Tehnike za brzu aproksimaciju kvadratnog korena

Dragana Ilijašević and Miroslav Lutovac, *Senior Member, IEEE*

***Sadržaj* — Analiziran je algoritam za procenu kvadratnog korena neke vrednosti korišćenjem iterativnih metoda. Pokazano je da konstante koje su date u publikovanim radovima, a koje su određene empirijski, nisu optimalne. Korišćenjem softvera za simboličko procesiranje, izvršena je optimizacija korišćenjem kombinovano simboličko-numeričke metode.**

***Ključne reči* — nelinearni sistemi, simboličko procesiranje.**

I. UVOD

PRIBLIŽNO izračunavanje kvadratnog korena je potrebno u brojnim primenama obrade signala, kao što su (a) izračunavanje kvadratnog korena srednje vrednosti zbira kvadrata odbiraka nekog signala, (b) amplitude spektralne komponente dobijene Furijeovom transformacijom (kada su poznate realna i imaginarna komponenta), (c) u sistemima za automatsku regulaciju pojačanja, (d) procenu trenutne vrednosti anvelope signala dobijene demodulacijom amplitudski modulisanih signala u digitalnim prijemnicima, kao i u (e) brojnim algoritmima za analizu trodimenzionih grafičkih signala [1]. Pri izboru algoritma za specifične potrebe, na primer za implementaciju algoritma korišćenjem određenog hardvera, rezultati koji su publikovani ne mogu uvek da se koriste. Tako na primer, neki algoritmu su zasnovani na rezultatima koji su poznati samo autorima članka i ne mogu se direktno primeniti za drugačije definisano hardversko ili softversko okruženje. Jedan primer je detaljno objašnjen u radu [2] u kome je analiziran jedan takav algoritam, a testirane su i performanse algoritma u aritmetici sa fiksnim zarezom. U [2] su izvedeni izrazi na osnovu kojih je moguće uraditi optimizaciju u zavisnosti od željene tačnosti i brzine izračunavanja.

U ovom radu je analiziran i drugi algoritam izloženi u [1] tako da je moguće uraditi uporednu analizu algoritama pod istim uslovima. Kao i u slučaju opisanom u [2], i kod drugog algoritma su neki koeficijenti koji se koriste u algoritmu dati na osnovu rada autora koji nisu objasnili postupak kojim su dobili koeficijente [3]. Tačnije rečeno, autori su rekli da su koeficijenti dobijeni empirijski [1].

Detaljna analiza koja sledi urađena je na osnovu algoritma [1]. Biće pokazano da uporedna analiza iz [1] ne garantuje da su zaključci u [1] korektni, pre svega zato što nema tvrdnje da su koeficijenti tako odabrani da je greška minimalna. Čitaoci mogu pogrešno da zaključe da algoritam radi lošije nego što bi to bio slučaj da je izvršena optimizacija algoritma.

Drugi razlog koji može da dovede do pogrešnog zaključka je taj da je algoritam prilagođen hardverskoj implementaciji, iako bi softverska implementacija mogla da bude jednostavnija.

U ovom radu, u sledećem poglavlju će najpre biti opisan postupak koji je već izložen u [1] i [3]. U narednim poglavljima će biti izvedena analiza koja pokazuje da greška može da bude i manja, kao i da algoritam može da se optimizira tako da se odrede koeficijenti kojima se povećava tačnost algoritma.

II. NELINEARNI IIR FILTAR (NIIRF) METOD

U radu [3] prezentovana je iterativna tehnika koja je prilagođena implementaciji u aritmetici sa fiksnim zarezom.

Za analizu izbora algoritma za izračunavanje kvadratnog korena uzimaju se u obzir dva kriterijuma:

1. brzina izračunavanja,
2. preciznost – tačnost izračunavanja.

U ovom radu biće razmotrena oba kriterijuma uzimajući u obzir implementaciju sa procesorima sa fiksnim zarezom ili hardverska implementacija korišćenjem programabilnih kola. Simulacija i analiza biće rađene u programskim paketima MATLAB (implementacija fiksnog zareza) [4] i *Mathematica* (izvođenje izraza potrebnih za optimizaciju) [5].

