SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD BR.5060

Učinkovita izvedba trigonometrijskih funkcija polinomijalnom aproksimacijom

Andrea Jelavić Šako

Zagreb, lipanj 2017.

Sadržaj

1. Uvod

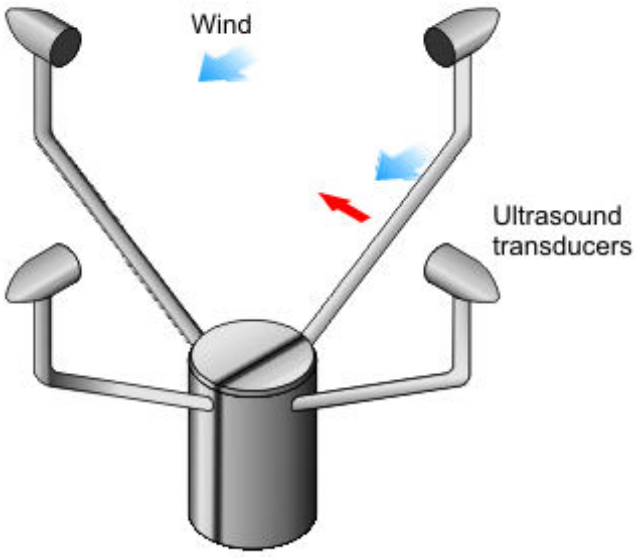
Anemometar je mjerni instrument koji mjeri brzinu i smjer vjetra. Postoje različite izvedbe uređaja, ali u okviru ovog rada razmatra se ultrazvučni anemometar temeljen na četiri ultrazvučna pretvarača. Izračun brzine i smjera vjetra temelji se na vremenu propagacije potrebnom bloku sinusnih signala da prijeđe put od odašiljača do prijemnika. S obzirom da blok ima djelomične podatke iz oba signala, ti podaci se ne mogu pravilno tumačiti i potrebno vrijeme kašnjenja se može odrediti iz faznih odnosa komponenti prisutnih u prijemnom signalu.

Procesor eZdsp TMS320VC5505 obrađuje skup podataka sa dva prijemnika anemometra. S obzirom da je procesor razvojnog sustava namijenjen radu sa cjelobrojnim brojevima, potrebno je pronaći način računanja faze kompleksnog broja operacijama koje procesor podržava i time odrediti fazne odnose između signala.

Za određivanje kuta korištena je Taylorova aproksimacija i po odsječcima polinomni modeli nižeg stupnja. Simulirani su modeli u kojima su ulazni podaci veličine 16 i 32 bita. Algoritam za određivanje kuta ostvaren je u C-u i Matlabu i zatim uspoređeni rezultati kako bi se pronašao najefikasniji način implementacije tražene funkcije na procesoru.

2. Ultrazvučni anemometar

Anemometar je mjerni instrument koji služi za mjerenje jačine vjetra i brzine strujanja zraka. Najčešće se koristi u vjetroelektranama, meteorologiji i industrijama koje se bave proizvodnjom električne energije. Postoji nekoliko vrsta anemometra, a neke od njih su anemometar sa lopaticama, ultrazvučni i laser doppler anemometar. Jedna od najstarijih verzija uređaja je anemometar sa zakretnim lopaticama na temelju čije se brzine zakretanja određuje brzina vjetra. Te verzije su pomalo neprecizne i neučinkovite pa ne mogu detektirati brzine vjetra malih iznosa. S obzirom na ta ograničenja, u okviru ovog završnog rada, promatrane su značajke rada ultrazvučnog anemometra koji služi mjerenju dvodimenzionalnih horizontalnih komponenata smjera i brzine vjetra a pritom koristi ultrazvučne valove. Temelji se na četiri ultrazvučna pretvarača raspoređena na uglovima romba čije su dijagonale orijentirane u skladu sa osnovnim geografskim osima. Jedan ultrazvučni senzor se ponaša kao odašiljač, dok je drugi prijemnik. Princip mjerenja brzine vjetra utemeljen je na mjerenju vremena propagacije potrebnog ultrazvučnom impulsu da pređe put između fiksnog odašiljača i fiksnog prijemnika. Također, neke od prednosti takve vrste anemometra su zvuk visoke frekvencije koji ima manje šumova i manja valna duljina. Implementacija i testiranja rada anemometra izvedena su na protoboardu i neki od zaključaka ovog rada bit će iskorišteni za primjenu na TMS320VC5505 procesoru za digitalnu obradu signala.



Slika 1 Ultrazvučni anemometar

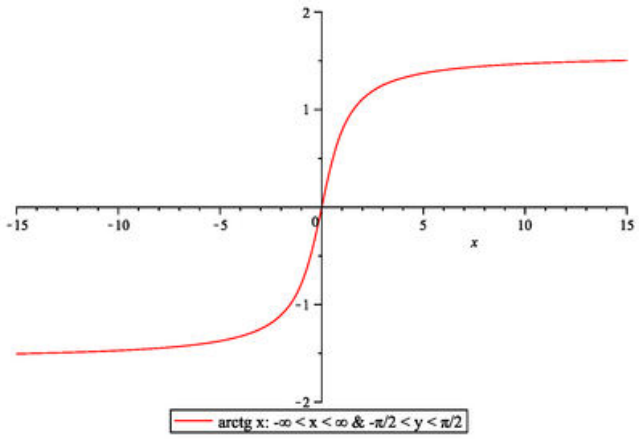
Kroz prethodne radove opisan je rad anemometra, točnije približen njegov rad simulacijom u Matlab-u. Da bi točnost tih izračuna bila što veća, simulirana su mjerenja po svim putevima između prijemnika, točnije osam kombinacija. Svaki signal sastoji se od tri frekvencijske komponente da bi bilo omogućeno odrediti pomak na prijemnicima. Za računanje pomaka primljenog signala spektar određujemo pomoću Fourierove transformacije (DFT) Goertzelovim algoritmom. Nakon izračuna faze primljenih signala, oduzimaju se inicijalne faze dobivene kalibriranjem. Zatim rezultate frakcionalnom aritmetikom pretvaramo u pozitivnu fazu na intervalu od 0 do 1. Nadalje, neki od izračuna su bili vrijeme kašnjenja, standardna devijacija kašnjenja i ukupno vrijeme kašnjenja. Iz svih vremena kašnjenja se rekonstruira brzina i smjer vjetra, a konačna brzina i smjer vjetra se izračunavaju uz pomoću rezultata po pojedinim osima i kompenziranjem bočnog vjetra. Kod svih izračuna i simulacija potrebno je znati izračunati kut potrebnog signala koji predstavljamo ulaznim kompleksnim brojem pa se na toj tematici temelji ovaj rad. Potrebno je prilagoditi izračune procesoru koji ne radi u cjelobrojnoj aritmetici pa se stoga koristimo frakcionalnom aritmetikom i aproksimacijom funkcije arkus tangens.

Slika 2 Kontrukcija anemometra

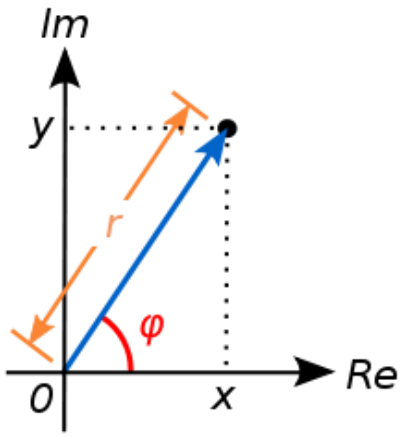
3. Trigonometrijska funkcija arctg i njene implementacije

Arkus tangens je funkcija koja je inverzna funkciji tangens na intervalu njene domene [-π/2,π/2].

Kada je potrebno izvoditi operacije sa kompleksnim brojevima, koristi se funkcija arg koja vraća kut između pozitivne realne osi i polupravca koji spaja traženu točku sa ishodištem, što je zapravo arctg omjera imaginarnog i realnog dijela ulaznog kompleksnog vektora.



Slika 3 Prikaz funkcije arkus tangens



Slika 4 Prikaz faznog kuta kompleksnog broja

U različitim okružjima to jest programskim jezicima postoje različite implementacije tražene funkcije. U programskom jeziku MATLAB-u se radi o funkciji atan i atan2.

