# Tietokoneohjelma epälineaaristen elektroniikkapiirien simulointiin

Matias Pekkarinen

25. huhtikuuta 2014

Pro Gradu tutkielma
Jyväskylän yliopisto, Fysiikan laitos
25. huhtikuuta 2014
Ohjaaja: Kari Loberg

**Tiivistelmä**

Tässä työssä esitetään kuinka lineaarisissa ja epälineaarisissa elektroniikkapiireissä toimiva piirisimulaattoriohjelma voidaan toteuttaa oliokielellä. Työ aloitettiin itse kehitetyn TICER-yksinkertaistusalgoritmin pohjalta ja yhtenä työn motivaationa olikin tutkia kuinka kyseistä algoritmia voi käyttää muiden analysointimenetelmien apuna. Työn edetessä osoittautuikin, että TICER-yksinkertaistusta kannattaa käyttää muiden analysointimenetelmien apuna, sillä se voi nopeuttaa huomattavasti RC-piirien analysointia. TICER ei kuitenkaan nopeuttanut ohjelman toimintaa poikkeuksetta, mutta se ei hidastanutkaan analysointia koskaan niin paljon, etteikö sitä olisi kannattanut käyttää. Ohjelman pääasiallisena analysointimenetelmänä käytettiin solmupisteanalyysiä ja sillä muodostetut yhtälöryhmät ratkaistiin lineaarisissa piireissä yleensä LU-hajotelmalla tai Gaussin eliminoinnilla, sekä epälineaarisissa piireissä Newton-Raphsonin menetelmällä. Ohjelma ohjelmoitiin javalla ja se kykenee suorittamaan elektroniikkapiireille niin tasavirta-, vaihtovirta-, kuin myös transienttianalyysejä.

# Sisältö

# 1 Johdanto

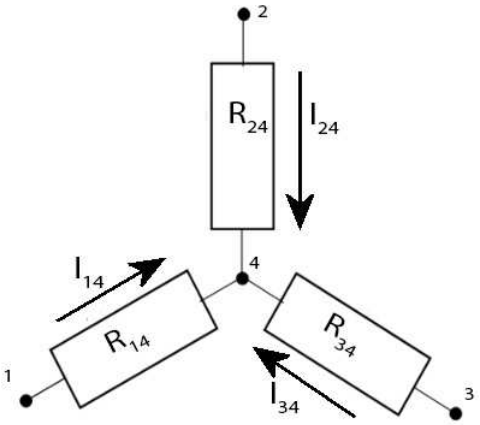
Tämän työn tarkoituksena on kehittää tietokoneohjelma, jota voidaan käyttää apuna, sekä lineaaristen, että epälineaaristen piirien tutkimisessa. Ohjelma ohjelmoidaan oliokielellä. Työn lähtökohtana on ohjelman edellinen kehitysvaihe, eli työssä [13] kehitetty TICER-yksinkertaistusalgoritmi, jota voidaan soveltaa yksinkertaistuksen lisäksi myös erilaisissa piirianalyyseissä. Tarkoituksena ei kuitenkaan ole pelkästään jatkokehittää kyseistä algoritmia, vaan myös tutkia kuinka muita piirien analysointimenetelmiä voidaan toteuttaa oliolähtökohtaisesti. Sekä tutkia kuinka hyvin yksinkertaistus toimii niiden apualgoritmina. Käytännössä yksinkertaistettu piiri on yleensä aina helpompi analysoida, mutta siitä ei kuitenkaan ole aina varmuutta, että maksaako yksinkertaistuksen tuoma helpotus analyysiin itse yksinkertaistuksen vaivan. Eli voidaanko ensin yksinkertaistus ja sen jälkeen yksinkertaistetun piirin analysointi suorittaa niin nopeasti, että se on nopeampi menetelmä kuin yksinkertaistamattoman piirin suora analysointi. Myös ohjelman toiminta-aluetta on tarkoitus laajentaa. Ohjelman edellisessä versiossa voitiin suorittaa vain tasavirta-analyysejä lineaarisille piireille, mutta tässä työssä kehitetyn ohjelman tarkoituksena on toimia myös epälineaarisissa piireissä, sekä tarjota mahdollisuudet tasavirta-, vaihtovirta- ja transienttianalyyseihin.

# 2 Piiriyhtälöiden muodostaminen

## 2.1 Solmupisteanalyysi

Solmupisteanalyysi on piirianalyysi, missä piiristä valitaan yksi kytkentäpiste vertailukohteeksi ja muut piirin kytkentäpisteiden jännitteet lasketaan verrattuna siihen. Vertailupisteeksi valitaan usein maadoitettu kytkentäpiste, sillä sen jännite tunnetaan jo ennalta ja sille ei tarvita omaa jännitemuuttujaa. Piiriyhtälöiden muodostaminen aloitetaan muodostamalla virtayhtälö piirin jokaiselle kytkentäpisteelle. Virtayhtälö muodostetaan Kirchoffin virtalain mukaan, jonka mukaan kaikkien kytkentäpisteeseen saapuvien virtojen summa on nolla. Eli esimerkiksi kuvan 1 mukaisen kytkennän kytkentäpisteen 4 virtayhtälö voidaan kirjoittaa.

$$i_{14} + i_{24} + i_{34} = 0 \tag{1}$$




Kuva 1: KCL-yhtälö

Tämän jälkeen saadut yhtälöt pyritään muokkaamaan siten, että siinä käytetään muuttujina vain jännitemuuttujia. Muokkaus onnistuu Kirchoffin jännitelain avulla, jonka mukaan esimerkiksi kytkentäpisteiden 1, 4 välinen jännite on $v_{14} = v_1 - v_4$ ja siten virtayhtälön muuttuja $i_{14}$ voidaan korvata yhtälöllä $i_{14} = \frac{V_1 - V_4}{R_{14}}$. Sama voidaan toistaa jokaisen virtamuuttujan kohdalla ja kun yhtälöstä (1) korvataan kaikki virtamuuttujat jännitemuuttujilla, saadaan

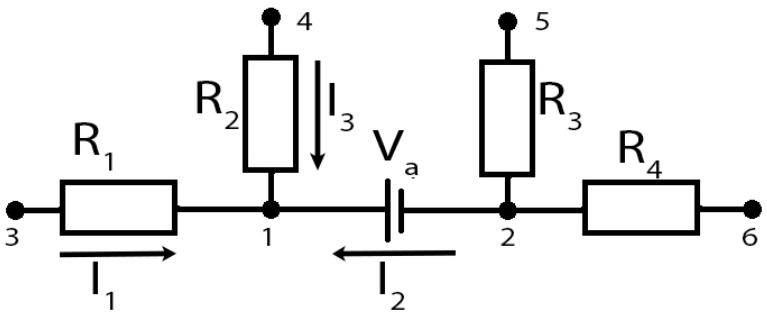
$$\frac{V_1 - V_4}{R_{14}} + \frac{V_3 - V_4}{R_{34}} + \frac{V_2 - V_4}{R_{24}} = 0.$$

Sen jälkeen kun virtamuuttujien korvaaminen on suoritettu jokaisen virtayhtälön kohdalla, niistä voidaan muodostaa yhtälöryhmä, jonka ratkaisusta saadaan selville kaikkien kytkentäpisteiden jännitteet.

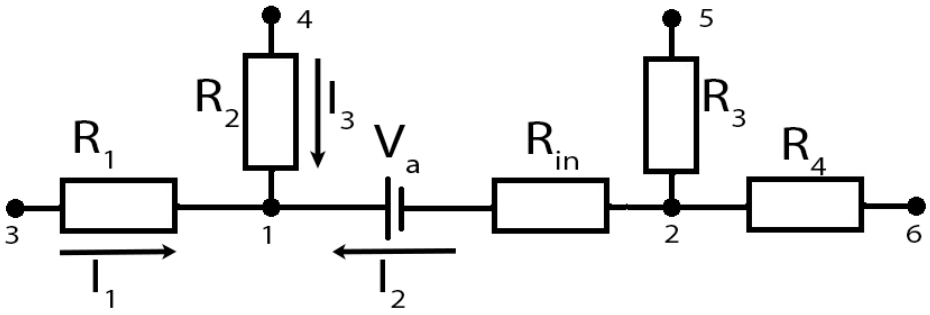
Solmupistemenetelmän käyttäminen edellyttää usein piirin muokkausta, sillä joskus kytkentäpisteen virtayhtälöä ei voida suoraan muuntaa pelkästään jännitemuuttujista riippuvaiseksi. Näin on esimerkiksi kuvan 2 mukaisen jännitelähteen kohdalla, koska kytkennän virtayhtälön $i_1 + i_2 + i_3 = 0$ muuttujaa $i_2$ ei voida suoraan kuvantaa kytkentävälin 1, 2 jännitemuuttujilla. Käytännössä jokaisella jännitelähteellä on kuitenkin myös oma sisäinen vastus ja siten kytkentä voidaan korvata. Esimerkiksi jos oletetaan, että kuvan 2 mukaisen jännitelähteen $V_1$ sisäinen vastus on $R_{in}$, niin se voidaan korvata kuvan 3 mukaisella kytkennällä [3]. Tämä ei kuitenkaan ole ratkaisu ongelmaan, sillä simulointiohjelman on kyettävä analysoimaan myös ideaalilähteitä. Ongelma voidaan korjata käsittelemällä kuvan 2 mukaista jännitelähdettä ns. superkytkentäpisteenä, jolloin kytkentäpisteistä 1, 2 vain toista käsitellään vertailupisteenä [15]. Eli yhtälöt muodostetaan oikeastaan kuvan 4 mukaisesta piiristä ja kyseisen kytkennän yhtälö voidaan kirjoittaa muotoon

$$\frac{V_1 - V_3}{R_1} + \frac{V_1 - V_4}{R_2} + \frac{V_1 - V_a - V_5}{R_3} + \frac{V_1 - V_a - V_6}{R_4} = 0.$$

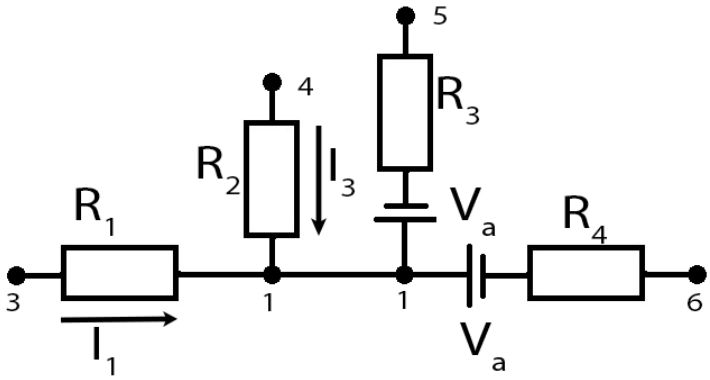
Tällainen kytkennän korvaaminen on kuitenkin turhaa, sillä usein etenkin laajoissa piireissä siitä voi tulla hyvin monimutkaista. Parhaana ratkaisuna ongelmaan toimii analysointimenetelmän muokkaaminen ja käytössä olevien muuttujien määrän lisääminen. Esimerkiksi kuvan 2 mukaisen kytkennän virtayhtälöön $i_1 + i_2 + i_3 = 0$ voidaan $i_2$ jättää muuttujaksi ja näin ongelma ohitetaan lisäämällä tuntemattomien määrää yhdellä. Kyseistä analysointimenetelmää kutsutaankin muokatuksi solmu-




Kuva 2: Superkytkentäpiste



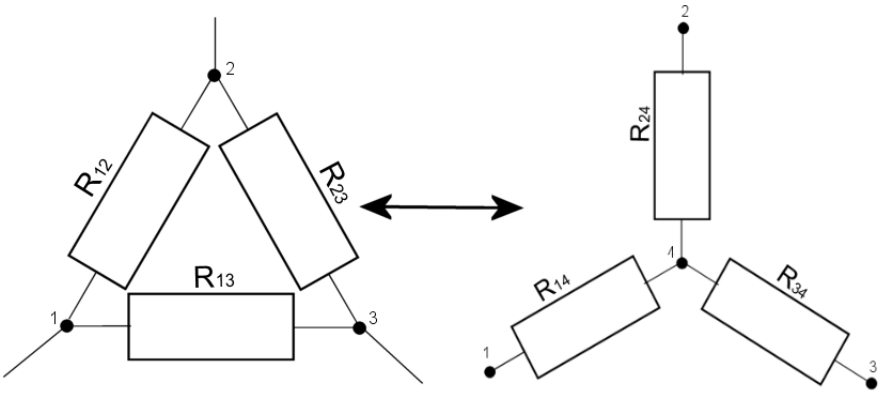

Kuva 3: Thevenin muunnos jännitelähteestä




Kuva 4: Kuvan 2 kytkennän muunnos

pisteanalyysiksi.

Muokattu solmupisteanalyysi on nimensä mukaisesti solmupisteanalyysistä kehitetty analyysimenetelmä. Muokattu solmupisteanalyysi ei paljoa eroa tavallisesta solmupisteanalyysistä. Ainoana erona voidaan pitää sitä, että kaikkia Kirchoffin virtalain mukaisesti muodostetuita yhtälöitä ei muokata pelkästään jännitemuuttujista riippuvaisiksi, vaan jännitemuuttujien lisäksi muuttujina käytetään myös virtamuuttujia. Kyseisten lisämuuttujien tarkoituksena ei kuitenkaan ole helpottaa yhtälöryhmien ratkaisua vaan niiden muodostamista [11]. Lisämuuttujat itse asiassa useimmiten vain vaikeuttavat yhtälöryhmän ratkaisua, mutta algoritmitasolla laajat yhtälöryhmät eivät ole niin suuri ongelma kuin tulkinnanvaraiset ja muokattavat kytkennät. Edellä osoitettiinkin jo kuinka riippumattoman jännitelähteen tapauksessa lisämuuttujaksi jätetty virta helpottaa solmupisteyhtälön muodostamista. Sama idea pätee myös useille muille komponenteille, kuten riippuvaisille jännitelähteille ja induktoreille. Piiriin jätetään muokatussa solmupisteanalyysissä tavallisesti myös ne virtamuuttujat, joista virtariippuvaiset lähteet ovat riippuvaisia.




Kuva 5: Kolmio-tähti muunnos

## 2.2 TICER

TICER on piirien yksinkertaistusmenetelmä, jota voidaan käyttää apuna piiriyhtälöiden muodostamisessa. TICERiä sovelletaan yleensä vain RC-piireissä, koska RLC-piireissä TICERissä käytettävä kolmio-tähti muunnos muuttaa piirin ominaisuuksia [14]. Yksinkertaistaminen aloitetaan kaikista rinnan- ja sarjaankytketyistä passiivi komponenteista, koska niiden yksinkertaistaminen voi luoda uusia tähti- ja kolmiokytkentöjä. Tämän jälkeen yksinkertaistamista jatketaan ns. kolmio-tähti muunnoksen avulla. Kolmio-tähti muunnoksessa kuvassa 5 oleva kolmion muotoinen kytkentä muunnetaan saman kuvan tähtikuvion mukaiseksi kytkennäksi tai toisinpäin.

Kyseinen muunnos kolmiosta tähdeksi yksinkertaistaa piiriä, sillä solmupisteiden määrä voi vähentyä kahdella. Muunnosta voidaan käyttää kytkentäpisteissä, joihin on kytketty vain vastuksia tai kondensaattoreita. Muunnos onnistuu yhtälöillä [4]

$$\begin{aligned} R_{14} &= \frac{R_{12} \cdot R_{13}}{R_{12} + R_{23} + R_{13}} \\ R_{24} &= \frac{R_{12} \cdot R_{23}}{R_{12} + R_{23} + R_{13}} \\ R_{34} &= \frac{R_{13} \cdot R_{23}}{R_{12} + R_{23} + R_{13}} \end{aligned}$$

ja muunnettaessa tähtikytkentä kolmiokytkennäksi voidaan käyttää seuraavia.

$$R_{12} = \frac{R_{12} \cdot R_{13} + R_{12} \cdot R_{23} + R_{13} \cdot R_{23}}{R_{34}}$$

$$R_{13} = \frac{R_{12} \cdot R_{13} + R_{12} \cdot R_{23} + R_{13} \cdot R_{23}}{R_{24}}$$

$$R_{23} = \frac{R_{12} \cdot R_{13} + R_{12} \cdot R_{23} + R_{13} \cdot R_{23}}{R_{14}}$$

Vaikka TICER on edellä mainitun mukaisesti yksinkertaistusmenetelmä, soveltuu se myös RC-piirien analysointiin, joissa sitä sovelletaan superpositiopiireittäin. Superpositiopiireittäin käytettynä TICER yksinkertaistaa piirit niin pitkälle, että yksinkertaistetussa piirissä ei ole enää yhtään tuntematonta muuttujaa. Yksinkertaistamisen jälkeen piirin yksinkertaistus voidaan purkaa ja samalla voidaan laskea palautetun piirin tuntemattomat tekijät. Lopulta kun koko superpositiopiirin yksinkertaistus on palautettu, ei superpositiopiirissä ole enää yhtään tuntematonta muuttujaa. Superpositiopiireittäin soveltaminen ei rajoita TICERin käyttöä vain riippumattomiin ja lineaarisiin piireihin. Sillä piirejä voidaan, sekä linearisoida, että analysoida riippuvuuksien mukaan. Tuohon piirin linearisointiin palataan myöhemmin, mutta jos analysoitavana on liitteen C mukainen piiri, jossa ilmenee riippuvuuksia, niin analysointi kannattaa aloittaa riippumattomista superpositiopiireistä. Riippumattomat superpositiopiirit lasketaan kuten tavallisesti, mutta riippuvaisista lähteistä muodostetuissa superpositiopiireissä riippuvaisten lähteiden arvot käsitellään symbolisina. Esimerkiksi lähteen $e1$ arvona käytetään muuttujaa $999000V_{12}$, koska se on riippuvainen kytkentävälin (1,2) jännitteestä ja sen kerroin on 999000. Lopuksi kaikkien superpositiopiirien arvot summataan yhteen ja jos oletetaan, että $V1 = 3$ V niin vaikuttavilla väleillä (1,2), (4,5), (8,7) superpositiopiirien jännitteiden summat ovat $3-666000V_{12}-333000V_{45}-0V_{87}$, $5-333000V_{12}-666000V_{45}+0V_{87}$, $0 - 499500V_{12} + 499500V_{45} - 499500V_{87}$. Kyseisistä arvoista voidaan edelleen muodostaa yhtälöryhmä

$$\begin{aligned} V_{12} &= 3 - 666000V_{12} - 333000V_{45} - 0V_{87} \\ V_{45} &= 5 - 333000V_{12} - 666000V_{45} + 0V_{87}. \\ V_{87} &= 0 - 499500V_{12} + 499500V_{45} - 499500V_{87} \end{aligned}$$

Yhtälöryhmän ratkaisemisen jälkeen vaikuttavien tekijöiden arvot tunnetaan $V_{12} =$

$1.00101 \cdot 10^{-6}$ V, $V_{45} = 6.99997 \cdot 10^{-6}$ V, $V_{87} = 6.00598 \cdot 10^{-6}$ V ja ne voidaan sijoittaa. Esimerkiksi sijoitettaessa arvot riippuvaisten jännitelähteiden yhtälöihin saadaan $V_3 = 999000V_{12} = 1,00001$ V, $V_6 = 999000V_{45} = 6,99999$ V, $V_9 = 999000V_{87} = 5,99997$ V.

