# Résolution d'équations différentielles

SIA Ensimag 3A

Estelle Duveau

# Inspiration

- **Andrew Witkin et David Baraff - Physically Based Modeling cours à Siggraph 2001**
- Cours de Marie-Paule Cani, François Faure, Nicolas Holzschuch

# Plan

# Plan

# Equations différentielles

- Relation entre une ou plusieurs fonctions inconnues et leurs dérivées
- Equations différentielles ordinaires (ODE) : fonction d'une variable
- Equation aux Dérivées Partielles : fonction de plusieurs variables
- Forme générique d'une ODE premier degré :

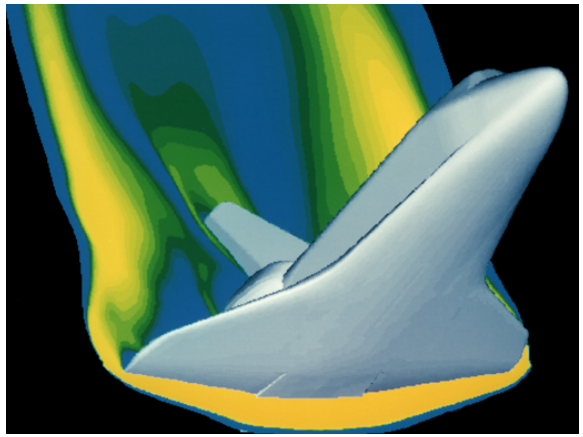
$$\frac{dX(t)}{dt} = F(X(t), t)$$

$$X : \mathbb{R} \rightarrow \mathbb{R}^n$$

$$F : \mathbb{R}^n x \mathbb{R} \rightarrow \mathbb{R}^n$$

# A quoi ça sert?

- A tout...ou presque!
  - Chimie
  - Physique
  - Ingénierie
  - Economie...
- En informatique graphique :
  - Animation
  - Modélisation
  - Rendu...

# Résolution des équations différentielles

- Au mieux : résolution analytique
- Nombreux problèmes sans solution analytique
  $\Rightarrow$ Résolution numérique
- Résolution numérique :

$$X(t_0) = X_0$$

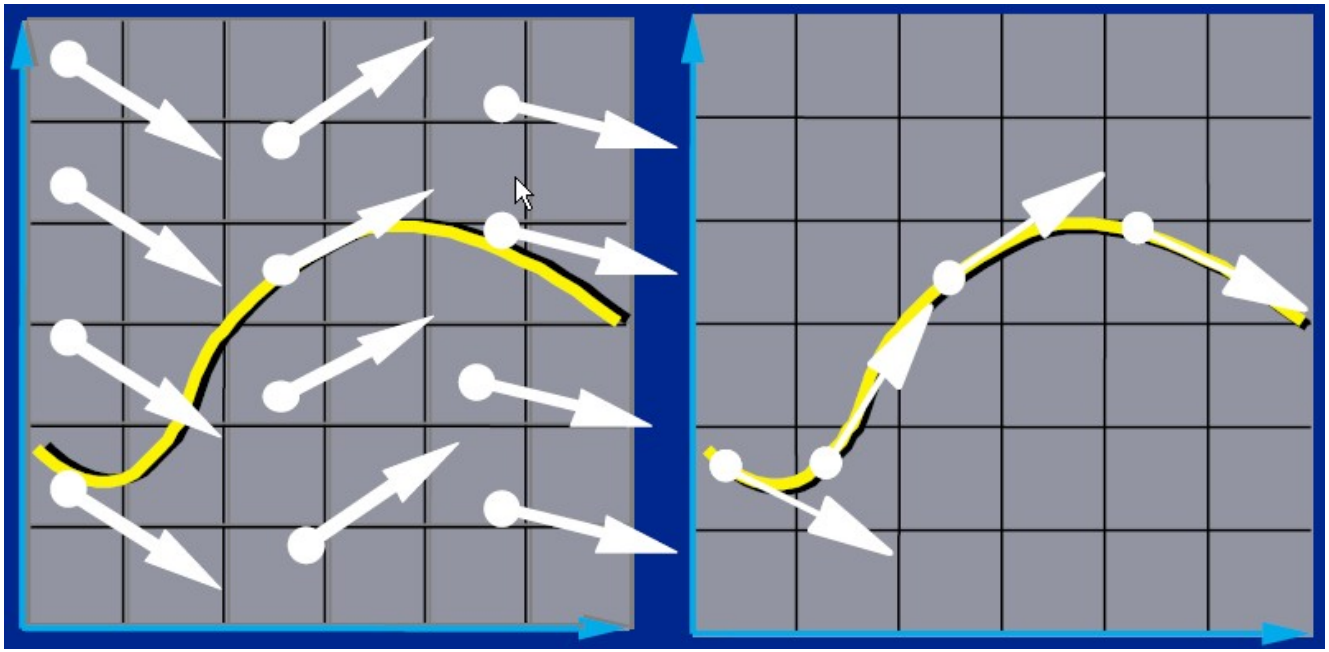
$$\frac{dX(t)}{dt} = F(X(t), t) \quad t > t_0$$

