1. Méthode d'Euler, c'est ce qui a été fait dans la série précédente

Formalisons la méthode utilisée dans la série précédente pour résoudre le 3e problème.
On a vu que l'évolution est régie par l'équation différentielle :

$a(t) = -\frac{k}{m} \cdot x(t) - \frac{c}{m} \cdot V(t)$ et

$V'(t) = a(t)$ et

$x'(t) = V(t)$ avec $V(0) = 0$ [m/s] et $x(0) = 0.10$ [m]

La **méthode d'Euler** consiste à remplacer :

V'(t) par $\frac{V(t+dt) - V(t)}{dt}$ et x'(t) par $\frac{x(t+dt) - x(t)}{dt}$.

L'équation différentielle est remplacé par la méthode numérique :

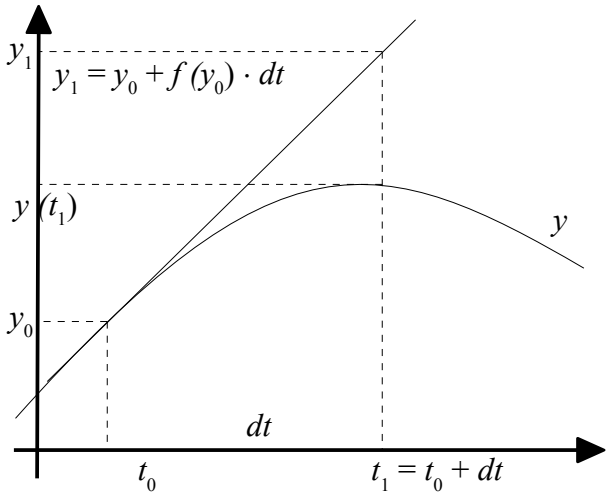
$a_t = (-k \cdot x(j \cdot dt) - c \cdot V(j \cdot dt)) / m$ et

$V((j+1) \cdot dt) = V(j \cdot dt) + dt \cdot a_t$ et

$x((j+1) \cdot dt) = x(j \cdot dt) + dt \cdot V(j \cdot dt)$, avec $j = 0, 1, 2, 3$, etc. ($t = j \cdot dt$)

La différence entre l'approximation numérique et la solution exacte est proportionnelle à la valeur numérique choisie pour dt.
On dit que la méthode d'Euler est d'ordre 1.

Illustration graphique de la méthode d'Euler pour résoudre : $\boldsymbol{y'(t) = f(y(t)),\ \ y(t_0) = y_0}$



$y(t_1)$ est la solution exacte,
y_1 est l'approximation numérique obtenue par la méthode d'Euler.

On remarque que l'approximation est rapidement mauvaise, si dt n'est pas assez petit.

Il existe de nombreuses améliorations, dont la méthode très connues de Runge-Kutta.
Nous ne les aborderons pas ici.
Seule la méthode **ode** intégrée dans le système de Scilab sera présentée.

Jusqu'à ici, il n'y a rien de pratique à faire, il faut comprendre la méthode d'Euler.

2. La méthode générale de Scilab

Vu que la résolution numérique de système d'équations différentielles est fréquente, Scilab a implémenté une fonction générale de résolution de système d'équations différentielles. De plus la méthode de Scilab est meilleure que celle de Runge-Kutta, qui était déjà très bonne.
La fonction de Scilab s'appelle **ode(...)** qui est l'abréviation de "Ordinary Differential Equation"

Voici un exemple d'utilisation de la fonction **ode(...)**

```
function ex2_1()
//=============
// Exercice 2.1.  Évolution d'une vitesse d'un corps qui chute avec frottement.
// Équation différentielle d'évolution de la vitesse.:
// V'(t) = 9.81 - (0.5/3.0) * V(t) , V(0) = 0
   function y_prime=f(t, y)
   //======================
   // t représente le temps  (non utilisé ici)
   // y  représente la vitesse
   // y_prime  représente la dérivée de la vitesse par rapport au temps
   g = 9.81; // [m/s^2]
   m = 3.0;  // masse du corps en [kg]
   kfrot = 1.5;   // coefficient de frottement [kg / s]
   y_prime = g - (kfrot/m)*y;  // car  vitesse ' = g - (kfrot/m) * vitesse
   endfunction

// données :
t0    =  0;   // temps initial
tFin  = 10;   // temps final
y0    = 0;  // vitesse initiale = 0 [m/s]
N  = 101;      // nombre de points d'évaluation
at = linspace(t0, tFin, N);  // vecteur LIGNE

// résolution numérique en appelant la fonction  ode  de SciLab
ay = ode( y0,..     // condition initiale
          t0,..     // t_initial,
          at,..     // série des t(j) pour lesquels  y(j)  sera calculé
          1e-3,..   // tolérance d'erreur relative
          1e-4,..   // tolérance d'erreur absolue
          f..       // fonction définissant l'équation différentielle
          );
// L'algorithme essaye d'avoir une erreur inférieure au
// maximum entre la tolérance d'erreur absolue et la tolérance d'erreur relative.

// graphique de la solution numérique
scf(1);  clf();
plot(at, ay, 'b');  // 'b' = blue, pour les positions

gcf().figure_name="Évolution de la vitesse en fonction du temps" // Titre de la fenêtre
title("Évolution de la vitesse en fonction du temps", 'fontsize', 3);
xlabel("Temps [s]", 'fontsize', 3);
ylabel("Vitesse [m/s]", 'fontsize', 3);
endfunction
```