Edellä osoitetun mukaisesti TICER on kätevä analysointimenetelmä, mutta sen avulla suoritettava yksinkertaistus, sekä nopeuttaa että helpottaa myös muiden analysointimenetelmien käyttöä. Esimerkiksi solmupisteanalyysissä yhtälöryhmän muuttujien määrä riippuvainen solmupisteiden määrästä ja TICERiä voidaan käyttää niiden vähentämisessä. Siihen kuinka paljon TICER nopeuttaa solmupisteanalyysissä palataan luvuissa 5.5.2 ja 5.6.

# 3 Piiriyhtälöiden ratkaiseminen

## 3.1 Lineaariset yhtälöryhmät

Lineaarisesta piiristä solmupistemenetelmällä muodostettu yhtälöryhmä on aina muotoa

$$\begin{cases} N_1: & a_{11}V_1+ & a_{12}V_2+ & ...+ & a_{1n}V_n = & b_1 \\ N_2: & a_{21}V_1+ & a_{22}V_2+ & ...+ & a_{2n}V_n = & b_2 \\ \vdots & & & & & \\ N_n: & a_{n1}V_1+ & a_{n2}V_2+ & ...+ & a_{nn}V_n = & b_n \end{cases} \tag{2}$$

, missä $V_1, V_2, ..., V_n$ ovat solmupisteiden tuntemattomia jännitteitä. Kyseinen yhtälöryhmä voidaan esittää myös muodossa

$$Ax = b \tag{3}$$

, missä $A$ on yhtälöryhmän muuttujien kertoimien arvoja esittävä matriisi, $x$ on muuttujien $V_1, V_2, ..., V_n$ vektori ja $b$ on vakioarvojen $b_1, b_2, ..., b_n$ vektori. Yhtälöryhmän ratkaisemiseksi yhtälöryhmä on yksinkertaistettava muotoon

$$\begin{cases} N_1: & V_1 & 0 & \cdots & 0 & = & r1 \\ N_2: & 0 & V_2 & \ddots & \vdots & = & r_2 \\ \vdots & \vdots & \ddots & \ddots & 0 & & \vdots \\ N_n: & 0 & \cdots & 0 & V_n & = & r_n \end{cases}$$

, missä $a_{ii} = 1$ ja $a_{ij} = 0$, kun $i \neq j$. Lineaarisen yhtälöryhmän ratkaisemisessa voidaan käyttää, sekä ns. suoria, että iteratiivisia menetelmiä. Suorissa menetelmissä yhtälöryhmä ratkaistaan äärellisellä määrällä lineaarisia laskutoimituksia. Suorilla menetelmillä saatu tulos on teoreettisesti tarkka, mutta pyöristysvirheet vaikuttavat kuitenkin osittain tulokseen. Suorat menetelmät eivät tarkkuudestaan huolimatta kuitenkaan ole aina se paras vaihtoehto yhtälöryhmän ratkaisemiseksi, vaan ne kannattaa korvata ns. iteratiivisilla menetelmillä. Iteratiivisissa menetelmissä oikean ratkaisun etsiminen aloitetaan ns. alkuarvauksesta ja alkuarvauksen

arvoa parannetaan vaihe vaiheelta. Menetelmä on sitä nopeampi, mitä lähempänä alkuarvaus on oikeaa tulosta ja esimerkiksi graafien pisteitä laskettaessa laskettavan pisteen arvo ei yleensä ole kovin kaukana edellisestä. Tämän takia ainakin graafeja laskettaessa suorat menetelmät kannattaa usein korvata iteratiivisilla. Iteratiivisten menetelmien etuna graafeissa on myös se, että se on tarkka vain haluttuun pisteeseen asti ja koska graafit voidaan esittää vain tiettyyn tarkkuuteen asti, ei suorien menetelmien tarkkuudesta ole hyötyä. Iteratiivisia menetelmiä ei kuitenkaan voida soveltaa ihan joka tilanteessa, sillä edellytyksenä iteratiivisten menetelmien käytölle on, että yhtälöryhmän (3) kerroinmatriisi $A$ suppenee [6]. Suppenemiseen vaikuttavat diagonaaliarvojen suuruus muihin arvoihin verrattuna ja suppeneminen on sitä nopeampaa, sekä varmempaa mitä suurempia diagonaalielementit ovat verrattuna muihin arvoihin. Suppenemisen varmistamiseksi ja nopeuttamiseksi yhtälöryhmää tavallisesti muokataankin siten, että diagonaaliarvot ovat suuria. Muokkaus onnistuu mm. permutaatiomatriisin avulla. Iteratiivisia menetelmiä ei kannata soveltaa, jos tälläinen muokkaus ei onnistu.

### 3.1.1 Gaussin eliminointi

Gaussin eliminointi on ns. suora menetelmä yhtälöryhmän ratkaisemiseksi. Gaussin eliminoinnissa yhtälöryhmä (2) saatetaan lineaarisia laskutoimituksia

$$N_i = \lambda N_i \tag{4}$$

$$N_i = N_i \pm N_k \lambda \tag{5}$$

hyödyntäen yläkolmiomuotoon

$$\begin{cases} N_1: & a_{11}V_1+ & a_{12}V_2+ & ...+ & a_{1n}V_n = & y_1 \\ N_2: & 0 & a_{22}V_2+ & ...+ & a_{2n}V_n = & y_2 \\ \vdots & \vdots & \ddots & \ddots & & \\ N_n: & 0 & \cdots & 0 & a_{nn}V_n = & y_n \end{cases}$$

, missä $a_{ij} = 0$, kun $i > j$. Kun yhtälöryhmä on saatu tähän yläkolmio muotoon, on alin yhtälö helppo ratkaista, saadaan $V_n = y_n/a_{nn}$ ja tästä taaksepäin sijoittamalla

saadaan edelleen ratkaistua yhtälö $N_{n-1}$ ja sijoituksia jatkamalla on lopulta koko yhtälöryhmä ratkaistu.

**Esimerkki Gaussin eliminoinnista** Ratkaistaaksemme yhtälöryhmän

$$\begin{cases} N_1: & 3V_1+ & V_2+ & 2V_3 = & 1 \\ N_2: & 3V_1+ & 5V_2+ & V_3 = & 2 \\ N_3: & 3V_1+ & 2V_2+ & 8V_3 = & 3 \end{cases}$$

Gaussin menetelmällä on kyseinen yhtälöryhmä ensin saatava muokattua yläkolmiomatriisiksi. Muokkaus onnistuu eteenpäin etenevillä lineaarisilla laskutoimituksilla.

$$\begin{aligned} N_2 &= N_2 - N_1 \\ N_3 &= N_3 - N_1 \end{aligned}$$

$$\begin{cases} N_1: & 3V_1+ & V_2+ & 2V_3 = & 1 \\ N_2: & 0+ & 4V_2- & V_3 = & 1 \\ N_3: & 0+ & V_2+ & 6V_3 = & 2 \end{cases}$$

$$N_3 = N_3 - \frac{1}{4}N_2$$

$$\begin{cases} N_1: & 3V_1+ & V_2+ & 2V_3 = & 1 \\ N_2: & 0+ & 4V_2- & V_3 = & 1 \\ N_3: & 0+ & 0+ & 6.25V_3 = & 1.75 \end{cases}$$

Nyt on matriisi saatu yläkolmiomuotoon ja muokkausta jatketaan tästä taaksepäin etenevillä sijoituksilla tavoitteena tehdä kerroin matriisista yksikkömatriisi.

$$N_3 = N_3/6.25$$

$$\begin{cases} N_1: & 3V_1+ & V_2+ & 2V_3 = & 1 \\ N_2: & 0+ & 4V_2- & V_3 = & 1 \\ N_3: & 0+ & 0+ & V_3 = & 7/25 \end{cases}$$

$$N_2 = \frac{N_2 + N_3}{4}$$

$$\begin{cases} N_1: & 3V_1+ & V_2+ & 2V_3 = & 1 \\ N_2: & 0+ & V_2+ & 0 = & 8/25 \\ N_3: & 0+ & 0+ & V_3 = & 7/25 \end{cases}$$

$$N_1 = \frac{N_1 - N_2 - 2N_3}{3}$$

$$\begin{cases} N_1: & V_1+ & 0+ & 0 = & 1/25 \\ N_2: & 0+ & V_2+ & 0 = & 8/25 \\ N_3: & 0+ & 0+ & V_3 = & 7/25 \end{cases}$$

### 3.1.2 LU-hajotelma

LU-hajotelma on toiminnaltaan hyvin paljon Gaussin eliminoinnin kaltainen. Kummassakin menetelmässä yhtälöryhmää muokataan lineaarisilla operaatioilla, mutta kun Gaussin eliminoinnissa etsitään ratkaisua yhtälöryhmälle (3), niin LU-hajotelmassa etsitään ratkaisua yhtälöryhmälle

$$Ax = LUx = b \tag{6}$$

, missä $LU$ on matriisin $A$ kolmiohajotelma eli $A = LU$ [2]. Kolmiohajotelman matriiseista $L$ on alakolmiomatriisi ja $U$ on yläkolmiomatriisi. Niiden laskemiseen on olemassa useita eri menetelmiä ja ne saadaan laskettua myös Gaussin eliminointia hyödyntäen. Gaussin eliminoinnilla laskien LU-hajotelman yläkolmio matriisi U on sama, joka saadaan Gaussin eteenpäin eliminoinnissa ja alakolmiomatriisi on niiden kerrointen matriisi, joilla kerroinmatriisia $A$ muokattiin eteenpäin eliminoinnin aikana [16]. Esimerkiksi jos kappaleessa 3.1.1 esitetyn Gaussin eliminointi esimerkin mukaisesta kerroinmatriisista muodostetaan yläkolmio ja alakolmio matriisit. Niin

yläkolmiomatriisi on muotoa

$$U = \begin{matrix} 3 & 1 & 2 \\ 0 & 4 & -1 \\ 0 & 0 & 6.25 \end{matrix}$$

, eli se on sama kuin Gaussin eliminointi esimerkissä eliminointivaiheen jälkeen tulokseksi saatu yläkolmiomatriisi. Kyseisen esimerkin LU-hajotelman alakolmiomatriisi taas on eteenpäin suoritettujen eliminointien $N_2 = N_2 - N_1,\ N_3 = N_3 - N_1$, sekä $N_3 = N_3 - \frac{1}{4}N_2$ mukainen.

$$L = \begin{matrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & \frac{1}{4} & 1 \end{matrix}$$

Ylä- ja alakolmio matriisien muodostamisen jälkeen yhtälö (6) ratkaistaan kuten Gaussin eliminoinnissa, mutta eteenpäin eliminointi vaihe suoritetaan yhtälölle

$$Ly = b$$

ja sen jälkeen taaksepäin sijoitus yhtälölle

$$Ux = y.$$

### 3.1.3 Permutaatiomatriisi

Permutaatiomatriisi on matriisi, jossa on yksi nollasta poikkeava alkio jokaista riviä ja saraketta kohti, sekä jokaisen nollasta eroavan alkion arvo on 1. Permutaatiomatriisi toteuttaa seuraavat yhtälöt

$$\begin{aligned} \sum_{i=1}^{n} p_{ij} &= 1, \qquad 1 \leq j \leq n \\ \sum_{j=1}^{n} p_{ij} &= 1, \qquad 1 \leq i \leq n \end{aligned}$$

, koska siinä on yksi yksikköalkio jokaista riviä ja saraketta kohden. Eli permutaatiomatriisi on kuten yksikkömatriisi, mutta permutaatiomatriisin jokainen nollasta poikkeva alkio ei sijaitse matriisin diagonaalilla, kuten yksikkömatriisissa. Siten jokainen permutaatiomatriisi voidaan johtaa yksikkömatriisista muuttamalla yksikkömatriisin rivien järjestystä.

Permutaatiomatriisia tarvitaan iteratiivisissa menetelmissä, kun diagonaalilukuja on tarve muuttaa. Sitä käytetään myös joskus Gaussin eliminoinnissa tai LU-hajotelmassa, kun jokin matriisin diagonaalialkioista $a_{ii}$ on arvoltaan 0. Tällöin yhtälöryhmä on mahdotonta ratkaista muokkaamatta sitä, koska tuntemattoman muuttujan kerroin on 0. Gaussin eliminoinnissa tilanteen pystyy kylläkin sitä todennäköisemmin korjaamaan permutaatiomatriisin lisäksi myös yhtälöillä (4), (5), mitä varhaisemmassa vaiheessa yksinkertaistus on. LU-hajotelmassa tilanteen korjaaminen pelkästään yhtälöitä (4), (5) käyttäen on kuitenkin mahdotonta, koska matriisit L ja U ovat kolmiomatriiseja. Tilanteen korjaamiseksi tarvitaan rivien vaihtoa, joka onnistuu juuri permutaatiomatriisin avulla. Jos esimerkiksi rivien 1 ja 2 järjestystä täytyy muuttaa, on permutaatiomatriisi muotoa

$$P = \begin{matrix} 0 & 1 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & & \\ \vdots & \vdots & & \ddots & \\ 0 & 0 & & & 1 \end{matrix}.$$

Rivien järjestys muuttuu kertomalla yhtälöryhmä (3) permutaatiomatriisilla sen vasemmalta puolelta

$$PAx = Pb.$$

Permutaatiomatriisia voi käyttää myös sarakkeiden järjestyksen muuttamiseen, mutta silloin yhtälöryhmä (3) kerrotaan permutaatiomatriisilla sen oikealta puolelta, eikä vakioarvoja kerrota

$$AxP = b.$$

Permutaatiomatriisia voidaan käyttää myös suorien ja iteratiivisten menetelmien

tarkkuuksien parantamiseen. Kyseisten menetelmien tarkkuus paranee kun diagonaalielementtien arvot kasvavat verrattuna muihin arvoihin ja permutaatiomatriisin avulla diagonaaliarvoja voidaan joskus suurentaa verrattuna muihin arvoihin.

### 3.1.4 Gauss-Jacobin menetelmä

Gauss-Jacobin menetelmä ei ole Gaussin eliminoinnin ja LU-hajotelman tavoin suora menetelmä, vaan se on iteratiivinen menetelmä. Yhtälöryhmän (3) kerroinmatriisi A hajotetaan muotoon $A = L + D + U$ [19], missä

$$
\begin{aligned}
L &= \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ a_{n1} & \cdots & a_{n,n-1} & 0 \end{bmatrix} \\
D &= \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix} \\
U &= \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{n-1,n} \\ 0 & \cdots & 0 & 0 \end{bmatrix}.
\end{aligned}
$$

Kyseisessä hajotelmassa esiintyvät matriisit $L$, $U$ eivät kuitenkaan ole samat kuin kappaleessa 3.1.2 esitetyt LU-hajotelman $L$, $U$ matriisit. Yhtälön $A = L + D + U$ avulla yhtälö (3) saadaan muotoon

$$
\begin{aligned}
(L + D + U)x &= b \qquad (7) \\
Dx &= b - (L + U)x
\end{aligned}
$$

, josta saadaan johdettua

$$x = D^{-1}b - D^{-1}(L+U)x \tag{8}$$

ja johon sijoitetaan alkuarvaus mahdollisen ratkaisun arvoista. Yleensä alkuarvauksena käytetään arvoja $x = [0, ..., 0]^T$. Alkuarvaus sijoitetaan yhtälöön (8) ja laskuvaiheen k iteraatio voidaan esittää seuraavasti

$$x^{(k)} = D^{-1}b - D^{-1}(L+U)x^{(k-1)} \tag{9}$$

Suoraan kerroinmatriisin $A$ kertoimilla laskettuna yhtälö (9) voidaan kirjoittaa myös muotoon

$$x_i^{(k)} = \frac{1}{a_{ii}} \left( \sum_{\substack{j=1 \\ j \neq i}}^{n} (-a_{ij} x_j^{(k-1)}) + b_i \right), \qquad i = 1, 2, ..., n.$$

**Esimerkki Gauss-Jacobin menetelmän käytöstä**

$$\begin{bmatrix} 3 & 1 & 2 \\ 3 & 5 & 1 \\ 3 & 2 & 8 \end{bmatrix} x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Yhtälöryhmän ratkaiseminen aloitetaan tekemällä kerroinmatriisista hajotelma.

$$L = \begin{bmatrix} 0 & 0 & 0 \\ 3 & 0 & 0 \\ 3 & 2 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 8 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Hajotelman tekemisen jälkeen on valittava alkuarvaus $x^{(0)} = [0, 0, 0]^T$, joka sijoitetaan yhtälöön (9) ja saatu yhtälöryhmä ratkaistaan.

$$
\begin{aligned}
Dx^{(1)} &= b - (L+U)x^{(0)} \\
\begin{bmatrix} 3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 8 \end{bmatrix} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} &= \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 2 \\ 3 & 0 & 1 \\ 3 & 2 & 0 \end{bmatrix} \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \\ x_3^{(0)} \end{bmatrix} \\
\begin{bmatrix} 3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 8 \end{bmatrix} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} &= \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 2 \\ 3 & 0 & 1 \\ 3 & 2 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\
\begin{bmatrix} 3x_1^{(1)} & 0 & 0 \\ 0 & 5x_2^{(1)} & 0 \\ 0 & 0 & 8x_3^{(1)} \end{bmatrix} &= \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \\
x_1^{(1)} &= 1/3 \\
x_2^{(1)} &= 2/5 \\
x_3^{(1)} &= 3/8
\end{aligned}
$$

Yhtälöryhmän ratkaisemisen jälkeen saatu tulos sijoitetaan uudelleen yhtälöön (9) ja yhtälöryhmä ratkaistaan.

$$
\begin{aligned}
\begin{bmatrix} 3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 8 \end{bmatrix} \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{bmatrix} &= \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 2 \\ 3 & 0 & 1 \\ 3 & 2 & 0 \end{bmatrix} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} \\
\begin{bmatrix} 3x_1^{(2)} & 0 & 0 \\ 0 & 5x_2^{(2)} & 0 \\ 0 & 0 & 8x_3^{(2)} \end{bmatrix} &= \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} - \begin{bmatrix} 1.15 \\ 1.375 \\ 1.8 \end{bmatrix} \\
\begin{bmatrix} 3x_1^{(2)} & 0 & 0 \\ 0 & 5x_2^{(2)} & 0 \\ 0 & 0 & 8x_3^{(2)} \end{bmatrix} &= \begin{bmatrix} -0.15 \\ 0.625 \\ 1.2 \end{bmatrix} \\
x_1^{(2)} &= -0.05 \\
x_2^{(2)} &= 0.125 \\
x_3^{(2)} &= 0.15
\end{aligned}
$$

Tätä jatketaan niin kauan kunnes tulos on niin tarkka kuin on tarpeen.