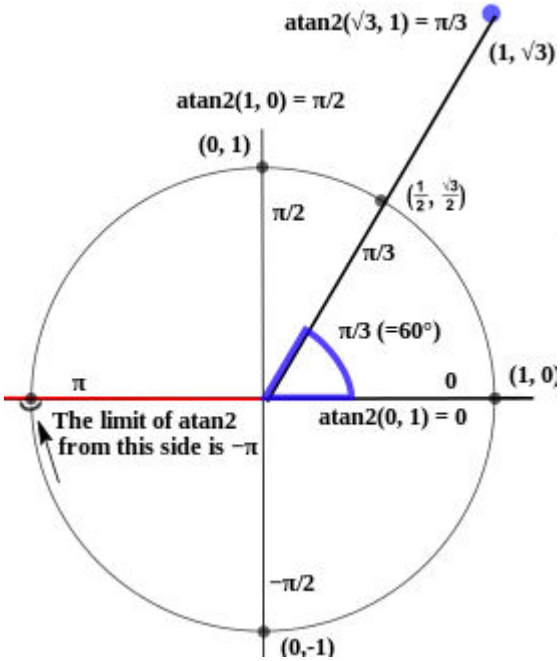
Y = atan(X) vraća tan^{-1} elemenata od X i funkcija je koja se može izvoditi na poljima. Za realne elemente iz X, vraća vrijednost iz intervala [-pi/2, pi/2], a za kompleksne vrijednosti iz X povratne vrijednosti su također kompleksne. Traženi kut je u radijanima.

Atan2 je arkus tangens funkcija koja prima 2 argumenta i računa polarni kut (x/y) točke zadane Kartezijevim koordinatama. Široko je primijenjena u područjima koja se bave obradom digitalnih signala, npr. za određivanje faze ili rekonstrukciju faze nekog signala. Za bilo koji realni broj (floating point) argumente x i y različite od 0, izlazna vrijednost funkcije je kut u radijanima između pozitivne x osi ravnine i točke

dane koordinatama (x,y) u ravnini. Kut je pozitivan za gornju poluravninu gdje je y >0 a negativan za donju poluravninu. Domena funkcije je $-\pi < \text{atan2}(y, x) \leq \pi$ za razliku od atan funkcije koja je ograničena na interval [-pi/2, pi/2]. Svrha korištenja dva argumenta , to jest korištenja funkcije atan2 umjesto atan je skupljanje informacija o predznacima ulaznih argumenata kako bi znali u kojem se kvadrantu nalazi dani kut, što nije moguće odrediti iz funkcije s jednim argumentom. Također izbjegavamo problem dijeljenja sa 0.

$$\operatorname{atan2}(y,x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0, \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{if } x < 0 \text{ and } y \geq 0, \\ \arctan\left(\frac{y}{x}\right) - \pi & \text{if } x < 0 \text{ and } y < 0, \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0, \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$

$$\operatorname{atan2}(y,x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0, \\ \frac{\pi}{2} - \arctan\left(\frac{x}{y}\right) & \text{if } y > 0, \\ -\frac{\pi}{2} - \arctan\left(\frac{x}{y}\right) & \text{if } y < 0, \\ \arctan\left(\frac{y}{x}\right) \pm \pi & \text{if } x < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$

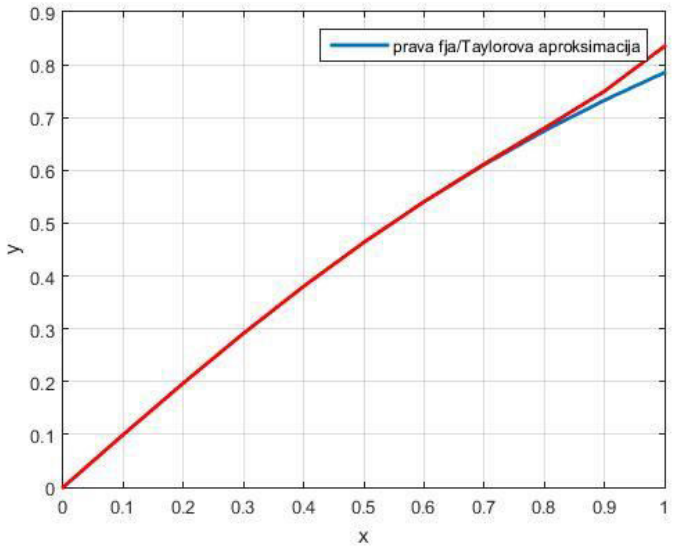


Slika 5 Prikaz faznih kutova na brojevnoj kružnici

3.1. Taylorova aproksimacija funkcije arctg

Postoje brojni algoritmi dostupni za implementaciju arkus tangens funkcije kada su nebitne posljedice s programske i memorijske strane. Najlakše rješenje je koristiti aproksimaciju Taylorovim redom na intervalu [-1,1]. Za mnoge funkcije potrebno je naći jednostavniji i efektivniji način izračuna arkus tangens funkcije koji se onda može lako implementirati na hardweru s ograničenom memorijom i veličinom registara.
Korišteni eZdsp ne podupire implementaciju u cjelobrojnoj aritmetici pa nije moguće koristiti ansi-C funkcije unutar math.h biblioteke C programskog jezika za ostvarivanje tražene funkcije. Potrebno je pronaći način računanja faze kompleksnog broja, ali koristeći se osnovnim operacijama koje procesor koji koristimo podržava i pritom pokušati skratiti složenost samog programa. Jedan od načina kojim to možemo ostvariti je aproksimacija funkcije arkus tangens Taylorovim polinomom jer polinomijalna aproksimacija predstavlja jedan od najboljih izbora za implementaciju nelinearnih funkcija kao što je atan2(x) na DSP-u. Razvojem u Taylorov red možemo prikazati trigonometrijske funkcije u obliku beskonačnih polinoma što omogućuje izračun vrijednosti funkcije preko vrijednosti polinoma u nekoj točki koristeći se samo osnovnim računskim operacijama. Za dani stupanj polinoma i funkciju f(x) = atan2(x) na interval [0,1], teorija polinomijalne aproksimacije traži polinom koji minimizira vrijednost |P(x) – f(x)|, gdje je P(x) aproksimativni polinom.
S obzirom da aproksimaciju trebamo prilagoditi frakcionalnoj aritmetici i procesoru, poželjno je smanjiti broj operacija množenja kako bi spriječili velika odstupanja i nepreciznost konačnog rezultata. Zato je izabrana polinomijalna aproksimacija četvrtog reda kako bi izbjegli preveliko nepodudaranje rezultata kojeg dobivamo kodom u C-u i onog kojeg vraća atan2 funkcija. Koristimo polinom četvrtog reda oko točke z= 0.5 koja je približno na sredini prvog oktanta.

$$Tn(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)(x - x_0)^2}{2!} + \frac{f'''(x_0)(x - x_0)^3}{3!} + \frac{f^{iv}(x_0)(x - x_0)^4}{4!}$$



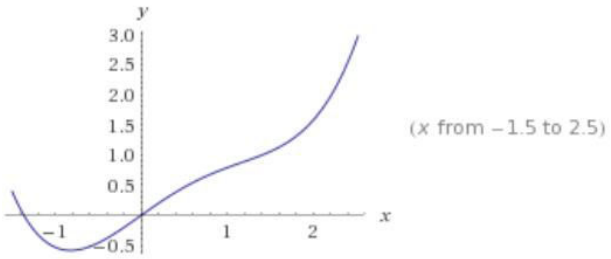
Slika 6 Prikaz arkus tangens funkcije I Taylorove aproksimacije

Polinom zapisujemo Hornerovom shemom kao uzastopne operacije množenja.

$$f(z) = z * \left(z * \left((0.1536 * z - 0.349867) * z - 0.0255995\right) + 1.0112\right) - 0.00141863 \quad (1)$$

$$0.1536\,x^4 - 0.349867\,x^3 - 0.0256\,x^2 + 1.0112\,x - 0.00141905$$

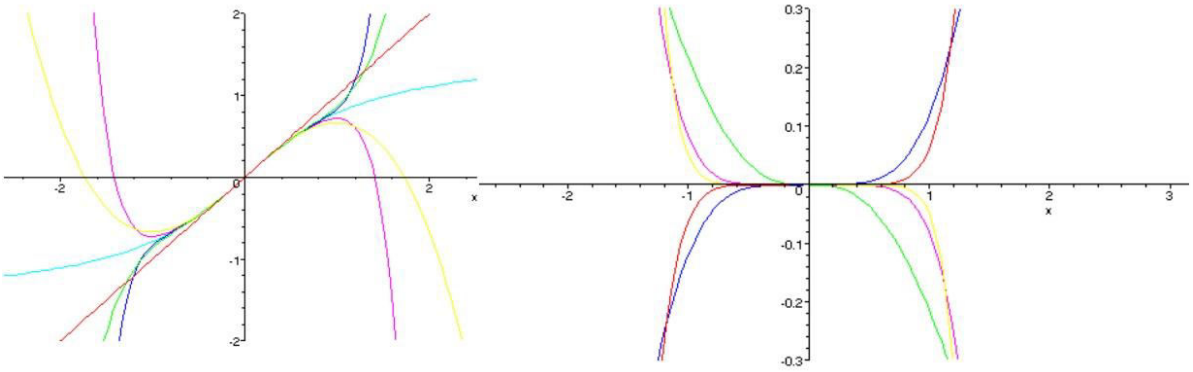
Plots:



Slika 7 Graf aproksimativne funkcije

Aproksimacija i svi koeficijenti polinoma moraju se prilagoditi procesoru na kojem radimo pa koristimo skaliranje koeficijenata i odabiremo onaj dio funkcije za koje ista daje vrijednosti iz intervala pogodnog za primjenu frakcionalne aritmetike.

