Shenran Wang, 711467

Tietotekniikka, Syksy 2018

23/04/2019

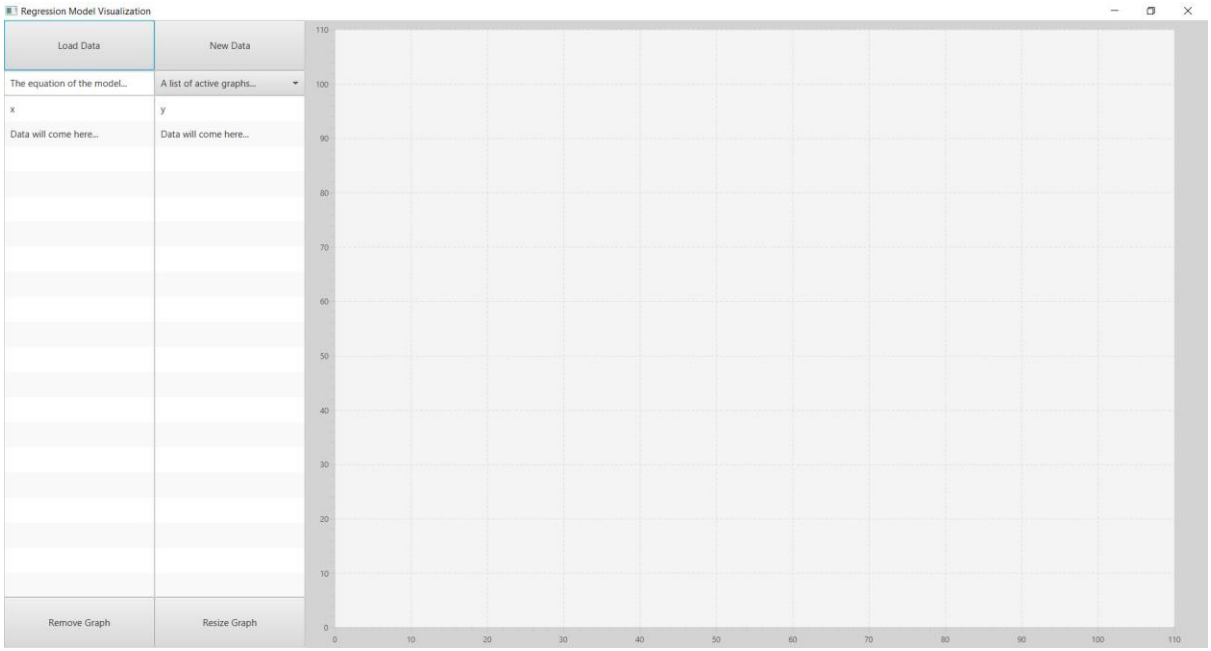
Regressiomallin sovittaminen ja visualisointi: Dokumentti

Yleiskuvaus

Vaikealla tasolla toteuttu ohjelma pystyy sovittamaan regressionmallin käyttäjän valitsemaan datajoukkoon ja hyödyntämään saatuja tuloksia sekä analyyttisesti että visuaalisesti. Kahdesta muuttujasta (x, y) koostuvan datan muuttujien välistä suhdetta arvioidaan kahdella eri menetelmällä:

1) $y = a * x + b$ (Yksinkertainen lineaariregressio)
2) $y = a_0 * x^n + a_1 * x^{n-1} + \cdots + a_{n-1} * x + a_n$ (Polynominen regressio, jossa n on luonnollinen luku, ei nollaa)

Käyttäjä pystyy lataamaan datajoukkonsa valitsemastaan tiedostosta. Ohjelmassa tuettuja tiedostoformaatteja ovat csv ja xml. Kun data on syötetty ohjelmaan, käyttöliittymään piirtyy kuvaaja sekä alkuperäisistä datapisteistä että niihin sovitetusta regressiomallista. Kuvaajan yhtälö ja ladatun datan (x,y)-pisteet näkyvät käyttöliittymässä. Käyttäjä voi säätää kuvaajan asetuksia muuttamalla akselien päätepisteitä. Jos käyttäjä on saman käyttösession aikana ladannut useamman tiedoston, jolloin on useampaan datajoukkoon sovitettu regressiomalli, niin käyttäjä voi vaihtaa aktiivisen kuvaajan nopeasti kuvaajavalikosta.



Kuva 1: Käyttöliittymä käynnistettäessä.

Käyttöohje

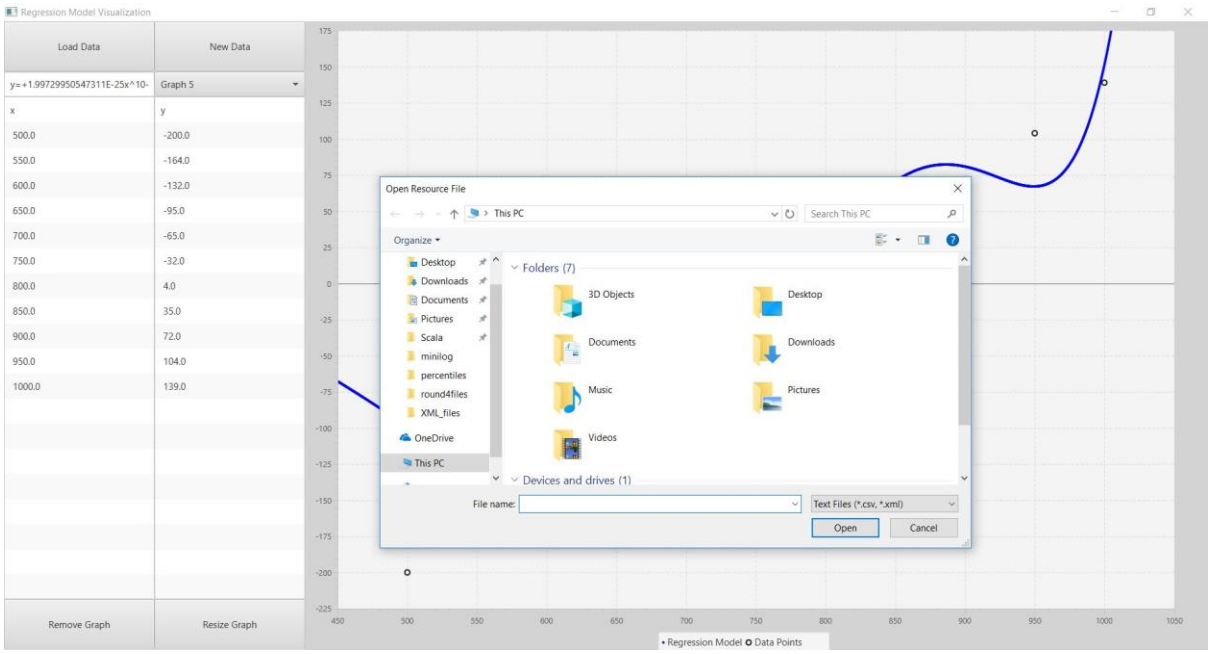
Ohjelma käynnistetään Eclipsestä valitsemalla projektin tiedosto ”RegressionModelGUI” ja käynnistämällä se joko ”Run” -valikon kautta tai pikanäppäimellä (Windows: Alt+Shift+X, S). Käyttäjä pystyy mm. sovittamaan regressiomallin valitsemaansa datajoukkoon, muuttamaan kuvaajan akseleita sekä kopioimaan regressiomallin yhtälön muuta käyttöä varten.

Sovelluksen perimmäisenä tarkoituksena on regressiomallin sovittaminen käyttäjän haluamaan datajoukkoon. ”Load Data” -painikkeen kautta käyttäjä voi valita tiedoston, joka sisältää datajoukon. Ohjelma tukee csv- ja xml-tiedostoformaatteja. Vaihtoehtoisesti käyttäjä voi ”New Data” -painikkeen kautta syöttää suoraan dataa, johon sovitetaan regressiomalli.