  - Etant donné $F(X(t), t)$, calculer $X(t)$
  - Pour l'animation, échantillons de $X(t)$ :

$$X(t_i) \quad t_i = t_0, t_1, t_2, \ldots$$

# Intuition

- $F(X, t)$ est un champ de vecteurs de $\mathbb{R}^n$
- $X(t)$ est un chemin dans ce champ / une trajectoire
- $X_0$ est le point de départ
- Résolution :
  1. Etat courant $X(t_i)$
  2. Calculer $F(X(t_i), t)$ : valeur du champ de vecteurs à l'état courant
  3. **Avancer d'un pas**
  4. Changer l'état courant $X(t_{i+1})$



# ODE d'ordre 2

- Exemple : 2ème loi de Newton :

$$\frac{d^2}{dt^2}x = \frac{f}{m}$$

- On se ramène à une ODE d'ordre 1 :

$$\begin{cases} \frac{d}{dt}x(t) = v(t) \\ \frac{d}{dt}v(t) = \frac{1}{m}f(x, v, t) \end{cases}$$

$$X = \begin{pmatrix} x \\ v \end{pmatrix} \quad F(X, t) = \begin{pmatrix} v \\ \frac{f(x,v,t)}{m} \end{pmatrix}$$

# Plan

# Plan

# Méthode d'Euler

- Méthode la plus intuitive
- Pas de temps donné $h$
- Avancer d'un pas :

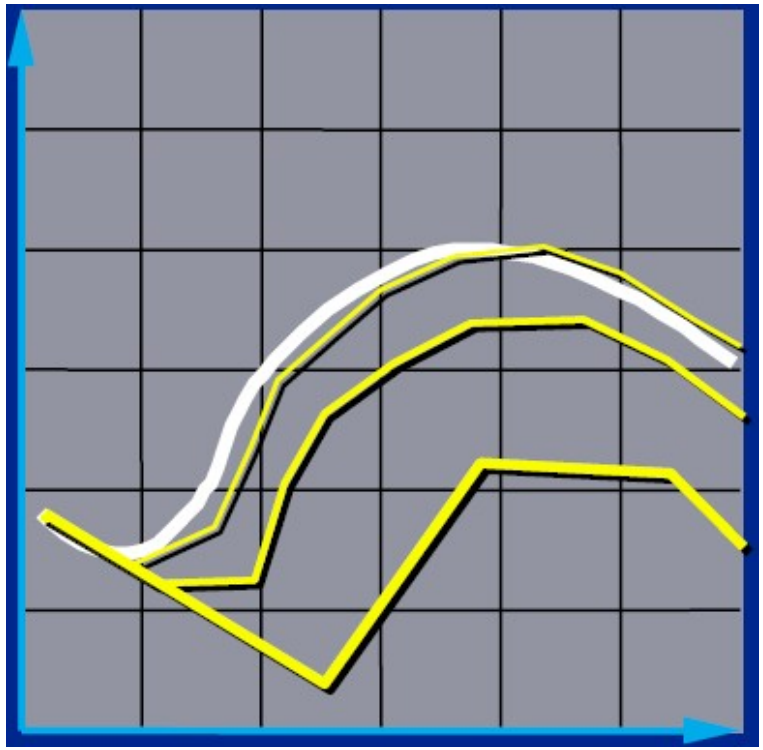
$$t_{i+1} = t_i + h$$

$$X(t_{i+1}) = X(t_i) + hF(X(t_i), t_i)$$

- $\Rightarrow$ Approximation linéaire par morceau de la trajectoire : précision en $O(h)$