Usvojena su sledeća ograničenja:

- Dozvoljeno je korišćenje operacija množenja, i to množenje sa konstantom ili množenje dve promenljive.
- Dozvoljeno je korišćenje operacije sabiranja i oduzimanja.
- Nije dozvoljena računska operacija deljenja.
- Dozvoljen je mali broj iteracija za izračunavanja.

Dozvoljena je upotreba male *look-up* tabele koja se može realizovati softverski bez operacija grananja.

A. Opis originalnog algoritma

Na slici 1 je prikazan algoritam koji je predložen u [3] a detaljno objašnjen u [1]. Engleski naziv algoritma je *Nonlinear IIR Filter* (NIIRF) *method*, iako se u ovom slučaju ne radi o filtriranju signala; naime, ulazni podatak

Ovaj rad je delimično finansiralo Ministarstvo nauka Republike Srbije, projekat TR-110002.

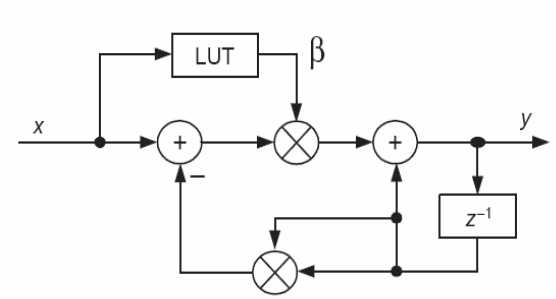
D. Ilijašević, QSS d.o.o. Beograd, Srbija (e-mail: dragana.ilijasevic@gmail.com).

M. Lutovac, Elektrotehnički fakultet u Beogradu, Bulevar kralja Aleksandra 73, 11120 Beograd i DUNP, Srbija; (e-mail: lutovac@etf.rs).

je promenljiva x a rezultat algoritma, odnosno obrade, je promenljiva y, koja je približna vrednost kvadratnog korena $y \approx \sqrt{x}$.

Naziv algoritma najverovatnije potiče od strukture koja liči na filtre sa beskonačnim impulsnim odzivom (IIR – *Infinite Impulse Response*). Naime, izlazna promenljiva se vraća na ulaz ovog sistema i sabira se sa ulaznom promenljivom, a to liči na strukture filtara sa beskonačnim impulsnim odzivom. U suštini radi se rekurzivnoj implementaciji koja se može primeniti i na IIR i na FIR filtre (FIR – *Finite Impulse Response*), pa i u tom smislu naziv algoritme može da zbuni i proizvede pogrešan utisak o vrsti sistema.

Da naziv sistema nije adekvatan govori i to da u ovom algoritmu niti postoji pobudni signal, niti se računa impulsni odziv. Osim toga, ako formalno i prihvatimo izlaznu promenljivu kao signal, u povratnoj sprezi koji se oduzima od "*pobudnog*" signala, stvarno se koristi kvadrat signala, a to onda više nije linearni sistem. Zato je u nazivu algoritma dodata reč koja govori o nelinearnom sistemu.



Sl. 1. NIIRF implementacija predložena u [3].

Preciznije rečeno, algoritam bi trebalo da ima naziv – nelinearni rekurzivni sistem.

Dodatni problem koji se može uočiti na osnovu originalne slike je taj da množenje sa promenljivom β nije potpuno jasno. U objašnjenju algoritma u [1], β se tretira kao koeficijent koji ima nepromenjenu vrednost tokom izvršavanja algoritma, pri čemu ta vrednost može da zavisi od vrednosti promenljive x. Na osnovu slike, i toga da se radi o iterativnom rekurzivnom algoritmu, nije jasno da li se koeficijent β izračunava u svakoj iteraciji, odnosno da li je deo iterativnog algoritma. Zbog toga će početna relacija koja opisuje iterativni algoritam biti napisana u nešto drugačijem obliku od onog u [1]:

$$y[k+1] = \beta\left(n - (y[k])^2\right) + y[k]. \qquad (1)$$

U prethodnom izrazu, umesto nezavisno promenljive x upotrebljena je oznaka n, da bi se istakli da se ne radi o signalu već o konstantnoj vrednosti, a koja je ulazni podatak za algoritam. Da bi se jasno razgraničilo šta se menja tokom iterativnog postupka, redni broj iteracije je predstavljen sa k, a vrednost promenljive u k-toj iteraciji se označava uglastim zagradama, na primer sa $[k]$. Kako se promenljive β i n ne menjaju u iterativnom postupku, to one nisu funkcije od k.

