N-Body Interactions

An (Almost) Trigonometry-Free Introduction

Nick Bennett

Position Vectors of Two Bodies

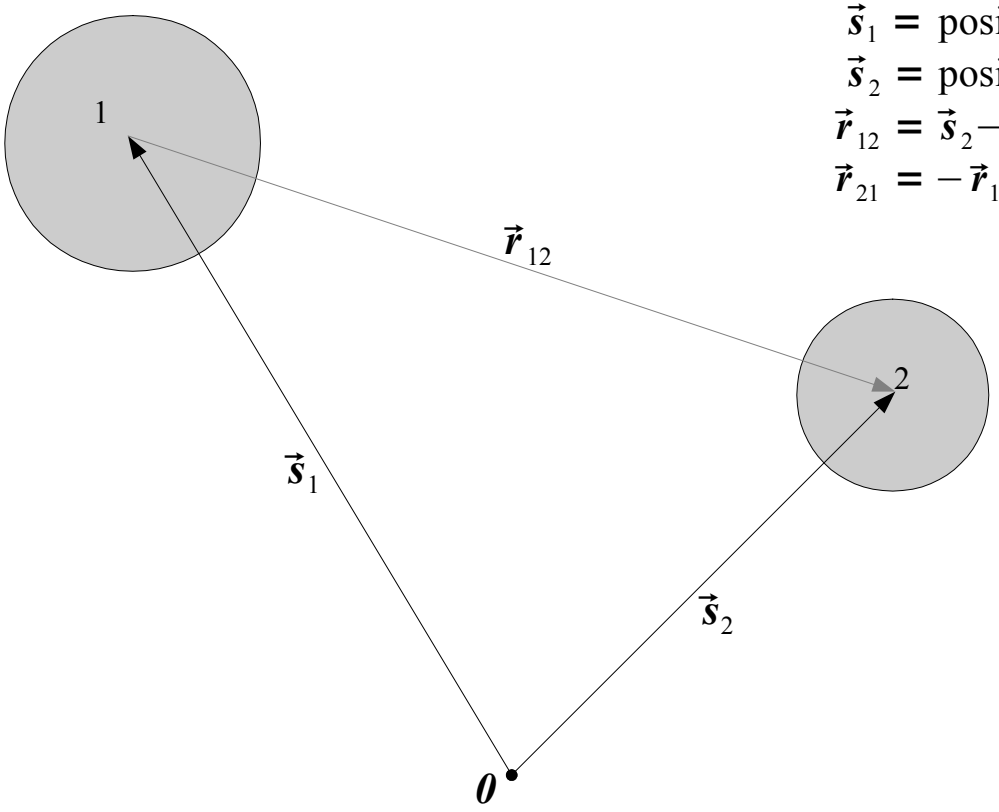
$\boldsymbol{0}$ = origin (not necessarily center of mass of the system; the mathematical result is independent of the choice of origin)

$\vec{\boldsymbol{s}}_1$ = position of body 1

$\vec{\boldsymbol{s}}_2$ = position of body 2

$\vec{\boldsymbol{r}}_{12} = \vec{\boldsymbol{s}}_2 - \vec{\boldsymbol{s}}_1$

$\vec{\boldsymbol{r}}_{21} = -\vec{\boldsymbol{r}}_{12}$



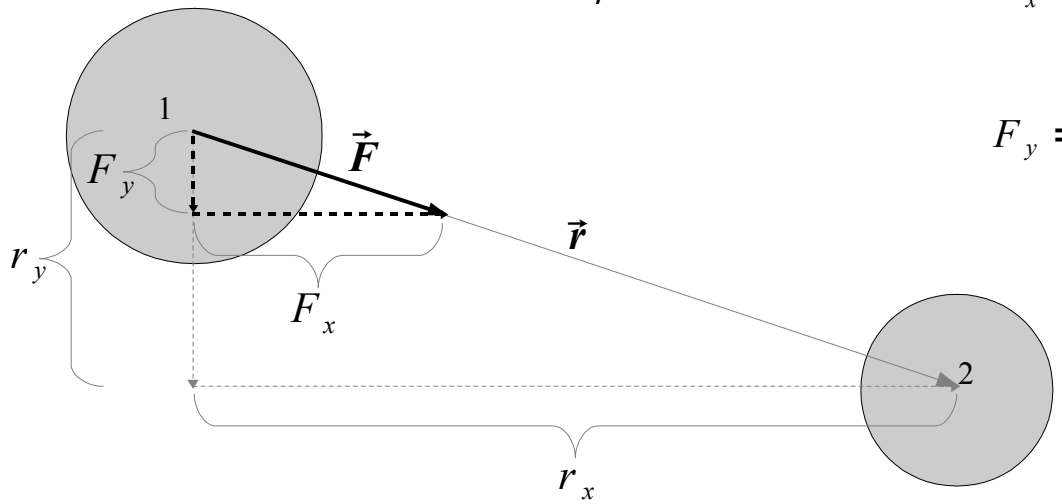
Distance and Force: Similar Triangles

$$\frac{F_x}{F} = \frac{r_x}{r}$$

$$F_x = F\frac{r_x}{r}$$

$$\frac{F_y}{F} = \frac{r_y}{r}$$

$$F_y = F\frac{r_y}{r}$$



$r = |\vec{r}|$ = distance between bodies 1 and 2

r_x = Distance between the bodies in the X direction

r_y = Distance between the bodies in the Y direction

$\vec{F}$ = Vector force (parallel to $\vec{r}$) acting on body 1, as a result of the gravitational attraction between body 1 and body 2

