TP. RÉSOLUTION NUMÉRIQUE DES EDO D'ORDRE UN

PR. BOUZIANI ABDELFATAH

Méthodes d'Euler (explicites, implicites).
Méthode d'Euler modifiée (Méthode de Heun).
Méthodes Runge-Kutta d'ordre 2, 3 et 4

1. MÉTHODES D'EULER

1.1. **Méthodes d'Euler explicite.** La méthode d'Euler explicite est l'une des techniques numériques les plus simples et les plus couramment utilisées pour résoudre des équations différentielles ordinaires (EDO). Cette méthode est un exemple de procédure de pas unique où la dérivée à un point est utilisée pour extrapoler la valeur de la fonction à l'étape suivante.

1.1.1. *Description de la Méthode d'Euler Explicite.* La méthode d'Euler explicite est une méthode de résolution d'équations différentielles du premier ordre qui peut s'écrire sous la forme générale suivante :

$$y_{i+1} = y_i + hf\left(x_i, y_i\right), \tag{1.1}$$

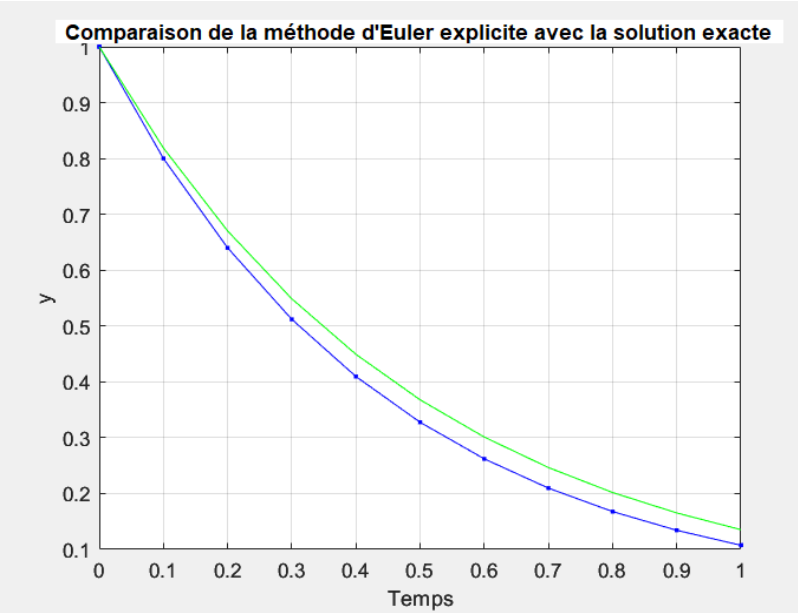
1.1.2. *Exemple Simple.* Supposons que nous voulons résoudre l'équation différentielle $\frac{dy}{dt} = -2y$ avec une condition initiale $y\left(0\right) = 1$. L'application de la méthode d'Euler explicite peut être visualisée avec les étapes suivantes :

1. Choisir un pas de temps h, par exemple $h = 0.1$.
2. Calculer la dérivée $f\left(t_i, y_i\right)$ dans cet exemple.
3. Utiliser la formule d'Euler explicite pour estimer y à chaque pas :

$$y_{i+1} = y_i - 2hy_i \tag{1.2}$$

1.1.3. *Exemple en MATLAB.* Pour coder cela en MATLAB et tracer la solution approximée par rapport à la solution exacte $y\left(t\right) = e^{-2t}$, voici un script simple :

```
% Paramètres
h = 0.1; % pas de temps
t_final = 1; % temps final
t = 0:h:t_final; % vecteur de temps
% Initialisation des vecteurs de solutions
y = zeros(1, length(t));
y(1) = 1; % condition initiale
% Calcul par la méthode d'Euler Explicite
for i = 1:length(t)-1
y(i+1) = y(i) - 2 * h * y(i);
end
% Solution exacte
y_exact = exp(-2 * t);
```



% Tracer les résultats
figure;
plot(t, y, 'b.-', 'DisplayName', '*Euler Explicite*');
hold on;
plot(t, y_exact, 'g', 'DisplayName', '*Solution exacte*');
title('*Comparaison de la méthode d"Euler explicite avec la solution exacte*');
xlabel('*Temps*');
ylabel('*y*');
legend show;
grid on;