B. Izbor početne vrednosti i konstante ubrzanja

U ovom delu treba da se objasni uticaj konstante β i izbor početne vrednosti iteracija $y[0]$.

Konstanta β se naziva faktor ubrzanja iterativnog postupka. Na osnovu jednačine (1) može se zaključiti da se iterativni postupak završava kada se vrednost $y[k+1]$ ne menja tokom iteracija u odnosu na prethodnu vrednost $y[k]$. Drugim rečima, kada $y[k+1]$ postane jednako sa n, tada se može prekinuti iterativni postupak. Razlika $(y[k+1]-n)$ je vrednost koja se koristi da se doda prethodnoj vrednosti $y[k]$, pa ukoliko je konstanta β veća, to će se brže doći do tačne vrednosti. Međutim, ako je ova vrednost previše velika, tada može doći do oscilacija oko tačne vrednosti, što može da napravi veliku grešku ukoliko se koristi konačan broj iteracija. Očigledno je da postoji neka optimalna vrednost za β kada se najbrže dolazi do rezultata koji ima zadovoljavajuću tačnost. U originalnom radu je data tabela koja je odredila optimalne vrednosti za određene opsege vrednosti čiji koren treba da se izračuna, kao što je prikazano na slici 2.

4 MSBs of x	β fixed-point	β flt-point
0100	0x7b20	0.961914
0101	0x6b90	0.840332
0110	0x6430	0.782715
0111	0x5e10	0.734869
1000	0x5880	0.691406
1001	0x53c0	0.654297
1010	0x4fa0	0.622070
1011	0x4c30	0.595215
1100	0x4970	0.573731
1101	0x4730	0.556152
1110	0x4210	0.516113
1111	0x4060	0.502930

Sl. 2. Vrednosti konstante β iz [3].

Nekada su realizacije sa *look-up* tabelama imale smisla u implementacijama jer se njima ostvaruje ubrzanje algoritma, tako da se sa neke memorijske lokacije pročita vrednost nelinearne funkcije, umesto da se ona izračunava.

Nedostatak predloženog rešenja u [1] je u tome da danas više nije ograničavajući faktor memorijski prostor. Umesto da se koristi delimična linearizacija nelinearne funkcija i zamena vrednosti na nekom segmentu sa konstantom, može da se koriste tabele većih dimenzija ili da se paralelno izračunava vrednost nelinearne funkcije (kod programabilnih hardverskih rešenja). U tom slučaju predloženo rešenje iz [1] nije adekvatno zato što nije data ni funkcija, ni kako da se izračunaju vrednosti konstante za manju grešku algoritma.

Drugi parametar koji treba da se odredi je početna vrednost iteracije od koje veoma zavisi i brzina i tačnost algoritma. Predloženo je da se koristi:

$$y[0] = \frac{2n}{3} + 0.354167. \qquad (2)$$

S obzirom da je formula dobijena empirijski, nije poznato da li je ovo optimalna vrednost i da li postoji bolje početno rešenje.

C. *Izračunavanje faktora ubrzanja*

Umesto da se koristi tabela sa predefinisanim vrednostima za faktor ubrzanja, predloženo je nekoliko izraza koji mogu da se koriste za izračunavanje ovog faktora. S obzirom da su postavljeni uslovi da se koriste samo operacije sabiranja i množenja, najjednostavnije je da se koristi polinomska predstava za faktor ubrzanja. Predloženi su sledeći izrazi:

$$\beta = (0.763n - 1.5688)n + 1.314 \quad (3)$$

$$\beta = -0.61951n + 1.0688 \quad (4)$$

$$\beta = 0.64. \quad (5)$$

Faktor ubrzanja se može izračunati sa dva množenja i dva sabiranja, ili sa jednim množenjem i jednim sabiranjem ili da bude konstanta. U prvom slučaju su potrebne dve instrukcije pomnoži i saberi, u drugom jedna takva instrukcija, a u trećem slučaju to je konstantna vrednost za algoritme koji se softverski implementiraju. U hardverskim implementacijama sa programabilnim hardverom, ova izračunavanja ne moraju biti ništa složenija od pravljenje tabele sa predefinisanim vrednostima. Prema tome, sadašnje mogućnosti procesora i programabilnih hardvera prevazilaze razloge zbog kojih je korišćeno rešenje sa *look-up* tabelama.

