eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
ZIENTZIA ETA TEKNOLOGIA FAKULTATEA
FACULTAD DE CIENCIA Y TECNOLOGÍA

Gradu Amaierako Lana
Ingeniaritza Elektronikako Gradua

N Gorputzen Simulazioa

Talkak jasaten dituzten gorputz masiboen higidura

Egilea:
Nerea Gurrutxaga Asiain
Zuzendaria:
Mikel Peñagaricano Badiola

Leioa, 2021eko Ekainaren 18a

Gaien Aurkibidea

1 Sarrera eta Helburuak

Orokorrean naturako problemak analitikoki ebaztea ezinezkoa denez, gero eta erabiliagoak dira simulazioak zientzian. Horrela, etengabeko elkarrekintzak jasaten dituzten partikulen higidura deskribatzea posible da. Lan honen interes berezia astrofisikan erabiltzen diren N gorputzen simulazioen lehen hurbilketa egitea izanik, fisikaren arlo honetan oso erabiliak diren grabitazio-simulazioak burutuko ditugu.

N gorputzen grabitate-simulazioen aurrekaria Erik Holmberg astronomoa izan zen. 1941. urtean galaxien arteko elkarrekintzak aztertzeko simulazioa burutu zuen ordenagailu optiko baten bitartez. Galaxia bakoitza deskribatzeko 37 bonbilla erabili zituen, eta fotozelulak eta galvanometro bat erabiliz, grabitate-indar simulatua neurtu zuen. Garapen teknologikoei esker, gaur egun milioika gorputzeko simulazioak burutzen dira milaka galaxien elkarrekintza aztertzeko [1]. Galaxiak izarrez eta bestelako gorputz zerutarrez osatutako gorputz-multzo erraldoiak izanik, bertako gorputzek elkar topo egitea oso gertaera inprobablea dela kontsideratzen da. Hortaz, gorputzen artean menderatuko duen elkarrekintza bakarra grabitatorioa dela kontsideratu ohi da. Eskala txikiagoetara jotzen badugu, ordea, posible da gorputzen arteko talkak ematea. Esaterako, milaka asteroide sartuko balira gure Eguzki-sistemara, zenbat gorputzek egingo lukete talka gure planetaren aurka? Are gehiago, nola perturbatuko lukete asteroideen ibilbidea euren arteko talkek? Horrelako problema batean, ez da nahikoa grabitatearen dinamikan oinarritzen den simulazioa burutzea.

Konkretuki landuko den problema, distantzia handietara grabitate-indarra eta distantzia laburretara talkak jasaten dituzten partikulen kasua da. Jarraian azalduko den eredu matematiko bakar bat erabiliz (2.2. atalean), eta eskura egon diren baliabide konputazionalekin, helburua N gorputzen problema konkretu honen ahalik eta algoritmo eraginkorrena lortzea izan da. Bikoteka elkarrekintzak jasaten dituzten partikulen problemetan, bikoteka indar guztiak kalkulatu eta batu behar dira, horrelako kode baten abiadura koadratikoki haziko delarik. N gorputz kopurua oso handia izateak kostu konputazional handia dakarrenez, kodea bektorizatzea funtsezkoa izan da.

Kodea inplementatzeko Python lengoaia aukeratu da, kode irekiko programazio lengoaia honen kalitate, produktibitate, eramangarritasun eta integraziorako optimizazioa direla medio [2]. Nahiz eta simulazioetarako lengoaia efizienteena, eta ondorioz, erabiliena Fortran izan, graduan zehar sendotu diren Pythonen inguruko ezagutzak aplikatzeko eta erabilera esparru berri bat lantzeko aukera lehenetsi da. Gainera, *NumPy* oso erabilia den Python lengoaiako paketeari esker, Python *Array* objektu multidimentsionalekin lan egiteko erabiltzen da, eta hainbat errutina ezberdin erabiltzeko aukera ematen du. Lan honetan gehienbat *array*-en arteko eragiketa aljebraiko linealak eta logikoak egin direnez, Python aukera ona izan da N gorputzen simulazioak lantzeko.

Sarrera honi amaiera emateko, lanaren abiapuntua zein izan den azpimarratu nahi da. Hasierako motibazioa Ilargiaren sorreraren egunera arteko hipotesirik sendoena simulatzen duen algoritmoa sortzea izan da; protolurra eta Theia planeten artean egon zen talka [3]. Kasu honetan, kodearen ikuspuntutik planeta bakoitza N masa txikiagotan banatzen da, eta

hasierako baldintza egokiak eta potentzia handiko konputazio-tresnak erabilita, sistemaren eboluzioa aztertu ahalko litzateke. Erreferentziatzat hartu den artikuluak proposatzen duen eredu matematikoaren ezegonkortasuna dela eta, hau baztertzea erabaki da. Saiakuntza ezberdinak eta gero, talkak deskribatzeko zenbait artikulutan proposatutako eredua jarraitzea aukeratu da [4, 5, 6]. Aurrerago ikusiko da simulazio bat burutzeko hasierako baldintzen fisikako azterketa sakona egin behar dela. Ilargiaren sorrera emango duen sistemaren konplexutasuna dela eta, analisi hau lan honen helburuetatik at geratzen da. Gainera, hasierako baldintza egokiak lortuko balira ere, eskura egon diren baliabide konputazionalen ezingo litzateke simulazioa gauzatu. Horregatik, sistema deskribatzeko erabiltzen den ereduaren algoritmoa inplementatu da, baina beste aplikazio batzuk jorratu dira.

2 Problemaren analisia

Analitikoki ebatzi ahal diren problemak linealak izan ohi dira. Naturako gertakari gehienak, ordea, ez dira linealak; aldagai baten aldaketa txiki batek beste aldagai baten aldaketa handiak eragin ditzake. Problema ez-lineal konkretu batzuk ebazpen analitikoa duten arren, orokorrean ebazpen analitikoak problema sorta murriztu batera mugatzen gaitu. Konputazioari esker edozein problema ez-lineal ebatzi daiteke. Horrez gain, problemen parametrizazio ahalbidetzen du; problemek askatasun graduak izan ditzakete, eta parametroen balioen arabera soluzio oso ezberdinak aztertu ahal dira [7]. Lan honetan aztertu nahi den problema ez-lineala izanik, beharrezkoa da konputaziora jotzea sistemaren denbora-eboluzioa aztertzeko. Gure probleman zentratu aurretik, lehenengo problema fisiko bat konputazionalki nola inplementatu daitekeen aztertuko da modu orokorrean.

Lehenengo eta behin, sistema modelizatu behar da. Pausu honetan sistemaren eboluzioan zehar aldatuko diren aldagaiak identifikatu behar dira. Horrez gain, aldagai hauen balioaren aldaketan eragina izango duten ezaugarriak identifikatu behar dira, gainerako efektu guztiak arbuiatu ahal izateko. Esaterako, ohiko tiro-parabolikoaren probleman, aldagaiak jaurti den objektuaren posizioa eta abiadura dira. Lurraren grabitate-indarra kontuan hartzen da, honek nabarmen aldatuko duelako jaurti den gorputzaren ibilbidea. Lurra errotatzen ari dela, ordea, arbuiatu egiten da, bi ekintzen denbora-eskalak oso ezberdinak baitira. Batez ere, sistemaren eboluzioan eragina izango duen fisika aztertzen da atal honetan.

Modelizazioan garrantzitsua da eredua ahalik eta orokorrena izatea. Horrela, inplementatutako kodea problema ezberdinak ebazteko gai izango da. Eredu berdinarekin deskribatzen diren problema batetik bestera simulazioa burutzeko erabaki beharreko bi parametro mota nagusi ezberdintzen dira:

- **Aldagai espezifikoak**: sistema deskribatzen duten parametroak dira eta ez daude lotuta eboluzioan zehar aldatuko diren aldagaiekin. Hala ere, exekuzioan zehar balioa alda dezakete. Balioaren aldaketa hau, ordea, ez dago zuzenean lotuta denboraren eboluzioarekin, eboluzioaren ondorioz gertatzen diren prozesuekin baizik. Tiro parabolikoaren adibidera itzuliz, objektuaren masa ereduko aldagai espezifiko bat da, problema espezifiko bakoitzaren arabera aldatuko dena. Masaren balio hau aldatu daiteke eboluzioan zehar, baina prozesu konkretu bat gertatu beharko da horretarako; esaterako, masa aldatu egingo da gorputza leherketa baten ondorioz bitan banatzen bada.

- **Hasierako baldintzak**: sistemaren eboluzioan zehar aldatuko diren aldagaien aldiune bateko balioak dira. Simulazio gehienetan hasierako aldiuneko balioak erabiltzen dira, hemendik aurrerako sistemaren eboluzioa kalkulatzeko. Tiro parabolikoaren adibidean, hasierako baldintzak posizioren eta abiaduraren koordenatuak dira.

Behin sistema deskribatzen duen eredua eta eboluzioa menderatuko duen fisika aztertuta, honen deskribapen matematikoa lortu behar da. Ereduak deskribatzen dituzten adierazpen matematikoak ekuazio diferentzialak izaten dira. n. ordenako ekuazio diferentziala eta n hasierako baldintza baditugu, problemaren soluzioa bakarra izango da [8]. Kasu honetan,

hasierako baldintzez hitz egiten da, denbora-eboluzioa kalkulatu nahi delako. Hortaz, ekuazio diferentzialetako deribatuak denborarekiko izango dira.

Ereduaren ekuazio diferentziala lortu denean, ekuazioa ebazteko erabiliko den metodoa aukeratu behar da ekuazioaren formaren eta ebazpenaren zehaztasunaren arabera [9]. Behin ebazpen metodoa aukeratuta, kodea idatzi daiteke.

2.1 Modelizazioa

Lan honetan talkak jasaten dituzten N gorputz masiboren sistema aztertuko da. Sistemaren denbora-eboluzioan eragina izango duten gorputzen ezaugarriak ondokoak dira:

- Gorputz bakoitzak dimentsio-espazial bakoitzeko posizio eta abiadura propioa du, erreferentziatzat hartuko den koordenatu-sistema kartesiar arbitrario batekiko. Problema hiru dimentsiotan ebatziko denez, gorputz bakoitzak denboran aldatuko diren sei aldagai ditu: $\vec{r}_i(t) = (x_i(t), y_i(t), z_i(t))$ posizioa eta $\vec{v}_i(t) = (v_{x_i}(t), v_{y_i}(t), v_{z_i}(t))$ abiadura.
- Gorputz bakoitzak m_i masa aldaezina du.
- Gorputz bakoitzak D_i diametroko bolumen esferiko aldaezina du $\vec{r}_i(t)$ posizioaren inguruan. Horrela, bi gorputzen arteko talka gertatuko den edo ez bereizi ahalko da (ikus 1. irudia). Bi gorputz ukitzen egongo dira euren zentroen arteko distantzia diametroen baturaren erdia baino txikiagoa bada, hau da, ondorengo baldintza betetzen bada: $|\vec{r}_i - \vec{r}_j| < (D_i + D_j)/2$.

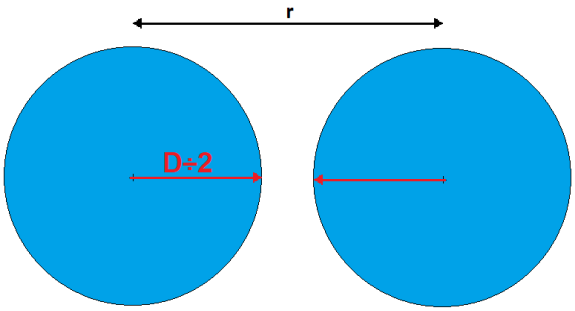
Problemaren fisikari dagokionez, partikulen arteko elkarrekintza partikula-bikoteka aztertuko da. Partikulak masadunak izanik, bi gorputz elkar ukitzen ari ez direnean grabitate-indarraren bitartez erakarriko dira. Newton-en grabitazio unibertsalaren legean oinarrituz, i eta j partikulen arteko indarra ondokoa da:

$$\vec{F}_{i,j} = G\frac{m_i m_j}{r_{i,j}^2}\hat{r}_{i,j} \tag{2.1}$$

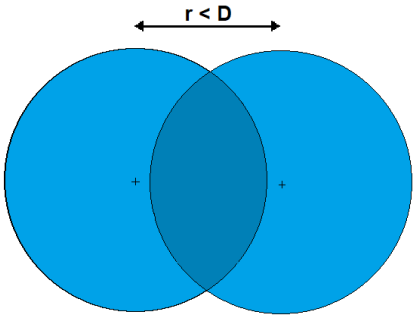
non $\hat{r}_{i,j} = \frac{\vec{r}_i - \vec{r}_j}{|\vec{r}_i - \vec{r}_j|}$ den, m_i eta m_j gorputzen masak, eta azkenik, G grabitazio unibertsalaren konstantea.

Berdin du simulazioak zein magnitude eskalatan egiten diren. Garrantzia duena magnitude eskalen arteko lotura da. Adibidez, luzera-unitatea = 1 m, denbora-unitatea = 1 s eta masa-unitatea = 1 kg badira, jakinik grabitazio unibertsalaren konstantea hiru magnitude hauen araberakoa dela, nahitaez bere balioa $G = 6.674 \cdot 10^{-11}\ \frac{\mathrm{N \cdot m^2}}{\mathrm{kg^2}}$ izango da. Hala ere, alderantzizkoa egitea ere posible da; G-ren balioa eta beste bi magnituderen eskalak finkatuz[1] gero (adb. masa eta distantziarena), hirugarren magnitudearen eskala finkatuta gertuko da (denborarena). Eskalak aukeratu ahal izateko G ereduaren aldagaia izango da.

[1] Eskalak finkatzea unitatea zein den adieraztea da. Esaterako, masaren kasuan masa-unitatea = 1 kg edota masa-unitatea = 8 kg izan daiteke, erabateko askatasuna dago.



(a) Bananduta dauden bi gorputz. Grabitate-indarraren ondorioz erakarri egingo dira.



(b) Ukitzen ari diren bi gorputz. Partikulen arteko talka gertatuko da.

1 Irudia: Bi partikula berdinen arteko elkarrekintza motak. $D = D_i = D_j$ partikulen diametroa eta $r = |\vec{r}_i - \vec{r}_j|$ zentroen arteko distantzia.

Bi gorputz elkar ukitzen ari direnean hauen arteko talka gertatuko da. Talkak deskribatzeko, ondorengo kontzeptu fisikoetan oinarritzen da eredua [10, 11]:

- n talka-parametroak talkaren elastikotasuna adieraziko du, hau da, talkan bero moduan galdutako energia kuantifikatzen du. Talka elastikoak ($n = 1$), inelastikoak ($0 < n < 1$) eta guztiz inelastikoak ($n = 0$) bereizten dira.

- Talka mota guztietan bi gorputzen momentu lineal totala kontserbatzen da[2]:

$$m_i\vec{v}_i + m_j\vec{v}_j = m_i\vec{v}'_i + m_j\vec{v}'_j \tag{2.2}$$

- Talka elastikoetan bi gorputzen energia zinetiko totala kontserbatzen da.

$$\frac{1}{2}m_i v_i^2 + \frac{1}{2}m_j v_j^2 = \frac{1}{2}m_i v'^2_i + \frac{1}{2}m_j v'^2_j \tag{2.3}$$

[2]Abiadura primatua (v') talka ondorengo abiadura adierazteko erabiltzen da.

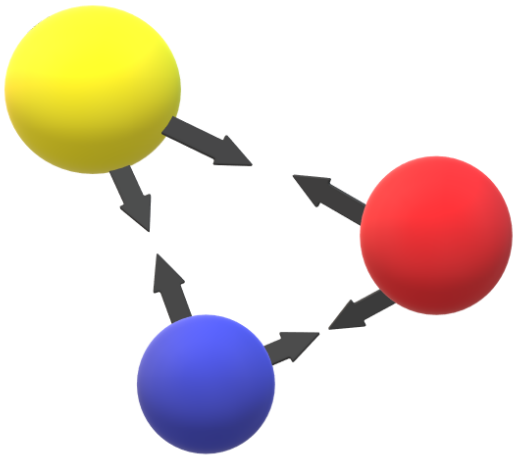
- Talka inelastikoetan ez da energia zinetiko totala kontserbatzen. Momentu lineal totala kontserbatzen denez, masa-zentroko energia zinetikoa, ordea, kontserbatu egingo da. Talka jasango duten i eta j gorputzen masa zentroko energia zinetikoa:

$$E_{MZ} = \frac{1}{2} M v_{MZ}^2 \tag{2.4}$$

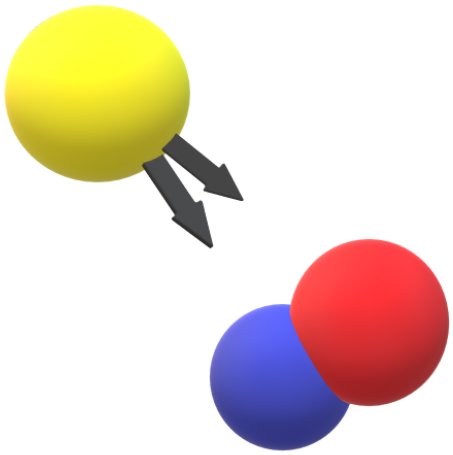
  non $M = m_i + m_j$ masa totala eta $\vec{v}_{MZ} = \frac{1}{M}(m_i\vec{v}_i + m_j\vec{v}_j)$ masa-zentroko abiadura diren.

- Talka guztiz inelastikoa bada, sistemaren amaierako energia zinetikoa masa-zentroko energia zinetikoaren berdina izango da.

- Talka gertatzen den aldiunean bi gorputzeko sistematik kanpo dauden indarrak arbuiatu egingo dira: $\vec{F}_{kanpo} \ll \vec{F}_{barne}$. Esaterako, i eta j gorputzen arteko talka gertatzen ari bada, w gorputzak duen eragina i eta j gorputzetan arbuiagarria da. i eta j gorputzek w-ren gainean izango duten eragina, ordea, ez da arbuiagarria izango (ikus 2. irudia).



(a) Grabitatearen ondorioz erakartzen diren hiru gorputz. Geziek indarrak erakarleak direla adierazten dute.



(b) Talka jasaten ari diren partikulek kanpoko partikularekin duten elkarrekintza arbuiagarria da. Kanpoko partikulak jasaten dituen indarrak, ordea, ez dira arbuiagarriak.

2 Irudia: Hiru partikulen arteko elkarrekintza.

Laburbilduz, sistemaren eboluzioan zehar aldatuko diren aldagaiak gorputzen posizioak eta abiadurak dira, eta hortaz, hasierako baldintzak $t = 0$ aldiuneko posizioak eta abiadurak izango dira. Sistemaren eboluzioan eragina izango duen fisika partikula bikoteen arteko elkarrekintza grabitatorioa eta talkak dira. Azkenik, sistemaren aldagai espezifikoak N partikula kopurua, hauen m_i masak, D_i diametroak, G grabitazio unibertsalaren konstantea eta n talka-parametroa dira.

