SAGE-OHJELMISTO LUKION MATEMATIIKAN OPETUKSESSA

Lauri Ruotsalainen

Pro gradu -tutkielma
Marraskuu 2011

MATEMATIIKAN LAITOS
TURUN YLIOPISTO

TURUN YLIOPISTO
Matematiikan laitos

RUOTSALAINEN, LAURI: Sage-ohjelmisto lukion matematiikan opetuksessa
Pro gradu -tutkielma, 132 s., 26 liites.
Matematiikka
Marraskuu 2011

Sage on verkkoselaimessa toimiva symbolisesti laskeva ohjelmisto, joka rakentuu useista avoimen lähdekoodin periaatetta noudattavista moduuleista. Komentokielenä se käyttää Python-ohjelmointikieltä. Sage soveltuu sekä opetus- että tutkimuskäyttöön. Opetuskäytössä sen tärkeimpiä ominaisuuksia ovat ilmaisuus, helppokäyttöisyys, vuorovaikutteiset ja graafiset toiminnot, dokumenttien muotoilutyökalut sekä verkkoympäristön edut, kuten käyttöjärjestelmäriippumattomuus ja ryhmätyömahdollisuudet.

Tutkielmassa esitellään aihealueittain, miten Sagea voidaan hyödyntää lukion matematiikan opetuksessa kursseilla, joilla käsitellään funktioita ja yhtälöitä, geometriaa ja trigonometriaa, differentiaali- ja integraalilaskentaa, todennäköisyyslaskentaa sekä numeerisia menetelmiä. Aihepiireihin liittyvien metodien toimintaa havainnollistetaan esimerkkien ja opetusta tukevien interaktiivisten sovellusten avulla. Lisäksi tutkielma sisältää 21 Sagen käyttöön perustuvaa tehtävää ratkaisuineen.

Suurin osa tutkielmaa varten ohjelmoiduista sovelluksista on julkaistu osana Sagea versiosta 4.7.1 alkaen. Tutkielma perustuu Sage-ohjelmiston versioon 4.7.

Asiasanat: lukio, matematiikka, Python, Sage, tietokoneavusteinen opetus

Sisältö

Esipuhe

Matemaattiset ohjelmistot ja verkkosovellukset ovat yleistymässä matematiikan perus- ja toisen asteen kouluopetuksessa. Parhaimmillaan matemaattisen aineiston käsittelyyn erikoistuneet ohjelmistot havainnollistavat opittavaa asiaa, antavat uusia näkökulmia matemaattisiin sisältöihin sekä innostavat opiskelijoita tutkimaan matemaattisia ilmiöitä ja ongelmia omatoimisesti.

Tässä tutkielmassa perehdytään *Sage*-nimiseen matemaattiseen ohjelmistoon ja sen käyttöön lukio-opetuksessa. Sage on verkkoselaimessa toimiva symbolisesti laskeva ohjelma, joka rakentuu useista avoimen lähdekoodin periaatetta noudattavista moduuleista. Se soveltuu niin lukiossa käsiteltävien matemaattisten aiheiden kuin myöhemmin korkeakouluissa opetettavien osa-alueiden tarkasteluun. Opetuskäytössä sen tärkeimpiä ominaisuuksia ovat ilmaisuus, helppokäyttöisyys, interaktiiviset ja graafiset toiminnot, dokumenttien muotoilutyökalut sekä verkkoympäristön edut, kuten käyttöjärjestelmäriippumattomuus ja ryhmätyömahdollisuudet.

Kiinnostuin Sagesta matematiikan opetuksen työvälineenä etsiessäni avointa ja maksutonta vaihtoehtoa kaupallisille ohjelmistoille. Sagen kehitysideologia perustuu ajatukselle yhdistää useiden avoimen lähdekoodin ohjelmien toimivat ratkaisut yhdeksi, kaikki matematiikan aihepiirit kattavaksi kokonaisuudeksi. Selkeä, helposti omaksuttava syntaksi ja hyvät käyttömahdollisuudet verkossa tekevät Sagesta monilta osin ihanteellisen opetuskäyttöön.

Tutkielmani kirjoittamista varten olen perehtynyt Sagen käyttöön sekä ohjelmoinut erilaisia vuorovaikutteisia ohjelmia, joita voidaan käyttää opetuksen tukena. Tutkielman tavoitteena on antaa yleiskuva Sagen käyttömahdollisuuksista ja sen ominaisuuksien monipuolisesta hyödyntämisestä lukion matematiikan opetuksessa.

Tutkielman ensimmäinen luku, johdanto, käsittelee yleisesti matemaattisia ohjelmistoja ja niiden käyttöä opetuksessa. Didaktisesti kiinnostavia kysymyksiä ovat, mitä lisäarvoa symbolisesti laskevat ohjelmistot tuovat matematiikan opetukseen ja miten niitä voidaan parhaiten soveltaa osana tietokoneavusteista oppimisympäristöä.

Toinen luku keskittyy Sagen yleisimpien ominaisuuksien esittelyyn ja johdattelee lyhyesti Python-ohjelmointikielen perusteisiin. Python on syntaksiltaan yksinkertainen korkeantason ohjelmointikieli, joka toimii Sagen komentokielenä muun muassa algoritmien ja interaktiivisten sovellusten laadinnassa. Se soveltuu erinomaisesti ensimmäiseksi ohjelmointikieleksi ohjelmoinnista kiinnostuneille opiskelijoille. Sagen peruskäyttö, kuten lausekkeiden symbolinen käsitteleminen, yhtälöiden ratkaiseminen ja funktioiden integroiminen, ei kuitenkaan edellytä ohjelmointitaitoa.

Luvut 3-7 esittelevät aihealueittain, miten Sagea voidaan hyödyntää lukion matematiikan opetuksessa kursseilla, joilla käsitellään funktioita ja yhtälöitä, geometriaa ja trigonometriaa, differentiaali- ja integraalilaskentaa, todennäköisyyslaskentaa sekä numeerisia menetelmiä. Kussakin luvussa esitetään aihepiiriin liittyviä metodeja ja havainnollistetaan niiden toimintaa esimerkein. Lisäksi esitellään erityisesti tätä tutkielmaa varten ohjelmoituja, aihealueen opetusta tukevia vuorovaikutteisia sovelluksia.

Luvut päättyvät tehtäväosioihin, joiden ratkaisut ovat tutkielman liitteenä. Monet tehtävistä perustuvat vuorovaikutteisista sovelluksista tehtäviin päättelyihin. Tehtävänä voi olla esimerkiksi tutkia, miten tehtävään liittyvien parametrien muuttaminen vaikuttaa tarkasteltavaan ilmiöön. Osassa tehtävistä käsitellään sellaisia realistisia ongelmia, joiden ratkaiseminen perinteisin keinoin kynän ja paperin avulla olisi erityisen työlästä tai hankalaa. Useat tehtävistä perustuvat matematiikan ylioppilaskokeiden tehtäviin, mikä saattaa osaltaan motivoida lukiolaisia. Näissä tehtävissä ongelmaa tarkastellaan eri näkökulmasta kuin alkuperäisessä yhteydessä, esimerkiksi numeerisin menetelmin tai ohjelmiston suorittamaan symboliseen laskentaan perustuen.

Tutkielma perustuu Sagen kirjoitushetkellä uusimpaan versioon 4.7. Aiheet ja niiden käsittelytavat on pyritty valitsemaan siten, että ne pysyisivät relevantteina myös myöhemmissä versioissa. Suurin osa tutkielmaa varten kirjoitetuista sovelluksista on julkaistu osana Sagea versiosta 4.7.1 alkaen. Näitä sovelluksia voidaan käyttää Sagessa ilman, että ohjelmakoodeja tarvitsee erikseen kopioida työarkkeihin.

Haluan kiittää tutkielmani ohjaajaa professori Matti Vuorista saamastani palautteesta ja ideoista, veljeäni Aarni Koskelaa teknisestä tuesta sekä Sagen kehitysyhteisöä avusta ja kiinnostuksesta projektia kohtaan.

1 Johdanto

Matemaattiset ohjelmistot on perinteisesti jaettu numeerisesti ja symbolisesti laskeviin ohjelmistoihin. Englanninkielisessä kirjallisuudessa symboliseen laskentaan kykeneviin ohjelmistoihin viitataan usein nimityksillä Computer Algebra System (CAS) tai Computer Algebra Environment. Useimmilla tällaisilla ohjelmistoilla voidaan suorittaa sekä numeerisia että symbolisia laskutoimituksia sekä piirtää kaksi- ja kolmiulotteista grafiikkaa. Osalla ohjelmista voidaan myös luoda vuorovaikutteisia sovelluksia ja laatia saaduista tuloksista dokumentteja käyttäen matemaattista esitystapaa.

Tutkimus- ja opetuskäyttöön kehitettyjä matemaattisia ohjelmistoja on lukuisia. Tunnetuimmat näistä ovat kaupallisia sovelluksia, jotka ovat lisensioitavissa opetuskäyttöön. Esimerkkejä kaupallisista ohjelmistoista ovat *Magma* [1], *Maple* [2], *Mathematica* [3] ja *MATLAB* [4]. Näiden ohjelmistojen etuina ovat käyttäjätuen hyvä saatavuus ja valmiiden sovellusten suuri määrä. Ne ovat usein huolellisesti viimeisteltyjä ja ohjelmistojen toiminnasta on saatavilla kattava ja yksityiskohtainen dokumentaatio. Toisaalta lisensseistä aiheutuvat kustannukset ovat huomattavia, eikä ohjelmistojen koko kapasiteettia useinkaan päästä täysin hyödyntämään opetuskäytössä. Linsessit sallivat harvoin ohjelmistojen vapaan asentamisen opiskelijoiden henkilökohtaisiin tietokoneisiin, jolloin ohjelmistojen hyödyntäminen rajoittuu oppitunneille tai ainoastaan opettajan käyttöön.

Myös ilmaisia ohjelmia ja verkkosovelluksia on saatavilla. Useimmat verkossa käytettävät *Java*- ja *Flash*-pohjaiset ohjelmistot ovat erikoistuneet johonkin tiettyyn tehtävään, kuten kuvaajien piirtämiseen tai numeerisiin laskutoimituksiin. Yksi tällä hetkellä suosituimmista opetuskäyttöä varten kehitetyistä sovelluksista on *GeoGebra* [5], jolla voidaan havainnollistaa muun muassa geometrian, trigonometrian ja algebran aihepiirejä. Muita geometrisia aihepiirejä käsitteleviä dynaamisia ohjelmistoja ovat muun muassa *Geonext* [6] sekä kaupalliset *Cabri* [7] ja *Geometers' Sketchpad* [8]. Symboliseen laskentaan ja matemaattisten erikoisalojen, kuten graafiteorian ja lukuteorian, tutkimukseen soveltuvia ilmaisia ohjelmistoja ovat tässä tutkielmassa esiteltävän Sagen lisäksi muun muassa *Maxima* [9] sekä Python-ohjelmointikielen luokkakirjastoista *SymPy* [10] ja *PyDSTools Symbolic* [11].

1.1 Matemaattiset ohjelmistot opetuskäytössä

Kilpatrick, Swafford ja Findell (2001, [12]) jakavat matemaattisen osaamisen viiden ominaisuuden avulla:

• **Konseptuaalinen ymmärtäminen:** Matemaattisten käsitteiden, operaatioiden ja relaatioiden ymmärtäminen. Toisiinsa liittyvien asioiden yhteyksien ja samankaltaisuuksien havainnoiminen.

• **Proseduraalinen sujuvuus:** Prosessien hallitseminen ja käyttö tarkasti ja tehokkaasti. Sopivien prosessien valitseminen ongelman mukaan. Esimerkiksi yhtälöiden ratkaisumenetelmien ja derivoimisen hallinta.

• **Strateginen osaaminen:** Kyky muodostaa, esittää ja ratkaista matemaattisia ongelmia.

• **Mukautuva päättely:** Kyky loogiseen ajatteluun ja päättelyyn. Reflektoimisen ja todistamisen sujuvuus.

• **Yritteliäisyys:** Matematiikan kokeminen järkevänä, hyödyllisenä ja arvokkaana. Opiskelija uskoo omiin kykyihinsä ja on motivoitunut pitkäjänteiseen työskentelyyn.

Matemaattiset ohjelmistot laskevat nopeasti ja tarkasti. Korkean tason metodien avulla voidaan suorittaa monimutkaisia laskutoimituksia ilman, että käyttäjän tarvitsee olla tietoinen käytettävän algoritmin toimintavaiheista. On esitetty, että matemaattisen ohjelmiston käytöllä voidaan tehostaa konseptuaalista oppimista, kuten matemaattisten käsitteiden ja ilmiöiden ymmärtämistä sekä ongelmanratkaisussa tarvittavien taitojen oppimista realististen ongelmien avulla (Heid 1988, [13]; Hillel 1993, [14]). Esimerkiksi kuvaajien tarkasteluun voidaan käyttää enemmän aikaa, kun niitä ei piirretä käsin (Fey 1989, [15]).

Erityisesti heikompitasoiset oppijat voivat hyötyä matemaattisen ohjelmiston välittömyydestä ja tarkkuudesta. Opiskelija voi keskittyä tutkimaan ongelmaa ja kehittämään konseptuaalista ymmärrystään, vaikka hänellä olisi vaikeuksia esimerkiksi lausekkeiden rutiininomaisessa mekaanisessa käsittelyssä. (Hillel 1993, [14]; Kieran & Damboise 2007, [16].) Tämä voi myös parantaa motivaatiota aiheen oppimista kohtaan. Matemaattisen ohjelman avulla oppija voi kehittää strategista osaamistaan tutkimalla esimerkiksi algebrallisia lausekkeita eri esitystapojen avulla, kuten graafisesti, numeerisesti ja symbolisesti, ja reflektoimalla oppimaansa (Porzio 1999, [17]; Pierce & Stacey 2004, [18]).

Voidaan kysyä, heikentääkö matemaattisten ohjelmien käyttö proseduraalisen osaamisen tasoa, jos ongelman ratkaisussa sivuutetaan tarkastelun välivaiheet. Mayes (1997, [19]) vertasi kokeellisia tutkimuksia, joissa selvitettiin ohjelmistojen vaikutusta oppimistuloksiin. Vertailussa havaittiin enemmän heikosti menestyneitä kokeiluja

tapauksissa, joissa matemaattisia ohjelmistoja käytettiin pelkästään perinteisten opetusmenetelmien tehokkuuden ja laskemisen nopeuden kasvattamiseen. Ohjelmistojen pinnallisen ja tuottamattoman käytön välttämiseksi opetusmenetelmien valinnalla tulee olla selkeät ja pätevät pedagogiset sekä käytännölliset perusteet (Pierce & Stacey 2004, [18]).

On useita viitteitä, että matematiikan konseptuaalisen ymmärryksen kehittyminen matemaattisten ohjelmien avulla voi myös tukea proseduraalista oppimista (ks. Nabb 2010, [20]). Keller & Russel (1997, [21]) tutkivat symboliseen laskentaan kykenevien laskinten käyttöä oppijakeskeisessä opetuksessa. Yliopisto-opiskeljoista koostuneessa ryhmässä matemaattista ohjelmaa käyttäneet menestyivät paremmin loppukokeen tehtävien symbolisessa ratkaisemisessa kuin vertailuryhmä. Samansuuntaisia tuloksia saatiin suomalaisessa tutkimuksessa, jossa selvitettiin derivaatan käsitteen havainnollistamista ohjelmiston avulla, kun kohderyhmänä oli lukiolaiset (Repo 1994, [22]).

Tässä tutkielmassa tarkastellaan useita vuorovaikutteisia sovelluksia, joiden avulla matemaattisia käsitteitä ja ilmiöitä voidaan tutkia dynaamisesti. Kaikissa sovelluksissa on tarkasteltavaan aiheeseen liittyviä parametreja, joiden muuttamisen vaikutus havaitaan aihetta esittävissä kuvissa ja datassa. Esimerkki tällaisesta interaktiivisesta sovelluksesta on ohjelma, jossa toisen asteen polynomifunktion vakiokertoimien arvoja voidaan muuttaa liukusäätimien avulla, ja tutkia, miten parametrien muutos vaikuttaa funktion kuvaajaan (ks. luku 3.5.1). Kuvan alla ohjelma esittää toisen asteen yhtälön ratkaisukaavan, johon ohjelma sijoittaa käytettävät parametrit ja tulostaa vastauksen. Ratkaisukaavan diskriminantin väri riippuu yhtälön ratkaisujen lukumäärästä. Ohjelman tavoitteena on havainnollistaa polynomifunktion kuvaajasta nähtävien nollakohtien ja käyrän muodon yhteyttä toisen asteen yhtälön juuriin ja ratkaisujen lukumäärään. Toisaalta sovellus painottaa analyyttisen tarkastelun proseduraalista osuutta, eli ratkaisukaavan soveltamista käytännössä.

Drijvers (2003, [23]) käsittelee empiirisessä tutkimuksessaan algebran ja erityisesti parametrin käsitteen opettamista tietokoneavusteisessa ympäristössä. 14-15 -vuotiaat opiskelijat perehtyivät symbolista laskinta käyttäen parametrin käsitteen eri merkityksiin. Esimerkiksi yhtälössä $y = ax + b$ kirjainta a voidaan ajatella määrättynä vakiona, muuttuvana arvona tai edustavan jotain joukkoa, kuten kokonaislukuja. Eri merkityksiä voidaan havainnollistaa tilanteesta riippuen esimerkiksi ratkaisemalla yhtälö eri muuttujien suhteen, tai graafisesti yksittäisten kuvien, animaatioiden tai dynaamisten sovellusten avulla.

Tutkimuksen tuloksena havaittiin symbolisen ohjelman selkeyttävän ongelmanratkaisustrategioita joidenkin oppilaiden kohdalla, mutta osalle oppilaista parametrien käyttö

vaikeutti asian ymmärtämistä. Erilaisten muuttujien määrä saattoi vaikuttaa hämmentävältä niiden opiskelijoiden mielestä, jotka kokivat kirjainten merkitysten tunnistamisen hankalaksi. Tutkimuksessa esitetään, että tietokoneavusteinen opetus voi tukea algebrallista ymmärtämistä, mutta sen vaikutus riippuu ennen kaikkea opetuksen didaktisista lähtökohdista ja opetusympäristön käytöstä. Tutkimuksessa korostetaan opettajan merkitystä tulosten seurausten, yhteyksien ja merkitysten tulkinnassa esimerkiksi luokkakeskustelun keinoin.

Edellä esitettyjen tutkimusten valossa tässä tutkielmassa esitettävät, Sage-ohjelman avulla tuotetut esimerkit ja vuorovaikutteiset ohjelmat soveltuvat ensisijaisesti oppimisen apuvälineiksi, joilla voidaan täydentää oppitunneilla käytettäviä oppimismenetelmiä. Symboliseen laskentaan kykenevä ohjelma voi auttaa monimutkaisten asioiden hahmottamisessa ja analysoinnissa. Ongelmanratkaisussa painopiste siirtyy laskuteknisistä seikoista kokonaisuuksien ymmärtämiseen ja hallitsemiseen, mikä voi tukea myös proseduraalista oppimista. Tärkeää on myös työn ohjaaminen ja ohjelmiston antamien tulosten tulkitseminen, missä opettajan toiminnalla on merkittävä rooli.

2 Johdatus Sage-ohjelmiston käyttöön

Tässä luvussa tarkastellaan Sage-ohjelmiston perusominaisuuksia ja kehitystyön taustaa, esimerkkejä ohjelman laskinkäytöstä sekä erilaisten kuvaajien ja animaatioiden laatimisesta. Lisäksi perehdytään Sagen komentokielen Pythonin perusteisiin.

2.1 Sage-ohjelmiston kehitystyön taustaa ja ominaisuuksia

Ensimmäinen versio Sagesta julkaistiin vuonna 2005, jonka jälkeen ohjelmistoa on laajennettu huomattavan nopeasti vapaaehtoisen yhteisön voimin. Sen jakelu ja kehitystyö perustuvat vapaaseen ohjelmistolisenssiin ja avoimen lähdekoodin periaatteelle. Sagen *GNU General Public License version 2* -lisenssi sallii ohjelmiston vapaan opetuskäytön, kopioimisen ja muokkaamisen. [24]

Sage-projektin päätavoitteena on luoda kattava ja ilmainen vaihtoehto kaupallisille matemaattisille ohjelmistoille, kuten *Mathematicalle*, *MATLABille*, *Magmalle* ja *Maplelle*. Toinen motiivi on tutkimuslähtöinen: kaupallisten ohjelmistojen lähdekoodia ja algoritmeja ei useinkaan ole julkisesti saatavilla, mikä tekee tulosten varmistamisesta mahdotonta. Tällaisten ohjelmien sisäänrakennettujen toimintojen käyttäminen voi olla riittämätöntä erityisesti sellaisissa matemaattisissa tarkasteluissa, jotka perustuvat pääasiassa algoritmien antamiin tuloksiin. Avoimen lähdekoodin periaatetta noudattavien ohjelmistojen algoritmit ovat kaikkien arvioitavissa. [25, 26]

Sagen toiminta-ajatuksena on yhdistää valmiiksi pitkälle kehitettyjä ohjelmapaketteja yhdeksi, laajasti eri matematiikan osa-alueita kattavaksi kokonaisuudeksi. Sage sisältää muun muassa seuraavat matemaattiset moduulit:

- *Maxima* ja *SymPy* (symbolinen laskenta ja differentiaalilaskenta),
- *GSL*, *SciPy*, *NumPy* ja *ATLAS* (numeerinen laskenta),
- *GAP*, *Maxima* ja *Singular* (algebra),
- *NetworkX* (graafiteoria),
- *Symmetrica* (kombinatoriikka),
- *ATLAS*, *BLAS*, *LAPACK*, *NumPy*, *Linbox*, *GSL* ja *IML* (lineaarialgebra),
- *PARI/GP*, *NTL* ja *FLINT* (lukuteoria),
- *SciPy* ja *R* (tilastollinen laskenta).

Yhteensä Sagessa on lähes sata ohjelmapakettia. [27]

Samoin kuin *Maple*, *Mathematica* ja *MATLAB*, Sage on tulkki, eli se ei käännä ohjelmakoodia konekieliseksi. Sagen komentokielenä toimiva *Python* on suosittu ja laajalle

levinnyt korkeantason kieli, jota opetetaan monissa yliopistoissa ensimmäisenä ohjelmointikielenä. [28] Sen syntaksi on yksinkertaista ja helposti opittavaa. Sagen käyttäminen laskimen tapaan yksirivisillä syötteillä ei kuitenkaan edellytä esitietoja ohjelmoinnista, joten se soveltuu hyvin toisen asteen koulutuksessa käytettäväksi ohjelmistoksi. Toisaalta ohjelmoinnista kiinnostuneet opiskelijat voivat tehdä Sagen avulla näyttäviä ja monimutkaisia sovelluksia.

Sagea voi käyttää usealla eri tavalla. Sen voi asentaa käyttäjän henkilökohtaiselle tietokoneelle tai etäkäyttää palvelimelle asennettua ohjelmistoa verkkoselaimen avulla. [29] Etäkäyttö mahdollistaa Sagen ja palvelimelle tallennettujen muistikirjojen käytön miltä tahansa verkkoyhteydelliseltä tietokoneelta. Haittapuolena on viive tiedonsiirrossa, jota ei esiinny paikallisesti asennetussa ohjelmistossa. Toisaalta etäkäytettynä kaikki prosessit tapahtuvat palvelimella, mikä ei vaadi päätetietokoneelta juurikaan laskentatehoa ja toimii siten vanhemmallakin laitteistolla. Sagessa on kaksi käyttöliittymää: komentorivikonsoli ja verkkoselaimen graafinen *Notebook*-ympäristö [30]. Näistä jälkimmäinen on luontevampi opetuskäytössä.

Sagen kehitystyössä painotetaan opetuskäyttöä tukevia ominaisuuksia, kuten vuorovaikutteisia toimintoja. Muita matematiikan opetuksessa hyödyllisiä työkaluja ovat erilaisten kuvaajien ja animaatioiden laadintaan liittyvät ohjelmafunktiot. Sagen työarkkeja voi muotoilla, niihin voi liittää matemaattisia symboleja sisältävää tekstiä, linkkejä, kuvia ja muuta sisältöä. Työarkkeihin voidaan myös upottaa kolmansien osapuolien tuottamia HTML-ympäristössä toimivia ohjelmia, kuten GeoGebra-sovelluksia. [31]

Työarkit sisältävät kaikki Sagen istunnon aikana suoritetut laskutoimitukset ja niihin liittyvän grafiikan sekä mahdolliset käyttäjän lisäämät muistiinpanot ja huomiot. Työarkkeja voi julkaista palvelimella, jolloin ne ovat kaikkien nähtävissä, tai lähettää yksittäisille käyttäjille, esimerkiksi opettajalle tai opetusryhmän muille jäsenille. [31, 32] Eri medioita yhdistelemällä on mahdollista tuottaa monipuolista ja interaktiivista oppimateriaalia, jota voidaan jakaa palvelimen käyttäjien kesken. Vastaavasti opiskelijat voivat laatia dokumentteja ja raportteja työnsä tuloksista.

Opetuskäytön kannalta yksi Sagen mielenkiintoisimmista ominaisuuksista on mahdollisuus ohjelmoida vuorovaikutteisia sovelluksia. Tällaisten interaktiivisten ohjelmien parametreja voidaan manipuloida ajon aikana erilaisilla säätimillä, kytkimillä ja valikoilla. Ohjelman ajon aikana muutettavissa oleva syöte voi olla myös esimerkiksi lukuarvo tai matemaattinen funktio. [33] Opiskelija voi halutessaan tutustua tarkemmin ohjelman toimintaan tutkimalla sen lähdekoodia.

Opetukseen liittyvien didaktisten ja teknisten ongelmien keskustelemista varten on perustettu keskusteluryhmä *sage-edu*. [34]

2.2 Sagen laskinkäyttö

Notebook eli *muistikirja* on Sagen graafinen käyttöliittymä, joka toimii rajapintana Sage-palvelimen ja käyttäjän välillä. Muistikirja sallii muun muassa laskutoimitusten suorittamisen palvelimella, ohjelmakoodin kirjoittamisen ja ajamisen, kaksi- ja kolmiulotteisten kuvaajien esittämisen sekä työarkkien organisoimisen ja jakamisen muiden Sagen käyttäjien välillä.

Worksheet eli *työarkki* on Sagen dokumentti, jossa tehdyt laskutoimitukset ja objektit on järjestetty niin sanottuihin *soluihin* (*cell*). Työarkki koostuu käyttäjän syötteistä ja työarkin yhteyteen talletetuista objekteista, kuten kuva- ja tekstitiedostoista.

Käyttäjän tekemät muuttujien ja funktioiden määrittelyt pysyvät palvelimen muistissa niin kauan, kun työarkki pysyy aktiivisena. Työarkkeja voidaan ladata muistikirjaan ja tallentaa tiedostoiksi. Työarkkien tiedostoformaatin pääte on *sws* (*Sage WorkSheet*).

Tarkastellaan seuraavaksi Sagen laskentaympäristöä. Tyhjässä työarkissa on valmiina yksi solu, johon laskutoimitus tai ohjelmalause voidaan kirjoittaa. Solu evaluoidaan joko klikkaamalla tekstiä *evaluate* tai näppäinkomennolla *shift+enter*. Suoritetaan aluksi yksinkertainen laskutoimitus:

```
3 + 2
```

```
5
```

Uusi solu luodaan viemällä hiiren osoitin solun alapuolelle ja klikkaamalla ilmestyvää sinistä vaakasuoraa palkkia. Toinen tapa on käyttää solun evaluointiin näppäinyhdistelmää *alt+enter*, jolloin uusi solu luodaan automaattisesti laskutoimituksen suorituksen jälkeen. Uusi solu tulee näkyviin myös siinä tapauksessa, että evaluoitava solu on laskentaympäristön viimeinen aktiivinen solu. Solu poistetaan tyhjentämällä sen sisältö ja painamalla *backspace*-näppäintä solun ollessa aktiivinen.

Laskennan voi milloin tahansa keskeyttää *escape*-painikkeella. Kirjoittamalla soluun *"restart"* Sage tyhjentää työarkkiin liittyvän muistin, muun muassa muuttujat ja sovelluksien tilan.

Lisätietoa työkirjan näppäinkomennoista on viitteessä [35].

2.2.1 Peruslaskutoimitukset

Sagessa käytetään seuraavia merkintöjä peruslaskutoimituksista yhteen- ja vähennyslaskuoperaattoreiden (+ ja -) lisäksi:

Matemaattinen notaatio	Merkintä Sagessa
ab	a*b
$\frac{a}{b}$	a/b
a^n	a^n *tai* a**n
$\sqrt{a}$	sqrt(a)
$\sqrt[n]{a}$	a^(1/n)
$\lvert a \rvert$	abs(a)
$\log_k x$	log(x, k)
$\ln x$	ln(x) *tai* log(x)

Taulukko 1: Peruslaskutoimitukset Sagessa.

On huomattava, että kertolaskussa on aina käytettävä asteriskia (*) tulontekijöiden välissä. Sage pyrkii sieventämään lausekkeita aina, kun se on mahdollista.

Esimerkkejä: Peruslaskutoimituksia

```
1/2 + 2/3 + 3/4
```

$$\frac{23}{12}$$

```
show(30/sqrt(5) + log(11,7) + log(25,5))
```

$$6*\sqrt{5} + \frac{\log(11)}{\log(7)} + 2$$

Monimutkaiset lausekkeet voidaan esittää matemaattisella notaatiolla kirjoitettuna. Tämä tehdään joko valitsemalla *Typeset*-tila aktiiviseksi muistikirjan asetuksista tai kirjoittamalla soluun *show(lauseke)*.

Esimerkki: Kaavan esittäminen show-metodin avulla

```
show( abs(-sqrt(7) / 3^log(2)) )
```

$$\left|\frac{-\sqrt{7}}{3^{\log 2}}\right|$$

2.2.2 Matemaattisia vakioita

Seuraavassa taulukossa on esitetty yleisimpien matemaattisten vakioiden ja symbolien syntaksimerkinnät [36]:

Symboli	Merkintä Sagessa
π	pi
e	e
i	i
ϕ	golden_ratio
∞	oo

Taulukko 2: Matemaattisia vakioita Sagessa.

Esimerkkejä:

Show-metodi esittää matemaattiset vakiot symboleina.

```
show(3*e^(-2*pi))
```

$$3e^{(-2\pi)}$$

Seuraavassa esimerkissä lauseke sieventyy, kun käytetään Sagen sisäänrakennettuja vakioita:

```
ln(e^2) + cos(pi)
```

$$1$$

2.2.3 Desimaaliarvot

Sage suorittaa laskutoimitukset aina symbolisesti, ellei erikseen määritellä käytettäväksi numeerisia menetelmiä. Tulokselle voi pyytää desimaaliarvon funktiolla *n(tulos)*. Komento *n(tulos, b)* palauttaa desimaaliluvun, joka on laskettu vähintään *b* bitin tarkkuudella. Jos tarkkuutta ei erikseen määritellä, likiarvo määritetään 53 bitin tarkkuudella. Desimaaliarvossa esiintyvien numeroiden lukumäärä (d) voidaan määrätä *digits*-parametrilla: *n(tulos, digits=d)*. [37]

Esimerkki: Piin likiarvon määrittäminen n-metodin avulla

Määritetään piin likiarvo *n*-metodin oletusasetuksilla ja 200 numeron tarkkuudella.

```
n(pi)
```

```
3.14159265358979
```

```
n(pi, digits=200)
```

```
3.14159265358979323846264338327950288419716939937510582097494459230
78164062862089986280348253421170679821480865132823066470938446095 50
58223172535940812848111745028410270193852110555964462294895493038 20
```

Laskennassa käytetään desimaaliarvoja, jos jokin lausekkeessa esiintyvistä luvuista on desimaalimuodossa.

Esimerkki:

Merkitsemällä luku 5 muodossa 5.0 ohjelma käyttää laskennassa likiarvoja tarkkojen arvojen sijaan.

```
sqrt(5) + sqrt(4)
```

```
sqrt(5) + 2
```

```
sqrt(5.0) + sqrt(4)
```

```
4.23606797749979
```

2.3 Symbolinen laskenta

Symbolisella laskennalla tarkoitetaan muuttujia sisältävien lausekkeiden käsittelyä, kuten sieventämistä, yhtälöiden ratkaisemista, derivointia ja integrointia. Sagessa voidaan määritellä tuntemattomia muuttujia sisältäviä lausekkeita, kun kaikki tarvittavat muuttujat on ensin luotu *var*-komennon avulla.

Esimerkki:

```
x = var("x")
3*x^2
```

```
3*x^2
```

Metodin *var* argumenteissa määritellään muuttujan nimi, joka lähes poikkeuksetta kannattaa valita samaksi kuin laskennassa käytettävä muuttujakirjain. Samalla rivillä voidaan esitellä myös useampia muuttujia samanaikaisesti.

Esimerkkejä:

```
z = var("z")
sqrt(2)*z + 2*z + 2/3*z
```

```
sqrt(2)*z + 8/3*z
```

```
x, y = var("x, y")
show(e^(x+y) + abs(x^2-y^2))
```

$$e^{x+y} + |x^2 - y^2|$$

Muuttujille annetaan arvoja käyttämällä yhtäsuuruusmerkkiä (=) asetuslauseen tunnuksena. Yhtälöissä yhtäsuuruutta ja totuusoperaattoreista ekvivalenssia merkitään kahdella peräkkäisellä yhtäsuuruusmerkillä (==). Ennestään esittelemätöntä muuttujaa ei tarvitse alustaa erikseen, jos sille asetetaan välittömästi jokin arvo.

Esimerkki:

```
x = 2
y = 3
x^2 + y^2
```

```
13
```

Lausekkeeseen voidaan sijoittaa arvoja muuttujien paikalle siten, että asetuslauseet kirjoitetaan lausekkeen jälkeen sulkeisiin. Myös lausekkeen tulee tällöin olla sulkeissa.

Esimerkki:

```
x, y = var("x, y")
(x^2 + y^2)(x=2, y=3)
```

```
13
```

Lausekkeita voidaan nimetä samaan tapaan kuin muuttujia:

```
x = var("x")
a = x^2 - 2
b = 3*x
a + b
```

```
x^2 + 3*x - 2
```

Nimettyjen lausekkeiden sieventäminen tapahtuu komennolla *full_simplify* ja auki kertominen metodilla *expand*.

Esimerkki:

```
x = var("x")
a = (x - 1)^4
b = a.expand()
b
```

```
x^4 - 4*x^3 + 6*x^2 - 4*x + 1
```

Hyödyllinen metodi on myös *factor*, joka jakaa lausekkeen tekijöihinsä. Lausekkeita muokkaavia metodeita voidaan kohdistaa suoraan laskulausekkeisiin, kuten seuraavassa esimerkissä tehdään.

Esimerkki:

```
(x^2 - 2*x + 1).factor()
```

```
(x - 1)^2
```

Lisätietoa Sagen lausekkeiden käsittelystä löytyy viitteestä [38]. Symboliseen laskentaan sekä funktioiden ja yhtälöiden määrittämiseen palataan tarkemmin seuraavassa luvussa.

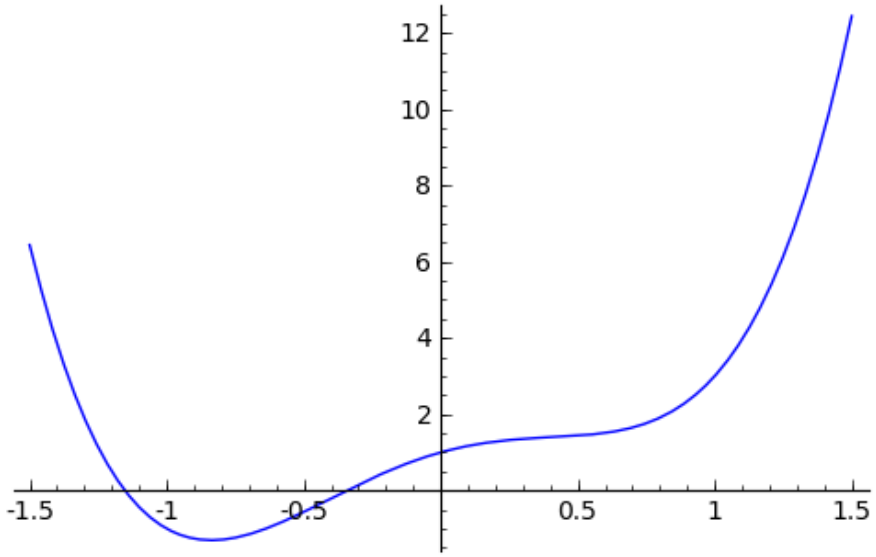
2.4 Kuvaajat

Kaksiulotteisia, muotoa $y = f(x)$ olevia kuvaajia, piirretään metodilla *plot(funktio, muuttuja, piirtoväli)*. Yksinkertaisimmillaan metodille voidaan antaa syötteenä pelkkä laskulauseke, jolloin Sage päättelee, minkä muuttujan suhteen kuvaaja piirretään. Oletusasetuksilla kuvaaja piirretään välillä $(-1, 1)$.

Esimerkki: Kuvaajan piirtäminen

Piirretään polynomin $3x^4 - 3x^2 + 2x + 1$ kuvaaja välillä $(-1.5, 1.5)$:

```
plot(3*x^4 - 3*x^2 + 2*x + 1, x, (-1.5, 1.5))
```



Kuva 1: Yksinkertainen kuvaaja *plot*-metodin avulla piirrettynä.

Kuvaajalle voidaan antaa muun muassa seuraavia lisämääreitä eli optioita [39]:

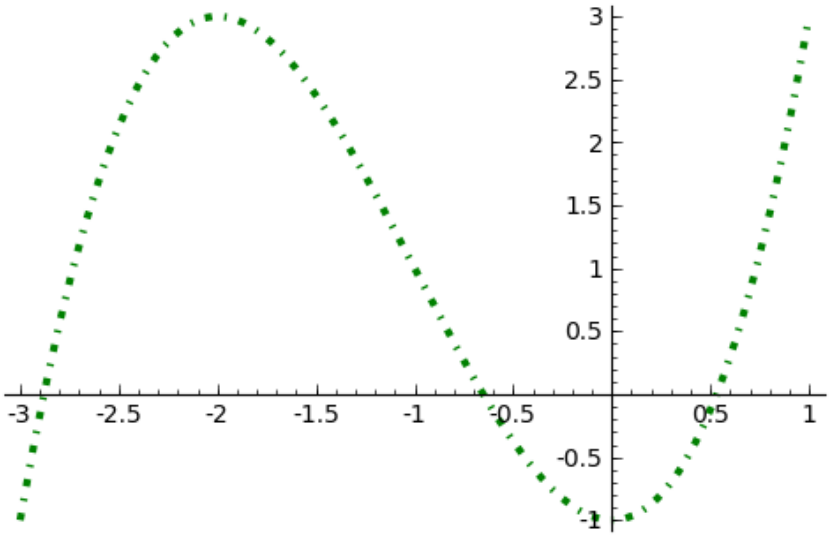
Optio	**Parametrin tyyppi**	**Kuvaus**
xmin, xmax, ymin, ymax	reaaliluku	Määrittelee piirtovälin. Oletuksena *ymin* ja *ymax* ovat funktion pienin ja suurin arvo tarkasteluvälillä.
rgbcolor	värin nimi, *(r, g, b)-tuple* tai *html*-värikoodi	Määrittelee käyrän värin, esim. *"red"*, *(0.6, 0.7, 0.2)* tai *"#aaff0b"*.
hue	reaaliluku väliltä 0-1	Toinen tapa määritellä viivan värisävy.
thickness	reaaliluku	Viivan leveys.
alpha	reaaliluku väliltä 0-1	Viivan läpinäkyvyys. Arvo 0 tarkoittaa näkymätöntä ja 1 täysin näkyvää viivaa.
linestyle	”-”, ”-.”, ”–”, ”:”	Viivan tyyli.

Taulukko 3: Metodin **plot** lisämääreitä.

Esimerkki: Plot-metodin optioita

Piirretään polynomi $x^3 + 3x^2 - 1$ välillä $[-3, 1]$ siten, että viivan väriksi on määritelty vihreä, leveydeksi arvo kolme ja tyyliksi eräänlainen katkoviiva.

```
plot(x^3 + 3*x^2 - 1, (-3, 1), rgbcolor="green", thickness=3,
   linestyle="-.")
```



Kuva 2: Esimerkki *plot*-metodin optioiden käytöstä.

Sagen kuvaajat ovat graafisia objekteja, joita voidaan nimetä ja käsitellä muiden graafisten objektien tapaan. Tällaisia objekteja voidaan esittää kuten laskulausekkeita metodilla *show*. [40] Useita kuvia voidaan yhdistää yhdeksi kuvaksi käyttämällä ”+”-symbolia.

Seuraavassa taulukossa esitellään *show*-metodin optioita. Nämä lisämääreet voidaan antaa myös *plot*-metodille, jolloin ne välittyvät automaattisesti metodin *show* optioiksi.

Optio	**Parametrin tyyppi**	**Kuvaus**
figsize	Muodossa [*leveys*, *korkeus*], missä annetut arvot ovat reaalilukuja.	Kuvan kehyksen koko.
aspect_ratio	reaaliluku	Kuvan korkeuden suhde leveyteen. Arvolla 1 ympyrät näkyvät pyöreinä.
axes	*True* tai *False*	Määrittää piirretäänkö koordinaattiakselit.
axes_labels	("*x-akselin nimi*", "*y-akselin nimi*")	Koordinaattiakselien nimet.

Taulukko 4: Metodin **show** lisämääreitä.

Kuvaajiin voidaan lisätä tekstiä metodilla *text(teksti, (x, y))*. Ensimmäisenä argumenttina tekstiobjektille annetaan tulostettava teksti ja toisena tekstin sijainti koordinaatistossa. Tekstin avulla voidaan esimerkiksi nimetä kuvaajassa esiintyviä funktioita.

Tekstiobjektille voidaan antaa muun muassa seuraavia lisämääreitä [41]:

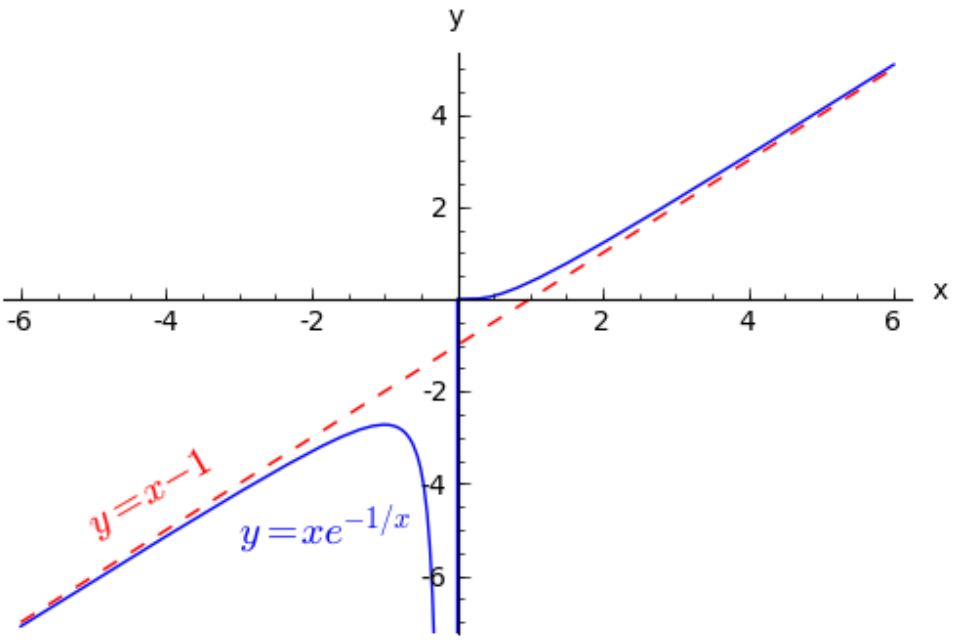
Optio	**Parametrin tyyppi**	**Kuvaus**
fontsize	kokonaisluku	Fontin koko.
rgbcolor	värin nimi, *(r, g, b)-tuple* tai *html*-värikoodi	Tekstin väri.
rotation	asteluku, "*vertical*" tai "*horizontal*"	Tekstin kallistuskulma suhteessa vaakatasoon.
vertical_alignment	"*top*", "*center*"tai "*bottom*"	Tekstin sijoittelu pystysuunnassa.
horizontal_alignment	"*left*", "*center*"tai "*right*"	Tekstin sijoittelu vaakasuunnassa.
axis_coords	*True* tai *False*	Jos True, niin piste (0, 0) vastaa kuvaajan vasenta alakulmaa ja (1, 1) oikeaa yläkulmaa koordinaatistosta riippumatta. Oletuksena False, jolloin teksti sijoitetaan annettuun koordinaattipisteeseen.

Taulukko 5: Metodin **text** lisämääreitä.

Esimerkki: Kuvien yhdistäminen ja *text*-metodi

Havainnollistetaan käyrän $y = xe^{-1/x}$ asymptoottia $y = x - 1$ kuvaajan avulla. Asymptoottisuoran ja käyrän erottaa parhaiten toisistaan, kun ne piirretään erivärisinä ja kummallekin kuvaajalle valitaan erilaiset viivan tyylit. Esitetään lisäksi funktioiden lausekkeet koordinaatistossa *text*-metodin avulla siten, että lausekkeiden värit vastaavat kuvaajien värejä. Kirjoittamalla lausekkeen ympärille $-merkit lauseke näkyy matemaattisella notaatiolla kirjoitettuna. *Show*-metodissa yhdistetään kuvaajat ja tekstit yhdeksi kuvaksi sekä nimetään x- ja y-akselit.

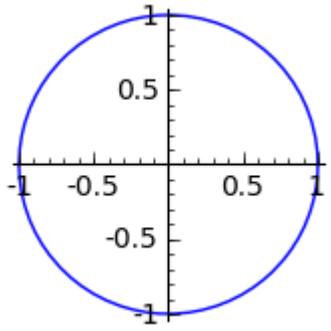
```
a = plot(x*e^(-1/x), (-6, 6), ymin=-7)
b = plot(x-1, (-6, 6), ymin=-7, rgbcolor="red", linestyle="--")
a_teksti = text("$y = xe^{-1/x}$", (-1.8, -5), rgbcolor="blue",
   fontsize=15)
b_teksti = text("$y = x-1$", (-4.2, -4.2), rgbcolor="red",
   fontsize=15, rotation=30)
show(a + b + a_teksti + b_teksti, axes_labels=("x", "y"))
```



Esimerkki: Useita kuvaajia samassa koordinaatistossa

Useita kuvaajia voidaan piirtää samaan koordinaatistoon *plot*-metodin avulla kirjoittamalla lausekkeet sulkeiden sisään ja erottamalla ne pilkuilla. Tällöin valitut optiot ovat samoja molemmissa kuvaajissa.

```
plot((sqrt(1-x^2), -sqrt(1-x^2)), aspect_ratio=1, figsize=[2,2])
```



2.5 Python-ohjelmointi Sagessa

Sage käyttää komentokielenään Pythonia, joka on monipuolinen ja laajalle levinnyt korkeantason ohjelmointikieli. Sen erityispiirre on helposti omaksuttava, hyvin selkeä syntaksi. Sagessa on Python-tulkki, joka lukee ohjelmakoodin ja suorittaa sen komennot kääntämättä koodia ensin konekieliseksi. Sagen peruskäyttö ei edellytä Python-kielen hallitsemista, mutta sovellusten laadinnassa ohjelmointikielen perusteiden tunteminen on välttämätöntä.

Tässä luvussa esitellään lyhyesti Pythonin tärkeimpiä ominaisuuksia niiltä osin kuin niitä myöhemmin tutkielmassa hyödynnetään. Suppean käsitellyn vuoksi esimerkiksi Pythonin laajat olio-ohjelmointiominaisuudet sivuutetaan. Suositeltavaa Python-ohjelmointiin liittyvää kirjallisuutta ovat muun muassa teokset *Learning Python* [42] ohjelmoinnin vasta-alkajille ja *Dive Into Python* [43] ennestään muilla kielillä ohjelmoinneille.

2.5.1 Tulostus näytölle

Tekstin tulostus tapahtuu Pythonissa *print*-komennolla. Ohjelmointikielen versiosta 3.0 alkaen *print*-metodin yhteydessä tulee käyttää sulkeita. Suuri osa kolmansien osapuolten ohjelmamoduuleista ei kuitenkaan vielä ole yhteensopivia Pythonin uusimman versiosukupolven kanssa, joten tässä tutkielmassa ei käytetä sulkeita *print*-metodin yhteydessä. Sagessa kumpikin tapa on sallittu. Vastaavasti johdonmukaisuuden vuoksi käytetään tekstisyötteissä pääsääntöisesti lainausmerkkejä ("), mutta myös heittomerkkien (') käyttö on mahdollista. Tämä on tarpeellista etenkin silloin, kun tekstissä itsessään halutaan käyttää lainausmerkkejä.

Esimerkki:

```
print "Testi", 3, '"Lainaus"'
```

```
Testi 3 "Lainaus"
```

Sagessa sovellukset voivat tulostaa myös HTML-merkintää *html*-metodin avulla. Tässä tutkielmassa ei perehdytä tarkemmin HTML-ohjelmointiin, mutta hyviä oppaita on runsaasti saatavilla, muun muassa *w3schoolsin* [44] tutoriaalit verkossa. HTML-merkinnän avulla sovellukset voivat esittää dynaamisesti muuttuvia taulukoita, kuvia, linkkejä ja muita HTML-objekteja.

Esimerkki: HTML-linkki Pythonin kotisivuille

```
html("Pythonin kotisivut: <a href='http://www.python.org'>
   www.python.org</a>")
```

```
Pythonin kotisivut: www.python.org
```

Useimmat kaavat ja matemaattiset merkinnät voidaan esittää LaTeX-ladottuna. Tästä on hyötyä erityisesti HTML-tilassa, jossa LaTeX-merkinnällä ladottuja matemaattisia lausekkeita voidaan esittää kirjoittamalla dollarimerkit ($) lausekkeen alkuun ja loppuun. Metodi *latex(objekti)* pyrkii muuntamaan annetun syötteen LaTeX-merkinnän mukaiseksi. LaTeX-ladontaan voi tutustua muun muassa suomeksi käännetyssä teoksessa *Pitkänpuoleinen johdanto LaTeX 2e:n käyttöön* [45], joka on saatavilla sähköisessä muodossa.