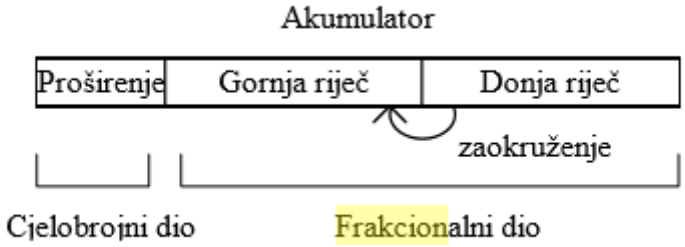
Sljedeća slika prikazuje prvih nekoliko stupnjeva Taylorovog polinoma aproksimacije funkcije arkus tangens oko točke 0 i pogreške aproksimacije koje se povećavaju u rubnim točkama -1 i 1 (zbog polumjera konvergencije Taylorovog reda)



Slika 8 Taylorove aproksimacije arkus tangens funkcije

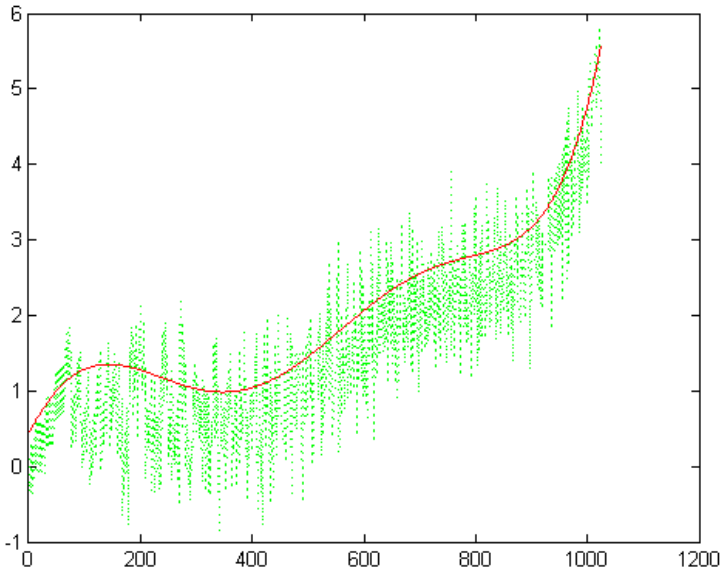
4.Frakcionalna aritmetika

S obzirom da je implementaciju potrebno provesti na eZdsp TMS320VC5505 koji ne podržava rad s decimalnim podacima, postavlja se pitanje kako realizirati potrebne matematičke operacije na cjelobrojnim procesorima. Ako se svi koeficijenti i signal u svim točkama sustava tako skaliraju da im se vrijednosti nalaze između -1 i 1, tada se za prikaz takvih brojeva može koristiti frakcionalna aritmetika. Za izračune u procesorima koji nemaju mogućnost računanja s pomičnim zarezom koriste se podaci izraženi Q formatom. U ovom slučaju koristi se Q15 format. To je format od 15 podatkovnih bitova i 1 bita predznaka. Za dani Qm.n format opseg je $[0, 2^m - 2^n]$. Npr. Q15.1 zahtjeva m+n = 16 bitova, opseg $[-\mathbf{2^{14}}, \mathbf{2^{14}} - \mathbf{2^{-1}}]$, $\mathbf{rezolucija\ 2^{-1}}$ ili ako je broj spremljen u Q2.13 formatu, 2 bita se koriste za cijeli, 13 za frakcionalni dio i 1 bit je bit predznaka. Veza između cjelobrojne i frakcionalne aritmetike je jednostavna: Iz 2komplement (float) zapisa u zapis Q formata dolazimo množenjem s $\mathbf{2^n}$ tj. $\mathbf{2^{15}}$. Ovaj DSP procesor je 16-bitni procesor s cjelobrojnom aritmetikom koji dakle radi sa 16-bitnim internim registrima. Ulazni argumenti aritmetičko-logičkih operacija su 16-bitni. Zbrajalo je 16-bitno, a množilo 32-bitno. U sklopu množila postoji i jedan akumulator koji omogućuje proširenje. Prilikom realizacije funkcije imamo više uzastopnih množenja (Hornerova shema) te postoji opasnost od izlaska iz dinamike. Zbog toga se mogu koristiti i 32bitni ulazni podaci pa se privremena vrijednost čuva u akumulatoru. Prilikom pretvorbe iz decimalnog zapisa, to jest zapisa s pomičnim zarezom stvara se 32bitna riječ koja se zaokruživanjem i posmicanjem svodi na 16bitnu riječ. Uzima se 16bita gornje riječi kao izlaz. Da bi smanjili pogrešku uslijed ispuštanja donje riječi provodi se zaokruženje gornje riječi prilikom svakog množenja.



Slika 9 Prikaz registra

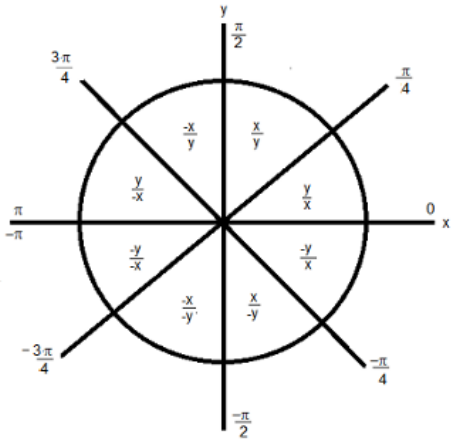
Zaokruživanje također unosi pogrešku, ali ne tako veliku s obzirom da smo grešku već unijeli činjenicom da smo koristili aproksimaciju polinomom.



Slika 10 Prikaz pogreški zbog aproksimacije

4.1.Primjena frakcionalne aritmetike na atan2 funkciju

Ostvarivanje funkcije atan2 preko frakcionalne aritmetike zahtjeva da podijelimo ulazne apsolutne vrijednosti kako bi dobili 16bitnu vrijednost između 0 i 1. Predznaci x i y ulaznih podataka određuju u kojem kvadrantu leži traženi kut. Ulazna vrijednost sa većom apsolutnom vrijednošću se koristi kao vrijednost s kojom dijelimo. Izračunamo 16bitnu ili 32bitnu vrijednost ovisno o operaciji pa zaokruživanjem svedemo na 16bitnu vrijednost koja u konačnici predstavlja kut u radijanima. S obzirom na simetrije same funkcije arkus tangens rezultate iz prvog oktanta prevodimo u konačan rezultat koji može biti u bilo kojem oktantu.



Slika 11 Argument funkcije prema oktantima

5. Implementacije funkcije

Postoji više načina implementacije tražene funkcije u različitim programskim jezicima kako bi simulirali njen rad i izračune potrebne za primjenu na procesoru. Jedan od mogućih načina je implementacija korištenjem floating-point aritmetike generičkim ručno napisanim C programom. Zatim implementacija korištenjem obične cjelobrojne aritmetike s uobičajenim cjelobrojnim podatkovnim tipovima i operacijama množenja i zbrajanja nad takvim tipovima .Također, implementacija korištenjem frakcionalnih podatkovnih tipova i ugrađenih funkcija za aritmetičke operacije koje sadrži sam DSP procesor u svojim DSP bibliotekama. U ovom radu fokus je na prve dvije metode, tj. Kod generiran u C jeziku i Matlab programskoj okolini.

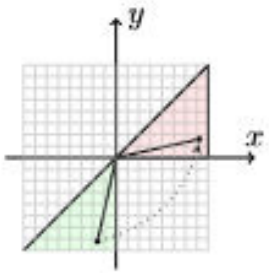
Ključna razlika između ovih metoda je brzina izvođenja, točnije prosječni broj ciklusa procesora potreban za izračun traženog kuta. Druga razlika je numerička točnost za koju je potrebno dokazati da je ista za sve cjelobrojne izvedbe , ali manja u odnosu na floating-point implementaciju

5.1. Implementacija u C-u

U c programskom jeziku implementirana je funkcionalnost sustava izvedenog u Matlabu. Način implementacije u C-u i je prikaz implementacije na DSP-u. U ovoj izvedbi koristi se cjelobrojni 16-bitni zapis za koeficijente polinoma. Kako C jezik ne poznaje frakcionalne tipove, koristimo short za 16 bita i int za 32 bita.Točnije koristimo format Q15, svih 15bita za frakcionalni dio. Za aritmetičke operacije množenja i zbrajanja koriste se operatori + i *. Operacije zbrajanja, oduzimanja i posmaka izvode se na isti način kao i kod rada s cjelobrojnim tipovima podataka, ali to nije slučaj kod množenja. Umnožak dva 16 bitna broja s predznakom daje rezultat u Q30 formatu s 2 bita predznaka. Taj rezultat se ne može spremiti u 16 bitni registar pa ga pohranjujemo u int varijablu uz posmak za jedan korak u lijevo zbog emulacije frakcionalnog moda rada. Također, potrebno je dodati 0x4000 na akumulirani long podatak uz aritmetički shift za 15 koraka u desno, što odgovara čitanju gornje značajnije riječi uz zaokruženje. Na ovaj način, kod množenja ne može doći do preljeva već samo do podljeva koji nastaje kada se nakon množenja u gornjoj riječi registriraju sve 0. (Preljev: max+1=min, podljev: min-1= max).

