TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Infotehnoloogia eriala

Oskar Gross

Neurovõrkude kasutamine dünaamiliste süsteemide juhtimisel

Bakalaureusetöö (4 AP)

Juhendaja: Sven Laur, D.Sc. (Tech.)

Autor: ..... "....." Juuni 2009
Juhendaja: ..... "....." Juuni 2009

Lubada kaitsmisele
Jaak Vilo (Ph.D.): ..... "....." Juuni 2009

TARTU 2009

Sisukord

1 Sissejuhatus

Antud bakalaureusetöö kirjeldab neurovõrkude kasutamist dünaamiliste süsteemide juhtimisel. Töö üheks eesmärgiks on anda lihtne ülevaade tehisneurovõrkudest ja nende toimimise põhimõtetest ning töö peaks olema ka hea õppevahend inimesele, kes soovib antud teemaga tutvust teha. Kuna autoril oli enne töö kirjutamist väike kokkupuude masinõppega, siis on üritatud ära katta kõik need teemad, mis võivad alguses raskusi valmistada.

Töö koosneb kahest osast. Esimene osa annab teoreetilise ülevaate neurovõrkudest ning teine osa on praktiline töö. Neurovõrkude teoreetilises ülevaates käsitletakse inimese bioloogilise närvisüsteemi ehitust ning seda, kuidas on neurovõrkude toimimissüsteem neile sarnane. Neurovõrkude puhul räägitakse erinevatest neuroni tüüpidest ning nende omadustest, näidatakse, kuidas erinevatest neuronitest moodustada neurovõrku ning kuidas neurovõrk tulemusi arvutab.

Neurovõrkude treenimise peatükis kirjeldatakse, mis on regressiooniülesanne ning kuidas neurovõrk aitab seda lahendada. Peatükis tutvustatakse ka populaarsemaid neurovõrgu treenimise meetodeid ning tuuakse välja meetodite plussid ja miinused. Kirjeldatakse ka neurovõrgu valideerimise võimalusi. On kirjeldatud, mis on regulariseerimine ning miks seda on vaja teha. Lugejale tutvustatakse neurovõrgu pügamise eesmärke ja põhimõtteid ning antakse ülevaade kahest populaarsemast neurovõrgu pügamise algoritmist. Lisaks sellele kirjeldatakse, mis on dünaamiline süsteem ning mis on selle omadused. Dünaamiliste süsteemide juures on käsitletakse ka süsteemi BIBO ja asümptootilist stabiilsust ning antakse ülevaade *open-loop* ja *closed-loop* kontrolleritest ning nende toimimise põhimõtetest.

Töö praktilises osas näidatakse inverteeritud pendli abil, kuidas luua tehisneurovõrke, mis suudavad süsteemi kontrollida. Täpsemaks eesmärgiks on hoida pendlit vertikaalses asendis. Ühtlasi antakse ülevaade süsteemi identifitseerimisest ja kontrollimisest ning seotakse omavahel erinevad praktilised sammud ja teooria. Praktilises osas on kasutatud kolme erinevat kontrolleri tüüpi - *Optimal*, NPC (*ing.k. Nonlinear Predictive Control*) ja DIC (*ing.k. Direct Inverse Control*).

Praktilise tööga on kaasas CD plaat, kus on olemas kõik tarvilikud failid ja andmed, et praktilist osa saaks ka oma arvutis läbi viia. Praktilise osa iseseisvaks läbiviimiseks on tarvilik MATLAB®'i (The MathWorks, Inc., 2007a) ja Simulink®'i (The MathWorks, Inc., 2007b) olemasolu.

Lisaks teoreetilisele ja praktilisele osale on NNSYSID tööriistale juurde programmeeritud võimalus neurovõrgu treenimisel töötada korraga mitmete aegridade fragmentidega. Antud modifikatsiooni leiab CD plaadilt kaustast *NNSYSID*, kus failis *Demo.m* on demonstreeritud, kuidas implementeeritud funktsioone ka-

sutada saab.

2 Neurovõrgud

2.1 Üldine ülevaade

Peamiseks väljakutseks tehisintellekti loomisel on luua süsteem, mis suudab kogemusest õppida ja sellele tuginedes teha iseseisvalt otsuseid. Kui me vaatame klassikalist programmi, siis suudab see teha andmete põhjal otsuseid, mis on vastavate tingimuste abil programmeeritud. Sõltumata andmete hulgast, mida programm töötleb, otsuste tegemise skeem jääb samaks ja ei muutu. Tänapäeval ollakse üha enam veendunud selles, et inimese aju kujutab endast mingit teatud tüüpi arvutit (Searle, 1980). Kuna tehisintellekt peaks täitma sama funktsiooni, mida inimese aju, siis sellest tulenevalt on tehisintellekti loomisel proovitud luua süsteem, mis oleks võimalikult lähedane inimese närvisüsteemi toimimise mehhanismidele. Antud süsteemi või programmi nimetamegi neurovõrguks.

Neurovõrke hakati uurima 1943. aastal, mil Warren S. McCulloch ja Walter Pitts avaldasid teose „A logical calculus of the ideas immanent in nervous activity" ning lõid sellega aluse tehisintellekti uurimiseks. McCulloch'i ja Pitts'i töö näitas, et on võimalik konstrueerida neurovõrk, kasutades selleks ainult matemaatikat ja algoritme. Nad pakkusid välja, et ühendades omavahel hulga piiratud tuletisvõrke (*ing.k. Bounded Derivative Network*), on võimalik luua selline võrk, mis suudab lahendada arvutamist vajavaid ülesandeid. Neurovõrgu treenimist tutvustas 1962 aastal Frank Rosenblatt, kes näitas, kuidas treenida tuletisvõrke. Treenimisel Rosenblatt muutis erinevate neuronite vahelisi kaale seni, kuni ta sai õige vastuse.

Inimese närvisüsteem jaguneb kaheks osaks: perifeerne- ja kesknärvisüsteem. Kesknärvisüsteemi ülesandeks on organismi välis- ja sisekeskkonnast saadud informatsiooni kogumise ja töötlemisega ning vastavalt nendele andmetele vastuse väljatöötamisega. Perifeerse närvisüsteemi ülesandeks on informatsiooni vahendamine kesknärvisüsteemi ja keskkonna vahel. Inimese närvisüsteemis saab neuron sisendi dendriitidest ning väljund liigub neuriiti ehk aksonisse. Neuronid on omavahel ühendatud sünapsiga, kus ühe neuroni neuriit puutub kokku teise neuroni dendriidiga. Üks neuron võib olla seotud tuhandete teiste neuronitega sünapsite abil. Sünapsid võivad olla kahte tüüpi: keemilised või elektrilised. Elektrilised sünapsid saadavad signaali viivitamatult ja muutmata kujul edasi ning keemilised sünapsid ei kanna signaali viivitamatult edasi, vaid kasutavad sünaptilist pilu selleks, et erutamata rakku saata närviimpulss, kuid erutatud rakku mitte. Neurovõrkudes on sünapsid elektrilised, ehk ühe neuroni informatsioon kantakse otse edasi järgmisesse neuronisse. Sünapsite puhul on oluline ka see,

et mida vähem sünapsit kasutatakse, seda nõrgemaks jääb tema poolt tuleva signaali mõju järgmisele rakule ning on ka võimalik, et antud sünaps ühel hetkel enam signaali edasi ei anna. Seda olukorda ilmestab unustamine, kus mingid sidemed neuronite vahel on katkenud ning inimene ei suuda soovitud informatsiooni kätte saada. Bioloogiliselt tähendab õppimine seda, et neurovõrk muudab oma struktuuri - neuronite vahele tekivad teatava kaaluga sidemed ning põhimõtteliselt muudab aju oma struktuuri seni, kuni inimene harjutab. See on ka põhjus, miks peab ennast treenima - ajus puuduvad algselt sidemed konkreetse tegevuse sooritamiseks.

Tehisneurovõrgud mudeldavad päris neurovõrke. Tehisneurovõrgud kirjeldavad inimese kesknärvisüsteemile sarnast süsteemi, mis töötleb perifeersest närvisüsteemist saadud informatsiooni. Samamoodi saab tehisneurovõrk sisendi(d) ning analoogiliselt bioloogilisele närvisüsteemile liigub see mööda sidemeid edasi järgmisesse neuronisse. Tehisneurovõrkudes kasutatavad neuronid on kõik n.ö. elektrilised, ehk signaal liigub viivitamatult edasi järgmisesse neuronisse. Samamoodi nagu bioloogilistes närvivõrkudes kõikide neuronite vahelistel seostel oma kaal, siis samamoodi on ka tehisneurovõrkude puhul.

Tehisneurovõrke võime vaadelda kui keerukat funktsioonide klassi, mis lubab lähendada igasuguseid funktsioone, teisisõnu saame õpetada neurovõrku kirjeldama mingit funktsiooni. Kui kasutada mitte-bioloogilist terminit, siis õppimine on funktsiooni lähendamine näidete põhjal - samamoodi on ajul vaja mingit hulka näiteid vaja selleks, et õppida. Väga primitivset näidet kasutades, siis näiteks tennise servi õppimisel võivad olla eeskujuks profitennisisti serv, joonis soovitavast liigutusest või hoopis sõnaline kirjeldus.

Neurovõrke kasutatakse peamiselt diskreetsete või pidevate tunnuste ennustamiseks teistest andmetest. Esimesel juhul on reeglina tegemist klassifitseerimisning teisel juhul regressiooniülesandega. Lihtsa klassifitseerimisülesanne näiteks on see, kui anda süsteemi sisendiks käsitsi kirjutatud allkiri ning süsteem peab otsustama, millisesse klassi see paigutada. Antud süsteem peab sel juhul allkirja näidisest kasutama piksleid x_1, ..., x_n ning nende põhjal otsustama millisesse klassi antud allkiri kuulub. Regressiooniülesanne on näiteks palli veeremise ennustamine mingil tasapinnal või mingi muu pidevate väärtustega süsteemi käitumise ennustamine. Regressiooniülesande puhul soovitakse leida ligikaudset regressiooni funktsiooni, klassifitseerimisülesande puhul me soovime leida ligikaudseid funktsioone, mis suudavad sisendi põhjal anda ligikaudse hinnangu mingisse klassi kuuluvuse kohta.

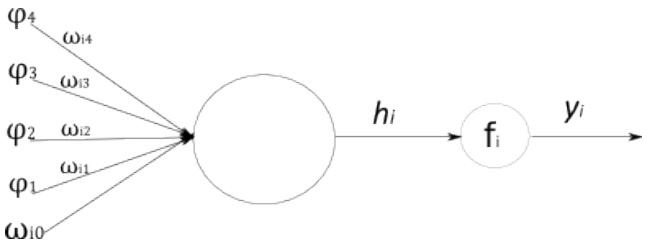
Neurovõrkude treenimisele saab läheneda kahel viisil: järelvalvega treenimine (*ing.k. supervised*) ja järelvalveta treenimine (*ing.k. unsupervised*). Suur osa neu-

rovõrkudest kasutab järelvalvega treenimist. Järelvalvega treenimine tähendab seda, et neurovõrgule on ette antud paarid $(\mathbf{x}_n, y_n)$ ning nende paaride abil arvutab neurovõrk mingi algoritmi alusel kaalud. Järelvalveta treenimist kasutatakse selleks, et teostada algset sisendite iseloomustust. Järelvalveta treening tähendab seda, et neurovõrk peab ise suutma aru saada sisenditest ilma välise abita. Regressiooni- ja klassifitseerimisülesannete puhul järelvalveta treeningut ei kasutata. Sellised ülesanded, kus kasutatakse järelvalveta treeningut, on SOM ja klasterdamine. Täielikult iseõppivat neurovõrku ei ole põhimõtteliselt võimalik luua, seega järelvalveta treeningut saame kasutada väga piiratult.

Tehisneurovõrkude kohta saab informatsiooni paljudest raamatutest, Tartu Ülikooli raamatukogus on neist kättesaadavad (Bishop, 1995) ja (Swingler, 1996).

2.2 Tehisneurovõrkude ehitus

Tehisneurovõrk koosneb neuronitest, mis on omavahel ühendatud erinevate sidemetega. Neuron on element, mis saab mingi arvu sisendeid, korrutab neid kaaluga (*ing.k. weight*), summeerib ja kasutab väljundit argumendina ühe muutuja funktsioonile, mida nimetatakse aktiveerimisfunktsiooniks (*ing.k. activation function*). Vaata joonist 2.1.



Joonis 2.1: Viie sisendi ja ühe väljundiga neuron

Neuroni sisendiks võivad olla teiste neuronite väljundid või välised sisendid. Neuronil, mis on joonisel 2.1, saame väljundi y_i arvutada järgnevalt

$$y_i = f_i\left(\sum_{j=1}^{n} \varphi_i \omega_{i,j} + \omega_{i,0}\right), \tag{2.1}$$

kus y_i on väljundsignaal, f_i on aktiveerimisfunktsioon, φ_i on sisend, $\omega_{i,j}$ on vastava sisendi kaal ning sisendit ω_{i0} kutsutakse kõrvalekaldeks (*ing.k. bias*), mida võime tõlgendada nii, et meil on pseudosisend väärtusega 1, mille kaal on ω_{i0}. Neuroni aktiveerimisfunktsioon määrab ära neurovõrgu õpetatavuse ja paindlikkuse.

Enimlevinud neuroni aktiveerimisfunktsioonid on lineaarsed ja sigmoidsed. Lineaarseks neuroniks nimetame me sellist neuronit, mille aktiveerimisfunktsioon

on lineaarne

$$f(x) = \alpha x \tag{2.2}$$

ning sigmoid-neuronite aktiveerimisfunktsiooniks on enamasti hüperboolne tangens

$$f(x) = tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}. \tag{2.3}$$

Enimkasutatud sigmoidfunktsioon on *tanh* funktsioon - sellist neuronit nimetatakse *tanh*−neuroniks. Antud töös ülejäänud neuroni tüüpe ei vaadelda.

Neuroneid saab võrku ühendada paljudel erinevatel viisidel, kuid kõige levinumaks viisiks on neuronite ühendamine võrku nii, et nad jagunevad kihtidesse. Sel juhul koosneb neurovõrk mitmest kihist. Sisendkihist, mis võtab vastu sisendeid, peidetud kihist, kus tehakse suurem osa neurovõrgu arvutustest ning väljundkihist, kust me saame kätte süsteemi väljundi. Peidetud kihi sees võib olla omakorda mitmekihiline neurovõrk et cetera. Kõige lihtsam mitmekihiline närvivõrk jagab kõik neuronid erinevatesse kihtidesse nii, et iga neuron saab sisendiks võtta ainult eelmisel kihil olevate neuronite või välist informatsiooni. Kui võrgul on selliseid kihte 2, siis seda nimetatakse kahekihiliseks neurovõrguks. Vastavalt sellele struktuurile nimetatakse sellist neurovõrku ka pärileviga neurovõrguks (*ing.k. feedforward network*).

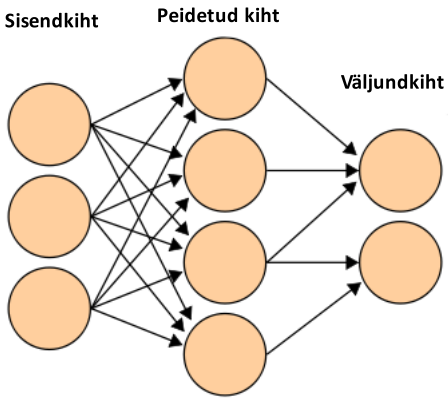
Samamoodi nagu iga neuroni sisendil on oma kaal, on ka neuronite vahelistel sidemetel kaal, mis korrutatakse neuronisse suubuva signaaliga.

Neurovõrkude struktuuri määrab ka see, milliste neuronite vahel sidemed eksisteerivad. Kui ühe kihi iga neuron on seotud järgmise kihi iga neuroniga, siis sellist neurovõrku nimetame me täielikuks neurovõrguks. Praktikas on neurovõrgud tihti mittetäielikud, sest sellisel juhul on vähem parameetreid, mida andmetest hinnata ning neurovõrgu treenimine annab stabiilsemaid tulemusi.

Neurovõrgu struktuuri omadusi ja treenitavust mõjutavadki peamiselt neuronite tüüp, neuronite vahelised sidemed ning nende sidemete kaalud.

Praktikas kasutatakse tihti kahe-kihilisi neurovõrke, kuna nende struktuur on piisavalt lihtne ning samas võimaldavad nad kirjeldada keerukaid süsteeme. Kahe-kihilises neurovõrgus on kaks kihti: peidetud kiht ja väljundkiht. Peidetuks nimetatakse kihti sellepärast, et ta asub sisend- ja väljundkihi vahel, ehk antud kihti ei ole võimalik kohe näha. Seega on kahe-kihilisel neurovõrgul 1 peidetud kiht, kolme-kihilisel neurovõrgul kaks peidetud kihti jne. Võrku nimetatakse täielikult ühendatuks juhul, kui kõik ühes kihis olevad neuronid on seotud kõikide järgmises kihis olevate neuronitega (Joonis 2.2).

Kahe-kihilise neurovõrgu väljundit y_i saab arvutada järgmiselt:



Joonis 2.2: Täielikult ühendatud kahe-kihiline neurovõrk

$$y_i = g_i\left(\boldsymbol{\varphi}, \boldsymbol{\theta}\right) = F_i\left(\sum_{j=1}^{n_h} W_{i,j}\, f_j\left(\sum_{l=1}^{n_\varphi} \omega_{j,l}\varphi_l + \omega_{j,0}\right) + W_{i,0}\right), \tag{2.4}$$

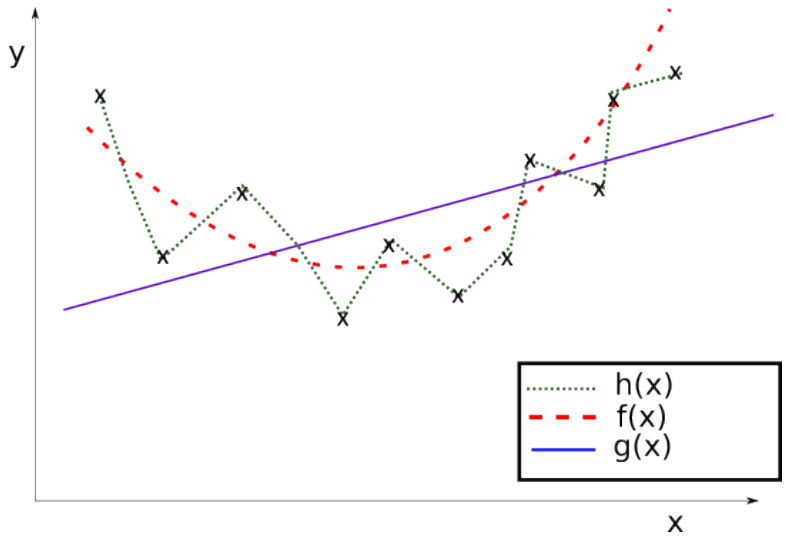
kus g_i on aktiveerimisfunktsioon,$\boldsymbol{\varphi}$ on sisendite vektor, $\boldsymbol{\theta}$ on parameetrite vektor, F_i on väljundneuroni aktiveerimisfunktsioon, f_j on vastava eelneva kihi neuroni aktiveerimisfunktsioon, $\omega_{j,l}$ on vastava neuroni ühenduse kaal, φ_l on vastava ühenduse kaudu tulev sisend, $\omega_{j,0}$ on vastava neuroni kõrvalekalle ning $W_{i,0}$ on kõrvalekalle, mis lisandub väljundneuronile. Suurus $W_{i,j}$ tähistab väljundneuronisse suubuvate ühenduste kaalusid. Tähis i tähistab väljundneuronit ja suurus j tähistab temasse suubuva kaalu väärtust.