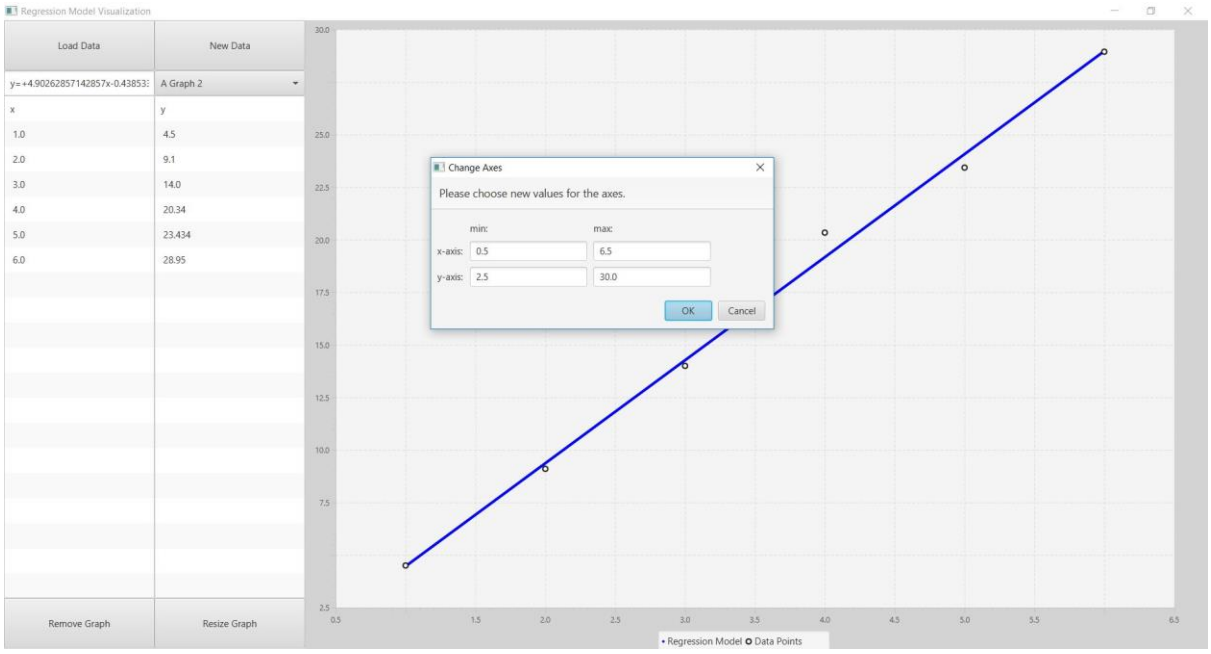
Tämän jälkeen käyttäjä nimeää kuvaajan ja valitsee regressiomallin: yksinkertainen lineaariregressio tai polynominen regressio, jonka yhteydessä valitaan myös asteluku. Kun ohjelma on suorittanut tarvittavat laskut, käyttöliittymän kuvaajaan ilmestyy regressiomalli datapisteineen. Sen lisäksi, vasemalle reunalle ilmestyy datapisteiden arvot, regressiomallin yhtälö, ja uusi malli lisätään kuvaajavalikkoon.

Jos useampaan datajoukkoon on sovitettu regressiomalli session aikana, niin käyttäjä voi kuvaajavalikon kautta vaihdella kuvaajien välillä. Valikossa lukee kuvaajien nimet. Käyttäjä voi kopioida regressiomallin yhtälön ja käyttää sitä sovelluksen ulkopuoliseen tarkoitukseen.

Kuvaajan akselien rajoja on mahdollista säätää ”Resize Graph” -painikkeella. Tällöin käyttäjä pystyy muuttamaan kuvaajaa haluamansa näköiseksi. Kuvaajan myös halutessaan poistaa ”Remove Graph” -painikkeella, jolloin kuvaaja poistetaan kuvaajavalikosta.



Kuva 2: ”Load Data” -painikkeen ensimmäinen näkymä, jossa valitaan tiedosto (csv tai xml).



Kuva 3: ”Resize Graph” -painikkeen näkymä, jossa annetaan uudet ääriarvot akseleille.

Ohjelman rakenne

Ohjelma jakaantuu kolmeen ydinluokkaan: RegrModel, Graph ja Data. Käyttöliittymän toiminta taas jakantuu luokkiin GUImethods ja RegressionModelGUI (kts. Liitteet-osio, jossa on UML-kaavio).

1) Abstraktissa luokassa RegrModel tapahtuu regressiomallin parametrien laskeminen. RegrModel ottaa parametrikseen Data-olion, jonka tiedoilla mallin tuottaminen on mahdollista. RegrModelin tärkein metodi on plotModel, joka suorittaa laskut. Metodin algoritmi käsitellään myöhemmin (kts. Algoritmit-osio). Se käyttää apunaan metodeja xMatrix ja yVector, jotka muodostavat datajoukon avulla aineistomatriisin ja vastevektorin. xMatrix riippuu myös regressiomallin tyypistä. Sen lisäksi on vielä parameters -metodi, joka tekee plotModel -metodista saatujen parametrien käsittelystä sujuvampaa muissa luokissa, kuten Graph -luokassa. RegrModel -luokan metodeissa käytetään matriisi- ja lineaarialgebrakirjastoa EJML[1].

    a. Luokka SimpleModel mallintaa yksinkertaista lineaariregressiota. Tällöin aineistomatriisilla on dimensiot nx2 ja vastevektorilla nx1, jossa n on datajoukon pisteiden lukumäärä. Vastaavasti plotModel tuottaa 2x1-parametrivektorin.

    b. Luokka PolynomialModel mallintaa polynomista regressiomallia. Tällöin aineistomatriisilla on dimensiot nxm ja vastevektorilla nx1, jossa n on datajoukon pisteiden lukumäärä, ja m on polynomin asteluku. Vastaavasti plotModel tuottaa mx1-parametrivektorin.

2) Graph -luokassa tapahtuu regressiomallin ”visualisointi”. Se ottaa parametrikseen RegrModel -olion. Graph-luokan metodi dataCoordinates laskee regressiomallin parametrien avulla regressiokäyrän pisteet tietyn dx:n välein, jossa x:n aloitus- ja päätepiste on määritetty. Tämän avulla drawGraph -metodi siirtää lasketut käyrän pisteet käyttöliittymälle sopivaksi kokoelmatyypiksi. Metodi equation palauttaa regressiomallin yhtälön käyttöliittymää varten.

[1] http://ejml.org/javadoc/

3) Data -luokka sisältää tekstitiedostosta saadun tai käyttäjän syöttämän datajoukon jäsenneltynä sopivasti RegrModel -olioille käsiteltäväksi. Parametrikseen se ottaa kaksi reaalilukuja sisätävää taulukkoa, jotka vastaavat datajoukon x- ja y-arvoja. Data -luokan perimmäisenä tarkoituksena on toimia kuin eräänlainen tietorakenne, joka helpottaa datajoukon pisteiden käsittelyä huomattavasti.

4) RegressionModelApp -olio toimii käyttöliittymän ja muun ohjelman linkkinä. Oliolla kokoelma models, joka sisältää kaikki session aikana luodut kuvaajat, positettuja lukuunottamatta. Sen lisäksi sillä on Parser -olion instanssi. RegressionModelApp sisältää paljon tärkeitä metodeja, kuten createGraph, createSimpleModel ja createData, joita hyödynnetään käyttöliittymän puolella. Metodi throwTaskEnded taas heittää poikkeuksia (tapausluokka TaskEnded). Parser -olio toimii tiedostonlukijana (metodi loadData, csv- ja xml-tiedostoformaatit) sekä merkkijonomanipulaattorina ohjelmassa (metodit checkPosInt ja checkDouble).

5) GUImethods -olio sisältää pääosan käyttöliittymän toiminnallisuuksista. Sillä on RegressionModelApp -olion instanssi. Oliot ovat erillään selkeyden vuoksi sekä erittäin suurien ja monipuolisten luokkien välttämiseksi. Suurin osa olion metodeista kutsutaan itse käyttöliittymän painikkeilla. Esimerkiksi painike ”LoadData” kutsuu metodia loadDataProcess, joka taas kutsuu metodeja fileChooser, chooseModel ja chooseGraphName. Metodia alert käytetään moneen otteeseen varoittaessa käyttäjää mahdollisista virheistä.

6) Ohjelman graafinen käyttöliittymä RegressionModelGUI sisältää alkutilassaan vain painikkeita ja tyhjiä elementtejä, kuten kuvaaja, kuvaajavalikko ja datajoukkolista. Ohjelmassa ei siis ole Data-, Graph-, tai RegrModel-olioita alussa. RegressioModelGUI -olio sisältää GUImethods -olion instanssin, joka taas sisältää RegressionModelApp -olion instanssin. Kaikki toiminnallisuudet tapahtuvat painikkeiden kautta tässä käyttöliittymässä. Kun käyttäjä painaa joko ”Load Data”- tai ”New Data” -painiketta, niin ohjelma luo kaikista yllä mainetuista luokista yhden olion kunhan poikkeuksia tai virheitä ei tapahdu, kuten datapisteiden puute (vähintään kaksi) tai väärä tiedostoformaatti. Käyttöliittymä päivittää itseään automaattisesti onnistuneessa prosessissa, jolloin mallin yhtälö, datajoukko, ja kuvaaja ilmestyvät käyttöliittymään.

Painike ”Resize Graph” muuttaa valitun kuvaajan akseleita ja ”Remove Graph” poistaa valitun kuvaajan kuvaajavalikosta. Käyttöliittymässä sovelletaan ScalaFX-kirjastoa[2].

Tähän ratkaisuun olen päätynyt hyvistä syistä. Olen erottanut käyttöliittymän muusta ohjelman luokista siten, että se ei pääsisi käsiksi niihin suoraan, vaan ainoastaan RegressionModelApp-olion kautta. Tämä selkeyttää ohjelman jakoa. Yleisesti pienemmät luokat selkeyttävät luokkien tehtävää, eikä tee yhdestä luokasta liian monipuolista.

Algoritmit

Tässä osiossa käsitellään kolme keskeisintä algoritmia: regressiomallin määrittäminen, tiedostojen lukeminen sekä kuvaajan piirtäminen.

