

# Regressiomallin visualisointi

Sampo Paukkonen

656645

Tietotekniikka

2018

23.4.2019

## Yleiskuvaus

Ohjelma on regressiomallin visualisointi, eli käyttäjä voi luoda regressiomallin (suoran tai muun käyrän) käyttäjän itsensä antamaan datajoukkoon, jolloin ohjelma piirtää halutun käyrän, sekä annetut pisteet. Ohjelma luo halutun käyrän annetun datan perusteella pyrkien sovittamaan sen tähän mahdollisimman hyvin.

Ohjelmassa on mahdollista luoda yksinkertainen lineaarinen regressiomalli, joka on tavallinen  $y = kx + b$  suora, monimutkaisempi polynominen regressiomalli, joka on muotoa  $y = b + a_1 * x + a_2 * (x)^2 + \dots + a_n * (x)^n$ , missä  $1 \leq n \leq 89$  tai koneoppimisessa esiintyvä logistinen regressiomalli, joka pyrkii selittämään dikotomista dataa (tosi/epätosi) numeerisella vastemuuttujalla.

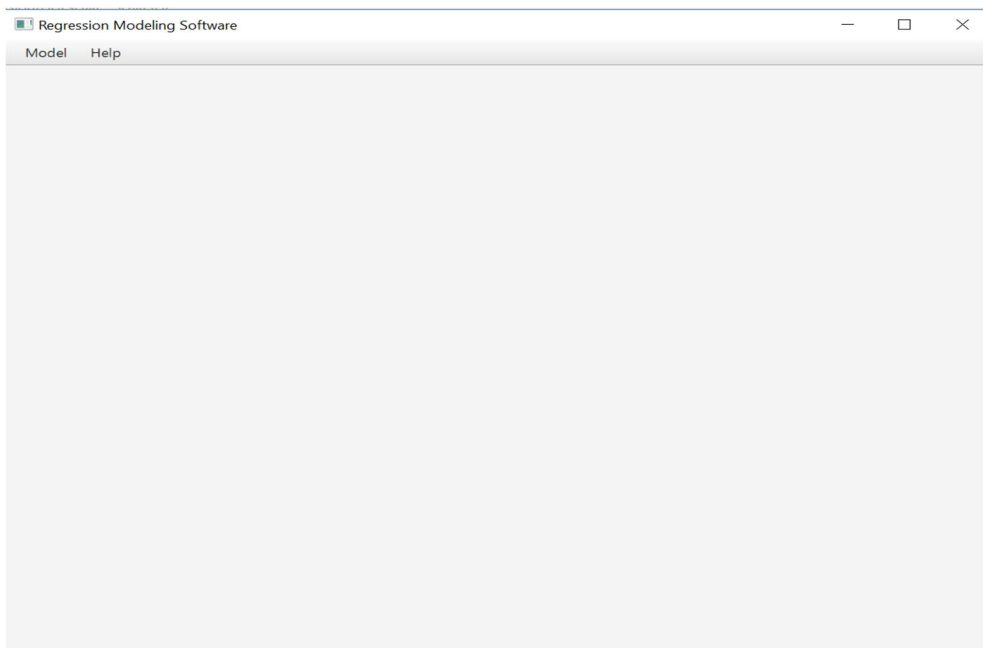
Työ on mahdollista toteuttaa keskivaikeana tai vaikeana. Työ on mielestäni toteutettu vaikean arvostelun mukaan.

## Käyttöohje

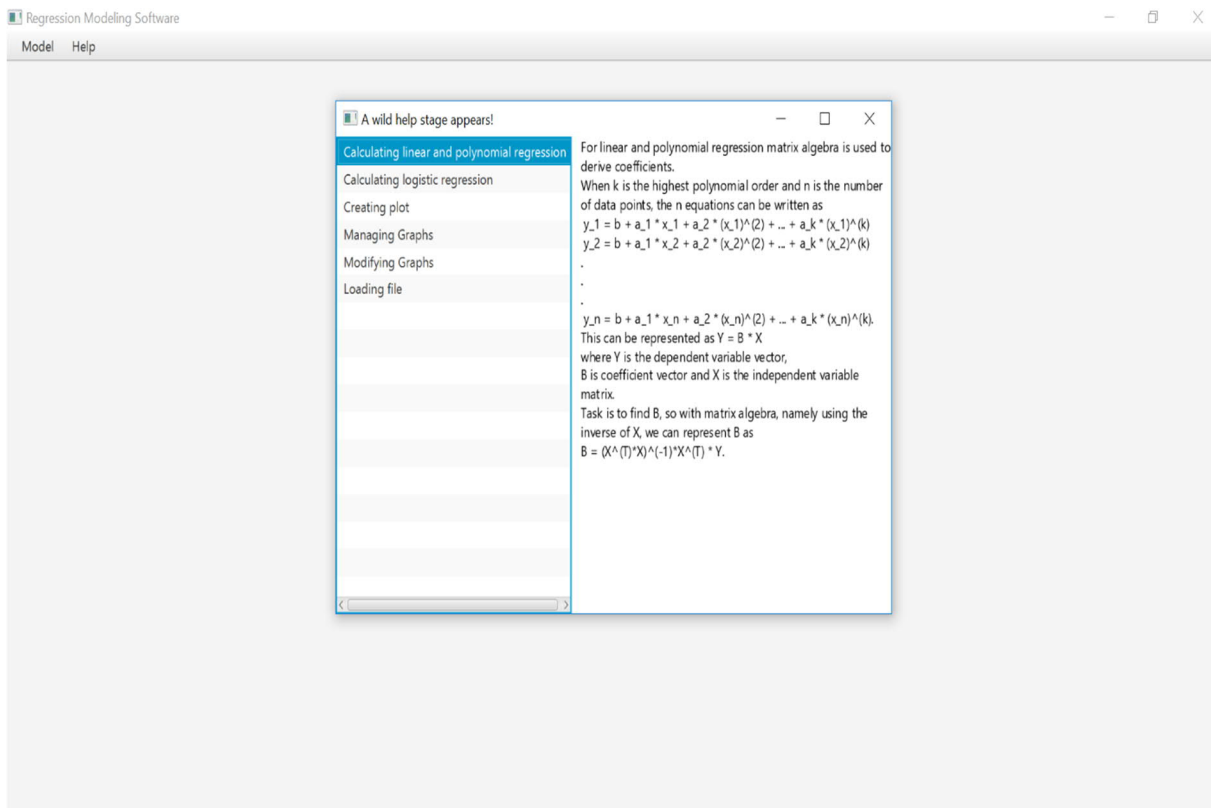
Kun projektin zip-tiedosto on tuotu esimerkiksi Eclipsen työkansioon, on tehtävä muutoksia projektin java build pathiin. Valitse projekti, paina hiiren oikeaa näppäintä avataksesi valikon ja valitse kohta Properties. Mene tämän jälkeen Libraries-näkymään,

josta valitse JRE System Library, paina Edit-painiketta ja valitse System libraryksi Workspace default JRE. Nyt kaiken pitäisi toimia normaalisti.

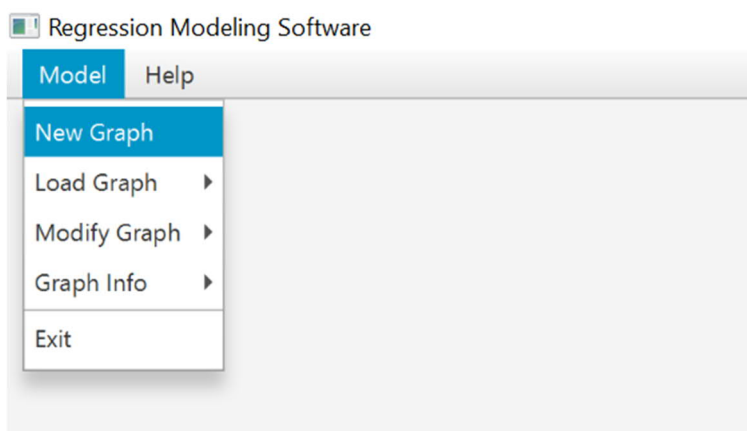
Ohjelma käynnistetään GUI-olion avulla. Kun ohjelma on käynnistetty, avautuu käyttäjälle aloitusnäky. Näkymässä on kaksi painiketta; Model ja Help.



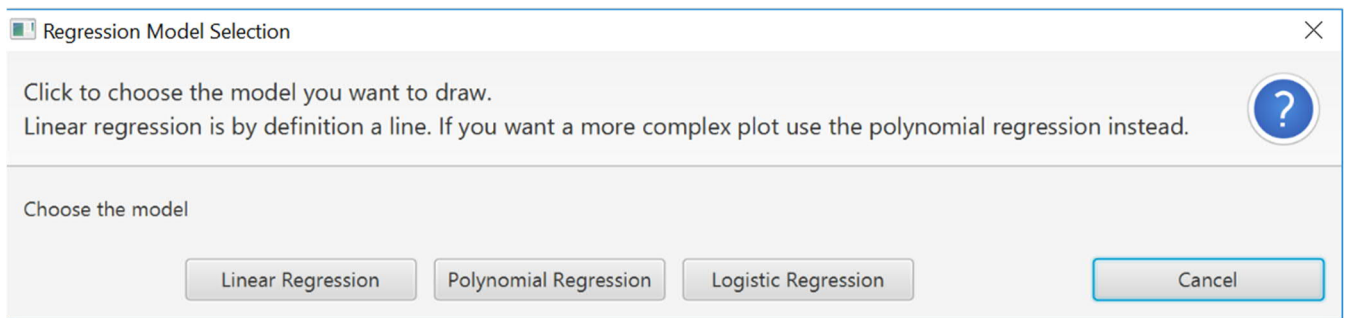
Painamalla Help-valikkoa tulee näkyviin Instructions-nappi. Instructions-nappia painamalla avautuu käyttäjälle apuikkuna. Apuikkunassa käyttäjä voi painaa jotain ikkunan aihetta, jolloin apuikkunan valikon oikealla puolella tulee näkyviin tekstuaalinen selostus aiheesta.



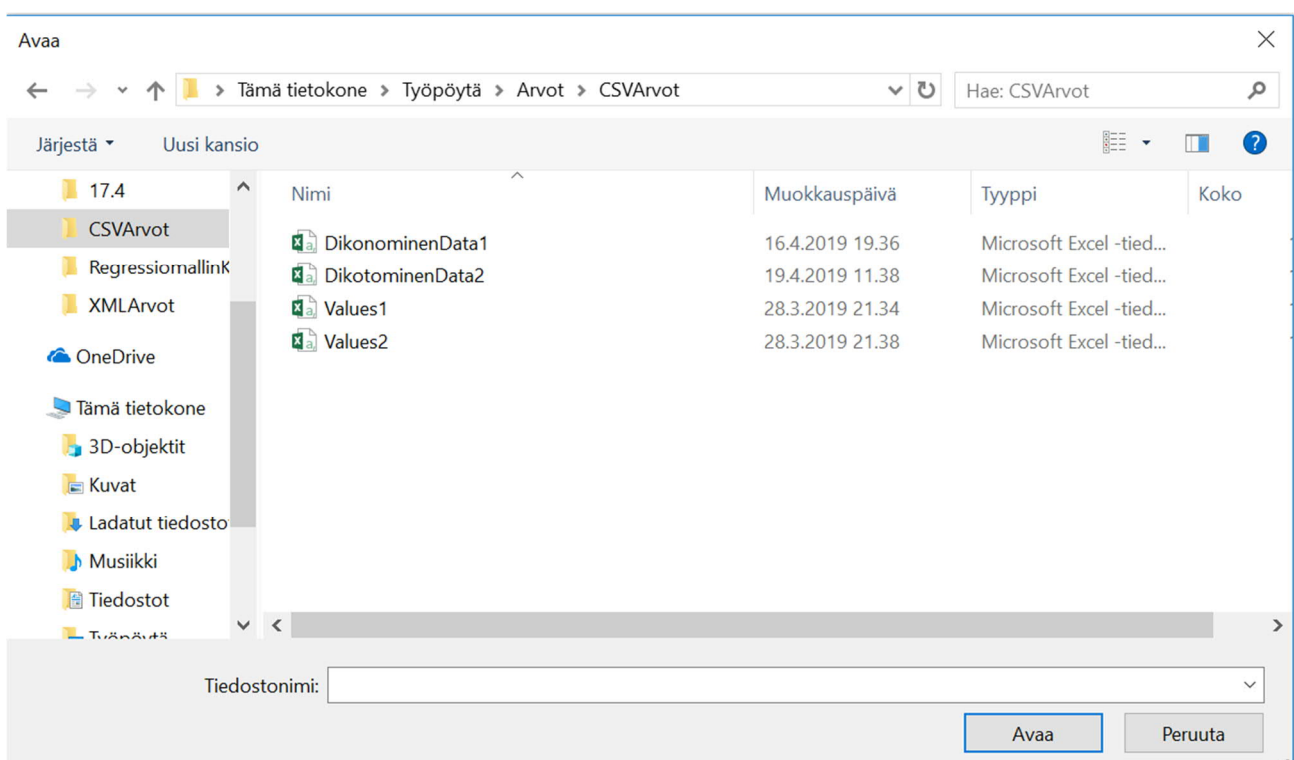
Painamalla Model-valikkoa avautuu käyttäjälle ohjelman päävalikko.