Valemis väljendab $\boldsymbol{\theta}$ parameetrite vektorit, mis sisaldab endas kõiki neurovõrgu muudetavaid parameetreid, s.t. kaalude ja kõrvalekallete väärtusi $\{\omega_i, W_{i,i}\}$. Kuna me võime käsitleda kõrvalekallet kui kaalu, mis antakse sisendile väärtusega 1, siis mõeldakse edaspidi „kaalude“ all nii kõrvalekaldeid kui ka kaalusid.

3 Neurovõrkude treenimine

3.1 Regressiooniülesanne

Neurovõrgust on kasu alles siis, kui sellele on selgeks õpetatud mingi süsteemi käitumine. Selleks tuleb koguda süsteemist kindel hulk näiteid, mis kirjeldavad süsteemi sisenditele vastavaid väljundeid. Seejärel tuleb valida neurovõrgu mudeli struktuur, mida me õpetama hakkame. Kui me oleme neurovõrgu mudeli struktuuri valinud, siis leiame treenimise abil antud hulgast need parameetrid, mis kirjeldavad süsteemi kõige paremini. Neuronite vaheliste kaalude leidmise protsessi, nimetatakse treenimiseks (*ing.k. training*) või õppimiseks. Antud töös käsitletakse neurovõrgu treenimist kui regressiooniülesande lahendamist neurovõrgu abil.



Joonis 3.1: Regressiooniülesande graafikud, kus $f(x)$ on reaalne funktsioon ning andmepunktid on saadud koos müraga. Kõik graafikul olevad funktsioonid illustreerivad funktsioonide klassi $\mathcal{F}$

Regressioonülesande puhul on antud mingi lõplik hulk andmepunkte, mis kirjeldavad süsteemi ning nende andmete põhjal soovime me leida funktsiooni, mis oleks võimalikult lähedane reaalsele süsteemi kirjeldavale funktsioonile. Näiteks, olgu meil süsteem, mida saab iseloomustada funktsiooniga $y(x)$, sel juhul soovime mingist funktsioonide hulgast leida sellise funktsiooni $\hat{y}(\mathbf{x})$, mille korral $y(\mathbf{x}) \approx \hat{y}(\mathbf{x})$. Matemaatiliselt väljendatuna, olgu meil antud treeningvalim

$$Z_N = \{[\mathbf{x}_i, y_i],\ i = 1...N\}, \tag{3.1}$$

kus $\mathbf{x}_i \in \mathbb{R}^n$ tähistab süsteemi sisendeid ning $y_i \in \mathbb{R}$ tähistab süsteemi väljundeid. Selleks, et leida funktsioon, mis antud neurovõrgu puhul kirjeldab süsteemi kõige paremini, peame määrama kandidaatfunktsioonide hulga $\mathcal{F}$, mis on määratud neurovõrgu struktuuri ja parameetritega

$$\mathcal{F} \subseteq \{f : \mathbb{R}^n \to \mathbb{R}\}. \tag{3.2}$$

Selleks, et saaksime leida optimaalset neurovõrgu, peame defineerima kaofunktsiooni (*ing. k. loss function*)

$$L = \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}, \tag{3.3}$$

mis väljendab erinevust oodatava või õige väärtuse ja saadud väärtuse vahel. Näiteks majandusarvutustes kasutatakse kaofunktsiooni selleks, et teisendada vale hinnang majanduslikuks kahjuks.

Klassikaliseks kaofunktsiooniks on

$$L(\hat{y},\, y) = (\hat{y} - y)^2, \tag{3.4}$$

kus $\hat{y}$ on ennustatav väärtus ning y on õige väärtus. Samas kasutatakse kaofunktsioonina ka

$$L(\hat{y},\, y) = |\hat{y} - y|. \tag{3.5}$$

Kaofunktsioon (3.4) vastab eeldusele

$$y = f(\mathbf{x}) + \varepsilon,\ \text{kus}\ \varepsilon \sim N(0, \delta), \tag{3.6}$$

ehk kus viga vastab normaaljaotuse tingimusele, mis on täidetud enamiku füüsikaliste katsete korral. Kaofunktsioon (3.5) vastab jällegi vea mudelile

$$y = f(\mathbf{x}) + \varepsilon, \tag{3.7}$$

kus ε vastab Laplace'i jaotusele.

Selleks, et leida funktsioonide klassist $\mathcal{F}$ funktsioon, mis iseloomustaks kõige paremini süsteemi, peame leidma funktsiooni $f \in \mathcal{F}$ nii, et oodatav keskmine kadu (3.8) oleks minimaalne üle tulevast sisendite ja väljundite jaotuse

$$\mathbf{E}(L(f(\mathbf{x}), y)) = \int L(f(\mathbf{x}), y) p(\mathbf{x}, y) d\mathbf{x} dy,\ kus \tag{3.8}$$

$L(f(\mathbf{x}), y)$ on kadu ning $p(\mathbf{x}, y)$ on tihedusfunktsioon, mis näitab, kui suure tõenäosusega asub andmepunkt selles kohas. Praktikas pole keskmine kadu arvutatav sellepärast, et jaotus ei ole teada. Me saaksime arvutada välja keskmise kao juhul, kui jaotus D oleks teada, ent sel juhul ei oleks lahendus enamasti analüütiline. Seoses sellega tuleb leida selline funktsioon $f \in \mathcal{F}$, mis minimeeriks empiirilist kadu

$$V_N = \frac{1}{N} \sum_{i=1}^{n} L(f(\mathbf{x}_i), y_i). \tag{3.9}$$

Suurte arvude seadusest lähtuvalt - kui andmete hulk on piisavalt suur, siis läheneb empiiriline kadu keskmisele kaole

$$\frac{1}{N} \sum_{i=1}^{n} L(f(\mathbf{x}_i), y_i) \approx \mathbf{E}(L(f(\mathbf{x}),\ y)). \tag{3.10}$$

Kui treeningandmed on võetud samast jaotusest D, mis tulevased andmed, siis suurte arvude seaduse korral on iga funktsiooni f korral kehtib (3.9), aga kuna

me minimeerime f sõltuvalt Z_N, siis tekib loomulik hälve

$$\frac{1}{N}\sum_{i=1}^{n} L(f(\mathbf{x}_i), y_i) \lesssim \mathbf{E}(L(f(\mathbf{x}),\ y)). \tag{3.11}$$

See tähendab valides $f \in F$ selliselt, et me minimeerime V_N üle treeningandmete. Seega tuleb arvestada ka võimalusega, et neurovõrku saab üle treenida (*ing. k. overfitting*). Üldiselt öeldakse, et neurovõrk on ületreenitud siis, kui see suudab paremini ennustada neid andmeid, mis on teada (treeningandmeid) ja kehvemini neid, mida sellel ei ole teada (test- ja valideerimisandmed), see tähendab, et vahe vasaku ja parema poole vahel (3.11) on märgatav.

Joonisel 3.1 on kujutatud kolme funktsiooni ning andmete kogumisel saadud andmepunkte. $f(\mathbf{x})$ tähistab funktsiooni, mida otsime ning $g(\mathbf{x})$ ja $h(\mathbf{x})$ on funktsioonid, mida võime neurovõrku treenides saada. Mida rohkem me neurovõrku treenime, seda väiksemaks muutub ka empiiriline kadu V_N. Seega mida rohkem me neurovõrku treenime, seda enam läheneb otsitav funktsioon funktsioonile $h(\mathbf{x})$, st. neurovõrk kirjeldab funktsiooni, mis läbib võimalikult täpselt treeningpunkte. Nagu näeme ka jooniselt, siis neurovõrgu tulemused sõltuvad tugevalt treeningandmetest ning andmete kogumisel tekkinud müra mõjutab oluliselt neurovõrgu toimimist. Tõenäosus, et saavutatud funktsioon on ka süsteemi kirjeldav funktsioon on põhimõtteliselt olematu ning sellest tulenevalt ületreenimist välditakse.

Üle- ja alatreenimise vältimiseks tuleb arvestada kõrvalekalde (*ing. k. bias*) ja varieeruvuse (*ing. k. variance*) suhtega. Kui treenimisel on saadud mingi funktsioon, siis kõrvalekaldeks nimetatakse otsitava funktsiooni ning leitud funktsiooni erinevust. Varieeruvus näitab funktsiooni sõltuvust andmepunktidest. Kui neurovõrk on alatreenitud, siis varieerub treenimisel saadud funktsioon vähe, ent viga on väga suur, samas kui neurovõrk on ületreenitud, siis kõrvalekalle on väga väike ja varieeruvus väga suur.

Kõrvalekalde ja varieeruvuse suhte abil saab selgeks teha, kui hästi on neurovõrk süsteemi ära õppinud. Kui varieeruvus on äärmiselt väike ja kõrvalekalle väga suur, siis neurovõrgu struktuur ei sõltu andmetest absoluutselt ning seega on ebareaalne, et see suudaks süsteemi käitumist ennustada. Samas jällegi, kui neurovõrgul on treenimisel saadud kõrvalekalle väga väike ning varieeruvus lialt suur, siis on suhteliselt tõenäoline, et neurovõrk ei suuda ennustada süsteemi käitumist andmete muutumisel.

Neurovõrgu treenimise puhul on oluline teada, et igat neurovõrku ei saa treenida selliselt, et ta suudaks lõpuks ülesannet lahendada. Sellisel juhul tuleb muuta võrgu struktuuri ja erinevate seoste arvu, treenida võrk uuesti ning proovida leida sel moel võrk, mis suudaks antud ülesannet lahendada. Lihtsa näitena võib tuua

olukorra, kus me loome neurovõrgu, milles on üks neuron. Kuna ei ole võimalik, et ühe neuroniga võrk suudaks lahendada kõiki ülesandeid, siis sellest tulenevalt peab leiduma teistsuguse struktuuriga (erinevate neuronite tüübi, arvu ja seoste arvuga) võrk, mis suudab antud ülesannet lahendada.

Treenimise puhul on ka oluline andmehulk, kuna liialt väheste andmete puhul ei ole võimalik süsteemi treenimist lõpuni viia ning sel juhul ei suuda neurovõrk ülesannet lahendada. Kui andmeid on piisavalt palju, siis saab need jagada kahte ossa: treenimisandmed ja testandmed. Treeningandmeid kasutatakse neurovõrgu treenimisel ning testandmeid saab kasutada selleks, et kontrollida, kui hästi suudab neurovõrk töötada andmetega, mida ei ole tema treenimiseks kasutatud (vt. ka valemit (3.11)).

3.2 Enamlevinud optimeerimismeetodid

Miinimumi otsimine. Kui neurovõrku treenitakse ennustusvea meetodil, siis on eesmärgiks leida neurovõrgu kaalud nii, et oleks minimeeritud kriteerium

$$\hat{\boldsymbol{\theta}} = arg\,\underset{\boldsymbol{\theta}}{min} V_N\left(\boldsymbol{\theta}, Z^N\right), \tag{3.12}$$

mis tähendab seda, et me peame leidma argumendi $\boldsymbol{\theta}$, mille puhul on funktsiooni $V_N(\boldsymbol{\theta}, Z^N)$ väärtus minimaalne.

Neurovõrgu treenimisel leiame parameetrite $\boldsymbol{\theta}$ ja treeningandmete hulga Z_N ning teist järku Taylori rea laiendamiseks esitatud kriteeriumi $\boldsymbol{\theta}^\star$ põhjal

$$\begin{aligned} V_N\left(\boldsymbol{\theta}, Z^N\right) &\approx V_N\left(\boldsymbol{\theta}^\star, Z^N\right) + \left(\boldsymbol{\theta} - \boldsymbol{\theta}^\star\right)^T G\left(\boldsymbol{\theta}^\star\right) \\ &+ \frac{1}{2}\left(\boldsymbol{\theta} - \boldsymbol{\theta}^\star\right)^T H\left(\boldsymbol{\theta}^\star\right)\left(\boldsymbol{\theta} - \boldsymbol{\theta}^\star\right). \end{aligned} \tag{3.13}$$

G on gradientvektor punktis $\boldsymbol{\theta}^\star$ ja H on Hessiaan punktis $\boldsymbol{\theta}^\star$. Gradientvektor avaldub kujul

$$G\left(\boldsymbol{\theta}^\star\right) = \begin{bmatrix} \frac{\partial V_N(\boldsymbol{\theta}, Z^N)}{\partial \theta^{(1)}} \\ \frac{\partial V_N(\boldsymbol{\theta}, Z^N)}{\partial \theta^{(2)}} \\ \vdots \\ \frac{\partial V_N(\boldsymbol{\theta}, Z^N)}{\partial \theta^{(n)}} \end{bmatrix}, \tag{3.14}$$

ja tähistab funktsiooni muutumise suunda. Teist järku tuletise maatriks, Hessiaan,

on defineeritud järgnevalt

$$H\left(\boldsymbol{\theta}^{\star}\right)=\left[\begin{array}{cccc}\frac{\partial^{2}V(\boldsymbol{\theta},Z^{N})}{\partial\boldsymbol{\theta}^{(1)}\partial\boldsymbol{\theta}^{(1)}} & \frac{\partial^{2}V_{N}(\boldsymbol{\theta},Z^{N})}{\partial\boldsymbol{\theta}^{(1)}\partial\boldsymbol{\theta}^{(2)}} & \cdots & \frac{\partial^{2}V_{N}(\boldsymbol{\theta},Z^{N})}{\partial\boldsymbol{\theta}^{(1)}\partial\boldsymbol{\theta}^{(n)}}\\ \frac{\partial^{2}V(\boldsymbol{\theta},Z^{N})}{\partial\boldsymbol{\theta}^{(2)}\partial\boldsymbol{\theta}^{(1)}} & \frac{\partial^{2}V(\boldsymbol{\theta},Z^{N})}{\partial\boldsymbol{\theta}^{(2)}\partial\boldsymbol{\theta}^{(2)}} & \cdots & \frac{d^{2}V(\boldsymbol{\theta},Z^{N})}{\partial\boldsymbol{\theta}^{(2)}\partial\boldsymbol{\theta}^{(n)}}\\ \vdots & \vdots & \ddots & \vdots\\ \frac{\partial^{2}V(\boldsymbol{\theta},Z^{N})}{\partial\boldsymbol{\theta}^{(n)}\partial\boldsymbol{\theta}^{(1)}} & \frac{\partial^{2}V(\boldsymbol{\theta},Z^{N})}{\partial\boldsymbol{\theta}^{(n)}\partial\boldsymbol{\theta}^{(2)}} & \cdots & \frac{\partial^{2}V(\boldsymbol{\theta},Z^{N})}{\partial\boldsymbol{\theta}^{(n)}\partial\boldsymbol{\theta}^{(n)}}\end{array}\right]. \tag{3.15}$$

Hessiaan kirjeldab mitmeargumendilise funktsiooni kumerust.

Kui gradient on null, siis on funktsioon selles punktis statsionaarne. Kuna Hessiaan kirjeldab funktsiooni kumerust, siis kui Hessiaani maatriks on iga nullist erineva vektori korral määratud positiivne, siis on funktsioon ∪-kumer (*ing. k. convex cup*). Kui funktsioon on ∪-kumer ja statsionaarne, siis järelikult oleme jõudnud mingisse funktsiooni miinimumi.

Seega selleks, et oleks täidetud tingimus (3.12) on tarvis seda, et gradient oleks null ja Hessiaan oleks iga nullist erineva vektori v korral määratud postiivne, st.

$$G(\boldsymbol{\theta}^{\star}) = 0, \tag{3.16}$$

$$\mathbf{v}^{T}H(\boldsymbol{\theta}^{\star})\mathbf{v} > 0. \tag{3.17}$$

Miinimumi otsimist alustatakse sellega, et alguses määratakse parameetrid $\boldsymbol{\theta}^{(0)}$ esialgse arvamuse põhjal. Seejärel kohandatakse kaalusid vastavalt mingile treeningmeetodile. Kui kriteeriumiks on mittelineaarne funktsioon, siis miinimumi leitakse tavaliselt kasutades suundotsingu (*ing. k. line search*) meetodit. Selle meetodiga avaldub järgmine iteratsioon kujul

$$\boldsymbol{\theta}^{(i+1)}=\boldsymbol{\theta}^{(\mathbf{i})}+\mu^{(i)}\mathbf{f}^{(i)}, \tag{3.18}$$

kus $\boldsymbol{\theta}^{(i)}$ määratleb praeguse iteratsiooni (number i), $\mathbf{f}^{(i)}$määrab otsingu suuna ja $\mu^{(i)}$määrab sammu suuruse. Iteratsioone jätkatakse seni, kuni usutakse, et $\boldsymbol{\theta}^{(i)}$on piisavalt lähedal miinimumile $\hat{\boldsymbol{\theta}}$.

Kriteeriumeid (3.17) ja (3.16) täidab üldiselt rohkem kui üks miinimum, seega eelnevalt vaadeldud otsingu tüüpide puhul me ei saa garanteerida, et oleme leidnud globaalse miinimumi. See, milline miinimum saavutatakse, sõltub sellest, millised parameetrid $\boldsymbol{\theta}^{(0)}$ olid alguses valitud. Sellest tulenevalt on ka mõtet neurovõrku treenida mitu korda, et leida parim tulemus.

Suundotsingu meetodi koondumine miinimumiks sõltub sammu suuruse $\mu^{(i)}$ ja algparameetrite $\boldsymbol{\theta}^{(0)}$ valikust. Sammu suurus saab olla konstantne või muutuv. Konstantse sammu suuruse valimine ei pruugi olla kõige optimaalsem, kuna

sõltumata sammu suuruse valikust, on gradiendi meetodi koondumiskiirus lineaarne, seega eksisteerib $c \in [0,1]$ nii, et

$$\left|\boldsymbol{\theta}^{(i+1)} - \boldsymbol{\theta}^{\star}\right| \leq c \left|\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}^{\star}\right|. \tag{3.19}$$

Lineaarne koonduvuskiirus on halb sellepärast, et algoritmi täitmisaeg on aeglane.

Kui me valime liiga suure sammu, siis võib juhtuda, et me gradiendi suunas liikudes „astume" miinimumpunktist üle, ning sammu suurusest tulenevalt ei saa me jõuda kunagi miinimumini. Liiga väikese sammu suuruse valiku korral jõuame küll miinimumini, kuid see on liialt ajakulukas. Sellest tulenevalt oleks kõige mõistlikum valida iga sammu alguses suurus nii, et liikudes igal sammul gradiendi suunas, jõutakse vähimate sammudega miinimumi. Seda tehnikat nimetatakse muutuvaks sammuks.

Muutuvat sammu suurust kontrollitakse adaptiivselt. Näiteid adaptiivse sammu suuruse kontrolli kohta saab vaadata näiteks (Demuth and Beale, 1998, lk 162-164).

