Tampereen yliopisto

Jimi Asikainen

GALTONIN KUUTIO

Kolmiulotteinen Galtonin lauta ja moniulotteinen normaalijakauma

Kandidaatintyö
Tekniikan ja luonnontieteiden tiedekunta
Tarkastaja: Yliopistonlehtori Terhi Kaarakka
Joulukuu 2023

TIIVISTELMÄ

Jimi Asikainen: Galtonin kuutio
Kandidaatintyö
Tampereen yliopisto
Tekniikka ja luonnontieteet, TkK
Joulukuu 2023

Tämän työn tavoitteena on luoda Francis Galtonin suunnittelemasta kaksiulotteisesta Galtonin laudasta kolmiulotteinen malli, joka simuloituna tuottaa sillä toteutetuilla toistokokeilla samaa todennäköisyysjakaumaa noudattavan hajonnan kuin kaksiulotteinen lauta. Koska simulaatio tullaan toteuttamaan kaksiulotteisessa tilassa toimivan Galtonin laudan sijaan kolmiulotteisena, Galtonin laudan synnyttämä yhden muuttujan todennäköisyysjakauma tulee esiintymään simulaatiossa kahden muuttujan moniulotteisena jakaumana. Tavoitteena on myös selittää syy sille, miksi kyseisen todennäköisyysjakauma myötäilee niin kaksi- kuin kolmiulotteisessa mallissa normaalijakaumaa.

Työssä tutkitaan ensin Galtonin laudan konseptia informaatioteoriaa hyödyntäen. Sen avulla voidaan selittää, miksi toistokokeen aikainen kaoottinen tila johtaa toistuvasti järjestäytyneeseen lopputulokseen normaalijakauman muodossa. Tämän jälkeen tutustutaan Galtonin laudan ja Pascalin kolmion väliseen yhteyteen, sekä siihen, millä tavalla Pascalin kolmiota voidaan osaltaan käyttää ennustamaan ja selittämään normaalijakauman syntyä. Työn teoreettinen osuus päättyy Galtonin kuutiona työssä nimitetyn kolmiulotteisen Galtonin laudan idean esittelyyn, jossa Galtonin kaksiulotteiselle laudalle ominaisilla ilmiöillä havainnollistetaan kolmiulotteisessa tilassa toimivan mallin toimintaperiaatteita.

Työssä tarvittavaa Galtonin kuution simulointia varten laadittiin ohjelma MatLab-nimistä tietokoneohjelmistoa sekä samannimistä ohjelmointikieltä käyttäen. Simulaation malli luotiin Galtonin laudan ominaisuuksien pohjalta. Jo aiemmin työssä tehdyn rajauksen seurauksena simulaation toimintamalli perustettiin fysikaalisten voimien tarkastelun sijaan puhtaasti matemaattisten todennäköisyysmallien alaisen sattumanvaraisuuden varaan. Ohjelmasta tehtiin lisäksi käyttäjäystävällinen, jotta sen käyttö myös tutkielmasta ulkopuolisen henkilön toimesta olisi mahdollisimman vaivatonta.

Tuloksena saatujen arvojen pohjalta todetaan simuloidun Galtonin kuution mallin noudattavan tuottamassaan hajonnassa Galtonin laudan tapaan normaalijakaumaa. Päätelmää tukee havainto siitä, että toistokokeiden odotusarvo kummankin akselin suhteen lähenee sitä arvoa, jota sen normaalijakaumaa noudattavassa mallissa tulee teoriassa lähestyä. Toinen johtopäätöstä tukeva havainto on simulaatiolla saatu todennäköisyys toistokokeen yksittäisen toistokerran tuloksen päätymiselle enintään yhden keskihajonnan päähän odotusarvosta. Vaikka todennäköisyys lähestyykin arvoa, joka eroaa hieman teoreettisesti oikeasta arvosta, virheen syy saadaan selitettyä eikä sillä ole vaikutusta päätelmän oikeellisuuteen. Lisäksi otoskoon suurentamisen havaitaan kasvattavan hajonnan tuottaman graafin ulkoista yhdennäköisyyttä normaalijakauman kanssa. Suuremman otoskoon huomataan myös vähentävän toistokokeen jakauman odotusarvolle sekä keskihajonnalle mitattujen arvojen keskinäistä vaihtelevuutta.

Avainsanat: Francis Galton, Galtonin kuutio, kolmiulotteinen Galtonin lauta, kaksiulotteinen normaalijakauma, entropia

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -ohjelmalla.

ALKUSANAT

Kandidaatintyö sisältää ohjelman, jonka koodi löytyy liitteestä A. Ohjelma on kirjoittajan alusta loppuun laatima, eikä siinä olla käytetty muiden tekemiä osia. Ohjelma on laadittu siten, että se on lukijalle tai muulle sitä käyttämään kiinnostuneelle helposti käytettävissä sekä visuaalisesti miellyttävä. Tästä syystä siitä löytyy kandidaatintyön kappaleiden tai tulosten saamisen suhteen tarpeettomia asioita, kuten animaatiomahdollisuus kahdella eri animaatiolla sekä ohjelmoinnin kultaisten ohjenuorien mukainen pitkäjänteinen kommentointi ja käyttäjälle tarkoitettujen muuttujien laajempi selitys heti ohjelman alussa.

Koodi on laadittu käyttäen MatLab-ohjelmaa. Kyseiseen ohjelmaan kopioituna koodin ajaminen käynnistää käyttäjän valitsemien parametrien mukaisen simulaation. Lukijaa kannustetaan kokeilemaan ohjelmaa itse. Näin sekä tämän työn tulosten että Galtonin kuution idean hahmotus paranee. Simulaatio on verrattaen selkeästi tulkittavissa jo tähän kandidaatintyöhön tarkemmin tutustumatta.

Kiitokset tämän kandidaatintyön ohjaajalle Terhi Kaarakalle, joka omista kiireistään huolimatta löysi aikaa johtaa kesken kandidaatintyön suoritusta perheelliseksi tulleen kirjoittajan alempi korkeakoulututkinto joulun alla päätökseensä!

Tampereella, 20. joulukuuta 2023

Jimi Asikainen

SISÄLLYSLUETTELO

1. Johdanto 1
2. Kaksiulotteinen Galtonin lauta 2
2.1 Yleiskuvaus 2
2.2 Teoreettinen malli 3
2.3 Pascalin kolmio 9
3. Kolmiulotteinen Galtonin lauta 11
3.1 Idean esittely 11
3.2 Pascalin pyramidi 11
3.3 Ideaali sekä realistinen malli 11
3.4 Simulaatio 12
3.5 Tulokset. 13
3.6 Tulosten analysointia 14
4. Yhteenveto 17
Lähteet. 18
Liite A: Galtonin kuution simulaatio - MatLab-koodi 19

LYHENTEET JA MERKINNÄT

$\mathbb{R}$	reaaliluvut
$\mathbb{R}^+$	positiiviset reaaliluvut
$\sum_{i=m}^{n}$	summa, jossa i käy luvusta m lukuun n.
$\int$	integraali
ln	luonnollinen logaritmi $\log_e$
$\sqrt{x}$	luvun x neliöjuuri
$x \in A$	luku x on joukon A alkio
sin	trigonometrinen funktio sini
cos	trigonometrinen funktio kosini

1. JOHDANTO

Arkemme on täynnä sattumanvaraisia tapahtumia ja epätietoisuutta. Ihmisen luonteelle on jo historian alkuajoista ollut ominaista parantaa selviytymismahdollisuuksiaan tunnistamalla toistuvia aspekteja, joiden avulla ennustaa ennalta tuntemattoman tapahtuman lopputulos. Jos tapahtuman yksittäisiä lopputuloksia voidaan erotella, usean toiston jälkeen voidaan oppia tunnistamaan yleisin lopputulos ja todennäköisyys sille, kuinka useasti tapahtuma päätyy tuohon lopputulemaan. Matematiikan saralla tällaiseen empiiriseen tutkimukseen ja havainnointiin verrattavia työkaluja ovat *toistokokeet*, joissa useiden toistokertojen tuloksista voidaan esittää erilaisia malleja kerättyn informaatioon nojaten. Näiden mallien avulla voidaan pyrkiä ennustamaan tuntematon mahdollisimman todenmukaisesti myös jatkossa.

Jos toistokokeen vaihtoehtoja on kaksi, joista toisen todennäköisyys tunnetaan, voidaan *binomijakaumaa* käyttämällä ennustaa todennäköisyys toistokokeen eri tuloksille. Mitä todenmukaisempi on ennusteemme toisen vaihtoehdon todennäköisyydelle, sitä tarkemmin binomijakaumaa tarkastelemalla osataan ennustaa, kuinka monta kertaa kyseinen tulos saadaan kaikista toistojen määrästä. Esimerkkinä tällaisesta olisi vaikkapa kolikon heittäminen, jonka mahdollisia lopputuloksia on kaksi: kruuna tai klaava. Koska kummankin tuloksen todennäköisyyden tiedetään olevan yhtä suuri, binomijakauman avulla voidaan vastata esimerkiksi kysymykseen, kuinka todennäköisesti kymmenestä heitosta kolmen lopputulos on kruuna.

