





## 5. Diskrečioji Furjė transformacija

### Furjė transformacijos šeima

Laiko ir dažnio domenai yra alternatyvūs tų pačių signalų vaizdavimo būdai. Furjė transformacija, pavadinta jos pradinio iniciatoriaus, prancūzų matematiko ir fiziko Žano Batisto Žozefo Furjė (Jean-Baptiste Joseph Fourier), vardu, yra matematinis ryšys tarp šių dviejų vaizdavimo būdų. Jei signalas pakeičiamas vienoje srityje, jis taip pat pasikeis ir kitoje srityje, nors paprastai ne tokiu pačiu būdu. Pavyzdžiui, kompozicija (*convolution*) laiko srityje yra lygiavertė daugybai dažnių srityje. Kitos matematinės operacijos, pavyzdžiui, sudėties, mastelio keitimo ir perstūmimo, taip pat turi atitikmenį priešingoje srityje. Šie ryšiai vadinami Furjė transformacijos savybėmis, kurios apibūdina, kaip matematinis pokytis vienoje srityje sukelia matematinį pokytį kitoje srityje.

Bendrajį *Furjė transformacijos terminą* galima suskirstyti į keturias kategorijas, atsirandančias dėl keturių pagrindinių signalų tipų, kurie įprastai yra sutinkami. Signalas gali būti *tolydus* arba *diskretus*, taip pat *periodinis* arba *aperiodinis*. Šių dviejų požymių derinys sukuria keturias kategorijas, pavaizduotas 1 pav.

Type of Transform	Example Signal
Fourier Transform <i>signals that are continuous and aperiodic</i>	
Fourier Series <i>signals that are continuous and periodic</i>	
Discrete Time Fourier Transform <i>signals that are discrete and aperiodic</i>	
Discrete Fourier Transform <i>signals that are discrete and periodic</i>	

1 pav. Keturi Furjė transformacijų tipai

#### Tolydus aperiodinis signalas

Tai gali būti, pavyzdžiui, eksponentiškai slopstantis signalas ar Normaliojo (Gauso) skirstinio kreivė. Šie signalai tęsiasi be galo tiek į neigiamą, tiek į teigiamą pusę be jokio periodinio pasikartojimo. Šio tipo signalo transformacija vadinama tiesiog *Furjė Transformacija*.

#### Tolydus periodinis signalas

Signalų pavyzdžiui: sinusinės bangos, stačiakampės bangos ir bet kuri kita bangos forma, kuri kartojasi reguliariai nuo neigiamos begalybės iki teigiamos begalybės. Ši Furjė transformacijos versija vadinama *Furjė Eilutė*.

#### Diskretus aperiodinis signalas

Šie signalai apibrėžti tik atskiruose taškuose tarp teigiamos ir neigiamos begalybės ir nesikartoja periodiškai. Šis Furjė transformacijos tipas vadinamas Diskrečiojo Laiko Furjė Transformacija.

#### Diskretus periodinis signalas

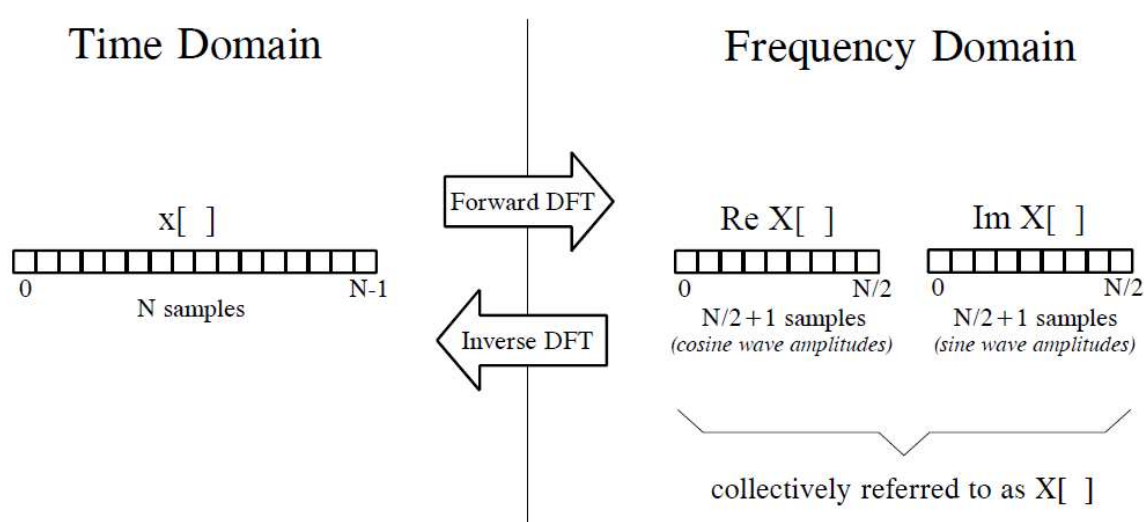
Tai diskretus signalas, kuris periodiškai kartojasi nuo neigiamos iki teigiamos begalybės. Šis Furjė transformacijos tipas kartais vadinamas diskrečiąja Furjė eilute, bet dažniausiai vadinamas *Diskrečiąja Furjė transformacija*.

Kadangi skaitmeniniai kompiuteriai gali dirbti tik su diskrečiąja ir baigtinio ilgio informacija, vienintelis Furjė transformacijos tipas, kurį galima naudoti SSA, yra **diskrečioji Furjė transformacija (DFT)**.

Kiekvieną iš keturių Furjė transformacijų taip pat galima suskirstyti į realiąją ir kompleksinę versijas. Realioji versija yra paprasčiausia, nes sintezei ir skaidymui naudojami paprasti skaičiai ir algebra. Furjė transformacijų kompleksinės versijos yra nepaprastai sudėtingos, joms reikia naudoti kompleksinius skaičius.

## Realiosios DFT žymėjimas ir formatas

Diskrečioji Furjė transformacija keičia  $N$  taškų įvesties signalą į du  $N/2+1$  taškų išvesties signalus. Įvesties signalas yra skaidomas signalas, o dviejuose išvesties signaluose yra sudedamųjų sinusinių ir kosinusinių bangų amplitudės.



2 pav. DFT terminologija

Sakoma, kad įvesties signalas yra *laiko srityje*, nes dažniausiai į DFT patenka signalas, sudarytas iš vienodais laiko intervalais paimtų signalo imčių. Terminas *dažnių sritis* vartojamas sinusinių ir kosinusinių bangų amplitudėms apibūdinti.