Regressiomalli

Ohjelma käyttää datalle sopivan regressiomallin luomiseen matriiseja (johdettu PNS:stä) sillä ehdolla, että Gauss-Markov -oletukset ovat voimassa[3]. Matriisimuotoa käytetään siksi, että ohjelma olisi tällöin helpommin laajennettavissa, jos haluttaisiin esim. sovittaa regressiomalli useamman muuttujan datajoukkoon. Polynomisen regressiomallin löytäminen perustuu algoritmiin, jossa ratkaistaan yhtälön $y_i = \beta_0 + \beta_1 * x_i + \beta_2 * x_i^2 + \cdots + \beta_m * x_i^m + \varepsilon_i$ (i = 1,2,…,n) kertoimet $\beta_0 \ldots \beta_m$ matriiseilla[4]. Tässä n on datajoukon suuruus ja m on polynomin asteluku. Malli voidaan kirjoittaa matriisimuodossa:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^m \\ 1 & x_2 & x_2^2 & \cdots & x_2^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^m \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_m \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

Yksinkertaisemmin: $\vec{Y} = X\vec{\delta} + \vec{\varepsilon}$, jossa X on aineistomatriisi, Y on vastevektori, δ on parametrivektori ja ε on residuaalitermi. Kun residuaalitermi merkitään nollaksi ja muokataan yhtälöä matriisilaskennan säännöillä, saadaan:

$$\hat{\delta} = (X^T X)^{-1} X^T Y$$

[2] www.scalafx.org/api/8.0/index.html#package
[3] http://www.stat.cmu.edu/~cshalizi/mreg/15/lectures/13/lecture-13.pdf
[4] https://fi.wikipedia.org/wiki/Lineaarinen_regressioanalyysi

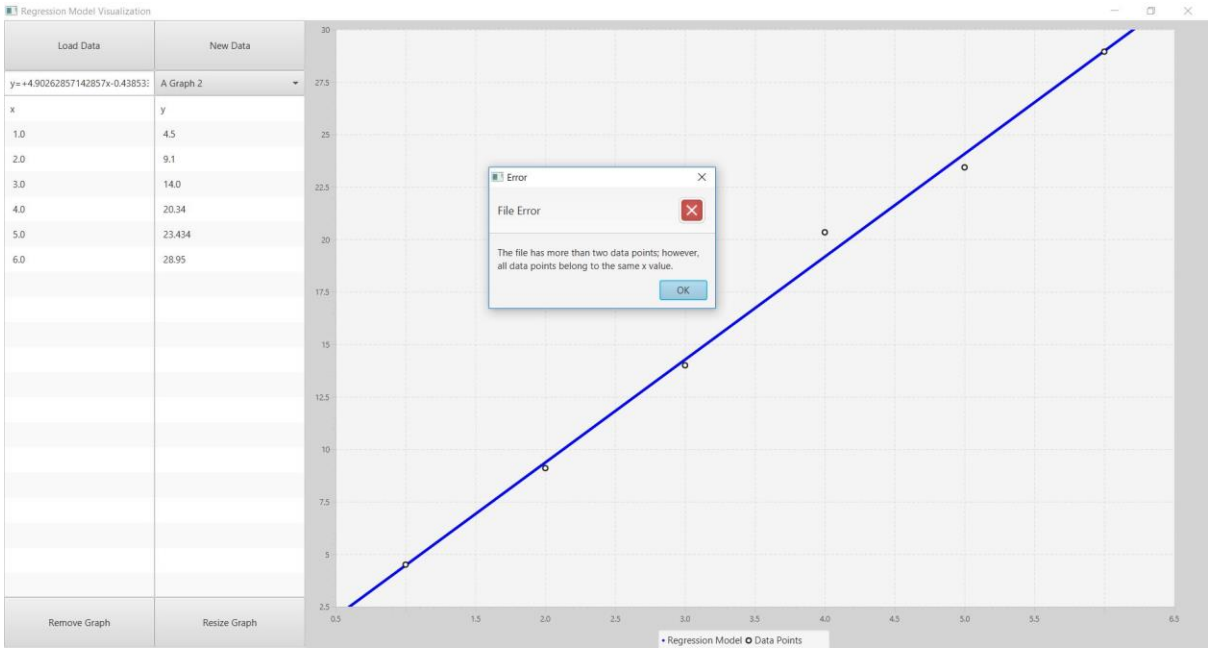
Tästä saadaan kertoimet $\beta_0 \ldots \beta_m$, joiden avulla sovitetaan regressiomalli[5]. Yksinkertaisen lineaariregression tapauksessa m=1.

Saadulla Data-oliolla voidaan yksinkertaisesti sijoittaa kaikki datajoukon x:n arvot aineistomatriisiin ja y:n arvot vastevektoriin. Parametrivektori saadaan yllä esitetyllä kaavalla: kertolasku aineistomatriisin pseudoinverssin ja vastevektorin välillä.

PNS-menetelmä perinteisessä muodossaan olisi myös ollut hyvä menetelmä, mutta ohjelman laajentamisen kannalta huono, sillä useamman muuttujan datajoukon tapauksessa regressiomallin määrittäminen on hyvin vaikeaa/mahdotonta.

Tiedostojen lukeminen

Tiedostojen lataaminen niin csv:llä kuin xml:llä toimii samalla metodilla: Parser-olion loadData. Aluksi luetaan kaikki tiedoston rivit ja kitketään kaikki tyhjät rivit pois. Sen jälkeen tarkistetaan, onko kyseessä xml-tiedosto. Jos on, sen formaatti muutetaan csv-tiedostojen formaatin näköiseksi, eli tiedoston rivien merkkijonot muutetaan. Tämän jälkeen varmistetaan, ettei tiedostoissa ole virheitä, esim. data ei koostu täysin reealiluvuista tai x-arvolta puuttuu vastaava y-arvo.



Kuva 4: Tiedoston lataaminen epäonnistui: kaikki datajoukon pisteet sijaitsevat samalla x:n arvolla, jolloin ei ole mahdollista sovittaa datajoukkoon regressiomallia.

[5] https://towardsdatascience.com/introduction-to-linear-regression-and-polynomial-regression-f8adc96f31cb

Tämän jälkeen varmistetaan, että tiedostojen ensimmäisessä rivissä lukee ”x;y” tai ”y;x”; muuten ohjelma heittää poikkeuksen. Onnistuneessa tapauksessa Parser-olio lukee tiedoston läpi riveittäin, tallentaen samalla datapisteet kokoelmaan. Lopuksi etsitään taas mahdollisia virheitä, esim. datapisteitä on vain yksi tai datapisteet sijaitsevat samalla x:n arvolla. Jos ei löydy ongelmia, niin loadData palauttaa Data-olion, joka sisältää tiedoston kaikki datapisteet.

Ongelman olisi voinut myös ratkaista csv- ja xml-kirjastoilla. Scala-kielelle näitä ei kuitenkaan ole montaa, ja niiden dokumentaatio ei ole parasta luokkaa, joten oli helpompaa ja vaivattomampaa ohjelmoida Parser-olio omin käsin.

Regressiomallin käyrä

Käyrän ”piirtäminen” eli käyrän pisteiden määrittäminen dx:n välein tapahtuu Graph -luokan metodeissa dataCoordinates ja drawGraph. Suurin osa työstä tehdään ensin mainitussa. Graph -luokan regressiomallin parametrivektorin avulla saadaan laskettua millä tahansa x:n arvolla vastaava y:n arvo. Tämä arvo määritetään seuraavasti (m on asteluku):

$$y_i = \beta_0 + \beta_1 * x_i + \beta_2 * x_i^2 + \cdots + \beta_m * x_i^m$$

Käyrää varten lasketaan y:n arvoja dx:n välein. Se määritetään ottamalla pienin mahdollinen arvo seuraavista:

1) Datajoukon suurin x-arvo vähennettynä pienimmällä x-arvolla.
2) Datajoukon suurin y-arvo vähennettynä pienimmällä y-arvolla.
3) Kuvaajan x-akselin maksimi vähennettynä minimillä. (Painettaessa ”Resize Graph”)
4) Kuvaajan y-akselin maksimi vähennettynä minimillä. (Painettaessa ”Resize Graph”)

Tämä on kokeilun kautta tuottanut parhaimman tuloksen (kts. Heikkoudet, missä esitetään yksi ongelma tällä tavalla; muilla ratkaisuilla on ollut vielä vakavampia ongelmia). Y-arvojen laskemista varten määritetään aloitus- ja lopetuspiste. Kun laskut on suoritettu, kaikki datapisteet on säilytetty kokelmassa. Tämän jälkeen drawGraph -metodissa muutetaan kokoelmien, joissa säilytetään dataCoordinatesin tuottaman käyrän pisteitä ja varsinaisen datajoukon pisteitä, tietorakenne XYChart.Data:ksi, jota graafinen käyttöliittymä pystyy käyttämään.

Eräänä vaihtoehtona olisi ollut laskea jokaisen x:n kohdalla derivaatta. Tällöin olisi tangenttifunktiolla voitu piirtää hyvin lyhyitä suoria, jotka muodostavat käyrän. ScalaFX:llä ei kuitenkaan ole mahdollista määrittää regressiomallin derivaattaa, eli olisi pitänyt käyttää toista kirjastoa tai tehdä uusi algoritmi derivaatan määrittämiseen käsin, mikä vaikuttaa hyvin työläältä verrattuna nykyiseen ratkaisuun.

Tietorakenteet

Ohjelmassa käytetään muuttuvatilaisia listapuskureita ja muuttumattomia taulukkoja, tarkemmin ottaen ListBuffer ja Array, tietorakenteena. Data -luokka käyttää esimerkiksi taulukoita parametreina, ja yleisesti tilanteissa, jossa kokoelmien kokoa ei tarvitse muuttaa, käytetään taulukoita tehokkuuden vuoksi.