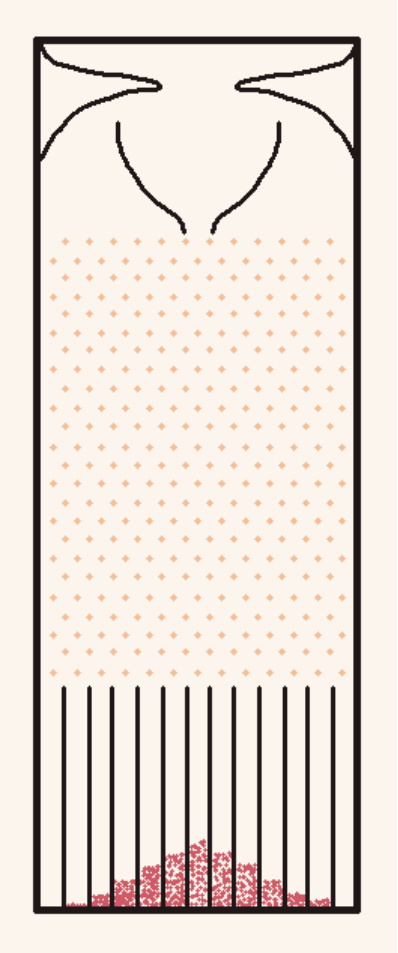
Monilla luonnossa esiintyvistä tuntemattomista asioista on kuitenkin mittaamaton määrä vaihtoehtoisia tuloksia. Satunnaisen, sademetsässä sijaitsevan köynnöksen pituutta ei voida kertoa kyllä-ei -kysymykseen vastaamalla, vaan mahdollisia pituuksia on lukematon määrä. On kuitenkin havaittu *normaalijakauman* olevan sellainen malli, jota tällaiset "luonnon toistokokeet" suurella todennäköisyydellä jollain tavalla noudattavat. Jos kaikkien sademetsän köynnösten pituuden *odotusarvo* sekä pituuksien välistä hajontaa kuvastava *varianssi* olisivat tiedossa, normaalijakauman avulla voitaisiin verrattaen luotettavasti ennustaa minkä tahansa pituisen köynnöksen esiintymistodennäköisyyttä luonnossa.

Tässä työssä lukijalta oletetaan ymmärrystä todennäköisyyksien syntymisestä, todennäköisyysjakaumista sekä erityisesti binomi- ja normaalijakaumista. Taustojen ymmärryksen tueksi sekä aiheesta muuten kiinnostuneille löytyy luettavaa esimerkiksi lähteestä [5].

2. KAKSIULOTTEINEN GALTONIN LAUTA

2.1 Yleiskuvaus

Francis Galton rakennutti vuonna 1873 itselleen laitteen, jolla hän demonstroisi luennoillaan Iso-Britannian kuninkaallisessa instituutissa yhteyttä binomi- ja normaalijakauman välillä. Laite koostui yläosan suppilosta, keskiosassa sijaitsevista lukuisista tapeista sekä alaosan korkeista ja kapeista lokeroista. Galton itse antoi laitteelleen nimeksi *quincunx*, joka vapaasti suomennettuna tarkoittaa viistoristiin asetettua. Tämä viittaa laudan tappeihin sekä niiden vinottaiseen asetteluun toisiinsa nähden.[3] Myöhemmin laitetta on nimitetty keksijänsä mukaan Galtonin laudaksi.



Kuva 2.1. *Quincunxin malli uudelleen piirrettynä Galtonin kuvailun mukaan. Kuvassa tapit kuvattu oranssilla ja helmet pinkillä värillä.*

Laitetta käytetään täyttämällä ensin yläosan suppilo helmillä tai vastaavilla pienillä ja pyöreillä

esineillä. Tämän jälkeen helmet päästetään vapaaksi, jolloin ne törmäilevät keskiosan tappeihin sekä toisiinsa matkalla alaspäin. Lopulta helmet päätyvät johonkin alaosan lokeroista muodostaen aina normaalijakaumaan verrattavan asetelman riippumatta toistojen määrästä.

2.2 Teoreettinen malli

Galtonin lautaa tarkasteltaessa herää kysymys siitä, miten ilmiön tarkastelu voidaan rajata puhtaasti matemaattiseksi. Kyseessähän on selvästi fysiikan lakeihin nojautuva ilmiö, jossa painovoima vetää helmiä kohti lokeroita ja helmien kimpoilu riippuu puhtaasti niiden kimmoisuudesta sekä muista ominaisuuksista.

Entropia fysiikassa kertoo epäjärjestyksen määrästä systeemissä, mutta Claude Shannonin esittämässä informaatioteoriassa sillä merkitään muuttujan kantaman informaation määrää. Systeemin entropialla voidaan mitata epätietoisuuttamme tulevasta. Jos entropia on nolla, tapahtuman tulos tiedetään ennalta täysin. Jos taas kokeen tulos on täysin ennustamaton, eikä yhdestäkään kokeen pysäytetystä tilasta ja siihen mennessä tapahtuneen antamasta informaatiosta voida ennustaa mitään lopputuloksesta, kokeen entropia on korkein mahdollinen.[7] Koska helmet ovat tippuessaan fysikaalisten voimien alaisia, on niillä tieto siitä, mitä reittiä ne ovat tippuneet. Kuitenkin, koska helmet törmäilevät matkallaan lukemattomia kertoja sekä toisiinsa että Galtonin laudan tappeihin, ei aikaisemmin kuljetun reitin perusteella voida tietää, mihin lokeroon helmi tulee tippumaan. Täten kokeen tulos hakee suurinta mahdollista entropiaa, eli epäjärjestyksen määrää.

Galtonin lauta on suljettu systeemi. Helmiä ei siis poistu tai tule lisää kokeen aikana. Siispä sillä on rajoitettu odotusarvo ja varianssi. Kokeesta ei kuitenkaan voida näiden lisäksi tietää mitään muuta. Todistetaan seuraavaksi, että normaalijakauma omaa suurimman mahdollisen entropian. Toisin sanoen, normaalijakauma on paras jakauma käytettäväksi tilanteessa, jossa ei voida tietää toistokokeesta mitään muuta kuin jakauman odotusarvo sekä varianssi. Seuraava todistus pohjautuu lähteeseen [4].

Määritetään ensin Shannonin entropian kaava diskreetille satunnaismuuttujalle.

Määritelmä 2.1. *Shannonin entropia* $H(X)$ diskreetille satunnaismuuttujalle X on

$$H(X) = \sum_{i=1}^{n} P(x_i) \log \frac{1}{P(x_i)}, \tag{2.1}$$

missä x_i on yksittäinen satunnaismuuttujan X mahdollisesti saama arvo, $P(x_i)$ todennäköisyys sille, että satunnaismuuttuja X saa arvon x_i ja n mahdollisten satunnaismuuttujan X saamien arvojen lukumäärä.

Kaavalla saatu entropian arvo on informaation määrää kuvaava luku, joka ei itsessään vielä kerro muuta kuin sen, onko tulos täysin ennalta tiedetty vai ei. Kuten aiemmin todettiin, entropian ollessa nolla on tuleva täysin tiedossa. Loogisesti tällaisesta tilanteesta tiedetään, että mahdollisia satunnaismuuttujan arvoja voi olla vain yksi. Kyseisen arvon todennäköisyyden täytyy siis myös

olla sata prosenttia, eli $P(x_1) = 1$. Tällöin logaritmin kannasta riippumatta $\log \frac{1}{P(x_1)} = 0$. Koska mahdollisia satunnaismuuttujan arvoja on vain yksi, Shannonin entropian kaavan summa on nolla.

Varman tuloksen tapauksessa voitaisiin myös ajatella satunnaismuuttujan mahdollisten arvojen määrän olevan suurempi, mutta muiden arvojen esiintyminen estettäisiin asettamalle niiden todennäköisyydeksi nolla. Määrittämäämme Shannonin entropian kaavaa käytettäessä ei kuitenkaan yhdenkään tapahtuman todennäköisyytenä voida käyttää nollaa. Tällöin logaritmin sisälle saataisiin $\frac{1}{0}$, mitä ei ole määritelty. Emme siis ota kaavaan mukaan diskreettejä tapahtumia, joiden todennäköisyys on nolla.

Jos taas entropia on suurempi kuin nolla, saadaan sille jokin nollasta poikkeava arvo. Saatu arvo riippuu kaavassa esiintyvän logaritmin kannasta. Jotta eri satunnaismuuttujien entropioita voidaan verrata toisiinta ja siten todeta normaalijakauman entropian olevan niistä suurin, on käytettävä johdonmukaisesti samaa kantaa. Tästä eteenpäin todistuksessa käytetään logaritmin kantana Neperin lukua, eli logaritmina luonnollista logaritmia. Näin yhtälöt ovat vertailukelpoisia ja myöhemmin käsiteltävässä normaalijakauman tiheysfunktiossa esiintyvän Neperin luvun sieventäminen helpottuu.

Jos mahdollisia diskreettejä tapahtumia asetetaan tietylle välille ääretön määrä äärettömän lähelle toisiaan, voidaan ajatella diskreetistä satunnaismuuttujasta saatavan jatkuva. Esimerkiksi satunnaismuuttuja $a \in \{1, 3, 6\}$ on diskreetti, sillä se voi saada vain kolme eri arvoa. Sen sijaan satunnaismuuttuja $b \in [0, 2]$ on jatkuva, sillä yksittäisen arvon vierestä löytyy aina äärettömän läheltä seuraava mahdollinen arvo. Jos tällaiselle äärettömän "tiheälle" diskreetille satunnaismuuttujalle laskettaisiin entropiaa, edellisessä kaavassa summattaisiin äärettömän monta arvoa tietyltä väliltä. Sen sijaan voidaan siirtyä käsittelemään jatkuvan satunnaismuuttujan tiheysfunktioiden integraalia ja jatkuvaa entropiaa, kuten tehdään seuraavassa kaavassa jatkuvan satunnaismuuttujan entropian laskemiseksi.

Määritelmä 2.2. *Jatkuva entropia* $h(X) : \mathbb{R} \to \mathbb{R}^+$ jatkuvalle satunnaismuuttujalle $X \in \mathbb{R}$ on

$$h(X) = \int f(x) \ln \frac{1}{f(x)} dx, \tag{2.2}$$

missä $f(x)$ on todennäköisyysjakauman tiheysfunktio. Entropiaa voidaan merkitä myös $h(f) = h(X)$, missä satunnaismuuttuja X noudattaa tiheysfunktiota $f(x)$.

Lauseen 2.11 todistusta varten määritetään seuraavaksi suhteellinen entropia. Sitä käytetään kahden eri todennäköisyysjakauman välisen eroavaisuuden mittaamiseksi.

Määritelmä 2.3. Olkoot $f, g : \mathbb{R} \to \mathbb{R}^+$ jatkuvia tiheysfunktioita. Tällöin niiden *suhteellinen entropia* $D(f||g)$ on

$$D(f||g) = \int f(x) \ln \frac{f(x)}{g(x)} dx. \tag{2.3}$$

Siinä missä jatkuvalla entropialla kuvataan satunnaismuuttujan informaation määrää, suhteellinen entropia kertoo, kuinka lähellä toisiaan kaksi satunnaismuuttujaa on hajontansa suhteen. Tätä

tukee huomio siitä, että jatkuvan entropian kaavassa tiheysfunktiolla jaetaan luku yksi, kun taas suhteellisen entropian kaavassa jakajana toimii toinen tiheysfunktio.

