2016

Numeerisen data visualisointikirjasto

OHJELMOINNIN PERUSKURSSI Y2: PROJEKTITYÖ

TUOMAS TAKKO (BIO 2.VSK) 427612

Tuomas.takko@aalto.fi

13.05.2016

Sisällys

1. Yleiskuvaus

Projektin päämääränä oli tuottaa ohjelmakirjasto, joka visualisoi numeerista data graafisesti. Ohjelmakirjasto pystyy visualisoimaan viivadiagrammin lisäksi pylväs- ja piirakka diagrammin. Data luetaan käyttäjän antamasta tiedostosta ja se muotoillaan ohjelmakirjaston tietorakenteiden mukaiseksi. Luetun datan perusteella lasketaan akselien paikka ja resoluutio, sekä kuvaajaan lisättävän lukemista helpottavan ruudukon ruutujen määrä. Datan mukaan ohjelma generoi kullekin kuvaajalle tai selitteelle oman yksilöllisen värin, jonka mukaan muodostetaan selitteet. Sekä selitteet että ruudukko on molemmat mahdollista poistaa näkyvistä.

Kun ohjelma tulostaa pyQt4:n avulla datan visuaalisesti ikkunaan, voi käyttäjä vaikuttaa näkymään ikkunan yläpalkin valintojen avulla. Käyttäjä voi vaihtaa luettavan tiedoston, jolloin ohjelma lukee tiedoston datan ja kuvaaja piirtyy uudestaan. Mahdollista on myös kytkeä ruudukko ja selitteet pois päältä, mikäli siihen on tarvetta. Ruudukon resoluutio voidaan vaihtaa, jolloin sekä akselien selitteet, että ruudukon ruudut muuttuvat. Akselien selitteet on myös mahdollista vaihtaa datan luettavuuden selkeyttämiseksi. Kuvaajan voi tallentaa kuvatiedostoksi.

Muutoksia alkuperäiseen suunnitelmaan tehtiin zoomauksen poisjättämisen osalta. Työ on toteutettu vaikeimmalla vaikeusasteella.

2. Käyttöohje

Ohjelmakirjaston käyttäminen on mahdollista niin itsenäisesti, kuin myös toisen ohjelman yhteydessä modulaarisesti. Pääasiallisesti ohjelmaa on tarkoitettu käytettävän tiedostosta main.py, joka kutsuu uuden pyQt applikaation. Ajettaessa main.py kysyy käyttäjältä tämän tiedoston nimen, olettaen tiedoston olevan samassa tiedostosijainnissa, tai sitten tiedoston sijainnin tiedostopolkuna. Kun käyttäjä on antanut haluamansa tiedoston, lukee ohjelma sen file_IO.py tiedostossa. Datasta on nyt muodostettu ohjelman mukaiset tietorakenteet, jotka siirtyvät GUI.py tiedoston käyttöön. GUI.py avaa ikkunan, johon piirtyy käyttäjän antaman datan mukainen kuvaaja.

Ohjelman piirtämässä ikkunassa käyttäjä voi tehdä vielä omia valintojaan. Ikkuna skaalautuu sisältöineen ja käyttäjä voi venyttää sitä haluamansa kokoiseksi. Ikkunan yläosassa on palkki, jossa on komentoja. Käyttäjä voi vaihtaa tiedostoa (”file”), kuvaajan ruudukon näkyvyyttä (”Grid”), kuvaajan ruudukon resoluutiota (”Grid res”), selitteiden näkyvyyttä (Legend”), tallentaa kuvaajan (”Save Graph”), vaihtaa akselien nimiä (”Labels”) ja sulkea ikkunan (”Exit”).

Ohjelman käyttö: step-by-step

1. Tiedoston antaminen ohjelmalle

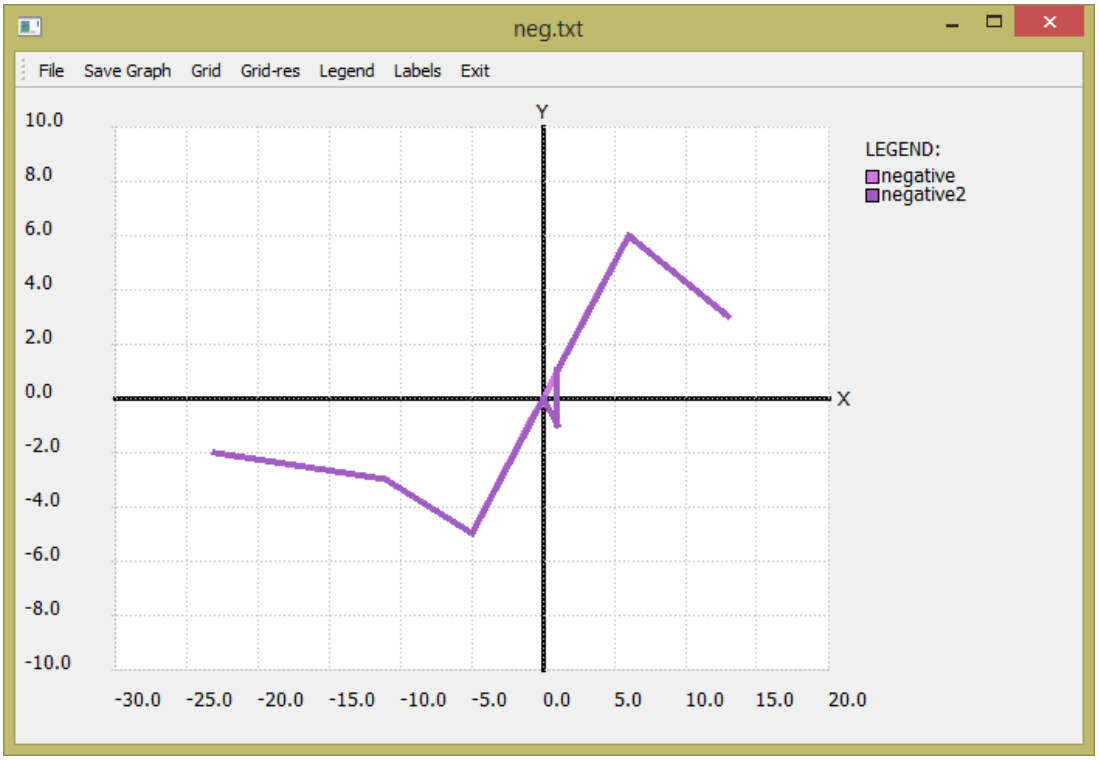
Ajettaessa main.py kysyy käyttäjältä luettavan tiedoston nimen:

Give filename:neg.txt

2. Ohjelman graafinen ikkuna

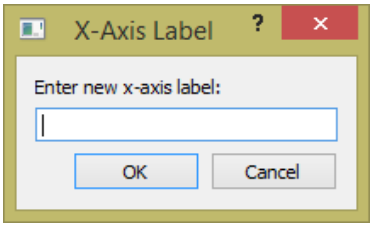
Ikkunan yläpalkista käyttäjä voi vaihtaa tiedostoa (”file”), kuvaajan ruudukon näkyvyyttä (”Grid”), kuvaajan ruudukon resoluutiota (”Grid res”), selitteiden näkyvyyttä (Legend”), tallentaa kuvaajan (”Save Graph”), vaihtaa akselien nimiä (”Labels”) ja sulkea ikkunan (”Exit”).

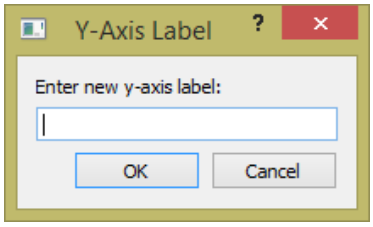
Akselien numeroinnit näkyvät vasemmalla sivulla ja kuvaajan alapuolella. Akselien nimet näkyvät akselien päissä. Selitteet piirtyvät oikeaan reunaan kukin oman värinsä viereen. Ikkunan nimi on käyttäjän antaman tiedoston nimi.



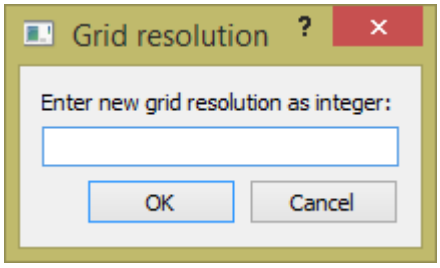
3. Valintaikkunat

- Valinta ”Labels” avaa kyseiset ikkunat, joihin käyttäjä voi syöttää uudet nimet akseleille.

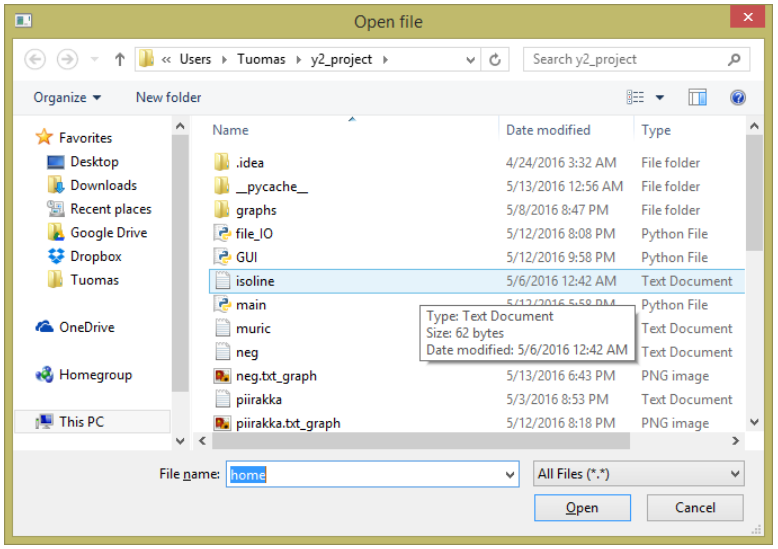




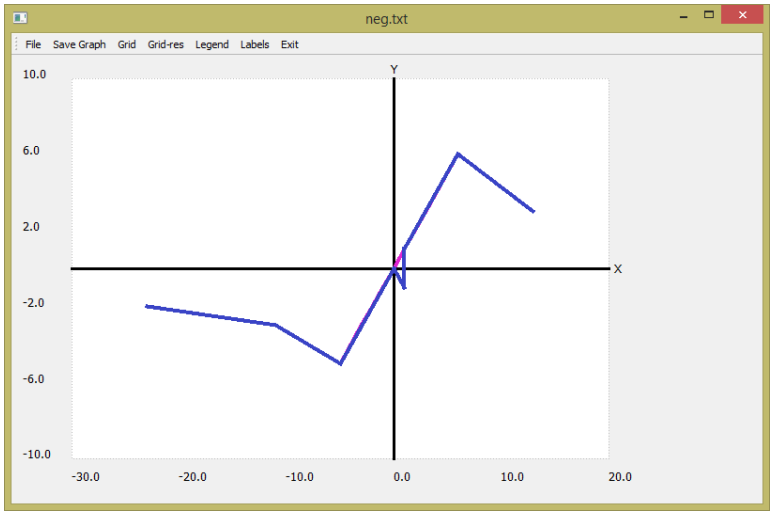
- Valinta ”Grid-res” avaa kyseisen ikkunan, johon käyttäjä voi syöttää uuden ruudukon resoluutioarvon.



- Valinta ”File” avaa kyseisen ikkunan, josta käyttäjä voi valita uuden datatiedoston, jonka ohjelma lukee.



- Kun käyttäjä painaa ”Grid” ja ”Legend”, poistuvat ruudukko ja selitteet näkymästä.



3. Ohjelman rakenne

Ohjelma rakentuu kolmen tiedoston ympärille. Nämä tiedostot ovat main.py, file_IO.py ja GUI.py. Main.py on itsenäisen ajettavuuden kannalta olennainen tiedosto, mutta kaikki toiminnallisuus tapahtuu muissa tiedostoissa. Main.py luo pyQt applikaation ja antaa tälle ensimmäisen tiedoston. Tämän toiminnon voi toteuttaa myös komentoriviltä tai toisesta ohjelmasta.