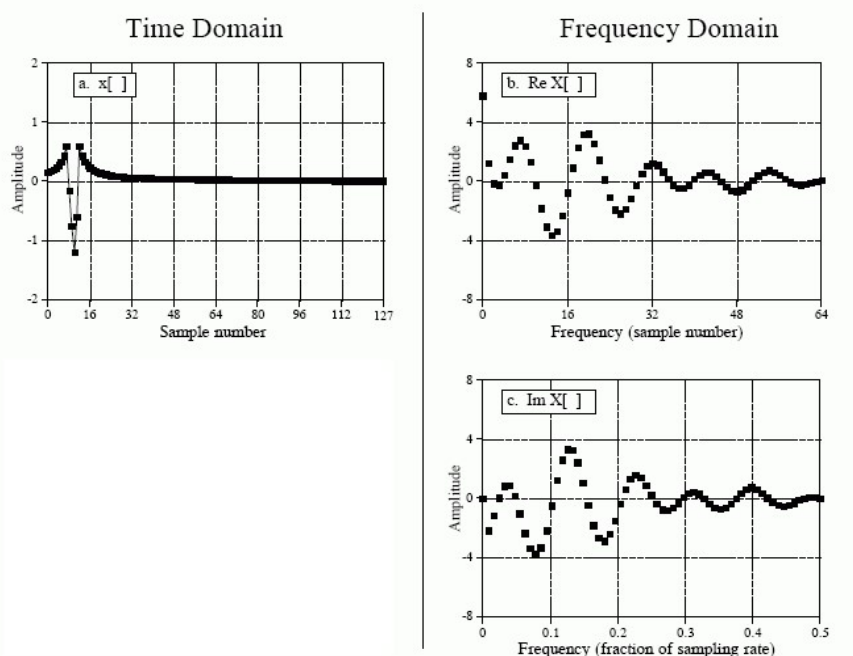
Dažnių srityje pateikiama lygiai tokia pati informacija kaip ir laiko srityje, tik kita forma. Jei yra žinoma viena sritis, tuomet galima apskaičiuoti ir kitą. Turint laiko srities signalą, dažnių srities skaičiavimo procesas vadinamas *dekompozicija*, *analize*, *tiesiogine DFT* arba *tiesiog DFT*. Jei žinote dažnių sritį, laiko srities skaičiavimas vadinamas *sintezė* arba *atvirkštinė DFT*.

Imčių skaičius laiko srityje paprastai žymimas kintamuoju  $N$ . Nors  $N$  gali būti bet koks teigiamas sveikasis skaičius, paprastai pasirenkama  $2^n$ , t. y. 128, 256, 512, 1024 ir t. t. Tai lemia dvi priežastys. Pirma, skaitmeninėje duomenų saugykloje naudojamas dvejetainis adresavimas, todėl natūralus signalo ilgis yra  $2^n$ . Antra, efektyviausias DFT skaičiavimo algoritmas – *Greitoji Furjė transformacija* (angl. *Fast Fourier Transform, FFT*) – įprastai atliekami skaičiavimai su masyvų dydžiu  $N$ , kuris yra  $2^n$ . Paprastai  $N$  pasirenkama nuo 32 iki 4096.

Standartinis SSA žymėjimas laiko srities signalų atvaizdavimui naudoja mažąsias raides, pavyzdžiui,  $x[n]$ ,  $y[n]$  ir t.t. Atitinkamai didžiosios raidės naudojamos šių signalų dažnių srities atvaizdavimui, pavyzdžiui,  $X[k]$ ,  $Y[k]$  ir t.t. Sakykime,  $N$  taškų laiko srityje signalas saugomas  $x[n]$ . Dažnių srityje šis signalas bus atitinkamai žymimas  $X[k]$ , kuris susidės iš dviejų dalių –  $N/2+1$  ilgio masyvų, realiosios dalies  $\text{Re}X[k]$  ir menamosios dalies  $\text{Im}X[k]$ .

## Dažnių srities nepriklausomas kintamasis

Pavyzdžiui, laiko srities signalas yra saugomas masyve, kurio ilgis  $N = 128$ : nuo  $x[0]$  iki  $x[127]$ . Dažnio srityje signalai yra dviejuose masyvuose:  $ReX[0]$  iki  $ReX[64]$  ir  $ImX[0]$  iki  $ImX[64]$ .



3 pav. Horizontalios ašies žymėjimas dažnių srityje

Dažnių srityje horizontalioji ašis gali būti žymima keturias skirtingais būdais, kurie visi yra įprasti SSA:

- Pirmuoju atveju horizontalioji ašis žymima skaičiais nuo 0 iki 64, kas atitinka signalo imtis masyvuose nuo 0 iki  $N/2$ . Kuomet naudojamas toks žymėjimas, indeksas dažnių srityje yra sveikasis skaičius, pavyzdžiui,  $ReX[k]$  ir  $ImX[k]$ , kur  $k$  kinta 0 iki  $N/2$  žingsniu po vienetą.
- Antruoju atveju horizontalioji ašis žymima diskretizavimo dažnio dalimis, t.y. horizontaliosios ašies vertės kinta nuo 0 iki 0,5 (kadangi dažnių dedamosios gali kisti nuo 0 iki pusės diskretizavimo dažnio). Šiuo atveju naudojamas indeksas  $f$ , realioji ir menamoji signalų dalis žymima  $ReX[f]$  ir  $ImX[f]$ , kur  $f$  įgyja  $N/2+1$  vienodai išdėstytų reikšmių nuo 0 iki 0,5.
- Trečiasis būdas panašus į antrąjį, tik horizontalioji ašis padauginta iš  $2\pi$ . Šiame žymėjime naudojamas indeksas  $\omega$  - mažoji graikiška raidė *omega*. Šiame užrašė realioji ir menamoji dalys rašomos  $ReX[\omega]$  ir  $ImX[\omega]$ , kur  $\omega$  įgyja  $N/2+1$  vienodai išsidėsčiusių reikšmių tarp 0 ir  $\pi$ .
- Ketvirtasis būdas - horizontaliąją ašį žymėti pagal tam tikru atveju naudojamus analoginius dažnius. Pavyzdžiui, jei tiriamos sistemos diskretizavimo dažnis yra 10 kHz, dažnių srities grafikai kistų nuo 0 iki 5 kHz.

## DFT bazinės funkcijos

DFT naudojamos sinusinės ir ko-sinusinės bangos paprastai vadinamos DFT bazinėmis funkcijomis. Kitaip tariant, DFT išvestis yra skaičių, kurie nurodo amplitudes, rinkinys. Bazinės funkcijos yra sinusinių ir kosinusinių bangų su vienetine amplitude rinkinys.

Jei kiekvieną amplitudę (dažnių srityje) priskirsime atitinkamai sinusinei arba ko-sinusinei bangai (bazinėms funkcijoms), gausime atitinkamo mastelio sinusines ir ko-sinusines bangas, kurias galima tarpusavyje sudėti ir tokiu būdu suformuoti signalą laiko srityje.

DFT bazinės funkcijos sugeneruotos remiantis šiomis lygtimis:

$$c_k[i] = \cos\left(\frac{2\pi ki}{N}\right) \quad (1)$$

$$s_k[i] = \sin\left(\frac{2\pi ki}{N}\right) \quad (2)$$

kur:  $c_k[i]$  – ko-sinusinė banga su  $ReX[k]$  amplitude;  $s_k[i]$  – sinusinė banga su  $ImX[k]$  amplitude.