Gradiendi meetodi põhimõte seisneb selles, et parameetreid muudetakse gradiendile vastupidises suunas. Kuna gradient näitab, mis suunas funktsioon kõige rohkem suureneneb, siis gradiendile vastu liikumine on loogiline samm. Gradiendi meetodis valitakse otsingu suunaks $\mathbf{f}^{(i)} = -G(\boldsymbol{\theta}^{(i)})$. Seega avaldub valemist 3.18

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \mu^{(i)} G(\boldsymbol{\theta}^{(i)}). \tag{3.20}$$

Gauss - Newtoni meetod. Gauss-Newtoni algoritmi eeliseks teiste algoritmide ees on see, et funktsiooni teise tuletise arvutamine ei ole vajalik, millest tulenevalt on algoritm kiirem. Selle asemel kasutatakse Gauss-Newtoni meetodis ennustusvea

$$\varepsilon_i(\boldsymbol{\theta}) = y(\mathbf{x}_i) - \hat{y}(\mathbf{x}_i) \tag{3.21}$$

hinnangut. Gauss-Newtoni algoritmi puhul on vaja funktsiooni teist tuletist e. Hessiaani maatriksit, mis on $n \times n$ ruutmaatriks, kus n on neurovõrgu parameetrite arv. Selleks paneme tähele, et Hessiaani on kujul

$$R(\boldsymbol{\theta}^{(i)}) = \frac{1}{N} \sum_{t=1}^{N} \psi(t, \boldsymbol{\theta}^{(i)}) \psi^T(t, \boldsymbol{\theta}^{(i)}),$$

kus funktsioon ψ on ennustusvea (3.21) esimest järku tuletis. Saadud ruutmaatriksit nimetatakse *Gauss-Newtoni Hessiaan*'iks. Gauss-Newtoni meetodis aval-

dub gradient kujul

$$G(\boldsymbol{\theta}^{(i)}) = \frac{1}{N}\sum_{t=1}^{N}\psi(t,\boldsymbol{\theta}^{(i)})\varepsilon(t,\boldsymbol{\theta}). \tag{3.22}$$

Antud meetodis leitakse järgmine iteratsioon selliselt

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - R^{-1}\left(\boldsymbol{\theta}^{(i)}\right) G\left(\boldsymbol{\theta}^{(i)}\right). \tag{3.23}$$

Oluline on teada, et antud valemit saab kasutada siis, kui kaofunktsiooniks on valitud keskmine ruutviga (Nørgaard et al., 2000, lk 53-54).

Kuna pöördmaatriksi leidmine vajab palju arvutusi ning võib tekkida ka olukordi, kus Hessiaan on singulaarne, siis arvutatakse praktikas Gauss-Newtoni meetodis suund ilma pöördmaatriksit leidmata, lahendades lineaarvõrrandi

$$R(\boldsymbol{\theta}^{(i)})\mathbf{f}^{(i)} = -G(\boldsymbol{\theta}^{(i)}). \tag{3.24}$$

Antud meetodis on otsingu samm konstantne $\mu^{(i)} = 1$. Seoses konstantse otsingusammu probeemidega, oleks mõistlik Gauss-Newtoni meetodit täiendada suundotsingu meetodiga.

Pseudo-Newtoni meetod. Pseudo-Newtoni meetod on lihtsustatud Gauss-Newtoni meetod, mis rajaneb sellel, et Gauss-Newtoni Hessiaanil jäetakse välja arvutamata need elemendid, mis ei asetse diagonaalil. Sellega saavutatakse järgmine iteratsioon nii:

$$\boldsymbol{\theta}_k^{(i+1)} = \boldsymbol{\theta}_k^{(i)} - \mu^{(i)} G_k(\boldsymbol{\theta}^{(i)}) / R_{kk}(\boldsymbol{\theta}^{(i)}). \tag{3.25}$$

Antud meetodi eeliseks on see, et järgmise suuna valimiseks ei ole vaja teha palju arvutusi ning see meetod vajab vähem mälu, kuna arvutusteks on vaja ainult Hessiaani diagonaali. Puuduseks on meetodil see, et koonduvus on märksa aeglasem kui Gauss-Newtoni meetodi puhul.

Levenberg-Marguardti meetod. Gauss-Newtoni või Newtoni meetodi abil valitud otsingu suund ei pruugi alati optimaalne olla. Kuna Taylori aproksimatsioon on adekvaatne vaid teatud piirides, siis on mõistlik valida ümbruseks, kust me $L^{(i)}(\boldsymbol{\theta})$ miinimumi otsime, kera raadiusega $\delta^{(i)}$, seega minimeerimise probleemi saame formuleerida

$$\boldsymbol{\theta}^{(i+1)} = arg\min_{\boldsymbol{\theta}} L^{(i)}(\boldsymbol{\theta}),\ kus\ \left|\boldsymbol{\theta} - \boldsymbol{\theta}^{(i)}\right| \leq \delta^{(i)}, \tag{3.26}$$

kus $L^{(i)}(\boldsymbol{\theta})$ on kaofunktsioon sellel iteratsioonil. Lahendades piiratud optimiseerimise probleemi, saame uuendamise reegliks:

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} + \mathbf{f}(i) \tag{3.27}$$

$$\left[R\left(\theta^{(i)}\right) + \lambda^{(i)} I\right] f^{(i)} = -G\left(\theta^{(i)}\right), \tag{3.28}$$

kus $R\left(\theta^{(i)}\right)$ on Hessiaan'i maatriks, $G\left(\theta^{(i)}\right)$ on gradientvektor, $\mathbf{f}^{(i)}$ on otsingu suund, $\lambda^{(i)}$ on suurus, mida muudetakse algoritmi käigus ja I on ühikmaatriks. $\lambda^{(i)}$ja kera raadiuse $\delta^{(i)}$vahel on monotoonne seos, kuid üldiselt see ei ole lihtsalt leitav. Kui $\lambda^{(i)} = 0$, siis arvutatakse järgmine iteratsioon nii, nagu Gauss-Newtoni meetodis (3.24). Samas kui $\lambda \to \infty$, siis lahendatakse võrrand gradiendi meetodiga (3.20), kus sammu suurus läheneb nullile.

Algorithm 3.1 Levenberg-Marquardti algoritm

1. Valida algsed väärtused parameetrite vektorile $\theta^{(0)}$ ja algne väärtus $\lambda^{(0)}$
2. Teha kindlaks otsingu suund valemist (3.28).
3. $r^{(i)} > 0.75 \Rightarrow \lambda^{(i)} = \lambda^{(i)}/2$
4. $r^{(i)} < 0.25 \Rightarrow \lambda^{(i)} = 2\lambda^{(i)}$
5. Kui $V_N(\boldsymbol{\theta}^{(i)} + f^{(i)}, Z_N) < V_N(\boldsymbol{\theta}^{(i)}, Z_N)$ siis valida uueks iteratsiooniks $\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} + \mathbf{f}^{(i)}$ ja olgu $\lambda^{(i+1)} = \lambda^{(i)}$
6. Kui peatumise kriteerium (3.17 ja 3.16) ei ole täidetud, siis minna tagasi sammu 2 juurde.

Algoritmis 3.1 avaldub koefitsent $r^{(i)}$ kujul

$$r^{(i)} = \frac{V_N\left(\boldsymbol{\theta}, Z^N\right) - V_N\left(\boldsymbol{\theta}^{(i)} + f^{(i)}, Z^N\right)}{V_N\left(\boldsymbol{\theta}, Z^N\right) - L^{(i)}\left(\boldsymbol{\theta}^{(i)} + f^{(i)}\right)}. \tag{3.29}$$

Nagu valemist näeme, siis koefitsent $r^{(i)}$ väljendab empiirilise vea (3.9) ja kao (3.4) vahelist seost antud suunal. Vastavalt koefitsendi $r^{(i)}$ väärtusele muudetakse adaptiivselt kera raadiust, ehk kui antud suunas viga väheneb $r^{(i)} \to 1$, siis kera raadiust vähendatakse, ning kui viga suureneb $r^{(i)} \to 0$, siis kera raadiust suurendatakse.

3.3 Tulemuste valideerimine

Valideerimise abil kontrollitakse, kas treenitud mudel kirjeldab soovitud süsteemi adekvaatselt. Ideaalne võimalus neurovõrgu valideerimiseks on selle rakendamine reaalsele mudelile. Tihti pole see praktikas võimalik ning sellest tulenevalt võetakse kasutusele testandmed. Mudeli testimisel kasutatakse neid andmeid, mida ei kasutata mudeli treenimisel. Sellist andmehulka nimetatakse testvalimiks (*ing. k. test set*). On vajalik, et testandmed rahuldavad samu nõudmisi, mida rahuldasid treeningandmed. Peamine valideerimise kriteerium on empiiriline testviga

$$T_N = \frac{1}{N}\sum_{i=1}^{n} L(f(\mathbf{x}_i), y_i),$$

kus $L(f(\mathbf{x}_i), y_i)$ on kaofunktsioon ja $Z^N = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$ on testandmete hulk.

Teine kriteerium, mida valideerimisel kasutatakse, on keskmine üldistusviga. Ka keskmine üldistusviga on hea mudeli valideerimiseks, kuid selle peamine funktsioon avaldub mudeli struktuuri valikul. Üldistusviga on hea selleks, et kiiresti hinnata, milline mudeli struktuur on tõenäoliselt parim. Keskmine üldistusviga avaldub kujul

$$\hat{V}_M \simeq \frac{N + p_1}{N - p_1} V_N(\hat{\boldsymbol{\theta}}, Z^N),$$

kus N on andmepunktide arv, $V_N(\hat{\boldsymbol{\theta}}, Z^N)$ on empiiriline ennustusviga üle testhulga, $p_1$on koefitsent, mis kirjeldab neurovõrgu keerukust. Täpsemalt saab keskmise üldistusvea kohta lugeda raamatust (Nørgaard et al., 2000, lk 88-94).

Regulariseerimine ja varajase peatumise meetod. Neurovõrgu ületreenimise vältimiseks kasutatakse tihti regulariseerimist. Regulariseerimisel arvestatakse vea ja varieeruvuse suhet. Mida rohkem on neurovõrgul kaale, seda väiksem on viga. Samas kui neurovõrgul on kaale liiga palju, siis sellest tulenevalt suureneb neurovõrgu varieeruvus. Regulariseerimine tähendab seda, et selle asemel, et minimeerida kriteeriumit (3.9), lisatakse antud kriteeriumile juurde neurovõrgu keerukust väljendav suurus. Seega regulariseerimisel minimeeritakse kriteeriumit

$$W_N(\boldsymbol{\theta}, Z^N) = \frac{1}{2N}\sum_{i=1}^{n} L(f(\mathbf{x}_i), y_i)) + \frac{1}{2N}\boldsymbol{\theta}^T D\boldsymbol{\theta}, \tag{3.30}$$

kus $\boldsymbol{\theta}$ väljendab neurovõrgu parameetreid, $D = \alpha I$, kus I on ühimaatriks. Kujundlikult võib öelda, et regulariseerimine on funktsiooni silumine.

Selleks, et saada sarnast efekti, mille saavutame regulariseerimisel, on võima-

lik neurovõrgu treenimine katkestada enne seda, kui oleme leidnud lokaalse miinimumi. Seda nimetataksegi varajaseks peatumismeetodiks (*ing. k. early stopping*). Kui treenida neurovõrku Levenberg-Marquardt'i meetodiga, siis märkame, et treeningviga ja testviga ei ole samaväärsed. Levenberg-Marquardt'i algoritmist tulenevalt on treeningviga monotoonselt kahanev funktsioon, seejuures testveal sellist omadust ei ole.

Praktiline on kasutada varajase peatamise meetodit selleks, et hoida ära võrgu ületreenimist ning treening katkestada siis, kui testviga on saavutanud miinimumi. Kuigi varajase peatamise ja regulariseerimise lähenemised on üsna sarnased, siis sellest hoolimata eelistatakse kasutada regulariseerimist. Regulariseerimise kohta saab lugeda raamatust (Bishop, 1995, lk 338 - 346).

Neurovõrkude pügamine. Keskne probleem masinõppel on süsteemi keerukuse minimeerimine. Neurovõrgu pügamisega saame vähendada ületreenituse probleemi. Samas jällegi, kui ühenduste arv on liiga väike, siis võib juhtuda, et neurovõrk ei suuda treeningandmeid ära õppida. Kui meil on treenitud neurovõrk, millel on liiga palju seoseid, tekivad järgmised küsimused: Milliseid seoseid eemaldada? Kuidas allesjäävaid seoseid kohandada nii, et nad toimiksid võimalikult hästi? Kuidas pügada neurovõrku nii, et see ei võtaks arvutuslikult liiga palju aega?

Kui neurovõrgul on palju parameetreid ning neurovõrgu treenimiseks on vähe andmeid, siis saadud mudeli varieeruvus on suur. Samas kui parameetreid on jällegi vähe, siis on tulemuseks mudeli suur süstemaatiline viga. Seega kui andmeid on vähe, on mõistlik kõigepealt luua mudel, millel on palju parameetreid ning seejärel jätta alles ainult mudeli olulised komponendid. Järgnevalt vaatleme kahte neurovõrgu pügamise algoritmi.

Üks võimalik meetod on kaotada ära need kaalud, mille väärtus on vähim. Paraku kaasneb sellise lahendusega tihti valede kaalude eemaldamine, kuna väikesed kaalud ei pruugi alati olla ebaolulised.

OBS (*ing. k. Optimal Brain Surgeon*) meetodi põhimõte seisneb selles, et kustutatakse ära kaal, mille eemaldamisel väheneb lõplik ennustusviga kõige rohkem. Selleks, et sellist kaalu eemaldada, peab kõigepealt välja uurima, kuidas iga kaal lõplikku ennustusviga mõjutab ja seejärel peab vastava kaalu eemaldama.

OBD (*ing. k. Optimal Brain Damage*) kasutab hindamiskriteeriumina minimaalset treeningvea suurenemist kaalu eemaldamisel. OBD meetodi puuduste tõttu leiab algoritm harva kasutust. Täpsemalt saab neurovõrkude pügamise kohta lugeda (Nørgaard et al., 2000, lk 102-113).

Algorithm 3.2 Üldine neurovõrgu pügamise algoritm.

1. Valida ja treenida neurovõrgu mudel, mis on „piisavalt" suur
2. Arvutada välja testviga ja treeningviga. Kui mõlemad vead on saavutanud soovitud miinimumi, siis minna punkti 5.
3. Arvutada välja iga neurovõrgus oleva kaalu olulisus.
4. Kõige vähem oluline kaal eemaldada. Treenida võrk uuesti, salvestada kaalud, minna punkti 2.
5. Luua neurovõrk vastavalt etteantud arhitektuurile. Treenida võrk uuesti.

4 Dünaamilised süsteemid

Lühiülevaade. Dünaamilised süsteemid võivad olla pidevad ja diskreetsed. Pidevate süsteemide puhul muutub süsteemi olek ühtlaselt, kuid diskreetsete süsteemide puhul muutub olek kindla ajalise intervalli tagant. Dünaamiliseks nimetame süsteemi siis, kui tal on järgmised omadused:

1. süsteemil on olek $x(t)$, mis muutub ajas
2. süsteemil on sisend $u(t)$, mis muutub ajas
3. süsteemil on väljund $y(t)$, mis muutub ajas
4. sisendolek $x(t)$ ja sisend $u(t_1)$, kus $t_1 > t$, määrab süsteemi väljundi.

Edaspidi käsitleme antud töös diskreetseid süsteeme. Pidevad süsteemid muutuvad diskreetseks siis, kui me mõõdame nende olekuid kindlate ajavahemike tagant. Paljud süsteemid on lineaarsed või lähedased lineaarsetele siis, kui neid piisavalt suure sagedusega mõõta. Me nimetame diskreetset süsteemi lineaarseks siis, kui ta vastab järgmistele tingimustele. Olgu signaalid $\mathbf{y}_1$ ja $\mathbf{y}_2$ süsteemide

$$\begin{cases} \mathbf{x}[0] = \mathbf{x}_1 \\ \mathbf{u}[k] = \mathbf{u}_1[k], \quad k \geq 0 \end{cases} \quad ja \quad \begin{cases} \mathbf{x}[0] = \mathbf{x}_1 \\ \mathbf{u}[k] = \mathbf{u}_2[k], \quad k \geq 0 \end{cases} \tag{4.1}$$

väljundid. Siis sellisel juhul süsteemi

$$\begin{cases} \mathbf{x}[0] = \alpha_1\mathbf{x}_1 + \alpha_2\mathbf{x}_2 \\ \mathbf{u}[k] = \alpha_1\mathbf{u}_1[k] + \alpha_2\mathbf{u}_2[k], \quad k \geq 0 \end{cases} \tag{4.2}$$

väljundit saab esitada lineaarse kombinatsioonina

$$\mathbf{y}[k] = \alpha_1 \mathbf{y}_1[k] + \alpha_2 \mathbf{y}_2[k]. \tag{4.3}$$

Lühidal on lineaarne süsteem nii homogeenne kui ka aditiivne. Homogeensus tähendab seda, et kui süsteemi sisend suureneb, siis peab ka süsteemi väljund suurenema sama palju kordi. Näiteks kui inimese hääl muutub kaks korda kõvemaks, siis eeldusel, et kuulmine on lineaarne süsteem, peaks ka see reageerima kaks korda tugevamalt. Samamoodi, kui süsteemi sisend väheneb kaks korda, siis peab ka süsteemi väljund kaks korda vähenema. Süsteemi aditiivsuse omadus tähendab seda, et reaktsioon kahe signaali summale on võrdne kummagi signaali põhjustatud reaktsioonide summaga.

Täpsemalt saab dünaamiliste süsteemide kohta lugeda raamatust (Chen, 1998) ja loengumaterjalidest (Laur, 2008).

BIBO ja asümptootiline stabiilsus. Olgu meil lineaarne süsteem, mille väljund on sisendsignaali $\mathbf{u} \equiv 0$ korral $\mathbf{y}_{zi}$ (*ing. k. zero-input*), ning olgu süsteemi väljund seisundi $\mathbf{x}_0 = 0$ korral $\mathbf{y}_{zs}$ (*ing. k. zero-state*). Seega saame avaldada süsteemi väljundi $\mathbf{u} \equiv 0$ korral

$$\mathbf{y}_{zi} = \mathbf{y}_{zi}[\mathbf{x}_0, k], \tag{4.4}$$

kus $\mathbf{x}_0$ on süsteemi algolek ning k ajahetk. Ning süsteemi väljund avaldub nullseisundi $\mathbf{x}_0 = 0$ korral kujul

$$\mathbf{y}_{zs} = \mathbf{y}_{zs}[\mathbf{u}, k], \tag{4.5}$$

kus $\mathbf{u}$ on kontrollsignaal ning k ajahetk. Valemitest (4.1) - (4.5) näeme, et saame süsteemi väljundi iga sisendseisundi $\mathbf{x}_0$ korral avaldada kujul

$$\mathbf{y} = \mathbf{y}_{zi} + \mathbf{y}_{zs}. \tag{4.6}$$

Reeglina on lihtne kontrollida süsteemi väljundit $\mathbf{y}_{zs}$, st. leida iga referentssignaali $\mathbf{r}(k)$ korral sisendsignaal $\mathbf{u}$ nii, et

$$\mathbf{y}_{zs}[\mathbf{u}, k] = \mathbf{r}[k]. \tag{4.7}$$

Kuna süsteem ei pea alguses olema null-olekus, siis sama kontrollsignaali korral on süsteemi tegelik väljund

$$\mathbf{y} = \mathbf{r}[k] + \mathbf{y}_{zi}[\mathbf{x}_0, k]. \tag{4.8}$$

Valemist (4.8) näeme, et väljundsignaal koosneb kahest komponendist - referentssignaalist $\mathbf{r}[t]$ ja null-sisendi komponendist $\mathbf{y}_{zi}[\mathbf{x}_0, k]$, mida nimetame häirituseks või veaks võrreldes soovitud käitumisega. Selle komponendi suurust mõjutab süsteemi stabiilsus ning järgnevalt vaatamegi asümptootilist ja BIBO stabiilsust.

