VÕRGU KASUTAJALIIDESE LOOMINE FÜÜSIKALISTE ÜLESANNETE ARVUTAMISEKS JA VISUALISEERIMISEKS

Bakalaureusetöö

Üliõpilane: Artur Raag
Juhendaja: Mihhail Klopov, Küberneetika instituut, dotsent
Õppekava: Rakendusfüüsika

Tallinn 2023

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Artur Raag
[allkiri ja kuupäev]

Töö vastab bakalaureusetööle/magistritööle esitatavatele nõuetele.
Juhendaja: Mihhail Klopov
[allkiri ja kuupäev]

Töö on lubatud kaitsmisele.
Kaitsmiskomisjoni esimees: [nimi]
[allkiri ja kuupäev]

Sisukord

1 Annotatsioon

Antud töö eesmärgiks oli luua veebivõrgus kasutatavad kasutajaliidesed kolme erineva füüsika nähtuse arvutamiseks, visualiseerimiseks ja interakteerimiseks. Et tagada võimalikult täpne visualisatsioon füüsika nähtustest, vaadati üle vastav füüsika õppekirjandus, mille alusel programmid ka loodi. Programmid kirjutati kasutades programmeerimiskeelt JavaScript, kuna see oli kõige loomulikum valik veebiarenduse jaoks. Lisaks kasutati töös visualiseerimise ja kasutajaliidese loomise hõlbustamiseks P5JS teeki. Visualisatsioonide arvutustäpsuse hindamisel hinnati süsteemi koguenergia muutumist pikkema ajavahemiku puhul. Kahe programmi puhul oli koguenergia muutuste suurusjärk nulli lähedane, kuid kolmanda programmi puhul esines koguenergias ebastabiilsusi, mis võis olla tingitud programmikoodis esinevatest ümardamisvigadest ning algoritmi valemite ebatäpsusest. Samuti võib esineda programmides väiksemaid graafilisi tõrkeid, kuid need tõrked ei takista kasutajaid programme veebis kasutamast ega nendega interakteerimast. Ebastabiilsuste ning lihtsuse tõttu ei kõlba hetkel programmid teaduslike arvutuste jaoks, kuid on piisavalt täpsed klassi- või loenguruumides kasutamiseks nähtuste demonstreerimiseks.

2 Abstract

The aim of this work was to create cross platform web-based user interfaces for calculating, visualizing and interacting with three different physics phenomena. To ensure the most accurate visualization of given physics phenomena, the corresponding physics educational literature was reviewed, on the basis of which the programs were also created. The program scripts were written using the JavaScript programming language, as it was the most obvious choice for web development. In addition, the P5JS library was used to facilitate visualization and user interface creation. When evaluating the calculation accuracy of the visualizations, the change in the total energy of the system was evaluated over a longer period of time. For two of the programs, the magnitude of the total energy changes was negligible, however for the third program there were fluctuations in the total energy calculations, which may have been due to faulty formulas in the program code. Users may also experience minor graphical glitches in these programs, but these glitches have minimal impact on the user experience when interacting with them online. Due to the total energy fluctuations, the programs in their current state are not suitable for scientific calculations, however, are accurate enough to provide insight into the observable physics phenomena which can justify their use in classrooms or lectures for demonstration purposes.

3 Sissejuhatus

Erinevate digilahenduste välja arendamine ning kasutamine koolides on praeguses tehnoloogiliselt kiiresti arenevas ühiskonnas olnud vaieldamatu vajadus. Hiljutise kriisiolukorrast tekitatud distantsõppe tagajärjel selgus, et vajadus kvaliteetsele digitaalse õppevara järel on aga veelgi suurem kui kunagi varem [1]. Antud probleemi peetakse riiklikul tasemel niivõrd oluliseks, et selle tagajärjel otsustas Eesti riik 2021. aastal toetada digiõppevara arendamist ja kasutuselevõtmist 1.75 miljon euroga [1]. Meie töö motivatsiooniks ongi panustada uute füüsika digi õppevahendite välja arendamisele.

Selle töö üheks eesmärgiks on luua põhiliste füüsikaülesannete jaoks numbrilised mudelid, mis võimaldavad füüsilist nähtust võimalikult täpselt jäljendada. Oma numbrilistes mudelites määrame ära kehade arvu, keha parameetrid ning kehadele mõjuvad jõud. Kehadele mõjuvad jõud aitavad meil luua keha liikumise kirjeldamiseks liikumisvõrrandit, mida lahendame oma programmis iteratiivseid meetodeid kasutades. Iga ajasammu (iteratsiooni) tagant arvutame süsteemis olevatele kehadele mõjuvad jõud, mille tulemusel saame määrata keha järgneva asukoha järgmises iteratsioonis. Uute keha koordinaatidega arvutame nendele mõjuvad jõud uuesti ja määrame taaskord järgmises iteratsioonis uued koordinaadi. Kirjeldatud protsess töötab tsüklis lõputult, või kuni mõne tingimuse täitumiseni.

Liikumisvõrrandite iteratiivse arvutamise puhul kasutame algoritme, mis on arvutamistäpsuse poolest võimalikult täpsed, kuid kui kehade arv ületab teatud lävendi, siis võib iga iteratsioonil tehtavate arvutuste kiirus märkimisväärselt kahaneda. Seega üheks motivatsiooniks on ohverdada mitme keha füüsika ülesande modelleerimisel täpsust, arvutamiskiiruse hõlbustamiseks. Numbrilisi mudeleid, mida kasutame füüsikaülesannete lahendamiseks, jaotame selle tõttu kaheks. Esimene neist on piiratud arvu kehadega ning teine n-kehadega. Mudeli arvutamistäpsuse hindamiseks, arvutame iga iteratsiooni tagant lisaks kehadele mõjuvatele jõududele ka süsteemi koguenergiat. Kui koguenergia väärtus isoleeritud süsteemis, kus puudub hõõrdumine, ei muutu, või muutub aeglaselt, siis võime järeldada, et süsteemis arvutatud vead on minimaalsed ja seega on arvutamistäpsus piisav. Vastasel juhul, kui koguenergia väärtus kasvab või kahaneb liiga kiiresti, siis on arvutamisvead liiga suured ja järeldame, et algoritmi arvutamistäpsus on kehvem.

Bakalaureusetöö teiseks eesmärgiks on luua kasutajaliides, mis on veebivõrgust lihtsasti ligipääsetav ja kasutajasõbralik. Kasutajaliides peab suutma füüsilist nähtust võimalikult täpselt visualiseerida ning võimaldama kasutajal süsteemi või keha parameetreid muuta. Kuna kasutajaliideste kasutajad ei pruugi olla füüsika haridusliku taustaga, siis oleks mõistlik sõltuvalt nähtusest initsialiseerida sellised algtingimused, mille korral nähtus töötaks nii nagu autor ette näeb. Seejärel jääb kasutajal vabadus proovida erinevaid algtingimusi, mille seas võivad olla ka algtingimused, mille korral nähtus ei tööta nii nagu autor seda ette nägi. Antud otsus tagab, et kasutaja ei kaota esimeste tingimuste sisestamise korral huvi ega motivatsiooni, vaid proovib initsialiseeritud algtingimusi hoopis varieerida. Loodud programmid laetakse kasutamiseks Tallinna Tehnika Ülikooli Parseki serverile.

4 Teoreetilised alused

4.1 Harmoonilised võnkumised

Võtame vaatluse alla süsteemi, kus meil on fikseeritud vedru elastsuskoefitsendiga *k*, ning vedru otsas on keha massiga *m*. Vedru otsas tasakaaluasendis olevale kuulikesele massiga *m* mõjub vaid üks väline jõud, näiteks raskusjõud mg. Tasakaalu oleku säilitamiseks mõjub seega vedrule ka elastsusjõud $k\Delta l_0$, ehk

$$mg = k\Delta l_0. \qquad (4.1)$$

Kui vedru otsas olevat kuulikest nihutatakse tasakaaluolekust *x* võrra kõrvale, siis pikeneb vedru $\Delta l_0 + x$ võrra, mille tagajärjel meie *x*-teljele langev resultantjõu projektsioon omandab kuju

$$F = mg - k(\Delta l_0 + x) \qquad (4.2)$$

Arvestades valemit (4.1), saame

$$F = -kx. \qquad (4.3)$$

Keha nihutamiseks tuleb teha tööd

$$A = \frac{kx^2}{2}, \qquad (4.4)$$

mis saab ka meie potentsiaalseks energiaks

$$E_p = \frac{kx^2}{2}. \qquad (4.5)$$

Kineetilist energiat arvutame aga

$$E_k = \frac{mv^2}{2}. \qquad (4.6)$$

Meie töös on vaja luua keha liikumise kirjeldamiseks liikumisvõrrand. Kirjutades võrrand (4.3) ümber Newtoni teise seaduse jaoks ning jagades mõlemad võrduse pooled keha massiga m läbi, siis saame

$$\ddot{x} + \frac{k}{m}x = 0, \qquad (4.7)$$

mis on teistjärku lineaarne homogeenne konstantsete kordajatega diferentsiaalvõrrand. Kuna meie oma mudelites kasutame diferentsiaalvõrrandite integreerimiseks neljandat järku Runge-Kutta (edaspidi RK4) algoritmi, mis suudab integreerida vaid esimest järku võrrandeid, siis tuleb meil konstrueerida esimest järku võrrandite süsteem. Asendades $\frac{k}{m} = \omega^2$ ning $\dot{x} = v_x$, konstrueerime kaks esimestjärku diferentsiaalvõrrandit, mida lahendame numbriliselt RK4 meetodiga [2].

$$\begin{cases} \dot{x} = v_x \\ \dot{v_x} = -\omega^2 \cdot x \end{cases} \qquad (4.8)$$

Võrrandite numbriliseks lahendamiseks on vaja ka määrata algtingimused *x* ja v_x. Need määrame ära oma programmi alguses, aga sisuliselt võivad nad olla mistahes suurused.

Samuti saab kasutada harmooniliste võnkumiste kontrollimiseks analüütilist valemit, mis näeb välja kujul

$$x(t) = x_0 cos(\omega_0 \cdot t + \phi_0), \qquad (4.9)$$

kus x_0 on võnke amplituud, ω_0 võnkumise ringsagedus ning ϕ_0 on algfaas.

4.2 Sumbuvad võnkumised

Sumbuvad võnkumised erinevad vabavõngetest selle poolest, et neile mõjub veel väline takistusjõud, mille tagajärjel võnkuv keha kaotab energiat. Väikeste kiiruste puhul on takistusjõud võrdeline keha kiirusega.

$$\vec{F}_{takistus} = -\alpha\vec{v}, \qquad (4.10)$$

kus α on välise takistusjõu tegur. Liikumisvõrrandi saame analoogselt nagu vabavõngete puhul, ehk pannes Newtoni teise seaduse võrduma kõikide süsteemis mõjuvate jõududega. Süsteemis mõjub lisaks takistusjõule ka eelnevalt mainitud elastsusjõud.

$$m\ddot{x} = -kx - \alpha\dot{x}. \qquad (4.11)$$

Jagades mõlemad võrduse pooled massiga *m*, ning viies kõik liikmed vasakule poole, saame

$$\ddot{x} + \frac{\alpha}{m}\dot{x} + \frac{k}{m}x = 0 \qquad (4.12)$$

Asendades veel

$$\beta = \frac{\alpha}{2m} \qquad (4.13)$$

$$\omega = \sqrt{\frac{k}{m}} \qquad (4.14)$$

saame viimaks [3] teistjärku lineaarse homogeense konstantsete kordajatega diferentsiaalvõrrandi kujul

$$\ddot{x} + 2\beta\dot{x} + \omega^2 x = 0. \qquad (4.15)$$

Sellest piisab, et moodustada esimestjärku diferentsiaalvõrrandite süsteem, mis kirjeldaks meie süsteemi liikumist.

$$\begin{cases} \dot{x} = v_x \\ \dot{v_x} = -2\beta\dot{x} - \omega^2 x \end{cases} \qquad (4.16)$$

Nagu ka eelnevalt, v_x ning *x* on algtingimused, mis määrame oma programmi alguses, kuid võivad olla mistahes suurused.

Analüütiline valem sumbuvate võnkumiste jaoks avaldub aga kujul

$$x(t) = x_0 \cdot e^{-\beta \cdot t} \cdot cos(\omega \cdot t + \phi_0), \qquad (4.17)$$

kus $\omega = \sqrt{\omega_0^2 - \beta^2}$, $\omega_0 = \frac{k}{m}$ ning ϕ_0 on algfaas.

