



## Représentations graphiques (Python et Scilab)

◇ Objectifs : représenter graphiquement des suites, des fonctions (et des mesures expérimentales). Donner des noms aux courbes, aux axes, à la figure.

◇ Les différentes étapes pour effectuer le tracé avec PYTHON :

```
import numpy as np
import matplotlib.pyplot as plt
plt.plot(x, y, ...)
```

```
plt.legend(...)
plt.xlabel(...)
plt.ylabel(...)
plt.title(...)
plt.show()
```

(Si besoin)

Importer le module de tracé

Représenter graphiquement les points donc les abscisses sont dans  $x$  et les ordonnées dans  $y$  ( $x$  et  $y$  sont des listes, ou des tableaux, de même taille). Répéter cette étape pour chaque courbe à tracer

Donner les noms des différentes courbes

Donner les noms des axes

Donner le nom du dessin

Afficher la fenêtre graphique

◇ Avec SCILAB, pas de module particulier à utiliser et les commandes sont très semblables :

```
scf()
plot(x, y, ...)

legend(...)
xlabel(...)
ylabel(...)
title(...)
```

Créer une nouvelle fenêtre graphique

Représenter graphiquement les points donc les abscisses sont dans  $x$  et les ordonnées dans  $y$  ( $x$  et  $y$  sont des matrices ligne ou colonne de même taille). Répéter cette étape pour chaque courbe à tracer

Donner les noms des différentes courbes

Donner les noms des axes

Donner le nom du dessin

*Remarque.* On peut ajouter à la commande **plot** des indications supplémentaires pour préciser le style du tracé sous forme d'une chaîne de caractères : "**ro**" pour un tracé avec des ronds rouges, "**b--**" pour un tracé avec une ligne en pointillés bleus, "**g-**" pour un tracé avec une ligne pleine verte, etc. On se reportera aux exemples vus en classe. □

# Exemples de représentations graphiques

## Suites

*Exemple.* On considère la suite  $(u_n)_{n \geq 0}$  définie par :

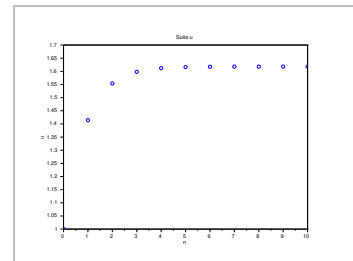
$$u_0 = 1$$
$$\forall n \in \mathbb{N}, u_{n+1} = \sqrt{u_n + 1}$$

Représenter graphiquement les valeurs prises par la suite  $(u_n)$  pour  $0 \leq n \leq 10$ .

```
import matplotlib.pyplot as plt
def u(n):
    u=1
    for k in range(n):
        u=(u+1)**0.5
    return u
plt.plot([u(k) for k in range(11)], 'bo')
plt.title("Suite u")
plt.xlabel("n")
plt.ylabel("u")
```

```
plt.show()
```

```
function y=u(n)
    y=1
    for k=1:n
        y=(y+1)^0.5
    end
endfunction
scf()
plot(0:10, feval(0:10,u), 'bo')
title("Suite u")
xlabel("n")
ylabel("u")
```



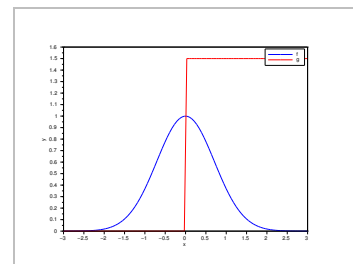
## Fonctions

**Exemple.** Représenter les fonctions  $f: x \mapsto e^{-x^2}$  et  $g: x \mapsto \begin{cases} 0 & \text{si } x < 0 \\ 1.5 & \text{si } x \geq 0 \end{cases}$  sur  $[-3, 3]$ .

```
import numpy as np
import matplotlib.pyplot as plt
def g(x):
    if x<0:
        return 0
    else:
        return 1.5
x=np.linspace(-3,3,100)
plt.plot(x,np.exp(-x**2), 'b-')
plt.plot(x,[g(xi) for xi in x], 'r-')
plt.legend(["f", "g"])
plt.xlabel("x")
plt.ylabel("y")
```

```
plt.show()
```

```
function y=g(x)
    if x<0 then
        y=0
    else
        y=1.5
    end
endfunction
x=linspace(-3,3,100)
scf()
plot(x,exp(-x.^2), 'b-')
plot(x,feval(x,g), 'r-')
legend("f", "g")
xlabel("x")
ylabel("y")
```