File_IO

File_IO tunnistaa halutun diagrammin tyypin ja muodostaa sen mukaisesti datan. Data luetaan rivi kerrallaan tyypin mukaisella funktiolla. Struct_line-funktio lukee viivadiagrammin, struct_pie piirakkadiagrammin ja struct_column pylväsdiagrammin. Jokainen näistä funktioista saa parametrina luettavan tiedoston ja Numeeriset arvot tallennetaan jokainen omaan Point-luokkaansa. Point-olio sisältää arvot x ja y. X-arvo voi olla joko numeerinen (viivadiagrammi) tai merkkijono (piirakka ja pylväsdiagrammit). Y-arvon tulee olla aina numeerinen. Jokainen Point-olio lisätään luomisensa jälkeen Graph_data-olion point-array attribuuttiin. Point-array on pythonin oma listarakenne, joka sisältää point-olioiden muistiosoitteet. Viivadiagrammin tapauksessa näitä Graph_data-olioita on yksi jokaiselle ”viivalle”. Jokainen Graph_data olio lisätään Graph_Set-olion graphlist-attribuuttiin. Tiedoston lukemisen jälkeen Graph_Set lisätään Graph-olion graph-array attribuuttiin.

GUI

Tässä vaiheessa File_IO-luokka on toteuttanut edellämainitun tietorakenteen, joka siirtyy GUI.py tiedoston GUI-luokan käyttöön. GUI-luokka toteuttaa kaiken visuaalisen toiminnallisuuden. GUI luokka toteuttaa paintEventin, joka piirtää oikeanlaisen kuvaajan.

GUI luokassa on seuraavat funktiot:

- initUI: Tallentaa luokan omat attribuutit filename, palette, data, name, gridDesc, x_marg, y_marg, grid ja legend:
    - Filename on tiedoston nimi
    - Palette on lista pyQt:n väriobjekteista, jotka generoidaan RGB tekniikalla satunnaisina arvoina välillä 0-255. Nämä värit tallennetaan listaan myöhempää käyttöä varten.
    - Data on File_IO:n muodostama datarakenne (Graph-olio)
    - GridDesc on ruudukon resoluutio, eli sen pysty ja vaakaviivojen määrä. Oletuksena 5.
    - X_marg on skaalautuva pituusarvo. Lasketaan ikkunan leveys jaettuna kymmenellä.
    - Y_marg on skaalautuva pituusarvo. Lasketaan ikkunan korkeus jaettuna kymmenellä.
    - Grid on boolean-muuttuja, jonka oletusarvo on True. Jos arvo on True, ruudukko piirretään ja jos arvo on False, ruudukkoa ei piirretä.

    - Legend on boolean-muuttuja, jonka oletusarvo on True. Jos arvo on True, selitteet piirretään, jos arvo on False, selitteitä ei piirretä.
- paintEvent: Piirtää oikeantyyppisen kuvaajan self.data:n mukaisesti muita funktioita apunaan käyttäen:
    - Viivadiagrammille kutsutaan drawBackground (piirtää taustan), drawLines (piirtää akselit ja niden selitteet), drawGrid (piirtää ruudukon), showLegend (piirtää selitteet), sekä jokaiselle ”viivalle” kutsutaan drawGraph (piirtää viivan). Luokalle luodaan attribuutti lgnd, joka on lista, jonka alkioina ovat selite ja väri yhdistelmät.
    - Piirakkadiagrammia varten luodaan muuttujalista lgnd (selitteet). Koska piirakkadiagrammin piirtäminen on yksinkertainen operaatio, lasketaan tässä kohtaa datan arvojen summa, jonka mukaan lasketaan kunkin alueen osuus piirakasta.
    - Pylväsdiagrammille kutsutaan drawBackground (piirtää taustan) ja countColumns (laskee akselien arvon ja skaalauksen, sekä piirtää ne). Uusia muuttujia luodaan lgnd-lista selitteille ja väreille. Tässä vaiheessa piirretään myös jokainen pylväs.
- countColumns: Laskee y-akselin korkeuden pylväsdiagrammille. Suurin arvo pyöristetään seuraavaan kymmenlukuun. Y-akselin selitteiden arvot lasketaan y-akselin pituudella jaettuna gridDesc:n arvolla ja tallennetaan ycrd listaan. Akselit ja niiden selitteet piirretään.
- drawLines: Piirtää viivadiagrammille akselit ja niiden selitteet. Hyödyntää akselien ja selitteiden laskemiseen funktiota countAxis. Tallentaa luokan muuttujat xlbl ja ylbl, jotka ovat luetun datan mukaiset akselien selitteet.
- drawGraph: Käyttää countAxis funktiossa luotua axisArr-listaa, jossa on x ja y akselian pyöristetyt ääriarvot. Tämän avulla piirretään jokaisen datapisteen välille viiva.
- drawGrid: Kutsuu countAxis-funktiota, jonka perusteella piirtää ruudukon oikean resoluution (gridDesc), mukaan.
- drawBackground: Piirtää valkoisen taustan viiva- ja pylväsdiagrammille. Taustan koko on viivadiagrammin tapauksessa pienempi, jotta selitteet mahtuvat.
- countAxis: Kuvaajan oikeaoppisen piirtämisen kannalta olennaisin funktio. Aluksi funktio määrittää akseleiden maksimiarvot lukemalla kaikkien samaan kuvaan piirrettävien graafien x- ja y-arvojen minimit ja maksimit. Näistä maksimiarvoista funktio muodostaa axisArr-listan pyöristämällä maksimiarvot seuraavaan kymmenlukuun ja minimiarvot pyöristämällä edelliseen kymmenlukuun tai nollaan. Funktio myös muodostaa akselien numeroinnin coords-listaan jakamalla pyöristettyjen maksimi ja minimiarvojen erotuksen gridDesc-muuttujan mukaisella resoluutiolla.
- Color: Muodostaa satunnaisen QColor olion laskemalla satunnaisen RGB-värin. RGB-värit muodostetaan seuraavasti (0-255, 0-255, 0-255)