Listapuskureita taas käytetään mm. Graph- ja RegressionModelApp -luokissa. Syynä siihen on ListBufferin muokattavuus sekä sisään rakennettu järjestys, joilla on helpompaa käsitellä tilanteita, joissa kokoelman koko on tuntematon ja voi muuttua milloin tahansa[6] [7] [8].

Ohjelman eräänlainen oma tietorakenne on muuttumaton Data -luokka, joka toimii hyvin pitkälti samalla tavalla kuin taulukot; siinä on vain yhdistetty kaksi taulukkoa, jotka sisältävät vastaavat x- ja y-arvot.

Näiden lisäksi ohjelmassa käytetään ScalaFX:n XYChart.Data tietorakennetta, jota vaaditaan ScalaFX:n kuvaajia piirtävää ScatterChart-luokkaa varten[9].

Tiedostot ja verkossa oleva tieto

Ohjelma käsittelee tekstitiedostoja, tarkalleen ottaen csv ja xml, koska kyseiset tiedostomuodot ovat sopivia datan käsittelyyn. Tiedostot sisältävät datajoukon, johon käyttäjä haluaa sovittaa regressionmallin. Tiedostojen sisällöllä on eroavaisuuksia; csv-tiedoston ensimmäinen rivi täytyy olla joko ”x;y” tai ”y;x”. Datapisteiden erottimena

[6] https://stackoverflow.com/questions/5446744/difference-between-mutablelist-and-listbuffer
[7] https://stackoverflow.com/questions/434761/array-versus-listt-when-to-use-which
[8] https://docs.scala-lang.org/overviews/collections/performance-characteristics.html
[9] http://www.scalafx.org/api/8.0/index.html#scalafx.scene.chart.ScatterChart

käytetään aina puolipistettä ”;”, sillä desimaalierottimena voidaan käyttää sekä pilkkua että pistettä. Sen sijaan xml-tiedostoissa ensimmäisessä rivissä on pakollinen prologi, kuten seuraava: <?xml version=”1.0” encoding=”UTF-8”?>. Kaikki datapisteet ovat juurielementin <DataInfo> lapsia. Täten ohjelma tunnistaa tiedoston xml:ksi. Datajoukkoa vastaavat tiedoston sisällöt on esitetty alla (x1, x2, y1, y2 ovat reaalilukuja):

Datajoukko	
x1	y1
x2	y2
...	...

CSV
x;y x1;y1 x2;y2 ...

XML

```
<?xml version='1.0' encoding='UTF-8'?>
<DataInfo>
   <Data>
      <x>x1</x>
      <y>y1</y>
   </Data>
   <Data>
      <x>x2</x>
      <y>y2</y>
   </Data>
...
</DataInfo>
```

Tämän lisäksi ohjelman kuvaajan datapisteille ja käyrälle on ominaiset värinsä ja kokonsa, jotka on css-tiedostossa valmiina projektin kansiossa. Tiedosto on myös dokumentin liitteenä lähdekoodin kanssa. Ohjelma ei ole millään tavalla yhteydessä verkkoon.

Testaus

Projektin aikana on pääosin testattu ohjelman toimivuutta suoraan käyttöliittymän kautta. Käyttöliittymän painikkeiden toimivuutta kokeiltiin aluksi mock- ja stub-metodien avulla, ja sen jälkeen implementointiin painikkeiden toiminnot. Painikkeiden ja ohjelman metodien toimivuutta testattiin siis rinnakkain.

Esimerkiksi tiedoston latautumista on testattu ”Load Data” -painikkeen kautta lukuisaan otteeseen. On varmistettu oikeassa formaatissa olevien tiedostojen toimivuuden siten, että käyttöliittymän vasemmalla puolelle ilmestyvät datajoukon x- ja y-arvot täsmäävät tiedostojen kanssa, esim. desimaalierottimina pilkut muutetaan pisteiksi ja tiedoston tyhjät rivit ei oteta huomioon.

On testattu myös vääränlaisten tiedostojen kohdalla, että ohjelma heittää poikkeuksen. Esimerkiksi datajoukossa on liian vähän datapisteitä, kaikki datapisteet sijaitsevat samalla x:n arvolla tai datajoukon arvot eivät kaikki ole reaalilukuja.

Luokkien Graph ja Data vastaavuutta käyttöliittymän kanssa on testattu tutkimalla mm. datapisteiden oikeinsijoittumista kuvaajassa ja kuvaajan akseleiden muuttamista.

Regressiomallien toimivuutta kokeiltiin vertaamalla käyttöliittymään ilmestyvän yhtälön kertoimia matemaattisten laskinten tuottamiin kertoimiin samalla datajoukolla. Esim. linearModel -metodia testataan antamalla sille Data-luokan olio, joka sisältää testattavan datajoukon. Olkoon datajoukko seuraavanlainen:

x	2.5	4.5	7	8.5	11	13	15.5	17
y	7	12.5	19	24	32.5	38.5	46.5	53

Tällöin metodi tuottaa regressiomallin, jonka pitäisi vastata laskimen tuottamaa regressiomallia: $y = 3.155183 \ldots * x - 2.032441 \ldots$ (kymmenen desimaalin tarkkuudella yksinkertaisessa lineaariregressiossa, polynomisessa regressiossa vähenevällä tarkkuudella asteluvun kasvaessa)[10].

Projektin loppuvaiheilla käytettiin myös yksikkötestejä seuraavissa ydintoiminnoissa varmuuden vuoksi, vaikka niitä testattiin läpi projektin käyttöliittymän kautta:

1) Metodi loadData: verrataan sen tuottaman Data-olion x- ja y-arvoja oikeaan vastaukseen. Kaikki viattomat tiedostot menivät testissä läpi. Viallisia ei testattu, sillä metodin sisällä olevat poikkeukset estävät testin läpäisyn (käyttöliittymän kautta kaikki vialliset tiedostot jäivät kiinni johonkin poikkeukseen).

2) Metodi plotModel: luokkaa SimpleModel testattiin, ja kaikki toimii vähintään kymmenen desimaalin tarkkuudella ongelmitta. Polynomisessa regressiossa voi kuitenkin ilmestyä ongelmia suuremmilla asteluvuilla. Esim. erään datajoukon polynomisen mallin kolmannella asteluvulla virhe on edelleen 10^-11 luokkaa, kun viidennellä asteluvulla virhe oli jo 10^-6 luokkaa. Yhden erittäin hajanaisen datajoukon polynomisen mallin kuudennella asteella virhe oli 10^-2 luokkaa, ja

[10] https://arachnoid.com/polysolve/

yhdennellätoista asteluvulla virhe oli erittäin suuri, 10^4 luokkaa. Hajanaisuus tietysti vaikutti paljon virheeseen, sillä säännöllistä sinifunktion näköistä rataa liikkuvan datajoukon kahdeksannen asteluvun virhe olikin vain 10^-8 luokkaa. Tästä tarkemmin Heikkoudet-osiossa.

Testaukset tehtiin kokonaisuudessaan hyvin pitkälti suunnitelman mukaisesti. Ohjelma ei suoriutunut esimerkillisesti polynomisen regressiomallin määrittämisessä kaikissa tapauksissa, mutta muuten kaikista tärkeimmät testit niin yksikkötestauksessa kuin käyttöliittymän testauksessa sujuivat hyvin.

Puutteet ja viat

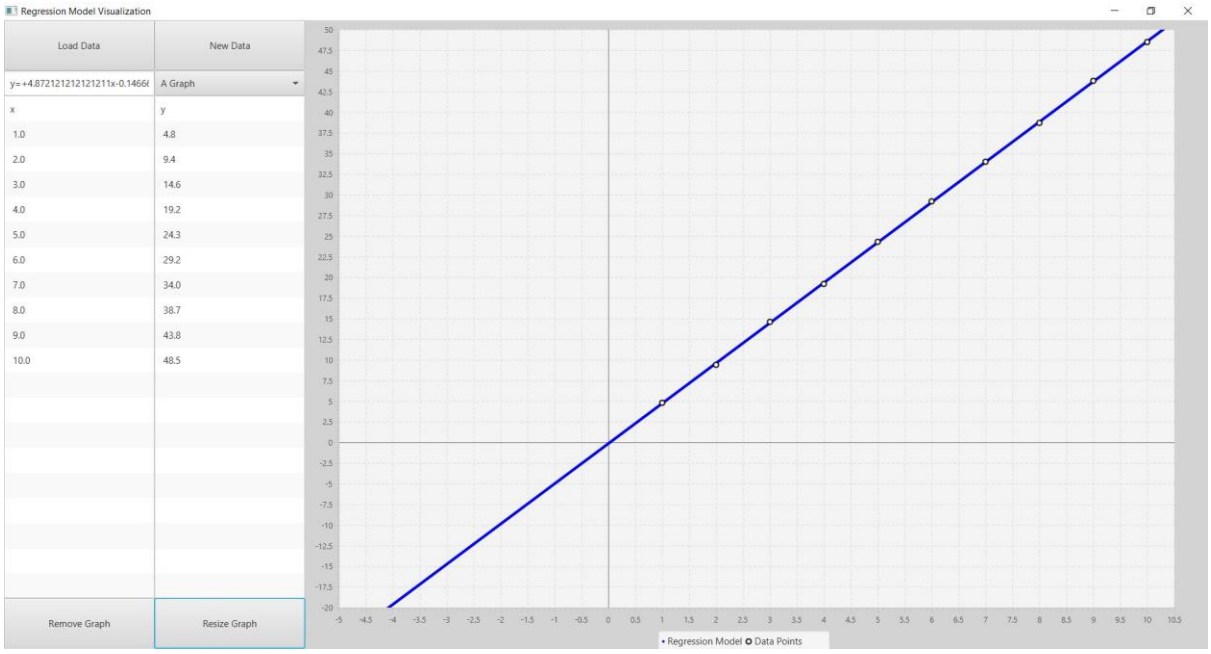
Tärkeimmät ongelmat on käsitelty Heikkoudet-osiossa. Tässä osiossa käsitellään muita puutteita ja vikoja.