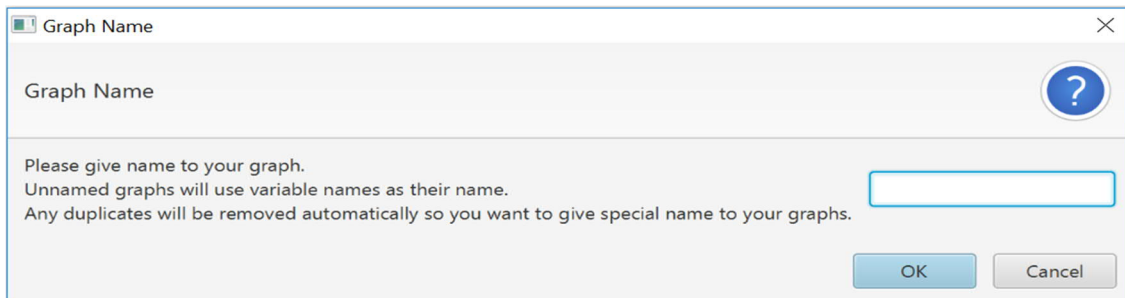
Päävalikosta käyttäjä voi luoda uuden regressiomallin painamalla New Graph -nappia. Tämän jälkeen käyttäjä voi valita lineaarisen, polynomisen tai logistisen regression valintaikkunasta.



Mallin valinnan jälkeen käyttäjälle avautuu näkymä, jossa hän voi valita datapisteiden kohdetiedoston. Ohjelma tukee CSV- sekä XML-tiedostoja.



Tiedoston valinnan jälkeen käyttäjä voi nimetä mallinsa. Jos nimeämistä ei tapahdu, niin mallin nimenä käytetään oletusarvoisesti "Y in terms of X", missä Y on y-muuttujan nimi ja X on x-muuttujan nimi. Jokaisen ohjelmassa olevan mallin nimen on oltava uniikki. Jos käyttäjällä on esimerkiksi lineaarinen regressiomalli nimeltä "Testi" ja hän luo uuden logistisen regressiomallin nimeltä "Testi", korvataan vanha malli uudella. Jos käyttäjä on valinnut piirrettäväsi polynomisen regression, on nimeämisen jälkeen vielä annettava suurin polynomissa esiintyvä aste. Aste on pakko antaa.

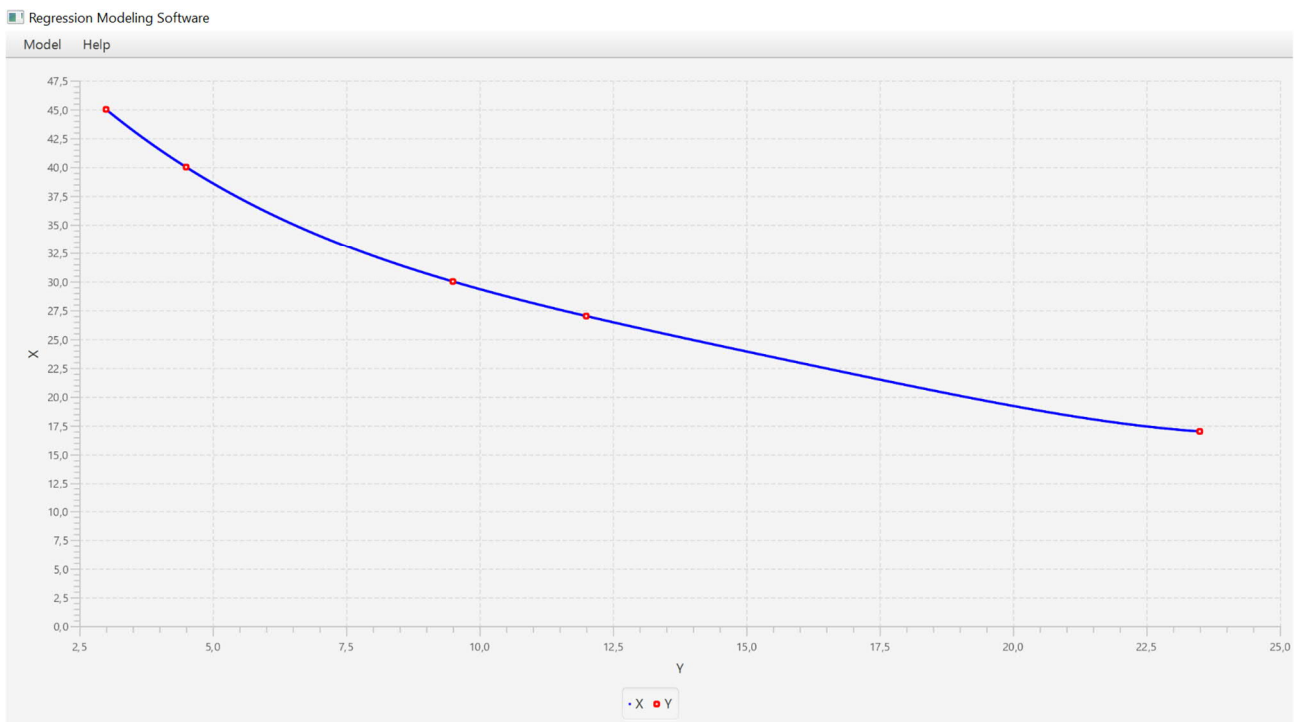


Graph Name

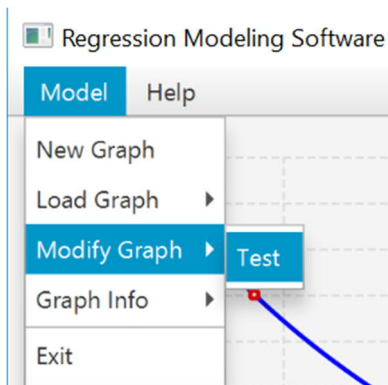
Please give name to your graph.  
Unnamed graphs will use variable names as their name.  
Any duplicates will be removed automatically so you want to give special name to your graphs.

OK Cancel

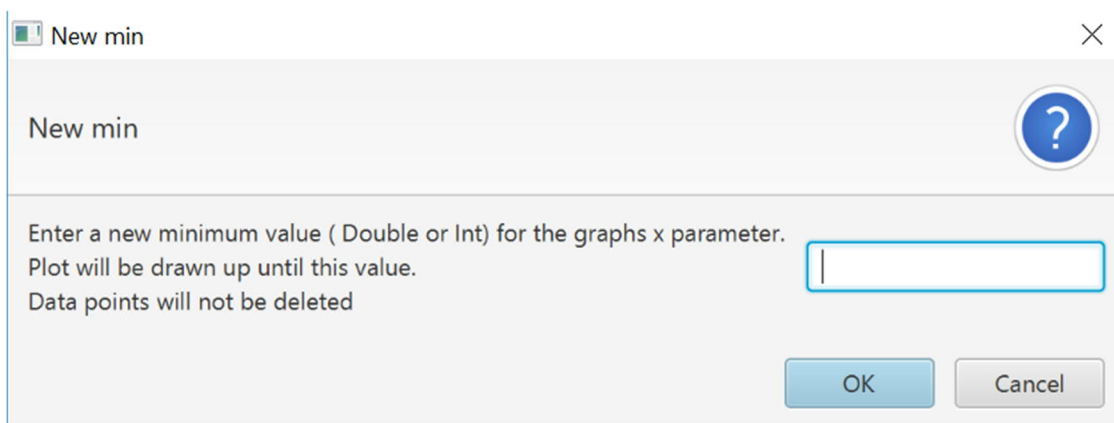
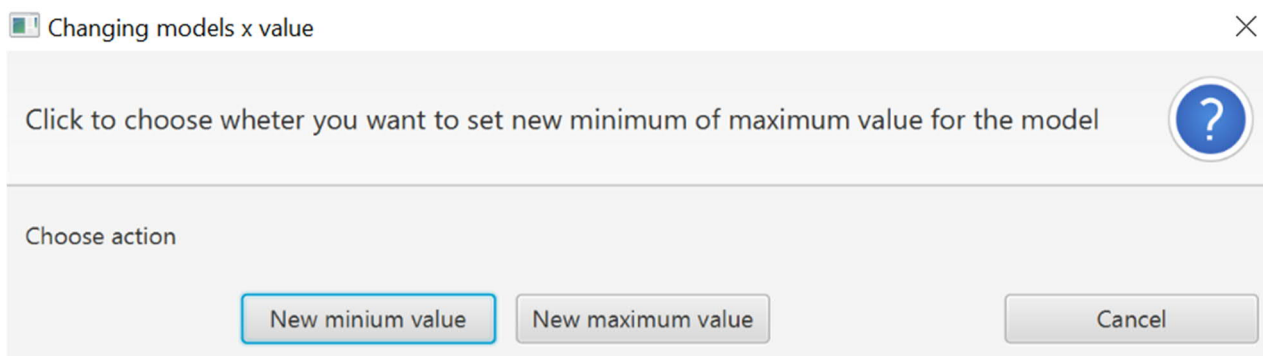
Tämän jälkeen ohjelma piirtää mallin mukaisen kuvaajan. Ohjelma piirtää kuvaajan datan perusteella, joten mallin minimi ja maksimi x-akselin arvo määräytyy annetun datan arvoista. Jos käyttäjä voi halutessaan muuttaa minimi- tai maksimi-arvoa. Kuvaajan piirtämiseen käytettävien ScalaFX:n työkalujen sisäisten ominaisuuksien takia on näkyvän x-akselin väli suurempi, kuin mallin mini ja maksimi x-arvojen väli. Ks. puutteet ja viat yksityiskohtaisempaa selitystä varten.



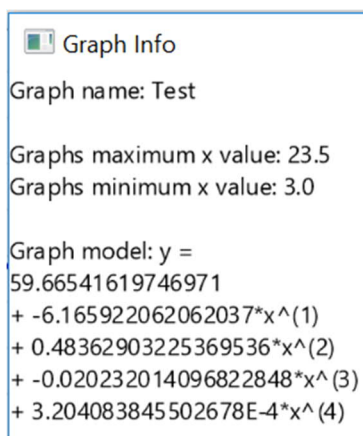
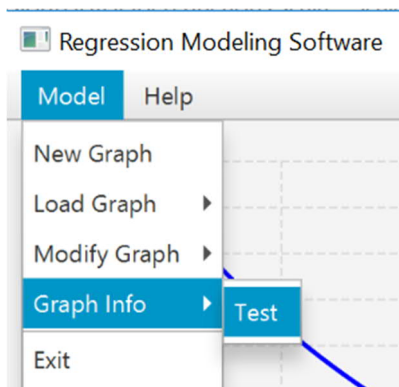
Kun käyttäjä on luonut ainakin yhden mallin, voi hän muokata mallia Model-valikon Modify Graph -napista.