- gridIO: Kutsuttaessa muuttaa luokan grid-muuttujan toiseen boolean-arvoon. Jos grid on True, muuttuu se Falseksi ja toisinpäin. Näin voidaan ruudukko piilottaa tai näyttää.
- legIO: Kutsuttaessa muuttaa luokan legend-muuttujan toiseen boolean arvoon. Jos legend on True, muuttuu se Falseksi ja toisinpäin. Näin voidaan selitteet piilottaa tai näyttää.
- saveIMG: Hyödyntää pyQt4:n QtGui:n QPixmap:in ominaisuutta tallentaa ikkunan sisältö. Tallennettava alue on rajattu niin, ettei yläpalkki tallennu otettavaan kuvaan.
- showLegend: Piirtää piirakka- ja viivadiagrammeille selitteet oikeaan ylänurkkaan. Kunkin selitteen edessä on pieni laatikko, joka vastaa viivan tai sektorin väriä.
- fileSwap: Avaa QtGui:n tiedostonvalintaikkunan, josta käyttäjä voi valita uuden käsiteltävän tiedoston. Funktio ajaa GUI-luokan initUI-metodin uudestaan kyseiselle tiedostolle.
- Gridcount: Avaa pienen ikkunan joka pyytää käyttäjältä uuden integer-arvon ruudukon resoluutiolle. Kun käyttäjä painaa ”ok”, funktio päivittää näkymän.
- lblSwap: Avaa kaksi ikkunaa, joilla käyttäjä voi nimetä uudestaan akselit. Funktio päivittää ikkunan kun käyttäjä painaa ”ok”.
- menuBar: Muodostaa ikkunaan yläpalkin, jossa aiemmin kuvaillut komennot ”File”, ”Save Graph”, ”Grid-res”, ”Legend”, ”Labels” ja ”Exit” ovat. Yhdistää kunkin nappulan ja sen mukaisen funktion.

4. Algoritmit

Ohjelmassa tapahtuu kuvaajatyypistä riippuen erilaisia algoritmeja. Tiedostonluvussa käytetään lähinnä pythonin valmista sorted-funktiota datan x-akselin järjestämiseksi. Tyypillisiä algoritmeja graafisen kuvan muodostuksessa ovat kuvaajan viivojen ja akselien skaalaukset, sektorien koot ja pylväiden leveydet.

Akselien skaalaus

GUI-luokan funktio countAxis iteroi for-loopilla läpi jokaisen samaan ikkunaan piirrettävän datapisteen ja tallentaa näistä niiden minimi ja maksimiarvot. Nämä minimi ja maksimiarvot pyöristetään pythonin math-kirjaston ceil-funktiolla lähimpään kymmenlukuun tai nollaan. Alimmat arvot pyöristetään riippuen niiden suuruudesta. Mikäli arvot ovat negatiivisia, pyöristetään ne alaspäin negatiiviseen kymmenlukuun. Muuten alimmat arvot ovat oletuksena nolla. Akselit numeroidaan käyttämällä muuttujaa sclx ja scly, jotka muodostetaan alimpien arvojen mukaan. Arvot joko summataan tai vähennetään toisistaan, jotta niiden välinen etäisyys aksellilla saadaan muuttujaan sclx tai scly. Akselien numerointi tapahtuu jakamalla scl-arvot gridDesc:n resoluutioarvolla ja kertomalla aina kukin arvo sen osuudella. Mikäli alimmat arvot ovat negatiivisia, tulee akselien origo silti nollaan.

Viivadiagrammin piirtäminen

Viivadiagrammin piirtäminen niin että sen arvot ovat oikeat ja skaalautuvat oikein ikkunassa on olennainen osa oikeanlaisen diagrammin muodostamista. Viivan piirtäminen pyQt:ssa tapahtuu antamalla drawLine-komennolle viivan alkukoordinaatit (x-koordinaatti, y-koordinaatti) ja loppukoordinaatit. Oikean lopputuloksen aikaansaamiseksi pitää ensin hahmottaa akselien skaalaus. Akselit piirretään seuraavasti:

- X-akseli: ikkunan leveys – 2*x_marg-100, eli 80% ikkunan leveydestä, jonka lisäksi jätetään selitteitä varten 100 pikselin levyinen marginaali.
- Y-akseli: ikkunan korkeus – 2*y_marg, eli 80% ikkunan korkeudesta.

Jotta akselit saadaan suhteutettua kuvaajaan, jaetaan ne countAxis-funktiossa laskettujen kunkin akselin minimi ja maksimiarvojen välillä. Tällöin saadaan suhteelliset koordinaatit. Kukin koordinaattipiste saadaan siis kertomalla suhteellinen koordinaatti kuvaajan pisteen arvolla. Algoritmi on seuraavanlainen esimerkiksi y-koordinaatille:

$$korkeus - datapiste + abs(minimiarvo * (korkeus - 2 * y_{marg})/(y_{max} - y_{\min}\quad)$$

Näin saadaan laskettua alkupisteiden arvo. Loppupisteen arvo on datapisteiden jonon seuraavan pisteen arvo, joka lasketaan samalla kaavalla. Kaava on yleisesti pätevä ja toimii myös arvojen ollessa negatiivisia.

Piirakkadiagramin sektorien laskeminen

Oikeanlaisen piirakkadiagrammin piirtämiseksi ohjelma laskee ensin kaikkien datapisteiden summan, josta lasketaan seuraavasti:

$$round(datapiste/(summa * 16 * 360)$$

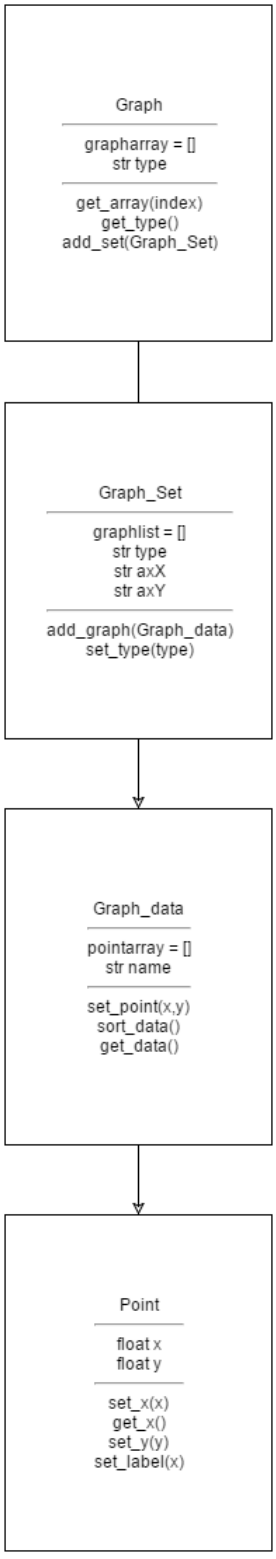
Näin saadaan kunkin datapisteen kulman suuruus. PyQt:n drawPie-komento ottaa vastaan alkukulman ja kulman suuruuden, joten aloituskulmaan lisätään kullakin iteraatiokierroksella piirrettävän kulman suuruus, jolloin seuraavan sektorin piirtäminen alkaa oikeasta kohdasta.