Kõikide eelnevalt mainitud mudelite puhul tuleb kasuks ka koguenergia arvutamine, näiteks mudeli arvutustäpsuse hindamisel. Kineetilist ning potentsiaalset energiat saab kõikide ülesannete puhul arvutada valemitega (4.5) ja (4.6). Nende summa annab meile aga süsteemi koguenergia [2]

$$E_{total} = E_{kin} + E_{pot}. \qquad (4.18)$$

4.3 Kepleri ülesanne

4.3.1 Liikumisvõrrand

Et modelleerida keha liikumist ümber valitud taevakeha tuleb meil leida kuidas keha koordinaadid $x(t), y(t)$, kiirusvektorid $v_x(t), v_y(t)$ sõltuvad ajast ning millised on keha kineetiline, potentsiaalne ning koguenergia erinevatel ajahetkedel.

Kui me eeldame, et satelliit kehale mõjub vaid gravitatsiooni jõud, siis saame Newtoni teise seaduse kaudu avaldada liikumisvõrrandi nagu eelnevaltki.

$$F_g = -G\frac{m \cdot M}{r^3}\vec{r} \qquad (4.19)$$

Siinkohal mainime ka, et potentsiaalne energia [4] avaldub valemiga

$$E_{pot} = -G\frac{m \cdot M}{r} \qquad (4.20)$$

Antud valemites on *G* gravitatsiooni konstant, *M* on taevakeha mass, *m* satelliidi mass ning *r* on kahe keha massikeskmete vaheline kaugus. Ehk hetkel eeldame, et taevakeha massikese asub taevakeha tsentris. Viimast suurust saab ka avaldada kujul $r = R_M + h$, kus R_M on keha keskmine raadius, ning h on satelliidi kõrgus maapinna suhtes.

Asendades F_g Newtoni II seadusesse ning jagades mõlemad võrduse pooled satelliidi massiga *m*, saame

$$\vec{a} = -G\frac{M}{r^3} \cdot \vec{r}. \qquad (4.21)$$

Kuna edaspidiseid arvutusi on märkimisväärselt lihtsam teha *x* ja *y* projektsioonidena, siis jaotame vektoriaalsed suurused projektsioonideks. Asukoha teist järku tuletis annab meile kiirenduse ning esimest järku tuletis kiiruse. Seega

$$\begin{cases} \ddot{x} = -G\frac{M}{r^3} \cdot x \\ \ddot{y} = -G\frac{M}{r^3} \cdot y \end{cases}, \qquad (4.22)$$

kusjuures $r = \sqrt{x^2 + y^2}$. Kuna meie programmis kasutatud algoritm lahendab vaid esimest järku diferentsiaalvõrrandeid, siis teisendame teistjärku võrrandid esimest järku võrranditeks, võttes $\dot{x} = v_x$ ning $\dot{y} = v_y$, saame uue esimest järku nelja võrrandisüsteemi.

$$\begin{cases} \dot{x} = v_x \\ \dot{y} = v_y \\ \dot{v_x} = -G\dfrac{M}{r^3} \cdot x \\ \dot{v_y} = -G\dfrac{M}{r^3} \cdot y \end{cases} \quad (4.23)$$

Diferentsiaalvõrrandite süsteemi lahendamiseks kasutame RK4 algoritmi. Edukaks lahendamiseks on aga vaja määrata ka algtingimused x, y, v_x ning v_y, kuid seekord algtingimuste määramine nii meelevaldne ei ole.

Eeldusel, et meie süsteemi koordinaatteljestik fikseeritakse ligikaudu taevakeha keskpunktis, siis ei saa meie satelliidi kaugus taevakeha tsentrist olla väiksem kui taevakeha raadius ise. Lisaks, kuna meie mudelis eeldame, et satelliit alustab enda liikumist juba taevakeha orbiidist, siis peaks satelliidi kauguse moodul olema suurem või võrdne taevakeha raadiuse ja orbiidi kõrguse mooduli summaga.

Samuti on tarvis leida sobiv algne kiirus, sest kui algkiirus on liiga väike, langeb satelliit taevakeha pinnale. Liiga suurt kiirust ei saa ka valida, kuna sellisel juhul lendab satelliit hoopis taevakeha mõjusfäärist/orbiidist välja. Seega algtingimused arvutame selles ülesandes ette ära, et tagada mudeli töökindlust.

4.3.2 Algtingimuste määramine

Nagu eelnevalt mainitud, satelliidi kaugus taevakeha keskpunktist peaks olema suurem või võrdne taevakeha raadiuse R_M ja oribiidi kõrgusega taevakeha pinnast h. Kui fikseerida koordinaatteljestik taevakeha keskpunkti, siis võiks lihtsuse nimel määrata satelliidi algkoordinaatideks

$$\begin{cases} x = R_M + h \\ y = 0 \end{cases}. \quad (4.24)$$

Leiame ka algkiiruse, mille puhul satelliidi orbiit säilitaks ringjoonelise trajektoori, ehk esimese kosmilise kiiruse [2]. Kui eeldada, et keha liigub ringjoonelist mööda taevakeha pinda, siis saaksime keha liikumise kirjeldamiseks järgmise liikumisvõrrandi

$$m \cdot a = F = m \cdot g, \quad (4.25)$$

kus g on raskuskiirendus taevakeha pinna lähedal. Keha, mille omapäraks on ringjooneline liikumine, saab kirjeldada ka tsentrifugaaljõu abil

$$F_c = \frac{mv^2}{R_M}. \quad (4.26)$$

Asetades tsentrifugaaljõudu Newtoni II seadusesse, saame avaldada keha vähima kiiruse, mis on tarvis ringjooneliseks orbiidiks taevakeha pinnal.

$$\frac{mv^2}{R_M} = mg \rightarrow v = \sqrt{gR_M} \quad (4.27)$$

Kuid ideaalis on orbiidis olev keha ikkagi taevakehade pinnast veidi eemal, seega tuleb meil viimast valemit veidi täiendada. Olgu seekord liikumisvõrrand sõltuv satelliidi kõrgusest taevakeha pinnast.

$$ma = -G\frac{m \cdot M_M}{r^2}, \qquad (4.28)$$

kusjuures $r = R_M + h$. Tsentrifugaaljõudu tuleb samuti üldistada, kuna see peab samuti kõrgusest sõltuma.

$$F_C = \frac{mv^2}{(R_M + h)} \qquad (4.29)$$

Ringjoonelisel orbiidil peab tsentrifugaaljõud võrduma gravitatsioonijõuga, mis kehale mõjub [5]. Seega paneme kaks suurust üksteisega võrduma ja avaldame kiiruse v.

$$G\frac{m \cdot M_M}{(R_M + h)^2} = \frac{mv^2}{(R_M + h)} \qquad (4.30)$$

$$v_I = \sqrt{\frac{GM_M}{(R_M + h)}} \qquad (4.31)$$

Viimase valemi alusel saame väita, et kui keha kõrgusel h taevakeha pinnast liigub aeglasemini kui kiirusega v_I, siis langeb satelliit tagasi vaadeldava taevakeha pinnale. Seega, et meie mudelis satelliit liikuma panna, peaks algkiirus olema suurem või võrdne esimese kosmilise kiirusega.
Selles töös määrasime satelliidi algkoordinaatideks $x = R_M + h$ ning $y = 0$. Seega satelliidi kiiruse võiks määrata ära nii, et see oleks taevakeha suhtes puutujasihiline. Ehk antud algkoordinaatide puhul

$$\begin{cases} v_x = 0 \\ v_y = v_I \end{cases}. \qquad (4.32)$$

Samuti sobiks ka $v_y = -v_I$, kuna see oleks samuti tangentsiaalne, kuid lihtsalt sihiga teises suunas.

Üleliigselt suure kiiruse puhul võib samuti soovimatu tulemuseni jõuda. Nimelt, kui kiirus on liiga suur, siis lendab satelliit taevakeha orbiidist/mõjusfäärist välja ja tagasi enam ei naase [2]. Sellist kiirust nimetatakse teiseks kosmiliseks kiiruseks. Ka selle kiiruse määrame enda töös ära, kuna see aitab meil tagada mugavamat liidese kasutamist.

Vähim kiirus, mille korral keha taevakeha mõjusfäärist välja lendab, saame avaldada energiajäävuse seaduse kaudu

$$E_{tot} = E_{pot} + E_{kin}. \qquad (4.33)$$

Siin E_{tot}, E_{pot} ja E_{kin} on koguenergia, potentsiaalne energia ning kineetiline energia. Kahe keha vahelist potentsiaalset energia saab arvutada valemiga

$$E_{pot} = -G\frac{m \cdot M}{R} \qquad (4.34)$$

ning kineetilist energiat valemiga

$$E_{kin} = \frac{mv^2}{2} \qquad (4.35)$$

Seega keha koguenergia taevakeha pinnalt õhku tõustes on

$$E_0 = -\frac{G \cdot m \cdot M}{R} + \frac{m v_{II}^2}{2}, \qquad (4.36)$$

kus $R = R_m + h$, $R_m$on taevakeha raadius ning h on satelliidi või muu keha kõrgus taevakeha maapinnast. Samuti on v_{II} otsitav teine kosmiline kiirus. Lõpmata kaugel peaks keha koguenergia võrduma nulliga [5], ehk $E_1 = 0$. Energiajäävuse seaduse tõttu, peab energia alghetkel olema sama, mis viimasel hetkel, ehk võrdsed. Seega

$$E_0 = E_1 \qquad (4.37)$$

$$-\frac{G \cdot m \cdot M}{R} + \frac{m v_{II}^2}{2} = 0. \qquad (4.38)$$

Kui avaldada teine kosmiline kiirus v_{II}, siis saame

$$v_{II} = \sqrt{\frac{2G \cdot M}{R}} \qquad (4.39)$$

Seega, et meie satelliit edukalt ümber vaadeldava taevakeha ringleks, võiksid kiirused olla vahemikus v_I ja v_{II}.

4.3.3 Ühikute teisendamine

Lähtudes näiteks eelmises peatükist toodud esimesest ja teisest kosmilisest kiirusest, võime märgata, et kiiruste suurusjärgud selles mudelis on SI süsteemi järgi liiga suured. See võib liidese loomist, kasutamist ja andmete lugemist ebamugavaks teha. Seega teisendame kiirused meeter/sekundilt hoopis kilomeeter/sekundiks.

Kui kiiruse ühikuks on m/s, siis selle teisendamisel km/s-ks tuleb meil kiirust laiendada 1000-ga. Suuruse laiendamine meil tegelikult midagi ei muuda, kuna 1000-ed taanduksid ära. Kuid põhjus, miks me seda teeme, on sellepärast, et kiirus, mis on esialgu m/s, muutuks 1000-ga läbi jagades km/s-ks, mis oleks omakorda veel läbi korrutatud 1000-ga. Ehk teisisõnu, kiiruste ja kauguste teisendamisel meetrilt kilomeetrile, tuleb meil suurus lihtsalt 1000-ga läbi korrutada.
Demonstreerides seda matemaatiliselt

$$v_x\left[\frac{m}{s}\right] = \frac{1000 \cdot v_x[m/s]}{1000} = 1000 \cdot v_x\left[\frac{km}{s}\right]. \qquad (4.40)$$

Seega võrrandite

$$\begin{cases} \dot{x} = v_x \\ \dot{y} = v_y \end{cases} \qquad (4.41)$$

puhul tuleb meil mõlemad võrduse pooled korrutada 1000-ga. Saame

$$\begin{cases} 1000 \cdot \dot{x} = 1000 \cdot v_x \\ 1000 \cdot \dot{y} = 1000 \cdot v_y \end{cases}. \qquad (4.42)$$

Mõlemalt võrduse poolelt taanduvad 1000-d ära, mis tähendab, et võime vabalt esimesed kaks võrrandit teisendada meeter/sekundilt kilomeeter/sekundiks kasutamata igasuguseid koefitsiente. Võrrandite

$$\begin{cases} \dot{v_x} = -G\dfrac{M}{r^3}\cdot x \\ \dot{v_y} = -G\dfrac{M}{r^3}\cdot y \end{cases} \qquad (4.43)$$

puhul analoogselt käitudes saame

$$\begin{cases} 1000\cdot\dot{v_x} = -G\dfrac{M}{(10^3\cdot r)^3}\cdot 1000\cdot x \\ 1000\cdot\dot{v_y} = -G\dfrac{M}{(10^3\cdot r)^3}\cdot 1000\cdot y \end{cases} \rightarrow \begin{cases} \dot{v_x} = -G\dfrac{M\cdot 10^{-9}}{r^3}\cdot x \\ \dot{v_y} = -G\dfrac{M\cdot 10^{-9}}{r^3}\cdot y \end{cases}. \qquad (4.44)$$

Ehk kiirenduste puhul tuleb meil eduka teisenduse jaoks korrutada võrrandite paremad pooled koefitsiendiga 10^{-9}.