To sve dovodi do određenog gubitka preciznosti. S toga koristimo skaliranje podataka prije obrade. Faktor skaliranja određuje se eksperimentalno, a referentni dio funkcije je onaj koji daje vrijednosti prvog oktanta.

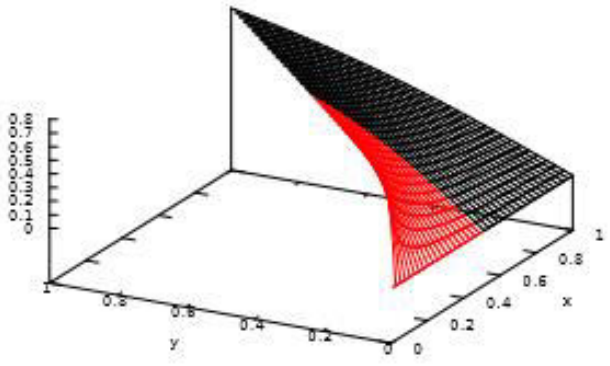
$$\begin{cases} x < 0 \\ y < 0 \\ y < x \end{cases} \Rightarrow \operatorname{atan2}(y,x) = -\tfrac{\pi}{2} - \operatorname{atan2}(|x|,|y|)$$



Slika 12 Simetrija funkcije

Argument funkcije je kvocijent imaginarnog i realnog dijela. Označavamo ga sa z i manji je od 1 s obzirom da uzimamo vrijednosti iz prvog oktanta.

Aproksimacija je kao već spomenuto, 4.reda oko točke 0.5 koja je otprilike na sredini prvog oktanta. Koeficijenti polinoma su skalirani na vrijednosti manje od 1 faktorom skaliranja 1.10 pa je rješenje polinoma pomnoženo sa $\frac{S}{2\pi}$. Vrijednosti [0,2pi] su prikazane na intervalu [0,1).



Slika 13 atan2 funkcija u 3D mode

```
short int atan_2(short int Re, short int Im){
        short int absRe;
         short int absIm;
         short int rez;
         short int z;
         long int pom_var;
        long int pom1, pom2, pom3, pom4;
        short int s_pom1, s_pom2, s_pom3, s_pom4;
// faktor skaliranja 1.10
// koef.podijelili sa 1.10 i zatim *2^15 da dobijemo Q format
        short int x1 = 10422;
        short int x2 = 763;
        short int x3 = 30123;
        short int x4 = 42;
// a=pi/2, b=pi,c=-pi/2
        short int a = 8192;
        short int b = 16384;
         short int c = 24576;

if (Re <= 0){ absRe = -Re; }
                                    else{ absRe = Re; }
                                    if (Im <= 0){ absIm = -Im; }
                                    else{ absIm = Im; }
```

```
if ((Re > 0 && Im == 0) || (Re == 0 && Im == 0)){ return 0; } //0
        else if (Re == 0 && Im > 0){ return  8192; } //pi/2(0.25)
        else if (Re < 0 && Im == 0){ return 16384; } //pi(0.5)
        else if (Re == 0 && Im < 0){ return 24576; } //-pi/2(0.75)

if (absRe == absIm){
                if (Re > 0){
                        if (Im > 0){ return 4096; } // 1/8
                        else{ return 28672; } // 7/8
                }
                if (Re < 0){
                        if (Im>0){ return 12288; } // 3/8
                        else{ return 20480; } // 5/8
                }
        }
        if (absRe < absIm){
                z= (long int)(absRe<<15) / absIm;
        }
        else if (absRe > absIm){
                  z = (long int)(absIm << 15) / absRe;
        }
       pom1 = (long int)(4576 * z) - (long)(x1 << 15);
        pom1 = pom1 + 0x4000;
        s_pom1 = (short int)(pom1 >> 15);

        pom2 = (long int)(s_pom1*z) - (long int)(x2 << 15);
        pom2 = pom2 + 0x4000;
         s_pom2 = (short int)(pom2 >> 15);
```

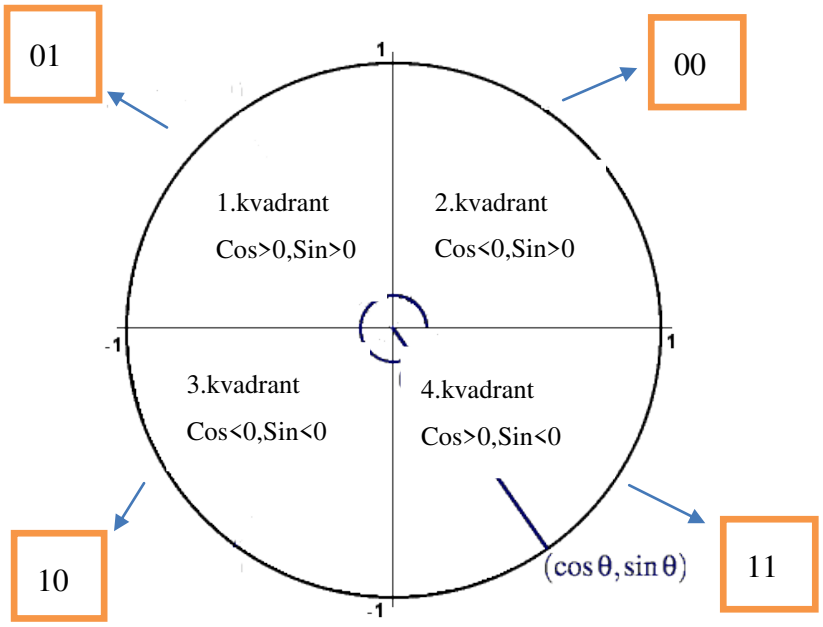
```
        pom3 = (long int)(s_pom2*z) + (long int)(x3 << 15);
        pom3 = pom3 + 0x4000;
        s_pom3 = (short int)(pom3 >> 15);

        pom4 = (long int)(s_pom3*z) - (long int)(x4 << 15);
        pom4 = pom4 + 0x4000;
        s_pom4 = (short int)(pom4 >> 15);
        pom_var = (long int)(5737 * s_pom4); // S/2pi

// ako nije u prvom oktantu ,izrazi rezultat preko vrijednosti iz prvog oktanta
if (absIm > absRe && Re > 0 && Im > 0){ pom_var = (long int)(a << 15) - pom_var; }
 // drugi oktant
else if (absIm > absRe && Re<0 && Im>0){ pom_var = (long int)(a << 15) + pom_var;
}
// treci
else if (absIm < absRe && Re<0 && Im>0){ pom_var = (long int)(b << 15) - pom_var;
}
 // cetvrti
else if (absIm < absRe && Re<0 && Im<0){ pom_var = (long int)(b << 15) + pom_var;
} //peti
else if (absIm > absRe && Re<0 && Im<0){ pom_var = (long int)(c << 15) - pom_var;
} //sesti
else if (absIm > absRe && Re>0 && Im<0){ pom_var = (long int)(c << 15) + pom_var;
} //sedmi
else if (absIm < absRe && Re>0 && Im<0){ pom_var = (long int)(b << 15) - pom_var;
//osmi
       pom_var = pom_var + (long int)(b << 15);
        };
        pom_var = pom_var + 0x4000;
        rez = (short int)(pom_var >> 15);
        return rez;
        }
```

5.2. Implementacija u Matlab-u

Implementacija funkcije u Matlabu služi za simulaciju izgleda podataka koje prima eZdsp TMS320VC5505. Uspoređuju se rezultati kada su ulazni podaci 16-bitni te kada su 32-bitni. Cilj je kao i u implementaciji u C-u, naći vrijednost faznog kuta u radijanima i to na intervalu [0,1]. Funkciju arkus tangens ne aproksimiramo Taylorovim polinomom već nizom Matlab ugrađenih funkcija dijelimo funkciju na intervalu [0,1] na četiri intervala te svaki opisujemo polinomom nižeg stupnja. Ovisno u kojem se kvadrantu nalazi argument (kompleksni broj) određujemo dva najviša bita izlazne 16-bitne riječi.



Slika 14 2 najviša bita izlazne riječi s obzirom na kvadrante

Zatim je potrebno pronaći apsolutnu vrijednost sinusa i kosinusa ulaznog argumenta, to jest ako je ulazna vrijednost z= x + yi, absCos=abs(cos(x)) i absSin=abs(sin(x)). Nadalje, potrebno je provjeriti koja je veličina veća iteriranjem kroz oktante. Naprimjer u prvom, četvrtom, petom i osmom oktantu je absSin>absCos.