2.2 Adierazpen matematikoa

Fisikako problema ugari ekuazio diferentzial arrunt edo partzialen bitartez adierazi daitezke. Gure eredua lehen ordenako ekuazio diferentzialen bitartez deskribatuko da, mugalde baldintzak hasierako aldiunean emango direlarik [9]:

$$\frac{dy(t)}{dt} = f(y(t), t) \qquad y(t=0) = y_0 \tag{2.5}$$

non $y(t)$ sistema osoa deskribatuko duen egoera-bektorea den. Bektore honen osagaiak denboran zehar aldatuko diren aldagaiak izango dira. Helburua $y(t)$ bektorearen denbora-eboluzioa numerikoki kalkulatzea da. Konputazionalki kalkulua egiteko t denbora-aldagaia diskretizatuko da, aldiune batetik besterako denbora-pausua h izanik:

$$t_{n+1} = t_n + h \tag{2.6}$$

t_n aldiuneko egoera-bektorea ezaguna izanik, t_{n+1} aldiuneko bektorea kalkulatzeko ondorengo ekuazioa ebatzi beharko da:

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(y(t'), t')\, dt' \tag{2.7}$$

$y(t_{n+1})$ kalkulatzeko $y(t)$ egoera-bektorearen eta $f(y(t), t)$ deribatuaren adierazpen matematikoak behar dira. Gure kasuan, denboran zehar aldatuko diren aldagaiak N partikulen posizioa eta abiadura direnez, bektoreen osagaiak ondokoak izango dira:

$$y(t) = (\vec{r}_1, \vec{v}_1, \vec{r}_2, \vec{v}_2, ..., \vec{r}_N, \vec{v}_N) \tag{2.8}$$

$$f(y(t), t) = \frac{dy(t)}{dt} = (\vec{v}_1, \vec{a}_1, \vec{v}_2, \vec{a}_2, ..., \vec{v}_N, \vec{a}_N) \tag{2.9}$$

Egoera-bektoreak eta bere deribatuak $6N$ osagai dituztenez, $6N$ ekuazio diferentzial izango ditugu. Modelizazioaren atalean deskribatu den ereduan oinarrituz, abiadura eta azelerazioaren adierazpenak lortu behar dira ekuazioen soluzioak bilatzeko. Aldagai hauen adierazpenak elkarrekintza motaren araberakoak izango dira:

2.2.1 Elkarrekintza grabitatorioa

Bi gorputz masadun erakarri egingo dira euren zentroen arteko distantzia erradioen batura baino handiagoa denean. Baldintza hau betetzen ez bada, bi gorputzek talka egingo dute, eta aldiune horretan gorputz hauen 12 ekuazioetan gainerako gorputzekin izango duten elkarrekintza grabitatorioa arbuiatuko da. Hortaz, atal honetan lortuko den adierazpena beste $N-1$ partikulekin talkarik jasaten ez duen gorputzarena izango da. i gorputzarengan j gorputzak eragingo lukeen indarra 2.1. ekuazioak deskribatzen du. Partikula batek jasaten duen indar totala partikula guztiek bere gainean eginiko indarren batura da. Hortaz, Newton-en 2. legea erabiliz, honela adierazi daiteke grabitatearen ondoriozko i partikularen azelerazioa:

$$\vec{a}_i = \frac{\sum_{j=1, j\neq i}^{N} \vec{F}_{i,j}}{m_i} = \sum_{j=1, j\neq i}^{N} \frac{G\, m_j}{r_{i,j}^2} \hat{r}_{i,j} \tag{2.10}$$

Partikulak bere buruarekin ez duenez elkarrekintzarik jasango N-1 batugai daude ($j \neq i$). 2.5. ekuazioko egoera-bektorearen deribatuan agertzen diren osagaiak partikula bakoitzaren azelerazioa eta abiadura izanik, azkenengoaren adierazpena lortzea falta da. 2.7. ekuazioa erabiliz:

$$\vec{v}_i(t_{n+1}) = \vec{v}_i(t_n) + \int_{t_n}^{t_{n+1}} \vec{a}_i(t')\, dt' \tag{2.11}$$

2.2.2 Bi gorputzen arteko talka

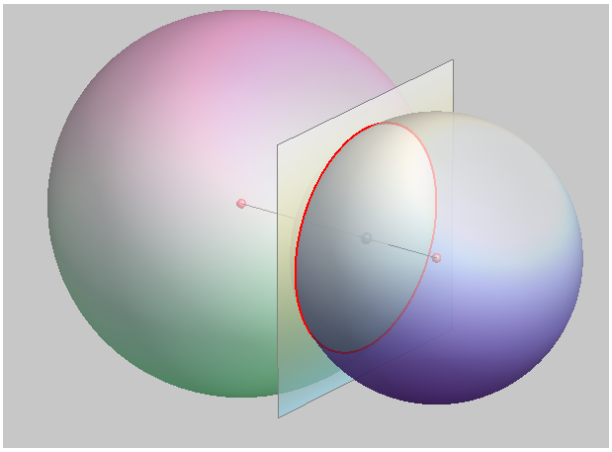
Bi gorputzen zentroen arteko distantzia hauen erradioen batura baino txikiagoa denean, gorputzen arteko talka gertatuko da. Erabiliko den ereduan [4, 5, 6], talkaren eragina gorputzen abiadura erlatiboaren aldaketaren bitartez deskribatuko da:

$$v_n^{'} = -n \cdot v_n \tag{2.12}$$

$$v_{t_1}^{'} = v_{t_1} \tag{2.13}$$

$$v_{t_2}^{'} = v_{t_2} \tag{2.14}$$

non abiadurak i eta j partikulen arteko abiadura erlatiboaren ($\vec{v} = \vec{v}_i - \vec{v}_j$) osagai normala ($\vec{v} \cdot \hat{u}_n = v_n$) eta tangentzialak ($\vec{v} \cdot \hat{u}_{t_1} = v_{t_1}$ eta $\vec{v} \cdot \hat{u}_{t_2} = v_{t_2}$) diren. n talka-parametroa da eta primak talka ondorengo abiadura dela zehazten du. Bi partikula esferikoren arteko talkan, abiadura erlatibo normalaren norabidea bi esferen zentroak lortzen dituen lerroaren berdina da. Abiadura erlatibo tangentzialak, berriz, ebakidura-planoan aurkitzen diren bektoreak dira (ikus 3. irudia).



3 Irudia: Bi partikula esferikoren arteko talka hiru dimentsiotan. Bektore normalaren norabidea bi esferen zentroak lortzen dituen lerroaren berdina da, eta ebakidura-planoko edozein bektore tangentziala izango da. Iturria: https://www.grad.hr/geomteh3d/prodori/prodor_sf_eng.html

i eta j partikulen arteko distantzia $\vec{r} = \vec{r_i} - \vec{r_j}$ eta talka aurretiko abiadura erlatiboa $\vec{v} = \vec{v_i} - \vec{v_j}$ izanik, abiaduraren osagai normala:

$$v_n = \vec{v} \cdot \hat{r} = \frac{1}{|\vec{r}|}(r_x v_x + r_y v_y + r_z v_z) \tag{2.15}$$

Ebakidura-planoa definitzeko, nahikoa dira planoaren bektore normala ($\hat{r}$) eta planoko puntu bat ($(0,0,0)$ puntua sinplifikazioagatik). Horrela, ondorengo ekuazioa betetzen duen edozein $\vec{v}_t = (v_x, v_y, v_z)$ abiadura erlatiboaren osagai tangentziala da:

$$r_x v_x + r_y v_y + r_z v_z = 0 \tag{2.16}$$

Hiru dimentsiotako problema ebazteko, linealki independenteak diren bi osagai tangentzial behar dira: $\vec{v}_{t1}$ eta $\vec{v}_{t2}$, non $\vec{v}_{t1} \cdot \vec{v}_{t2} = 0$ perpendikularrak direlako. Printzipioz edozein bi bektore perpendikular aukeratzeko askatasuna dago. Hala ere, kontuan izan behar da honen kodea inplementatu behar dela, eta ebazpen analitikoan agertzen ez diren arazoak eduki ditzakegula. Hau argiago ikusteko, demagun ondorengo bektore unitario tangentzialak hartzen ditugula:

$$\hat{v}_{t1} = \frac{1}{\sqrt{r_x^2 + r_y^2}}(-r_y, r_x, 0)$$

$$\hat{v}_{t2} = \frac{1}{\sqrt{r_x^2 r_z^2 + r_y^2 r_z^2 + (r_x^2 + r_y^2)^2}}(r_x r_z, r_y r_z, -r_x^2 - r_y^2)$$

Bi bektore hauek orain arte azaldutako baldintza guztiak betetzen dituzte, eta beraz, ez lukete arazorik eman beharko. Demagun, ordea, $D = 2$ diametroko i eta j partikulak ditugula, i partikularen posizioa $\vec{r}_i = (1,1,1)$ dela eta j partikularena $\vec{r}_j = (1,1,2)$. $\vec{r} = \vec{r}_i - \vec{r}_j = (0,0,-1)$ izanik, talka gertatuko da. Bektore unitario tangentzialak kalkulatzen baditugu:

$$\hat{v}_{t1} = \frac{1}{\sqrt{0}}(0,0,0)$$

$$\hat{v}_{t2} = \frac{1}{\sqrt{0}}(0,0,0)$$

Emaitzan ikus daiteke zeroak ditugula zatitzen, eta honek simulazioan nahi ez ditugun efektuak sortuko ditu. Inplementatu behar den kodeak kasu guztiak hartu behar dituenez kontuan, ekuazioak aukeratzeko dagoen askatasuna singularitate arazoak ekiditeko erabiliko da. Aurreko adibidean ikusi denez, osagai tangentzialetan r_x, r_y eta r_z osagai guztiak agertzea komeni da termino gurutzatuak erabili gabe. Azpimarratzekoa da $\vec{r} = (0,0,0)$ kasua inoiz ez dela gertatuko suposatuko dela problemaren ebazpena posible izateko. Onarpen honek zentzua galduko luke eskuz bi partikula bata bestearen gainean jarriko balira. Hau, ordea, zentzu fisikorik ez duen kasu bat besterik ez da. Hori dela eta, aukeratu diren bektore unitario tangentzialak ondokoak dira:

$$\hat{v}_{t1} = \frac{1}{\sqrt{2r_y^2 + (r_x + r_z)^2}}(r_y, -r_x - r_z, r_y)$$

$$\hat{v}_{t2} = \frac{1}{\sqrt{(r_y^2 + r_z^2 + r_x r_z)^2 + r_y^2(r_x - r_z)^2 + (r_x^2 + r_y^2 + r_x r_z)^2}}(-r_y^2 - r_z^2 - r_x r_z, r_y(r_x - r_z), r_x^2 + r_y^2 + r_x r_z)$$

Osagai normalak eta tangentzialak eskuartean izanik, ondorengo adierazpen orokorra erabiliko dugu abiadura erlatiboaren osagai kartesiarrekin erlazionatzeko:

$$\begin{pmatrix} v_n \\ v_{t1} \\ v_{t2} \end{pmatrix} = \mathbb{P} \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \tag{2.17}$$

non $\mathbb{P}$ jada kalkulatu den oinarri ortonormala den.

$$\mathbb{P} = \begin{pmatrix} \hat{v}_n \cdot \hat{i} & \hat{v}_n \cdot \hat{j} & \hat{v}_n \cdot \hat{k} \\ \hat{v}_{t_1} \cdot \hat{i} & \hat{v}_{t_1} \cdot \hat{j} & \hat{v}_{t_1} \cdot \hat{k} \\ \hat{v}_{t_2} \cdot \hat{i} & \hat{v}_{t_2} \cdot \hat{j} & \hat{v}_{t_2} \cdot \hat{k} \end{pmatrix} \tag{2.18}$$

$\mathbb{P}$ oinarri ortonormalean idatzita dagoenez, matrizearen alderantzizkoa bere iraulia da: $\mathbb{P}^{-1} = \mathbb{P}^T$. Hortaz, talka ondorengo abiadura erlatiboaren osagai kartesiarrak:

$$\begin{pmatrix} v'_x \\ v'_y \\ v'_z \end{pmatrix} = \mathbb{P}^T \begin{pmatrix} -n \cdot v_n \\ v_{t_1} \\ v_{t_2} \end{pmatrix} \tag{2.19}$$

v'_n, v'_{t1}, eta v'_{t2} idatzi ordez, zuzenean 2.12., 2.13. eta 2.14. ekuazioak ordezkatu dira.

Eragiketa honekin v'_x, v'_y eta v'_z osagaiak kalkulatu ondoren, partikula bakoitzari dagokion talka ondorengo abiadura kalkulatu behar da. Printzipioz sei ezezagun ($\vec{v}'_i$ eta $\vec{v}'_j$) eta hiru ekuazio ($\vec{v}' = \vec{v}'_i - \vec{v}'_j$) ditugu. Talka guztietan momentu lineala kontserbatzen dela jakinik, beste hiru ekuazio lortzen dira:

$$m_i\vec{v}_i + m_j\vec{v}_j = m_i\vec{v}'_i + m_j\vec{v}'_j \tag{2.20}$$

Guztia era matrizialean jarriz, partikula bakoitzaren talka ondorengo abiadurak lortzeko ondorengo sistema askatu behar da:

$$\begin{pmatrix} m_i & m_j & 0 & 0 & 0 & 0 \\ 0 & 0 & m_i & m_j & 0 & 0 \\ 0 & 0 & 0 & 0 & m_i & m_j \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} v'_{i_x} \\ v'_{j_x} \\ v'_{i_y} \\ v'_{j_y} \\ v'_{i_z} \\ v'_{j_z} \end{pmatrix} = \begin{pmatrix} m_i v_{i_x} + m_j v_{j_x} \\ m_i v_{i_y} + m_j v_{j_y} \\ m_i v_{i_z} + m_j v_{j_z} \\ v'_x \\ v'_y \\ v'_z \end{pmatrix} \tag{2.21}$$

Atal honetan, talkak jasaten dituzten gorputzen abiadura nola aldatzen den aztertu da. Hau nahikoa da 2.8. ekuazioko osagai guztiak kalkulatzeko, eta beraz, ez da beharrezkoa azelerazioa kalkulatzea.

2.3 Ebazpen metodoak

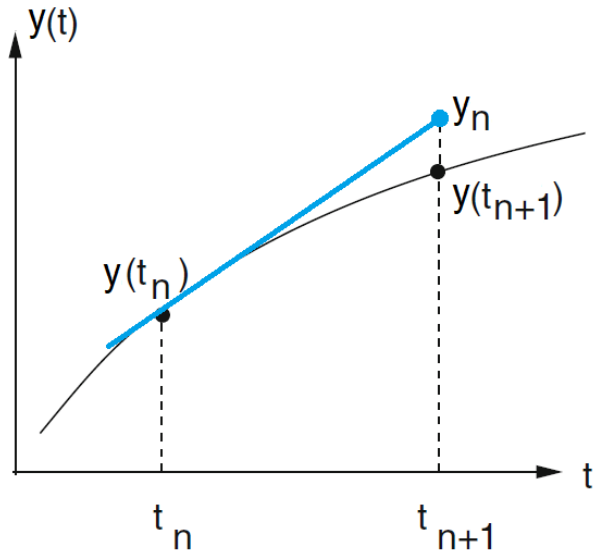
Behin ekuazioak idatzita, ebazpen metodoa aukeratu behar da. Emaitzan nahi den zehaztasunaren arabera, metodo ezberdinekin ebatzi daiteke problema. Hainbat algoritmo ezberdin existitzen dira sistema baten denbora-eboluzioa kalkulatzeko. Hauen artean bi metodo landu dira: Euler-en metodo esplizitua eta Leapfrog algoritmoa [9]. Bi metodoak Taylor-en garapenean oinarritzen dira funtzioaren hurbilketa bat lortzeko:

$$y(t+h) = y(t) + y'(t) \cdot h + y''(t) \cdot \frac{h^2}{2!} + y'''(t) \cdot \frac{h^3}{3!} + ... \tag{2.22}$$

2.3.1 Euler-en algoritmoa

Euler-en metodo esplizituan Taylor-en garapen osoaren lehenengo gaia bakarrik hartzen da kontuan. Ekuazio diferentziala $y'(t) = f(y(t), t)$ izanik (ikus 2.5. ekuazioa), ondorengo algoritmoa inplementatu behar da:

$$y_{n+1} = y_n + f(y_n, t_n) \cdot h + O(h^2) \tag{2.23}$$



4 Irudia: Euler esplizituaren deskribapen grafikoa. y_n erabiliz y_{n+1} kalkulatzen da pausu bakarrean.

Euler metodoaren errore lokala, hau da, pausu bakoitzean emango dena, $O(h^2)$ ordenakoa da. Denbora-tarte osoan emango den errore globala, berriz, $O(h)$ ordenakoa da. Denbora-tarte osoa t eta iterazio kopurua *loop* izanik:

$$\text{Errore totala} = loop \cdot h^2 = \frac{t}{h} \cdot h^2 = \mathrm{t} \cdot h \tag{2.24}$$

Euler-en metodoa inplementatzea oso erraza izan arren, hartu beharreko h pausua nahi den zehaztasunaren ordenakoa izan behar da. Adibidez, 10^4-ko zehaztasuna nahi bada, 10 000 pausu egin behar dira denbora-unitate batean.

Gure probleman ondorengo moduan inplementatuko da Euler-en metodoa:

$$\vec{r}_{n+1} = \vec{r}_n + \vec{v}_n \cdot h \tag{2.25}$$

$$\vec{v}_{n+1} = \vec{v}_n + \vec{a}_n \cdot h \tag{2.26}$$

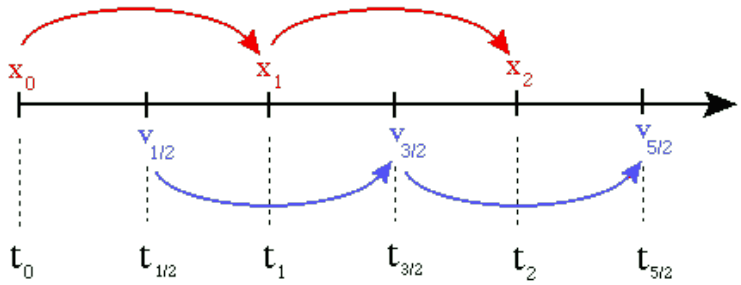
non $\vec{r} = (x, y, z)$ partikulen posizioaren koordenatuak, $\vec{v} = (v_x, v_y, v_z)$ partikulen abiadurak, $\vec{a} = (a_x, a_y, a_z)$ azelerazioak eta h denbora-pausua diren. Azpi-indizeak aldiunea adierazten

du: n azpi-indizeak t_n aldiuneko aldagaia dela adierazten du, eta $n+1$-ek t_{n+1} aldiunekoa.

Talken ebazpenak zentzua izateko, $\vec{r}_{n+1}$ kalkulatu ondoren talkarik gertatzen den edo ez egiaztatu behar da. Talkarik ez badago, $\vec{v}_{n+1}$ 2.26. ekuazioa erabiliz kalkulatuko da. Talka detektatzen bada, ordea, $\vec{v}_{n+1} = \vec{v}'_n$ talka ondorengo abiadura izango da.

2.3.2 Leapfrog algoritmoa

Leapfrog metodoan $\vec{r}_{n+1}$ posizioa kalkulatzeko, $\vec{r}_n$ aurreko pausuko posizioa eta $\vec{v}_{n+\frac{1}{2}}$ tarteko pausuko abiadura erabiltzen dira. $\vec{v}_{n+\frac{1}{2}}$ kalkulatzeko, berriz, $\vec{v}_{n-\frac{1}{2}}$ eta $\vec{r}_n$ erabiltzen dira. Dimentsio batean x_n kalkulatzeko prozedura 5. irudian deskribatzen da.