# Taille des pas

- Contrôle la précision
- Petits pas : plus long mais suit la courbe de plus près
- Grands pas : plus rapide mais moins précis



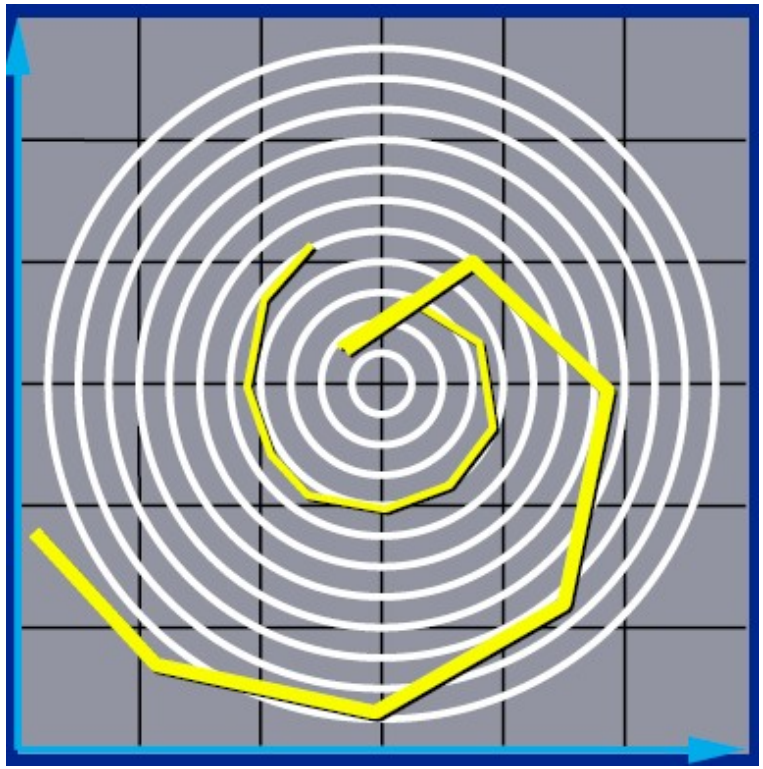
# Problème I : Précision

- Exemple :

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = F(\begin{pmatrix} x \\ y \end{pmatrix}, t) = \begin{pmatrix} -y \\ x \end{pmatrix}$$

- Solution : cercles

- Euler part en spirale même avec des pas très petits



# Problème II : Instabilité

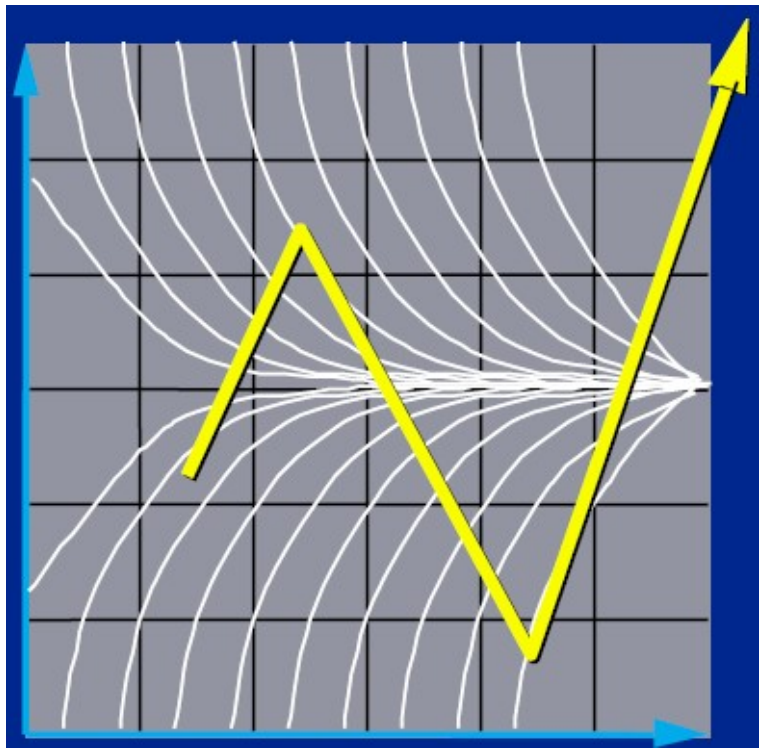
- Exemple :