Apulauseen 2.7 todistusta varten määritetään jatkuvan satunnaismuuttujan odotusarvo.

Määritelmä 2.4. *Odotusarvo* $E\{f(X)\}$ jatkuvalle satunnaismuuttujalle $X \in \mathbb{R}$ on

$$E\{f(X)\} = \int f(x)P(x)dx, \tag{2.4}$$

missä $P(x)$ on satunnaismuuttujan X tiheysfunktio [10].

Todistukseen tarvitaan myös Jensenin epäyhtälöä. Esitetään se seuraavaksi lähteen [11] mukaan.

Lause 2.5. Jensenin epäyhtälö*: Olkoot $p_1, ..., p_n$ positiivisia lukuja, joiden summa on yksi, ja olkoon lisäksi reaalifunktio $f : \mathbb{R} \to \mathbb{R}$ konveksi. Tällöin*

$$\sum_{i=1}^{n} p_i f(x_i) \geq f(\sum_{i=1}^{n} p_i x_i). \tag{2.5}$$

Diskreetin satunnaismuuttujan odotusarvo saadaan summaamalla yhteen jokaisen mahdollisen satunnaismuuttujan arvon sekä sen todennäköisyyden tulo. Kaikkien todennäköisyyksien summan tulee olla yksi. Huomataan, että Jensenin epäyhtälön $\sum_{i=1}^{n} p_i x_i$ vastaa diskreetin satunnaismuuttujan odotusarvon yhtälöä. Samalla tavalla kuin entropian kaava saatiin diskreetin satunnaismuuttujan johdosta myös jatkuvalle satunnaismuuttujalle, saadaan Jensenin epäyhtälön kaava myös ikään kuin jatkuvalle epäyhtälölle jatkuvan satunnaismuuttujan odotusarvoa hyväksi käyttämällä. Esitetään yleistys, jonka todistus löytyy lähteestä [8].

Lause 2.6. Jensenin epäyhtälön yleistys*: Olkoon jatkuva reaalifunktio $f : \mathbb{R} \to \mathbb{R}$ konveksi. Tällöin*

$$E\{f(g(X)\} \geq f(E\{g(X)\}). \tag{2.6}$$

Suhteellinen entropia kahdelle eri jatkuvalle tiheysfunktiolle on aina suurempi kuin nolla [4]. Tästä saadaan ensimmäinen apulause.

Apulause 2.7. *Olkoot $f, g : \mathbb{R} \to \mathbb{R}^+$ jatkuvia tiheysfunktioita. Suhteelliselle entropialle on voimassa*

$$D(f||g) \geq 0. \tag{2.7}$$

Todistus. Määritelmän 2.3 mukaan suhteellinen entropia jatkuville tiheysfunktioille f ja g on

$$D(f||g) = \int f(x) \ln \frac{f(x)}{g(x)} dx. \tag{2.8}$$

Olkoon tiheysfunktio $q(x) = \ln \frac{f(x)}{g(x)}$. Sijoittamalla $q(x)$ yhtälöön saadaan

$$\int f(x) \ln \frac{f(x)}{g(x)} dx = \int f(x) q(x) dx. \tag{2.9}$$

Olkoon $f(x)$ jatkuvan satunnaismuuttujan X tiheysfunktio. Tällöin

$$\begin{aligned} \int q(x) f(x) dx &= E\{q(X)\} && \text{(2.4) perusteella} \\ &= E\{\ln \frac{f(X)}{g(X)}\} \\ &= E\{-(-\ln \frac{f(X)}{g(X)})\} \\ &= E\{-\ln (\frac{f(X)}{g(X)})^{-1}\} \\ &= E\{-\ln \frac{g(X)}{f(X)}\}. \end{aligned}$$

Tiheysfunktiot voivat saada vain positiivisia arvoja, joten $\frac{g(X)}{f(X)} \geq 0$. Siispä $\ln \frac{g(X)}{f(X)}$ on määritelty kaikilla $X \in \mathbb{R}$. Koska $-\ln x$ on jatkuva sekä konveksi funktio, Jensenin epäyhtälön (2.6) mukaan

$$\begin{aligned} E\{-\ln \frac{g(X)}{f(X)}\} &\geq -\ln E\{\frac{g(X)}{f(X)}\} \\ &= -\ln \int \frac{g(x)}{f(x)} f(x) dx \\ &= -\ln \int g(x) dx. \end{aligned}$$

Aivan kuten diskreetin satunnaismuuttujan saamien arvojen todennäköisyyksien summan tulee olla yksi, jatkuvan satunnaismuuttujan tiheysfunktion integraali on aina yksi. Täten

$$-\ln \int g(x) dx = -\ln 1 = 0. \tag{2.10}$$

□

Viimeistä apulausetta tarvitaan, jotta voidaan verrata kahden eri jatkuvan tiheysfunktion entropiaa toisiinsa. Sen avulla voidaan osoittaa, ettei löydy toista jatkuvaa jakaumaa, jonka entropia olisi normaalijakauman entropiaa suurempi.

Apulause 2.8. *Olkoot $f, g : \mathbb{R} \to \mathbb{R}^+$ jatkuvia tiheysfunktioita. Tällöin*

$$h(f) \leq -\int f(x) \ln g(x) dx. \tag{2.11}$$

Todistus. Suhteellisen entropia yhtälöön (2.3) sijoittamalla sekä integraalin ja logaritmin laskusääntöjä hyödyntämällä jatkuvien tiheysfunktioiden f ja g suhteellinen entropia saadaan muo-

toon

$$
\begin{aligned}
D(f||g) &= \int f(x) \ln \frac{f(x)}{g(x)} dx \\
&= \int f(x)(\ln f(x) - \ln g(x) dx \\
&= \int f(x) \ln f(x) dx - \int f(x) \ln g(x) dx \\
&= - \int f(x)(-\ln f(x)) dx - \int f(x) \ln g(x) dx \\
&= - \int f(x) \ln f(x)^{-1} dx - \int f(x) \ln g(x) dx \\
&= - \int f(x) \ln \frac{1}{f(x)} dx - \int f(x) \ln g(x) dx \\
&= -h(f) - \int f(x) \ln g(x) dx. && \text{(2.2) perusteella}
\end{aligned}
$$

Apulauseen 2.7 avulla saadaan epäyhtälö

$$
\begin{aligned}
0 \leq -h(f) - \int f(x) \ln g(x) dx \Leftrightarrow \\
h(f) \leq - \int f(x) \ln g(x) dx.
\end{aligned}
$$

□

Ennen viimeistä todistusta määritetään vielä varianssin yhtälö ja normaalijakauman tiheysfunktio. Näitä tarvitaan normaalijakauman entropian laskemiseksi sekä todistuksessa käytettävän epäyhtälön (2.11) sieventämiseksi. Määritelmien jälkeen todistetaan normaalijakauman olevan jakauma, jonka entropia on suurin tai yhtä suuri kuin millään muulla jakaumalla.

Määritelmä 2.9. *Jatkuvan jakauman varianssi* σ^2 on

$$\sigma^2 = \int f(x)(x-\mu)^2 dx, \tag{2.12}$$

missä $f(x)$ on jakauman tiheysfunktio [12].

Määritelmä 2.10. *Normaalijakauman tiheysfunktio* on

$$\phi(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{2.13}$$

missä μ on tiheysfunktion odotusarvo ja σ^2 on tiheysfunktion varianssi [6].

Lause 2.11. *Kaikista jakaumista, joilla on rajoitettu odotusarvo sekä varianssi, normaalijakauman entropia on suurin tai yhtä suuri kuin millään muulla jakaumalla.*

Todistus. Olkoon f mielivaltainen todennäköisyysjakauma ja olkoon ϕ normaalijakauma. Normaalijakauman entropia kaavan (2.2) mukaan on

$$h(\phi) = \int \phi(x) \ln \frac{1}{\phi(x)} dx. \tag{2.14}$$

Korvaamalla jälkimmäinen tiheysfunktio $\phi(x)$ normaalijakauman tiheysfunktiolla (2.13) saadaan

$$h(\phi) = \int \phi(x) \ln \frac{1}{\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}} dx.$$

Logaritmin laskusääntöjä käyttämällä yhtälö sievenee muotoon

$$\begin{aligned}
\int \phi(x) \ln \frac{1}{\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}} dx &= \int \phi(x)(\ln 1 - \ln \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}) dx \\
&= \int \phi(x)(-\ln \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}) dx \\
&= \int \phi(x)(-\ln \frac{1}{\sqrt{2\pi\sigma^2}} - \ln e^{-\frac{(x-\mu)^2}{2\sigma^2}}) dx.
\end{aligned}$$

Koska logaritmi on e-kantainen, saadaan sen sisällä olevan Neperin luvun potenssi kertoimeksi integraaliin. Saadaan yhtälö muotoon

$$\begin{aligned}
\int \phi(x)(-\ln \frac{1}{\sqrt{2\pi\sigma^2}} - \ln e^{-\frac{(x-\mu)^2}{2\sigma^2}}) dx &= \int \phi(x)(\ln \sqrt{2\pi\sigma^2} - (-\frac{(x-\mu)^2}{2\sigma^2})) dx \\
&= \int \phi(x)(\frac{1}{2} \ln 2\pi\sigma^2 + \frac{(x-\mu)^2}{2\sigma^2}) dx.
\end{aligned}$$

Jakamalla integraali summan kohdalta kahteen eri integraaliin saadaan

$$\begin{aligned}
\int \phi(x)(\frac{1}{2} \ln 2\pi\sigma^2 + \frac{(x-\mu)^2}{2\sigma^2}) dx &= \int \phi(x) \frac{1}{2} \ln 2\pi\sigma^2 + \int \phi(x) \frac{(x-\mu)^2}{2\sigma^2} dx \\
&= \frac{1}{2} \ln 2\pi\sigma^2 \int \phi(x) + \frac{1}{2\sigma^2} \int \phi(x)(x-\mu)^2 dx.
\end{aligned}$$

