

# Simulation du système solaire

Konrad HINSEN

Centre de Biophysique Moléculaire, CNRS Orléans

et

Synchrotron SOLEIL, Saint Aubin

Un des exemples utilisés dans les travaux pratique de cette formation est un simulateur du système solaire. Ce programme lit d'un fichier une description du système solaire (soleil et planètes) qui contient les masses ainsi que les positions et vitesses initiales pour chaque corps céleste. Par la suite, le programme trace les trajectoires de ces objets en appliquant l'algorithme suivant dans une boucle :

1. Calculer les forces d'interaction :

$$\mathbf{f}_i = -G \sum_{j \neq i} \frac{m_i m_j}{|\mathbf{r}_i - \mathbf{r}_j|^3} (\mathbf{r}_i - \mathbf{r}_j) \quad (1)$$

où  $G$  est la constante gravitationnelle,  $m_i$  est la masse du corps  $i$ , et  $\mathbf{r}_i$  est sa position. On voit que chaque terme dans la somme contribue aux forces agissantes sur deux corps,  $i$  et  $j$ . Pour gagner un facteur 2 en vitesse, on utilise donc une boucle sur les paires à l'intérieur de laquelle on rajoute une contribution aux deux forces.

2. Appliquer les équations de Newton,

$$\frac{d^2}{dt^2} \mathbf{r}_i = \frac{1}{m_i} \mathbf{f}_i, \quad (2)$$

sous la forme discrétisée suivante :

$$\begin{aligned} \mathbf{v}_i(t + \Delta t/2) &= \mathbf{v}_i(t) + \frac{\Delta t}{2} \frac{1}{m_i} \mathbf{f}_i(t) \\ \mathbf{r}_i(t + \Delta t) &= \mathbf{r}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t/2) \\ \mathbf{v}_i(t + \Delta t) &= \mathbf{v}_i(t + \Delta t/2) + \frac{\Delta t}{2} \frac{1}{m_i} \mathbf{f}_i(t + \Delta t) \end{aligned} \quad (3)$$

$$(4)$$

Cette discrétisation, choisie ici pour sa simplicité, est connue comme l'algorithme de Verlet et couramment utilisée dans les simulations de dynamique moléculaire pour sa stabilité.

À première vue, cet algorithme semble nécessiter deux évaluations des forces par pas de temps. Mais le  $\mathbf{f}_i(t + \Delta t)$  est identique au  $\mathbf{f}_i(t)$  du pas suivant. Il y a donc en total  $n + 1$  évaluations des forces pour  $n$  pas d'intégration.

Le fichier d'entrée, dont le nom est `systeme_solaire`, consiste de cinq lignes pour chacun des corps célestes avec les données suivantes :

1. son nom
2. sa position initiale (x, y, z) en AU
3. son vecteur de vitesse initial (x, y, z) en AU/h
4. sa masse en multiples de 1028 livres américains
5. son diamètre en AU

Par exemple, les cinq lignes qui décrivent le soleil sont

Sun

```
0.642737 -0.0447062 -0.0154626
0.000104211 0.001252 -1.30241e-05
1.98864e+06
0.696
```

Une version élargie de cette description du système solaire, qui contient toutes les lunes et un bon nombre d'astéroïdes, se trouve dans le fichier `systeme_solaire_complet`. Les données ont été obtenues du site <http://www.orbitsimulator.com/> et proviennent ultérieurement de la base de données HORIZONS de la NASA (<http://ssd.jpl.nasa.gov/?horizons>).

La version basique (et peu efficace) de ce programme est commentée ci-dessous. Au cours des travaux pratiques, il sera amélioré et optimisé en appliquant plusieurs techniques.

Les unités choisies pour les calculs sont

- longueur :  $10^9$  m
- temps :  $10^5$  s
- masse :  $10^{24}$  kg
- force :  $10^{23}$  N

Avec ce choix d'unités, les équations données ci-dessus conservent leur forme sans aucun préfacteur supplémentaire.

## Le programme commenté

### Importation de modules

```
import itertools
from Scientific.Geometry import Vector
import visual
```

Le programme utilise

- le module `itertools` de la bibliothèque standard Python
- le module `Geometry` de la bibliothèque Scientific Python
- le module `visual` de la bibliothèque de visualisation VPython

## Quelques constantes (en unités internes)

```
h = 60*60/1.e5 # Heure
G = 6.67428e-11/(1e9**3/1.e24/1.e5**2) # Constante gravitationnelle

delta_t = 0.05 # Pas d'integration
```

## Lectures des données

La fonction `group` regroupe les  $n$  éléments successifs d'une liste. Elle est utilisée pour convertir la liste des lignes lues du fichier dans une liste de spécifications (5 lignes chacune) de corps célestes.

```
def group(seq, n):
    return itertools.izip(*[itertools.islice(seq, i, None, n)
                           for i in range(n)])

names = []
masses = []
radii = []
positions = []
velocities = []
for data in group(file('systeme_solaire').readlines(), 5):
    names.append(data[0].strip())
    positions.append(Vector(*[float(x) for x in data[1].split()]))
    velocities.append(Vector(*[float(x) for x in data[2].split()]))
    masses.append(float(data[3].strip()))
    radii.append(float(data[4].strip()))
```

## Visualisation

La visualisation consiste d'une boule jaune pour chaque corps céleste et une lignes verte pour sa trajectoire depuis le début de la simulation. Ici ces objets sont créés avec les positions initiales. Au cours de l'intégration, les boules sont avancées en affectant une nouvelle valeur à l'attribut `pos`, et les lignes sont prolongées en rajoutant un nouveau point.

```
spheres = [visual.sphere(pos=visual.vector(p[0], p[1], p[2]),
                                     radius = 10.,
                                     color = visual.color.yellow)
            for p in positions]
curves = [visual.curve(color = visual.color.green, radius=3.)
          for i in range(len(masses))]
```

## Calcul des forces

Les forces sont calculées dans une boucle sur tous les pairs de corps célestes.

```
def calc_forces(masses, positions):
    n = len(masses)
    assert n == len(positions)
    forces = [Vector(0., 0., 0.) for i in range(n)]
    for i in range(n):
        for j in range(i+1, n):
            r_ij = positions[i]-positions[j]
            r_ij_abs = r_ij.length()
            f_ij = G*masses[i]*masses[j]*r_ij/r_ij_abs**3
            forces[i] = forces[i] - f_ij
            forces[j] = forces[j] + f_ij
    return forces
```

## Intégration des équations de mouvement

Il s'agit d'une implémentation directe de l'algorithme de Verlet.

```
def move(masses, positions, velocities, time):
    n = len(masses)
    assert n == len(positions)
    assert n == len(velocities)
    assert time >= 0
    steps = int(time/delta_t)
    forces = calc_forces(masses, positions)
    v_midstep = n*[None]
    for k in range(steps):
        for i in range(n):
            v_midstep[i] = velocities[i] + 0.5*delta_t*forces[i]/masses[i]
            positions[i] = positions[i] + delta_t*v_midstep[i]
            spheres[i].pos = positions[i]
            curves[i].append(pos = positions[i])
        forces = calc_forces(masses, positions)
        for i in range(n):
            velocities[i] = v_midstep[i] + 0.5*delta_t*forces[i]/masses[i]

# Lancer la simulation pour un an
move(masses, positions, velocities, 365*24*h)
```