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = F(\begin{pmatrix} x \\ y \end{pmatrix}, t) = \begin{pmatrix} -kx \\ 1 \end{pmatrix}$$

- Solution : exponentielles
- Solution numérique dépend de la taille du pas $h$ :

$$x(t_{i+1}) = x(t_i)(1 - kh)$$

  - $h \leq 1/k$ ok
  - $1/k < h \leq 2/k$ oscillations
  - $2/k < h$ divergence



# Ordres supérieurs

- Méthode du trapèze (ordre 2)
- Méthode du point milieu (ordre 2)
- Méthode de Runge-Kutta (ordres plus élevés)

# Plan

# Exercice 1 : Coder Euler explicite

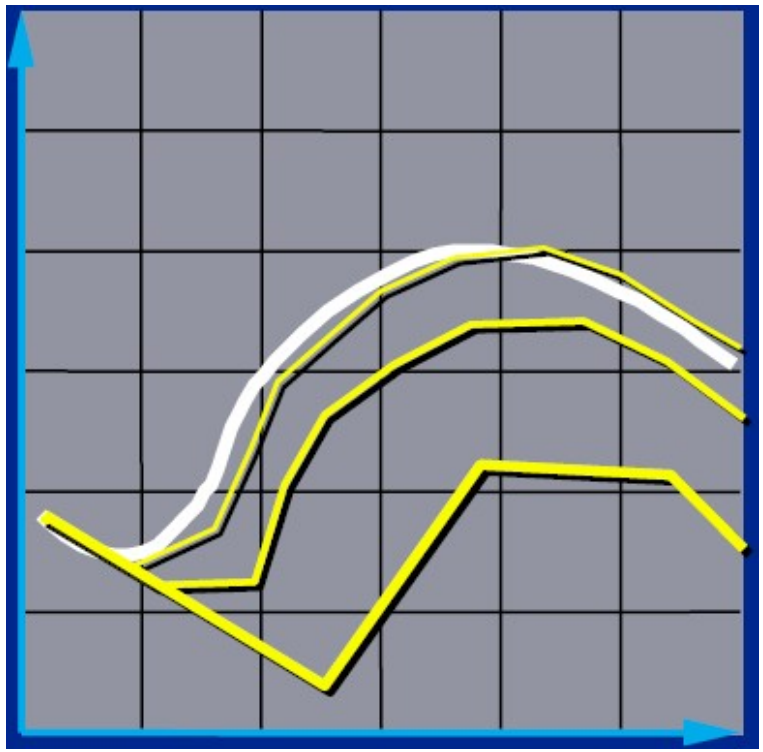
- `animPhys-ODE.cpp` : affichage de la trajectoire solution (2D), interface (flèches : déplacement, `a`/`z` : zoom, `Echap` : quitter), `main`
- module `Matrix` : calcul matriciel
- module `Vec` : calcul vectoriel
- module `Solver` :
  - L'état courant est stocké dans *pos* : c'est donc ce que l'on cherche à calculer
  - 2 problèmes : décroissances exponentielles et cercles (vus précédemment)
  - Paramètres à expérimenter (dans constructeur de `Solver`) :
    - ★ *_k* : raideur dans le cas de la décroissance exponentielle
    - ★ *_step* : pas de temps
    - ★ *_problem* : problème choisi
    - ★ *pos*0 : conditions initiales
  - Fonctions :
    - ★ `callProblem(Vec x, Vec* r)` calcule $F(X(t_i), t_i)$ avec $X(t_i) = x$ et stocke le résultat dans $r$ : $r = F(x)$
    - ★ `eulerExpFixStep()` avance d'un pas dans la résolution en utilisant Euler explicite : modifie donc *pos*
- Exercice :
  - Codez *eulerExpFixStep()*
  - Expérimentez avec les paramètres et les problèmes fournis
  - Retrouvez-vous les comportements vus en cours?