### 3.1.5 Gauss-Seidelin menetelmä

Gauss-Seidelin menetelmä on Gauss-Jacobin menetelmän tavoin iteratiivinen ja muutenkin hyvin samankaltainen kuin Gauss-Jacobin menetelmä. Ainoana erona voidaan pitää sitä, että yhtälön (8) sijasta ratkaistaan yhtälö

$$\begin{aligned} (L+D)x &= b - Ux \\ x &= (L+D)^{-1}b - (L+D)^{-1}x \end{aligned} \tag{10}$$

, eli yhtälöä (7) muokataan vain toisin ja siten yhtälöryhmän ratkaisulle saadaan erilaiset lähtökohdat, kuin Gauss-Jacobin menetelmässä. Yhtälö (10) voidaan kirjoittaa kerroinmatriisin $A$ kertoimien avulla myös muotoon [7].

$$x_i^{(k)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} (a_{ij} x_j^{(k)}) - \sum_{j=i+1}^{n} (a_{ij} x_j^{(k-1)}) \right)$$

Kyseisestä yhtälöstä voidaan havaita, että iteraatioissa hyödynnetään myös saman iteraatiovaiheen aiemmin laskettuja arvoja $x_j^{(k)}$. Eli myös tältä osin Gauss-Seidelin menetelmä poikkeaa Gauss-Jacobin menetelmästä, sillä Gauss-Jacobin menetelmässä hyödynnetään jokaisessa iteraatiossa vain edellisessä iteraatiovaiheessa saatuja arvoja.

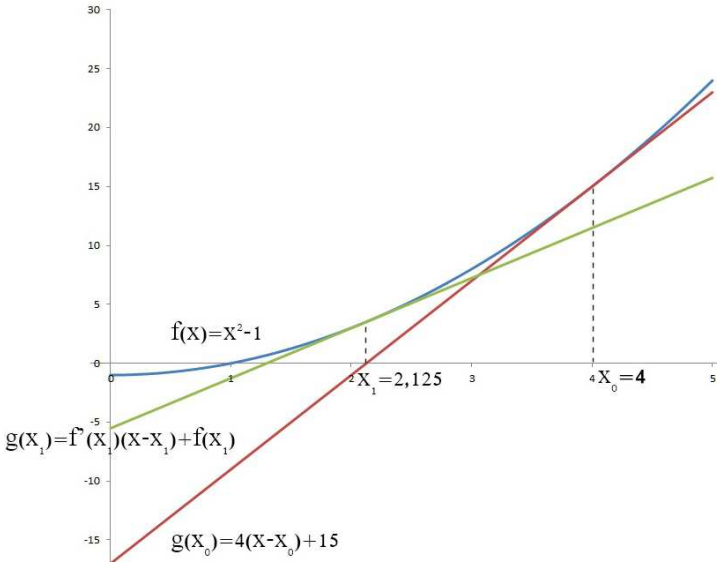
**Esimerkki Gauss-Seidelin menetelmän käytöstä** Yhtälöryhmän ratkaiseminen Gauss-Seidelin menetelmällä etenee hyvin samanlaisesti kuin Gauss-Jacobin menetelmässä. Vain yhtälö (10), johon hajotelman osat ja alkuarvaus sijoitetaan on erilainen.

$$\begin{bmatrix} 3 & 1 & 2 \\ 3 & 5 & 1 \\ 3 & 2 & 8 \end{bmatrix} x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$L = \begin{bmatrix} 0 & 0 & 0 \\ 3 & 0 & 0 \\ 3 & 2 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 8 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{aligned}
(D+L)x^{(1)} &= b - Ux^{(0)} \\
\begin{bmatrix} 3 & 0 & 0 \\ 3 & 5 & 0 \\ 3 & 2 & 8 \end{bmatrix} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} &= \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 2 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \\ x_3^{(0)} \end{bmatrix} \\
\begin{bmatrix} 3x_1^{(1)} & 0 & 0 \\ 3x_1^{(1)} & 5x_2^{(1)} & 0 \\ 3x_1^{(1)} & 2x_2^{(1)} & 8x_3^{(1)} \end{bmatrix} &= \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \\
x_1^{(1)} &= 1/3 \\
x_2^{(1)} &= (2-1)/5 = 1/5 \\
x_3^{(1)} &= (3-1-2/5)/8 = 0.2
\end{aligned}$$

$$\begin{aligned}
\begin{bmatrix} 3 & 0 & 0 \\ 3 & 5 & 0 \\ 3 & 2 & 8 \end{bmatrix} \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{bmatrix} &= \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 2 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} \\
\begin{bmatrix} 3x_1^{(2)} & 0 & 0 \\ 3x_1^{(2)} & 5x_2^{(2)} & 0 \\ 3x_1^{(2)} & 2x_2^{(2)} & 8x_3^{(2)} \end{bmatrix} &= \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} - \begin{bmatrix} 0.6 \\ 0.2 \\ 0 \end{bmatrix} \\
\begin{bmatrix} 3x_1^{(2)} & 0 & 0 \\ 3x_1^{(2)} & 5x_2^{(2)} & 0 \\ 3x_1^{(2)} & 2x_2^{(2)} & 8x_3^{(2)} \end{bmatrix} &= \begin{bmatrix} 0.4 \\ 1.8 \\ 3 \end{bmatrix} \\
x_1^{(2)} &= -2/15 \\
x_2^{(2)} &= (1.8-0.4)/5 = 0.28 \\
x_3^{(2)} &= (3-0.4-2\cdot 0.28)/8 = 0.255
\end{aligned}$$




Kuva 6: Newtonin menetelmän eteneminen yksiulotteisesti

## 3.2 Epälineaariset yhtälöryhmät

### 3.2.1 Yksiulotteinen Newtonin menetelmä

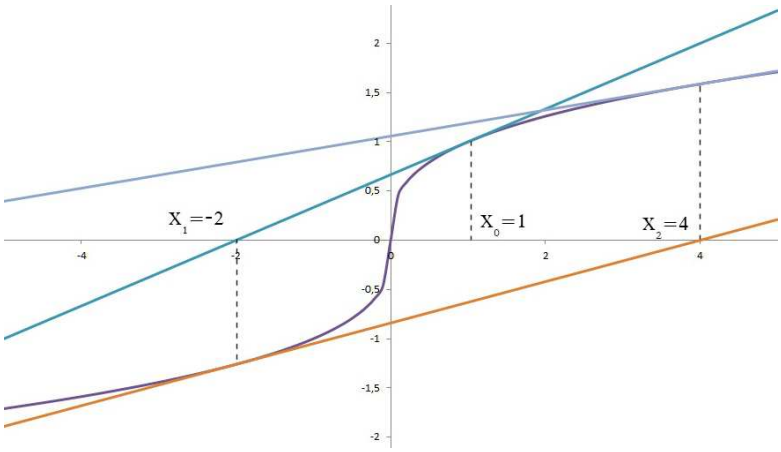
Newtonin menetelmä on Sir Isaac Newtonin kehittämä iteratiivinen laskumenetelmä funktioiden nollakohtien etsimiseksi. Se tunnetaan myös Newton-Raphsonin menetelmänä ja sitä sovelleteen usein erityisesti epälineaarisissa yhtälöissä. Sitä voidaan käyttää yhtälöiden ratkaisemiseen muuttamalla yhtälö $f(x) = a$ muotoon $f(x) - a = 0$, jolloin funktion $f(x) - a = 0$ nollakohta on samalla myös yhtälön $f(x) = a$ ratkaisu. Edellytyksenä menetelmän käytölle on kuitenkin, että funktio on juuren läheisyydessä kahdesti derivoituva ja että $f'(x) \neq 0$. Menetelmä lähtee liikkeelle alkuarvauksesta, jonka arvoa parannetaan iteratiivisesti vaihe vaiheelta. Graafisesti kuvan 6 mukaan tarkasteltuna arvon parantaminen tapahtuu seuraavasti. Funktiolle $f(x)$ lasketaan tangentti $g(x)$ alkuarvauspisteessä $x_0$. Tangentti voidaan laskea yhtälön (11) mukaisesti

$$g(x) - f(x_0) = f'(x_0)(x - x_0) \tag{11}$$

ja saadaan

$$g(x) = f'(x_0)(x - x_0) + f(x_0). \tag{12}$$

Lähtöarvauksen parantamiseksi täytyy saada selville tangenttiyhtälön $g(x)$ nollakohta, joten yhtälöön (12) voidaan asettaa $g(x) = 0$ ja saadaan




Kuva 7: Lukujonon hajaantuminen

$$x = x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Saatu luku $x_1$ on uusi ja samalla todennäköisesti myös tarkempi approksimaatio funktion $f(x)$ nollakohdalle. Approksimaation parantamiseksi iteraatio toistetaan ja lasketaan nollakohta funktion $f(x)$, kohdan $x_1$ tangentille. Samaa toistetaan kunnes $x_{n+1} - x_n < \textit{haluttu tarkkuus}$. Yleisesti iteraation eteneminen voidaan esittää seuraavassa muodossa.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{13}$$

Newtonin menetelmässä käytetty alkuarvaus $x_0$ voi olla satunnainen, mutta menetelmä toimii sitä varmemmin mitä lähempänä alkuarvaus on funktion nollakohtaa. Esimerkiksi kuvan 7 mukaisessa graafissa alkuarvaus on liian kaukana oikeasta juuresta ja yhtälön (13) mukaan laskettu lukujono ei suppene kohti funktion f(x) oikeaa juurta.

Menetelmän toimivuus on siis hyvin riippuvainen siitä, mikä alkuarvaus juurelle asetetaan. Alkuarvon asettaminen on usein myös menetelmän soveltamisen hankalin vaihe, etenkin tietokoneella sovellettuna ja epälineaaristen yhtälöiden tapauksessa. Epälineaaristen yhtälöiden kohdalla alkuarvauksen asettamisen tekee hankalaksi se, että niillä on hyvin tavallisesti useampi kuin yksi juuri ja jokaista juurta kohden tarvitaan vähintään yksi alkuarvaus. Yksi alkuarvaus jokaista juurta kohden ei epälineaarisilla yhtälöillä kuitenkaan aina riitä, sillä kaksi eri alkuarvausta voi johtaa samaan juureen. Hyvien alkuarvausten saamiseksi voidaan kuitenkin soveltaa mui-

ta, toiminnaltaan varmempia, mutta hitaampia laskumenetelmiä, kuten esimerkiksi jyrkän kasvun menetelmää.

### 3.2.2 Moniulotteinen Newtonin menetelmä

Edellä Newtonin menetelmää sovellettiin yhden muuttujan yhtälöissä. Sitä voidaan kuitenkin käyttää myös useamman muuttujan yhtälöiden ja yhtälöryhmien ratkaisuissa. Oletetaan esimerkiksi, että funktio $F(x) = 0$ on useamman muuttujan yhtälöryhmä

$$\begin{aligned} f_1(x_1, x_2, ..., x_n) &= 0 \\ f_2(x_1, x_2, ..., x_n) &= 0 \\ \vdots \qquad\qquad &\vdots \quad \vdots \\ f_n(x_1, x_2, ..., x_n) &= 0 \end{aligned} \tag{14}$$

, jonka nollakohtaa approksimoidaan Newtonin menetelmällä. Moniulotteisen nollakohdan löytämiseksi yhtälö (12) on kuitenkin täydennettävä moniulotteiseen muotoon, eli jokaiselle funktiolle lasketaan tangettiviivan sijasta tangenttitaso ja funktiosta on laskettava tangentti sen jokaisen muuttujan mukaan.

$$g_i(x) = f_i(x^{(k)}) + \frac{\partial f_i}{\partial x_1}\bigg|_{x^{(k)}} (x_1 - x_1^{(k)}) + \frac{\partial f_i}{\partial x_2}\bigg|_{x^{(k)}} (x_2 - x_2^{(k)}) + \cdots + \frac{\partial f_i}{\partial x_n}\bigg|_{x^{(k)}} (x_n - x_n^{(k)}), \qquad i = 1, ..., n$$

Kyseinen yhtälö voidaan esittää myös muodossa [17]

$$G_k(x) = f(x^{(k)}) + \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} (x - x^{(k)}) = f(x^{(k)}) + J(x^{(k)})(x - x^{(k)}) \tag{15}$$

, missä matriisi $J(x)$ tunnetaan Jacobian matriisina. Tangenttitasojen yhtymäkohtien $x$ selvittämiseksi 0-tasossa voidaan asettaa $G_k(x) = 0$, ja yhtälö (15) saa muodon

$$x = x^{(k)} - \frac{f(x^{(k)})}{J(x^{(k)})}$$

ja yleisemmin pätevänä iteraatioyhtälönä se voidaan esittää muodossa

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{J(x^{(k)})}. \tag{16}$$

Yhtälön (16) mukaan laskettuna Jacobian matriisista täytyisi kuitenkin laskea käänteismatriisi, joka on aikaa vievää ja yhtälö kannattaa ennemmin kertoa Jacobian matriisilla.

$$J(x^{(k)})(x^{(k+1)} - x^{(k)}) = -f(x^{(k)}) \tag{17}$$

Saatu yhtälöryhmä (17) on kätevämpi ja nopeampi ratkaista, jos yhtälöön sijoitetaan

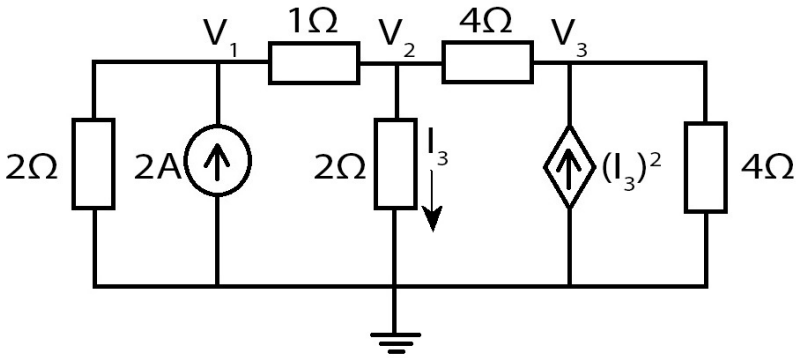
$$y^{(k)} = x^{(k+1)} - x^{(k)} \tag{18}$$

ja yhtälöryhmä (17) saa muodon

$$J(x^{(k)})(y^{(k)}) = -f(x^{(k)}). \tag{19}$$

Kyseessä on lineaarinen yhtälöryhmä, joka voidaan ratkaista esimerkiksi jotakin suoraa menetelmä, kuten Gaussin eliminointia käyttäen. Yhtälöryhmän ratkaisun jälkeen saatu tulos $y^{(k)}$ voidaan sijoittaa yhtälöön (18) ja vaiheen $(k+1)$ approksimaatio saadaan lasketuksi. Tämän jälkeen tulokseksi saatu approksimaatio sijoitetaan yhä uudelleen yhtälöön (17), kunnes approksimaatio on niin tarkka, kuin on tarpeen.

Teoreettisesti tarkasteltuna Newtonin menetelmä saattaa vaikuttaa hyvinkin monimutkaiselta menetelmältä epälineaaristen yhtälöryhmien ratkaisemiseksi. Käytännössä Newtonin menetelmä ei kuitenkaan ole kovinkaan monimutkainen ja pieni aiheen mukainen esimerkki osoittaa sen parhaiten. Oletetaan esimerkiksi, että ratkaistavana on kuvan 8 mukaisesta epälineaarisesta piiristä muodostettu epälineaarinen yhtälöryhmä.