Parametru  $k$  nustatomas kiekvienos sinusoidės dažnis. Pavyzdžiui,  $c_1[i]$  yra kosinuso banga, kuri sudaro vieną pilną ciklą per  $N$  taškų,  $c_5[i]$  yra kosinuso banga, kuri sudaro penkis pilnus ciklus per  $N$  taškų ir t. t. Tai svarbi sąvoka, padedanti suprasti bazines funkcijas; dažnio parametras  $k$  yra lygus pilnų ciklų, kurie įvyksta per  $N$  signalo taškų, skaičiui.

$ReX[0]$  masyvo elementas saugo visų signalo imčių laikinėje srityje vidutinę vertę. Elektronikoje tai atitinka signalo nuolatinę dedamąją (*DC offset*).

$ImX[0]$  ir  $ImX[N/2]$  neneša jokios informacijos, šių masyvo elementų vertė visada lygi nuliui.

### Atvirkštinės DFT skaičiavimas (Sintezė)

Sintezės lygtis:

$$x[i] = \sum_{k=0}^{\frac{N}{2}} Re\bar{X}[k] \cos\left(\frac{2\pi ki}{N}\right) + \sum_{k=0}^{\frac{N}{2}} Im\bar{X}[k] \sin\left(\frac{2\pi ki}{N}\right) \quad (3)$$

Remiantis 3-iaja lygtimi, bet koks  $N$  taškų signalas  $x[i]$  gali būti atkurtas sudedant  $N/2+1$  ko-sinusinių bangų ir  $N/2+1$  sinusinių bangų. Ko-sinusinių ir sinusinių bangų amplitudės saugomos atitinkamai masyvuose  $Im\bar{X}[k]$  ir  $Re\bar{X}[k]$ . Sintezės lygtyje šios amplitudės padaugintos iš bazinių funkcijų, kad būtų sukurtos atitinkamos amplitudės ko-sinusinės ir sinusinės bangos.

Sintezės lygčiai naudojamos amplitudžių reikšmės rasti naudojamas paprastas normalizavimas:

$$Re\bar{X}[k] = \frac{ReX[k]}{N/2} \quad (4)$$

$$Im\bar{X}[k] = -\frac{ImX[k]}{N/2} \quad (5)$$

$$Re\bar{X}[0] = \frac{ReX[0]}{N} \quad (6)$$

$$Re\bar{X}[N/2] = \frac{ReX[N/2]}{N} \quad (7)$$

Atsižvelgiant į pateiktas formules, prieš atliekant signalo sintezę, turi būti atlikti šie veiksmai:

- Visos amplitudžių reikšmės dažnių srityje turi būti padalintos iš  $N/2$
- Pakeistas menamosios dalies amplitudžių ženklas (padauginta iš  $-1$ )
- Padalinti pirmąją ir paskutiniąją realios dalies masyvo reikšmes  $ReX[0]$  ir  $ReX[N/2]$  iš dviejų

## DFT skaičiavimas (Analizė)

DFT galima apskaičiuoti trimis visiškai skirtingais būdais. visi trys šie metodai duoda vienodą rezultatą.

### DFT skaičiavimas tiesioginiu būdu (*DFT by Simultaneous Equations*)

Šis metodas yra naudingas norint suprasti DFT, tačiau jis yra per daug neefektyvus, kad būtų praktiškai naudingas. Jis reikalauja labai daug skaičiavimų ir praktiškai niekada nenaudojamas DSP.

Jei seka turi  $N$  atskaitų, tai DFT vienai atskaitai apskaičiuoti reikia atlikti  $N$  kompleksinių sandaugų ir  $N-1$  kompleksinių sudėčių, o visoms  $N$  atskaitoms apskaičiuoti – maždaug po  $N^2$  kompleksinių sandaugų ir sudėčių. Sandaugų ir sudėčių skaičius gana gerai atspindi apskaičiavimų sudėtingumą. Taigi tiesioginiu būdu skaičiuojant sekos iš  $N$  atskaitų DFT, reikia sugaišti laiką, proporcingą  $N^2$ .

### DFT pagal koreliaciją

Šis metodas remiasi žinomos bangos aptikimu kitame signale. Norint tai padaryti, reikia abu signalus sudauginti ir sudėti visus gauto signalo taškus. Gautas rezultatas – skaičius, nurodantis abiejų signalų panašumą. Praktikoje koreliacijos metodas yra priimtinausias skaičiavimo būdas, jei DFT turi mažiau nei 32 taškus. Kitu atveju naudojama GFT.

### GFT – Greitoji Furjė transformacija (*FFT – Fast Furje transformation*)

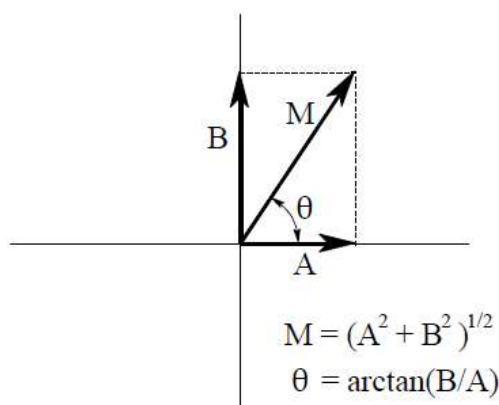
Greitoji Furjė transformacija - tai algoritmas, kuris išskaido DFT su  $N$  taškų į  $N$  DFT su vienu tašku. FFT paprastai yra šimtus kartų greitesnė už kitus metodus. Plačiau apie FFT algoritmo veikimą<sup>1</sup>

## Polinė koordinačių sistema

Dažnių sritis - tai ko-sinusinių ir sinusinių bangų amplitudžių grupė (su nedideliais mastelio pakeitimais). Tai vadinama *stačiakampiu žymėjimu* (stačiakampe koordinačių sistema).

Alternatyviai dažnių sritis gali būti išreikšta *poline forma* (polinė koordinačių sistema). Polinėje koordinačių sistemoje  $ReX[]$  ir  $ImX[]$  pakeičiami kitais dviem masyvais – Magnitude  $MagX[]$  ir faze  $PhaseX[]$ .

Magnitudė ir fazė „vienas prie vieno“ atitinka realiąją ir menamąją signalo dažnių srityje dalis. Pavyzdžiui,  $MagX[0]$  ir  $PhaseX[0]$  reikšmės yra apskaičiuojamos atitinkamai naudojant tik  $ReX[0]$  ir  $ImX[0]$ .