Tiheysfunktion integraali $\int \phi(x)$ koko määrittelyjoukkonsa yli on aina yksi. Jälkimmäinen integraali taas on määritelmän 2.9 mukaisesti varianssi. Normaalijakauman entropia on siis

$$\begin{aligned}
h(\phi) &= \frac{1}{2} \ln 2\pi\sigma^2 + \frac{1}{2\sigma^2} \sigma^2 \\
&= \frac{1}{2} \ln 2\pi\sigma^2 + \frac{1}{2} \\
&= \frac{1}{2}(\ln 2\pi\sigma^2 + 1).
\end{aligned}$$

Nyt kun normaalijakauman entropia on tiedossa, lähdetään todistamaan lause hyödyntäen epäyhtälöä (2.11), jossa esiintyvänä jälkimmäisenä tiheysfunktiona käytetään normaalijakauman tiheys-

funktiota. Epäyhtälö sievenee logaritmin laskusääntöjen avulla muotoon

$$\begin{aligned}
h(f) &\leq -\int f(x)\ln\phi(x)dx \\
&\leq -\int f(x)\ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}\right)dx \\
&\leq -\int f(x)\left(\ln\frac{1}{\sqrt{2\pi\sigma^2}}+\ln e^{-\frac{(x-\mu)^2}{2\sigma^2}}\right)dx \\
&\leq \int -f(x)\left(-\frac{1}{2}\ln 2\pi\sigma^2-\frac{(x-\mu)^2}{2\sigma^2}\right)dx \\
&\leq \int \frac{1}{2}f(x)\ln 2\pi\sigma^2+f(x)\frac{(x-\mu)^2}{2\sigma^2}dx.
\end{aligned}$$

Jaetaan integraali kahdeksi, kuten normaalijakauman entropiaa laskettaessa. Saadaan siis epäyhtälö

$$\begin{aligned}
h(f) &\leq \int \frac{1}{2}f(x)\ln 2\pi\sigma^2+\int f(x)\frac{(x-\mu)^2}{2\sigma^2}dx \\
&\leq \frac{1}{2}\ln 2\pi\sigma^2\int f(x)+\frac{1}{2\sigma^2}\int f(x)(x-\mu)^2dx.
\end{aligned}$$

Kuten aikaisemmin todettiin, tiheysfunktion integraali koko määrittelyjoukkonsa yli on yksi, ja jälkimmäinen integraali on varianssi. Mielivaltaisen todennäköisyysjakauman f entropialle saadaan näin ollen epäyhtälö

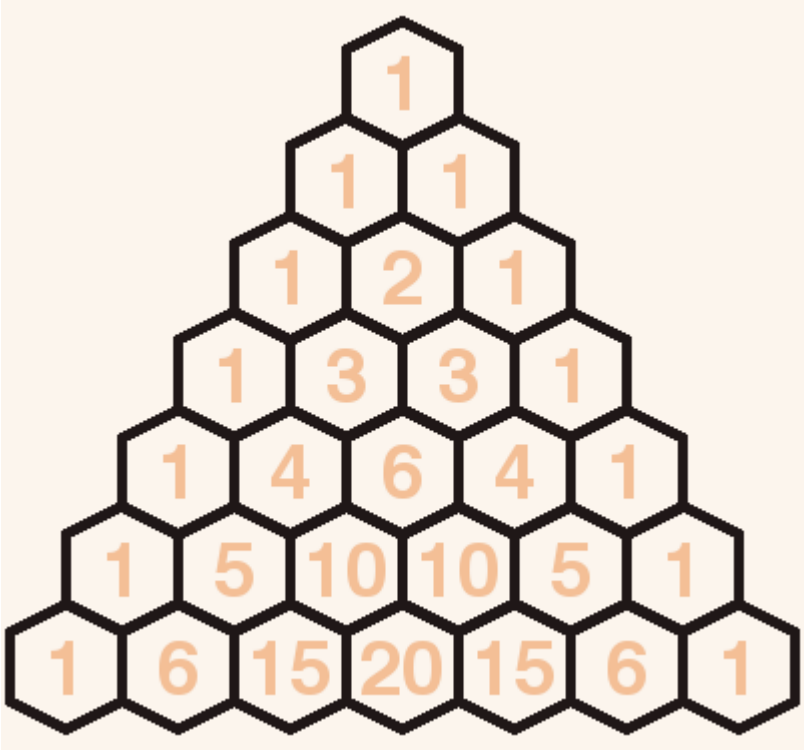
$$h(f) \leq \frac{1}{2}\ln 2\pi\sigma^2+\frac{1}{2\sigma^2}\sigma^2=\frac{1}{2}\ln 2\pi\sigma^2+\frac{1}{2}=\frac{1}{2}(\ln 2\pi\sigma^2+1)=h(\phi). \tag{2.15}$$

Täten jatkuvan todennäköisyysjakauman f entropia on aina pienempi tai yhtä suuri kuin normaalijakauman entropia, kun niillä on sama odotusarvo ja varianssi. □

2.3 Pascalin kolmio

Galtonin laudan sekä Pascalin kolmion välillä on myös havaittavissa yhteys. Havainnollistaaksemme tätä käytämme hyödyksi alla olevaa kuvaa. Yhteyden idea on saatu lähteestä [1].

Kuvitellaan, että jokainen Pascalin kolmion numeron kohdalla on yksi Galtonin laudan tappi. Ylimpään numeroon saapuessaan helmellä on ollut siihen vain yksi reitti: suoraan ylhäältäpäin. Suuresta epäjärjestyksestä johtuen voidaan approksimoida helmen valitsevan tapin kohdalla reitin joko vasemmalle tai oikealle. Ylimmästä tapista helmi voi siis tippua joko vasemman, tai oikean puoleiseen tappiin. Molempiin näihin on siis edelleen vain yksi reitti, ja numero näissä molemmissa onkin yksi. Kumpaan tahansa tappiin helmi päätyi, on sillä taas mahdollisuus valita joko vasen tai oikea puoli, jolle tippua. Nyt kuitenkin yksi kolmannen rivin tapeista omaa numeron kaksi. Tämä johtuu siitä, että kyseinen tappi on ensimmäinen, johon löytyy useampi eri reitti. Siihen päästään joko kaatumalla ensimmäisen rivin tapista vasempaan ja toisen rivin tapista oikeaan, tai ensimmäisestä oikeaan ja toisesta vasempaan. Seuraavia rivejä katsomalla voidaan huomata jokaisen Pascalin kolmion luvun kertovan suoraan sen, kuinka monta eri reittiä kyseisessä kohdassa sijaitsevaan tappiin on, kun helmi tiputetaan keskeltä ylhäältä. Esimerkiksi viidennen rivin toiseen



Kuva 2.2. *Pascalin kolmion ensimmäiset seitsemän riviä.*

tappiin on neljä reittiä: vasen, vasen, vasen, oikea; vasen, vasen, oikea, vasen; vasen, oikea, vasen, vasen; oikea, vasen, vasen, vasen.

Koska Galtonin laudan keskimmäisiin alalokeroihin on eniten reittejä, yksittäisen helmen on todennäköisempää valita niitä kohti vievä reitti, sillä jokainen reitti on yhtä todennäköinen. Siksi toistokokeen tuloksena niihin myös päätyy eniten helmiä.

3. KOLMIULOTTEINEN GALTONIN LAUTA

3.1 Idean esittely

Galtonin lautaa käsiteltäessä tarkastellaan kaksiulotteista koordinaatistoa, jolloin laudan läpi tippuvat helmet muodostavat normaalijakauman mukaisen käyrän. Nyt siirrytään tarkastelemaan tilannetta kolmiulotteisessa avaruudessa, jossa laudan sijaan tapit sijaitsevatkin kuution sisällä. Näin syntyneen kolmiulotteisen Galtonin laudan läpi tippuvat helmet voivat siis liikkua laudan yhden akselin sijaan nyt kahden akselin välillä. Kuution pohjaan tippuneet helmet myös muodostavat yksiulotteisen normaalijakauman sijasta kaksiulotteisen normaalijakaumatason. Tästä eteenpäin tässä työssä voidaan nimittää kolmiulotteista Galtonin lautaa myös Galtonin kuutiona.

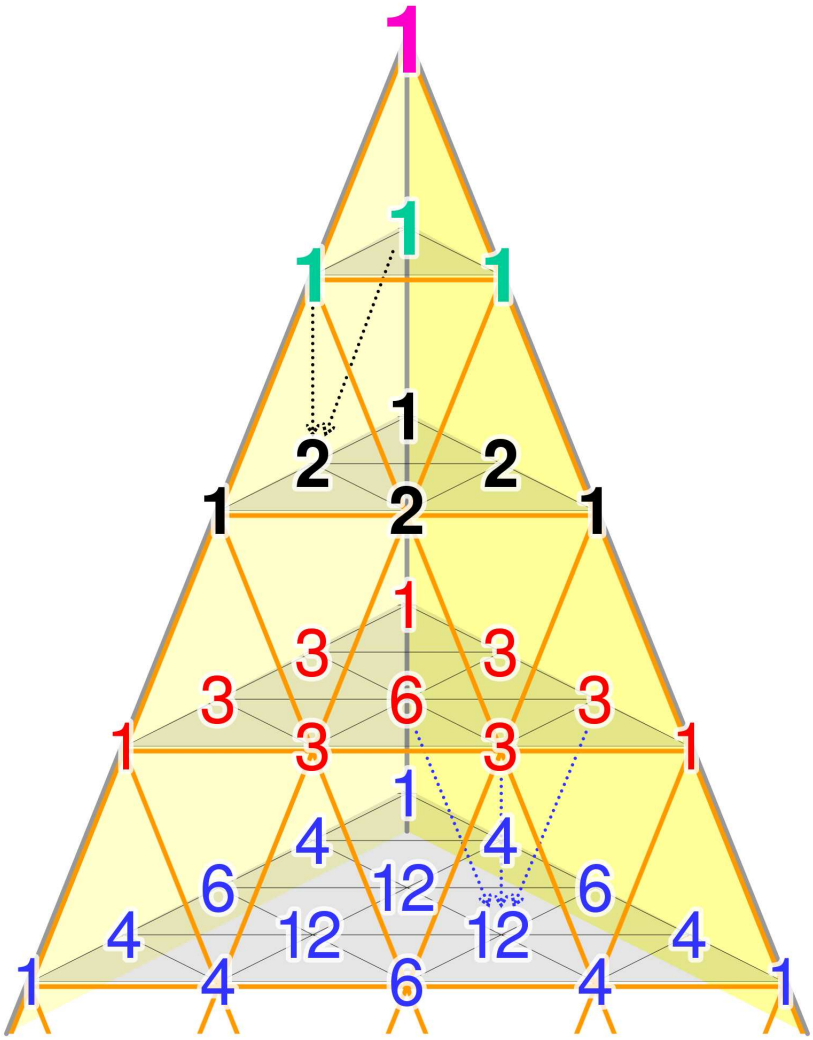
3.2 Pascalin pyramidi

Kuten Pascalin kolmio Galtonin laudasta, niin Pascalin pyramidi kertoo tärkeää informaatiota Galtonin kuutiosta. Pascalin kolmiosta voidaan nähdä binomilauseen summan termien kertoimet. Pascalin kolmiopohjainen pyramidi on kolmisivuinen, ja sitä voidaan käyttää hyödyksi kolmitermisen polynomin potenssin trinomikerrointen ratkaisemiseksi. Jos jokainen numero vastaa yhtä tappia, Pascalin kolmion osoittamat binomikertoimet ja pyramidin trinomikertoimet kertovat samalla sen, kuinka monta eri reittiä kyseiseen tappiin on olemassa.