Esimerkki: Kaavan esittäminen LaTeX-ladottuna

```
kaava = latex(x*e^x + sqrt(3))
print kaava
html("LaTeX-ladottu kaava: $%s$"%kaava)
```

```
x e^x + \sqrt3
LaTeX-ladottu kaava: $xe^x + \sqrt{3}$
```

2.5.2 Tietotyypit

Pythonissa on täysin dynaaminen muuttujien tyypitys. Tämä tarkoittaa, ettei muuttujien tyyppiä tarvitse määritellä tai alustaa erikseen. Tyyppi voi myös muuttua ajon aikana. Sagessa käytetään jonkin verran sen omia tietorakenteita, jotka poikkevat Pythonin vastaavista. Esimerkiksi Pythonin kokonais- ja liukulukuliteraalit (*int* ja *float*) Sage muuntaa sisäisesti omiin muotoihinsa. Muuttujan tyypin voi tulostaa komennolla *type(muuttuja)*.

```
a = 2
print type(a)
b = 2.0
print type(b)
```

```
<type 'sage.rings.integer.Integer'>
<type 'sage.rings.real_mpfr.RealLiteral'>
```

Python sisältää laajan valikoiman perustietotyyppejä, joista yleisimpiä on lueteltu seuraavassa taulukossa. Python sallii myös uusien tietotyyppien luomisen. Mutatoituvien ja mutatoitumattomien listojen sekä hakemistojen määrittelyt eroavat syntaktisesti toisistaan erilaisilla sulkeilla. Mutatoitumattomuudella tarkoitetaan, ettei tietotyypin sisältöä voida muuttaa määrittämisen jälkeen. Python antaa *TypeError*-virheilmoituksen, jos tätä sääntöä yritetään rikkoa. [46, 47]

Tyyppi	**Kuvaus**	**Esimerkki syntaksista**
str, unicode	Mutatoitumaton merkkijono.	"merkkijono", u"merkkijono"
bool	Totuusarvo.	True tai False
list	Mutatoituva lista.	[1.5, "merkkijono", False]
tuple	Mutatoitumaton lista.	(1.5, "merkkijono", False)
set	Järjestämätön joukko.	set([1.5,"merkkijono", False])
dict	Avain-arvo -pareista koostuva hakemisto.	{"avain1": 1.5, "avain2": True}

Taulukko 6: Pythonin yleisimpiä tietotyyppejä.

Matemaattisten joukkojen käsittelyä varten Sagessa ovat metodit *Set* ja *EnumeratedSet* [48]. Sekä Pythonin että Sagen metodeilla voidaan määrittää muun muassa joukkojen unioneita, leikkauksia ja erotuksia.

String-tietotyyppejä voidaan katenoida eli yhdistää peräkkäin plus-operaattorilla (+), toistaa tulo-operaattorilla (*) ja "leikata" (*slicing*) kaksoispisteellä (:). Semanttisesti mielettömät operaatiot, esimerkiksi merkkijonon lisääminen kokonaislukuun, antavat *TypeError*-virheilmoituksen.

Esimerkki: Merkkijonojen katenoiminen, leikkaaminen ja toistaminen

```
merkkijono1 = "Python"
merkkijono2 = "Sage"
print merkkijono1 + ", " + merkkijono2
print merkkijono1[2:5]
print (merkkijono1 + merkkijono2)*2
```

```
Python, Sage
tho
PythonSagePythonSage
```

Esimerkki: Listojen käsittely plus- ja tulo-operaattoreilla

```
t = [1, 2, 3]
print 3*t
print t + [4, 5]
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
[1, 2, 3, 4, 5]
```

Esimerkki: Joukkojen unioni, leikkaus ja erotus

```
s = set([1, 2, 3])
r = set([2, 3, 4])
print s.union(r)
print s.intersection(r)
print s.difference(r)
```

```
set([1, 2, 3, 4])
set([2, 3])
set([1])
```

2.5.3 Listat ja range-metodi

Pythonin mutatoituva lista korvaa monissa ohjelmointikielissä käytetyn taulukon. Listat ovat dynaamisia, eli niiden sisältämiä alkioita ja listan kokoa voidaan muokata ajon aikana. Alkiot voivat olla mitä tahansa olioita, myös toisia listoja. Listoilla on paljon sisäänrakennettuja metodeja, joista osa esitellään taulukossa 7. [49]

Metodi	**Kuvaus**
append(alkio)	Lisää alkion listan viimeiseksi.
extend(lista)	Lisää annetun listan tai muun sarjamuotoisen datan alkiot listan loppuun.
insert(indeksi, alkio)	Lisää alkion indeksin osoittamaan paikkaan listassa.
remove(alkio)	Poistaa alkion listasta.
pop(indeksi)	Palauttaa ja poistaa listasta indeksin osoittaman alkion. Oletuksena pop poistaa viimeisen alkion.
reverse()	Kääntää listan järjestyksen.
sort()	Järjestää listan kasvavaan järjestykseen.

Taulukko 7: Pythonin listoja muokkaavia metodeja.

Pythonissa listan ensimmäinen indeksi on aina 0. Tyhjän listan voi luoda kirjoittamalla *[ ]*. Metodi *len(lista)* palauttaa listan pituuden. Komento *lista[i]* palauttaa listan alkion, jonka indeksi on i. Annetun indeksin ollessa negatiivinen, *lista[-i]*, metodi palauttaa listan lopusta laskien järjestyksessä *i* olevan alkion.

Esimerkki:

Lisätään tyhjään listaan luvut 1 ja 2 sekä listan [4, 5] alkiot:

```
l = []
l.append(1)
l.append(2)
l.extend([4,5])
print l
```

```
[1, 2, 4, 5]
```

Lisätään luku 3 listan paikkaan, jonka indeksi on 2.

```
l.insert(2, 3)
print l
```

```
[1, 2, 3, 4, 5]
```

Poistetaan luku 3 listasta ja käännetään listan järjestys päinvastaiseksi.

```
l.remove(3)
l.reverse()
print l
```

```
[5, 4, 2, 1]
```

Tulostetaan lopuksi listasta alkiot, joista ensimmäisen indeksi on 1 ja jälkimmäinen on listan viimeinen alkio.

```
print l[1]
print l[-1]
```

```
4
1
```

Usein on tarpeellista luoda erilaisia listoja ja lukujonoja, joita voidaan käydä läpi eli iteroida alkio kerrallaan. Tätä tarkoitusta varten Pythonista löytyy metodi *range([lähtöarvo,] loppuarvo[, askel])*. Se palauttaa listan, jonka ensimmäinen alkio on *lähtöarvo*, viimeinen *loppuarvo - 1* ja kahden peräkkäisen alkion välinen erotus on *askel*. Oletusarvoiltaan *lähtöarvo* on 0 ja *loppuarvo* on 1. Esimerkiksi komento *range(i, j)*, missä *i* ja *j* ovat kokonaislukuja, palauttaa listan *[i, i+1, i+2, ..., j-1]*. Jos iteroitava lukualue on suuri, metodi *xrange* on usein tehokkaampi kuin *range*.

Esimerkki:

Luodaan lista, jossa on lueteltu luvut nollasta kuuteen:

```
range(7)
```

```
[0, 1, 2, 3, 4, 5, 6]
```

Tulostetaan kolmen kertotaulu välillä $[3, 30]$.

```
range(3, 31, 3)
```

```
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
```

Vähenevän listan voi tehdä valitsemalla askelen pituuden arvoksi negatiivisen luvun ja sellaiset lähtö- ja lopetusarvot, että lähtöarvo on suurempi kuin lopetusarvo.

Esimerkki:

```
range(15, 1, -3)
```

```
[15, 12, 9, 6, 3]
```

Kirjoittamalla listaan määrittelyvaiheessa kaksi pistettä Sage osaa useimmissa tapauksissa täydentää puuttuvan osan. Tämä on toinen tapa määritellä iteroitava lista.

Esimerkki:

Luodaan lista, jossa on alkiot yhdestä kymmeneen.

```
[1, 2, .. , 10]
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Määritellään lista välillä [3, 15] kahden yksikön välein.

```
[3, 5, .. , 15]
```

```
[3, 5, 7, 9, 11, 13, 15]
```

Tehdään samanlainen lista kuin edellisessä esimerkissä, mutta vähenevänä.

```
[15, 13, .. , 3]
```

```
[15, 13, 11, 9, 7, 5, 3]
```

2.5.4 Aliohjelmat

Monista tunnetuista ohjelmointikielistä poiketen Pythonissa sisennyksillä on syntaktinen merkitys. Ohjelmakoodin sisennys jäsentää lauseet ryhmiksi ehto- ja silmukkarakenteissa sekä alisteisissa ohjelmissa. Lauseet erotellaan toisistaan rivinvaihdoilla eikä monissa kielissä käytettyjä puolipisteitä rivien lopussa ole välttämätöntä käyttää. Sisentäminen yhdenmukaistaa ohjelmakoodin esitystapaa ja sen katsotaan yleensä parantavan luettavuutta.

Funktiot ovat aliohjelmia, jotka tavallisesti palauttavat jonkin arvon. Ohjelmafunktioita voidaan kutsua eri kohdissa pääohjelmaa ja muissa aliohjelmissa. Pythonissa funktiot ovat ensimmäisen luokan objekteja: niitä voidaan tallettaa muuttujiin, välittää funktioiden argumentteina ja palauttaa toisten funktioiden paluuarvoina. Funktio määritellään *def*-avainsanalla, jota seuraa funktion nimi ja sulkuihin kirjoitetut argumentit. Määrittelyrivin päättää aina kaksoispiste. [50]

```
def funktion_nimi(argumentit):
    lause1
    lause2
    ...
    return paluuarvot
```

Funktion parametrit voivat olla mitä tahansa Pythonin olioita. Parametreille voidaan antaa oletusarvoja, joita käytetään, jos argumenttien arvoja ei ole määritetty funktiota kutsuttaessa. Jos *return*-lausetta ei ole määritelty, funktio palauttaa *None*-olion.

2.5.5 Ehtorakenteet

Ehtolause määritellään seuraavasti:

```
if ehto:
   lauseet
elif ehto:
   lauseet
else:
   lauseet
```

Jos ehto on tosi, ensimmäinen ohjelmalohko suoritetaan. Muussa tapauksessa lohko ohitetaan. Tapaukset, joissa ehto on epätosi, voidaan käsitellä else-lausella. Pythonissa *else if*-rakenne lyhennetään avainsanalla *elif*. [51]

Esimerkki:

```
if a > 0.0:
   print "a on positiivinen"
elif a < 0.0:
   print "a on negatiivinen"
else:
   print "a on nolla"
```

Pythonissa on käytössä seuraavat vertailuoperaattorit:

<	pienempi kuin
>	suurempi kuin
<=	pienempi tai yhtä suuri kuin
>=	suurempi tai yhtä suuri kuin
==	yhtä suuri kuin
!=	erisuuri kuin

Taulukko 8: Vertailuoperaattorit Pythonissa.

Loogisia operaattoreita $\wedge$, $\vee$ ja $\neg$ vastaavat englannin kielen sanat *and*, *or* ja *not*. Totuusarvoja vertailtaessa ei ole välttämätöntä käyttää vertailuoperaattoreita.

Esimerkki: Totuusarvojen vertaileminen

```
if a and b:
   print "a ja b ovat tosia"
if a or not b:
   print "a on tosi tai b on epätosi"
if not a:
   print "a on epätosi"
```

2.5.6 Toistorakenteet

Toistorakenteet toteutetaan Pythonissa muista ohjelmointikielistä tutuilla *for*- ja *while*-lauseilla. *While*-lause suorittaa ohjelmalohkoa yhä uudelleen niin kauan, kun lauseessa määritelty ehto on tosi tai silmukan suoritus keskeytetään komennolla *break*. *Else*-lauseella voidaan määrittää, miten ohjelma käyttäytyy ehdon ollessa epätosi. [51]

```
while ehto:
   lauseet
else:
   lauseet
```

Pythonissa *for*-silmukan käyttö poikkeaa monista muista ohjelmointikielistä. *For*-rakenteessa iteroidaan sarjamuotoista dataa alkio kerrallaan, esimerkiksi listaa alkioittain tai merkkijonoa kirjain kerrallaan. Lukualueen läpikäynti onnistuu esimerkiksi edellä esitetyn *range*-funktion avulla.

Esimerkkejä:

Seuraava ohjelma tulostaa luvut 3, 4, 5 ja 6 omille riveilleen:

```
for x in range(3, 7):
   print x
```

Merkkijonoa iteroiva silmukka tulostaa kirjaimet P, y, t, h, o ja n omille riveilleen:

```
for c in "Python":
   print c
```

Jos halutaan luoda uusi lista jonkin säännön mukaisesti toisen listan alkioista tai muusta iteroitavasta datasta, voidaan kirjoittaa *for*-silmukka listan määrittelevien hakasul-

keiden sisälle. Esimerkiksi seuraava koodi kertoo jokaisen listan *a* alkion luvulla kaksi ja tallentaa tuloksen listaksi *b*:

```
a = [1, 2, 3, 4, 5]
b = [k*2 for k in a]
print b
```

```
[2, 4, 6, 8, 10]
```

2.5.7 Tiedostoon kirjoittaminen ja lukeminen

Kirjoitetaan ohjelma, joka kirjoittaa tiedostoon *tulokset.txt* sanan "testi". Aluksi määritellään tiedostoa vastaava muuttuja, johon myöhemmässä ohjelmakoodissa viitataan. Esimerkin tapauksessa muuttujan nimi on *tiedosto*. Kirjain *a* tarkoittaa, että tulostus lisätään (*append*) tiedoston loppuun. Jos tiedoston aikaisempi sisältö halutaan korvata, kirjoitetaan *w*-kirjain (*write*). Lopuksi kolmannella rivillä suljetaan tiedosto. Ohjelma palauttaa linkin tiedostoon *tulokset.txt*, joka on talletettu työarkin yhteyteen. Tämän tiedoston voi kopioida omalle tietokoneelleen tai sitä voi käyttää toisaalla työarkissa.

```
tiedosto = file("tulokset.txt", "a")
print >>tiedosto, "testi"
tiedosto.close()
```

tulokset.txt

Vaihtoehtoisesti tiedostoon voidaan kirjoittaa seuraavalla tavalla:

```
with file("tulokset.txt", "a") as tiedosto:
   print >>tiedosto, "testi"
```

tulokset.txt

Jälkimmäisessä esimerkissä tiedosto on avoinna lohkon suorituksen ajan, jonka jälkeen Python huolehtii automaattisesti tiedoston sulkemisesta.

Esimerkki: Matriisin tallentaminen tiedostoon ja lukeminen tiedostosta

Seuraavaksi esitettävä ohjelma tallentaa kaksiulotteisen listan muodossa olevan matriisin tiedostoon, jonka jälkeen tiedosto luetaan. Oletetaan, että listassa on yhtä monta pysty- ja vaakariviä. Iteroidaan listan rivejä yksi kerrallaan *for*-silmukassa siten, että jokaisen rivin alkiot yhdistetään toisiinsa *join*-metodin avulla ennen tiedostoon tallentamista. Vierekkäisiä alkioita erottaa välilyönti. Ohjelmakoodissa metodi *map(str, rivi)*

muuntaa alkioiden datan merkkijonoiksi.

Matriisia luettaessa iteroidaan tiedostoa riveittäin. Poistetaan ensiksi rivien lopussa esiintyvät välilyönnit metodin *rstrip* avulla. Tämän jälkeen *split*-metodi erottelee alkiot toisistaan ja palauttaa listan rivin alkioista. Merkkijonoina luetut alkiot muunnetaan tämän jälkeen metodin *map(float, alkiot)* avulla takaisin liukuluvuiksi. Lopuksi ohjelma tulostaa tiedostosta luetun matriisin.

```
a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# Kirjoitetaan matriisi a tiedostoon matriisi.txt
with file("matriisi.txt", "w") as tiedosto:
   for rivi in a:
      print >>tiedosto, " ".join(map(str, rivi))

# Luetaan matriisi tiedostosta.
matr = [map(float, rivi.rstrip().split()) for rivi in
   file("matriisi.txt")]
print matr
```

```
[[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]]
```

matriisi.txt

Lisätietoja Pythonin tiedoston käsittelyyn liittyvistä metodeista on viitteessä [52].

2.5.8 Esimerkkiohjelma: Kertoman laskeva funktio

Tarkastellaan lopuksi havainnollistavaa esimerkkiä, jossa käytetään edellä esitettyjä Pythonin ominaisuuksia: toistorakenteita, ehtolauseita ja ohjelmafunktioita. Tehtävänä on kirjoittaa ohjelma, joka laskee luvun n kertoman. Positiivisen kokonaisluvun n kertoma, merkitään $n!$, on luvun n ja kaikkien sitä pienempien positiivisten kokonaislukujen tulo, eli $1 \cdot 2 \cdot ... \cdot (n-1) \cdot n$. On sovittu, että nollan kertoma on yksi, eli $0! = 1$.

Käyttäen *for*-toistorakennetta saadaan esimerkiksi seuraavanlainen ohjelmakoodi:

```
def kertoma(n):
   k = 1
   if n > 1:
      for i in range(2, n+1):
         k = k*i
   return k
```

Jos n on nolla tai yksi, neljännellä rivillä alkavaa koodilohkoa ei suoriteta ollenkaan, vaan funktio palauttaa toisella rivillä muuttujalle k asetetun arvon yksi. Muussa tapauksessa *for*-lause käy listan $[2, 3, .., n]$ läpi ja kertoo jokaisella iteraatiokierroksella luvun k apumuuttujalla i asettaen samalla tuloksen uudeksi muuttujan k arvoksi. Näin ollen viimeisellä rivillä palautettava arvo on luvun n kertoma $1 \cdot 2 \cdot ... \cdot (n-1) \cdot n$.

Ohjelman voi toteuttaa myös *while*-toistorakenteella. Tällöin apumuuttuja i tulee esitellä ennen toistorakennetta. *While*-silmukan sisällä tulee huolehtia apumuuttujan kasvattamisesta yhdellä joka kierroksen lopussa. Merkintä *i += 1* on lyhennelmä komennolle *i = i + 1*.

```
def kertoma(n):
   k = 1
   i = 1
   if n > 1:
      while i <= n:
         k = k*i
         i += 1
   return k
```

Vielä yksi vaihtoehtoinen tapa ohjelmoida kertoma on käyttää rekursiota. Siinä ohjelmafunktio kutsuu itseään toistuvasti, jolloin funktiokutsuista muodostuu pino. Jonkin ennalta määrätyn ehdon toteutuessa rekursio päättyy ja pinosta kerätään välitulokset käänteisessä järjestyksessä. Rekursiivinen ohjelma, joka laskee kertoman, on seuraavanlainen:

```
def kertoma(n):
   if n < 2:
      return 1
   else:
      return n*kertoma(n-1)
```

Jos n on nolla tai yksi, funktio palauttaa luvun yksi ja suoritus lopetetaan. Muussa tapauksessa palautetaan n kerrottuna yhtä pienemmän luvun kertomalla. Näin jatketaan, kunnes n saavuttaa arvon yksi. Tämän jälkeen ohjelma palaa käänteisessä järjestyksessä funktiokutsut läpi ja palauttaa lopulta alkuperäisen luvun n kertoman.

Sagessa on myös valmis funktio kertoman laskemiseksi, *factorial(n)*. [53]

2.6 Animaatiot

Matematiikan opetuksessa asioita voidaan havainnollistaa erilaisten animaatioiden avulla. Voidaan esimerkiksi tutkia, miten parametrin asteittainen muuttaminen vaikuttaa kuvaajan käyttäytymiseen. Sagessa animaatioita voidaan luoda mistä tahansa graafisista objekteista koostuvasta kuvasarjasta, jonka kuvat on tallennettu listan alkioiksi. Metodi *animate(lista)* muodostaa listan perusteella *gif*-tiedostoformaatissa olevan animaation. Tiedoston voi tallentaa tietokoneen kovalevylle, upottaa internet-sivulle tai näyttää työarkissa komennolla *show(animaatio)*.

Show-metodille voidaan antaa seuraavassa taulukossa esitettyjä optiota, kun sitä käytetään animaatioiden esittämiseen. [54]

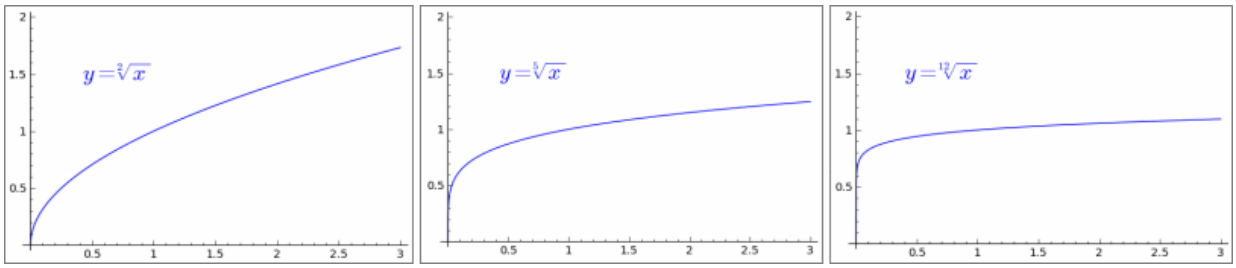
Optio	**Kuvaus**
delay	Kuvien välisen tauon pituus sadasosasekunneissa. Oletusarvo on 20, eli viisi kuvaa sekunnissa.
iterations	Esityskertojen lukumäärä. Oletusarvona 0, mikä tarkoittaa jatkuvaa toistoa.

Taulukko 9: Optiot metodille **show**(animaatio, optiot).

Kuvaajassa tapahtuvaa muutosta voidaan havainnollistaa siten, että kuvasarja koostuu useista itsenäisistä kuvaajista, joista jokainen esittää tuloksen tietyillä parametrien arvoilla. Toinen tapa on muodostaa animaatio, jossa kaikki edellä piirretyt kuvaajat kertautuvat jokaisessa seuraavassa kuvassa. Muuttuvan parametrin ei välttämättä tarvitse olla funktion argumentin arvo, vaan se voi olla myös esimerkiksi kuvaajan väri tai tarkastelualueen rajat.

Tarkastellaan seuraavaksi esimerkkiä, jossa tutkitaan, miten funktion $\sqrt[n]{x}$ kuvaaja käyttäytyy, kun parametrin n arvo kasvaa yhden yksikön välein välillä $[2, 12]$:

```
v = []
for n in range(2, 13, 1):
   kuva = plot(x^(1/n), [0, 3])
   teksti = text("$y = \sqrt[%s]{x}$" % n, (0.7, 1.5), fontsize=20)
   v.append(kuva + teksti)

animaatio = animate(v, ymax=2)
show(animaatio, delay=40)
```



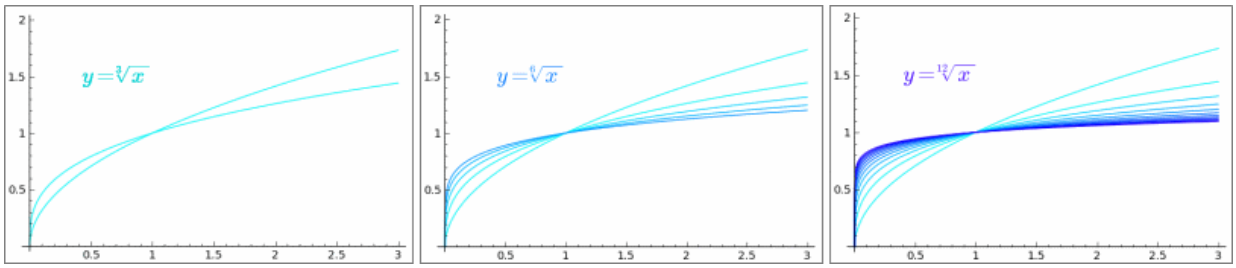
Kuva 3: Funktion $\sqrt[n]{x}$ käyttäytymistä kuvaavan animaation pysäytyskuvia.

Ohjelman aluksi alustetaan lista *v*, johon animaation kuvat kerätään. *For*-silmukassa piirretään funktion kuvaaja muuttujan *x* suhteen eri parametrin *n* arvoilla välillä $[0,3]$. Lopuksi kuva lisätään listaan *v* metodin *append* avulla. Silmukan jälkeen kuvat animoidaan siten, että jokaisessa kuvassa koordinaatisto piirretään välillä $[0,3]$x$[0,2]$. Määritetään animaation yksittäisten kuvien välille 40 millisekunnin tauko, eli esitetään 2,5 kuvaa sekunnissa. (Kuva 3.)

Tarkastellaan vielä vastaavaa animaatiota, jossa piirretyt kuvaajat toistuvat kaikissa myöhemmissä kuvissa. Asetetaan myös yksi kuvaajan optioista, viivan väri, muuttumaan animaation aikana.

```
v = []
G = Graphics()

for n in range(2, 13, 1):
   G += plot(x^(1/n), [0, 3], hue=0.45+1/50*n)
   H = text("$y=\sqrt[%s]{x}$" % n, (0.7, 1.5), hue=0.47+0.02*n,
      fontsize=20)
   v.append(G + H)

animaatio = animate(v, ymax=2)
show(animaatio, delay=40)
```

Ohjelma toimii muuten samoin kuin edellä, mutta nyt kuvia ei lisätä suoraan listaan, vaan yleiseen graafiseen objektiin *G*. Se alustetaan ohjelman alussa määrittelyllä *G = Graphics()*. Jokaisella silmukan kierroksella uusi kuvaaja piirretään entisten päälle ja saatu kuva lisätään animoitavaan listaan *v*. Viivan väri on sidottu parametriin *n* siten, että arvon *n* kasvaessa viivan sininen sävy muuttuu tummemmaksi.



Kuva 4: Pysäytyskuvia animaatiosta, jossa piirretyt juurifunktion kuvaajat kertautuvat kaikkiin seuraaviin kuviin.

Didaktisessa mielessä jälkimmäinen tapa vaikuttaa tässä tapauksessa paremmalta. Kuvaajien piirtäminen samaan koordinaatistoon havainnollistaa hyvin, miten kaikki juurifunktioiden käyrät kulkevat pisteen *(1,1)* kautta. Värien käyttö helpottaa käyrien erottamista toisistaan: mitä tummempi käyrä, sitä suurempi muuttujan n arvo on. Animaatiossa parametrin muutoksen vaikutus nähdään selkeästi: mitä suurempi juuri muuttujasta x otetaan, sitä lähempänä käyrä kulkee suoraa $y = 1$.

2.7 Interaktiiviset sovellukset

Opetuskäytön kannalta yksi Sagen mielenkiintoisimmista ominaisuuksista on mahdollisuus ohjelmoida interaktiivisia sovelluksia. Tällä tarkoitetaan ohjelman parametrien ajonaikaista manipulointia erilaisten säätimien, kytkimien, valikoiden ja tekstikenttien avulla. Vuorovaikutteisten ohjelmien avulla opiskelija voi tutkia jonkin parametrin vaikutusta esimerkiksi kuvaajan käyttäytymiseen. Parametri voi myös liittyä suoraan ohjelman toimintaan, kuten tarkasteluvälin valintaan. Seuraavissa luvuissa esitellään useita esimerkkejä interaktiivisten ominaisuuksien soveltamisesta.

Interaktiivinen funktio määritellään lisäämällä teksti *@interact* ennen funktion määrittelyriviä (niin sanottu funktiodekoraattori). Tämän jälkeen määrittelyrivillä argumenttien esittelemisen yhteydessä määritellään, millä tavalla argumenttien arvoja halutaan manipuloida, mitkä ovat sallittuja parametrien arvoja ja mitkä ovat niiden oletusarvot. Erilaisia säätimiä ja niiden ominaisuuksia on lueteltu taulukossa 10. [55]

Jokaisessa säätimessä ovat myös optiot *default*, joka määrittää muuttujan oletusarvon, ja *label*, joka määrittää käyttäjälle näkyvän muuttujan nimen. Oletuksena molemmat saavat arvon *None*, ellei toisin mainita. Säätimille on olemassa myös syntaktiset lyhennysmerkinnät, jotka on esitetty viitteessä [55].

Säädin	**Kuvaus**
input_box(type=None)	Tekstikenttä. Jos tyypin halutaan olevan syötteestä riippumatta merkkijono, valitaan *type=str*.
slider(vmin, vmax=None, step_size=1)	Liukusäädin, joka saa syötteenä listan mahdollisista arvoista tai lukuvälin. Esim. *slider(1, 3)* tuottaa säätimen, joka saa portaattomasti arvoja lukujen 1 ja 3 väliltä. Säätimessä *slider(1, 3, 1)* askeleen väli on 1. Metodi palauttaa säätimellä valitun arvon.
range_slider(vmin, vmax=None, step_size=1)	Liukusäädin lukuvälin määrittämiseen. Metodi palauttaa valitun välin päätepisteet muodossa *(a, b)*.
checkbox(default=True)	Kytkin. Palauttaa arvon *True* tai *False* sen mukaan, onko kytkin päällä vai ei.
selector(values, nrows=None, ncols=None, buttons=None)	Valikko, jos syötteenä annetaan lista arvoista ja parametrit *buttons*, *nrows* ja *ncols* saavat arvon *None* tai *False*. Jos parametrin *buttons* arvo on *True*, valinta tehdään keskenään vaihtoehtoisilla painikkeilla. Parametrit *nrows* ja *ncols* määrittävät rivien ja kolumnien lukumäärän. Metodi palauttaa valitun arvon.
color_selector(default=(0, 0, 1), widget='farbtastic', hide_box=False)	Värinvalitsin. Parametri widget voi olla myös *"jpicker"* tai *"colorpicker"*. Tekstikenttää ei näytetä, jos *hide_box=. True*. Metodi palauttaa värin koodin *rgb*-muodossa.

Taulukko 10: Interact-ympäristön erilaisia säätimiä.

2.8 Erikoiskomennot työarkissa

Lopuksi esitellään lyhyesti laskentaympäristön erikoistoimintoja, joiden avulla voidaan muotoilla laskentaympäristön dokumentteja, lisätä HTML-merkintää työarkkeihin, mitata laskutoimitusten suorittamisen kestoa ja saada lisätietoja Sagen funktioista.

HTML-soluja luodaan työarkissa klikkaamalla solun alla olevaa sinistä palkkia *shift*-painike pohjaan painettuna. Uudeksi soluksi avautuu helppokäyttöinen HTML-editori, jonka avulla työarkkeihin voidaan lisätä muun muassa tekstiä, linkkejä ja kuvia. Toiminto on erityisen hyödyllinen, kun halutaan kirjoittaa opetusmateriaalia tai matemaattisia raportteja, jotka sisältävät sekä tekstiä että laskulausekkeita. LaTeX-ladottuja kaavoja voi HTML-editorissa upottaa tekstin joukkoon käyttämällä $-merkkejä kaavan alussa ja lopussa. Kaksi dollarimerkkiä lausekkeen molemmin puolin, $$, latoo kaavan keskitetysti omalle rivilleen.

Kommenttirivi ohjelmakoodissa saadaan lisäämällä rivin alkuun #-merkki.

Tiettyjä erikoismääreitä voi kirjoittaa ohjelmasolun ensimmäisille riveille. Näitä ovat muun muassa

- **#auto**: evaluoi solun automaattisesti, kun laskentaympäristö avataan.

- **%hide**: piilottaa ohjelmakoodin. Tämä toiminto on hyödyllinen opetusmateriaalia kirjoitettaessa, kun ohjelmakoodi ei ole mielenkiintoinen osa dokumenttia ja sen piilottaminen selkeyttää ulkoasua.

- **%time**: laskee ohjelmakoodin suorittamiseen kuluvan ajan. Suorituksen jälkeen Sage tulostaa kaksi lukua. *CPU time* tarkoittaa aikaa, jonka palvelimen prosessori käytti laskentaan ja *Wall time* kuvaa prosessin suorittamiseen kulunutta kokonaisaikaa.

Lisätietoa ohjelmafunktioista saa kirjoittamalla metodin nimen jälkeen kysymysmerkin (?). Tällöin Sage tulostaa metodin kuvauksen, mahdolliset parametrit ja esimerkkejä metodin käytöstä. Kaksi peräkkäistä kysymysmerkkiä (??) kirjoittamalla tulostuu lisäksi metodin lähdekoodi.

3 Funktiot ja yhtälöt

Funktiot ja yhtälöt ovat keskeisessä asemassa lukion opetussuunnitelmassa. Tässä luvussa tarkastellaan, miten matemaattinen funktio määritellään Sagessa ja miten sen arvoja voidaan määrittää muuttujan eri arvoilla. Lisäksi käsitellään yhtälöiden ja epäyhtälöiden määrittelemistä sekä yhtälössä esiintyvän tuntemattoman muuttujan ratkaisemista. Myös epäyhtälöitä voidaan monissa tapauksissa ratkaista symbolisesti. Opetuskäyttöön liittyen tässä luvussa esitellään funktioiden ja toisen asteen yhtälön ratkaisujen havainnollistamiseen soveltuvia interaktiivisia ohjelmia.

3.1 Funktion määritteleminen

Sagessa matemaattinen funktio voidaan määritellä usealla eri tavalla. [56] Yksi tapa on määritellä funktio ohjelmafunktiona:

```
def f(x):
    return x^2+1
```

Tällä tavalla määriteltyjä funktioita voidaan piirtää, mutta niihin ei voida kohdistaa erilaisia matemaattisia operaatioita, kuten derivointia tai integrointia. Usein parempi tapa on luoda niin sanottu *kutsuttava symbolinen lauseke* (*callable symbolic expression*). Sen määrittely muistuttaa perinteistä matemaattista notaatiota:

```
f(x) = x^2 + 1
print f
print f(x)
```

```
x |-> x^2 + 1
x^2 + 1
```

Symbolisia lausekkeita voidaan derivoida ja integroita symbolisesti sekä määritellä toisten funktioiden avulla. Niitä voidaan myös piirtää Sagen *plot*-metodin avulla ja tutkia tarkemmin erilaisilla menetelmillä, esimerkiksi etsiä funktion nollakohtia tai ääriarvoja.

Funktioon sijoittaminen tapahtuu kirjoittamalla funktion argumentiksi sijoitettava luku tai toinen muuttuja:

```
f(x) = x^2 + 1
print f(3)
n = var("n")
print f(n)
```

```
10
n^2+1
```

Esimerkki: Määritellään funktio h funktioiden $f(x) = x^2$ ja $g(x) = x - 1$ yhdistettynä funktiona:

```
f(x) = x^2
g(x) = x - 1
h(x) = f(g(x))
print "h(x) = ", h(x)
print f(g(3))
```

```
h(x) = (x - 1)^2
4
```

Funktio voidaan määritellä myös edellisessä luvussa esitetyllä tavalla lausekkeena, esimerkiksi f = x^2 + 1. Näin määritelty funktio on myös symbolinen lauseke, mutta se eroaa toiminnaltaan edellä kuvatusta. Ongelmana on, ettei ohjelma voi tunnistaa, minkä muuttujan suhteen funktio on määritelty. Erityisesti tämä näkyy usean muuttujan funktioissa: sijoittamalla luvut suoraan argumenttien paikalle ohjelma antaa varoituksen eikä välttämättä tee sijoitusta odotetulla tavalla.

Esimerkki: Sijoittaminen funktioihin f ja g tulkitaan eri tavalla, vaikka määrittelyt ovat lähes samanlaiset:

```
f(x, k) = x - k
print "f(0, 1) = ", f(0, 1)
g = x - k
print "g(0, 1) = ", g(0, 1)
```

```
f(0, 1) = -1
__main__:6: DeprecationWarning: Substitution using function-call
syntax and unnamed arguments is deprecated...; you can use named
arguments instead, like EXPR(x=..., y=...)
g(0, 1) = 1
```

Nimeämällä muuttujat ongelmilta vältytään:

```
print g(x=0, k=1)
```

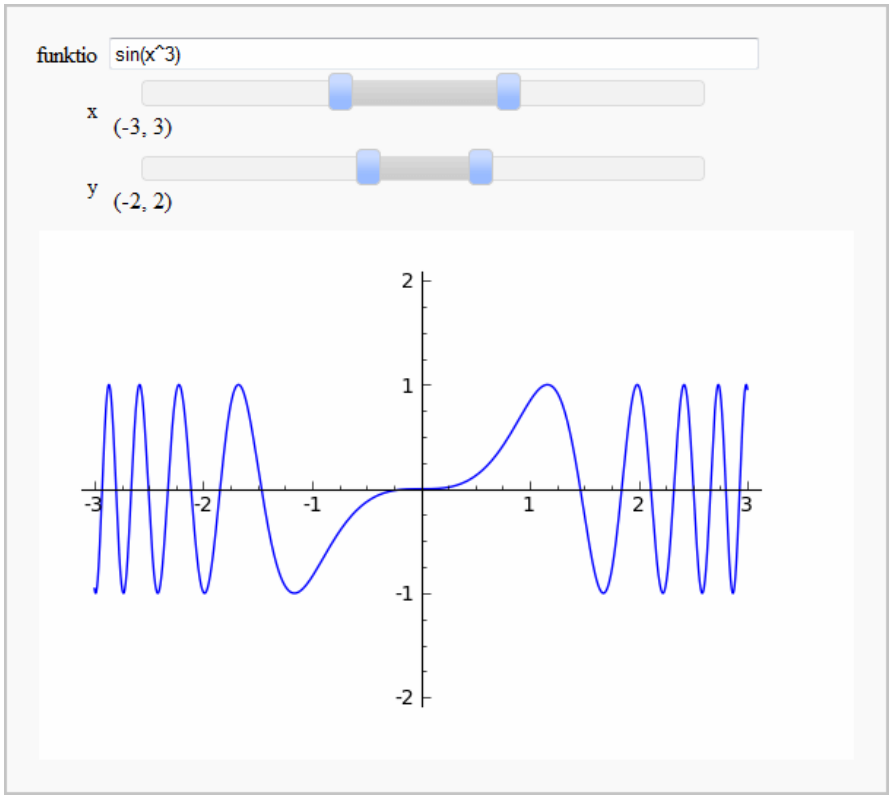
```
-1
```

3.2 Funktiot kuvaajissa ja interaktiivisissa sovelluksissa

3.2.1 Yksinkertainen piirto-ohjelma

Lukion matematiikan opetustilanteissa halutaan usein piirtää erilaisia kuvaajia koordinaatistoon. Tätä tarkoitusta varten voidaan kirjoittaa yksinkertainen ohjelma, joka piirtää syötteenä annetun funktion. Käyttäjä voi myös valita sopivan piirtovälin.

```
@interact
def Piirto(funktio = input_box(default = sin(x^3)),
         x_vali = range_slider(-10,10,1, default=(-5,5), label="x"),
         y_vali = range_slider(-10,10,1, default=(-5,5), label="y")):
   f(x) = funktio
   show(plot(f(x), x_vali, ymin=y_vali[0], ymax=y_vali[1]))
```



Kuva 5: Yksinkertainen piirto-ohjelma.

3.2.2 Sovellus: Kaksi funktiota samassa koordinaatistossa

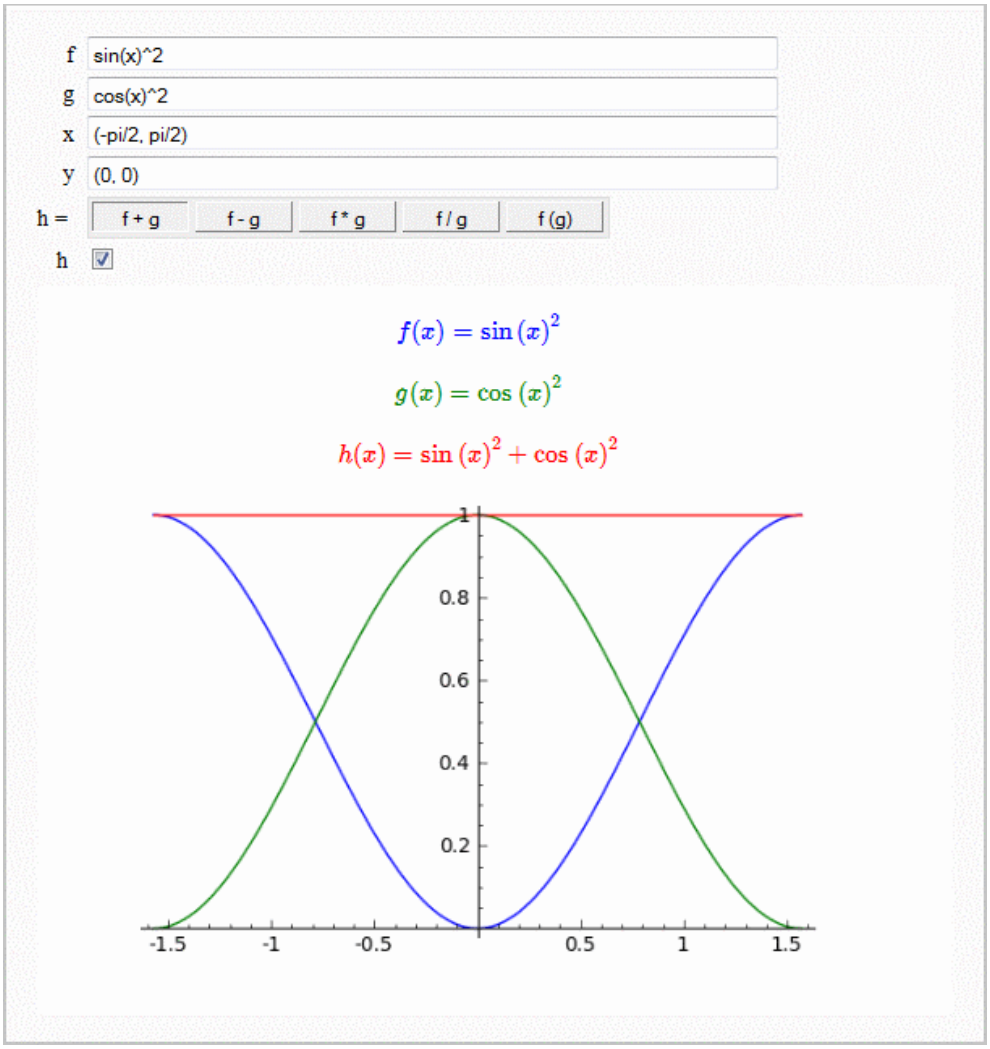
Laajennetaan edellistä piirto-ohjelmaa siten, että se piirtää kahden funktion käyrät sekä kolmannen käyrän, joka on käyttäjän valinnasta riippuen kahden annetun funktion summa, erotus, tulo, osamäärä tai yhdistetty funktio. Ohjelmalla voidaan tutkia näin muodostetun funktion käyttäytymistä ja riippuvuutta kahdesta annetusta funktiosta.

Ohjelma 1: Kaksi funktiota samassa koordinaatistossa

```
@interact
def _(f = input_box(default = x+1), g = input_box(default = 2*x),
      x_vali = input_box(default=(-2,2), label="x"),
      y_vali = input_box(default = (0,0), label = "y"),
      asetus = selector(["f + g", "f - g", "f * g", "f / g", "f (g)"], nrows = 1,
          width = 10, label = "h = "),
      kytkin = checkbox(default = True, label = "h ")):
    f(x) = f; g(x) = g
    # Funktioiden f ja g kuvaajat.
    f_kuva = plot(f(x), x_vali, color = "blue")
    g_kuva = plot(g(x), x_vali, color = "green")
    kuva = f_kuva + g_kuva

    # Funktion h määrittely ja funktioiden f, g ja h tulostaminen.
    html("<center> $\color{Blue}{f(x) = %s}$ </center>"%latex(f(x)))
    html("<center> $\color{Green}{g(x) = %s}$ </center>"%latex(g(x)))
    if kytkin:
        if asetus == "f + g":
            h(x) = f + g
        elif asetus == "f - g":
            h(x) = f-g
        elif asetus == "f * g":
            h(x) = f*g
        elif asetus == "f / g":
            h(x) = f/g
        else:
            h(x) = f(g)
        h_kuva = plot(h(x), x_vali, color="red")
        kuva += h_kuva
        html("<center> $\color{Red}{h(x) = %s}$ </center>"%latex(h(x)))

    # Yhdistetyn kuvan esittäminen.
    if y_vali == (0,0):
        show(kuva, xmin=x_vali[0], xmax=x_vali[1])
    else:
        show(kuva, xmin=x_vali[0], xmax=x_vali[1], ymin=y_vali[0], ymax=y_vali[1])
```

Tämä ohjelma eroaa edellisestä interaktiivisesta sovelluksesta siinä, että piirtoväli kirjoitetaan sulkeisiin sen sijaan, että se valittaisiin erilaisten säätimien avulla. Näin määriteltynä tarkasteluvälin valinta on vapaampaa, sillä päätearvot voivat olla myös murto- tai desimaalilukuja. Toisaalta säätimien käyttö voi olla intuitiivisempaa.

Oletuksena y-akselilla on arvo (0,0), jolloin ohjelma valitsee piirtoväliksi pystysuunnassa kolmen funktion saaman pienimmän ja suurimman arvon annetulla välillä. Ohjelma määrittää uuden funktion h valitulla tavalla ja tulostaa kaikkien kolmen funktion lausekkeen. Funktio h voidaan myös piilottaa näkyvistä kytkimellä. Funktioiden lausekkeiden värit vastaavat koordinaatistoon piirrettyjen käyrien värejä, mikä helpottaa funktioiden kuvaajien tunnistamista ja erottamista toisistaan.



Kuva 6: Kaksi funktiota samassa koordinaatistossa.

3.2.3 Paloittain määritellyt funktiot

Funktio voidaan määritellä paloittain metodilla *Piecewise*. [57] Syötteenä metodille annetaan lista, jonka alkiot ovat funktion ja sen avoimen määrittelyvälin sisältäviä listoja. Paloittain määriteltyjä funktioita voidaan piirtää sekä muun muassa derivoida ja integroida.

Esimerkki:

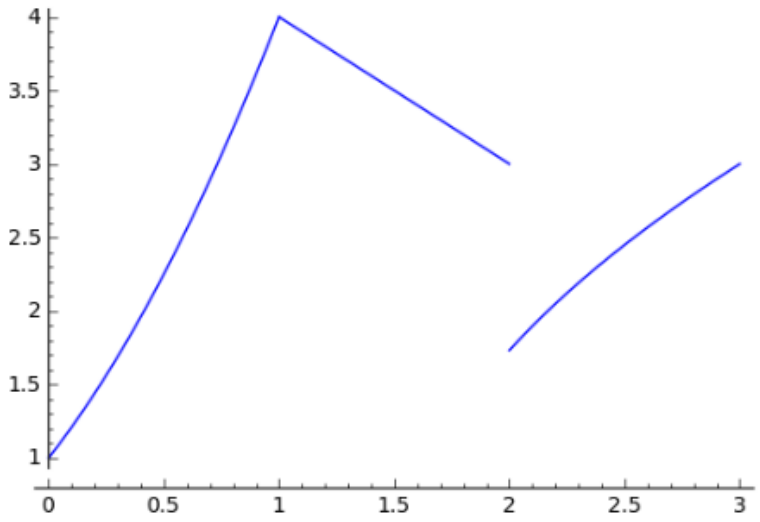
Tarkastellaan esimerkkinä paloittain määritellyn funktion piirtämistä, kun funktiona on

$$f(x) = \begin{cases} (x+1)^2, & \text{kun } 0 < x < 1 \\ -x+5, & \text{kun } 1 < x < 2 \\ \sqrt{6x-9}, & \text{kun } 2 < x < 3 \end{cases}$$

Kirjoitetaan osaväleillä määritellyt lausekkeet funktioiksi *f1*, *f2* ja *f3*. Tämän jälkeen luodaan paloittain määritelty funktio *Piecewise*-metodin avulla:

```
f1(x) = (x+1)^2
f2(x) = -x+5
f3(x) = sqrt(6*x-9)
f = Piecewise([[(0, 1), f1], [(1, 2), f2], [(2, 3), f3]])
plot(f)
```

Ohjelma piirtää seuraavan kuvan:



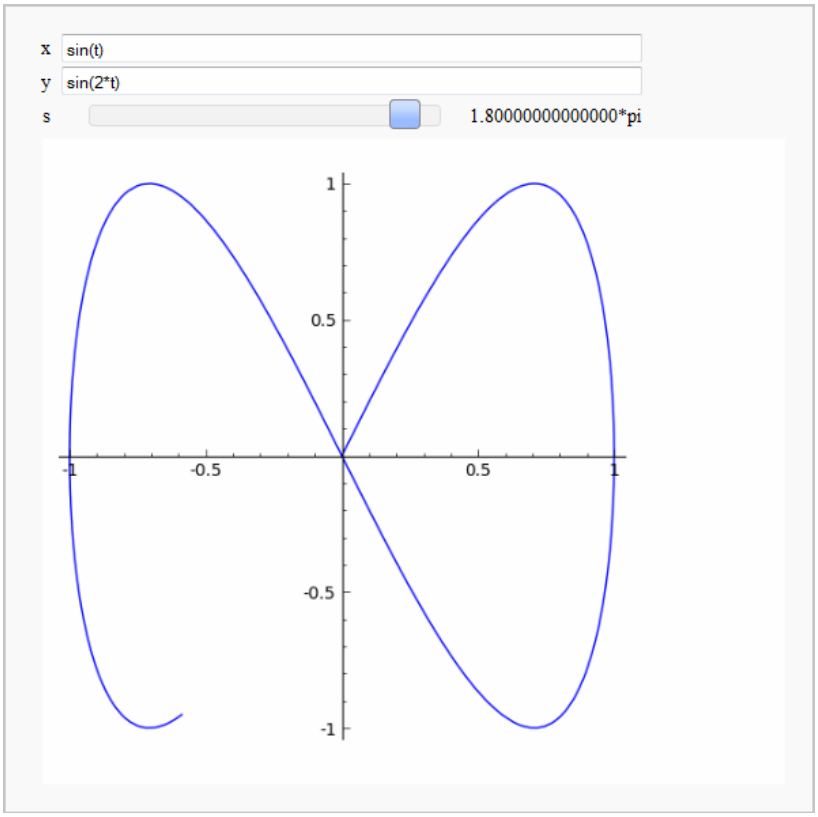
Kuva 7: Paloittain määritelty funktio $f(x)$.

3.3 Parametristen käyrien piirtäminen

Parametristen käyrien kuvaajia piirretään komennolla *parametric_plot(x(t), y(t), (t, a, b))*, missä muuttuja t saa arvoja välillä (a, b). [58] Sagen interaktiivisten toimintojen avulla käyrän piirtämistä voidaan havainnollistaa tekemällä käyräparametrin t piirtovälistä säädettävä.

Esimerkki: Piirretään parametrisesti määritelty pistejoukko $(sin(t), sin(2t))$, missä t saa arvoja käyttäjän määrittelemältä väliltä $(0, s)$. Käyräparametrista t riippuvat funktiot $x(t)$ ja $y(t)$ ovat ohjelmassa vapaasti valittavissa.

