0. Théorie.

On veut résoudre numériquement une équation différentielle : $\boldsymbol{y'(x) = f(x, y(x)),\quad y(x_0) = y_0}$

Le schéma pour résoudre numériquement une équation différentielle selon la **méthode d'Euler** peut s'écrire de la manière standard suivante :

$$\boxed{\begin{aligned} k_1 &= f(x_0\ ;\ y_0) \\ y_1 &= y_0 + k_1 \cdot h \end{aligned}}$$

y_0, f, k_1 et y_1 peuvent être des vecteurs, x_0 et h sont des scalaires.

Cette méthode de résolution numérique est simple, mais peu efficace, car elle est d'ordre 1, c'est-à-dire que l'erreur est proportionnelle au pas h.
Par exemple, un pas de 0,01 donne une erreur proportionnelle à 0,01.

La **méthode de Runge** décrite ci-dessous est légèrement plus compliquée, mais elle est d'ordre 2, c'est-à-dire que l'erreur est proportionnelle au carré du pas h, donc proportionnelle à h^2.
Par exemple, un pas de 0,01 donne une erreur proportionnelle à 0,0001.
Dans les faits, le coefficient de proportionnalité est un peu plus petit que celui de la méthode d'Euler, *donc pour un pas de 0,01, le résultat est plus de 100 fois plus précis qu'avec Euler* !!!
Le schéma de la **méthode de Runge** est le suivant :

$$\boxed{\begin{aligned} k_1 &= f(x_0\ ;\ y_0) \\ k_2 &= f\left(x_0 + \frac{h}{2}\ ;\ y_0 + k_1 \cdot \frac{h}{2}\right) \\ y_1 &= y_0 + k_2 \cdot h \end{aligned}}$$

Écrivons un algorithme général de résolution utilisant la méthode de Runge.
On veut résoudre numériquement une équation différentielle : $\boldsymbol{y'(x) = f(x, y(x)),\quad y(x_0) = y_0}$
On veut obtenir nMax+1 valeurs de y répartis régulièrement entre x0 et xFin.
Écrivons une fonction qui résolve numériquement notre équation différentielle :

```
function y=Runge_o2_d1(x0, xFin, y0, nMax, f)
//=============================================
// o2, signifie que l'ordre numérique est de 2, soit moyennement précis.
// d1, signifie que la dimension de  y0  et de  y  est de  1, soit assez limité.
// Résolution numérique de l'équation différentielle y'(x) = f(x, y(x))
// y(x0) = y0  est la condition initiale
// retourne un vecteur de  nMax+1  valeurs de  y.
// f  est la fonction décrivant l'équ. diff.
y(1) = y0;
dx = (xFin - x0) / nMax;
for j=1:nMax do
   x = x0 + (j-1) * dx;
   k1 = f(x, y(j));
   k2 = f(x + dx/2, y(j) + k1 * dx/2);
   y(j+1) = y(j) + dx*k2;
   end;
endfunction
```

On remarque que cette fonction est très similaire à celle utilisant la méthode d'Euler.

Pour le problème de l'exercice 1, il suffit de faire ce qui suit :

```
function z=f(x, y)
//================
// z  représente  y'
z = 9.81 - (1.5/3)*y;  // car  y' = f(x, y) = 9.81 - (1.5/3) * y
endfunction

nMax = 20;
// Résolution numérique
ay = Runge_o2_d1( 0..    // temps initial
                 ,10..   // temps final
                 ,0..    // vitesse initiale
                 ,nMax..  // nombre de points = nMax
                 ,f);    // f  est la fonction définissant l'équ. diff.

// graphique de la solution numérique
ax = linspace(0, 10, nMax+1)';
scf(1);
clf();
plot(ax, ay, 'k');
```

1. Évolution de la vitesse d'un corps qui chute avec frottement.

Le but est de calculer numériquement l'évolution de la vitesse d'un corps qui chute avec frottement.
Un corps de masse $m = 3.0$ [kg] chute verticalement, en subissant une force résultante égale à :
$F_{résultante} = m \cdot g - 1.5 \cdot V(t)$
où V(t) est la vitesse du corps au temps t. $g = 9.81$ [m/s²].
Au temps t = 0 [s], le corps est immobile.