Ensimmäinen regressiomallin käyrä, joka syntyy ladattaessa tai syötettäessä datapisteitä sovitettavaksi, on piirretty ainoastaan datajoukon x-arvojen minimin ja maksimin välille. Jos painaa ”Resize Graph” ja hyväksyy samat ääriarvot akseleille kuten ne alun perin olivat, niin käyrä ylettyykin kuvaajan reunoille. Ongelman voisi korjata siten, että ScatterChartin (kuvaajan) metodi autoSize on false, jolloin Graph -luokan metodeissa dataCoordinates ja drawGraph pitäisi määrittää jollain tavalla akselien ääriarvoiksi ne, jotka tulisi myös autoSizen ollessa true.

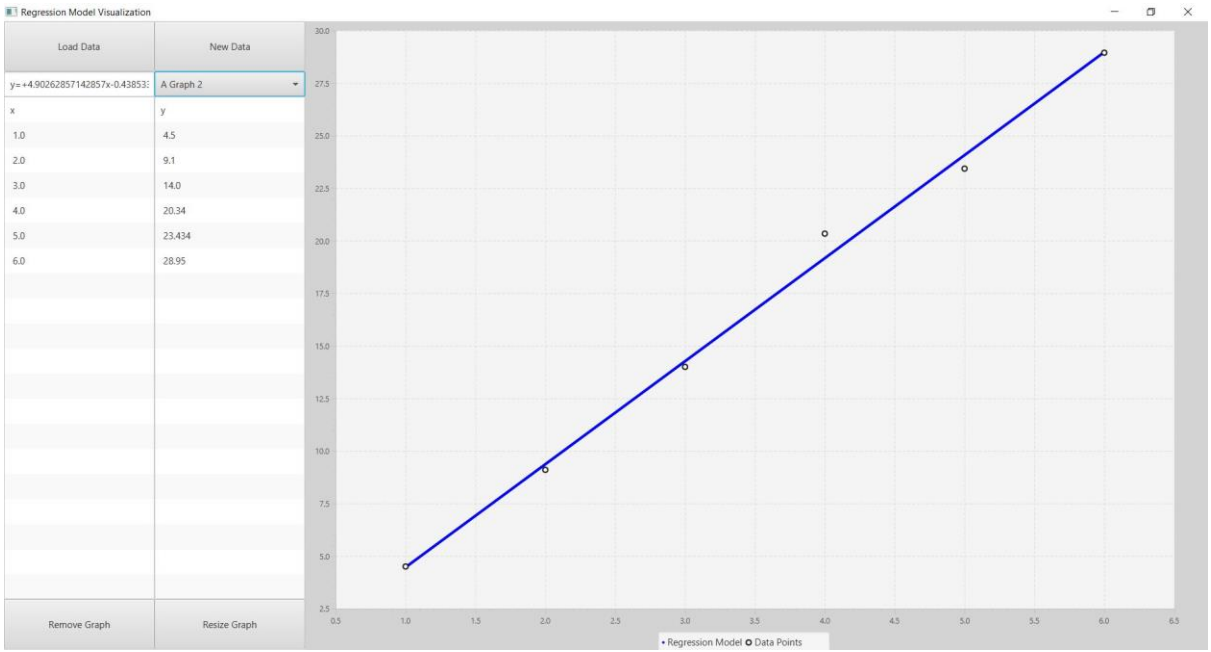
Jos valitun kuvaajan akseleita on muutettu, ja kuvaajaa vaihdetaan toiseen kuvajaan ja takaisin, niin alkuperäisen kuvaajan akselit eivät pysy muutetuissa arvoissaan. Tämän voisi korjata siten, että Graph -oliolla olisi muuttuvatilainen kokoelma, joka sisältää aina viimeisimmän muutoksen aikana olleet akselien ääriarvot. Toisaalta on varsin helppoa ”resetoida” kuvaajan tila vain vaihtamalla kuvaajia nopeasti edestakaisin.

Kun jokin kuvaaja poistetaan, käyttöliittymä ei muutu lainkaan, jos poistettu kuva oli ylimpänä kuvaajavalikossa. Muuten poistetun kuvaajan tilalle tulee automaattisesti sitä ylempänä kuvaajavalikossa oleva kuvaaja. En ole täysin varma, miten tämän voi korjata, sillä tästä ei

löytynyt tietoa ScalaFX:n dokumentaation ComboBox -luokan osiosta. Sama ongelma myös ScatterChartin kanssa, sillä välillä kaikki akselien arvot eivät näy kuvaajaa päivitettäessä[11].

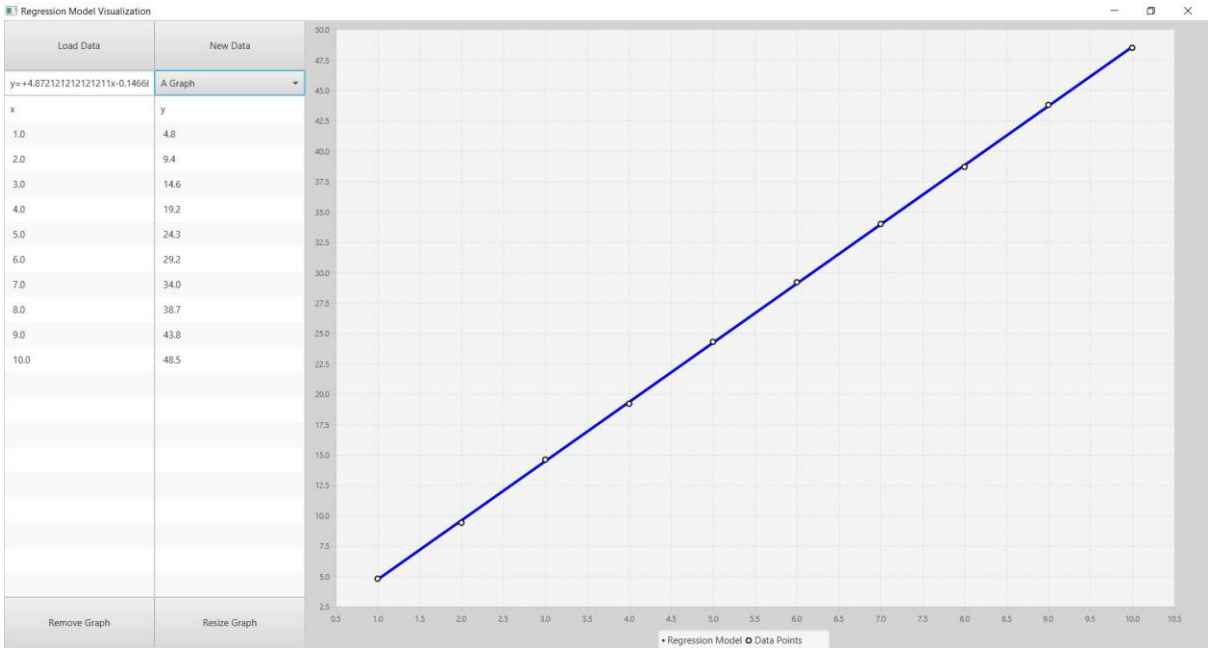


Kuva 5: Alkuperäinen kuvaaja muutetuilla akselien ääriarvoilla.



Kuva 6: Valittu toinen kuvaaja kuvaajavalikosta.