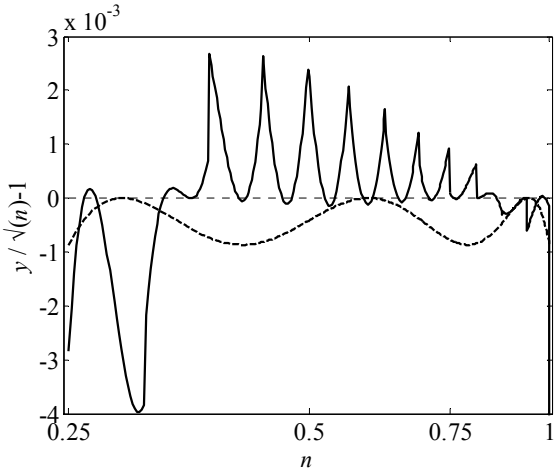
U originalnom radu se dalje analiziraju greške u funkciji izbora rešenja za određivanje konstante ubrzanja i broja iteracija originalnog algoritme. Iako su rezultati tih analiza korisni, kada treba da se implementira algoritam na drugačijim platformama, teško je odrediti šta je optimalno rešenje, tim pre ako treba menjati i broj bita koji je raspoloživ u implementacijama sa fiksnim zarezom.

U narednom poglavlju biće ponovljeni rezultati originalnog rada, kao referentno rešenje, a zatim će biti urađena analiza mogućnosti da se urade izmene u originalnom algoritmu sa ciljem da se poveća tačnost.

III. PROVERA RAD ORIGINALNOG ALGORITMA

Provera rada urađena generisanjem koda u programskom paketu MATLAB. Osim ograničenja koja su navedena u uvodnom delu, poželjno je da program nema grananja, odnosno da se ne koriste naredbe kao što su *if* i *case*. Drugo ograničenje je da je opseg vrednosti broja čiji kvadratni koren treba da se izračuna u opsegu od 1/4 do 1, zato što se jednostavnom normalizacijom može dovesti bilo koji opseg u opseg vrednosti $1/4 \le n < 1$. Naime, ako je broj manji, tada se on pomnoži sa 4, što je ekvivalentno pomeranju zareza za dva bita u desno. Nakon dobijanja kvadratnog korena, rezultat se pomeri za jedan zarez u levo, tako da se uvek koristi maksimalni broj bita u izračunavanju. Da bi izbegli operacije grananja, zbog keširanja instrukcija, *look-up* tabela je realizovana tako da se broj pomnoži sa 16, što je ekvivalentno pomeranju za 4 bita u desno, a zatim se vrši odsecanje celobrojnog dela, da bi se na kraju broj pomerio za 4 bita u levo. Ovako dobijen broj se može koristiti kao indeks niza, ili registra, iz koga se čita vrednost faktora ubrzanja za željeni opseg vrednosti broja *n*.

Na slici 3 su upoređene greške algoritma NRI iz rada [2] sa algoritmom NIIRF. U [1] je izneta tvrdnja da NRI metod tačniji ali da zahteva veći broj iteracija. Na slici se vidi da NRI algoritam daje manju grešku sa samo jednom iteracijom u odnosu na NIIRF koji je realizovan sa dve iteracije. Slika takođe dokazuje da je NIIRF algoritam korektno implementiran u ovom radu jer se dobija ista slika za grešku kao u [1].