Kuva 8: Epälineaarinen piiri

$$f(v_1, v_2, v_3, i_3) = \begin{cases} \frac{v_2 - v_1}{1} + \frac{v_1}{2} - 2 & = 0 \\ \frac{v_2 - v_3}{4} + \frac{v_2 - v_1}{1} + \frac{v_2}{2} & = 0 \\ \frac{v_3 - v_2}{4} + \frac{v_3}{4} - i_3^2 & = 0 \\ \frac{v_2}{2} - i_3 & = 0 \end{cases}$$

Yhtälöryhmän linearisoimiseksi on yhtälöryhmästä laskettava Jacobian matriisi, joka voidaan laskea yhtälön (15) mukaisesti ja Jacobian matriisin yleiseksi muodoksi saadaan

$$J(x^{(n)}) = \begin{pmatrix} 1.5 & -1 & 0 & 0 \\ -1 & 1.75 & -0.25 & 0 \\ 0 & -0.25 & 0.5 & -2I_3 \\ 0 & 0,5 & 0 & -1 \end{pmatrix}.$$

Ennen Newtonin menetelmän soveltamista on ratkaisulle kuitenkin vielä asetettava alkuarvaus, joka on hyvä asettaa nollapisteeseen

$$x^{(0)} = (0, 0, 0, 0).$$

Tämän jälkeen ensimmäisen approksimaation laskemiseen voidaan käyttää yhtälöä

(19) .

$$J(x^{(0)})y^{(0)} = -f(x^{(0)})$$

$$\begin{pmatrix} 1.5 & -1 & 0 & 0 \\ -1 & 1.75 & -0.25 & 0 \\ 0 & -0.25 & 0.5 & -2 \cdot 0 \\ 0 & 0.5 & 0 & -1 \end{pmatrix} \begin{pmatrix} y_1^{(0)} \\ y_2^{(0)} \\ y_3^{(0)} \\ y_4^{(0)} \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Kyseisen yhtälöryhmän ratkaisuksi saatu tulos,

$$y^{(0)} = \begin{pmatrix} 2.260870, & 1.391304, & 0.695652, & 0.695652 \end{pmatrix}$$

sijoitetaan lausekkeeseen (18) ja ensimmäisen approksimaation arvoksi saadaan,

$$x^{(1)} = x^{(0)} + y^{(0)} = \begin{pmatrix} 2.260870, & 1.391304, & 0.695652, & 0.695652 \end{pmatrix}.$$

Saatu tulos voidaan sijoittaa uudelleen yhtälöön (19)

$$J(x^{(1)})y^{(1)} = -f(x^{(1)})$$

ja toisen approksimaation arvoksi saadaan

$$x^{(2)} = x^{(1)} + y^{(1)} = (2.525094,\ 1.787640,\ 2.413108,\ 0.893820).$$

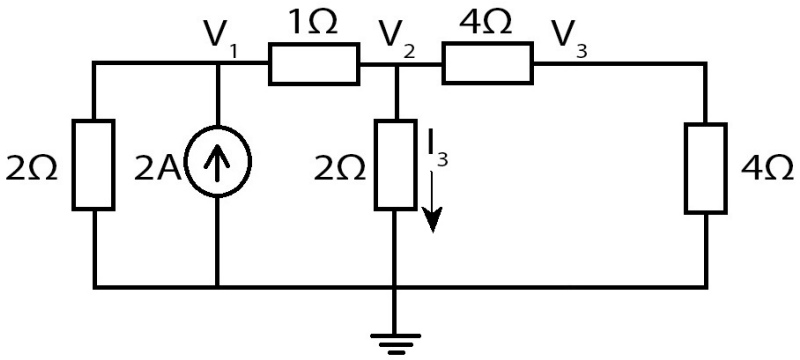
Tätä voidaan toistaa yhä uudelleen, mutta jo neljännessä iteraatiossa saatu tulos

$$x^{(4)} = (2.55,\ 1.83,\ 2.58,\ 0.91)$$

, on lähtöarvojen tarkkuuksien mukainen.

### 3.2.3 Piirin linearisointi

Luvussa 3.2.2 Newtonin menetelmää käytettiin epälineaaristen yhtälöryhmien ratkaisussa. Yhtälöryhmien ratkaisussa tarvittiin Jacobin matriisia, joka voitiin laskea epälineaarisesta yhtälöryhmästä. Piirin linearisointi vastaa Newtonin menetelmää lähes täysin, mutta piirin linearisoinnissa Jacobian matriisia ei lasketa epälineaarisesta yhtälöryhmästä, vaan se muodostetaan linearisoidusta piiristä. Menetelmä lähtee liikkeelle Newtonin menetelmän tavoin alkuarvauspisteestä ja piiri linearisoidaan kyseisessä pisteessä. Esimerkiksi jos tarkastellaan kuvan 8 mukaisesta piiriä ja sen ainoata epälineaarista komponenttia, virtariippuvaista virtalähdettä, jonka virtaa $i_3^2$ kuvaava yhtälö voidaan linearisoida yhtälön (12) mukaisesti. Tulokseksi saatu tangenttiyhtälö $g(x) = 2x_0(x - x_0^2) + x_0$ kuvaa virtalähteen käyttäytymistä pisteen $x_0 = i_3^{(0)}$ läheisyydessä. Luvun 3.2.2 mukaisessa esimerkissä alkuarvaus asetettiin kyseisen arvon kohdalla arvoon $i_3^{(0)} = 0$. Asettamalla nyt sama alkuarvaus saadaan tangenttiyhtälöksi $g(x) = 0$, eli pisteessä $i_3^{(0)} = 0$ linearisoitu piiri on kuvan 9 mukainen.




Kuva 9: Kuvan 8 mukainen piiri linearisoituna pisteessä $i_3^{(0)} = 0$

Piirin linearisoinnin tuloksena saadusta lineaarisesta piiristä voidaan muodostaa

lineaarinen yhtälöryhmä

$$f(v_1, v_2, v_3, i_3) = \begin{cases} \frac{v_2 - v_1}{1} + \frac{v_1}{2} - 2 & = 0 \\ \frac{v_2 - v_3}{4} + \frac{v_2 - v_1}{1} + \frac{v_2}{2} & = 0 \\ \frac{v_3 - v_2}{4} + \frac{v_3}{4} & = 0 \\ \frac{v_2}{2} - i_3 & = 0 \end{cases}$$

, josta voidaan havaita, että se vastaa täysin luvussa 3.2.2 esitetyn esimerkin ensimmäisen approksimaatiovaiheen mukaista yhtälöryhmää. Yhtälöryhmän ratkaisusta saadaan ensimmäisen iteraation arvo $i_3^{(1)}$ ja piiri voidaan linearisoida saadulla arvolla taas uudelleen. Tätä toistetaan yhä uudelleen kuten Newtonin menetelmässä, kunnes tulos on riittävän tarkka.

### 3.2.4 Jyrkän kasvun menetelmä (Steepest Descent method)

Piirin linearisointi ja Newtonin menetelmä ovat, sekä nopeita, että tarkkoja menetelmiä epälineaaristen yhtälöryhmien ratkaisussa. Ne ovat kuitenkin tarkkuudestaan ja nopeudestaan huolimatta toiminnaltaan epävarmoja. Niiden toiminnan varmuus riippuu hyvin pitkälti alkuarvauksesta ja mitä lähempänä alkuarvaus on ratkaisua, sitä varmempia ne toiminnaltaan ovat. Joskus alkuarvaus voi olla liian huono ratkaisun löytämiseksi ja sen takia ne tarvitsevatkin tuekseen laskentamenetelmän, jota voidaan käyttää paremman alkuarvauksen tuottamiseen. Tällaiseksi menetelmäksi sopii hyvin jyrkän kasvun menetelmä. Jyrkän kasvun menetelmä ei kuitenkaan ole toiminnaltaan yhtä nopea ja yksinkertainen ratkaisumenetelmä kuin Newtonin menetelmä on ja sen takia alkuarvausta kannattaakin tarkentaa sillä vain parin iteraation verran. Nopean kasvun menetelmä lähtee Newtonin menetelmän tavoin liikkeelle alkuarvauksesta, mutta ratkaisua ei lähdetä etsimään suoraan yhtälöryhmästä (14) vaan yhtälöryhmästä johdetusta funktiosta [18]

$$g(x_1, x_2, ..., x_n) = \sum_{i=0}^{n} [f_i(x_1, x_2, ..., x_n)]^2 \tag{20}$$

, jonka nollakohta on samassa pisteessä kuin yhtälöryhmän (14) ratkaisu. Nollakohdan selvittämiseksi on ensin laskettava suunta, johon funktion (20) arvo lähtee pienenemään jyrkimmin. Suunta saadaan laskettua funktion gradientin avulla.

$$\nabla g(x) = \left( \frac{\partial g(x)}{\partial x_1}, \frac{\partial g(x)}{\partial x_2}, ..., \frac{\partial g(x)}{\partial x_\mathrm{n}} \right)^t$$

Funktion gradientti kertoo kuitenkin vasta suurimman muunnoksen suunnan ja sen takia on vielä laskettava kuinka suurella kertoimella $\alpha > 0$ alkuarvauksen arvoa kannattaa muuttaa.

$$x^{(n+1)} = x^{(n)} - \alpha z = x^{(n)} - \alpha \frac{\nabla g(x^{(n)})}{|\nabla g(x^{(n)})|} \tag{21}$$

Kyseisen $\alpha$ muuttujan arvo voitaisiin laskea suoraan yhtälöstä

$$h(\alpha) = g\left( x^{(n)} - \alpha \frac{\nabla g(x^{(n)})}{|\nabla g(x^{(n)})|} \right) \tag{22}$$

, mutta arvon suora määrittely differentiaalin avulla olisi kuitenkin todennäköisesti liian työlästä. Sen takia funktion (20) käyttäytymistä alkuarvauksen läheisyydessä kannattaa vain approksimoida. Approksimointi onnistuu parhaiten Newtonin interpolaation toisen asteen polynomin avulla

$$P(\alpha) = g(x^{(n)}) + h_1\alpha + h_3\alpha(\alpha - \alpha_2) \tag{23}$$

, missä

$$\begin{aligned} h_1 &= \frac{g(x^{(n)} - \alpha_2 z) - g(x^{(n)} - \alpha_1 z)}{\alpha_2 - \alpha_1} \\ h_3 &= \frac{h_2 - h_1}{\alpha_3 - \alpha_1} \\ h_2 &= \frac{g(x^{(n)} - \alpha_3 z) - g(x^{(n)} - \alpha_2 z)}{\alpha_3 - \alpha_2} \end{aligned}$$

ja joka interpoloi funktion (22) pisteissä $\alpha_1, \alpha_2$ ja $\alpha_3$. Kyseiset kolme funktion (22) interpolaatiopistettä on valittava siten, että ehdot $\alpha_1 = 0$, $h(\alpha_3) < h(\alpha_1)$ ja $\alpha_2 = \alpha_3/2$ toteutuvat. Näin ollen $\alpha_1$ on aina arvoltaan 0 ja $\alpha_3$ on alkuarvaus, joka valitaan siten että ehto toteutuu $h(\alpha_3) < h(\alpha_1)$. Tämän jälkeen vielä lopuksi viimeinen kerroin $\alpha_2$ sijoitetaan arvojen $\alpha_1$, sekä $\alpha_3$ puoliväliin ja koska $\alpha_1 = 0$ niin voidaan suoraan laskea $\alpha_2 = \alpha_3/2$. Kyseiset arvot määrittelevät funktion (23) ja paras yhtälön (21) kertoimen $\alpha$ arvo on joko $\alpha_0$, jolla $P'(\alpha_0) = 0$ tai $\alpha_3$. Näistä kahdesta arvosta $\alpha_0$

ja $\alpha_3$ parempi on se, joka tuo yhtälölle (22) pienemmän arvon. Paremman arvon määrittelemisen jälkeen parempi sijoitetaan yhtälöön (21) ja iteraation seuraava arvo voidaan laskea. Tämän jälkeen lasketaan taas paras $\alpha$:n arvo, sekä sen avulla seuraavan iteraation arvo ja tätä toistetaan kunnes haluttu tarkkuus saavutetaan.

# 4 Piirin analysointi

Elektroniikkapiirien käyttäytymistä voidaan analysoida usealla eri tavalla. Näitä analysointitapoja ovat mm. tasavirta-, vaihtovirta-, transientti-, herkkyys- ja häiriöanalyysi. Nimensä mukaisesti jokainen näistä on kehitetty tietyn tilanteen simuloimista varten. Tässä työssä kehitetty ohjelma osaa soveltaa tasavirta-, vaihtovirta- ja transienttianalyysejä.

## 4.1 Tasavirta-, eli DC-analyysi

Analogisten elektroniikkapiirien käyttäytymisen simulointi aloitetaan tavallisesti aina tasavirta-analyysistä. Analyysin avulla saadaan selville piirin operaatiopiste. Tasavirta-analyysissä kaikki lähteet on tasavirtaisia, joten piirin jännitteet tai virrat eivät muutu ajan myötä ja energiaa varaavat komponentit voidaan korvata. Esimerkiksi kondensaattorin läpikulkeva virta on riippuvainen sen yli olevan jännitteen muutoksista

$$i = C\frac{dv}{dt} \tag{24}$$

ja koska jännite on aina vakio.

$$i = C\frac{dv}{dt} = C \cdot 0 = 0$$

Eli kondensaattorin läpi ei kulje koskaan virta ja se voidaan korvata virtalähteellä, joka ei tuota virtaa ollenkaan, eli $i = 0$ A. Induktoria sen sijaan ei voi korvata virtalähteellä, sillä sen yli oleva jännite on riippuvainen sen läpi kulkevan virran muutoksista

$$v = L\frac{di}{dt} \tag{25}$$

ja koska virta on aina vakio

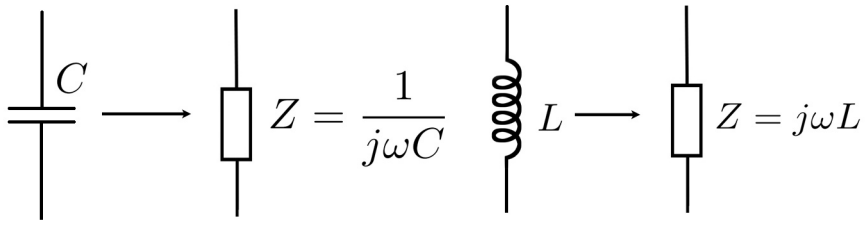
$$v = L\frac{di}{dt} = L \cdot 0 = 0$$

Eli induktorin yli oleva jännite on aina nolla ja se voidaan korvata jännitelähteellä, jonka yli oleva jännite on 0 V.

Energiariippuvaisten komponenttien korvaamisen jälkeen piiristä muodostetaan yhtälöt. Yhtälöt voidaan muodostaa esimerkiksi solmupisteanalyysin mukaisesti. Lineaarisesta piiristä muodostetut yhtälöt ovat lineaarisia ja niiden ratkaisemiseen voidaan käyttää sekä suoria, että iteratiivisia menetelmiä. Työssä toteutettu ohjelma ratkaisee lineaariset yhtälöryhmät tavallisesti suoria menetelmiä kuten Gaussin eliminointia ja LU-hajotelmaa soveltaen, mutta käyttää myös iteratiivisia menetelmiä, kun ratkaisun nopeus on ratkaisun tarkkuutta merkitsevämpi tekijä. Epälineaarisista piireistä muodostetut yhtälöt ratkaistaan sen sijaan vain iteratiivisia menetelmiä käyttäen. Ohjelmassa ne yritetään aluksi ratkaista Newtonin menetelmän avulla, mutta jos iteraatioissa saadut tulokset lähtevät hajaantumaan sovelletaan jyrkän kasvun menetelmää muutaman iteraation verran ja lopuksi arvoa tarkennetaan Newtonin menetelmän avulla.

## 4.2 Vaihtovirta- eli AC-analyysi

Tasavirta-analyysin jälkeen tiedossa on jo piirin operaatiopiste, eli piirin kaikkien kytkentäpisteiden jännitteet tunnetaan. Tämän jälkeen on usein vielä saatava selville piirin taajuusriippuvuus, eli miten vaihtovirtaisen ja vakaan signaalin taajuus vaikuttaa piirin käyttäytymiseen. Kyseinen analysointi onnistuu vaihtovirta-analyysin avulla. Vaihtovirta-analyysin suorittamisen ehtona on kuitenkin, että kaikkien piirissä olevien vaihtovirtaisten lähteiden signaalit ovat sinusoidaalisia ja niiden taajuus on sama. Vaihtovirta-analyysi aloitetaan luvussa 4.1 esitellyn tasavirta-analyysin mukaisesti. Operaatiopisteen laskemisen jälkeen esimerkiksi diodit korvataan operaatiopisteen mukaan lasketuilla sijaiskytkennöillä. Myös taajuusriippuvaiset komponentit korvataan kuvan 10 mukaisilla linearisoiduilla sijaiskytkennöillä.




(a) Kondesaattorin sijaiskytkentä (b) Induktorin sijaiskytkentä

Kuva 10: Taajuusriippuvaisten komponenttien sijaiskytkennät

Piensignaalimallin muodostamisen jälkeen piiriä analysoidaan, kuten tasavirta-analyysissä. Ainoana erona tavalliseen tasavirta-analyysin voidaan pitää sitä, että kun tasavirta-analyysissä yhtälöt muodostettiin kokonaan reaalisista arvoista niin vaihtovirta-analyysissä mukana on myös kompleksisia arvoja. Ohjelman toiminnan laajentaminen tasavirta-analyysistä vaihtovirta-analyysiin ei tuo ohjelman koodaukseen kovin suuria muutoksia. Vaihtovirta-analyysin toteuttamiseksi ohjelmaan on vain lisättävä mahdollisuus piensignaalimallin luomiseen sekä kompleksisten arvojen käsittelyyn.