[11] www.scalafx.org/api/8.0/index.html#package



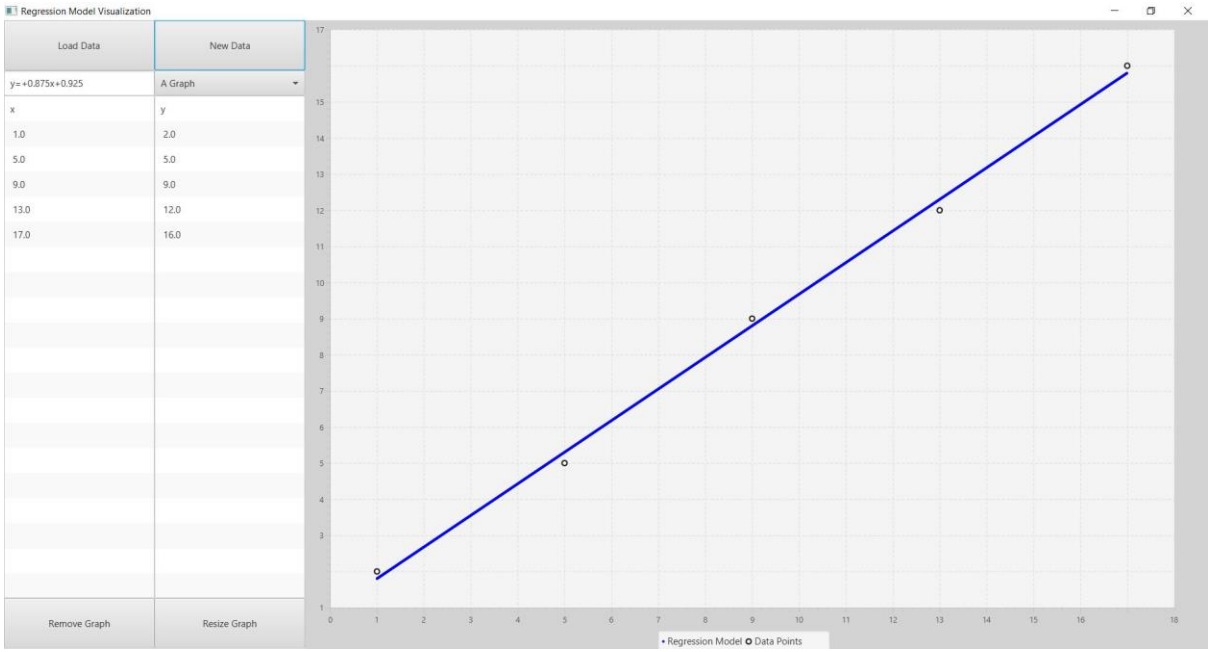
Kuva 7: Valittu taas alkuperäinen kuvaaja; akselien ääriarvot eivät säilyneet, vaan palautuivat niihin arvoihin, mitä saatiin kuvaajaa ladattaessa ensimmäistä kertaa.

Käyttöliittymässä näkyvä yhtälö ei ole täysin tarkka; se on pyöristetty kymmenen desimaalin tarkkuuteen Javan BigDecimal -luokan avulla, joten liukuluvut voivat vaikuttaa tarkkuuteen. Jos pyöristys olisi esimerkiksi neljän desimaalin tarkkuuteen, ongelmaksi voi tulla, että suuruusluokkaa 10^-6 ja pienemmät luvut pyöristyisivät nollaan tarkoituksetta, vaikka liukulukujen vaikutus pienenisi.

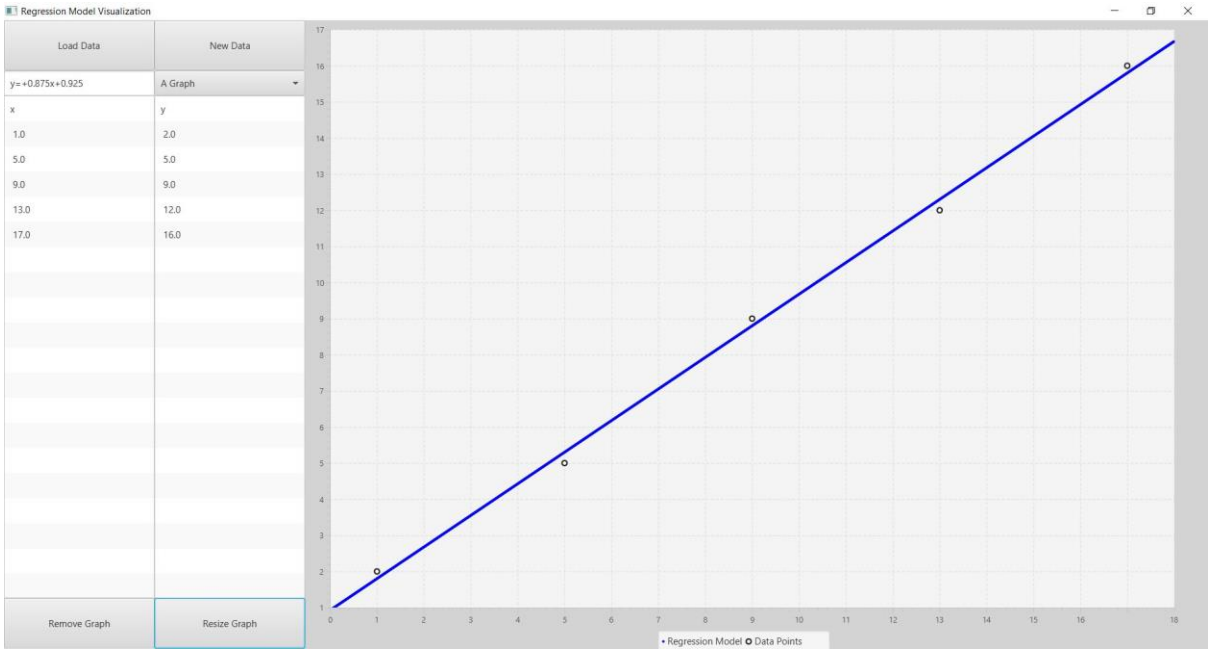
Käyttäjä ei tiedä kuinka kauan aikaa jonkin kuvaajan päivittämiseen kestää. Tämän voisi korjata soveltamalla ScalaFX:n luokkia ProgressBar ja ProgressIndicator, kunhan ne pääsevät käsiksi esimerkiksi Graph- luokan ja Parser -olion metodien etenemiseen.

Kun polynomisen regressiomallin yhtyeydessä akselien rajat asetetaan hyvin suuriksi verrattuna datajoukon x-arvojen maksimin ja minimin erotukseen verrattuna, ohjelman tehokkuus kärsii ja siltä voi loppua muisti kesken laskun ohjelman dx:n määritelmän takia. Tätä voi parantaa optimaalisemmalla algoritmilla regressiomallin käyrän pisteiden valitsemisessa.

Käyttöliittymän leveyttä ja korkeutta on mahdollista säätää, mutta elementtien koot eivät muutu. Tämä on tarkoituksellinen, sillä käyttöliittymä on tarkoitettu koko näytöllä käsiteltäväksi ohjelmaksi. Toisaalta tämä on kätevä ominaisuus, sillä käyttäjä voi tehdä ikkunasta erittäin pienen ja asettaa sen ruudun alanurkkaan, ja avata sen näppärästi maksimoimalla ikkunaa.



Kuva 8: Kuvaaja heti datan lataamisen jälkeen.



Kuva 9: Kuvaaja, kun on painettu ”Resize Graph”.

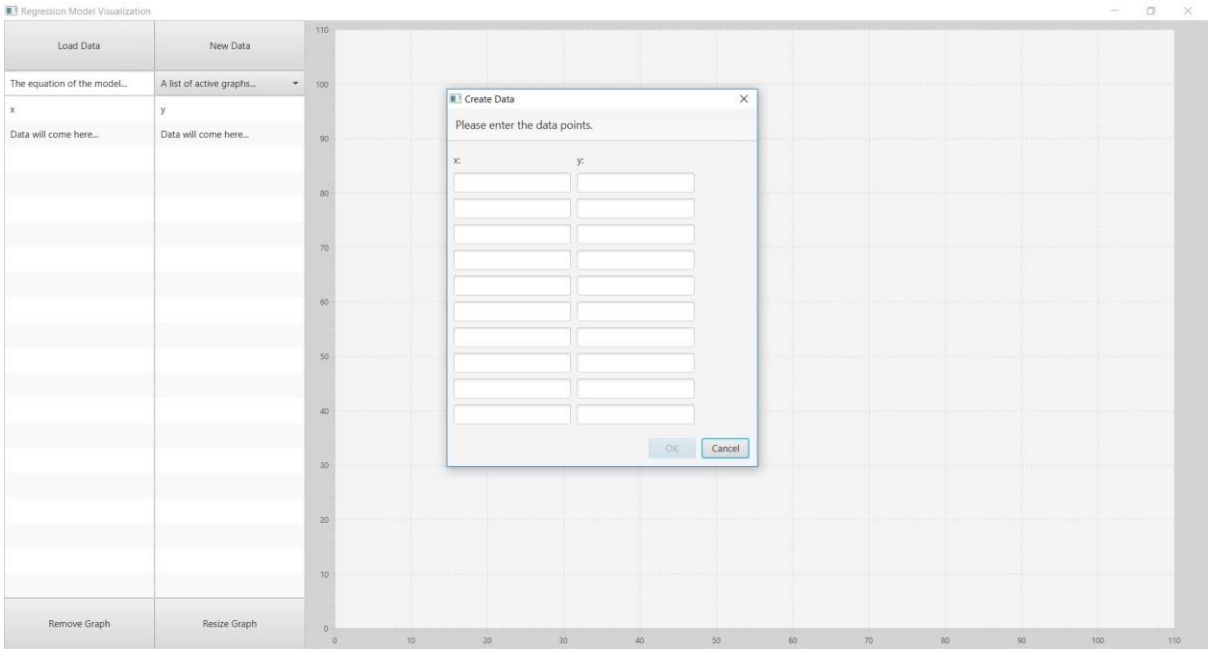
Yleisimmän csv-tiedoston lataaminen toimii (CSV (comma delimited) *.csv), mutta jotkut taas eivät toimi (CSV UTF-8 (comma delimited) *.csv). Tämä johtuu siitä, että jälkimmäisessä tiedostossa alkuun tulee kolme kirjainta, jotka edustavat ”Byte order mark”:ia. Ongelmaa ei kuitenkaan saanut kitkettyä merkkijonoa manipuloimalla, joten sen ratkaisu on vielä tuntematon.

