-l*t0))*np.exp(-l*(t-t0))]
12     return np.piecewise(t, conds, funcs)
13
14  t0 = 3 # hours
15  t = np.linspace(0, 10, 1000)
16  n = carbon11(t, t0)
17
18  import matplotlib.pyplot as plt
19  plt.plot(t, n)
20  plt.xlabel("temps [heures]")
21  plt.ylabel(r"$n(^{11}\mathrm{C})$")
22
23  plt.show()

```

Télécharger la solution [↗](#)



À titre de comparaison, la fonction suivante

```

1 def carbon11bis(t, t0):
2     import math
3     y = np.empty(len(t))
4     for i, val in enumerate(t):
5         if t[i] <= t0:
6             y[i] = ni/l*(1-math.exp(-l*val))
7         else:
8             y[i] = ni/l*(1-math.exp(-l*t0))*math.exp(-l*(val-t0))
9     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
```

6.3 *Iris setosa, Iris virginica et Iris versicolor*

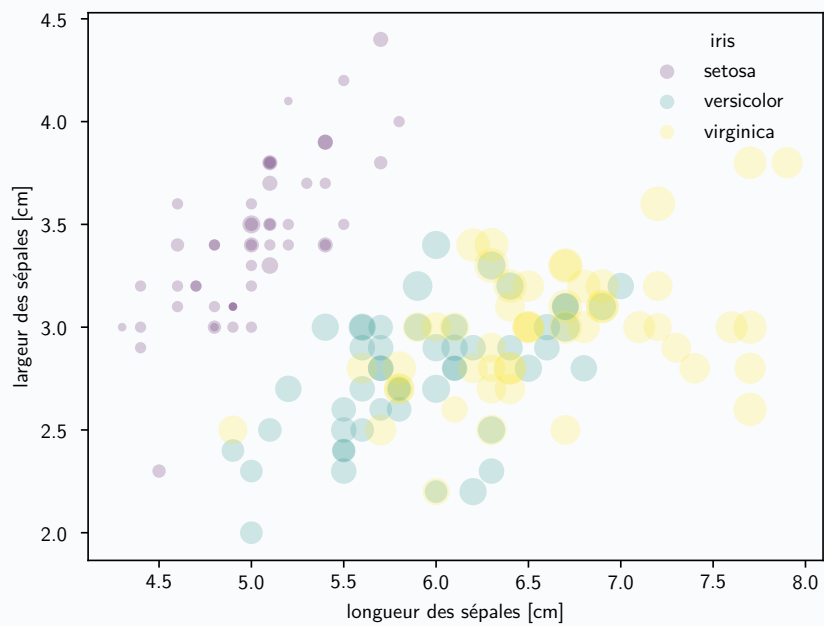
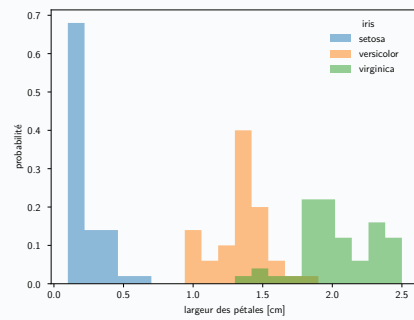
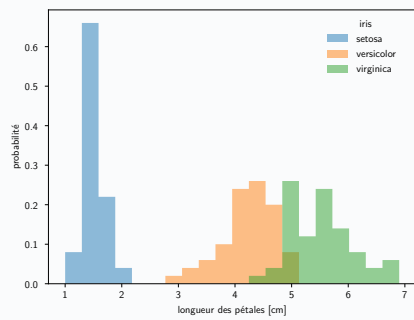
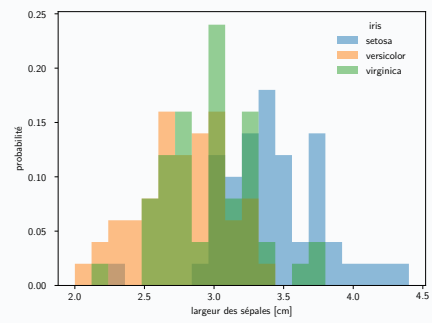
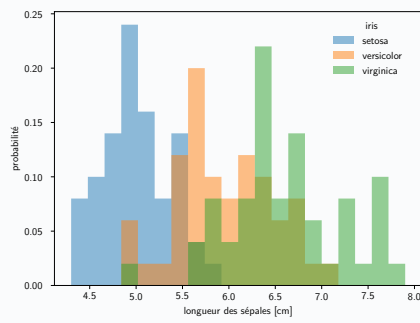
Le jeu de données *Iris* contient les propriétés morphologiques de 3 espèces de fleur d'iris collectées par Edgar Anderson. Ce jeu de données est surtout réputé par l'utilisation faite en 1936 par Ronald Fisher pour démontrer la puissance de son algorithme d'analyse discriminante linéaire à même de séparer 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` [ici](#) 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.