5 Irudia: Leapfrog metodoaren deskribapen grafikoa. Lehenengo $v_{n-\frac{1}{2}}$ eta x_n erabiliz $v_{n+\frac{1}{2}}$ kalkulatzen da. Ondoren, x_n eta $v_{n+\frac{1}{2}}$ erabiliz, x_{n+1}. Iturria: http://www.physics.drexel.edu/~steve/Courses/Comp_Phys/Integrators/leapfrog/

h denbora-pausua izanik, $n+\frac{1}{2}$ aldiuneko abiadura eta $n+1$ aldiuneko posizioak kalkulatzeko ondorengo ekuazioak erabiltzen dira:

$$\vec{v}_{n+\frac{1}{2}} = \vec{v}_{n-\frac{1}{2}} + \vec{a}_n \cdot h \tag{2.27}$$

$$\vec{r}_{n+1} = \vec{r}_n + \vec{v}_{n+\frac{1}{2}} \cdot h \tag{2.28}$$

$\vec{v}_n$ ez bada agertzen ekuazioetan, metodo honen errore lokala $O(h^3)$ ordenakoa da. Errore globala, berriz, $O(h^2)$ ordenakoa da. Metodo honen bitartez ezin da $\vec{v}_n$ zuzenean lortu. Horretarako, zehaztasunean eragin negatiboa izango duten aldaketa batzuk egin behar dira:

$$\vec{v}_n = \frac{\vec{v}_{n+\frac{1}{2}} + \vec{v}_{n-\frac{1}{2}}}{2} + O(h^2) \tag{2.29}$$

Leapfrog algoritmoa oso erabilia da grabitate-simulaziotan, Newtonen grabitazio unibertsalaren legeko ekuazioetan ez baita abiadura agertzen. Hortaz, Euler-en metodoarekin egin beharreko pausu kopuru berdinak egiten dira, baina zehaztasuna hobeagoa lortuta. Hala ere, metodo honek kontuan hartu beharreko desabantaila bat du. Hasierako baldintzak ez

dira aldiune berekoak, $\vec{r}_0$ eta $\vec{v}_{\frac{1}{2}}$ baizik. t_0 aldiuneko abiadura erabiliz $t_{\frac{1}{2}}$ aldiunekoa kalkulatzeko, Euler-en ataleko 2.26. ekuazioa erabili behar da[3] talkarik ez badago:

$$\vec{v}_{\frac{1}{2}} = \vec{v}_0 + \vec{a}_0 \cdot \frac{h}{2} \tag{2.30}$$

Talka badago t_0 aldiunean:

$$\vec{v}_{\frac{1}{2}} = \vec{v}'_0 \tag{2.31}$$

[3] h-ren ordez $\frac{h}{2}$ ordezkatu behar da, denbora-pausu erdia kalkulatzen ari gara.

3 Kodearen Inplementazioa

Behin problemaren garapen matematikoa eta ebazteko algoritmoa aztertuta, kodearen inplementazioa burutu daiteke. Horretarako, kontuan izan behar da ondoren kodea partikula kopuru handiekin erabiliko dela, eta hortaz, ezinbestekoa dela kodea efizientea izatea. Horretarako, kodearen bektorizazioa landu behar da.

Kodea inplementatu denean, kodea zuzena dela egiaztatu behar da. Kodea zuzena kontsideratuko da, exekuzioak esperotako emaitza itzultzeaz gain, exekuzio-denbora minimizatzea lortu denean. Partikula asko daudenean emaitza zuzena den edo ez egiaztatzea zaila denez, egiaztapen-prozesu gehienak partikula kopuru oso murriztu batekin egin daitezke.

3.1 Kodearen bektorizazioa

Gaur egun existitzen diren muga konputazionalak direla medio, kodea inplementatzean honen efizientziak garrantzi handia du. N gorputzen problemetan, datu multzo handiekin egiten da lan, eta elkarrekintzak bikoteka aztertzean, N partikulek N-1 partikulekin duen elkarrekintza aztertu behar da denbora-pausu bakoitzean. Ondorioz, exekuzio-denbora koadratikoki haziko da. Kostu konputazionala murrizteko, kode bektorizatuekin egiten da lan [12]. Kode hauen funtsa, ahal den neurrian kontrol egiturarik gabeko funtzioak eraikitzea da, hau da, if, for eta $while$ sententziarik gabeko kode zatiak lortzea. Horrela, eragiketak elementuz elementu burutzea ekidingo da.

Kodea bektorizatzeko, Pythonek eskaintzen duen konputazio zientifikorako **NumPy** paketea erabili da [13], NumPy klaseko array batek Pythoneko ohiko lista batekin alderatuz dituen abantailak direla eta. Numpy klaseko array-ek eragiketa matematikoak modu efizienteagoan exekutatu eta kodea sinplifikatzea ahalbidetzen dute. Listekin, aldiz, eragiketak elementuz elementu burutzen dira, hauek zeharkatzeko begiztak beharko direlarik. Zergatik dira azkarragoak, ordea, NumPy klaseko objektuak listak baino? Kode bektorizatuan begiztak eta indexazioak ez dira esplizituki agertzen. Jakina, inplizituki gertatzen den zerbait da, baina aurretik konpilatutako C edo Fortran lengoaian edota mikroprozesadorearen instrukzio natiboan idatzitako kode optimizatu batean. Elementuz elementu eragiketak inplizituki burutzeari *Broadcasting* deritzo. NumPy motako bi objekturen arteko eragiketa aritmetiko zein logikoetan, array txikiena 'hedatu' egiten da handienaren tamainara.

Kode bektorizatuaren eta bektorizatu gabekoaren arteko ezberdintasuna argiago ikusteko, ondorengo adibide sinplea inplementatu da: Izan bedi $a1$, $b1$ eta $c1$ Pythoneko ohiko hiru lista, eta $a2$, $b2$ eta $c2$ NumPy klaseko hiru 30 milioi elementuko array-ak. c, a eta b array-en batura bada, kode bektorizatu gabea ondokoa litzateke:

```
a1 = [1, 2, 3]*10000000
b1 = [-1, 0, 4]*10000000
c1 = list()
for i in range(0, len(a1)):
    c1.append(a1[i] + b1[i])
```

Saiakuntza ezberdinak eta gero, kode honen exekuzioak 7.5 s behar izan ditu batez bestean. Bektorizatutako kodea ondorengoa litzateke:

```
a2 = np.array([1, 2, 3]*10000000)
b2 = np.array([—1, 0, 4]*10000000)
c2 = a2 + b2
```

Aurreko kodea exekutatzeko erabilitako erreminta berdinekin[4], kode bektorizatuaren exekuzio-denbora batez bestean 0.104 s izan da. Kodea sinplifikatzeaz gain, ikus daiteke exekuzio-denboran alde handia dagoela tamaina handiko array-ekin lan egiten dugunean. 30 milioi elementuko array-ak beharrean, 300 milioi elementukoak badira, exekuzio-denborak ~ 70 s eta 1.04 s izan dira hurrenez hurren.

3.2 Kodearen eraikuntza eta egiaztapena

Talkak jasaten dituzten N gorputz masiboren higidura ebazteko idatzi den kodea hiru bloke handitan banatu da. Batetik, grabitatearen ondoriozko partikulen azelerazioa kalkulatzen duen funtzioa idatzi da (ikus A. eranskina). Bestetik, partikulen arteko talkak detektatu eta ebazteko funtzioak sortu dira fitxategi batean (ikus B. eranskina). Azkenik, bi programa hauek inportatuko dituen programa nagusiak eraiki dira. Landutako Euler eta Leapfrog algoritmoak azken hauetan inplementatu dira.

3.2.1 Grabitatea eta Eguzki-sistema

Bi partikulen artean elkarrekintza grabitatorioa dagoenean, 2.1. ekuazioaren bitartez deskribatzen da bi gorputzen arteko indarra. Talkarik ez badago, hau da, bi gorputzen arteko distantzia beren erradioen batura baino handiagoa bada, elkarrekintza hau izango dugu. Partikula bakoitzaren azelerazioa kalkulatzeko bektorizatutako funtzio bakar bat inplementatu da (ikus A. eranskina).

Funtzioa egiaztatzeko, gure Eguzki-sistema simulatu da. Sisteman zortzi planeta eta beste zenbait gorputz zerutar Eguzkiaren inguruan orbitatzen ari dira. Gorputz bakoitzak Eguzkiaren inguruan orbita bat burutzeko behar duen denbora Keplerren legeak erabiliz kalkulatu daiteke. Lege hauen bitartez periodoa kalkulatzean, gainerako planeta eta gorputzen eragina arbuiatzen da, hau da, Eguzkia eta gorputzak osatzen duten sistema bitarraren problema ebazten da [11]. Horrela, posible da periodoa analitikoki kalkulatzea. Emaitzak behaketekin konparatzean, hau hurbilketa ona dela ikusi izan da.

Gure programa gai izango da elkarrekintza guztiak kontuan hartuta periodoak kalkulatzeko. Eguzkia, zortzi planeta[5] eta Pluton planeta nanoaren simulazioko burutuko da, guztien ezaugarriak ezagunak baitira. Hortaz, $N = 10$ gorputzekoa izango da simulazioa. Atal honen helburua, simulazioan hasierako baldintza batzuk ezarriz, gorputz bakoitzak orbita itxia

[4]Simulazioa egiteko ordenagailu eramangarri arrunt bat erabili da. Interesgarria exekuzio-denboren arteko ezberdintasuna da, balio zehatzak baino.

[5]Merkurio, Artizarra , Lurra, Marte, Jupiter, Saturno, Urano eta Neptuno.

egiteko behar duen denbora aurresatea da. Balioak egiaztatzeko, lortutako datuak astrofisikako liburuetako datuekin konparatuko dira. Eguzki-sistemako gorputz handienen masak ondokoak dira:

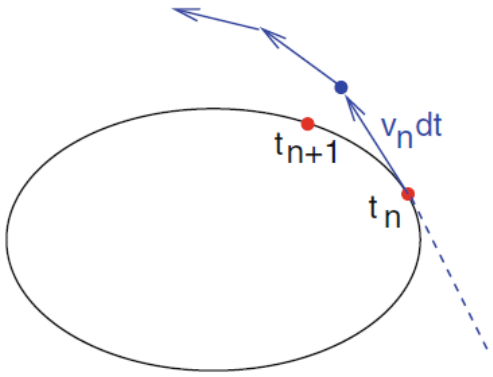
Gorputza	Masa ($M_\oplus$)
Eguzkia	333000
Merkurio	0.05528
Artizarra	0.81500
Lurra	1.0000
Marte	0.10745
Jupiter	317.83
Saturno	95.159
Urano	14.536
Neptuno	17.147
Pluton	0.0021

1 Taula: Eguzki-sistemako gorputzen masak [14]. Unitatea Lurraren masa da: $M_\oplus = 5.972 \cdot 10^{24}$ kg.

Problemaren zailtasuna hasierako baldintza egokiak aurkitzean egon da. Objektu zerutarren posizioa adierazteko astronomian erabiltzen den koordenatu-sistema ohikoena, Lurrarekiko posizioa neurtzen duen koordenatu-sistema ekuatoriala da. Koordenatu-sistema honetatik abiatuz simulazioko koordenatu-sistema kartesiarrera aldatzeko zailtasuna lan honen helburuetatik kanpo dagoenez, aukera hau baztertu egin da. Hasierako baldintzak lortzeko, *Skyfield* biblioteka erabili da [15]. Biblioteka honek kodea exekutatzen ari garen aldiuneko izarren, planeten eta sateliteen posizioak eta abiadurak ematen ditu koordenatu kartesiarretan, Estatu Batuetako Itsas Armadako Behatokiko datuetan oinarrituta. Datu hauen zehaztasuna 0.0005 $''$-koa[6] da eta jada aipatutako *NumPy* biblioteka erabiltzen du eragiketak eraginkorragoak izateko.

Problema ebazteko aukeratu den algoritmoa LeapFrog izan da. Euler-en metodoa ez da erabiltzen gorputzen orbitak kalkulatzeko, metodo honen errore globala $O(h)$ ordenakoa izanik, oso pausu txikia hartu beharko litzatekeelako zehaztasun onargarri bat lortzeko (ikus 6. irudia). Leapfrog algoritmoa inplementatu denez, 2.3. atalean aipatutako hasierako abiaduraren arazoa konpondu behar dela gogoratu behar da. Erabili den Skyfield bibliotekak gorputzen hasierako (exekuzio-aldiuneko) posizioak eta abiadurak emango ditu, hau da, $\vec{r}_0$ eta $\vec{v}_0$. Leapfrog algoritmoa erabiltzeko, ordea, $\vec{r}_0$ eta $\vec{v}_{\frac{1}{2}}$ hasierako baldintzak behar dira. Horregatik, beharrezkoa da denboraren garapena kalkulatzen hasi aurretik 2.30. ekuazioko konbertsioa egitea. Simulazioaren interesa gorputzen orbiten periodoen kalkulua denez, denbora-eboluzioa kalkulatzen hasi aurretik hasierako posizioak gorde behar dira. Horrela, gorputzak hasierako posiziora itzultzean periodo bat bete dutela detektatu ahalko da.

[6] $1'' = \frac{1}{3600}^\circ$



6 Irudia: Euler-en metodoaren bitartez orbitak kalkulatzerakoan, pausu bakoitzean egindako $O(h^2)$ ordenako errorea metatu egiten da, errore globala edo totala $O(h)$ ordenakoa izanik [9].

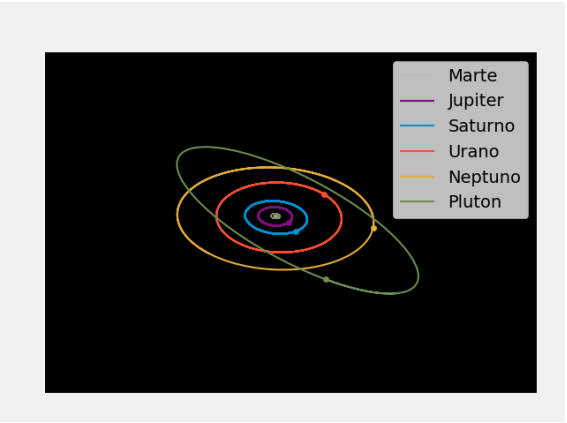
Gorputza	Simulatutako P (urte)	Benetako P (urte)
Merkurio	0.241	0.241
Artizarra	0.614	0.615
Lurra	1.000	1.000
Marte	1.879	1.881
Jupiter	11.866	11.862
Saturno	29.441	29.457
Urano	84.019	84.011
Neptuno	164.849	164.79
Pluton	248.011	247.68

2 Taula: Eguzki-sistemako gorputzen periodo simulatuak eta bibliografiakoak [14]. Simulazioa egunetan egin da eta urtetarako konbertsioa egiteko simulatutako Lurraren periodoarekin zatitu da.

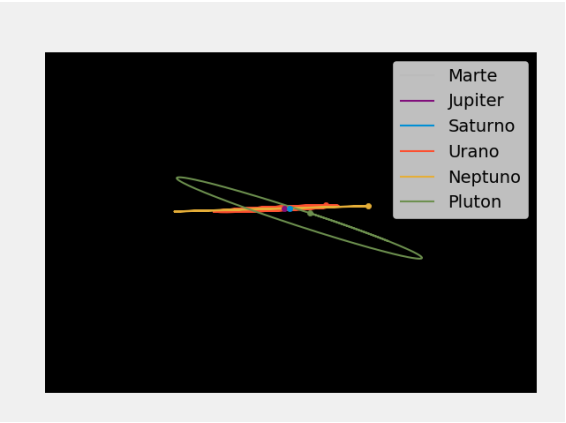
Orain artekoa kontuan hartuta, programa nagusia idatzi da (ikus A.2). Programaren exekuzioa hasi aurretik, h denbora-pausuaren balioa aukeratu da: h = 1 egun = 0.0027 urte. Denbora-pausu honekin, exekuzioak 14.11 s behar izan ditu. 2. taulako bigarren zutabean simulazioaren bitartez lortutako periodoak adierazi dira, eta hirugarren zutabean, berriz, astrofisikako liburuetatik [14] lortutako datuak. Taulan ikus daitekeenez, emaitza hobeak lortu dira periodo txikiagoa duten gorputzen kasuan. Gorputz baten periodoa handiagoa bada, iterazio gehiago beharko ditu programak honen periodoa kalkulatzeko, eta errore handiagoa metatuko da. Errore totala $= P \cdot h^2$ izanik (ikus 2.24. ekuazioa), adibidez, Pluton planeta nanoaren periodoaren errorea $\Delta P_{Pluton} = 0.670$ urtekoa da. Balio zehatzagoak lortzeko h denbora-pausua txikitu beharko litzateke, kontuan izanik honek simulazioaren denbora-exekuzioa luzatuko lukeela.

Simulazio hau hiru dimentsiotan egin denez, planetek Eguzkia eta Lurraren arteko plano

ekliptikoarekiko inklinazio bat dutela behatu daiteke. Planetak plano ekliptikotik oso gertu daude, baina 7(b). irudian ikus daiteke Plutonen orbitak inklinazio nabaria duela.



(a) Puntu lodiek simulazioa egin den eguneko planeten posizioa adierazten dute (2021/05/14).



(b) Plutonen orbitaren inklinazioa plano ekliptikoarekiko.

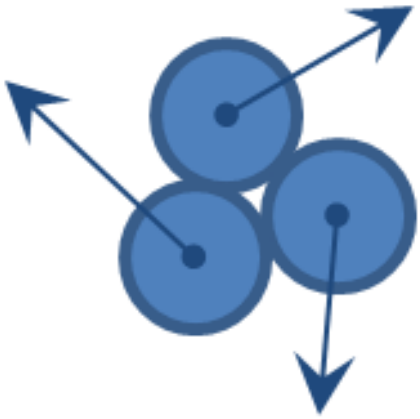
7 Irudia: Eguzki-sistemako planeten eta Plutonen orbiten simulazioa.

3.2.2 Talkak

Bi partikulen artean talka gertatzen denean, 2.12., 2.13. eta 2.14. ekuazioen bitartez deskribatzen da bi gorputzen arteko elkarrekintza. Talkaren inelastikotasunaren arabera n talka-parametroak $0 \leq n \leq 1$ tarteko balio bat hartuko du; $n = 1$ talka elastikoa bada, eta $n = 0$ talka guztiz inelastikoa bada.

Talkak ebazteko inplementatu den kodearen zatia grabitatearena baino konplexuagoa izan da. Lehenengo, denbora-pausu bakoitzean zein gorputzen arteko talka gertatzen den identifikatu behar da. Ez du zentzu fisikorik hiru gorputzen arteko talka aldiune berean gertatzeak, errealitatean beti egongo baitira bi gorputz hirugarrena iritsi baina lehenago talka egingo dutenak (ikus 8. irudia). Hau ez gertatzeko, h denbora-pausu txikia aukeratu behar da. Hala ere, badaude kasuak zeinetan hiru gorputzak elkar talka egingo dutela detektatuko duen programak. Hortaz, arazo honi aurre egiteko gai izan beharko da kodea.