Lineaarne süsteem on asümptootiliselt stabiilne juhul, kui iga algoleku $\mathbf{x}_0$ korral tekib piiratud koste $\mathbf{x}[\cdot]$, mis läheneb nullile. Praktikas tähendab see seda, et kui süsteemil sisendid puuduvad, siis tema olek läheb nulli. See on oluline sellepärast, et kui süsteemi olek kasvaks, siis muutuks ta kontrollimatuks. Lineaarsete süsteemide omadusest lähtudes, kui süsteemi olek $\mathbf{x}$ läheb nulli, siis ka $\mathbf{y}_{zi}$ läheb nulli (4.4). Antud omadusest näeme, et kui me oleme disaininud kontrolleri $\mathbf{y}_{zs}$ jaoks (4.8), siis kui $\mathbf{x} \to 0$, ehk süsteemi olek läheneb nullile, siis järelikult ka komponent $\mathbf{y}_{zi}[\mathbf{x}_0, k]$ läheneb nullile, seega süsteemi väljundsignaal (4.8)

$$\mathbf{y} \to \mathbf{r}[k]$$

Süsteem on BIBO stabiilne (*ing. k. Bounded-Input Bounded-Output stable*) siis, kui kõikide piiratud sisendite $\mathbf{u}$ korral on süsteemi sisendseisundi väljund $\mathbf{y}_{zs}$ samuti piiratud. Nagu on näha, siis BIBO stabiilsus näitab $\mathbf{y}_{zs}$ komponendi omadusi, kuid ei ütle mitte midagi komponendi $\mathbf{y}_{zi}$ kohta. Õnneks saab näidata, et kui süsteem on BIBO stabiilne, siis on ta ka asümptootiliselt stabiilne, seega ka komponent $\mathbf{y}_{zi}$ on BIBO stabiilsuse puhul kahanev. Järelikult iga piiratud sisendi $\mathbf{u}_i < \infty$ ja süsteemi seisundi $\mathbf{x}_0$ korral on väljund $\mathbf{y}[\mathbf{x}]$ samuti piiratud. Rohkem saab süsteemi stabiilsuse kohta raamatust (Sarachik, 1997).

Kandefunktsioon. Kandefunktsiooniks nimetatakse funktsiooni, mis on süsteemi impulsskoste z-transformatsioon ehk Laplace'i transformatsioon diskreetsete süsteemide jaoks. Lineaarse süsteemi kontrollitavust mõjutavad kandefunktsiooni poolused ja nullid (*ing. k. poles and zeros*). Olgu meil lineaarse süsteemi kandefunktsioon $\hat{g}[z]$, mis avaldub ratsionaalse funktsioonina

$$\hat{g}[z] = \frac{N[z]}{D[z]},$$

kus $N[z]$ ja $D[z]$ on polünoomid ning $deg\, N[z] \leq deg\, D[z]$. Funktsiooni pooluseks me nimetame punkti z, mille korral

$$\hat{g}[z] = \infty \quad \Leftrightarrow \quad D[z] = 0$$

ja nullideks nimetame sellist punkti, kus

$$\hat{g}[z] = 0 \Leftrightarrow N[z] = 0.$$

Ehk lihtsamalt öeldes on funktsiooni poolused punktis, kus nimetaja on null ning ning funktsiooni nullid on punktis, kus lugeja on null.

Tagasisidestuseta kontroller. Tagasisidestuseta *ing. k. open-loop* kontroller on kontrolleri tüüp, millel puudub tagasisidestus, vaid otsused võetakse vastu vastavalt hetkeseisundile ja süsteemi mudelile. Sellest tulenevalt puudub kontroll selle üle, kas sisend saavutas soovitud eesmärgi või mitte. Lihtsaim tagasisidestuseta kontroller on näiteks kastmissüsteem, mis mingitel kindlatel tundidel käivitub, kontrollimata pinnase niiskusastet. Sel juhul võib juhtuda, et vihma sajab ning reaalselt poleks kastmine tarvilik, kuid süsteem seda ei kontrolli.

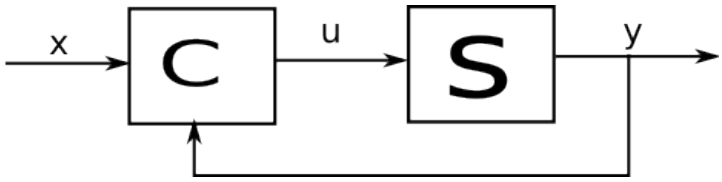
Tagasisidestuseta kontrolleri eeliseks on see, et seda on lihtne ja odav implementeerida.

Tagasisidestuseta kontrollerit on lineaarsetele süsteemidele lihtne konstureerida kasutades kandefunktsiooni. Kuigi süsteem võib olla lineaarne, siis ei pruugi see olla hästi kontrollitav tagasisidestuseta kontrolleriga. Esiteks on tihti keeruline teada täpseid kandefunktsiooni parameetreid ning teiseks ei saa luua täiuslikku kontrollerit. Sellest tulenevalt võib süsteemi lineaarsusest hoolimata kaduda tema üle kontroll. Täpsemalt võimaldab kandefunktsioon kirjeldada y_{zs}, seega peame eraldi modelleerima ka viga $y_{zi}(\varepsilon)$. Kui aga viga läheb välja nendest piiridest, mis on modelleeritud, siis kaotab tagasisidestuseta kontroller ka süsteemi üle kontrolli.

Tagasisidestuseta kontrollerit ei saa kasutada sellistel juhtudel, kus süsteemi käitumist võivad mõjutada ettearvamatud tegurid. Olgu meil tehases liin, mille kiirust me soovime hoida ühtlasel kiirusel. Tagasisidestuseta kontrolleri puhul me eeldame, et kui mootorisse anda mingi ühik volte, siis sellele vastab kindel liini liikumise kiirus, aga kui asetada liinile suure massiga asi, siis liini kiirus väheneb. Sellisel juhul tuleks mootori võimsust suurendada, kuid sellega tagasisidestuseta kontroller hakkama ei saa, kuna ta ei saa süsteemilt tagasisidet reaalse kiiruse kohta. Kirjeldatud olukorra puhul on süsteem küll endiselt stabiilne, kuid eksisteerib teatav kõrvalekalle.

Otsese tagasisidestusega kontroller. Olgu meil süsteem S ning kontroller C (vaata joonist 4.1). Kuna süsteemi väljund liigub ilma muutusteta või vaheplokkideta kontrollerini, siis sellist süsteemi nimetatakse otsese tagasisidestusega *ing. k. unity-feedback* süsteemiks. Tagasisidestusega kontrollereid kasutatakse sell-

istel puhkudel, kus tagasisidestuseta kontroller süsteemi kontrollimisega hakkama ei saa. Tagasisidestusega kontrollerite ehitus ja toimimismehhanism on keerukam kui tagasisidestuseta kontrolleritel.



Joonis 4.1: Otsese tagasisidestusega süsteem.

Otsese tagasisidestusega kontrolleri puhul on suur võimalus, et tekib müra. Müra saab süsteemi tekkida kolme signaali kaudu: väljundsignaali y, sisendsignaali x või kontrollsignaali u kaudu. Põhjuseid selleks võib olla mitmeid, näiteks väljundsignaalil võib olla teatav kõrvalekalle seetõttu, et sensor ei tööta hästi, samamoodi võivad ka kontrolleri ja süsteemi ühenduses olla mingid pisivead et cetera. Süsteemide modelleerimise ja mudelite testimise juures on väga oluline see, et kontroller suudaks süsteemi kontrolli all hoida ka siis, kui mingile signaalile lisatakse juurde müra. See on oluline seetõttu, et reaalses elus on müra puudumine äärmiselt ebatõenäoline ning kui kontroller ei suuda müra puhul süsteemi kontrollida, siis pole temast ka reaalse mudeli puhul kasu.

Eelmises peatükis toodud tehase liini näite puhul näeme, et tagasisidestuse puhul saab kontroller informatsiooni liini reaalse kiiruse kohta ning seetõttu saab mootori võimsust reguleerida. Populaarseimaks tagasisidestusega kontrolleriks on PID *(ing. k. Proportional–Integral–Derivative)* kontroller, mis kasutab süsteemi kontrollimisel vea parandamiseks kõrvalekallet, mis on oodatava väljundi x ja reaalse väljundi y vahe.

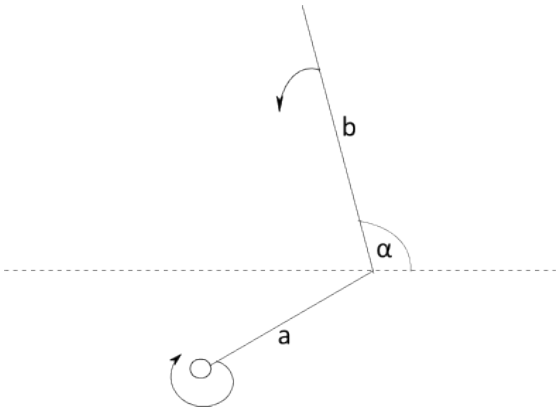
Otsese tagasisidestusega kontrollerite kohta saab lugeda raamatust Chen (1998, lk 277-281) ning dünaamiliste süsteemide stabiilsuse ja kontrollimise kohta saab lugeda loengumaterjalidest (Laur, 2008).

5 Praktiline töö

5.1 Süsteemi kirjeldus

Praktilise töö eesmärgiks on kontrollida üheastmelist inverteeritud pendlit. Inverteeritud pendel on levinud kontrolliteooria ülesanne, mida kasutatakse tihti kontrollialgoritmide testimiseks. Pendli mudel koosneb kolmest komponendist: pöörlevast mootorist, selle külge kinnitatud alusest ning aluse külge kinnitatud

pendlist (vaata joonist 5.1). Antud praktilise töö ülesandeks on hoida pendli ja maa vahelist nurka võimalikult lähedal üheksakümnele kraadile. Pendli asendit saab mõjutada mootori vasakule või paremale liikumisega.

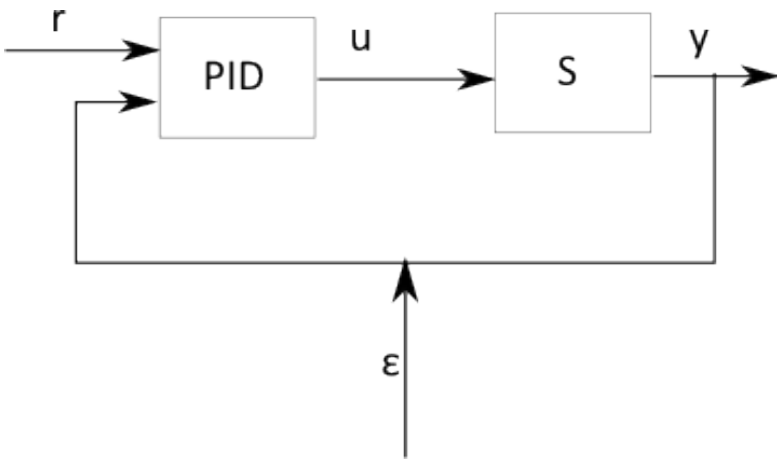


Joonis 5.1: Üheastmeline inverteeritud pendel, kus a tähistab pendli „kätt", b tähistab pendlit ning α maa ja pendli vahelist nurka.

Antud töös kasutasime kaheastmelise pendli *Simulink*'i mudelit (Nair, 2007), mis on muudetud üheastmelise pendli mudeliks. Töö lõpus on kirjeldatud ka põhjuseid, miks kaheastmelise pendli mudeliga praktilist osa realiseeritud ei ole. Neurovõrgu identifitseerimiseks töös kasutasime MATLAB'is realiseeritud neurovõrgu tööriista NNSYSID ja mudeli kontrollimiseks tööriista NNCTRL (Nørgaard et al., 2000).

Andmete kogumiseks kasutasime eelpool mainitud *Simulink*'i mudelit. Mudeli väljundite hulka kuuluvad: pendli nurk α, pendli kiirus, „käe" (joonisel 5.1 tähega b) kiirus ja „käe" nurk. Antud konktestis on meie jaoks vajalikuks väljundiks ainult pendli nurk, kuna sellega on üheselt määratud pendli vertikaalne seisund. Mudeli sisendiks on mootorile avaldatav jõud. Andmete kogumisel tuleb kõigepealt otsustada, mida neurovõrk tulevikus kontrollima peab. Antud juhul on eesmärgiks kontrollida pendli nurka maa suhtes. Mudeli puhul on süsteemi väljundiks 0 siis, kui pendel on maa suhtes 90 kraadise nurga all ja suunatud ülesse ning π siis, kui pendel on 90 kraadise nurga all ja suunatud maa poole. Seega saame antud mudeli puhul öelda, et pendel on püsti siis, kui süsteemi väljund $\alpha \approx 0$.

Selleks, et saada „kvaliteetsed" andmed, tuleb kõigepealt mudelit mingil viisil kontrollida. Selleks kasutame lihtsat ja väga levinud tagasisidestusega PID kontrolleritüüpi, mille kalibreerisime käsitsi, kasutades Ziegler–Nichols meetodit (Leigh, 1982).



Joonis 5.2: Andmete kogumisel kasutatud PID kontroller ja müra.

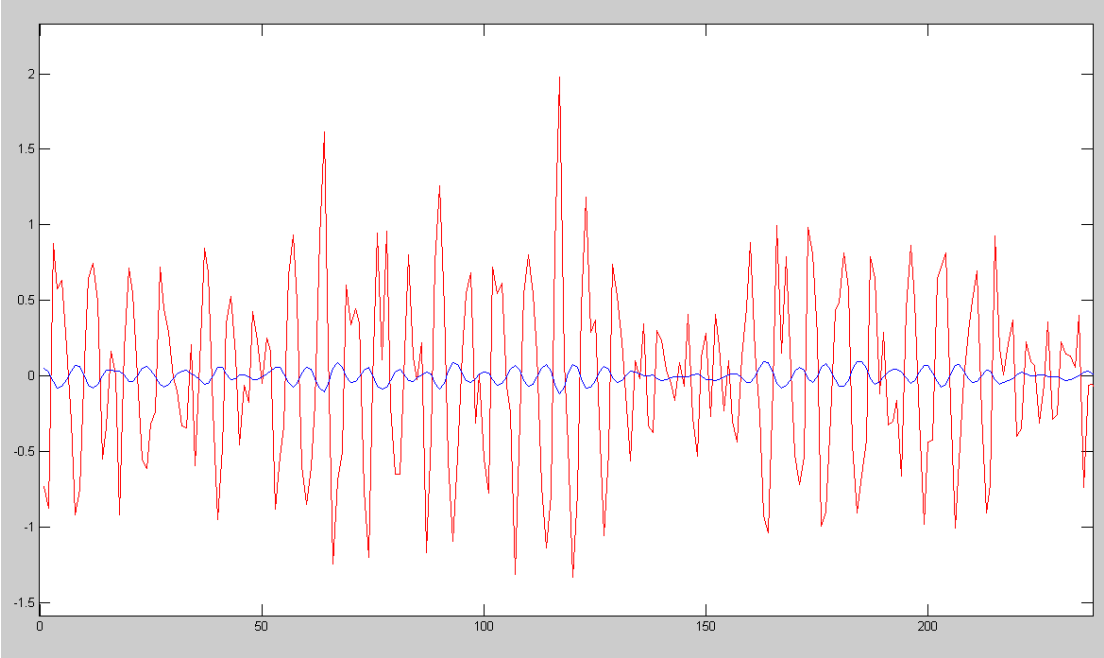
5.2 Andmete kogumine

Andmete kogumise eesmärk on saada sellised andmed, mis kirjeldavad süsteemi käitumist. Andmete kogumisel peame saama kaks tulemit - sisendid ja väljundid. Kuna neurovõrgu eesmärgiks on leida sisendite ja väljundite vaheline seos, siis ei ole meil mõtet koguda süsteemi kohta andmeid nii, et sisend puuduks või oleks ühtlane. Samamoodi ei ole meil mõtet koguda andmeid nii, et väljundid oleksid ühtlased. Kui me soovime tulevikus süsteemi kontrollida kindlas vahemikus, siis on mõistlik andmete kogumisel süsteemi kontrollida mõne lihtsa ja robustse kontrolleriga. Üks võimalus on anda admete kogumise ajal süsteemile sisendiks valget müra või kasutada sisendina signaali, mis muudab juhuslike ajavahemike tagant oma amplituudi. Viimast meetodit nimetatakse *random-walk* meetodiks. Rohkem saab andmete kogumise meetodi kohta lugeda raamatust (Nørgaard et al., 2000, lk 38 - 41). Andmeid tuleb koguda konstantse sagedusega, vastasel juhul ei ole põhjus-tagajärg suhe üheselt määratud. Diskreetsete süsteemide korral on see oluline sellepärast, et kui *sample*'i intervall on erinev, siis neurovõrk ei suuda kirjeldada seda, kui kiiresti süsteemi olek muutub. Seega kui me süsteemi hiljem kontrollima hakkame, siis ennustused tõenäoliselt ei ole õiged.

Praktilises töös on kasutatud andmete kogumiseks PID kontrollerit, mis hoiab pendlit enam-vähem tasakaalu asendi ümber (vt. joonis 5.2). Kuna kontroller töötas „liiga" hästi, siis sellest tulenevalt on sisendile eksperimendi käigus lisatud juurde ka valget müra, et pendlit natukene rohkem kõigutada. Antud andmete kogumise viis rahuldab meie nõudmisi - kui pendel kõigub, siis sisendsignaali muutused näitavad, kuidas süsteem erinevatele sisenditele reageerib. Andmete kogumine viidi läbi mudeliga, mille leiab tööga kaasas olevalt CD'lt asukohast *Mudelid/sinuglar _pid.mdl*.

Andmete kogumisel on valitud sageduseks 50 Hz ning kokku on kogutud 5001 andmepunkti. Kui andmeid on võimalik koguda palju, siis tuleks mõelda vea ja varieeruvuse suhtele. Mida rohkem on andmeid, seda suuremad võimalused on leida neurovõrk, mis suudab süsteemi hästi kirjeldada.

Andmete kogumisel saadud sisendsignaalid jäid lõiku $[-2,0812,\ 1,978]$, samas kui väljundsignaalid jäid lõiku $[-0,1422,\ 0,1524]$ ehk $[97,9°,\ 81,3°]$. Sisend- ja väljundsignaale on võimalik näha jooniselt (5.3).



Joonis 5.3: Esimesed 300 sisendit ja väljundit, kus punane on kontrollsignaal ja sinine on väljundsignaal.

Nagu jooniselt näeme, siis sisendsignaal muutub märksa aktiivsemalt kui väljundsignaal. Andmete osadeks jagamist saab vaadata failist *PrepareData.m*.

5.3 Süsteemi identifitseerimine

Süsteemi identifitseerimine on neurovõrgu loomisel kõige olulisem protsess. Süsteemi identifitseerimine on regressiooniülesande lahendamine (vt. peatükk 3.1). Süsteemi identifitseerimise eesmärk on leida selline funktsioon, mis kirjeldab süsteemi käitumist. Antud protsessis vaatleme erinevaid mudeli struktuure ja seejärel leiame konkreetse mudeli. Leitud neurovõrgu „headust" kontrollime kaofunktsiooni abil, milleks oleme valinud ruutvea $L(\hat{y},\ y) = (\hat{y} - y)^2$. Seega keskmine ruutviga avaldub kujul

$$V_N = \frac{1}{N}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

ja normaliseeritud ruutviga on

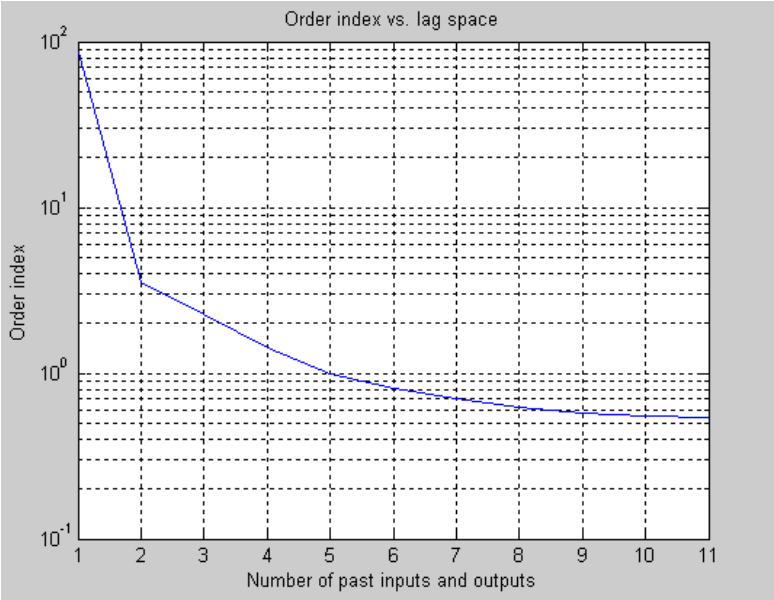
$$E_N = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} {y_i}^2} \cdot 100\%.$$

Antud töös analüüsime kõigepealt lineaarseid mudeli struktuure ning seda sellepärast, et lineaarsed mudeli struktuurid on robustsemad kui sigmoidsed. Seega analüüsime kõigepealt ARX'i, ARMAX'i ja OE mudeli struktuure. Kui leiame mõne struktuuri, mis suudab süsteemi käitumist hästi ennustada, siis lisame sellele mudelile ka sigmoidseid neuroneid, mis võivad tulemusi parandada. Sigmoidsete neuronite lisamine suurendab funktsioonide klassi, millest me süsteemi kirjeldavat funktsiooni otsime. Kui selle tulemusena oleme seatud tingimusi rahuldava neurovõrgu leidnud, siis tegeleme neurovõrgu optimeerimisega ning seejärel vaatame, kas antud neurovõrk suudab ka Simulink'i mudelit kontrollida.