4 pav. Stačiakampių koordinačių perskaičiavimas į polinę sistemą

<sup>1</sup> How the FFT works: <https://www.dspguide.com/ch12/2.htm>

Poliarinėje sistemoje  $MagX[k]$  atitinka ko-sinusinės bangos amplitudę, o  $PhaseX[k]$  – ko-sinusinės bangos fazės kampą. Toliau pateiktomis lygtimis dažnių sritis iš stačiakampės sistemos perskaičiuojama į polinę ir atvirkščiai:

$$MagX[k] = \sqrt{ReX[k]^2 + ImX[k]^2} \quad (8)$$

$$PhaseX[k] = \arctan(ImX[k]/ReX[k]) \quad (9)$$

$$ReX[k] = MagX[k] \cos(PhaseX[k]) \quad (10)$$

$$ImX[k] = MagX[k] \sin(PhaseX[k]) \quad (11)$$

Polinėje sistemoje DFT išskaido  $N$  taškų signalą į  $N/2+1$  ko-sinuso bangas, kurių kiekviena turi tam tikro dydžio amplitudę (vadinamą magnitude) ir fazės poslinkį. Kodėl polinėje sistemoje naudojamos ko-sinusinės, o ne sinusinės bangos? Sinusinės bangos negali perteikti signalo nuolatinės srovės komponentės, nes nulinio dažnio sinusinę bangą sudaro visi nuliai.

Kada reikia naudoti stačiakampę, o kada polinę koordinatų sistemą? Stačiakampis užrašymas paprastai geriausiai tinkamas atliekant skaičiavimus, pavyzdžiui, lygtyse ir kompiuterinėse programose. Palyginimui, grafikai beveik visuomet vaizduojama poline sistema. Žmonės beveik neįmanoma suprasti dažnių srities signalo charakteristikų žiūrint į realiąją ir menamąją dalis. Tipinėje kompiuterinėje programoje dažnių srities signalas saugomas stačiakampėje sistemoje, kol prireikia jį atvaizduoti. Tuomet atliekamas konvertavimas iš stačiakampės į polinę sistemą.

## Polinės sistemos niuansai

Naudojant polinę sistemą kyla tam tikrų niuansų - nepatogumų. Šie nepatogumai nėra labai dideli, tačiau vis tiek gali erzinti.

### 1 nepatogumas: radianai ir laipsniai

Fazė gali būti išreikšta tiek laipsniais, tiek radianais. Kuomet fazė išreikšta laipsniais, fazės reikšmės kinta nuo -180 iki 180. Naudojant radianus reikšmės kinta nuo  $-\pi$  iki  $\pi$ , arba atitinkamai nuo -3.141592 iki 3.141592. Daugelyje programų trigonometrinėse funkcijose naudojami radianai, todėl gali būti nepatogu nurodyti dešimtainius skaičius. Pavyzdžiui, jei signalui reikia pridėti 90° fazės poslinkį, tokiu atveju radianais reiktų pridėti skaičių 1.570796.

Geriausias būdas spręsti šią problemą - apibrėžti konstantą  $PI = 3,141592$ .

### 2 nepatogumas: dalyba iš nulio

Konvertuojant iš stačiakampės sistemos į polinę, dažnai galima sutikti dažnių, kurių realioji dalis yra lygi nuliui, o menamoji dalis tam tikras skaičius. Tai paprasčiausiai reiškia, kad fazė yra lygiai 90 arba -90 laipsnių. Bandant programoje apskaičiuoti fazę pagal formulę:  $PhaseX[k] = \arctan(ImX[k]/ReX[k])$ , bus gauta dalybos iš nulio klaida.

Galimas būdas šiai problemai apeiti – prieš dalybos operaciją patikrinti, ar realioji dalis yra lygi nuliui. Jei ji lygi nuliui, galimi sprendimai:

- Patikrinti menamosios dalies ženklą ir fazę nustatyti atitinkamai  $\pi/2$  arba  $-\pi/2$ , o dalybą praleisti.
- Vietoje nulio įrašyti tam tikrą mažą reikšmę, taip išvengiant dalybos iš nulio.

### 3 nepatogumas: neteisingas arktangento apskaičiavimas

Šis atvejis nutinka, kuomet realioji ir menamoji dalys yra neigiamos. Pavyzdžiui,  $ReX[k] = 1$  ir  $ImX[k] = 1$ . Šiuo atveju polinėje sistemoje gausime  $MagX[k] = 1.414$  ir  $PhaseX[k] = 45^\circ$ . Tačiau lygiai

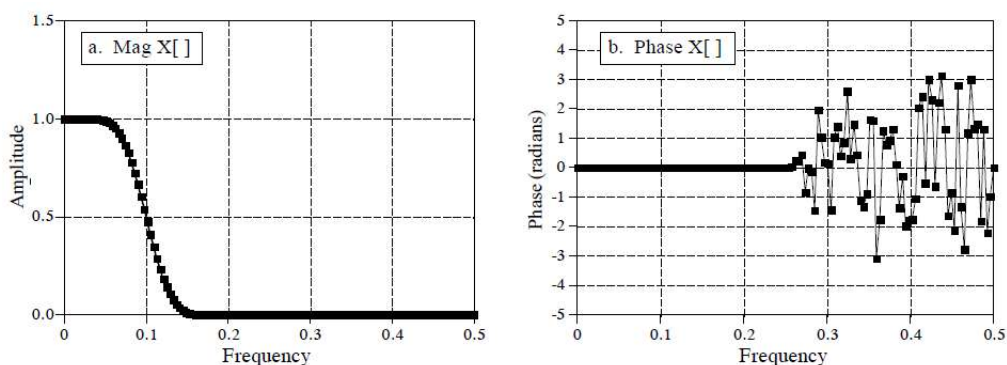
toks pats skaičiavimų rezultatas bus esant  $ReX[k] = -1$  ir  $ImX[k] = -1$ , tik šiuo atveju fazė turėtų iš tikrųjų turi būti  $-135^\circ$ . Ši klaida įvyksta, kai realioji dalis yra neigiama.

Sprendimas - patikrinti realiąją ir menamąją dalis:

- Jei abi yra neigiamos, iš apskaičiuotos fazės atimti  $180^\circ$
- Jei realioji dalis neigiama, o menamoji teigiama, prie apskaičiuotos fazės pridėti  $180^\circ$

#### 4 nepatogumas: fazės svyravimas esant mažai magnitudėi

Dažniuose, kuriuose magnitudė sumažėja iki labai mažos vertės, ji tampa nereikšminga ir pranyksta apvalinimo „triukšme“. Tuo tarpu fazę sudaro atsitiktiniai skaičiai, kurie svyruoja nuo  $-\pi$  iki  $+\pi$ . Deja, tai gali atrodyti kaip realus signalas.

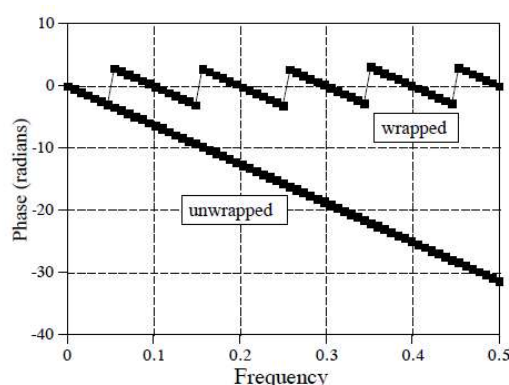


5 pav. Fazė prie mažos magnitudės

#### 5 nepatogumas: fazės $2\pi$ dviprasmiškumas

Pažvelgus į 6 pav. matyti keletas duomenų nutrūkimų. Kiekvieną kartą, kuomet atrodo, kad sekantis taškas nusileis žemiau  $-3,1259$  reikšmės, jis vėl peršoka atgal į  $3,141592$  reikšmę. Tai sinusoidės periodinio pobūdžio rezultatas. Jei prie fazės pridėsime  $2\pi$  (arba  $2\pi$  padauginimą iš bet kokio sveiko skaičiaus), sinusoidė išliks nepakitusi.