Pylväsdiagrammin algoritmit

Pylväsdiagrammissa lasketaan ensiksi kaikkien pylväiden leveys. Tämä lasketaan jakamalla käytössä oleva tila (ikkunan leveys*0,8) pylväiden lukumäärällä kerrottuna kahdella. Jokaisen pylvään väliin jää siis pylvään leveyden verran tilaa. Jokaisen pylvään korkeus lasketaan laskemalla suhteellinen koordinaatisto samalla tavalla kuin viivadiagrammin kanssa, mutta nyt vain y-akselille. Jokaisen pylvään arvo kerrotaan suhteellisella koordinaatilla ja vähennetään ikkunan korkeudesta, jolloin saadaan oikeankorkuiset pylväät.

5. Tietorakenteet

Tiedostonluvun suorittava luokka File_IO, luo ohjelman käyttöön tulevat tietorakenteet. Tietorakenteita on kuvailtu jo edellä. Pääpiirteissään tietorakenne on hyvin yksinkertainen, olioista ja dynaamisista listoista koostuva puu-rakenne. Ainoa ei-dynaaminen rakenne, jonka ohjelma tallentaa on kuvatiedosto mikäli käyttäjä sen tallentaa. Kaikki file_IO:n rakenteet ovat dynaamisia.



Yliluokka on Graph-olio, johon on tallennettu dynaamiseen listaan Graph_Set-olio, eli sen muistiosoite. Tämä tietorakenne luotiin, jotta ohjelmaan pystyttäisiin mahdollisesti lisäämään samaan Graph-rakenteeseen useita Graph_Set-rakenteita. Näin ei kuitenkaan tehty, koska siihen ei ollut tarvetta. Graph toimii siis yläluokkana ja pitää sisällään kaiken tiedostosta luetun datan.

Graph_Set-rakenne sisältää dynaamisen listan johon tulee jokaisen Graph_datan muistiosoite, jota kautta voidaan päästä käsiksi tämän rakenteen sisältämään pistelistaan. Tämä rakenne on olennainen osa viivadiagrammin piirtämistä, sillä jokaiselle graafille (”viivalle”) muodostetaan oma Graph_data-olio. Näin on helppoa päästä käsiksi useisiin eri graafeihin saman tietorakenteen kautta ilman kummempaa etsimistä. Graph_Set sisältää dynaamisen listan ja tyypin lisäksi myös viivadiagrammille olennaiset akselien nimet, joiden tulisi olla sama kaikille samaan kuvaajaan piirrettäville graafeille, koska eihän muuten voida tehdä järkevää vertailua tai havainnointia kuvaajasta.

Graph_data sisältää dynaamisen listan Piste-olioista, eli jokaisen pisteen muistiosoitteen. Näin on helppo päästä käsiksi jokaiseen pisteeseen, sekä pisteiden järjestäminen niin että arvoparit säilyvät onnistuu helposti. Dynaamisen listan kautta sort_array-funktio pystyy käyttämään pythonin sorted-funktiota listan järjestämiseen avaimella point.get_x.

Kokonaisuudessaan tietorakenne mahdollistaa datan siirrettävyyden tiedostosta toiseen ja sen helpon lukemisen. Dynaamiset listat eivät voi mennä keskenään sekaisin, dataparit pysyvät eheinä ja jokaisen kuvaajan muodostaminen onnistuu erikseen. Tämä mahdollistaa myös virheiden estämisen, sillä yksittäinen piste tai pistejono on helppo jättää piirtämisen ulkopuolelle. Myös yksittäisen pisteen tai jonon poistaminen on yksinkertaista. Tietorakenne on hyvin sovellettavissa ja laajennettavissa, mikäli käyttäjä niin haluaa.

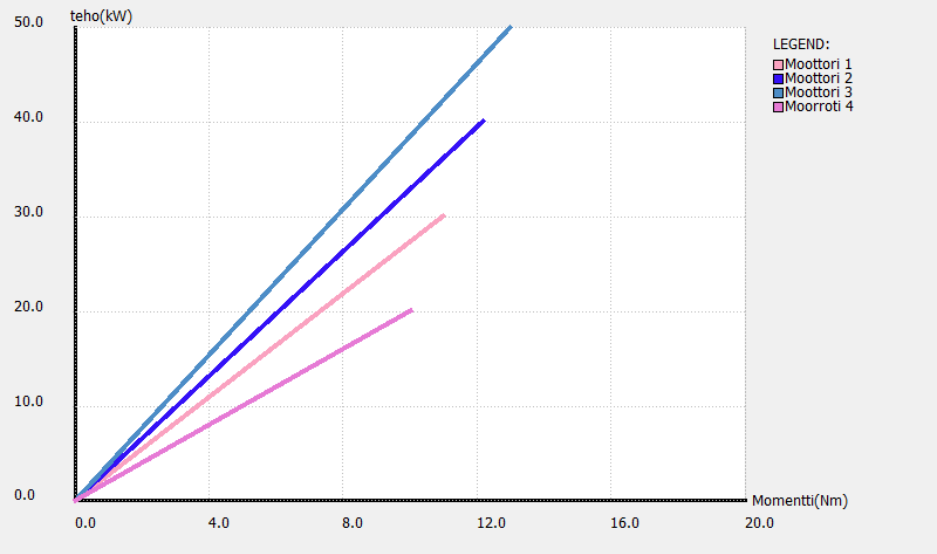
6. Tiedostot

Ohjelmisto lukee sen omalle syntaksille tehtyjä txt-tiedostoja, joiden lukeminen noudattaa seuraavaa kaavaa:

- Tyyppitunniste kullekkin kuvaajalle on muotoa ”#tyyppi”, eli viivadiagrammille ”#line”, piirakkadiagrammille ”#pie” ja pylväsdiagrammille ”#column”. Rajoitteena tiedostolle on yksi tyyppi, eli tiedosto ei voi sisältää sekä viiva- että pylväsdiagrammeja. Jos näin on, ohjelma lukee vain ensimmäisenä tulevan datan. Eri viivadiagrammit erotetaan toisistaan tyyppitunnisteella.
- Kunkin kuvaajan nimi annetaan tyyppitunnisteen jälkeen kaksoispisteellä erotettuna. Oletusarvo on ”Unknown”
- Akseleiden nimet annetaan muodossa ”##x:y”, jossa x kuvaa vaaka-akselin nimeä ja y pystyakselin nimeä. Nämä voidaan jälkikäteen vaihtaa ja oletuksena nimet ovat ”X” ja ”Y”.
- Viivadiagrammin data annetaan muodossa ”x-arvo:y-arvo”, mikäli nämä pisteet eivät ole numeerisia, ei pistettä lisätä kuvaajaan. Kukin arvopari on omalla rivillään.
- Pylväs ja piirakkadiagrammien data annetaan muodossa ”selite:arvo”. Kukin arvopari on omalla rivillään.
- Tyhjät rivit lasketaan tiedoston loppumiseksi.