Eelpool mainitud neurovõrkude loomiseks kasutame MATLAB'is realiseeritud tööriista NNSYSID, mis sisaldab erinevaid neurovõrgu struktuuri loomise, treenimis- ja valideerimisalgoritme. NNSYSID tööriist on hea vahend SISO (*ing. k. Single-Input Single-Output*) neurovõrkude loomiseks, MIMO (*ing. k. Multi-Input Multi-Output*) süsteemide identifitseerimist toetab vaid NNSSIF mudeli struktuuri realiseeriv funktsioon. Teiseks puuduseks on NNSYSID paketil see, et sisend- ja väljundandmeid saab funktsioonile ette anda vaid ühe vektorina. Üheks puuduseks, mis töö kirjutamise käigus avastasime, on see, et NNSSYID pakett ei suuda treenida selliseid neurovõrke, mis kasutavad erinevat arvu eelnevaid sisendeid ja väljundeid. Kuna iga eelneva sisendi või väljundi kasutamine lisab neurovõrgule keerukust, siis võib juhtuda, et osade neurovõrkude puhul oleks kasulikum kasutada rohkem sisendeid kui väljundeid. Antud tööriista korral peame kasutama mõlemaid võrdselt, seega on neurovõrgu optimeerimise võimalused piiratud. Näiteks saab NNSYSID pakett hakkama siis, kui nii eelnevaid sisendeid kui ka väljundeid kasutatakse 2, kuid algoritmid jooksevad kokku siis, kui eelnevaid sisendeid on näiteks 3 ja eelnevaid väljundeid 2.

Antud tööga kaasas oleva CD peal on funktsioonidele *nnarmax*, *nnarx*, ja *nnoe* realiseeritud võimalus sisestada maatriksi ja indeksivektori abil mitu aegrida, millega ka neurovõrgu treenimisel arvestatakse.

Enne konkreetsete neurovõrgu mudelitega tegelemist tuleb leida *lag*-struktuur, mis defineerib, mitu sisendit ja väljundit minevikust kasutatakse. Nende arvude leidmiseks on implementeeritud NNSYSID paketis Lipschitzi teoreemi põhjal *lag*-struktuuri välja pakkuv funktsioon *lipschit*, mis väljendab sisendite ja väljundite mõju süsteemile. Antud funktsiooni väljunditeks on Lipschitzi arvud ja ka visuaalsed graafikud. *Lag*-struktuuri leidmist saab vaadata failist *Identification/Find-*



Joonis 5.4: Lipschitzi arvude graafik

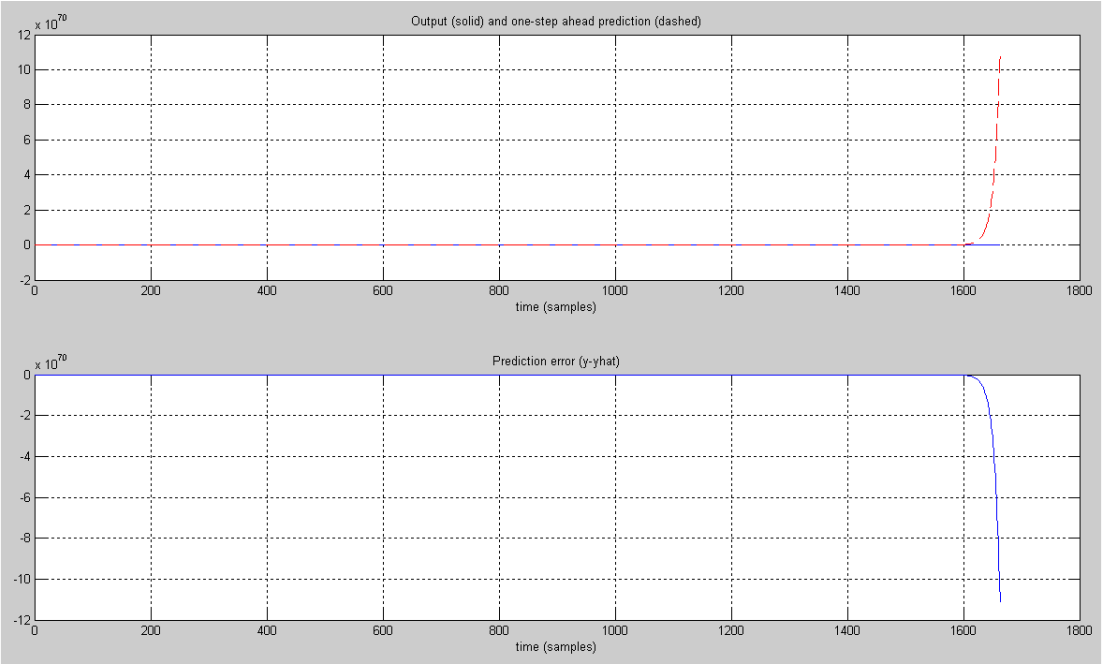
	1	2	3	4	5	6	7	8	9	10	11
1	86.7499	9.3578	7.4889	6.7783	6.1274	5.7533	5.6683	5.4761	5.3381	5.3157	5.2338
2	6.2515	3.5056	3.1244	3.1944	3.2413	3.2983	3.315	3.28	3.2033	3.1776	3.168
3	2.2201	2.3861	2.2705	2.1112	2.1016	2.1171	2.1167	2.149	2.1647	2.1549	2.1505
4	1.211	1.2805	1.3688	1.4163	1.4087	1.4243	1.4661	1.4844	1.5055	1.5167	1.5191
5	0.8099	0.8561	0.9067	0.9559	0.9889	1.016	1.0507	1.071	1.084	1.0966	1.1121
6	0.633	0.6692	0.7057	0.7417	0.7758	0.8043	0.8289	0.8548	0.8778	0.8961	0.9145
7	0.5375	0.5672	0.5966	0.6249	0.652	0.6774	0.7009	0.7232	0.7448	0.7639	0.7819
8	0.4687	0.4926	0.516	0.5386	0.5601	0.5808	0.6004	0.6187	0.6361	0.653	0.6692
9	0.4292	0.4494	0.4692	0.4881	0.5063	0.5237	0.5406	0.5568	0.5724	0.5873	0.6019
10	0.4111	0.429	0.4464	0.4631	0.4791	0.4946	0.5097	0.5242	0.5384	0.5518	0.5646
11	0.3991	0.4151	0.4306	0.4456	0.46	0.474	0.4876	0.5008	0.5136	0.5261	0.538

Joonis 5.5: Lipschitzi arvud

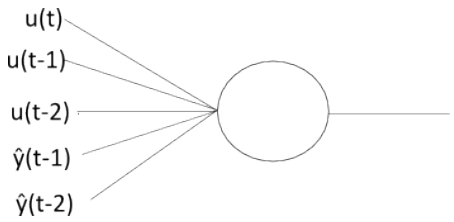
LagSpace.m (joonis 5.4 ja 5.5). Lipschitzi arvude tabelis on kirjeldatud 11 eelneva sisendi ja 11 eelneva väljundi Lipschitzi arvud. Ülevalt alla näitavad need arvud sisendite mõju ning paremalt vasakule näitavad väljundite mõju. Mida väiksem Lipschitzi arv on, seda tõenäolisemalt ennustab neurovõrk vastava *lag*-struktuuriga õigesti.

Jooniselt 5.4 näeme, et kõige mõistlikumaks arvuks eelnevate sisendite ja väljundite kasutamiseks on 2. Seda hindame selle järgi, kus punktis graafiku muutumine järsult väheneb. Teoreetiliselt peaks olema see kõige optimaalsem *lag*-struktuur, mida valida, kuid samas, vaadates Lipschitzi arve jooniselt 5.5, siis tegelikult saame märksa väiksemad arvud siis, kui me kasutame 4 sisendit ja väljundit, kuna sellest hetkest alates arvud enam märkimisväärselt ei vähene. Seega ei ole alati mõistlik hinnata *lag*-struktuuri visuaalse graafiku järgi, vaid seda tulemust tuleks ka kinnitada Lipschitzi arvude analüüsimisega.

Nüüd, kui *lag*-struktuur on leitud, vaatleme järgnevalt 3 lineaarset mudelit.



Joonis 5.7: OE mudeli treenimisel saadud neurovõrgu ennustuse täpsuse graafik.

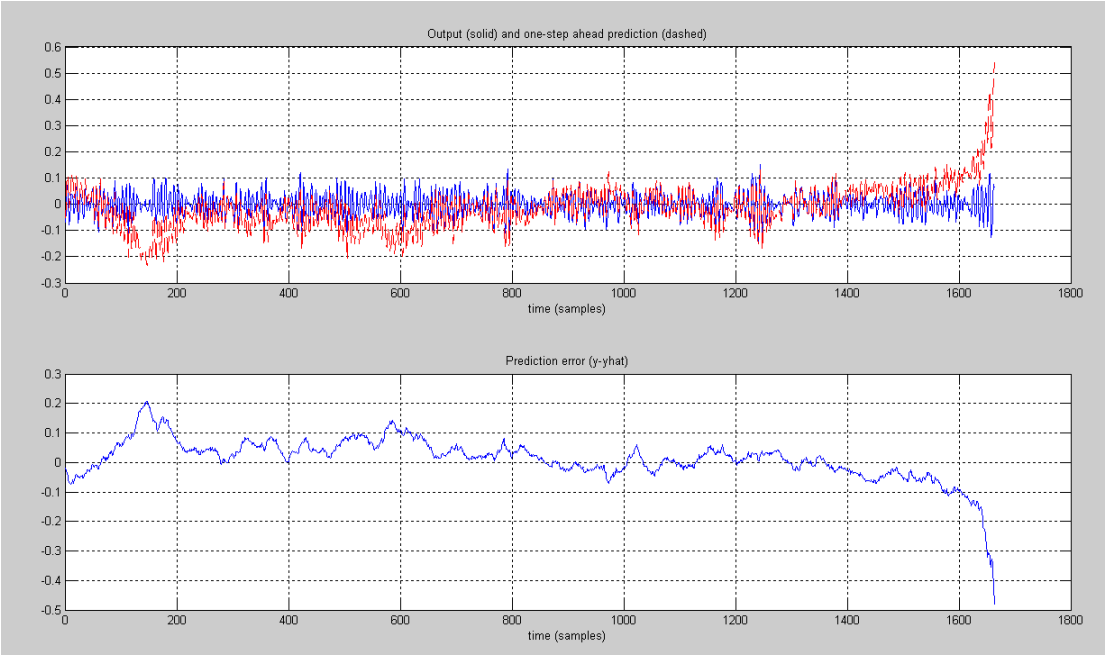


Joonis 5.6: OE mudeli joonis.

OE mudel. OE (*ing. k. Output Error*) mudel kasutab ennustuste tegemiseks eelnevaid sisendeid ja ennustusi (vt. joonis 5.6). OE mudeli loomise ja valideerimise leiab CD plaadilt failist *Identification/OE.m*. Selleks, et teada saada, kas antud mudel suudab süsteemi kirjeldada, tuleb kõigepealt luua neurovõrk ja see treeningvalimi abil treenida. Seejärel tuleb vaadata, kui hästi suudab neurovõrk ennustada süsteemi käitumist testvalimi peal. OE mudeli treenimiseks kasutasime funktsiooni *nnoe* ning treenintud neurovõrgu hindamiseks funktsiooni *nnvalid*. Lisaks sellele arvutasime välja ka ennustuste keskmise ruutvea funktsiooniga *meanSquaredrror*, mille leiab samanimelisest failist kaustast *Abifailid* (vt. ka 3.9).

Jooniselt 5.7 on näha lineaarse OE mudeli ennustusvõimet. Ennustuse normaliseeritud ruutveaks tuli OE mudelil $3,4142 \cdot 10^{143}\%$. Ilmselgelt näeme, et OE mudelit me süsteemi kontrollimiseks kasutada ei saa.

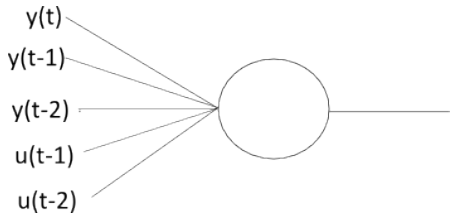
Hoolimata sellest, et treenitud OE mudel ei suuda süsteemi käitumist ennustada, siis proovime siiski OE mudelit treenida uuesti nii, et on juurde lisatud ka 6 *tanh* neuronit (vt. ka 2.3). Kuna funktsiooni lokaalseid miinimume võib olla mi-



Joonis 5.8: NNOE mudel sigmoidsete neuronite ja suurendatud *lag*-struktuuriga.

tu, siis treenisime neurovõrku 5 korda. Parimat tulemust, mis on saadud 6 tanh neuroni ja *lag*-struktuuriga 4 on näha joonisel 5.8.

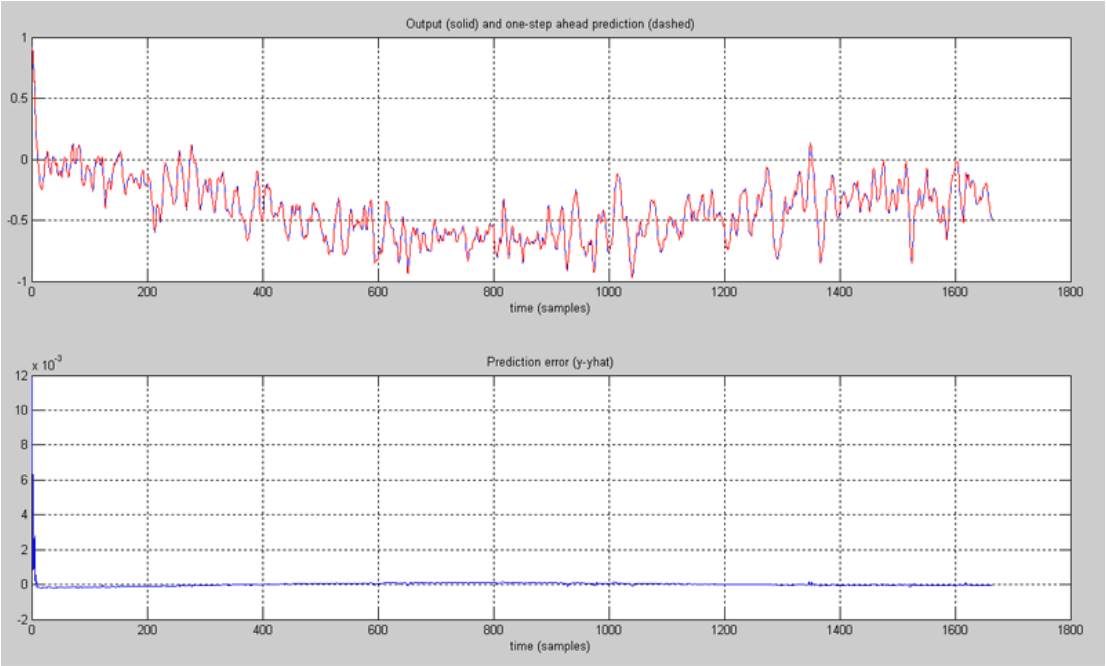
Hoolimata sellest, et saadud neurovõrk suudab mingeid süsteemi omadusi isegi kirjeldada, on ennustusvea keskmine ruutviga $262,1447\%$ siiski liialt kõrge.



Joonis 5.9: ARX mudeli joonis.

ARX mudel. Kuna OE mudel ei suutnud süsteemi kirjeldada, siis sellest tulenevalt leiame ARX mudeli. ARX (*ing. k. Autoregressive exogenous*) mudel kasutab ennustuste tegemiseks süsteemi eelnevaid sisendeid ja väljundeid. ARX mudeli puuduseks on tundlikkus häiritussignaali suhtes. Kui süsteemi kontrollimisel tekkiv müra ei ole valge müra, siis see kahandab ARX mudeli ennustuste täpsust.

ARX mudeli loomist ja valideerimist saab vaadata failist *Indentification/ARX.m*. Neurovõrgu lõime ja treenisime funktsiooniga *nnarx* ja tulemusi valideerisime funktsiooniga *nnvalid*. Funktsioon *nnvalid* annab tulemuseks graafikud, kust saab



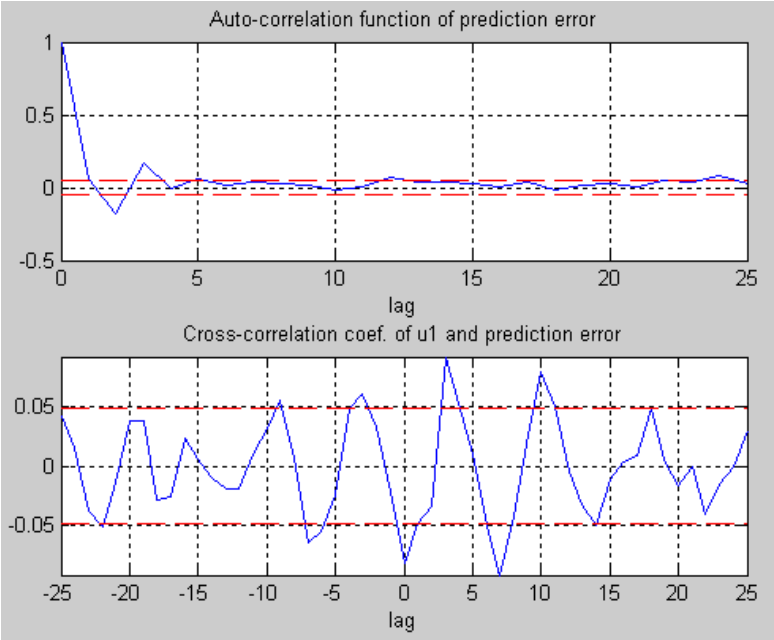
Joonis 5.10: ARX mudeli treenimisel saadud neurovõrgu ennustuse täpsuse graafik.

visuaalselt hinnata, kui hästi neurovõrk toimib.

Joonisel 5.10 on näidatud, kui hästi suudab treenitud mudel testvalimi puhul ühe sammu ette ennustada. Ülemisel graafikul on joonestatud ühele graafikule nii testandmed kui ka ennustuste väärtused. Alumisel graafikul on näha ennustusvea muutmist. Nagu näha, on ennustusviga suhteliselt madal, jäädes vahemikku $-2.5 \cdot 10^{-3}$ ja $5 \cdot 10^{-3}$ ning ennustuse normaliseeritud ruutveaks üle testvalimi on $0,0423\%$.