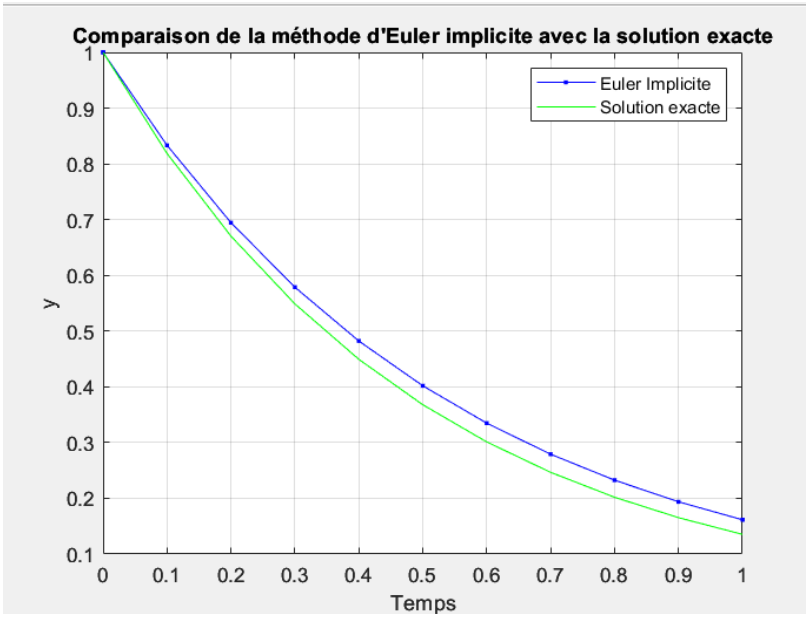
1.2. **Méthodes d'Euler implicites.** La méthode d'Euler implicite est une technique numérique pour résoudre des équations différentielles ordinaires (EDO).

1.2.1. *Description de la Méthode d'Euler Implicite.* La méthode d'Euler implicite est décrite par la formule suivante :

$$y_{i+1} = y_i + hf\left(x_{i+1}, y_{i+1}\right), \tag{1.3}$$

1.2.2. *Exemple en MATLAB.* Prenons l'équation $\dfrac{dy}{dt} = -2y$ avec une condition initiale $y\left(0\right) = 1$. Voici un exemple d'implémentation de la méthode d'Euler implicite en MATLAB :

% Paramètres
h = 0.1; **% pas de temps**
t_final = 1; **% temps final**
t = 0:h:t_final; **% vecteur de temps**
% Initialisation des vecteurs de solutions
y = zeros(1, length(t));
y(1) = 1; % condition initiale




```
% Calcul par la méthode d'Euler Implicite
for i = 1:length(t)-1
 % Méthode de Newton pour résoudre y_{n+1} = y_n - 2h y_{n+1}
y_next = y(i); % Initialisation de y_{n+1} pour l'itération
for j = 1:5 % Boucle de Newton avec un nombre fixe d'itérations
f = y_next - y(i) + 2 * h * y_next; % Fonction à annuler
df = 1 + 2 * h; % Dérivée de f
y_next = y_next - f / df; % Mise à jour de Newton
end
y(i+1) = y_next;
end
% Solution exacte
y_exact = exp(-2 * t);
% Tracer les résultats
figure;
plot(t, y, 'b.-', 'DisplayName', 'Euler Implicite');
hold on;
plot(t, y_exact, 'g', 'DisplayName', 'Solution exacte');
title('Comparaison de la méthode d"Euler implicite avec la solution exacte');
xlabel('Temps');
ylabel('y');
legend show;
grid on;
```

2. Méthode d'Euler modifiée (Méthode de Heun)

La méthode d'Euler modifiée, souvent appelée méthode de Heun, est une amélioration de la méthode d'Euler explicite qui offre une meilleure précision.