Kodean eskuz sarturiko lehentasunik ez egoteko, lehenengo talka-bikote guztiak identifikatuko dira eta hauek lista batean gorde. Lista honetan dauden talkak ausazko ordenan exekutatuko dira. Esaterako, demagun i, j eta w gorputzen arteko aldibereko talka detektatu dela. Gure listan hiru talka gordeko dira; i eta j-ren artekoa, i eta w-ren artekoa, eta j eta w-ren artekoa. Lista modu sasiausazkoan ordenatuko da. Adibidez, lehenengo i eta w-ren arteko talka ebazten da. Hortaz, partikula hauen abiadurak aldatu egingo dira. Ondorengoa i eta j partikulen arteko talka bada, talka hau ebaztean j partikulak talken exekuzioa hasi aurretiko abiadura du, baina i partikulak w-rekin izandako talka eta geroko abiadura izango du. Exekuzioa talken sekuentzia bat burutuz egingo denez, ezin izango da kodea guztiz bektorizatu grabitatearekin egiten zen moduan; 2. talka ezingo da exekutatu 1.



8 Irudia: Hiru gorputzen talka aldiune berdinean. Hau errealitatean ez da inoiz gertatuko. Iturria: https://sites.google.com/site/hjwhitney/portfolio/gravitational-n-body-problem-with-elastic-collisions

talka ebatzi arte. Ebazteko modu honek kostu konputazionala nabarmen igoko duen arren, emaitza errealistagoa izatea lortzen da. Inplementatu den kodea B. eranskinean ageri da.

Talka-parametro ezberdinak Talka-parametroaren balioa $n = 1$ ezartzean, gorputzen arteko talka elastikoa gertatuko da. Horrelako talka batean, bi partikulen energia zinetiko totala kontserbatu behar da. $0 < n < 1$ denean, talka ondorengo energia zinetikoa hasierakoa baino txikiagoa izango da, baina masa-zentroarena baino handiagoa (ikus 2.4. ekuazioa). $n = 0$ denean, berriz, talka ondorengo energia zinetiko totala masa-zentroarena izango da. Gure kodea zuzena izan behar da n-ren edozein baliotarako.

Bi partikulen arteko talka exekutatu da $n = 1$, $n = 0.9$, $n = 0.4$ eta $n = 0$ balioetarako, eta 9(a). irudiko energia zinetikoak lortu dira (ikus B.4. eranskina). Kodearen egiaztapenean grabitatea kontuan ez denez hartu, talkaren unean izan ezik, abiadura konstantez higitzen dira partikulak. Hortaz, Euler eta Leapfrog algoritmoa berdinak dira kasu espezifiko honetan. [4] artikuluan aipatzen den moduan, lortutako emaitzak h denbora-pausuaren mendekotasun handia duela ikusi da. h handiegia ezartzen badugu, baliteke partikulen bideak gurutzatzea baina kodeak hau ez detektatzea, pausu batetik bestera egindako distantzia diametroa baino handiagoa delako. Partikulak oso azkar gerturatzen badira, denbora-pausu txikiagoa beharko dugu distantzia-aldaketa nahikoak detektatzeko. h denbora-pausua aukeratzeko ondorengo irizpidea proposatzen da:

Demagun gerturatzen ari diren i eta j partikulak aztertzen ari garela eta bien elkarrekintza dimentsio batean deskribatu daitekeela. Bi partikulen zentroen arteko distantzia $r_n^{(i)} - r_n^{(j)} = D_{i,j} + \epsilon$ bada ($\epsilon \to 0$), t_n aldiunean ez da talka detektatuko. Hurrengo pausuko i partikularen posizioa $r_{n+1}^{(i)} = r_n^{(i)} + v_{n+\frac{1}{2}}^{(i)} \cdot h$ izanik, $v_{n+\frac{1}{2}}^{(i)} = v_{n-\frac{1}{2}}^{(i)}$ izango da talkarik ez badago, eta $v_{n+\frac{1}{2}}^{(i)} = {v^{(i)}}'_{n-\frac{1}{2}}$ talka badago. t_{n+1} aldiunean talka ez da detektatuko $r_{n+1}^{(i)} - r_n^{(i)} > D_{i,j} + \epsilon$ bada. Era berean j partikularen kasuan $r_{n+1}^{(j)} - r_n^{(j)} < -(D_{i,j} + \epsilon)$. Gainera, 2.2.2. atalean suposatu dugu partikulak ez direla inoiz bata bestearen gainean egongo, beraz, muga zorrotzagoa jarriko dugu. Bete beharreko baldintza:

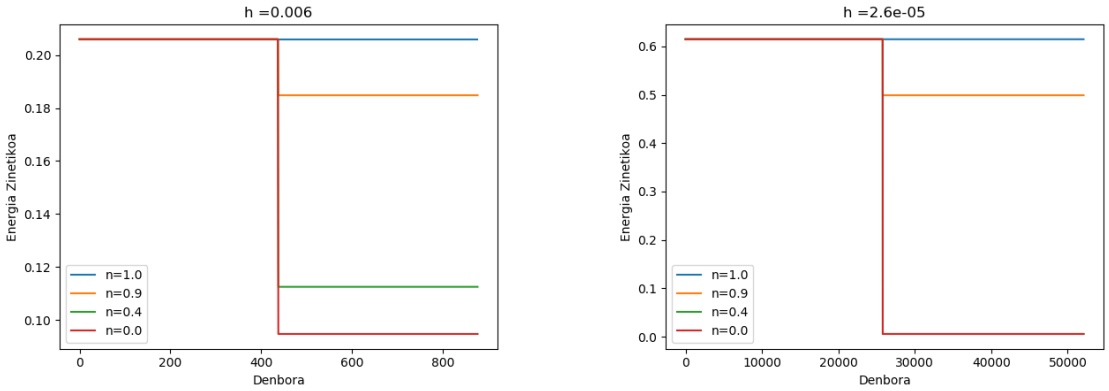
$$r_{n+1} - r_n = v_{n+\frac{1}{2}} \cdot h < \frac{D}{4} \tag{3.1}$$

Partikula guztiek ez dutenez abiadura berdina, batez bestekoa erabiliko dugu:

$$h < \frac{D_{i,j}}{4 \cdot \bar{v}} \tag{3.2}$$

Irizpide honek h handiegia hautatzearen arazoa konpondu arren, ereduak [4] artikuluan aipatzen ez den arazo bat aurkezten duela ikusi da $h \ll \frac{D_{i,j}}{4 \cdot \bar{v}}$ ezartzean. Denbora-pausua txikiegia denean, partikulen arteko talka inelastikoetan partikulak itsatsita geratu daitezke guztiz inelastikoak ez izan arren ($1 > n > 0$). Talka inelastikoetan, beti partikulen abiadura erlatiboa txikitzen da, eta baliteke abiadura hau ez izatea nahikoa hurrengo denbora-pausuan talkatik ateratzeko. Hau gertatzen bada, hurrengo aldiunean berriro talka detektatuko da, eta partikulak berriro gerturatu egingo dira, abiadura erlatiboaren zeinua talka bakoitzean aldatzen denez gero. Horrela, etengabeko talkak detektatuko dira, eta abiadura erlatiboa moduluz txikitzen joango da aldiune bakoitzean zerorantz gerturatuko delarik. Adibidez, 9(b). irudiko simulazioan h txikiegia aukeratu da, eta $n = 0.4$ kasua talka guztiz inelastikoaren berdina da. Ez da lortu h-ren azpi-mugaren balio hurbildu bat ematea, baino froga ezberdinak eginez ikusi da ez dela gomendagarria (batez ere n txikietarako) 3.2. ekuazioko baldintzan lortzen den goi-mugatik asko urruntzea.

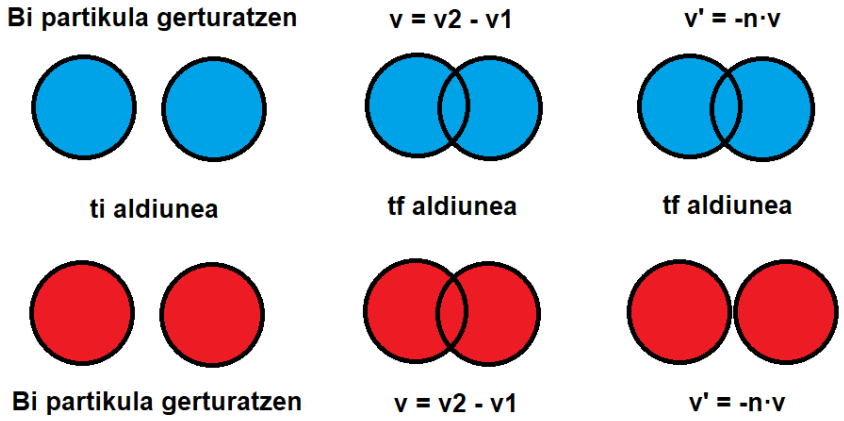
Hortaz, idatzitako kodean h-ren hautaketa egiteko, partikula kopuru murriztu batekin partikulen ezaugarri intrintsekoen eta batez besteko abiaduraren araberako simulazioak egin behar dira. Esperotako emaitza zein h-ren magnitude ordenarako lortzen den aurkitzean, partikula kopurua igo daiteke. h txikiegia izatearen arazoa konpontzeko beste aukera bat talkak ebazteko modua aldatzea da. Orain arte, talka eta geroko abiadurak kalkulatzen direnean, talkaren parte diren partikulen hurrengo pausua kalkulatzeko talka detektatu deneko posizioak hartu dira. Posizio hauen arteko distantzia erlatiboa partikulen erradioen batura baino txikiagoa da, talka gertatu denez gero. Horregatik, partikulek bata bestearengandik ihes egitea ez badute lortzen, hurrengo aldiunean berriro ere partikula berdinen arteko talka detektatuko da. Hau ekiditeko modu bat talkak exekutatzean partikulen arteko distantziak erradioen baturetaraino aldentzea da ($r_{i,j} = \frac{D_i + D_j}{2} + \epsilon$). Hortaz, partikulek ihes egingo dute hurrengo denbora-pausuan (ikus 10. irudia). Talkak ebazteko modu honetan, ordea, partikula bera bi partikula edo gehiagorekin talka egiten ari denean arazo bat agertuko da. Talka



(a) $h \sim \frac{D}{5 \cdot \bar{v}}$ (b) $h \sim \frac{D}{500 \cdot \bar{v}}$

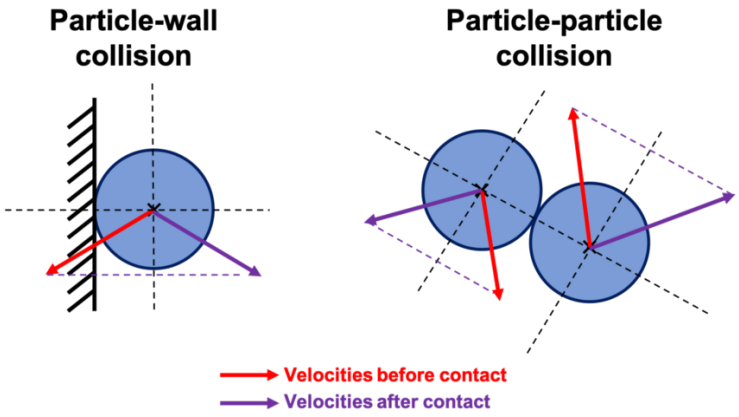
9 Irudia: Bi partikulen energia zinetikoak n talka-parametro ezberdinetarako.

exekutatzean, partikulen posizioak aldatzen ari gara. Ondorioz, i partikulak j eta w partikulekin talka egiten badu, eta lehenengo i eta j partikulen arteko talka exekutatzen bada, i partikularen posizioa aldatu egingo da w partikularekin talka exekutatu aurretik. Hortaz, berriro kalkulatu beharko litzateke i eta w-ren arteko talka emango den edo ez. Kodea partikula kopuru handi batekin erabili nahi denez, hau ez da bideragarria.



10 Irudia: Talkak exekutatzeko bi modu ezberdin. Bi kasutan, t_i aldiunean ez dago talkarik eta t_f aldiunean talka detektatu eta exekutatu deneko aldiunea da. Goiko irudian, partikula urdinen talka exekutatzean partikulen abiadura erlatiboa aldatu egiten da (v-tik v'-ra), baina posizioak mantendu egiten dira. Inplementatutako kodea eredu honetan oinarritu da. Beheko irudian, partikula gorrien talkak exekutatzen diren aldiunean partikulen posizioak aldatu egiten dira. Hortaz, iterazio berdinean partikulen abiadura erlatiboa aldatzeaz gain (v-tik v'-ra) posizio erlatiboa ere aldatu egiten da ($r < D$-tik $r' > D$-ra).

Gas-partikulen higidura Presio baxua[7] duen gas erreal bat gas ideala kontsideratu daiteke. Gas ideal bateko partikulen artean indar alderatzaile zein erakarleak arbuiatu daitezke, eta horrela, hauen arteko talkak izango dira kontuan hartu beharreko elkarrekintza bakarrak. Hortaz, gas ideal batean elkarrekintza grabitatorioa arbuiatu dezakegu [16]. Teoria zinetiko molekularraren arabera, gas-partikulen arteko talkak elastikoak dira, eta beraz, N gorputzek gas ideala eratzen dutenean talka-parametroaren balioa $n = 1$ izango da. Gasak espazioan hedatzeko joera du, eta horregatik, hainbat partikula hasieratzen baditugu espazioko zona batean, hauek berehala sakabanatuko dira espazio osora. Hau ekiditeko, *Kutxa* izeneko klasea sortu da (ikus B.5. eranskina). Gas-partikulak kutxaren barruan kokatuko dira eta ezingo dute kutxatik atera. Partikula batek kutxako pareta baten kontra jotzean, partikularen abiaduraren norabidea aldatuko da. Zehazki, paretarekiko normala den osagaiaren noranzkoa aldatzen da (ikus 11. irudia).



11 Irudia: Pareta baten kontra talka egiten duen partikula baten abiaduraren aldaketa, eta bi partikulek talka egitean abiadura bakoitzaren aldaketa [17].

Atal honetan kutxa batean gas-partikulak sartuko ditugu, eta ausazko partikula baten ibilbidea behatuko dugu. Gasaren dentsitatea handitzean, hau da, partikula kopurua handitzean, behatzen ari garen partikulak talka gehiago jasango dituela ikusiko dugu. Simulazioan zehar partikula batek jasango dituen talka-kopuruaren balio hurbildua ondorengoa da:

$$\text{Talka Kopurua} \approx \text{Iterazio Kopurua} \cdot (N-1) \cdot \frac{\frac{4\pi}{3}\left(\frac{D}{2}\right)^3}{l^3} \tag{3.3}$$

non Iterazio Kopurua simulazioaren denbora-pausu kopurua eta l kutxaren aldea diren. Simulazioa burutzeko, ereduko aldagai espezifikoak eta hasierako baldintzak aukeratu behar dira. Jarraian, aldagaien balioak aukeratzeko jarraitutako irizpideak azalduko dira[8]:

- Kutxa: sinpleena kutxaren zentroa $(0, 0, 0)$ puntuan kokatzea eta $l = 1$-eko aldea ematea da.

[7] Presioa baxua dela kontsideratzen da 2 atm baino gutxiago denean.
[8] Aldagai batzuen balioak amaieran aukeratuko dira, beste aldagaiekiko duten dependentzia dela eta.

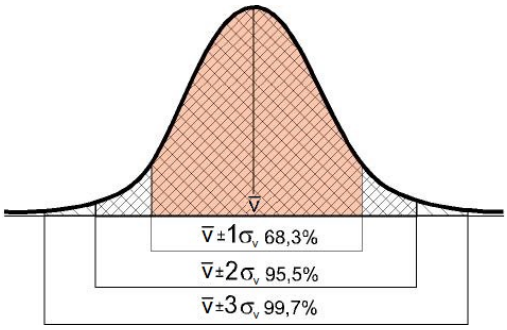
- Partikula kopurua (N): froga batetik bestera aldatuko den balioa izango da, partikula batek jasandako talka-kopurua nola aldatzen den ikusteko. Froga $N = 50$-tik $N = 1000$-ra bitarteko balioekin egitea tarte interesgarria[9] da.

- Partikulen masa (m): partikula guztiei masa berdina emango zaie. Elkarrekintza grabitatorioa arbuiatuko denez, soilik talkak ebatzi beharko dira. 2.21. ekuazioa behatzen badugu, ikus daiteke masak sinplifikatu egingo direla, eta beraz, masaren edozein balio eman diezaiokegu gas-partikula berdinei. Masa oso handia bada, elkarrekintza grabitatorioa arbuiatzeak zentzu fisikoa galtzen du. Hala ere, magnitude eskalak ez direnez zehazten simulazioan, erabateko askatasuna dago. Sinplifikazioagatik $m = 1$ aukeratuko da.

- Partikulen diametroa (D): partikulak berdinak direnez, guztiei diametro bera emango zaie. Kutxaren aldea $l = 1$ bada, partikulen diametroa hau baino txikiagoa izan behar da simulazioak zentzua izateko. Diametro txikiegia aukeratzen bada, aukeratu dezakegun N handienetan ere talkak gertatzea ez da probablea izango.

- Partikulen hasierako posizioa: modu sasiausazkoan banatuko dira partikulak kutxan zehar.

- Partikulen hasierako abiadura: Gauss-en banaketa erabiliko da (ikus 12. irudia). Batez bestekoa $\bar{v} = 0$ aukeratzen bada dimentsio guztietan, abiadura positiboak zein negatiboak egongo dira, eta σ_v desbiderapen estandarraren arabera partikulek abiaduraren modulu handiagoa edo txikiagoa izango dute.



12 Irudia: Gauss-en banaketa. $\bar{v}$ batez bestekoa eta σ_v desbiderapen tipikoa dira.

- Talka-parametroa (n): talkak elastikoak direnez, $n = 1$ balioa finkatu da.

- Iterazio kopurua (*loop*): simulazioaren iraupena mugatuko du.