Joonisel 5.11 on samuti kaks graafikut. Need graafikud näitavad, kui tugevalt on andmed korreleeritud. Punased jooned on teoreetilised kvantiilid korreleeritud suurustele ning kui graafik jääb nende vahele, siis põhimõtteliselt võib öelda, et korrelatsioon on piisavalt nõrk. Ülemine graafik näitab ennustusvea autokorrelatsiooni tugevust ning alumine graafik näitab kontrollsignaali ja ennustusvea ristkorrelatsiooni tugevust. Autokorrelatsioon on halb sellepärast, et see näitab, et ennustusviga on sõltuv eelnevast ennustusveast.

Oluliseks mudeli omaduseks on ka k-sammu ennustamise võime. Selleks on NNSYSID tööriistakastis implementeeritud funktsioon *kpredict*, mille sisendparameetriteks on neurovõrgu parameetrid, struktuur ja andmevalim ning väljundiks on $YHat$, kus on neurovõrgu ennustused vastavalt andmevalimile. Funktsiooni *kpredict* puhul kasutame andmehulgana valideerimisvalimit. k-sammu ennustamise tulemusi saab näha tabelist 5.1 ning jooniselt 5.12 on näha ARX mudeli 3 sammu ennustamise tulemus.



Joonis 5.11: ARX mudeli valideerimisel saadud korrelatsioonigraafikud.

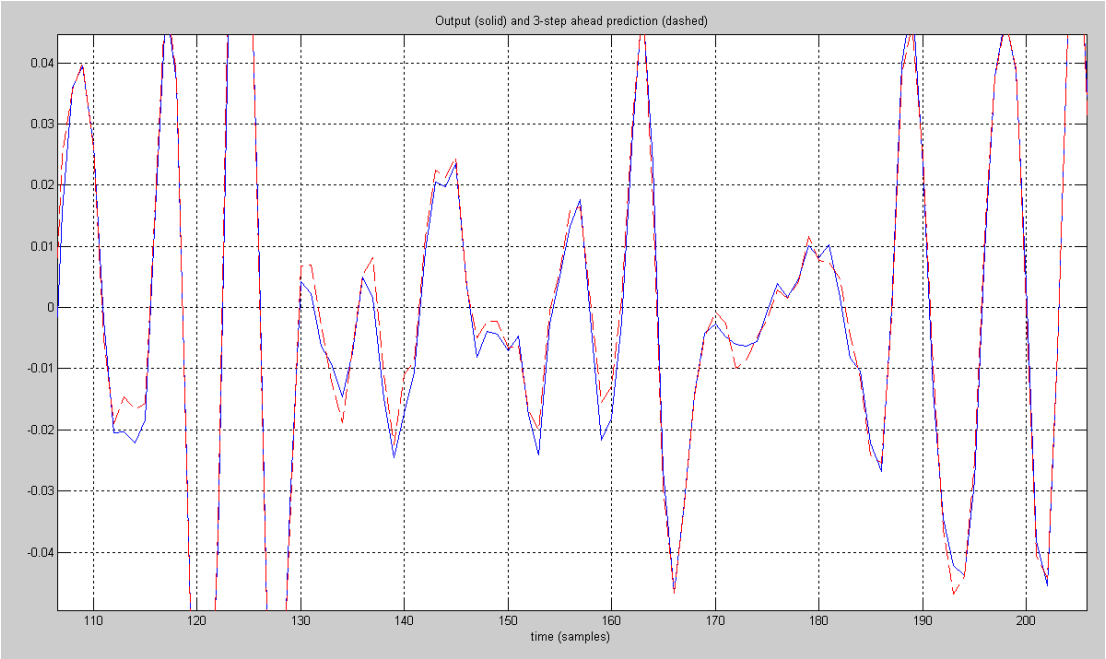
Sammude arv	**Ennustuse keskmine ruutviga**
1	0.3366%
3	1.049%
20	386.5531%

Tabel 5.1: ARX mudeli k-sammu ennustamise tabel

Nagu näeme, siis kuni kolm sammu ette ennustab mudel hästi. 20 sammu ennustamisel on viga juba märkimisväärselt suur. Olude sunnil tegime ka identifitseerimisprotsessi läbi 500Hz sagedusega. Mudeli võrdluseid saab vaadata tabelist 5.2. Ka tabelist on näha, et väikesel ajavahemikul suudab ka väga tiheda samplimisega kogutud andmetega treenitud neurovõrk täitsa hästi ennustada, kuid juba kaks või viis korda pikema aja jooksul kasvab viga märgatavalt, samas kui madalama samplimissageduse korral on pikemaajaline ennustusvõime parem. Seega kuigi ühe sammu ennustus on 500Hz mudelil parem, on tegelik ajalise usaldusväärsuse piir sellise samplimissagedusega mudeli korral oluliselt madalam kui 50Hz mudelil.

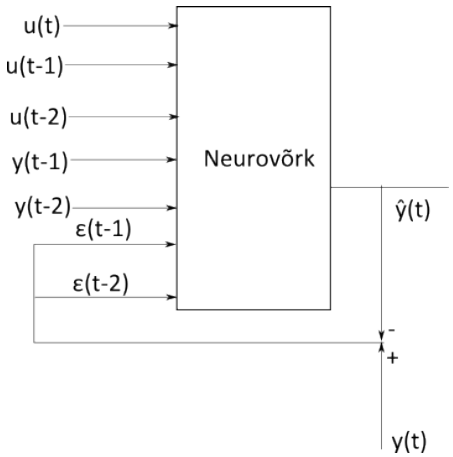
Sagedus\Aeg	0.02 sekundit	0.06 sekundit	0.1 sekundit	0.2 sekundit	0.5 sekundit
50Hz	0.043%	1.05%	4.1%	28.63%	1175.19%
500Hz	0.027%	1.17%	6.8%	87.7%	10007%

Tabel 5.2: 500Hz samplimissageduse ja 50Hz samplimissagedusega saadud neurovõrkude ennustusvõime võrdlus.



Joonis 5.12: ARX mudeli 20 sammu ennustamise graafik.

ARMAX mudel.

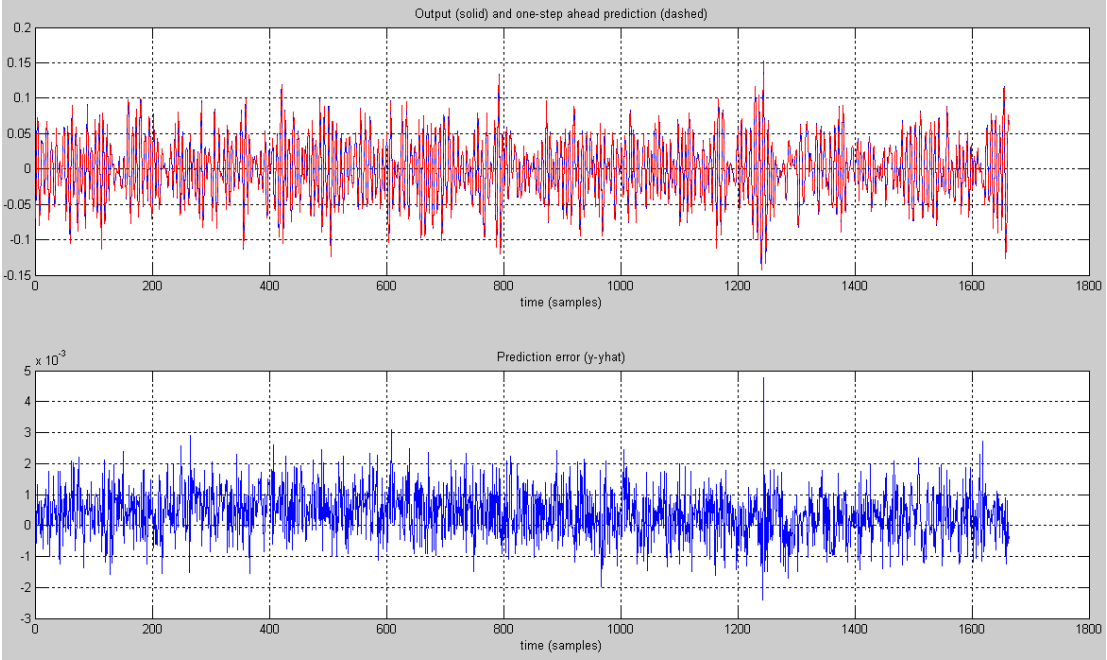


Joonis 5.13: ARMAX struktuuriga neurovõrgu joonis.

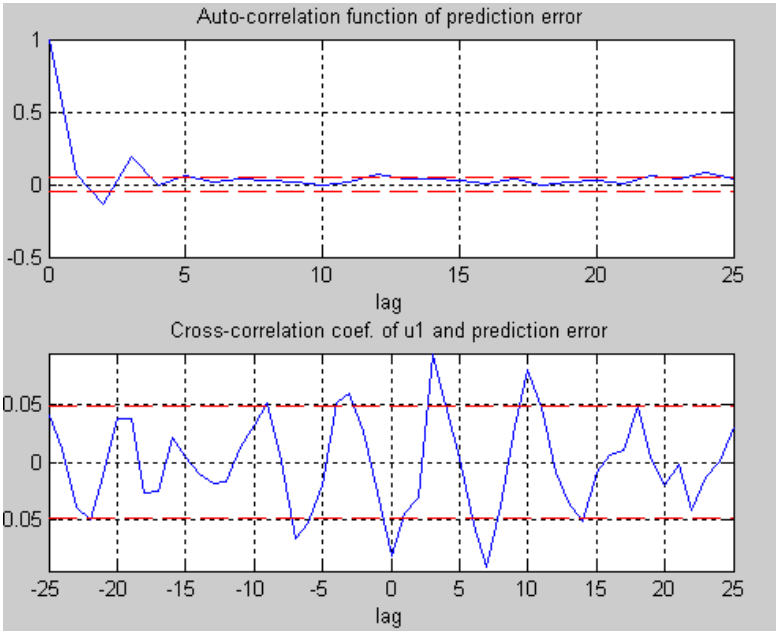
Kuigi ARX mudel suudab süsteemi käitumist väga hästi ennustada, siis sellest hoolimata vaatame, kuidas leida ARMAX mudelit. ARMAX (*ing. k. Autoregressive Moving Average exogenous*) mudel kasutab väljundite ennustamiseks eelnevaid sisendeid, väljundeid ja ennustusviga. ARMAX mudeli eeliseks ARX mudeli ees on madalam tundlikkus müra suhtes ning puuduseks on see, et ARMAX mudel on keerukam. ARMAX mudeli loomise ja valideerimise programmikoodi leiab failist *ARMAX.m*.

Analoogselt ARX mudelile, treenime kõigepealt neurovõrgu ära funktsiooniga

*nnarmax*2 ja seejärel valideerime saadud neurovõrku funktsiooniga *nnvalid* (vt. tulemusi joonistelt 5.14 ja 5.15).



Joonis 5.14: NNARMAX mudeli treenimisel saadud neurovõrgu ennustamise täpsuse graafik.



Joonis 5.15: NNARMAX mudeli treenimisel saadud neurovõrgu korrelatsioonigraafikud.

Võime öelda, et saadud tulemused on põhimõtteliselt samad, mis ARX mudeli struktuuri korral. Sama kehtib ka ennustuse normaliseeritud ruutvea kohta, mis

Sammude arv	Ennustuse normeeritud ruutviga
1	0.04%
3	1.07%
20	393.9%

Tabel 5.3: ARMAX mudeli k-sammu ennustamise keskmised ruutvead.

on 0.0432%. Arvestades nende mudeli struktuuride sarnasust, siis on see tulemus oodatav.

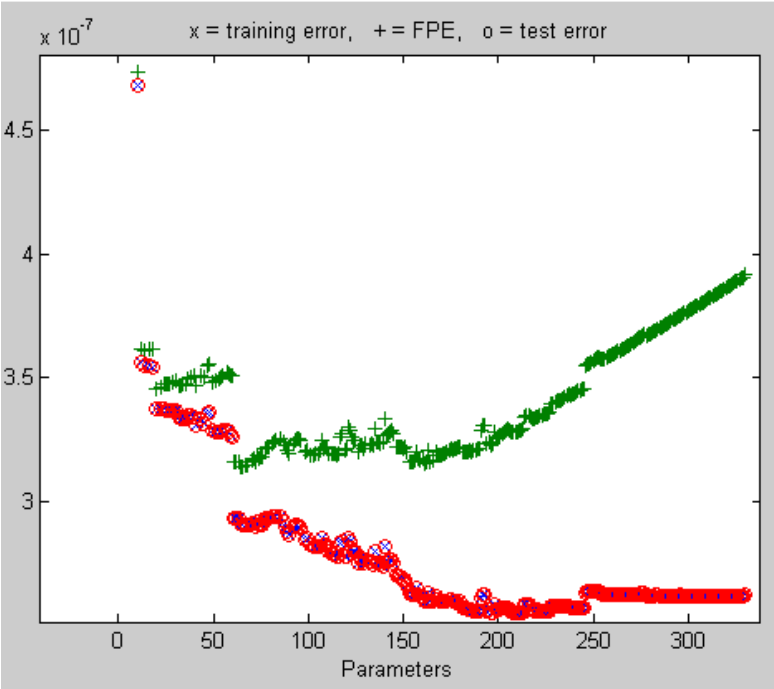
Analoogselt, nagu ARX mudeli korral, leidsime ka ARMAX mudeli korral 1,3 ja 20 sammu ennustamisel tekkiva normaliseeritud ruutvea. Tulemusi saab vaadata tabelist 5.3. Nagu võrdluses näeme, siis NNARMAX ja NNARX mudelid suudavad ennustada süsteemi käitumist sama hästi. Sellisel juhul, kui on kaks põhimõtteliselt sama head neurovõrgu mudelit, siis on mõistlikum valida see, mille struktuur on lihtsam. Seega edasiseks tegevuseks valime NNARX mudeli.

Neurovõrgu täiendamine. Valitud neurovõrgu mudeli struktuur suudab süsteemi omadusi kirjeldada hästi. Kuna aga süsteemil võib olla selliseid omadusi, mida lineaarse mudeliga kirjeldada ei saa, siis kasutades sedasama neurovõrgu struktuuri, lisame neurovõrgu peidetud kihti juurde 32 *tanh* neuronit. Antud neuronite arv on valitud suvaliselt. Põhimõtteliselt saab ka antud probleemile läheneda süstemaatiliselt, kuid ajakulu tõttu valisime lihtsalt intuitiivselt „piisavalt" suure neurovõrgu. Kui hiljem neurovõrku pügatakse, siis need kaalud, mis on ebavajalikud, eemaldatakse. Seega, pärast 32 *tanh* neuroni lisamist teeme läbi sama protsessi, mis lineaarsete mudelite puhul - treenime neurovõrgu ära ning vaatame kui hästi see testvalimi peal toimib.

Antud juhul ei ole ennustusvõime graafikud eraldi välja toodud, kuna ennustusviga märkimisväärselt suurenenud pole. Korrelatsioon on täpselt samamoodi mõningatest punktides üle usalduspiiri. Ennustuse normaliseeritud ruutveaks saime 0.05%, mis on küll veidi suurem, kui ARX mudelil, kuid ehk saab neurovõrgu ennustust ja korrelatsiooniprobleemi parandada võrgu pügamisega.

Neurovõrgu optimeerimine. Nüüd, kui meil on üsnagi suure struktuuriga neurovõrk, püüame neurovõrgu optimiseerimise teel vähendada ennustusvea ja kontrollsignaali ristkorrelatsiooni neurovõrgu pügamise teel. Neurovõrgu pügamiseks kasutame OBS algoritmi. Teoreetilises osas puudatasime seda teemat peatükis 3.3. Koodi neurovõrgu pügamise kohta leiab failist *Indetification/NNARXPRUNE.m*. Neurovõrgu optimeerimisega üritame saavutada seda, et antud struktuurist leida see neurovõrk, mille puhul test- ja treeningviga oleksid võimalikult võrdsed.

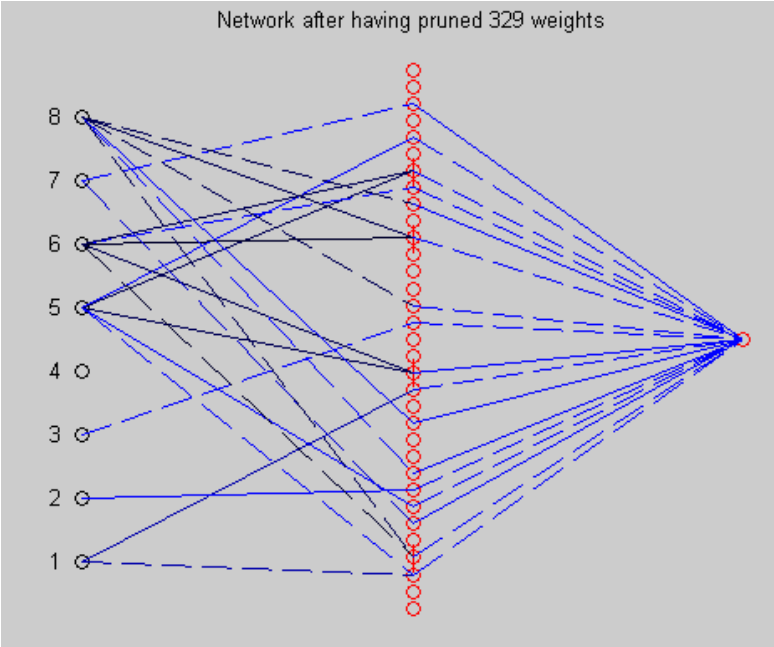
Joonisel 5.16 näeme neurovõrgu pügamise käigus loodud graafikut, kus on näidatud treening- ja testviga peale vastava parameetrite eemaldamist. Pärast nende tulemuste saamist võetakse välja see neurovõrk, mille korral on ennustusviga valdieerimisvalimil vähim. Jooniselt 5.17 näeme, milline näeb neurovõrk välja pärast seda, kui ta on pügatud.



Joonis 5.16: Neurovõrgu pügamise käigus saadud test ja treeningvead.

Joonisel 5.18 näeme pügatud neurovõrgu korrelatsioonigraafikuid. On näha, et ennustusvea autokorrelatsioon on märgatavalt tugevnenud ning sama kehtib ka ristkorrelatsiooni kohta. Suurenenud on tundmatute andmetega neurovõrgu ennustamise normaliseeritud ruutviga, milleks on 0.0613%. Antud juhul ei aidanud neurovõrgu pügamine eriti sellepärast, et treeningvalim on suur, 1667 andmepunkti. Näiteks 500 andmepunkti juures oleks neurovõrgu pügamisest suurem kasu. Alles jäi 17 aktiivset neuronit. Täielikul neurovõrgul, millel on 17 aktiivset neuronit, on enne pügamist ennustuse normaliseeritud ruutviga 0.0448%.

Kuna pügatud neurovõrgu korrelatsiooninäitajad ja ennustusviga on kehvemad kui lineaarsel ARX neurovõrgul, siis selleks, et otsustada, kumba neurovõrku edasi kasutada, vaatame, kui hästi nad sammude ennustamisega hakkama saavad. Tulemusi saab vaadata tabelist 5.4.



Joonis 5.17: Pügatud neurovõrk.