Kolmiopohjainen pyramidi ei ole ainut mahdollisuus. Pascalin pyramidi voidaan muodostaa myös neliöpohjaisena. Tällöin sitä voitaisiin edellisten tapaan käyttää monitermisen, tässä tapauksessa nelitermisen, polynomin potenssin sarjan kerrointen selvittämiseksi. Toistokokeen simulaatiossa tullaan käyttämään neliöpohjaista pyramidia.

3.3 Ideaali sekä realistinen malli

Kuten Galtonin lauta, myös Galtonin kuutio voidaan pyrkiä rakentamaan reaalimaailmassa. Tässä kuitenkin huomataan, että suurempaan ulottuvuuteen siirryttäessä pistemäisten tappien asettaminen muuttuu vaikeammaksi. Kaksiulotteisessa laudassa tappien tarvitsee olla pistemäisiä vain yksiulotteisesti, sillä helmet kimpoilevat tapeista tippuessaan yhden akselin suuntaisesti. Täten tapit voivat olla pitkiä tikkuja Galtonin laudan pohjan sekä sen kannen välillä ja samalla säilyttää pistemäisyytensä helmiin ja niiden liikkumismahdollisuuteen nähden. Kun siirrytään tarkastelemaan neliöpohjaista pyramidia, helmet voivat kimpoilla tapeista kahden akselin suuntaisesti. Kuutioon ei voida fyysisesti luoda leijuvia pisteitä, joista helmet kimpoisivat alaspäin mennessään. Voidaan



Kuva 3.1. *Pascalin kolmiopohjaisen pyramidin ensimmäiset viisi kerrosta. [2]*

kuitenkin luoda erilaisia malleja, joissa tapit kiinnitetään johonkin kuution reunaan. Esimerkiksi jokainen piste voisi olla risteämäkohta kahdelle, kuution eri seinistä kohtisuoraan osoittavalle tikulle. Tällöin saataisiin siivilän tapainen ratkaisu, joka vastaisi teoreettisesti ideaaleja leijuvia pisteitä kohtalaisen hyvin. Simulaatiossa käytetään "leijuvia tappeja". Jos vastaavanlainen toistokoe tahdottaisiin suorittaa reaalimaailmassa, tulisi kuution sisälle rakentaa mahdollinen, mutta leijuvien pisteiden tapainen ratkaisu.

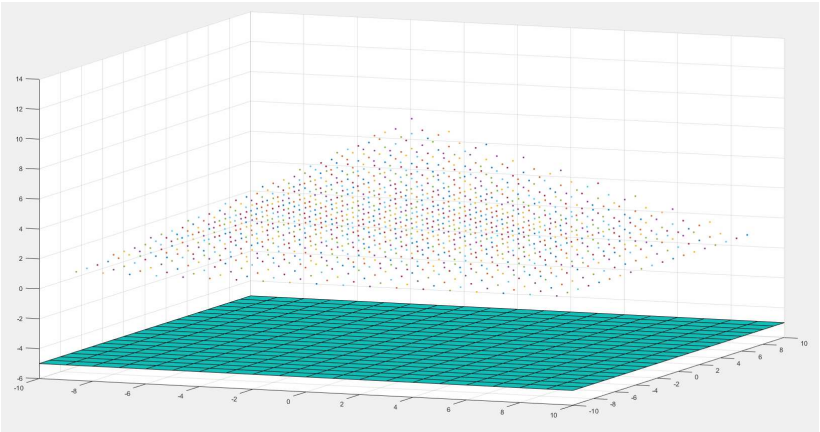
3.4 Simulaatio

Toistokokeen simulaatio toteutettiin MatLab-ohjelmalla. Käytetty ohjelma on ohjelmoitu tätä työtä varten, ja sen koodi löytyy työn lopusta liitteestä A. Koodissa luodaan tapeista neliöpohjaisen Pascalin pyramidin malli kolmiulotteiseen tilaan, jonka jälkeen helmiä voidaan simuloida putoamaan pyramidin huipulta. Pyramidi koostuu kymmenestä kerroksesta.

Simuloinnissa jokaisen tapin kohdalla helmelle annetaan satunnainen suunta, jonka mukaan helmi

tippuu seuraavaan kerrokseen. Helmelle arvotaan yksi kahdeksasta ilmansuunnasta, johon helmi tippuu. Tämä tapahtuu arpomalla ensin kokonaisluku $a \in [1, 8]$. Arvotulla luvulla kerrotaan luku $2\pi/8$, jolloin saadaan yksi kahdeksaan osaan jaetun yksikköympyrän kulmista. Lopuksi saadaan helmen reitti x-akselilla kokonaisuudessaan kaavalla $\cos a2\pi/8$ ja y-akselilla kaavalla $\sin a2\pi/8$. Cosinin ja sinin antama negatiivinen tai positiivinen arvo tarkoittavat omilla akseleillaan negatiiviseen tai positiiviseen suuntaan siirtymistä.

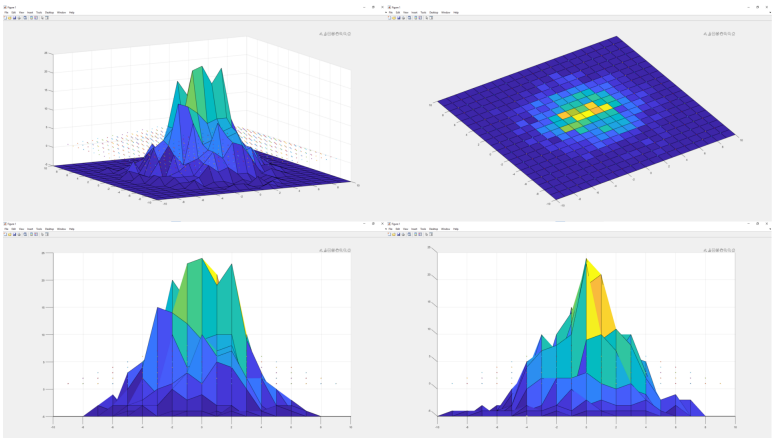
Satunnainen suunta oltaisiin voitu myös saada arpomalla erikseen sekä x- että y-akseleille kokonaisluku väliltä $[-1, 0]$. Kuitenkin, yksikköympyrää käyttämällä saavutetaan kaksi etua tähän verrattuna. Ensinnäkin, yksikköympyrän kulmaa käyttämällä saadaan yhdellä arvotulla luvulla helmen suunta sekä x- että y-akseleilla. Jos luvut arvottaisiin suoraan, täytyisi lukuja arpoa aina kaksi kappaletta erikseen. Tämän lisäksi yksikköympyrä ei voi antaa suuntaa $(0, 0)$, jonka suoraan kokonaislukujen arpominen voisi antaa. Tämä suunta on mahdoton, sillä helmi ei voi tippua tapin läpi. Helmen tulee aina tapin kohdatessaan kimmota siitä johonkin suuntaan, eikä se voi koskaan jatkaa liikettään suoraan alaspäin.



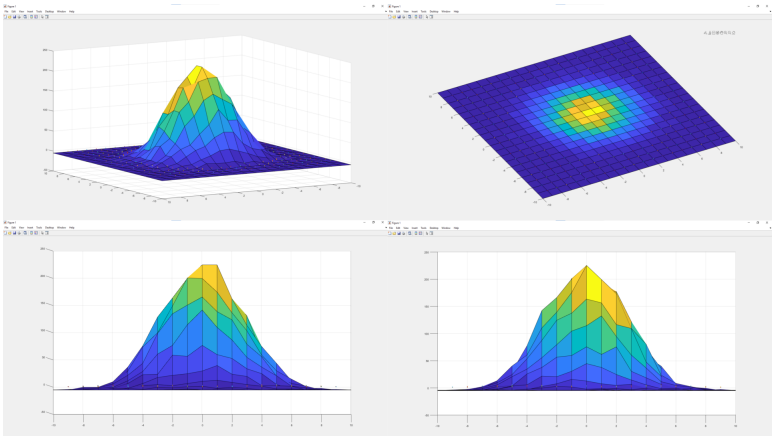
Kuva 3.2. *Pascalin neliöpohjainen pyramidi MatLab-ohjelmassa.*