# Plan

# Plan

# Comment choisir le pas?

- Trop grand : erreurs
- Trop petit : long
- Pas idéal :
  - Aussi grand que possible sans trop d'erreurs
  - Grands pas dans les endroits faciles
  - Petits pas dans les endroits difficiles
- $\Rightarrow$ Adapter la taille du pas aux difficultés au cours du calcul



# Pas variable automatique

1. Pas initial $h$
2. **Estimer l'erreur commise**
3. Si l'erreur est petite : valider le résultat (et augmenter $h$)
4. Si le résultat n'est pas validé : **diminuer** $h$ et recommencer le calcul

# Estimer l'erreur

- Calcul :
  - 1 pas de temps avec $h \Rightarrow X_a$
  - 2 pas de temps avec $h/2 \Rightarrow X_b$
  - Erreur estimée $= \|X_a - X_b\|$
- Estimation :
  - Facile à calculer
  - Peut être erroné
  - $\Rightarrow$ Raisonnablement efficace
- Choix du nouveau pas :
  Pour une méthode d'ordre $j$, erreur on $O(h^{j+1})$

$$h_{new} = h_{old}\left(\frac{tolerance}{erreur}\right)^{\frac{1}{j+1}}$$

# Plan

# Exercice 2 : Coder Euler à pas variable

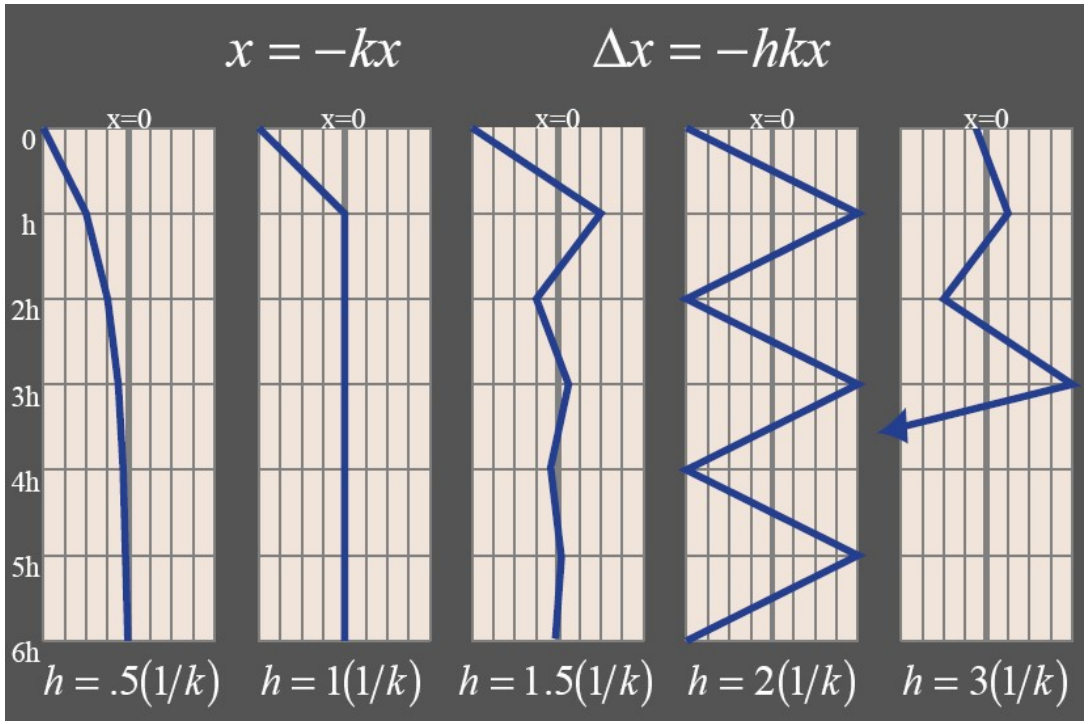
- Module `Solver` :
  - Dans le constructeur, *_method* permet de choisir la méthode pour avancer d'un pas
  - Dans le constructeur, *_tol* permet de régler la tolérance à l'erreur
  - `eulerExpVarStep()` avance d'un pas dans la résolution en utilisant Euler à pas variable
- Exercice :
  - Codez `eulerExpVarStep()`
  - Expérimentez avec les paramètres et les problèmes fournis : en particulier, observez l'évolution du pas de temps
  - Voyez-vous une amélioriation de la qualité de l'approximation?