2.1. **Formule de la méthode de Heun.** La méthode de Heun est une méthode prédictive-corrective qui procède en deux étapes :

1. Étape prédictive : Une première estimation de y_{i+1} est calculée en utilisant la méthode d'Euler explicite

$$y_{i+1}^{(0)} = y_i + hf\left(x_i, y_i\right), \tag{20}$$

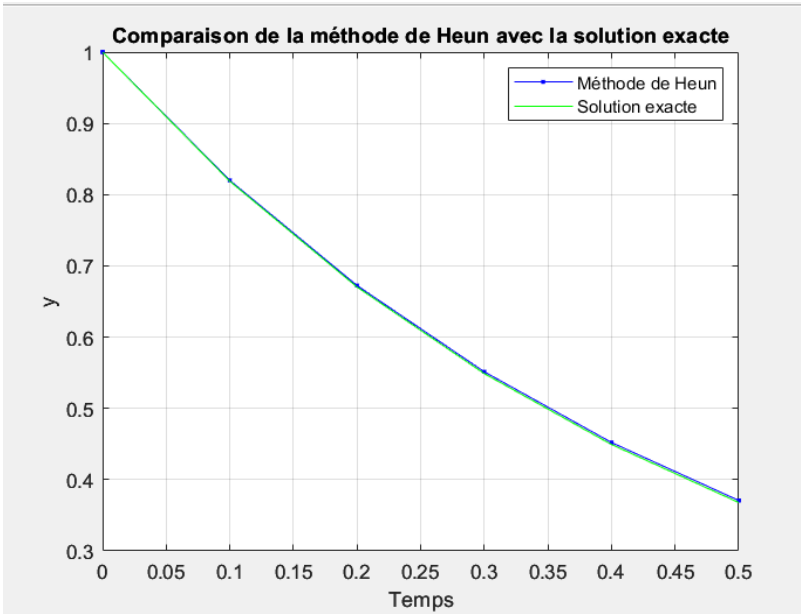
2. Étape corrective : Une meilleure estimation de la pente est obtenue en évaluant la fonction f à cette estimation préliminaire, puis en calculant la moyenne des pentes au début et à la fin de l'intervalle

$$y_{i+1} = y_i + \frac{h}{2}\left[f\left(x_i, y_i\right) + f\left(x_{i+1}, y_{i+1}^{(0)}\right)\right]. \tag{22}$$

2.1.1. *Exemple en MATLAB.* Prenons l'équation $\dfrac{dy}{dt} = -2y$ avec une condition initiale $y\left(0\right) = 1$. Voici un exemple d'implémentation de la méthode de Heun en MATLAB :

```
% Paramètres
h = 0.1; % pas de temps
t_final = 0.5; % temps final
t = 0:h:t_final; % vecteur de temps
% Initialisation des vecteurs de solutions
y_heun = zeros(1, length(t));
y_heun(1) = 1; % condition initiale
% Fonction définissant l'EDO
f = @(y) -2 * y;
% Calcul par la méthode de Heun
for i = 1:length(t)-1
y_predict = y_heun(i) + h * f(y_heun(i)); % Étape prédictive
y_correct = y_heun(i) + (h/2) * (f(y_heun(i)) + f(y_predict)); % Étape
corrective
y_heun(i+1) = y_correct;
end
% Solution exacte
y_exact = exp(-2 * t);
% Tracer les résultats
figure;
plot(t, y_heun, 'b.-', 'DisplayName', 'Méthode de Heun');
hold on;
plot(t, y_exact, 'g', 'DisplayName', 'Solution exacte');
title('Comparaison de la méthode de Heun avec la solution exacte');
xlabel('Temps');
ylabel('y');
legend show;
grid on;
```

2.1.2. ***Explications du code.*** 1. **Initialisation :** Nous commençons par définir le pas de temps **h**, le temps final **t_final**, et nous créons un vecteur de temps **t**. Le vecteur **y_heun** est initialisé avec la condition initiale.



2. **Définition de la fonction :** **f** est une fonction handle qui prend **y** comme argument et retourne **-2y**, qui est la dérivée de **y** selon notre équation différentielle.

3. **Boucle de calcul :**
   - **Étape prédictive :** Nous utilisons la valeur actuelle pour prédire la valeur suivante.
   - **Étape corrective :** Nous calculons une correction en utilisant la moyenne des pentes au début et à la fin de l'intervalle de temps, puis nous mettons à jour **y_heun**.

