3.4 N-body Simulation

Goal. Determine the motion of N particles, moving under their mutual Newtonian gravitational forces.

Ex. Planets orbit the sun.

N-Body: Applications

Applications to astrophysics.
- Orbits of solar system bodies.
- Stellar dynamics at the galactic center.
- Stellar dynamics in a globular cluster.
- Stellar dynamics during the collision of two galaxies.
- Formation of structure in the universe.
- Dynamics of galaxies during cluster formation.

N-Body Problem

Goal. Determine the motion of N particles, moving under their mutual Newtonian gravitational forces.

Context. Newton formulated the physical principles in Principia.

\[ F = m a \]
\[ F = \frac{G m_1 m_2}{r^2} \]

Newton's second law of motion
Newton's law of universal gravitation

Galaxies NGC 3327 and IC 3165

Kepler Bernoulli Newton Euler Lagrange Delaunay Poincaré
2-Body Problem

2 body problem.
- Can be solved analytically via Kepler’s 3rd law.
- Bodies move around a common barycenter (center-of-mass) with elliptical orbits.

3-Body Problem

3-body problem. No solution possible in terms of elementary functions; moreover, orbits may not be stable or periodic!

Consequence. Must resort to computational methods.

N-Body Simulation

N-body simulation. The ultimate object-oriented program: simulate the universe.

Body Data Type

Body data type. Represent a particle.

```
public class Body
{
    private Vector r;  // position
    private Vector v;  // velocity
    private double mass;  // mass
    // instance variables

    Body(Vector r, Vector v, double mass)
    void move(Vector f, double dt)
    void draw()
    Vector forceFrom(Body b)
}
```

Vector notation. Represent position, velocity, and force using vector.

```
public class Body {
    private Vector r;  // position
    private Vector v;  // velocity
    private double mass;  // mass
    // instance variables
```
Moving a Body

Moving a body. Assuming no other forces, body moves in straight line.

Moving a Body

\[ r = r.\text{plus}(v.\text{times}(dt)); \]
\[ r_x = r_x + dt \cdot v_x \]
\[ r_y = r_y + dt \cdot v_y \]

Force Between Two Bodies

Newton’s law of universal gravitation.
- \( F = G \frac{m_1 m_2}{r^2} \).
- Direction of force is line between two particles.

Body Data Type: Java Implementation

```
public class Body {
    private Vector r; // position
    private Vector v; // velocity
    private double mass; // mass

    public Body(Vector r, Vector v, double mass) {
        this.r = r;
        this.v = v;
        this.mass = mass;
    }

    public void move(Vector f, double dt) {
        Vector a = f.times(1/mass);
        v = v.plus(a.times(dt));
        r = r.plus(v.times(dt));
    }

    public Vector forceFrom(Body that) {
        double G = 6.67e-11;
        Vector delta = a.r.minus(b.r);
        double dist = delta.magnitude();
        double F = (G * a.mass * b.mass) / (dist * dist);
        Vector force = delta.direction().times(F);
    }

    public void draw() {
        StdDraw.setPenRadius(0.025);
        StdDraw.point(r.cartesian(0), r.cartesian(1));
    }
}
```
Universe Data Type

Universe data type. Represent a universe of N particles.

```java
public class Universe {
    private double radius; // radius of universe
    private int N; // number of particles
    private Body[] orbs; // the bodies

    constructor
    public Universe() {
        N = StdIn.readInt();
        radius = StdIn.readDouble();
        StdDraw.setXscale(-radius, +radius);
        StdDraw.setYscale(-radius, +radius);
        // read in the N bodies
        orbs = new Body[N];
        for (int i = 0; i < N; i++) {
            double rx   = StdIn.readDouble();
            double ry   = StdIn.readDouble();
            double vx   = StdIn.readDouble();
            double vy   = StdIn.readDouble();
            double mass = StdIn.readDouble();
            double[] position = { rx, ry }; // x, y
            double[] velocity = { vx, vy }; // vx, vy
            Vector r = new Vector(position);
            Vector v = new Vector(velocity);
            orbs[i] = new Body(r, v, mass);
        }
    }

    main simulation loop
    public static void main(String[] args) {
        Universe newton = new Universe();
        double dt = Double.parseDouble(args[0]);
        while (true) {
            StdDraw.clear();
            newton.increaseTime(dt);
            newton.draw();
            StdDraw.show(10);
        }
    }
}
```

Data-Driven Design

File format.

```
% more 4body.txt
4  N  velocity  mass
5.0e10  radius
-3.5e10 0.0e00 0.0e00 1.4e03 3.0e28
-1.0e10 0.0e00 0.0e00 1.4e04 3.0e28
1.0e10 0.0e00 0.0e00 -1.4e04 3.0e28
3.5e10 0.0e00 0.0e00 -1.4e03 3.0e28
```

```
Principle of Superposition

Principle of superposition. Net gravitational force acting on a body is the sum of the individual forces.

```math
\[ F_i = \sum_{j \neq i} \frac{G m_i m_j}{|r_i - r_j|^2} \]
```
public class Universe {
    private final double radius; // radius of universe
    private final int N; // number of bodies
    private final Body[] orbs; // array of N bodies

    public Universe() { /* see previous slide */ }

    public void increaseTime(double dt) {
        Vector[] f = new Vector[N];
        for (int i = 0; i < N; i++)
            f[i] = new Vector(new double[2]);
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                if (i != j)
                    f[i] = f[i].plus(orbs[j].forceTo(orbs[i]));
        for (int i = 0; i < N; i++)
            orbs[i].move(f[i], dt);
    }

    public void draw() {
        for (int i = 0; i < N; i++)
            orbs[i].draw();
    }

    public static void main(String[] args) { /* see previous slide */ }
}