```
@interact
def _(x = input_box(default = sin(t)),
     y = input_box(default = sin(2*t)),
     s = slider(0.1*pi, 2*pi, 0.1*pi)):
   x(t) = x; y(t) = y
   show(parametric_plot((x(t), y(t)), (t, 0, s)), aspect_ratio=1,
      xmin=-1, xmax=1, ymin=-1, ymax=1)
```



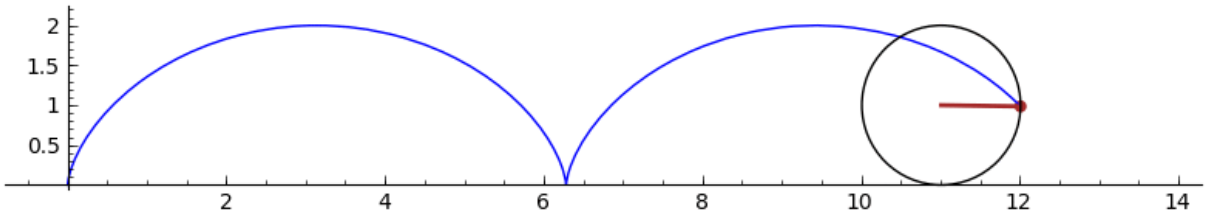
Kuva 8: Parametrisen funktion $(sin(t), sin(2t))$ kuvaaja, kun $t \in [0; 1, 8\pi]$.

3.3.1 Animaatio: sykloidi

Parametristen käyrien sovelluksena esitetään animaatio, joka kuvaa koordinaattiakselilla vierivän ympyrän kehäpisteen piirtämää käyrää, sykloidia. Tälle käyrälle voidaan muodostaa parametriesitys $(t - sin(t), 1 - cos(t))$. Ohjelmassa käytetään graafisia objekteja: ympyrää, pistettä ja viivaa. Graafisiin objekteihin palataan tarkemmin seuraavassa luvussa. Metodi *srange* toimii *range*-metodin tavoin, mutta sallii myös muiden kuin kokonaislukujen käyttämisen metodin parametreina.

Ohjelma 2: Animaatio: sykloidi

```
askel = 0.2
v = []
t = var("t")

# Animaation yksittäisten kuvien piirtäminen.
for s in srange(0.01, 4.0*pi, askel):
   sykloidi = parametric_plot((t-sin(t), 1-cos(t)), (t, 0, s))
   kehapiste = (s-sin(s), 1-cos(s))
   ympyra = circle((s, 1), 1)
   piste = point(kehapiste, pointsize=25, rgbcolor="brown")
   viiva = line([(s, 1), kehapiste], thickness=2, rgbcolor="brown")
   v.append(sykloidi + ympyra + piste + viiva)

# Animointi ja kuvasarjan esittäminen.
a = animate(v, figsize=[10,2], aspect_ratio=1, xmin=-0.5, xmax=14, ymin=0, ymax=2.2)
show(a, delay=10)
```



Kuva 9: Sykloidin käyrän piirtymistä havainnollistavan animaation pysäytyskuva.

3.4 Yhtälöiden ja epäyhtälöiden määritteleminen

Yhtälö määritellään Sagessa kirjoittamalla yhtä suurten lausekkeiden välille vertailuoperaattori ==. Yhtälöitä voidaan laskea yhteen ja vähentää toisistaan sekä kertoa ja jakaa puolittain. Yhtälöihin kohdistetut laskutoimitukset suoritetaan niin ikään puolittain. Metodi *left()* palauttaa yhtälön vasemman puolen, *right()* vastaavasti oikean puolen. Yhtälöitä voidaan tallentaa symbolisiksi lausekkeiksi kuten funktioitakin.

Esimerkki:

```
a = 3*x^2 == 3*x+3
b = x^2 == 3
print a-b
print a/b
print a/3
print (a - (3*x + 3))/3
print b.right()
```

```
2*x^2 == 3*x
3 == x + 1
x^2 == x + 1
x^2 - x - 1 == 0
3
```

Jos halutaan selvittää, onko jokin yhtälö symbolisesti tosi, voidaan käyttää totuusarvon palauttavaa metodia *bool*. Sen käytössä on kuitenkin huomioitava, että arvo *False* voi tarkoittaa paitsi sitä, että yhtälö on epätosi, myös, ettei Sagen avulla pystytty selvittämään asiaa.

Esimerkki:

```
d = (x + 1)^2 == x^2 - 2*x + 1
print bool(d)
e = (x - 1)^2 == x^2 - 2*x + 1
print bool(e)
```

```
False
True
```

Epäyhtälöitä määritellään samaan tapaan kuin yhtälöitäkin vertailuoperaattoreilla <=, <, >= ja >. Myös epäyhtälöitä voidaan laskea yhteen, vähentää toisistaan, kertoa ja jakaa toisilla epäyhtälöillä, mutta lopputulos ei välttämättä ole algebrallisesti järkevä. On esimerkiksi huomattava, ettei Sage automaattisesti käännä epäyhtälön suuntaa, jos tulon tekijä on negatiivinen. Epäyhtälöillä laskutoimituksia tehtäessä lausekkeiden välisten relaatioiden tulee olla samanlaisia, muussa tapauksessa ohjelma antaa virheilmoituksen.

Esimerkki:

```
a = 3*x^2 <= 3*x+3
b = x^2 <= 3
print a/b
print a*(-1)
print a+b
c = x > 2
print a+c
```

```
3 <= x + 1
-3*x^2 <= -3*x - 3
4*x^2 <= 3*x + 6
Traceback (click to the left of this block for traceback)
...
TypeError: incompatible relations
```

Yhtälöön voidaan sijoittaa metodilla *substitute()*. Muuttujan paikalle voidaan sijoittaa luku, toinen muuttuja tai lauseke.

Esimerkki:

```
x, n, z = var("x, n, z")
a = x^(1/n) == 4
print a.substitute(n=2)
print a.substitute(x=16, n=2)
print a.substitute(x=2*z, n=1/2)
```

```
sqrt(x) == 4
4 == 4
4*z^2 == 4
```

3.5 Yhtälöiden, yhtälöryhmien ja epäyhtälöiden ratkaiseminen

Sage pystyy ratkaisemaan useimpia tavallisia yhtälöitä, yhtälöryhmiä ja epäyhtälöitä. Metodi *solve(yhtälö, muuttuja)* palauttaa listan yhtälön ratkaisuista.

Esimerkki:

```
x, y = var("x, y")
solve(x^2 == 4, x)
```

$$[x = (-2),\ x = 2]$$

Yhtälöryhmiä voidaan ratkaista antamalla yhtälöt metodille listan alkioina.

Esimerkki:

```
solve([x+y == 3, 2*x+y == 6], x, y)
```

```
[[x == 3, y == 0]]
```

Lisäämällä parametrin *solution_dict=True* metodi palauttaa hakemiston, joka sisältää yhtälön ratkaisut. Tämä on monesti hyödyllistä interaktiivisten sovellusten toteutuksessa, sillä hakemistoja on käytännöllistä iteroida läpi.

Esimerkki: Seuraava koodi ratkaisee yhtälön $5x^4 - 23x^2 + 19 = 0$ juuret. Neljännen asteen yhtälön monimutkaiset, useita neliöjuuria sisältävät juuret voidaan muuntaa likiarvoiksi iteroimalla lista läpi samalla kohdistaen *n*-metodi jokaiseen yksittäiseen juuren arvoon.

```
ratkaisut = solve(5*x^4 - 23*x^2 + 19 == 0 , x, solution_dict=True)
show(ratkaisut)
```

$$\left[\left\{x : -\tfrac{1}{10}\sqrt{\sqrt{149} + 23}\sqrt{10}\right\}, \left\{x : \tfrac{1}{10}\sqrt{\sqrt{149} + 23}\sqrt{10}\right\},\right.$$

$$\left.\left\{x : -\tfrac{1}{10}\sqrt{-\sqrt{149} + 23}\sqrt{10}\right\}, \left\{x : \tfrac{1}{10}\sqrt{-\sqrt{149} + 23}\sqrt{10}\right\}\right]$$

```
[n(i[x]) for i in ratkaisut]
```

```
[-1.87634100354210, 1.87634100354210,
-1.03891502945459, 1.03891502945459]
```

Yhtälön ratkaisussa voi esiintyä niin sanottuja vapaita muuttujia, jotka voivat saada minkä tahansa arvon. Vapaiden muuttujien nimi koostuu kirjaimesta ja numerosta. Muuttujan numerolla ei ole merkitystä ratkaisun kannalta, sillä se liittyy ratkaisualgoritmin toimintaan.

Esimerkki: Seuraavassa yhtälöryhmän ratkaisussa esiintyy vapaa muuttuja $r2$.

```
solve([x+y == 3, 2*x+2*y == 6], x, y)
```

```
[[x == -r2 + 3, y == r2]]
```

Myös useimpia epäyhtälöitä voidaan ratkaista samalla tavalla kuin yhtälöitä.

```
solve(x^2 > 8, x)
```

$$\left[\left[x < -2\sqrt{2}\right], \left[x > 2\sqrt{2}\right]\right]$$

Lisätietoja Sagen yhtälöistä ja epäyhtälöistä sekä niiden ratkaisemisesta on viitteessä [59].

3.5.1 Sovellus: toisen asteen yhtälön ratkaisukaava

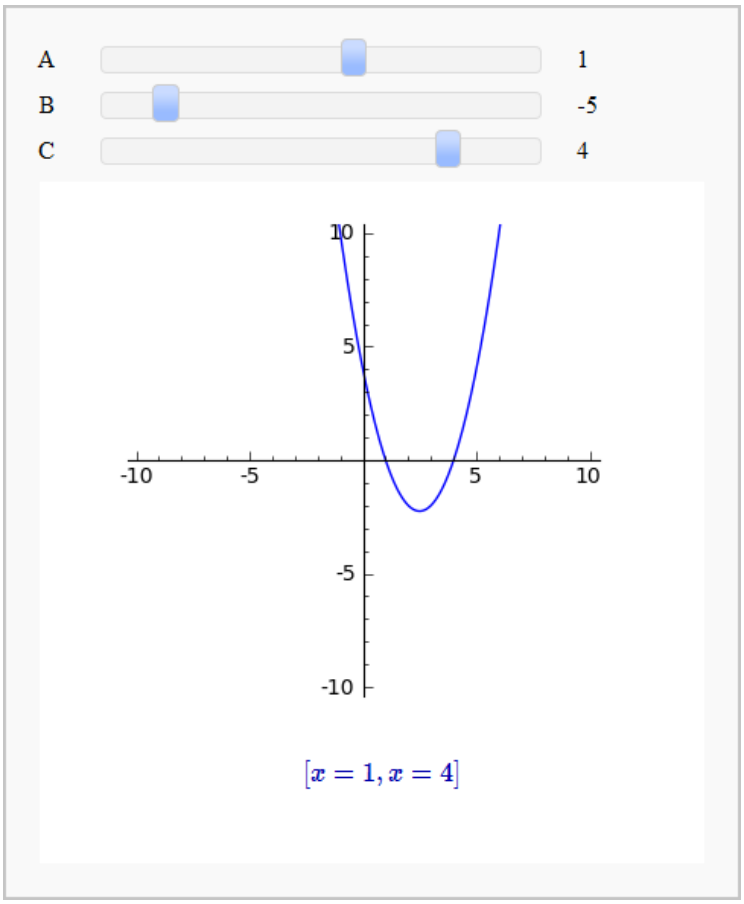
Selvitetään Sagen avulla toisen asteen yhtälön $Ax^2 + Bx + C = 0$ yleinen ratkaisukaava ja esitetään se matemaattisella notaatiolla kirjoitettuna:

```
x, A, B, C = var("x, A, B, C")
show(solve(A*x^2 + B*x + C == 0, x))
```

$$\left[x = \frac{-\left(B+\sqrt{-4\,AC+B^2}\right)}{2\,A}, \; x = \frac{-\left(B-\sqrt{-4\,AC+B^2}\right)}{2\,A}\right]$$

Kirjoitetaan seuraavaksi opetusohjelma, joka havainnollistaa toisen asteen yhtälön ratkaisuja kuvaajan avulla. Yksinkertaisimmillaan ohjelma piirtää toisen asteen polynomin kuvaajan käyttäjän antamien kertoimien perusteella ja esittää käyrän nollakohdat:

```
@interact
def _(A=slider(-7, 7, 1, 1),
      B=slider(-7, 7, 1, 1),
      C=slider(-7, 7, 1, -2)):
   show(plot(A*x^2 + B*x + C, (-10, 10), ymin=-10, ymax=10),
      aspect_ratio=1, figsize=4)
   show(solve(A*x^2 + B*x + C == 0, x))
```



Kuva 10: Yksinkertainen toisen asteen yhtälön ratkaisuja havainnollistava ohjelma.

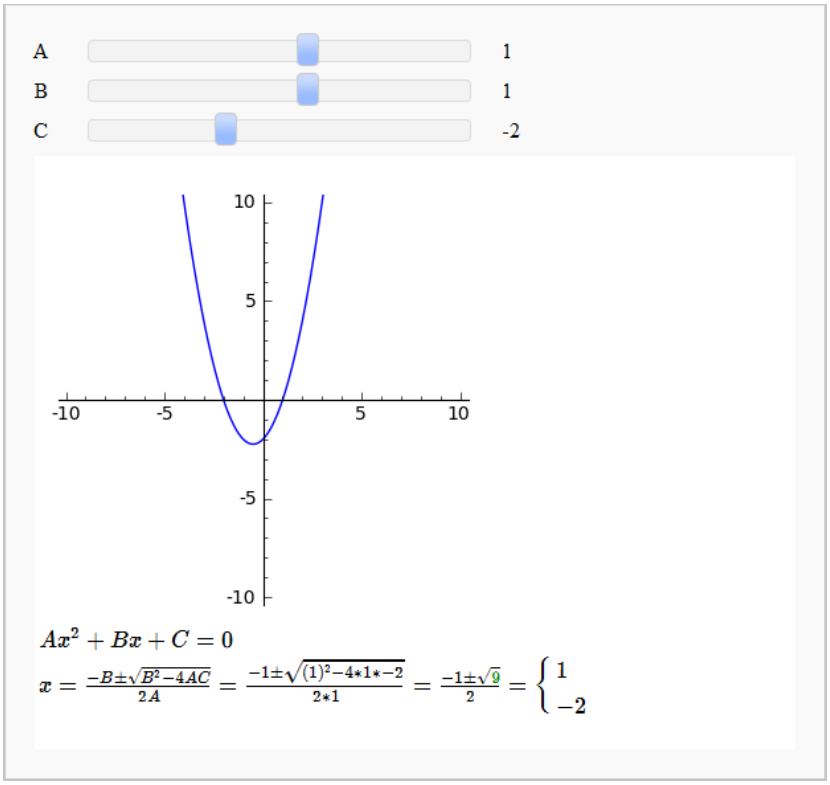
Ohjelma ei kuitenkaan selvitä tapaa, jolla nollakohdat lasketaan toisen asteen yhtälön ratkaisukaavan avulla. Ei ole myöskään ilmeistä, missä tapauksissa toisen asteen yhtälöllä on kaksi ratkaisua, milloin yksi tai ei ollenkaan ratkaisua. Lisäksi *solve*-metodi palauttaa kompleksilukuratkaisut silloin, kun yhtälöllä ei ole reaalisia ratkaisuja, mikä saattaa vaikuttaa hämmentävältä lukion ensimmäisillä matematiikan kursseilla, joilla polynomifunktiot esitellään.

Seuraavassa esitetään ohjelma, joka huomioi nämä kysymykset. Se esittää toisen asteen yhtälön ratkaisun vaiheittain ratkaisukaavan avulla. Ohjelmassa ratkaisukaavan diskri-

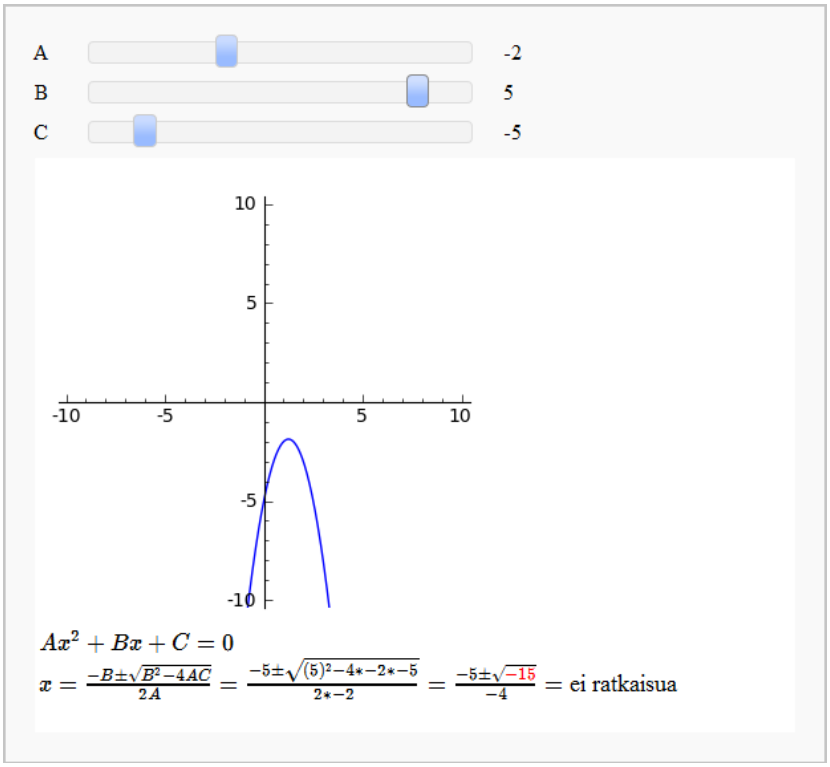
minantin väri riippuu siitä, onko sen arvo positiivinen, nolla tai negatiivinen luku. Vastaavasti yhtälöllä on joko kaksi, yksi tai nolla ratkaisua.

Ohjelma 3: Toisen asteen yhtälön ratkaiseminen

```
@interact
def _(A = slider(-7, 7, 1, 1),
      B = slider(-7, 7, 1, 1),
      C = slider(-7, 7, 1, -2)):

   # Toisen asteen polynomin piirtäminen.
   show(plot(A*x^2 + B*x + C, (-10,10), ymin=-10, ymax=10), aspect_ratio=1,
       figsize=4)

   # Diskriminantin laskeminen.
   d = B^2 - 4*A*C

   # Ratkaisujen määrittäminen diskriminantin arvon perusteella.
   if d < 0:
      color = "Red"
      ratkaisu = "\\text{ei ratkaisua}"
   elif d == 0:
      color = "Blue"
      ratkaisu = -B/(2*A)
   else:
      color = "Green"
      a = (-B + sqrt(B^2 - 4*A*C))/(2*A)
      b = (-B - sqrt(B^2 - 4*A*C))/(2*A)
      ratkaisu = "\\begin{cases}%s\\\\%s\\end{cases}"%(latex(a),latex(b))

   # Toisen asteen yhtälön ratkaisukaava.
   dis1="(%s)^2-4*%s*%s"%(B, A, C)
   dis2="\color{%s}{%s}"%(color, d)

   html("$Ax^2 + Bx + C = 0$")
   html("$x = \\frac{-B\pm\sqrt{B^2-4AC}}{2A} = \\frac{-%s\pm\sqrt{%s}}{2*%s} =
       \\frac{-%s\pm\sqrt{%s}}{%s} = %s$"%(B, dis1, A, B, dis2, 2*A, ratkaisu))
```



Kuva 11: Toisen asteen yhtälön ratkaiseminen, kun yhtälöllä on kaksi ratkaisua.



Kuva 12: Toisen asteen yhtälö, jolla ei ole reaaliratkaisuja.

3.6 Tehtäviä

Tehtävä 3-1: *(yo-K10p:1)* Suorita seuraavat tehtävät analyyttisesti ja tarkista saamasi vastaukset Sagen avulla:

a) Ratkaise yhtälö $7x^7 + 6x^6 = 0$.

b) Sievennä lauseke $(\sqrt{a} + 1)^2 - a - 1$.

c) Millä x:n arvoilla pätee $\frac{3}{3-2x} < 0$?

Tehtävä 3-2: *(yo-K08l:13)* Piirrä funktioiden $x^3 - 2x$ ja $-x + 3$ kuvaajat samaan koordinaatistoon. Etsi yhtälön $x^3 - 2x = -x + 3$ juuri haarukoimalla. Anna vastaus yhden desimaalin tarkkuudella. Suorita tehtävä Sagessa tarkastelemalla funktioiden erotusfunktiota ohjelman 1 (luvussa 3.2.2) avulla.

Tehtävä 3-3: Tarkastellaan toisen asteen polynomia $f(x) = Ax^2 + Bx + C$ sovelluksen 3 (luvussa 3.5.1) avulla.

a) Tutki, miten parametrien A, B ja C muuttaminen vaikuttaa toisen asteen polynomin kuvaajaan. Etsi sellaiset parametrien arvot, että yhtälöllä on nolla, yksi tai kaksi ratkaisua.

b) Ratkaise sovelluksen avulla yhtälöt

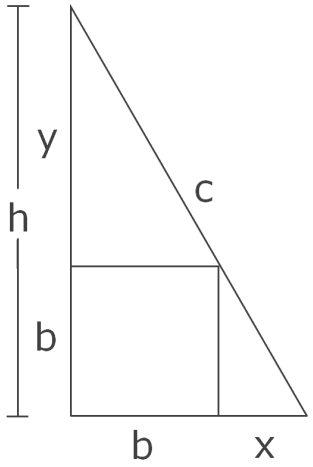
   i) $2x^2 - 2x - 4 = 0$,
   ii) $-x^2 + 2x - 1 = 0$,
   iii) $-x^2 + 5 = 0$,
   iv) $2x^2 - 6x + 5 = 0$.

c) Etsi ohjelman parametreja muuttamalla sellainen polynomi, joka avautuu ylöspäin ja jonka huippu on suoralla $x = 2$. Etsi tämän jälkeen edellisen polynomin yhtä parametria muuttamalla sellainen toisen asteen polynomi, joka aukeaa ylöspäin ja jonka huippu on pisteessä $(2, -4)$.

Tehtävä 3-4: Kun tunnetaan kaksi suoran pistettä (x_1, y_1) ja (x_2, y_2), saadaan suoran yhtälöksi $y - y_1 = \frac{y_1 - y_2}{x_1 - x_2}(x - x_1)$. Ratkaise Sagen avulla symbolisesti kahden suoran leikkauspiste, kun toinen suorista kulkee edellä mainittujen pisteiden kautta ja toinen pisteiden (x_a, y_a) ja (x_b, y_b) kautta. Milloin suorilla ei ole leikkauspistettä?

Tehtävä 3-5: Tikkaat, joiden pituus on c metriä, lepäävät seinää vasten siten, että ne juuri ja juuri koskettavat seinässä kiinni olevan laatikon kulmaa. Kuution muotoisen laatikon sivujen pituus on b metriä.

Olkoon tikkaiden pituus $c = 5\ m$ ja laatikon sivujen pituus $b = 1\ m$. Laske Sagea apuna käyttäen, mikä on suurin korkeus h, jolla tikkaiden yläpää koskettaa seinää.



Kuva 13: Tehtävän 5 asetelma.

4 Geometria ja trigonometria

Geometria on matematiikan osa-alueista graafisin. Perinteisesti geometrisia ongelmia on havainnollistettu oppikirjoissa ja oppitunneilla staattisilla kuvilla, joissa yleisistä tapauksista on esitetty jokin erityistapaus. Tietotekniikan avulla on mahdollista luoda dynaamisia kuvia, joiden parametreja voidaan muuttaa reaaliaikaisesti. Tällaisia parametreja voivat olla esimerkiksi kolmion kulmien suuruus tai geometrisen kappaleen mittasuhteet. Ne voivat liittyä myös itse graafisen objektin tarkasteluun, esimerkiksi kolmiulotteisissa kuvissa havainnoitsijan katselukulmaan ja etäisyyteen tarkasteltavasta kappaleesta. Tässä luvussa perehdytään Sagen graafisten objektien käsittelyyn ja trigonometrisia funktioita sisältävien lausekkeiden symboliseen laskentaan.

4.1 Trigonometrisia funktioita

Sagessa on tuettuna kaikki trigonometriset perusfunktiot: sini, kosini, tangentti, kotangentti, sekantti ja kosekantti. Lisäksi on määritelty metodit sinin, kosinin ja tangentin käänteisfunktioille (arkusfunktiot) sekä hyperbolisille funktioille ja niiden käänteisfunktioille (areafunktiot). [38]

Perusfunktiot	**Käänteisarvot**	**Arkusfunktiot**	**Hyperboliset funktiot**	**Areafunktiot**
sin(x)	csc(x)	asin(x)	sinh(x)	asinh(x)
cos(x)	sec(x)	acos(x)	cosh(x)	acosh(x)
tan(x)	cot(x)	atan(x)	tanh(x)	atanh(x)

Taulukko 11: Trigonometrisia funktioita vastaavia metodeja Sagessa.

Trigonometriset funktiot käyttävät syötteinään pääasiallisesti radiaaneja. Lukion matematiikan kursseilla on usein tarpeellista käyttää radiaanien ohella asteita. Muunnokset radiaaneista asteiksi ja päinvastoin voidaan tehdä muun muassa Pythonin funktioiden *math.degrees(x)* ja *math.radians(x)* avulla. Pitkät komennot on käytännöllistä korvata lyhyemmillä nimillä esimerkiksi seuraavasti:

```
aste = math.degrees
rad = math.radians
print aste(3/2*pi)
print rad(270)
```

```
270.0
4.7123889038
```

Trigonometrisia funktioita sisältäviä lausekkeita voidaan käsitellä symbolisesti. Sage tunnistaa useita trigonometrisia identiteettejä ja pyrkii yksinkertaistamaan lausekkeita aina kun se on mahdollista.

Esimerkki:

```
f = tan(x)*cos(x)
print f, " = ", f.full_simplify()
g = sin(x)^2+cos(x)
print g, " = ", g.full_simplify()
h = sin(2*x)/(2*sin(x))
print h, " = ", h.full_simplify()
```

```
cos(x)*tan(x) = sin(x)
sin(x)^2 + cos(x)^2 = 1
1/2*sin(2*x)/sin(x) = cos(x)
```

Full_simplify-metodi ei aina pysty sieventämään trigonometrisia funktioita sisältäviä lausekkeita. Tehokkaampi menetelmä on Maximan metodi *trigreduce*, jota Sagessa kutsutaan esimerkiksi seuraavasti:

```
q1, q2 = var("q1, q2")
f = cos(q1)*cos(q2) - sin(q1)*sin(q2)
maxima.trigreduce(f)
```

```
cos(q1 + q2)
```

4.1.1 Trigonometristen yhtälöiden ratkaiseminen

Trigonometrisia yhtälöitä ratkaistaessa tulee huomioida, ettei *solve*-metodi välttämättä palauta kaikkia yhtälön ratkaisuja. [60] Lisäämällä *solven* lisämääreisiin parametrin *to_poly_solve="force"* ohjelma ilmoittaa jaksollisen funktion kaikki ratkaisut.

Esimerkki: Ratkaistaan yhtälö $sin(2x - \frac{\pi}{6}) = \frac{1}{2}$.

```
print solve(sin(2*x - pi/6) == 1/2, x)
print solve(sin(2*x - pi/6) == 1/2, x, to_poly_solve="force")
```

```
[x == 1/6*pi]
[x == 1/6*pi + pi*z2, x == 1/2*pi + pi*z0]
```

Yhtälön ratkaisuksi saadaan siis *to_poly_solve="force"* -parametria käyttämällä

$$x = \frac{1}{6}\pi + n \cdot \pi \quad \text{tai} \quad x = \frac{1}{2}\pi + n \cdot \pi,$$

missä $n \in \mathbb{Z}$.

4.1.2 Esimerkki: Kolmion kulmien suuruus

Kosinilauseen mukaan kolmion sivujen ja kulmien suuruuden välillä on riippuvuussuhde

$$c^2 = a^2 + b^2 - 2ab\cos\alpha,$$

missä a ja b ovat kulman α viereisiä sivuja ja c on kulman vastainen sivu. Kun kolmion sivujen pituudet tunnetaan, voidaan kulman α kosinin suuruus ratkaista yllä olevasta yhtälöstä:

```
a, b, c, alpha = var("a, b, c, alpha")
show(solve(c^2 == a^2 + b^2 - 2*a*b*cos(alpha), cos(alpha)))
```

$$\left[\cos(\alpha) = \frac{(a^2+b^2-c^2)}{2ab}\right]$$

Kirjoitetaan tuloksen perusteella ohjelma, joka saa syötteenä kulman viereiset sivut ja vastaisen sivun. Ohjelma palauttaa kulman asteluvun.

```
def kulma(a, b, c):
   return math.degrees(acos((b^2 + c^2 - a^2)/(2*b*c)))
```

4.2 Tasokuvioita

Sagessa voidaan piirtää geometrisia tasokuvioita opetuksen tukena käytettäväksi. Seuraavassa esitellään näistä tavallisimmat: jana, piste ja ympyrä sekä niihin liittyviä esimerkkejä.

4.2.1 Jana

Jana piirretään metodilla *line*. Argumentteina se saa listan pisteistä (x, y), joiden kautta jana piirretään. Jos listassa on enemmän kuin kaksi pistettä, piirretään järjestyksessä

jokaisen pisteen kautta kulkeva murtoviiva. Janalle voidaan antaa samoja ulkonäköön vaikuttavia optioita kuin kuvaajan piirrossa käyrälle. [61]

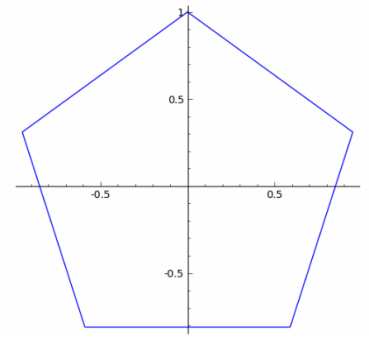
Esimerkki: Monikulmion piirtäminen

Seuraava funktio palauttaa annettujen pisteiden kautta kulkevan suljetun murtoviivan, eli monikulmion graafisena objektina:

```
def monikulmio(pisteet):
   pisteet.append(pisteet[0])
   return line(pisteet)
```

Näin määritelty monikulmio voidaan piirtää *show*-metodilla:

```
viisikulmio = [(0, 1), (0.95, 0.31), (0.59, -0.81), (-0.59, -0.81),
   (-0.95, 0.31)]
show(monikulmio(viisikulmio), aspect_ratio=1)
```



Kuva 14: *Line*-metodin avulla piirretty viisikulmio.

4.2.2 Piste

Metodi *point* saa argumenttina listan piirrettävistä pisteistä. Pisteen lisäparametreja ovat muun muassa läpinäkyvyys (*alpha*), pisteen koko (*pointsize*) ja väri (*rgbcolor*). [62]

4.2.3 Ympyrä

Circle((x,y), r) on metodi, jonka avulla voidaan piirtää ympyröitä. Argumentteina metodi saa ympyrän keskipisteen (x, y) ja säteen r. Ympyrälle voidaan antaa samoja optioita kuin viivallekin, minkä lisäksi sen sisäosan voi värittää valitsemalla option *fill* totuusarvoksi *True*. Oletuksena *fill* saa totuusarvon *False*. Ympyrän täyttövärin voi määrittää optiolla *facecolor*. [63]

4.3 Sinin, kosinin ja tangentin määrittäminen yksikköympyrästä

Toteutetaan tasokuvioiden sovelluksena ohjelma, joka havainnollistaa trigonometristen funktioiden sinin, kosinin ja tangentin määrittämistä yksikköympyrästä.

Ohjelma näyttää kaksi kuvaa, joista vasemmanpuoleinen esittää yksikköympyrää ja oikeanpuoleisessa on koordinaatisto, johon trigonometrisen funktion kuvaaja piirretään. Ohjelma on interaktiivinen: säätimellä valitaan radiaaniluku, jonka funktio saa argumenttinaan. Tutkittava funktio valitaan pudotusvalikosta.

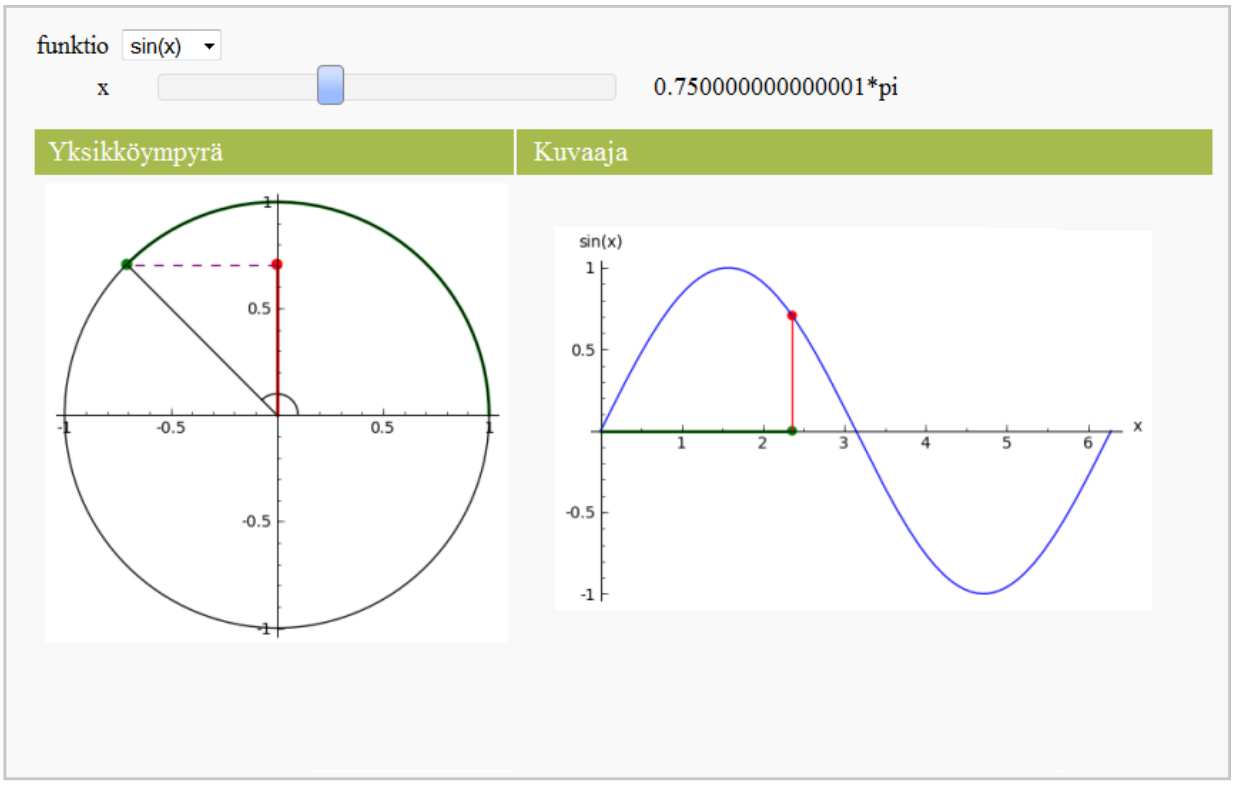
Ohjelma näyttää kulmaa vastaavan pisteen yksikköympyrän kehällä ja havainnollistaa eri väristen viivojen avulla, miten esimerkiksi sinifunktio määritellään pisteen y-koordinaattina. Funktion kuvaajaan on piirretty vastaavat viivat samoilla väreillä kuin yksikköympyrässä. Tämä havainnollistaa hyvin trigonometristen funktioiden määrittelyn luonnetta yksikköympyrän ja funktion kuvaajan välillä.

Vähäisillä muutoksilla ohjelma voidaan muuntaa käyttämään radiaanien sijaan asteita. Tämän luvun myöhemmissä sovelluksissa esitetään, miten ohjelmalle annettava syöte voi olla asteluku, vaikka ohjelma laskeekin sisäisesti radiaaneilla.

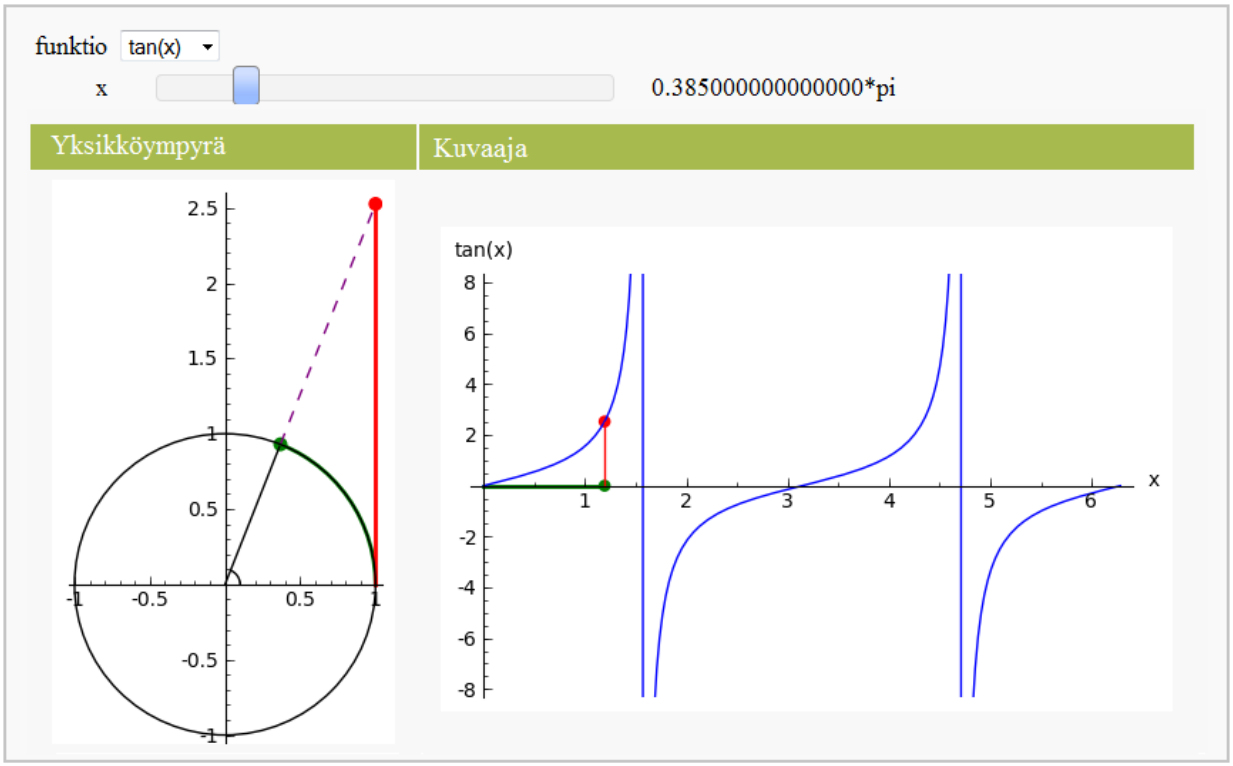
Ohjelma 4: Sinin, kosinin ja tangentin määrittäminen yksikköympyrästä

```
t = var('t')
@interact
def yksikkoympyra(funktio=selector([(0, sin(x)), (1, cos(x)), (2, tan(x))]),
                  x=slider(0, 2*pi, 0.005*pi, 0)):
    xy = (cos(x), sin(x))

    # Yksikköympyrään liittyvä grafiikka.
    ympyra = circle((0,0), 1, figsize=[5,5], aspect_ratio=1)
    ympyra_viiva = line([(0,0), (xy[0],xy[1])], rgbcolor="black")
    ympyra_piste = point((xy[0],xy[1]), pointsize=40, rgbcolor="green")
    ympyra_kulma_keha = parametric_plot((cos(t),sin(t)), (t, 0, x+0.001),
        color="green", thickness=2)
    ympyra_kulma_keskus = parametric_plot((0.1*cos(t), 0.1*sin(t)), (t, 0, x+0.001),
        color="black")

    # Kuvaajaan liittyvä grafiikka.
    kuvaaja_viiva = line([(0,0), (x,0)], rgbcolor="green", thickness=2)
    kuvaaja_piste = point((x,0), pointsize=30, rgbcolor="green")
```

```
# Sini.
if funktio == 0:
   ympyra_funktio_piste = point((0,xy[1]), pointsize=40, rgbcolor="red")
   ympyra_funktio_viiva_akseli = line([(0,0), (0,xy[1])], rgbcolor="red",
       thickness=2) ympyra_funktio_viiva_keha = line([(0,xy[1]), (xy[0],xy[1])],
       rgbcolor="purple", linestyle='--')
   kuvaaja_funktio = plot(sin(t), t, 0, 2*pi, axes_labels=("x", "sin(x)"))
   kuvaaja_funktio_piste = point((x, sin(x)), pointsize=30, rgbcolor="red")
   kuvaaja_funktio_viiva = line([(x,0), (x,sin(x))], rgbcolor="red")
# Kosini.
elif(funktio==1):
   ympyra_funktio_piste = point((xy[0],0), pointsize=40, rgbcolor="red")
   ympyra_funktio_viiva_akseli = line([(0,0), (xy[0],0)], rgbcolor="red",
       thickness=2) ympyra_funktio_viiva_keha = line([(xy[0],0), (xy[0],xy[1])],
       rgbcolor="purple", linestyle='--')
   kuvaaja_funktio = plot(cos(t), t, 0, 2*pi, axes_labels=("x", "cos(x)"))
   kuvaaja_funktio_piste = point((x,cos(x)), pointsize=30, rgbcolor="red")
   kuvaaja_funktio_viiva = line([(x,0), (x,cos(x))], rgbcolor="red")
# Tangentti.
else:
   ympyra_funktio_piste = point((1,tan(x)), pointsize=40, rgbcolor="red")
   ympyra_funktio_viiva_akseli = line([(1,0), (1,tan(x))], rgbcolor="red",
       thickness=2)
   ympyra_funktio_viiva_keha = line([(xy[0],xy[1]), (1,tan(x))],
       rgbcolor="purple", linestyle='--')
   kuvaaja_funktio = plot(tan(t), t, 0, 2*pi, ymin=-8, ymax=8, axes_labels=("x",
       "tan(x)"))
   kuvaaja_funktio_piste = point((x,tan(x)), pointsize=30, rgbcolor="red")
   kuvaaja_funktio_viiva = line([(x,0), (x,tan(x))], rgbcolor="red")

# Yksikköympyrään liittyvän grafiikan yhdistäminen yhdeksi kuvaksi.
yksikkoympyra = ympyra + ympyra_piste + ympyra_viiva + ympyra_kulma_keha +
    ympyra_kulma_keskus + ympyra_funktio_piste + ympyra_funktio_viiva_akseli +
    ympyra_funktio_viiva_keha

# Funktion kuvaajaan liittyvän grafiikan yhdistäminen.
kuvaaja = kuvaaja_viiva + kuvaaja_piste + kuvaaja_funktio +
    kuvaaja_funktio_piste + kuvaaja_funktio_viiva

 # Esitetään yksikköympyrä ja kuvaaja rinnakkain.
html.table([["$\\text{Yksikköympyrä}$","$\\text{Kuvaaja}"], [yksikkoympyra,
    kuvaaja]], header=True)
```



Kuva 15: Sinifunktion määrittäminen yksikköympyrästä.



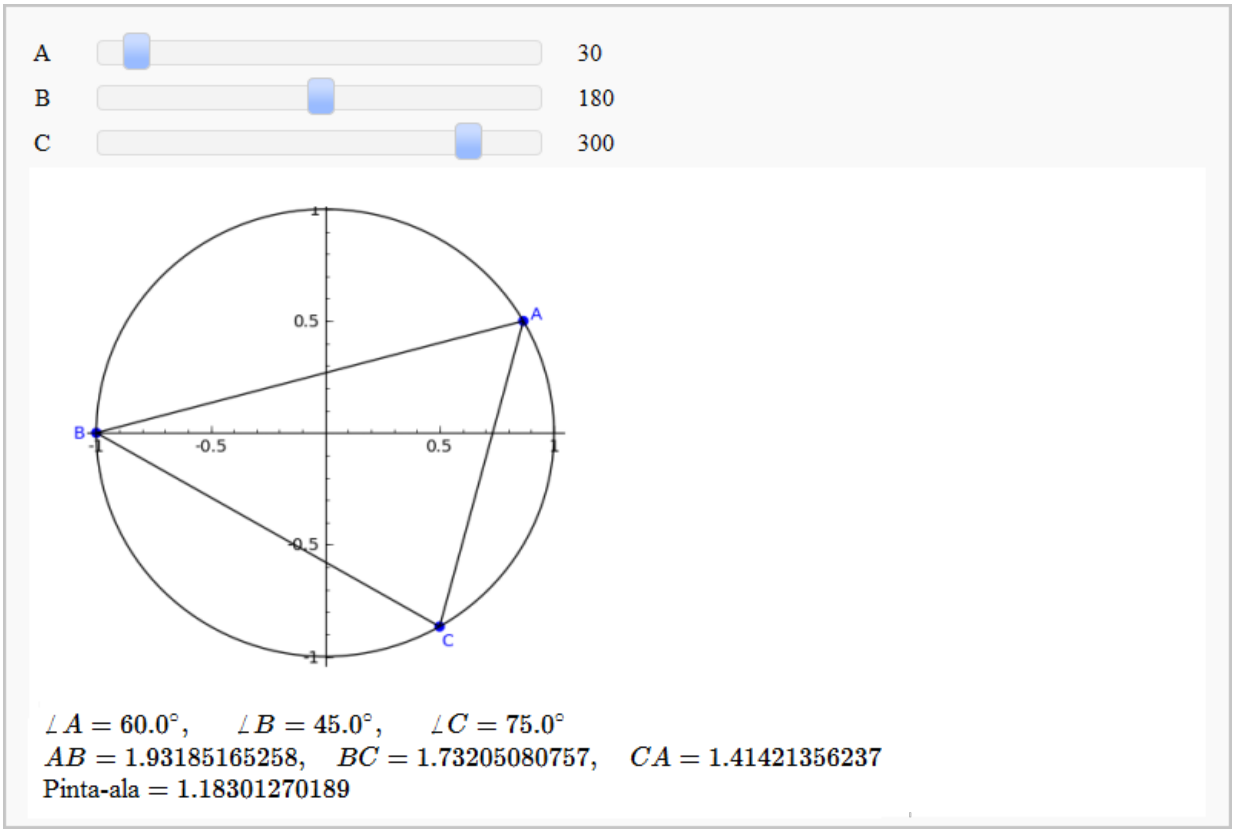
Kuva 16: Tangenttifunktion arvon määrittäminen.

4.4 Kolmion trigonometrisia ominaisuuksia

Vuorovaikutteiset ohjelmat soveltuvat erityisen hyvin kolmion trigonometristen ominaisuuksien ja niiden välisten riippuvuussuhteiden tarkasteluun. Kirjoitetaan ohjelma, jolla voidaan piirtää mielivaltaisia kolmioita ja määrittää piirretyn kolmion kulmien suuruudet asteissa, sivujen pituudet sekä pinta-ala. Sovellusta laajennetaan myöhemmin kolmion merkillisten pisteiden tutkimiseen.

Sovelluksessa kolmion kulmat sijaitsevat yksikköympyrän kehällä. Käyttäjä voi vapaasti valita kolmion mittasuhteet kolmen parametrin avulla, jotka ovat ympyrän kehällä olevien kolmion kulmien sijainnit asteissa. Näin määriteltynä kolmio voidaan valita yhdenmuotoiseksi minkä tahansa mielivaltaisen kolmion kanssa.

Kolmion sivujen pituudet lasketaan Pythagoraan lauseen avulla, kun kulmien koordinaatit tunnetaan. Edellä on määritelty funktio *kulma*, joka palauttaa kolmion kulman suuruuden, kun kulman viereiset sivut ja vastainen sivu tunnetaan. Kolmion pinta-ala voidaan laskea edellisen tuloksen avulla kaavasta $A = \frac{1}{2}ab\sin(\alpha)$, missä a ja b ovat kulman α viereisiä sivuja. Käytännöllisyyden vuoksi ohjelma laskee sisäisesti radiaaneilla, mutta esittää vastaukset käyttäjälle asteissa.



Kuva 17: Kolmion trigonometristen ominaisuuksien määrittäminen.

Ohjelma 5: Kolmion trigonometrisia ominaisuuksia

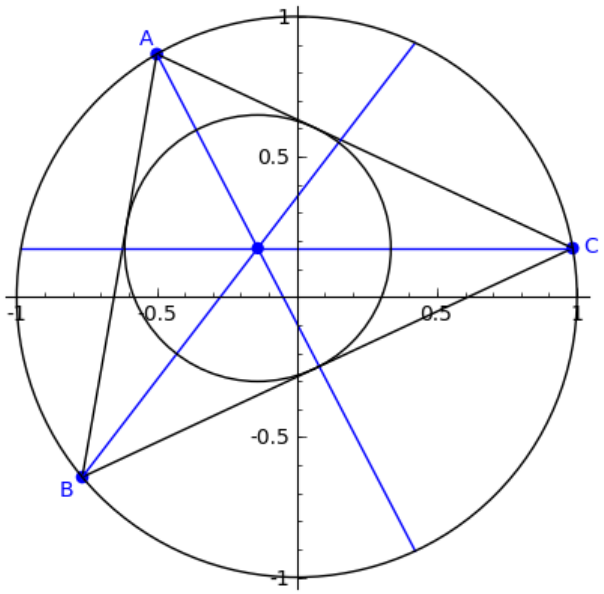
```
# Palauttaa pisteiden (x1,y1) ja (x2,y2) välisen etäisyyden.
def etaisyys((x1,y1), (x2,y2)):
   return sqrt((x2-x1)^2 + (y2-y1)^2)