```

1  import numpy as np
2
3  data = np.loadtxt("../data/iris.csv", delimiter=",")
4
5  sepal_length = data[:, 0]
6  sepal_width  = data[:, 1]
7  petal_length = data[:, 2]
8  petal_width  = data[:, 3]
9  species      = data[:, 4]
10
11 # Distributions des longueurs
12 import matplotlib.pyplot as plt
13
14 style = dict(histtype="stepfilled", alpha=0.5, bins=20)
15
16 iris = {0 : "iris setosa", 1 : "iris versicolor", 2 : "iris virginica"}
17 labels = {"longueur des sépales [cm]" : sepal_length,
18          "largeur des sépales [cm]"  : sepal_width,
19          "longueur des pétales [cm]" : petal_length,
20          "largeur des pétales [cm]"  : petal_width}
21
22 for xlabel, data in labels.items():
23     # Determine best range and bin probability
24     r=(np.min(data), np.max(data))
25     plt.figure()
26     for key, name in iris.items():
27         d = data[species == key]
28         w = np.ones_like(d)/len(d)
29         plt.hist(d, **style, label=name, range=r, weights=w)
30         plt.xlabel(xlabel)
31         plt.ylabel("probabilité")
32         plt.legend()
33
34 # Diagrammes longueur vs. largeur sépales
35 plt.figure()
36 plt.scatter(sepal_length, sepal_width, s=100*petal_width,
37            c=species, cmap="viridis", alpha=0.2)
38 plt.xlabel("longueur des sépales [cm]")
39 plt.ylabel("largeur des sépales [cm]")
40
41 # Création d'une légende à partir d'un scatter plot vide
42 color = plt.cm.get_cmap("viridis")
43 for key, name in iris.items():
44     rgba = color(key/2)
45     plt.scatter([], [], c=rgba, alpha=0.2, label=name)
46     plt.legend()

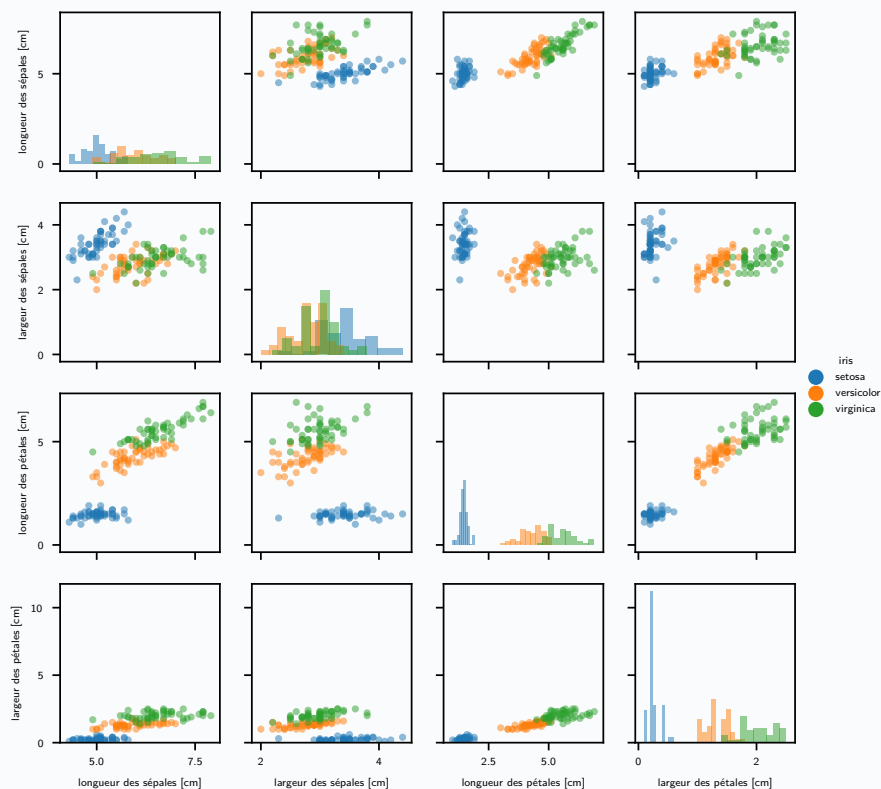
```



```

1  # Changement de taille de police uniquement pour cette figure
2  with plt.rc_context({"font.size": 5}):
3      # Définition d'une grille de sous-figures
4      fig, ax = plt.subplots(len(labels), len(labels),
5                             sharex="col", sharey="row",
6                             figsize=(1.5*len(labels), 1.5*len(labels)))
7
8      for l1, d1 in labels.items():
9          i1 = list(labels.keys()).index(l1)
10         for l2, d2 in labels.items():
11             i2 = list(labels.keys()).index(l2)
12             for key, name in iris.items():
13                 sc = (species == key)
14                 if l1 == l2:
15                     ax[i1, i2].hist(d1[sc], alpha=0.5, bins=10, normed=True)
16                 else:
17                     ax[i1, i2].scatter(d2[sc], d1[sc], s=5, alpha=0.5)
18                     ax[-1, i1].set_xlabel(l1)
19                     ax[i1, 0].set_ylabel(l1)
20
21     # Création d'une légende à partir d'un scatter plot vide
22     for key, name in iris.items():
23         plt.scatter([], [], label=name)
24         plt.legend(title="iris", bbox_to_anchor=(1, len(iris)/2+1), loc="upper left")
25     fig.subplots_adjust(right=0.9)
26     plt.show()

```



Télécharger la solution [↗](#)