1.1 Juste avant la fonction qui calculera l'évolution de la vitesse en fonction du temps, écrivez en commentaire l'équation différentielle qui régit l'évolution de cette vitesse.
1.2 Écrivez une "function ex1()" qui calcule par la **méthode d'Euler, comme décrit ci-dessus,** l'évolution de la vitesse du corps durant 10 secondes, en prenant nMax = 100.
1.3 À la fin de votre fonction, tracez la courbe d'évolution de la vitesse en fonction du temps.

---

2. Évolution du niveau d'eau d'un réservoir.

Le but est de calculer numériquement l'évolution du niveau d'eau d'un réservoir qui se remplit.
Le réservoir est cylindrique de section de base valant 1 *dm*².
Le débit d'eau qui entre dans le réservoir vaut :
$D(t) = 2 - h(t)/6$, où $h(t)$ est la hauteur d'eau dans le réservoir au temps t.
La hauteur est exprimée en *dm* et le débit en *litres par seconde*.
Rappelons que le débit est la variation du volume par unité de temps.
Initialement, le réservoir est vide.

2.1 Juste avant la fonction qui calculera l'évolution de la hauteur en fonction du temps, écrivez en commentaire l'équation différentielle qui régit l'évolution de hauteur h(t).
2.2 Écrivez une "function ex2()" qui calcule par la **méthode d'Euler, comme décrit ci-dessus,** l'évolution de la vitesse du corps durant 20 secondes, en prenant nMax = 200.
2.3 À la fin de votre fonction, tracez la courbe d'évolution de la vitesse en fonction du temps.

---

3. Théorie à compléter par vous.

Vous avez vu dans la série précédente comment écrire une fonction générale de résolution numérique d'équation différentielle de dimension quelconque, par la méthode d'Euler.

Vous avez vu dans le début de cette série comment écrire une fonction générale de résolution numérique d'équation différentielle de dimension 1, par la méthode de Runge.

En s'inspirant des deux fonctions précédentes, écrivez une fonction générale de résolution numérique d'équation différentielle de dimension quelconque, par la méthode de Runge.

---

4. Oscillateur amorti

Le but est de calculer numériquement l'évolution d'un oscillateur qui subit un frottement
Soit un corps d'une masse de 1 [kg] subissant une force résultante égale à
$F_{résultante} = – k \cdot x(t) – c \cdot v(t)$ où
$x(t)$ est la position du corps au temps t.
$V(t)$ est la vitesse du corps au temps t ;
Au temps $t = 0$ [s], le corps est immobile et se trouve en $x(0) = 0.1$ [m].
$k = 2$ [N / m].
$c = 0.3$ [N · s / m].
Reprenez l'énoncé 4 de la série 4.

4.1 Juste avant la fonction qui calculera l'évolution de cet oscillateur en fonction du temps, écrivez en commentaire l'équation différentielle qui régit son l'évolution.
4.2 Écrivez une "function ex4()" qui calcule par la **méthode d'Euler, comme décrit ci-dessus,** l'évolution de la vitesse du corps durant 20 secondes, en prenant nMax = 2000.
4.3 À la fin de votre fonction, tracez la courbe d'évolution de la position en fonction du temps.

---

5. Étude de la précision de la méthode de Runge

Le but est de calculer numériquement la solution d'une équation différentielle, puis de la comparer avec la solution exacte, qui est connue, pour étudier la précision de la méthode de Runge.
Résolvez numériquement :
$x'(t) = v(t)$
$v'(t) = -x(t)$
$x(0) = 0$; $v(0) = 1$;
Déterminez (trouvez, devinez) la solution exacte de ce système.