2.1 Écrivez le programme ci-dessus.

Testez-le.

3. La méthode générale de Scilab, en plusieurs dimensions

Sans regarder le code de la page suivante, saurez-vous modifier le code ci-dessus pour obtenir également l'évolution de la position ?
Les modifications sont mineures, pas évidentes et vectorielles !

Voici le code demandé, remarquez sa similitude avec le code précédent.

```
function ex3()
//=============
// Exercice 3.  Évolution d'une vitesse d'un corps qui chute avec frottement.
// Avec l'évolution de la position en fonction du temps.
// Équation différentielle d'évolution de la vitesse.:
// X'(t) = V(t) , X(0) = 0 [m]
// V'(t) = 9.81 - (0.5/3.0) * V(t) , V(0) = 0 [m/s]

   function y_prime=f(t, y)
   //======================
  // t représente le temps  (non utilisé ici)
  // y(1)  représente la position
  // y(2)  représente la vitesse
  // y_prime(1)  représente la dérivée de la position par rapport au temps
  // y_prime(2)  représente la dérivée de la vitesse par rapport au temps
   // y et y_prime sont des vecteurs colonne
   g = 9.81; // [m/s^2]
   m = 3.0;  // masse du corps en [kg]
   kfrot = 1.5;   // coefficient de frottement [kg / s]
   V = y(2)
   y_prime = [ V; ..   // car  position' = vitesse
               (g - (kfrot/m)*V)];  // car  vitesse ' = g - (kfrot/m) * vitesse
   // Autre écriture possible
   //y_prime(1) = y(2);     // car  position' = vitesse
   //y_prime(2) = g - (kfrot/m)*y(2);  // car  vitesse ' = g - (kfrot/m) * vitesse
   endfunction

// données :
t0    =  0;   // temps initial
tFin  = 10;   // temps final
V0 = 0;       // vitesse initiale = 0 [m/s]
X0 = 0;       // position = 0 [m]
y0    = [X0; V0];  // vecteur COLONNE
N  = 101;      // nombre de points d'évaluation
at = linspace(t0, tFin, N);  // vecteur LIGNE

// résolution numérique en appelant la fonction  ode  de SciLab
ay = ode( y0,..     // condition initiale
          t0,..     // t_initial,
          at,..     // série des t(j) pour lesquels  y(j)  sera calculé
          1e-3,..   // tolérance d'erreur relative
          1e-4,..   // tolérance d'erreur absolue
          f..       // fonction définissant l'équation différentielle
          );
// L'algorithme essaye d'avoir une erreur inférieure au
// maximum entre la tolérance d'erreur absolue et la tolérance d'erreur relative.

// graphique de la solution numérique
scf(1);  clf();
plot(at, ay(2,:), 'b');  // 'b' = blue, pour les vitesses

gcf().figure_name="Évolution de la vitesse en fonction du temps" // Titre de la fenêtre
title("Évolution de la vitesse en fonction du temps", 'fontsize', 3);
xlabel("Temps [s]", 'fontsize', 3);
ylabel("Vitesse [m/s]", 'fontsize', 3);

scf(2);  clf();
plot(at, ay(1,:), 'g');  // 'g' = red, pour les positions

gcf().figure_name="Évolution de la position en fonction du temps" // Titre de la fenêtre
title("Évolution de la position en fonction du temps", 'fontsize', 3);
xlabel("Temps [s]", 'fontsize', 3);
ylabel("Position [m]", 'fontsize', 3);
endfunction
```

4. La méthode générale de Scilab, en plusieurs dimensions

Reprenez les exercices 2 et 3 de la série précédente et résolvez-les avec la méthode **ode** de Scilab.