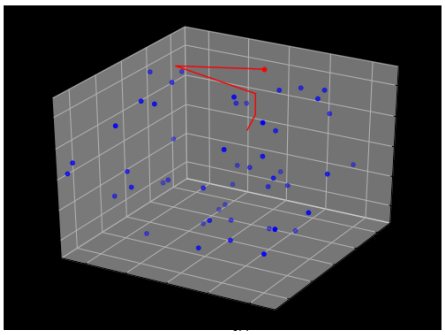
- Denbora-pausua (h): iterazio batetik bestera igaroko den denbora aukeratu behar da. h-ren balioa handiegia bada, partikulen arteko talkak ez dira detektatuko elkar gurutzatuta ere. Hau ez gertatzeko, 3.2. ekuazioko baldintza erabiliko dugu. Abiaduraren batez bestekoa kalkulatzeko, Gauss-en banaketako zorizko aldagaiaren trinkotasun funtzioa erabiliz [18]:

$$\bar{v^2} = \int_{-\infty}^{\infty} v^2 \frac{1}{\sqrt{2\pi}\sigma_v} e^{-\frac{v^2}{2\sigma_v^2}} dv = \sigma_v^2 \tag{3.4}$$

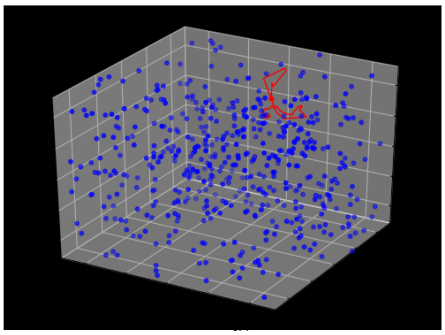
[9]Eskura ditugun baliabide konputazionalekin partikula kopurua handitzeak kostu gehiegi dakar.

$$h < \frac{D}{4 \cdot \sigma_v} \tag{3.5}$$

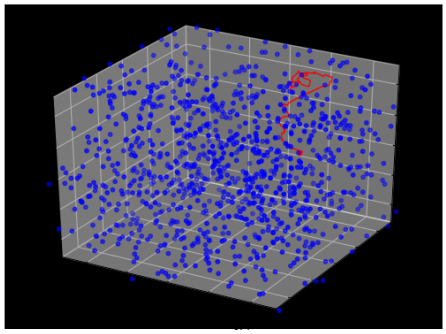
Behin aldagai guztiak aztertuta, falta zaizkigun aukeraketak egingo ditugu. Alde batetik, 100 iterazioko simulazioa burutuko da. Bestetik, $N = 50$, $N = 500$ eta $N = 1000$ partikula-kopuruko simulazioak egingo dira. Bi aldagai hauek gure baliabide konputazionalek mugatzen dizkigute. Hemendik abiatuz, gainerako parametroak finkatu daitezke. $N = 50$ partikula gutxieneko simulazioan, 3.3. ekuazioari jarraituz, partikula bakoitzak gutxienez talka bat jasateko $D = 0.08$ aukeratu behar da. $\sigma_v = 10$ aukeratzen bada, $h = 0.001$ aukeratzea egokia da $h < \frac{D}{4 \cdot \sigma_v}$ irizpideari jarraituta.



(a) $N = 50$. Beste partikulen kontra talka bakarra jasan du partikula gorriak.



(b) $N = 500$. Beste partikulen kontra 88 talka jasan ditu partikula gorriak.



(c) $N = 1000$. Beste partikulen kontra 246 talka jasan ditu partikula gorriak.

13 Irudia: Gas-partikula multzo baten higiduraren simulazioa. Ez dago elkarrekintza grabitatoriorik. Gorriz irudikatu den partikula ausaz aukeratu den bat da. $m = 1$, $D = 0.08$, $n = 1$, $loop = 100$, $h = 0.001$, $l = 1$. Posizioak ausaz aukeratu dira eta abiadura Gauss-en banaketaren bitartez: $\bar{v} = 0$ eta $\sigma_v = 10$ dimentsio bakoitzean.

13. irudian ikus daiteke esperotako emaitzak lortu direla; partikula batek jasandako talka-kopurua nabarmen igotzen da kutxa barruko partikula kopurua igotzen denean. Kasu honetan beste partikulen aurka jasandako talka-kopurua zenbatu da, eta ez dira kontuan hartu kutxaren pareten aurka jasandako talkak. Azkenik, simulazio hauen kostu konputazionalaren ideia bat izateko, $N = 1000$ partikulako simulazioak $\sim 45\,\mathrm{s}$ behar izan ditu kalkuluak egin eta GIF animatua eratzeko ordenagailu eramangarri arrunt batean.

3.2.3 Grabitatea eta Talken simulazio bateratua

Behin bi kasuak independenteki inplementatu eta frogatu direla, partikulen arteko bi elkarrekintza motak bateratu dira Leapfrog algoritmoa erabilita. $\vec{r}_n$ eta $\vec{v}_{n+\frac{1}{2}}$ jakinik, ondorengoa egingo da:

1. $\vec{r}_{n+1}$ kalkulatu 2.28. ekuazioa erabilita.
2. Gorputz batek talka jasaten badu posizio berrian, $\vec{v}'_{n+\frac{1}{2}}$ kalkulatu B.3. eranskinean ageri den **talkaEbatzi** funtzioa erabilita. $\vec{v}_{n+\frac{1}{2}} = \vec{v}'_{n+\frac{1}{2}}$ egin.
3. Grabitatearen ondoriozko $\vec{a}_{n+1}$ azelerazioa kalkulatu A. eranskinean ageri den **axyz** funtzioa erabilita.
4. $\vec{a}_{n+1}$ azelerazioa izanik, $\vec{v}_{n+\frac{3}{2}}$ kalkulatu[10] 2.27. ekuazioa erabilita. Talka jasaten duten partikulen kasuan $\vec{a}_{n+1} = 0$ da, eta beraz, $\vec{v}_{n+\frac{3}{2}} = \vec{v}'_{n+\frac{1}{2}}$ egiten ariko ginateke.

Leapfrog algoritmoak behar dituen hasierako baldintzak $\vec{r}_0$ eta $\vec{v}_{\frac{1}{2}}$ dira. Orokorrean problemaren $\vec{v}_0$ jakingo denez, $\vec{v}_{\frac{1}{2}}$ kalkulatzeko ondorengoa egingo da:

1. $\vec{v}_0$ eta $\vec{r}_0$ izanik hasierako baldintzak, t_0 aldiunean talkarik dagoen egiaztatu eta $\vec{v}_0 = \vec{v}'_0$ gorde **talkaEbatzi** funtzioa erabilita.
2. $\vec{a}_0$ hasierako azelerazioa kalkulatu **axyz** funtzioaren bitartez.
3. $\vec{v}_{\frac{1}{2}}$ kalkulatu 2.30. ekuazioa erabilita.

C. eranskinean programa nagusiaren adibide bat gehitu da. Honekin prest gaude gure eredua jarraitzen duen edozein problema simulatzeko. Hainbat kasu ezberdin aztertu dira orain arte aipatutakoez gain (Eguzki-sistema eta gas-partikulen higidura). Idatzi honetan azterketa fisiko sinpleenetarikoa duen problema jorratuko da:

N partikulen grabitazio-kolapsoa Eskuarteko problema elkarrekintza grabitatorioaren ondorioz erakartzen diren eta distantzia laburretara talkak jasaten dituzten partikula-multzo baten denbora-eboluzioa simulatzea da. Horretarako, jada idatzi diren funtzioak eta algoritmoak erabiliko dira.

Partikula-multzo handiekin hasten garenean lanean, kostu konputazionala nabarmen handitzen da. Simulazio bera behin eta berriz ez exekutatzeko, efizienteena simulazioko datuak

[10] Ekuazioa aplikatzeko $\vec{v}_{n+\frac{3}{2}} \longrightarrow \vec{v}_{n+\frac{1}{2}}$ eta $\vec{v}_{n+\frac{1}{2}} \longrightarrow \vec{v}_{n-\frac{1}{2}}$ ordezkatu.

gordetzea da. Ondoren, beste programa bat sortu da simulazio honen GIF animatuak sortzeko. Horrez gain, simulazio bat amaitzean simulazioa bera luzatu dezakeen programa sortu da. Honen erabilgarritasuna ikusteko, demagun 1000 partikulako simulazio bat burutu dugula eta simulazio horren exekuzioak 10 minutu behar izan dituela. Hasierako baldintzak eta bestelako datuak gordetzen baditugu, simulazio bera berriro exekutatu ahalko genuke[11]. Demagun, ordea, simulazio hau ez dela nahikoa izan 1000 partikula horien eboluzioa aztertzeko, eta beste 10 minutu nahi ditugula simulatu. Kasu honetan, hasierako baldintzak hartu eta 20 minutuko simulazioa exekutatu beharko genuke. Denbora aurrezteko modu bat, simulazioko azken egoeratik abiatzeko gai den programa bat idaztea da.

Hortaz, hiru programa idatzi dira:

1. Programa nagusia (ikus D.1. eranskina). Simulazioko aldagai espezifikoak eta hasierako baldintzak zehaztu behar dira programa honetan. Hauek *Informazioa.txt* izeneko fitxategi batean gordeko dira, eta denbora-eboluzioko posizioak *Datuak.txt* fitxategian gordeko dira ondoren GIF animatuak sortzeko. Exekuzio bakoitzari dagozkion fitxategiak karpeta propio batean gordeko dira automatikoki.

2. Simulazioa luzatzeko programa (ikus D.2. eranskina). Luzatu nahi den simulazioa eta luzapen-denbora adierazi behar dira. Programa honek berak automatikoki editatuko ditu *Informazioa.txt* eta *Datuak.txt* fitxategietako datuak.

3. Simulazioa irudikatzeko programa (ikus D.3. eranskina). Zein simulazioren GIF animatua sortu nahi den adierazi beharko da eta automatikoki GIF animatua simulazioaren karpetan gordeko da (.txt fitxategiak dauden karpeta berdinean).

Berriro ere, simulazioa burutzeko ereduko aldagai espezifikoak eta hasierako baldintzak aukeratu behar dira. 2.1. atalean magnitude eskalak aukeratzeko dagoen askatasuna gogora ekarriz:

- Grabitazio unibertsalaren konstantea $G = 1$ aukeratuko da.
- Partikula kopurua $N = 500$ aukeratuko da. Partikula kopuru handiagoak kostu konputazional handiegia suposatzen du.
- Partikulen masa (m): partikula guztiei masa berdina emango zaie. $m = 1$ aukeratuko da.
- Partikulen diametroa (D): partikula guztiei diametro berdina emango zaie. $D = 1$ aukeratuko da.
- Partikulen hasierako posizioa: Gauss-en banaketa erabiliko da. Batez bestekoa koordenatu kartesiarren jatorrian kokatuko da. Kolapsoa gertatzeko partikulak bata bestearengandik urrun kokatu behar direnez, desbiderapen tipikoa aukeratzeko ondorengo irizpidea proposatzen da:

$$\frac{N \cdot V_{partikula}}{V_{totala}} \approx \frac{N \cdot D^3}{\sigma_r^3} \ll 1 \tag{3.6}$$

[11] Emaitza ez da guztiz berdina izango, talkak ausazko ordenan exekutatzen dira eta.

$\sigma_r = 100$ hartzea aukeraketa ona da.

- Partikulen hasierako abiadura: kolapsoa aztertu nahi denez, partikulak pausagunetik abiatuko dira[12].
- Talka-parametroa (n): talkak inelastikoak dira. $n = 0.1$ aukeratuko da [6].
- Denbora-pausua (h): h-ren balioa handiegia bada, partikulen arteko talkak ez dira detektatuko elkar gurutzatuta ere. Grabitate-indarra dugunez, irizpide berri bat proposatu behar da h-ren balio egokia aukeratzeko:

Demagun i eta j partikulak aztertzen ari garela eta bien elkarrekintza dimentsio batean deskribatu daitekeela. Bi partikulen zentroen arteko distantzia $r = D + \epsilon$ bada t_n aldiunean ($\epsilon \to 0$), ez da hauen arteko talka gertatuko. $r^{(i)}_{n+1}$ kalkulatzeko behar den $v^{(i)}_{n+\frac{1}{2}} = v^{(i)}_{n-\frac{1}{2}} + a^{(i)}_n \cdot h$ izango da, non $a^{(i)}_n \approx G\frac{m}{r^2_{i,j}} \approx G\frac{m}{D^2}$, j partikula besteak baino gertuago egonik, eragin handiena baitu azelerazio totalean. Abiadura erlatiboa handiegia bada, partikulak elkar gurutzatuko dira eta t_{n+1} aldiunean talka ez da detektatuko. Hau ez gertatzeko, denbora-pausu txikia aukeratu behar da. Berriro ere, 2.2.2. atalean suposatu dugunez partikulak ez direla inoiz bata bestearen gainean egongo, muga zorrotzagoa jarriko dugu. Bete beharreko baldintza:

$$r^{(i)}_{n+1} - r^{(i)}_n < \frac{D}{4} \tag{3.7}$$

$$v^{(i)}_{n+\frac{1}{2}} \cdot h < \frac{D}{4}$$

$$(v^{(i)}_{n-\frac{1}{2}} + a^{(i)}_n \cdot h) \cdot h \approx \sigma_v \cdot h + G\frac{m}{D^2} \cdot h^2 < \frac{D}{4}$$

$$h < -\frac{\sigma_v\, D^2}{2\, G\, m} + \frac{\sqrt{\sigma_v^2 D^4 + G\, m\, D^3}}{2\, G\, m} \tag{3.8}$$

Kasu honetan partikulak pausagunetik abiatzen direnez $\sigma_v = 0$ izango da. 3.8. ekuazioa honela geratuko zaigu:

$$h < \sqrt{\frac{D^3}{4\, G\, m}} = 0.5 \tag{3.9}$$

h=0.2 aukeratuko da.

- Iterazio-kopurua (*loop*): aukeratu diren magnitudeak ikusita, zenbat denborako simulazioa burutu beharko da partikulak kolapsatzen ikusteko? Gutxi gorabeherako balioa lortzeko hainbat hurbilketa egingo dira. Demagun partikula bat zentrotik 3σ distantziara dagoela. 12. irudian agertzen den moduan, partikula gehienak (%99, 7) erradio honen barruan aurkituko dira. Zenbat denbora beharko du puntu honetan kokatzen den

[12]Partikulei abiadura ematea kolapsoa eragotzi dezakeen fenomenoa da. Esaterako, biraketa erresultante batek indar zentrifugoa eragiten du, partikulak zentrora erortzea eragotziko duena.

partikula batek zentrora iristeko? Talkak arbuiatzen baditugu, Newtonen grabitazio unibertsalaren legea erabiliz [10]:

$$a \approx G \cdot \frac{N \cdot m}{9\sigma_r^2} \tag{3.10}$$

Higidura zuzen uniformeki azeleratua izanik eta pausagunetik abiatzen dela onartuz:

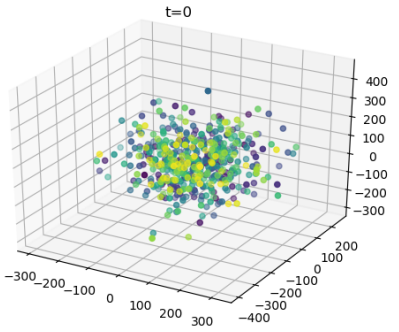
$$3\sigma_r = a \cdot \frac{t^2}{2} \tag{3.11}$$

Ondoko adierazpen orokorrera iristen gara:

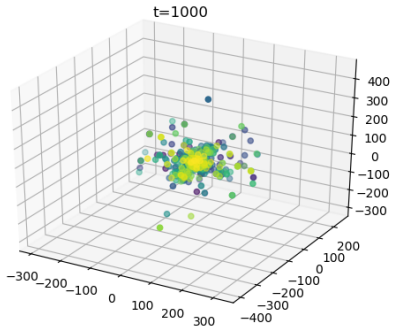
$$t = h * loop \approx \sqrt{\frac{54\,\sigma_r^3}{G\,N\,m}} \approx 330 \tag{3.12}$$

Simulazioa luzatzeko funtzioari esker, aukeratzen den iterazio kopuruaren balioa ez da horren erabakigarria, baina irizpideari jarraituz lehenengo simulaziotik emaitza esanguratsuak ikusteko $loop > 1650$ aukeratuko da.

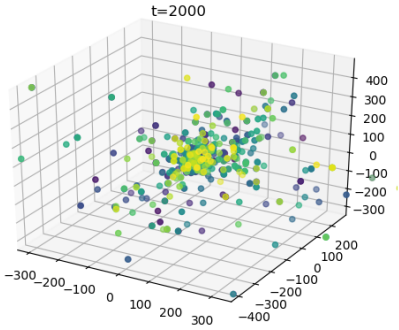
Simulazioari esker, partikulen higidura deskribatzea lortu da (ikus 14. irudia) eta 11 500 iterazio burutzeko ~ 1 ordu behar izan dira ordenagailu eramangarri arrunt batean. Lortu diren emaitzen analisi estatistikoa egitea interesgarria da emaitzak interpretatzeko. Lanaren helburua kodea eraikitzea denez eta ez simulazioaren emaitzak interpretatzea, ez gara honetan sartuko. Hala ere, merezi du aipatzea $t = 0$ aldiunean desbiderapen estandarra ~ 100 dela, $t = 1000$ kolapsoa gertatzen ari den aldiunean $\sigma \sim 50$ dela, eta amaieran $t = 11\,500$ denean $\sigma \sim 1200$ izatera igaro dela. Partikula askok kolapso ondoren ihes egitea lortzen dutenez, σ-ren balioa asko handitzen da. Azterketa hau *statistics* biblioteka erabiliz egin da [19].



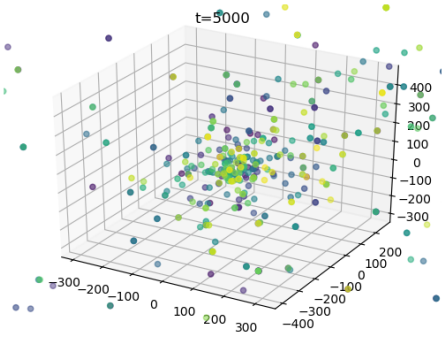
(a) $t = 0$



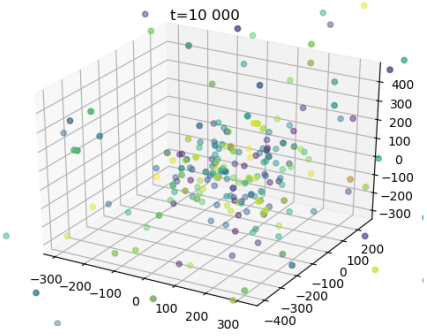
(b) $t = 1\,000$



(c) $t = 2\,000$



(d) $t = 5\,000$



(e) $t = 10\,000$

14 Irudia: $N = 500$ gorputz berdinen grabitazio-kolapsoa. Aldagai espezifikoak: $G = 1$, $m = 1$, $D = 1$, $n = 0.1$, $h = 0.2$ eta $loop = 11\,500$. Hasierako baldintzak: posizioak Gauss-en banaketaren bitartez hasieratu ($\bar{r} = 0$, $\sigma_r = 100$) dira eta partikulak pausagunetik abiatu dira.

4 Ondorioak

Lan honetan talkak jasaten dituzten N gorputz masiboren higidura simulatzeko kodea eraiki da. Helburua kode ahalik eta eraginkorrena sortzea izanik, sistemaren aurretiko analisia eta ondorengo kodearen egiaztapen prozesuak ezinbestekoak izan dira.

Aurretiko analisia burutzeko, metodologia konkretu bat jarraitu da. Lehenengo eta behin, sistemaren modelizazioa landu da, kontuan hartu behar diren lege fisikoak eta simulazioan eragina izango duten aldagaiak identifikatu direlarik. Ondoren, sistema deskribatzen duen ekuazio diferentziala lortu eta eskuz talkak ebazteko ariketa aljebraikoa ebatzi da. Adierazpen matematikoa konputazionalki inplementatzeko, bi algoritmo landu dira: Euler-en metodo esplizitua eta Leapfrog algoritmoa. Probako simulazioetan azken hau erabili den arren, Euler-en metodoa lantzea erabilgarria izan da Leapfrog algoritmoak hasierako baldintzekin duen problema konpontzeko.