5.1 Juste avant la fonction qui calculera l'évolution de cet oscillateur en fonction du temps, écrivez en commentaire l'équation différentielle qui régit son l'évolution.
5.2 Écrivez une "function ex5()" qui calcule par la **méthode de Runge, comme décrit ci-dessus,** la solution numérique de ce système en $t = 2 \cdot \pi$.
5.3 À la fin de votre fonction, tracez la courbe obtenue numériquement de $v(t)$ et celle exacte, pour la comparer.
5.4 Pour différentes valeurs de nMax, entre 100 et 10'00, affichez la valeur de
$nMax^2 \cdot (v_{numérique}(2 \cdot \pi) – v_{exacte}(2 \cdot \pi))$.
Constatez qu'on obtient presque chaque fois la même valeur. Déduisez-en une information sur la précision de la méthode de Runge. On dit qu'elle est d'**ordre 2**.
5.5 Une méthode d'**ordre 2** a la propriété que : $nMax^2 \cdot (v_{numérique}(2 \cdot \pi) – v_{exacte}(2 \cdot \pi))$ ne varie presque pas en fonction de nMax. Décrivez son avantage.
5.6 La méthode de Runge-Kutta est d'**ordre 4**.
Que cela signifie-t-il ?
Quelle est son avantage ?

Information supplémentaire pour la curiosité :

On veut résoudre numériquement une équation différentielle : $y'(x) = f(x, y(x)),\quad y(x_0) = y_0$

La **méthode de Runge-Kutta** décrite ci-dessous est d'ordre 4, c'est-à-dire que l'erreur est proportionnelle à h^4.
Par exemple, un pas de 0,01 donne une erreur proportionnelle à 0,00000001.

Le schéma de la **méthode de Runge-Kutta** est le suivant : (c.f. CRM page 97)

$$k_1 = f(x_0\ ;\ y_0)$$
$$k_2 = f\left(x_0+\frac{h}{2}\ ;\ y_0+k_1\cdot\frac{h}{2}\right)$$
$$k_3 = f\left(x_0+\frac{h}{2}\ ;\ y_0+k_2\cdot\frac{h}{2}\right)$$
$$k_4 = f(x_0+h\ ;\ y_0+k_3\cdot h)$$
$$y_1 = y_0 + (k_1+2\cdot k_2+2\cdot k_3+k_4)\cdot\frac{h}{6}$$

Saurez-vous écrire un algorithme général de résolution utilisant la méthode de Runge-Kutta ?
On veut obtenir nMax+1 valeurs de y répartis régulièrement entre x0 et xFin.
(Une méthode d'ordre 5 ou plus est nettement plus compliquée, mais suit un schéma similaire.)

Après tous ces efforts, sachez qu'une méthode générale de résolution numérique d'équations différentielles est implémenté dans SciLab. Voici un exemple d'utilisation de la fonction **ode(...)**

```
// Définition de l'équation différentielle :  [y(1) ; y(2)]' = [y(2) ; -k*y(1)]
// Rien ne change par rapport à ce que l'on a déjà vu.
function z=f(x, y)
//================
// x représente le temps
// y(1)  représente la position
// y(2)  représente la vitesse
// z(1)  représente la dérivée de la position par rapport au temps
// z(2)  représente la dérivée de la vitesse par rapport au temps
k = 2.0;
z(1) = y(2);     // car  position' = vitesse
z(2) = -k*y(1);  // car  vitesse' = -k * position
endfunction

// données :
t0    =  0;   // temps initial
tFin  = 10;   // temps final
y0    = [0.1; 0];  // position = 0.1 [m] ; vitesse initiale = 0 [m/s]], vecteur COLONNE
N  = 200;      // nombre de points d'évaluation
ax = linspace(t0, tFin, N+1);  // vecteur LIGNE

// résolution numérique en appelant la fonction  ode  de SciLab
ay = ode( y0,..     // condition initiale
          t0,..     // x_initial,
          ax,..     // série des x(j) pour lesquels  y(j)  sera calculé
          1e-4,..   // tolérance d'erreur relative
          1e-5,..   // tolérance d'erreur absolue
          f..       // fonction définissant l'équation différentielle
          );
// L'algorithme essaye d'avoir une erreur inférieure au
// maximum entre l'erreur absolue et l'erreur relative.

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