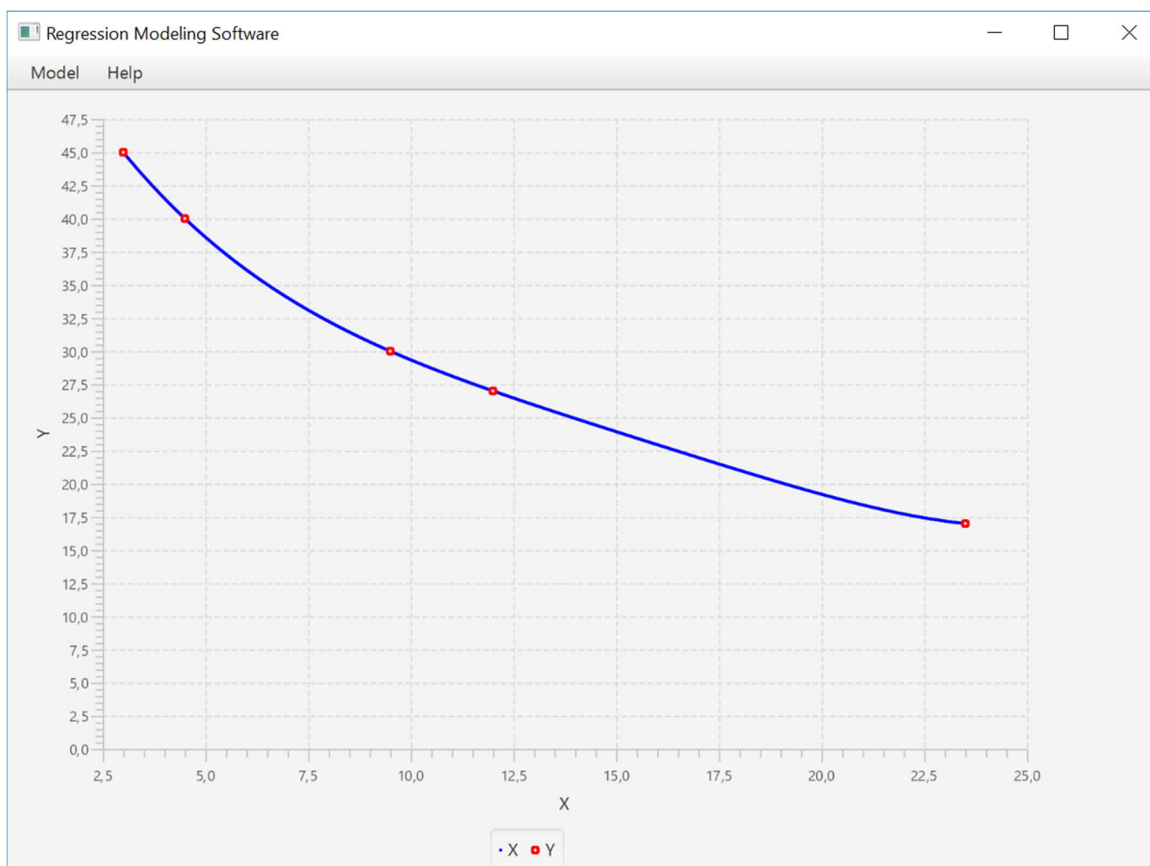
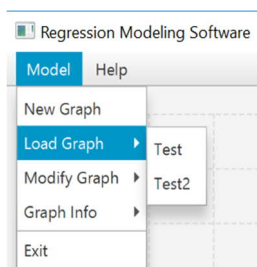
Kun käyttäjä on valinnut nimen perusteella haluamansa mallin, avautuu apuikkuna, jossa mallia voi muokata antamalla sille uuden minimi- tai maksimiarvon. Uuden minimiarvon on oltava pienempi kuin nykyinen maksimiarvo ja uuden maksimiarvon on oltava suurempi kuin nykyinen minimiarvo. Jos käyttäjä antaa niin suuren arvon, että kääntäjä tulkitsee sen äärettömäksi (Infinity), varoittaa ohjelma käyttäjää.

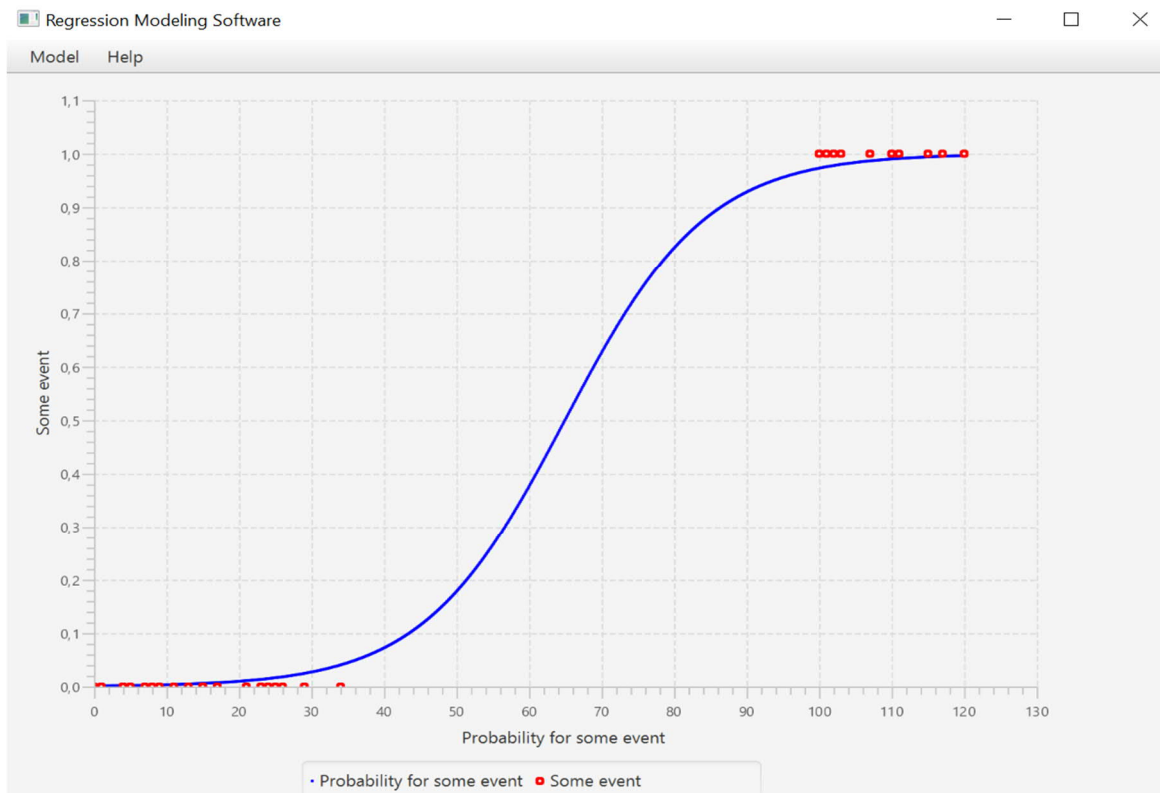


Käyttäjä pääsee myös tarkastelemaan luotujen mallien tietoja valitsemalla Model-valikosta Graph Info -valikon ja hakemalla halutun mallin nimen perusteella. Haluttua mallia painettaessa avautuu apuikkuna, josta käyttäjä näkee seuraavat tiedot mallista: nimi, minimi x-arvo, maksimi x-arvo, sekä mallin kertoimet.



Kun käyttäjä on luonut useamman mallin, voi hän ladata helposti vanhan näkyviin valitsemalla Model-valikosta valikko Load Graph ja hakemalla haluttu malli nimen perusteella.





Ohjelman voi lopettaa painamalla Model-valikossa olevaa Exit-nappia tai näkymän oikean yläaidan ruutua. Piirretyt mallit ovat ohjelman muistissa vain ajon ajan eikä niitä tallenneta minnekään.

## Ohjelman rakenne

Ohjelman luokat jakaantuvat selkeästi front ja back endiin. Syy erottelulle on se, että ohjelman lähdekoodia on helpompi lukea, sekä ylläpitää, kun osa luokista ja olioista on luotu vain graafisen käyttöliittymän tarpeisiin, toisten luokkien ja olioiden hoitaen muita tehtäviä kulisseissa.

Back end -luokkia ovat DataCollector, Parser, Plotter, Loader, GeneralGraph, GeneralRegressionMatrixCalculator, PolynomialGraph, LinearGraph, LogisticGraph, LogisticRegressionCalculator, sekä RegressionMatrixCalculator. Myös poikkeusluokat InvalidFileData, sekä InvalidInput voidaan lukea back end -luokkiin.

Front end -luokkiin kuuluvat luokat GUI, sekä olio GUIHelperMethods.



Luokan GUI, sekä GUIHelperMethods:n avulla mallinnetaan graafinen käyttöliittymä. Luokka GUI pitää sisällään graafisen käyttöliittymän muotoilun, sekä tapahtumakuuntelut käyttäjän toiminnalle.

## GUIHelperMethods

Olio GUIHelperMethods on täynnä erinäisiä funktioita, joita GUI tarvitsee ja jotka on siirretty yksittäisolioon luettavuuden, selkeyden, sekä ylläpidettävyyden takia. Graafista käyttöliittymää on helpompi jatkokehittää olemassa olevan pohjalta, kun monet rajapintaan liittyvät toiminnot on abstrahoitu yksittäisolioin metodeiksi: suunnittelijan ei tarvitse tietää metodien sisäistä toimintaa, kunhan hän tietää mihin mitäkin käytetään.

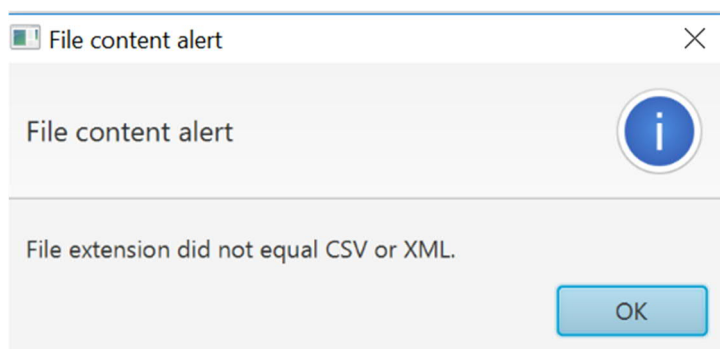
GUIHelperMethods:n keskeiset metodit

```
def formGraph(model: String, graphName: String, yAndXNames: (String, String),  
rawDataValues: Buffer[(Double, Double)]): Option[General Graph]
```

Metodi formGraph ottaa parametreikseen käytettävän mallin, piirrettävän graafin nimen, y ja x muuttujien nimet, sekä datapisteet. Metodi palauttaa Option-kääreessä tyyppiä GeneralGraph olevan palautusarvon. Metodi tunnistaa sisäisesti, jos mallin luonnissa tulee ongelmia, kuten esimerkiksi, jos annetussa datassa on niin suuria arvoja, että lineaarista, sekä polynomista regressiomallia varten tehtävät matriisioperaatiot eivät onnistu. Tällöin metodi ilmoittaa asiasta käyttäjälle toisen GUIHelperMethods:n metodin invalidDataDetected avulla ja palauttaa None-arvon.

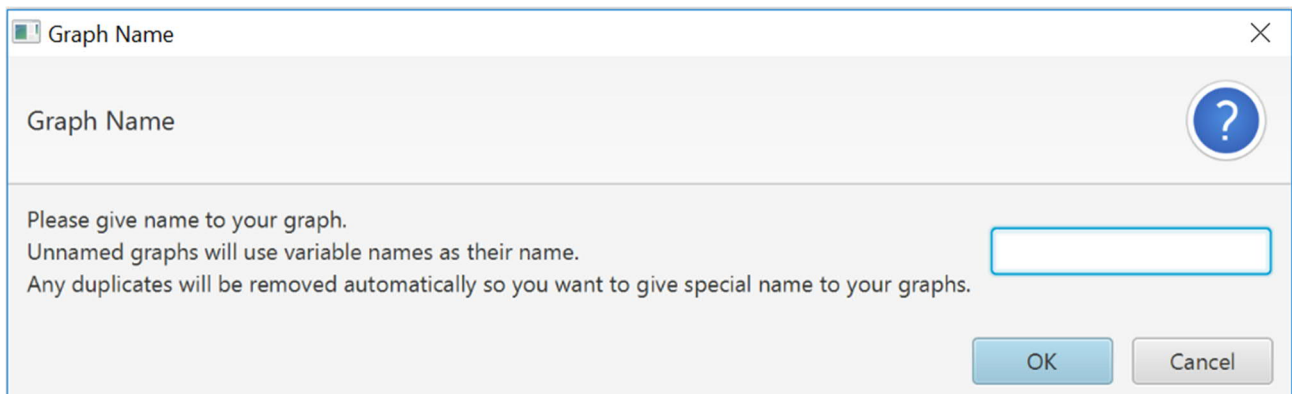
```
def invalidDataDetected(content: String, titleString: String, header: String)
```

Metodi invalidDataDetected luo varoituksen metodia kutsuttavaan nimiavaruuteen. Metodi ottaa parametreikseen käytettävän sisällön (content), otsikon (titleString), sekä ylätunnisteen (header). Alla on esimerkki, jossa ohjelmassa on annettu vääränlainen tiedosto.



```
def userInputBox(content: String, title: String, header: String):  
ObjectProperty[String]
```

Metodi userInputBox ottaa täsmälleen samanlaiset parametrit, kuten invalidDataDetected. Kyseisen metodin ero on se, että tällä metodilla käyttäjä voi antaa ohjelmalle tekstuaalista tai numeerista syötettä. Metodi palauttaa ScalaFX:n TextInputDialog:n palautusarvona ObjectProperty[String]. Alla on esimerkkikuva.



```
def getGraphOrder(): Int
```

Metodi getGraphOrder pyytää käyttäjältä korkeimman polynomin asteen polynomista regressiota varten. Käyttäjän on annettava metodille jokin lukuarvo. Kaikki annetut arvot pyöristetään alaspäin lähimpään kokonaislukuun.

```
def helpWindow()
```

Metodi helpWindow luo kutsuttaessa täysin uuden apuikkunan, jossa käyttäjä voi painaa jotain aihealuetta ja saada tähän liittyvää tietoa. Metodi vastaa Help-valikon Instruction-napin luomasta näkymästä.