Aurretiko analisia burututa, kodea idatzi da. Hau hiru zatitan banatu da. Batetik, partikula bakoitzak jasaten duen grabitate-indarraren ondoriozko azelerazioa kalkulatzeko funtzio bektorizatua idatzi da, eta Eguzki-sistemako objektuen orbitak kalkulatzeko erabili da. Bestetik, talkak ebazteko hainbat funtzio idatzi dira, eta gas-partikulen higidura simulatzeko erabili da. Azkenik, bi elkarrekintzak bateratuko dituen programa nagusiaren zati garrantzitsuena eraiki da. Problema konkretu bakoitzak bere hasierako baldintzak eta aldagai espezifikoak izan arren, beti errepikatuko den Euler (hasierako baldintzetan) eta Leapfrog (denbora-eboluzioan) algoritmoak erabiltzen dituen kode zatia da hau. Kodearen erabilera lantzeko, $N = 500$ gorputzen grabitazio-kolapsoa aztertu da.

Lana aurrera eramateko, hainbat arazoei aurre egin behar izan zaie. Batetik, talkak deskribatzeko eredu egonkorra bilatzea zeregin neketsua izan da, lanean azaldu den ereduaz gain beste bi aztertu direlako aurretik ([3] artikulukoa eta osziladore indargetua). Horrez gain, problema bakoitzaren aldagai espezifikoen analisia egiteko fisikako zein matematikako ezagutzak behar izan dira. Horrela, denbora-pausuaren zein bestelako aldagaien balio egokiak aukeratzeko irizpideak proposatu ahal izan dira.

Etorkizunari begira, ondorengoak egin daitezke lanari jarraipena emateko:

- h txikiegia aukeratzeak talka inelastikoetan duen eragina sakonki aztertu eta azpimuga zein den kalkulatu. Beste aukera bat, 10. irudiaren bitartez azaldu den talkak exekutatzeko modua aldatzea da. Hau inplementatu nahi bada, kontuan izan behar da kostu konputazionala asko handitu daitekeela.
- Kodearen exekuzio-denbora hobetzeari dagokionez, zuhaitzaren metodoan oinarritzen den *Barnes-Hut* algoritmoa inplementatzea. Espazioa oktanteetan banatuz, urrun dauden partikulen arteko indar-bikoteak arbuiatzen ditu. Horrela, iterazio bakoitzean kalkulu gutxiago burutu beharko dira.
- Hasierako baldintzak aztertuz eta potentzia handiko konputazio-tresnak erabiliz, hasierako motibazio gisa hartu zen Ilargiaren sorreraren simulazioa posible izan daiteke.

Erreferentziak

[1] E. Bertschinger and J. M. Gelb, "Cosmological n-body simulations: Modern simulations use millions of particles to follow thousands of galaxies in a large volume of space," *Computers in Physics*, vol. 5, no. 2, pp. 164–179, 1991.

[2] D. Ascher and M. Lutz, *Learning Python*. O'Reilly, 1999.

[3] J. C. Eiland, T. C. Salzillo, B. H. Hokr, J. L. Highland, and B. M. Wyatt, "N-body simulation of the formation of the earth-moon system from a single giant impact," *arXiv preprint arXiv:1307.7062*, 2013.

[4] H. Rein and S.-F. Liu, "Rebound: an open-source multi-purpose n-body code for collisional dynamics," *Astronomy & Astrophysics*, vol. 537, p. A128, 2012.

[5] J. Aguirregabiria, A. Hernandez, and M. Rivas, "A simple model for inelastic collisions," *American Journal of Physics*, vol. 76, no. 11, pp. 1071–1073, 2008.

[6] T. Sasaki and N. Hosono, "Particle number dependence of the n-body simulations of moon formation," *The Astrophysical Journal*, vol. 856, no. 2, p. 175, 2018.

[7] H. Gould, J. Tobochnik, and W. Christian, *An introduction to computer simulation methods*, vol. 1. Addison-Wesley New York, 1988.

[8] J. M. Aguirregabiria Aguirre, *Fisika ikasleentzako ekuazio diferentzial arruntak*. Servicio Editorial de la Universidad del País Vasco/Euskal Herriko ..., 2009.

[9] P. O. Scherer, *Computational Physics*. Springer, 2010.

[10] P. A. Tipler and G. Mosca, *Física para la ciencia y la tecnología. I*, vol. 1. Reverté, 2004.

[11] J. M. Aguirregabiria, *Mekanika klasikoa*. Universidad del País Vasco, 2004.

[12] "What is vectorized code?." http://www.cs.cornell.edu/courses/cs1112/2015sp/Exams/exam2/vectorizedCode.pdf, 2015.

[13] T. S. community, "NumPy." https://numpy.org/doc/stable/user/whatisnumpy.html.

[14] B. W. Carroll and D. A. Ostlie, *An introduction to modern astrophysics*. Cambridge University Press, 2017.

[15] R. Brandon, "Skyfield: Generate high precision research-grade positions for stars, planets, moons, and earth satellites." https://rhodesmill.org/skyfield/, 2020.

[16] M. W. Zemansky, "Heat and thermodynamics: an intermediate textbook," 1968.

[17] C. Mehl, "Modeling of disease propagation using a particle/collision formalism." https://medium.com/@cedric.mehl/modeling-of-disease-propagation-using-a-particle-collision-formalism-a2e4b5dba133.

[18] K. F. Riley, M. P. Hobson, S. J. Bence, and M. Hobson, *Mathematical methods for physics and engineering: a comprehensive guide*. Cambridge university press, 2002.

[19] P. S. Foundation, "statistics — mathematical statistics functions." https://docs.python.org/3/library/statistics.html, 2021.

Eranskinak

A Grabitatearen kodea

A.1 Funtzio Nagusia

```
import numpy as np


def axyz(N, G, Dij, m, x, y, z):
    """
        Funtzio honek partikula bakoitzaren grabitatearen ondoriozko
        azelerazioa kalkulatzen du hiru dimentsiotan.

    Jaso:
        N: partikula kopurua (int)
        G: grabitazio unibertsalaren konstantea (float)
        Dij: partikulen erradioen batura (NXN tamainako np array-a,
                                          elementuak float)
        m: partikulen masak (N tamainako np array-a, elementuak float)
        x: partikulen x koordenatuko posizioak (N tamainako np array-a,
                                                elementuak float)
        y: partikulen y koordenatuko posizioak (N tamainako np array-a,
                                                elementuak float)
        z: partikulen z koordenatuko posizioak (N tamainako np array-a,
                                                elementuak float)
    Itzuli:
        ax: partikulen x koordenatuko azelerazioak (N tamainako np array-a,
                                                    elementuak float)
        ay: partikulen y koordenatuko azelerazioak (N tamainako np array-a,
                                                    elementuak float)
        az: partikulen z koordenatuko azelerazioak (N tamainako np array-a,
                                                    elementuak float)
    """
    # Partikulen arteko posizio erlatiboak kalkulatu
    xij = np.transpose(np.ones([N, N])*x) - np.ones([N, N])*x
    yij = np.transpose(np.ones([N, N])*y) - np.ones([N, N])*y
    zij = np.transpose(np.ones([N, N])*z) - np.ones([N, N])*z
    R = np.power(np.power(xij, 2) + np.power(yij, 2) + np.power(zij, 2), 0.5)

    # Zatitzaile nulurik ez egoteko laguntzailea
    Rlagun = R + np.identity(N)

    # Talkarik ez badago bi elementuen artean batak besteari eragingo liokeen
    # azelerazioa kalkulatu. Talka badago azelerazioa 0.
```

```
    matrixx = np.where(R > Dij, -G*np.transpose(m)*np.power(Rlagun, -3)*xij, 0)
    matrixy = np.where(R > Dij, -G*np.transpose(m)*np.power(Rlagun, -3)*yij, 0)
    matrixz = np.where(R > Dij, -G*np.transpose(m)*np.power(Rlagun, -3)*zij, 0)

    # Azelerazioak batu. Talka jasaten duten elementuen azelerazio totala nulua.
    lagun = np.where(Rlagun + Dij*np.identity(N) > Dij, True, False)
    ax = np.prod(lagun, axis=1)*np.sum(matrixx, axis=1)
    ay = np.prod(lagun, axis=1)*np.sum(matrixy, axis=1)
    az = np.prod(lagun, axis=1)*np.sum(matrixz, axis=1)

    return ax, ay, az
```

A.2 Eguzki-sistemaren simulazioa

```
import math
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import numpy as np
import time
from skyfield.api import load
import Grabitatea3DOrokor as grab


if __name__ == "__main__":

    N = 10
    G = 8.888*math.pow(10, -10)
    eps = math.pow(10, -2)  # Orbitak bira oso bat eman duen bereizteko distantzia
    loop = 100000  # Zenbat denbora pausu kalkulatuko diren
    jump = 100  # Zenbat datu grafikatuko diren
    h = 1
    x_all = np.empty((N, int(loop/jump)), dtype=float)
    y_all = np.empty((N, int(loop/jump)), dtype=float)
    z_all = np.empty((N, int(loop/jump)), dtype=float)
    m = np.array([333000, 0.05528, 0.81500, 1.0, 0.10745,
                  317.83, 95.159, 14.536, 17.147, 0.0021])

    ts = load.timescale()
    tnow = ts.now()

    planets = load('de421.bsp')

    sun = planets['sun']
    earth = planets['earth']
```

```
mars = planets['mars']
mercury = planets['mercury']
venus = planets['venus']
jupiter = planets['JUPITER BARYCENTER']
saturn = planets['SATURN BARYCENTER']
uranus = planets['URANUS BARYCENTER']
neptune = planets['NEPTUNE BARYCENTER']
pluton = planets['PLUTO BARYCENTER']

sun_pos = sun.at(tnow).position.au
earth_pos = earth.at(tnow).position.au
mars_pos = mars.at(tnow).position.au
mercury_pos = mercury.at(tnow).position.au
venus_pos = venus.at(tnow).position.au
jupiter_pos = jupiter.at(tnow).position.au
saturn_pos = saturn.at(tnow).position.au
uranus_pos = uranus.at(tnow).position.au
neptune_pos = neptune.at(tnow).position.au
pluton_pos = pluton.at(tnow).position.au

sun_vel = sun.at(tnow).velocity.au_per_d
earth_vel = earth.at(tnow).velocity.au_per_d
mars_vel = mars.at(tnow).velocity.au_per_d
mercury_vel = mercury.at(tnow).velocity.au_per_d
venus_vel = venus.at(tnow).velocity.au_per_d
jupiter_vel = jupiter.at(tnow).velocity.au_per_d
saturn_vel = saturn.at(tnow).velocity.au_per_d
uranus_vel = uranus.at(tnow).velocity.au_per_d
neptune_vel = neptune.at(tnow).velocity.au_per_d
pluton_vel = pluton.at(tnow).velocity.au_per_d
```

 # Gorputzen posizioak gorde

```
x = np.array([sun_pos[0], mercury_pos[0], venus_pos[0], earth_pos[0],
              mars_pos[0], jupiter_pos[0], saturn_pos[0], uranus_pos[0],
              neptune_pos[0], pluton_pos[0]])
y = np.array([sun_pos[1], mercury_pos[1], venus_pos[1], earth_pos[1],
              mars_pos[1], jupiter_pos[1], saturn_pos[1], uranus_pos[1],
              neptune_pos[1], pluton_pos[1]])
z = np.array([sun_pos[2], mercury_pos[2], venus_pos[2], earth_pos[2],
              mars_pos[2], jupiter_pos[2], saturn_pos[2], uranus_pos[2],
              neptune_pos[2], pluton_pos[2]])

x_all[:, 0] = x
y_all[:, 0] = y
z_all[:, 0] = z
```

```
# Gorputzen abiadurak gorde
vx = np.array([sun_vel[0],  mercury_vel[0], venus_vel[0], earth_vel[0],
               mars_vel[0], jupiter_vel[0], saturn_vel[0], uranus_vel[0],
               neptune_vel[0], pluton_vel[0]])
vy = np.array([sun_vel[1],  mercury_vel[1], venus_vel[1], earth_vel[1],
               mars_vel[1], jupiter_vel[1], saturn_vel[1], uranus_vel[1],
               neptune_vel[1], pluton_vel[1]])
vz = np.array([sun_vel[2],  mercury_vel[2], venus_vel[2], earth_vel[2],
               mars_vel[2], jupiter_vel[2], saturn_vel[2], uranus_vel[2],
               neptune_vel[2], pluton_vel[2]])

exet = time.time()

# t=1/2 aldiuneko abiadura lortzeko
ax, ay, az = grab.axyz(N, G, 0, m, x, y, z)  # Dij = 0, talkarik ez
vx = vx + ax*h/2
vy = vy + ay*h/2
vz = vz + az*h/2

# Periodoak gordetzeko
per = np.full([N], None)

for t in range(1, loop):
    # Leapfrog algoritmoaren hasieran, x(t=0), v(t=1/2) eta a(t=0) ditugu
    x = x + vx*h
    y = y + vy*h
    z = z + vz*h
    ax, ay, az = grab.axyz(N, G, 0, m, x, y, z)
    vx = vx + ax*h
    vy = vy + ay*h
    vz = vz + az*h

    # Orbita oso bat bada hasierako posiziotik True gorde
    per_bete = np.logical_and(abs(x - x_all[:, 0]) < eps,
                              abs(y - y_all[:, 0]) < eps,
                              abs(z - z_all[:, 0]) < eps)
    if t % jump == 0:
        x_all[:, int(t/jump)] = x
        y_all[:, int(t/jump)] = y
        z_all[:, int(t/jump)] = z

    # Periodoa kalkulatzeko (t>10 hasierako posizioak kontuan ez hartzeko)
    for i in range(0, N):
        if per_bete[i] == True and per[i] == None and t > 10:
```

```
            per[i] = h*t

# Lurraren periodoarekin normalizatu
per = np.around(per.astype(np.double)/per[3], decimals=3)

#--------------------GRAFIKATZEA---------------------
axis = plt.axes(projection='3d', facecolor='black')

axis.plot3D(x_all[0, :], y_all[0, :], z_all[0, :], linewidth=0.01)
axis.plot3D(x_all[1, :], y_all[1, :], z_all[1, :], linewidth=0.1)
axis.plot3D(x_all[2, :], y_all[2, :], z_all[2, :], linewidth=0.1)
axis.plot3D(x_all[3, :], y_all[3, :], z_all[3, :], linewidth=0.1)
axis.plot3D(x_all[4, :], y_all[4, :], z_all[4, :], linewidth=0.1,
            label='Marte')
axis.plot3D(x_all[5, :], y_all[5, :], z_all[5, :], linewidth=1.5,
            label='Jupiter')
axis.plot3D(x_all[6, :], y_all[6, :], z_all[6, :], linewidth=1.5,
            label='Saturno')
axis.plot3D(x_all[7, :], y_all[7, :], z_all[7, :], linewidth=1.5,
            label='Urano')
axis.plot3D(x_all[8, :], y_all[8, :], z_all[8, :], linewidth=1.5,
            label='Neptuno')
axis.plot3D(x_all[9, :], y_all[9, :], z_all[9, :], linewidth=1.5,
            label='Pluton')

# Simulazioa egin den eguneko planeten posizioak markatu
axis.scatter(x_all[0, 0], y_all[0, 0], z_all[0, 0], s=0)
axis.scatter(x_all[1, 0], y_all[1, 0], z_all[1, 0], s=0)
axis.scatter(x_all[2, 0], y_all[2, 0], z_all[2, 0], s=0)
axis.scatter(x_all[3, 0], y_all[3, 0], z_all[3, 0], s=0)
axis.scatter(x_all[4, 0], y_all[4, 0], z_all[4, 0], s=0)
axis.scatter(x_all[5, 0], y_all[5, 0], z_all[5, 0])
axis.scatter(x_all[6, 0], y_all[6, 0], z_all[6, 0])
axis.scatter(x_all[7, 0], y_all[7, 0], z_all[7, 0])
axis.scatter(x_all[8, 0], y_all[8, 0], z_all[8, 0])
axis.scatter(x_all[9, 0], y_all[9, 0], z_all[9, 0])

axis.grid(False)
axis.set_axis_off()

plt.legend()
plt.show()

#----------Periodoak pantailan idatzi----------
print('Merkurioren periodoa ' + str(per[1]) + ' urte da.')
```

```
    print('Artizarraren periodoa ' + str(per[2]) + 'urte da.')
    print('Lurraren periodoa ' + str(per[3]) + ' urte da.')
    print('Marteren periodoa ' + str(per[4]) + ' urte da.')
    print('Jupiterren periodoa ' + str(per[5]) + ' urte da.')
    print('Saturnoren periodoa ' + str(per[6]) + ' urte da.')
    print('Uranoren periodoa ' + str(per[7]) + ' urte da.')
    print('Neptunoren periodoa ' + str(per[8]) + ' urte da.')
    print('Plutonen periodoa ' + str(per[9]) + ' urte da.\n\r')

    print('Exekuzio-denbora ' + str(round(time.time() - exet, 2)) +
          's izan da.')
```

B Talken kodea

Kode hau erabiltzeko ondorengo hiru paketeak inportatu behar dira:

```
import numpy as np
import random
import math
```

B.1 Talkak detektatzeko funtzioa

```
def talkaExis(N, Dij, x, y, z):
    """
        Talka jasango duten partikula bikoteak detektatu eta modu aleatorio
        batean partikula bikoteak ordenatzen dituen funtzioa.

    Jaso:
        N: partikula kopurua (int)
        Dij: partikulen erradioen batura (NXN tamainako np array-n,
                                          elementuak float)
        x: partikulen x koordenatuko posizioak (N tamainako np array-n,
                                                elementuak float)
        y: partikulen y koordenatuko posizioak (N tamainako np array-n,
                                                elementuak float)
        z: partikulen z koordenatuko posizioak (N tamainako np array-n,
                                                elementuak float)

    Itzuli:
        listOfCoordinates: tuplaz osatutako lista, tuplako osagaiak talka-
                           bikotean parte hartzen duten partikulen indizeak
                           dira: [(part1, part2), (part2, part5),....]
```

```
    """

    # Diagonaleko elementuen arteko talka eta talka bakoitza bi aldiz
    # ez kalkulatzeko
    eps = math.pow(10, -2)
    Rlagun = dist(N, x, y, z) + (Dij + eps)*np.tril(np.ones([N, N]))

    # talka gertatuko bada True gorde.
    talka = Rlagun < Dij

    # bere baitan True parametroa duten elementuen indizeak gorde tuplatan
    result = np.where(talka == True)
    listOfCoordinates = list(zip(result[0], result[1]))

    # tuplak ausaz berrordenatu
    random.shuffle(listOfCoordinates)
    return listOfCoordinates
```

Funtzio honek ondorengo funtzioa erabiltzen du:

```
def dist(N, x, y, z):
    """
        Partikulen arteko distantzia kalkulatzeko funtzioa.

    Jaso:
        N: partikula kopurua (int)
        x: partikulen x koordenatuko posizioak (N tamainako np array-n,
                                                elementuak float)
        y: partikulen y koordenatuko posizioak (N tamainako np array-n,
                                                elementuak float)
        z: partikulen z koordenatuko posizioak (N tamainako np array-n,
                                                elementuak float)

    Itzuli:
        R: partikulen arteko distantzia (NXN tamainako np array-n,
                                         elementuak float)
    """

    xij = np.transpose(np.ones([N, N])*x) - np.ones([N, N])*x
    yij = np.transpose(np.ones([N, N])*y) - np.ones([N, N])*y
    zij = np.transpose(np.ones([N, N])*z) - np.ones([N, N])*z
    R = np.power(np.power(xij, 2) + np.power(yij, 2) + np.power(zij, 2), 0.5)
    return R
```