## 4.3 Transienttianalyysi

Vaihtovirta- ja tasavirta-analyysissä analysoidaan miten piiri käyttäytyy vakaassa tilassa. Piiri ei kuitenkaan aina ole vakaassa tilassa. Piirissä voi esiintyä häiriösignaaleja, jotka häiritsevät ja muuttavat piirin käyttäytymistä. Vaikka häiriösignaaleja ei olisi, niin piiri ei kuitenkaan saavuta vakaata tilaa välittömästi virran kytkemisen jälkeen. Piirin suunnittelussa on otettava huomioon myös mahdolliset epävakaat tilat ja tutkittava miten piiri niissä käyttäytyy. Epävakaiden tilojen simulointiin käytetään aikakehitys eli transienttianalyysiä, joka tutkii miten piirin käyttäytyminen muuttuu aikakehityksen myötä. Resistiivisissä piireissä aikakehityksen tutkiminen on melko helppoa ja tavallisesti niiden aikakehitys voidaan hahmottaa jo nopealla päättelyllä, sillä niissä kaikki piirin arvot hetkellä $t$ ovat suoraan riippuvaisia lähteiden arvoista hetkellä $t$. Resistiivisillä piireillä ei kuitenkaan ole kovinkaan laaja sovellettavuus ja sen takia usein käytetään myös energiaa varaavia komponentteja, kuten kondesaattoreita ja induktoreita, joissa jännitteen ja virran välinen riippuvuus on lineaarisen sijasta differentiaalinen (24), (25). Kyseiset yhtälöt voidaan esittää myös integraalisessa muodossa

$$v(t) = \frac{1}{C}\int_{-\infty}^{t} i(t)dt \tag{26}$$

$$i(t) = \frac{1}{L}\int_{-\infty}^{t} v(t)dt \tag{27}$$

ja ne osoittavat, että kondensaattorin jännite, sekä induktorin virta hetkellä $t$ on riippuvainen kaikesta siitä, mitä on tapahtunut ennen hetkeä $t$. Integraali on line-

aarinen operaatio ja yhtälöiden (26), sekä (27) integraalit voidaan jakaa osiin. Jakamalla integraalin esimerkiksi väleille $[-\infty, t_0]$ ja $[t_0, t]$ voidaan yhtälöt (26) ja (27) kirjoittaa muotoon.

$$\begin{aligned} v(t) &= \frac{1}{C}\left(\int_{-\infty}^{t_0} i(t)dt + \int_{t_0}^{t} i(t)dt\right) \\ i(t) &= \frac{1}{L}\left(\int_{-\infty}^{t_0} v(t)dt + \int_{t_0}^{t} v(t)dt\right) \end{aligned}$$

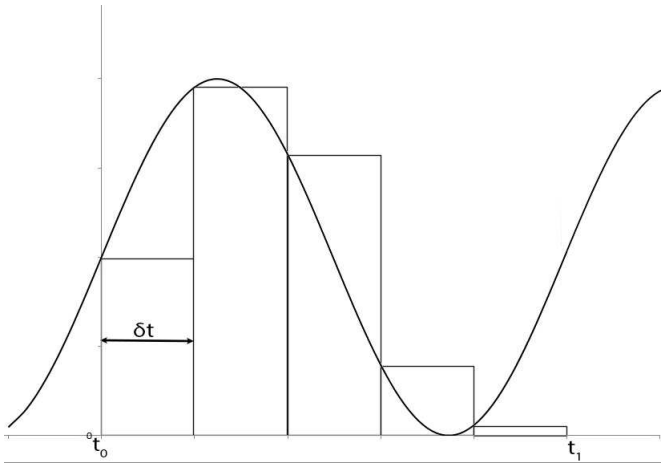
Kyseisissä yhtälöissä välin $[-\infty, t_0]$ määrätty integraali laskee esimerkiksi kondensaattorille, mikä jännite kondensaattorissa on hetkellä $t_0$. Eli välin $[-\infty, t_0]$ määrätty integraali voidaan korvata muuttujalla $v(t_0)$, jos kyseisen hetken jännite tunnetaan. Sama pätee myös induktorin virralle ja saadaan

$$v(t) = v(t_0) + \frac{1}{C}\int_{t_0}^{t} i(t)dt \tag{28}$$

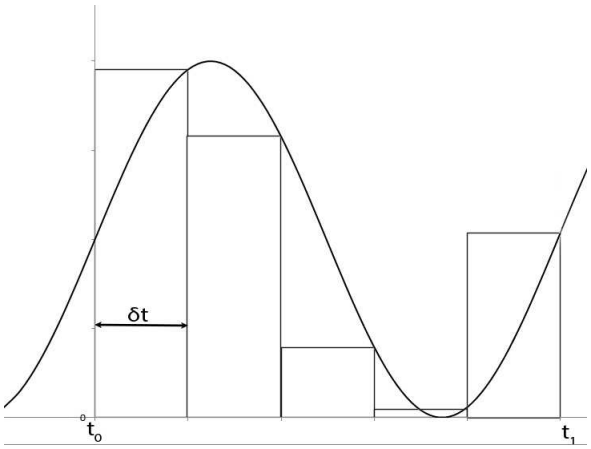
$$i(t) = i(t_0) + \frac{1}{L}\int_{t_0}^{t} v(t)dt \tag{29}$$

Hetken $t_o$ arvoja kutsutaan alkuarvoiksi ja tavallisesti ohjelman käyttäjä asettaa kyseiset arvot tai sitten ohjelma asettaa ne piirin operaatiopisteen arvojen mukaan. Alkuarvojen laskemisen tai asettamisen jälkeen yhtälöissä (28), (29) on kummassakin vielä yksi tuntematon arvo, joka on määrätyllä integraalilla määriteltävä. Integraalin tarkka laskeminen symbolisesti on haastavaa etenkin tietokoneella ja usein muutenkin mahdotonta. Tietokoneella määrätyn integraalin arvoa kannattaakin vain approksimoida mahdollisimman tarkasti. Approksimointi onnistuu numeerisen integroinnin avulla. Numeeriseen integrointiin on olemassa useita eri menetelmiä ja niiden tarkkuudet vaihtelevat funktion mukaan. Tärkeätä onkin valita aina kullekin funktiolle parhaiten sopiva eli tarkin menetelmä.

Transienttianalyysi aloitetaan tasavirta-analyysillä, jos ohjelman käyttäjä ei ole asettanut alkuarvoja. Operaatiopisteen laskemiseksi kaikki energiariippuvaiset komponentit korvataan tasavirta-analyysin mukaisilla sijaiskytkennöillä. Operaatiopisteen laskemisella saadaan selville piirin alkuarvot eli yhtälöiden (28), (29) arvot $v(t_0)$, $i(t_0)$ tunnetaan. Alkuarvojen laskemisen jälkeen piirin energiaa varaavat kom-




Kuva 11: Eulerin eksplisiittinen menetelmä




Kuva 12: Eulerin implisiittinen menetelmä

ponentit korvataan numeerisen integrointimenetelmän mukaisilla sijaiskytkennöillä ja hetken $t_0 + \delta t$ arvojen approksimaatio voidaan laskea. Tämän jälkeen sijaiskytkennät suoritetaan ajankohdan $t_0 + \delta t$ arvojen mukaisesti ja hetken $t_0 + 2\delta t$ arvot voidaan laskea. Näin jatketaan kunnes $t_0 + n\delta t$ on määrätyn integraalin ylärajan mukainen.

### 4.3.1 Eulerin menetelmät

Eulerin menetelmiin perustuvat numeeriset integroinnit ovat yksinkertaisia ja ne antavat tarkan tuloksen vain vakiofunktioilla. Eulerin menetelmät perustuvat derivaatan perusmääritelmään.

$$\frac{dy(t)}{dt} = \lim_{h \to 0} \frac{y(t+h) - y(t)}{h}$$

Muuttamalla raja-arvo $h$ nollasta poikkeavaksi voidaan derivaatan perusmääritelmää approksimoida hieman

$$\frac{dy(t)}{dt} \approx \lim_{h \to \delta t} \frac{y(t+h) - y(t)}{h} \approx \frac{y(t+\delta t) - y(t)}{\delta t}$$

ja saadun yhtälön avulla funktion $y(t+\delta t)$ arvoa voidaan approksimoida, kun arvo $y(t)$ tunnetaan.

$$y(t+\delta t) \approx y(t) + \delta t \frac{dy(t)}{dt} \tag{30}$$

Menetelmä tunnetaan nimellä Eulerin eksplisiittinen menetelmä ja se etenee kuvan 11 mukaisesti. Yhtälöllä (30) voidaan siis approksimoida myös määrätyn integraalin arvoa hetkellä $t + \delta t$, mutta tällöin derivaatta $dy(t)/dt$ on integroitava funktio ja esimerkiksi kondensaattorin jänniteyhtälön (28) arvoa hetkellä $t + \delta t$ voidaan approksimoida yhtälöllä [9]

$$v(t+\delta t) = v(t) + \delta t \frac{i(t)}{C} \tag{31}$$

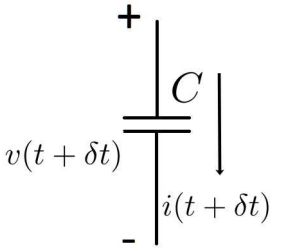
, missä $i(t)/C = dv(t)/dt$. Kyseisestä yhtälöstä (31) voidaan päätellä, että hetkellä $t + \delta t$ kondensaattori käyttäytyy kuin se olisi jännitelähde, jonka jännite on $v(t) + \delta t i(t)/C$. Tämän takia kondensaattori voidaan korvata kuvan 13b mukaisella jännitelähteellä. Myös induktori voidaan korvata, mutta se korvataan yhtälön (32) mukaisella ja kuvassa 13d esitetyllä virtalähteellä [9].

$$i(t+\delta t) \approx i(t) + \frac{\delta t}{L} v(t) \tag{32}$$

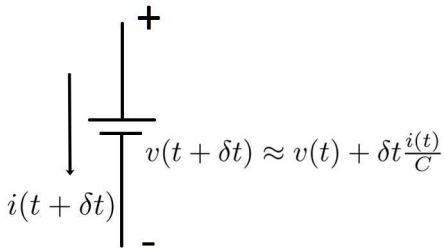
Sijaiskytkentöjen jälkeen piirin arvot voidaan laskea ja niistä voidaan johtaa approksimaation seuraavan vaiheen $t_0 + 2\delta t$ sijaiskytkennät.

$$\begin{aligned} v(t_0 + 2\delta t) &\approx v(t_0 + \delta t) + \delta t \frac{i(t_0 + \delta t)}{C} \\ i(t_0 + 2\delta t) &\approx i(t_0 + \delta t) + \delta t \frac{v(t_0 + \delta t)}{L} \end{aligned}$$

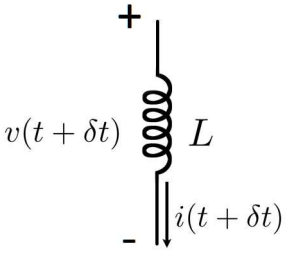
Tämän jälkeen piirin arvot analysoidaan uudelleen ja niistä asetetaan taas uudet sijaiskytkennät. Tätä toistetaan $n$ kertaa kunnes $t_0 + n\delta t = t_1$, eli määrätyn integraalin



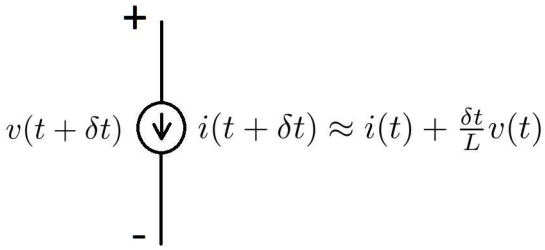

(a) Kondensaattori




(b) Kondensaattori sijaiskytkentä




(c) Induktori




(d) Induktorin sijaiskytkentä

Kuva 13: Kondensaattori ja induktori sekä niiden Eulerin eksplisiittisen menetelmän sijaiskytkennät

yläraja on saavutettu.

Eulerin eksplisiittisessä menetelmässä käytettävät sijaiskytkennät tuottavat joskus ongelmia analysointiin. Ongelmia syntyy esimerkiksi, jos monta silmukkaan kytkettyä kondensaattoria korvataan jännitelähteillä, jolloin jännitelähteet käytännössä aiheuttavat oikosulun. Kuvan 12 mukaisesti etenevä Eulerin implisiittinen menetelmä tuo tällaisiin tilanteisiin korjauksen, sillä implisiittisessä menetelmässä approksimointi suoritetaan yhtälön (30) sijasta yhtälöllä

$$y(t+\delta t) \approx y(t) + \delta t \frac{dy(t+\delta t)}{dt}$$

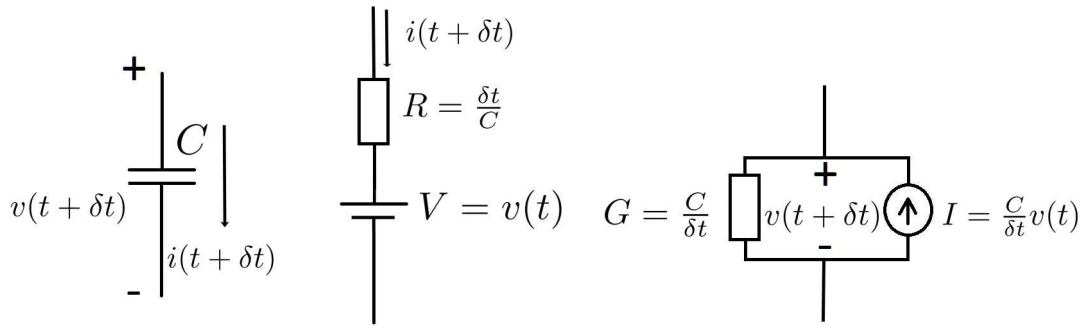
, josta voidaan edelleen johtaa kondensaattorin

$$v(t+\delta t) \approx v(t) + \delta t \frac{i(t+\delta t)}{C}$$

ja induktorin sijaiskytkentöjä kuvaavat yhtälöt [9].

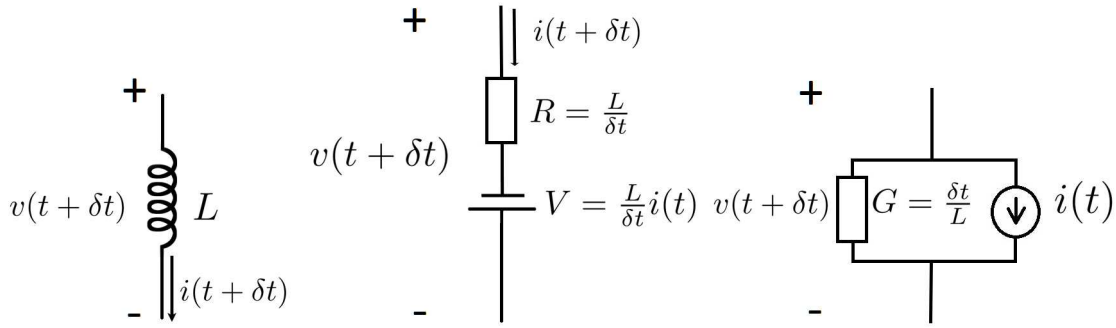
$$i(t+\delta t) \approx i(t) + \delta t \frac{v(t+\delta t)}{L}$$

Yhtälöistä voidaan havaita, että implisiittisessä menetelmässä esimerkiksi konden-




(a) Kondensaattori (b) Kondensaattorin Thevenin malli (c) Kondensaattorin Nortonin malli

Kuva 14: Kondensaattori ja sen Eulerin implisiittisen menetelmän mukaiset sijaiskytkennät



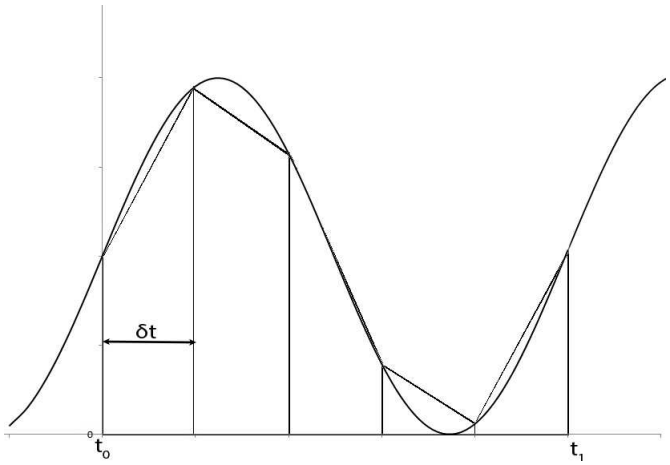

(a) Induktori (b) Induktorin Thevenin malli (c) Induktorin Nortonin malli

Kuva 15: Eulerin implisiittisen menetelmän sijaiskytkennät induktorille

saattorin jännite $v(t+\delta t)$ on riippuvainen saman ajankohdan $t+\delta t$ virrasta $i(t+\delta t)$, kun taas eksplisiittisessä menetelmässä ne ovat riippuvaisia eri ajankohdan arvoista. Tämän takia implisiittisessä menetelmässä kondensaattorin sijaiskytkentänä voidaan yhden jännitelähteen sijasta käyttää kuvan 14b mukaista kytkentää, missä on kytkettynä sarjaan, sekä jännitelähde $v(t)$, että vastus $\delta t/C$. Kyseiselle kytkennälle on olemassa sitä vastaava kuvan 14c mukainen Nortonin kytkentä. Vastaavat sijaiskytkennät induktorille ovat kuvissa 15b ja 15c.

### 4.3.2 Puolisuunnikassääntö

Eulerin menetelmät ovat yhden pisteen menetelmiä, eli määrätyn integraalin arvo approksimoidaan jokaisessa vaiheessa vain yhden funktion arvon avulla. Eksplisiittisessä käytettiin arvoa $y(t)$, kun taas implisiittisessä käytettiin arvoa $y(t+\delta t)$. Puolisuunnikassääntö on sen sijaan kahden pisteen menetelmä ja siinä approksimointiin

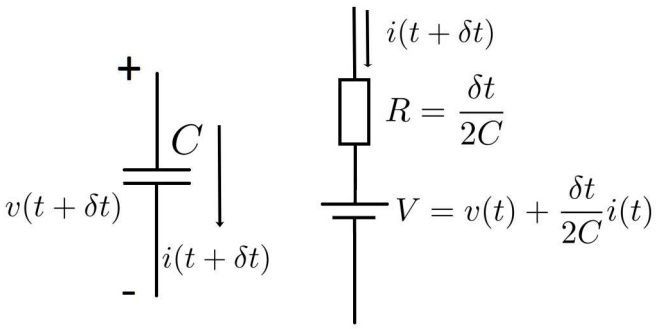



Kuva 16: Puolisuunnikassääntö

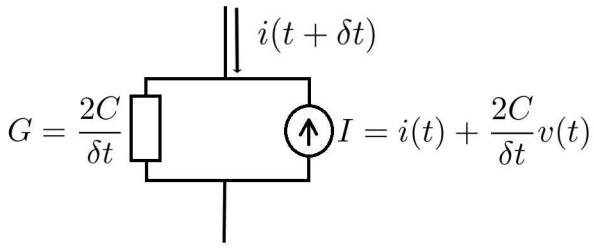
käytetään arvojen $dy(t)/dt$ ja $dy(t+\delta t)/dt$ keskiarvoa [9].

$$y(t+\delta t) \approx y(t) + \frac{\delta t}{2}\frac{dy(t)}{dt} + \frac{\delta t}{2}\frac{dy(t+\delta t)}{dt}$$

Kyseisestä yhtälöstä johdetut kuvien 17, 18 mukaiset sijaiskytkennät ovatkin ikäänkuin Eulerin eksplisiittisen ja implisiittisen menetelmän sijaiskytkentöjen yhdistelmä. Puolisuunnikassäännön mukainen integrointi etenee kuvan 16 mukaisesti ja kuvasta voidaan havaita, että approksimointi myötäilee hyvin funktiota. Puolisuunnikassääntö onkin tarkka menetelmä jaksollisille funktioille.