# Plan

# Plan

# Méthodes explicites

- Pas de temps trop grand : simulation instable
- Pas de temps trop petit : ça n'avance pas!
- Ressorts de rappel :
  - pas de temps limité par la plus grande raideur $k$
  - $\Rightarrow$ un seul ressort peut tout gâcher!
  - Systèmes rigides : systèmes avec quelques raideurs très élevées



# Euler implicite

- Connu : $X(t_i), t_i, t_{i+1} = t_i + h$
- Inconnues : $X(t_{i+1})$
- Euler explicite :

$$X(t_{i+1}) = X(t_i) + hF(X(t_i), t_i)$$

- Euler implicite :

$$X(t_{i+1}) = X(t_i) + hF(X(t_{i+1}), t_{i+1})$$

$\Rightarrow X(t_{i+1})$ définie par une équation implicite

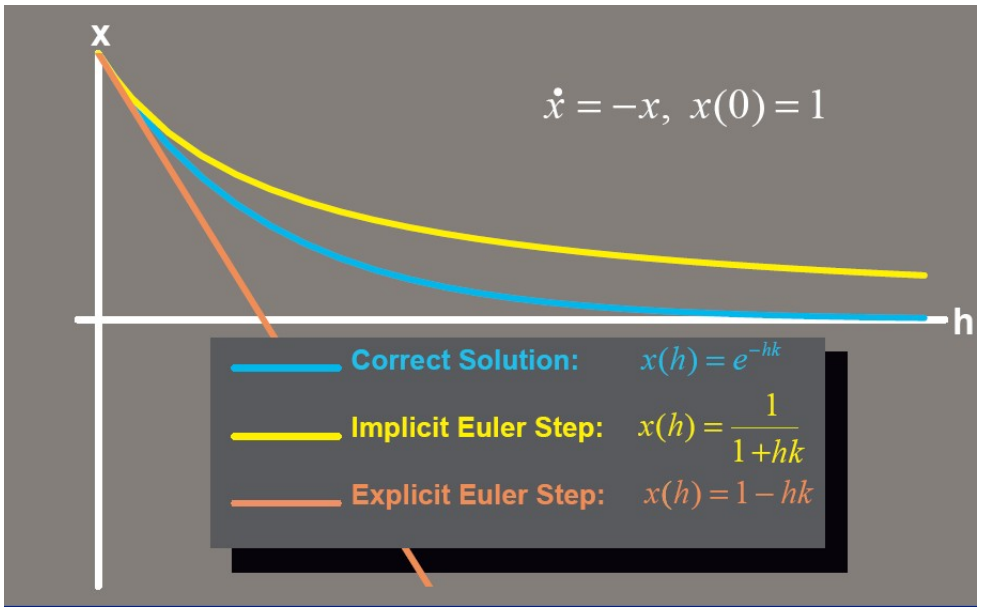
# Exemple : ressort de rappel

$$F(X(t), t) = -kX(t)$$

$$\begin{aligned} X(t_{i+1}) &= X(t_i) + hF(X(t_{i+1}), t_{i+1}) \\ &= X(t_i) - hkX(t_i + h) \end{aligned}$$

$$X(t_i + h) = \frac{X(t_i)}{1 + hk}$$

$\Rightarrow$ **Pas de limite sur** $h$



# Cas général

$$X(t_{i+1}) = X(t_i) + hF(X(t_{i+1}), t_{i+1})$$

Déplacement :

$$\Delta X = hF(X(t_i) + \Delta X, t_{i+1})$$

Linéarisation :

$$\frac{1}{h}\Delta X = F(X(t_i), t_{i+1}) + (\frac{\partial F}{\partial X})\Delta X$$