B.2 Talken lista exekutatzeko funtzioak

Kode honetan agertzen diren eragiketen jatorria ulertzeko ikus 2.2.2. atala.

```
def talkaExekutatu(N, n, listOfCoor, m, x, y, z, vx, vy, vz):
    """
       Talka jasango duten partikula–bikoteen lista exekutatuko duen
       funtzioa hiru dimentsiotan.

    Jaso:
        N: partikula kopurua (int)
        n: talka–parametroa (float, [0,1] tarteko balio)
        listOfCoor: tuplaz osatutako lista, tuplako osagaiak talka–
                    bikotean parte hartzen duten partikulen indizeak
                    dira: [(part1, part2), (part3, part5),...]
        m: partikulen masa (N tamainako np array–n, elementuak float)
        x: partikulen x koordenatuko posizioak (N tamainako np array–n,
                                                elementuak float)
        y: partikulen y koordenatuko posizioak (N tamainako np array–n,
                                                elementuak float)
        z: partikulen z koordenatuko posizioak (N tamainako np array–n,
                                                elementuak float)
        vx: partikulen x koordenatuko abiadurak (N tamainako np array–n,
                                                 elementuak float)
        vy: partikulen y koordenatuko abiadurak (N tamainako np array–n,
                                                 elementuak float)
        vz: partikulen z koordenatuko abiadurak (N tamainako np array–n,
                                                 elementuak float)

    Itzuli:
        vx: partikulen x koordenatuko abiadura berriak (N tamainako np array–n,
                                                        elementuak float)
        vy: partikulen y koordenatuko abiadura berriak (N tamainako np array–n,
                                                        elementuak float)
        vz: partikulen z koordenatuko abiadura berriak (N tamainako np array–n,
                                                        elementuak float)

    """


    # i iterazio bakoitzean talka bat exekutatuko da
    for i in range(0, len(listOfCoor)):
        m1 = m[listOfCoor[i][0]]
        m2 = m[listOfCoor[i][1]]
        matrix = np.array(([m1, m2, 0, 0, 0, 0], [0, 0, m1, m2, 0, 0],
                           [0, 0, 0, 0, m1, m2], [1, —1, 0, 0, 0, 0],
```

```
                        [0, 0, 1, -1, 0, 0], [0, 0, 0, 0, 1, -1]),
                         dtype=float)

        # vx, vy eta vz behin baino gehiagotan indexatzea ekiditeko
        vx1 = vx[listOfCoor[i][0]]
        vx2 = vx[listOfCoor[i][1]]
        vy1 = vy[listOfCoor[i][0]]
        vy2 = vy[listOfCoor[i][1]]
        vz1 = vz[listOfCoor[i][0]]
        vz2 = vz[listOfCoor[i][1]]

        # Koordenatu bakoitzeko partikulen arteko distantzia kalkulatu
        rx = x[listOfCoor[i][0]] - x[listOfCoor[i][1]]
        ry = y[listOfCoor[i][0]] - y[listOfCoor[i][1]]
        rz = z[listOfCoor[i][0]] - z[listOfCoor[i][1]]

        # Talka ondorengo abiadura erlatiboak kalkulatu
        vxij, vyij, vzij = vErlatibo3D(n, rx, ry, rz, vx1 - vx2, vy1 - vy2,
                                       vz1-vz2)

        # Partikula bakoitzaren talka ondorengo abiadura kalkulatu
        b = np.array(([m1*vx1 + m2*vx2, m1*vy1 + m2*vy2, m1*vz1 + m2*vz2, vxij,
                       vyij, vzij]))
        sol = np.linalg.solve(matrix, b)

        # Talkako emaitzak gorde
        vx[listOfCoor[i][0]] = sol[0]
        vx[listOfCoor[i][1]] = sol[1]
        vy[listOfCoor[i][0]] = sol[2]
        vy[listOfCoor[i][1]] = sol[3]
        vz[listOfCoor[i][0]] = sol[4]
        vz[listOfCoor[i][1]] = sol[5]

    return vx, vy, vz
```

Funtzio honek ondorengo funtzioa erabiltzen du:

```
def vErlatibo3D(n, rx, ry, rz, vx, vy, vz):
    """
        Bi partikulen arteko talka ondorengo abiadura erlatiboa kalkulatzen
        duen funtzioa hiru dimentsiotan.

    Jaso:
        n: talka-parametroa (float, [0,1] tarteko balio)
        rx: partikulen arteko x koordenatuko distantzia (float)
        ry: partikulen arteko y koordenatuko distantzia (float)
```

```
        rz: partikulen arteko z koordenatuko distantzia (float)
        vx: partikulen arteko x koordenatuko abiadura talka aurretik (float)
        vy: partikulen arteko y koordenatuko abiadura talka aurretik (float)
        vz: partikulen arteko z koordenatuko abiadura talka aurretik (float)

    Itzuli:
        vem[0]: partikulen arteko x koordenatuko abiadura talka ondoren (float)
        vem[1]: partikulen arteko y koordenatuko abiadura talka ondoren (float)
        vem[2]: partikulen arteko z koordenatuko abiadura talka ondoren (float)

    """

    # Normalizazio faktoreak
    rnor = math.pow(rx*rx + ry*ry + rz*rz, -0.5)
    tnor1 = math.pow(math.pow(rx + rz, 2) + 2*ry*ry, -0.5)
    tnor2 = math.pow(math.pow((rz*rx + ry*ry + rz*rz), 2) +
                     math.pow(rz*rx + ry*ry + rx*rx, 2) +
                     math.pow((rx - rz)*ry, 2), -0.5)

    # Abiadura normala eta tangentziala kalkulatu
    P = np.array(([rx*rnor, ry*rnor, rz*rnor], [tnor1*ry, -tnor1*(rx + rz),
                  tnor1*ry], [-tnor2*(rz*rx + ry*ry + rz*rz),
                  tnor2*(rx - rz)*ry, tnor2*(rz*rx + ry*ry + rx*rx)]))
    vem = np.array([vx, vy, vz])
    vkor = np.matmul(P, vem)

    # Osagai normala talka ondoren aldatu
    vkor[0] = -n*vkor[0]
    vem = np.matmul(P.T, vkor)

    return vem[0], vem[1], vem[2]
```

B.3 Funtzio nagusia

Talkak ebatziko dituzten funtzioak modu errazean erabiltzeko, funtzio bakar bat inplementatu da ondoren programa nagusitik deitu ahal izateko.

```
def talkaEbatzi(N, Dij, n, m, x, y, z, vx, vy, vz):
    """
        Partikulen arteko talka detektatu eta exekutatzen duen funtzioa
        hiru dimentsiotan.

    Jaso:
        N: partikula kopurua (int)
        Dij: partikulen erradioen batura (NXN tamainako np array-n,
```

```
                                        elementuak float)
    n: talka-parametroa (float, [0,1] tarteko balio)
    m: partikulen masa (N tamainako np array-n, elementuak float)
    x: partikulen x koordenatuko posizioak (N tamainako np array-n,
                                            elementuak float)
    y: partikulen y koordenatuko posizioak (N tamainako np array-n,
                                            elementuak float)
    z: partikulen z koordenatuko posizioak (N tamainako np array-n,
                                            elementuak float)
    vx: partikulen x koordenatuko abiadurak (N tamainako np array-n,
                                             elementuak float)
    vy: partikulen y koordenatuko abiadurak (N tamainako np array-n,
                                             elementuak float)
    vz: partikulen z koordenatuko abiadurak (N tamainako np array-n,
                                             elementuak float)

Itzuli:
    vx: partikulen x koordenatuko abiadura berriak (N tamainako np array-n,
                                                    elementuak float)
    vy: partikulen y koordenatuko abiadura berriak (N tamainako np array-n,
                                                    elementuak float)
    vz: partikulen z koordenatuko abiadura berriak (N tamainako np array-n,
                                                    elementuak float)
"""
listOfCoor = talkaExis(N, Dij, x, y, z)
vx, vy, vz = talkaExekutatu(N, n, listOfCoor, m, x, y, z, vx, vy, vz)
return vx, vy, vz
```

B.4 Talka-parametroaren aldaketa

```
import numpy as np
import matplotlib.pyplot as plt
import Talkak3DOrokor as talk
import random




if __name__ == "__main__":
    N = 2
    D = np.empty((N))
    D.fill(0.01)
    Dij = (np.transpose(np.ones([N, N])*D) + np.ones([N, N])*D)/2
    m = np.empty((N))
    m.fill(1.0)
```

```
n_list = np.array([1, 0.9, 0.4, 0.0])
while True:
    x1 = random.random()
    x2 = random.random()
    if abs(x1 — x2) > D[0]:
        break

v1 = random.random()
v2 = random.random()
sigma = (v1 + v2)/2
h = D[0]/(500*sigma)
print(h)
loop = int((x1 + x2)/sigma/h)
for n in n_list:

    x = np.array([x1, —x2])
    y = np.array([0.0, 0.0])
    z = np.array([0.0, 0.0])

    vx = np.array([—v1, v2])
    vy = np.array([0.0, 0.0])
    vz = np.array([0.0, 0.0])

    x_all = np.empty((N, loop), dtype=float)
    y_all = np.empty((N, loop), dtype=float)
    z_all = np.empty((N, loop), dtype=float)

    ez = np.empty((loop), dtype=float)
    x_all[:, 0] = x
    y_all[:, 0] = y
    z_all[:, 0] = z

    ez[0] = talk.energiaZinetiko(m, vx, vy, vz)

    for t in range(1, loop):
        vx, vy, vz = talk.talkaEbatzi(N, Dij, n, m, x, y, z, vx, vy, vz)
        x = x + vx*h
        y = y + vy*h
        z = z + vz*h

        x_all[:, t] = x
        y_all[:, t] = y
        z_all[:, t] = z
        ez[t] = talk.energiaZinetiko(m, vx, vy, vz)
```

```
        denbora = np.linspace(0, loop-1, num=loop, dtype=float)
        plt.plot(denbora, ez, label='n=' + str(n), linewidth=1.5)
    plt.title('h =' + str(round(h, 6)))
    plt.xlabel('Denbora')
    plt.ylabel('Energia Zinetikoa')
    plt.legend()
    plt.show()
```

B.5 Kutxa Klasea

```
import numpy as np

class Kutxa():

    def __init__(self, x, y, z, aldea):
        """
            Kutxa klaseko objektu bat hasieratzeko aldagaiak:

            x: kutxaren zentroaren posizioaren x koordenatua (float)
            y: kutxaren zentroaren posizioaren y koordenatua (float)
            z: kutxaren zentroaren posizioaren z koordenatua (float)
            aldea: kutxaren aldea (float)

        """


        self.zentroa = (x, y, z)
        self.aldea = aldea


    def txokatu(self, x, y, z):
        """
            Partikulek kutxako pareten kontra jasandako talkak detektatzen
            dituen metodoa.

        Jaso:
            x: partikulen x koordenatuko posizioak (N tamainako np array-a,
                                                    elementuak float)
            y: partikulen y koordenatuko posizioak (N tamainako np array-a,
                                                    elementuak float)
            z: partikulen z koordenatuko posizioak (N tamainako np array-a,
                                                    elementuak float)
        Itzuli:
            txokea: partikula batek pareten kontra talkarik jasaten badu True
                    itzuli (N tamainako np array-a, elementuak float)
        """
```

```
        txokeax = np.where(np.logical_and(x < self.zentroa[0] + self.aldea/2.0,
                           x > self.zentroa[0] - self.aldea/2.0),
                           True, False)
        txokeay = np.where(np.logical_and(y < self.zentroa[1] + self.aldea/2.0,
                           y > self.zentroa[1] - self.aldea/2.0),
                           True, False)
        txokeaz = np.where(np.logical_and(z < self.zentroa[2] + self.aldea/2.0,
                           z > self.zentroa[2] - self.aldea/2.0),
                           True, False)
        txokea = np.logical_not(txokeax*txokeay*txokeaz)
        return txokea

    def KutxaTxokea(self, n, x, y, z, vx, vy, vz):
        """
```

Partikulek kutxako pareten kontra talka egitean abiadurak nola aldatzen diren kalkulatzen duen metodoa.

Jaso:

n: talka-parametroa (float, [0,1] tarteko balio)
x: partikulen x koordenatuko posizioak (N tamainako np array-n, elementuak float)
y: partikulen y koordenatuko posizioak (N tamainako np array-n, elementuak float)
z: partikulen z koordenatuko posizioak (N tamainako np array-n, elementuak float)
vx: partikulen x koordenatuko abiadurak (N tamainako np array-n, elementuak float)
vy: partikulen y koordenatuko abiadurak (N tamainako np array-n, elementuak float)
vz: partikulen z koordenatuko abiadurak (N tamainako np array-n, elementuak float)

Itzuli:

vx: partikulen x koordenatuko abiadura berriak (N tamainako np array-n, elementuak float)
vy: partikulen y koordenatuko abiadura berriak (N tamainako np array-n, elementuak float)
vz: partikulen z koordenatuko abiadura berriak (N tamainako np array-n, elementuak float)

```
        """
        talka = self.txokatu(x, y, z)
```

```
        result = np.where(talka == True)
        try:
            for i in np.nditer(result):
                if x[i] > self.zentroa[0] + self.aldea/2.0:
                    vx[i] = -n*vx[i]
                elif x[i] < self.zentroa[0] - self.aldea/2.0:
                    vx[i] = -n*vx[i]
                if y[i] > self.zentroa[1] + self.aldea/2.0:
                    vy[i] = -n*vy[i]
                elif y[i] < self.zentroa[1] - self.aldea/2.0:
                    vy[i] = -n*vy[i]
                if z[i] > self.zentroa[2] + self.aldea/2.0:
                    vz[i] = -n*vz[i]
                elif z[i] < self.zentroa[2] - self.aldea/2.0:
                    vz[i] = -n*vz[i]
        except:
            pass
        return vx, vy, vz
```

B.6 Kutxaren simulazioa

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import time
import Paretak as pareta
from itertools import count
from matplotlib.animation import FuncAnimation, PillowWriter
import os
import sys
currentFile = os.path.dirname(os.path.realpath(__file__))
parentFile = os.path.dirname(currentFile)
sys.path.insert(0, parentFile)
import Talkak3DOrokor as talk


def update_plot1(i, xmax, xmin, ymax, ymin, zmax, zmin):
    axis.cla()
    axis.set_xlim3d(xmin, xmax)
    axis.set_ylim3d(ymin, ymax)
    axis.set_zlim3d(zmin, zmax)
    # axis.grid(False)
    # axis.set_axis_off()
```

```
    rx = x_all[:, i]
    ry = y_all[:, i]
    rz = z_all[:, i]
    axis.scatter(rx[0], ry[0], rz[0], color='red', s=10)
    axis.plot(x_all[0, 0:i+1], y_all[0, 0:i+1], z_all[0, 0:i+1], color='red', linewidth=1)
    axis.scatter(rx[1:N], ry[1:N], rz[1:N], color='blue',s=10)


if __name__ == "__main__":
    karpeta = 0
    while True:
        try:
            os.mkdir('Emaitza' + str(karpeta))
            karpetaIzena = 'Emaitza' + str(karpeta)
            break
        except OSError:
            karpeta = karpeta + 1

    if os.name == 'posix':
        slash = '/'
    else:
        slash = '\\'

    fitx1 = open(str(karpetaIzena) + str(slash) + 'Informazioa' +
                 str(karpeta) + '.txt', 'w')
    aldea = 1
    kutxa = pareta.Kutxa(0.0, 0.0, 0.0, aldea)
    N = 50
    D = np.empty((N))
    D.fill(0.06)
    m = np.empty((N))
    m.fill(1)
    n = 1
    x = np.random.choice([-1, 1], size=N)*aldea*np.random.random_sample(N)/2
    y = np.random.choice([-1, 1], size=N)*aldea*np.random.random_sample(N)/2
    z = np.random.choice([-1, 1], size=N)*aldea*np.random.random_sample(N)/2

    vx = np.random.normal(loc=0.0, scale=10, size=N)
    vy = np.random.normal(loc=0.0, scale=10, size=N)
    vz = np.random.normal(loc=0.0, scale=10, size=N)

    loop = 100  # Zenbat denbora pausu kalkulatuko diren
    jump = 1  # Zenbat datu grafikatuko diren
    h = 0.005
    x_all = np.empty((N, int(loop/jump)), dtype=float)
```

```
y_all = np.empty((N, int(loop/jump)), dtype=float)
z_all = np.empty((N, int(loop/jump)), dtype=float)

x_all[:, 0] = x
y_all[:, 0] = y
z_all[:, 0] = z


exet = time.time()

fitx1.write('N = ' + str(N) + '\n')
fitx1.write('n = ' + str(n) + '\n')
fitx1.write('aldea = ' + str(aldea) + '\n')
fitx1.write('h = ' + str(h) + '\n')
fitx1.write('loop = ' + str(loop) + '\n')

fitx1.write('\n----------Masa----------\n')
np.savetxt(fitx1, m, delimiter=' ', newline=' ')
fitx1.write('\n----------Diametroa----------\n')
np.savetxt(fitx1, D, delimiter=' ', newline=' ')
fitx1.write('\n----------Hasierako x----------\n')
np.savetxt(fitx1, x, delimiter=' ', newline=' ')
fitx1.write('\n----------Hasierako y----------\n')
np.savetxt(fitx1, y, delimiter=' ', newline=' ')
fitx1.write('\n----------Hasierako z----------\n')
np.savetxt(fitx1, z, delimiter=' ', newline=' ')
fitx1.write('\n----------Hasierako vx----------\n')
np.savetxt(fitx1, vx, delimiter=' ', newline=' ')
fitx1.write('\n----------Hasierako vy----------\n')
np.savetxt(fitx1, vy, delimiter=' ', newline=' ')
fitx1.write('\n----------Hasierako vz----------\n')
np.savetxt(fitx1, vz, delimiter=' ', newline=' ')

fitx2 = open(str(karpetaIzena) + str(slash) + 'Datuak' +
            str(karpeta) + '.txt', 'w')
np.savetxt(fitx2, x, delimiter=' ', newline=' ')
np.savetxt(fitx2, y, delimiter=' ', newline=' ')
np.savetxt(fitx2, z, delimiter=' ', newline=' ')
fitx2.write('\n')
Dij = (np.transpose(np.ones([N, N])*D) + np.ones([N, N])*D)/2
for t in range(1, loop):

    vx, vy, vz = talk.talkaEbatzi(N, Dij, n, m, x, y, z, vx, vy, vz)
    vx, vy, vz = kutxa.KutxaTxokea(n, x, y, z, vx, vy, vz)
    x = x + vx*h
```

```
    y = y + vy*h
    z = z + vz*h
    np.savetxt(fitx2, x, delimiter=' ', newline=' ')
    np.savetxt(fitx2, y, delimiter=' ', newline=' ')
    np.savetxt(fitx2, z, delimiter=' ', newline=' ')
    fitx2.write('\n')

    if t % jump == 0:
        x_all[:, int(t/jump)] = x
        y_all[:, int(t/jump)] = y
        z_all[:, int(t/jump)] = z

#------------------GRAFIKATZEA---------------------

index = count()
fig = plt.figure()
xmin = -kutxa.aldea/2
xmax = kutxa.aldea/2
ymin = -kutxa.aldea/2
ymax = kutxa.aldea/2
zmin = -kutxa.aldea/2
zmax = kutxa.aldea/2
axis = plt.axes(projection='3d', facecolor='black')