Dažniausiai fazę lengviau suprasti be šių nutrūkimų. Kaip parodyta apatinėje kreivėje, fazę galima išskleisti pridėdant arba atimant iš kiekvienos reikšmės sveiką skaičių, padauginimą iš  $2\pi$ .



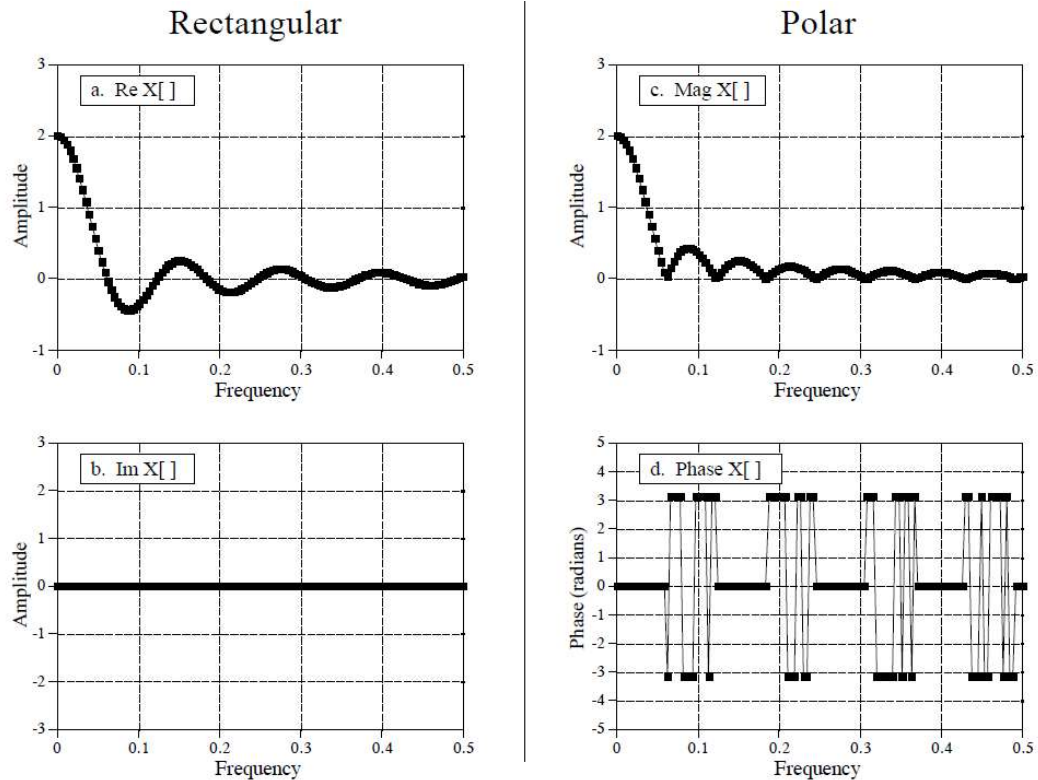
6 pav. Fazės išskleidimo pavyzdys

#### 6 nepatogumas: visuomet teigiama magnitudė

Paveikslėlyje pavaizduotas dažnių srities signalas stačiakampėje ir polinėje formoje. Realioji signalo dalis yra nuosekli ir lengvai suprantama, menama dalis yra lygi nuliui.

Tuo tarpu polinėje formoje yra staigūs nutrūkimai ir aštrūs kampai. Taip yra todėl, magnitudė pagal apibrėžimą visuomet turi būti teigiama. Kai realioji signalo dalis nukrenta žemiau nulio, magnitudė išlieka teigiama, keičiant fazę dydžiu  $\pi$ .





7 pav. Signalų pavyzdys stačiakampėje ir polinėje formoje

### 7 nepatogumas: Šuoliai tarp $\pi$ ir $-\pi$

Kadangi  $\pi$  ir  $-\pi$  reikšmės reiškia tokį patį fazės poslinkį, dėl apvalinimo triukšmo gretimi fazės taškai gali greitai „peršokti“ iš vienos į kitą reikšmę. Kaip matyti 6 pav. d) dalyje, tai gali sukelti nutrūkimus ir šuolius fazės kreivėje, kuri įprastai būtų tolygi.

## CMSIS-DSP Transform Functions

CMSIS DSP bibliotekoje galima rasti signalų apdorojimui ir transformavimui skirtas funkcijas:

- **Complex FFT Functions**

Greitosios Furjė transformacijos funkcijos, skirtos darbui su kompleksiniais skaičiais.

- **DCT Type IV Functions**

Diskrečioji ko-sinusinė transformacija (DCT – Discrete cosine transform). Plačiai naudojama signalų apdorojimui ir duomenų suspaudimui.

- **MFCC**

Mel Frequency Cepstral Coefficients (MFCCs) – plačiai naudojama technika audio signalo savybių išgavimui ir jo atpažinimui.

- **Real FFT Functions**

GFT funkcijos, skirtos realiesiems duomenims.



## Realųjų duomenų GFT funkcijos (Real FFT Functions)

[https://www.keil.com/pack/doc/CMSIS/DSP/html/group\\_\\_RealFFT.html](https://www.keil.com/pack/doc/CMSIS/DSP/html/group__RealFFT.html)

The CMSIS DSP library includes specialized algorithms for computing the FFT of real data sequences. The FFT is defined over complex data but in many applications the input is real. Real FFT algorithms take advantage of the symmetry properties of the FFT and have a speed advantage over complex algorithms of the same length.

The real length  $N$  forward FFT of a sequence is computed using the steps shown below.

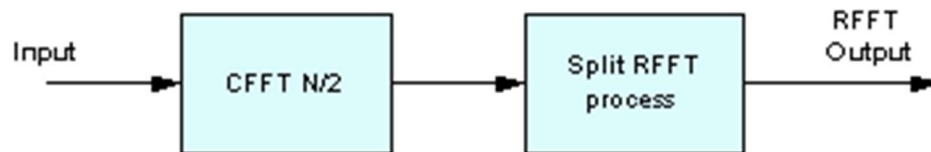


Fig. 8. Real Fast Fourier Transform

The real sequence is initially treated as if it were complex to perform a CFFT. Later, a processing stage reshapes the data to obtain half of the frequency spectrum in complex format. Except the first complex number that contains the two real numbers  $X[0]$  and  $X[N/2]$  all the data is complex. In other words, the first complex sample contains two real values packed.

There are algorithms for floating-point (f16, f32, f64), Q15, and Q31 data.