Factorisation :

$$(\frac{1}{h}I - \frac{\partial F}{\partial X})\Delta X = F(X(t_i), t_{i+1})$$

Résolution :

$$\Delta X = (\frac{1}{h}I - J(X(t_i), t_{i+1}))^{-1} F(X(t_i), t_{i+1})$$

# Euler implicite

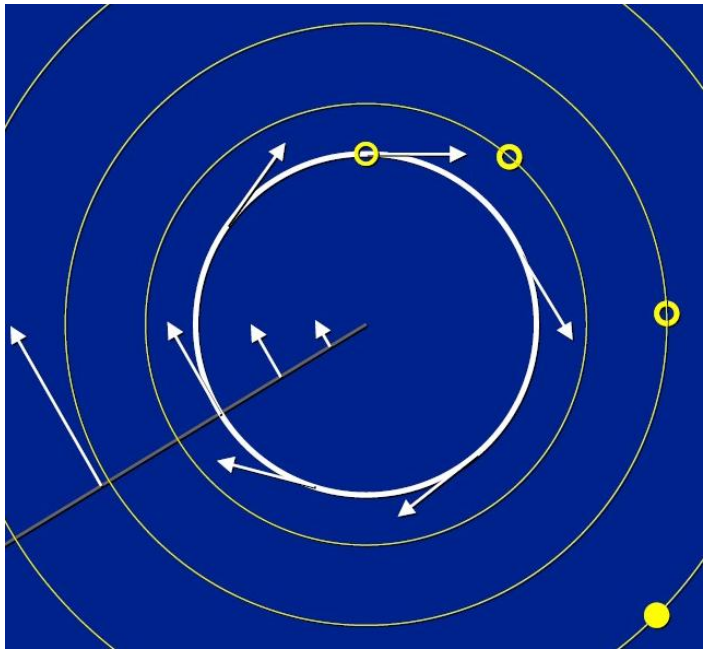
$$X(t_{i+1}) = X(t_i) + (\frac{1}{h}I - J(X(t_i), t_{i+1}))^{-1} F(X(t_i), t_{i+1})$$

- Besoin de calculer le jacobien
- Inversion d'une matrice à chaque étape
  - $J$ souvent creuse
  - $J$ souvent mal conditionnée ou singulière
- Programme plus compliqué
- Système **très** stable
- Pas de temps beaucoup plus grands sans divergence

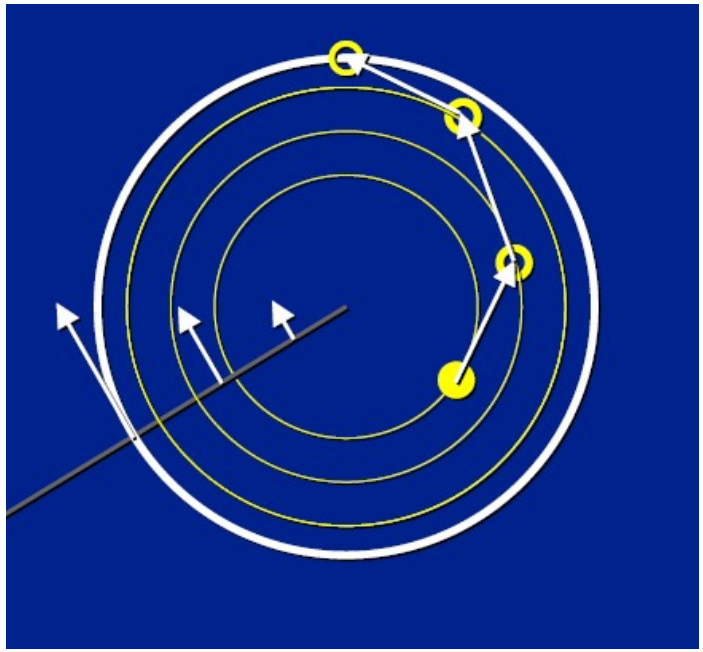
# Problème : précision

- Diminue les hautes fréquences
- Dissipation d'énergie
-

Euler explicite



Euler implicite



# Plan

# Exercice 3 : Coder Euler implicite

- Module `Solver` :
  - `setJacobien(Vec x, Matrix* J)` calcule le jacobien $J(X(t_i))$ où $X(t_i) = x$ et stocke le résultat dans $J$ : $J = J(X(t_i))$
  - `eulerImp()` avance d'un pas dans la résolution en utilisant Euler implicite
- Exercice :
  - Codez `eulerImp()`
  - Expérimentez avec les paramètres et les problèmes fournis
  - Voyez-vous une amélioriation de la qualité de l'approximation? du pas de temps nécessaire?
  - Observez-vous la dissipation d'énergie?

# Plan

# Bilan

- Euler explicite : très simple, plus ou moins rapide, instable
- Euler variable : simple, lent, stable
- Euler implicite : compliqué, rapide, stable
- Problème précision