4. **Tracé des résultats :** Nous traçons les résultats de la méthode de Heun et de la solution exacte pour comparer visuellement leur précision.

Ce script vous permettra de comprendre comment la méthode de Heun peut être implémentée et utilisée pour obtenir des résultats numériques plus précis que la méthode d'Euler simple, en utilisant un exemple concret dans MATLAB.

3. METHODES DE RUNGE-KUTTA

3.1. **Méthodes Runge-Kutta du second ordre.**

(3.1)

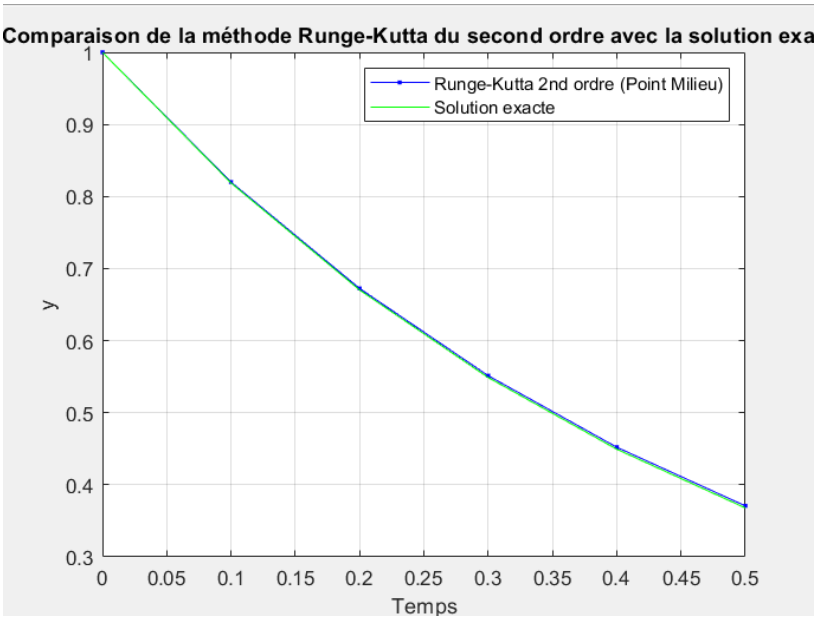
$$y_{i+1} = y_i + \frac{h}{2}\left(K_1 + K_2\right).$$

avec

$$K_1 = f\left(x_i, y_i\right)$$
$$K_2 = f\left(x_i + h, y_i + K_1 h\right)$$

3.1.1. *Exemple en MATLAB.* Prenons l'équation $\frac{dy}{dt} = -2y$ avec une condition initiale $y\left(0\right) = 1$. Voici un exemple d'implémentation de la méthode de Runge-Kutta du second ordre en MATLAB :

% Paramètres



h = 0.1; **% pas de temps**
t_final = 0.5; % **temps final**
t = 0:h:t_final; **% vecteur de temps**
% Initialisation des vecteurs de solutions
y_rk2 = zeros(1, length(t));
y_rk2(1) = 1; **% condition initiale**
% Fonction définissant l'EDO
f = @(t, y) -2 * y;
% Calcul par la méthode du Runge-Kutta du second ordre
for i = 1:length(t)-1
k1 = f(t(i), y_rk2(i));
k2 = f(t(i) + 0.5 * h, y_rk2(i) + 0.5 * h * k1);
y_rk2(i+1) = y_rk2(i) + h * k2;
end
% Solution exacte
y_exact = exp(-2 * t);
% Tracer les résultats
figure;
plot(t, y_rk2, 'b.-', 'DisplayName', '*Runge-Kutta 2nd ordre (Point Milieu)*');
hold on;
plot(t, y_exact, 'g', 'DisplayName', '*Solution exacte*');
title('*Comparaison de la méthode Runge-Kutta du second ordre avec la solution exacte*');
xlabel('*Temps*');
ylabel('*y*');
legend show;
grid on;