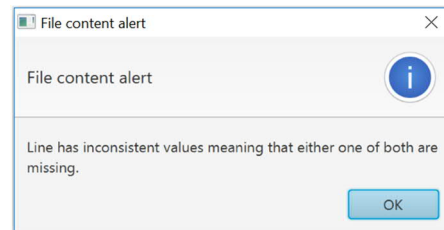
```
def addGraph(graph: General Graph, loader: Loader, menu: Menu*)
```

Metodi addGraph ottaa parametrinaan ohjelman muistiin lisättävän graafin, käytettävän loader-luokan instanssin, sekä vaikutuksen kohteena olevat valikot (menu). Metodi lisää graafin loaderin muistiin, sekä huolehtii samannimisten graafien poistamisesta loaderin oman addGraphAndRemoveDuplicates:n avulla.

```
def askForFileAndGetValues(file: File): (Buffer[(Double, Double)], (String, String))
```

Metodi askForFileAndGetValues huolehtii käytettävän tiedoston lataamisesta, sekä olioiden DataCollector ja Parser:n kanssa, että ladattava tiedosto sisältää oikeita arvoja

oikein esitettynä. Metodi ilmoittaa, jos tiedosto on väärän tyyppinen, arvoja puuttuu tai



löydetty arvot ovat väärä.

## DataCollector

DataCollector-olio huolehtii Parser-olion kanssa datan lukemisesta, sekä lataamisesta annetusta tiedostosta. DataCollectorin tehtävä on tarkastella, että onko annettu tiedostotyyppi oikeanlainen, valita tiedostotyyppin perusteella oikea metodi tiedoston lukemiseen, sekä muuttaa Parser:n palauttama String-tyyppinen data numeeriseksi, pitäen samalla huolen mahdollisista poikkeuksista muunnon yhteydessä.

DataCollector:n keskeiset metodit

```
def chooseTypeAndExtractFromFile(file: File)
```

Metodi chooseTypeAndExtractFromFile valitsee tiedoston lukemiseen oikean metodin ja palauttaa tiedostosta saamansa arvot. Jos annettu tiedosto ei ole päätteeltään XML tai CSV, niin heittää ohjelma uuden poikkeuksen.

```
def extractDataFromCSVFile(file: File): (Buffer[(Double, Double)], (String, String))
```

Metodi extractDataFromCSVFile saa parametrikseen CSV-tyyppisen tiedoston, ja palauttaa monikon, jonka arvot ovat puskurissa olevat monikkomuotoiset datapisteet, sekä datapisteiden muuttujien nimet monikkomuodossa

```
def extractDataFromXMLFile(file: File): (Buffer[(Double, Double)], (String, String))
```

Metodi extractDataFromXMLFile toimii täysin samoin, kuin extractDataFromCSVFile, mutta tyypiltään XML oleville tiedostoille.

## Parser

Parser-olio toimii DataCollector-olion suorassa alaisuudessa: DataCollector tunnistaa tiedostotyyppin ja valitsee Parser:lta tämän perusteella oikean metodin tiedoston lukemiseen. Parser osaa tarkastaa, että annetun tiedoston riveiltä löytyy vähintään kaksi arvoa y- ja x-muuttujaa varten. CSV-tiedostoissa Parser käyttää puolipistettä (;) kahden

arvon erottimena: Syy tähän on se, että Exceliä käytettäessä erotetaan sarakkeet toisistaan puolipisteellä.

Parser:n keskeiset metodit

```
def readFromCSVFile(file: File): Buffer[(String, String)]
```

Metodi readFromCSVFile saa parametrinaan CSV-tyyppisen tiedoston ja palauttaa puskurissa tyyppiltään (String, String) olevan monikon.

Tiedoston sarakkeista vain kaksi ensimmäistä oletetaan merkityksellisiksi. Jos jollain rivillä on puutteellinen määrä dataa, heittää metodi poikkeuksen.

Jos muuttujien nimet puuttuvat tiedostosta tai tiedostossa ei ole yhtään datapistettä muuttujien nimien lisäksi, heittää metodi poikkeuksen.

```
def readFromXMLFile(file: File): Buffer[(String, String)]
```

Metodi readFromXMLFile toimii pääpiirteittäin samoin, kuin metodi readFromCSVFile. Erona on se, että XML-tiedostossa on oltava tietyt tagit. Näitä ovat variableNames, variableNames:n sisällä, names, x, y, sekä tagi values, joka pitää sisällään XML-tiedoston data-arvot. Jokainen data-arvo on esitettävä omalla rivillään tagissa value.

Jos XML-tiedoston rakenne on puutteellinen (tageja puuttuu yms.) heittää metodi poikkeuksen, kuten myös jos em. tagit puuttuvat.

## Plotter

Plotter-olion tehtävä on muodostaa kuva käsiteltävästä graafista. Kuvan muodostus tapahtuu palauttamalla ScatterChart, joka pitää sisällään halutut arvot ja akselit.

Plotter:n keskeiset metodit

```
def drawGraph(xName: String, yName: String, graph: General Graph, step: Double, maximumX: Double): ScatterChart[Number, Number]
```

Metodi drawGraph ottaa parametreikseen x- ja y-akselien nimet (xName, yName), piirrettävän graafin (graph), x-arvojen askelpituuden (step), sekä maksimi x-arvon (maximumX). Metodi palauttaa ScatterChartin, jossa on halutut arvot ja akselit.

## Loader

```
class Loader(val graphs: Buffer[General Graph])
```

Loader-luokka pitää huolta ohjelman ajon aikana luotujen graafien varastoinnista.

Loader:n keskeiset metodit

```
def addGraphAndRemoveDuplicates(graph: General Graph): Unit
```

Metodi addGraphAndRemoveDuplicates lisää graafin Loader:n graphs puskuriin.

Puskurissa graphs olevien graafien tulee olla erinimisiä. Jos esimerkiksi on olemassa graafi Testi, ja ajon aikana luodaan toinen graafi nimeltä Testi, uusi graafi tallennetaan vanhan päälle.

## GeneralGraph

GeneralGraph on piirreluokka, joka pyrkii kuvaamaan kaikille graafeille yhteisiä ominaisuuksia. Ohjelma on rakennettu niin, että se toimii GeneralGraph-piirreluokan metodien varassa. Näin uuden graafin tulee toteuttaa vain piirreluokan metodit toimiakseen oikein.

### GeneralGraph:n keskeiset metodit

```
def calculatePlotPoints(step: Double, lowerBound: Double, upperBound: Double):  
Buffer[(Double, Double)]
```

Metodi calculatePlotPoints ottaa paremètreikseen askelpituuden (step), alarajan (lowerBound), sekä ylärajan (upperBound) arvot, jotka ovat kaikki tyypiltään Double. Metodi palauttaa puskurissa käyrän lasketut pisteet. GeneralGraphiä perivät luokat pitävät huolen monikon pisteiden järjestyksestä. Ohjelmassa oletetaan, että metodi palauttaa pisteet muodossa (y, x).

```
def dataPointsUntilMaxX(maximum: Double): Buffer[(Double, Double)]
```

Metodi dataPointsUntilMaxX ottaa parametrikseen käytettävän maksimin (maximum) ja palauttaa puskurissa alkuperäiset datapisteet annettuun maksimi x-arvoon saakka.

```
def name: String
```

Metodi name palauttaa graafin nimen.

```
def giveVariableNames: (String, String)
```

Metodi giveVariableNames palauttaa String-monikossa ajon aikana ohjelmalla annetusta tiedostosta löytyneet muuttujien nimet.

```
def maximumXDataPoint: Double
```

```
def minimumXDataPoint: Double
```

```
def setMaximumXDataPoint(newMax: Double): Unit
```

```
def setMinimumXDataPoint(newMin: Double): Unit
```

Metodit maximum- ja minimumXDataPoint palauttavat graafin sen hetkisen maksimi tai minimi x-arvon. Metodeilla setMaximum- ja setMinimumXDataPoint voidaan asettaa uusi haluttu maksimi x-arvo annetun parametrin mukaan.

```
def estimateY(value: Double): Double
```

Metodi estimateY palauttaa graafin regressiomallin arvion y-arvolla parametrin value mukaan.

```
def roundedMaximumXPoint(roundToThis: Double): Double
```

Metodi roundedMaximumXPoint palauttaa graafin maksimi x-arvon pyöristettynä annettuun roundToThis-parametriin.

## GeneralRegressionMatrixCalculator

GeneralRegressionMatrixCalculator on piirreluokka, jota käytetään kaikissa matriisipohjaisissa regressiomalleissa. Piirreluokka varmistaa, että kaikki matriisipohjaiset mallit palauttavat kerroinvektorin.

GeneralRegressionMatrixCalculator:n keskeiset metodit

```
def deliverCoefficients(): SimpleMatrix
```

Metodin deliverCoefficients tarkoitus on tuottaa matriiseihin perustuvan regressiomallin kerroinvektori.

## RegressionMatrixCalculator

```
class RegressionMatrixCalculator(val inputData: Buffer[(Double, Double)],
```

```
val polynomialOrder: Int) extends GeneralRegressionMatrixCalculator
```

Luokka RegressionMatrixCalculator on tarkoitettu tuottamaan kertoimet regressiomalleille, jotka voidaan esittää matriisimuodossa. Ohjelmassa nämä ovat polynominen, sekä lineaarinen regressio.

Luokka saa parametreinaan käytettävän datan puskurissa (inputData), sekä käytettävän polynomin suurimman asteen (polynomialOrder).

## PolynomialGraph

```
class PolynomialGraph(val name: String, val variableNames: (String, String),
```

```
val polynomialOrder: Int, private var originalData: Buffer[(Double, Double)])
```

```
extends GeneralGraph
```

Luokka PolynomialGraph on polynomisen regression kuvaamiseen tarkoitettu luokka. Se saa tarvitsemansa kertoimet käyttämällä RegressionMatrixCalculatoria sisäisesti. Se saa parametreinaan graafin nimen (name), datajoukon muuttujien nimet (variableNames), käytettävän polynomin korkeimman asteen (polynomialOrder), sekä tiedostosta ladatun datan (originalData). Polynomin asteen tulkitaan ohjelmassa arvona  $n - 1$  matriisiesityksen takia.

Käyttäjää pyydetään huomioimaan, että korkea-asteisissa polynomeissa vaara yli- tai alisovitukseen kasvaa huomattavasti.

PolynomialGraph:n keskeiset metodit

```
override def toString: String
```

GeneralGraph-piirreluokan metodien lisäksi PolynomialGraph-luokalla on oma toString-metodi, jossa polynomi esitetään muodossa  $y = b + a_1 \cdot x_1 + \dots + a_n \cdot (x_n)^k$ .

## LinearGraph

```
class LinearGraph(name: String, variableNames: (String, String),  
polynomialOrder: Int = 2, private var originalData: Buffer[(Double, Double)]) extends  
PolynomialGraph(name, variableNames, polynomialOrder, originalData)
```

Luokka LinearGraph toimii muuten täysin samoin, kuin luokka PolynomialGraph paitsi, että sen käytettävän polynomin korkein aste on vakioarvona kaksi.

## LogisticGraph

```
class LogisticGraph(val name: String, val variableNames: (String, String),  
private var originalData: Buffer[(Double, Double)]) extends GeneralGraph
```

LogisticGraph on logistisen regression kuvaamiseen tarkoitettu luokka.

Se saa parametreinaan graafin nimen (name), muuttujien nimet (variableNames), sekä tiedostosta ladatun datan (originalData).

LogisticGraph:n keskeiset metodit

```
override def toString: String
```

GeneralGraph-piirreluokan metodien lisäksi LogisticGraph:llä on oma toString-metodi, jossa palautetaan graafin kertoimet, sekä yhtälön muoto.

## LogisticRegressionCalculator

```
class LogisticRegressionCalculator
```

Luokka LogisticRegressionCalculator huolehtii kertoimien luomisesta logistista regressiota varten. Se ei käytä matriisimuotoista esitystä, kuten RegressionMatrixCalculator, vaan numeerisia algoritmeja.

LogisticRegressionCalculator:n keskeiset metodit

```
def deriveThetas(yData: Array[Double], xData: Array[Double]): Buffer[Double]
```



Metodi `deriveThetas` ottaa parametreikseen selitettävän datan (`yData`), selittävän datan (`xData`) ja palauttaa näistä numeerisilla menetelmillä saadut kertoimet logistista regressiota varten. Ks. kohta algoritmit lisäselvitystä varten.