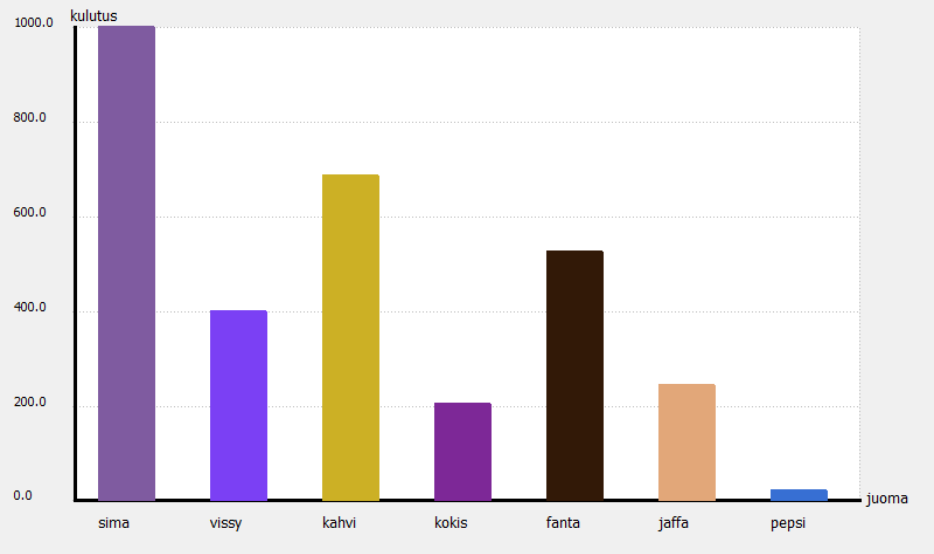
Esimerkki tiedoston syntaksista ja lopputuloksesta (viivadiagrammi):

```
#line:Moottori 1
##Momentti(Nm):teho(kW)
0:0
11:30
#line:Moottori 2
0:0
12.2:40
#line:Moottori 3
0:0
13:50
#line:Moorroti 4
0:0
10:20
```



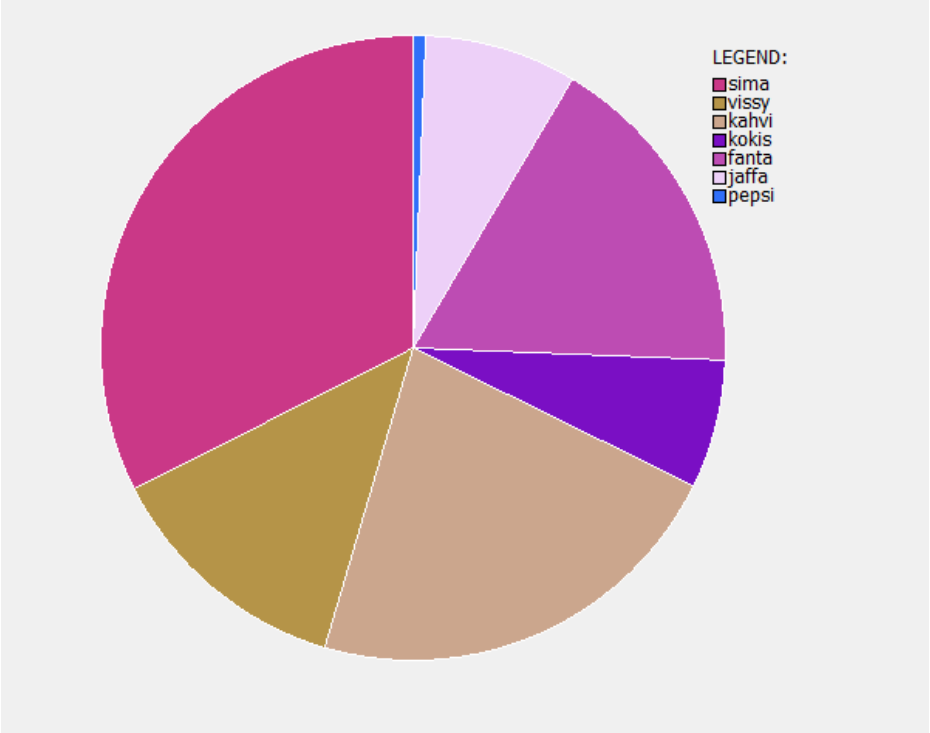
Esimerkki tiedoston syntaksista ja lopputuloksesta (pylväsdiagrammi):

```
#column:juomat
##juoma:kulutus
sima:1000
vissy:400
kahvi:687
kokis:205
fanta:527
jaffa:243
pepsi:20
```



Esimerkki tiedoston syntaksista ja lopputuloksesta (piirakkadiagrammi):

```
#pie:juomat
sima:1000
vissy:400
kahvi:687
kokis:205
fanta:527
jaffa:243
pepsi:20
```



7. Testaus

Yksikkötestauksessa painopiste on tiedostonlukemisen ja tietorakenteen muodostamisen suorittavan osan, file_IO:n, testaamisella. Testattavia asioita ovat tiedostonluku ja käsittely, sekä olioiden ja kuvan parametrien luominen. Graafisen puolen testaaminen tehtiin antamalla ohjelmalle erilaisia ääriarvoja ja arvioimalla visuaalisesti piirretystä lopputuloksesta ohjelman toiminnan laatu. Testitiedosto löytyy git-reposta tiedostosta test.py.

Testaaminen tehtiin syöttämällä erilaista dataa ohjelmaan ja tarkistamalla ohjelman toiminnan olevan tarkoituksenmukaista. Tiedoston lukeminen testattiin syöttämällä virheellisiä tiedostoja ohjelmaan. Näistä ohjelma odotetusti joko nosti virheilmoituksen ja keskeytti toimintansa, tai sitten ohitti viallisen datapisteen, mutta ilmoitti siitä.