### Floating-point real FFT

The main functions are `arm_rfft_fast_f32()` and `arm_rfft_fast_init_f32()`.

The older functions `arm_rfft_f32()` and `arm_rfft_init_f32()` have been deprecated but are still documented.

The FFT of a real  $N$ -point sequence has even symmetry in the frequency domain. The second half of the data equals the conjugate of the first half flipped in frequency. Looking at the data, we see that we can uniquely represent the FFT using only  $N/2$  complex numbers. These are packed into the output array in alternating real and imaginary components:

$$X = \{ \text{real}[0], \text{imag}[0], \text{real}[1], \text{imag}[1], \text{real}[2], \text{imag}[2] \dots \text{real}[(N/2)-1], \text{imag}[(N/2)-1] \}$$

It happens that the first complex number ( $\text{real}[0], \text{imag}[0]$ ) is actually all real.  $\text{real}[0]$  represents the DC offset, and  $\text{imag}[0]$  should be 0. ( $\text{real}[1], \text{imag}[1]$ ) is the fundamental frequency, ( $\text{real}[2], \text{imag}[2]$ ) is the first harmonic and so on.

The real FFT functions pack the frequency domain data in this fashion. The forward transform outputs the data in this form and the inverse transform expects input data in this form. The function always performs the needed bitreversal so that the input and output data is always in normal order. The functions support lengths of [32, 64, 128, ..., 4096] samples.

The function uses the standard FFT definition and output values may grow by a factor of  $\text{fftLen}$  when computing the forward transform. The inverse transform includes a scale of  $1/\text{fftLen}$  as part of the calculation and this matches the textbook definition of the inverse FFT.

**Instance structure** for the floating-point RFFT/RIFFT function

```
arm_rfft_fast_instance_f32 rFFT_S;
```

**Init Function**

There is an associated initialization function for each data type. The parameter *fftLen* specifies the length of RFFT/CIFFT process. Supported FFT Lengths are 32, 64, 128, 256, 512, 1024, 2048, 4096. This Function also initializes Twiddle factor table pointer and Bit reversal table pointer.

```
arm_status arm_rfft_fast_init_f32 (    arm_rfft_fast_instance_f32 *    S,
                                     uint16_t                        fftLen
                                     )
```

**Parameters**

[in,out] **S** points to an [arm\\_rfft\\_fast\\_instance\\_f32](#) structure  
 [in] **fftLen** length of the Real Sequence

**RFFT Function**

```
void arm_rfft_fast_f32 (    const arm_rfft_fast_instance_f32 *    S,
                           float32_t *                            p,
                           float32_t *                            pOut,
                           uint8_t                                ifftFlag
                           )
```

**Parameters**

[in] **S** points to an [arm\\_rfft\\_fast\\_instance\\_f32](#) structure  
 [in] **p** points to input buffer (Source buffer is modified by this function.)  
 [in] **pOut** points to output buffer  
 [in] **ifftFlag**

- value = 0: RFFT
- value = 1: RIFFT

**Complex Magnitude Function**

Computes the magnitude of the elements of a complex data vector. The *pSrc* points to the source data and *pDst* points to the where the result should be written. *numSamples* specifies the number of complex samples in the input array and the data is stored in an interleaved fashion (*real, imag, real, imag, ...*). The input array has a total of  $2 \cdot \text{numSamples}$  values; the output array has a total of *numSamples* values.

The underlying algorithm is used:

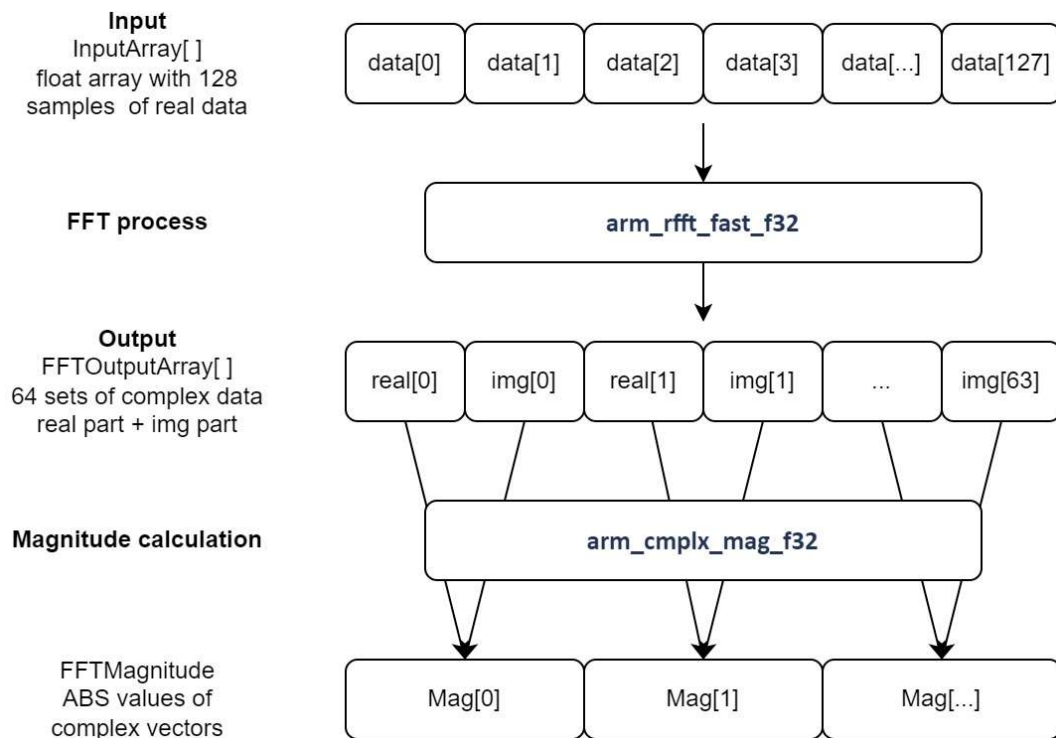
```
for (n = 0; n < numSamples; n++)
{
    pDst[n] = sqrt(pSrc[(2*n)+0]^2 + pSrc[(2*n)+1]^2);
}
```

```
void arm_cmplx_mag_f32 (    const float32_t *    pSrc,
                           float32_t *          pDst,
                           uint32_t              numSamples
                           )
```

**Parameters**

[in] **pSrc** points to input vector  
 [out] **pDst** points to output vector  
 [in] **numSamples** number of samples in each vector

Realųjų duomenų GFT skaičiavimo eiga parodyta 9 pav. struktūrinėje schemeje:



