```
open Maths
open Constantes
open Graph
open Types

(*fonction qui donne l'acceleration en fonction de dt, la position et la vitesse*)
let f _ y y' args =
  let blob = to_points y y' args.points in
  let vol = volume args.points in
  Graph.mapi
    (fun i p ->
      if collision p.pos then
      begin
        y.(i) <- fix_collision p.pos;
        y'.(i) <- zero
      end;
      (somme_forces (Force.bilan_des_forces p i vol blob args) p ) /$ p.mass
    )
  blob

let runge_kunta args =
  let h = dt in
  let h2 = h/.2.0 in
  let hh4 = h*.h/.4.0 in
  let h6 = h/.6.0 in
  let y = Graph.map (fun x -> x.pos) args.points in
  let y' = Graph.map (fun x -> x.vit) args.points in
  let k1 = f 0.0 y y' args in
  let k2 = f h2 (y +% h2 *% y') (y' +% h2 *% k1) args in
  let k3 = f h2 (y +% h2 *% y' +% hh4 *% k1) (y' +% h2 *% k2) args in
  let k4 = f h (y +% h *% y' +% (h*.h2) *% k2) (y' +% h *% k3) args in
  let ny = y +% h *% y' +% (h*.h6) *% (k1 +% k2 +% k3) in
  let ny' = y' +% h6 *% (k1 +% 2.0 *% k2 +% 2.0 *% k3 +% k4) in
  to_points ny ny' args.points

(*diminuer dt avant d'utiliser ces méthodes*)
let verlet args =
  let y' = Graph.map (fun x -> x.vit) args.points in
  let prec = Graph.map (fun x -> x.pos) args.points in
  let current = prec +% (dt *% y') in
  let next = 2.0 *% current -% prec +% (dt*.dt *% f dt current y' args) in
  to_points next ((1.0 /.dt) *% (next -% current)) args.points

let euler args =
  let y = Graph.map (fun x -> x.pos) args.points in
  let y' = Graph.map (fun x -> x.vit) args.points in
  let next = y +% dt *% y' in
  let next' = y' +% dt *% f dt y y' args in
  to_points next next' args.points

let integrate args =
  if not animate then args.points else
  match meth with
  | "rk" -> runge_kunta args
  | "euler" -> euler args
  | "verlet" -> verlet args
  | _ -> failwith "methode inconnue"
```