Na taj način smo odredili tri bita izlazne riječi, gdje treći bit sadrži informaciju u kojem smo oktantu.

Kako bi osigurali da je ulazni argument manji od 1, u oktantima gdje je absSin < absCos računamo absSin/ absCos i obrnuto. Od istog argumenta računamo atan(arg) i ako je ulazni argument u prvom oktantu, to je ujedno i rješenje koje je bilo potrebno. Međutim, ako je vektor smješten u nekom drugom oktantu potrebno je preko svojstava arkus tangens funkcije i njene simetrije potražiti ispravnu vrijednost.

Ako je ulazni argument smješten u 2.,3.,6. Ili 7. oktantu gdje je absSin>absCos, ulazni argument je absCos/absSin pa zapravo pronalazimo arcctg. S obzirom da je arkus kontanges zrcalno simetričan oko π/4 funkciji arkus tangens, koristimo vezu te dvije funkcije za izračun pravog argumenta.

$ctg\left(\frac{\pi}{2}-\rho\right)=tg(\rho)$ to jest $tg(\rho)=ctg\left(\frac{\pi}{2}-\rho\right)$ (2)

Za 16bitni zapis očekivani raspon vrijednosti unutar prvog oktanta je [0000-2000H] , gdje je 2000h vrijednost π/4. Za vrijednosti iz drugih oktanata potrebno je kut zrcaliti oko π/4 to jest odrediti 4000h – y što predstavlja operaciju π/2 – ρ i uzeti 13 bitova razlike jer su gornja tri bita već određena oktantom.

Ulazni vektor: 0.5 + 0.57735i
Kut: 0.8570719479 rad
Argument: aCos/aSin = $\sqrt{3}/2$ = 0.8660254
Atan(arg) =0.7137
Atan(arg)/2pi = 0.1136
Round(0.1136*2^16) = 7445 = 1D15(h)
Dec2hex(hex2dec('4000')-hex2dec('1D15')) = 22EB(h)
Hex2dec('2000')+hex2dec('2EB') = 22EB(h)
2000(h) predstavlja drugi kvadrant → 0010
Hex2dec('22EB') = 8939
8939/2^16 = 0.1364

Zadnja vrijednost predstavlja kut skaliran na 2pi. Kada podijelimo kut u radijanima sa 2pi dobijemo 0.1364 također što znači da aproksimacijom sigurno ne gubimo na prve 4 decimale.

Potrebno je funkciju prikazati preko polinomijalnih odsječaka što postižemo Matlab ugrađenim funkcijama polyval i polyfit. Matlab funkcija polyfit računa koeficijente Lagrangeovog interpolacijskog polinoma. Poziva se sa p=polyfit(x,y,n) gdje su x i y jednoredne matrice u kojima su pohranjene vrijednosti koordinata, a n stupanj traženog polinoma. Funkcija polyval je standardna Matlab funkcija čiji je poziv y=polyval(p,x) i računa vrijednost polinoma sa koeficijentima zadanim jednorednom matricom p u svim točkama od x.

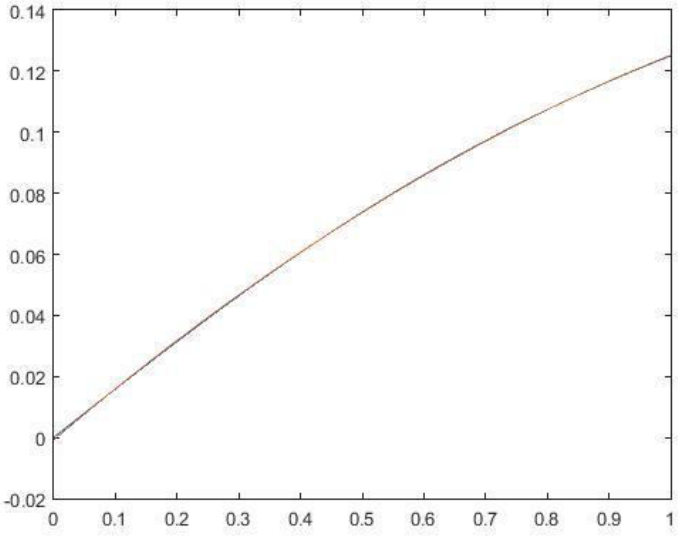
Na primjer, odredimo x kao raspon točaka od 0 do 1 i zatim funkcijom polyfit nađemo koeficijente polinoma stupnja 2 koji najbolje aproksimiraju tu funkciju.

x= [0:1000]/1000

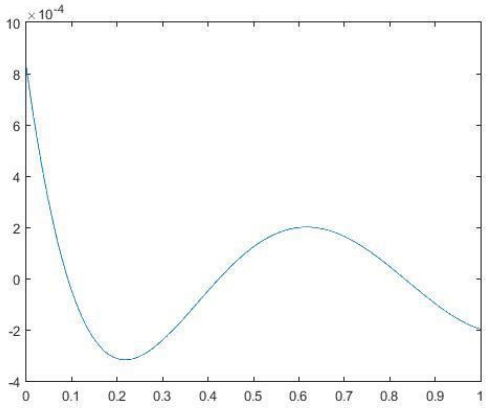
f=atan(xt)/(pi/4)* $\frac{1}{8}$

p=polyfit(xt,fja,2) (-0.0459 0.1719 -0.0008)

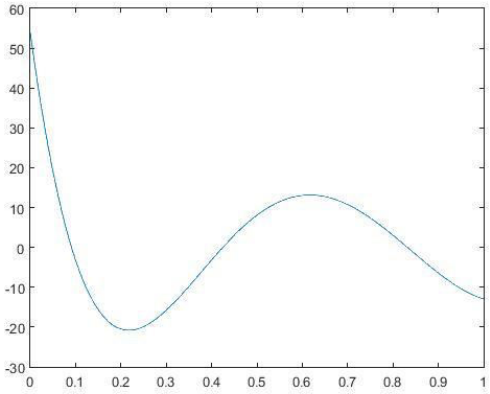
yp=polyval(p,xt) vraća vrijednost polinoma p stupnja 2 izračunatog u x.



Slika 15 Arkus tangens funkcija na intervalu [0,1]



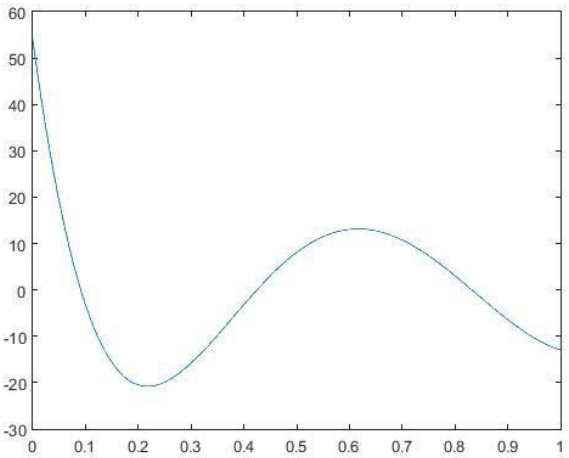
Slika 16 Pogreška (frakcionalna aritmetika)



Slika 17 Pogreška

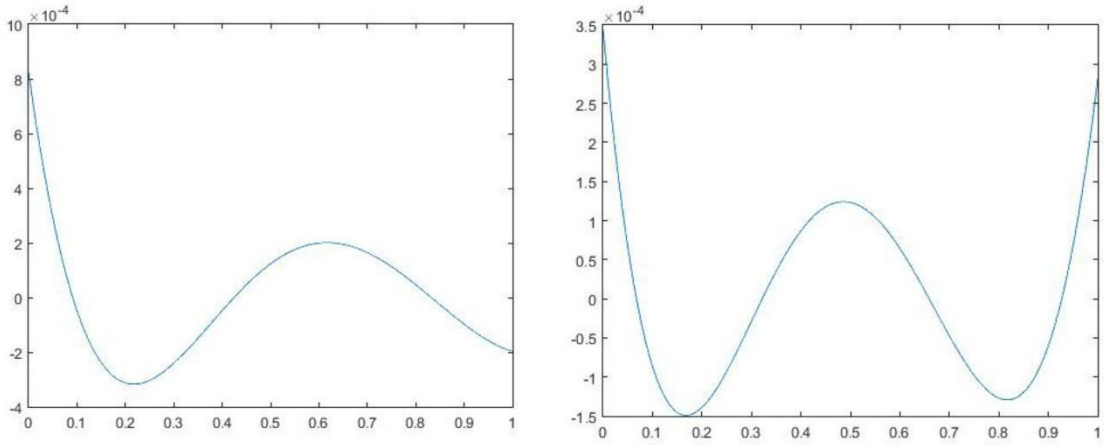
Kada funkciju aproksimiramo polinomom stupnja 3, smanji se pogreška, ali je i dalje jako velika oko graničnih točaka kao što je 1, a to je zato što smo uzimajući cijeli interval [0,1] najbolje opisali sredinu interval I nemamo nikakvih informacija o rubnim točkama i njihovim odnosima.