9 pav. Realųjų duomenų GFT eiga

GFT skaičiavimo funkcija

```

void FFT_calculation(void)
{
    float32_t FFTMaxValue, threshold;
    uint32_t MaxIndex;

    //rFFT init. Supported FFT Lengths are 32, 64, 128, 256, 512, 1024, 2048, 4096.
    arm_rfft_fast_init_f32(&rFFT_S, SAMPLES_QTY);

    //rFFT process. 0 = RFFT, 1 = RIFFT
    arm_rfft_fast_f32(&rFFT_S, InputArray, FFTOutputArray, 0);           //direct FFT
    FFTOutputArray[0] = 0;                                              //DC

    //Calculate the Magnitude
    arm_cmplx_mag_f32(FFTOutputArray, FFTMagnitude, SAMPLES_QTY/2);

    //Phase calculation
    for(int i=0; i<SAMPLES_QTY/2; i++)
    {
        FFTRePart[i] = FFTOutputArray[2*i];
        FFTImPart[i] = FFTOutputArray[(2*i)+1];
        FFTPhase[i] = (atan(FFTImPart[i]/FFTRePart[i]))*180/PI;
    }

    //Find max value
    arm_max_f32(&FFTMagnitude, SAMPLES_QTY/2, &FFTMaxValue, &MaxIndex);
}
    
```

```

//Mask out small values
threshold = FFTMaxValue / 1000;
for(int i=0; i<SAMPLES_QTY/2; i++)
{
    if(fabs(FFTImPart[i])<threshold) FFTImPart[i] = 0;
}

//Phase calculation
for(int i=0; i<SAMPLES_QTY/2; i++)
{
    FFTPhase_masked[i] = (atan(FFTImPart[i]/FFTRePart[i]))*180/PI;
}
}

```

## Programos rezultatų pavyzdys

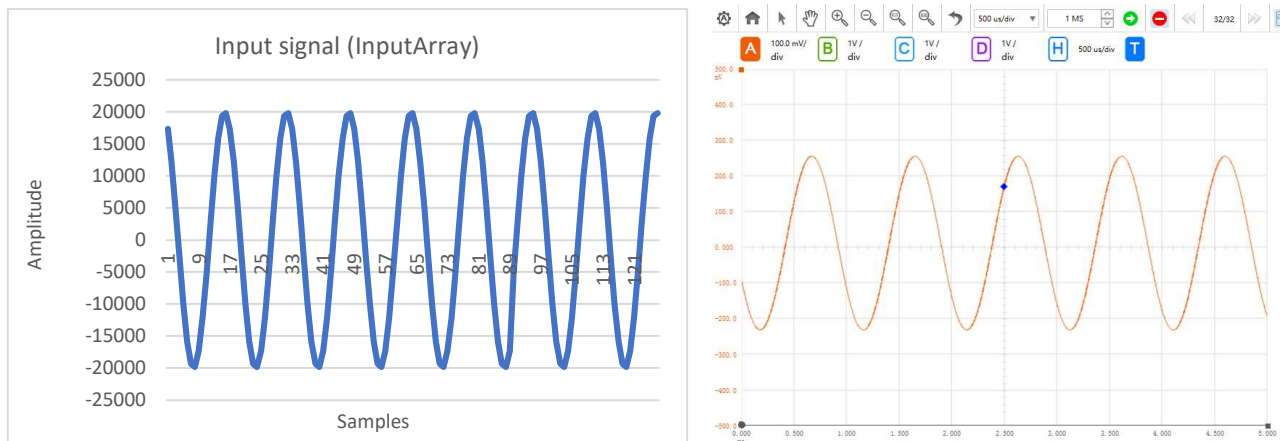
Funkcijos testavimui sugeneruotas 1kHz dažnio signalas su  $\pi/6$  fazės poslinkiu.

```

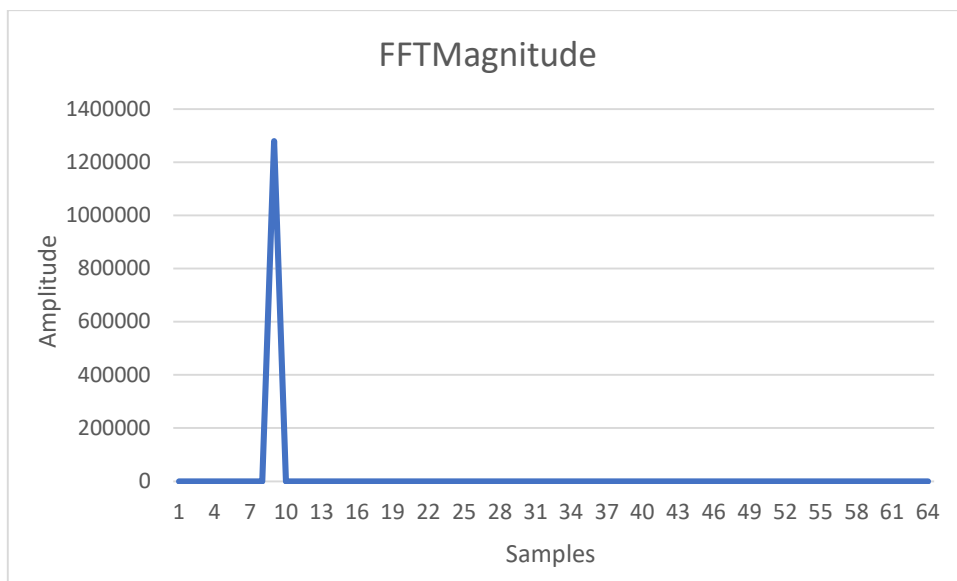
//1 kHz cos + PI/6 shift
InputArray[index] = ( 20000*cos((FREQUENCY/SAMPLING_FREQ)*2*PI*index + PI/6) );

```

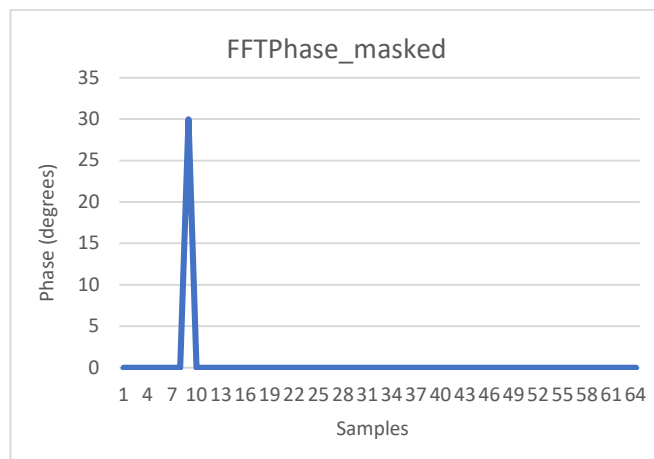
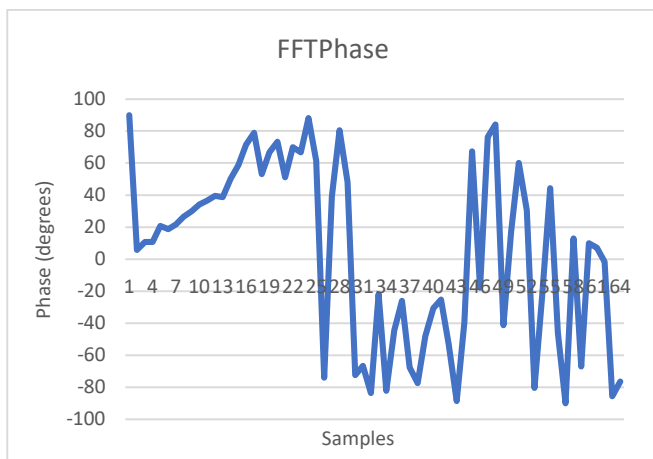
10-12 pav. parodytas signalas laiko bei dažnių srityje.