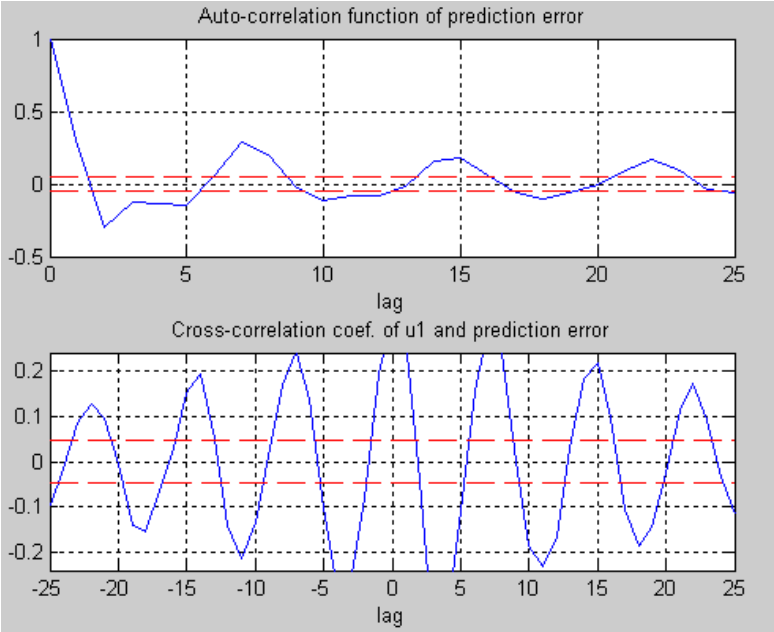
Mudel\Sammud	3 sammu ennus-tamisel saadud keskmine ruutviga	6 sammu ennus-tamisel saadud keskmine ruutviga	15 sammu ennus-tamisel saadud keskmine ruutviga	30 sammu ennus-tamisel saadud keskmine ruutviga
ARX	1.049%	6.60%	116.66%	3407.48%
NNARX	1.66%	13.094%	252.24%	672.97%

Tabel 5.4: NNARX ja ARX mudeli ennustamise võrdluste tabel.

Nagu näeme, siis kõige mõistlikum on jääda lineaarse ARX mudeli juurde, mille me leidsime kõige esimesena. Seega on meil neurovõrk identifitseeritud. Salvestame andmed $W1, W2, NN\ ja\ NetDef$ faili *NNARXFINAL.mat* ja kasutame seda neurovõrku edaspidi üheastmelise inverteeritud pendli mudeli kontrollimiseks.

5.4 Süsteemi kontrollimine

Antud peatükis kasutame Simulinki mudeli kontrollimiseks süsteemi identifitseerimise käigus saadud neurovõrku. Kõigepealt proovime, kas saame pendlit paigal hoida ning kui see kontrolleril õnnestub, siis vaatame, kui hästi suudab kontroller kastfunktsiooni jälgida ning kuidas ta müraga toime tuleb. Kontrollimisel kasutame kolme erinevat kontrollerit - *Optimal*, NPC (*ing.k. Nonlinear Predictive Control*) ja DIC (*ing.k. Direct Inverse Control*). Süsteemi kontrollimiseks kasutame



Joonis 5.18: Pügatud neurovõrgu korrelatsioonigraafikud.

NNCTRL tööriista.

NNCTRL pakett koosneb kolmest alamüksusest - CTRLDEMO, CTRLTOOL, TEMPLATE. Paketi idee seisneb selles, et TEMPLATE kaustas kirjeldatakse erinevad parameetreid - referentssignaalide generaator, ühe sampli pikkus, samplite arv, maksimaalne ja minimaalne kontrolli sisend et cetera. Kaustas CTRLTOOL on kontrollerite loogikat sisaldavad failid ning CTRLDEMO sisaldab faile, mis näitavad, kuidas antud tööriista kasutada.

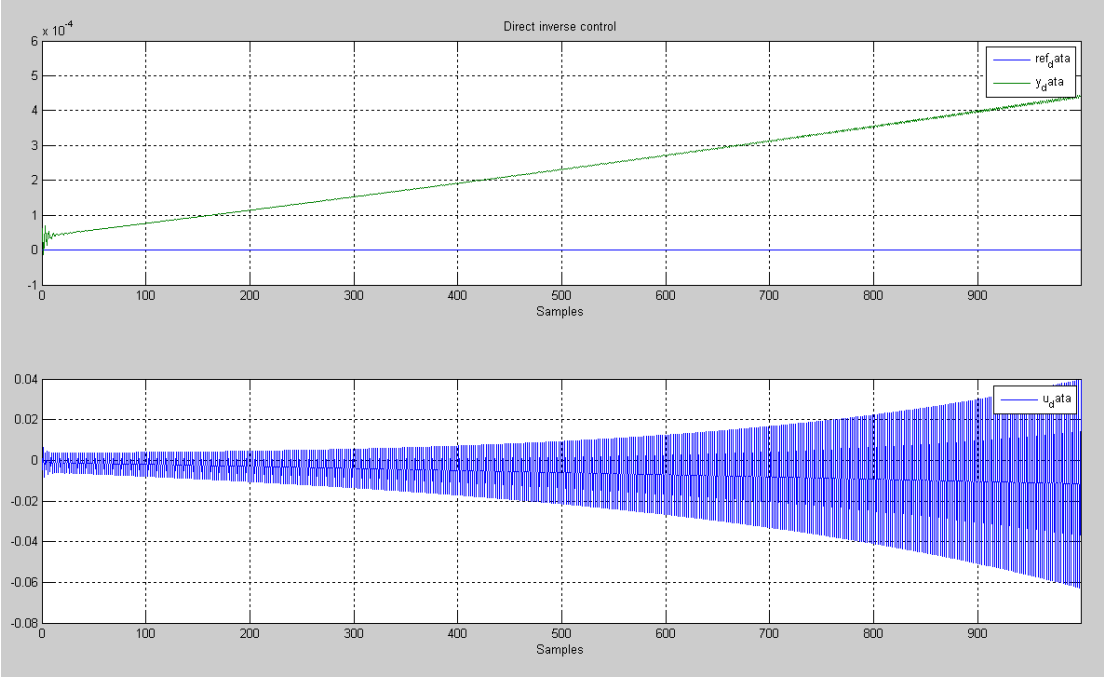
Kontrollitava mudeli leiab CD'lt *Mudelid/singular ctrl.mdl*. Kui on soov ka visuaalselt näha, kuidas kontroller mudeli kontrollimisega hakkama saab, siis tuleb antud mudel avada enne kontrolleri käivitamist.

DIC kontroller. DIC kontrolleri põhimõte seisneb selles, et ta kasutab nn. vastupidi töötavat neurovõrku mudeli kontrollimisel. Kui see neurovõrk, mille me leidsime identifitseerimise käigus, kasutas treenimisel sisendandmeid selleks, et leida väljundid, siis vastupidi töötaval neurovõrgu mudelil treenitakse neurovõrku nii, et väljundsignaalile seatakse vastavusse sisendsignaalid. DIC kontrolleri eripära teiste ees on see, et enne mudeli kontrollimist kasutatakse identifitseerimise käigus saadud neurovõrku kontrollitava mudeli kirjeldajana. Seda mudelit kasutatakse selleks, et luua selline neurovõrk, mis üritab kirjeldalda funktsiooni $f(y) = \mathbf{x}$, ehk luuakse tagurpidi mudel (*ing. k. inverse model*), mis üritab väljundi abil ennustada sisendeid. Identifitseerimise käigus saadud mudelit nimetatakse $forward$ mudeliks ning tagurpidi mudelit nimetatakse *inverse* mudeliks. Saadud *inverse*

mudelit treenitakse ka süsteemi kontrollimise käigus ja seda nimetatakse $on-line$ treeninguks. $Inverse$ kontrollerite puuduseks on see, et neid treenitakse kontrollimise ajal, millest tulenevalt see võtab rohkem ressurssi ning võib tekitada ka stabiilsusprobleeme, kui algoritm piisavalt kiiresti ei koondu. $Inverse$ mudeli leidsime *general* treeninguga, mille minimeerimiskriteeriumiks on

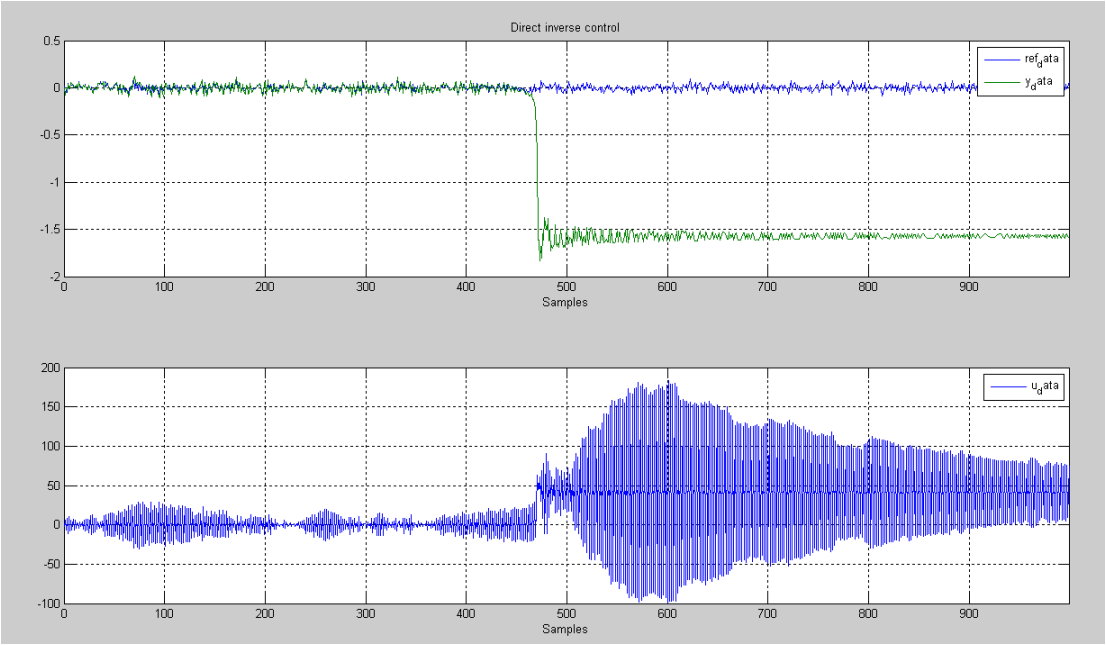
$$J(\boldsymbol{\theta}, Z^N) = \frac{1}{2N}\sum_{t=1}^{N}[u(t) - \hat{u}(t|\boldsymbol{\theta})]^2.$$

Antud kontrollerit saab konfigureerida failist *invinit.m*. Failis *invinit.m* tuleb defineerida mudeli *inverse* mudel ja $forward$ mudel. $Forward$ mudelina kasutame seda sama mudelit, mille saime süsteemi identifitseerimisel $Inverse$ mudeli leidmiseks tuleb konfigureerida fail $invinit1.m$ ning seejärel kirjutada MATLAB'i käsureale $special1$. Selle tulemusena saame treenitud *inverse* mudeli. Konfigureeritud faili saab vaadata CD'lt $Controllers/invinit1.m$. Kõigepealt püüdsime hoida pendlit püsti ilma mürata. Tulemus on näha graafikul 5.19.



Joonis 5.19: DIC kontrolleri tulemus ilma mürata ja konstantse 0 referentssignaaliga.

Analoogset graafikut näeme ka *optimal* kontrolleri kui ka NPC kontrolleri juures. Selleks, et näha, kuidas kontroller reaalselt töötab, lisasime kontrollerile juurde müra. Saadud tulemus on näha joonisel 5.20.



Joonis 5.20: DIC kontrolleri käitumine müraga, mille varieeruvus on 0.001.

Nagu jooniselt näeme, siis DIC kontroller ei suuda pendlit paigal hoida ning siis, kui pendli nurk läheb liiga suureks, kaob kontrolleril pendli üle kontroll.

Optimal kontroller. *Optimal* kontroller töötab samal põhimõttel, mis DIC kontroller, kuid nende kontrollerite erinevus on see, et *optimal* kontrolleril kasutatakse modifitseeritud treenimisalgoritmi (vt. täpsemalt lõigust *DIC kontroller*). *Optimal* kontroller kasutab teist kriteeriumit kui DIC kontroller, ta vaatab kui hästi väljundsignaal järgib referentssignaali ning jälgib ka seda, kui suur on sisendsignaali amplituud. See on hea sellepärast, et kui ka kontrollsignaali võimalikult nõrgana hoida, siis tehakse n.ö. vähim selleks, et süsteem kontrolli all oleks. Kui arvestada, et kontroller teeb vea mingi % ulatuses, siis tugevamate signaalide puhul on kõrvalekalle suurem. *Optimal* kontroller minimeerib kriteeriumit

$$J(\boldsymbol{\theta}) = \sum_t [r(t) - y(t)]^2 + \rho u^2(t) \quad \rho \geq 0.$$

Optimal kontroller kasutab mudeli kontrollimisel seda neurovõrku, mis on saadud identifitseerimise käigus. Kui meil on SISO (*ing. k. Single–Input Single–Output*) süsteem, siis võime kasutada $inverse$ ja $forward$ mudelina sama neurovõrku, mille oleme leidnud identifitseerimise käigus. Selleks, et saada nii $forward$ kui ka algne neurovõrk, teeme mudelist $NNARXFINAL$ kaks koopiat ning lisame $forward$ mudeli korral parameetritele $Netdef$, $W1$ ja $W2$ lõppu tähe f ning algsele mudelile lisame lõppu tähe c. Seega oleme saanud kolm faili: $NNAR$-

$MAXFINAL$, $NNARMAXFINALc$ ja $NNARMAXFINALf$. Teine võimalus on kasutada DIC kontrolleri eeltreenitud *inverse* mudelit.

Selleks, et kontrollida süsteemi *optimal* kontrolleriga, tuleb teha järgnevad sammud.

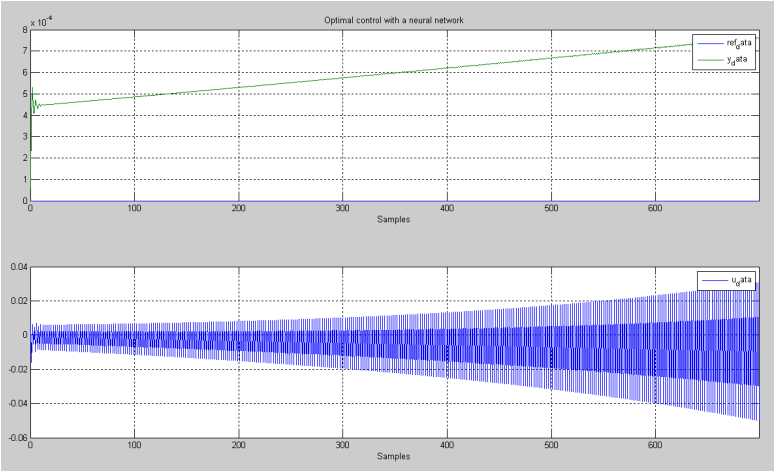
1. **Määrata dünaamilise süsteemi *inverse* mudel.** Kõigepealt tuleks konfigureerida faili *optrinit.m*. Failis *optrinit.m* tuleb muutuja $nnforw$ väärtuseks panna $NNARXFINALf$ ning *nnctr* muutuja väärtuseks $NNARXFINALc$. Ühtlasi tuleb sagedus muutujast Ts määrata 0.02 peale, kuna me oleme kogunud andmeid sama sagedusega. Nüüd, kui *optrinit.m* on konfigureeritud, tuleb MATLAB'i käsurealt sisestada käsk *opttrain*. Saadud neurovõrgu parameetrid $W1c, W2c, NetDefc$ ja NN tuleks salvestada faili $NNARMAXc$.

2. **Initsialiseerida inverse mudel.** *Optimal* kontrolleri seadistamiseks tuleb avada fail *optinit.m* kaustast TEMPLATE. Kõigepealt muudame parameetri $nnforw$ väärtuse $NNARMAXFINALf$ ning parameetrile $nnctrl$ väärtuse $NNARMAXFINALc$.

3. **Määrata treeningu parameetrid.** Kuna me soovime pendlit paigal olla, siis oluliseks sektsiooniks, millele tähelepanu pöörata on, $Reference\ signal$. Kuna me soovime, et pendel oleks püsti, kasutame referentssignaaliks DC signaali väärtusega 0. Ühtlasi tuleks tähelepanu pöörata kontrollimise sagedusele, mis peab olema sama, mis andmete kogumisel $50Hz$. Alguses me kontrollerile müra ei lisanud, seega muutuja $Nvar$ peab olema 0. Samplite arvuks valisime 500, kuna siis on enam vähem näha, kuidas kontroller pendliga käitub. Salvestame faili ja käivitame MATLAB'i käsurealt funktsiooni *optcon*.

Joonisel 5.21 on näha, kuidas süsteem antud kontrolleriga käitus. Nagu näeme, siis *optimal* kontroller suudab pendlit püsti hoida hästi, kuid näeme, et reaalselt hakkas pendel siiski „ära vajuma", kuna nurk aeglaselt kasvas. Selleks, et veenduda, et see seda ka „päriselt" suudab, lisame kontrollerile müra. Selleks tuleb omistada muutujale $Nvar$ mingi väärtus, antud juhul kasutasime väärtust 0.001. Saadud tulemusi saame vaadata jooniselt 5.22. Nagu näeme, siis *optimal* kontroller suutis müraga pendlit mingi aja kontrollida, kuid lõpuks väljus pendel kontrollitavast piirkonnast ja siis ei suutnud kontroller enam pendliga hakkama saada.

NPC kontroller. Mittelineaarne ennustav kontroller kasutab mudeli kontrollimisel identifitseerimise käigus saadud neurovõrgu ennustusi. Kontrolleri tööpõhimõte seisneb selles, et neurovõrgu ennustuse ja referentssignaali abil leitakse kontrollsignaal. Ehk siis neurovõrgule antakse sisendiks referentssignaal ning ennustuse abil korrigeeritakse kontrollsignaali nii, et oodatav väljund oleks sama, mis

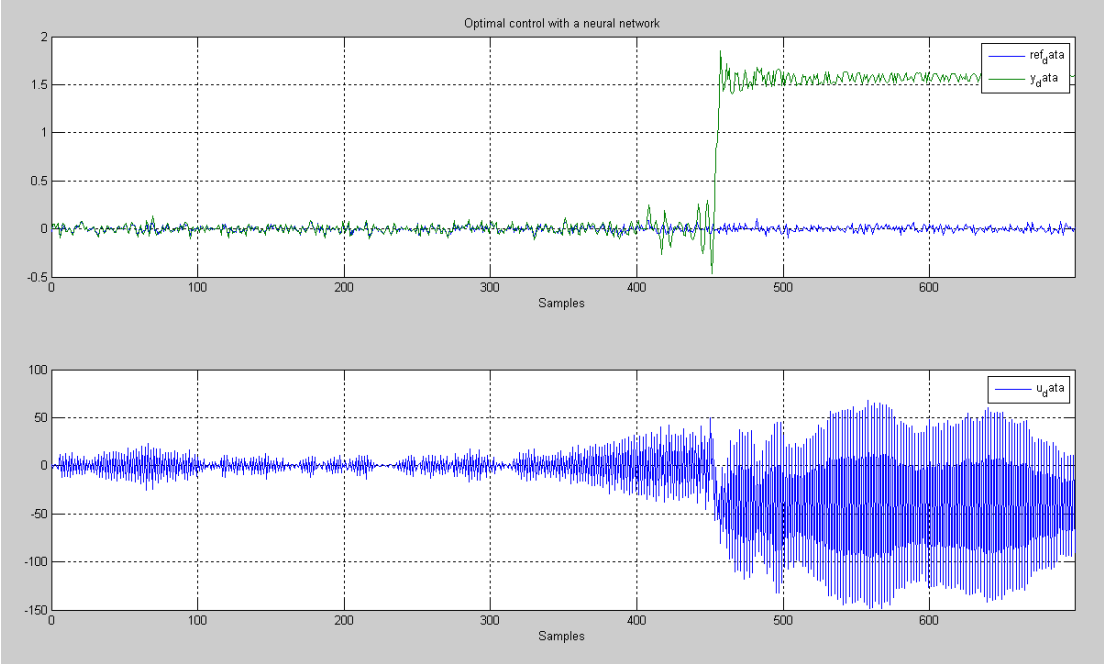


Joonis 5.21: Üheastmeline pendli kontrollimise tulemus 50Hz sagedusega.

referentssignaalil. Teoreetiliselt peaks ennustava kontrolli käsitlus lahendama probleeme, mis tekivad näiteks DIC'i või *optimal* kontrolli puhul. Esiteks ei pea neurovõrku süsteemi kontrollimise ajal uuesti treenima ning teiseks suudab mittelineaarne ennustav kontroll hakkama saada nendes olukordades, kus lineaarsed kontrollerid süsteemi keerukuse tõttu hakkama ei saa.