## InvalidFileData

```
case class InvalidFileData(description: String) extends  
java.lang.Exception(description)
```

Poikkeusluokkaa `InvalidFileData` käytetään pääasiallisena poikkeusluokkana projektissa.

## InvalidInput

```
case class InvalidInput(description: String) extends java.lang.Exception(description)
```

Poikkeusluokkaa `InvalidInput` käytetään projektissa poikkeuksissa, jotka liittyvät käyttäjän antamaan syötteeseen.

## Omat pohdinnat

Monille ohjelman osa-alueille on lukuisia vaihtoehtoisia toteutustapoja. Yksi vaihtoehto olisi esimerkiksi liittää GUI-olio vahvemmin back end -puoleen yhdistämällä yksittäisoliot, kuten `Plotter:n` tai `GUIHelperMethods:n` suoraan siihen. Tämä tosin tekisi GUI:sta vaikeammin luettavan rivimäärien kasvaessa valtavasti. Nykyinen toteutus GUI:n osalta tähtää yleisyyteen, sekä luettavuuteen, kun GUI pyytää yksittäisolioilta tarvitsemiaan asioita, pitäen itse huolen graafisen käyttöliittymän näkyvyydestä.

Toinen asia, jonka voisi toteuttaa toisin, olisi se, että Graph-olioiden sijasta GUI:ssa käsiteltäisiin vain regressiomallin kertoimia. Tämä liittyy vahvasti ajatukseen tuoda GUI:ta enemmän back end -puoleen. Jos esimerkiksi GUI:ssa tallennettaisiin tiedostosta ladatut arvot talteen, niin halutun mallin piirtoa varten tarvitaan vain kertoimet. Tällöin esimerkiksi samoja kertoimia voisi käyttää eri datajoukkoon ja katsoa miten kuvaaja suhteutuu annettuihin pisteisiin. Nykyinen toteutus poikkeaa tästä, sillä GUI pyytää yksittäisolioiden metodeilta arvoja ja ohjaa Graph-olioiden luontia, sekä käsittelyä yksittäisolioiden metodien avulla. Koska muut oliot, sekä luokat huolehtivat yksityiskohtaisesta toteutuksesta GUI pystyy ohjaamaan tapahtumankulkua korkean abstraktion tasolla.

# Algoritmit

Polynomisen regression kertoimien selvitys:

(Koska suora on polynomin erikoistapaus, seuraavassa käsitellään vain polynomisen regression kertoimien selvitys.)

Kun datajoukon yksittäinen havainto on  $y_i$ , on sitä selittämässä yhtälö

$y_i = b + a_1 * (x_i) + a_2 * (x_i)^2 + \dots + a_k * (x_i)^k$ , kun  $k$  on polynomin korkein aste. Jos yhtälöjä on yhteensä  $n$  kpl, niin ne voidaan kirjoittaa muodossa

$$y_1 = b + a_1 * (x_1) + a_2 * (x_1)^2 + \dots + a_k * (x_1)^k$$

$$y_2 = b + a_1 * (x_2) + a_2 * (x_2)^2 + \dots + a_k * (x_2)^k$$

⋮

$$y_n = b + a_1 * (x_n) + a_2 * (x_n)^2 + \dots + a_k * (x_n)^k.$$

Nämä  $n$  lineaarista yhtälöä saadaan helposti matriisiesitykseen, sillä nyt

$Y = X * B + \epsilon$ ,  $Y$  on pystyvektori  $n$  selitettävästi muuttujasta,

$$X \text{ on } n \times k \text{ matriisi muotoa } X = \begin{bmatrix} 1 & \dots & x_1^k \\ \vdots & \ddots & \vdots \\ 1 & \dots & x_n^k \end{bmatrix},$$

$$B \text{ on kerroinvektori muotoa } B = \begin{bmatrix} 1 \\ a_n \end{bmatrix}.$$

ja  $\epsilon$  on virhevektori, joka oletetaan nolllaksi.

Tällöin yhtälöstä  $Y = X * B$  saadaan kerroinvektori, mikäli  $X$  on kääntyvä.

Lineaarialgebrasta tulevan Moore-Penrosen näennäiskäänteismatriisin avulla yhtälö saadaan ratkaistua.

Nyt  $B = (X^T * X)^{-1} * Y$ , jolloin kerroinvektori on selvillä.

Käänteismatriisin, sekä matriisi tulon toteuttaa käytettävän EJML-kirjaston metodit `pseudoInverse()` ja `mult()` .

Logistisen regression kertoimien selvittäminen:

Toisin kuin lineaarisessa tai polynomisessa regressiossa, ei logistisessa regressiossa ole olemassa matriisiesitystä. Logistisessa regressiossa pyritään selittämään binääristä dataa numeerisen selittävän muuttujan avulla. Tällöin annettuun datajoukkoon pyritään sovittamaan sigmoidin mallinen logistinen käyrä, s-käyrä, jolloin käsiteltävä yhtälö on muotoa

$$h_{\theta}(X) = \frac{1}{1 + e^{-(\theta^T * X)}}$$

missä  $\theta$  on kerroinvektori ja  $X$  on havainnon piirrevektori. Ohjelman toteutuksen puitteissa selittäviä muuttujia voi olla vain yksi, jolloin  $X_i = \frac{1}{x_i}$ .

Ohjelmassa kertoimet ratkaistaan numeerisesti gradient descent -algoritmillä, jossa kertoimet  $\theta_0$  ja  $\theta_1$  selvitetään iteroimalla niiden osittaisderivaattoja.

Tämä onnistuu seuraavasti.

Muodostetaan ensin hypoteesifunktion  $h_{\theta}(X)$  ns. kustannusfunktio (cost function), eli funktio, joka kertoo kuinka paljon hypoteesifunktio poikkeaa oikeasta arvosta. Logistista regressiota varten kustannusfunktio on määritelty

$$Cost(h_{\theta}(x), y) = \begin{cases} -\log(h_{\theta}(x)) : y = 1 \\ -\log(1 - h_{\theta}(x)) : y = 0 \end{cases}$$

Ja tämä paloittain määritelty funktio voidaan saattaa muotoon

$$Cost(h_{\theta}(x), y) = -y \log(h_{\theta}(x)) - (1 - y) \log(1 - h_{\theta}(x)).$$

Huomautus notaatiosta:  $x^{(i)}$  tarkoittaa havainnon  $i$  selittävää piirrevektoria,  $y^{(i)}$  havainnon  $i$  selitettävää muuttujaa,  $x_j^{(i)}$  havainnon  $i$  selittävän piirrevektorin piirrearvoa  $j$ .

Nyt funktio, jota halutaan optimoida kertoimien  $\theta$  mukaan, kun datapisteitä on  $m$  kpl, on

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_{\theta}(x^{(i)}), y^{(i)})$$

$$J(\theta) = -\frac{1}{m} [\sum_{i=1}^m (y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})))] .$$

Optimointi tapahtuu kertoimien osittaisderivaattojen avulla.

*repeat until converge {*

*for i = 0 and i = 1*

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_j)$$

*}*

Missä  $\alpha$  on algoritmin oppimismisnopeus (learning rate). Iterointi alkaa asettamalla kertoimille jokin alkuarvaus, tyypillisesti nolla. Ohjelmassa kertoimien alkuarvo on nolla. Teoriassa algoritmia iteroidaan, kunnes nollakohtaan päästään, mutta käytännössä toteutuksessa esiintyy yläraja iteraatioilla, sekä virheraja  $\epsilon$ : jos  $\theta_j$  ja  $\theta_{j+1}$  etäisyys on epsilonin luokkaa, ollaan riittävän lähellä kustannusfunktion globaalia minimiä.

Kertoimet päivitetään samanaikaisesti, eli

$$\tau_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0)$$

$$\tau_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

$$\theta_0 = \tau_0$$

$$\theta_1 = \tau_1$$

Optimoinnissa esiintyvä funktion  $J(\theta)$  osittaisderivaatta on muotoa

$$\frac{\partial}{\partial \theta_j} J(\theta_j) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

## Pohdinnat

Logistisessa regressiossa esiintyvää gradient descent -algoritmia olisi voinut käyttää myös lineaarisen, sekä polynomisen regression kertoimien löytämiseen. Syy matriisialgoritmin käytölle on se, että lineaarinen, sekä polynominen regressio toteutettiin jo projektin alkuvaiheessa, kun taas logistinen regressio viimeisen kahden viikon aikana. Näin ollen aikaa muunnostyölle, sekä testaukselle ei ollut. Matriisiesitys on myös matemaattisesti kompakti, joten ohjelmiston rakentaminen sen ympärille on yksinkertaisempaa. Käytettävä EJML-kirjasto on ollut riittävän tehokas työssään, että matriisiesityksen mahdollisesti korkeampi aikavaativuus ei ole osoittautunut ongelmaksi.

## Tietorakenteet

Projektissa yleisimmin käytetty tietorakenne on puskuri, koska sen muuttuvatilaisuus on eduksi tilanteissa, joissa ei voida ennalta tietää pystytäänkö etenemään seuraavaan vaiheeseen tai kohdataanko suorituksen aikana virhettä. Esimerkiksi, kun tiedostosta ladataan datapisteitä, niin on parempi käyttää puskuria: Ei voida etukäteen tietää, kuinka iso tiedosto on, sekä onko tiedostossa oleva tieto oikeellista. Tieto tallennetaan puskuriin vain, jos se on annettujen ehtojen mukaista. Toinen esimerkki on Loader-olio: Ei voida ennalta tiedää, kuinka monta graafia käyttäjä haluaa luoda ajon aikana, joten on parempi antaa tämän luoda niin monta kuin hän haluaa.

Joidenkin luokkien algoritmeissa, esim. LogisticRegressionCalculator:n deriveThetas tai RegressionMatrixCalculator:n deliverCoefficients sisäisessä toteutuksessa käytetään puskurin sijasta taulukkoa, koska kyseisissä algoritmeissa käsiteltävän datan koko ei muutu, sisäisissä toteutuksissa esiintyvien for-looppien takia taulukon indeksiltä hakemisen nopeus on eduksi ja tärkeimpänä, käytettävä matriisikirjasto toimii hyvin taulukko-tietotyypin kanssa.

Projektin luonteen takia muita hyviä vaihtoehtoja tietotyypeille ei ole. Muuttuvatilaisista tietotyypeistä puskuri on yleiskäyttöisin, joten projektissa se soveltuu erinomaisesti ohjelman moniin eri välivaiheisiin. Tietotyypit, kuten vektori tai lista olisivat soveltuneet

tähän tarkoitukseen huonosti. Tehokkuuden puolesta vektoria voisi käyttää projektin vaativimmissa laskuoperaatioissa, mutta taulukko hoitaa tämän tehtävän paremmin, koska käytetyt kirjastot tukevat sitä jo valmiiksi.

## Tiedostot

Ohjelma tukee CVS- ja XML-tiedostotyyppejä. Jos tiedostoissa käytetään liian isoja data-arvoja voi esiintyä ongelmia. Esimerkiksi ScalaFX-kirjaston ominaisuuksien takia, liian isot arvot johtavat siihen, että piirretyssä kuvaajassa itse kuvaajaa ei näy, koska y- ja x-akselien arvot vievät koko tilan. Jos käytetyt arvot ovat hyvin lähellä Scalan `Double.MaxValue`:ta, niin koko kuvaajaa ei välttämättä piirretä laisinkaan. Tilanteissa, joissa liian suuret arvot johtavat matriisioperaatioiden epäonnistumiseen kehottaa ohjelma käyttäjää skaalaamaan annetut arvot uudelleen. Tiedostoissa on käytettävä pistettä (.) pilkun (,) sijasta desimaalierottimena.

CSV-tiedostoissa rakenne on hyvin yksinkertainen. Vain tiedoston kaksi ensimmäistä saraketta tulkitaan merkityksellisiksi. Ensimmäisen sarakkeen oletetaan kuvaavan y-arvoja ja toisen x-arvoja. Tiedoston ensimmäisellä rivillä oletetaan olevan muuttujien nimet. Jokaisen merkityksellisen rivin ensimmäisessä kahdessa sarakkeessa on esiinnyttävä arvo. Tyhjiä rivejä ei saa esiintyä merkityksellisten rivien välissä, sillä lukeminen lopetetaan tyhjään riviin. Käytetyt nimet voivat olla mielivaltaisia. Rivillä olevat arvot erotetaan toisistaan puolipisteen (;) avulla.