10 pav. 1 kHz dažnio ko-sinusinis signalas su  $\pi/6$  fazės poslinkiu



11 pav. Magnitudė

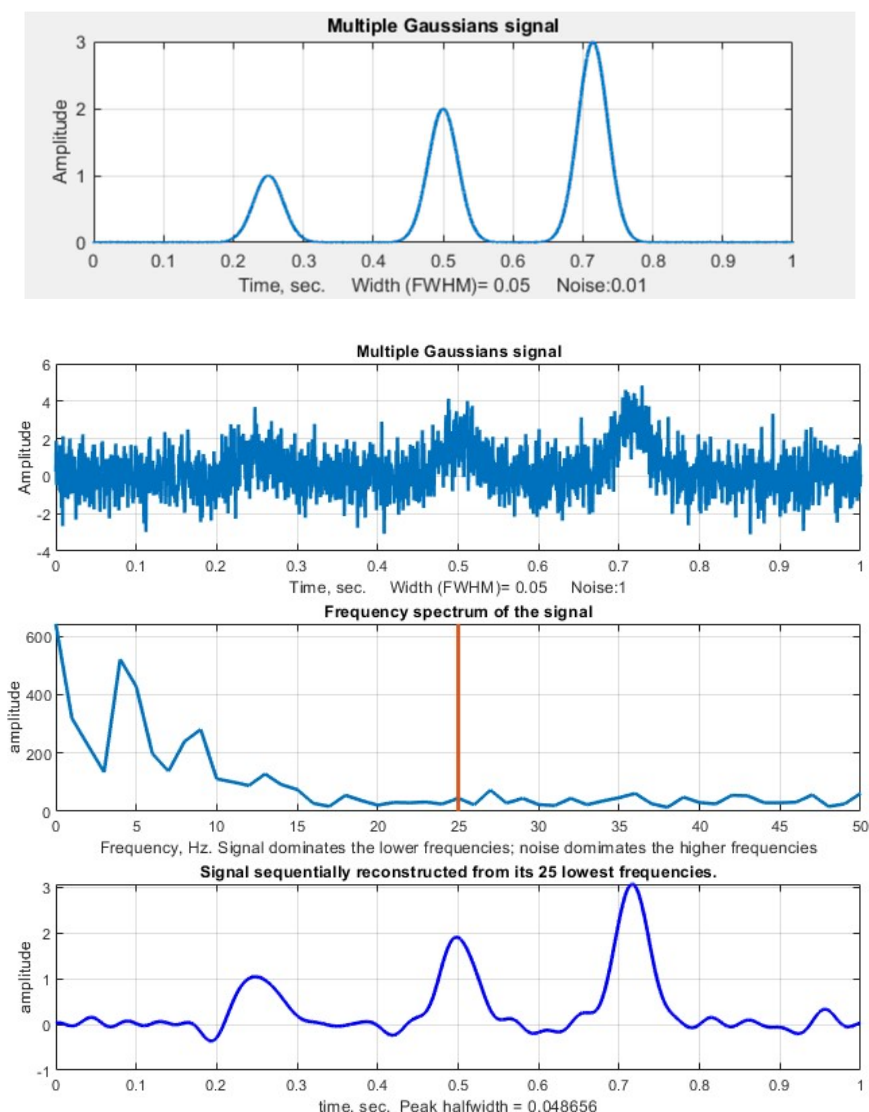


12 pav. Fazė: a) be maskavimo; b) su mažų verčių maskavimu

## Filtravimas su GFT

Furjė filtras - tai filtravimo funkcija, pagrįsta tam tikrų signalo dažnių dedamųjų manipuliavimu. Jis įgyvendinamas pirmiausia atliekant signalo Furjė transformaciją, tada susilpninant arba sustiprinant tam tikrus dažnius ir galiausiai atliekama atvirkštinė Furjė transformacija.

Daugelyje mokslinių matavimų, pavyzdžiui, spektroskopijoje ir chromatografijoje, signalai yra palyginti tolygių formų, kurias galima atvaizduoti stebėtinai mažu Furjė komponentų skaičiumi.



13 pav. Signalų pavyzdys su pridėtu baltu triukšmu<sup>2</sup>

Pateiktame paveikslėlyje pavaizduotas signalas su trimis pikais ir pridėtu atsitiktiniu baltuoju triukšmu, kurio standartinis nuokrypis lygus mažiausios viršūnės aukščiui. Baltojo triukšmo amplitudė visuose dažniuose yra vienoda, o didžioji dalis signalo sutelkta pirmuosiuose 25 Hz, todėl galima tikėtis, kad didžiąją dalį (bet ne visą) triukšmo galima pašalinti naudojant žemųjų dažnių filtrą, kurio atkirtos dažnis yra apie 25 Hz. Rezultate (apatinėje pav. dalyje) matyti trys signalo viršūnės, kurių padėtis ir forma atitinka pirminį signalą. Akivaizdu, kad triukšmas žemiau 25 Hz lieka, todėl signalas yra netobulas.

<sup>2</sup> <https://terpconnect.umd.edu/~toh/spectrum/FourierFilter.html>

Funkcija filtravimui su GFT:

```
void FFT_filtering(void)
{
    //rFFT init. Supported FFT Lengths are 32, 64, 128, 256, 512, 1024, 2048, 4096.
    arm_rfft_fast_init_f32(&rFFT_S, SAMPLES_QTY);

    //rFFT process. 0 = RFFT, 1 = RIFFT
    arm_rfft_fast_f32(&rFFT_S, InputArray, FFTOutputArray, 0);           //direct FFT

    //Remove some frequencies
    for(int i=20; i<SAMPLES_QTY; i++)
        FFTOutputArray[i] = 0;

    //rFFT process. 0 = RFFT, 1 = RIFFT
    arm_rfft_fast_f32(&rFFT_S, FFTOutputArray, OutputSignalArray, 1); //inverse FFT
}
```

## Praktinė dalis

1. Sugeneruoti poli-harmoninį signalą. Modifikuoti FFT filtravimo funkciją taip, kad:
  - a. Būtų filtruojamas vienas pasirinkto dažnio signalas.
  - b. Būtų filtruojami visi skirtingų dažnių signalai, paliekant didžiausios amplitudės signalą
2. Išmatuoti FFT filtro greitaveiką esant skirtingiems signalo masyvo dydžiams bei skirtingiems kompiliatoriaus optimizavimo lygiams.
3. Sugeneruoti kintančio dažnio (*Chirp*) signalą bei apskaičiuoti jo dažnių spektrą.