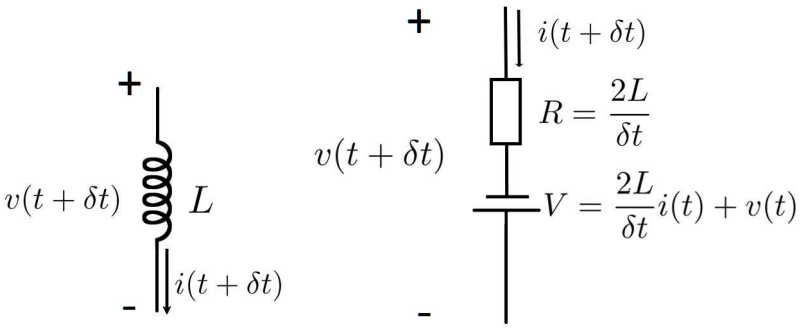



(a) Kondensaattori (b) Kondensaattorin Thevenin malli

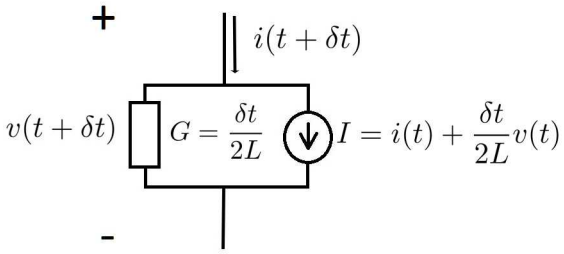



(c) Kondensaattorin Nortonin malli

Kuva 17: Kondensaattori ja sen puolisuunnikassäännön mukaiset sijaiskytkennät




(a) Induktori (b) Induktorin Thevenin malli




(c) Induktorin Nortonin malli

Kuva 18: Induktori ja sen puolisuunnikassäännön mukaiset sijaiskytkennät

# 5 Ohjelman toiminta

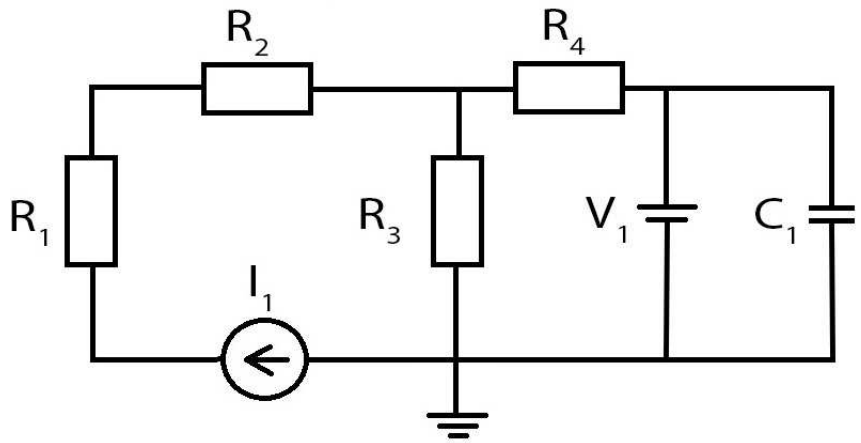
Oliokielellä kehitetty ohjelma koostuu nimensä mukaisesti oliosta. Oliot ovat ohjelman osia, jotka tallentavat tietoa ja vuorovaikuttavat keskenään. Ne muodostetaan useimmiten käytännön käsitteiden mukaan. Piirisimulaatio-ohjelmassa tällaisia käsitteitä ovat mm. kytkentäväli, kytkentäpiste ja komponentti. Tässä luvussa tutustutaankin minkälaisin olioluokin ja algoritmein tässä työssä kehitetty simulaatio-ohjelma on kehitetty.

## 5.1 Piirin topologia

Piirin kytkentöjen tutkiminen ja käsittely on ohjelman tärkeimpiä aihealueita, koska sitä sovelletaan kaikista eniten ohjelman suorittamisen aikana. Se on myös ohjelmanosa, joka kannattaa kehittää ohjelman matemaattisen koneiston ohella ensimmäisenä, sillä kaikki muut ohjelmanosat ovat niistä riippuvaisia. Kytkentöjen tutkimiseen on olemassa lukuisia eri menetelmiä, mutta valtaosa niistä on kuitenkin kehitetty juuri kytkentöjen tutkimista ja tulkitsemista eikä muokkaamista varten. Esimerkiksi mm. SPICEssäkin käytetty kytkentämatriisi [10] on käytännöllinen menetelmä, kun tarkoituksena on muodostaa solmupisteyhtälöitä. Yksinkertaistusalgoritmeissa se ei kuitenkaan ole kovin käytännöllinen, kun piirin kytkentöjä myös muunnellaan. Yksinkertaistamista varten täytyikin kehittää aivan oma menetelmä, jossa piirin kytkentöjä tulkitaan ja muokataan olion avulla. Kyseistä menetelmää voidaan soveltaa myös yhtälöiden muodostamisessa.

### 5.1.1 Kytkentämatriisi

Tietokoneohjelmissa piirin kytkentöjen tulkitsemiseen käytetään usein ns. kytkentämatriisia $M$, jossa on oma rivi $i$ jokaista piirin kytkentäpistettä $i$ ja oma sarake $k$ jokaista piirin komponenttia $e_k$ varten. Matriisin $M$ arvot $M_{ik}$ määräytyvät sen mukaan onko kyseessä kytkentäpisteeseen saapuva vai kytkentäpisteestä lähtevä



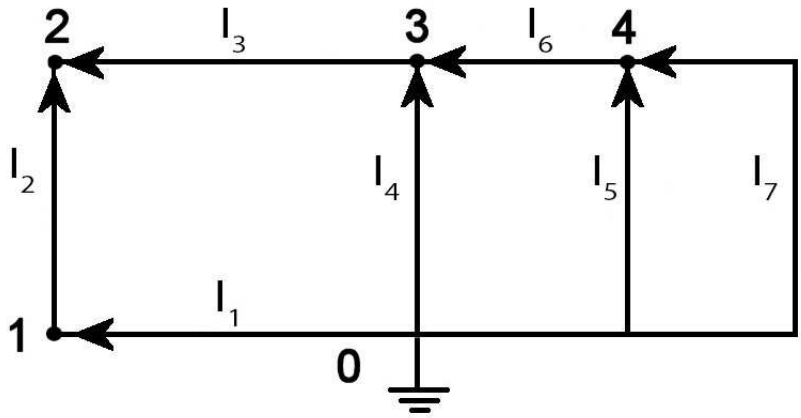

Kuva 19: RC-piiri

kytkentä [21].

$$M_{ik} = \begin{cases} +1 & \text{, jos kytkentä on pisteeseen saapuva} \\ -1 & \text{, jos kytkentä on pisteestä lähtevä} \\ 0 & \text{, jos kytkentää ei ole} \end{cases} \tag{33}$$

Näin ollen piirin jokaisesta kytkennästä on tehtävä päätös mihin pisteeseen se on saapuva ja mistä se on lähtevä. Passiivisilla komponenteilla kytkennän suunnalla ei ole mitään merkitystä. Lähteillä suunta sen sijaan kannattaa kuitenkin merkitä virtalähteen virransuunnan ja jännitelähteen napojen $- \rightarrow +$ mukaisesti. Käytännössä tämä onnistuu parhaiten suuntakytkentäkaavion avulla, jossa on määritelty jokaisen kytkennän suunta. Esimerkiksi kuvan 19 mukaisesta kytkentäkaaviosta muodostettu suunnattu kytkentäkaavio voisi olla kuvan 20 mukaisen suunnatun kytkentäkaavion mukainen, josta muodostettu kytkentämatriisi on:

$$M = \begin{bmatrix} I_1 & R_1 & R_2 & R_3 & V_1 & R_4 & C_1 \\ -1 & 0 & 0 & -1 & -1 & 0 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 1 \end{bmatrix}$$




Kuva 20: Kuvan 19 RC-piirin suunnattu kytkentäkaavio

### 5.1.2 Kytkentäoliot

Luvussa 5.1.1 esitelty kytkentämatriisi voidaan korvata oliokielessä olioluokan avulla. Tähän soveltuu esimerkiksi kuvan 21a mukainen olioluokka. Kyseinen olioluokka kehitettiin nimenomaan yksinkertaistusta varten ja sen avulla yksinkertaistusta voidaan nopeuttaa huomattavasti. Yksinkertaistus nopeutuu, sillä kyseinen olio ei sisällä nolla-alkioita kuten kytkentämatriisi. Kytkentämatriisin nolla-alkiot ovat oikeastaan turhia, sillä ne kertovat vain kytkennöistä, joita ei ole olemassa. Kytkentöjä käsitellään olioluokan avulla myös toisin, kun kytkentämatriisi käsittelee kytkentöjä kytkentäpistein ja komponentein, niin olioluokka *Kytkentäpisteet* käsittelee kytkentöjä pelkästään kytkentäpistein. Olioluokan *Kytkentäpisteet* attribuuttiin *viereisetpisteet* merkitään vain kaikki kyseisen kytkentäpisteen viereiset kytkentäpisteet ja esimerkiksi kuvan 20 mukaisen suunnatun kytkentäkaavion maadoituspisteestä muodostetun olion *viereisetpisteet* muuttuja olisi $\{1, 3, 4\}$. Kuvan 21b mukaista olioluokkaa apuna käyttäen kyseisten arvojen avulla voidaan viitata kytkentäväli-olioista muodostettuun taulukkoon ja tutkia esimerkiksi kytkentävälin $[0, 1]$ tai $[3, 0]$ ominaisuuksia. Vaikka kyseinen olioluokka on kehitetty yksinkertaistusta varten, voidaan sitä käyttää apuna myös solmupisteanalyysin mukaisia yhtälöitä muodostettaessa. Esimerkiksi KCL-yhtälöiden muodostaminen onnistuu hyvin yksinkertaisesti ja samankaltaisesti kuin käytännössä. Yhtälöiden muodostamisessa tarvitsee vain tutkia KCL-pisteiden attribuuttia *viereisetpisteet*, jonka arvoista saadaan selville miltä väleiltä kunkin KCL-pisteen yhtälöt muodostetaan. Kytkentävälien selvittämisen jälkeen niiltä voidaan tutkia niihin kytketyt komponentit ja

| Kytkentäpiste |
| --- |
| -piste |
| -viereisetpisteet |
| -jännite |
| -vaikuttava |
| -riippuvainen |

(a) Olioluokka kytkentäpiste

| Kytkentäväli |
| --- |
| -piste1 |
| -piste2 |
| -komponentit |
| -jännite |
| -virta |

(b) Olioluokka kytkentäväli

Kuva 21: Kytkentöjä käsittelevät olioluokat

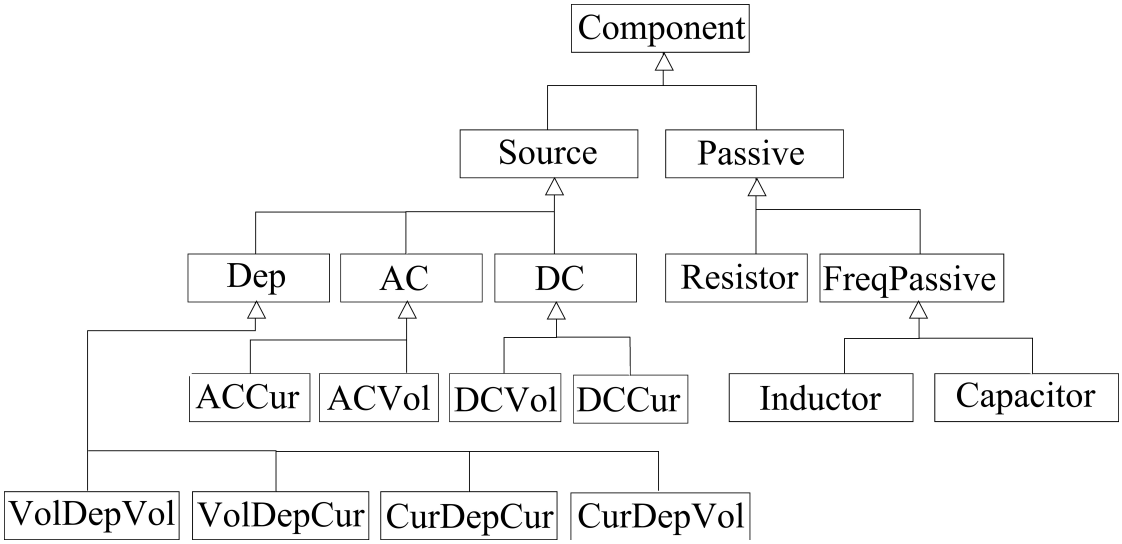
lopulta KCL-yhtälöt muodostetaan komponenttien arvojen mukaan.

## 5.2 Piirin komponentit ja niiden ominaisuudet

Piirin komponentit listataan ohjelmassa kytkentäväleittäin olioluokan *Kytkentäväli* attribuuttiin *komponentit*. Komponentteja on kuitenkin olemassa monenlaisia ja ohjelman on usein saatava selville, mikä komponentti on kyseessä. Joskus ohjelmalle riittää, että se saa selville onko kyseessä aktiivinen vai passiivinen komponentti. Kyseinen tieto riittää usein esimerkiksi, kun ohjelman täytyy saada selville onko jokin väli yksinkertaistettavissa. Tieto siitä, että onko komponentti passiivinen vai aktiivinen ei kuitenkaan ole aina riittävä, sillä esimerkiksi aktiivisia komponentteja on olemassa monenlaisia. Niitä ovat esimerkiksi kaikki lähteet, jotka taas voivat taas olla riippuvaisia, tasavirtaisia tai vaihtovirtaisia. Myös ohjelman komponentteja kuvaavat olioluokat muodostetaan näiden käsitteiden jakautumisen, sekä tarkentumisen mukaan ja tarkentamista jatketaan niin kauan kunnes käsite ei ole enää abstrakti. Kuvan 22 mukainen periytyvyyskaavio osoittaa miten ohjelman komponenttiluokat periytyvät. Komponenttiluokista kaikkein ylimpään tallennetaan kaikille komponenteille ominaiset piirteet kuten kytkentäväli, niiden yli oleva jännite, sekä niiden läpi kulkeva virta. Tietoa komponentin ominaisuuksista tarkennetaan sitä mukaan mitä tarkempi käsite on kyseessä.

## 5.3 Ohjelman matemaattiset muuttujat

Ohjelmassa käsitellään paljon matemaattisia muuttujia, joita käsitellään joko numeerisina tai symbolisina muuttujina. Kyseisten muuttujien käsittely numeerisesti




Kuva 22: Komponenttiluokkien periytyvyys

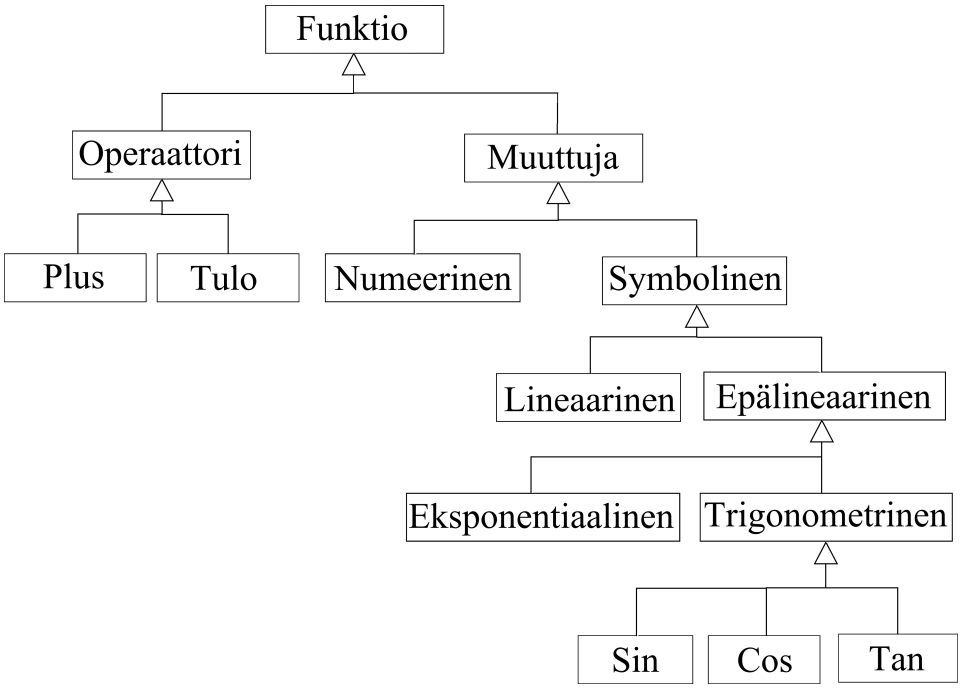
on yksinkertaista, sillä siihen riittää käytetyn ohjelmointikielen javan perusmuuttujat. Java ei kuitenkaan ole kovin matemaattinen kieli, eikä siinä ole perusmuuttujia lausekkeiden käsittelyä varten. Tämän takia lausekkeiden symboliseen käsittelyyn on luotava omat tietorakenteet.

Kuvan 23 mukainen matemaattisten olioluokkien periytyvyys luo mahdollisuuden käsitellä kaikkia ohjelmassa tarvittavia muuttujia. Kyseisen luokkakaavion olioluokkien avulla komponenttien arvoja voidaan käsitellä symbolisesti funktioina ja kyseiset funktiot voivat olla yksittäisiä lineaarisia tai epälineaarisia muuttujia, mutta ne voivat olla myös muuttujista ja operaattoreista muodostuvia kokonaisia lausekkeita. Mahdollisuus käsitellä kokonaisia lausekkeita symbolisesti helpottaa mm. symbolisen derivaatan laskemisessa, sillä kokonaisen lausekkeen derivointi määräytyy siinä olevien operaattorien ja muuttujien mukaan.

## 5.4 Ohjelman matemaattiset algoritmit

### 5.4.1 Gaussin eliminointialgoritmi

Liitteen A mukainen Gaussin eliminoinnin pohjalta kehitetty algoritmi on suhteellisen tarkka, sillä laskutoimituksen virhe syntyy vain pyöristysvirheistä. Pyöristysvirheetkin voidaan minimoida käyttäen skaalattua Gaussin eliminointia [5]. Tarkkuudestaan huolimatta Gaussin eliminoinnin pohjalta kehitetty algoritmi ei ole, joka tilanteessa se oikea laskentamenetelmä yhtälöryhmän ratkaisemiseksi, sillä se on




Kuva 23: Matemaattisten olioiden periytyvyys

laskentateholtaan melko vaativa. Algoritmi on vaativuudeltaan luokkaa $\mathcal{O}(n^3)$, joten laskentatoimituksien määrä kasvaa nopeasti muuttujien määrän kasvaessa.