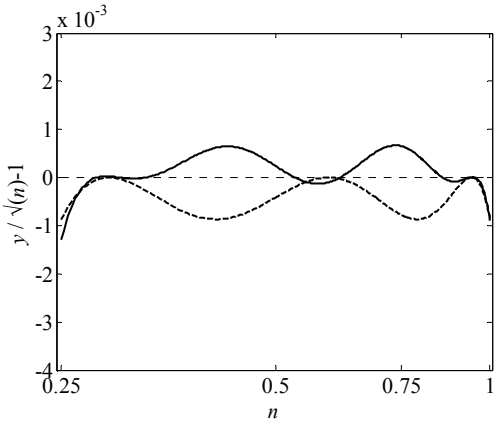


Sl. 3. Relativna greške za NIIRF algoritam sa 2 iteracije (puna linija); i greške za NRI sa jednom iteracijom (isprekidana linija).

Ovaj primer pokazuje da zaključke o uporednim analizama koji su publikovani treba proveriti, a posebno kada se daju paušalno bez konkretnih rezultata kao što je u ovom slučaju; a to je da je NRI metod tačniji (što je tačno) ali znatno složeniji (što nije tačno).

Nakon što je izvršena provera algoritma i rezultata koji su publikovani, sledi simbolička analiza algoritma sa ciljem da se proveri da li je predloženi algoritam optimalan. Na slici 4 je nacrtana greška kada se za faktor ubrzanja koristi jednačina (3) umesto *look-up* tabele.

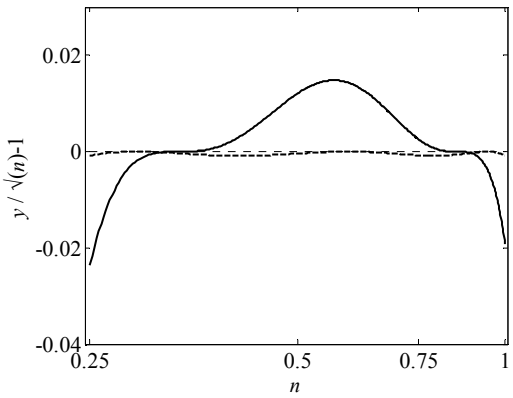
Može se zaključiti da je veća greška na slici 3 posledica zamene funkcije linearnim segmentima, tako da je greška najveća u tačkama kada se prelazi sa jednog na drugi segment funkcije.



Sl. 4. Relativna greške za NIIRF algoritam sa 2 iteracije i $\beta = (0.763n - 1.5688)n + 1.314$ (puna linija).

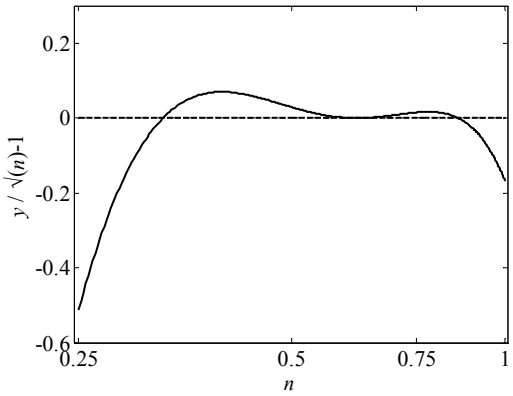
Na slici 5 je nacrtana greška kada se za izračunavanje faktora ubrzanja koristi izraz (4). Smanjenje složenosti za

jedno množenje i sabiranje povećava grešku skoro 10 puta.



Sl. 5. Relativna greške za NIIRF algoritam sa 2 iteracije i $\beta = -0.61951n + 1.0688$ (puna linija).

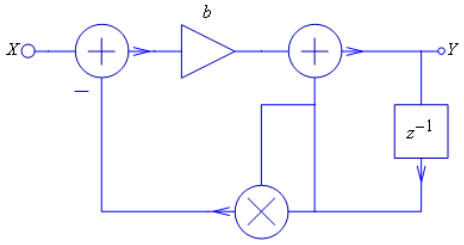
Na slici 6 je nacrtana greška kada se za izračunavanje faktora ubrzanja koristi izraz (5). Korišćenje konstante za faktor ubrzanja povećava grešku skoro za red veličina.



Sl. 6. Relativna greške za NIIRF algoritam sa 2 iteracije, $\beta = 0.685$ i $y[0] = 2n/3 + 0.339$.

