1 Wiskundige en fysische aanpak

1.1 programmeereenheden

In het SI-systeem zijn de fundamentele eenheden m, s en kg. De eenheid van de gravitationele constante G is van deze drie eenheden afgeleid. de eenheid van G kunnen we afleiden uit de versnelling van een m_1 door aantrekking tot een m_2:

$$\begin{aligned} m_1 a_1 = F = G\frac{m_1 m_2}{r^2} \\ \iff G = \frac{a_1 r^2}{m_2} \end{aligned} \tag{1}$$

Deze vergelijking klopt in gelijk welke eenheid. Evalueren we a_1, r en m_2 in SI-eenheden, dan wordt de SI-eenheid van G:

$$\begin{aligned} [G](SI) &= \frac{[a_1](SI) * [r^2](SI)}{[m_2](SI)} \\ &= \frac{[lengte^3](SI)}{[tijd^2](SI) \cdot [massa](SI)} \\ &= \frac{m^3}{s^2 kg} \end{aligned} \tag{2}$$

Dit geeft de eenheid van G. De tweede lijn kunnen we even goed in programmeerenheden (PU) opstellen:

$$[G](PU) = \frac{[lengte^3](PU)}{[tijd^2](PU) \cdot [massa](PU)} \tag{3}$$

Het verschil zit nu in de fundamentele eenheden van de programmeerenheden. We werken nu niet met (lengte, tijd, massa), maar met (lengte, massa, gravitationele constante). Tijd wordt dus een afgeleide eenheid! Vergelijking 3 geeft de eenheid van tijd in programmeerenheden, net zoals vergelijking 2 de eenheid van de gravitationele constante geeft in SI-eenheden:

$$[tijd](PU) = \sqrt{\frac{[lengte^3](PU)}{[G](PU) \cdot [massa](PU)}} \tag{4}$$

Wat we bedoelen met 'We kiezen $G = 1$' wilt dus zeggen dat we G kiezen als fundamentele eenheid in plaats van tijd.

De simulatie kan perfect lopen in de programmeereenheden; we gebruiken immers dezelfde wet van newton, enkel in een ander eenhedensysteem. Om de resultaten te interpreteren, moeten we wel een keuze maken wat we nu juist bedoelen met $[lengte](PU)$ en $[massa](PU)$.

1.2 berekening tijd- en snelheidseenheid

Als we een simulatie gerund hebben tot een bepaalde $tmax$, wil dat zeggen dat de simulatie in het echt $tmax$ maal de tijdseenheid in PU zou duren. Hiervoor moeten we deze $[tijd](PU)$ kunnen evalueren in eenheden die we kennen. Dit doen we door de volledige vergelijking 4 te evalueren in bv. het SI-stelsel:

$$\begin{aligned} [tijd](PU) \text{ in SI} &= \sqrt{\frac{[lengte^3](PU) \text{ in SI}}{[G](PU) \text{ in SI} \cdot [massa](PU) \text{ in SI}}} \\ [tijd](PU) \text{ in s} &= \sqrt{\frac{[lengte^3](PU) \text{ in m}^3}{[G](PU) \text{ in } \frac{m^3}{s^2 kg} \cdot [massa](PU) \text{ in kg}}} \end{aligned} \tag{5}$$

Wanneer we bijvoorbeeld de lengte-eenheid als 1 km nemen, en de massa-eenheid als 100 kg, dan zou de tijdseenheid $\sqrt{1000^3\ m^3/6.674 \cdot 10^{-11}\ \frac{m^3}{s^2 kg} \cdot 100\ kg} = 387 \cdot 10^6$ s zijn.
De programmeereenheid van snelheid staat hiermee in verband; deze is uiteraard $[lengte](PU)/[tijd](PU)$. In bovenstaand voorbeeld wordt dit $[snelheid](PU) = 1000\ m/387 \cdot 10^6\ s = 2.58 \cdot 10^{-6}$ m/s. De initiële snelheden in de inputfile worden geïnterpreteerd in deze eenheden!

1.3 gesimuleerd voorbeeld

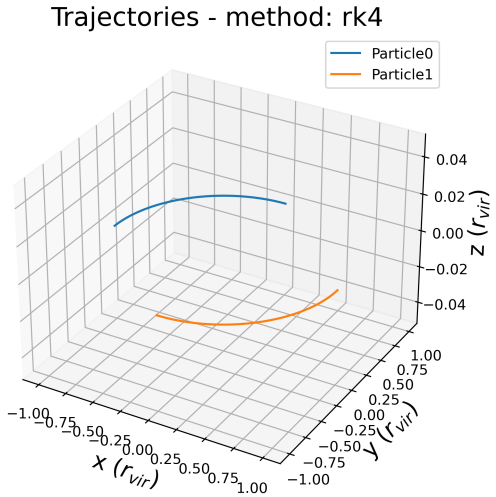
Om aan te tonen dat deze afleiding wel klopt, kunnen we een simulatie testen waarvan we weten wat het resultaat moet zijn; neem nu twee bollen met gelijke massa die een cirkelbaan rond elkaar beschrijven. We nemen bepaalde eenheden aan voor de massa en lengte, we berekenen hoeveel tijdseenheiden het zou duren om een kwart van een cirkelbaan af te leggen, en we kijken dan of de simulatie dat ook werkelijk toont.
Voor de cirkelstraal nemen we $r = 1$ km, voor de massa van de deeltjes nemen we $m = 100$ kg, en dit nemen we ook als lengte- en massa-eenheden. Dan weten we uit bovenstaand voorbeeld dat de tijd- en snelheidseenheden $387 \cdot 10^6$ s en $2.58 \cdot 10^{-6}$ m/s zijn. De snelheid van de deeltjes op de cirkelbaan halen we uit:

$$\begin{aligned} m\frac{v^2}{r} &= F = G\frac{m^2}{(2r)^2} \\ \iff v &= \sqrt{\frac{m\ G}{4\ r}} = 1.291 \cdot 10^{-6}\ \text{m/s} = 0.5005\ [\text{snelheid}](\text{PU}) \end{aligned} \tag{6}$$

Dit laatste getal wordt de snelheid in de inputfile. Om een kwart cirkelbaan te simuleren, moeten we de simulatie tot tmax $= 0.25 \cdot 2\pi \cdot r/v = 1216 \cdot 10^6$ s $= 3.1431$ [tijd](PU) laten lopen. Met onderstaande inputfile krijgen we dan:

```
1  2
2  0   1   0   -0.5005105259746108  0   0   1
3  0   -1  0   0.5005105259746108   0   0   1
```

(a) *inputfile*



(b) *traject*

De afstandsschaal op de figuur is nu ook de eenheid $[lengte](PU)$. Je kan met behulp van formule 5 nu de tijd- en snelheidseenheid voor gelijk welke afstand- en massa-eenheid berekenen. Het is niet nodig dat de totale massa gelijk aan 1 is in de inputfile, dat is in bovenstaand voorbeeld ook niet zo.

2 Vereenvoudigde aanpak

Er is ook een tweede manier om te garanderen dat de integratie in de correcte eenheden verloopt. Deze is eenvoudiger te implementeren, maar heeft een minder wiskundig en fysisch ondersteund principe.
Eerst moet de input getransformeerd worden naar de mks SI eenheden (meter, kilogram, seconde). Hiervan weten we dat het een consistent systeem is waarin de waarden van allerlei constanten welbekend zijn. Hierin is echter de gravitationele constante $G = 6.67\ 10^{-11} \frac{m^3}{kg\ s^2}$ terwijl de integratoren gebruik maken van $G = 1$. Wanneer we de stukken code bekijken waar G zou voorkomen indien die niet 1 zou zijn, valt er echter iets op.
Ten eerste zou G opduiken in het berekenen van de acceleraties. Merk op dat elke term in de totale acceleratievector voor het deeltje `i`, die hieronder berekend wordt, recht evenredig is met de massa van een deeltje `j`.

```
    for (int j=0; j<r.size(); ++j) {
            if (i!=j) {
                a -=  m[j] * (r[i] - r[j]) / (r[i] - r[j]).norm3();
            }
        }
```

Vervolgens dient G ook te verschijnen in het berekenen van de potentiële gravitationele energie, die adhv de code hieronder berekend wordt. Ook de computatie van de kinetische energie is gegeven. Merk op dat hierin de potentiële energie evenredig is met het kwadraat van de massa's en de kinetische energie lineair met de massa schaalt. Aangezien we enkel de energie berekenen op de relatieve energiefout te vinden, is het irrelevant als beide energieën met een constante vermenigvuldigd worden.

```
double potentialEnergy(const vector<double>& m, const vector<Vec>& x) {
    double sum = 0;
    for (int i=0; i<m.size(); ++i) {
        for (int j=0; j<m.size(); ++j) {
            if (i!=j) {
            sum += m[i]*m[j] / (x[i] - x[j]).norm();
            }
        }
    }
    return -sum/2.;
}

double kineticEnergy(const vector<double>& m, const vector<Vec>& v) {
    double sum = 0;
    for (int i=0; i<m.size(); ++i) {
        sum += m[i] * v[i].norm2();
    }
    return sum/2.;
}
```

Hierdoor is het mogelijk om, na de input in SI eenheden om te zetten, alle massa's met $G = 6.67\ 10^{-11}\frac{m^3}{kg\ s^2}$ te vermenigvuldigen. Hierdoor zou de acceleratie berekend worden aan de hand van de gekende zwaartekrachtwet met voorfactor G. Ook zowel de kinetische als potentiële energie correct zullen berekend zijn, hoewel ook vermenigvuldigd met G, maar die constante valt weg bij het bepalen van de relative energiefout. De output van de integratie -massa's, tijd, totale energie en positiecomponenten- zal dan eveneens in SI eenheden gegeven zijn, met uitzondering van de massa's en energie die eerst door $6.67\ 10^{-11}\frac{m^3}{kg\ s^2}$ gedeeld dienen te worden.
De aandachtige lezer heeft misschien opgemerkt dat deze methodiek ook op andere systemen dan het mks SI systeem, zolang de dan correcte waarde voor G ingegeven wordt. In dat geval zal de output gegeven zijn in de door de gebruiker gekozen eenheden nadat massa's en energieën door G gedeeld werden.