### 5.4.2 LU-hajotelman algoritmi

Matriisien $L$ ja $U$ alkioiden laskemiseen on olemassa useita eri algoritmeja, kuten Doolittlen, Croutin ja Gaussin algoritmi. Näistä Gaussin algoritmi on useimmiten käytännöllisin, koska se sallii matriisien täydellisen muokkaamisen hajotelman laskemisen edetessä [12]. LU-hajotelman laskemiseen tarkoitettu Gaussin algoritmin eduksi voidaan laskea myös se, että sen suorittamiseen tarvitaan vain niitä algoritmeja, joita käytetään myös Gaussin eliminointialgoritmissa. Sekä itse hajotelman tekeminen, että myös yhtälöryhmän ratkaiseminen onnistuu Gaussin eliminoinnissa sovelletuilla eteenpäin eliminointi ja taaksepäin sijoitus algoritmeilla. LU-hajotelman algoritmi ei siis paljon poikkea Gaussin eliminoinnissa käytetyistä algoritmeista. Niillä on kuitenkin vaativuuseroja. Vaativuuseroja syntyy, jos ratkaistavana on kerroinmatriisiltaan $A$ sama yhtälöryhmä useaan otteeseen peräkkäin ja ainoana erona yhtälöryhmillä on, että vakioarvot $b$ muuttuvat. Tällöin LU-hajotelman matriiseja $L$, $U$ ei tarvitse laskea uudelleen ja algoritmin vaativuus laskee vaativuusluokasta $\mathcal{O}(n^3)$ luokkaan $\mathcal{O}(n^2)$, kun taas Gaussin algoritmin vaativuus on aina $\mathcal{O}(n^3)$. Piiria-

nalyysissä LU-hajotelman algoritmi voi siten nopeuttaa laskentaa, kun esimerkiksi jollekin virtalähteelle suoritetaan tasavirta-analyysi virtalähteen arvoja muuttaen.

### 5.4.3 Moniulotteisen Newtonin menetelmän algoritmi

Newtonin menetelmän liitteen B mukainen algoritmi ei ole rakenteeltaan kovin monimutkainen, mutta se vaatii toimiakseen melko kehittyneitä tietorakenteita. Esimerkiksi tarkoitukseen sopivat matemaattiset muuttujat, joita käyttäen funktioihin voi sijoittaa arvoja, sekä laskea niistä derivaattoja. Monet ohjelmointikielet tarjoavat tähän jo itsestään tarpeeksi kehittyneen matemaattisen koneiston. Kyseinen ohjelma kehitettiin kuitenkin javalla, josta ei sellaista itsestään löydy. Tarpeeksi kehittyneitä kirjastoja olisi kyllä ollut olemassa, mutta työn periaatteena oli alusta asti olla soveltamatta toisten kehittämiä algoritmeja ja mm. tähän tarkoitukseen sopivat matemaattiset muuttujat esiteltiin luvussa 5.3. Newtonin menetelmän vaativuus ja toiminnanvarmuus vaihtelee hyvin paljon riippuen ratkaistavista funktioista ja alkuarvauksesta. Teoreettisesti se on kuitenkin nopea menetelmä epälineaaristen yhtälöryhmien ratkaisuun.

## 5.5 Ohjelman piirialgoritmit

### 5.5.1 Solmupistealgoritmi

Solmupisteanalyysin yhtälöt voitaisiin muodostaa suoraan kytkentämatriisin (33) avulla.

$$MI(s) = 0 \tag{34}$$

Tavallisesti kytkentämatriisi korvataan kuitenkin solmupisteanalyysissä supistetulla kytkentämatriisilla, sillä kytkentämatriisiin on merkitty myös maadoitetun kytkentäpisteen kytkennät, eikä solmupisteanalyysissä maadoituspisteelle luoda KCL-yhtälöä. Kytkentämatriisin rivin 0 arvot ovat siten turhia ja kytkentämatriisista kannattaa poistaa maadoitetun pisteen kytkentöjä esittävä rivi ja luoda supistettu kytkentämatriisi $A$. Supistetun kytkentämatriisin avulla yhtälö (34) voidaan kirjoittaa muotoon

$$AI(s) = 0$$

, josta virtamuuttujat voidaan korvata jännitemuuttujilla sijoittamalla $I(s) = Yu + s = yA^T v + s$ ja saadaan [8]

$$AyA^T v = Yv = -AI(s) \tag{35}$$

, missä $Y = yA^T$ on admittanssimatriisi ja $I(s)$ on virtalähdevektori. Esimerkiksi jos oletetaan, että kuvan 19 mukaisessa piirissä kaikki lähteet ovat tasavirtaisia ideaalilähteitä ja yhtälöt muodostetaan kuvan 20 suuntakaavion muokaisesti, niin yhtälön (35) matriisit ja vektorit kirjoitettaisiin seuraavasti

$$\begin{aligned}
A &= \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 1 \end{bmatrix} \\
y &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/R_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/R_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/R_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/R_4 \end{bmatrix} \\
v &= \begin{bmatrix} V_1 & V_2 & V_3 & V_4 \end{bmatrix}^T \\
I(s) &= \begin{bmatrix} I_1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T
\end{aligned}$$

### 5.5.2 TICER-algoritmi

Tietokoneella suoritettavissa solmupisteanalyyseissä yhtälöt muodostetaan usein supistetun kytkentämatriisin avulla. Supistettu kytkentämatriisi taas muodostetaan piirin jokaisen kytkentäpisteen, sekä komponentin mukaan maadoitettua pistettä lukuun ottamatta. Kytkentämatriisilla muodostetuissa yhtälöryhmissä (35) ei ole muuttujina pelkästään solmupisteiden jänniteet, vaan jokaisen kytkentäpisteen jännitteet. Yhtälöryhmän muodostaminen ja ratkaiseminen onnistuu näinkin, mutta yhtälöryhmässä on tällöin paljon enemmän muuttujia, kuin mitä niitä olisi, jos

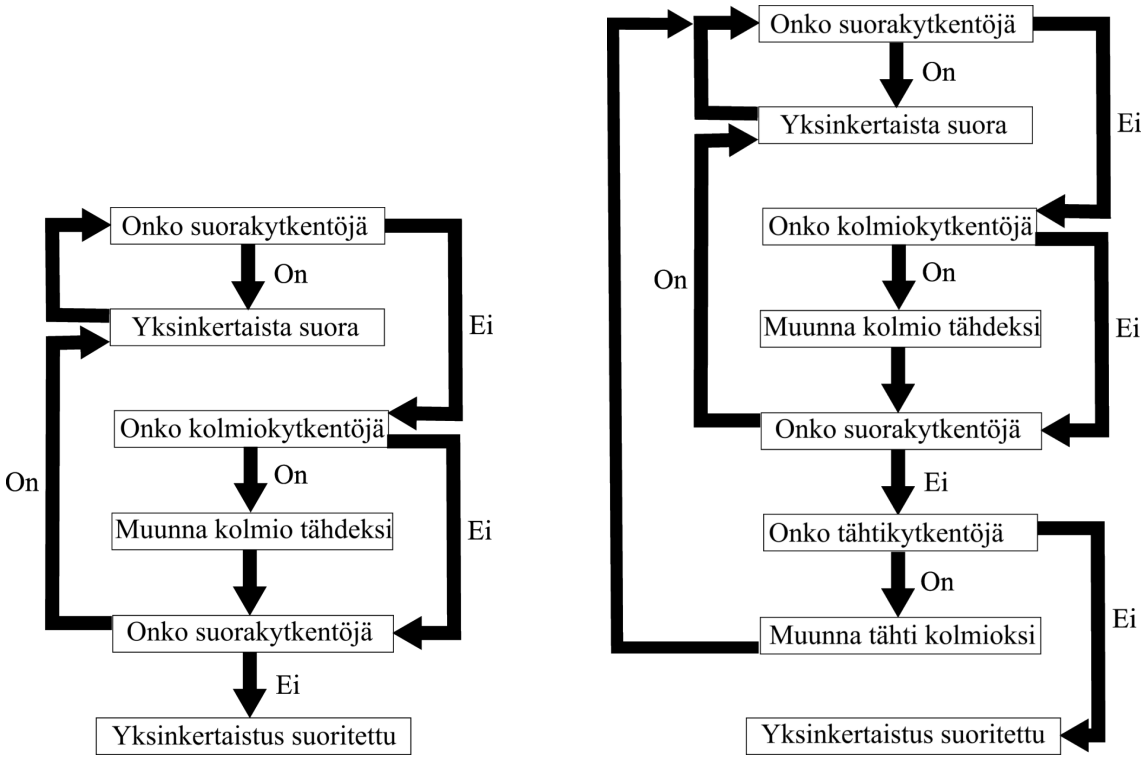
muuttujina olisi pelkästään solmupisteiden jännitteitä. Yhtälöryhmän (35) muuttujien vähentämiseksi on muodostettava kytkentämatriisi, johon on merkittynä lähinnä vain solmupisteitä koskevat kytkennät. Koko prosessi on oikeastaan yksinkertaistamista, sillä kaikki piirin sarjaankytketyt komponentit korvataan niitä vastaavilla teoreettisilla komponenteilla. Yksinkertaistamisprosessissa siis poistetaan kaikki kytkentäpisteet, jotka ovat kahden passiivisen sarjaan kytketyn komponentin yhteinen kytkentäpiste. Tällainen kytkentäpiste voidaan löytää kytkentämatriisista (33) tulkitsemalla ja jos ehto

$$\sum_{i=1}^{n} |M_{ik}| \leq 2 \tag{36}$$

toteutuu on kyseessä sarjakytkentäpiste. Tällainen sarjakytkentäpiste on siis melko yksinkertaista löytää, sillä yhtälön (36) mukaisesti ei tarvitse kuin tutkia onko kytkentämatriisin rivin summa pienempi tai yhtäsuuri kuin kaksi. Sarjakytkentäpisteiden löytämisen helppoudesta huolimatta aavistuksen hankalampaa on muokata kytkentämatriisia siten, että se vastaa yksinkertaistettua piiriä. Sitä paitsi pelkkä kytkentämatriisin muokkaaminen ei riitä, vaan myös kaikkia yhtälön (35) muita tekijöitä on muokattava. Muokkaaminen voitaisiin suorittaa suoraan matriiseihin, mutta koska ohjelma toteutetaan oliokielellä, on muokkaus kätevämpää suorittaa kuvan 21a kaavion mukaista olioluokkaa apuna käyttäen. Kyseisessä oliossa on attribuuttina listamuuttuja *viereisetpisteet*, johon merkitään vain kytkentäpisteen viereiset kytkentäpisteet. Listan laajuudesta voidaan jo päätellä onko kyseessä suorakytkentäpiste, sillä jos listassa on enintään kaksi tekijää, on kyseessä sarjakytkentäpiste. Myös tähtikytkentöjen olemassaolo voidaan päätellä listaobjektin laajuudesta. Muutos taulukosta olioon nopeuttaa etsintäprosessia taulukon vaativuusluokasta $\mathcal{O}(n^2)$ olion vaativuusluokkaan $\mathcal{O}(n)$. Kytkentöjen muokkaamisessa ei menetelmien välillä ilmene kovin suuria vaativuuseroja, mutta listaobjektien muokkaaminen on yleensä aavistuksen yksinkertaisempaa kuin kokonaisten taulukoiden muokkaaminen.

TICER yksinkertaistusalgoritmi vaatii toimiakseen paljon haasteellisia apualgoritmeja. Apualgoritmeja tarvitaan lähinnä piirin kytkentöjen tutkimiseen ja muokkaamisen. Niitä tarvitaan lisäksi niin paljon, että jos kaikki yksinkertaistuksessa piirin rakenteen tutkimiseen ja muokkaamiseen tarvittavat apualgoritmit kirjoitettaisiin suoraan samaan algoritmiin, niin se olisi noin 8 kertaa pitempi. Algoritmin

vaativuustaso on vaihteleva ja on hyvin riippuvainen piirin kytkentöjen monimutkaisuudesta, sekä siitä kuinka pitkälle piiri yksinkertaistetaan. Jos RC-piiristä poistetaan vain kaikki rinnan- ja sarjaankytketyt passiivikomponentit on algoritmi hyvin nopea, mutta jos myös piirin kolmiokytkentöjä muokataan, on algoritmi melko vaativa, sillä kolmiokytkentöjen etsiminen on melko työlästä. Kolmiokytkentöjä kannattaa kuitenkin muokata hyvin useissa eri tapauksissa. Esimerkiksi epälineaarisissa piireissä pitkälle yksinkertaistetun piirin analysointi on niin paljon nopeampaa, että yksinkertaistuksessa kannattaa käyttää myös kolmiokytkentöjen muokkausta. Algoritmi voidaan suorittaa joko kuvan 24a tai kuvan 24b mukaisien kaavioiden mukaisesti. Tavallisimmin yksinkertaistus suoritetaan kuvan 24a mukaisen kaavion mukaan, sillä se on nopeampi menetelmä eikä tähti-kolmioksi muunnos ole välttämätön vaihe yksinkertaistuksessa.




(a) Ilman tähti kolmioksi muunnosta (b) Tähti kolmioksi muunnos mukana

Kuva 24: TICER algoritmin etenemiskaavio

## 5.6 Piirialgoritmien nopeus

Työtä aloitettaessa suurena mielenkiinnon kohteena oli TICER-yksinkertaistuksella ja solmupisteyhtälöillä suoritettavien analyysien nopeudet verrattuna toisiinsa, sekä kuinka nopeita ne ovat yhteistyössä. Algoritmien nopeuksia verrattiin toisiinsa

ja vertailuarvoksi asetettiin ohjelman LTSpice nopeus. Työssä kehitetyt algoritmit pärjäsivätkin LTSpicen nopeudelle melko hyvin lineaarisissa piireissä, mutta epälineaarisissa piireissä ja transienttianalyyseissä LTSpice oli nopeampi. Epälineaarisissa piireissä ohjelman algoritmit suorittavat analyysin linearisoiduista piireistä ja yhtälöryhmät muodostetaan aina uudelleen jokaisen linearisoinnin jälkeen. Ohjelman toiminta nopeutuisi huomattavasti, jos piirin kytkentöjen tulkintavaihe voitaisiin jättää pois jokaisesta iteraatiosta.

Toisiinsa verrattuna ohjelman algoritmien nopeus vaihteli piiristä ja analyysimenetelmästä riippuen. Yksittäistä operaatiopistettä laskettaessa solmupisteanalyysin ja TICER-yksinkertaistuksen yhteistyö oli tavallisesti aavistuksen muita menetelmiä nopeampi, mutta graafien arvoja laskettaessa ero kasvoi usein jo melko suureksi. Yksistään TICER-algoritmilla superpositiopiireittäin suoritettavan analysoinnin nopeus oli hyvin pitkälle riippuvainen lähteiden määrästä, mutta kovin nopea se ei ollut edes yksilähteisissä piireissä. Taulukkoon 1 on koottu, joidenkin testipiirien tarkemmat arvot. Kyseisessä taulukossa $t_{LTS}$, $t_{TIC}$, $t_{NA}$ tarkoittavat LTSpicen, TICERin ja solmupisteanalyysin kuluttamaa aikaa. $t_{TICNA}$ on aika, joka kuluu kun TICER ja solmupisteanalyysi analysoivat piirin arvot yhteistyössä. Taulukon ensimmäiseen sarakkeeseen on merkitty analysoitavana oleva piiri ja siihen mahdollisesti tehdyt muutokset. Taulukon arvoista voidaan havaita, että TICER-algorimi ei vielä toiminut epälineaarisissa piireissä. Saman algoritmin muista arvoista voidaan kuitenkin myös päätellä, että sellaista TICER-algoritmia ei kannata välttämättä edes kehittää, sillä se häviää nopeudessa muille.

| Piiri | $t_{LTS}/t_{LTS}$ | $t_{TIC}/t_{LTS}$ | $t_{NA}/t_{LTS}$ | $t_{TICNA}/t_{LTS}$ |
|---|---|---|---|---|
| Liite C | 1 | 1,54 | 0,52 | 0,54 |
| Liite D | 1 | 4,1 | 1,26 | 0,61 |
| Liite D<br>.step param R4 1 1000 1 | 1 | 4,95 | 0,48 | 0,41 |
| Liite E | 1 | - | 1,69 | 1,34 |
| Liite E<br>.step param R1 1 100 1 | 1 | - | 3,14 | 1,16 |

Taulukko 1: Nopeuksien suhteellinen vertailu

# 6 Ohjelman käyttö

Elektroniikkapiirien suunnittelijoilla on ollut mahdollisuus käyttää erilaisia simulaattoreita jo vuosikymmenien ajan. Kyseiset simulaattorit suorittavat analyysin mitä erilaisimmin keinoin, ohjelmointimetodein ja ohjelmointikielin toteutettuna. Valtaosalla kyseisistä ohjelmista on kuitenkin yksi yhteinen piirre: ne ymmärtävät piirisimulaattori SPICEn kuvantamiskoodausta ja analysointikäskyjä. Ilman tällaista yhteistä ominaisuutta ohjelmien käyttäjistä saattaisikin tuntua turhauttavalta siirtyä käyttämään toista ohjelmaa, koska aikaisemmin tehdyt piirit täytyisi piirtää tai kirjoittaa uudelleen. Tämän takia myös tässä työssä kehitetyn ohjelman käyttöliittymään on tehty mahdollisuus käyttää joitakin SPICEn käskyjä.

## 6.1 Piirin kuvantamiskäskyt

Kaikkien peruskomponenttien kuten vastusten, jännitelähteiden, virtalähteiden, kondensaattorien ja induktorien kuvantaminen tapahtuu koodiriveittäin. Eli jokainen komponentti kuvataan omalla koodirivillään ja koodirivin ensimmäinen kirjain määrittelee, mitä komponenttia piiriin ollaan lisäämässä [20].