4.4 Runge-Kutta-Fehlberg meetod

Kuna meie harmooniliste võngete ning Kepleri ülesande mudelites on liikumisvõrrandite arv suhtelist väike, siis saame liikumisvõrrandite arvutamiseks kasutada enimlevinud Runge-Kutta Fehlbergi (edaspidi RKF45) meetodit [6]. RKF45 meetod on oma iseloomu poolest arvutuste rohke, kuid piisavalt täpne. Samuti omab algoritm omadust enda integreerimissammu kohandada juhul kui arvutusviga liiga suureks või väikseks osutub.

Võrreldakse neljandat ja viiendat järku lahendeid [7]. Kui nende vahe on liiga suur või liiga väike, siis kohandatakse integreerimis sammu ja arvutatakse kogu algoritm uue ajasammuga läbi. Kui lahendite vahe on aga sobiv, siis võetakse lähendatud tulemus vastu. Ajasammu kohandamine nõuaks algoritmi korduvat arvutamist, mis ei ole hea kombinatsioon väikese ega suure ajasammuga, kuna see aeglustaks mudeli reaalajas töötamist . Seega oma programmis jätame selle operatsiooni välja. Lisaks on RKF45 omaduseks $k_1 - k_6$ kordajate arvutamisel kasutada varieeruvat ajasammu, kuid kuna meie liikumisvõrrandites sõltuvus ajast puudub, siis ei mõjuta varieeruv ajasamm meie tulemust kuidagi. Varieeruv ajasamm on aga RKF45 algoritmi oluline omadus ja kuna meie oma programmis jätame seda rakendamata, siis taandub algoritm hoopis neljandat järku Runge-Kutta (edaspidi RK4) meetodile.

Algoritmi edukaks kasutamiseks tuleb arvutada järgmist kuus erinevat kordajat [8]

$$\begin{aligned} k_1 &= h\cdot f(x + A(1)\cdot h, y) \\ k_2 &= h\cdot f(x + A(2)\cdot h, y + B(2,1)\cdot k_1) \\ k_3 &= h\cdot f(x + A(3)\cdot h, y + B(3,1)\cdot k_1 + B(3,2)\cdot k_2) \\ k_4 &= h\cdot f(x + A(4)\cdot h, y + B(4,1)\cdot k_1 + B(4,2)\cdot k_2 + B(4,3)\cdot k_3) \\ k_5 &= h\cdot f(x + A(5)\cdot h, y + B(5,1)\cdot k_1 + B(5,2)\cdot k_2 + B(5,3)\cdot k_3 + B(5,4)\cdot k_4) \\ k_6 &= h\cdot f(x + A(6)\cdot h, y + B(6,1)\cdot k_1 + B(6,2)\cdot k_2 + B(6,3)\cdot k_3 + B(6,4)\cdot k_4 + B(6,5)\cdot k_5) \end{aligned} \qquad (4.45)$$

kus $A(i,j)$ ja $B(i,j)$ on Fehlbergi poolt tuletatud vastavad kaalud [8] ning on leitavad tabelist (Tabel 1). Otsitav lahend järgmise ajasammu jaoks avaldub kujul

$$y(x+h) = y(x) + CH(1)\cdot k_1 + CH(2)\cdot k_2 + CH(3)\cdot k_3 + CH(4)\cdot k_4 + CH(5)\cdot k_5 + CH(6)\cdot k_6. \quad (4.46)$$

k	A(k)	B(k,1)	B(k,2)	B(k,3)	B(k,4)	B(k,5)	CH(k)	CT(k)
1	0	0	0	0	0	0	16/135	1/360
2	1/4	1/4	0	0	0	0	0	0
3	3/8	3/32	9/32	0	0	0	6656/12825	-128/4275
4	12/13	1932/2197	-7200/2197	7296/2197	0	0	28561/56430	-2197/75240
5	1	439/216	-8	3680/513	-845/4104	0	-9/50	1/50
6	1/2	-8/27	2	-3544/2565	1859/4104	-11/40	2/55	2/55

Tabel 1. Fehlbergi koefitsendid

4.5 Molekulaardünaamika ülesanne

Selles punktis on eesmärgiks luua mudel, mis suudab edukalt simuleerida gaasimolekulide liikumist kinnises tasapinnalises kastis. Molekulaardünaamika ülesande realiseerimiseks on vaja kasutada sobivat potentsiaal funktsiooni, mis kirjeldab osakeste vahel mõjuvaid jõudusid. Meie töös kasutame selle jaoks kõige lihtsamat potentsiaali, milleks on Lennard-Jones'i potentsiaal (edaspidi LJP), mis iseloomustab van der Waalsi jõudusid ning mis sõltub vaid osakeste vahelisest kaugusest. Lennard-Jones'i potentsiaal sisaldab kahte liiget, millest esimene on väga tugev tõuke- ja teine veidi nõrgem tõmbejõud [9]. Kahe osakese vahelise kauguse suurenemisel, muutub nende osakeste vaheline jõud väga kiiresti triviaalseks. See on märkimist väärt, kuna saame antud teadmist oma mudelis rakendada simulatsiooni arvutuskiiruse hõlbustamiseks.

4.5.1 Lennard-Jones'i potentsiaal

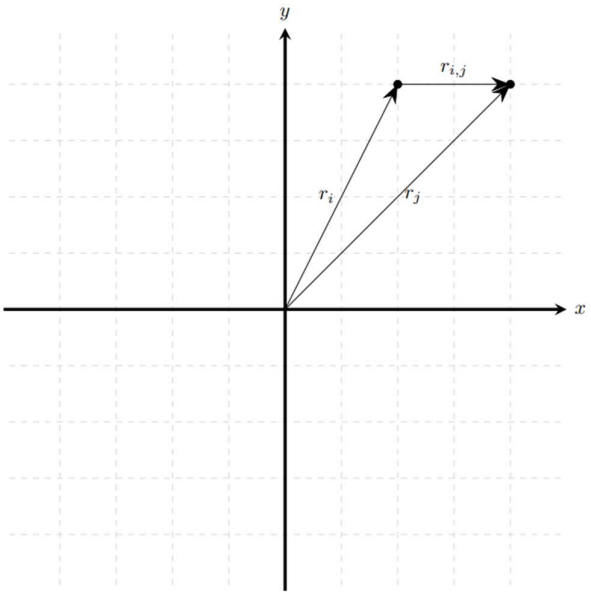
Lennard-Jonesi potentsiaal on defineeritud järgnevalt [10]

$$V(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \quad (4.47)$$

kus *V* on potentsiaal, *r* on osakeste vaheline kaugus, ϵ on potentsiaali augu sügavus, ehk energia ning σ on kahe osakese vaheline kaugus, mille korral potentsiaal *V* nende vahel on null. Osakeste vahelise kauguse *r* moodul on defineeritud kui

$$r_{i,j} = \sqrt{\left(x_j - x_i\right)^2 + \left(y_j - y_i\right)^2} \quad (4.48)$$

kus x_j ja y_j on esimese ning x_i ja y_i on teise osakese asukoha vektori koordinaadid (Joonis 1).



Joonis 1. Kahe osakese vahelise kaugus

Kui rakendada ∇ operaatorit Lennard-Jonesi potentsiaalile, siis saaksime jõu kahe osakese vahel

$$\vec{F} = -\nabla V \qquad (4.49)$$

kus *V* on Lennard-Jonesi potentsiaal ning

$$\nabla = \left[\frac{\delta}{\delta x}\hat{\imath} + \frac{\delta}{\delta y}\hat{\jmath}\right]. \qquad (4.50)$$

Kui rehkendused ära teha, siis saame järgmised jõu projektsioonid

$$f_x = 24\epsilon\sigma^6\left[2\sigma^6\left(\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2\right)^{-7} - \left(\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2\right)^{-4}\right] \cdot \left(x_i - x_j\right) \qquad (4.51)$$

$$\mathrm{f_y} = 24\epsilon\sigma^6\left[2\sigma^6\left(\left(\mathrm{x_i} - \mathrm{x_j}\right)^2 + \left(\mathrm{y_i} - \mathrm{y_j}\right)^2\right)^{-7} - \left(\left(\mathrm{x_i} - \mathrm{x_j}\right)^2 + \left(\mathrm{y_i} - \mathrm{y_j}\right)^2\right)^{-4}\right] \cdot \left(\mathrm{y_i} - \mathrm{y_j}\right) \qquad (4.52)$$

Kui aga asendada tagasi raadius

$$f_x = 24\epsilon\sigma^6[2\sigma^6 r^{-14} - r^{-8}] \cdot \left(x_i - x_j\right) \qquad (4.53)$$
$$f_y = 24\epsilon\sigma^6[2\sigma^6 r^{-14} - r^{-8}] \cdot \left(y_i - y_j\right).$$

Tõstame ühe raadiustes sulgudest välja. Nõnda oleme saanud Lennard-Jones'i potentsiaali poolt tekitatud jõu valemid, mida saame edaspidi kasutada liikumisvõrrandi loomisel. Kuid enne liikumisvõrrandi tuletamist oleks mõistlik teisendada kõik ühikud dimensioonita ühikuteks. See võimaldab meil luua üldist numbrilist mudelit. Vajadusel saab sellisel juhul väljundit lihtsalt mõne koefitsiendiga läbi korrutada [10].

$$f_x = 24\epsilon\sigma^6\left(2\sigma^6\frac{1}{r^{13}} - \frac{1}{r^7}\right)\cdot\frac{\left(x_i - x_j\right)}{r} \qquad (4.54)$$
$$f_y = 24\epsilon\sigma^6\left(2\sigma^6\frac{1}{r^{13}} - \frac{1}{r^7}\right)\cdot\frac{\left(y_i - y_j\right)}{r}$$

4.5.2 Dimensioonita suuruste tuletamine

4.5.2.1 Dimensioonita kaugus ja raadius

Võtame alustuseks dimensioonita kauguse *X*, mille saame kui jagame kauguse x või y kaugusega σ.

$$X = \frac{x}{\sigma} \qquad (4.55)$$

Seega, kui me määrame oma mudelis osakese kauguse dimensioonita ühikuga, siis selleks et teisendada see tagasi dimensiooniga ühikuks, tuleb meil ta lihtsalt σ-ga läbi korrutada.
Kuna liikumisvõrrandis läheb meil vaja ka raadiust, siis tuletame selle ka etteruttavalt ära. Asendades raadiuse mooduli valemis kõik dimensiooniga x-id ja y-id hoopis dimensioonita X-ide ja Y-itega, saame

$$r = \sqrt{\left(\sigma X_j - \sigma X_i\right)^2 + \left(\sigma Y_j - \sigma Y_i\right)^2} = \qquad (4.56)$$
$$= \sigma\sqrt{\left(X_j - X_i\right)^2 + \left(Y_j - Y_i\right)^2}.$$

Ehk

$$r = \sigma R \qquad (4.57)$$

4.5.2.2 Dimensioonita jõud

Asendades dimensioonita kaugused ja dimensioonita raadiuse LJP poolt tekitatud jõu valemisse (4.54), saame

$$\mathrm{f_x} = 24\epsilon\sigma^6\left(2\sigma^6\frac{1}{\sigma^{13}\mathrm{R}^{13}} - \frac{1}{\sigma^7\mathrm{R}^7}\right)\cdot\frac{\sigma\left(\mathrm{X_i} - \mathrm{X_j}\right)}{\sigma\,\mathrm{R}} \qquad (4.58)$$

$$f_y = 24\epsilon\sigma^6\left(2\sigma^6\frac{1}{\sigma^{13}R^{13}} - \frac{1}{\sigma^7R^7}\right)\cdot\frac{\sigma\left(Y_i - Y_j\right)}{\sigma\,R}$$

Kui tõsta sulgude ees olev σ^6 sulgude sisse, taandada ning tegurdada alles jäänud σ sulgude ette, saame viimse kuju, mis kirjeldab dimensioonita ja dimensiooniga jõu seost.

$$f_x = \frac{24\epsilon}{\sigma}\cdot\left(\frac{2}{R^6} - 1\right)\cdot\frac{(X_i - X_j)}{R^8} \qquad (4.59)$$

$$f_y = \frac{24\epsilon}{\sigma}\cdot\left(\frac{2}{R^6} - 1\right)\cdot\frac{(Y_i - Y_j)}{R^8}$$

Ehk üldisemalt

$$f = \frac{24\epsilon}{\sigma}F \qquad (4.60)$$

mida saame kasutada dimensioonita liikumisvõrrandi konstrueerimiseks.