Datan muotoilu tarkistettiin testaamalla että syötetty data on oikeassa formaatissa käyttämällä File_IO-luokan metodia is_number. Tämä metodi yrittää muuttaa annetun datan float-numeroksi, mutta muunnoksen epäonnistuessa se nostaa virheen. Jos tiedosto ei ole järjestetty, järjestää kirjasto x-akselin järjestykseen käyttämällä pythonin omaa sorted-toimintoa avaimena datapisteen get_x-metodi. Akselin järjestäminen testattiin syöttämällä ohjelmalle tiedosto virheellisellä x-akselilla. Akselien skaalaus testattiin syöttämällä erikoisia ääritapauksia sisältäviä arvoja ohjelmaan. Negatiivisten arvojen kohdalla ohjelma piirtää oikeaoppisesti koordinaattiakselien origon nollakohtaan.

File_IO:lle tehtiin seuraavat yksikötestit tiedostossa test.py:

- Test_line: Testattiin että tiedosto luetaan oikein muodostettaessa viivadiagrammia vertaamalla lopputulosta oikeaan tyyppiin, satunnaisen pisteen oikeaan arvoon, akseleiden nimiin ja graafien nimiin.
- Test_pie: Testattiin että tiedosto luetaan oikein muodostettaessa piirakkadiagrammia vertaamalla lopputulosta oikeaan tyyppiin, satunnaisen pisteen oikeaan arvoon ja graafien nimiin.
- Test_column: Testattiin että tiedosto luetaan oikein muodostettaessa pylväsdiagrammia vertaamalla lopputulosta oikeaan tyyppiin, satunnaisen pisteen oikeaan arvoon ja graafien nimiin.
- Test_sortbadlinechart: Testattiin että tiedostosta, jonka x-akseli on sekaisin, saadaan oikeanlainen kuvaaja ja x-akseli järjestettyä. Verrattiin lopputulosta oikeisiin pisteisiin oikeassa järjestyksessä.
- Test_badvalueslinechart: Testattiin että ohjelma nostaa oikeanlaisen virheilmoituksen kun ohjelmalle syötetään täysin vääränlaista dataa.

Graafisen muotoilun testaaminen tehtiin antamalla erilaisia ääriarvoja. Graafinen skaalaus ja tyylittely testattiin silmämääräisesti ja muokattiin tarpeen vaatiessa. Graafisen ikkunan toiminnot testattiin antamalla erilaisia syötearvoja ruudukon resoluutioksi ja erilaisia nimiä akseleille. Tiedostonvalinta ja kuvankaappaus testattiin kokeilemalla erilaisten tiedostojen avaamista ja niiden kuvien tallentamista.

8. Ohjelman puutteet ja tunnetut viat

Ohjelmassa on muutamia riskitekijöitä, jotka johtavat puutteisiin tai virhetilanteisiin. Oletuksena ohjelman toiminnallisuudelle on tietenkin se että käyttäjä pyrkii antamaan oikeanlaista dataa, eikä vääränmuotoista dataa anneta tahallaan. Mikäli tiedosto tai sen data ei täytä sille annettuja kriteereitä, ohjelma nostaa siitä virheen, eikä kuvaa piirry. Mikäli vain osa esimerkiksi viivadiagrammin graafeista on virheellisiä, piirtyvät vain oikein muotoillut graafit kuvaan. Ohjelman toiminta, kuten monen muunkin ohjelman toiminta, edellyttää käyttäjältä maalaisjärjen käyttöä, mikäli tämä haluaa järkeviä tuloksia.

Varsinaisena puutteena ohjelmassa on tekstien mahdollinen päällekkäisyys, mikäli käyttäjä antaa suuren pylväsdiagrammitiedoston tai nimet ovat muuten hyvin pitkiä. Tämäkin ratkeaa mahdollisesti käyttämällä kokoruudun näkymää, mutta pitkät nimet saattavat silti mennä päällekkäin. Tämä oltaisiin voitu korjata antamalla ohjelmalle jonkinlainen rivitys, jos tietyt ehdot merkkijonon pituudesta tai ikkunan leveydestä täyttyvät.

Virhetilanne syntyy myös, mikäli käyttäjä antaa vääränmuotoista dataa esimerkiksi ruudukon resoluutiota muokatessa. Tällöin ohjelma ei reagoi muutokseen. Mikäli käyttäjä antaa akseleille uudet nimet käyttämällä ääkkösiä, tulee tulosteeseen sekalaisia merkkejä, sillä ohjelma ei tue muita kuin tavallisia kirjainmerkkejä. Mikäli käyttäjä haluaa asettaa ruudukon resoluution hyvin suureksi, menettää ruudukko informatiivisuuttaan, sillä akseleiden numeroinnit saattavat mennä päällekkäin. Tämä oltaisiin voitu välttää tekemällä ehtolause ikkunan koon ja ruudukon resoluution välille.

9. 3 parasta ja 3 heikointa kohtaa

Parhaat kohdat

Ohjelmisto toimii kuten pitääkin ja olen siihen hyvin tyytyväinen. Näiden kohtien valinta oli haastavaa, eikä numeerinen järjestys kerro tässä tapauksessa mitään kohtien paremmuusjärjestyksestä.

1. Akseleiden skaalaus

Mielestäni akseleiden skaalaus toimii hienosti ja origon siirtyminen datan mukaan toimii niinikään. Jos käyttäjällä on dataa, joka alkaa negatiivisilla arvoilla, lukee ohjelma sen onnistuneesti ja tulostaa hienon graafin.

2. Ruudukko ja sen resoluutio

Ruudukko piirtyy hyvin ja tyylikkäästi kuvaan. Siitä saa paljon informaatiota niin viiva kuin pylväsdiagrammeissakin, joten se on hyvin hyödyllinen työkalu datan hahmottamiseksi. Ruudukon resoluution muuttaminen paremman hahmottamisen aikaansaamiseksi toimii järkevillä arvoilla ikkunan koosta riippuen. Uudet resoluution arvot päivittyvät ikkunaan hienosti.

3. Ikkuna ja graafinen ulkoasu

Mielestäni ikkuna on tarkoituksenmukainen, pelkistetty ja informatiivinen. Värit piirtyvät hyvin ja ovat äärimmäisen todennäköisesti erilaisia. Selitteet näkyvät selkeästi. Ikkunan yläpalkissa olevat toiminnot toimivat hyvin ja ovat tarpeellisia ja tärkeitä. Samoja funktioita voi toki kutsua ilman ikkunan nappien käyttöä, mutta yksinkertaisuuden kannalta on järkevää käyttää valintapalkkia. Ikkuna muistuttaa hieman ulkoasultaan kaupallista visualisointiohjelmaa.