# Palauttaa kolmion kulman suuruuden (radiaaneissa), kun kulman viereiset sivut a
    ja b sekä vastainen sivu c tunnetaan.
def kulma(a,b,c):
   return acos((b^2 + c^2 - a^2)/(2*b*c))

# Palauttaa kolmion pinta-alan, kun kulman alpha (radiaaneissa) viereiset sivut a
    ja b tunnetaan.
def ala(alpha,a,b):
   return 1/2*a*b*sin(alpha)

xy = [0]*3

@interact
def kolmio(a0 = slider(0, 360, 1, 30, label="A"),
           a1 = slider(0, 360, 1, 180, label="B"),
           a2 = slider(0, 360, 1, 300, label="C")):

   # Kulmien koordinaatit:
   a = [math.radians(a0), math.radians(a1), math.radians(a2)]
   for i in range(3):
      xy[i] = (cos(a[i]), sin(a[i]))

   # Kolmion kulmia (a,b,c) vastaavien sivujen pituudet (bc, ca, ab):
   al = [etaisyys(xy[1], xy[2]), etaisyys(xy[2], xy[0]), etaisyys(xy[0], xy[1])]

   # Kolmion kulmien (a,b,c) suuruus radiaaneissa:
   ak = [kulma(al[0], al[1], al[2]), kulma(al[1], al[2], al[0]), kulma(al[2],
       al[0], al[1])]

   # Kolmion pinta-ala:
   A = ala(ak[0], al[1], al[2])

   # Yksikköympyrän piirtäminen.
   ympyra = circle((0,0), 1, aspect_ratio=1)

   # Kolmion piirtäminen.
   kolmio = line([xy[0], xy[1], xy[2], xy[0]], rgbcolor="black")
   kolmio_pisteet = point(xy, pointsize=30)
```

```
# Kulmien tunnukset (piirretään hieman etäälle pisteistä).
a_tunnus = text("A", (xy[0][0]*1.07, xy[0][1]*1.07))
b_tunnus = text("B", (xy[1][0]*1.07, xy[1][1]*1.07))
c_tunnus = text("C", (xy[2][0]*1.07, xy[2][1]*1.07))
tunnukset = a_tunnus + b_tunnus + c_tunnus

show(ympyra + kolmio + kolmio_pisteet + tunnukset, figsize=[5,5], xmin=-1,
    xmax=1, ymin=-1, ymax=1)
html("$\\angle A = %s^\\circ,$ $\\angle B = %s^\circ,$ $\\angle C =
    %s^\circ$"%(math.degrees(ak[0]), math.degrees(ak[1]), math.degrees(ak[2])))
html("$AB = %s,$ $BC = %s,$ $CA = %s$"%(al[2], al[0], al[1]))
html("$\\text{Pinta-ala} = %s$"%A)
```

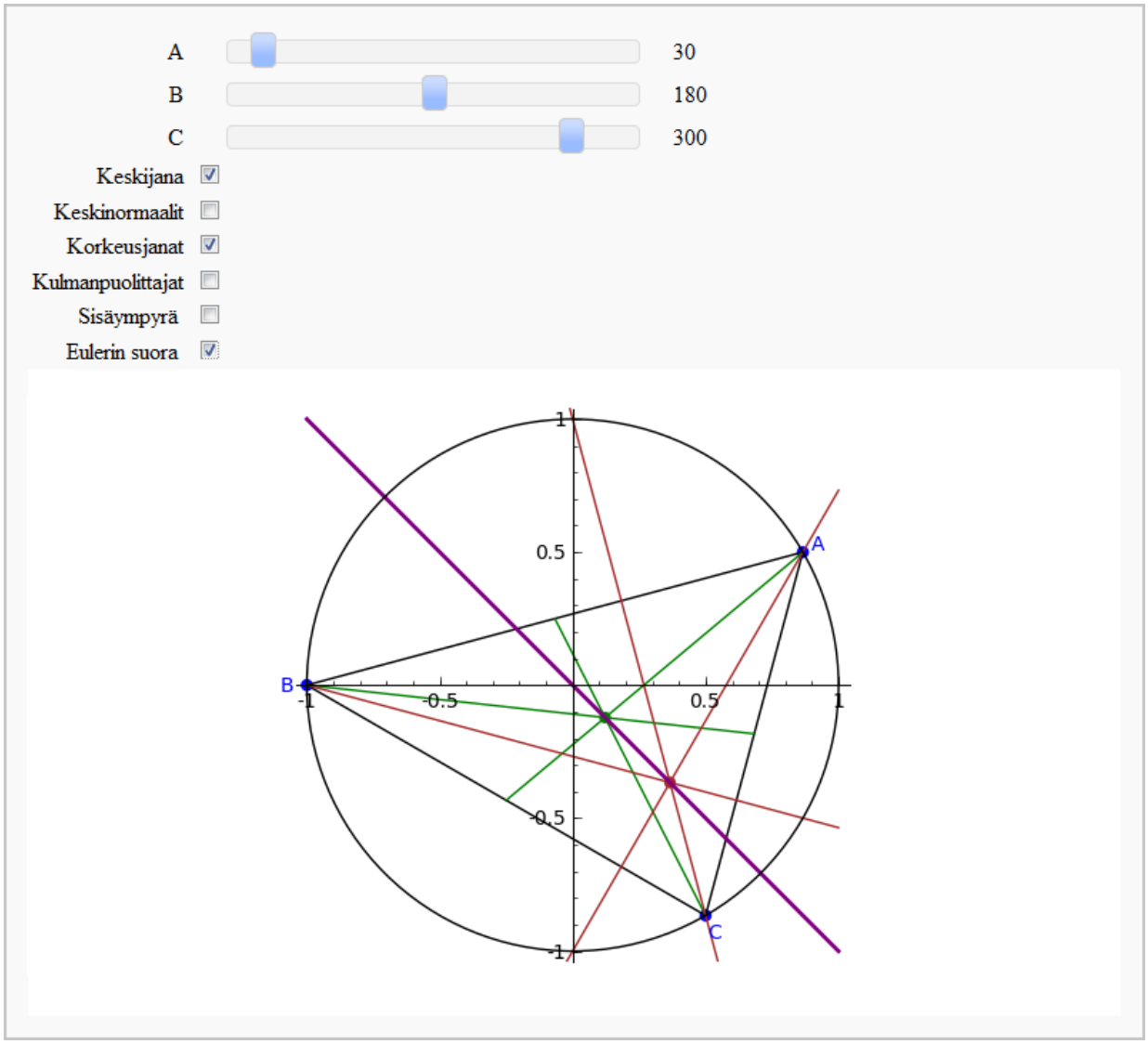
4.5 Kolmion merkilliset pisteet

Laajennetaan edellistä ohjelmaa siten, että tarkastelun kohteena ovat kolmion merkilliset pisteet. Näillä tarkoitetaan pisteitä, joissa kolmion kulmanpuolittajat, sivujen keskinormaalit, korkeusjanat ja keskijanat leikkaavat. Piirretään myös suora, joka kulkee edellä mainittujen pisteiden kautta kulmanpuolittajien leikkauspistettä lukuun ottamatta, eli niin sanottu Eulerin suora. Lisäksi piirretään kolmion sisälle suurin mahdollinen ympyrä. Sen keskipiste on sama kuin kulmanpuolittajien leikkauspiste.

Sovelluksen ohjelmakoodi on liitteenä (A.1).



Kuva 18: Kolmion kulmanpuolittajat ja sisäympyrä.



Kuva 19: Kolmion keski- ja korkeusjanat sekä Eulerin suora.

4.6 Kolmiulotteisia kappaleita

Sagen avulla voidaan mallintaa useita kolmiulotteisia objekteja, joista tässä esitellään lyhyesti pallo ja platoniset kappaleet tetraedri, kuutio, oktaedri, ikosaedri ja dodekaedri. [64] Lisäksi pisteellä, janalla ja muilla tasokuvioilla on kolmiulotteiset vastineensa, kuten metodit *point3d* ja *line3d*. [65] Kahden muuttujan funktioita ja muun muassa parametrisia käyriä voi piirtää kolmessa ulottuvuudessa metodeilla *plot3d* ja *parametric_plot3d*.

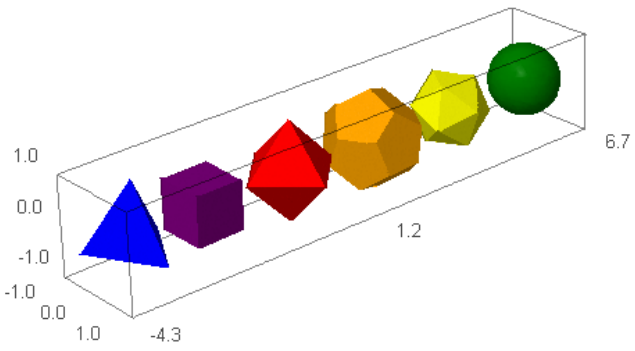
Seuraavaan taulukkoon on koottu kolmiulotteisia kappaleita määritteleviä metodeja ja niiden optioita:

Kappale	**Metodi**	**Optiot**
pallo	*sphere(center=(0,0,0), size=1, optiot)*	samat kuin ympyrällä
tetraedri	*tetrahedron(center=(0,0,0), size=1, optiot)*	color, opacity
kuutio	*cube(center=(0,0,0), size=1, optiot)*	color, opacity, frame_thickness frame_color
oktaedri	*octahedron(center=(0,0,0), size=1, optiot)*	color, opacity
ikosaedri	*icosahedron(center=(0,0,0), size=1, optiot)*	color, opacity
dodekaedri	*dodecahedron(center=(0,0,0), size=1, optiot)*	color, opacity

Taulukko 12: Platonisia kappaleita määritteleviä metodeja ja niiden optioita.

Esimerkki: Kolmiulotteisten kappaleiden piirtäminen

```
G = tetrahedron((0,-3.5,0), color="blue") + cube((0,-2,0),
   color="purple") + octahedron(color="red") + dodecahedron((0,2,0),
   color="orange") + icosahedron(center=(0,4,0), color="yellow") +
   sphere((0,6,0), size=0.7, color="green")
G.show(aspect_ratio=[1, 1, 1])
```



Kolmiulotteiset kappaleet ja kuvaajat esitetään *Jmol*-nimisessä *Java*-sovelluksessa, jossa kappaleita voidaan reaaliaikaisesti tarkastella eri suunnista ja etäisyyksiltä. Jmol tukee myös kuvien esittämistä stereokuvina. Tällaiset kuvat luovat vaikutelman kolmiulotteisuudesta, kun niitä katsotaan esimerkiksi puna-viher-laseilla.

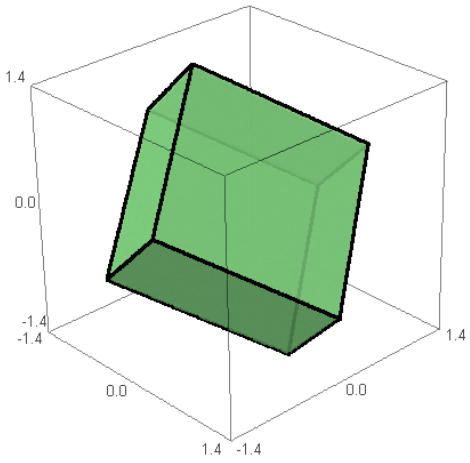
Edellä luetellut kappaleet ovat kaikki Sagen graafisia 3D-objekteja, joille on määritelty yhteisiä metodeja. Seuraavassa taulukossa on esitelty näistä metodeista kolme tärkeintä, joiden avulla objekteja voidaan siirtää, pyörittää ja skaalata [66]:

Metodi	**Kuvaus**
translate(x, y, z)	Siirtää objektia vektorin (x, y, z) verran
rotate(v, theta)	Pyörittää objektia vektorin v ympäri *theta* radiaania
rotateX(theta), rotateY(theta), rotateZ(theta)	Pyörittää objektia x-, y- tai z-akselin ympäri *theta* radiaania
scale([x, y, z]), scale(k)	Skaalaa objektia x-, y- ja z-akselin suuntaisesti kertoimilla x, y ja z. Jos argumenttina on vain yksi arvo, skaalataan koko objektia kertoimella k.

Taulukko 13: Kolmiulotteisten kappaleiden yhteisiä metodeja.

Esimerkki: Kappaleen skaalaus ja pyörittäminen

```
G = cube((0,0,0),1, rgbcolor="lightgreen", opacity=0.8,
   frame_color="black", frame_thickness=3)
G = G.scale([1,2,2]).rotateX(pi/4).rotateY(pi/4)
show(G, aspect_ratio=[1,1,1])
```



4.7 Esimerkki: Geometrisen ongelman numeerinen tarkastelu

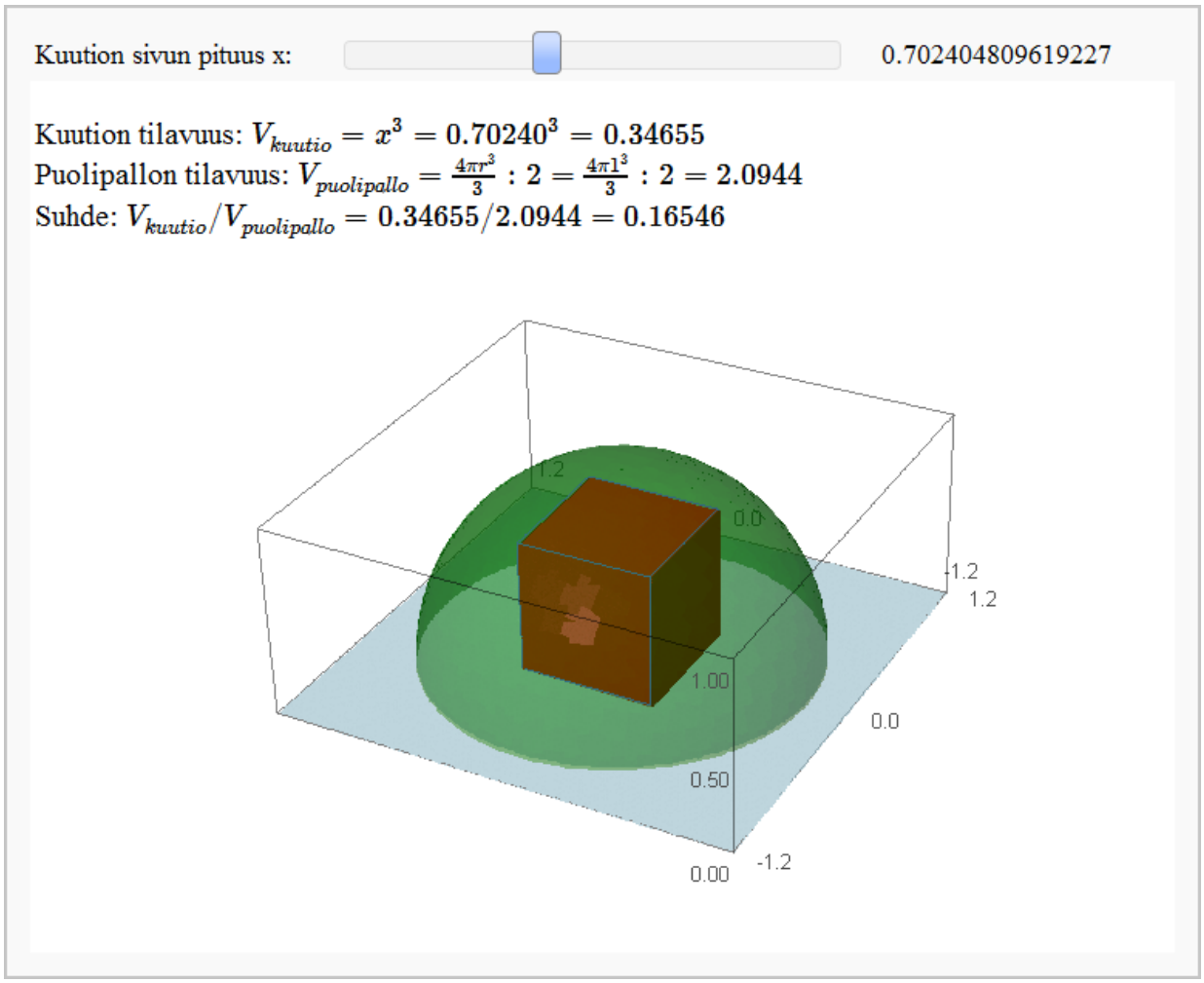
Kevään 2010 pitkän matematiikan ylioppilaskokeen tehtävässä 4 oli seuraava konstruktio: puolipallon sisälle on asetettu kuutio siten, että sen yksi sivutahko on puolipallon pohjatasolla ja vastakkaisen sivutahkon kärkipisteet ovat pallopinnalla. Tehtävässä kysytään, kuinka monta prosenttia kuution tilavuus on puolipallon tilavuudesta. Tarkastellaan ongelmaa numeerisesti Sagessa. Menetelmää voidaan käyttää esimerkiksi analyyttisesti lasketun arvon tarkistamiseen tai vastauksen suuruusluokan arvioimiseen kolmiulotteisten kappaleiden ominaisuuksia koskevassa ongelmassa.

Piirretään tehtävässä esiintyvät kappaleet koordinaatistoon ja lasketaan niiden tilavuudet. Olkoon pallon säde yksi; kappaleiden mittasuhteilla ei ole numeerisen tarkastelun kannalta merkitystä, koska kappaleiden tilavuuksien suhde on yhtä suuri riippumatta kappaleiden koosta. Puolipallo voidaan piirtää metodin *implicit_plot3d* avulla. Metodi piirtää implisiittisen yhtälön ratkaisun, kun muuttujat saavat arvoja annetuilta väleiltä. Valitaan kuution keskipistekoordinaatiksi $(0, 0, koko/2)$, missä *koko* on kuution sivun pituus. Tällöin yksi kuution sivutahkoista tulee puolipallon pohjatasolle. Tehtävässä pyydetyn suhteen määrittäminen tehdään tämän jälkeen silmämääräisesti säätämällä kuution kokoa, esimerkiksi haarukoimalla, kunnes pohjatasoa vastakkaisen sivutahkon kärkipisteet koskettavat pallopintaa.

Ohjelma 6: Kuution tilavuuden suhde puolipallon tilavuuteen

```
x, y, z = var("x, y, z")

@interact
def _(koko = slider(0.5, 1, label="Kuution sivun pituus x:")):

# Piirretään ja esitetään sovelluksen graafiset objektit.
  puolipallo = implicit_plot3d(x^2 + y^2 + z^2 == 1, (x,-1,1), (y,-1,1), (z,0,1),
      color="green", opacity=0.4)
    kuutio = cube((0, 0, koko/2), size=koko, opacity=0.9, color="red",
        frame_thickness=1)
   taso = plot3d(0, (x, -1.2, 1.2), (y, -1.2, 1.2), color="lightblue", opacity=0.6)
   show(puolipallo + kuutio + taso, aspect_ratio=1)

# Lasketaan kuution ja puolipallon tilavuudet.
   V_k = koko^3
   V_p = 4*pi*1^3/6
```

```
# Esitetään tehtävään liittyvät tunnusluvut.
   html("$\\text{Kuution tilavuus: }V_{kuutio}=x^3=%s^3=%s"%(N(koko,digits=5),
       N(V_k, digits=5)))
   html("$\\text{Puolipallon tilavuus: }V_{puolipallo}=\\frac{4\pi
       r^3}{3}:2=\\frac{4\pi 1^3}{3}:2=%s$"%N(V_p, digits=5))
   html("$\\text{Suhde: }V_{kuutio}/V_{puolipallo}=%s/%s=%s$"%(N(V_k,
       digits=5),N(V_p, digits=5),N(V_k/V_p, digits=5)))
```



Kuva 20: Geometrisen ongelman numeerinen tarkastelu.

4.8 Tehtäviä

Tehtävä 4-1: Ratkaise yhtälöt Sagen avulla:

a) $\sin^2 \alpha - 3\cos\alpha = 0$ **b)** $2\sin^2 \alpha + \sin\alpha = 0$

c) $2\sin(2\alpha - \frac{\pi}{2}) + \sqrt{3} = 0$ **d)** $\tan 2\alpha = \tan(\alpha + \frac{\pi}{2})$

Tehtävä 4-2: Piirrä funktion $\sin\frac{1}{x}$ kuvaaja ja tarkastele käyrän kulkua origon lähellä. Tutki kuvaajaa tarkemmin välillä $[\frac{1}{1000}, \frac{1}{100}]$ ja arvioi, kuinka monta nollakohtaa funktiolla on tällä välillä. Ratkaise Sagen avulla yhtälö $\sin\frac{1}{x} = 0$ ja selvitä, kuinka moni yhtälön juurista on välillä $[\frac{1}{1000}, \frac{1}{100}]$.

Tehtävä 4-3: Tee seuraavat trigonometristen funktioiden tarkastelut yksikköympyräsovelluksen (luvussa 4.3) avulla:

a) Määritä likimääräisesti kaikki ne muuttujan x arvot radiaaneissa, joilla $\sin x = \frac{1}{2}$, kun x on välillä $[0, 2\pi]$. Mitkä ovat vastaavat asteluvut? Minkä arvon sinifunktio saa (likimääräisesti) x:n arvolla $1,27\pi$?

b) Määritä likimääräisesti $\cos\frac{5\pi}{3}$. Millä muuttujan x arvoilla $\cos(x) = 0$?

c) Määritä yksikköympyrästä $\tan\frac{\pi}{4}$ ja likimääräisesti $\tan(0{,}35\pi)$. Määritä lisäksi välillä $[0, 2\pi]$ likimääräisesti ne muuttujan x arvot, joilla $\tan x = -4$. Mikä on tangenttifunktion jakson pituus? Miksi arvelet ohjelman antavan virheilmoituksen, jos yrität määrittää tangenttifunktion arvoa pisteessä $\frac{\pi}{2}$?

Tehtävä 4-4: Tutki esimerkin 4.7 ongelmaa ja määritä kuution tilavuuden suhde puolipallon tilavuuteen yhden desimaalin tarkkuudella.

5 Differentiaali- ja integraalilaskenta

Lukion matematiikan opetuksessa differentiaali- ja integraalilaskennan perusteita käsitellään usealla kurssilla. Tässä luvussa esitellään Sagen derivointiin ja integrointiin erikoistuneita metodeja sekä raja-arvon, erotusosamäärän ja tangentin havainnollistamiseen liittyviä ohjelmia ja animaatioita. Kirjoitetaan myös opetuskäyttöön soveltuva ohjelma, joka integroi annetun funktion ja havainnollistaa määrätyn integraalin laskemista koordinaatistossa.

5.1 Raja-arvojen määrittäminen

Raja-arvo kuvaa funktion käyttäytymistä, kun funktion muuttuja lähestyy tiettyä pistettä tai ääretöntä. Sagessa raja-arvoja voidaan määrittää metodilla *limit(lauseke, piste, dir=None, taylor=False)* [67]. Parametri *dir* tarkoittaa suuntaa, josta muuttujan arvo lähestyy pistettä. Sille voidaan antaa arvona joko "*minus*" (lähestyy vasemmalta) tai "*plus*" (lähestyy oikealta). Joissakin tapauksissa raja-arvon määrittäminen onnistuu paremmin muuttamalla lauseke ensin Taylorin sarjaksi. Tämä tapahtuu antamalla parametrille *taylor* totuusarvoksi *True*.

Esimerkki:

Määritetään raja-arvo

$$\lim_{x \to 4} \frac{\sqrt{x} - 2}{x - 4}.$$

```
x = var("x")
limit((sqrt(x)-2)/(x-4), x=4)
```

```
1/4
```

Tarkastellaan toisena esimerkkinä Neperin luvun määrittävää raja-arvoa

$$\lim_{x \to \infty} (1 + \frac{1}{x})^x.$$

```
x = var("x")
f = (1+1/x)^x
f.limit(x=oo)
```

```
e
```

5.1.1 Erotusosamäärä

Derivaatan määritelmä perustuu erotusosamäärän raja-arvoon. Pisteessä a funktion $f(x)$ derivaatta on

$$f'(a) = \lim_{x \to a} \frac{f(x) - f(a)}{x - a}.$$

Geometrisesti derivaattaa voidaan havainnollistaa funktion käyrän tangentin kulmakertoimena. Tarkastellaan sovellusta, joka piirtää pisteiden $(x0, f(x0))$ ja $(a, f(a))$ kautta kulkevan suoran, esittää sen yhtälön ja laskee kulmakertoimen erotusosamäärän avulla. Funktion $f(x)$ voi vapaasti valita, samoin muuttujien arvot ja tarkasteluvälin.

Sovelluksen ohjelmakoodissa metodi *f.find_minimum_on_interval(a, b)* palauttaa funktion pienimmän arvon välillä $[a, b]$.

Ohjelma 7: Erotusosamäärä

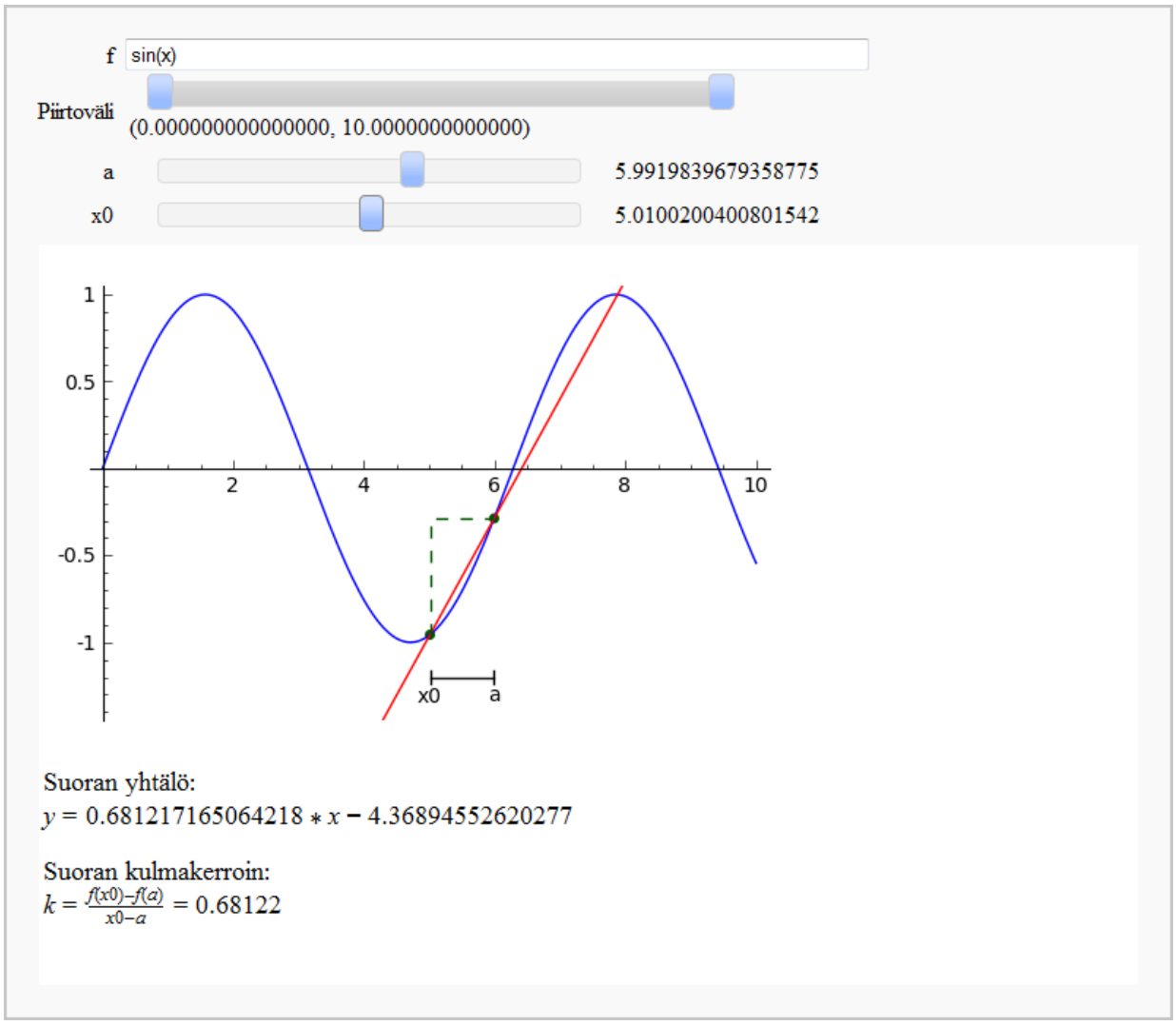
```
@interact
def erotusosamaara(f = input_box(default=sin(x)),
                   vali = range_slider(0, 10, 0.1, default=(0.0, 10.0),
                       label="Piirtoväli"),
                   a = slider(0, 10, None, 5.5),
                   x0 = slider(0, 10, None, 2.5)):
    f(x) = f
    fmax = f.find_maximum_on_interval(vali[0], vali[1])[0]
    fmin = f.find_minimum_on_interval(vali[0], vali[1])[0]
    f_korkeus = fmax - fmin

    # Välin [x0, a] esittämiseen liittyvä grafiikka.
    alaviiva_y = fmin - 0.1*f_korkeus
    alaviiva_0 = line2d([(x0, alaviiva_y), (a, alaviiva_y)], rgbcolor="black")
    alaviiva_1 = line2d([(x0, alaviiva_y + 0.02*f_korkeus), (x0,
        alaviiva_y-0.02*f_korkeus)], rgbcolor="black")
    alaviiva_2 = line2d([(a, alaviiva_y + 0.02*f_korkeus), (a,
        alaviiva_y-0.02*f_korkeus)], rgbcolor="black")
    teksti_x0 = text("x0", (x0, alaviiva_y - 0.05*f_korkeus), rgbcolor="black")
    teksti_a = text("a", (a, alaviiva_y - 0.05*f_korkeus), rgbcolor="black")
    alaviiva = alaviiva_0 + alaviiva_1 + alaviiva_2 + teksti_x0 + teksti_a

    # Pisteiden (x0, f(x0)) ja (a, f(a)) kulkevan suoran yhtälö.
    tanf(x) = (f(x0)-f(a))*(x-a)/(x0-a) + f(a)
```

```
    # Esitetään suora ja siihen liittyvä grafiikka.
    fplot = plot(f(x), x, vali[0], vali[1])
    tanplot = plot(tanf(x), x, vali[0], vali[1], rgbcolor=(1,0,0))
    pisteet = point([(x0,f(x0)), (a,f(a))], pointsize=20, rgbcolor="#005500")
    katkoviiva = line2d([(x0,f(x0)), (x0,f(a)), (a,f(a))], rgbcolor="#005500",
        linestyle="--")
    show(fplot + tanplot + pisteet + katkoviiva + alaviiva, xmin=vali[0],
        xmax=vali[1], ymin=fmin-0.2*f_korkeus, ymax=fmax)

    # Esitetään suoran yhtälö ja kulmakerroin.
    html("<br>$\\text{Suoran yhtälö:}$")
    html("$y = %s$<br>"%tanf(x))
    html("$\\text{Suoran kulmakerroin:}$")
    html("$k=\\frac{f(x0)-f(a)}{x0-a}=%s$<br>"%(N(derivative(tanf(x), x), digits=5)))
```

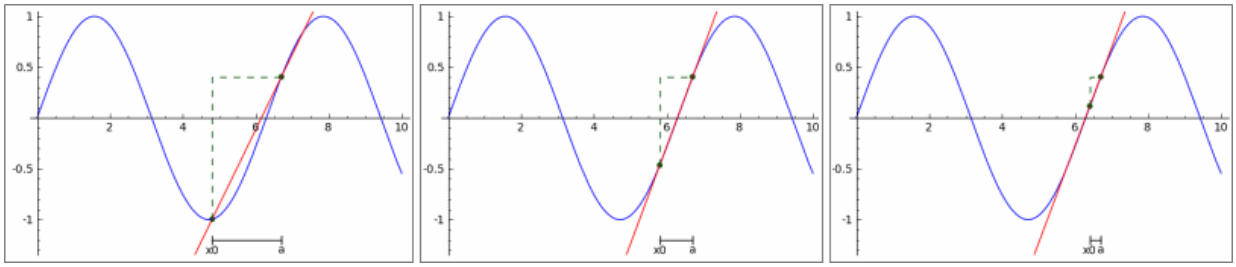
Ohjelman avulla voidaan konkreettisesti tutkia, miten suora käyttäytyy, kun muuttuja x lähestyy pistettä a. Muuttujien arvojen ollessa lähellä toisiaan, $x0 \approx a$, havaitaan suoran esittävän funktion tangenttisuoraa tässä pisteessä ja erotusmäärän arvon vastaavan likimäärin funktion derivaattaa. (Kuva 21.)

Interaktiivinen sovellus voidaan muuntaa animaatioksi kutsumalla ohjelmafunktiota toistuvasti eri parametrien arvoilla. Animointia varten sovelluksesta poistetaan interaktiiviset ominaisuudet määrittelyriviltä. Muokataan ohjelmakoodia lisäksi siten, että graafisten objektien piirtämisen sijaan ohjelma palauttaa yhden, kaikki objektit sisältävän komposiittikuvan, joka lisätään animoitavien kuvien listaan. Tämän jälkeen kuvasarja animoidaan metodin *animate* avulla. Erotusosamäärää havainnollistavasta ohjelmasta voidaan muodostaa animaatio esimerkiksi seuraavalla tavalla:

```
def erotusosamaara(f, vali, a, x0):

    ... # (sama kuin edellisessä ohjelmakoodissa)

    komposiitti = fplot + tanplot + pisteet + katkoviiva + alaviiva
    return komposiitti

v = []
for x0 in srange(3, 6.7, 0.2):
    v.append(erotusosamaara("sin(x)", (0,10), 6.7, x0))
a = animate(v, xmin = 0, xmax = 10, ymin = -1, ymax = 1)
a.show(delay = 40)
```



Kuva 21: Erotusosamäärän raja-arvoa ja sen yhteyttä derivaatan käsitteeseen havainnollistava interaktiivinen sovellus.



Kuva 22: Erotusosamäärän raja-arvon lähestymistä havainnollistavan animaation pysäytyskuvia.

Esimerkki: Derivaatan määrittäminen erotusosamäärän raja-arvon avulla

Derivaatan arvoja voidaan määrittää erotusosamäärän raja-arvon avulla. Määritetään potenssifunktion x^n yleinen derivointikaava laskemalla erotusosamäärän raja-arvo pisteessä a:

```
x, n, a = var("x, n, a")
g(x) = x^n
f = (g(x) - g(a))/(x-a)
f.limit(x=a)
```

```
n*a^(n - 1)
```

5.2 Derivointi

Sagessa funktio derivoidaan metodilla *derivative(lauseke, muuttuja)* tai *diff(lauseke, muuttuja)*. Argumenttina metodille annetaan muuttuja, jonka suhteen lauseke halutaan derivoida. Jos syötteenä on useampi muuttujakirjain, metodi derivoi funktion kunkin muuttujan suhteen siinä järjestyksessä, kun ne on annettu argumenttina. [68]

Esimerkki: Derivaatan määrittäminen

Määritetään polynomin $f(x) = x^3 + 3x^2 - 2x + 17$ ensimmäinen ja toinen derivaatta.

```
f(x) = x^3 + 3*x^2 - 2*x + 17
print derivative(f(x), x)
print derivative(f(x), x, x)
```

```
3*x^2 + 6*x - 2
6*x + 6
```

Korkeamman asteen derivaattoja laskettaessa voidaan käyttää lyhennysmerkintää. Esimerkiksi edellisen funktion kolmannen asteen derivaatta määritetään seuraavasti:

```
f(x) = x^3 + 3*x^2 - 2*x + 17
print derivative(f(x), x, 3)
```

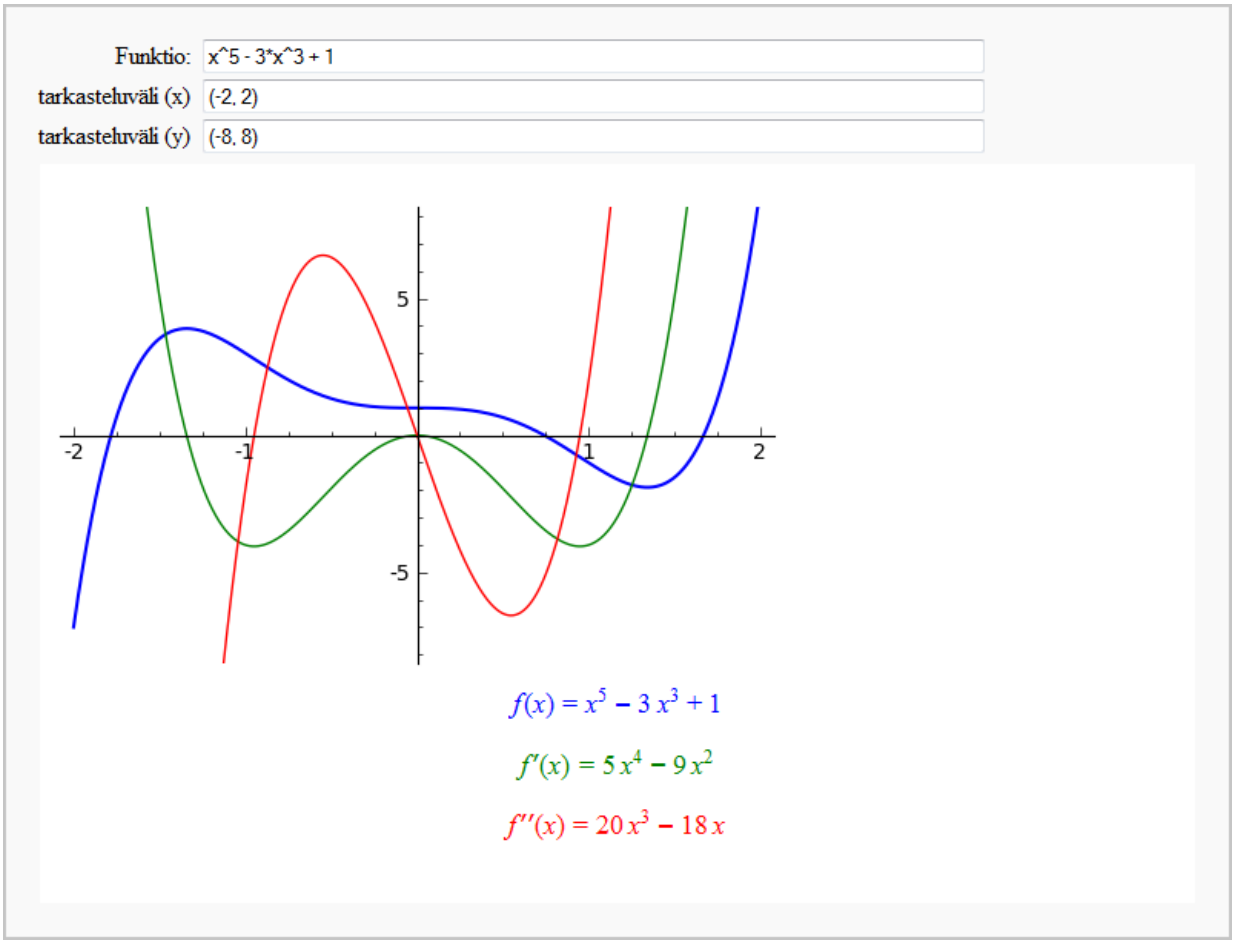
```
6
```

5.2.1 Funktion ensimmäinen ja toinen derivaatta samassa koordinaatistossa

Kirjoitetaan vuorovaikutteinen sovellus, joka piirtää syötteenä annetun funktion sekä sen ensimmäisen ja toisen derivaatan kuvaajat samaan koordinaatistoon. Käyttäjä voi valita piirtovälin kirjoittamalla tarkasteluvälin ala- ja ylärajat sulkeisiin. Pystyakselin oletusarvo (0, 0) piirtää funktion ja sen derivaattafunktiot niiden saavuttamien pienimmän ja suurimman arvon välillä. (Kuva 23.)

Ohjelma 8: Funktion ensimmäinen ja toinen derivaatta samassa koordinaatistossa

```
@interact
def _(funktio = input_box(default=x^5-3*x^3+1, label="Funktio:"),
       x_vali=input_box(default=(-2,2), label="tarkasteluväli (x)"),
       y_vali=input_box(default=(-8,8), label="tarkasteluväli (y)")):

   # Funktion ja sen derivaattojen määrittäminen.
   f(x) = funktio
   fd(x) = derivative(f(x), x)
   fdd(x) = derivative(fd(x), x)

   # Kuvaajien piirtäminen ja yhdistäminen.
   kuvaajat = plot(f(x), x_vali, thickness=1.5) + plot(fd(x), x_vali,
       color="green") + plot(fdd(x), x_vali, color="red")

   # Piirtovälin määrittely.
   if y_vali == (0,0):
      show(kuvaajat, xmin=x_vali[0], xmax=x_vali[1])
   else:
      show(kuvaajat, xmin=x_vali[0], xmax=x_vali[1], ymin=y_vali[0], ymax=y_vali[1])

   # Tulostetaan funktioiden lausekkeet.
   html("<center>$\color{Blue}{f(x) = %s}$</center>"%latex(f(x)))
   html("<center>$\color{Green}{f'(x) = %s}$</center>"%latex(fd(x)))
   html("<center>$\color{Red}{f''(x) = %s}$</center>"%latex(fdd(x)))
```



Kuva 23: Ohjelma, joka piirtää funktion sekä sen ensimmäisen ja toisen derivaatan samaan koordinaatistoon.

5.2.2 Derivoinnin osaamista testaava opetusohjelma

Sagen vuorovaikutteisten toimintojen avulla voidaan kirjoittaa opetusohjelmia, jotka luovat satunnaisesti tehtäviä ja tarkistavat käyttäjän antaman vastauksen oikeellisuuden. Tämän tyyppiset ohjelmat soveltuvat mekaanisen laskuvalmiuden kehittämiseen ja opiskelijan taitotason itsearviointiin. Esimerkkinä esitetään ohjelma, joka pyytää käyttäjää derivoimaan ohjelman generoiman satunnaisen polynomifunktion. Ohjelma tarkistaa vastauksen symbolisesti ja antaa palautteen suorituksesta. Halutessaan käyttäjä voi pyytää ohjelmaa näyttämään oikean ratkaisun. (Kuva 24.)

Polynomin generoimista varten määritellään polynomirengas R, jonka muuttujakirjaimena toimii x . Polynomin termien kertoimet valitaan kokonaislukujen joukosta $\mathbb{Z}$ (Sagessa ZZ[]). Metodikutsu *R.random_element(3, -10, 10)* palauttaa R:n satunnaisen alkion, joka tässä tapauksessa on kokonaislukukertoiminen polynomi, jonka termien korkein asteluku on kolme ja kertoimet ovat välillä [-10,10]. Lisätietoa polynomirenkaista ja polynomien laskutoimituksista on viittessä [69].

Avainsana *global* sallii moduulin tasolla määriteltyjen muuttujien arvojen muuttamisen funktion sisällä. Koska Sagessa ei vielä ole saatavilla interaktiivista painiketta, käytetään *checkbox*-objektia painikkeen tavoin. Rastilaatikon tilan muutos havaitaan käyttämällä apumuuttujia. Alla olevassa ohjelmassa tällaisia apumuuttujia ovat *uusitehtava* ja *naytaratkaisu*.

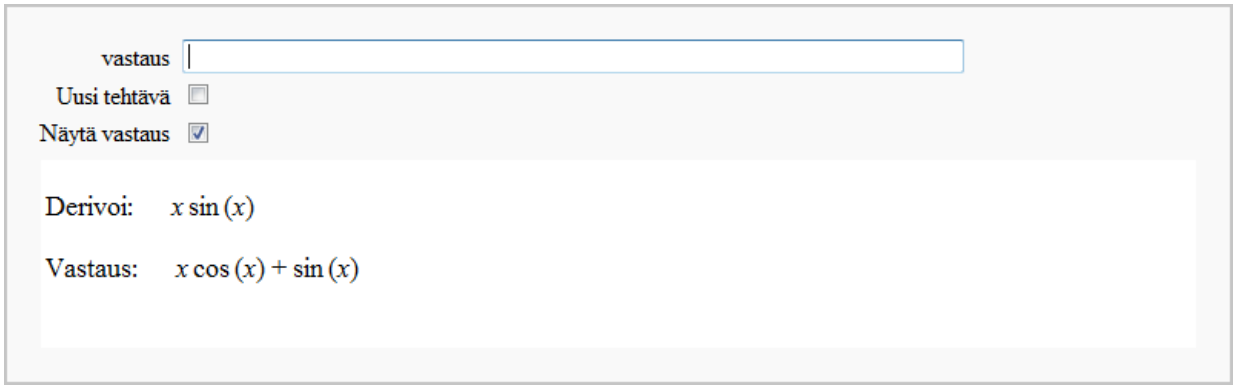
Ohjelma 9: Derivoinnin osaamista testaava opetusohjelma

```
# Luodaan polynomirengas R ja alustetaan apumuuttujat.
R.<x> = ZZ[]
funktio = R.random_element(3, -10,10)
uusitehtava = True
naytaratkaisu = True

@interact
def _(vastaus = input_box(),
       uusi = checkbox(default=False, label="Uusi tehtävä"),
       ratkaisu = checkbox(default=False, label="Näytä vastaus")):

   # Globaalit muuttujat.
   global funktio, uusitehtava, naytaratkaisu

   # Tarkistaa vastauksen oikeellisuuden.
   if vastaus == derivative(funktio(x)) or uusi == uusitehtava:
      uusitehtava = not uusi
      if vastaus == derivative(funktio(x)):
         html("$\\text{Oikein! Uusi tehtävä:}$")
      funktio = R.random_element(3, -10,10)
   elif vastaus and ratkaisu != naytaratkaisu and uusi != uusitehtava:
      html("$\\text{Virhe. Yritä uudestaan!}$")

   html("$\\text{Derivoi: }\;\;\;\;%s$<br>"%latex(funktio(x)))

   # Näyttää oikean vastauksen, jos rastilaatikon tila on muuttunut.
   if ratkaisu == naytaratkaisu:
      naytaratkaisu = not ratkaisu
      html("$\\text{Vastaus: }\;\;\;\;%s$<br>"%latex(derivative(funktio(x))))
```

Kuva 24: Opetusohjelma luo satunnaisia polynomifunktioita ja tarkistaa syötetyn vastauksen oikeellisuuden automaattisesti.

Ohjelmaa voidaan laajentaa kysymään muidenkin kuin polynomifunktioiden derivaattoja. Seuraavassa luvussa esiteltävien satunnaislukujen avulla voidaan generoida satunnaisia lausekkeita, joissa esiintyy esimerkiksi trigonometrisia tai logaritmisia funktioita sekä näiden tuloja ja yhdistettyjä funktioita.

Satunnaisesti luotuihin funktioihin liittyy myös ongelmia: sama funktio saattaa toistua lyhyellä aikavälillä tai generoitu funktio ei ole mielekäs. Tästä syystä tehtävien määritteleminen etukäteen saattaa joissakin tilanteissa olla järkevämpää kuin lausekkeiden satunnainen generoiminen. Liitteenä (A.2) on esitetty ohjelma, jossa derivoitavat funktiot luetaan tekstitiedostosta. Funktiot kirjoitetaan tiedostoon riveittäin niin, että jokaisella rivillä on yksi derivoitava funktio. Ohjelma laskee funktion derivaatan symbolisesti ja esittää lopuksi käyttäjälle yhteenvetona, kuinka monta tehtävää kokonaismäärästä oli laskettu oikein. (Kuva 25.)



Kuva 25: Derivoinnin hallitsemista testaavassa ohjelmassa ratkaistavat funktiot voidaan lukea generoimisen sijaan myös tiedostosta.

5.3 Integrointi

Metodi *integrate(lauseke, muuttuja)* palauttaa lausekkeen määräämättömän integraalin. On huomattava, että Sagen antamaan vastaukseen ei ole sisällytetty integroimisvakiota. Jos metodin argumenteissa määritetään myös integroimisväli, *integrate(lauseke, muuttuja, a, b)*, metodi palauttaa välillä $[a, b]$ määrätyn integraalin. Sage käyttää symboliseen integrointiin matemaattisen ohjelmiston *Maximan* kirjastoja. [70]

Esimerkkejä: Määräämättömän ja määrätyn integraalin laskeminen

Määritetään integraali $\int (4x^3 - 2x^2 + 4x - 1)\,\mathrm{d}x$:

```
integrate(4*x^3 - 2*x^2 + 4*x - 1, x)
```

```
x^4 - 2/3*x^3 + 2*x^2 - x
```

Integroidaan lauseke $\sin(x)x^2$:

```
integrate(sin(x)*x^2, x)
```

```
-(x^2 - 2)*cos(x) + 2*x*sin(x)
```

Lasketaan monomin x^2 määrätty integraali välillä $[3, 9]$:

```
integrate(x^2, x, 3, 9)
```

```
234
```

Esimerkki: Paloittain määritellyn funktion integroiminen

Olkoon $g(x)$ paloittain määritelty funktio

$$g(x) = \begin{cases} 1, & \text{kun } x < 1 \\ 2, & \text{kun } x \geq 1. \end{cases}$$

Määritetään integraalin $\int_0^2 g(x)\,\mathrm{d}x$ arvo *piecewise*-metodin avulla:

```
g(x) = piecewise([[(-1, 1), 1], [(1, 3), 2]])
print integrate(g, x, 0, 2)
```

```
3
```

Paloittain määritellyn funktion integroimisessa on huomioitava, ettei funktion määritteleminen Python-funktiona välttämättä palauta oikeaa tulosta:

```
var("x")
def g(x):
  if x<1:
     return 1
  else:
     return 2
print integrate(g(x), x, 0, 2)
```

```
4
```

5.3.1 Oletuksien määräminen

Sagessa voidaan antaa muuttujille oletuksia komennolla *assume(oletus)*. Tämä on hyödyllistä tapauksissa, joissa tulos riippuu muuttujan määrittelyjoukosta. Komento *forget()* poistaa kaikki annetut oletukset. [71]

Esimerkkejä

Tutkitaan funktion $f(x) = x^n$ integraaleja, kun $n \geq 1$ ja $n = -1$

```
x,n = var("x,n")
f = x^n
assume(n >= 1)
assume(n, "integer")
f.integrate(x)
```

```
x^(n + 1)/(n + 1)
```

```
forget()
assume(n == -1)
f.integrate(x)
```

```
log(x)
```

5.3.2 Numeerinen integrointi

Määrätyn integraalin määrittäminen symbolisesti voi epäonnistua, jos integrandi on hyvin epäsäännöllinen tai jos määrittäminen on teoreettisesti mahdotonta. Tällöin voidaan

käyttää numeerisia menetelmiä määrätyn integraalin laskemiseen. Muun muassa metodi *nintegrate(lauseke, muuttuja, a, b)* määrittää välillä $[a, b]$ määrätyn integraalin likiarvon. Metodi palauttaa neljä lukuarvoa, joista ensimmäinen on saatu likiarvo, toinen luku on arvio vastauksen suhteelliselle virheelle, kolmas kertoo kuinka monessa pisteessä integrandin arvo evaluoitiin ja viimeinen luku on virhekoodi.

Virhekoodin arvon ollessa nolla, ei ongelmia kohdattu algoritmin suorituksen aikana. Mahdollisia virhetekijöitä ovat muun muassa integrandin poikkeuksellinen epäsäännöllisyys tai rajoittamattomuus integrointivälillä. Lisätietoa numeerisesta integroinnista ja virhekoodeista on esitetty viitteessä [72].

Esimerkkejä:

```
f(x) = x^2
f.nintegrate(x, 0, 1)
```

```
(0.33333333333333343, 3.7007434154171903e-15, 21, 0)
```

```
f(x) = e^(x^3)
print "Tarkka arvo:", f(x).integrate(x, 0, 1)
print "Likiarvo:", f(x).nintegrate(x, 0, 1)
```

```
Tarkka arvo: -1/6*(I*sqrt(3) - 1)*(gamma(1/3) - gamma(1/3, -1))
Likiarvo: (1.34190441797742, 1.489813181684752e-14, 21, 0)
```

5.3.3 Määrätyn integraalin havainnollistaminen

Määrätyn integraalin laskemista voidaan visualisoida värittämällä funktion käyrän ja vaaka-akselin välinen ala, joka vastaa määrätyn integraalin arvoa. *Plot*-metodin parametri *fill* värittää valitun osan koordinaatistosta (taulukko 14).

Arvo	Kuvaus
True tai "axis"	Värittää käyrän ja x-akselin välisen alan.
"min", "max"	Värittää funktion ja sen saaman pienimmän (suurimman) arvon välisen alan.
reaaliluku	Värittää funktion ja vaakasuoran viivan y = c välisen alan, missä c on annettu reaaliluku.
funktio	Värittää alkuperäisen ja annetun funktion välisen alan.

Taulukko 14: Plot-metodin *fill*-option mahdollisia arvoja.

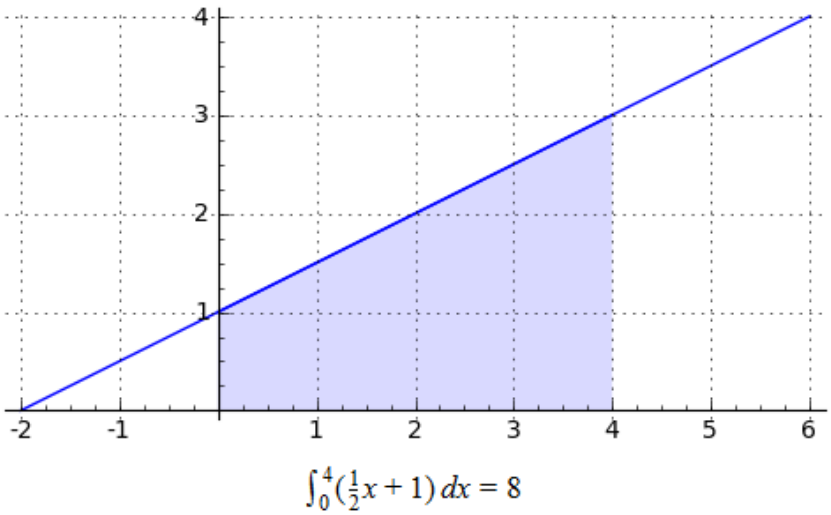
Täyttövärin ja läpinäkyvyyden tason voi valita parametreilla *fillcolor* ja *fillalpha*. Ne saavat samoja arvoja kuin *plot*-metodin vastaavat parametrit *color* ja *alpha*.

Koordinaatistoon voidaan piirtää apuviivoja, jotka helpottavat käyrän ja koordinaattiakselin välisen pinta-alan silmämääräistä arviointia. Parametri *gridlines* määrittää koordinaatistoon piirrettävän ruudukon. Kirjoittamalla *plot*- tai *show*-metodin argumentteihin *gridlines=True* ohjelma piirtää niin sanotut pääviivat koordinaatiston akseleille merkittyjen lukujen kohdille. Tiheämmän ruudukon saa parametrin arvolla *gridlines="minor"*.

Ruudukon viivat voi piirtää myös yksitellen määrittelemällä viivojen sijainnit parametrin arvolla *gridlines=[*L_x*,* L_y*]*, missä L_x on lista pystyviivojen x-koordinaateista ja L_y vaakaviivojen y-koordinaateista. [39]

Esimerkki:

```
funktio = plot(1/2*x + 1, (-2, 6))
integraali = plot(1/2*x + 1, (0, 4), fill=True, fillcolor="blue",
   fillalpha=0.15)
show(funktio + integraali, gridlines=True, aspect_ratio=1, xmin=-2,
   xmax=6)
html("<center>$\int_0^4(\\frac{1}{2}x+1)\,dx=8$</center>")
```



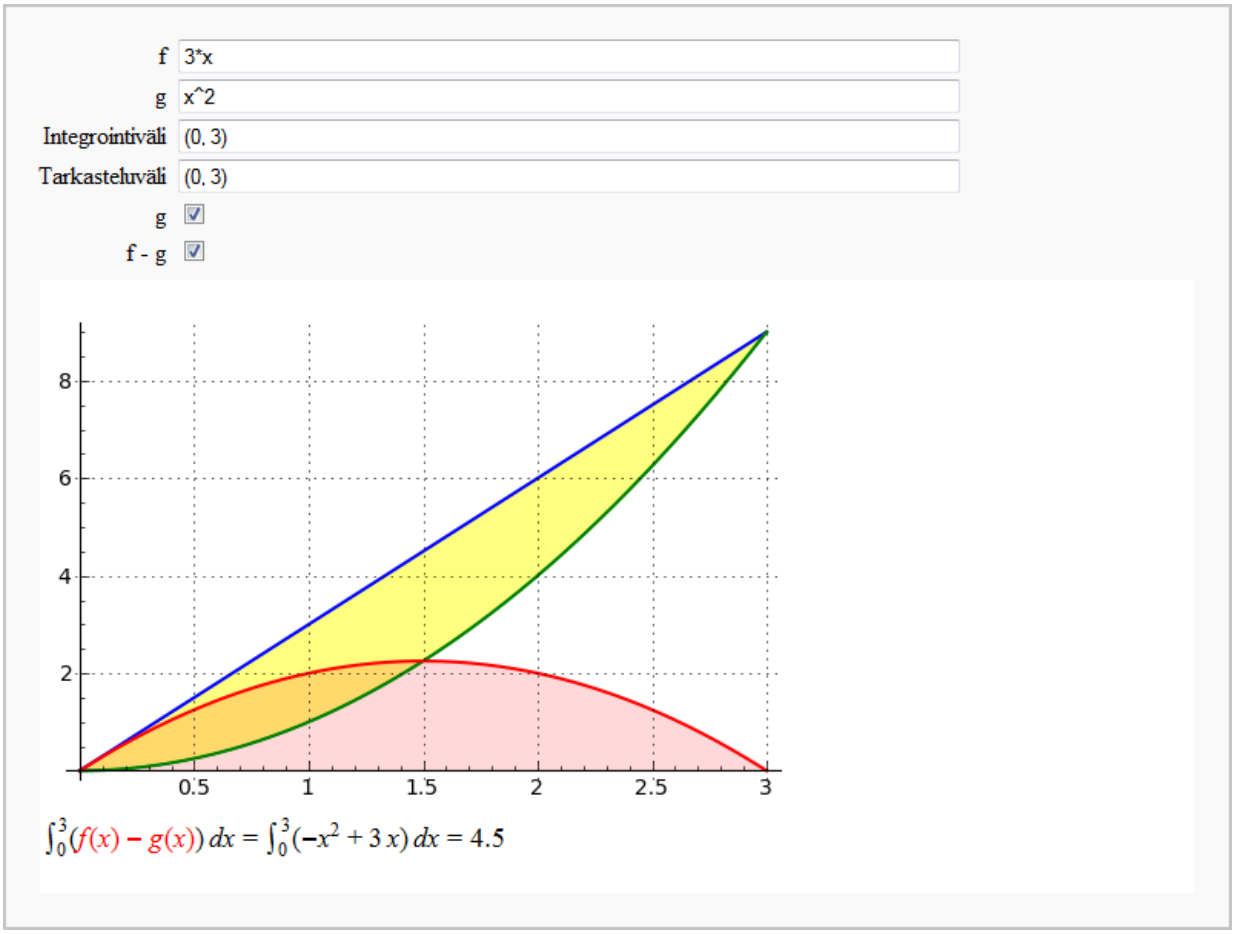
Kuva 26: Määrätyn integraalin havainnollistaminen plot-metodin lisäparametrien avulla.

5.3.4 Sovellus: Kahden käyrän rajaaman alan määrittäminen

Lopuksi esitetään sovellus, joka laskee annetun funktion f määrätyn integraalin käyttäjän antamalla välillä. Ohjelma piirtää integroitavan funktion koordinaatistoon ja värittää integrointivälillä funktion ja koordinaattiakselin välisen alan siniseksi.

Sovelluksen avulla voidaan myös tutkia kahden funktion välisen pinta-alan laskemista. Jos ohjelmalle määrittelee toisen funktion g ja valitsee sen kytkimestä aktiiviseksi, sovellus värittää funktioiden f ja g välisen alan keltaiseksi ja laskee erotusfunktion $f - g$ määrätyn integraalin annetulla välillä. Saatu tulos esitetään kuvaajan alle kirjoitettuna.

Käyttäjä voi halutessaan myös piirtää erotusfunktion. Tällöin ohjelma esittää erotusfunktion ja vaaka-akselin välisen alan punaisena. Koska väritys on osittain läpinäkyvä, erottuvat päällekkäin olevat alueet toisistaan. (Kuva 27.)



Kuva 27: Interaktiivisen ohjelman avulla voidaan tutkia kahden käyrän välistä pinta-alaa. Kuvassa erotusfunktion ja vaaka-akselin välinen ala on väritetty punaisella.

Ohjelma 10: Määrätyn integraalin havainnollistaminen interaktiivisesti

```
@interact
def _(f = input_box(default=3*x), g = input_box(default=x^2),
      int_vali = input_box(default=(0,3), label="Integrointiväli"),
      x_vali = input_box(default=(0,3), label = "Tarkasteluväli"),
      gt = checkbox(default=False, label="g"),
      erotus = checkbox(default=False, label="f - g")):
   f(x) = f; g(x) = g

   # Piirretään funktion g kuvaaja ja väritetään f:n ja g:n välinen ala keltaiseksi.
   if gt:
      g_kuva = plot(g(x), x, x_vali, color="green", thickness=1.5)
      g_kuva += plot(g(x), x, int_vali, color="green", fill=f, fillcolor="yellow",
          fillalpha=0.5)
      f_kuva = plot(f(x), x, x_vali, color="blue", thickness=1.5)
      tuloste = "$\int_{%s}^{%s}(\color{Blue}{f(x)} - \color{Green}{g(x)})\,
          dx=\int_{%s}^{%s}({%s})\, dx={%s}$"%(int_vali[0], int_vali[1],
          int_vali[0], int_vali[1], latex(f(x)-g(x)), (f(x)-g(x)).nintegrate(x,
          int_vali[0], int_vali[1])[0])
   # Piirretään pelkkä funktio f ja väritetään vaaka-akselin ja f:n välinen ala
       siniseksi.
   else:
      f_kuva = plot(f(x), x, x_vali, color="blue", thickness=1.5)
      f_kuva += plot(f(x), x, int_vali, color="blue", fill=True, fillcolor="blue",
          fillalpha=0.15)
      g_kuva = Graphics()
      tuloste = "$\int_{%s}^{%s}(\color{Blue}{f(x)})\, dx=\int_{%s}^{%s}({%s})\,
          dx={%s}$"%(int_vali[0], int_vali[1], int_vali[0], int_vali[1],
          latex(f(x)), f(x).nintegrate(x, int_vali[0], int_vali[1])[0])

   # Piirretään erotusfunktio sekä väritetään sen ja vaaka-akselin välinen ala
       punaiseksi.
   if erotus:
      h_kuva = plot(f(x) - g(x), x, int_vali, color="red", thickness=1.5,
          fill=True, fillcolor="red", fillalpha=0.15)
      tuloste = "$\int_{%s}^{%s}(\color{Red}{f(x)-g(x)})\,
          dx=\int_{%s}^{%s}({%s})\, dx={%s}$"%(int_vali[0], int_vali[1],
          int_vali[0], int_vali[1], latex(f(x)-g(x)), (f(x)-g(x)).nintegrate(x,
          int_vali[0], int_vali[1])[0])
   else:
      h_kuva = Graphics()

   show(f_kuva + g_kuva + h_kuva, gridlines=True)
   html(tuloste)
```

5.4 Tehtäviä

Tehtävä 5-1: Määritä erotusosamäärän raja-arvon avulla yleiset derivointikaavat seuraaville funktioille, missä x on vapaa muuttuja ja n mielivaltainen luku. Määritä funktioiden derivaatat myös metodin *derivative* avulla ja vertaa tuloksia.

a) $sin(nx)$ **b)** $ln(x)$ **c)** n^{n^x}

Tehtävä 5-2: Määritä funktion $f(x) = x^5 - x^3 - x^2 - 5x$ ensimmäisen ja toisen kertaluvun derivaatat ja piirrä kuvaajat sovelluksen 8 (luvussa 5.2.1) avulla. Määritä derivaattojen nollakohdat kuvaajasta ja laadi derivaattojen merkkikaaviot. Mitä voit kertoa funktion käyrän muodosta 1. ja 2. kertaluvun derivaattojen perusteella?

Tehtävä 5-3: Määritä seuraavien funktioiden määrätyt integraalit symbolisesti sekä ohjelman 10 (luvussa 5.3.4) avulla numeerisesti.

a) $\int_0^1 (e^x + 1)\,dx$ *(yo-K10p)* **b)** $\int_0^1 \sqrt[3]{x}\,dx$ *(yo-K09p)*

c) $\int_{-1}^{4/9} \frac{1}{\sqrt{x+5}}\,dx$ *(yo-S09p)* **d)** $\int_0^{\pi} (1 + \sin x)\,dx$ *(yo-S08p)*

Tehtävä 5-4: (yo-K98p:7) Euroopan unionin tarkastaja mallintaa satelliittikuvassa näkyvän, trombin tuhoaman metsän alueeksi, joka jää käyrien $y = \sin x$ ja $y = \sin 2x$ väliin, kun $x \in [0, \pi]$. Mikä on tuhoalueen pinta-ala mallin mukaan, kun pituusyksikkönä on kilometri? Tarkastele ongelmaa Sagen avulla esimerkiksi sovellusta 10 (luvussa 5.3.4) apuna käyttäen.

6 Tilastot ja todennäköisyyslaskenta

Sage sisältää kattavia tilastolliseen laskentaan erikoistuneita moduuleita, muun muassa tilastollisen ohjelmistopaketin *R*:n ja Pythonin *SciPy*-kirjaston moduulit. Tietotekniikan avulla tilastotieteen opetuksessa voidaan painottaa tilastollisten käsitteiden hallintaa ja käyttää laskennassa laajoja aineistoja, joiden käsitteleminen laskimella ei ole mielekästä.

Tässä luvussa esitellään tilastollisista tunnusluvuista keskiarvo, mediaani, moodi, varianssi ja keskihajonta. Lisäksi tarkastellaan kategoristen muuttujien frekvenssijakaumien havainnollistamista pylväs- ja sektoridiagrammien avulla. Tutkitaan myös tilannetta, jossa tilastoyksiköihin liittyy kaksi kvantitatiivista tilastollista muuttujaa, joiden välistä riippuvuussuhdetta voidaan mallintaa regressiosuoralla. Todennäköisyyslaskentaan liittyen käsitellään satunnaislukuja ja esitetään kaksi lukio-opetukseen soveltuvaa simulaatiota. Luvussa käytetään Sagen ja Pythonin laajennuskirjastojen ohjelmafunktioita.

6.1 Tilastollisia tunnuslukuja

Havaintoaineistoa ja sen ominaisuuksia voidaan kuvailla tunnusluvuilla, jotka saadaan tilastollisen muuttujan arvoista erilaisilla laskutoimituksilla. Sagessa tunnuslukuja voidaan laskea listamuotoisesta datasta, joka on joko annettu ohjelmakoodissa tai tekstitiedostossa. Seuraavassa taulukossa on listattuna metodeja, jotka laskevat yleisimpiä tilastollisia tunnuslukuja, kun syötteenä on lista *v*.

Metodi	**Kuvaus**
mean(v)	keskiarvo
median(v)	mediaani
mode(v)	moodi
variance(v)	varianssi

Metodi	**Kuvaus**
std(v)	keskihajonta
sum(v)	alkioiden summa
prod(v)	alkioiden tulo

Jos listassa on parillinen määrä alkioita, mediaani palauttaa kahden keskimmäisen alkion keskiarvon. Moodi palauttaa kaikkien niiden luokkien edustajat, joilla on suurin frekvenssi.

6.1.1 Esimerkki: Tilastollisia tunnuslukuja

Lasketaan aineiston [1, 2, 2, 4, 4, 6] tilastollisia tunnuslukuja.

```
v = [1, 1, 2, 4, 4, 6]
print "Keskiarvo: ", mean(v)
print "Mediaani: ", median(v)
print "Moodi: ", mode(v)
print "Varianssi: ", variance(v)
print "Keskihajonta: ", std(v)
print "Summa: ", sum(v)
print "Tulo: ", prod(v)
```

```
Keskiarvo: 3
Mediaani: 3
Moodi: [1, 4]
Varianssi: 4
Keskihajonta: 2
Summa: 18
Tulo: 192
```

6.2 Kategorisen muuttujan frekvenssijakauman kuvaaminen

Kategorisen muuttujan frekvenssijakaumaa voidaan kuvata muun muassa pylväs- ja sektoridiagrammeilla. Seuraavissa esimerkeissä käytetään Python-kirjastoa *matplotlib* ja sen moduulia *pylab*, jonka syntaksi muistuttaa matemaattisen ohjelmiston Matlabin syntaksia. Matplotlib on nimensä mukaisesti kokoelma erilaisia kuvaajien ja muun grafiikan piirtämiseen kehitettyjä metodeja.

Koska matplotlib-kirjaston metodit sisältävät lukuisia mahdollisia parametreja, ei tässä yhteydessä esitetä kaikkia toimintoja, vaan tilastokuvien piirtämistä havainnollistetaan yksinkertaisten, dokumentoitujen esimerkkien avulla. Lisätietoa matplotlib-kirjaston metodeista, muun muassa tässä esitettävistä pylväs- ja sektoridiagrammeista, on esitetty tarkemmin viitteessä [73].

6.2.1 Esimerkki: Pylväsdiagrammi

Pylväskuvioita voidaan piirtää metodilla *bar*. Metodi saa syötteenä listat indekseistä ja niitä vastaavista arvoista.

Kyselytutkimuksessa 20 vastaajaa pyydettiin valitsemaan yksi neljästä eri vaihtoehdosta. Kyselyn tuloksena saatiin seuraava aineisto:

Luokka	Frekvenssi	Suhteellinen frekvenssi
1	3	15 %
2	6	30 %
3	2	10 %
3	9	45 %

Seuraava ohjelmakoodi piirtää saadusta aineistosta pylväsdiagrammin (kuva 28):

```
# Tuo pylab-moduulin metodit ohjelman käytettäväksi.
from pylab import *
clf()                       # Tyhjentää piirrostilan
figure(figsize = (4,4))     # Määrittää kuvan mittasuhteet
ind = [1, 2, 3, 4]          # Pylväiden paikat x-akselilla
arvot = [3, 6, 2, 9]        # Pylväiden korkeudet (frekvenssit)
width = 0.5                 # Pylvään leveys (yksikköinä)
# Luokkien nimet.
labels = ("Luokka 1", "Luokka 2", "Luokka 3", "Luokka 4")
# x-akselin jaotus, muotoa xticks(lista x-koordinaateista, nimet).
xticks([x + width/2.0 for x in ind], labels)
# y-akselin jaotus, muotoa yticks(lista y-koordinaateista).
yticks([1, 2, .. , 9])
# Pylväsdiagrammi, muotoa bar(indeksit, arvot, pylvään leveys,
   lisämääreet).
bar(ind, arvot, width, color="blue")
title("Otsikko")            # Kuvan otsikko
xlabel("x-akseli")          # x-akselin nimi
ylabel("y-akseli")          # y-akselin nimi
# Tallentaa kuvan solun yhteyteen tiedostonimellä pylvas.png
savefig("pylvas.png")
```

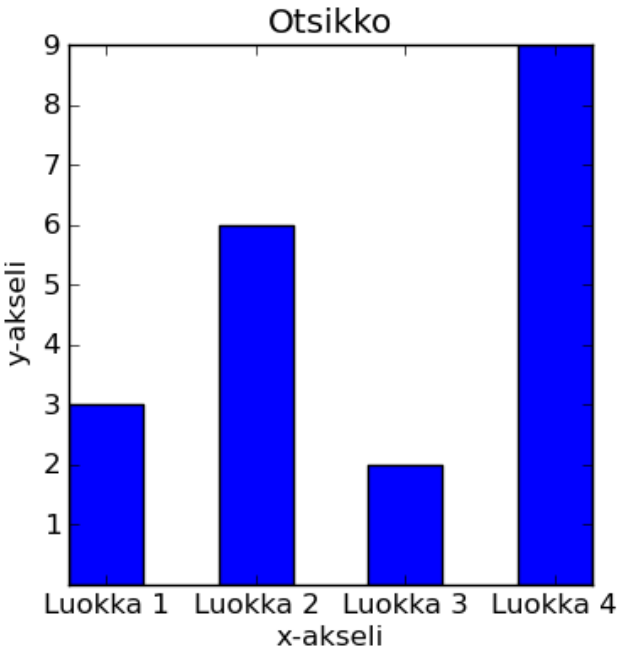
Myös Sagen piirrostilan koordinaatistoon voidaan piirtää pylväsdiagrammeja metodin *BarChart* avulla. Tämä mahdollistaa muun muassa käyrien esittämisen samassa koordinaatistossa pylväsdiagrammin kanssa. Metodia käytetään kuten yllä esitetyn *matplotlib*:n vastaavaa metodia, mutta se on toiminnaltaan rajoittuneempi. Esimerkki metodin *BarChart* käytöstä esitetään myöhemmin sovelluksessa 12 (luvussa 6.6).

6.2.2 Esimerkki: Sektoridiagrammi

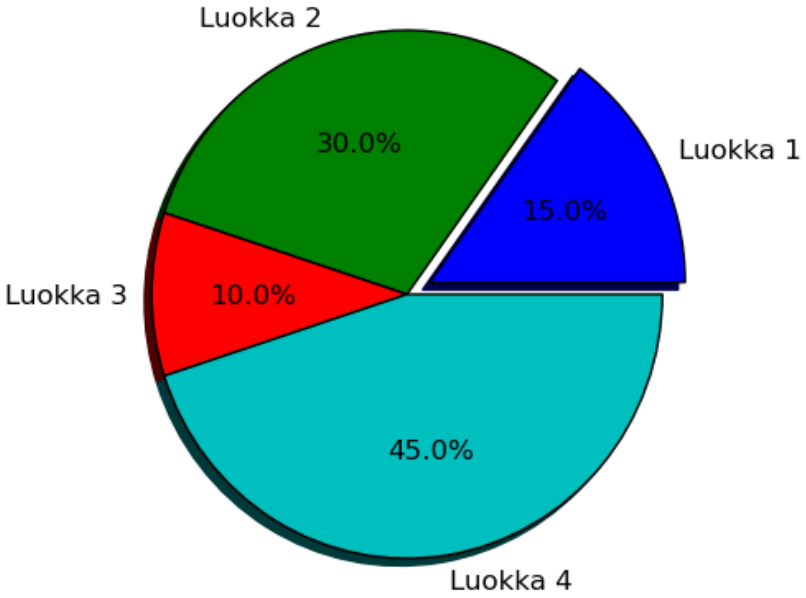
Laaditaan sektori- eli piirakkakuvio edellisen aineiston suhteellisten frekvenssien perusteella. Johonkin tiettyyn luokkaan kuuluvaa osuutta voidaan korostaa irrottamalla se kuviosta parametrin *explode* avulla. Parametri voidaan jättää myös pois, jos yksittäisiä luokkia ei haluta korostaa. Metodin *pie* parametrin *autopct* arvo "%1.1f%%"näyttää sektorien prosenttiosuudet kirjoitettuna diagrammin sisälle.

Kuvan 29 sektoridiagrammi on piirretty seuraavan koodin avulla:

```
from pylab import *            # Tuo pylab-moduulin metodit
clf()                          # Tyhjentää piirrostilan
figure(figsize=(5, 5))         # Määrittää kuvan mittasuhteet
# Luokkien nimet.
labels = ("Luokka 1", "Luokka 2", "Luokka 3", "Luokka 4")
# Osuudet prosentteina.
osuudet = [15, 30, 10, 45]
# Osuuksien etäisyys keskustasta.
explode = (0.1, 0, 0, 0)
# Sektoridiagrammi.
pie(osuudet, explode=explode, labels=labels, autopct="%1.1f%%",
   shadow=True)
savefig("sektoridiagrammi.png") # Tallentaa kuvan solun yhteyteen
```



Kuva 28: Pylväsdiagrammi esimerkin aineistosta.



Kuva 29: Sektoridiagrammi.

6.3 Kaksi kvantitatiivista tilastollista muuttujaa

Tarkastellaan tilannetta, jossa tilastoyksiköihin liityy kaksi kvantitatiivista tilastollista muuttujaa X ja Y. Havaintoaineisto muodostuu n:stä arvoparista

$$(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n).$$

Tutkitaan, muuttuvatko muuttujan Y arvot lineaarisesti, kun muuttujan X arvot kasvavat. Tätä varten piirretään havaintoaineistosta sirontakuvio ja sovitetaan siihen lineaarinen regressiomalli eli regressiosuora pienimmän neliösumman menetelmällä. Havaintojen välistä lineaarista riippuvuutta kuvaamaan lasketaan korrelaatiokerroin ja siihen liittyvä selitysaste.

6.3.1 Regressiosuora

Tarkastellaan seuraavaa aineistoa ja piirretään sen perusteella regressiosuora pienimmän neliösumman menetelmällä.

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14
y	6,5	8,3	11,7	15,9	17,5	24,6	26,6	28,7	31,9	34,9	40,4	38,3	45,4	48,5

Taulukko 15: Tarkasteltava havaintoaineisto.

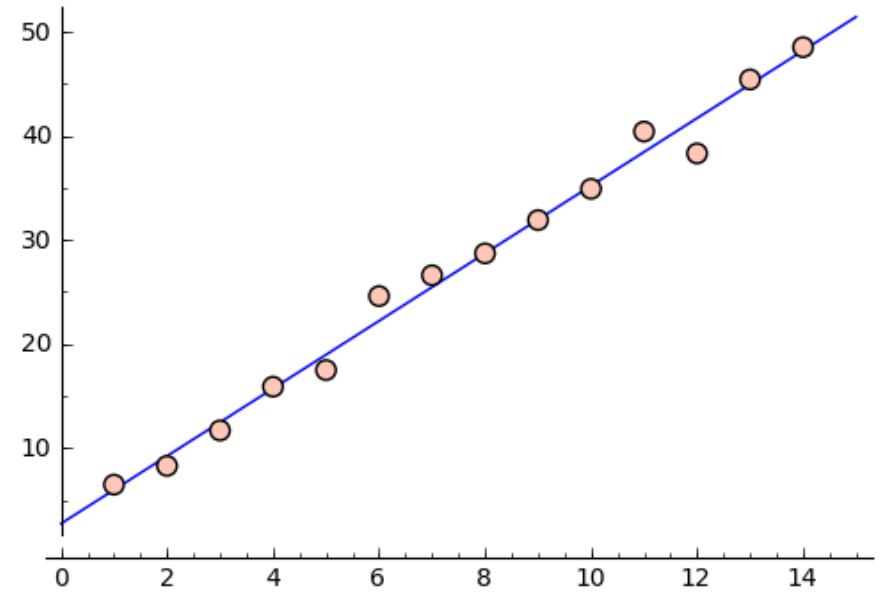
Sirontakuvio voidaan piirtää metodilla *scatter_plot(lista)*. [74] Metodi saa syötteenään listan, joka koostuu piirrettävistä pisteistä (x, y). Pisteiden muotoa, kokoa, väriä ja läpinäkyvyyden tasoa voidaan säätää parametreilla *marker*, *markersize*, *facecolor*, *edgecolor* ja *alpha*.

Regressiosuoran piirtämistä varten ladataan *scipy*-paketista moduuli *stats*. [75] Metodi *stats.linregress(lista)* palauttaa syötteenä saadun pisteistä koostuvan listan perusteella regressiosuoran kulmakertoimen ja vakion sekä korrelaatiokertoimen, kaksitahoisen t-testin p-arvon ja estimaatin keskivirheen.

Sagen ulkopuolisten pakettien metodit eivät välttämättä osaa käsitellä Sagen tietotyyppejä. Tästä syystä seuraavassa ohjelmakoodissa Pythonin *map*-metodin avulla muutetaan kaikki listan v koordinaattipisteiden arvot *float*-tyyppisiksi. Komento *Map(funktio, iteroitava objekti)* suorittaa annetun funktion jokaiselle iteroitavan objektin alkiolle ja palauttaa tuloksen listan muodossa.

Regressiosuoran sovittamiseksi annettuun havaintoaineistoon kirjoitetaan seuraava ohjelma:

```
from scipy import stats
# Havaintoaineisto:
v = [(1, 6.5), (2, 8.3), (3, 11.7), (4, 15.9), (5, 17.5), (6, 24.6),
   (7, 26.6), (8, 28.7), (9, 31.9), (10, 34.9), (11, 40.4), (12,
   38.3), (13, 45.4), (14, 48.5)]
# Lasketaan regressiosuoran tunnusluvut.
slope, intercept, r, p, stderr = stats.linregress([map(float, xy) for
   xy in v])
# Piirretään regressiosuora ja sirontakuvio sekä esitetään tunnusluvut
regressiosuora = plot(slope*x + intercept, (x,0,15))
sirontakuvio = scatter_plot(v, markersize=50, marker="o",
   facecolor="#fec7b8", edgecolor="black")
show(regressiosuora + sirontakuvio)
html("$\\text{Regressiosuora: } %sx+%s$" %(N(slope,digits=3),
   N(intercept,digits=3)))
html("$\\text{Korrelaatiokerroin: R = } %s$" %(N(r,digits=3)))
html("$\\text{Selitysaste: } R^2 = %s \%%$" %(N(r,digits=3)^2*100.0))
```



Regressiosuora: $3.24x + 2.76$
Korrelaatiokerroin: R = 0.995
Selitysaste: $R^2 = 98.9\%$

Kuva 30: Havaintoaineistoon sovitettu regressiosuora ja siihen liittyvät tunnusluvut.

6.4 Satunnaislukuja

Satunnaismuuttujia voidaan käyttää todennäköisyyslaskennan käsitteiden ja teoreemien havainnollistamiseen. Tässä luvussa esitellään kaksi simulaatiota, joista ensimmäinen demonstroi satunnaisuuden luonnetta kolikon heiton avulla. Jälkimmäinen liittyy tilanteeseen, jossa heitetään noppia ja lasketaan niiden silmälukujen summa. Sovellus havainnollistaa hyvin, miten tällaisen summamuuttujan jakauma on normaalisti jakautunut keskeisen raja-arvolauseen mukaisesti.

Seuraavassa taulukossa esitellään Pythonin satunnaislukuihin erikoistuneen *random*-moduulin yleisimpiä metodeja:

Metodi	**Kuvaus**
random()	Palauttaa satunnaisen desimaaliluvun väliltä [0,1)
randint(a,b)	Palauttaa satunnaisen kokonaisluvun väliltä [a,b]
uniform(a,b)	Palauttaa satunnaisen desimaaliluvun väliltä [a,b]
choice(lista)	Palauttaa syötteenä saadun listan satunnaisen alkion
shuffle(lista)	Sekoittaa satunnaisesti syötteenä annetun listan alkiot
normalvariate($\bar{x}$, σ^2)	Palauttaa satunnaisen normaalijakaumaa N($\bar{x}$, σ^2) noudattavan satunnaismuuttujan arvon

Taulukko 16: Satunnaislukuihin erikoistuneen *random*-moduulin metodeja.

Metodien *uniform* ja *normalvariate* lisäksi *random*-luokka sisältää useita jatkuvia jakaumia. [76] Sagessa on luokkakirjasto niin sanotun todennäköisyysavaruuden konstruoimiseen. Todennäköisyysavaruuden avulla voidaan käsitellä diskreettejä todennäköisyysmuuttujia ja laskea erilaisia tunnuslukuja, kuten muuttujien odotusarvoja ja variansseja sekä tarkastella erilaisia jakaumia. [77] Useimmissa tapauksissa Pythonin *random*-moduuli on kuitenkin ominaisuuksiltaan riittävä.

6.4.1 Esimerkki: Havaintoaineiston generointi

Generoidaan havaintoaineisto, jota voidaan käyttää lineaarisen regressiomallin tutkimista varten. Pisteet valitaan siten, että tasaisin muuttujan x välein määritetään funktion $f(x)$ arvo ja poikkeutetaan sitä satunnaisen luvun verran.

Tarkastellaan funktiota $f(x) = 3x + 5$ välillä $[1, 15]$. Valitaan pisteet yhden yksikön välein siten, että funktion arvoon lisätään kyseisessä pisteessä luku väliltä $[-3, 3]$. Esitetään koordinaatit neljän merkitsevän numeron tarkkuudella.

```
import random
f(x) = 3*x + 5
v = []
for i in range(1, 15):
   v.append((i, N(f(i) + random.uniform(-3, 3), digits=4)))
print v
```

```
[(1, 10.31), (2, 13.29), (3, 16.40), (4, 14.73), (5, 20.61), (6,
21.94), (7, 26.99), (8, 29.91), (9, 29.85), (10, 32.31), (11,
40.96), (12, 38.10), (13, 43.84), (14, 44.42)]
```

6.4.2 Esimerkki: Lottonumeroiden arvonta

Arvotaan seitsemän numeroa ja kolme lisänumeroa väliltä 1-39. Suoritetaan arvonta sekoittamalla ensin lottonumeroista koostuva lista, jonka jälkeen listan alkupäästä leikataan varsinaiset lottonumerot ja lisänumerot. Arvotut numerot ja lisänumerot järjestetään kasvavaan järjestykseen ja tulostetaan.

```
v = [1, 2, .. , 39]
shuffle(v)
numerot, lisanumerot = v[:7], v[7:10]
numerot.sort()
lisanumerot.sort()
print "Arvotut numerot: ", numerot
print "Arvotut lisänumerot: ", lisanumerot
```

```
Arvotut numerot: [9, 12, 20, 22, 24, 27, 35]
Arvotut lisänumerot: [11, 16, 39]
```

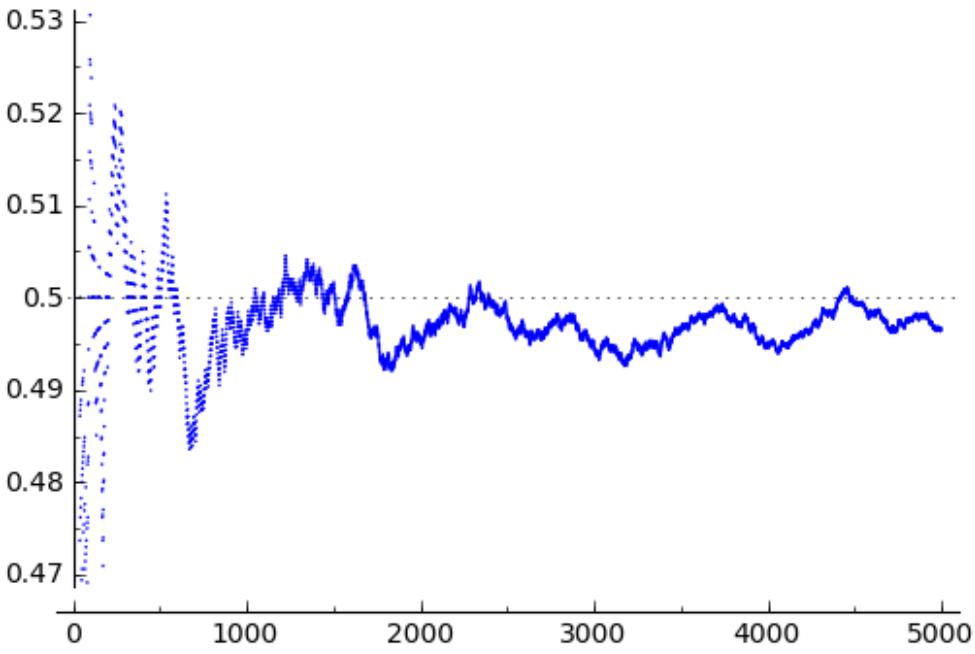
6.5 Simulaatio: Kolikon heitto

Heitetään kolikkoa toistuvasti ja lasketaan jokaisen heiton jälkeen kruunujen lukumäärän suhde kaikkien heittojen määrään. Kun kolikkoa on heitetty 5000 kertaa, esitetään suhteellisen osuuden kehitys kuvaajassa, jossa vaaka-akselilla on heiton järjestysnumero ja pystyakselilla kruunujen suhteellinen osuus kaikista siihen asti heitetyistä kolikoista.

Kuviosta (kuva 31) havaitaan, miten osuus aluksi heittelehtii puolelta toiselle, kunnes se alkaa tasoittua lähelle raja-arvoa 50 %. Kuvio on jossain määrin tyypillinen siinä mielessä, että suhteellisella osuudella on taipumusta pysytellä toisinaan yllättävänkin kauan raja-arvon toisella puolella sen sijaan, että se vaihtelisi usein raja-arvon kummallakin puolella. Tämä ilmiö voidaan todeta ajamalla sovellus useita kertoja ja vertaamalla käyrien muotoa. [78]

Ohjelma 11: Simulaatio: Kolikon heitto

```
import random
pisteet = []
k = 0
for n in range(1, 5000):
   k += random.randint(0, 1)
   pisteet.append((n, k/n))
point(pisteet, ymin=0.47, ymax=0.53, gridlines=[None, [0.5]], pointsize=1)
```



Kuva 31: Kolikon heitto -simulaatio. Kuvaajassa x-akselilla on heiton järjestysnumero ja pystyakselilla kruunujen suhteellinen osuus kaikista siihen asti heitetyistä kolikoista.

6.6 Simulaatio: Noppien heiton summa

Simulaatiossa heitetään k kertaa s-sivuista noppaa ja lasketaan noppien näyttämien silmälukujen summa. Yhden nopan heiton tulos X on diskreetti satunnaismuuttuja, joka noudattaa tasaista jakaumaa. Silmäluvun odotusarvo on

$$E(X) = \sum_{x=1}^{s} xP(X = x) = \frac{1}{s} \sum_{x=1}^{s} x.$$

Satunnaismuuttujan X varianssi saadaan kaavasta

$$Var(X) = E(X^2) - E(X)^2 = \frac{1}{s} \sum_{x=1}^{s} x^2 - \left(\frac{1}{s} \sum_{x=1}^{s} x \right)^2.$$

Heittotulosten summa,

$$Z = \sum_{i=1}^{k} X_i,$$

on diskreetti satunnaismuuttuja, joka on riippumattomien, samaa jakaumaa noudattavien satunnaismuuttujien X_i $(i = 1, 2, ..., k)$ summa. Näin ollen summan Z odotusarvoksi saadaan

$$E(Z) = E(\sum_{i=1}^{k} X_i) = \sum_{i=1}^{k} E(X_i) = kE(X).$$

Vastaavasti summan varianssille saadaan

$$Var(Z) = kVar(X).$$

Toistetaan koetta n kertaa ja lasketaan saatujen havaintojen suhteelliset frekvenssit. Esitetään jakauma graafisesti sekä lasketaan satunnaismuuttujan odotusarvo, varianssi ja keskihajonta. Kuvaajassa näkyy erityisen hyvin keskeisen raja-arvolauseen vaikutus, kun heittojen lukumäärää kasvatetaan: jakauma lähestyy normaalijakaumaa. Ilmiötä voidaan korostaa piirtämällä pylväsdiagrammin kanssa samaan koordinaatistoon normaalijakauman tiheysfunktio

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

missä $\mu = E(Z)$ on jakauman odotusarvo ja $\sigma^2 = Var(Z)$ varianssi. Pylväsdiagrammi piirretään koordinaatistoon käyttäen luokan *sage.plot* metodia *BarChart* [79].

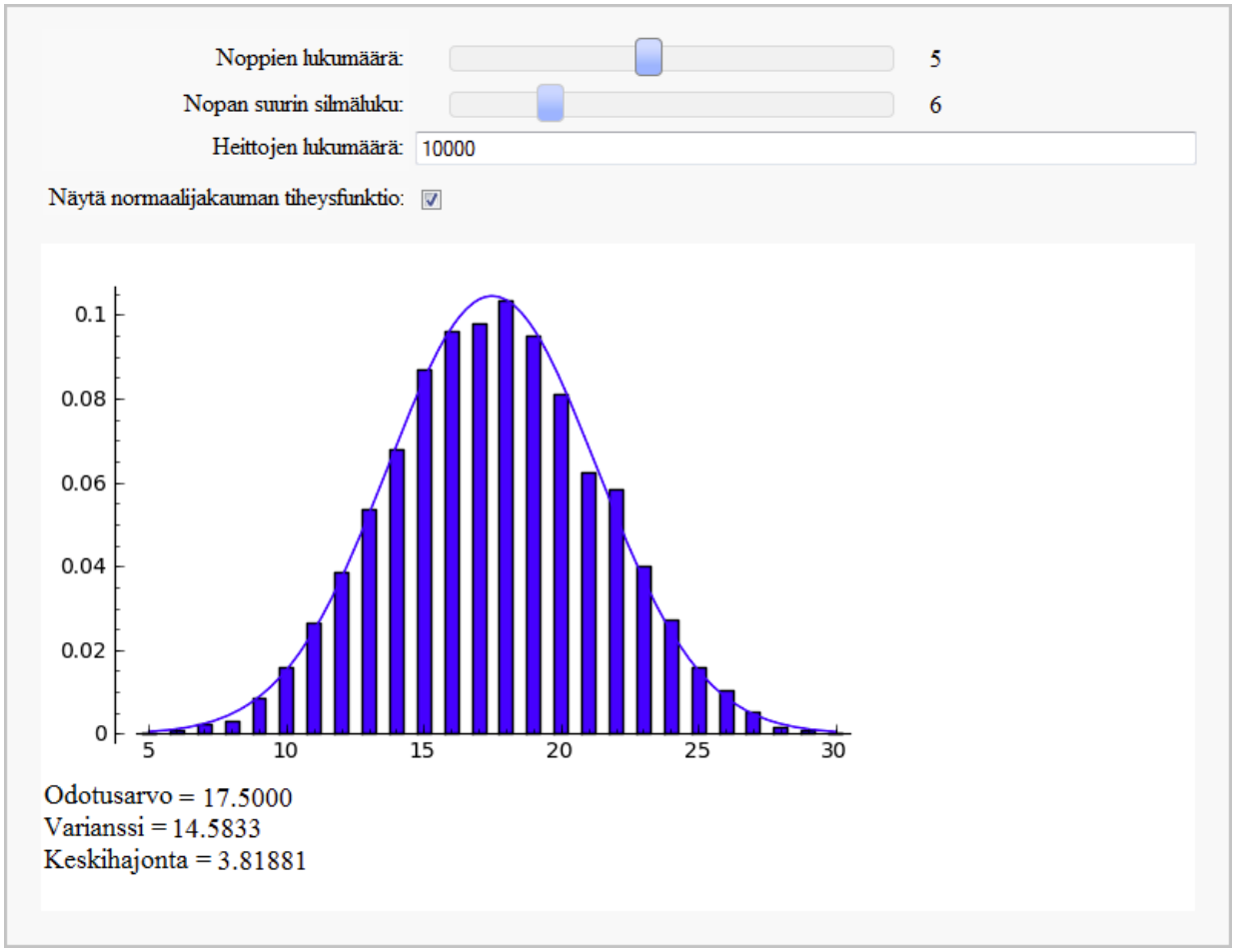
Ohjelma on toteutettu interaktiivisena sovelluksena, missä noppien lukumäärän, suurimman silmäluvun ja heittojen lukumäärän voi vapaasti valita. Normaalijakauman tiheysfunktion saa halutessaan näkyviin kytkimestä.

Ohjelman avulla voidaan tutkia odotusarvon ja varianssin käsitettä sekä keskeistä raja-arvolausetta ja normaalijakaumaa. Se havainnollistaa, miten satunnaismuuttujan todennäköisyysjakauma muodostuu eri parametrien arvoilla.

Ohjelma 12: Simulaatio: Noppien heitto

```
import random
from sage.plot.bar_chart import BarChart

@interact
def noppien_heitto(k = slider(1, 10, 1, default=3, label="Noppien lukumäärä:"),
                       s = slider(2, 20, 1, default = 6, label="Nopan suurin
                           silmäluku:"),
                       n = input_box(default=10000, label="Heittojen lukumäärä:"),
                           normaalijakauma = ("Näytä normaalijakauman
                           tiheysfunktio:", false)):
   # Arvotaan noppien summa ja lisätään se avain-arvo -pareista koostuvaan
       hakemistoon "havainnot". Hakemiston avaimia ovat summat ja avaimia vastaavat
       arvot ovat summaa vastaavien havaintojen lukumäärät.
   havainnot = {}
   for i in range(n):
      summa = 0
      for j in range(k):
         summa += random.randint(1, s)
      if summa not in havainnot:
         havainnot[summa] = 0
      havainnot[summa] += 1

   # Luodaan indeksilista mahdollisista summien arvoista ja lasketaan havaintojen
       suhteelliset osuudet.
   indeksit = range(k, s*k+1)
   suhteelliset_osuudet = [havainnot.get(i,0)/n for i in indeksit]

   # Pylväsdiagrammia varten poikkeutetaan indeksien arvoja -0.25, jolloin pylvään
       keskikohta osuu indeksin kohdalle, kun pylvään leveys on 0.5.
   poikkeutetut_indeksit = [c - 0.25 for c in indeksit]

   # Piirretään pylväskuvio.
   G = Graphics()
   G.add_primitive(BarChart(poikkeutetut_indeksit, suhteelliset_osuudet,
       {"width":0.5, "rgbcolor":(0,0,1), "legend_label":0}))
```

```
# Määritellään normaalijakauman tiheysfunktio ja lasketaan satunnaismuuttujan
    odotusarvo ja varianssi.
normaalijakauman_tiheysfunktio(E,V,x) = 1/sqrt(2*pi*V)*e^(-(x-E)^2/(2*V))
odotusarvo = sum([1, 2, .. , s])/s*k
i = var("i")
varianssi = (sum(i^2, i, 1, s)/s - (sum([1, 2, .. , s])/s)^2)*k

# Piirretään normaalijakauma.
if (normaalijakauma):
   G += plot(normaalijakauman_tiheysfunktio(odotusarvo, varianssi, x), xmin=k,
       xmax=k*s)