4.5.2.3 Dimensioonita aeg

Alustame Newtoni teisest seadusest

$$f = m\ddot{x}. \quad (4.61)$$

Avaldame $\ddot{x}$ ning asendame dimensioonita suurusega.

$$\sigma\ddot{X} = \frac{f}{m} \quad (4.62)$$

Tõstame σ teisele poole võrdust ning arvestame asjaolu, et siin f on LJP poolt tekitatud jõud , mida avaldasime eelmises punktis (4.60). Kuna motivatsiooniks on leida dimensioonita aeg, siis kirjutame $\ddot{X}$ asemel hoopis $\frac{d^2X}{dt^2}$. Saame

$$\frac{d^2X}{dt^2} = \frac{24\epsilon}{\sigma^2 m}F \quad (4.63)$$

Tõstes nüüd ϵ-i ja σ^2 teisele poole võrdust, saame

$$\frac{d^2X}{d\frac{t^2 24\epsilon}{\sigma^2 m}} = F \quad (4.64)$$

Seega dimensioonita aeg τ on

$$\tau = t\sqrt{\frac{24\epsilon}{\sigma^2 m}} \quad (4.65)$$

4.5.2.4 Dimensioonita kiirus

Nüüd võtame vaatluse alla kiiruse

$$v = \frac{dx}{dt}. \quad (4.66)$$

Kui avaldada eelmises punktis leitud dimensioonita ajast (4.65) suurus t ning asetada see kiirusesse koos dimensioonita kaugusega, siis saame

$$v = \frac{\sigma dX}{d\left(\sqrt{\frac{\tau^2\sigma^2 \cdot m}{24\epsilon}}\right)} = \frac{dX}{\sqrt{\frac{m}{24\epsilon}}d\tau} \quad (4.67)$$

kus $\frac{dX}{d\tau}$ on dimensioonita kiirus V. Ehk

$$V = v \cdot \sqrt{\frac{m}{24\epsilon}}. \quad (4.68)$$

4.5.2.5 Dimensioonita kiirendus

Kiirendus avaldub kujul

$$a = \frac{d^2x}{dt^2}. \quad (4.69)$$

Asendame kauguse dimensioonita kaugusega, ning aja dimensioonita ajaga.

$$a = \frac{dX\sigma}{d\tau^2\left(\frac{\sigma^2 m}{24\epsilon}\right)} = \frac{d^2X}{\left(\frac{\sigma m}{24\epsilon}\right)d\tau^2} \quad (4.70)$$

Suurus $\frac{d^2X}{d\tau^2}$ on dimensioonita kiirendus *A*, ehk

$$a = \frac{A}{\left(\frac{\sigma m}{24\epsilon}\right)} \quad (4.71)$$

$$A = a\left(\frac{\sigma m}{24\epsilon}\right) \quad (4.72)$$

4.5.2.6 Dimensioonita liikumisvõrrand

Kuna nüüd on meil dimensioonita kiirendus ja dimensioonita jõud olemas, siis saame koostada dimensioonita liikumisvõrrandit. Järjekordselt kasutame Newtoni teist seadust

$$a_x = \frac{f_x}{m} \quad (4.73)$$

Asendame jõu *f* dimensioonita LJP jõuga *F* (4.60), ning kiirenduse a_x dimensioonita kiirendusega A_x. Saame

$$\left(\frac{24\epsilon}{m\sigma}\right)A_x = \frac{24\epsilon}{\sigma m}\left(\frac{2}{R^6} - 1\right)\cdot\frac{X_i - X_j}{R^8} \quad (4.74)$$

$$A_x = \left(\frac{2}{R^6} - 1\right)\cdot\frac{X_i - X_j}{R^8}. \quad (4.75)$$

Analoogselt saame ka y projektsiooni.

$$A_y = \left(\frac{2}{R^6} - 1\right)\cdot\frac{Y_i - Y_j}{R^8}. \quad (4.76)$$

Antud liikumisvõrrandite projektsioone kasutame oma numbrilises mudelis molekulaardünaamika visualiseerimiseks.

4.5.2.7 Dimensioonita energia

Mudeli arvutustäpsuse hindamiseks on vaja ka koguenergiat arvutada, seega tuletame ka dimensioonita energia. Koguenergia on kineetilise ja potentsiaalse energia summa

$$E = \frac{mv^2}{2} + U \quad (4.77)$$

kus U on Lennard-Jones'i potentsiaalne energia dimensioonita kujul.

$$U = 4\epsilon\left[\left(\frac{1}{R}\right)^{12} - \left(\frac{1}{R}\right)^{6}\right] \quad (4.78)$$

Kineetilise energia saame, kui asendame kiiruse v dimensioonita kiirusega, mille tuletasime eelnevas punktis.

$$E_k = \frac{mv^2}{2} = \frac{m \cdot V^2 \cdot \frac{24\epsilon}{m}}{2} = 12\epsilon V^2 \quad (4.79)$$

Dimensioonita energia on seega

$$E = 12\epsilon V^2 + 4\epsilon \left[\left(\frac{1}{R}\right)^{12} - \left(\frac{1}{R}\right)^{6} \right]. \quad (4.80)$$

$$E_{dim} = \frac{E}{4\epsilon} \quad (4.81)$$

4.5.3 Raskuskiirenduse arvestamine

Raskuskiirenduse puhul tuleb arvestada asjaolu, et see mõjub vaid molekuli y projektsioonile. Seega koostame liikumisvõrrandi y projektsiooni jaoks, kus mõjub vaid raskuskiirendus.

$$F_y = ma_y \quad (4.82)$$

Maapinna lähedal on gravitatsiooni jõud

$$F_y = mg \quad (4.83)$$

Pannes need üksteisega võrduma saame

$$ma_y = mg \quad (4.84)$$

$$a_y = g \quad (4.85)$$

$$\frac{d^2y}{dt^2} = g \quad (4.86)$$

$$\frac{\sigma \cdot d^2Y}{\frac{\sigma^2 m}{24\,\epsilon} \cdot d\tau^2} = g \quad (4.87)$$

$$A = g \cdot \frac{\sigma m}{24\,\epsilon} \quad (4.88)$$

$$G_{dim} = g \cdot \frac{\sigma m}{24\,\epsilon} \quad (4.89)$$

4.5.4 Temperatuuri arvestamine

Osakeste keskmist ruutkiirust $\overline{v^2}$ saab arvutada summeerides iga osakese ruut kiirust ${v_i}^2$ ning jagades seda osakeste arvuga *n*, ehk

$$\langle v^2 \rangle = \frac{1}{n} \cdot \sum_{i}^{N} v_i^2 \qquad (4.90)$$

Gaaside kineetilises teoorias on osakese keskmine kineetiline energia molekuli kohta aga [11]

$$\frac{1}{2} m \overline{v^2} = \frac{3}{2} k_B T. \qquad (4.91)$$

Süsteemi kogu kineetiline energia on võrdne kõikide osakeste kineetilise energiaga. Ehk võrrandi vasakpool tuleb korrutada osakeste arvuga *n*. Kui võtta, et $\overline{E_k} = \frac{1}{2} n m \overline{v^2}$ ning avaldada eelmisest võrrandist *T*, siis saame

$$T = \frac{m\overline{v^2}}{3k_B} \qquad (4.92)$$

$$T = \frac{2}{3} \frac{\overline{E_k}}{n k_B} \qquad (4.93)$$

Avaldades kiiruse *v* sõltuvuse.

$$\overline{v^2} = 3 \cdot \frac{T \cdot k_B}{m} \qquad (4.94)$$

$$\bar{v} = \sqrt{3 \cdot \frac{T \cdot k_B}{m}} \qquad (4.95)$$

4.5.5 Tasakaalu kauguse määramine

Meie programmis on eesmärgiks paigutada osakesed nõnda, et need oleksid tasakaalu asendis. Ehk osakestele mõjuvad tõmbe ja tõukejõud peavad olema võrdsed. Seega, tuleb meil ennem määrata osakeste vaheline kaugus, mille puhul nad tasakaalu asendis oleksid. Teeme seda analüütiliselt ja seejärel kinnituseks graafiliselt/numbriliselt.

Lennard-Jonesi potentsiaali poolt tekitatud jõud sisaldab juba summaarset jõudu, ehk tõmbe ja tõukejõudu. Võttes LJP-st tuletist raadiuse järgi saame jõu ning kui panna ta võrduma nulliga, siis peaksime saama avaldada raadiuse, mille puhul tõuke ja tõmbejõud on võrdsed. Leiame liikmete tuletised eraldi

$$\left(\frac{1}{R^{12}}\right)' = -12R^{-13} \qquad (4.96)$$

$$\left(\frac{1}{R^6}\right)' = -6R^{-7} \qquad (4.97)$$

Kui panna nüüd nende kahe liikme summa võrduma nulliga, siis saame võrrandi, kust saame avaldada *R*-i mille puhul tõmbe ja tõukejõud on võrdsed.

$$-\frac{12}{R^{13}} + \frac{6}{R^7} = 0 \qquad (4.98)$$

$$\frac{6}{R^7}\left(-\frac{2}{R^6} + 1\right) = 0 \qquad (4.99)$$

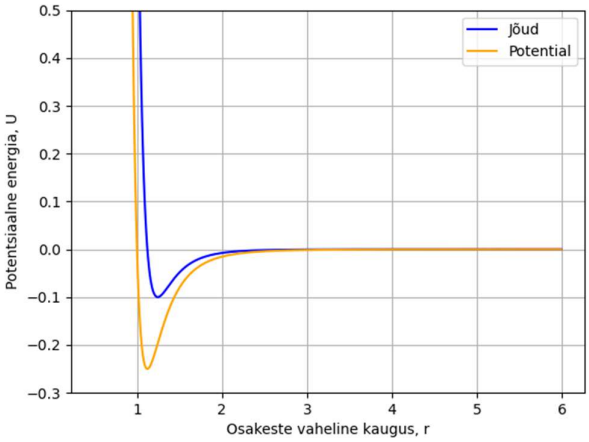
Ühel juhul on kahe osakese vahelise tõmbe ja tõukejõud võrdsed, kui $R = \infty$, kuid see ei ole väga praktiline kaugus, millega mudelit luua. Seega peaks saama sulgude sees olevast avaldisest sobiva *R*-i kätte.

$$-\frac{2}{R^6} + 1 = 0 \qquad (4.100)$$

$$R^6 = 2 \qquad (4.101)$$

$$R \approx 1.12 \qquad (4.102)$$

Kontrollime saadud tulemust ka numbriliselt/graafiliselt. Selle jaoks koostasime Pythoni skripti mille leiab lisast (Lisa 1). Numbrilise/graafilise lahendamise puhul ei ole meil vaja tuletist võtta ning piisab lihtsalt sellest, et leiame LJP potentsiaali miinimumi.



Joonis 2. Lennard-Jones'i potentsiaali ja selle poolt tekitatud jõu graafik

Lähtudes Pythoni skripti tulemusest ning ka skripti poolt koostatud joonisest (Joonis 2), leiame, et potentsiaali miinimum asub tõepoolest kaugusel 1.12.

4.6 Kiiruse Verlet algoritm

Molekulaardünaamika ülesande puhul on meil vaja arvutada kõikide süsteemis olevate osakeste liikumisvõrrandid. Osakeste arv sellise ülesande puhul on aga päris suur ja seega RK4 meetod enam ei kõlba. Ohverdame seega arvutustäpsust ja kasutame veidi kiiremat algoritmi osakeste võrrandite lahendamiseks.

Kiiruse Verlet algoritm on tuletatud klassikalisest mehaanikast kinemaatilisest võrrandist. Sisuliselt rakendataksegi kinemaatilist võrrandit, kuid igal ajasammul eraldi [12]. Kiiruse Verlet algoritm näeb välja järgnevalt

$$x(t+\Delta t) = x(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 \qquad (4.103)$$
$$v(t+\Delta t) = v(t) + \frac{a(t)+a(t+\Delta t)}{2}\Delta t.$$

Algoritmi realiseerimisel meie programmis, tuleb silmas pidada, et esimesel iteratsioonil on $x(t)$ ja $v(t)$ meie osakeste algtingimused, ehk algsed koordinaadid ja kiirused ning $a(t)$ on meie liikumisvõrrand, mis sõltub vaid algkoordinaatidest.