Eräissä testeissä ilmeni, että CSV-tiedoston lukemisessa voi tulla ongelma, jos käyttäjä on luonut merkityksellisten rivien jälkeen useita tyhjiä rivejä painamalla enter-näppäintä. Ongelma on, että kyseisille riveillä jää puolipisteitä, vaikka arvoja niissä ei enää ole. Parserin käyttämän `getLines`-metodi tulkitsee rivit merkityksellisiksi, vaikka niissä on ainoastaan puolipiste. Näin ei pitäisi käydä, mutta tilanteessa, jossa näin käy, käyttäjää pyydetään poistamaan turhat puolipisteet tiedostosta.

XML-tiedostoissa on oltava tietty rakenne. Testit ovat osoittaneet, että se ei ole välttämätöntä, mutta käyttäjää kuitenkin kehoitetaan kirjoittamaan XML:n versio tagi

tiedoston alkuun. Tiedoston on yleisesti oltava XML:n mukaista siten, että avaaville tageille löytyy sulkevä tagi jne.

Tagit, jotka tiedostosta on löydettävä, ovat variableNames, jonka alla on tagi name, jonka sisällä on oltava muuttujien nimet seuraavan esimerkin mukaisesti.

```
<name y="Testi" x="Onnistui"/>
```

Jos y:n tai x:n sijasta käytetään jotain muuta muuttujaa nimen varastointiin heittää Parser poikkeuksen. Y ja x voi sisältää arvonaan mielivaltaisen nimen.

Tiedoston datapisteet on oltava tagin values alla, jossa ne ovat tagin value sisällä. Jokainen datapiste on listattava tagin values alle seuraavan esimerkin mukaisesti.

```
<value y="17.0" x="23.5"/>
```

Kuten muuttujien nimissä, y:n ja x:n on löydettävä tagistä value. Ko. muuttujiin voi sijoittaa mielivaltaista numeerista annettujen huomautusten valossa (liian suuret arvot matriisioperaatioissa).

Kun tiedosto luetaan ajon aikana, niin pyrkii ohjelma aluksi tarkistamaan annetun tiedoston tiedostopäätteen. Jos tiedostopäätte ei ole XML tai CSV, niin heitetään poikkeus. Muut poikkeukset aiheutuvat vääristä arvoista (kirjaimia numeerisen datan sijasta), puuttuvista muuttujien nimistä tai jos riveillä ei ole vaadittua dataa ensimmäisessä kahdessa sarakkeessa.

## Testaus

Ohjelmaa testattiin antamalla sille vääränlaisia tiedostoja, tiedostoja, joissa oli roska-arvoja (kirjaimia ja erikoismerkkejä numeroiden sijasta) tai liian isoja arvoja (Double.MaxValue). Tiedostoihin liittyvät testit olivat hyvin samanlaisia, kuin mitä suunnitelmassa esitettiin.

Testaus toteutettiin pääasiassa kokeilemalla. Mitään tiedostoihin liittyvää testausta ei ollut automatisoitu, vaan se toteutettiin manuaalisesti. Sama pätee myös logistisen regression testaamiseen, sillä saadut kuvaajat olivat järkeviä käytettyyn dataan nähden, vaikka saadut kertoimet poikkesivat muiden ohjelmistojen antamista kertoimista.

Projektista löytyy myös luokka yksikkötestaukseen. Yksikkötestit on toteutettu lineaarista, sekä polynomista regressiota varten.

Ensimmäinen testi, jossa data on täysin satunnaista ja kertoimet on saatu ulkoisista ohjelmista, pärjää projektin matriisialgoritmi hyvin. Toisessa testissä luodaan yksikerrallaan 50 i-asteista polynomia ( $i = 1, 2, \dots, 50$ ) sekä näille 500 pistettä testidataa. Testissä tarkastetaan saako matriisialgoritmi saman kertoimen polynomin korkeimmalle asteelle. Matriisialgoritmi pärjää testissä hyvin.

Kolmannessa testissä luodaan muuttujan `buffSize` (alustettu arvoon 20) verran tapauksia. Yhdessä tapauksessa luodaan `buffSize` verran kertoimia `Scalan Random`-olion avulla niin, että yhden kertoimen arvo on välillä  $[0, \text{buffSize}]$ . Viimeinen saatu kerroin on yksittäisen tapauksen polynomin korkeimman asteen kerroin. Saadut kertoimet tallennetaan puskuriin, kertoimien avulla lasketaan haluttu määrä arvoja (oletusarvoisesti 3000) ja saatujen arvojen perusteella yritetään selvittää alkuperäiset kertoimet. Jos käytettävän polynomin asteesta ja datasta johtuen ei matriisioperaatioissa esiintyvä epätarkkuus ole huomattavaa, pärjää matriisialgoritmi testissä hyvin. Jos matriisioperaatioissa käytettävät arvot ovat suuria, kasvaa kertoimien epätarkkuus niin paljon, että ei algoritmi läpäise kyseistä yksikkötestiä.

Täsmällisen arvoalueen määrittäminen dokumentaatiossa viitatussa "suurelle arvolle" ei ollut aikataulun takia mahdollista. Liian suuri arvo riippuu käytettävän polynomin korkeimmasta asteesta, polynomin yleisestä muodosta (kumoavatko termit toisiaan), sekä itse käytettävien arvojen kokoluokasta.

## Ohjelman tunnetut puutteet ja viat

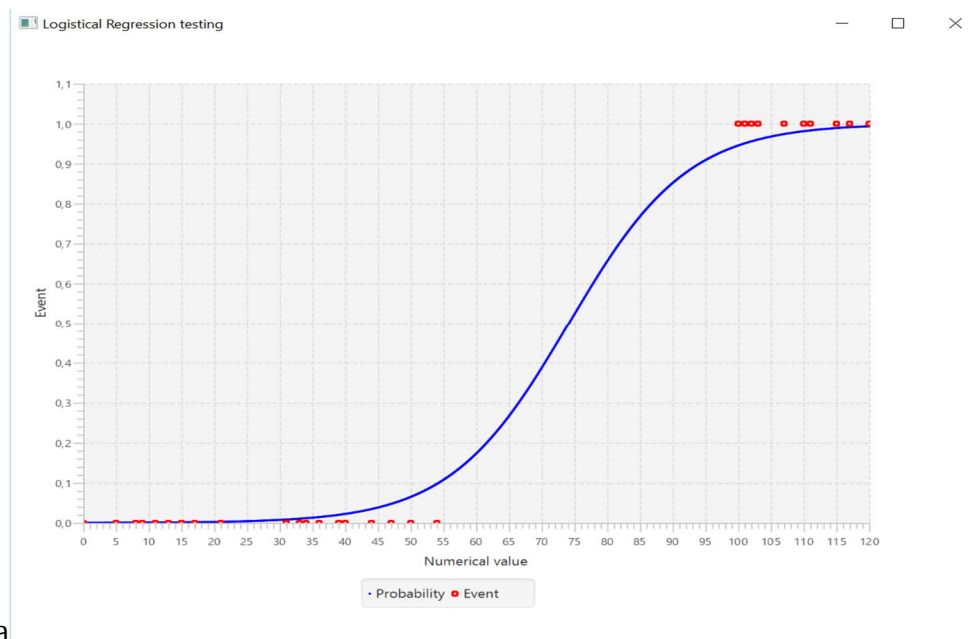


## Logistinen regressio

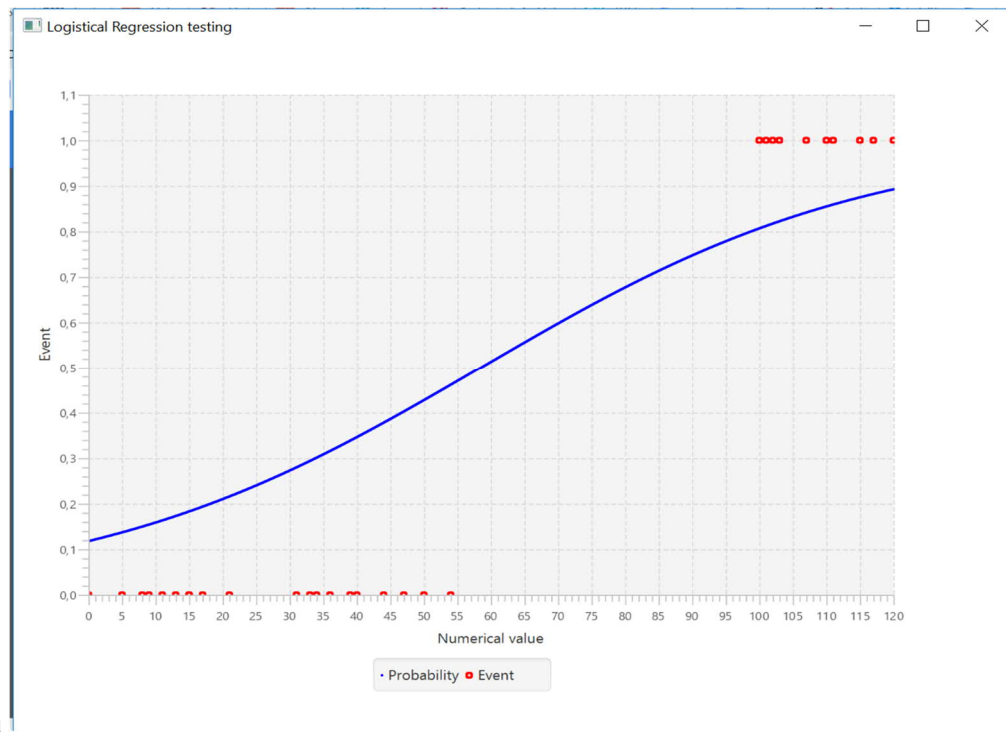
Logistiseen regressioon käytettävä algoritmi tuottaa kuvaajia, jotka käyvät järkeen datan perusteella. Syystä tai toisesta projektin algoritmin tuottamat kertoimet eivät vastaa tunnettujen ohjelmistojen, kuten R, tuottamia kertoimia. Kun kuvaaja piirretään samasta datasta tunnetuilla, sekä projektin ohjelmiston antamilla kertoimilla, ovat kuvaajat joko identtisiä tai vain projektin algoritmin antamat kertoimet tuottavat järkeviä kuvaajia.

Lukuiset testit näyttivät, että projekti tuottaa dataan suhteutettuna järkeviä kuvaajia, vaikka kertoimet eivät vastaakaan tunnettujen ohjelmistojen kuvaajia.

Alla on esimerkkitapauksia käyttäen <http://statpages.info/logistic.html>, sekä RStudio:n antamia kertoimia verrattuna projektin algoritmin antamiin kertoimiin.