3.5 Tulokset

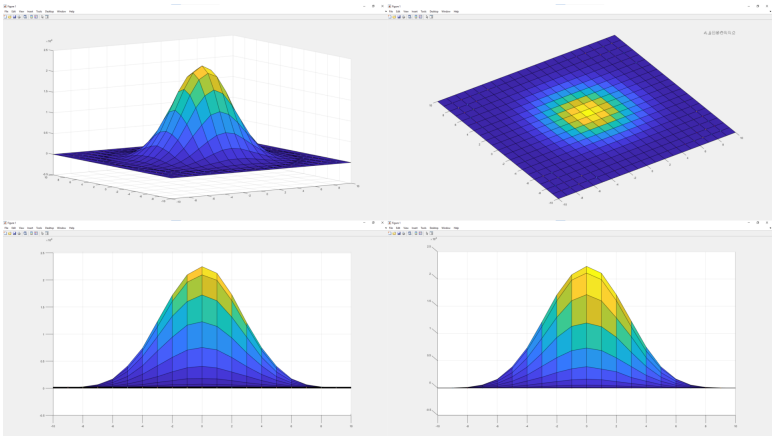
Toistokoe suoritettiin kuudella eri helmimäärällä: 10, 100, 1 000, 10 000, 100 000 sekä 1 000 000 helmellä. Koe toistettiin kymmenen kertaa jokaisella helmimäärällä. Laadittu ohjelma antaa helmien tiputusten jälkeen tiedon jokaisen alalokeroon tippuneen helmen x- ja y-koordinaatin odotusarvosta, sekä keskihajonnan erikseen kummankin koordinaattiakselin suhteen. Lisäksi ohjelma laskee, kuinka todennäköisesti helmi päätyi sekä x- että y-koordinaatillaan enintään yhden keskihajonnan päähän koordinaatin odotusarvosta. Jokaisen helmimäärän kymmenelle toistolle saadaan siis viisi eri arvoa. Taulukossa näille arvoille on laskettu varianssi, joka ilmaisee arvojen eroavaisuuden suuruutta. Lisäksi suurimmalle toistomäärälle, miljoonalle helmelle, ja niiden kymmenelle toistokerralle on jokaiselle laskettu todennäköisyys sille, että yksittäinen helmi tippuu enintään yhden keskihajonnan päähän odotusarvosta. Tätä voidaan verrata teoreettiseen todennäköisyyteen, jonka tulisi toteutua moniulotteisessa normaalijakaumassa.



Kuva 3.3. *Tuhannen toiston toistokokeen kuvaaja*



Kuva 3.4. *Kymmenen tuhannen toiston toistokokeen kuvaaja*



Kuva 3.5. *Miljoonan toiston toistokokeen kuvaaja*

3.6 Tulosten analysointia

Tuloksista huomataan toistokertojen välisen varianssin pienentyvän otoskokoa kasvattamalla. Havainto on ennakko-odotusten mukainen, sillä toistokokeiden on yleisesti ajateltu antavan tarkempia arvoja suuremmilla toistomäärillä. Simulaation jakauman odotusarvoa ja -hajontaa analysoitaessa voidaan siis tarkastella miljoonan toiston tuloksia, sillä se on suoritetuista toistokokeista suurin.

Keskiarvo_x	Toistojen määrä	10	100	1 000	10 000	100 000	1 000 000
Toistokerta							
1		1,2000	-0,2300	-0,0050	-0,0158	-0,0024	0,0016
2		-0,6000	-0,0800	-0,0020	0,0314	0,0009	-0,0002
3		0,6000	0,1100	0,0400	-0,0045	0,0008	-0,0034
4		-0,7000	-0,1700	0,1440	0,0162	-0,0072	0,0018
5		0,6000	-0,0200	0,0540	0,0024	-0,0012	-0,0018
6		0,7000	-0,0900	-0,2070	0,0066	-0,0016	-0,0029
7		1,2000	0,4000	0,0710	-0,0205	0,0042	0,0014
8		-0,5000	0,0700	0,1080	0,0488	-0,0130	0,0012
9		-0,5000	0,0300	-0,0200	0,0071	-0,0059	0,0052
10		-0,5000	-0,1400	0,0360	0,0129	-0,0042	-0,0017
Varianssi		0,5465	0,029276	0,00817349	0,00038704	2,14524E-05	6,2236E-06
Keskiarvo_y	Toistojen määrä	10	100	1 000	10 000	100 000	1 000 000
Toistokerta							
1		0,4000	0,1400	-0,0210	0,0032	-0,0052	-0,0006
2		-0,3000	-0,0500	0,0340	-0,0134	-0,0096	-0,0016
3		0,8000	-0,3300	0,0700	0,0484	0,0006	-0,0027
4		0,8000	0,4300	0,1050	-0,0139	-0,0006	0,0011
5		-1,1000	-0,1100	-0,0360	-0,0202	0,0035	0,0032
6		-0,3000	0,1100	-0,0700	0,0013	0,0047	0,0026
7		0,3000	0,0300	0,0420	-0,0322	-0,0067	-0,0002
8		0,4000	-0,2200	-0,0290	-0,0229	0,0089	-0,0002
9		0,5000	-0,0800	-0,0870	-0,0304	-0,0073	0,0036
10		0,0000	0,4500	0,0030	-0,0325	0,0033	-0,0038
Varianssi		0,3105	0,058461	0,00338889	0,000540908	3,35484E-05	5,5704E-06
Keskihajonta_x	Toistojen määrä	10	100	1 000	10 000	100 000	1 000 000
Toistokerta							
1		2,6758	2,6827	2,6648	2,7496	2,7317	2,7389
2		1,3565	2,7665	2,7928	2,7206	2,7418	2,7386
3		1,6852	2,8139	2,6575	2,7344	2,7271	2,7372
4		2,3685	2,4822	2,7881	2,7309	2,7416	2,7389
5		3,5553	2,3366	2,8148	2,7451	2,7363	2,7375
6		2,5318	2,7243	2,6725	2,7331	2,7410	2,7365
7		2,3580	2,8844	2,7240	2,7603	2,7412	2,7405
8		3,3838	3,0175	2,7753	2,7225	2,7428	2,7334
9		2,4597	2,7256	2,6671	2,7460	2,7426	2,7353
10		1,9105	2,7929	2,7674	2,7160	2,7415	2,7417
Varianssi		0,423365109	0,033674226	0,003470824	0,000181442	2,58704E-05	5,3285E-06

odotusarvo sekä x- että y-akselilla lähestyy nollaa. Tämä oli odotettavissa, sillä sekä helmien tiputuspiste että Pascalin neliöpohjaisen pyramidin huippu sijaitsevat simulaatiossa näiden akselien origossa. Täten mitä lähempänä origoa pohjalla sijaitseva lokero on, sitä enemmän siihen on yksittäisiä reittejä ja sitä todennäköisemmin helmi päätyy kyseiseen lokeroon.

Myös keskihajonta lähestyy samaa arvoa sekä x- että y-akseleilla. Keskihajonta kuvaa saatujen arvojen keskimääräistä poikkeamaa odotusarvosta. Simulaatiossa keskihajonta lähestyy kummallakin akselilla samaa lukua, joka kahden desimaalin tarkkuudella pyöristettynä on 2.74. Teoreettinen todennäköisyys sille, että yksi normaalijakaumaa noudattavan toistokokeen toisto osuu enintään yhden keskihajonnan päähän odotusarvosta, on 39.35% [9]. Tulosten taulukosta nähdään, että miljoonan toiston toistokokeiden odotusarvo tälle on 40.14%. Kuten nähdään, simulaatiolla saatu luku eroaa melkein yhden kokonaisen prosenttiyksikön verran teoreettisesta arvosta. Kuitenkin, taulu-

Keskihajonta_y	Toistojen määrä	10	100	1 000	10 000	100 000	1 000 000
Toistokerta							
1		2,7641	2,8285	2,6938	2,7318	2,7334	2,7382
2		2,5318	2,5822	2,6572	2,7241	2,7361	2,7403
3		1,6000	2,7643	2,7682	2,7794	2,7301	2,7374
4		2,5612	2,7577	2,7309	2,7334	2,7365	2,7363
5		3,3302	2,5452	2,6531	2,7547	2,7454	2,7382
6		3,3181	2,3954	2,8201	2,7046	2,7283	2,7390
7		3,4655	2,5197	2,6639	2,7509	2,7462	2,7401
8		1,8547	3,0613	2,7317	2,7812	2,7334	2,7369
9		3,6946	3,2823	2,7805	2,7320	2,7419	2,7389
10		2,1909	2,8262	2,7461	2,7389	2,7463	2,7380
Varianssi		0,458012113	0,06409682	0,002898649	0,000517058	4,10204E-05	1,4961E-06
Todennäköisyys	Toistojen määrä	10	100	1 000	10 000	100 000	1 000 000
Toistokerta							
1		0,4000	0,4000	0,4030	0,4012	0,4030	0,4013
2		0,4000	0,4100	0,4160	0,4083	0,4011	0,4014
3		0,5000	0,3800	0,4020	0,3955	0,4058	0,4015
4		0,4000	0,4990	0,3930	0,4037	0,4009	0,4020
5		0,2000	0,4600	0,4180	0,3981	0,4023	0,4013
6		0,4000	0,4900	0,3930	0,4088	0,4020	0,4011
7		0,6000	0,4800	0,4080	0,4013	0,4000	0,4007
8		0,4000	0,4400	0,4070	0,3954	0,4013	0,4023
9		0,3000	0,5000	0,3970	0,3972	0,3994	0,4016
10		0,5000	0,4100	0,3780	0,4038	0,4003	0,4003
Varianssi		0,0109	0,00181049	0,00012745	2,12761E-05	3,0169E-06	3,005E-07
						Keskiarvo	0,4014

kosta nähdään, että saatu luku ei ole simulaation kannalta yllättävä. Se käyttäytyy samalla tavalla kuin odotusarvo sekä keskihajonta siinä, että yksittäisten toistokokeiden välinen vaihtelu pienenee ja saadun arvon tarkkuus suurenee, mitä suuremmalla otoskoolla simulaatio suoritetaan.

Saadun arvon eroavuutta teoreettisesti oikeasta luvusta voidaan selittää sillä, että simulaation helmet päätyvät rajattuihin lokeroihin. Siispä saatu jakauma helmien loppusijoituksista on diskreetti, siinä missä teoreettinen arvo on annettu jatkuvalle normaalijakaumalle. Taulukosta nähdään, että odotusarvo lähestyy koordinaattia $(0, 0)$ ja keskihajonta arvoa 2.73. Diskreetissä jakaumassa helmen laskeutuessa jollakin akselillaan jatkuvalle välille $[2, 3]$ helmen tulee päätyä joko kokonaislukuarvolliseen koordinaattiin 2 tai 3. Koska normaalijakaumassa toistot sijoittuvat lähestyen otoksen odotusarvoa, simulaation tyylisiin lokeroihin jakaessa ikään kuin kahden lokeron väliin päätyvät tulokset sijoittuvat todennäköisemmin lähempänä odotusarvoa olevaan lokeroon. Simulaatiosta saatua suurempaa todennäköisyyttä osua yhden keskihajonnan päähän odotusarvosta voitaneen selittää tällä ilmiöllä.