Koeficijenti polinoma su : -0.0096 -0.0315 0.1662 -0.0003.



Slika 18 aproksimacija polinomom trećeg stupnja

Potrebno je interval podijeliti na odsječke kojima ćemo opisati funkciju i za to se primjenjuje polinom 2.stupnja. Prilikom izračuna se ne gubi u prevelikoj mjeri preciznost zbog uzastopnog množenja i zaokruživanjem jer smo smanjili broj tih operacija, a i dalje je dovoljno precizno za što bolju aproksimaciju funkcije.

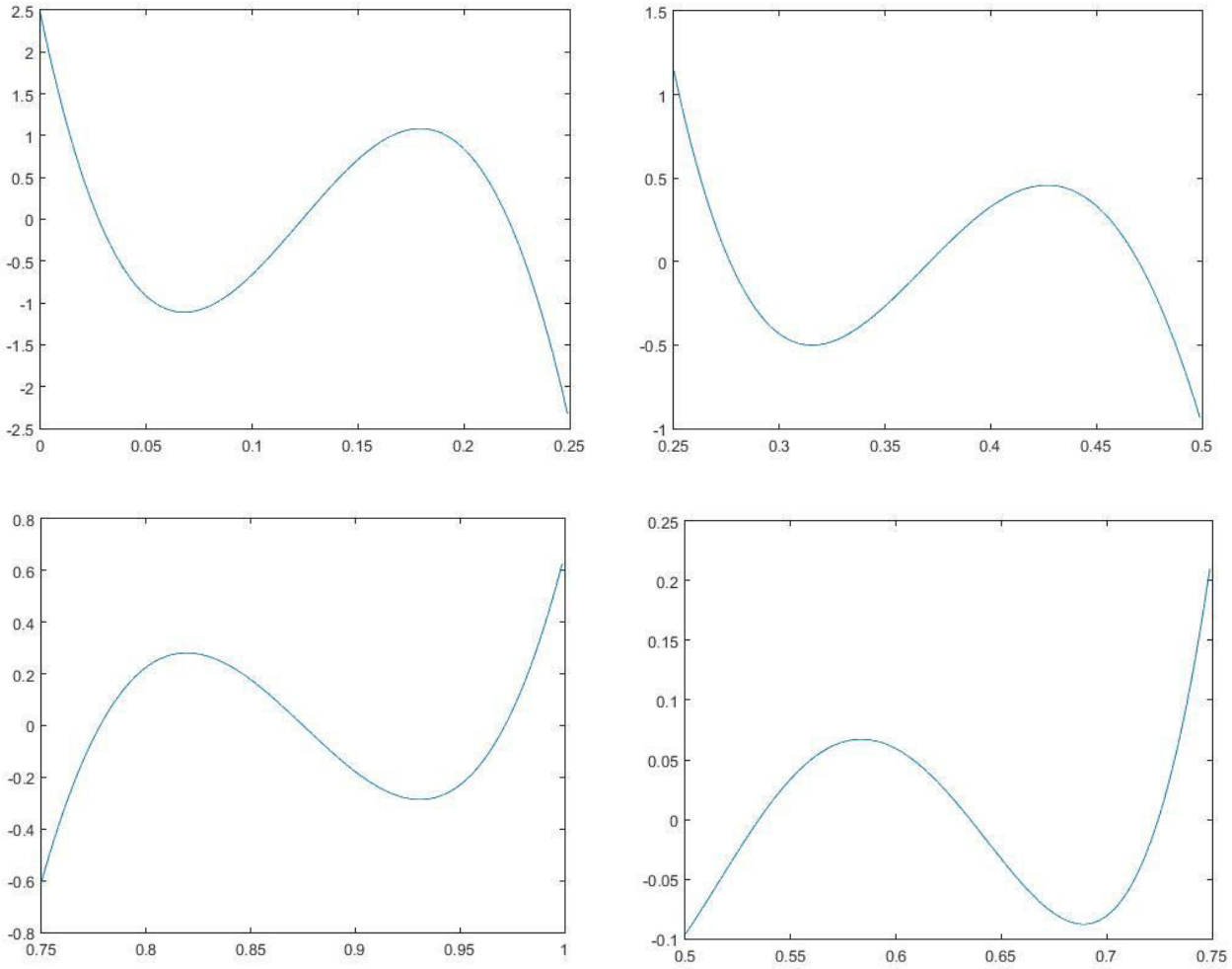


Slika 19 Pogreške nastale aproksimacijom polinoma trećeg stupnja

Ograničavajući funkciju na manje odsječke dobivamo precizniju aproksimaciju i manje pogreške na pojedinom intervalu.

Primjer za prvi odsječak :

```
p=polyfit(x(1:250),f(1:250),2)
yp=polyval(p,x)
plot(x(1:250),(f(1:250)- yp(1:250))*2^16)
```



Slika 20 Prikaz funkcije na odsječcima

Prilikom prilagodbe frakcionalnoj aritmetici preciznost koja se izgubila na zaokruživanju to jest množenju koeficijenata polinoma sa 2^15 i zatim dijeljenjem sa 2^15 je $1*10^{-4}$

Tablica 1 Koeficijenti [0,0.25]

Koeficijenti [0,0.25]	-0.151774484603907	1.287995199421376	-0.000299048675167
Round(p*2^15)*2^-15	-0.151763916015625	1.287994384765625	-0.000305175781250

Zatim funkciju na svakom od odabrana četiri intervala opisujemo polinomom točnije Hornerovim algoritmom. Prilikom svake operacije množenja ili zbrajanja potrebno je prilagođavati izlaze frakcionalnoj aritmetici to jest registrima procesora.

Hornerov algoritam, gdje je xq frakcionalna vrijednost ulaznog argumenta .

Polinom= $\left(xq * qp(1) + qp(2)\right) * xq + qp(3)$

Primjer ulazne vrijednosti 0.099 :

P1=$(xq * 2^{16}) * qp(1) * 2^{15}$ qp su ulazni koeficijenti polinoma u Q15 formatu

P1_b =$(xq * 2^{16}) * qp(1) * 2^{15} + qp(2) * 2^{15} * 2^{16}$

$p1_{\_b} = p1_b * 2^{-31}$ = 1.272969905287027

Rezultat bez operacija sa registrima i zaokruživanjima daje :

$xq * qp(1) + qp(2)$= 1.272969905287027

p1 $= round(p1_b * 2^{-16})$

p=$p1 * xq * 2^{16} * 2^{-31}$ = 0.126023750752211 dok rezultat bez zaokruživanja daje 0.126022777488743 što znači da se konačni rezultat razlikuje tek na šestoj decimali, što je zanemarivo.

Rj(q)=$p1 * xq * 2^{16} + qp(3) * 2^{31}$

Rj=rj(q)*2^{-31} = 0.125718574970961 dok je atan(xq)/(pi/4) = 0.12564008076482.

Rješenja u Q formatu se razlikuju na 5LSB (8234,8239).

Potrebno je provjeriti i mogućnost kada su ulazne vrijednosti i koeficijenti 32-bitne vrijednosti u formatu Q31 sa jednim bitom predznaka. Za isti primjer uzet kao i u implementaciji sa 16-bitnim koeficijentima dobivamo preciznije rješenje.

Tablica 2 Usporedba rješenja

	Potrebno rješenje	**Dobiveno rješenje**	**Potrebno rješenje- frakcionalna aritmetika**	**Dobiveno rješenje- frakcionalna aritmetika**
16-bitni podaci	0.12564008076482	0.125718574970961	8234	8239
32-bitni podaci	0.12564008076482	0.125724934305301	539625326	539984481

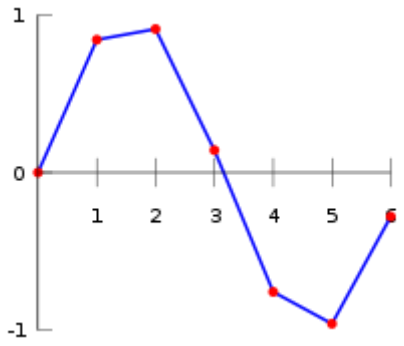
Uspoređivanjem dobivenih vrijednosti s rješenjima koda u C-u, postižu se različite vrijednosti jer je kod u C jeziku utemeljen na Taylorovoj aproksimaciji čime se dodatno gubi na preciznosti rješenja. Neka je ulazni vektor kojeg je potrebno obraditi jednak $z = \frac{1}{2} + \frac{\sqrt{3}}{2} i$. Taj vektor predstavlja fazni kut od $\frac{\pi}{3}$.