Vahvuudet ja heikkoudet

Vahvuudet

1) Graafinen käyttöliittymä on käyttäjän kannalta erittäin selkeä ja helposti omaksuttavissa ilman suurta perehtymistä. Ohjelman kaikki toiminnot ovat heti silmien edessä, painikkeiden tehtävät ovat yksiselitteisiä ja värit ovat silmille mukavia.

2) Ohjelman ominaisuus, jolla käyttäjä voi syöttää käsin datajoukon, johon sovitetaan regressiomalli, on erittäin kätevä. ”New Data” -painike antaa mahdollisuuden syöttää dataa nopeasti, koska aina ei tarvitse luoda uutta csv- tai xml-tiedostoa, jotta saisi sovitettua regressiomallin datajoukkoon.



Kuva 10: Datan syöttäminen käsin. Tässä on valittu 10 datapistettä syötettäväksi.

3) Ohjelman luokkajako on yksiselitteinen ja selkeä. Jokaisella luokalla/oliolla on omat tehtävänsä: Data -luokka toimii tietorakenteena syötetylle datajoukolle. Tähän luokkaan pääsee helposti käsiksi RegrModel, jonka tehtävänä on määrittää datajoukkoon sovitetun regressiomallin parametrit. Graph -luokka määrittää kuvaajan pisteitä dx:n välein näiden parametrien avulla. Sen lisäksi on myös Parser -olio, joka lukee tiedostoja. Kaikki nämä osat kuuluvat RegressionModelApp -olioon.

   Käyttöliittymän ainoa tapa päästä käsiksi ohjelman eri osiin on täten RegressionModelApp:in kautta, mikä pitää käyttöliittymän ja muun ohjelman erillään. GUImethods -luokka, joka sisältää käyttöliittymän toiminnot, on vielä näiden pääsovelluksen ja käyttöliittymän välissä. Luokan kaikki toiminnot voisivat olla myös RegressionModelApp- tai RegressionModelGUI -oliossa, mutta tämä selkeyttää ohjelman rakennetta eikä tee yhdestä luokasta/oliosta liian monipuolista.

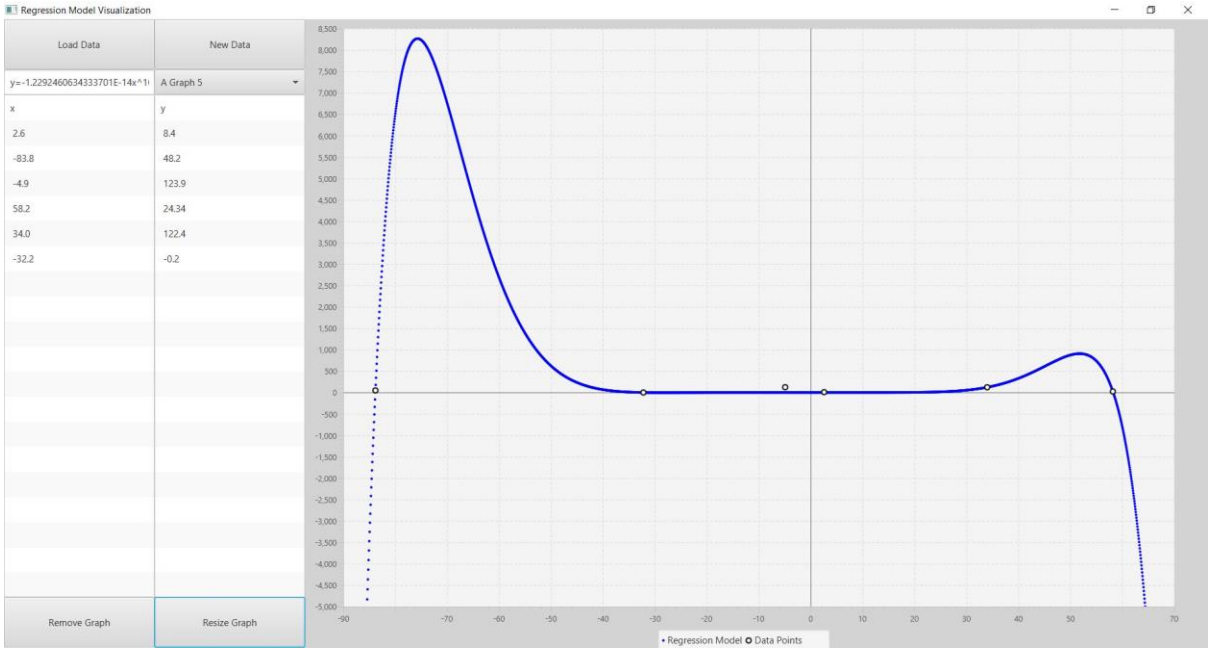
Heikkoudet

1) Ohjelma laskee kuvaajan pisteet kokonaan uudestaan aina, kun kuvaajaa muutetaan joko vaihtamalla kuvaajaa tai säätämällä akseleita. Jos x-akseli on erittäin leveä, niin tehokkuus kärsii. Pystyy korjaamaan säikeillä ja kokoelmalla, jossa säilyttäisi arvoja. Esimerkiksi alkuun lasketaan tarvittavat datapisteet, ja kuvaajan päivityttyä käyttöliittymää pystyy käyttämään, kun ohjelma laskee ohella datapisteitä akselien rajojen ulkopuolelta tiettyihin koordinaatteihin saakka omassa tahdissaan, ja ne säilytetään kokoelmassa mahdollista tulevaa käyttöä varten[12].

2) Kun kuvaajan derivaatta on erittäin suuri tai kuvaaja on tietyillä akseleiden ääriarvoilla hyvin jyrkkä, Graph-luokan drawGraph-metodin algoritmi pisteiden laskemiselle ei tuota hyvää kuvaajaa, sillä metodilla saatu dx, jonka välein piirretään regressiomallin piste, on liian suuri. Pisteet eivät tällöin ole tarpeeksi lähellä toisiaan muodostaakseen jatkuvan kuvaajan. Tämän pystyy korjaamaan, jos pääsisi käsiksi käänteisfunktioon (toisen kirjaston kautta), jolloin voidaan määrittää dy. Kun dy < dx, niin määritetään dy:n mukaan seuraava piste; kun dx < dy, niin määritetään dx:n mukaan seuraava

[12] Ohjelmointistudio 2: https://plus.cs.hut.fi/studio_2/k2019/threads/

piste. Seurauksena tehokkuus voi kuitenkin kärsiä. Ongelman voi myös kitkeä valitsemalla paremmat akselien ääriarvot.



Kuva 11: Kohdissa, joissa derivaatan arvo on erittäin suuri, ei kuvaajan jatkuvuus pysy ennallaan. Tapahtuu harvoin, ja useimmiten polynomisen regression suuremmilla asteluvuilla. Tässä asteluku on 10, mikä on suurempi kuin datajoukon pisteiden määrä. Tämän seurauksena regressiomalli tuottaa varsin erikoisen näköisen kuvaajan.

3) Korkeampien asteiden polynomisilla regressiomalleilla parametrien tarkkuus heikkenee huomattavasti johtuen liukuluvuista. Koska regressiomallien parametrien määrittämisessä suoritetaan erittäin paljon kertolaskuja, ja asteluvun kasvaessa vieläkin enemmän, niin liukulukujen aiheuttamat pienet virheet akkumuloituvat varsin suuriksi. Mitä hajanaisempia datajoukkojen arvot ovat verrattuna toisiinsa, sitä heikompi tarkkuus. Jos regressiomallin asteluku ylittää datajoukon pisteiden lukumäärän, voi malli tuottaa erikoisia vastauksia. Tällä hetkellä ongelmaan ei ole ratkaisua.

Poikkeamat suunnitelmasta

Projektin suunnitelmassa tapahtui monta muutosta. Ohjelmassa oli seuraavanlaisia poikkeamia suunnitelmasta:

1) Data -luokan parametrien tietorakenne vaihdettiin ListBufferista Arrayksi eli taulukoiksi. Tämä johtuu siitä, että datapisteiden lisääminen ja poistaminen ominaisuutena karsittiin pois, sillä viime kädessä se vaikutti varsin hyödyttömältä. Tällöin on järkevämpää käyttää taulukoita, sillä ne ovat paljon tehokkaampia, kun kokelmien suuruus ei muutu olion luomisen jälkeen. Seurauksena Data -luokka on huomattavasti yksinkertaisempi, ja datapisteiden läpikäyminen on nopeampaa. ListBufferia käytetään sen sijaan muiden luokkien metodeissa, sillä joissakin metodeissa on kätevämpää käyttää puskureita.

2) Jsonin sijaan käytetäänkin xml-tiedostoformaattia csv:n ohella. Ei ole varsinaista syytä, mutta xml-tiedostoja on helposti tehtävissä tekstieditorin avulla.

3) Luokat RegrModel ja Graph vaihtoivat perimisjärjestystä. Luonnollinen ja järkevä muutos, kun luokkia alkoi ohjelmoimaan.

