

TP Avancé : La Méthode Runge-Kutta (RK4)

Le « Couteau Suisse » de la simulation physique

Dans les sessions précédentes, nous avons utilisé la méthode d'Euler. Elle est simple, mais elle « dérape » vite car elle suppose que la vitesse est constante durant tout le pas de temps.

La méthode de Runge-Kutta d'ordre 4 (RK4) corrige ce problème en étant beaucoup plus astucieuse : au lieu de regarder la pente (dérivée) uniquement au début, elle « sonde » le terrain à 4 endroits différents pour trouver une trajectoire moyenne quasi-parfaite.

1. Le Concept : 4 sondes valent mieux qu'une

Imaginez que vous êtes aveugle et que vous devez traverser une vallée vallonnée en un pas de temps Δt .

- **Euler (k_1)** : Vous tendez le pied, sentez la pente actuelle, et faites un grand saut dans cette direction. → *Risqué*.
- **RK4** : Vous envoyez des éclaireurs virtuels avant de bouger.

1. **k_1 (Le Départ)** : Pente au début (comme Euler).
2. **k_2 (Le Milieu A)** : On utilise la pente k_1 pour avancer jusqu'à la moitié du pas ($\Delta t/2$) et on regarde la pente là-bas.
3. **k_3 (Le Milieu B)** : On se méfie de k_2 . On repart du début, mais cette fois on utilise la pente k_2 pour aller au milieu. On regarde la nouvelle pente.
4. **k_4 (L'Arrivée)** : On utilise la pente k_3 pour aller jusqu'à la fin du pas (Δt) et on regarde la pente finale.

Le Grand Final : On fait une moyenne pondérée de ces 4 pentes. On donne plus de poids aux pentes du milieu (k_2 et k_3).

$$\text{Pente Finale} = \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$$

2. Les Mathématiques (L'État du Système)

Pour implémenter RK4 proprement, nous devons regrouper nos variables (Position et Vitesse) dans un seul vecteur que nous appellerons l'État (Y).

Si on a un système physique simple :

$$Y = \begin{pmatrix} x \\ v \end{pmatrix}$$
$$\dot{Y} = f(t, Y) = \begin{pmatrix} v \\ a \end{pmatrix}$$

La dérivée de l'état (le changement), c'est la vitesse et l'accélération.

Voici l'algorithme complet pour un pas de temps h (ou Δt) :

Algorithme RK4 pour un pas h :

Soit la fonction `eval(état)` qui retourne la dérivée [vitesse, accélération].

1. k_1 (Début) = `eval(état_actuel)`
2. k_2 (Milieu) = `eval(état_actuel + $k_1 \times \frac{h}{2}$)`
3. k_3 (Milieu) = `eval(état_actuel + $k_2 \times \frac{h}{2}$)`
4. k_4 (Fin) = `eval(état_actuel + $k_3 \times h$)`

$$\text{Nouvel État} = \text{état_actuel} + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \times h$$

3. Implémentation (Pseudo-Code / JavaScript)

Contrairement à Euler où l'on écrit `pos += vel * dt` directement, RK4 nécessite une structure plus organisée.

```
// Définition d'un État
struct State {
    Vector3 position;
    Vector3 velocity;
};

// La dérivée de l'état, c'est ce qui change
struct Derivative {
    Vector3 d_position; // C'est la vitesse
    Vector3 d_velocity; // C'est l'accélération (Forces / Masse)
};

// Fonction qui calcule les forces et retourne la dérivée
function evaluate(initialState, t, dt, derivative) {
    // 1. Estimer l'état futur basé sur la dérivée précédente
    State state = initialState;
    state.position += derivative.d_position * dt;
    state.velocity += derivative.d_velocity * dt;

    // 2. Calculer les forces à cet endroit/vitesse là
    // (Exemple: Gravité + Vent + Ressort)
    Vector3 forces = calculateForces(state.position, state.velocity);
    Vector3 acceleration = forces / mass;

    // 3. Retourner la nouvelle dérivée
    return new Derivative(state.velocity, acceleration);
}

// BOUCLE PRINCIPALE (INTÉGRATEUR)
function integrateRK4(state, t, dt) {
    // a. Préparer les 4 échantillons
    Derivative a = evaluate(state, t, 0.0, new Derivative()); // k1
    Derivative b = evaluate(state, t, dt*0.5, a); // k2
    Derivative c = evaluate(state, t, dt*0.5, b); // k3
    Derivative d = evaluate(state, t, dt, c); // k4
```

```
// b. Moyenne pondérée pour la position (dxdt)
Vector3 dxdt = (a.d_pos + 2*(b.d_pos + c.d_pos) + d.d_pos) * 1/6;

// c. Moyenne pondérée pour la vitesse (dvdt)
Vector3 dvdt = (a.d_vel + 2*(b.d_vel + c.d_vel) + d.d_vel) * 1/6;

// d. Mise à jour finale
state.position += dxdt * dt;
state.velocity += dvdt * dt;
}
```