IV. Simboličko-numerička optimizacija

Za optimizaciju algoritma je korišćen softver za simboličko procesiranje [6]. Najpre je nacrtana šema nelinearnog vremenski diskretnog sistema, gde je korišćen množač za implementaciju faktora ubrzanja.

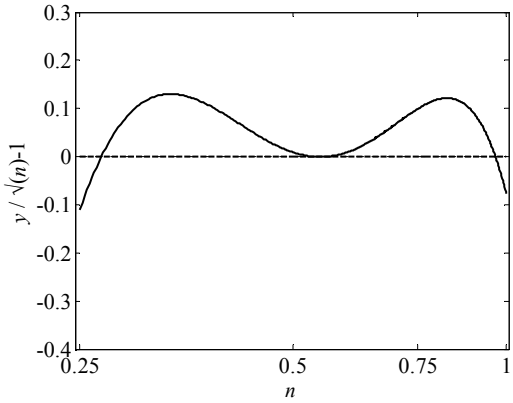


Sl. 7. Šematski opis nelinearnog NIIRF algoritma.

Pošto su svi parametri simbolički, dobijeni su izrazi u zatvorenoj formi, tako da je moguće u finalnom izrazu koristiti izraze (3), (4) i (5). Za slučaj implementacije faktora ubrzanja konstantom, urađena je numerička optimizacija i funkcija greške je nacrtana na slici 8. Maksimalna greška je smanjena 5 puta.

To znači da numeričke vrednosti koje su date u [1] nisu optimalne, i da greška može značajno da se smanji.

Dobijeni rezultati pokazuju da empirijski izvedeni izrazi, bez objašnjenog postupka kako su dobijeni, ne moraju da budu optimalni.



Sl. 8. Relativna greške za NIIRF algoritam sa 2 iteracije, $\beta = 0.685$ i $y[0] = 2n/3 + 0.339$.

Dodatna analiza uticaja konačne dužine reči može da se realizuje na sličan način kao i u [2]. Na osnovu uticaja implementacije korišćenja aritmetike sa fiksnim zarezom može se odabrati i onaj algoritam gde su greške usled kvantizacije i izabranog algoritma približno istog reda veličine. Takođe, upoređivanje dva algoritma (NRF opisanog u [2] i NIIRF koji je analiziran u ovom radu) može se izabrati optimalno rešenje u skladu sa raspoloživim hardverskim i softverskim resursima.

V. Zaključak

Algoritmi koji su izvedeni pre više godina, a koji su optimizirani za tehnologiju koja je tada bila raspoloživa, ne moraju nužno da budu optimalni i u tehnologiju koja je danas raspoloživa. Ako su algoritmi zasnovani na privatnim i nepublikovanim rešenjima, ili su dobijeni empirijski, mogu biti neadekvatni za implementaciju. U radu je opisan jedan postupak kako se mogu ponoviti publikovani rezultati a zatim i optimizovati.

Literatura

[1] R. G. Lyons, *Streamlining Digital Signal Processing, A Tricks of the Trade Guidebook*, Piscataway: IEEE Press, 2007, pp. 165–172.

[2] D. Ilijašević and M. Lutovac, "Implementacija algoritma za izračunavanje kvadratnog korena u aritmetici sa fiksnom tačkom," *ETRAN* 2009.

[3] N. Mikami et al., "A New DSP - Oriented Algorithm for Calculation of Square Root Using a Nonlinear Digital Filter," *IEEE Trans. on Signal Processing*, pp. 1663 – 1669, July 1992.

[4] MATLAB Version 7, MathWorks, Inc., 2005

[5] S. Wolfram, *The Mathematica Book*, Cambridge University Press, Wolfram Media, Cambridge, 2003.

[6] M. Lutovac and D. Tošić, *SchematicSolver* Version 2.2.

Abstract

Algorithm for estimating the square root of a single value using iterative methods is analyzed. It is shown that the constants in published papers, that were determined empirically, are not optimal. A software tool for symbolic processing is used for symbolic-numeric optimization.

HIGH - SPEED TECHNIQUES FOR APPROXIMATING THE SQUARE ROOT

Dragana Ilijašević and Miroslav Lutovac