4. YHTEENVETO

Galtonin kuutioon tutustumiseksi käytiin ensin läpi perinteistä, kaksiulotteista Galtonin lautaa. Galtonin laudan käsittely rajattiin puhtaasti matemaattiseksi Claude Shannonin informaatioteorian avulla. Informaatioteoriaa soveltaen todettiin Galtonin laudan helmien muodostavan täysin ennalta tietämättömän jakauman, jonka jälkeen todistettiin normaalijakauman entropian olevan kaikista jatkuvista todennäköisyysjakaumista suurin. Pascalin kolmion todettiin myös olevan merkittävä työkalu osaltaan selittämään todennäköisyysjakauman syntyä ja reittien lukumäärien vaikutusta sen syntyyn.

Kaksiulotteisesta Galtonin laudasta siirryttiin kolmiulotteiseen Galtonin lautaan, jota tässä työssä nimitettiin lyhyesti Galtonin kuutiona. Galtonin kuution ideaa tuettiin kaksiulotteisen laudan aspekteilla, jonka jälkeen todettiin ideaalin mallin ja sen simuloinnin olevan tämän työn intressejä mukailevampi tapa Galtonin kuution mallintamiseksi. Simulaatiota ja tätä työtä varten koodattiin liitteestä A löytyvä ohjelma, jonka avulla simuloitiin Galtonin kuutiolla toteutettuja toistokokeita. Toistokokeita suoritettiin useita erilaisilla otosko'oilla aina kymmenestä tuhannesta miljoonaan saakka, ja saadut tulokset odotusarvolle ja keskihajonnalle erikseen x- ja y-akselien suhteen taulukoitiin ylös.

Lopuksi taulukkoon kerättyä dataa sekä simulaation piirtämiä graafeja katsomalla havaittiin Galtonin kuution niin ikään noudattavan kaksiulotteista normaalijakaumaa. Tätä tukivat saadut odotusarvon ja keskihajonnan arvot, mutta myös se, että tiputettujen helmien todennäköisyydeksi tippua enintään yhden keskihajonnan päähän odotusarvosta saatiin teoreettista todennäköisyyttä lähellä oleva arvo. Heittoa teoreettisesta arvosta selitettiin simulaation lokeroiden diskreetillä luonteella ja sen aiheuttamalla jakauman vääristymällä.

LÄHTEET

[1] D!NG. 2018. *The Galton Board*. YouTube-video. Julkaistu 14.5.2018. Viitattu 20.12.2023. https://www.youtube.com/watch?v=UCmPmkHqHXk.

[2] Cmglee, CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons. Saatavilla: https://commons.wikimedia.org/wiki/File:Pascal_pyramid_3d.svg.

[3] M. Bulmer. *Francis Galton: Pioneer of Heredity and Biometry*. Baltimore: Johns Hopkins University Press, 2003.

[4] C. Marsh. Introduction to continuous entropy. *Department of Computer Science, Princeton University* 1034 (2013).

[5] J. Pitman. *Probability*. Springer, 1999.

[6] R. Rojas. Why the normal distribution. *Freis Universitat Berlin lecture notes* (2010).

[7] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal* (1984).

[8] T. O. To ja K. W. Yip. A generalized Jensen's inequality. (1975).

[9] B. Wang, W. Shi ja Z. Miao. Confidence analysis of standard deviational ellipse and its extension into higher dimensional Euclidean space. *PloS one* 10.3 (2015), e0118537.

[10] E. W. Weisstein. *Expectation Value. From MathWorld–A Wolfram Web Resource*. Viitattu 20.12.2023. URL: `https://mathworld.wolfram.com/ExpectationValue.html`.

[11] E. W. Weisstein. *Jensen's Inequality. From MathWorld–A Wolfram Web Resource*. Viitattu 20.12.2023. URL: `https://mathworld.wolfram.com/JensensInequality.html`.

[12] E. W. Weisstein. *Variance. From MathWorld–A Wolfram Web Resource*. Viitattu 20.12.2023. URL: `https://mathworld.wolfram.com/Variance.html`.