Tablica 3 Rezultati različith implementacija

Rezultat programa u C-u	**Rezultat simulacije u Matlabu**
0.523599526653246 = 34315	0.5235987755= 34314

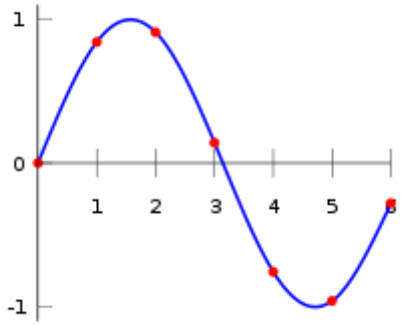
6. Oblici interpolacije

Interpolacija u matematičkom smislu označava metodu konstrukcije novih točaka podataka unutar raspona diskretnog skupa poznatih točaka podataka. U inženjerstvu i znanosti često ima mnogo točaka podataka prikupljenih uzorkovanjem, te se njome pokušava konstruirati funkcija koja približno odgovara svim točkama podataka. Interpolacija je specifični slučaj prilagodbe krivulje u kojem funkcija mora točno prolaziti točkama podataka. Česti problem koji je blisko povezan s interpolacijom je aproksimacija složene funkcije jednostavnom funkcijom. Taj problem se najčešće rješava tako da izaberemo nekoliko poznatih točaka iz složene funkcije, zatim izradimo preglednu tablicu i pokušamo interpolirati te točke i konstruirati jednostavnije funkcije. Jedan od najjednostavnijih oblika interpolacije je izračun aritmetičke sredine iz vrijednosti dviju susjednih točaka. Jedna od najjednostavnijih metoda je po dijelovima linearna interpolacija koja uzima dvije točke recimo (Xa,Ya) i (Xb,Yb).



Slika 21 Prikaz podataka s primijenjenom linearnom interpolacijom

Polinom u općem obliku zapisujemo kao $f(x)=a_n x^n+a_{n-1}x^{n-1}+...+a_1x+a_0$, dakle kao izraz s n+1 nepoznanice. Kako bi ih odredili potreban nam je n+1 podatak, odnosno točke interpolacije. Iz toga slijedi da s n točaka interpolacije možemo odrediti polinom stupnja n-1 koji kroz njih prolazi.

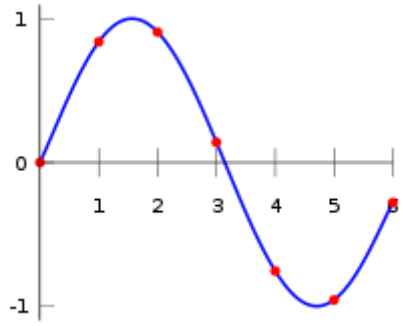


Slika 22 Prikaz podataka s primijenjenom polinomna interpolacijom

Polinomna interpolacija ipak ima neke nedostatke a to je da je računski u usporedbi s linearnom interpolacijom. Štoviše, polinomijalna interpolacija ne mora uopće biti točna, posebice na krajnjim Ovi nedostatci mogu se izbjeći uporabom spline interpolacije.

6.1. Spline funkcija

Kako po dijelovima linearna interpolacija koristi linearnu funkciju na svakom intervalu $[x_k,x_{k+1}]$, spline interpolacija koristi polinome manjeg stupnja na svakom intervalu i odabire polinomne dijelove tako da glatko pristaju. Rezultirajuća funkcija naziva se spline. Primjerice, prirodni kubični spline je segmentno kubičan i dvostruko kontinuirano diferencijabilan. Štoviše, njegova druga derivacija je nula na krajnjim točkama.



Slika 23 Prikaz podataka sa primijenjenom spline interpolacijom

Pitanje je kako ostvariti spline funkciju arkus tangens funkcije na intervalu [0,1] koji smo koristili i u implementaciji u Matlabu.

S obzirom da smo funkciju aproksimirali polinomima na intervalima [0,0.25], [0.25,0.5], [0.5,0.75] i [0.75,1] kao krajnje točke podataka za ostvarivanje spline funkcije po intervalima uzima se upravo 0,0.25,0.5, 0.75 i 1. Svaki od pojedinih intervala koji se nalazi između zadanih točaka prikazuje se polinomom drugog stupnja pa govorimo o kvadratnoj spline funkciji.

Tablica 4 Točke za ostvarivanje spline funkcije

x	atan(x)
0	0
0.25	0.244978663126864
0.5	0.463647609000806
0.75	0.643501108793284
1	0.78539163397448

S obzirom da imamo pet točaka, stvaramo četiri kvadratna polinoma koji prolaze kroz njih.

Intervali su opisani zadanim polinomima :

$$f(x) = a_1x^2 + b_1x + c_1, \qquad 0 \le x \le 0.25 \tag{3}$$

$$= a_2x^2 + b_2x + c_2, \qquad 0.25 \le x \le 0.5 \tag{4}$$

$$= a_3x^2 + b_3x + c_3, \qquad 0.5 \le x \le 0.75 \tag{5}$$

$$= a_4x^2 + b_4x + c_4, \qquad 0.75 \le x \le 1 \tag{6}$$

Svaka kvadratna jednadžba prolazi kroz dvije zadane točke na primjer $a_1x^2 + b_1x + c_1$, prolazi kroz x=0 i x=0.25 pa vrijedi :

$$a_1(0)^2 + b_1(0) + c_1 = 0 \tag{7}$$

$$a_1(0.25)^2 + b_1(0.25) + c_1 = 0.244978663126864 \tag{8}$$

Kvadratne spline funkcije imaju uvjet neprekidne derivacije u unutarnjim točkama pa to se to svojstvo iskorištava za dobivanje sljedećih jednadžbi :

$$2a_1(0.25)+b_1-2a_2(0.25)-b_2=0 \qquad \text{u x=0.25} \qquad (9)$$

$$2a_2(0.5)+b_2-2a_3(0.5)-b_3=0 \qquad \text{u x=0.5} \qquad (10)$$

$$2a_3(0.75)+b_3-2a_4(0.75)-b_4=0 \qquad \text{u x=0.75} \qquad (11)$$

Također pretpostavljamo da je prva funkcija $a_1x^2+b_1x+c_1$ linearna i stoga da je a_1 =1.

Kombinirajući sve zadane jednadžbe dolazi se do matrice koja daje zadano rješenje to jest potrebne koeficijente.

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.0625 & 0.25 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.0625 & 0.25 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.25 & 0.5 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0.5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5625 & 0.75 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5625 & 0.75 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0.5 & 1 & 0 & -0.5 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1.5 & 1 & -1.5 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ a_2 \\ b_2 \\ c_2 \\ a_3 \\ b_3 \\ c_3 \\ a_4 \\ b_4 \\ c_4 \end{bmatrix} =$$

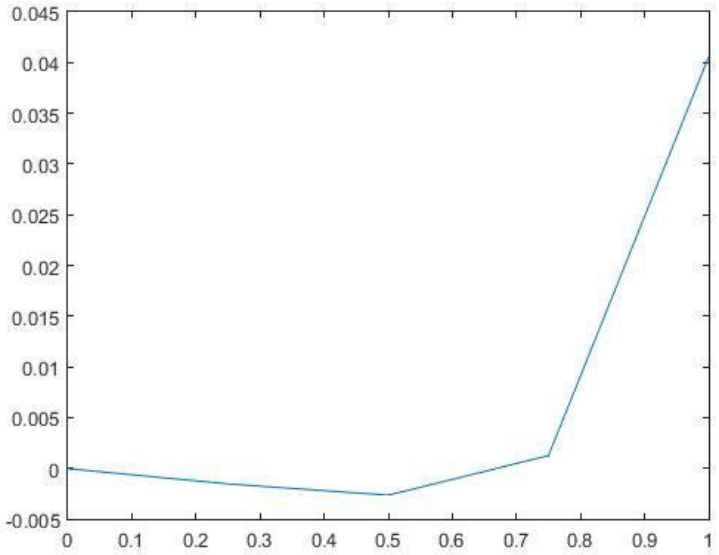
$$\begin{bmatrix} 0 \\ 0.24497866312684 \\ 0.24497866312684 \\ 0.463647609000806 \\ 0.463647609000806 \\ 0.643501108793284 \\ 0.643501108793284 \\ 0.785398163397448 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Dobiveni koeficijenti su :