# Esitetään kuva ja satunnaismuuttujan tunnusluvut.
G.show(xmin=k, xmax=s*k, ymin=0)
html("$\\text{Odotusarvo = %s}$" %(N(odotusarvo, digits=6)))
html("$\\text{Varianssi = %s}$" %(N(varianssi, digits=6)))
html("$\\text{Keskihajonta = %s}$" %(N(sqrt(varianssi), digits=6)))
```



Kuva 32: Noppien heitto -simulaatio.

6.7 Tehtäviä

Tehtävä 6-1: Opiskelijat saivat seuraavia arvosanoja matematiikan kokeesta, kun kokeet laitettiin aakkosjärjestykseen oppilaiden nimien perustella: [8, 7, 5, 7, 8, 8, 9, 10, 4, 7, 7, 8, 4, 6, 10, 9, 5, 8, 9, 8, 7, 5, 6, 7, 9]. Tee lista arvosanoista ja järjestä se kasvavaan järjestykseen. Tämän jälkeen määritä Sagen avulla arvosanojen keskiarvo, mediaani, moodi, varianssi ja keskihajonta.

Tehtävä 6-2: *(Perustuu osittain yo-K08l:13)* Alla olevassa taulukossa on kevään 2007 lyhyen matematiikan ylioppilaskokeen tehtävän 8 pistejakauma. Laadi tuloksista pylväsdiagrammi, jossa vaaka-akselilla ovat pisteet ja pylväiden korkeus osoittaa kyseisen pistemäärän saaneiden lukumäärän. Laadi myös vastaava sektoridiagrammi, josta ilmenee kunkin pistemäärän saaneiden suhteellinen osuus, ja ilmoita osuudet prosentteina.

Pistemäärä	0	1	2	3	4	5	6
Frekvenssi	2025	1245	2186	792	673	908	1471

Tehtävä 6-3: Tarkastellaan kahta havaintoaineistoa A ja B, jotka muodostuvat (x, y)-arvopareista:

x	1	2	3	4	5	6	7	8	9	10
y_A	18,5	15,4	16,0	12,2	8,6	8,58	6,20	5,34	3,77	0,10
y_B	2,96	1,21	7,79	14,5	25,4	56,3	108	208	401	788

Piirrä havainnoista sirontakuviot ja sovita aineistoihin regressiosuora pienimmän neliösumman menetelmällä. Voit käyttää tässä apuna luvun 6.3.1 esimerkin lähdekoodia. Vertaile regressiomallien korrelaatiokertoimia ja selitysasteita. Kumpaan aineistoon lineaarinen malli sopii paremmin? Jos lineaarinen malli kuvaa heikosti riippuvuussuhdetta, minkä tyyppinen malli sopisi paremmin?

Tehtävä 6-4: Tutkitaan satunnaismuuttujan jakaumaa noppien heittoa tarkastelevan ohjelman 12 (luvussa 6.6) avulla.

a) Heitetään viisi kappaletta 6-sivuisia noppia ja lasketaan silmälukujen summa. Onko noppien silmälukujen summa normaalisti jakautunut? Määrää summan odotusarvo, varianssi ja keskihajonta.

b) Jos viiden nopan sijasta heitetäänkin 10 kappaletta 6-sivuisia noppia, mikä on silmälukujen summan odotusarvo, varianssi ja keskihajonta. Vertaa vastausta edellisen tehtävän tuloksiin.

7 Numeerisia menetelmiä

Matemaattinen ongelma pyritään yleensä ratkaisemaan analyyttisesti, eli käsittelemään lausekkeita symbolisesti siten, että myös vastaus voidaan antaa tarkasti lausekkeen muodossa. On kuitenkin olemassa ongelmia, joihin ei edes teoriassa ole mahdollista löytää ratkaisua analyyttisesti. Tällöin tehtävää voidaan lähestyä numeerisesti ja pyrkiä etsimään vastaukselle likimääräinen arvo.

Tässä luvussa esitellään erilaisia Sagella toteutettuja numeerisia menetelmiä, joiden avulla voidaan tutkia funktion nollakohtia, integroida numeerisesti, approksimoida funktiota Taylorin polynomien avulla ja ratkaista differentiaaliyhtälöitä likimääräisesti. Ohjelmien laadinnassa on kiinnitetty erityistä huomiota algoritmien toimintaa selventävään grafiikkaan ja muihin havainnollistuksiin, joita voidaan käyttää opetuksen tukena. Luvun lopussa esitellään lyhyesti Pythonin lineaarialgebraan erikoistunut laajennuskirjasto *NumPy*.

7.1 Nollakohtien määrittäminen numeerisesti

Funktion nollakohtien likiarvoja voidaan määrittää laskemalla funktion arvoja nollakohdan ympäristössä erilaisilla tarkentuvilla menetelmillä. Yhtälön ratkaiseminen voidaan aina palauttaa nollakohtien määrittämiseksi siirtämällä kaikki yhtälössä esiintyvät termit yhtälön toiselle puolelle. Nollakohtien määrittäminen on tarpeellista myös esimerkiksi funktion ääriarvokohtien etsinnässä, kun tutkitaan derivaattafunktion nollakohtia. Seuraavassa esitetään puolitusmenetelmä ja sen muunnelma sekanttimenetelmä sekä Newtonin menetelmä, joka perustuu derivaatan laskemiseen.

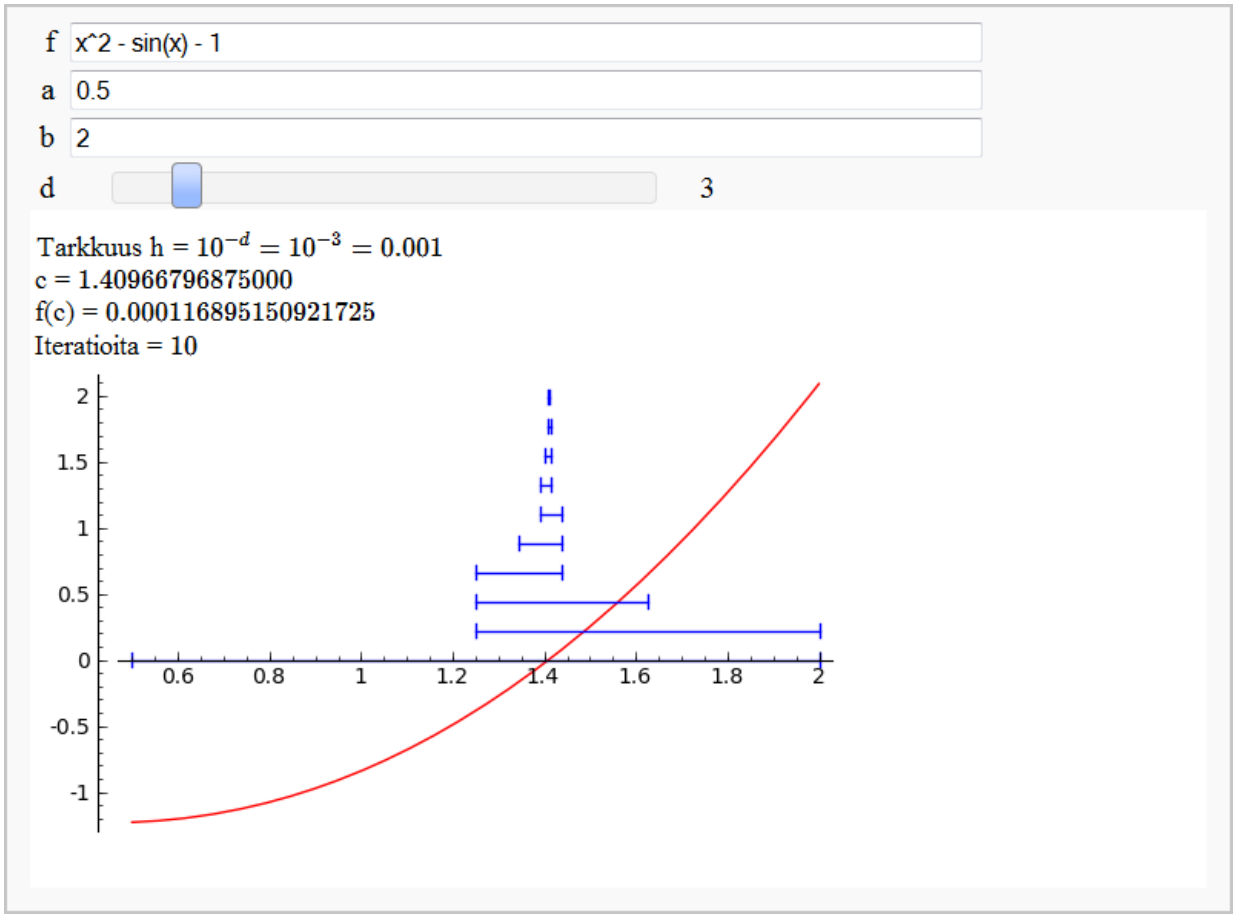
7.1.1 Puolitusmenetelmä

Puolitusmenetelmä perustuu *Bolzanon lauseeseen*: jos suljetulla välillä jatkuvan funktion arvot välin päätepisteissä ovat erimerkkiset, funktiolla on tällä välillä (ainakin yksi) nollakohta. Puolitusmenetelmässä nollakohdan sijaintia tarkennetaan valitsemalla uuden välin yhdeksi päätepisteeksi aina tarkasteluvälin keskipiste. Toinen päätepiste valitaan siten, että funktion arvot välin päätepisteissä ovat jälleen erimerkkiset. Näin jatketaan, kunnes nollakohta on määritetty halutulla tarkkuudella.

Alla olevassa ohjelmassa puolitusmenetelmä on toteutettu Sagen avulla. Ennen algoritmin käyttöä tulee etsiä väli, jonka päätepisteissä funktiolla on erimerkkiset arvot. Sopiva väli voidaan löytää funktion kuvaajasta. Metodi *puolitusmenetelma* saa syötteenään

funktion f ja välin päätepisteet a ja b. Muuttuja h merkitsee tarkkuutta, jolla nollakohta määrätään. Tarkkuus annetaan muodossa $h = 10^{-d}$, missä d on käyttäjän valittavissa säätimellä. Algoritmin suoritus pysähtyy, kun funktion arvo on pienempi kuin annettu tarkkuus, eli $f(c) < h$, missä c on haarukoitu nollakohdan estimaatti. Metodi palauttaa pisteen c sekä listan algoritmin suorituksen aikana tutkituista väleistä.

Ohjelma esittää tarkastellut välit piirrettynä samaan kuvaan kuvaajan kanssa. Lisäksi ohjelma tulostaa ratkaistun nollakohdan c, funktion arvon tässä pisteessä sekä iterointikierrosten lukumäärän. Ohjelma perustuu William Steinin kirjoittamaan sovellukseen [80].



Kuva 33: Puolitusmenetelmän toimintaa havainnollistava sovellus.

Ohjelma 13: Puolitusmenetelmä

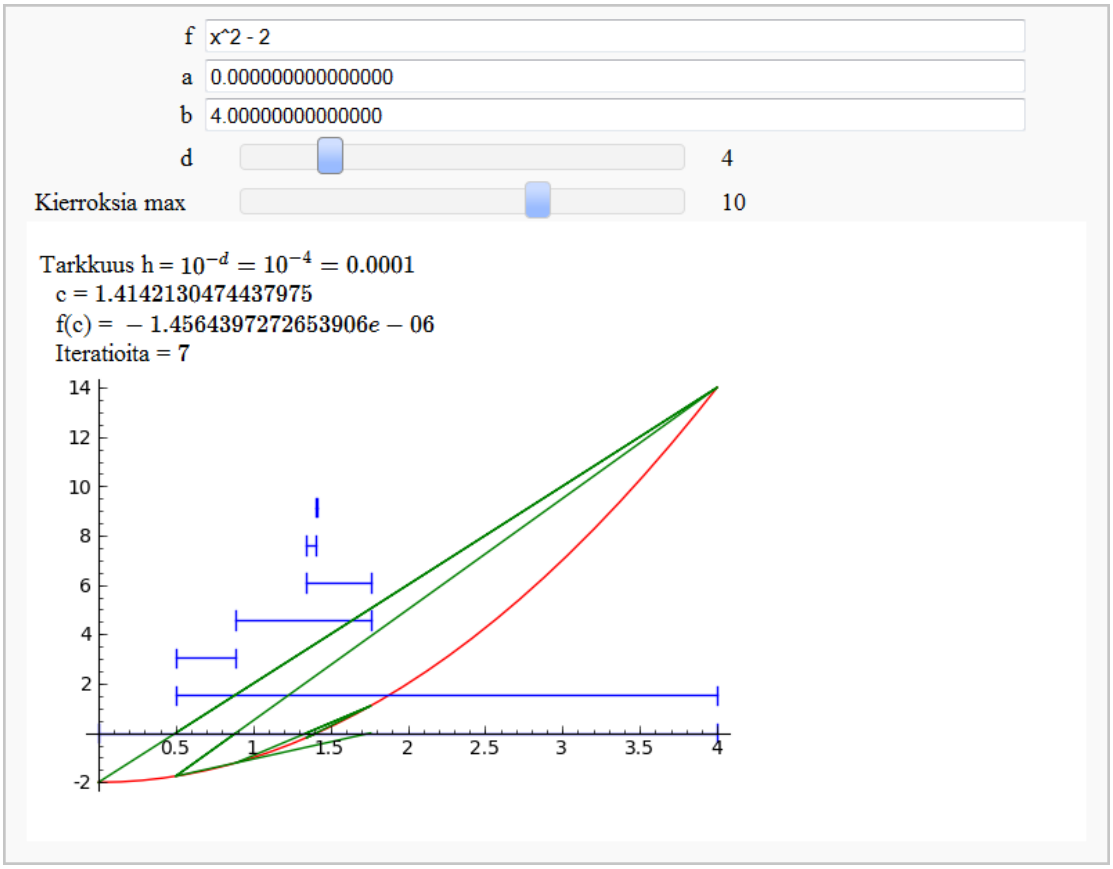
```
def puolitusmenetelma(f, a, b, h):
    f(x) = f
    intervallit = [(a, b)]
    while True:
        c = (a + b)/2.0
        fa = f(a); fb = f(b); fc = f(c)
        if abs(fc) < h: break
        if fa*fc < 0:
            a, b = a, c
        elif fc*fb < 0:
            a, b = c, b
        else:
            # Nostaa virheen, jos päätepisteet ovat saman merkkisiä.
            raise ValueError, "Päätepisteiden tulee olla erimerkkisiä."
        intervallit.append((a, b))
    return c, intervallit

@interact
def _(f = cos(x) - x, a = 0.0, b = 1.0, d = slider(1, 16, 1, 3) ):
    f(x) = f
    h = 10^(-d)
    # Kokeilee puolitusmenetelmän suorittamista.
    try:
        c, intervallit = puolitusmenetelma(f, float(a), float(b), float(h))
    # Jos menetelmän suoritus epäonnistuu ValueError:n takia, tulostaa
        virheilmoituksen ja kuvaajan uuden välin määrittämiseksi.
    except ValueError:
        print "Päätepisteiden tulee olla erimerkkisiä."
        show(plot(f, a, b, color="red"), xmin=a, xmax=b)
    else:
        html("$\\text{Tarkkuus = }10^{-d}=10^{-%s}=%s$" %(d, float(h)))
        html("$\\text{c = }%s$" %c)
        html("$\\text{f(c) = }%s" %f(c))
        html("$\\text{Iteratioita = }%s" %len(intervallit))
        # Piirretään algoritmin toimintaa kuvaava grafiikka.
        P = plot(f, a, b, color="red")
        k = (P.ymax() - P.ymin())/ (1.5*len(intervallit))
        L = sum(line([(c, k*i), (d, k*i)]) for i, (c, d) in enumerate(intervallit) )
        L += sum(line([(c, k*i - k/4), (c, k*i + k/4)]) for i, (c,d) in
            enumerate(intervallit) )
        L += sum(line([(d, k*i - k/4), (d, k*i + k/4)]) for i, (c,d) in
            enumerate(intervallit) )
        show(P + L, xmin=a, xmax=b)
```

7.1.2 Sekanttimenetelmä

Puolitusmenetelmä ei ole kovinkaan tehokas, sillä se hyödyntää lasketuista funktion arvoista vain tietoa funktion etumerkistä. Sekanttimenetelmän avulla päästään yleensä parempaan tulokseen, sillä se perustuu lineaarisen interpolointiin. Menetelmässä tarkasteltavan välin puolittamisen sijaan piirretään viiva pisteiden $(a, f(a))$ ja $(b, f(b))$ kautta. Välin uudeksi päätepisteeksi valitaan suoran ja x-akselin leikkauskohta c. Leikkauspiste voidaan esittää välin päätepisteiden ja funktion arvojen avulla seuraavasti:

$$c = b - \frac{b-a}{f(b)-f(a)} \cdot f(b).$$

Toinen päätepiste voidaan valita siten, että uuden välin päätepisteet ovat erimerkkisiä, kuten puolitusmenetelmässä tehtiin. Toinen vaihtoehto, jota seuraavaksi esitettävä algoritmi käyttää, on ajatella välien päätepisteiden muodostavan jonon: kolme jonon ensimmäistä jäsentä ovat a, b ja c, jonka jälkeen seuraava piste d lasketaan välillä, jonka päätepisteet ovat kaksi edellistä pistettä b ja c. Näin jatketaan, kunnes nollakohta on saatu lasketuksi halutulla tarkkuudella.



Kuva 34: Sekanttimenetelmän toimintaa havainnollistava sovellus.

Sekanttimenetelmä ei välttämättä löydä nollakohtaa tai sen tuottama ratkaisu voi olla alkuperäisen tarkasteluvälin ulkopuolella. Ikuisen silmukan välttämiseksi algoritmille annetaan syötteenä enimmäismäärä kierroksia (*maxn*), joiden jälkeen algoritmin suoritus viimeistään pysähtyy ja ohjelma palauttaa viimeksi lasketun pisteen. Jos käyttäjä valitsee parametrin *maxn* arvoksi nollan, ohjelma suorittaa algoritmia loputtomasti tai kunnes nollakohta löydetään halutulla tarkkuudella. Laskenta keskeytyy myös silloin, jos funktion arvot päätepisteissä ovat yhtä suuret, eli $f(b) - f(a) = 0$. Ohjelma antaa virheilmoituksen nollalla jaettaessa.

Algoritmin toimintaa havainnollistetaan ohjelmassa piirtämällä interpolaatiosuoraa kuvaava jana jokaisen kierroksen jälkeen pisteiden $(a, f(a))$ ja $(b, f(b))$ välille. Kuvaajasta havaitaan, miten seuraavan välin päätepiste valitaan suoran ja x-akselin leikkauspisteen perusteella. Jos välin päätepisteet ovat samanmerkkisiä, janaa jatketaan suoran ja x-akselin leikkauspisteeseen asti. (Kuva 34.)

Ohjelma 14: Sekanttimenetelmä

```
def sekanttimenetelma(f, a, b, maxn, h):
    f(x) = f
    intervallit = [(a,b)]
    kierros = 1
    while True:
        c = b-(b-a)*f(b)/(f(b)-f(a))
        if abs(f(c)) < h or kierros == maxn:
            break
        a, b = b, c
        intervallit.append((a,b))
        kierros += 1
    return c, intervallit

@interact
def _(f = x^2 - 2, a = 0.0, b = 4.0, d = slider(1, 16, 1, 3),
        maxn = slider(0, 15, 1, 10, label="Kierroksia max") ):
    f(x) = f
    h = 10^(-d)
    c, intervallit = sekanttimenetelma(f, float(a), float(b), maxn, h)
    ... # (kts. puolitusmenetelmä)

    # Piirretään algoritmin toimintaa kuvaava grafiikka.
    S = sum(line([(c, f(c)), (d, f(d)), (d-(d-c)*f(d)/(f(d)-f(c)), 0)],
        color="green") for (c, d) in intervallit)
    show(P + L + S, xmin=a, xmax=b)
```

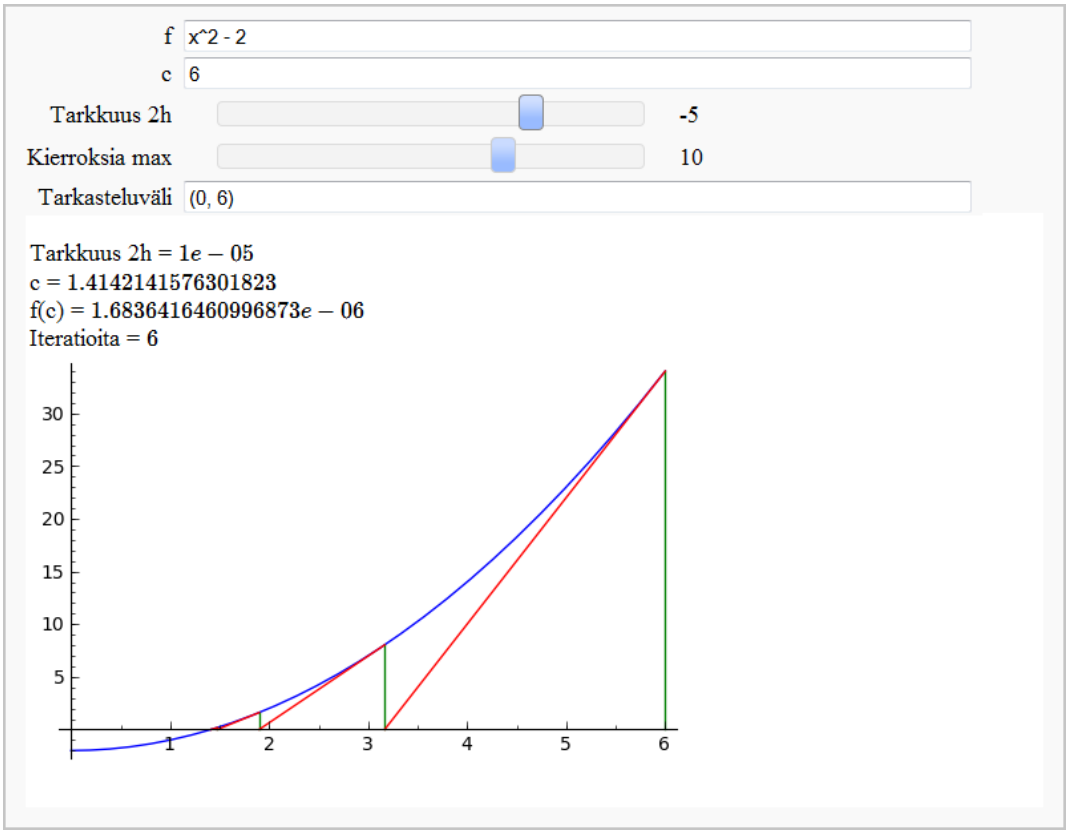
7.1.3 Newtonin menetelmä

Sekanttimenetelmässä käytetty lauseke $\frac{b-a}{f(b)-f(a)}$ voidaan tulkita funktion f erotusosamäärän käänteisluvuksi. Jos f on derivoituva ja a on lähellä pistettä b, erotusosamäärä vastaa likimäärin funktion derivaattaa pisteessä b. Newtonin menetelmässä piste c lasketaan kuten sekanttimenetelmässä, mutta korvaamalla erotusosamäärä derivaatalla:

$$c = b - \frac{f(b)}{f'(b)}.$$

Laskennassa edellytetään, että $f'(b) \neq 0$. Seuraava piste lasketaan aina edellisen perusteella, kunnes nollakohta löydetään halutulla tarkkuudella. Metodi *newtonin_menetelma* saa syötteenä funktion, aloituspisteen, tarkkuuden ja iteraatioiden enimmäismäärän. Algoritmi käyttää tarkkuutena arvoa $2h$: ohjelma pysähtyy, kun tulo $f(c-h)f(c+h)$ saa negatiivisen arvon, toisin sanoen funktion arvot pisteen c lähellä ovat erimerkkiset. Tällöin funktion nollakohta sijaitsee välillä $[c-h, c+h]$.

Kuten sekanttimenetelmä, myös Newtonin menetelmä voi jäädä ikuiseen silmukkaan tai nollakohtaa ei löydetä. Lisäksi laskenta keskeytyy, jos algoritmin toiminnan aikana derivaatta saa arvon nolla. Metodi palauttaa viimeisen iteraatiopisteen c sekä kaikki lasketut välipisteet.



Kuva 35: Newtonin menetelmän toimintaa demonstroiva sovellus.

Seuraavassa ohjelmassa Newtonin algoritmin etenemistä kuvataan piirtämällä funktion tangenttisuoran suuntainen jana tarkastelupisteestä x-akselille. Janan ja vaaka-akselin leikkauspiste määrää seuraavan tarkastelupisteen paikan. (Kuva 35.)

Ohjelma 15: Newtonin menetelmä

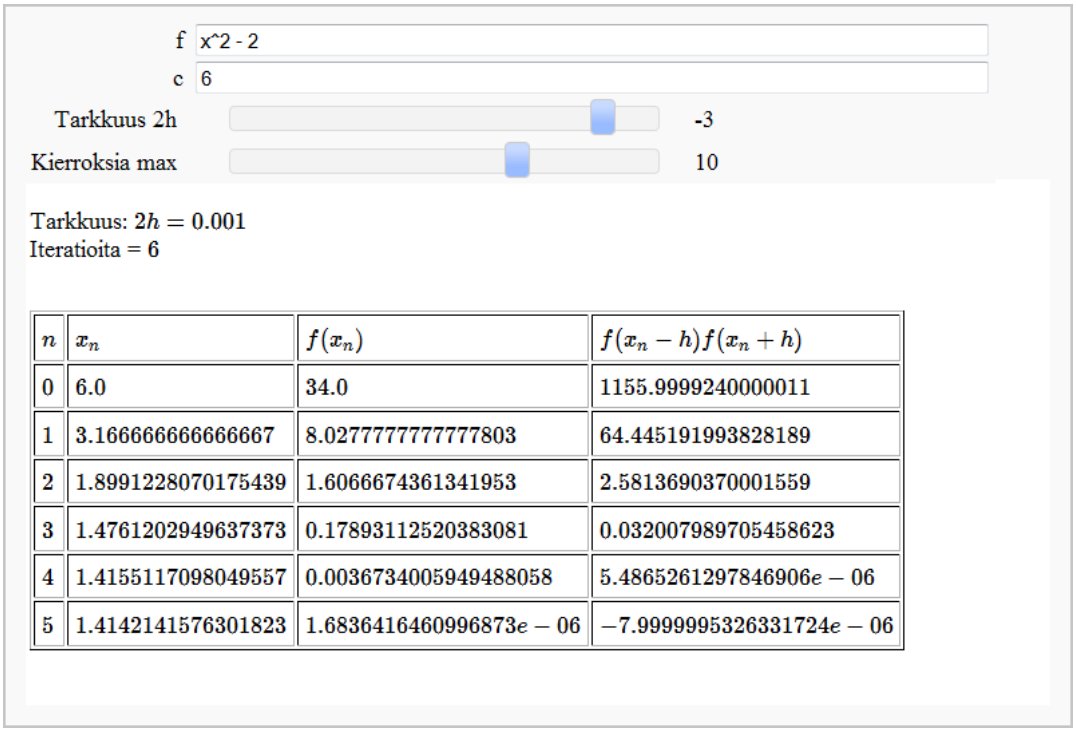
```
def newtonin_menetelma(f, c, maxn, h):
   f(x) = f
   valipisteet = [c]
   kierros = 1
   while True:
      c = c - f(c)/derivative(f(x))(x=c)
      valipisteet.append(c)
      if f(c-h)*f(c+h) < 0 or kierros == maxn:
         break
      kierros += 1
   return c, valipisteet

@interact
def _(f = x^2 - 2,
      c = 6,
      hh = slider(-16, -1, 1, -3, label="Tarkkuus 2h"),
      maxn = slider(0, 15, 1, 10, label="Kierroksia max"),
      tarkasteluvali = input_box(default = (0,6), label="Tarkasteluväli")):
   f(x) = f
   h = 10^hh
   c, valipisteet = newtonin_menetelma(f, float(c), maxn, h/2.0)
   html("$\\text{Tarkkuus = }%s$" %float(h))
   html("$\\text{c = }%s$" %c)
   html("$\\text{f(c) = }%s" %f(c))
   html("$\\text{Iteratioita = }%s" %len(valipisteet))

   # Piirretään algoritmin toimintaa kuvaava grafiikka.
   P = plot(f, x, tarkasteluvali, color="blue")
   L = sum(line([(c, 0), (c, f(c))], color="green") for c in valipisteet[:-1])
   for i in range(len(valipisteet) - 1):
      L += line([(valipisteet[i], f(valipisteet[i])), (valipisteet[i+1], 0)],
          color="red")
   show(P + L, xmin=tarkasteluvali[0], xmax=tarkasteluvali[1], ymin=P.ymin(),
       ymax=P.ymax())
```

Menetelmän toimintaa voidaan tarkastella myös tulostamalla algoritmin suoritukseen liittyvien muuttujien arvot jokaisen iteraatiokierroksen jälkeen. Seuraava ohjelma tulostaa *HTML*-taulukon, jossa on listattuna iteraatiokierroksen järjestysnumero n, piste

x_n, funktion arvo $f(x_n)$ tässä pisteessä ja tulon $f(c-h)f(c+h)$ arvo. Listauksesta nähdään, kuinka Newtonin menetelmän suoritus pysähtyy ensimmäisessä pisteessä, jossa $f(c-h)f(c+h) < 0$. (Kuva 36.)

Ohjelma 16: Newtonin menetelmän vaiheet taulukoituna

```
@interact
def _(f = x^2 - 2, c = 6, hh = slider(-16, -1, 1, -3, label="Tarkkuus 2h"),
      maxn = slider(0, 15, 1, 10, label="Kierroksia max")):
   f(x) = f
   h = 10^hh
   c, valipisteet = newtonin_menetelma(f, float(c), maxn, h/2.0)
   html("$\\text{Tarkkuus: } 2h = %s$"%float(h))
   html("$\\text{Iteratioita = }%s<br>"%len(valipisteet))
   # HTML-taulukon tulostaminen.
   s = "<br><br><table border=1 cellpadding=5>"
   s +=
       "<tr><td>$n$</td><td>$x_n$</td><td>$f(x_n)$</td><td>$f(x_n-h)f(x_n+h)$</td></tr>"
   for i, c in enumerate(valipisteet):
      s += "<tr><td>$%s$</td><td>$%s$</td><td>$%s$</td><td>$%s$</td></tr>"%(i, c,
         f(c), f(c-h)*f(c+h))
   s += "</table>"
   html(s)
```



n	x_n	$f(x_n)$	$f(x_n-h)f(x_n+h)$
0	6.0	34.0	1155.9999240000011
1	3.166666666666667	8.027777777777803	64.445191993828189
2	1.8991228070175439	1.6066674361341953	2.5813690370001559
3	1.4761202949637373	0.17893112520383081	0.032007989705458623
4	1.4155117098049557	0.0036734005949488058	$5.4865261297846906e-06$
5	1.4142141576301823	$1.6836416460996873e-06$	$-7.9999995326331724e-06$

Kuva 36: Newtonin menetelmän vaiheet taulukoituna.

7.2 Numeerinen integrointi

Määrätyn integraalin arvo voidaan usein laskea analyyttisesti, mutta aina se ei ole mahdollista. Esimerkiksi funktion e^{-x^2} integraalia ei voida esittää alkeisfunktiona suljetussa muodossa. Jos funktio on riittävän säännöllinen, määrätyn integraalin likimääräinen arvo voidaan laskea käyttämällä erilaisia numeerisia menetelmiä. Numeerista integrointia voidaan hyödyntää apuna myös tilanteissa, joissa vain osa funktion pisteistä tunnetaan tai integraalin analyyttinen määrittäminen on vaikeaa.

Kaikille numeerisen integroinnin menetelmille on yhteistä, että integraalia arvioidaan summalla, jossa integrandin arvoja tarkasteluvälin eri kohdissa x_i kerrotaan painoilla w_i. Geometrisesti perusteltuna funktion ja x-akselin välistä alaa arvioidaan sovittamalla tarkasteluvälin osaväleille geometrisia tasokuvioita, joiden pinta-ala voidaan määrittää. Osavälien lukumäärää kasvattamalla määrätyn integraalin arvo voidaan määrittää halutulla tarkkuudella.

Numeerisista integrointimenetelmistä seuraavassa perehdytään suorakulmio- ja puolisuunnikasmenetelmiin sekä Simpsonin menetelmään.

7.2.1 Suorakulmiomenetelmät

Suorakulmiomenetelmissä integrointivälin osaväleille sovitetaan suorakulmioita, joiden pinta-alojen yhteissummasta saadaan approksimaatio määrätylle integraalille. Yksinkertaisin tapa on valita suorakulmion korkeudeksi funktion arvo osavälin keskipisteessä. Merkitään välien keskipisteitä $x_1,\ x_2,\ ...,\ x_n$. Kun osavälit oletetaan yhtä pitkiksi, funktion f kuvaajan ja x-akselin välillä [a, b] rajaaman alueen pinta-ala on likimäärin

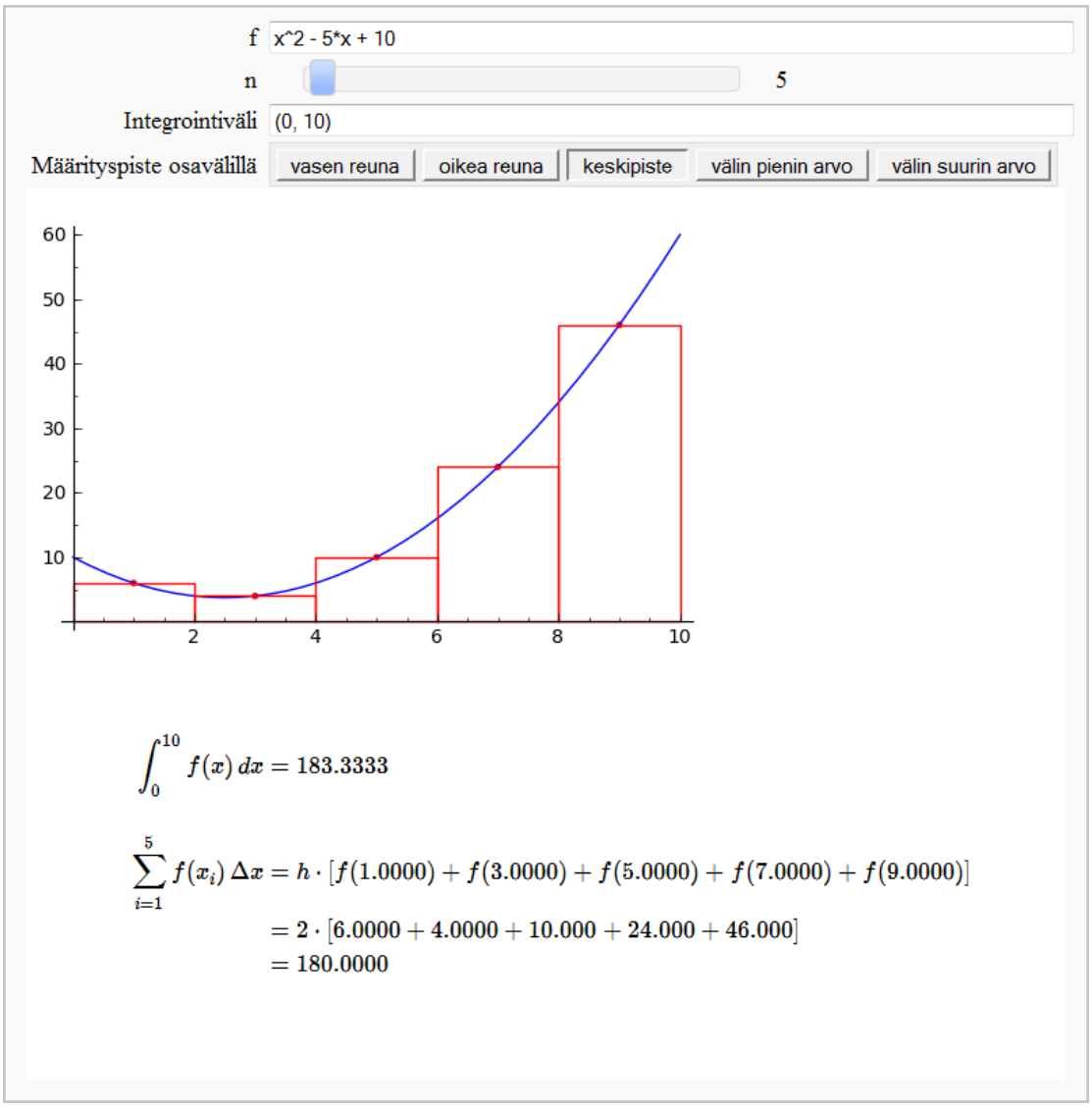
$$hf(x_1) + hf(x_2) + ... + hf(x_n) = h \cdot [f(x_1) + f(x_2) + ... + f(x_n)]\,,$$

missä h on osavälin pituus. Muita tapoja määrittää suorakulmioiden korkeus on valita välin suurin tai pienin arvo (niin sanotut ylä- ja alasummat) tai osavälin vasemman tai oikean reunan arvo.

Seuraavan ohjelman avulla voidaan graafisesti tarkastella suorakulmiomenetelmän toimintaa. Käyttäjän valittavana on funktio, integrointiväli, osavälien määrä ja sääntö, jonka perusteella suorakulmion korkeus määrätään osavälillä. Ohjelma piirtää funktion kuvaajan ja suorakulmiot valitun säännön mukaisesti. Kuvaajan alla esitetään laskulausekkein miten menetelmässä määrätyn integraalin arvo arvioidaan painotetun summan avulla. Sovellus laskee myös Sagen oman kehittyneen numeerisen integrointimetodin avulla mahdollisimman hyvän likiarvon määrätylle integraalille. Näin käyttäjä voi

verrata suorakulmiomenetelmän avulla laskettua arvoa tarkkaan arvoon. Lisäksi ohjelmalla voidaan tutkia Riemannin integraalin määritelmään liittyvien ala- ja yläsummien käyttäytymistä, kun osavälien määrää kasvatetaan. (Kuva 37.)

Pythonissa on mahdollista määritellä nimeämättömiä funktioita käyttämällä *lambda*-avainsanaa. Ohjelman lähdekoodissa tätä hyödynnetään, kun määritellään säännöt eri määrityspisteille. Metodi *find_minimum_on_interval(funktio, a, b)* palauttaa funktion pienimmän arvon ja pisteen, jossa tämä arvo saavutetaan välillä $[a, b]$. Vastaavasti metodi *find_maximum_on_interval* etsii välin suurimman arvon. Ohjelma perustuu Marshall Hamptonin ja Nick Alexanderin kirjoittamaan sovellukseen [81].



Kuva 37: Numeerinen integrointi suorakulmiomenetelmällä.

Ohjelma 17: Numeerinen integrointi suorakulmiomenetelmällä

```
maarityspiste_saannot = (
   ("vasen reuna",        lambda f, i, h, a: a + i*h),
   ("oikea reuna",        lambda f, i, h, a: a + i*h + h),
   ("keskipiste",         lambda f, i, h, a: a + i*h + h/2),
   ("välin pienin arvo",  lambda f, i, h, a: find_minimum_on_interval(f, a + i*h, a
       + i*h + h)[1]),
   ("välin suurin arvo",  lambda f, i, h, a: find_maximum_on_interval(f, a + i*h, a
       + i*h + h)[1])
)

maarityspiste_nimet = [r[0] for r in maarityspiste_saannot]
maarityspiste_saannot = dict(maarityspiste_saannot)

@interact
def _(f = input_box(default = x^2-5*x + 10),
      n = slider(1, 100, 1, 5),
      vali = input_box(default=(0, 10), label="Integrointiväli"),
      maarityspiste = selector(maarityspiste_nimet, nrows=1, label="Määrityspiste
          osavälillä")):

   xs = []; ys = []
   h = (vali[1]-vali[0])/n
   f(x) = f
   maarityspiste_funktio = maarityspiste_saannot[maarityspiste]

   # Piirretään suorakulmiot määrityspistesäännön mukaisesti.
   suorakulmiot = Graphics()
   for i in range(n):
      xi = maarityspiste_funktio(f, i, h, vali[0])
      yi = f(xi)
      xi0 = vali[0] + i*h
      suorakulmiot += line([[xi0, 0], [xi0, yi], [xi0 + h, yi], [xi0 + h, 0], [xi0,
          0]], rgbcolor=(1, 0, 0))
      suorakulmiot += point((xi, yi), rgbcolor=(1,0,0))
      xs.append(xi)
      ys.append(yi)

   show(plot(f, vali[0], vali[1]) + suorakulmiot, xmin = vali[0], xmax = vali[1])

   # Lasketaan määrätyn integraalin approksimaatio suorakulmiomenetelmän ja Sagen
       numeerisen metodin avulla.
   approksimaatio = h * sum([ys[i] for i in range(n)])
   numeerinen_arvo = integral_numerical(f, vali[0], vali[1])[0]
```

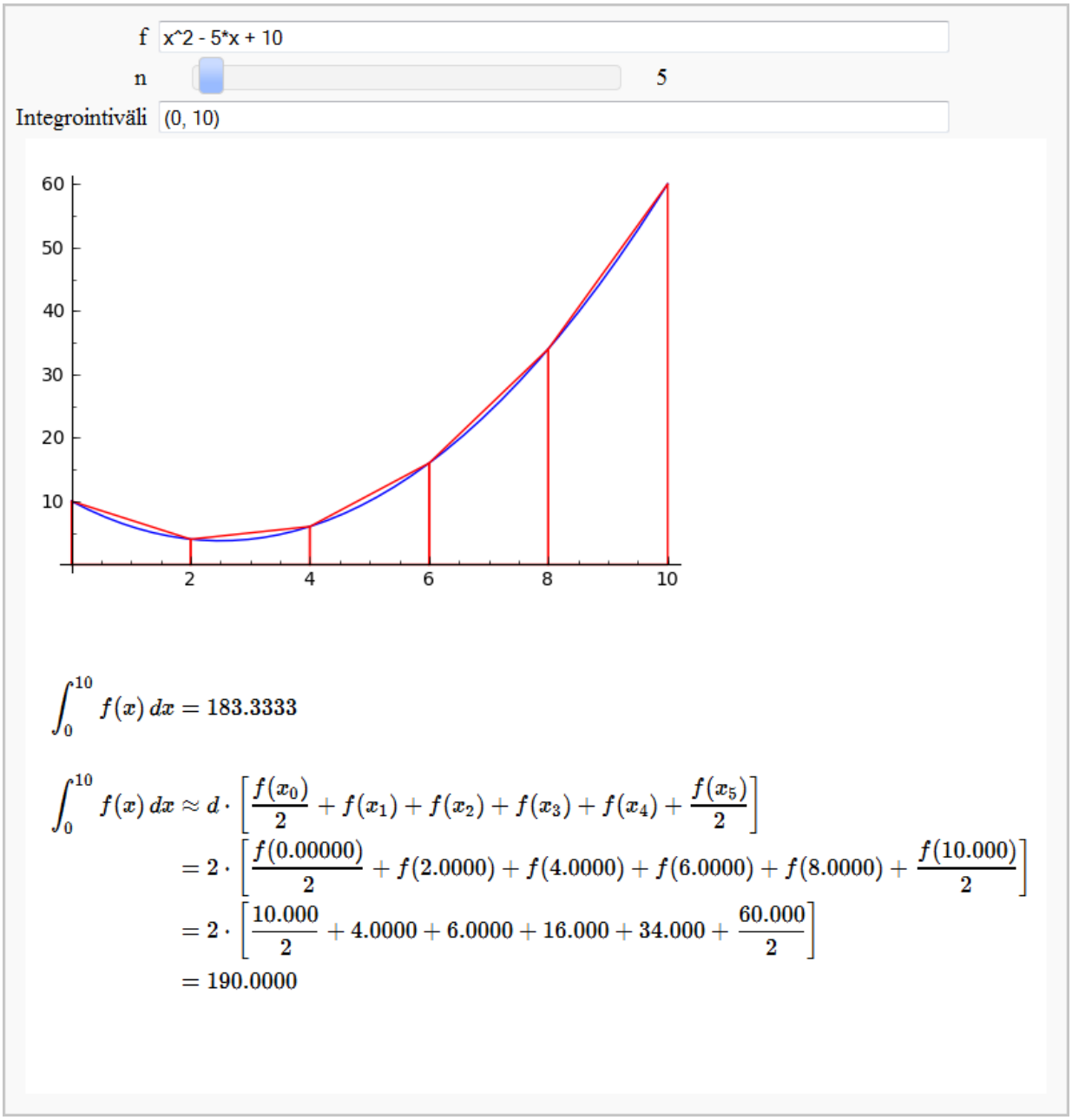
```
# Tulostetaan laskulausekkeet.
summa_sijoitus_html = "h \cdot \\left[ %s \\right]" %(join(["f(%s)"% N(i,
    digits=5) for i in xs]," + "))
summa_arvot_html = "%s \cdot \\left[ %s \\right]" %(h, join(["%s"% N(i,
    digits=5) for i in ys]," + "))

html(r'''
<div class="math">
\begin{align*}
\int_{%s}^{%s} {f(x) \, dx} & = %s \\\
\\\
\sum_{i=1}^{%s} {f(x_i) \, \Delta x}
   & = %s \\\
   & = %s \\\
   & = %s
\end{align*}
</div>
''' % (vali[0], vali[1], N(numeerinen_arvo, digits=7), n, summa_sijoitus_html,
    summa_arvot_html, N(approksimaatio, digits=7)))
```

7.2.2 Puolisuunnikasmenetelmä

Toinen tapa arvioida määrätyn integraalin arvoa on approksimoida käyrän muotoa lineaarisen approksimaation avulla. Puolisuunnikasmenetelmässä jokaisella osavälillä funktion käyrä korvataan janalla, joka kulkee funktion päätepistearvojen kautta. Funktion kuvaajan ja x-akselin rajaaman alueen pinta-ala on tällä välillä likimäärin sama kuin sovitetun puolisuunnikkaan ala. Koko integrointivälin pinta-alaksi saadaan tällöin

$$\begin{aligned}&\frac{f(x_0)+f(x_1)}{2}\cdot h+\frac{f(x_1)+f(x_2)}{2}\cdot h+\ldots+\frac{f(x_{n-1})+f(x_n)}{2}\cdot h\\&=h\cdot(\frac{f(a)}{2}+f(x_1)+f(x_2)+\ldots+f(x_{n-1})+\frac{f(b)}{2}).\end{aligned}$$

Puolisuunnikasmenetelmän toimintaa havainnollistava sovellus on tehty muokkaamalla edellistä sovellusta ja sen ohjelmakoodi on liitteenä (A.3). (Kuva 38.)



$$\int_0^{10} f(x)\,dx = 183.3333$$

$$\begin{aligned}\int_0^{10} f(x)\,dx &\approx d \cdot \left[\frac{f(x_0)}{2} + f(x_1) + f(x_2) + f(x_3) + f(x_4) + \frac{f(x_5)}{2}\right] \\ &= 2 \cdot \left[\frac{f(0.00000)}{2} + f(2.0000) + f(4.0000) + f(6.0000) + f(8.0000) + \frac{f(10.000)}{2}\right] \\ &= 2 \cdot \left[\frac{10.000}{2} + 4.0000 + 6.0000 + 16.000 + 34.000 + \frac{60.000}{2}\right] \\ &= 190.0000\end{aligned}$$

Kuva 38: Numeerinen integrointi puolisuunnikasmenetelmän avulla.

7.2.3 Simpsonin menetelmä

Siinä missä suorakulmiomenetelmässä käyrää approksimoitiin vakiofunktiolla ja puolisuunnikasmenetelmässä suoralla, Simpsonin menetelmässä funktiota arvioidaan osavälillä toisen asteen polynomilla. Tehtävänä on sovittaa osavälille paraabeli, joka kulkee välin päätepisteiden ja keskipisteen kautta. Kolmen eri pisteen kautta kulkevan paraabelin $y = Ax^2 + Bx + C$ vakiot A, B ja C voidaan ratkaista muodostamalla yhtälöryhmä

$$\begin{cases} y_1 = Ax_1^2 + Bx_1 + C \\ y_2 = Ax_2^2 + Bx_2 + C \\ y_3 = Ax_3^2 + Bx_3 + C, \end{cases}$$

missä (x_1, y_1), (x_2, y_2) ja (x_3, y_3) ovat pisteitä, joiden kautta paraabeli kulkee. Sagen avulla voidaan kirjoittaa ohjelmafunktio, joka ratkaisee ongelman. Funktio palauttaa paraabelin käyrän funktion $y = f(x)$.