LIITE A: GALTONIN KUUTION SIMULAATIO - MATLAB-KOODI

```
% @Author: Jimi Asikainen
%
% This code has been written in its entirety by the author. No pieces of
% it have been taken from elsewhere.
% At the start, there are multiple variables and information on how they
% can be used. The simulation can be greatly modified with them. The
% user doesn't have to understand the rest of the code to simulate
% Galton's cube successfully. Nevertheless, comments have been added
% throughout the rest of the software for anyone interested in the
% working principles.
%
%
%
% z_scale: the height
z_scale = -5:14;
z_scale_limits = z_scale(size(z_scale));
bottom_coordinate = z_scale_limits(1);
top_coordinate = z_scale_limits(2);
% Galton_cube_height: The number of layers of pegs that are there for the
% beans to bounce off. The recommended value is 10. Use values between 1
% and 10.
Galton_cube_height = 10;
% repeats: how many balls are dropped through the Galton's cube. For
% example, you can try these values: 10, 100, 1 000, 10 000, 100 000,
% 1 000 000.
repeats = 100000
% animate: true, if you want to animate beans dropping. false, if you
% want only to plot the result. Setting this false makes the process
% extremely quicker.
animate = false;
% animate_balls: if animate is true, setting this true animates using
% red balls. false uses lines instead of balls. Setting this false makes
```

```
% the process greatly faster.
animate_balls = true;
% animated_balls_amount: how many red balls are plotted between two
% individual locations. The recommended value is 4.
animated_balls_amount = 4;

% coordinate map for saving dropped balls' end locations. y dimension
y_map = [-10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10, ...
    -10 -10 -10 -10 -10 -10;
    -9 -9 -9 -9 -9 -9 -9 -9 -9 -9 -9 -9 -9 -9 -9 -9 -9 -9 -9 -9 -9;
    -8 -8 -8 -8 -8 -8 -8 -8 -8 -8 -8 -8 -8 -8 -8 -8 -8 -8 -8 -8 -8;
    -7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7 -7;
    -6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6 -6;
    -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5;
    -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4;
    -3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3 -3;
    -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2;
    -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1;
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1;
    2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2;
    3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3;
    4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4;
    5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5;
    6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6;
    7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7;
    8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8;
    9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9;
    10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10;];
% x dimension
x_map = y_map';

% starting plot area
box = plot3(1,1,z_scale);

% starting camera
v = [0,15];
view(v)

hold on
grid on
```

```
z = Galton_cube_height;
% Set the cube's pegs
for z_index = 1:z
    for x_index = -(z-z_index):(z-z_index)
        for y_index = -(z-z_index):(z-z_index)
            plot3(x_index, y_index, z_index, '.')
        end
    end
end

% Set the bottom surf plot for normal distribution
normal_surface = zeros(21) + bottom_coordinate;
surf_plot = surf(x_map, y_map, normal_surface);

% starting level of dropped balls
ball_starting_height = top_coordinate;

% Count the normal distribution, either with ball animation, line
% animation or no animation at all.
% Note: There are three almost identical functions used here, which
% repeats plenty of code and therefore is against the standard rules of
% coding. However, this way we dodge having multiple different
% if-statements inside the code, which would have an effect on the speed
% of the repeated trial. Instead, here we have only two "if"-statements,
% followed by calling the fitting function. That is why the programmer
% has decided this to be the best way to approach the problem, even
% though it requires more lines of code.
if (animate == false)
    normal_surface = ND_not_animated(repeats, ball_starting_height, ...
    bottom_coordinate, Galton_cube_height, normal_surface);
else
    if (animate_balls == true)
        normal_surface = ND_animated_balls(repeats, ...
        ball_starting_height, bottom_coordinate, Galton_cube_height, ...
        normal_surface, x_map, y_map, animated_balls_amount);
    else
        normal_surface = ND_animated_lines(repeats, ...
        ball_starting_height, bottom_coordinate, Galton_cube_height, ...
        normal_surface, x_map, y_map);
    end
end
```

```
% Plot the result distribution
surf_plot = surf(x_map, y_map, normal_surface);

% Fix the values for the calculations.
normal_surface = normal_surface + 5;

mean_coordinate = get_mean_coordinate(normal_surface, repeats)

variance_coordinate = get_variance_coordinate(normal_surface, ...
    repeats, mean_coordinate)

% Standard deviation is the square root of variance.
standard_deviation = sqrt(variance_coordinate)

probability = get_standard_deviation_probability(normal_surface, ...
    standard_deviation, mean_coordinate, repeats)
```

```
function target_position = set_target_position(current_position, x, y)
    arguments
        current_position
        x {mustBeNumeric}
        y {mustBeNumeric}
    end
    % Add x and y
    target_position(1) = current_position(1) + x;
    target_position(2) = current_position(2) + y;
    % Lower the bean location by one.
    target_position(3) = current_position(3) - 1;
end

function [x, y] = random_walk()
    % Eight possible paths to fall to.
    rand_int = (randi(8) - 1);
    % Use the unit circle to get random values for x and y. Possible
    % values are -1, 0 and 1.
    rand_deg = rand_int*(pi/4);
    x = cos(rand_deg);
    y = sin(rand_deg);

    % Fix the remainder mistake caused by having a calculation done with
    % float variables, where the result should be 0, but is something
```

```
    % between [ -1*10^(-5), 1*10^(-5) ] instead.
    if abs(x) < 0.001
        x = 0;
    end
    % If x is not zero, it gets the value of -1 or 1.
    if x ~= 0
        % If cos (x) is positive, x = 1.
        if x > 0
            x = 1;
        % Otherwise it is negative and x = -1.
        else
            x = -1;
        end
    end

    % Fix the remainder mistake (when y is actually zero).
    if abs(y) < 0.001
        y = 0;
    end
    if y ~= 0
        % If sin (y) is positive, y = 1.
        if y > 0
            y = 1;
        % Otherwise it is negative and y = -1.
        else
            y = -1;
        end
    end
end

function normal_surface = ND_not_animated(repeats, ...
    ball_starting_height, bottom_coordinate, Galton_cube_height, ...
    normal_surface)
    % Repeat drops for repeats.
    for i = 1:repeats
        % Set current position to starting position (middle and starting
        % height).
        current_position = [0, 0, ball_starting_height];
        % Current position(3) equals height
        while current_position(3) > bottom_coordinate
            % If the height is inside the area of pebbles, get the new
            % coordinate after hitting a pebble.
```

```
            if current_position(3) > 0 && current_position(3) <= ...
                    Galton_cube_height
                [x, y] = random_walk();
            % Otherwise move the bean downwards
            else
                x = 0;
                y = 0;
            end
            target_position = set_target_position(current_position, ...
                x, y);
            % Update the position of the bean
            current_position = target_position;
        end
        % Values of the plot are between [-10, 10] and values in the
        % coordinate maps are between [1, 21]. We fix this by adding
        % 1 - (-10) = 11.
        x = 11 + current_position(2);
        y = 11 + current_position(1);
        % Add a bean to it's ending compartment
        normal_surface(x, y) = normal_surface(x, y) + 1;
    end
end

function normal_surface = ND_animated_balls(repeats, ...
    ball_starting_height, bottom_coordinate, Galton_cube_height, ...
    normal_surface, x_map, y_map, animated_balls_amount)
    % Repeat drops for repeats.
    for i = 1:repeats
        % Set current position to starting position (middle and starting
        % height).
        current_position = [0, 0, ball_starting_height];
        % Set spin for the rotation of Galton's cube
        spin = 1;
        % Set tag_counter for deleting the path plots of an individual
        % bean.
        tag_counter = 1;
        % Current position(3) equals height
        while current_position(3) > bottom_coordinate
            % If the height is inside the area of pebbles, get the new
            % coordinate after hitting a pebble.
            if current_position(3) > 0 && current_position(3) <= ...
            Galton_cube_height
```

```
            [x, y] = random_walk();
        % Otherwise move the bean downwards
        else
            x = 0;
            y = 0;
        end
        target_position = set_target_position(current_position, ...
            x, y);

        % The path to the next pebble
        target_vector = (target_position - current_position);
        % Plot balls for the amount of animated_balls_amount.
        for i = 1:animated_balls_amount
            ball_coordinate = current_position + ...
                i*(1/animated_balls_amount)*target_vector;
            ball_marker = plot3(ball_coordinate(1), ...
                ball_coordinate(2), ball_coordinate(3), ...
                'ro','MarkerFaceColor','r');
            % Tag the plot to delete it later.
            tag_str = int2str(tag_counter);
            set(ball_marker,'tag',tag_str)
            tag_counter = tag_counter + 1;
            % Rotate the view around the cube.
            [caz, cel] = view();
            v = [caz, cel] + [spin, 0];
            view(v)

            % Time waited between ball plots
            pause(0.001)
        end
        % Update the position of the bean
        current_position = target_position;
    end
    % Values of the plot are between [-10, 10] and values in the
    % coordinate maps are between [1, 21]. We fix this by adding
    % 1 - (-10) = 11.
    x = 11 + current_position(2);
    y = 11 + current_position(1);
    % Add a bean to it's ending compartment
    normal_surface(x, y) = normal_surface(x, y) + 1;
    % Update the normal surface plot after each bean.
    surf_plot = surf(x_map, y_map, normal_surface);
```

```
        % Delete the plotted path the bean fell.
        remove_path(tag_counter)
    end
end

function normal_surface = ND_animated_lines(repeats, ...
    ball_starting_height, bottom_coordinate, Galton_cube_height, ...
    normal_surface, x_map, y_map)
    % Repeat drops for repeats.
    for i = 1:repeats
        % Set current position to starting position (middle and starting
        % height).
        current_position = [0, 0, ball_starting_height];
        % Set spin for the rotation of Galton's cube
        spin = 1;
        % Set tag_counter for deleting the path plots of an individual
        % bean.
        tag_counter = 1;
        % Current position(3) equals height
        while current_position(3) > bottom_coordinate
            % If the height is inside the area of pebbles, get the new
            % coordinate after hitting a pebble.
            if current_position(3) > 0 && current_position(3) <= ...
                Galton_cube_height
                [x, y] = random_walk();
            % Otherwise move the bean downwards
            else
                x = 0;
                y = 0;
            end
            target_position = set_target_position(current_position, ...
                x, y);

            % Plot the line between the current pebble and the next one.
            line_plot = plot3([current_position(1), ...
                target_position(1)], [current_position(2), ...
                target_position(2)], [current_position(3), ...
                target_position(3)], 'Color','r');
            % Tag the plot to delete it later.
            tag_str = int2str(tag_counter);
            set(line_plot,'tag',tag_str)
            tag_counter = tag_counter + 1;
```

```
            % Rotate the view around the cube.
            [caz, cel] = view();
            v = [caz, cel] + [spin, 0];
            view(v)
            % Update the position of the bean
            current_position = target_position;
            % Time waited between line plots.
            pause(0.001)
        end
        % Values of the plot are between [-10, 10] and values in the
        % coordinate maps are between [1, 21]. We fix this by adding
        % 1 - (-10) = 11.
        x = 11 + current_position(2);
        y = 11 + current_position(1);
        % Add a bean to it's ending compartment
        normal_surface(x, y) = normal_surface(x, y) + 1;
        % Update the normal surface plot after each bean.
        surf_plot = surf(x_map, y_map, normal_surface);
        % Delete the plotted path the bean fell.
        remove_path(tag_counter)
    end
end

function remove_path(tag_counter)
    % Plots are marked with tags starting from number 1. Amount of tagged
    % plots is saved as tag_counter.
    for i = 1:tag_counter
        % Delete plot by tag number.
        tag_str = int2str(i);
        plot_to_remove = findobj('tag',tag_str);
        delete(plot_to_remove)
    end
end

function mean_coordinate = get_mean_coordinate(normal_surface, repeats)
    % Sum of all beans on each x coordinate (all columns summed)
    Sx = sum(normal_surface, 1);
    % Collect all x values from dropped beans.
    x_values = zeros(repeats, 1);
    % Index to mark values on x_values.
    counter = 1;
    for coordinate = -10:10
```

```
        column = Sx(coordinate+11);
        % Add x coordinate to x_values for each bean dropped on each one.
        for index = 1:column
            x_values(counter) = coordinate;
            counter = counter + 1;
        end
    end
    % Calculate the mean for the x coordinate.
    mean_coordinate_x = mean(x_values);

    % Sum of all beans on each y coordinate (all rows summed)
    Sy = sum(normal_surface, 2);
    % Collect all y values from dropped beans.
    y_values = zeros(repeats, 1);
    % Index to mark values on y_values.
    counter = 1;
    for coordinate = -10:10
        row = Sy(coordinate+11);
        % Add y coordinate to y_values for each bean dropped on each one.
        for index = 1:row
            y_values(counter) = coordinate;
            counter = counter + 1;
        end
    end
    % Calculate the mean for y coordinate.
    mean_coordinate_y = mean(y_values);

    % Return mean_coordinate containing mean values for both axes.
    mean_coordinate = [mean_coordinate_x, mean_coordinate_y];
end

function variance_coordinate = get_variance_coordinate(normal_surface, ...
    repeats, mean_coordinate)
    % Collect mean values of both axes from mean_coordinate.
    mean_x = mean_coordinate(1);
    mean_y = mean_coordinate(2);
    % Sum of all beans on each x coordinate (all columns summed)
    Sx = sum(normal_surface, 1);
    % Collect the value of (x-mean_x)^2 for each bean.
    variance_portion_x = zeros(repeats, 1);
    % Index to mark values on variance_portion_x.
    counter = 1;
```

```
    for coordinate = -10:10
        column = Sx(coordinate+11);
        for index = 1:column
            variance_portion_x(counter) = (coordinate - mean_x)^2;
            counter = counter + 1;
        end
    end
    % Calculate the x coordinate variance for the data collected.
    variance_coordinate_x = sum(variance_portion_x)/repeats;

    % Sum of all beans on each y coordinate (all rows summed)
    Sy = sum(normal_surface, 2);
    % Collect the value of (y-mean_y)^2 for each bean.
    variance_portion_y = zeros(repeats, 1);
    % Index to mark values on variance_portion_y.
    counter = 1;
    for coordinate = -10:10
        row = Sy(coordinate+11);
        for index = 1:row
            variance_portion_y(counter) = (coordinate - mean_y)^2;
            counter = counter + 1;
        end
    end
    % Calculate the y coordinate variance for the data collected.
    variance_coordinate_y = sum(variance_portion_y)/repeats;

    % Return both variances.
    variance_coordinate = [variance_coordinate_x, variance_coordinate_y];
end

function probability = get_standard_deviation_probability( ...
    normal_surface, standard_deviation, mean_coordinate, repeats)
    % Count how many beans found between Mu +- Sigma (mean +- standard
    % deviation)
    counter = 0;
    % Go through every compartment on the bottom
    for coordinate_x = -10:10
        for coordinate_y = -10:10
            % amount of how many beans in a compartment (x,y)
            coordinate_xy_amount = normal_surface(coordinate_x + 11, ...
                coordinate_y + 11);
            % if amount is zero (i = 1:0), this is skipped. If not, this
```

```
            % compartment is checked if it is between Mu +- Sigma
            for i = 1:coordinate_xy_amount
                if ( (coordinate_x < mean_coordinate(1) + ...
                        standard_deviation(1)) && (coordinate_x > ...
                        mean_coordinate(1) - standard_deviation(1)) && ...
                     (coordinate_y < mean_coordinate(2) + ...
                     standard_deviation(2)) && (coordinate_y > ...
                     mean_coordinate(2) - standard_deviation(2)) )
                    counter = counter + 1;
                end
            end
        end
    end
    % Lastly, count the probability that a random bean has landed at a
    % coordinate inside Mu +- Sigma
    probability = counter / repeats;
end
```