Tablica 5 Koeficijenti spline funkcije

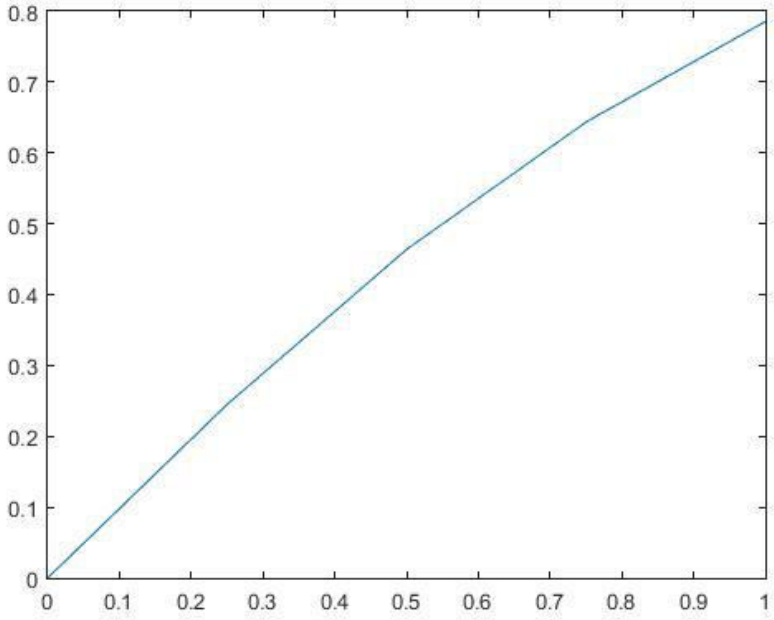
i	a_i	b_i	c_i
1	0	0.979919678148594	0
2	-0.420952380952380	1.190391459074733	-0.02631578947368
3	-0.200091785222579	0.9695290858725761	0.028901734104046
4	0.6260330578512396	-0.527972027972028	0.687331536388140

Koeficijente polinoma koje smo dobili već korištenim funkcijama polyval i polyfit također možemo prikazati na grafu. Kao i očekivano taj graf neće izgledati pravilno. Naime uzeli smo koeficijente polinoma kojima smo aproksimirali manje intervale i oni se međusobno ne nadovezuju jedni na druge. Upravo zbog toga koristimo spline funkciju koja omogućava glatke prijelaze između polinomnih odsječaka.



Slika 24 Graf polinoma po odsječcima

Kada bi na isti način išli prikazati funkciju koeficijentima gore navedenima, izračunatima ručnim putem i matricom, također ne bi dobili savršeno glatku krivulju jer nam nedostaje točaka i podataka kao što su vrijednosti prvih derivacija. Zato Matlab ima ugrađenu funkciju spline koja računa kubične splinove na određenim intervalima i sa određenim točkama prekida. Upravo na taj način vidimo važnost i učinkovitost spline funkcije.



Slika 25 Primjena Spline Matlab funkcije

Iz svih danih primjera vidljiva je važnost spline funkcije. Naime ona je korisna za izračunavanje vrijednosti funkcije u nepoznatim točkama ako su nam poznate samo neke točke (Xi,Yi). U ovom radu spline funkcija nema bitnu ulogu jer je primarni zadatak bio naći vrijednost kuta, a glatkoća funkcije i prijelaz između točaka polinoma ne doprinosi nikakvom pogreškom rezultatu.

7. Zaključak

Koristeći teorijsku podloge, brojne proračune i upoznavanjem sa razvojnim sustavom koji je temeljen na procesoru koji je namijenjen radu u frakcionalnoj aritmetici realiziran je algoritam koji određuje faznu razliku ultrazvučnog signala. Koristeći podatke tipa *int* i *long* napravljene su implementacije u C jeziku i Matlab- u. Cjelina gdje se računa fazni kut je samo dio koji se mora nadovezati na neke algoritme koje su ostvarili druge kolege koji su se usredotočili na istu tematiku, kao što je Goertzelov algoritam, izračun potrebnog vremena propagacije i implementacija na procesoru. Tek tada čine cjelinu i mogu se upotrijebiti, a za tu svrhu korišten je razvojni sustav eZdsp TMS320VC5505 proizvođača Texas Instruments. Važnost ovog algoritma i rezultata dobivenim radom je određivanje konačnog vremena kašnjenja iz faznih odnosa komponenata prisutnih u prijemnom signalu. Rezultati pokazuju da je jedan od boljih načina implementacije upravo korištenje 16-bitnih koeficijenata i aproksimacija Taylorovim polinomom. Problem ostaju pogreške koje se akumuliraju uzastopnim množenjima i zbrajanjima u pokušaju prebacivanja podataka u frakcionalnu aritmetiku pa sigurno postoje čak i bolja rješenja, a to bi bili intrinzici. Instrinsic funkcije se koriste u nekim programskim jezicima i poznate su prevodiocu ,a još se nazivaju i buit-in funkcije jer približavaju programski jezik i naredbe na najnižu, to jest procesorsku razinu. Učinkovitije su od običnih poziva funkcija, zbog toga što nije potrebno povezivanje pri pozivu funkcije, već prevodioc direktno mapira intrinzičnu funkciju u niz asemblerskih instrukcija. Na taj način bi mogli izravno utjecati na registre što bi rezultiralo manjom greškom.

8. Literatura

[1] Wikipedia, Anemometar, 2016, URL: https://hr.wikipedia.org/wiki/Anemometar#Vrste_anemometara

[2] Mathworks, atan2, URL: https://www.mathworks.com/help/matlab/ref/atan2.html

[3] Wikipedia, atan2, 2016, URL: https://en.wikipedia.org/wiki/Atan2

[4] Račun beskonačnih polinoma , Diferencijalni račun, available on: www.fsb.unizg.hr

[5] Projektiranje FIR filtara metodom vremenskih otvora, FER, URL; https://www.fer.hr/_download/repository/DOS0304_vj3.pdf

[6] Ohio.Edu, Polynomial and Spline interpolation, URL: https://www.ohio.edu/cas/math/undergrad/courses-resources/upload/lecture19.pdf

[7] Mathworks, fixed point, URL: https://www.mathworks.com/examples/matlab-fixed-point-designer/mw/fixedpoint_product-fixpt_atan2_demo-calculate-fixed-point-arctangen

[8] Spectrum Digital, DSP Development System, 2009
URL: TMS320VC5505 eZdsp USB Stick, Technical Reference, http://support.spectrumdigital.com/boards/usbstk5505/revb/files/usbstk5505_TechRef_revb.pdf, 1.6.2016

[9] Mathworks, spline, URL: https://www.mathworks.com/help/matlab/ref/spline.html

[10] Direktna digitalna sinteza visoke točnosti korištenjem kubične B-spline interpolacije, Marko Brezović, 2009

[11] Wikipedia, Interpolacija, 2013, URL: https://hr.wikipedia.org/wiki/Interpolacija

[12] Stackoverflow, URL: https://stackoverflow.com/questions/41920162/calculate-matrix-in-cubic-spline-interpolation

[13] Stackoverflow, URL: https://stackoverflow.com/questions/31006923/adding-piecewise-polynomials-in-matlab

[14] Kratke upute za korištenje Matlaba, Tomislav Petković,Zagreb travanj 2005
URL: https://www.fer.unizg.hr/_download/repository/matlab_upute.pdf

[15] Wireless World 2012, converter, URL: http://www.rfwireless-world.com/calculators/floating-vs-fixed-point-converter.html

9. Naslov, sažetak i ključne riječi

UČINKOVITA IZVEDBA TRIGONOMETRIJSKIH FUNKCIJA POLINOMIJALNOM APROKSIMACIJOM

Sažetak

U okviru završnog rada potrebno je istražiti mogućnosti aproksimativnih programskih izvedbi izračuna trigonometrijskih funkcija korištenjem cjelobrojne aritmetike. Zadatak je namijenjen za primjenu na anemometru,a to je mjerni instrument koji mjeri brzinu i smjer vjetra. U ovom rada razmatra se ultrazvučni anemometar sa četiri ultrazvučna pretvarača to jest senzora. Izračun brzine i smjera vjetra temelji se na vremenu propagacije potrebnom bloku sinusnih signala da prijeđe put od odašiljača do prijemnika. Iz određenog vremena kašnjenja potrebno je izračunati fazni kut i fazne odnose komponenti prisutnih u prijemnom signalu. Za određivanje kuta korištena je Taylorova aproksimacija i po odsječcima polinomni modeli nižeg stupnja. Algoritmi su ostvareni u C-u i Matlabu i zatim uspoređeni njihovi rezultati kako bi se pronašao najefikasniji način implementacije funkcije na procesoru. Korišteni procesor je eZdsp TMS320VC5505.

Ključne riječi:

anemometar, implementacija trigonometrijske funkcije, matlab, atan2, polinomijalna aproksimacija, spline

EFFICIENT IMPLEMENTATION OF TRIGONOMETRIC FUNCTIONS USING POLYNOMIAL APPROXIMATION

Summary

The purpose of this paper was to implement trigonometric functions using polynomial approximations. Therefore we used anemometer, a device used for measuring the speed and the wind direction. This paper is mainly based on ultrasonic continuous - wave anemometer properties which has two pair of ultrasonic transducers. The measure of wind speed and direction is based on the time of flight of sonic pulses between pairs of transducers. Using the delay time as a starting point, we need to calculate the phase ratios of the components and the phase angle. In the process was used Taylor's approximation method to determine the phase angle and also the other way was using lower degree polynomial models. Algorithms that were used to implement the function are in C and Matlab. Their results were compared in order to find the most efficient way of implementing the function on the processor. The processor used for the implementation is eZdsp TMSV320VC5505.

Keywords:

Anemometer, implementation of trigonometric function, matlab, atan2, polynomial approximation