## Un exemple en chimie

**Exemple.** On mesure la concentration  $C(t)$  d'une espèce chimique à différents instants. On obtient le tableau ci-contre. Par ailleurs, une étude théorique montre que l'évolution de cette concentration peut être modélisée par la fonction :

$$C: t \mapsto \frac{1}{kt + \frac{1}{C_0}}$$

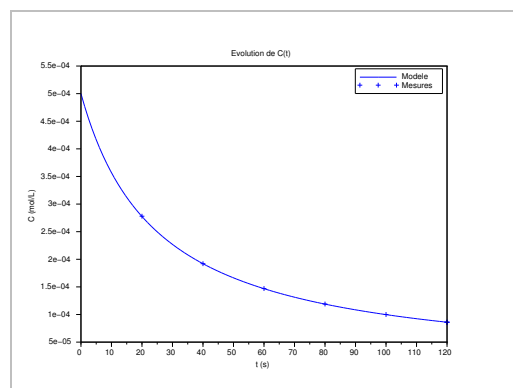
avec  $k = 80.2 \text{ mol}^{-1} \cdot \text{L} \cdot \text{s}^{-1}$  et  $C_0 = 5.0 \cdot 10^{-4} \text{ mol} \cdot \text{L}^{-1}$ . Représenter sur le même graphique la courbe théorique et les valeurs expérimentales.

```
import matplotlib.pyplot as plt
import numpy as np
t_mes=[20, 40, 60, 80, 100, 120]
C_mes=[2.78e-4, 1.92e-4, 1.47e-4, 1.19e-4, 1.0e-4, 0.86e-4]
k=80.2
C0=5.0e-4
t=np.linspace(0, 120, 100)
plt.plot(t, (k*t+1/C0)**(-1))
plt.plot(t_mes, C_mes, '+', markersize=12)
plt.legend(["Modele", "Mesures"])
plt.title("Evolution de C(t)")
plt.xlabel("t (s)")
plt.ylabel("C (mol/L)")
```

```
plt.show()
```

```
t_mes=[20, 40, 60, 80, 100, 120]
C_mes=[2.78e-4, 1.92e-4, 1.47e-4, 1.19e-4, 1.0e-4, 0.86e-4]
```

```
k=80.2
C0=5.0e-4
t=linspace(0, 120, 100)
scf()
plot(t, (k*t+1/C0).^(-1))
plot(t_mes, C_mes, '+')
legend("Modele", "Mesures")
title("Evolution de C(t)")
xlabel("t (s)")
ylabel("C (mol/L)")
```



Un exemple en physique

**Exemple.** On considère l'équation différentielle du second ordre sous forme canonique :

$$\frac{1}{\omega_0^2} \frac{d^2 u_c}{dt^2} + \frac{2\xi}{\omega_0} \frac{du_c}{dt} + u_c = 0$$

avec  $\omega_0 = 10^4 \text{ rad} \cdot \text{s}^{-1}$  et  $\xi = 0.5$  ainsi que la solution :

$$u_c(t) = A \cos(\omega_a t) \exp(-\xi \omega_0 t)$$

avec  $A = 1 \text{ V}$  et  $\omega_a = \omega_0 \sqrt{1 - \xi^2}$  (régime pseudopériodique). Représenter la fonction  $u_c$  en traits pleins ainsi que les fonctions  $t \mapsto \pm A \exp(-\xi \omega_0 t)$  en pointillés sur  $[0, 4T]$  avec  $T = 2\pi/\omega_0$ .

```
import numpy as np
import matplotlib.pyplot as plt
w_0=1e4
xi=0.5
A=1
w_a=w_0*np.sqrt(1-xi**2)
T=2*np.pi/w_0
t=np.linspace(0,4*T,100)
plt.plot(t,A*np.cos(w_a*t)*np.exp(-xi*w_0*t), 'b-')
plt.plot(t,A*np.exp(-xi*w_0*t), 'r--')
plt.plot(t,-A*np.exp(-xi*w_0*t), 'r--')
plt.xlabel("t (s)")
plt.ylabel("u_c (V)")
plt.title("Regime pseudoperiodique")
```

```
plt.show()
```

```
w_0=1e4
xi=0.5
A=1
w_a=w_0*sqrt(1-xi^2)
T=2*pi/w_0
t=linspace(0,4*T,100)
plot(t,A*cos(w_a*t).*exp(-xi*w_0*t), 'b-')
plot(t,A*exp(-xi*w_0*t), 'r--')
plot(t,-A*exp(-xi*w_0*t), 'r--')
xlabel("t (s)")
ylabel("u_c (V)")
title("Regime pseudoperiodique")
```