### 6.1.1 Vastukset

Vastusten kuvantaminen onnistuu seuraavan yleisen muodon mukaisesti

```
1 RNIMI N1 N2 ARVO
```

, missä rivin ensimmäinen kirjain $R$ määrittelee, että piiriin ollaan lisäämässä vastusta ja $R$:n jälkeinen osa nimi täydentää lisättävän vastuksen symbolin halutun mukaiseksi. $N1$ ja $N2$ määrittelevät mille kytkentävälille vastusta ollaan lisäämässä ja $ARVO$ määrittää vastuksen arvon. Esimerkiksi vastus $R1$, joka on kytketty kytkentäpisteiden 5, 6 välille ja jonka vastusarvo on 100 ohmia kuvannettaisiin seuraavasti

```
1 R1 5 6 100
```

### 6.1.2 Riippumattomat lähteet

Riippumattomien tasavirtaisten lähteiden kuvantaminen on yleiseltä muodoltaan hyvin samanlainen, kuin vastusten. Lähteessä on vastuksista poiketen otettava kuitenkin huomioon myös kytkennän suunta ja siten tasavirtaisen jännitelähteen yleisessä muodossa

```
1 VNIMI N+ N- DC ARVO
```

, missä $N+$ on sen positiivinen ja $N-$ sen negatiivinen napa, kun taas tasavirtaisella virtalähteen yleisessä muodossa.

```
1 INIMI N+ N- DC ARVO
```

Virtalähteen virran suuntaa symboloiva nuoli osoittaa pisteestä $N+$ pisteeseen $N-$. Kummassakin määritelmässä $DC$ kuvantaa, että kyseessä on tasavirtainen lähde. Vaihtovirtaisilla lähteillä määritelmä $DC$ korvataan määritelmällä $AC$ ja koko määritelmän perään lisätään vielä lähteen kulma. Esimerkiksi kytkentävälille $4, 6$ lisätty jännitelähde $V2$, jonka jännite on $v(t) = 8\sin(2t)$ V määriteltäisiin seuraavasti.

```
1 V2 4 6 AC 8 -90
```

### 6.1.3 Kondensaattorit ja induktorit

Kondensaattorit ja induktorit voidaan kuvantaa kuten riippumattomat vaihtovirtaiset lähteet. Ainoana erona on, että kondensaattorille ja induktorille määritellään hetken $t = 0$ s virta, eikä kulmaa. Tämä määritelmä on vaihtovirtaisten lähteiden kulman tavoin myös optionaalinen. Yleinen muoto kondensaattorin kuvantamiseksi on

```
1 CNIMI N+ N- ARVO 0VIRTA
```

ja induktorin yleinen muoto on.

```
1 LNIMI N+ N- ARVO 0VIRTA
```

Esimerkiksi kytkentävälille $4, 5$ kytketty kondensaattori $C1$, jonka hetken $t = 0$ s virta on $0.1$ A ja jonka kapasitanssi on $0.2$ F, kuvannettaisiin seuraavasti.

```
1 C1 4 6 0.2 IC=0.1
```

Edellä mainitun mukaisesti hetken $t = 0$ s virta on kuitenkin optionaalinen.

### 6.1.4 Riippuvaiset lähteet

Riippumattomista lähteistä poiketen jänniteriippuvaisissa lähteissä on kuvannettava myös ne kytkentäpisteet, joiden välisestä jännitteestä se on riippuvainen. Yleinen muoto jänniteriippuvaiselle jännitelähteelle on

```
1 ENIMI N+ N- NC+ NC- ARVO
```

ja jänniteriippuvaiselle virtalähteelle

```
1 GNIMI N+ N- NC+ NC- ARVO
```

, missä $N+$ ja $N-$ määrittelee lähteen suunnan samoin kuin riippumattomissa lähteissä ja $NC+$ on vaikuttavien kytkentäpisteiden positiivinen ja $NC-$ on negatiivinen napa.

Virtariippuvaiset lähteet voidaan kuvantaa riippuvaisiksi vain komponentin virrasta. Virtariippuvaisen virtalähteen yleinen muoto on

```
1 ENIMI N+ N- SYMBOLI ARVO
```

ja virtariippuvaisen jännitelähteen yleinen muoto on

```
1 HNIMI N+ N- SYMBOLI ARVO
```

, missä SYMBOLI on sen komponentin symboli, jonka virrasta lähde on riippuvainen.

## 6.2 Piirin analysointikäskyt

Piirien kuvantamisen lisäksi ohjelman on vielä saatava tietää, mitä piirin ominaisuuksia sen halutaan laskevan. Tämä onnistuu ns. analyysikäskyillä, jotka voidaan jakaa kolmeen eri kategoriaan tasavirta eli DC-analyysiin, vaihtovirta eli AC-analyysiin ja transientti- eli aikakehitysanalyysiin.

### 6.2.1 Tasavirta- ja vaihtovirta-analyysi

Jos piirissä on vain tasavirtaisia lähteitä, eikä mahdollisiin piirissä oleviin kondensaattoreihin tai induktoreihin ole asetettu mitään hetken $t = 0\,s$ virtaa, laskee ohjelma piirin operaatiopisteen automaattisesti. Muissa tapauksissa ohjelma pitää ohjelmoida laskemaan operaatiopiste

```
.op
```

, jolla ohjelma laskee piirin operaatiopisteen ja tulostaa kaikkien kytkentäpisteiden jännitteet, sekä komponenttien virrat. Ohjelma voidaan kuitenkin asettaa tekemään myös yksityiskohtaisempia ja rajatumpia analyysejä. Ohjelma voidaan esimerkiksi rajata laskemaan vain tiettyjen komponenttien läpi menevät virrat ja yli olevat jännitteet. Ohjelma voidaan asettaa laskemaan vain tiettyjen kytkentäpisteiden ja niiden väliset jännitteet. Rajaus onnistuu $.PRINT$ käskyllä. Esimerkiksi käsky

```
.PRINT DC V(1) V(2,3) V(R1) I(R2)
```

, rajaa ohjelman tulostamaan vain kytkentäpisteen 1 jännitteen, vastuksen $R1$ yli olevan jännitteen ja vastuksen $R2$ läpi menevän virran. Käsky laajentaa kuitenkin samalla ohjelman laskenta-aluetta ja asettaa sen laskemaan myös kytkentäpisteiden 2 ja 3 välisen jännitteen. Vaihtovirta-, eli AC-analyysi suoritetaan kuten tasavirta-analyysi, mutta käskyyn vaihdetaan rajauksen $DC$ tilalle rajaus $AC$.

### 6.2.2 Transienttianalyysi

Transienttianalyysi voidaan suorittaa käskyllä,

```
.TRAN 3ns 300ns
```

, missä 3ns on askelväli ja 300ns on ajankohta mihin asti analyysi suoritetaan.

# 7 Johtopäätökset

Työssä kehitetty ohjelma toteutettiin java-ohjelmointikielellä ja työn edetessä java ei osoittautunut ainakaan huonoksi valinnaksi. Informaatiota muilla kielillä, kuten fortranilla ja C:llä toteutettavista simulointiohjelmista oli tosin tarjolla enemmän, mutta kaikki se informaatio oli sovitettavissa myös javalle. Internetin keskustelu palstoilla javaa sanottiin myös hitaaksi ja suhteellisen huonoksi kieleksi matemaattisissa ohjelmissa, mutta siitä huolimatta javalla pystyttiin tekemään suhteellisen nopea simulointiohjelma. Esimerkiksi ilmaisjakeluohjelma LTSpice oli lineaarisissa piireissä n. 2 kertaa hitaampi kuin kehitetyn ohjelman nopein analyysimenetelmä. Keskimäärin nopeimmaksi analyysimenetelmäksi osoittautui solmupisteanalyysin ja TICER-yksinkertaistuksen yhteistyöllä toteutettu menetelmä. Kyseisessä menetelmässä piiri aluksi yksinkertaistetaan TICER-yksinkertaistuksen avulla, jonka jälkeen yksinkertaistettu piiri analysoidaan solmupisteanalyysin avulla ja lopuksi yksinkertaistetun piirin loput arvot lasketaan, kun yksinkertaistettu piiri palautetaan yksinkertaistamattomaksi. Epälineaarisissa piireissä ja transienttianalyysissä kehitetty ohjelma oli tosin keskimäärin n. 2 kertaa hitaampi, mutta niiden osalta ohjelmaa ei saatu vielä täysin toimintavarmaksikaan.

Työn edetessä ohjelmointityön haastellisimmiksi osiksi osoittautuivat piirin topologia ja ohjelman matemaattinen koneisto. Niistä löytyi aina vähän väliä jotain uutta kehittämisen varaa ja pari kertaa ne täytyikin kehittää aivan lähes alusta asti uusiksi. Siksi suosittelisinkin muille, jotka mahdollisesti ovat aloittamassa samankaltaisen simulaattorin kehittämistä, että käyttäisivät jo kehitettyjä matemaattisia kirjastoja tai aloittaisivat ohjelmoinnin toimivan matemaattisen koneiston kehittämisestä. Ohjelmointia aloitettaessa tosin on hyvin vaikea tietää, mitä kaikkea matemaattiselta koneistolta vaaditaan ja tärkeintä onkin että se suunniteltaan rakenteeltaan helposti laajennettavaksi.

# Viitteet:


[1] All About Circuits. Example circuits and netlists. www.allaboutcircuits.com, 04 2014.

[2] Biswa Nath Datta. *Numerical Linear Algebra and Applications*, volume 530. SIAM, 2 edition, 2010, sivut (129-131).

[3] David Gorham David Crecraft. *Electronics*, volume 448. 2 edition, 2003, sivut (24-33).

[4] Ernst A. Guillemin. *Introductory circuit theory*, volume 550. Wiley, 5 edition, 1953, sivut (131-137).

[5] Steven Frankel Joe D. Hoffman. *Numerical Methods for Engineers and Scientists*, volume 840. CRC Press, 2 edition, 2001, sivut (30-45).

[6] Steven Frankel Joe D. Hoffman. *Numerical Methods for Engineers and Scientists*, volume 840. CRC Press, 2 edition, 2001, sivut (59-67).

[7] Erwin Kreyszig. *Advances Engineering Mathematics*. Wiley, 10 edition, 2010, sivut: (858-860).

[8] K. S. Suresh Kumar. *Electric Circuits and Networks*, volume 840. Pearson Education India, 1 edition, 2009, sivut (770-774).

[9] Chandramouli Visweswariah Lawrence T. Pillage, Ronald A. Rohrer. *Electronic circuit and system simulation methods*, volume 392. McGraw-Hill, 1995, sivut (75-86).

[10] Laurence W. Nagel. Spice2: A computer program to simulate semiconductor circuits. *University of California, Berkeley*, pages 59–72, 1975.

[11] Farid N. Najm. *Circuit Simulation*, volume 318. Wiley, 4 edition, 2010, sivut (32-42).

[12] Farid N. Najm. *Circuit Simulation*, volume 318. Wiley, 4 edition, 2010, sivut (60-71).

[13] Matias Pekkarinen. Tietokonealgoritmi elektroniikkapiirien simulointiin. *Jyväskylän Yliopisto, Fysiikan laitos*, 2013.

[14] E. Jan W. Ter Maten Peter Benner, Michael Hinze. *Model Reduction for Circuit Simulation*, volume 328. Springer, 4 edition, 2011, sivut (149-160).

[15] James A. Svoboda Richard C. Dorf. *Introduction to Electric Circuits*, volume 886. John Wiley and Sons, 2010, sivut (115-119).

[16] J. Douglas Faires Richard L. Burden. *Numerical Analysis*, volume 872. Richard Stratton, 9 edition, 2011, sivut (403-406).

[17] J. Douglas Faires Richard L. Burden. *Numerical Analysis*, volume 872. Richard Stratton, 9 edition, 2011, sivut (638-643).

[18] J. Douglas Faires Richard L. Burden. *Numerical Analysis*, volume 872. Richard Stratton, 9 edition, 2011, sivut (654-660).

[19] Lawrence Turyn. *Advanced Engineering Mathematics*, volume 1455. CRC Press, 2013, sivut (682-687).

[20] Andrei Vladimirescu. *THE SPICE BOOK*, volume 411. Wiley, 1993.

[21] C.L. Wadhwa. *Network Analysis and Synthesis*, volume 833. New Age International, 2 edition, 2006, sivut (113-116).

# A Liite: Gaussin-eliminointi algoritmi

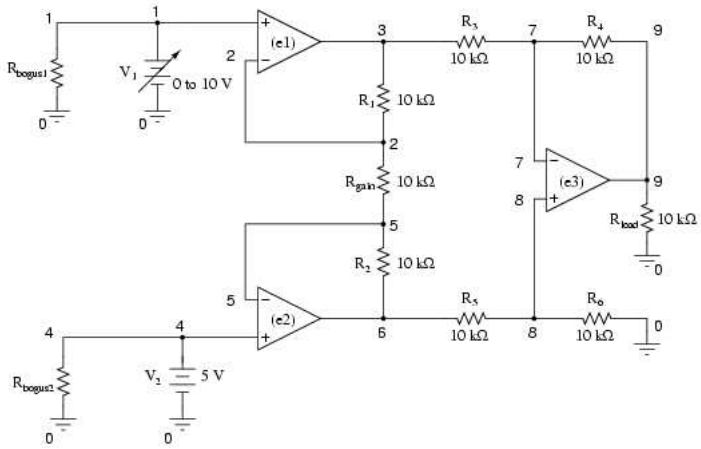
```
public static double[] gaussEliminate(double[][] A, double[] b) {
    for (int i = 0; i < A.length - 1; i++) {
    //eliminoidaan kertoimia alaspäin
            double diag= A[i][i];
            if(diag==0){
                //jos diagonaalielementti on 0 rivejä vaihdetaan
            }
          Vector.divide(A[i], diag);//Kaikki rivin i elementit
            jaetaan
            //diagonaalielementillä
            b[i] = b[i] / diag;
            for (int j = i + 1; j < A.length; j++) {//
                double kerroin = A[j][i];
                A[j] = Vektori.miinus(A[j], Vektori.tulo(A[i],
                  kerroin));
                //Rivistä j eliminoidaan sarakkeen i muuttuja
                b[j] = b[j] - b[i] * kerroin;
            }
        }
        double kerroin = A[A.length - 1][A.length - 1];
        //Viimeisen rivin muuttujan kertoimeksi muutetaan 1
          jakamalla koko rivi muuttujan arvolla
        A[A.length - 1] = Vektori.jako(A[A.length - 1], kerroin);
        b[b.length - 1] = b[b.length - 1] / kerroin;
        for (int i = A.length - 1; i > 0; i--) {
        //Eliminoidaan kertoimia ylöspäin
            double diag = A[i][i];
            A[i] = Vektori.jako(A[i], diag);//Rivin i
            b[i] = b[i] / diag;
            for (int j = i - 1; j >= 0; j--) {
                double kerroin = A[j][i];
                A[j] = Vektori.miinus(A[j], Vektori.tulo(A[i],
                  kerroin));
                //Rivistä j eliminoidaan sarakkeen i muuttuja
              b[j] = b[j] - b[i] * kerroin;
           }
        }
```

```
34          double kerroin = A[0][0];
35          A[0] = Vektori.jako(A[0], kerroin);
36          //Ensimmäisen rivin elementit jaetaan saman rivin
              diagonaalielementillä
37          b[0]=b[0]/kerroin;
38  }
```

# B Liite: Newtonin menetelmän algoritmi

```
/**
*
* @param coefmatrix Yhtälöryhmän kerroinmatriisi
* @param eqgroup Yhtälöryhmän muodostavat yhtälöt
* @param var Yhtälöryhmän muuttujat
* @param xn Alkuarvaus
* @param tol Toleranssi
* @param max Maksimimäärä iteraatioita
* @return Yhtälöryhmän ratkaisun approksimaatio
     */
public static Muuttuja[] newtonMethod(Muuttuja[][] coefmatrix,
  Muuttuja[] eqgroup, SVar[] var,Muuttuja[] xn,double tol,int max)
  {
    double res=999999999;
    int j=0;
    Muuttuja[] prev;//Edellisen approksimaation n-1 arvo
    while(res>tol){//Toistetaan kunnes on riittävän tarkka
        if(j>max)// tai iteraatioita on tehty maksimimäärä
            break;
        prev=xn;
        j++;
        for (int i = 0; i < xn.length; i++) {
            var[i].setValue(xn[i].toNumVar());//Muuttujiin
              asetetaan
        }//niiden approksimaatiot
        Muuttuja[][] J=Matrix1.createJacobian(coefmatrix, var);
        Muuttuja[] Fn=Vektori1.toNumVar(eqgroup,true);
        //osa F(xn) yhtälöstä J(xn)=-F(xn)
        Muuttuja[] yn=Matrix1.LUfact(J, Fn);
        //Yhtälöryhmä ratkaistaan
        xn=Vektori1.plus(yn, prev);
        //lasketaan vaiheen n approksimaatio xn
        res=Vektori1.distance(xn,prev).getReal();
        //Lasketaan tarkkuus
    }
    return xn;
}
```

# C Liite: Vahvistin




Kuva 25: Ideaalisilla operaatiovahvistimilla toteutettu vahvistin [1]

**Algoritmi 1** Vahvistimen SPICE koodi

```
v1 1 0
rbogus1 1 0 9e12
v2 4 0 dc 5
rbogus2 4 0 9e12
e1 3 0 1 2 999k
e2 6 0 4 5 999k
e3 9 0 8 7 999k
rload 9 0 10k
r1 2 3 10k
rgain 2 5 10k
r2 5 6 10k
r3 3 7 10k
r4 7 9 10k
r5 6 8 10k
r6 8 0 10k
.op
.end
```

# D Liite: Lineaarinen piiri

**Algoritmi 2** Lineaarisesti riippuvaisen piirin SPICE koodi

```
R1 08 12 2
R2 04 08 3
R3 05 04 2
R4 06 05 4
R5 09 06 5
R6 14 09 1
R7 12 13 3
R8 13 14 4
R9 01 07 2
R10 10 15 7
R11 03 02 5
R12 0 16 8
I1 13 0 2
I2 08 07 4
V1 02 05 10
V2 10 09 5
R13 02 01 1
R14 11 07 5
R15 0 15 7
R16 03 10 6
17 03 06 2
R18 14 15 6
R19 12 11 10
R20 15 14 4
E1 16 11 11 12 0.8
.op
.end
```

# E Liite: Epälineaarinen piiri

**Algoritmi 3** Epälineaarisesti riippuvaisen piirin SPICE koodi

```
R1 10 14 2
R2 04 10 3
R3 05 04 2
R4 06 05 4
R5 11 06 5
R6 16 11 1
R7 14 15 3
R8 15 16 4
R9 01 09 2
R10 12 17 7
R11 03 02 5
R12 0 20 8
I1 15 0 2
I2 10 09 4
V1 02 05 10
V2 12 11 5
R13 02 01 1
R14 13 09 5
R15 0 17 7
R16 03 12 6
R17 03 06 2
R18 16 17 6
R19 14 13 10
R20 17 16 4
B1 20 13 V=(sin(V(05)))
G1 1 2 8 7 1.2
.op
.end
```