```
def paraabeli(a, b, c):
    A, B, C, x = var("A, B, C, x")
    K = solve([A*a[0]^2 + B*a[0] + C == a[1], A*b[0]^2 + B*b[0] + C ==
        b[1], A*c[0]^2 + B*c[0] + C == c[1]], [A, B, C],
        solution_dict=True)[0]
    f = K[A]*x^2 + K[B]*x + K[C]
    return f
```

Simpsonin menetelmässä jakovälejä tulee olla parillinen määrä. Voidaan osoittaa, että koko välillä $[a, b]$ funktion kuvaajan ja x-akselin rajaaman alueen alaksi saadaan Simpsonin menetelmällä

$$A = \frac{h}{3} \cdot (f(a) + 4f(x_1) + 2f(x_2) + ... + 2f(x_{n-2}) + 4f(x_{n-1}) + f(b)),$$

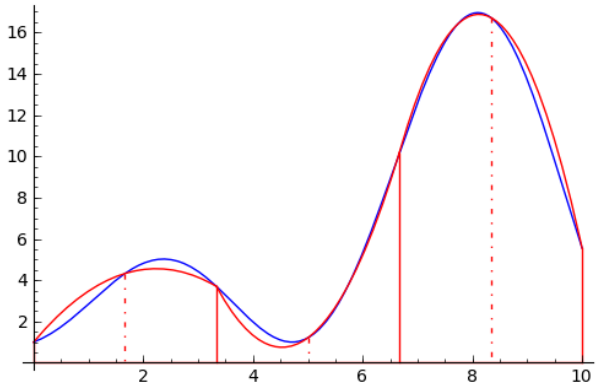
missä funktion arvojen kertoimet 4 ja 2 vuorottelevat.

Simpsonin menetelmän ohjelmakoodi on esitetty liitteessä (A.4).

f x*sin(x) + x + 1

n 6

Integrointiväli (0, 10)



$$\int_0^{10} f(x)\,dx = 67.84669$$

$$\begin{aligned}\int_0^{10} f(x)\,dx &\approx \frac{d}{3}\cdot[f(x_0)+4\cdot f(x_1)+2\cdot f(x_2)+4\cdot f(x_3)+2\cdot f(x_4)+4\cdot f(x_5)+f(x_6)]\\ &= \frac{5/3}{3}\cdot[f(0.00000)+4\cdot f(1.6667)+2\cdot f(3.3333)+4\cdot f(5.0000)+2\cdot f(6.6667)+4\cdot f(8.3333)+f(10.000)]\\ &= \frac{5/3}{3}\cdot[1.0000+4\cdot 4.3257+2\cdot 3.6981+4\cdot 1.2054+2\cdot 10.161+4\cdot 16.727+5.5598]\\ &= 68.50670\end{aligned}$$

Kuva 39: Numeerinen integrointi Simpsonin menetelmällä.

7.3 Funktion approksimointi Taylorin polynomien avulla

Funktion approksimointi on hyödyllistä muun muassa silloin, kun alkuperäisen funktion arvoja on vaikea laskea tai kun funktiosta tunnetaan vain yksittäisiä arvoja. *Weierstrassin approksimaatiolauseen* mukaan jatkuvaa funktiota voidaan suljetulla välillä approksimoida polynomilla kuinka tarkasti tahansa. Yksi tapa johtaa tällaisia polynomeja on määrittää funktion Taylorin sarjan termejä. Funktion f astetta n oleva Taylorin polynomi kehityskeskuksena x_0 on

$$\sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k.$$

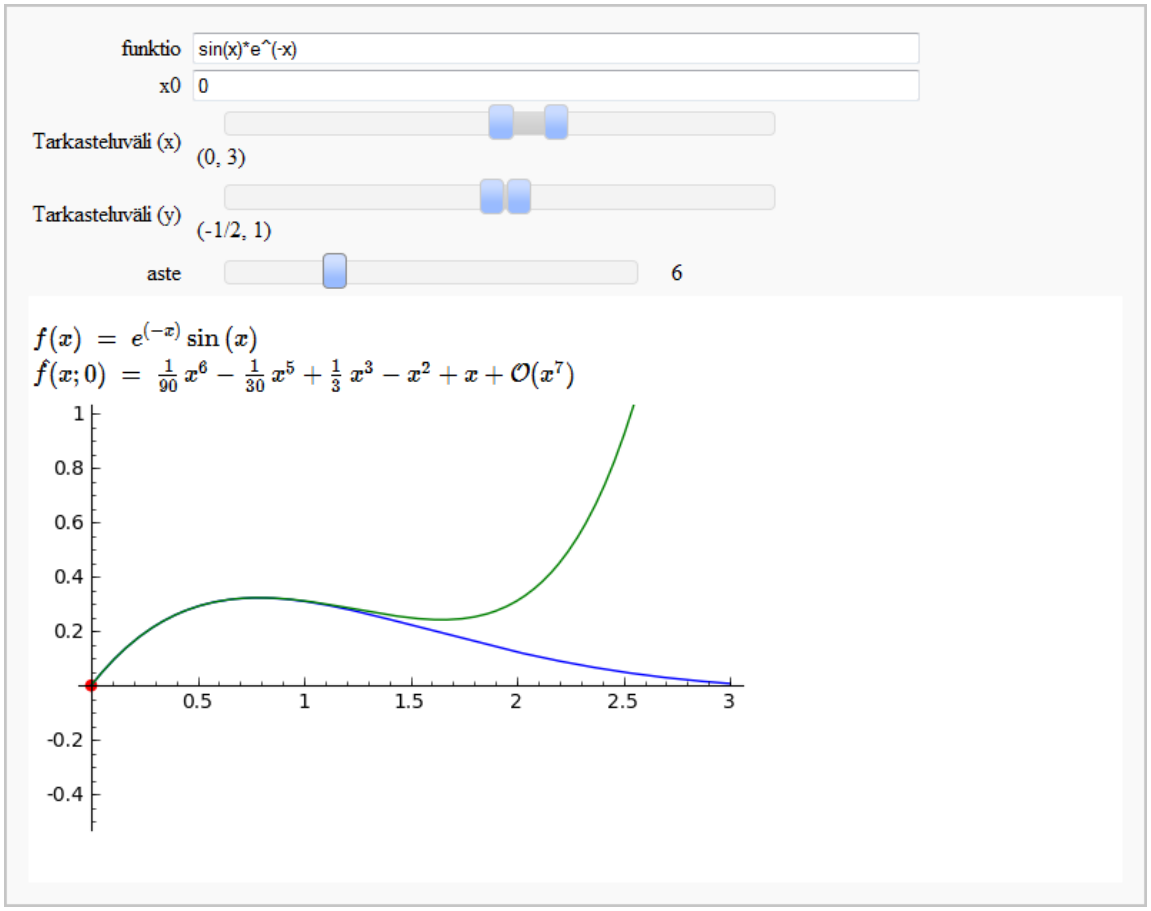
Kehityskeskuksen läheisyydessä polynomi antaa usein varsin tarkan approksimaation funktiolle, mutta asteluvun n pienillä arvoilla tarkkuus heikkenee nopeasti siirryttäessä kauemmaksi kehityskeskuksesta. Taylorin polynomi ei välttämättä suppene kohti funktiota asteen kasvaessa kuin vain jollain rajoitetulla välillä kehityskeskuksen ympärillä (ks. tehtävä 3).

Seuraava ohjelma laskee Taylorin polynomin käyttäjän antamalle funktiolle. Polynomin tulostuksessa esiintyvä *ordo*-merkintä kertoo Taylorin sarjassa jäljellä olevien termien pienimmän asteluvun. Oletuksena approksimaatio lasketaan origon ympäristössä, $x_0 = 0$, eli kyseessä on niin sanottu *Maclaurinin polynomi*. Ohjelma piirtää samaan kuvaan alkuperäisen funktion f ja approksimaatiopolynomin $\hat{f}(x; x_0)$ kuvaajat. Piirtoväli on käyttäjän valittavissa. (Kuva 40.)

Sovelluksen avulla voidaan tutkia Taylorin polynomien käyttäytymistä eri asteluvuilla ja kehityskeskuksen x_0 arvoilla. Voidaan myös yrittää selvittää alue, jossa Taylorin sarja suppenee. Ohjelmakoodissa metodi *f(x).taylor(x, x0, n)* laskee automaattisesti astetta n olevan Taylorin polynomin kehityskeskuksen $x0$ ympäristössä [82]. Ohjelma perustuu Harald Schillyn kirjoittamaan ohjelmakoodiin [83].

Ohjelma 18: Funktion approksimointi Taylorin polynomien avulla

```
var('x')
@interact
def _(funktio = input_box(default = "sin(x)*e^(-x)", type=str),
   x0 = input_box(default = 0),
   xvali = range_slider(-15, 15, 1/2, default=(0,3), label="Tarkasteluväli (x)"),
   yvali = range_slider(-15, 15, 1/2, default=(-0.5,1), label="Tarkasteluväli (y)"),
   aste=slider(1, 20, 1, 1)):
  f(x) = funktio
  # Lasketaan Taylorin polynomi kehityskeskuksen x0 ympärillä.
  f_taylor(x) = f(x).taylor(x, x0, aste)
  # Piirretään funktio f ja siihen liittyvä Taylorin polynomi.
  f_kuvaaja = plot(f(x), xvali, color="blue")
  taylor_kuvaaja = plot(f_taylor(x), xvali, color="green")
  piste = point((x0, f(x0)), pointsize=30, rgbcolor=(1,0,0))
  html("$f(x)\;=\;%s$"%latex(f(x)))
  html("$\hat{f}(x;%s)\;=\;%s+\mathcal{O}(x^{%s})$"%(x0, latex(f_taylor(x)), aste+1))
  show(f_kuvaaja + taylor_kuvaaja + piste, xmin=xvali[0], xmax=xvali[1],
      ymin=yvali[0], ymax=yvali[1])
```



Kuva 40: Funktion approksimointi Taylorin polynomien avulla.

7.4 Differentiaaliyhtälöiden likimääräinen ratkaiseminen

Differentiaaliyhtälön ratkaiseminen analyyttisesti onnistuu vain tietyissä erikoistapauksissa, minkä vuoksi on usein käytettävä numeerisia menetelmiä vastauksen selvittämiseksi. Seuraavassa tarkastellaan muotoa $y' = f(t, y)$ olevia differentiaaliyhtälöitä, missä $f(t, y)$ on funktiosta $y(t)$ ja muuttujasta t riippuva funktio. Yhtälöön liittyy lisäksi alkuarvoehto $y(t_0) = y_0$.

Eksplisiittisistä numeerisista differentiaaliyhtälöiden ratkaisualgoritmeista käsitellään seuraavassa Eulerin menetelmää, keskipistemenetelmää ja Rungen-Kuttan menetelmää. Algoritmeissa tarkasteluväli $[a, b]$ jaetaan yhtä pitkiin osaväleihin. Merkitään osavälien pituutta eli askelväliä kirjaimella h ja jakopisteitä $[t_0 = a,\ t_1,\ t_2,\ \ldots,\ t_n = b]$. Oletetaan vielä, että osavälejä on n kappaletta, eli $h = \frac{b-a}{n}$.

7.4.1 Eulerin menetelmä

Eulerin menetelmässä ratkaisufunktiota approksimoidaan osaväleillä suorilla, joiden kulmakerroin lasketaan jokaisen osavälin alkupisteessä. Jos tarkasteluvälien jako on riittävän tiheä, funktion käyrä poikkeaa jokaisella osavälillä vain vähän osavälin alkupisteeseen piirretystä tangenttisuorasta.

Ensimmäisellä osavälillä tangentin kulmakerroin voidaan laskea suoraan differentiaaliyhtälöstä alkuarvoehdon ollessa voimassa: $y' = f(t_0, y_0)$. Osavälin päätepisteessä (t_1, y_1) ratkaisufunktion y arvo on $y_1 = y_0 + h \cdot f(t_0, y_0)$. Näin jatkamalla saadaan funktion arvo määritettyä likimääräisesti jokaisessa jakopisteessä. Eulerin menetelmä voidaan esittää rekursiivisesti differenssiyhtälönä:

$$y_0 = y(t_0), \qquad y_{k+1} = y_k + h \cdot f(t_k, y_k), \qquad t_{k+1} = t_k + h.$$

Kirjoitetaan ohjelmafunktio, joka saa syötteenä funktion $f(t, y)$, osavälin pituuden h, osavälien lukumäärän n ja alkuarvoehdon $y(t_0) = y_0$ muuttujien arvot t_0 ja y_0. Metodi palauttaa listan ratkaisufunktion pisteistä (x_i, y_i), missä y_i on ratkaisufunktion arvo, kun $x_i = t_0 + i \cdot h$.

```
def eulerin_menetelma(f, h, n, t0, y0):
   f(t, y) = f
   xy = [(t0, y0)]
   for k in range(n):
      tk = xy[-1][0]
      yk = xy[-1][1]
      xy.append((tk + h, yk + h*f(tk, yk)))
   return xy
```

7.4.2 Keskipistemenetelmä

Eulerin menetelmää parempi tulos saadaan, kun funktion käyrän osavälillä korvaavan janan kulmakerroin lasketaan välin alkupisteen sijaan osavälin keskipisteessä. Näin ollen menetelmän rekursiiviseksi kaavaksi saadaan

$$y_{k+1} = y_k + h \cdot f(t_k + \frac{h}{2},\ y_k + f(t_k, y_k) \cdot \frac{h}{2}), \qquad t_{k+1} = t_k + h.$$

Keskipistemenetelmän avulla ratkaisufunktion arvoja laskeva ohjelmafunktio saadaan muokkaamalla Eulerin menetelmää:

```
def kp_menetelma(f, h, n, t0, y0):
   f(t, y) = f
   xy = [(t0, y0)]
   for k in range(n):
      tk = xy[-1][0]
      yk = xy[-1][1]
      xy.append((tk + h, yk + h*f(tk + h/2, yk + f(tk, yk)*h/2)))
   return xy
```

7.4.3 Neljännen asteen Rungen-Kuttan menetelmä

Voidaan osoittaa, että Eulerin menetelmällä laskettujen arvojen virhe on verrannollinen askelvälin pituuteen. Eulerin menetelmän sanotaan olevan ensimmäistä astetta. Keskipistemenetelmä on toista astetta: virhe on verrannollinen askelvälin neliöön. Seuraavassa esitettävä Rungen-Kuttan menetelmä on neljättä astetta, mikä on riittävän tarkka käytännön laskentaan.

Rungen-Kuttan menetelmän laskukaava voidaan kirjoittaa neljässä osassa seuraavasti:

$$y_{k+1} = y_k + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4),$$

missä

$$K_1 = f(t_k,\ y_k),$$
$$K_2 = f(t_k + \frac{1}{2}h,\ y_k + \frac{h}{2}K_1),$$
$$K_3 = f(t_k + \frac{1}{2}h,\ y_k + \frac{h}{2}K_2),$$
$$K_4 = f(t_k + h,\ y_k + hK_3).$$

Runge-Kuttan menetelmän ohjelmafunktio on esitetty alla.

```
def runge_kutta(f, h, n, t0, y0):
   f(t, y) = f
   xy = [(t0, y0)]
   for k in range(n):
      tk = xy[-1][0]
      yk = xy[-1][1]
      k1 = f(tk, yk)
      k2 = f(tk + h/2, yk + k1*h/2)
      k3 = f(tk + h/2, yk + k2*h/2)
      k4 = f(tk + h, yk + k3*h)
      k = (k1 + 2*k2 + 2*k3 + k4)/6
      xy.append((tk + h, yk + k*h))
   return xy
```

7.4.4 Differentiaaliyhtälöiden numeeristen ratkaisumenetelmien vertailu

Lopuksi esitetään ohjelma, joka laskee differentiaaliyhtälön numeeriset ratkaisut edellä esitettyjen menetelmien avulla ja piirtää ne samaan koordinaatistoon yhdessä tarkan ratkaisufunktion kuvaajan kanssa. Lisäksi ohjelma esittää taulukossa kaikkien eri menetelmien antamat arvot ja vertailuarvoina ratkaisufunktion tarkat arvot pisteissä t_k.

Ohjelmalle annetaan syötteenä tarkasteltava differentiaaliyhtälö, askelvälin pituus, askelien määrä ja alkuarvoehdot. Käyttäjä voi myös syöttää differentiaaliyhtälön tunne-

tun ratkaisun, jos sellainen on tiedossa. Muussa tapauksessa voidaan kokeilla differentiaaliyhtälön ratkaisemista Maximan avulla. Tällöin ohjelma kutsuu metodia *desolve(f, y, [t0, y0])*, joka pyrkii ratkaisemaan differentiaaliyhtälön $f(t, y) = 0$ alkuarvoehdolla $y(t0) = y0$.

Desolve-metodin käyttö edellyttää, että y on alustettu funktioksi ohjelmakoodissa. Määrittelemätön funktio voidaan alustaa komennolla *function("y", t)*, missä y on muuttujasta t riippuva funktio. Määrittelemättömän funktion derivaattafunktio luodaan metodilla *diff(y, t)*. Lisätietoja *desolve*-metodin käytöstä ja differentiaaliyhtälöiden ratkaisemisesta Sagen avulla on esitetty viitteessä [84].

Jos ratkaisua ei tunneta eikä differentiaaliyhtälön ratkaiseminen analyyttisesti onnistu, voidaan tarkkojen arvojen laskeminen sivuuttaa. Halutessaan käyttäjä voi valita piirrettäviksi differentiaaliyhtälön suuntaelementit, eli säännöllisin välimatkoin tason pisteisiin (t, y) piirretyt lyhyet janat, joiden kulmakerroin määräytyy differentiaaliyhtälöstä $y' = f(t, y)$. Suuntaelementit piirretään välillä $[a, b]x[c, d]$ metodilla *plot_slope_field(f, (x, a, b), (y, c, d))*. Suuntaelementtejä seuraamalla voidaan hahmottaa yksittäisen ratkaisun kulku. [85]

Kuvaajan alle ohjelma listaa lasketut ratkaisufunktion arvot pisteissä t_k. Sovelluksen avulla voidaan tutkia numeeristen ratkaisumenetelmien toimintaa eri askelpituuden arvoilla ja arvioida menetelmien virhettä tarkkaan ratkaisuun verrattuna.

Ohjelma 19: Differentiaaliyhtälöiden numeeristen ratkaisumenetelmien vertailu

```
@interact
def difVertailu(f = input_box(default = "y - 2*sin(t)", label="y' ="),
                  g = input_box(default = "sin(t) + cos(t)", label="Tunnettu
                      ratkaisu: y ="),
                  h = input_box(default=0.5, label="Askelväli: h ="),
                  n = input_box(default=5, label="Askeleita: n ="),
                  t0 = input_box(default=0.0, label="t0 ="),
                  y0 = input_box(default=1.0, label="y0 ="),
                  valinta = selector([(0, "Tunnettu ratkaisu"), (1, "Maxima"), (2,
                      "Ei ratkaista")], label="Tarkkojen arvojen laskeminen"),
                  suunnat=checkbox(default = False, label="Suuntaelementit:")):

   f(y,t) = f
   g(y,t) = g
```

```
# Valitaan, näytetäänkö tarkka ratkaisu (tunnettu tai Maximan määrittämä).
if valinta == 0:
   g(t) = g
elif valinta == 1:
   s = var("s")
   u = function("u", s)
   g(t) = desolve(diff(u,s)-f(u,s), u, [t0,y0])(s=t)
else:
   g(t)=0.0
g_kuva = Graphics()
if valinta != 2:
   g_kuva = plot(g(t), (t0, t0+n*h), color="blue", thickness=1.5)

# Määritetään ratkaisut eri menetelmien avulla.
eu_xy = eulerin_menetelma(f, h, n, t0, y0)
kp_xy = kp_menetelma(f, h, n, t0, y0)
rk_xy = runge_kutta(f, h, n, t0, y0)

# Piirretään ratkaisuun liittyvä grafiikka.
pisteet = point(eu_xy, color="red")
pisteet += point(kp_xy, color="green")
pisteet += point(rk_xy, color="brown")
viivat = line(eu_xy, color="red")
viivat += line(kp_xy, color="green")
viivat += line(rk_xy, color="brown")
kuva = g_kuva + pisteet + viivat

# Suuntaelementtien piirtäminen kuvaajaan.
if suunnat == True:
   kuva += plot_slope_field(f, (t, t0, t0 + n*h), (y, kuva.ymin(), kuva.ymax()))
show(kuva)

# Tulostetaan taulukko ratkaisufunktioiden arvoista.
s = "<br><br><table border=1 cellpadding=5>"
s += "<tr><td> $n$ </td><td> $t$ </td><td> $\color{Red}{\\text{Euler}}$
    </td><td> $\color{Green}{\\text{Eulerin kp}}$ </td><td>
    $\color{Brown}{\\text{Runge-Kutta}}$ </td><td> $\color{Blue}{\\text{Tarkka
    arvo}}$ </td></tr> "
for i in range(n+1):
   s += "<tr><td> $%s$ </td><td> $%s$ </td><td> $%s$ </td><td> $%s$ </td><td>
      $%s$ </td><td> $%s$ </td></tr>" %(i, N(t0+i*h, digits=4), N(eu_xy[i][1],
      digits=10), N(kp_xy[i][1], digits=10), N(rk_xy[i][1], digits=10),
      N(g(t0+i*h), digits=10))
s += "</table>"
html(s)
```

y' = y - 2*sin(t)

Tunnettu ratkaisu: y = sin(t) + cos(t)

Askelväli: h = 0.500000000000000

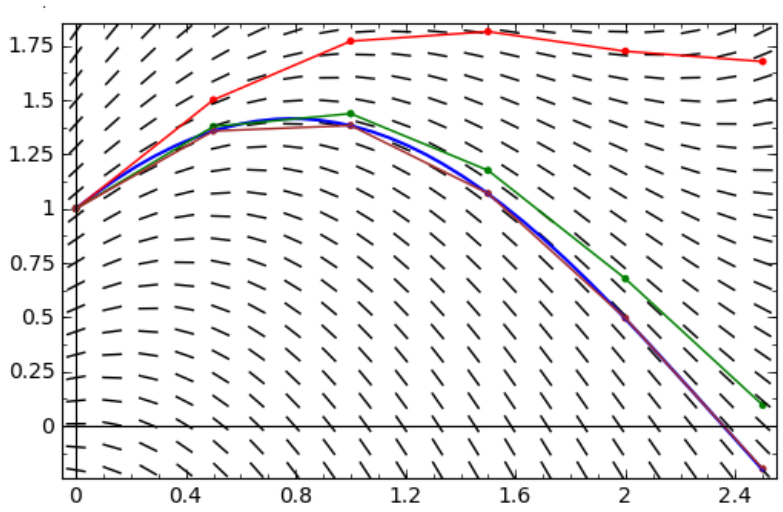
Askeleita: n = 5

t0 = 0.000000000000000

y0 = 1.00000000000000

Tarkkojen arvojen laskeminen 'Tunnettu ratkaisu'

Suuntaelementit:



n	t	Euler	Eulerin kp	Runge-Kutta	Tarkka arvo
0	0.0000	1.000000000	1.000000000	1.000000000	1.000000000
1	0.5000	1.500000000	1.377596041	1.357209028	1.357008101
2	1.000	1.770574461	1.437098422	1.382454561	1.381773291
3	1.500	1.814390707	1.175932569	1.069768694	1.068232188
4	2.000	1.724091074	0.6775307318	0.4960590861	0.4931505903
5	2.500	1.676839185	0.09558988561	−0.1976311238	−0.2026714714

Kuva 41: Differentiaaliyhtälöiden numeeristen ratkaisumenetelmien vertailu.

7.5 NumPy-moduuli

NumPy (*Numerical Python*) on numeerisiin menetelmiin ja matriiseihin erikoistunut Pythonin laajennuskirjasto. Se tukee suurikokoisten ja moniulotteisten taulukoiden ja matriisien nopeaa käsittelyä. NumPy sisältää suuren kokoelman erilaisia lineaarialgebran operaatioita sekä muun muassa satunnaislukuihin ja Fourier'n muunnoksiin liittyviä metodeja. Tarkastellaan lyhyesti NumPy-kirjaston taulukkorakennetta ja matriisioperaatioita sekä lineaaristen yhtälöryhmien ratkaisemista Sagen avulla. [86]

7.5.1 NumPy:n matriisioperaatioita

NumPy-moduulin taulukko määritellään *array*-metodin avulla Pythonin listoja käyttäen. Kaksiulotteisille taulukolle on kirjoitettu runsaasti matriisioperaatioita, joista tärkeimpiä on lueteltu taulukossa 17. *Transpose*-metodia lukuun ottamatta funktiot kuuluvat NumPy:n lineaarialgebraan erikoistuneeseen *linalg*-alipakettiin. [87]

transpose(**A**)	Palauttaa transpoosin.
det(**A**)	Palauttaa determinantin.
inv(**A**)	Palauttaa käänteismatriisin.
eigvals(**A**)	Palauttaa ominaisarvot.
eig(**A**)	Palauttaa ominaisarvot ja -vektorit.

Taulukko 17: NumPy-kirjaston matriisioperaatioita.

Esimerkki: Matriisin transpoosin, determinantin ja ominaisarvojen laskeminen

Lasketaan seuraavan neliömatriisin transpoosi, determinantti ja ominaisarvot:

$$\mathbf{A} = \begin{pmatrix} 3 & 2 & -1 \\ 2 & -2 & 4 \\ -1 & 0.5 & -1 \end{pmatrix}.$$

```
from numpy import *
A = array([[ 3, 2, -1], [2, -2, 4], [-1, 1/2, -1]])
print "Transpoosi: "
print transpose(A)
print "Determinantti: ", linalg.det(A)
print "Ominaisarvot: ", linalg.eigvals(A)
```

```
Transpoosi:
[[ 3. 2. -1. ]
[ 2. -2. 0.5]
[-1. 4. -1. ]]
Determinantti: -3.0
Ominaisarvot: [3.62952412 -3.84451968 0.21499555]
```

7.5.2 Esimerkki: Lineaarisen yhtälöryhmän ratkaiseminen

Tarkastellaan yhtälöryhmää

$$\begin{cases} 3x + 2y - z = 1 \\ 2x - 2y + 4z = -2 \\ -x + \frac{1}{2}y - z = 0 \end{cases}$$

Ryhmässä on kolme yhtälöä ja yhtä monta tuntematonta muuttujaa. Yhtälöryhmä voidaan kirjoittaa matriisiyhtälönä

$$\mathbf{A}\mathbf{x} = \mathbf{c},$$

missä

$$\mathbf{A} = \begin{pmatrix} 3 & 2 & -1 \\ 2 & -2 & 4 \\ -1 & 1/2 & -1 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 1 \\ -2 \\ 0 \end{pmatrix}.$$

Ratkaistaan matriisiyhtälö NumPy:n linalg-alipaketin metodin *solve(A, c)* avulla. [88] Syötteenä metodi saa kerroinmatriisin **A** ja vakiovektorin **c**, joiden perusteella vektorin **x** tuntemattomat muuttujat voidaan ratkaista.

```
from numpy import *
A = array([[ 3, 2, -1],
           [ 2, -2, 4],
           [-1, 1/2, -1]])
c = array([1, -2, 0])
linalg.solve(A, c)
```

```
array([ 1., -2., -2.])
```

Yhtälöryhmän ratkaisuksi saatiin siis $x = 1, y = -2$ ja $z = -2$.

Lisää esimerkkejä NumPy:n menetelmien käytöstä on koottu viitteeseen [89].

7.6 Tehtäviä

Tehtävä 7-1:
a) Vertaile puolitus-, sekantti- ja Newtonin menetelmien toimintaa, kun seuraavat yhtälöt halutaan ratkaista kolmen desimaalin tarkkuudella:

i) $2^x - x^3 = 0$ **ii)** $x - 2 = \sin x$ **iii)** $e^{-x} = x$

b) Tarkastellaan kosinifunktiota, $f(x) = \cos x$. Miten Newtonin menetelmä käyttäytyy, kun lähdetään alkuarvauksesta $c = 3$, joka on lähinnä yhtälön $f(x) = 0$ juurta $\pi/2$?

c) Tutki, miten Newtonin menetelmä käyttäytyy, kun tutkittavana funktiona on $f(x) = x^3 - x - 3$ alkuarvauksella $c = 0$.

d) Tarkastele funktion $f(x) = \frac{x\sqrt{|x|}}{|x|}$ nollakohdan määrittämistä Newtonin menetelmällä eri lähtöarvoja käyttäen. Funktiota ei ole määritelty pisteessä 0, mutta $\lim_{x\to 0} f(x) = 0$. Tutki yhtälön $\frac{(x-a)\sqrt{|x-a|}}{|x-a|} = 0$ ratkaisemista Newtonin menetelmän avulla eri muuttujan a arvoilla.

Tehtävä 7-2:
Vertaile numeeristen integrointimenetelmien välisiä eroja vaihdellen osavälien määrää, kun funktiona on

a) $f(x) = 2x^2 - 5x + 2$ välillä $[0, 3]$ **b)** $g(x) = |x \sin(3x)|$ välillä $[0, 2\pi]$
c) $h(x) = x(\frac{\sin(4x)}{4} + \frac{\sin(9x)}{9}) + 4$ välillä $[0, 10]$

Tehtävä 7-3:
Tutki seuraavien funktioiden Taylorin polynomeja vaihtelemalla kehityskeskusta ja astelukua. Yritä erityisesti selvittää alue, jossa Taylorin sarja suppenee. Voidaan todistaa, että Taylorin sarja suppenee avoimella välillä, jonka keskipisteenä on kehityskeskus. Puolet avoimen välin pituudesta on sarjan *suppenemissäde*. Teorian mukaan suppenemissäde on yhtä suuri kuin kehityskeskuksen etäisyys lähimmästä funktion erikoispisteestä. Tutki, miten havaintosi sopivat yhteen tämän tuloksen kanssa. [90]

a) $\cos x$ **b)** $\ln x$ **c)** $\sqrt{x+1}$ **d)** $\frac{1}{x^2+1}$

Tehtävä 7-4:

a) Tutki seuraavien differentiaaliyhtälöiden numeerisia ratkaisuja eri menetelmien avulla askelmäärää ja askelpituutta vaihdellen. Selvitä differentiaaliyhtälön analyyttinen ratkaisu Maximan avulla ja vertaa sitä numeeristen menetelmien antamiin tuloksiin.

i) $y' = ty$, kun $y(0) = 1$

ii) $y' = y - 2sin(t)$, kun $y(0) = 1$

iii) $y' = y + \frac{y}{t}$, kun $y(1) = e$

b) Määritä Rungen-Kuttan menetelmän avulla $y(0{,}4)$, kun y on yhtälön $y' = y^2 + t$, $y(0) = -3$ ratkaisu. Käytä askelväliä 0,1.

Viitteet

Sage-ohjelmiston verkkosivusto on osoitteessa `http://www.sagemath.org`.
Kaikki verkkolähteet on tarkistettu 15.8.2011.

[1] Magma-ohjelmiston kotisivu, `http://magma.maths.usyd.edu.au/magma/`

[2] Maple-ohjelmiston kotisivu, `http://www.maplesoft.com/products/Maple/index.aspx`

[3] Mathematica-ohjelmiston kotisivu, `http://www.wolfram.com/products/mathematica/`

[4] MATLAB-ohjelmiston kotisivu, `http://www.mathworks.com/products/matlab/`

[5] GeoGebra-ohjelmiston kotisivu, `http://www.geogebra.org`

[6] GEONExT-ohjelmiston kotisivu, `http://geonext.uni-bayreuth.de`

[7] Cabri-ohjelmiston kotisivu, `http://www.cabri.com`

[8] Geometers' Sketchpad -ohjelmiston kotisivu, `http://dynamicgeometry.com`

[9] Maxima-ohjelmiston kotisivu, `http://maxima.sourceforge.net/`

[10] SymPy, verkkosivusto, `http://www.sympy.org`

[11] PyDSTool, verkkosivusto, `http://sourceforge.net/projects/pydstool/`

[12] Kilpatrick, J., Swafford, J. & Findell, B. *Adding It Up: Helping Children Learn Mathematics* (Washington, DC: National Academy Press, 2001)

[13] Heid, M. K. Resequencing skills and concepts in applied calculus using the computer as a tool. *Journal for Research in Mathematics Education* **19(1)**, 3–25 (1988)

[14] Hillel, J. Computer algebra systems as cognitive technologies: Implications for the practice of mathematics education. In Ruthven, C. K. . K. (ed.) *Learning from computers: Mathematics education and technology*, 18–47 (Berlin: Springer-Verlag, 1993)

[15] Fey, J. Technology and mathematics education: A survey of recent developments and important problems. *Educational Studies in Mathematics* **20**, 237–272 (1989)

[16] Kieran, C. & Damboise, C. "How Can We Describe the Relation between the Factored Form and the Expanded Form of These Trinomials? - We Don't even Know If Our Paper-and-Pencil Factorizations are Right": The Case for Computer Algebra Systems (CAS) with Weaker Algebra Students in PME Conference, 2007, 31; vol. 3, p. 105-112. Proceedings of the 31st Conference of the International Group for the Psychology of Mathematics Education. Saatavilla verkossa: `http://www.emis.de/proceedings/PME31/3/105.pdf`

[17] Porzio, D. Effects of differing emphases in the use of multiple representations and technology on students' understanding of calculus concepts. *Focus on Learning Problems in Mathematics* **21(3)**, 1–29 (1999)

[18] Pierce, R. & Stacey, K. A framework for monitoring progress and planning teaching towards effective use of computer algebra systems. *International Journal of Computers for Mathematical Learning* **9**, 59–93 (2004)

[19] Mayes, R. Current state of research into cas in mathematics education. In Berry J., K. . M. J., Kronfellner M. (ed.) *The State of Computer Algebra in Mathematics Education*, 171–189 (Chartwell-Bratt, Bromley, 1997)

[20] Nabb, K. A. CAS as a restructuring tool in mathematics education. Proceedings of the 22nd International Conference on Technology in Collegiate Mathematics, Chicago IL, 2010. Saatavilla verkossa: `http://www.keithnabb.com/yahoo_site_admin/assets/docs/CAS_As_A_Restructuring_Tool_in_Mathematics_Education.28990404.pdf`

[21] Keller, B. A. & Russell, C. A. Effects of the ti-92 on calculus students solving symbolic problems. *The International Journal of Computer Algebra in Mathematics Education* **4(1)**, 77–98 (1997)

[22] Repo, S. Understanding and reflective abstraction: Learning the concept of derivative in a computer environment. *International DERIVE Journal* **1(1)**, 97–113 (1994)

[23] Drijvers, P. The concept of parameter in a computer algebra environment. In Chick, S. K. V. J. . V. J., H. (ed.) *Proceedings of the 12th ICMI study conference the future of the teaching and learning of algebra*, vol. 1, 221–227 (Melbourne: The University of Melbourne., 2003)

[24] History and License, Sage Reference v4.7, verkkosivu, `http://www.sagemath.org/doc/reference/history_and_license.html`

[25] Beezer, R. A. Sage (version 3.4). SIAM Review, 51(4), 785-807, (2009). Preprint-versio saatavilla verkossa: `http://buzzard.ups.edu/bookreview/sage-beezer-review.pdf`

[26] Eröcal, B. & Stein, W. The sage project: Unifying free mathematical software to create a viable alternative to magma, maple, mathematica and matlab. ICMS 2010: Proceedings of the Third International Congress on Mathematical Software. Springer, Lecture Notes in Computer Science, 6327, 12-27, (2010). Saatavilla verkossa: `http://www.wstein.org/papers/icms/icms_2010.pdf`

[27] Components, Sagen verkkosivut, `http://www.sagemath.org/links-components.html`

[28] Schools using python, Python wiki -verkkosivusto, `http://wiki.python.org/moin/SchoolsUsingPython`

[29] Installation, Sage Tutorial v4.7, verkkosivu, `http://www.sagemath.org/doc/tutorial/introduction.html#installation`

[30] Interactive Shell, Sage Tutorial v4.7, verkkosivu, http://www.sagemath.org/doc/tutorial/interactive_shell.html

[31] The Sage Notebook, Sage Reference v4.7, verkkosivu, http://sagemath.org/doc/reference/notebook.html

[32] The Sage Notebook, verkkosivu, http://nb.sagemath.org/

[33] Interact Functions in the Notebook, Sage Reference v4.7, verkkosivu, http://sagemath.org/doc/reference/sagenb/notebook/interact.html

[34] Sage-edu -keskusteluryhmä Google Groups -palvelussa, verkkosivu, http://groups.google.com/group/sage-edu

[35] Notebook Keybindings, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sagenb/notebook/config.html

[36] Mathematical Constants, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/symbolic/constants.html

[37] N-metodi, Symbolic Expressions, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/symbolic/expression.html#sage.symbolic.expression.Expression.N

[38] Symbolic Expression, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/symbolic/expression.html

[39] Plot-metodi, 2D-plotting, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/plot/plot.html#sage.plot.plot.plot

[40] Show-metodi, 2D-plotting, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/plot/plot.html#sage.plot.plot.Graphics.show

[41] Text in plots, 2D Graphics, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/plot/text.html

[42] Mark Lutz, D. A. *Learning Python* (O'Reilly Media, 1999)

[43] Pilgrim, M. *Dive Into Python* (Apress, saatavilla verkossa: http://diveintopython.org/, 2004)

[44] w3schools, verkkosivusto, http://www.w3schools.com/

[45] Oetiker, T., Partl, H., Hyna, I. & Schlegl, E. *Pitkänpuoleinen johdanto LaTeX 2e:n käyttöön* (suomennos Timo Hellgren, 2005, saatavilla verkossa: ftp://ftp.funet.fi/pub/TeX/CTAN/info/lshort/finnish/lyhyt2e.pdf)

[46] Data Structures, Python v2.7.2 documentation, verkkosivu, http://docs.python.org/tutorial/datastructures.html

[47] Using Python as a Calculator, Python v2.7.2 documentation, verkkosivu, http://docs.python.org/tutorial/introduction.html#using-python-as-a-calculator

[48] Sets, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/sets/set.html

[49] Mutable Sequence Types, Built-in Types, Python v2.7.2 documentation, verkkosivu, http://docs.python.org/library/stdtypes.html#mutable-sequence-types

[50] Function definitions, Compound statements, Python v2.7.2 documentation, verkkosivu, http://docs.python.org/reference/compound_stmts.html#function-definitions

[51] Compound statements, Python v2.7.2 documentation, verkkosivu, http://docs.python.org/reference/compound_stmts.html

[52] File Objects, Built-in Types, Python v2.7.2 documentation, verkkosivu, http://docs.python.org/library/stdtypes.html#file-objects

[53] Factorial, Miscellaneous arithmetic functions, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/rings/arith.html#sage.rings.arith.factorial

[54] Animated plots, 2D Graphics, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/plot/animate.html

[55] Interact-metodi, Interact Functions in the Notebook, Sage Reference v4.7, verkkosivu, http://sagemath.org/doc/reference/sagenb/notebook/interact.html#sagenb.notebook.interact.interact

[56] Some Common Issues with Functions, Sage Tutorial v4.7, verkkosivu, http://www.sagemath.org/doc/tutorial/tour_functions.html

[57] Piecewise-defined Functions, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/functions/piecewise.html

[58] Parametric_plot-metodi, 2D Graphics, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/plot/plot.html#sage.plot.plot.parametric_plot

[59] Symbolic Equations and Inequalities, Symbolic Calculus, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/symbolic/relation.html

[60] Find all roots of a trigonometric equation, Sage Development Organization Page, verkkosivu, http://trac.sagemath.org/sage_trac/ticket/8390

[61] Line Plots, 2D Graphics, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/plot/line.html

[62] Points, 2D Graphics, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/plot/point.html

[63] Circles, 2D Graphics, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/plot/circle.html

[64] Platonic Solids, 3D Graphics, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/plot/plot3d/platonic.html

[65] Classes for Lines, Frames, Rulers, Spheres, Points, Dots, and Text; 3d Graphics, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/plot/plot3d/shapes2.html

[66] Base classes for 3D Graphics objects and plotting, 3D Graphics, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/plot/plot3d/base.html

[67] Limit-metodi, Symbolic Calculus, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/calculus/calculus.html#sage.calculus.calculus.limit

[68] Derivative-metodi, Symbolic Expressions, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/symbolic/expression.html#sage.symbolic.expression.Expression.derivative

[69] Polynomial Rings, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/polynomial_rings.html

[70] Integrate-metodi, Symbolic Expressions, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/symbolic/expression.html#sage.symbolic.expression.Expression.integrate

[71] Assume-metodi, Symbolic Expressions, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/symbolic/expression.html?highlight=assume#sage.symbolic.expression.Expression.assume

[72] Nintegrate-metodi, Symbolic Expressions, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/calculus/calculus.html#sage.calculus.calculus.nintegrate

[73] Matplotlib: Python plotting - Matplotlib v1.0.1. documentation, verkkosivu, http://matplotlib.sourceforge.net/

[74] Scatter Plots, 2D Graphics, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/plot/scatter_plot.html

[75] SciPyPackages/Stats, verkkosivu, http://www.scipy.org/SciPyPackages/Stats

[76] random-luokka, Numeric and Mathematical Modules, Python v2.7.2 documentation, verkkosivu, http://docs.python.org/library/random.html

[77] Random variables and probability spaces, Probability, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/probability/random_variable.html

[78] Heikki Ruskeepää: Todennäköisyyslaskenta, luentomoniste, Turun yliopisto 1996

[79] Bar Charts, 2D Graphics, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/plot/bar_chart.html

[80] Root Finding Using Bisection by William Stein, Sage Interactions - Calculus, verkkosivu, http://wiki.sagemath.org/interact/calculus#Root_Finding_Using_Bisection

[81] Numerical integrals with various rules by Nick Alexander (based on the work of Marshall Hampton), Sage Interactions - Calculus, verkkosivu, http://wiki.sagemath.org/interact/calculus#Numerical_integrals_with_various_rules

[82] Taylor Series by Harald Schilly, Sage Interactions - Calculus, verkkosivu, http://wiki.sagemath.org/interact/calculus#Taylor_Series

[83] Power series, Calculus, Sage Constructions v4.7, verkkosivu, http://www.sagemath.org/doc/constructions/calculus.html#power-series

[84] Solving ordinary differential equations, Symbolic Calculus, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/calculus/desolvers.html

[85] Plotting fields, 2D Graphics, Sage Reference v4.7, verkkosivu, http://www.sagemath.org/doc/reference/sage/plot/plot%_field.html

[86] NumPy Reference, verkkosivu, http://docs.scipy.org/doc/numpy/reference/

[87] Linear algebra, NumPy Reference, verkkosivu, http://docs.scipy.org/doc/numpy/reference/routines.linalg.html

[88] NumPy:n solve-metodi, Linear algebra, NumPy Reference, verkkosivu, http://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.solve.html

[89] Tentative NumPy Tutorial, verkkosivu, http://www.scipy.org/Tentative_NumPy_Tutorial

[90] Taylorin polynomi, MatTa-projekti, Simo K. Kivelä, verkkosivu, http://matta.hut.fi/webMathematica/matta/taylor.jsp

[91] Ladder Problems, Jürgen Köller 2004, verkkosivu, http://www.mathematische-basteleien.de/ladder.htm

[92] Weideman, J. A. C. Numerical integration of periodic functions: A few examples. *The American Mathematical Monthly* **109 (1)**, 21–36 (2002)

[93] Cruz-Uribe, D. & Neugebauer, C. J. Sharp error bounds for the trapezoidal rule and simpson's rule. *Journal of Inequalities in Pure and Applied Mathematics* (2002)

A Ohjelmakoodeja

A.1 Kolmion merkilliset pisteet

Seuraavan ohjelmakoodin avulla voidaan tutkia kolmion merkillisiä pisteitä. Ohjelman tarkempi esittely on luvussa 4.5.

Sovellusta on mahdollista muokata esimerkiksi siten, että kolmion kulmapisteiden koordinaatit xy[i] syötetään ohjelmalle suoraan sen sijaan, että kulmapisteet määräytyvät yksikköympyrän kehältä astelukujen perusteella.

Ohjelma 20: Kolmion merkilliset pisteet

```
xy = [0]*3

# Palauttaa kulman <A[a],A[c],A[b] puolittajasuoran ja yksikköympyrän
    leikkauspisteen. Kulmat annetaan radiaaneissa.
def puol(A, a, b, c):
   if (A[a] < A[b] and (A[c] < A[a] or A[c] > A[b])) or (A[a] > A[b] and (A[c] >
       A[a] or A[c] < A[b])):
      p = A[a] + (A[b]-A[a])/2.0
   else:
      p = A[b] + (2*pi-(A[b]-A[a]))/2.0
   return (cos(p), sin(p))

# Palauttaa pisteiden välisen etäisyyden.
def etaisyys((x1,y1), (x2,y2)):
   return sqrt((x2-x1)^2 + (y2-y1)^2)

# Palauttaa kahden pisteen kautta kulkevan suoran yhtälön GraphicPrimitive-objektin.
def suora((x1,y1), (x2,y2), vari=None, koko=1):
   return plot((y2-y1)/(x2-x1)*(x-x1)+y1, x, rgbcolor=vari, thickness=koko)

@interact
def merkilliset(a0 = slider(0,360,1,30,label="A"),
                a1 = slider(0,360,1,180,label="B"),
                a2 = slider(0,360,1,300,label="C"),
                nayta_keskijanat = checkbox(False, label="Keskijana"),
                nayta_keskinormaalit = checkbox(False, label="Keskinormaalit"),
                nayta_korkeusjanat = checkbox(False, label="Korkeusjanat"),
                nayta_kulmanpuolittajat = checkbox(False,
                    label="Kulmanpuolittajat"),
                nayta_sisaympyra = checkbox(False, label="Sisäympyrä "),
                nayta_euler = checkbox(False, label="Eulerin suora ")):
```

```
# Kulmien koordinaatit.
a = [math.radians(a0), math.radians(a1), math.radians(a2)]
for i in range(3):
   xy[i] = (cos(a[i]), sin(a[i]))

# Kulmien tunnukset.
a_tunnus = text("A", (xy[0][0]*1.07, xy[0][1]*1.07))
b_tunnus = text("B", (xy[1][0]*1.07, xy[1][1]*1.07))
c_tunnus = text("C", (xy[2][0]*1.07, xy[2][1]*1.07))
tunnukset = a_tunnus + b_tunnus + c_tunnus

# Yksikköympyrä.
ympyra = circle((0,0), 1, aspect_ratio=1)

# Kolmio.
kolmio = line([xy[0], xy[1], xy[2], xy[0]], rgbcolor="black")
kolmio_pisteet = point(xy, pointsize=30)

# Kolmion kulmia (a,b,c) vastaavien sivujen pituudet (bc, ca, ab).
al = [etaisyys(xy[1], xy[2]), etaisyys(xy[2], xy[0]), etaisyys(xy[0], xy[1])]

# Kolmion kulmia (a,b,c) vastaavien sivujen keskipisteet.
a_kesk = [((xy[1][0]+xy[2][0])/2.0, (xy[1][1]+xy[2][1])/2.0),
    ((xy[2][0]+xy[0][0])/2.0,(xy[2][1]+xy[0][1])/2.0),
    ((xy[0][0]+xy[1][0])/2.0,(xy[0][1]+xy[1][1])/2.0)]

# Kolmion sisälle piirretyn ympyrän keskipiste.
piiri = al[0] + al[1] + al[2]
sisa_keskipiste = ((al[0]*xy[0][0]+al[1]*xy[1][0]+al[2]*xy[2][0])/piiri,
    (al[0]*xy[0][1]+al[1]*xy[1][1]+al[2]*xy[2][1])/piiri)

# Kolmion sisälle piirretty ympyrä.
if nayta_sisaympyra:
   s = piiri/2.0
   sisa_sade = sqrt((s-al[0])*(s-al[1])*(s-al[2])/s)
   sisa_ympyra = circle(sisa_keskipiste, sisa_sade)
else:
   sisa_ympyra = Graphics()

# Kulmanpuolittajat.
if nayta_kulmanpuolittajat:
   a_kp = line([xy[0],puol(a,1,2,0)], rgbcolor="blue")
   b_kp = line([xy[1],puol(a,2,0,1)], rgbcolor="blue")
   c_kp = line([xy[2],puol(a,0,1,2)], rgbcolor="blue")
```

```
        kp_piste = point(sisa_keskipiste, rgbcolor="blue", pointsize=28)
        kulmanpuolittajat = a_kp + b_kp + c_kp + kp_piste
    else:
        kulmanpuolittajat = Graphics()

    # Keskijanat.
    if nayta_keskijanat:
        a_kj = line([xy[0], a_kesk[0]], rgbcolor="green")
        b_kj = line([xy[1], a_kesk[1]], rgbcolor="green")
        c_kj = line([xy[2], a_kesk[2]], rgbcolor="green")
        kj_piste = point(((xy[0][0]+xy[1][0]+xy[2][0])/3.0,
            (xy[0][1]+xy[1][1]+xy[2][1])/3.0), rgbcolor="green", pointsize=28)
        keskijanat = a_kj + b_kj + c_kj + kj_piste
    else:
        keskijanat = Graphics()

    # Keskinormaalit.
    if nayta_keskinormaalit:
        a_kn = suora(a_kesk[0], puol(a, 1, 2, 0), "red")
        b_kn = suora(a_kesk[1], puol(a, 2, 0, 1), "red")
        c_kn = suora(a_kesk[2], puol(a, 0, 1, 2), "red")
        kn_piste = point((0,0), rgbcolor="red", pointsize=28)
        keskinormaalit = a_kn + b_kn + c_kn + kn_piste
    else:
        keskinormaalit = Graphics()

    # Korkeusjanat.
    if nayta_korkeusjanat:
        xA = xy[0][0]; xB=xy[1][0]; xC=xy[2][0]
        yA = xy[0][1]; yB=xy[1][1]; yC=xy[2][1]
        a_koj = plot(((xC-xB)*x+(xB-xC)*xA)/(yB-yC)+yA, x, rgbcolor="brown")
        b_koj = plot(((xA-xC)*x+(xC-xA)*xB)/(yC-yA)+yB, x, rgbcolor="brown")
        c_koj = plot(((xB-xA)*x+(xA-xB)*xC)/(yA-yB)+yC, x, rgbcolor="brown")
        koj_lx = (xA*xB*(yA-yB) + xB*xC*(yB - yC) + xC*xA*(yC-yA) -
            (yA-yB)*(yB-yC)*(yC-yA))/(xC*yB - xB*yC + xA*yC - xC*yA + xB*yA - xA*yB)
        koj_ly = (yA*yB*(xA-xB) + yB*yC*(xB-xC) + yC*yA*(xC-xA) -
            (xA-xB)*(xB-xC)*(xC-xA))/(yC*xB - yB*xC + yA*xC - yC*xA + yB*xA - yA*xB)
        koj_leikkauspiste = point((koj_lx,koj_ly), rgbcolor="brown", pointsize=28)
        korkeusjanat = a_koj + b_koj + c_koj + koj_leikkauspiste
    else:
        korkeusjanat = Graphics()
```

```
    # Eulerin suora.
    if nayta_euler:
        euler_suora = suora((0,0), ((xy[0][0]+xy[1][0]+xy[2][0])/3.0,
            (xy[0][1]+xy[1][1]+xy[2][1])/3.0), "purple", 2)
    else:
        euler_suora = Graphics()

    # Yhdistetään ja esitetään kaikki piirretyt kuvaajat.
    show(ympyra + kolmio + kolmio_pisteet + tunnukset + kulmanpuolittajat +
        keskijanat + keskinormaalit + korkeusjanat + sisa_ympyra + euler_suora,
        figsize=[5,5], xmin=-1, xmax=1, ymin=-1, ymax=1)
```

A.2 Derivoinnin osaamista testaava opetusohjelma, jossa tehtävät luetaan tiedostosta

Derivoinnin osaamista harjoittavia ohjelmia käsitellään luvussa 5.2.2. Seuraava ohjelma lukee derivoitavat funktiot tiedostosta *funktioita.txt,* joka on liitetty työarkin yhteyteen.

Työarkin yhteyteen talletettuihin tiedostoihin viitataan DATA-avainsanalla.

Ohjelma 21: Derivoinnin osaamista testaava opetusohjelma (funktiot luetaan tiedostosta)

```
# Luetaan funktiot tiedostosta.
funktiot = [l.strip() for l in file(DATA+"funktioita.txt", "rt")]
# Alustetaan apumuuttujat.
funktio(x) = funktiot[0]
laskuri = 0
oikein = 0
uusitehtava = True
naytaratkaisu = True

@interact
def _(vastaus = input_box(),
      uusi = checkbox(default = False, label = "Uusi tehtävä"),
      ratkaisu=checkbox(default = False, label = "Näytä vastaus")):
    # Globaalit muuttujat.
    global funktiot, funktio, laskuri, oikein, uusitehtava, naytaratkaisu

    if laskuri < len(funktiot):
        if vastaus == derivative(funktio(x)):
            html("$\\text{Oikein!}$")
```