Analoogselt optimaalse kontrolliga, saab muuta NPC kontrolleri tööparameetreid failist $TEMPLATES/npcinit.m$. Selleks, et näha, kas NPC kontroller suudab pendlit paremini püsti hoida, tõstame samplite arvu 1000 sammu peale. Neurovõrgu failiks määrame $NNARXFINAL$. Muutuja $N1$ määrab minimaalse ennustuse horisondi, mis peab olema võrdne ajalise nihkega, ehk siis ühega. $N2$ ja Nu on vastavalt maksimaalne ennustushorisont ja kontrollimisel kasutatav eelnevate sisendite ja väljundite arv. NPC kontroller vaatab ennustustava väljundi ja referentssignaali käitumist vahemikus $N1$ ja $N2$. Muutuja $N2$ suurus mõjutab seda, mitut referentssignaali me teadma peame. Muutujate $N1$ ja $N2$ valik aitab kontrollsignaali adaptiivsemalt valida - otsitakse sellist kontrollsignaali, mis sunnib süsteemi käituma hästi üle terve vahemiku. Maksimaalne ennustushorisont Nu näitab seda, mitme sammu jooksul me peame süsteemi väljundi õigeks saama. Mida suurem on Nu, seda rohkem signaale meil kasutada on. Kõikide mainitud muutujate väärtused peavad olema sellised, et neurovõrk suudab nende raames adekvaatselt ennustada. $N2$ sammu ennustus peaks olema $10-20\%$ ennustusveaga ning Nu väärtus ei tohiks olla liiga suur, kuna vastasel juhul ei suuda kontroller jooksvalt optimeerida. Alguses määrame need võrdseks *lag*-struktuuriga, milleks on 4.

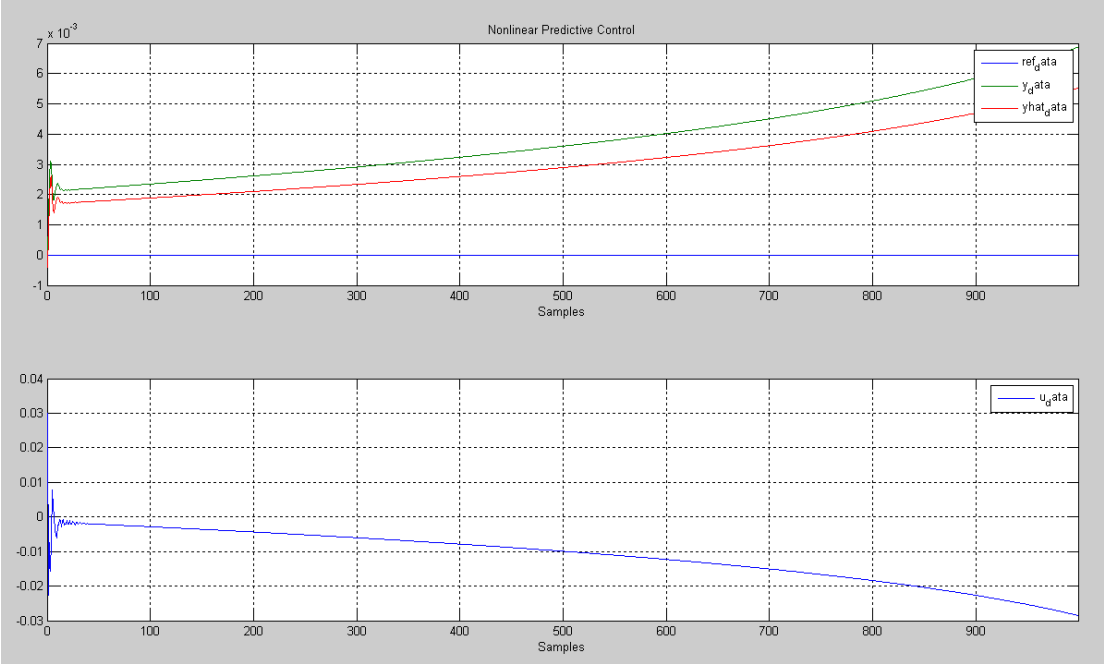
Referentssignaaliks määrame konstantse 0 signaali ning muutuja $Nvar$, mis reguleerib valget müra, määrame ka nulliks. Seejärel käivitame MATLAB'i käsu-



Joonis 5.22: Optimal kontroller 50Hz kontrollimissagedusega

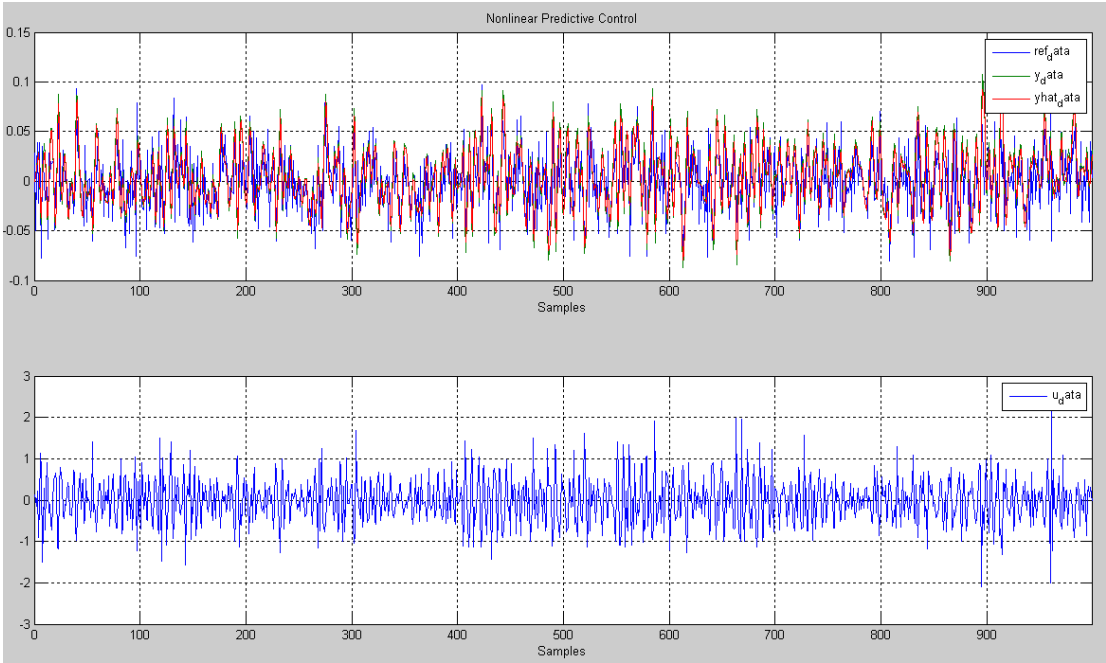
realt funktsiooni *npccon*2. NPC kontolleri skriptide juures avastasime, et kui neurovõrgus on ainult üks lineaarne neuron, siis skriptis tekivad vead. Selle probleemi lahenduseks on, et lisame neurovõrgule juurde 1 lineaarse neuroni ja treenime neurovõrgu uuesti ning salvestame faili $NNARXFINAL2L$ ning kasutame funktsiooni *npccon*2 asemel funktsiooni *npccon*1.

Antud kontrolliparameetritega tulemus on näha joonisel 5.23.



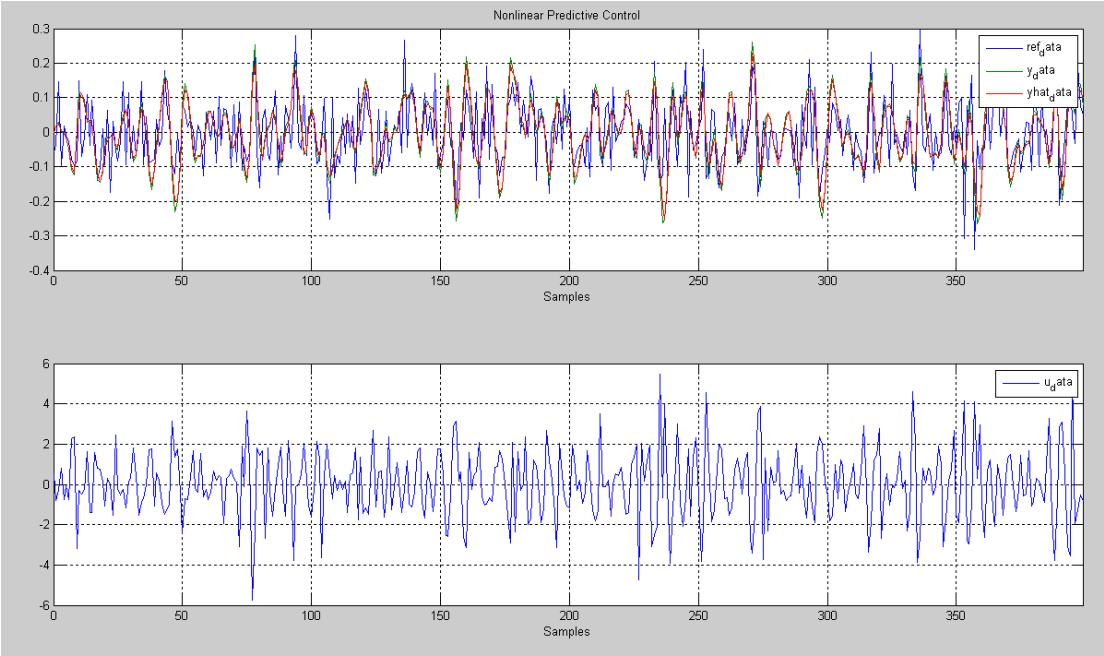
Joonis 5.23: NPC kontroller 50 Hz kontrollsagedusega.

Näeme, et sarnaselt *optimal* kontrollerile hakkas ka NPC kontrolleril pendel „ära vajuma", kuid lisame kontrollerile müra ja vaatame, mis on siis tulemuseks. Muutuja *Nvar*'i väärtuseks paneme 0.001 ning samplite arvuks 1000. Vaadates joonist 5.25, siis näeme, et NPC saab pendli kontrollimisega ka müraga väga hästi hakkama.



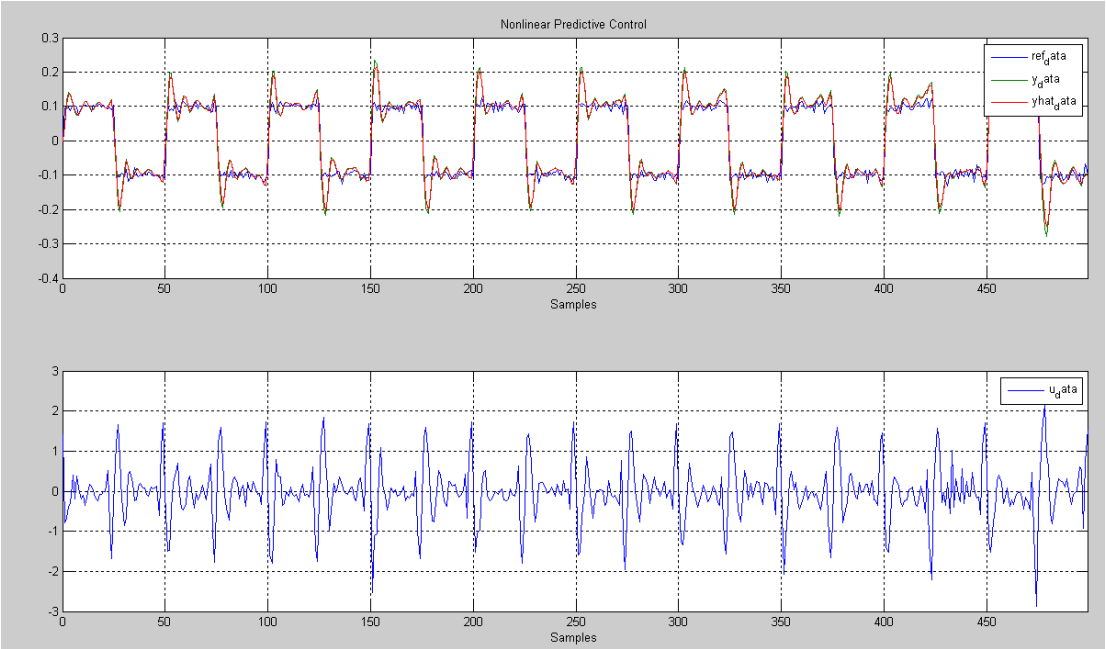
Joonis 5.24: NPC kontrolleri tulemus, kui signaalile on lisatud müra.

Huvi pärast suurendasime müra 10 korda ja vaatasime, kas NPC kontroller suudab ka siis pendlit paigal hoida. Tulemust näeb joonisel 5.25. Näeme, et kontroller saab ka selle müraga hästi hakkama. Seega saame öelda, et NPC kontroller suudab antud mudelit kontrollida.



Joonis 5.25: NPC kontrolleri tulemus, kui signaalile on lisatud 10 korda suurem müra.

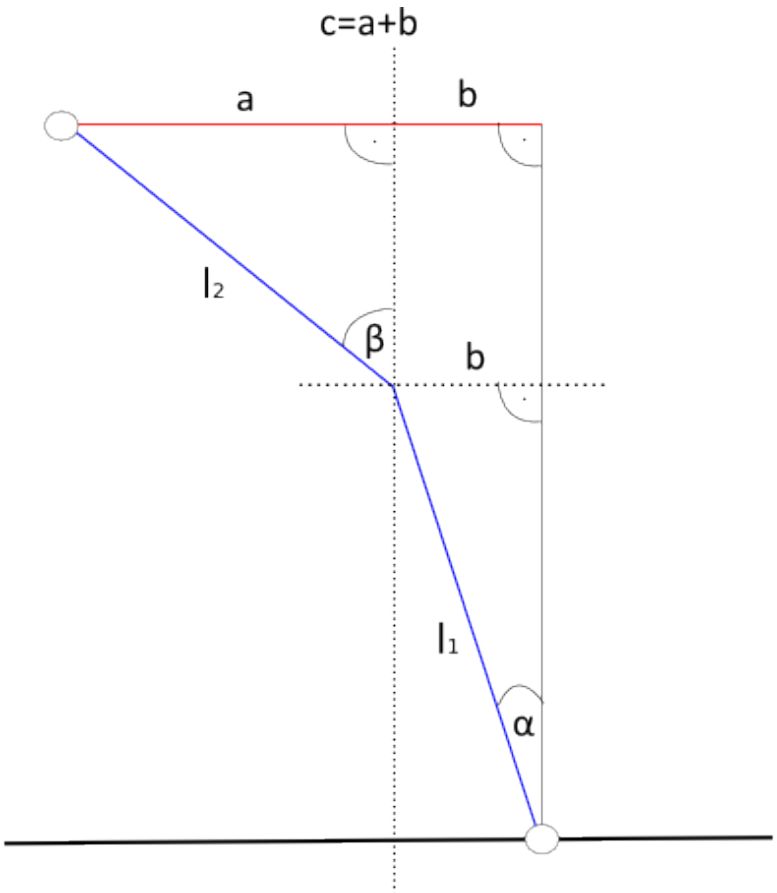
Nüüd, kui näeme, et kontroller suudab pendlit hoida üheksakümne kraadise nurga all, siis vaatame, kas kontroller suudab ka järgida kastfunktsiooni. See tähendab siis seda, et pendel on alguses otse, siis liigub mingi nurga alla ning siis kontroller seab ta uuesti otse. Kastfunktsiooni genereerimiseks muutsime muutuja *sq_amp* väärtuseks0.1 ning *sq_freq* 1 peale. Tulemus on näidatud joonisel 5.26. Nagu näha, suudab kontroller ka koos väikese müraga kastfunktsiooni järgida.



Joonis 5.26: Mudeli käitumine NPC kontrolleriga kastfunktsiooni korral.

Kokkuvõte. Praktilise töö eesmärgiks oli hoida inverteeritud pendlit paigal ning selle tulemuse ka saavutasime. Nägime, et üks kontroller kolmest suutis pendli kontrollimisega hakkama saada väga hästi.

Alguses oli praktilise töö eesmärk indetifitseerida ja kontrollida kahekordset inverteeritud pendlit. Seoses NNSYSID tööriista puudusega, et neurovõrgul saab olla üks sisend ja üks väljund, proovisime kaheastmelisel pendlil kontrollida tipu kaugust mootori keskristsirgest (vt. joonis 5.27).



Joonis 5.27: Kaheastmelise inverteeritud pendli joonis, kus tipu kaugus mootori keskristsirgest on c

Neurovõrguga süsteemi kontrollimisel juhtus see, et kaugus jäi õigeks ehk $c = 0$, kuid α ja β muutusid - pendel n.ö. „kukkus kokku". Kuna me ei suutnud välja mõelda ühte väljundit, mis annaks kontrollerile ühest informatsiooni, siis sellest tulenevalt otsustasime kontrolliülesande lahendada üheastmelise pendliga.

6 Using Neural Networks for Controlling Dynamic Systems

Bachelor thesis (6 ECTS)

Oskar Gross

Summary

The thesis is divided into two main parts: the first part is about the theory of neural networks and the second is a case-study.

The first part describes what the artificial neural networks are and gives a short overview of history and connections to human nervous system. We give an overview of the regression problem and how we can use neural networks to solve it. Also, we give an overview of the most popular training algorithms. We cover briefly the main reasons and methods for optimizing and pruning neural networks.

The last topic in the first part is about dynamic systems. We give a short overview of what dynamic systems are and explain what is BIBO and asymptotic stability. Also, we introduce two types of controllers of which a closed loop controller is used in the case-study part. The aim of the case-study was to give a step-by-step guide how to control an inverted pendulum. In the case-study, we cover the following topics: how to collect data from a Simulink mode l, how to indentify the system and how can we use the neural network to control the system afterwards. We also included a CD with the paper so the reader could carry out the process by him or herself.

Kasutatud kirjandus

C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford University Press, November 1995.

C.-T. Chen, *Linear System Theory and Design*, third. ed., ser. Oxford Series in Electrical and Computer Engineering. Oxford University Press, October 1998.

H. Demuth and M. Beale, *Neural Network Toolbox: User's Guide, Version 3.0*. The MathWorks, Inc., Natick, MA, 1998.

S. Laur, *Properties of Open-Loop Controllers*, 2008, Available from http://courses.cs.ut.ee/2008/modelling-and-control/extras/open-loop-controllers-howto.pdf.

J. R. Leigh, *Applied control theory*. Peregrinus, Stevenage, 1982.

G. Nair, *Rotary double inverted pendulum;sim mechanics model*, April 2007, published in Matlab Central: http://www.mathworks.com/matlabcentral/fileexchange/14533.

W. Nienstedt, O. Hänninen, A. Arstila, and S.-E. Björkqvist, *Inimese füsioloogia ja anatoomia*. Medicina, 2005.

M. Nørgaard, O. E. Ravn, N. K. Poulsen, and L. K. Hansen, *Neural Networks for Modelling and Control of Dynamic Systems: A Practitioner's Handbook*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2000.

P. E. Sarachik, *Principles of linear systems*. Polytechnic University, New York, 1997, available in books.google.com.

J. R. Searle, *Minds, brains, and programs*, 1980, vol. 3.

K. Swingler, *Applying Neural Networks: A Practical Guide*. NY: Academic Press, 1996.

The MathWorks, Inc., *MATLAB® R2007a*, 2007, version 7.4.0.287, January 29, 2007.

——, *Simulink®, R2007a*, 2007, version 6.6.