Heikoimmat kohdat

1. Tekstien mahdollinen päällekkäisyys

Mikäli käyttäjän tiedostossa on hyvin pitkiä merkkijonoja niminä esimerkiksi viivadiagrammeille tai pylväsdiagrammin osille, saattavat kirjaimet mennä päällekkäin tai suorastaan ulos ikkunasta, näyttäen hyvin rumalta.

2. Ruudukon ja akselien numerointi

Ohjelma ei ole täysin ”idioottivarma”, eli mikäli käyttäjä päättää antaa vääränlaisia arvoja tai valtavia resoluutioita ruudukolle, menevät akselien numerot päällekkäin.

3. Kuvan/Graafin tallentaminen

Ohjelma käyttää valmista funktiota tähän, mutta graafin tallentaminen pakatuksi jpg-kuvaksi ei ole välttämättä se kaikkein elegantein ratkaisu.

10. Poikkeamat suunnitelmasta

Toisin kuin suunnitelmassa alun perin kerrottiin, jäi projektin lopputuotteesta pois muutamia ominaisuuksia. Ensimmäinen puuttuva ominaisuus on tuetut tiedostoformaatit. Ohjelma ei tue ”mitä tahansa” tiedostoformaattia, vaan luettavan datan tulee olla hyvin spesifisti file_IO:n tukeman syntaksin mukainen. Tässä omassa tiedostosyntaksissa on omat etunsa, mutta puutteena tietenkin se, ettei käyttäjä voi suoraan sisällyttää esimerkiksi Excelistä saamiaan arvoja kuvaajaan. Toinen puuttuva, mutta suunniteltu, ominaisuus on zoomauksen puuttuminen. Tämän ominaisuuden implementointi olisi tuottanut paljon ylimääräistä työtä, vaikka sen hyödyllisyys käytössä olisikin ollut suuri.

Muutoksia tehtiin myös suunniteltuihin datarakenteisiin lisäämällä yksi abstraktiotaso Graph-luokkien väliin. Visuadata-luokkaa ei toteutettu, mutta sen toiminnallisuus ja ominaisuudet löytyvät hyvin pitkälti GUI-luokasta. Muuten ohjelma vastaa pääpiirteissään suunniteltua.

11. Toteutunut työjärjestys ja aikataulu

Työjärjestys

Ohjelman työjärjestys toteutui pitkälti suunnitellusti. Ensiksi toteutettiin tiedostonluku ja tietorakenteiden muodostus, jonka jälkeen keskityttiin graafisen toteutuksen aikaansaamiseen. Yksikkötestaus ja viimeistely jätettiin aivan viimeiseksi asiaksi ennen dokumentointia. Poikkeus suunniteltuun työjärjestykseen tehtiin testauksen jättämisellä loppuun.

Itse projektin edistyminen löytyy hyvin selkeästi git-repositoryn commit-historiasta, johon pyrin lisäämään aina commitin yhteydessä tilanneraporttia.

Pääpiirteissään aikataulu oli seuraava:

21.03. Git-projektin luominen

24.03. Tiedostojen luominen ja jäsentäminen

25.03. File_IO lähes valmis

13.04. File_IO valmis ja testattu manuaalisesti. PyQt:n implementointi alkaa

03.05. Graafinen toteutus valmis, testaus ja dokumentointi kesken

Ajankäyttö

Arvioitu aikamäärä on sulkeissa.

Algoritmien ja tiedostonkäsittelyn toteuttaminen 30h(30h)

Visualisoivan osan toteuttaminen 40h(30h)

Viimeistely, testaaminen ja korjaaminen 20h(20h)

Dokumentointi 10h(10h)

YHTEENSÄ: 100h(100h)

12. Yhteenveto lopputuloksesta

Omasta mielestäni ohjelma täyttää kaikki sille asetetut tavoitteet. Se lukee käyttäjän antaman datan, muodostaa siitä oikeat tietorakenteet ja piirtää näiden mukaisen kuvaajan. Kuvaajaan on mahdollista vaikuttaa ruudukon ja seilitteiden, sekä akselien nimien osalta. Ohjelma tukee viivadiagrammin lisäksi piirakka ja pylväsdiagrammeja. Ulkoasu on selkeä ja funktionaalinen. Kuvaaja on mahdollista tallentaa kuvana ja uuden kuvaajan saa avattua tiedostonvalinta ikkunasta. Luettu data tarkistetaan ja ohjelma on läpäissyt sille asetetut yksikkötestit. Graafinen ulkoasu on testattu manuaalisesti.

Itse olen ohjelmaan tyytyväinen. Asioita, jotka olisi voinut tehdä paremmin, on muutama, kuten ohjelman laajennettavuus sisällytettävyys muihin ohjelmointitöihin mahdollisimman yksinkertaisesti ja tekstien pituuksien tarkistaminen. Näin jälkikäteen ajateltuna ohjelman olisi voinut toteuttaa vielä modulaarisemmin ja useampaan tiedostoon, jotta sisällytettävyys olisi ollut helpompaa muihin koodeihin. Nämä asiat voitaisiin tulevaisuudessa korjata hyvinkin yksinkertaisesti.

Mielestäni ohjelman luokkajako ja muut ratkaisut ovat tarkoituksenmukaisia ja perusteltuja. Ohjelmaa voidaan helposti laajentaa esimerkiksi lisäämällä puuttuva zoomaus-ominaisuus, kuvaajan tarkan datan kertova kursori tai mahdollisuus poistaa ikkunasta yksittäisiä kuvaajia. Nämä kaikki on mahdollista toteuttaa ohjelman pohjalta, koska tiedostonkäsittelyn luoma tietorakenne on hyvin itsenäinen ja modulaarinen.

13. Viitteet

Hyödynnetyt lähteet:

PyQt4 Dokumentaatio: http://pyqt.sourceforge.net/Docs/PyQt4/classes.html

Python 3 Dokumentaatio: https://docs.python.org/3/

Kurssin luentokalvot A+:ssa

Kurssin tehtäväkierros 4

Stackoverflow-foorumi http://stackoverflow.com/