```
            oikein += 1
            uusitehtava = not uusi
        elif vastaus and ratkaisu != naytaratkaisu and uusi != uusitehtava:
            html("$\\text{Virhe. Yritä uudestaan!}$")
        if uusi == uusitehtava:
            uusitehtava = not uusi
            laskuri += 1
            if laskuri < len(funktiot):
                html("$\\text{Uusi tehtävä:}$")
                funktio(x) = funktiot[laskuri]
    if laskuri == len(funktiot):
        html("$\\text{Tehtävät loppuivat. Sait %s/%s
            oikein!}$"%(oikein,len(funktiot)))
    else:
        html("$\\text{Derivoi: }\;\;\;\;%s$<br>"%latex(funktio(x)))
        if ratkaisu == naytaratkaisu:
            naytaratkaisu = not ratkaisu
            html("$\\text{Vastaus: }\;\;\;\;%s$<br>"%latex(derivative(funktio(x))))
```

A.3 Puolisuunnikasmenetelmä

Puolisuunnikasmenetelmän toimintaa havainnollistava ohjelma on laajennettu alunperin Marshall Hamptonin ja Nick Alexanderin kirjoittamasta sovelluksesta, joka demonstroi erilaisten suorakulmiomenetelmien käyttöä. [81]

Ohjelma 22: Puolisuunnikasmenetelmä

```
@interact
def puolisuunnikas(f = input_box(default = x^2-5*x + 10, type=SR),
                   n = slider(1, 100, 1, 5),
                   vali = input_box(default=(0, 10), label ="Integrointiväli")):
    xs = []
    ys = []
    h = (vali[1] - vali[0])/n
    f(x) = f
    puolisuunnikkaat = Graphics()

    # Piirretään puolisuunnikkaat.
    for i in range(n):
        xi = vali[0] + i*h
        yi = f(xi)
        puolisuunnikkaat += line([[xi, 0], [xi, yi], [xi + h, f(xi + h)], [xi + h,
            0],[xi, 0]], rgbcolor = (1,0,0))
```

```
        xs.append(xi)
        ys.append(yi)

    xs.append(xi + h)
    ys.append(f(xi + h))

    # Esitetään funktion käyrä ja puolisuunnikkaat kuvaajassa.
    show(plot(f, vali[0], vali[1]) + puolisuunnikkaat, xmin = vali[0], xmax =
        vali[1])

    # Lasketaan määrätyn integraalin approksimaatio puolisuunnikasmenetelmän ja
        Sagen numeerisen metodin avulla.
    numeerinen_arvo = integral_numerical(f, vali[0], vali[1])[0]
    approksimaatio = h *(ys[0]/2 + sum([ys[i] for i in range(1,n)]) + ys[n]/2)

    # Esitetään approksimaation laskemisen välivaiheet.
    summa_kaava_html = "d \cdot \\left[\\frac{f(x_0)}{2} + %s + \\frac{f(x_{%s})}{2}
        \\right]" %(join([ "f(x_{%s})"%i for i in range(1,n)], " + "), n)
    summa_sijoitus_html = "%s \cdot \\left[\\frac{f(%s)}{2} + %s + \\frac{f(%s)}{2}
        \\right]" %(h, N(xs[0], digits=5), join([ "f(%s)" %N(i, digits=5) for i in
        xs[1:-1]]," + "), N(xs[n], digits=5))
    summa_arvot_html = "%s \cdot \\left[\\frac{%s}{2} + %s + \\frac{%s}{2}\\right]"
        %(h, N(ys[0], digits=5), join([ "%s" % N(i, digits=5) for i in ys[1:-1]], "
        + "), N(ys[n], digits=5))

    html(r'''
    <div class="math">
    \begin{align*}
    \int_{%s}^{%s} {f(x) \, dx} & = %s \\\
    \\\
    \int_{%s}^{%s} {f(x) \, dx}
        & \approx %s \\\
        & = %s \\\
        & = %s \\\
        & = %s
    \end{align*}
    </div>
    ''' % (vali[0], vali[1], N(numeerinen_arvo, digits=7), vali[0], vali[1],
        summa_kaava_html, summa_sijoitus_html, summa_arvot_html, N(approksimaatio,
        digits=7)))
```

A.4 Simpsonin menetelmä

Seuraava Simpsonin menetelmän toimintaa kuvaava ohjelma on laajennettu alunperin Marshall Hamptonin ja Nick Alexanderin kirjoittamasta erilaisia suorakulmiomenetelmiä havainnollistavasta sovelluksesta. [81]

Ohjelma 23: Simpsonin menetelmä

```
# Sellaisen paraabelin funktio, joka lukee pisteiden a, b ja c kautta.
def paraabeli(a, b, c):
   A, B, C, x = var("A, B, C, x")
   K = solve([A*a[0]^2+B*a[0]+C==a[1], A*b[0]^2+B*b[0]+C==b[1],
       A*c[0]^2+B*c[0]+C==c[1]], [A, B, C], solution_dict=True)[0]
   f=K[A]*x^2+K[B]*x+K[C]
   return f

@interact
def simpson(f = input_box(default = x*sin(x)+x+1),
             n = slider(2,100,2,6),
             vali = input_box(default=(0,10), label="Integrointiväli")):
   f(x) = f
   xs = []; ys = []
   dx = (vali[1]-vali[0])/n

   # Piirretään paraabelit.
   for i in range(n+1):
      xs.append(vali[0] + i*dx)
      ys.append(f(xs[-1]))
   paraabelit = Graphics()
   viivat = Graphics()
   for i in range(0, n-1, 2):
      p(x) = paraabeli((xs[i], ys[i]), (xs[i+1], ys[i+1]), (xs[i+2], ys[i+2]))
      paraabelit += plot(p(x), x, xmin=xs[i], xmax=xs[i+2], color="red")
      viivat += line([(xs[i], ys[i]), (xs[i], 0), (xs[i+2], 0)], color="red")
      viivat += line([(xs[i+1], ys[i+1]), (xs[i+1], 0)], linestyle="-.",
          color="red")
   viivat += line([(xs[-1],ys[-1]), (xs[-1],0)], color="red")

   # Esitetään funktion käyrä ja paraabelit kuvaajassa.
   show(plot(f(x), x, vali[0], vali[1]) + paraabelit + viivat, xmin = vali[0], xmax
       = vali[1])
```

```
# Lasketaan määrätyn integraalin approksimaatio Simpsonin menetelmän ja Sagen
    numeerisen metodin avulla.
numeerinen_arvo = integral_numerical(f,vali[0],vali[1])[0]
approksimaatio = dx/3 *(ys[0] + sum([4*ys[i] for i in range(1,n,2)]) +
    sum([2*ys[i] for i in range(2, n, 2)]) + ys[n])

# Esitetään approksimaation laskemisen välivaiheet.
summa_kaava_html = "\\frac{d}{3} \cdot \\left[ f(x_0) + %s + f(x_{%s})\\right]"
    %(join([ "%s \cdot f(x_{%s})" %(i%2*(-2)+4, i+1) for i in range(0,n-1)], " +
    "), n)
summa_sijoitus_html = "\\frac{%s}{3} \cdot \\left[ f(%s) + %s + f(%s)\\right]"
    %(dx, N(xs[0], digits=5), join([ "%s \cdot f(%s)" %(i%2*(-2)+4, N(xk,
    digits=5)) for i, xk in enumerate(xs[1:-1])], " + "), N(xs[n], digits=5))
summa_arvot_html = "\\frac{%s}{3} \cdot \\left[ %s %s %s\\right]" %(dx, "%s +
    "%N(ys[0], digits=5), join([ "%s \cdot %s" %(i%2*(-2)+4, N(yk, digits=5))
    for i, yk in enumerate(ys[1:-1])], " + "), " + %s"%N(ys[n],digits=5))

html(r'''
<div class="math">
\begin{align*}
\int_{%s}^{%s} {f(x) \, dx} & = %s \\\
\\\
\int_{%s}^{%s} {f(x) \, dx}
   & \approx %s \\\
   & = %s \\\
   & = %s \\\
   & = %s
\end{align*}
</div>
''' % (vali[0], vali[1], N(numeerinen_arvo, digits=7), vali[0], vali[1],
    summa_kaava_html, summa_sijoitus_html, summa_arvot_html, N(approksimaatio,
    digits=7)))
```

B Tehtävien ratkaisut

Tehtävä 3-1:

a)

```
solve(7*x^7 + 6*x^6 == 0, x)
```

```
[x == (-6/7), x == 0]
```

Vastaus: $x = -\frac{6}{7}$ tai $x = 0$

b)

```
f(a) = (sqrt(a) + 1)^2 - a - 1
f(a).full_simplify()
```

```
2*sqrt(a)
```

Vastaus: $2\sqrt{a}$

c)

```
solve(3/(3 - 2*x) < 0, x)
```

```
[[x > (3/2)]]
```

Vastaus: $x > \frac{3}{2}$

Tehtävä 3-2:

Valitaan sovelluksessa funktioiksi *f(x) = x^3 - 2*x* ja *g(x) = -x + 3*, jolloin havaitaan kuvaajien risteävän jossakin kohtaa välillä $[1, 2]$. Helpotetaan leikkauspisteen määrittämistä valitsemalla funktioksi h erotusfunktio $f - g$. Tällöin yhtälö on tosi sillä x:n arvolla, jolla $h(x) = 0$. Suoritetaan haarukointi piirtoväliä säätämällä, kunnes ratkaisu löydetään vähintään yhden desimaalin tarkkuudella. Juuren arvoksi saadaan $x \approx 1,7$.

Saatu ratkaisu voidaan todeta yhtälön ainoaksi juureksi tarkastelemalla funktioiden kulkua. Esimerkiksi derivoimalla jatkuva funktio h saadaan $h'(x) = 3x^2 - 1$, mistä lokaaliksi maksimiksi saadaan piste $x = -\frac{1}{\sqrt{3}}$ ja lokaaliksi minimiksi $x = \frac{1}{\sqrt{3}}$. Funktion arvot ovat molemmissa kriittisissä pisteissä negatiivisia. Funktion arvot ovat negatiivisia, kun $x \leq \frac{1}{\sqrt{3}}$ ja kasvavat monotonisesti, kun $x > \frac{1}{\sqrt{3}}$. Näin ollen funktiolla voi olla enintään yksi nollakohta.

Vastaus: $x \approx 1,7$.

Tehtävä 3-3:

a) Tutkimalla erilaisilla parametrien A, B ja C arvoilla toisen asteen polynomifunktion käyttäytymistä voidaan tehdä esimerkiksi seuraavia havaintoja:

- Paraabelikuvaaja avautuu ylöspäin, jos $A > 0$, ja vastaavasti alaspäin, jos $A < 0$. Jos $A = 0$, funktio redusoituu ensimmäisen asteen funktioksi.
- A vaikuttaa muutosnopeuteen, jolla funktio kasvaa (tai vähenee) paraabelin huipun ympäristössä. Mitä suurempi A:n itseisarvo on, sitä sulkeutuneemmalta paraabelin kuvaaja näyttää.
- Paraabelin huipun sijaintiin sivusuunnassa vaikuttavat parametrit A ja B (suhteessa $x = -\frac{B}{2A}$).
- Vain kerroin B vaikuttaa siihen, missä kulmassa käyrä leikkaa y-akselin.
- Parametri C vaikuttaa huipun korkeuden ja määrittää pisteen, jossa käyrä leikkaa y-akselin.

b) Yhtälöiden juuret ovat

i) 2 ja -1, **ii)** 1,

iii) $-\sqrt{5}$ ja $\sqrt{5}$, **iv)** yhtälöllä ei ole ratkaisua.

c) Tehtävän a-kohdasta tiedetään, että ylöspäin aukeavalla paraabelilla kerroin A on positiivinen. Huipun paikkaan vaikuttaa sekä kerroin A että B. Oikeat kertoimet löytyvät tutkimalla, miten kertoimien muuttaminen vaikuttaa huipun paikkaan. Parametrilla C voidaan vaikuttaa paraabelin huipun korkeuteen ilman, että sen muut ominaisuudet muuttuvat.

Esimerkki ehdot täyttävästä polynomista on $x^2 - 4x$.

Tehtävä 3-4:

Suorien yhtälöt ovat siis $y_{12} = \frac{y_1 - y_2}{x_1 - x_2}(x - x_1) + y_1$ ja $y_{ab} = \frac{y_a - y_b}{x_a - x_b}(x - x_a) + y_a$.

Määritellään tarvittavat muuttujat ja merkitään suorien y-koordinaatit yhtä suuriksi. Ratkaistaan saadusta yhtälöstä x:

```
x1, x2, y1, y2, x_a, x_b, y_a, y_b, x, y = var("x1, x2, y1, y2, x_a,
   x_b, y_a, y_b, x, y")
show(solve((y1-y2)/(x1-x2)*(x-x1) + y1 == (y_a-y_b)/(x_a-x_b)*(x-x_a)
   + y_a, x))
```

$$\left[x = \frac{(x_1-x_2)x_a y_b-(x_1-x_2)x_b y_a+(x_2 x_a-x_2 x_b)y_1-(x_1 x_a-x_1 x_b)y_2}{(x_a-x_b)y_1-(x_a-x_b)y_2-(x_1-x_2)y_a+(x_1-x_2)y_b}\right]$$

Sekä osoittajasta että nimittäjästä voidaan erottaa yhteisiä tekijöitä, jolloin ratkaisu sievenee:

$$x = \frac{(x_1-x_2)(x_a y_b-x_b y_a)+(x_a-x_b)(x_2 y_1-x_1 y_2)}{(x_a-x_b)(y_1-y_2)-(x_1-x_2)(y_a-y_b)}$$

Suorat ovat yhdensuuntaiset, jos edellisen lausekkeen nimittäjä on nolla. Nyt y-koordinaatti ratkeaa sijoittamalla saatu x:n arvo suoran yhtälöön y_{12}, paitsi jos $x_1 - x_2 = 0$. Tällöin kyseinen suora on y-akselin suuntainen, jolloin leikkauspisteen y-koordinaatti on ratkaistava toisen suoran yhtälöstä. Jos tämäkin suora on y-akselin suuntainen, suorat ovat yhdensuuntaiset eikä ratkaisua ole.

Sijoitetaan x:n arvo ensimmäisen suoran yhtälöön, jolloin y-koordinaatiksi saadaan:

```
y == ((y1-y2)/(x1-x2)*(x-x1)(x =
   ((x1-x2)*(x_a*y_b-x_b*y_a)+(x_a-x_b)*(x2*y1-x1*y2)) /
   ((x_a-x_b)*(y1-y2)-(x1-x2)*(y_a-y_b)))).full_simplify()
```

$$y = -\frac{(x_a-x_b)y_1^2-(x_a-x_b)y_1 y_2+((x_1-x_a)y_1-(x_1-x_a)y_2)y_b-((x_1-x_b)y_1-(x_1-x_b)y_2)y_a}{(x_a-x_b)y_1-(x_a-x_b)y_2-(x_1-x_2)y_a+(x_1-x_2)y_b}$$

Myös tämä lauseke yksinkertaistuu erottamalla yhteisiä tekijöitä:

$$y = -\frac{(x_a-x_b)(y_1^2-y_1 y_2)+(x_1-x_a)(y_1-y_2)y_b-(x_1-x_b)(y_1-y_2)y_a}{(x_a-x_b)(y_1-y_2)-(x_1-x_2)(y_a-y_b)}$$

Tehtävä on esimerkki ongelmatyypistä, jonka tehtävänasettelu on yksinkertainen ja ratkaisun vaiheet helposti ymmärrettävissä, mutta jonka laskeminen olisi käsin työlästä. Ohjelmiston avulla voidaan tilapäisesti sivuuttaa algrebrallisten yksityiskohtien tarkastelu ja keskittyä periaatteseen, jolla ratkaisu löydetään.

Tehtävä 3-5:

Pythagoraan lauseen mukaan $c^2 = (b+x)^2 + (b+y)^2$. Toisaalta tikkaiden ja laatikon väliin jäävät kaksi pienempää kolmiota ovat yhdenmuotoiset (kk-lause), joten saadaan $\frac{x}{b} = \frac{b}{y}$.

Ratkaisemalla jälkimmäisestä yhtälöstä muuttujan y ja sijoittamalla vastauksen edelliseen yhtälöön saadaan

```
b, x, y, c = var("b, x, y, c")
yhtalo = c^2 == (b + x)^2 + (b + y)^2
yhtalo2 = yhtalo(y = b^2/x).full_simplify()
show(yhtalo2)
```

$$c^2 = \frac{b^4+2b^3x+2b^2x^2+2bx^3+x^4}{x^2}$$

Huomataan, että ratkaisu on 4. asteen polynomiyhtälö, kun yhtälö kerrotaan puolittain termillä x^2.

```
yhtalo3 = yhtalo2*x^2
show(yhtalo3)
```

$$c^2x^2 = b^4 + 2b^3x + 2b^2x^2 + 2bx^3 + x^4$$

Sijoitetaan nyt yhtälöön laatikon sivun pituus $b = 1$ ja tikkaiden pituus $c = 5$:

```
yhtalo4 = yhtalo3(b=1, c=5)
show(yhtalo4)
```

$$25x^2 = x^4 + 2x^3 + 2x^2 + 2x + 1$$

Tämä neljännen asteen yhtälö voidaan ratkaista Sagen *solve*-metodin avulla:

```
ratkaisut = solve(yhtalo4, x, solution_dict=True)
show(ratkaisut)
```

$$\left[\left\{x : -\tfrac{1}{2}\sqrt{2\sqrt{26}+23} - \tfrac{1}{2}\sqrt{26} - \tfrac{1}{2}\right\}, \left\{x : \tfrac{1}{2}\sqrt{2\sqrt{26}+23} - \tfrac{1}{2}\sqrt{26} - \tfrac{1}{2}\right\},\right.$$
$$\left.\left\{x : -\tfrac{1}{2}\sqrt{-2\sqrt{26}+23} + \tfrac{1}{2}\sqrt{26} - \tfrac{1}{2}\right\}, \left\{x : \tfrac{1}{2}\sqrt{-2\sqrt{26}+23} + \tfrac{1}{2}\sqrt{26} - \tfrac{1}{2}\right\}\right]$$

Edellä valittiin metodin lisämääreeksi *solution_dict=True*, eli ratkaisut palautetaan hakemiston muodossa. Nyt lista voidaan iteroida alkio kerrallaan ja muuntaa tarkat lausekkeet likiarvoiksi seuraavalla komennolla:

```
[n(i[x]) for i in ratkaisut]
```

```
[-5.93039673076750, -0.168622782825287, 0.260518352903236,
3.83850116068955]
```

Arvot voidaan muuntaa desimaaliluvuiksi myös käyttämällä *n*-metodia yksitellen jokaiseen lausekkeeseen. Havaitaan, että arvoista kaksi on negatiivisia, joten ne hylätään. Tehtävässä pyydettiin määrittämään suurin korkeus, jolla tikkaat koskettavat seinää, joten vastaukseksi saadaan (laatikon korkeus mukaan huomioiden) $1\ m + 3,84\ m = 4,83\ m$.

Tehtävä on esimerkki ongelmasta, jonka asettelu ja ratkaisun vaiheet ovat helposti ymmärrettävissä lukiossa opetettavan matematiikan tiedoin, mutta jonka selvittäminen olisi hyvin vaikeaa, ellei mahdotonta, ilman matemaattisen ohjelmiston käyttöä.

Tehtävän 4. asteen yhtälö on ratkaistavissa myös sijoitusmenetelmän avulla tai trigonometrisena ongelmana. Lisätietoa analyyttisistä ratkaisuista on viitteessä [91].

Vastaus: 4,83 m.

Tehtävä 4-1:

a) Ratkaistaan yhtälöt käyttäen *solve*-metodia parametrin *to_poly_solve* arvolla *"force"*. Näin varmistutaan, että kaikki ratkaisut esitetään.

```
x = var("x")
ratkaisut = solve(sin(x)^2 - 3*cos(x)^2 == 0, x,
   to_poly_solve="force")
show(ratkaisut)
```

$$\left[x = -\frac{1}{3}\pi + \pi z_1,\ \ x = \frac{1}{3}\pi + \pi z_2\right]$$

Vastaus: $\alpha = -\frac{1}{3}\pi + n \cdot \pi$ ja $\alpha = \frac{1}{3}\pi + n \cdot \pi$, missä $n \in \mathbb{Z}$.

b)

```
ratkaisut = solve(2*sin(x)^2 + sin(x) == 0, x,
   to_poly_solve="force")
show(ratkaisut)
```

$$\left[x = 2\pi z_3,\ \ x = -\frac{5}{6}\pi + 2\pi z_4,\ \ x = -\frac{1}{6}\pi + 2\pi z_5,\ \ x = \pi + 2\pi z_6\right]$$

Vastaus: Listan ensimmäinen ja viimeinen ratkaisu, $x = n \cdot 2\pi$ ja $x = \pi n \cdot 2\pi$, ovat yhdistettävissä yhdeksi lausekkeeksi: $x = n \cdot \pi$. Muut ratkaisut ovat $x = -\frac{1}{6}\pi + n \cdot 2\pi$ ja $x = -\frac{5}{6}\pi + n \cdot 2\pi$, missä $n \in \mathbb{Z}$.

c)

```
ratkaisut = solve(2*sin(2*x-pi/2)+sqrt(3)==0, x,
   to_poly_solve="force")
show(ratkaisut)
```

$$\left[x = \frac{1}{12}\pi + \pi z_7,\ \ x = -\frac{1}{12}\pi + \pi z_8\right]$$

Vastaus: $x = \frac{1}{12}\pi + n \cdot \pi$ tai $x = -\frac{1}{12}\pi + n \cdot \pi$, missä $n \in \mathbb{Z}$.

d)

```
ratkaisut = solve(tan(2*x) == tan(x+pi/2), x,
   to_poly_solve="force")
show(ratkaisut)
```

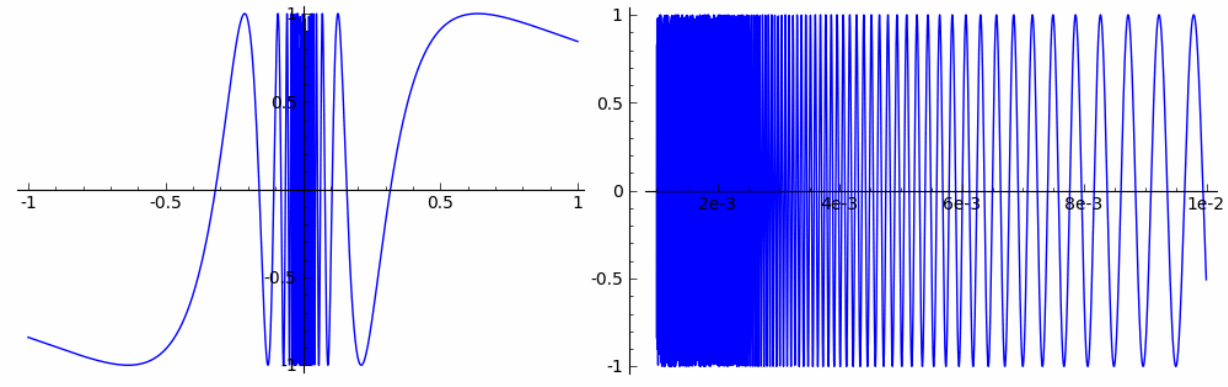
$$[]$$

Vastaus: Yhtälöllä ei ole juuria.

Tehtävä 4-2:

Piirretään ensin funktio $sin\frac{1}{x}$ (kuvassa 42 vasemmalla):

```
plot(sin(1/x))
```



Kuva 42: Funktio $sin\frac{1}{x}$ piirrettynä välillä $[-1,1]$ ja tarkemmin välillä $[\frac{1}{1000}, \frac{1}{100}]$.

Havaitaan, että funktion nollakohtien esiintymistiheys näyttäisi kasvavan, mitä lähemmäksi origoa lähestytään. Tarkastellaan nyt kuvaajaa tarkemmin välillä $[\frac{1}{1000}, \frac{1}{100}]$:

```
plot(sin(1/x), [1/1000, 1/100])
```

Kuvaajasta (kuvassa 42 oikealla) on mahdotonta arvioida nollakohtien määrää. Ratkaistaan yhtälö $\sin\frac{1}{x} = 0$ ohjelman avulla:

```
f(x) = sin(1/x)
show( solve(f(x) == 0, x, to_poly_solve="force") )
```

$$\left[x = \frac{1}{2\pi z_{10}},\ x = \frac{1}{\pi+2\pi z_{11}}\right]$$

Sagen antamat kaksi ratkaisua ovat ekvivalentteja ratkaisun $x = \frac{1}{n\pi}$ kanssa, missä n on nollasta poikkeava kokonaisluku. Lasketaan vielä välillä $[\frac{1}{1000}, \frac{1}{100}]$ olevien juurien lukumäärä. Ratkaisemalla n edellisestä yhtälöstä saadaan $n = \frac{1}{x\pi}$. Selvitetään siis, kuinka monta sellaista $n \in \mathbb{Z}$ on, jotka toteuttavat ehdon $\frac{1}{(1/100)\pi} < n < \frac{1}{(1/1000)\pi}$. Funktio $\frac{1}{x\pi}$ on monotonisesti vähenevä, joten riittää laskea pienin ja suurin muuttujan n arvo, jolle edellä esitetty ehto on voimassa.

Koska

```
print n(1/(1/100*pi))
print n(1/(1/1000*pi))
```

```
31.8309886183791
318.309886183791
```

saadaan, että välillä $[\frac{1}{1000}, \frac{1}{100}]$ on $318 - 31 = 287$ juurta.

Tehtävä 4-3:

a) Säätämällä ohjelman avulla muuttujaa x voidaan funktion $\sin x$ arvo lukea joko yksikköympyrän tai kuvaajan y-akselilta. Yhtälö $\sin x = \frac{1}{2}$ toteutuu välillä $[0, 2\pi]$ muuttujan arvoilla $x \approx 0{,}17\pi$ ja $x \approx 0{,}84\pi$.

   Arvon $\sin(1{,}27\pi)$ näkee parhaiten yksikköympyrän y-akselilta: $\sin 1{,}27\pi \approx -0{,}75$.

b) Valitaan ensin kosinifunktio valikosta. Nyt funktion arvo voidaan lukea yksikköympyrän x-akselilta tai kuvaajan y-akselilta, $\cos \frac{5\pi}{3} = 0{,}5$.

   Yksikköympyrästä nähdään, että $\cos x = 0$, kun $x = 0{,}5\pi$ tai $x = 1{,}5\pi$. Yleisemmin saadaan, että $\cos x = 0$, kun $x = (\frac{1}{2} + n)\pi$ kaikilla $n \in \mathbb{Z}$.

c) Yksikköympyrästä tai kuvaajasta luettuna $\tan \frac{\pi}{4} \approx 1{,}0$ ja $\tan 0{,}35\pi \approx 2{,}0$. Tutkimalla tangenttifunktion kulkua löydetään yhtälön $\tan x = -4$ likimääräisiksi ratkaisuiksi $x \approx 0{,}58\pi$ ja $x \approx 1{,}58\pi$. Havaitaan tangenttifunktion jakson pituuden olevan π.

   Ohjelma antaa virheilmoituksen x:n arvolla $\frac{\pi}{2}$, koska tangenttifunktiota ei ole määritelty tässä pisteessä. Jos tangenttia yritettäisiin määrittää yksikköympyrästä, seuraisi väistämättä, että tangentin määrittävän kolmion kulmista kaksi olisi suoria kulmia, mikä ei ole mahdollista.

Tehtävä 4-4:

Kuution kulma läpäisee puolipallon pinnan juuri ja juuri kuution sivun pituuden ollessa n. 0,82. Näin ollen kuution tilavuuden suhde puolipallon tilavuuteen on ohjelman mukaan 26 %.

Tehtävä 5-1:

Määritetään erotusosamäärän raja-arvo pisteessä a metodin *limit* avulla.

a)

```
x, n, a = var("x, n, a")
f(x) = sin(n*x)
erotusosamaara = (f(x) - f(a))/(x-a)
limit(erotusosamaara, x=a)
```

```
n*cos(a*n)
```

```
derivative(f(x), x)
```

```
n*cos(a*n)
```

Vastaus: $n\cos(xn)$

b)

```
f(x) = ln(n*x)
erotusosamaara = (f(x) - f(a))/(x-a)
limit(erotusosamaara, x=a)
```

```
1/a
```

```
derivative(f(x), x)
```

```
1/x
```

Vastaus: $\frac{1}{x}$

c)

```
f(x) = n^n^x
erotusosamaara = (f(x) - f(a))/(x-a)
limit(erotusosamaara, x=a)
```

```
n^(n^a + a)*log(n)^2
```

```
derivative(f(x), x)
```

```
n^x*n^(n^x)*log(n)^2
```

Vastaus: $n^{n^x+x}(\ln n)^2$

Kaikissa kohdissa erotusosamäärän raja-arvon ja Sagen *derivative*-metodin avulla saadaan sama tulos.

Tehtävä 5-2:

Määritetään ensin funktion f(x) ensimmäinen ja toinen derivaatta Sagen *derivative*-metodin avulla:

```
f(x) = x^5 - x^3 - x^2 - 5*x
print "f'(x) = ", derivative(f(x), x)
print "f''(x) = ", derivative(f(x), x, x)
```

```
f'(x) = 5*x^4 - 3*x^2 - 2*x - 5
f''(x) = 20*x^3 - 6*x - 2
```

Derivaattojen nollakohtia on vaikea määrittää analyyttisesti, koska ensimmäinen derivaatta on neljännen asteen ja toinen derivaatta kolmannen asteen polynomi. Nollakohdat voidaan kuitenkin määrittää likimääräisesti joko kuvaajan perusteella tai käyttämällä *solve*-metodia. Kuvaajasta tehtävää tarkastelua voidaan tarkentaa rajoittamalla tarkasteluväli tilapäisesti derivaattojen nollakohtien ympäristöön.

Kuvaajan perusteella $f'(x) = 0$, kun $x \approx -1{,}06$ tai $x \approx 1{,}25$.

Tutkitaan nyt tarkemmin funktion käyttäytymistä. Todetaan ensin, että polynomifunktiona se on jatkuva. Kohdassa $x \approx -1,06$ on funktion lokaali maksimi ja pisteessä $x \approx 1{,}25$ lokaali minimi. Tämä nähdään paitsi kuvaajasta, myös siitä, että $f''(-1{,}06) < 0$ ja $f''(1{,}25) > 0$, kun ensimmäinen derivaatta näissä pisteissä on nolla. Minimi- ja maksimipisteiden välillä funktio on aidosti vähenevä, muualla aidosti kasvava. Nopeinta väheneminen on muuttujan arvolla $x \approx 0,67$, joka on toisen derivaatan nollakohta ja ensimmäisen derivaatan lokaalinen minimi. Piste $x \approx 0,67$ on myös funktion käännepiste, eli kohta, jossa käyrän kaarevuussuunta vaihtuu: käännepisteeseen piirretyn käyrän tangentti lävistää käyrän.

Tehtävä 5-3:

Lasketaan määrätyt integraalit Sagen *integrate*-metodin avulla:

a)

```
integrate(e^x + 1, x, 0, 1)
```

```
e
```

b)

```
integrate(x^(1/3), x, 0, 1)
```

```
3/4
```

c)

```
integrate(1/sqrt(x+5), x, -1, 4/9)
```

```
2/3
```

d)

```
integrate(1+sin(x), x, 0, pi)
```

```
pi + 2
```

Verrataan vielä saatuja tuloksia numeerisesti laskevan sovelluksen 10 (luvussa 5.3.4) antamiin arvoihin. Valitaan sovelluksessa funktioksi f tehtävässä tarkasteltava integrandi. Toisella funktiolla g ei ole tässä merkitystä ja se voidaan jättää tyhjäksi. Tarkasteluväli voidaan valita samaksi kuin integrointiväli. Vertaamalla saatuja tuloksia keskenään havaitaan, että tulokset vastaavat hyvin toisiaan.

Tehtävä 5-4:

Merkitään sovelluksessa 10 (luvussa 5.3.4) funktioiksi $f(x) = sin(x)$ ja $g(x) = sin(2x)$. Valitaan piirrettäväksi sekä funktio g että erotusfunktio $f - g$. Integrointi- ja tarkasteluväliksi valitaan $[0, \pi]$. Kuvasta havaitaan, että erotusfunktio kulkee aluksi x-akselin alapuolella, kunnes funktioiden f ja g käyrät risteävät siten, että f:n arvot ovat suurempia kuin g:n arvot. Funktioiden rajaamaa pinta-alaa ei saada suoraan laskemalla määrättyä integraalia funktioiden erotuksesta koko tarkasteluvälillä, vaan integraali pitää määrittää osissa.

Ratkaistaan funktioiden leikkauspiste *solve*-metodin avulla:

```
ratkaisu = solve(sin(x)==sin(2*x), x, to_poly_solve="force")
show(ratkaisu)
```

$$\left[x = -\frac{1}{3}\pi + \frac{2}{3}\pi z_2,\ \ x = -2\pi z_1\right]$$

Ratkaisuista 0, $\frac{\pi}{3}$ ja π ovat tarkasteluvälillä. Lasketaan sovelluksen avulla määrättyjen integraalien itseisarvojen summa osaväleillä $[0, \frac{\pi}{3}]$ ja $[\frac{\pi}{3}, \pi]$. Välillä $[0, \frac{\pi}{3}]$ määrätyn integraalin itseisarvoksi saadaan 0,25 ja välillä $[\frac{\pi}{3}, \pi]$ integraalin arvo on 2,25.

Vastaus: 2,5 km^2

Tehtävä 6-1:

Arvosanat järjestetään kasvavaan järjestykseen *sort*-metodin avulla:

```
arvosanat = [8, 7, 5, 7, 8, 8, 9, 10, 4, 7, 7, 8, 4, 6, 10, 9, 5, 8,
   9, 8, 7, 5, 6, 7, 9]
arvosanat.sort()
print arvosanat
```

```
[4, 4, 5, 5, 5, 6, 6, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9,
   10, 10]
```

Määritetään seuraavaksi tehtävässä pyydetyt tunnusluvut. Keskiarvosta, varianssista ja keskihajonnasta lasketaan likiarvot metodin *n* avulla.

```
print "Keskiarvo: ", n(mean(arvosanat), 3)
print "Mediaani: ", median(arvosanat)
print "Moodi: ", mode(arvosanat)
print "Varianssi: ", n(variance(arvosanat), 3)
print "Keskihajonta: ", n(std(arvosanat), 3)
```

```
Keskiarvo: 7.0
Mediaani: 7
Moodi: [7, 8]
Varianssi: 3.0
Keskihajonta: 2.0
```

Tehtävä 6-2:

Laaditaan pylväsdiagrammi pistejakaumasta esimerkin 6.2.1 mukaisesti (kuvassa 43 vasemmalla):

```
from pylab import *
clf()                                # Tyhjentää piirrostilan
figure(figsize=(6,4))                # Määrittää kuvan mittasuhteet
ind = [1, 2, 3, 4, 5, 6, 7]          # Pylväiden paikat x-akselilla
# Pylväiden korkeudet (frekvenssit).
arvot = [2025, 1245, 2186, 792, 673, 908, 1471]
width = 0.5                          # Pylvään leveys (yksikköinä)
labels = ("0", "1", "2", "3", "4", "5", "6")     # Luokkien nimet
```

```
# x-akselin jaotus, muotoa xticks(lista x-koordinaateista, nimet).
xticks([x+width/2.0 for x in ind], labels)
# y-akselin jaotus, muotoa yticks(lista y-koordinaateista).
yticks([0, 300, 600, .. ,2100])
# Pylväsdiagrammi, bar(indeksit, arvot, pylvään leveys, lisämääreet).
bar(ind, arvot, width, color="green")
title("Pistejakauma")     # Kuvan otsikko
xlabel("Pisteet")         # x-akselin nimi
ylabel("Frekvenssi")      # y-akselin nimi
savefig("pylvas.png")     # Tallentaa kuvan solun yhteyteen
```

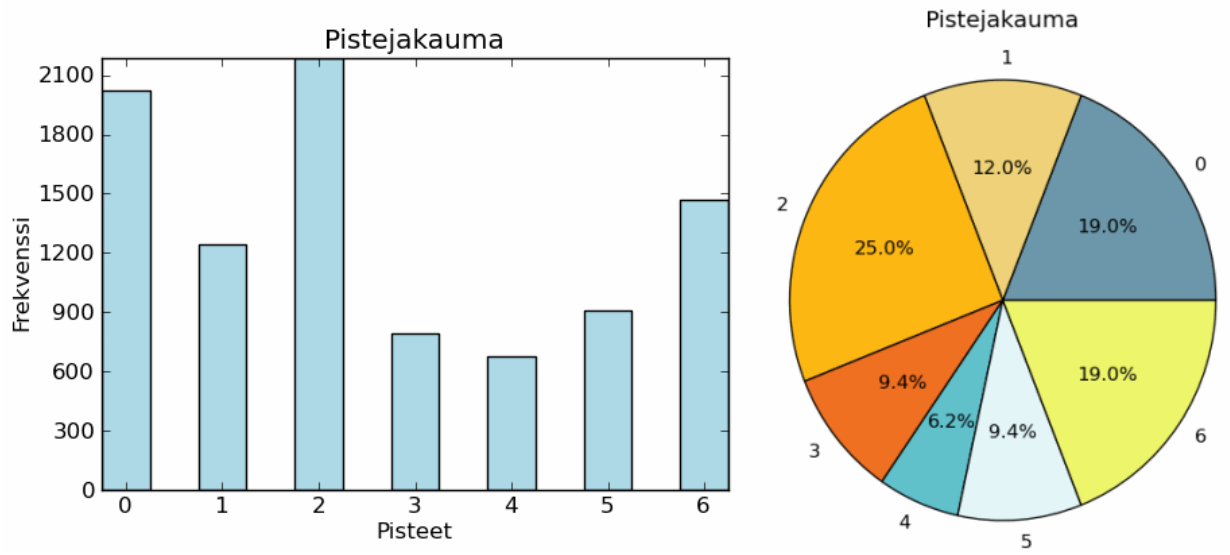
Lasketaan sektoridiagrammia varten pistejakauman suhteelliset osuudet kahden merkitsevän numeron tarkkuudella.

```
arvot = [2025.0, 1245.0, 2186.0, 792.0, 673.0, 908.0, 1471.0]
summa = sum(arvot)
[n(x/summa, 2) for x in arvot]
```

```
[0.19, 0.12, 0.25, 0.094, 0.062, 0.094, 0.19]
```

Sektoridiagrammi voidaan piirtää pistejakaumasta esimerkkiä 6.2.2 mukaillen (kuvassa 43 oikealla):

```
from pylab import *       # Tuo pylab-moduulin metodit
clf()                     # Tyhjentää piirrostilan
figure(figsize=(6,6))     # Määrittää kuvan mittasuhteet
labels = ("0", "1", "2", "3", "4", "5", "6") # Luokkien nimet
osuudet = [19, 12, 25, 9.4, 6.2, 9.4, 19] # Osuudet prosentteina
title("Pistejakauma")     # Kuvan otsikko
# Sektoridiagrammi.
pie(osuudet, labels=labels, colors=["#6D98AB", "#EFD279", "#FDB813",
   "#F17022", "#62C2CC", "#E4F6F8", "#EEF66C"], autopct="%1.1f%%")
savefig("sektori.png") # Tallentaa kuvan solun yhteyteen
```



Kuva 43: Pylväs- ja sektoridiagrammit pistejakaumasta.

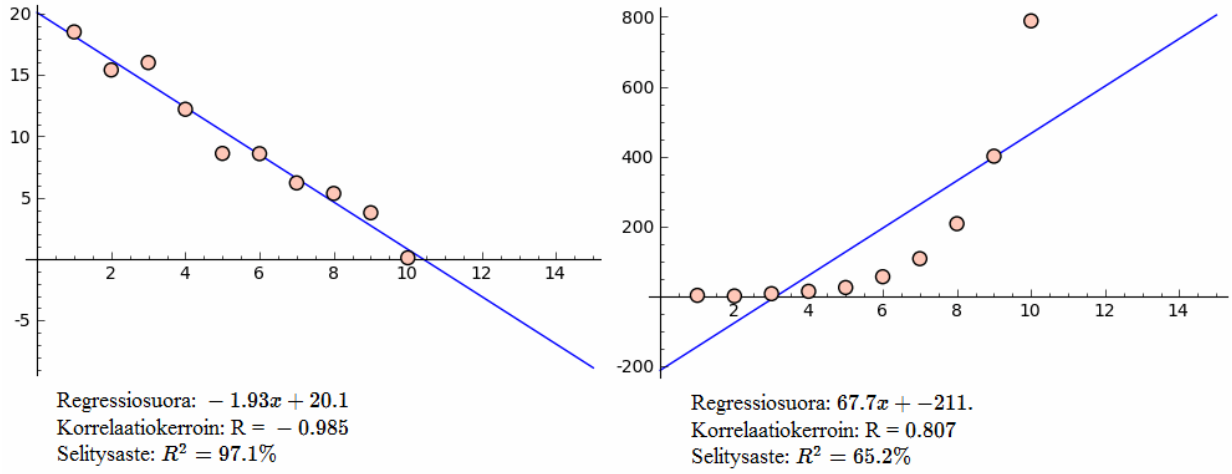
Tehtävä 6-3:

Tutkitaan ensin kvantitaavista aineistoa (x, y_A) esimerkin 6.3.1 ohjelmakoodin avulla:

```
from scipy import stats
v = [(1, 18.5), (2, 15.4), (3, 16.0), (4, 12.2), (5, 8.6), (6, 8.58),
   (7, 6.2), (8, 5.34), (9, 3.77), (10, 0.1)]
k, c, r, p, stderr = stats.linregress([map(float, xy) for xy in v])
regressiosuora = plot(k*x+c,(x,0,15))
sirontakuvio = scatter_plot(v, markersize=50, marker="o",
   facecolor="#fec7b8", edgecolor="black")
show(regressiosuora + sirontakuvio)
html("$\\text{Regressiosuora: } %sx+%s$"%(N(k, digits=3), N(c,
   digits=3)))
html("$\\text{Korrelaatiokerroin: R = } %s$"%(N(r, digits=3)))
html("$\\text{Selitysaste: } R^2 = %s \%%$"%(N(r, digits=3)^2*100.0))
```

Saatu regressiosuora näyttäisi sopivan erinomaisesti havaintoaineistoon (kuvassa 44 vasemmalla). Selitysaste tukee havaintoa negatiivisesta lineaarisesta riippuvuudesta.

Tutkitaan pistejoukkoa (x, y_B) saman esimerkin avulla (kuvassa 44 oikealla). Kuvaajasta ja selitysasteesta havaitaan, että lineaarinen malli kuvaa riippuvuutta heikosti. Kuvaajan perusteella y_B näyttäisi kasvavan x:n suhteen eksponentiaalisesti.



Kuva 44: Havaintojoukkoihin (x, y_A) ja (x, y_B) sovitetut regressiosuorat.

Tehtävä 6-4:

a) Piirtämällä normaalijakauman tiheysfunktion pylväsdiagrammin päälle nähdään, että havaintoarvot ovat normaalisti jakautuneet. Sovellus antaa summalle odotusarvon 17,5, varianssin 14,58 ja keskihajonnan 3,82.

b) Jos noppia heitetään kymmenen, odotusarvo on 35, varianssi 29,17 ja keskihajonta 5,4. Havaitaan, että odotusarvo ja varianssi ovat kaksinkertaiset edelliseen tehtävään nähden. Keskihajonta on varianssin neliö, joten se ei kasva samassa suhteessa.

Tehtävä 7-1:

a) i) Kaikki menetelmät löytävät yhtälön juuren. Valitaan puolitus- ja sekanttimenetelmässä $a = 0$ ja $b = 2$. Kuvaajan mukaan nollakohta sijaitsee näiden pisteiden välissä. Käytettäessä kolmen desimaalin tarkkuutta puolitusmenetelmän suorittaminen vaatii 15 iteraatiota ja sekanttimenetelmä 13 iteraatiota. Newtonin menetelmällä sama tarkkuus saavutetaan huomattavan nopeasti, kun lähtöpiste valitaan nollakohdan lähistöltä. Jos lähtöpisteeksi valitaan $c = 2$, päästään vain neljässä iteraatiossa tarkempaan tulokseen kuin puolitus- ja sekanttimenetelmissä.

Vastaus: $x \approx 1,373$.

ii) Sekanttimenetelmä vaikuttaisi tässä paljon tehokkaammalta kuin puolitusmenetelmä. Newtonin menetelmässä lähtöpisteen valinta vaikuttaa merkittävästi suppenemisnopeuteen eikä menetelmä välttämättä suppene ollenkaan,

jos lähtöpiste on valittu väliltä, jossa funktion kasvunopeus on hidasta, eli derivaatta on lähellä nollaa. Esimerkiksi lähtöarvolla $c = 6$ Newtonin menetelmä ei löydä juurta edes sadassa iteraatiossa, mutta lähtöarvoilla $c = 4$ ja $c = 7$ juuri löytyy alle kymmenessä iteraatiossa.

Vastaus: $x \approx 2,554$.

iii) Jälleen sekanttimenetelmä näyttäisi huomattavasti tehokkaammalta kuin puolitusmenetelmä, samoin Newtonin menetelmä. Newtonin menetelmä suppenee nopeasti juurikaan riippumatta lähtöpisteestä.

Vastaus: $x \approx 0,567$.

b) Koska funktion derivaatta pisteessä $x = 6$ on lähellä nollaa, menetelmä ajautuu heti kauas lähimmästä nollakohdasta, ja suppeneekin kohti juurta $x = -\frac{3}{2}\pi$.

c) Menetelmä jää ikuiseen nelivaiheiseen silmukkaan {0, -3,0, -1,96, -1,147, 0, -3,0, -1,96, ...} eikä löydä yhtälön juurta.

d) Ensimmäisen funktion tapauksessa riippumatta lähtöarvosta menetelmä ajautuu välittömästi nollakeskiseen kaksivaiheiseen silmukkaan {c, -c, c, -c, c, -c, ...}, missä c on lähtöarvo.

Jälkimmäinen funktio yleistää edellisen tapauksen siten, että kaksivaiheinen silmukka alternoi erikoispisteen a ympärillä etäisyydellä, joka vastaa lähtöpisteen c ja erikoispisteen välistä etäisyyttä seuraavasti: {c, 2a-c, c, 2a-c, ...}. Kyseessä on erikoistapaus, jossa Newtonin menetelmä ei suppene riippumatta lähtöarvon valinnasta.

Tehtävä 7-2:

Voidaan osoittaa, että yläraja keskipistemenetelmän virheen itseisarvolle on puolet pienempi kuin puolisuunnikasmenetelmässä, kun integrandin toinen derivaatta on jatkuva. Tehtävässä havaitaan, miten yleisesti ottaen Simpsonin menetelmä toimii parhaiten kaarevien muotojen pinta-alojen approksimoinnissa. Toisaalta puolisuunnikasmenetelmä saattaa supeta hyvin nopeasti jaksollisten funktioiden tapauksessa. [92] Jos integrandin korkeamman asteen derivaatat eivät ole jatkuvia, puolisuunnikasmenetelmä voi supeta nopeammin kuin Simpsonin menetelmä. [93]

Varsinkin pienillä parametrin n arvoilla eri menetelmien välillä voi olla huomattavia eroja. Jos funktiossa on paljon vaihtelua, menetelmien väliset erot tulevat esiin vasta suuremmalla osavälien määrällä eikä luotettaviin arvioihin päästä pienillä n:n arvoilla. Sagen metodi *integral_numerical* antaa tehtävän määrätyille integraaleille seuraavat likiarvot:

a) 1.5 **b)** 12,566 **c)** 37,226

Tehtävä 7-3:

Suppenemisväli nähdään parhaiten kasvattamalla asteluku mahdollisimman suureksi. Havainnot vastaavat hyvin teoriaa: esimerkiksi luonnollisen logaritmifunktion tapauksessa Taylorin polynomi näyttäisi suppenevan välillä $(0, 2x_0)$, missä $x_0 > 0$ on kehityskeskus. Suppenemisalue on siis sitä laajempi, mitä kauempaa funktion erikoispisteestä, $x = 0$, kehityskeskus valitaan.

Tehtävä 7-4:

a) Maxima pystyy määrittämään kaikkien tehtävässä esiintyvien differentiaaliyhtälöiden ratkaisut symbolisesti:

```
t = var('t')
y = function('y', t)
desolve(diff(y,t) - y*t, y, [0, 1])
```

```
e^(1/2*t^2)
```

```
desolve(diff(y,t) - y + 2*sin(t), y, [0, 1])
```

```
sin(t) + cos(t)
```

```
desolve(diff(y,t) - y - y/t, y, [1, e])
```

```
t*e^t
```

Numeerisia ratkaisuja voidaan tutkia ohjelman 19 (luvussa 7.4.4) avulla. Koska tiedetään, että symboliset ratkaisut voidaan määrittää, valitaan valikosta "Tarkkojen arvojen laskeminen" vaihtoehto "Maxima". Nyt sovellus piirtää tarkan ratkaisun numeeristen ratkaisujen rinnalla, mikä mahdollistaa numeeristen menetelmien tuottamien ratkaisujen virheen arvioinnin.

b) Määritetään vastaus ohjelman 19 (luvussa 7.4.4) avulla. Koska askelvälin pituus on 0,1, riittää $y(0,4)$ määrittämiseksi laskea vain neljä askelta, $n = 4$. Maxima ei pysty määrittämään tämän differentiaaliyhtälön tarkkaa ratkaisua (virheilmoitus *NotImplementedError: Maxima was unable to solve this ODE.*), joten valitaan valikosta "Tarkkojen arvojen laskeminen" vaihtoehto "Ei ratkaista". Nyt sovellus tulostaa taulukon, jossa on listattuna lasketut arvot. Tehtävän vastaus on luettavissa 4. askeleen kohdalta ($t = 0,4$).

Vastaus: $y(0{,}4) = -1{,}31$