$F = |\vec{F}|$ = Magnitude of force vector

F_x = Magnitude of force component acting in the X direction, i.e. parallel to the Y axis

F_y = Magnitude of component of force acting in the Y direction, i.e. parallel to the X axis

Force Formulas

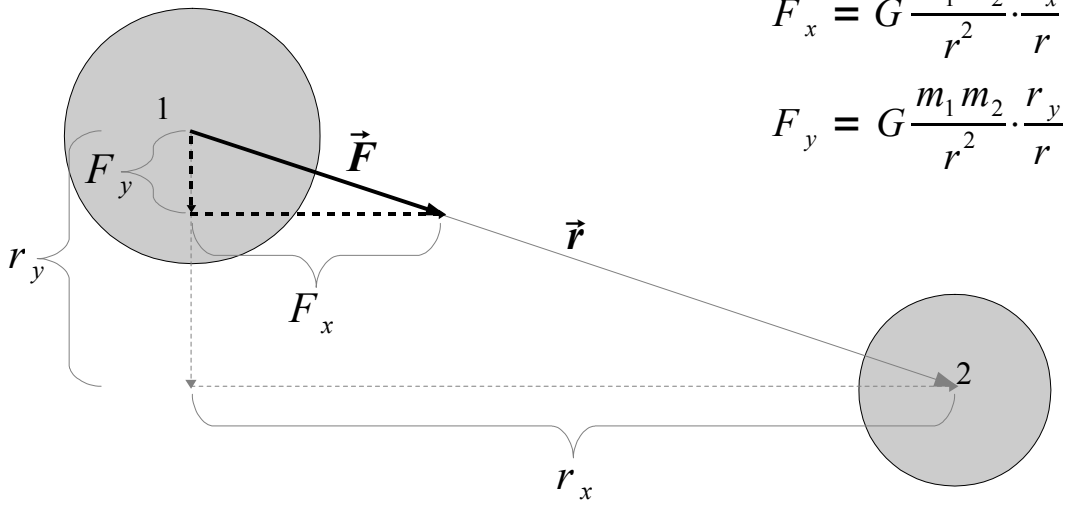
G = Universal gravitational constant

$$\vec{F} = G\frac{m_1 m_2}{r^2}\cdot\frac{\vec{r}}{r} = G\frac{m_1 m_2}{r^3}\vec{r}$$

$$F = \left|\vec{F}\right| = G\frac{m_1 m_2}{r^2}$$

$$F_x = G\frac{m_1 m_2}{r^2}\cdot\frac{r_x}{r} = G\frac{m_1 m_2}{r^3}r_x$$

$$F_y = G\frac{m_1 m_2}{r^2}\cdot\frac{r_y}{r} = G\frac{m_1 m_2}{r^3}r_y$$



Java Force Calculation Example

Assume we have the following variable and constant declarations in place. These might be found as class variables – in fact, body 1 and body 2 would probably be instances of the same class. (Note that the declaration of `G` is incomplete, since it is missing a value; a value appropriate for the units of measurement must be used. Also, `Point2D.Double` is a class in the `java.awt.geom` package, and an `import` statement would be required to use the class as seen in this example.)

```
static final double G = ...;            // Gravitational constant
double mass1;                           // Mass of body 1
double mass2;                           // Mass of body 2
Point2D.Double position1;               // Position of body 1
Point2D.Double position2;               // Position of body 2
```

Given the above variables, with appropriate values for a given moment in time, the following calculations would compute `force1` (the force acting on body 1) and `force2` (the force acting on body 2, with the same magnitude as – but opposite in direction to – `force1`), at that moment:

```
Point2D.Double displacement = new Point2D.Double(
    position2.x - position1.x, position2.y – position1.y);
double distance = Math.hypot(displacement.x, displacement.y);
double quotient = G * mass1 * mass2 / Math.pow(distance, 3);
Point2D.Double force1 = new Point2D.Double(
    displacement.x * quotient, displacement.y * quotient);
Point2D.Double force2 = new Point2D.Double(-force1.x, -force2.y);
```

Java Velocity & Position Calculation Example

Assume we have the following variable declarations in place. Again, these would probably be class variables, and an **`import`** statement would be required for this use of **`Point2D.Double`**.

```
double mass;                        // Mass of the body
Point2D.Double position;            // Position of the body
Point2D.Double velocity;            // Velocity of the body
```

Given the above, the following methods could be used to update the velocity (given a force applied over an interval of time) and position (given an interval of time). Note that these are approximations that use the Euler method, the most basic form of numerical integration; this method is usually not appropriate for use where a high level of accuracy is required. Also, if velocity and position are both updated in the same time step (as an alternative, some numerical integration techniques update these in a "leapfrog" fashion), most methods dictate that we update the velocities of all of the bodies, and then update all of their positions.

```
void updateVelocity(Point2D.Double force, double deltaTime) {
    velocity.x += (deltaTime * force.x / mass);
    velocity.y += (deltaTime * force.y / mass);
}

void updatePosition(double deltaTime) {
    position.x += (deltaTime * velocity.x);
    position.y += (deltaTime * velocity.y);
}
```