5 Programmide realiseerimine

Kuna töö motivatsiooniks oli luua füüsika programme veebivõrgu jaoks, siis pidi hoolikalt valima sobivat programmeerimiskeelt, mis seda ilma suurema vaevata võimaldaks. JavaScript osutus parimaks kandidaadiks, kuna see on üks populaarsemaid programmeerimiskeeli veebi arenduse jaoks [13]. JavaScripti eeliseks on see, et seda suudab iga arvuti ja ka nutiseade brauseris interpreteerida, sellesse on integreeritud HTML (HyperText Markup Language) ning Java ja seda on relatiivselt lihtne selgeks õppida [14]. Lisaks, et meie mudeleid veidi lihtsamini visualiseerida oleks, kasutame ka P5JS teeki, mis loob meie jaoks HTML *canvas* objekti, võimaldab interakteerida veebilehe objektidega ning omab ka mitmeid erinevaid visualiseerimis funktsioone, mida me ise looma ei pea [15]. Iga programmi puhul tuleb meil visualiseerida ka graafikuid. Graafikute joonestamiseks on ka olemas teeke, kuid meie oma töös neid ei kasuta, kuna üleliigsete teekide kasutamisega võivad kaasneda turva riskid [16].

5.1 Kepleri ülesanne

Nagu tavaks on, deklareerime enda programmi alguses kõik programmis kasutatavad muutujad ja keskkonna parameetrid. Meie programmis oli loodud kolm erinevat fikseeritud keskkonda, milleks on kolm erinevat taevakeha. Nende taevakehade hulka kuuluvad Maa, Mars ning Veenus. Kuna ainukesed parameetrid, mis meie mudelis mingit kasutust leiavad, on mass ja raadius, siis defineerisimegi iga taevakeha jaoks vaid need suurused. Lisaks taevakehadele, määrasime ära ka satelliidi massi ja raadiuse ning süsteemi algtingimused liikumisvõrrandite jaoks. Keskkond initsialiseeritakse funktsiooniga „init_model(num)“, kus parameeter „num“ on keskkonna järjekorranumber. Kokku on neid kolm ning initsialiseeritakse esimene mudel, mis sisaldab Maa massi ning raadiust. Algtingimused on programmis algselt vaikimisi fikseeritud koordinaatidega $x = R_M + h, y = 0$ ning $v_x = 0$ ja $v_y = 0$. Kus R_M on taevakeha keskmine raadius ja h on oribiidi kõrgus pinnast.

Kuna Kepleri ülesande ning harmooniliste võngete ülesannete programmides on liikumisvõrrandite integreerimiseks kasutuses RKF45 meetod, siis on meil vaja esmalt defineerida suur hulk Fehlbergi kordajaid, mis olid meil välja toodud tabelis (Tabel 1). Otsustasime selle jaoks luua funktsiooni „coefficients()“, mille käivitamisel deklareeritakse antud kordajad. Kordajad salvestati kahemõõtmelisse andmemassiivi nii, et kordajate indeksid langeksid kokku RK4 algoritmis kasutatavate indeksitega, et kood loetavam oleks.

Funktsioonis „setup()“ käivitame nii „coefficients()“ kui ka „init_model(num)“ funktsioonid. Lisaks loome programmi jaoks kõik vajalikud DOM (Document Object Model) elemendid, ehk kõik vajalikud slaiderid, sisendkastid ja nupud, et kasutaja nendega interakteerida saaks. Nõuab ka mainimist, et „setup()“ funktsioon iseenesest veel integreerimist ei alusta, vaid on lihtsalt mõeldud kogu liidese ja arvutuste jaoks vajalike andmete initsialiseerimiseks.

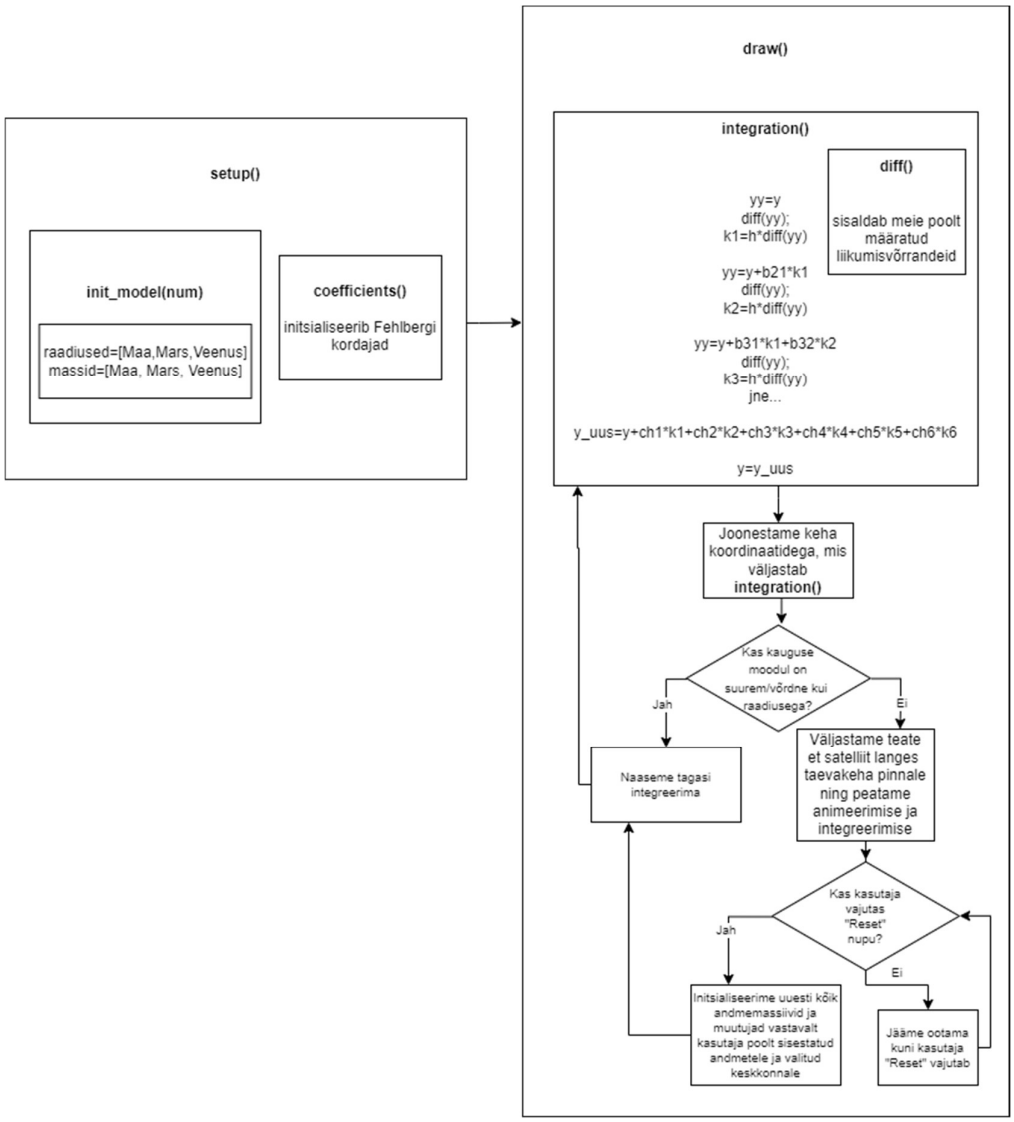
Liikumisvõrrandite integreerimine ning visualiseerimine algab funktsioonis „draw()“. Selles funktsioonis käivitame funktsiooni „integration()“, mis integreerib meie liikumisvõrrandeid iteratiivselt RK4 meetodi alusel. Integreerimise funktsioonis on kasutuses ka „diff()“ funktsioon,

milles on defineeritud meie liikumisvõrrandid. Integreerimis funktsiooni väljundiks on uued algtingimused, mida järgmises iteratsioonis uuesti integreerimis funktsioonis kasutatakse. Kuna väljundite seas on ka satelliidi koordinaadid, siis saame neid koordinaate kasutada satelliidi liikumise visualiseerimiseks.

Juhul kui kasutaja poolt sisestatud algkoordinaatide või algkiiruse projektsioonide tagajärjel peaks satelliit langema taevakeha pinnale, siis väljastatakse kasutajale vastav teade. Teade väljastatakse kasutajale vaid juhul, kui satelliidi kauguse moodul taevakeha tsentrist on väiksem või võrdne taevakeha raadiusega

$$\sqrt{x^2 + y^2} \leq R_M. \qquad (5.1)$$

Tingimuse täitumisel jääb ka animatsioon seisma ja selle taas käivitamiseks tuleb „Reset“ nuppu vajutada. Allpool on välja toodud joonis (Joonis 3), mis kirjeldab programmi töötamist.

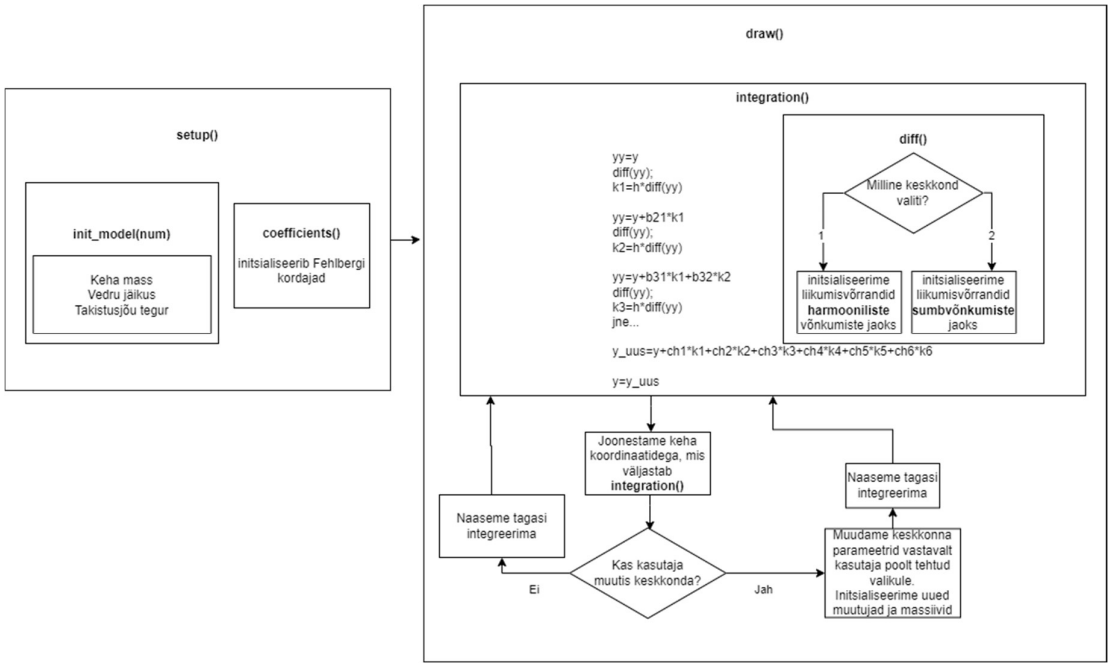


Joonis 3.Lihtsustatud Kepleri programmi protsessi diagramm

5.2 Harmoonilised võnkumised

Antud programm erineb eelmisest vaid liikumisvõrrandite, keskkonna parameetrite kui ka visuaaliseeringu poolest. Muu struktuur ühtib. Keskkondi on kokku kaks. Esimene nendest kirjeldab harmoonilisi võnkumisi ja teine sumbuvaid võnkumisi. Olulisemad parameetrid, mida nendes mudelites initsialiseerime on: elastsuskonstant k, võnkuva keha mass m ning takistusjõutegur α. Algtingimused liikumisvõrrandite jaoks fikseerisime nii, et vedru oleks algselt tasakaalu olekus, ehk mille puhul nihe oleks 0. Seega algkoordinaat $x = 0$ ning kiirus samuti $v_x = 0$. Kasutaja saab programmi akna siseselt klõpsata soovitud kaugusele, nihutades nõnda mudelis olevat keha ning määrata sellega uued liikumisvõrrandi algtingimused. Samuti on ka võimalus hiire vasakut nuppu all hoides lohistada keha soovitud kaugusele. Selle lahti laskmiseks lastakse lahti ka hiire vasak nupp.

Liikumisvõrrandite integreerimine toimib sama eeskirja järgi nagu ka Kepleri ülesande programm, kuid ühe väikse erinevusega. Nimelt kui „integration()“ funktsioon „diff()“ funktsiooni käivitab, siis on selles funktsioonis kõigi kolme keskkonda kirjeldavad liikumisvõrrandid. Et integreerida vaid valitud mudeli liikumisvõrrandit, määrasime ära iga liikumisvõrrandi jaoks tingimuslaused. Ehk, kui valitud mudeli järjekorra number on 2, siis kasutab vaid teist keskkonda kirjeldavaid liikumisvõrrandeid ning muid ignoreerib. Allpool on välja toodud joonis (Joonis 4), mis kirjeldab programmi töötamist.