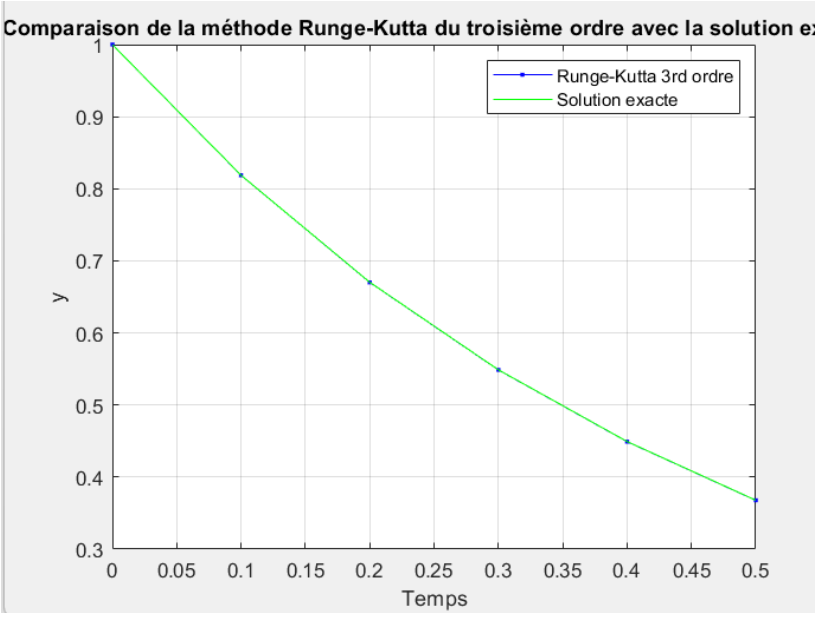
3.2. **Méthode Runge-Kutta du troisième ordre.** La méthode de Runge-Kutta du troisième ordre est une approche encore plus précise pour résoudre numériquement les équations différentielles ordinaires (EDO) par rapport aux méthodes du premier et du second ordre. Elle utilise trois évaluations intermédiaires de la fonction dérivée à chaque pas, permettant une meilleure approximation de la solution. Les méthodes de Runge-Kutta du troisième ordre sont particulièrement utiles pour les problèmes où une plus grande précision est nécessaire sans passer directement aux méthodes d'ordre supérieur comme celles du quatrième ordre.

3.2.1. *La forme générale de la méthode Runge-Kutta du troisième ordre classique est:*

$$
\begin{aligned}
& y_{i+1} = y_i + \frac{h}{6}\left(K_1 + 4K_2 + K_3\right), \\
& \text{où} \\
& K_1 = f\left(x_i, y_i\right), \\
& K_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}K_1 h\right), \\
& K_3 = f\left(x_i + h, y_i - K_1 h + 2K_2 h\right).
\end{aligned} \tag{37}
$$

3.2.2. *Exemple en MATLAB.* Prenons l'équation $\frac{dy}{dt} = -2y$ avec une condition initiale $y\left(0\right) = 1$. Voici un exemple d'implémentation de la méthode de Runge-Kutta du troisième ordre classique en MATLAB :

```
% Paramètres
h = 0.1; % pas de temps
t_final = 0.5; % temps final
t = 0:h:t_final; % vecteur de temps
% Initialisation des vecteurs de solutions
y_rk3 = zeros(1, length(t));
y_rk3(1) = 1; % condition initiale
% Fonction définissant l'EDO
f = @(t, y) -2 * y;
% Calcul par la méthode de Runge-Kutta du 3ème ordre
for i = 1:length(t)-1
k1 = f(t(i), y_rk3(i));
k2 = f(t(i) + 0.5 * h, y_rk3(i) + 0.5 * h * k1);
k3 = f(t(i) + h, y_rk3(i) - h * k1 + 2 * h * k2);
y_rk3(i+1) = y_rk3(i) + h * (1/6 * k1 + 2/3 * k2 + 1/6 * k3);
end
% Solution exacte
y_exact = exp(-2 * t);
% Tracer les résultats
figure;
plot(t, y_rk3, 'b.-', 'DisplayName', 'Runge-Kutta 3rd ordre');
hold on;
plot(t, y_exact, 'g', 'DisplayName', 'Solution exacte');
title('Comparaison de la méthode Runge-Kutta du troisième ordre avec la
solution exacte');
```




```
xlabel('Temps');
ylabel('y');
legend show;
grid on;
```