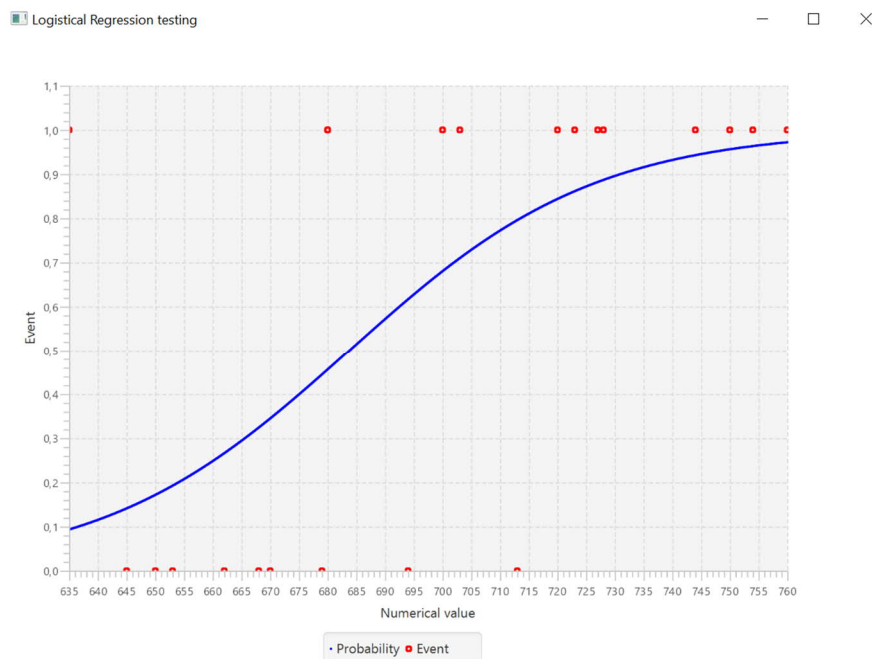


Projektin kuvaaja

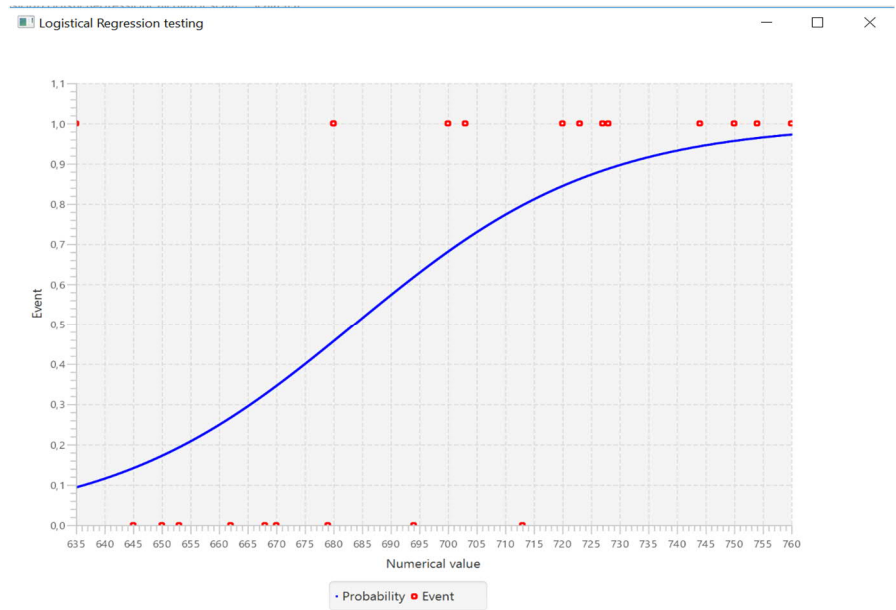


Sivuston kuvaaja

Tässä tapauksessa ero on järjestyttävä, sillä sivuston kertoimien kuvaaja väittää, että vaikka tapahtuma tapahtuu ensimmäisen kerran vasta numeerisella arvolla 100, on silti numeerisella arvolla 5 n. 12% todennäköisyys tapahtua. Projektin algoritmin kuvaajan mukaan n. 12% tapahtumatodennäköisyyteen vaaditaan n. 58 numeerinen arvo.

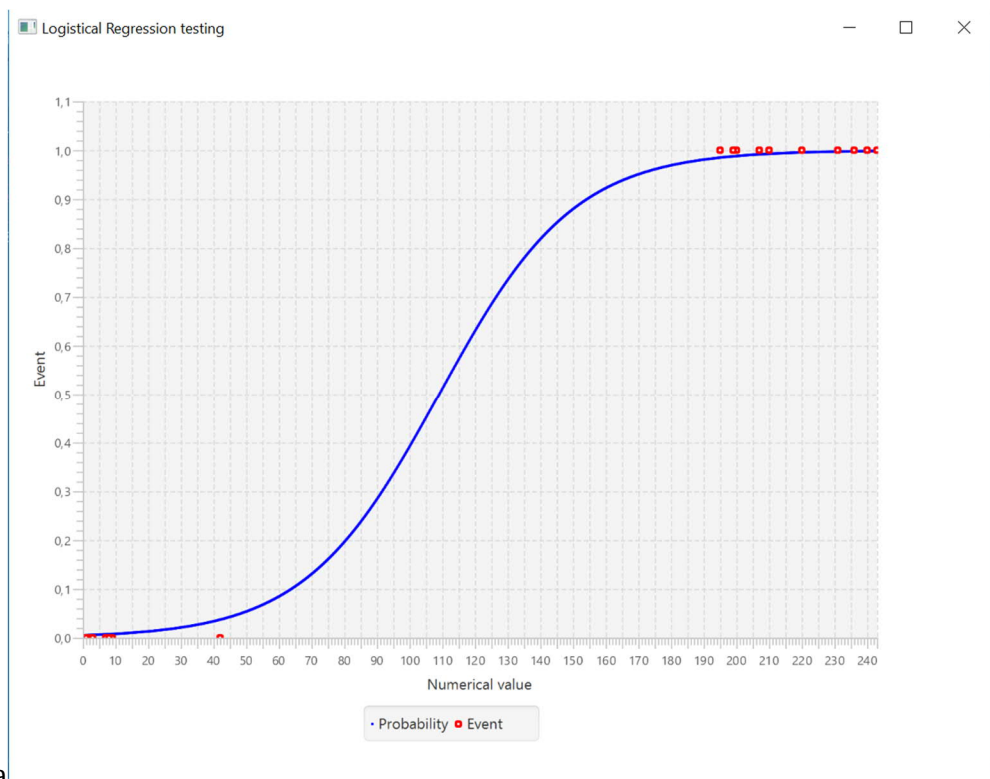


Projektin kuvaaja

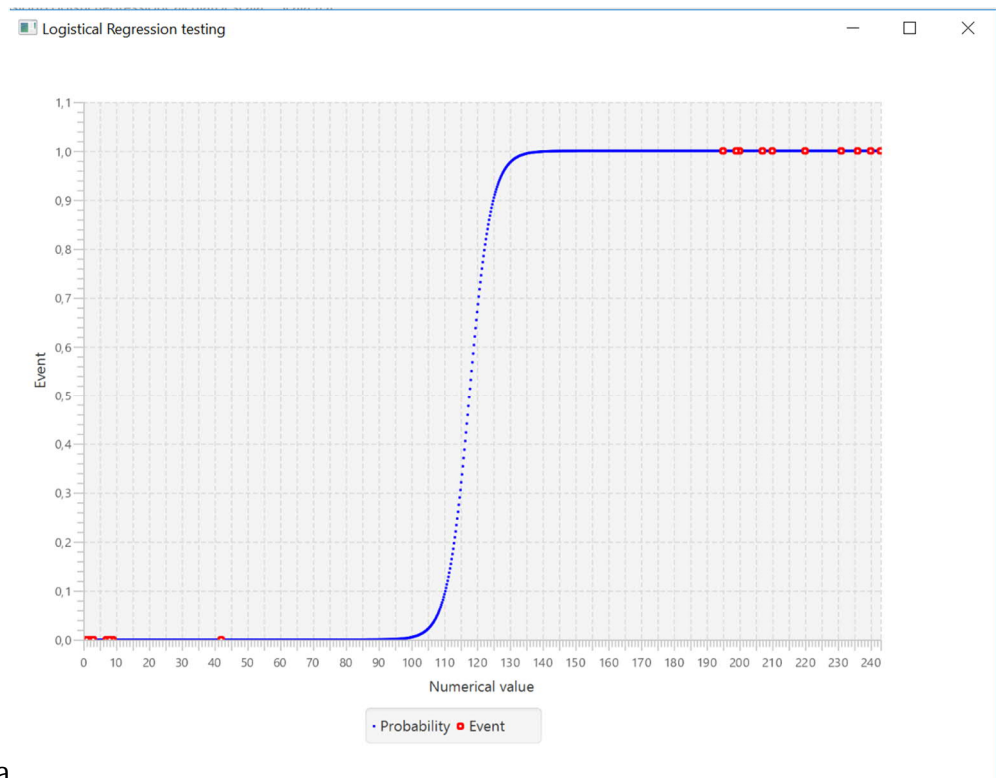


RStudion kertoimien kuvaaja

Testit näyttivät kuvaajien arvojen poikkeavan toisistaan luokkaa  $1E-6$ .



Projektin kuvaaja



RStuudioon kuvaaja

Tässä tapauksessa RStuudioon kertomien kuvaaja asettuu dataan varsin huonosti.

## Gradient descent -algoritmi

Gradient descent -algoritmissa käytetään iteraatioiden ylärajana  $1E6$ , joten logistisen käyrän piirto voi kestää koneesta riippuen. Aina ylärajaan ei kuitenkaan päästä, sillä kahden vierekkäisen pisteen ero on riittävän pientä iteroinnin lopettamiseen. Kuitenkin, koska logistinen regressio toteutettiin projektiin vasta viimeisten viikkojen aikana niin ei aikaa optimoinnille ollut. Käytännön testaukset ovat osoittaneet nykyisen iteraatiolukumäärän tuottavan hyviä kuvaajia, vaikka tätäkin voitaisiin optimoida.

## Erikoismerkit

Ohjelmaa on myös testattu kaikilla Excelin tarjoamilla CSV:n tyypeillä eikä toistaiseksi ole tullut vastaan mitään vikoja. Kuitenkin muuttujien nimissä merkit kuten ä, å tai ö jätetään usein pois. Tämä johtuu Parser:ssa käytettävästä Codec:n arvosta UTF-8.

```
implicit val codec = Codec("UTF-8")
```

```
codec.onMalformedInput(CodingErrorAction.REPLACE)
```

```
codec.onUnmappableCharacter(CodingErrorAction.REPLACE)
```

```
codec.onMalformedInput(CodingErrorAction.IGNORE)
```

## Polynominen regressio

Matriisipohjaisista regressiomalleista etenkin polynomista rajoittavat tilanteesta riippuen suuret luvut. Jos regressiomallin polynomin korkein aste on esimerkiksi 10, niin se voi saada jo arvoilla  $1 < x < 20$  suuria arvoja. Suurien arvojen, sekä käytettävän EJML-kirjaston sisäisestä toiminnasta johtuen epätarkkuus polynomin kertoimissa voi kasvaa hyvinkin merkittävästi. Testit ovat näyttäneet, että epätarkkuus korkeimpien asteiden kertoimissa on pientä tai olematonta, mutta näitä pienemmissä se on huomattavaa. Tämän takia korkeilla polynomin asteilla käyttäjää suositellaan skaalaamaan selittävän muuttujan ( $x$ ) arvoja reilusti pienemmäksi. Gradient descent -algoritmin käyttö matriisien sijasta voi ratkaista tämän ongelman, koska kyseisissä algoritmissa pyritään pääsemään kustannusfunktion (Cost function) globaaliin minimiin iteroimalla kertoimien osittaisderivaattoja. Matriisioperaatioissa suoritetaan matriisin koosta riippuen satoja tai tuhansia kertolaskuja kertoimien selvittämiseksi, jolloin epätarkkuus, suurilla arvoilla, kasvaa hyvinkin nopeasti.

Täsmällisen arvoalueen määrittäminen dokumentaatiossa viitatusalle "suurelle arvolle" ei ollut aikataulun takia mahdollista. Liian suuri arvo riippuu käytettävän polynomin korkeimmasta asteesta, polynomin yleisestä muodosta (kumoavatko termit toisiaan), sekä itse käytettävien arvojen kokoluokasta.

Lukujen epätarkkuudesta johtuen takia ohjelma ei takaa, että esimerkiksi tilanteessa, jossa mallinnettava funktio on  $f(x) = x^2$ , pystyisi ohjelma antamaan vakioksi, sekä ensimmäisen asteen kertoimeksi nolla. Riippuen annetusta datasta kertoimet voivat olla äärimmäisen lähellä nollaa, mutta eivät kuitenkaan täsmällisesti sitä ole.

Jos ohjelmassa luodaan korkea-asteinen polynomi, niin sen tietoja tarkastettaessa voi käydä niin, että kaikki asteet eivät mahdu ruudulle. Tämän voi korjata esimerkiksi niin, että PolynomialGraph:n toString-metodin palautusarvo tallennetaan johonkin rullattavaan listaan, jolloin kaikki graafin tiedot näkyvät.

## Suoritus aika

Ohjelmassa ei myöskään ole rinnakkaisuutta, joten hetkellistä käyttöjärjestelmän jäätymistä voi esiintyä. Yksi tapa toteuttaa rinnakkaisuus voisi olla, että taustalaskenta sijoitetaan omaan säikeeseensä ja käyttöliittymä omaansa. Tällöin käyttäjä voi käyttää graafista käyttöliittymää, vaikka taustalaskenta olisikin erityisen suurta.

## Logistisen regression yleisyys

Logistisessa regressiossa voi selittäviä muuttujia olla vain yksi, vaikka käytännössä niitä on useampia. Tämä ohjelmiston lisäyksen toteutus on teoriassa hyvin suoraviivainen: kahden muuttujan sijasta gradient descent -algoritmia iteroidaan kaikkien tuntemattomien kertoimien kesken. Kuitenkin käytännössä saadusta datasta kaikki selittävät muuttujat eivät ole yhtä arvokkaita. Ja monesti datan merkityksellinen tulkinta riippuu kontekstista. Joten, jotta saadut tulokset olisivat järkeviä pitäisi itse algoritmiin lisätä ylimääräinen vakinaistamistermi (regularization term), jolla eri kertoimia voidaan painottaa, sekä itse ohjelmaa muokata enemmän tilastollisten ohjelmistojen, kuten R, suuntaan.