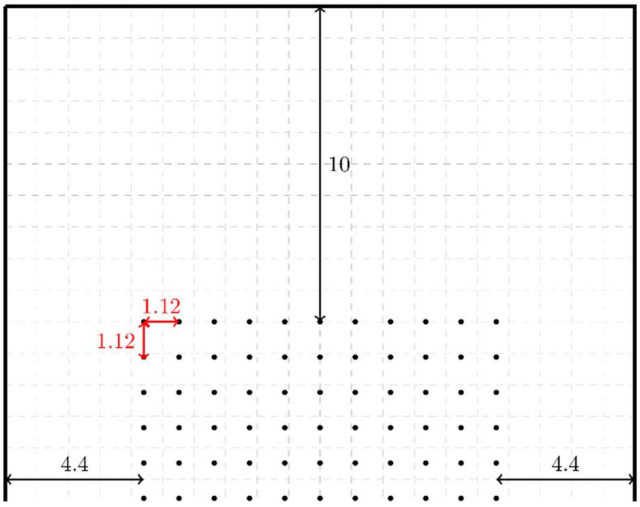
Joonis 4. Lihtsustatud võnkumiste programmi protsessi diagramm

5.3 Molekulaardünaamika

5.3.1 Muutujate, parameetrite ja algtingimuste deklareerimine

Raudvara alapeatükkis „Molekulaardünaamika ülesanne“ teisendasime kõik ühikud dimensioonita ühikuteks. Seega selles programmis ei ole meil vaja osakeste omadusi defineerida, kuna nende liikumine sõltub vaid üksteise vahelisest kaugusest. Küll aga on vaja fikseerida ära keskkond, milles osakesed viibivad põrgete arvutamiseks ning määrata ära ka osakeste arv. Osakeste arvu määramine ei ole meelevaldne, sest kui osakesi on liiga vähe, siis ei pruugi programm olla kasutaja jaoks nähtuse uurimiseks piisavalt informatiivne. Samas kui osakesi on liiga palju, siis võib osakeste liikumisvõrrandite hulk olla liiga suur, et visualisatsioon piisavalt kiiresti töötaks. Seega oma töös leppisime 200 osakesega.

Anum, milles 200 osakest viibivad on ristküliku kujuline laiusega 20 dimensioonita ühikut ning kõrgusega 40 dimensioonita ühikut. Esimeses implementatsioonis üritasime osakeste koordinaadid määrata programmis juhuslikult JavaScripti „Math.random()“ funktsiooni kasutades. Kuid selgus, et nõnda võivad osakesed juhtumisi sattuda liiga lähestikku, mille tagajärjel mõjub osakestele väga tugev tõukejõud ja seega omandavad nad ka väga suure kiirenduse. Suure kiirenduse tõttu võivad osakesed isegi väga väikse ajasammu korral kastist välja hüpata või jätta simulatsioonist väga ebaloomuliku mulje. Seega otsustasime probleemi lahendada nõnda, et paigutasime osakesed võre kujuliselt, kus osakeste vahelised kaugused on võrdsed osakeste tasakaalu oleku kaugusega. Antud kauguse saime aga juba kätte raudvara punktis 4.5.5, milleks tuli dimensioonita $r = 1.12$. 200 osakesest koosnev võre on samuti ristküliku kujuline, mõõtmetega 10x20, kus iga osakese vahel on tasakaalu oleku kaugus. Osakeste võre oli omakorda nihutatud anuma keskele nii *x* kui ka *y* koordinaadi suunas, et ülevalt, alt, vasakult ja paremalt oleks seina ja osakeste vahel väike vahe (Joonis 5), mis võimaldab vajadusel juhuslike kiirusprojektsioonide genereerimisel liikuda igas suunas.



Joonis 5. Võre paigutus kastis ning selle konfiguratsioon

Osakeste algkiiruste projektsioonide määramiseks on meil kaks erinevat lähenemist. Esimesel juhul on kõikide osakeste algkiiruste projektsioonid juhuslikult genereeritud vahemikust 0-1, et

jäljendada osakeste kaootilist liikumist. Teisel juhul on osakeste algkiirused võrdsed nulliga. Viimase variandi korral osakesed liikuma ei hakka, kuna nad on paigutatud tasakaalu olekus oleva võre kujuliselt. Kuid kui tuua sisse raskuskiirendus, siis langeb osakeste võre kasti põhja, mille tagajärjel tekkib samuti osakeste kaootiline liikumine. Nende kahe mudeli vahel laseme kasutajal ise valiku teha liideses.

Kui algtingimused on fikseeritud, siis alustame liikumisvõrrandite integreerimisega. Algselt käivitub funktsioon „setup()“, mis initsialiseerib funktsiooniga „forces()“ osakestele mõjuvad jõud ehk kiirendused.

5.3.2 Funktsiooni „forces()“ kirjeldus

Initsialiseerime kiirenduse projektsioonide massiivid, kus indeks kirjeldab iga osakese summaarse kiirenduse. Algselt on kiirenduste projektsioonide summad 0. Seejärel arvutame iteratiivselt topelt *for*-tsükliga iga osakese kiirenduse.

Kusjuures, osakeste vahelisi jõude ei arvutata, kui osakeste vaheline kaugus piisavalt suur on, sest sellisel juhul on osakesele mõjuv jõud triviaalne. See aitab meil arvutamiskiirust veidi hõlbustada. Meie programmis on kirjeldatud tingimus kirjas järgmisel kujul

$$if\ (dist < rcut),$$

kus *dist* on osakeste vaheline kauguse moodul ning *rcut* on kaugus mille puhul on jõu ignoreerimine õigustatud. Sobivat *rcut* väärtust saab arvutada kas analüütiliselt või määrata graafiliselt LJP graafikult. Meie oma programmis määrasime meelevaldselt, et $rcut = 3$. Kui oleme vaadeldava osakeste paari vahelise jõu projektsioonid kätte saanud, siis liidame vastava osakese kiirenduse projektsiooni selle osakese kiirenduse projektsiooni summale. Paralleelselt arvutame samas tsükli iteratsioonis ka teise osakese summaarsed kiirendused. Jõud mis mõjub teisele osakesele esimese osakese poolt on võrdne, kuid vastupidise suunaga. Seega seda me osakese summale juurde mitte ei liida, vaid lahutame. Veidi jälgitavamad ja arusaadavamad kommentaarid koos koodiga on leitavad lisast (Lisa 2).

Kui oleme kõikide osakeste summaarsed kiirenduste projektsioonid välja arvutanud, siis võime sõltuvalt kasutaja soovist võtta arvesse ka raskuskiirenduse. Raskuskiirenduse olemasolu või ignoreerimist määrab kasutaja ära kasutajaliideses raadionupuga. Juhul, kui kasutaja soovib raskuskiirendust arvestada, siis lahutatakse igalt kiirenduse *y*-projektsioonilt maha programmis määratud raskuskiirenduse konstant a_g.

Kuigi raudvara punktis oli meil dimensioonita raskuskiirendus tuletatud, siis kahjuks olime sunnitud oma programmis kasutama abstraktset raskuskiirenduse väärtust, kuna vastasel juhul oli kiirendus liiga suur ja osakesed hüppasid isegi väiksema ajasammu korral kastist välja. Meie hinnangul abstraktselt valitud raskuskiirenduse väärtus mudeli kasutamise kogemust kehvemaks ei tee, kuna motivatsioon on seda ikkagi kasutada õpetamise eesmärkidel, mitte päris teadus uuringute jaoks.

5.3.3 Funktsiooni „calc1()“ kirjeldus

Kui algtingimused koos esimeste kiirendustega on välja arvutatud, siis algab „draw()“ funktsioonis liikumisvõrrandite integreerimise protsess. Liikumisvõrrandeid integreerib „calc1()“ funktsioon. Funktsiooni ehitus on analoogne teoreetilistes alustes punktis 4.6 kirja pandud kiiruse Verlet algoritmiga, kuid esineb ka üksikuid nüansse.

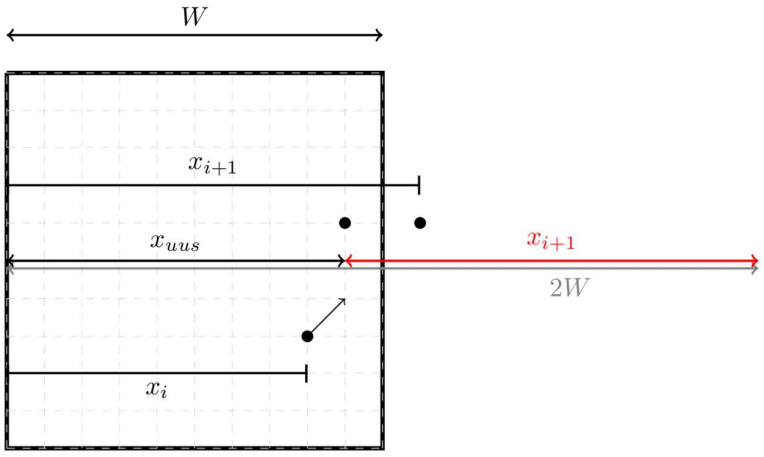
Nimelt on funktsioonis kaks eraldi tsüklit. Esimeses tsüklis arvutame algkoordinaatide, algkiiruse kui ka algkiirenduse alusel järgmise ajasammu osakese koordinaadid. Järgmisena oleks vaja arvutada osakeste kiiruste projektsioonid järgmisel ajahetkel, kuid selle arvutuse jaotasime oma programmis kaheks eraldi liikmeks, ehk võrrandi

$$v(t+\Delta t) = v(t) + \frac{a(t)+a(t+\Delta t)}{2}\Delta t \qquad (5.2)$$

jaotasime hoopis kujule

$$v(t+\Delta t) = \underbrace{v(t) + \frac{1}{2}\cdot a(t)\cdot \Delta t}_{A} + \overbrace{\frac{1}{2}\cdot a(t+\Delta t)\Delta t}^{B}. \qquad (5.3)$$

Ehk esimeses tsüklis arvutasime vaid järgmise ajasammu kiiruse liiget, mis on tähistatud *A*-ga. Seega esimese tsükli väljunditeks on järgmise ajasammu koordinaadid, ning järgmise ajasammu kiiruse projektsiooni A osa.



Joonis 6. Osakese kasti sisse tagasi paigutamine põrke korral

Selle kahe tsükli vahele jääb meil ka põrgete kontroll tingimused. Esmalt kontrollime põrgete x-projektsioone. Kui osakese x-koordinaat järgmisel ajasammul peaks langema kastist välja paremale poole, siis tõstame ta kasti sisse tagasi valemiga

$$x(t+\Delta t)_{uus} = 2 \cdot W - x(t+\Delta t), \qquad (5.4)$$

kus *W* on kasti laius, $x(t+\Delta t)$ on osakese *x* koordinaat, mis langeb kastist välja paremalt poolt ning $x(t+\Delta t)_{uus}$ on kasti tagasi asetatud osakese koordinaat (Joonis 6). Kui osake langeb aga kastist välja vasakult poolt, siis on muudame lihtsalt selle koordinaadi märki

$$x(t+\Delta t)_{uus} = -x(t+\Delta t). \qquad (5.5)$$

Analoogselt käitume ka *y*-projektsiooni põrgetega. Kui osake hüppab kastist välja alla poole, siis saame ta kasti tagasi asetada valemiga

$$y(t+\Delta t)_{uus} = 2 \cdot H - y(t+\Delta t) \qquad (5.6)$$

ning kui osake langeb kastist välja ülevalt, siis

$$y(t+\Delta t) = -y(t+\Delta t). \qquad (5.7)$$

Loomulik oleks põrgete korral ka muuta vastavate projektsioonide kiiruste märke, ehk põrke korral

$$v_{x_{uus}} = -v_x \qquad (5.8)$$
$$v_{y_{uus}} = -v_y$$

Kui osake ei põrku, ehk kastist välja ei hüppa, siis salvestame järgmise pooliku ajasammu arvutatud kiiruse projektsiooni uueks kiiruse väärtuseks, mille saime esimeses tsüklis.

$$v_{x_{uus}} = v_x\left(t + \frac{\Delta t}{2}\right) \qquad (5.9)$$
$$v_{y_{uus}} = v_y\left(t + \frac{\Delta t}{2}\right)$$

Nüüd kui meil on osakesed tagasi kasti sisse asetatud ning kiiruste suunad ka ära parandatud, siis arvutame järgmise ajasammu kiirendused välja, kuna neid on vaja järgmise ajasammu kiiruse arvutamiseks.

Selle jaoks teostame arvutused taaskord “forces()” funktsiooniga, mille töökorda kirjeldasime juba eespool. Seejärel kasutades uusi kiirendusi ja eelmises tsüklis arvutatud pooliku ajasammu kiiruseid, arvutame järgmise ajasammu kiiruse, liites pooliku sammu kiirusele A juurde liikme *B* nagu valemis (5.3).