4) SaveAndLoad- ja Parser-olio yhdistettiin yhdeksi olioksi. Olio pystyy myös manipuloimaan merkkijonoja. Sen lisäksi luotiin uusi tapausluokka TaskEnded.

5) Kuvaajien tallentaminen poistettiin ohjelman omaisuuksista.

6) Uusi GUImethods -olio, joka sisältää metodeja, joita graafinen käyttöliittymä käyttää. Se on erotettu käyttöliittymästä ja RegressionModelApp -oliosta selkeyden vuoksi ja liian suurien luokkien välttämiseksi.

7) Equation- tai Axes -luokat poistettiin ohjelmasta. Equationin korvasi merkkijono ja ScalaFX:llä oli valmis NumberAxis -luokka käytettävissä.

Aikataulun kannalta suunnitelmassa poikettiin huomattavasti. Projektin ulkopuolisten tekijöiden takia aloitin projektin tekemisen noin puolitoista viikkoa aikataulusta jäljessä. Suurin poikkeus oli yksikkötestien kannalta, sillä ne suoritettiin vasta projektin lopuksi (21.4 - 22.4) eikä samassa yhteydessä kuin metodeja tehtiin. Muuten regressiomallien toimivuutta testattiin csv- ja xml-tiedostoilla läpi projektin.

Aloitin projektin tekemällä käyttöliittymää ensin (alkoi 4.3) ja sitten vasta ohjelman luokkia (esim. RegrModel 17.3 ja Graph 18.3), toisin kuin suunnitelmassa. Kaikkia ohjelman osia työstettiin läpi projektin, ja kaikki osat valmistuivat suurin piirtein samanaikaisesti.

Ajankäyttöarviot olivat myös ailahtelevia, sillä käyttöliittymän tekemisessä meni huomattavasti vähemmän aikaa ScalaFX-kirjaston avulla ja ohjelman luokkien ohjelmoimisessa kesti kauemmin kuin odotettu. Refaktorointia ja pientä korjailua tapahtui läpi projektin eikä ainoastaan lopuksi.

Kokonaisarvio

Ohjelma on perusominaisuuksiltaan ja toimivuudeltaan varsin pätevä. Sillä pystyy helposti syöttämään joko tiedostolla tai käsin dataa, johon regressiomalli sovitetaan. Kuvaajan yhtälö ja datajoukon pisteet ilmestyvät itse kuvaajan lisäksi käyttöliittymään. On myös mahdollista selata kuvaajien välillä helposti kuvaajavalikon kautta.

Ohjelmalla on kuitenkin heikkouksia. Esimerkiksi polynomisen regressiomallin kuvaajan tarkkuus voi olla hyvinkin suuri suurilla asteluvuilla, tapauksesta riippuen. Sen lisäksi, aina kun kuvaajaa muutetaan, ohjelma laskee kaikki käyrän pisteet uudestaan, mikä vaikuttaa tehokkuuteen. Jos käyrä näyttää käyttöliittymässä jyrkältä akselien muuttamisen jälkeen, todennäköisesti se ei enää näytä jatkuvalta (kts. Vahvuudet ja heikkoudet -osio tarkempaa tietoa varten).

Ohjelman tämänhetkiset ratkaisumenetelmät toimivat suurimmalta osin. Esimerkiksi regressiomallin parametrien määrittäminen antaa hyvin tarkkoja vastauksia yksinkertaisella lineaariregressiolla ja polynomisen regression pienemmillä asteluvuilla. On olemassa muita ratkaisumenetelmiä myös, kuten ”Gradient descent”: optimointialgoritmi, jota pystyy soveltamaan myös regressiomallin määrittämiseen[13]. Tietorakenteita valitessa on tutkittu niiden käyttöyhteyksiä ja valittu niiden perusteella taulukot ja listapuskurit optimaalisimmiksi muuttumattomiksi ja muuttuvatilaisiksi kokoelmiksi.

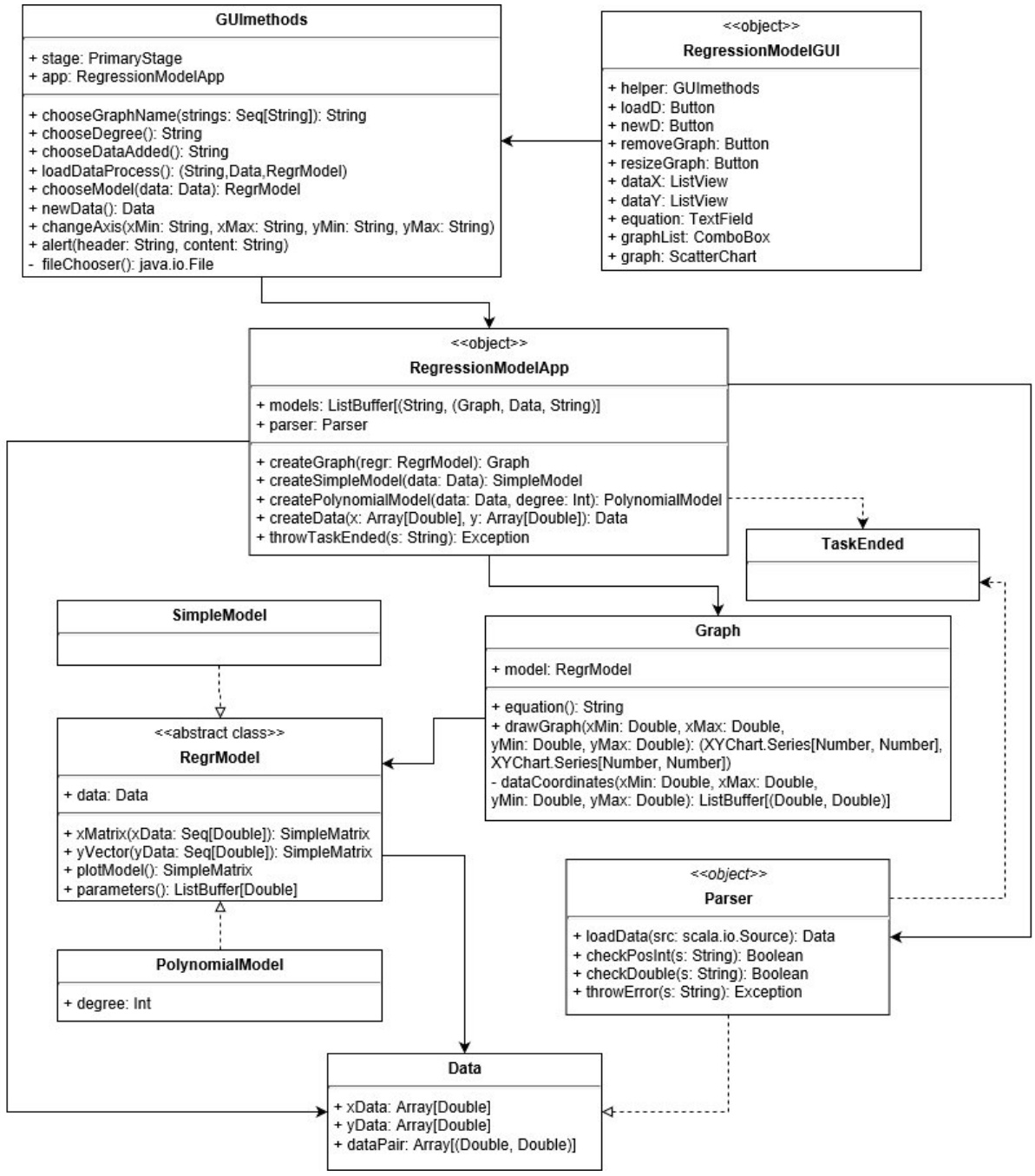
Ohjelma on rakenteeltaan selkeä ja luokilla on omat tehtävänsä, jotka muodostavat yhdessä sujuvan kokonaisuuden. Tietysti luokat voisi halutessaan jakaa vielä pienimmiksi, mutta silloin useammalla luokalla olisi samankaltaisia tehtäviä. Rakenteellisesti laajennusten tekeminen ei ole hankalaa, sillä käyttöliittymä on erotettu muusta ohjelmasta, joten ohjelmaan tehtävät

[13] https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/

lisäykset eivät aiheuta ongelmia. Uusien regressiomallien luominen hoituu tekemällä aliluokkia abstraktista luokasta RegrModel.

Jos aloittaisin projektin uudelleen alusta, tekisin ehdottomasti yksikkötestit luokkien ohella, eikä vasta projektin lopussa, kun kaikki muu on jo valmiina. Tällöin olisi todennäköisempää välttää ikäviä yllätyksiä ohjelmassa (näin ei kuitenkaan tapahtunu tällä kertaa onneksi). Aloittaisin myös projektin tekemisen aikaisemmin, niin kuin suunnitelmassa oli tarkoituksena. Viimeiseen kahteen viikkoon on tullut tehtyä paljon enemmän töitä kuin aikaisemmilla viikoilla. Viimeiseksi, perehtyisin heti alkuun paremmin projektissa käytettyihin kirjastoihin, etenkin ScalaFX:ään. Tällöin olisi voitu välttää turhaa refaktorointia projektin aikana, mikä vei kallisarvoista aikaa.

Liitteet



Liite 1: UML-kaavio.