## Ulkoisten ohjelmistojen XML-tiedostot

Kaupallisten ohjelmistojen, kuten Excel, tallentamia XML-tiedostoja ei todennäköisesti voida lukea, koska kyseinen ohjelmisto tallentaa XML-tiedoston omien spesifikaatioidensa mukaisesti. Näin ollen, jos käyttäjä haluaa välttämättä ladata juuri Excelin tallentamia XML-tiedostoja, pitää hänen todennäköisesti muokata ne tämän dokumentaation ohjeiden mukaiseksi. Tästä johtuen ohjelmalle annettavat XML-tiedostot pitää todennäköisesti tuottaa käsin.

## Kuvaajan yhtenäisyys

Jos ohjelmaan ladattava data mallintaa kuvaajaa, jonka derivaatta annetulla välillä on jonkin pisteen  $x$  lähistöllä hyvin jyrkkä, ei piirrettävä kuvaaja välttämättä ole yhtenäinen. Syy tähän on se, että piirrettäviä pisteitä laskettaessa, valitaan kahden vierekkäisen  $x$ -arvon etäisyys  $dx$  seuraavasti.

```
val dx = (graph.maximumXDataPoint - graph.minimumXDataPoint) / scene.width.get
```

Kun  $x$ -arvojen väli jaetaan ruudun sen hetkisellevä leveydellä, saadaan jokaista leveyspikseliä vastaava  $x$ -arvo. Näin piirrettävä kuvaaja skaalautuu pääasiassa hyvin ruudun leveyden vaihdellessa. Ongelmaksi muodostuvat tilanteet, joissa kuvaajan derivaatta on suuri, sillä silloin ei välttämättä lasketa riittävästi pisteitä kuvaajalle, jolloin siitä tulee katkonainen. Yksi ratkaisu voisi olla, että laskettaessa pisteitä  $f(x)$  ja  $f(x + dx)$  tarkistetaan, että onko

$dy = |f(x + dx) - f(x)| > dx$  ja jos on, niin asetetaan, että  $dx = dy$ . Tällöin ongelmaksi muodostuu se, että ilman muita apuvälineitä ei voida etukäteen tietää missä kohdissa kuvaajan derivaatta on hyvin suuri. Tällöin laskenta hidastuu, koska vertailua joudutaan tekemään koko ajan ja uuden  $dx$  arvon löytyessä joudutaan arvot ottamaan talteen, prosessi pysäyttämään ja jatkamaan siitä edellisestä lopetuskohdasta uudella  $dx$  arvolla. Jos piirrettävän kuvaajan halutaan ehdottomasti näyttävän jatkuvalta kaikilla arvoilla, on ainoa mahdollinen tapa käyttää jotakin ulkoista kirjastoa funktion toisen derivaatan löytämiseen, laskea tämän arvot annetulla välillä ja tämän perusteella muodostaa käytettävä  $dx$ . Tämä, uuden kirjaston käyttöönottamisen lisäksi, tekisi laskennasta hitaampaa, koska tarkistuksia pitäisi tehdä enemmän, etenkin käyttäjän muuttaessa minimi ja maksimi x-arvoja.

## Kolme parasta ja kolme heikointa kohtaa

### Kolme parasta

1. Käyttöliittymä. Olen mielestäni saanut käyttöliittymän hyvin intuitiiviseksi, sekä helppokäyttöiseksi. Uusien graafien luonti on hyvin suoraviivaista ja graafien lataus tai niiden tietojen tarkempi tutkiminen onnistuu vaivattomasti
2. Logistinen regressio. Vaikka kohdassa "Ohjelman puutteet ja viat" logistinen regressio nousikin esille, on se silti yksi tämän ohjelman kruununjalokivistä. Datasta, jossa on vain yksi selittävä muuttuja, piirretyt kuvaajat näyttävät varsin järkeviltä.
3. Matriisien käyttö polynomisen, sekä lineaarisen regression toteutuksessa. Matriisien käytön avulla on polynomisen, sekä lineaarisen regressio mahdollista esittää hyvin kompaktisti. Etenkin lineaarinen regressio toimii erityisen hyvin matriisiesityksen avulla.

### Kolme heikointa

1. Gradient descent -algoritmi, nykytilassaan. Koska gradient descent -algoritmi tuli ohjelmaan mukaan vasta viimeisillä viikoilla, ei sen koko potentiaalia voitu hyödyntää työssä. Kyseistä algoritmia pystyy käyttämään myös lineaarisen, sekä polynomisen kertoimien etsimiseen matriisien sijasta, mutta kertoimet löytyvät

lähteiden mukaan nopeammin. Koska ylimääräistä aikaa algoritmin optimisoinnille ei ollut, niin nykytilassaan logistisen regression laskemiseen kulutetaan turhaa aikaa.

2. GUI-olion rakenne paikoittain. GUI:sta löytyy kohtia, joiden toteutus on paikoittain toisteista tai heikohkoa. Esimerkki tästä on GUI:n graafin muokkaus kohdassa `modifyGraph.onAction`, missä vaihtoehtoinen toteutus olisi funktio, mikä ottaa valtavan määrän parametreja graafin minimi tai maksimin muokkausta varten. Toinen heikko kohta on GUI:n apuolio `GUIHelperMethods`:n metodi `getGraphOrder`. Kyseinen metodi vaatii, että sille annetaan jokin sallittu kokonaisluku ennen, kuin kyselyikkuna katoaa. Tämän voi korjata ohjelman rakenteen muutoksella. Tosin silloin tarvittaisiin ylimääräinen kytkin (flag) tai try-catch -rakenne tarkastamaan onko graafia luotu tilanteissa, joissa lähdekoodin seuraavat osat riippuvat graafin olemassaolosta.
3. Ei automaattista arvojen skaalausta polynomisessa regressiossa. Jos polynomisessa regressiossa olisi mukana automaattinen arvojen skaalaus, niin laskettavien kertoimien tarkkuus olisi nykyistä parempi. Nykyisillään korkeimpien asteiden kertoimet ovat hyvinkin tarkkoja, mutta koska korkea-asteisten polynomien arvot kasvavat nopeasti, alkaa tulomuotoisissa matriisioperaatioissa esiintymään huomattavaa epätarkkuutta pienempien asteiden kertoimissa.

## Poikkeamat suunnitelmasta, toteutunut työjärjestys ja aikataulu

Poikkesin aikataulusta merkittävästi projektin aikana, eikä suunnitelma vastannut todellisuutta.

11 - 17.2 Sain valmiiksi matriisipohjaisen algoritmin polynomiselle regressiolla, luokat polynomista regressiota varten, sekä alkeellisen käyttöliittymän.

17.2-3.3 Aloitin tiedostosta lukemisen kehittämisen, sekä työstin käyttöliittymää lisää. Graafeja pystyy nimeämään, sekä tallentamaan ajon aikana.



3.3–17.3 Paransin tiedostosta lukemista lisää, paransin käyttöliittymän reagoitua poikkeuksiin, toteutin GeneralRegressionModel-piirreluokan, alkeellisen apuikkunan käyttäjälle, sekä refaktoroin lähdekoodia toteuttamalla GUIHelperMethods-olion.

17.3-31.3 Paransin XML-tiedostojen lukemista, sekä vähensin toisteisuutta käyttämällä vain yhtä poikkeusluokkaa.

31.3-14.4 Tein selkeän eron PolynomialGraph, sekä LinearGraph:n välille: LinearGraph perii PolynomialGraph:n, mutta sille on valittuna aste jo valmiiksi. Tein myös kuvaajan piirrosta skaalautuvamman siten, että x-arvojen askelpituus  $dx$  riippuu ruudun leveydestä. Logistisen regression työstämisen. Toteutin myös GeneralGraph-piirreluokan, jonka avulla ohjelma on dynaamisempi.

14-24.4 Toteutin logistisen regression loppuun, sekä viimeistelin lähdekoodia. Paransin myös ohjelman poikkeuksen tunnistusta liian isojen arvojen osalta.

## Kokonaisarvio lopputuloksesta

Itsearvio lopputuloksesta on erinomainen. Ohjelma on intuitiivinen, sekä helppokäyttöinen. Se esittää mallinnettavan datan selkeästi. Ohjelmassa on alkuperäisen kahden regressiomallin (yksinkertainen lineaarinen regressio, sekä oma valinta) sijasta kolme regressiomalli: lineaarinen regressio, polynominen regressio, sekä logistinen regressio, joista viimeisen on ohjelmassa ehdottomasti hienoin.

Ohjelmassa ei ole merkittäviä puutteita, vaikka sitä voisi hioa monesta paikasta. Erityisesti laskennan (matriisialgoritmi, gradient descent) osalta on varmasti monia asioita, joissa suoritusaikaa ja tarkkuutta voi parantaa. Käyttöliittymää pystyy myös parantamaan sen ulkoasun, sekä olemuksen osalta. Myöskin GUI:sta saisi varmasti ns. kompaktimman, sen nykyisen olemuksen sijasta. Korkeamman asteen funktiolla saisi lyhennettyä monta eri kohtaa, esimerkiksi modifyGraph:n osalta.

Haluttaessa, graafista ja alkuperäisestä datasta voitaisiin kertoa käyttäjälle enemmän tietoa. Esimerkiksi mallin kasvunopeuden ilmoittaminen eri väleillä on yksi asia, jolla ohjelmaa voi parantaa.

Jos aloittaisin projektin nyt alusta, käyttäisin enemmän aikaa gradient descent -algoritmin toteutukseen. Kyseisen algoritmin avulla ohjelman laskenta-aika, sekä

kirjasto riippuvuus pienenisi. Käyttäisin myös piirreluokkia alusta asti enemmän, koska ne tekevät myöhemmästä refaktoroinnista helpompaa.

Nykyisillään ohjelma on hyvin laajennettavissa. Uuden regressiomallin tekeminen vaatii vain GeneralGraph-piirreluokan laajentamista, uuden valintanapin lisäämistä graafiseen käyttöliittymään, sekä uuden case:n lisäämistä GUIHelperMethods:n formGraph-metodiin. Halutessaan graafiseen käyttöliittymään voi myös helposti lisätä uusia valikoita ja nappeja.

## Viitteet

Käytetyt kirjastot

ScalaFX: <http://www.scalafx.org/>

ScalaFX API: <http://www.scalafx.org/api/8.0/#package>

ScalaXML: <https://github.com/scala/scala-xml>

EJML: [http://ejml.org/wiki/index.php?title=Main\\_Page](http://ejml.org/wiki/index.php?title=Main_Page)

EJML SimpleMatrix: <http://ejml.org/wiki/index.php?title=SimpleMatrix>

JUnit: <https://github.com/junit-team/junit4/wiki/Download-and-Install>

Käytetyt lähteet

Logistinen regressio, sekä gradient descent -algoritmi

- [https://en.wikipedia.org/wiki/Logistic\\_regression](https://en.wikipedia.org/wiki/Logistic_regression)
- <https://www.internalpointers.com/post/linear-regression-one-variable>
- <https://www.internalpointers.com/post/gradient-descent-function>
- <https://www.internalpointers.com/post/multivariate-linear-regression>
- <https://www.internalpointers.com/post/gradient-descent-action>
- <https://www.internalpointers.com/post/optimize-gradient-descent-algorithm>
- <https://www.internalpointers.com/post/introduction-classification-and-logistic-regression>
- <https://www.internalpointers.com/post/cost-function-logistic-regression>
- <https://www.internalpointers.com/post/problem-overfitting-machine-learning-algorithms>

Lineaarinen regressio, sekä sen matriisiesitys

- [https://en.wikipedia.org/wiki/Linear\\_regression](https://en.wikipedia.org/wiki/Linear_regression)
- <https://newonlinecourses.science.psu.edu/stat501/node/382/>

Polynominen regressio

- [https://en.wikipedia.org/wiki/Polynomial\\_regression](https://en.wikipedia.org/wiki/Polynomial_regression)

Testaukseen käytetty logistisen regression sivusto

- <http://statpages.info/logistic.html>

Testaukseen käytetty lineaarisen regression sivusto

- <https://www.socscistatistics.com/tests/regression/default.aspx>

ScalaFX:n käyttö, sekä käyttöliittymän toteutukseen liittyvät kysymykset

- <https://stackoverflow.com/questions/55327982/how-to-efficiently-plot-continuous-functions-in-scalafx-javafx>
- <https://stackoverflow.com/questions/54607234/changing-the-size-and-symbol-of-scatter-chart-plot-points-in-scalafx>
- <https://stackoverflow.com/questions/55078589/cant-move-textflow-inside-a-new-scene>
- Mark Lewis Object-Orientation, Abstraction and Data Structures Using Scala
  - <https://www.youtube.com/watch?v=85bHg5AipvU&list=PLLMXbkbDbVt8JLumqKj-3BIHmEXPIfR42&index=1>; Jaksot 1 – 18, 67- 74, 76 – 78, 82 – 84.