Kasutajal on võimalik kasutajaliideses sisse lülitada ka temperatuuri slaider. Temperatuuri olemasolul on põrgete tagajärjel uuendatud kiirused veel omakorda kohandatud sõltuvalt temperatuuri slaideri väärtusest. Anumas olevate osakeste kiiruste kohandamise süsteem temperatuurist sõltuvalt näeb välja järgnevalt.

Eeldame, et kui anumas olev osake põrkub seinaga, siis põrkub ta hoopis ühe teise osakesega, mille impulss ehk kiirus on määratud temperatuuri slaideri väärtusega ning on seotud seosega (4.95). Oma programmis kasutame lihtsustatud impulsi edastamise mehhanismi. Anuma seintelt peegelduvate osakeste kiiruste projektsioonid arvutame järgmiste seostega:

$$v_x = v_{slider} \cdot \frac{v_x}{v_{moodul}} \qquad (5.10)$$
$$v_y = v_{slider} \cdot \frac{v_y}{v_{moodul}},$$

kus v_{slider} on liidese temperatuuri slaideri poolt määratud kiiruse väärtus ning v_{moodul} on osakese kiiruse moodul. Võrduste paremal pool olevad v_x ja v_y on osakese kiiruse projektsioonid ning võrduste vasakul pool olevad v_x ja v_y on uued osakese kiiruse projektsioonid.

5.3.4 Animatsiooni kiiruse suurendamine

Väikse integreerimis ajasammu korral tuleb küll täpsem tulemus, kuid animatsiooni kiirus osutub aeglaseks. Üks viis kuidas antud probleemi lahendada, ajasammu suurendamata, on luua animeerimis tsükli sisse veel üks eraldi tsükkel, mis arvutab ühes animatsiooni iteratsioonis näiteks kümme ajasammu. Ehk iga kord, kui meie programmis “draw()” funktsioon iteratiivselt joonise loob, integreeritakse liikumisvõrrandeid 10 korda. Selline meetod säilitab meie väikese ajasammuga määratud arvutustäpsuse ning jätab animeerimisel mulje, nagu oleks kasutuses suurem ajasamm.

5.4 Graafikute joonestamine

Kõikide ülevalpool mainitud programmide korral arvutasime lisaks koordinaatidele ka iteratiivselt „draw()“ funktsioonis kehade kineetilised, potentsiaalsed ning koguenergiad. Arvutuste tulemused salvestati vastavatesse andmemassiividesse, mis võimaldasid meil joonestada animeeritud graafikuid. Graafiku *y*-teljele kandsime energiad ning *x*-teljele aja.

Kuna programmi koordinaadistiku nullkoht asub veebibrauseri üleval vasakul nurgas ning graafikute kast asub animatsiooni kastist veidi madalamal, siis tuleb kõik arvutatud tulemused nihutada veebibrauseri suhtes allapoole, et graafik langeks täpselt graafiku kasti sisse. Näiteks, kui animatsiooni kasti kõrgus on 500 *px* ning graafiku kasti kõrgus on 300 *px* ja graafikute nullkoht asub täpselt graafiku kasti keskel, siis tuleb kõikide arvutatud punktide nihutamiseks liita *y*-komponentidele 650 *px*. Lisaks tuleb arvestada ka asjaoluga, et veebibrauseri *y*-telg on suunatud alla, mis tähendab, et kui me soovime graafikut joonestada hoopis *y*-teljega ülesse poole, siis tuleb kõikide arvutatud väärtuste märki muuta. Viimaks, kui arvutatud väärtused on oma suurusjärgu poolest liiga suured või liiga väiksed, siis on mõistlik mõne sobiva konstandiga need suurused läbi korrutada. Näiteks Kepleri ülesande korral olid energia väärtused suurusjärguga 10^8. Et antud suurus 300 piksli kõrgusega kasti sisse mugavalt ära mahutada, jagasime kõik energia väärtused graafiku joonestamisel 10^8-ga läbi.

Juhul kui graafikule kantav suurus on pärast teisendusi ikkagi suurem kui meie graafiku kasti kõrgus, siis selleks, et graafikule kantavad punktid animatsiooni peale ei langeks, koostasime järgmise tingimuse.

$$if(abs(arvutatud\_suurus) \leq graafikute\_kõrgus/2) \qquad (5.11)$$

Antud tingimus kontrollib, kas arvutatud suuruse absoluutväärtus on väiksem või võrdne pool graafiku kasti kõrgusest. Graafiku punktid joonestataksegi vaid antud tingimuse täitumisel, et vältida graafiku joonise langemist animatsiooni peale.

Kui kanda graafikule vaid punktid, siis võib juhtuda et ühes ajaühikus muutub arvutatud suurus väga kiiresti, mille tagajärjel võib jääda punktide vahele märgatavaid vahesid. Selle vältimiseks ühendasime punktid lihtsalt joonega ära, alustades teisest graafikule kantud punktist.

5.4.1 Graafiku punktide nihutamine

Kuna andmemassiive on palju ning kõik nendest täituvad iteratiivselt uute väärtustega, siis võib pärast mõnda aega töötamist avastada, et programmid jäävad oma animeerimise kiiruse poolest aeglasemaks. Põhjus seisneb selles, et kuna andmemassiivid kasvavad lõputult ning me üritame neid igal iteratsioonil graafiku joonestamisel kasutada, siis saab arvuti operatiivmälu väga kähku täis.

Selle probleemi lahendamiseks koostasime ühe staatilise piiratud suurusega ajamassiivi, mis kirjeldab graafikute puhul *x*-telge. Antud massiivi numbrilised väärtused ega suurus ei muutu. Lisaks iga kord kui me „draw()“ funktsioonis graafikule kantavaid suurusi arvutame ja massiividesse salvestame, siis kontrollime, et ega massiivi pikkus meie poolt määratud pikkust ei

ületa. Kui ületab, siis lisame andmemassiivi lõppu hiljuti arvutatud suuruse ning eemaldame massiivi algusest esimese suuruse. Ehk nihutame piltlikult öeldes massiivi täitumisel väärtused koguaeg vasakule.

Graafiku joonestamiseks kasutamegi x-telje koordinaatidena fikseeritud muutumatut ajamassiivi ning y-teljel on koordinaatideks arvutatud suurused, mis asuvad nihkuvates massiivides.

5.5 Programmide üleslaadimine TTÜ Parseki serverile

Programmid said edukalt valmis ja töötavad ilma tõrgeteta. Programmid olid laetud ülesse TTÜ Parseki serverile kasutajatele vaatamiseks ja kasutamiseks. Link Parseki serverile on leitav lisast (Lisa 3).

6 Mudelite arvutustäpsuste hindamine

Mudelite arvutustäpsuse hindamiseks kasutame kõige elementaarsemat meetodit, analüüsides süsteemide koguneriga muutumist.

Kepleri ülesande ning harmooniliste võngete ülesande korral on meil võimalik luua kinniseid süsteeme, kus välised jõud puuduvad. Ehk Kepleri ülesande korral mõõdame koguenergiat vaid olukorras, kus satelliidile mõjub ainult gravitatsiooni jõud ning võngete korral uurime vaid harmoonilise võnkumise koguenergiat. Analoogselt on meil võimalik luua ka kinnine süsteem molekulaardünaamika programmi korral, elimineerides sõltuvuse raskuskiirendusest ja temperatuurist.

Täpsuse hindamiseks kasutasime järgmisi etappe

1. Käivitasime mudeli ning panime kirja koguenergia väärtuse esimesel ajahetkel
2. Lasime simulatsioonil töötada ligikaudu 10 minutit
3. Panime kirja mudeli koguenergia väärtuse pärast 10 minutit töötamist
4. Seejärel jagasime mudeli alguses saadud koguenergia väärtuse E_1 viimase kirja pandud koguenergia väärtusega E_2
5. Leidsime koguenergia väärtuse muutumise protsendi valemiga $\frac{|E_1-E_2|}{E_1} \cdot 100\%$

Protsentuaalse muutumise põhjal hindamegi mudeli täpsust.

6.1 Tulemused

Kepleri ülesande puhul kasutasime simulatsioonis algtingimusteks $x = R_M + 100, y = 0$ ja $v_x = 0$, $v_y = v_I$ ehk esimene kosmiline kiirus. Esimesel ajahetkel oli arvutatud koguenergia väärtuseks

$$E_1 = -3079909420.259659\,J$$

Pärast 10 minutit oli arvutatud koguenergia väärtus

$$E_2 = -3079909420.5218773\,J$$

Seega protsentuaalselt muutus koguenergia väärtus

$$\frac{|-3079909420.259659 + 3079909420.5218773|}{-3079909420.259659} \cdot 100\% \approx -8.5 \cdot 10^{-9}\,\%$$

Antud muutuse suurusjärk on niivõrd väike, et seda võib sisuliselt pidada olematuks. Seega järeldame, et antud ülesandes arvutustäpsus on piisavalt hea, et kasutada nähtuse simuleerimiseks.

Harmooniliste võnkumiste puhul kasutasime algtingimusteks $x = 15, v_x = 0$. Esimesel ajahetkel oli koguenergia väärtuseks

$$E_1 = 562.5000002550144\,J.$$

Pärast 10 minutilist võnkumist oli koguenergia väärtus

$$E_2 = 562.5012656377827\,J.$$

Saame, et protsentuaalselt muutus koguneriga

$$\frac{|562.5000002550144\ -\ 562.5012656377827|}{562.5000002550144} \cdot 100\% \approx 2.25 \cdot\ 10^{-4}\,\%$$

mis on suurusjärgu poolest küll suurem kui Kepleri ülesande puhul, aga siiski piisavalt väike, et kinnitada, et simulatsioon on piisavalt täpne.

Molekulaardünaamika programmis jätsime raskuskiirenduse ning temperatuuri sõltuvuse välja ning initsialiseerisime osakesed juhuslike kiiruste projektsioonidega. Lisaks, kuna koguenergia väärtus kõikus väga palju, siis otsustasime võtta aritmeetilise keskmise kümnest mõõtmisest. Mõõdetud tulemused kanname mugavama ülevaate jaoks tabelisse (Tabel 2).

Algne	318.52	291.53	319.76	377.55	421.22	381.79	369.96	295.18	323.03	372.80
Lõplik	367.30	341.81	316.42	349.06	437.81	415.52	441.46	311.70	370.91	413.36
%	15.31 %	17.25 %	1.04%	7.55%	3.94%	8.83%	19.33 %	5.60%	14.82 %	10.88 %

Tabel 2. Molekulaardünaamika koguenergia alg- ja lõppväärtuste mõõtmised

Aritmeetiline keskmine on seega

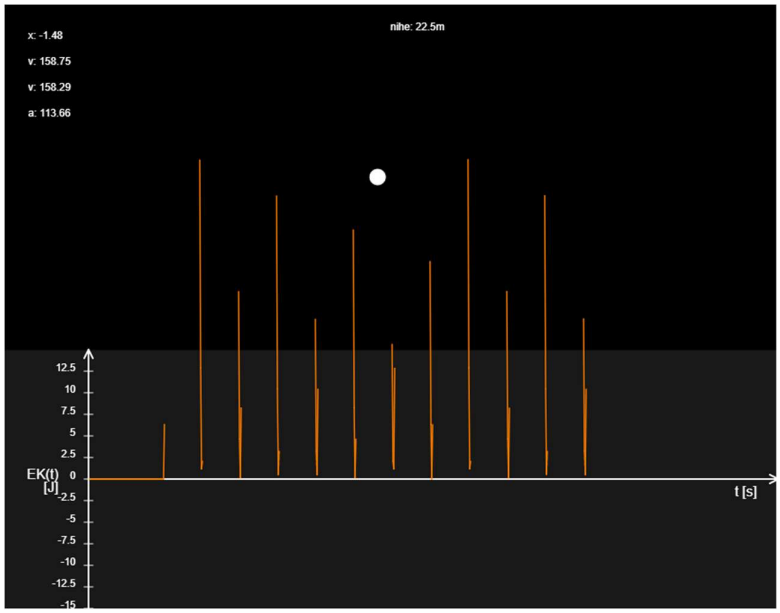
$$\frac{15.31 + 17.25 + 1.04 + 7.55 + 3.94 + 8.83 + 19.33 + 5.60 + 14.82 + 10.88}{10} = 10.45\%$$

Saadud tulemuse suurusjärk on võrreldes eelmiste mudelitega palju suurem. Teoreetiliselt on tegemist veidi kehvema arvutustäpsuse algoritmiga ja seega oli kehvem tulemus oodatud. Mis antud olukorra haruldaseks teeb on see, et koguenergia väärtus on simulatsiooni käigus kõikuv, kuid see peaks olema stabiilne ehk enamvähem konstante. See viitab sellele, et kas koguenergia valemid, mida eelnevalt tuletasime on vigased või on kuskil programmi koodis mõni arvutusviga, mida töö autor märkamata jättis. Kuna koguenergia väärtused kõikusid nii suurema kui ka väiksema väärtuse suunas, siis võib järeldada, et koguenergia ei ole ainult kasvav vaid kohati ka kahanev, isegi vigaste arvutuste olemasolul, ja seega koguenergia väärtus tegelikult algsest väärtusest väga palju erineda ei tohiks. Kaldume arvama, et tegemist on tõesti mõne koodis oleva veaga ja seega deklareerime, et teaduslike arvutuste jaoks praegune programm, oma kehva arvutustäpsuse tõttu, ei kõlba.