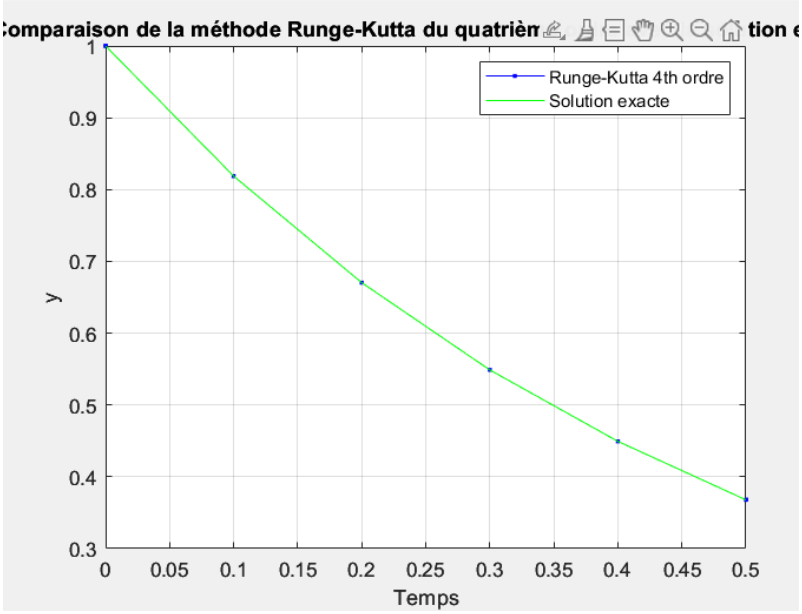
3.3. **Méthode Runge-Kutta du quatrième ordre.** La méthode de Runge-Kutta du quatrième ordre (RK4) est l'une des techniques numériques les plus utilisées pour résoudre des équations différentielles ordinaires (EDO) en raison de son excellent compromis entre précision et coût de calcul. Elle est très prisée dans divers domaines scientifiques et d'ingénierie pour sa capacité à produire des résultats précis avec un pas de temps relativement large.

La forme générale des méthodes Runge-Kutta du quatrième ordre est:

$$
\begin{aligned}
&y_{i+1} = y_i + \frac{h}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right), \quad i = 1, \cdots, n, \\
&\text{où} \\
&K_1 = f\left(x_i, y_i\right) \\
&K_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}K_1 h\right) \\
&K_3 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}K_2 h\right) \\
&K_4 = f\left(x_i + h, y_i + K_3 h\right)
\end{aligned}
\tag{38}
$$

3.3.1. *Exemple en MATLAB.* Pour illustrer l'application de la méthode RK4, considérons l'équation différentielle $\frac{dy}{dt} = -2y$ avec une condition initiale $y(0) = 1$. Voici comment vous pourriez implémenter cette méthode en MATLAB :

```
% Paramètres
h = 0.1; % pas de temps
t_final = 0.5; % temps final
```




```
t = 0:h:t_final; % vecteur de temps
% Initialisation des vecteurs de solutions
y_rk4 = zeros(1, length(t));
y_rk4(1) = 1; % condition initiale
% Fonction définissant l'EDO
f = @(t, y) -2 * y;
% Calcul par la méthode RK4
for i = 1:length(t)-1
k1 = f(t(i), y_rk4(i));
k2 = f(t(i) + 0.5 * h, y_rk4(i) + 0.5 * h * k1);
k3 = f(t(i) + 0.5 * h, y_rk4(i) + 0.5 * h * k2);
k4 = f(t(i) + h, y_rk4(i) + h * k3);
y_rk4(i+1) = y_rk4(i) + (h/6) * (k1 + 2*k2 + 2*k3 + k4);
end
% Solution exacte
y_exact = exp(-2 * t);
% Tracer les résultats
figure;
plot(t, y_rk4, 'b.-', 'DisplayName', 'Runge-Kutta 4th ordre');
hold on;
plot(t, y_exact, 'g', 'DisplayName', 'Solution exacte');
title('Comparaison de la méthode Runge-Kutta du quatrième ordre avec la
solution exacte');
xlabel('Temps');
ylabel('y');
legend show;
grid on;
```