#----------DATUAKGORDE---------------
fitx1.write('\n----------Azken x--------------\n')
np.savetxt(fitx1, x, delimiter=' ', newline=' ')
fitx1.write('\n----------Azken y--------------\n')
np.savetxt(fitx1, y, delimiter=' ', newline=' ')
fitx1.write('\n----------Azken z--------------\n')
np.savetxt(fitx1, z, delimiter=' ', newline=' ')
fitx1.write('\n----------Azken vx--------------\n')
np.savetxt(fitx1, vx, delimiter=' ', newline=' ')
fitx1.write('\n----------Azken vy--------------\n')
np.savetxt(fitx1, vy, delimiter=' ', newline=' ')
fitx1.write('\n----------Azken vz--------------\n')
np.savetxt(fitx1, vz, delimiter=' ', newline=' ')
fitx1.close()
fitx2.close()
print('Exekuzio-denbora ' + str(round(time.time() - exet, 2)) +
      's izan da.')
axis.cla()
axis.set_xlim3d(xmin, xmax)
axis.set_ylim3d(ymin, ymax)
```

```
axis.set_zlim3d(zmin, zmax)
axis.grid(True)
rx = x_all[:, loop-1]
ry = y_all[:, loop-1]
rz = z_all[:, loop-1]

axis.scatter(rx[0], ry[0], rz[0], color='red', s=20)
axis.plot(x_all[0, 0:loop], y_all[0, 0:loop], z_all[0, 0:loop], color='red', linewidth=1)
fig.savefig(str(karpetaIzena) + str(slash) + 'PartikulaBakarra' +
            str(karpeta) + '.png')
axis.cla()
axis.set_xlim3d(xmin, xmax)
axis.set_ylim3d(ymin, ymax)
axis.set_zlim3d(zmin, zmax)
axis.grid(True)
axis.scatter(rx[0], ry[0], rz[0], color='red', s=30)
axis.plot(x_all[0, 0:loop], y_all[0, 0:loop], z_all[0, 0:loop], color='red',
          linewidth=1)

axis.scatter(rx[1:N], ry[1:N], rz[1:N], color='blue', s=10)
fig.savefig(str(karpetaIzena) + str(slash) + 'PartikulaGuztiak' +
            str(karpeta) + '.png')

#----------------GIF-A SORTU----------------
axis.cla()
axis.grid(False)
axis.set_axis_off()
axis.set_xlim3d(xmin, xmax)
axis.set_ylim3d(ymin, ymax)
axis.set_zlim3d(zmin, zmax)
zein = np.linspace(0, loop-1, num=int(loop/jump), dtype=int)
franerate = 100
frame = iter(list(zein))
anim = FuncAnimation(fig, update_plot1, frames=frame, interval=1.0/franerate,
                     repeat=True, fargs= (xmax, xmin, ymax, ymin, zmax,
                                          zmin),  save_count=int(loop/jump))
plt.show()
```

C Leapfrog simulazio bateratua

Programa nagusiko denboraren garapena:

```
#Hasierako baldintzak ezarri
vx, vy, vz = talkaEbatzi(N, Dij, n, m, x, y, z, vx, vy, vz)
```

```
    ax, ay, az = axyz(N, G, Dij, m, x, y, z)
    vx = vx + ax*h/2
    vy = vy + ay*h/2
    vz = vz + az*h/2

    #Leapfrog algoritmoa
    for t in range(1, iterazio_kop):
        x = x + vx*h
        y = y + vy*h
        z = z + vz*h
        vx, vy, vz = talkaEbatzi(N, Dij, n, m, x, y, z, vx, vy, vz)
        ax, ay, az = axyz(N, G, Dij, m, x, y, z)
        vx = vx + ax*h
        vy = vy + ay*h
        vz = vz + az*h
```

D Gorputz baten grabitazio-kolapsoa

D.1 Programa nagusia

```
import numpy as np
import time
import sys
import os

currentFile = os.path.dirname(os.path.realpath(__file__))
parentFile = os.path.dirname(currentFile)
sys.path.insert(0, parentFile)
import Talkak3DOrokor as talk
import Grabitatea3DOrokor as grab


if __name__ == "__main__":
    N = 500
    D = 1.0
    n = 0.1
    G = 1.0
    mtot = 1.0
    sigma = 100
    m = np.full((N), mtot)
    x = np.random.normal(loc=0, scale=sigma, size=N)
    y = np.random.normal(loc=0, scale=sigma, size=N)
    z = np.random.normal(loc=0, scale=sigma, size=N)
```

```
vx = np.random.normal(loc=0.0, scale=0, size=N)
vy = np.random.normal(loc=0.0, scale=0, size=N)
vz = np.random.normal(loc=0.0, scale=0, size=N)

aber = time.time()
h = 0.2
loop = 1000
kont = 1
karpeta = 0

while True:
    try:
        os.mkdir('Emaitza' + str(karpeta))
        karpetaIzena = 'Emaitza' + str(karpeta)
        break
    except OSError:
        karpeta = karpeta + 1

if os.name == 'posix':
    slash = '/'
else:
    slash = '\\'

fitx1 = open(str(karpetaIzena) + str(slash) + 'Informazioa' +
             str(karpeta) + '.txt', 'w')

fitx1.write('N = ' + str(N) + '\n')
fitx1.write('D = ' + str(D) + '\n')
fitx1.write('n = ' + str(n) + '\n')
fitx1.write('G = ' + str(G) + '\n')
fitx1.write('h = ' + str(h) + '\n')
fitx1.write('loop = ' + str(loop) + '\n')
fitx1.write('mtot = ' + str(mtot) + '\n')
fitx1.write('xmax = ' + str(np.amax(x)) + '\n')
fitx1.write('xmin = ' + str(np.amin(x)) + '\n')
fitx1.write('ymax = ' + str(np.amax(y)) + '\n')
fitx1.write('ymin = ' + str(np.amin(y)) + '\n')
fitx1.write('zmax = ' + str(np.amax(z)) + '\n')
fitx1.write('zmin = ' + str(np.amin(z)) + '\n')

fitx1.write('\n----------Hasierako x----------\n')
np.savetxt(fitx1, x, delimiter=' ', newline=' ')
fitx1.write('\n----------Hasierako y----------\n')
np.savetxt(fitx1, y, delimiter=' ', newline=' ')
```

```
fitx1.write('\n----------Hasierako z----------\n')
np.savetxt(fitx1, z, delimiter=' ', newline=' ')
fitx1.write('\n----------Hasierako vx----------\n')
np.savetxt(fitx1, vx, delimiter=' ', newline=' ')
fitx1.write('\n----------Hasierako vy----------\n')
np.savetxt(fitx1, vy, delimiter=' ', newline=' ')
fitx1.write('\n----------Hasierako vz----------\n')
np.savetxt(fitx1, vz, delimiter=' ', newline=' ')


fitx2 = open(str(karpetaIzena) + str(slash) + 'Datuak' +
            str(karpeta) + '.txt', 'w')
np.savetxt(fitx2, x, delimiter=' ', newline=' ')
np.savetxt(fitx2, y, delimiter=' ', newline=' ')
np.savetxt(fitx2, z, delimiter=' ', newline=' ')
fitx2.write('\n')

Dij = (np.transpose(np.ones([N, N])*D) + np.ones([N, N])*D)/2
```

 # Hasierako baldintzak ezarri

```
vx, vy, vz = talk.talkaEbatzi(N, Dij, n, m, x, y, z, vx, vy, vz)
ax, ay, az = grab.axyz(N, G, Dij, m, x, y, z)
vx = vx + ax*h/2
vy = vy + ay*h/2
vz = vz + az*h/2
```

 # Leapfrog algoritmoa

```
for t in range(1, loop):
    x = x + vx*h
    y = y + vy*h
    z = z + vz*h
    vx, vy, vz = talk.talkaEbatzi(N, Dij, n, m, x, y, z, vx, vy, vz)
    ax, ay, az = grab.axyz(N, G, Dij, m, x, y, z)
    vx = vx + ax*h
    vy = vy + ay*h
    vz = vz + az*h

    np.savetxt(fitx2, x, delimiter=' ', newline=' ')
    np.savetxt(fitx2, y, delimiter=' ', newline=' ')
    np.savetxt(fitx2, z, delimiter=' ', newline=' ')
    fitx2.write('\n')
    if t % int(loop/5) == 0:
        print(str(loop-t) + "  iterazio geratzen dira.")

fitx1.write('\nTime = ' + str(time.time() - aber))
```

```
    fitx1.write('\n----------Azken x----------\n')
    np.savetxt(fitx1, x, delimiter=' ', newline=' ')
    fitx1.write('\n----------Azken y----------\n')
    np.savetxt(fitx1, y, delimiter=' ', newline=' ')
    fitx1.write('\n----------Azken z----------\n')
    np.savetxt(fitx1, z, delimiter=' ', newline=' ')
    fitx1.write('\n----------Azken vx----------\n')
    np.savetxt(fitx1, vx, delimiter=' ', newline=' ')
    fitx1.write('\n----------Azken vy----------\n')
    np.savetxt(fitx1, vy, delimiter=' ', newline=' ')
    fitx1.write('\n----------Azken vz----------\n')
    np.savetxt(fitx1, vz, delimiter=' ', newline=' ')
    fitx1.close()
    fitx2.close()
    print('Programa honek ' + str(time.time() - aber) + 's behar izan ditu exekutatzeko')
```

D.2 Simulazioa luzatzeko programa

```
import numpy as np
import time
import sys
import os
import Talkak3DOrokor as talk
import Grabitatea3DOrokor as grab


if __name__ == "__main__":

    aber = time.time()
    karpeta = 4  # Zein simulazio luzatu nahi den
    loop = 10000  # Simulazioa zenbat luzatu nahi den
    kont = 1
    karpetaIzena = 'Emaitza' + str(karpeta)

    if os.name == 'posix':
        slash = '/'
    else:
        slash = '\\'
    fitx1 = open(str(karpetaIzena) + str(slash) + 'Informazioa' +
                 str(karpeta) + '.txt', 'r')
    esaldi = fitx1.readline()
    N = int((esaldi.split())[2])
    esaldi = fitx1.readline()
```

```
D = float((esaldi.split())[2])
esaldi = fitx1.readline()
n = float((esaldi.split())[2])
esaldi = fitx1.readline()
G = float((esaldi.split())[2])
esaldi = fitx1.readline()
h = float((esaldi.split())[2])
esaldi = fitx1.readline()
loop0 = int((esaldi.split())[2])
esaldi = fitx1.readline()
mtot = float((esaldi.split())[2])
m = np.full((N), mtot)
while True:
    esaldi = fitx1.readline()
    if esaldi == '----------Hasierako x----------\n':
        esaldi = fitx1.readline()
        x0 = esaldi.split()
        x0 = np.array(x0, dtype=float)
    elif esaldi == '----------Hasierako y----------\n':
        esaldi = fitx1.readline()
        y0 = esaldi.split()
        y0 = np.array(y0, dtype=float)
    elif esaldi == '----------Hasierako z----------\n':
        esaldi = fitx1.readline()
        z0 = esaldi.split()
        z0 = np.array(z0, dtype=float)
    elif esaldi == '----------Hasierako vx----------\n':
        esaldi = fitx1.readline()
        vx0 = esaldi.split()
        vx0 = np.array(vx0, dtype=float)
    elif esaldi == '----------Hasierako vy----------\n':
        esaldi = fitx1.readline()
        vy0 = esaldi.split()
        vy0 = np.array(vy0, dtype=float)
    elif esaldi == '----------Hasierako vz----------\n':
        esaldi = fitx1.readline()
        vz0 = esaldi.split()
        vz0 = np.array(vz0, dtype=float)
        esaldi = fitx1.readline()
        Time = float((esaldi.split())[2])
    elif esaldi == '----------Azken x----------\n':
        esaldi = fitx1.readline()
        x = esaldi.split()
        x = np.array(x, dtype=float)
    elif esaldi == '----------Azken y----------\n':
```

```
            esaldi = fitx1.readline()
            y = esaldi.split()
            y = np.array(y, dtype=float)
        elif esaldi == '----------Azken z----------\n':
            esaldi = fitx1.readline()
            z = esaldi.split()
            z = np.array(z, dtype=float)
        elif esaldi == '----------Azken vx----------\n':
            esaldi = fitx1.readline()
            vx = esaldi.split()
            vx = np.array(vx, dtype=float)
        elif esaldi == '----------Azken vy----------\n':
            esaldi = fitx1.readline()
            vy = esaldi.split()
            vy = np.array(vy, dtype=float)
        elif esaldi == '----------Azken vz----------\n':
            esaldi = fitx1.readline()
            vz = esaldi.split()
            vz = np.array(vz, dtype=float)
            break
    fitx1.close()

    fitx2 = open(str(karpetaIzena) + str(slash) + 'Datuak' +
                 str(karpeta) + '.txt', 'a')

    Dij = (np.transpose(np.ones([N, N])*D) + np.ones([N, N])*D)/2
    for t in range(0, loop):
        x = x + vx*h
        y = y + vy*h
        z = z + vz*h
        vx, vy, vz = talk.talkaEbatzi(N, Dij, n, m, x, y, z, vx, vy, vz)
        ax, ay, az = grab.axyz(N, G, Dij, m, x, y, z)
        vx = vx + ax*h
        vy = vy + ay*h
        vz = vz + az*h
        np.savetxt(fitx2, x, delimiter=' ', newline=' ')
        np.savetxt(fitx2, y, delimiter=' ', newline=' ')
        np.savetxt(fitx2, z, delimiter=' ', newline=' ')
        fitx2.write('\n')
        if t % int(loop/5) == 0:
            print(str(loop-t) + "  iterazio geratzen dira.")

    fitx1 = open(str(karpetaIzena) + str(slash) + 'Informazioa' +
                 str(karpeta) + '.txt', 'w')
    fitx1.write('N = ' + str(N) + '\n')
```

```
fitx1.write('D = ' + str(D) + '\n')
fitx1.write('n = ' + str(n) + '\n')
fitx1.write('G = ' + str(G) + '\n')
fitx1.write('h = ' + str(h) + '\n')
fitx1.write('loop = ' + str(loop + loop0) + '\n')
fitx1.write('mtot = ' + str(mtot) + '\n')
fitx1.write('xmax = ' + str(np.amax(x0)) + '\n')
fitx1.write('xmin = ' + str(np.amin(x0)) + '\n')
fitx1.write('ymax = ' + str(np.amax(y0)) + '\n')
fitx1.write('ymin = ' + str(np.amin(y0)) + '\n')
fitx1.write('zmax = ' + str(np.amax(z0)) + '\n')
fitx1.write('zmin = ' + str(np.amin(z0)) + '\n')

fitx1.write('\n----------Hasierako x----------\n')
np.savetxt(fitx1, x0, delimiter=' ', newline=' ')
fitx1.write('\n----------Hasierako y----------\n')
np.savetxt(fitx1, y0, delimiter=' ', newline=' ')
fitx1.write('\n----------Hasierako z----------\n')
np.savetxt(fitx1, z0, delimiter=' ', newline=' ')
fitx1.write('\n----------Hasierako vx----------\n')
np.savetxt(fitx1, vx0, delimiter=' ', newline=' ')
fitx1.write('\n----------Hasierako vy----------\n')
np.savetxt(fitx1, vy0, delimiter=' ', newline=' ')
fitx1.write('\n----------Hasierako vz----------\n')
np.savetxt(fitx1, vz0, delimiter=' ', newline=' ')
fitx1.write('\nTime = ' + str(Time + time.time() - aber))
fitx1.write('\n----------Azken x----------\n')
np.savetxt(fitx1, x, delimiter=' ', newline=' ')
fitx1.write('\n----------Azken y----------\n')
np.savetxt(fitx1, y, delimiter=' ', newline=' ')
fitx1.write('\n----------Azken z----------\n')
np.savetxt(fitx1, z, delimiter=' ', newline=' ')
fitx1.write('\n----------Azken vx----------\n')
np.savetxt(fitx1, vx, delimiter=' ', newline=' ')
fitx1.write('\n----------Azken vy----------\n')
np.savetxt(fitx1, vy, delimiter=' ', newline=' ')
fitx1.write('\n----------Azken vz----------\n')
np.savetxt(fitx1, vz, delimiter=' ', newline=' ')
fitx1.close()
fitx2.close()
print('Programa honek ' + str(time.time() - aber) +
      's behar izan ditu exekutatzeko')
```

D.3 GIF animatua sortzeko programa

```
import numpy as np
from matplotlib.animation import FuncAnimation, PillowWriter
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from itertools import count
from os import name


def update_plot(i, xmax, xmin, ymax, ymin, zmax, zmin):
    ax.cla()
    ax.set_xlim3d(xmin, xmax)
    ax.set_ylim3d(ymin, ymax)
    ax.set_zlim3d(zmin, zmax)
    x = r[i, 0:N]
    y = r[i, N:2*N]
    z = r[i, 2*N:3*N]
    ax.scatter(x, y, z, c=colors)


if __name__ == "__main__":

    if name == 'posix':
        slash = '/'
    else:
        slash = '\\'
    karpeta = 7  # Karpetaren zenbakia finkatu
    irudiKop = 500  # Zenbat irudiz eratutako gif–a nahi dudan
    franerate = 100  # Zenbat fotograma milisegundoko
    karpetaIzena = 'Emaitza' + str(karpeta)

    fitx1 = open(str(karpetaIzena) + str(slash) + 'Informazioa' +
                 str(karpeta) + '.txt', 'r')
    kont = 0
    while True:
        esaldi = fitx1.readline()
        if esaldi.split()[0] == 'N':
            N = int((esaldi.split())[2])
        elif esaldi.split()[0] == 'loop':
            loop = int((esaldi.split())[2])
        elif esaldi.split()[0] == 'xmax':
            xmax = 1.1*float((esaldi.split())[2])
            esaldi = fitx1.readline()
            xmin = 1.1*float((esaldi.split())[2])
```

```
            esaldi = fitx1.readline()
            ymax = 1.1*float((esaldi.split())[2])
            esaldi = fitx1.readline()
            ymin = 1.1*float((esaldi.split())[2])
            esaldi = fitx1.readline()
            zmax = 1.1*float((esaldi.split())[2])
            esaldi = fitx1.readline()
            zmin = 1.1*float((esaldi.split())[2])
            break
        kont = kont + 1

    fitx1.close()

    filename2 = karpetaIzena + slash + 'Datuak' + str(karpeta) + '.txt'
    r = np.array(np.loadtxt(filename2, dtype=float))



    colors = np.linspace(0, 1, N)
    colors = colors.tolist()
    fig = plt.figure()
    ax = plt.axes(projection='3d')
    ax.set_xlim3d(xmin, xmax)
    ax.set_ylim3d(ymin, ymax)
    ax.set_zlim3d(zmin, zmax)
    zein = np.linspace(0, loop-1, num=irudiKop, dtype=int)
    frame = iter(list(zein))
    anim = FuncAnimation(fig, update_plot, frames=frame, interval=1.0/franerate,
                         repeat=True, fargs= (xmax, xmin, ymax, ymin, zmax,
                                              zmin),  save_count=irudiKop)
    plt.show()
    writergif = PillowWriter(fps=franerate)
    anim.save(karpetaIzena + slash + 'Gif' + str(karpeta) +'.gif',
              writer=writergif)
```