7 Esinenud probleemid edaspidiseks uurimiseks

Kuigi programmid said edukalt valmis ning töötavad ilma suuremate riketeta, siis esineb kohati väiksemaid rikkeid ja puudujääke.

Kõige ilmselgem probleem on võnkumiste programmides graafikutega. Kuigi eelmises punktis määrasime tingimuse, mille puhul graafikut joonestatakse, siis kahe punkti ühendamisel joonega seda tingimust osaliselt ignoreeritakse. Jooniselt (Joonis 7) on näha, et kui eelnev punkt asub graafiku kastist väljas, siis punkti ise ei joonestata, kuid ühendatakse sellest hoolimata ta ikkagi ära järgneva punktiga, mis asub graafiku kasti sees. Samuti on näha, et kui eelnev punkt asub graafiku kasti sees, aga järgmine asub juba kastist väljas, siis sellisel juhul kahte punkti joonega ei ühendata.

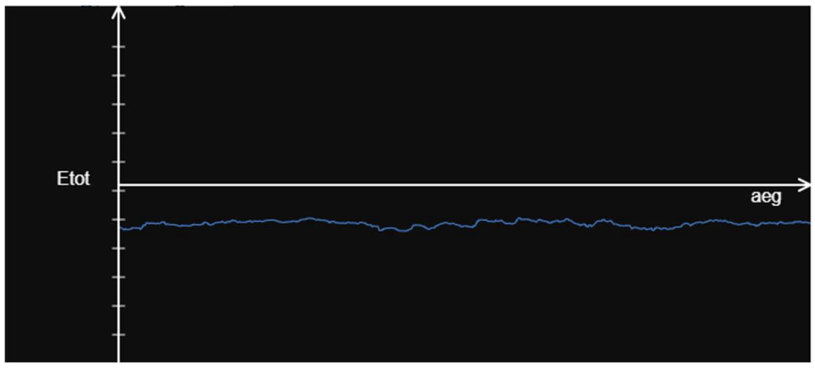


Joonis 7. Graafiku jooned kastist välja ronimas

Teine probleem, mis esineb on see, et graafikute kõrgused on kohati kas liiga madalad või liiga kõrged, seega kas ei mahu graafiku kasti ära või on väärtused liiga madalad et mingisuguseid muudatusi näha. Siinkohal oleks hea luua adaptiivne graafik, mis kõiki oma *y*-teljele kantavaid väärtusi kohandab vastavalt massiivi maksimum väärtusele. Ehk kui graafiku *y*-teljele kantav väärtuse maksimum on liiga väike, siis korrutatakse neid mõne suurema numbriga läbi ning kui väärtus on liiga suur, siis korrutatakse väiksema numbriga läbi. Antud number, millega y-teljele kantavate väärtustega massiivi kõiki elemente läbi korrutatakse võiks olla programmis automaatselt arvutatav iga iteratsiooni tagant ja kohandatud nii, et massiivi maksimum või miinimum väärtus langeks täpselt graafiku kasti kummagi ääre peale, ehk kas alla minimaalse väärtuse korral või ülesse maksimaalse väärtuse korral. Selline ettepanek teeks graafiku kasutajale veidi jälgitavamaks ning lahendaks ka esimese mainitud probleemi ära.

Tekkib ka kahtlus molekulaardünaamika simulatsiooni juures koguenergia graafiku puhul (Joonis 8). Nimelt on meil süsteem kinnine ja seega peaks koguenergia olema stabiilne, kuid meie programmis on energia hetkel liialt kõikuv. Kuigi programmis tehtavad arvutused on autori poolt juba korduvalt

üle vaadatud, siis ei teeks halba, kui mõni kolmas isik, kes värske pilguga arvutused üle vaataks ja potentsiaalse vea ülesse leiaks.



Joonis 8. Kõikuv koguenergia graafik

8 Kokkuvõtte

Töö peamiseks eesmärgiks oli luua kolme erineva füüsika ülesande jaoks veebivõrgu kasutajaliidesed, mis võimalikult täpselt visualiseeriks füüsika nähtuste käitumist, ning laadida need kasutamiseks TTÜ Parseki serverile. Selle jaoks loeti läbi vastav õppekirjandus ning kirjutati JavaScriptiga teoorial põhinevad programmid. Et tagada programmide töökindlust kohandati ja lihtsustati ka programmides teooriast tulenevaid valemeid, näiteks molekulaardünaamikas arvutatav raskuskiirendus.

Programmid visualiseerivad füüsilisi nähtusi enam vähem korrektselt ning kasutajaliidesed võimaldavad ka kasutajal simulatsiooniga mugavalt interakteeruda. Töö käigus esinenud suuremad tõrked said lahendatud, kuid väiksemad probleemid veel endiselt säilivad, näiteks graafiku joonte ilmumine animatsiooni kasti sisse ning molekulaardünaamika koguenergia arvulise väärtuse kõikumine. Kõik see viitab sellele, et kõik kolm kasutajaliidest said edukalt valmis ning olid laetud vaatamiseks ja kasutamiseks TTÜ Parseki serverile.

Kuna programmides esines üksikuid valemite lihtsustusi, siis ei ole soovitatav programme teaduslikel eesmärkidel kasutada vaid pigem demonstreerimiseks klassi- või loenguruumis.

Et programmid veelgi täiuslikumad oleksid, oleks vaja lahendada mõned väiksemad probleemid. Esmalt võiks läbi mõelda, kuidas teha nii, et graafiku jooned animatsiooni kasti sisse üle ei kanduks. Lisaks võiksid graafikud olla ka adaptiivsed, nii et nende kõige suurim või väikseim väärtus langeks täpselt graafiku kasti ääre peale. Samuti tuleks kontrollida üle koguenergia arvutused molekulaardünaamika programmis, kuna hetke seisuga on selle kõikumine liiga suur, mis ei ole loomulik.

9 Tänuavaldused

Soovin eelkõige tänada enda juhendajat, Mihhail Klopovit, kes oma rohkete kohustuste kõrvalt leidis alati aega minu suunamiseks ja aitas raskesti haaratavaid kontseptsioone lahti mõtestada. Samuti sooviks tänada kõiki TalTechi rakendusfüüsika õppekava õppejõudusid, kes on olnud minu õppeteel suurepärasteks eeskujudeks.

10 Kasutatud kirjandus

[1] Riigi Tugiteenuste Keskus. *Riik toetab digitaalse õppevara arendamist ja kasutuselevõttu 1,75 miljon euroga*. [www] https://www.rtk.ee/uudised/riik-toetab-digitaalse-oppevara-arendamist-ja-kasutuselevottu-175-miljoni-euroga. Kasutatud: 27.12.2022.

[2] I.Saveljev, *Füüsika üldkursus*, Tallinn: Valgus, 1978, pp. 169-174.

[3] D.Morin, „Damped harmonic motion," in *Introductory classical mechanics: with problems and solutions*, Cambridge: Cambridge University Press, 2012. [Online]. Doi: https://doi.org/10.1017/CBO9780511808951. Kasutatud: 27.12.2022.

[4] D.Halliday, R.Resnick and J. Walker, „Gravitation," in *Fundamentals of physics*, 10th ed, New York: Wiley, 2014. [E-book]. Loetud aadressil: https://elearn.daffodilvarsity.edu.bd/pluginfile.php/1057506/mod_label/intro/fundamentals-of-physics-textbook.pdf. Kasutatud: 27.12.2022.

[5] J.J Lissauer and I. de Pater, „Bound and Unbound Orbits," in *Fundamental Planetary Science: Physics, Chemistry and Habitability*, Cambridge: Cambridge University Press, 2019. [Online]. Doi: https://doi.org/10.1017/9781108304061.003. Kasutatud: 27.12.2022.

[6] L. Zhang, „Studying and comparing numerical methods for ordinary differential equations", Massachusetts Institute of Technology, USA, 2015. Leitav aadressilt: [Online] https://www.researchgate.net/publication/316266911_STUDYING_AND_COMPARING_NUMERICAL_METHODS_FOR_ORDINARY_DIFFERENTIAL_EQUATIONS. Kasutatud: 27.12.2022.

[7] J.H. Mathews ja K.D Fink, „Runge-Kutta Methods" in *Numerical Methods using MATLAB, 3rd ed,* New Jersey: Prentice Hall, 1999. [E-book]. Loetud aadressil: https://www.academia.edu/11293619/Numerical_Methods_using_MATLAB_Mathews_and_Fink. Kasutatud: 27.12.2022.

[8] E.Fehlberg, „Low-order classical Runge-Kutta formulas with stepsize Control and their application to some heat transfer problems", George C. Marshall Space Flight Center (MSFC), Marshall, Alabama, USA, 1969. Leitav aadressilt: [Online] https://ntrs.nasa.gov/api/citations/19690021375/downloads/19690021375.pdf. Kasutatud: 27.12.2022.

[9] D. Langer, „Efficient Verlet-List Implementation for KNL Processors", [Bakalaureusetöö], Department of Informatics, Technical University of Munich, München, Saksamaa, 2018. [Online]. Loetud aadressil https://mediatum.ub.tum.de/doc/1459358/1459358.pdf. Kasutatud: 27.12.2022.

[10] D. C. Rapaport, *The Art of Molecular Dynamics Simulation*, 2nd ed, Cambridge: Cambridge University Press, 2004, pp. 11-15.

[11] D. W. Ball and J. A. Key, „Kinetic Molecular Theory of Gases“ in *Introductory Chemistry,* Victoria, B.C: BCcampus, 2014. [E-book]. Loetud aadressil: https://opentextbc.ca/introductorychemistry/. Kasutatud: 27.12.2022.

[12] W. C. Swope and H. C. Andersen, „A Computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters“, *The Journal of Chemical Physics*, vol. 76, p. 637, 1982, doi: https://doi.org/10.1063/1.442716. Kasutatud: 27.12.2022.

[13] StackOverflow. *StackOverflow 2022 Developer Survey.* [www] https://survey.stackoverflow.co/2022/#section-most-popular-technologies-programming-scripting-and-markup-languages. Kasutatud: 27.12.2022.

[14] Netscape. *Netscape and Sun Announce JavaScript, the Open, Cross-Platform Object Scripting Language for Enterprise Networks and the Internet.* [www] https://web.archive.org/web/20070916144913/https://wp.netscape.com/newsref/pr/newsrelease67.html. Kasutatud: 27.12.2022.

[15] p5js. [www] https://p5js.org/. Kasutatud: 27.12.2022.

[16] R. Sun, Q. Wang and L. Guo, „Research Towards Key Issues of API Security“ in CNCERT 2021, Singapore, 2021, pp. 179-192. Loetud aadressil: https://link.springer.com/chapter/10.1007/978-981-16-9229-1_11. Kasutatud: 27.12.2022.

Lisa 1

```
import numpy as np
import matplotlib.pyplot as plt

sigma=1 #Angstroms
epsilon=1 # kJ/mol


def ljp_Force(r):
   return (2/r**6-1)*(r/r**8)

def ljp_Potential(r):
   return ((1/r)**12-(1/r)**6)


x=np.arange(0.5,6,0.01)

F=ljp_Force(x)

V=ljp_Potential(x)

plt.plot(x,F,color='blue', label='Jõud')
plt.plot(x,V, color='orange', label='Potential')
plt.legend()
plt.ylim([-0.3,0.5])
plt.show()

print(np.where(V==min(V)))
print("r=",x[62])
```

Lisa 2

Harmooniliste võnkumiste kood:
https://github.com/ArturRaag/Parsek_programmid/blob/main/Harmoonilised_vonkumised.js

Kepleri kood: https://github.com/ArturRaag/Parsek_programmid/blob/main/Kepler.js

Molekulaardünaamika kood:
https://github.com/ArturRaag/Parsek_programmid/blob/main/LJP_MD.js

Lisa 3

Link Parseki serverile: http://parsek.yf.ttu.ee/~eksleja/