Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Kosdi Dávid

Szinuszbecslő eljárások érzékenységének vizsgálata a mintavételi időpont bizonytalanságára

KONZULENS

Dr. Pálfi Vilmos

BUDAPEST, 2018

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Kosdi Dávid**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2018. 12. 16.

...…….
Kosdi Dávid

Összefoglaló

A méréstechnikában fontos szerepet tölt be a szinuszjelek paramétereinek becslése. Zajjal terhelt környezetben, digitális műszerekkel végzett mérés esetén elengedhetetlen az adatok előfeldolgozása. Az előfeldolgozás végezhető például legkisebb négyzetek módszerét alkalmazó becslő algoritmusokkal.

A zajjal terhelt bemeneti szinuszos jel becsült paramétereit felhasználhatjuk például egy háromfázisú hálózat minőségi jellemzésére, analóg-digitális átalakító minősítésére, rendszeridentifikációra, vagy impedancia mérésre. Dolgozatom célja a legkisebb négyzetek módszerét alkalmazó Least Squares és Total Least Squares algoritmusok érzékenységének vizsgálata a mintavételezés bizonytalanságának függvényében. Mintavételi bizonytalanság esetén előfordulhat, hogy a digitális mintákhoz nem megfelelő időbélyeget rendelünk, ami hatással van a becslési algoritmusok statisztikai tulajdonságaira. Least Squares algoritmus használata esetén a minták időbeni eltolódásának figyelembevételére nincs lehetőség, míg a Total Least Squares algoritmus tekintettel van a mintavételezés időpontjának hibáira is.

Az algoritmusok becslési hibáját szimulációs környezetben vizsgálom, majd az elmélet megvalósíthatóságának bizonyítékaként bemutatom az algoritmusok C nyelvű implementációit, amelyeket egy Cortex M4 alapú mikrovezérlő segítségével ellenőrzök.

Abstract

The estimation of the signal parameters of a sine wave has an important role in the measurement technology. While using the digital data processing in a noisy environment, the preprocess of the data is essential. Preprocessing can be done by using the least squares method.

The estimated parameters of the input sine wave can be used to qualify a three-phase system, or an analog-to-digital converter,or for system identification, or to measure impedance. My thesis is about examining the sensitivity of the Least Squares and the Total Least Squares methods, depending on the uncertainty of sampling. In the sampling uncertainty, we may assign a later timestamp to the digital samples that affect the statistical properties of the estimation algorithms. When using the Least Squares method, it is not possible to take care of the shift of the samples, while the Total Least Squares method has a solution to count with the errors in the sampling time.

I examine the estimation error of the algorithms in a simulation environment, and as a proof of the feasibility of the theory, I present a C language implementation, which is verified by running the code on a Cortex M4-based microcontroller.

1 Bevezetés

A méréstechnikában fontos szerepet tölt be a szinuszjelek paramétereinek becslése. A zajjal terhelt bemeneti szinuszos jel becsült paramétereit felhasználhatjuk például egy háromfázisú hálózat minőségi jellemzésére, analóg-digitális átalakító minősítésére, rendszeridentifikációra, vagy impedancia mérésre.

A becslési algoritmusok vizsgálatát számos aspektusból lehet végezni, például konvergencia, statisztikai tulajdonságok vagy számításigény. Dolgozatomban két, legkisebb négyzetek módszerét alkalmazó becslési algoritmust statisztikai tulajdonságainak vizsgálatát végzem el különböző hibaforrások mellett. A vizsgálatot zajjal terhelt szinuszos jelekkel végzem és kiemelt figyelmet fordítok a mintavételezés bizonytalanságából eredő torzító hatásra.

Az algoritmusok vizsgálata előtt meg kell határozni egy, a becsléshez alkalmas jelmodellt. A következő jelleírásban szereplő paraméterek között A jelöli a jel csúcsértékét, f a frekvenciáját, ϕ_0 a kezdőfázisát, C pedig az eltolását, ez utóbbi nevezhető egyenáramú jelszintnek is.

$$y(t) = A * \cos(2\pi * f * t + \varphi_0) + C \tag{1}$$

Az (1)-es kifejezésben szereplő jelleírás problémája, hogy két paraméterben is nemlineáris. A becslések elvégzéséhez a kifejezést át kell alakítani, hogy csökkenjen a nemlineáris paraméterek száma. Ezt a következő jelleírás alkalmazásával érhetjük el:

$$y(t) = A * \cos(2\pi * f * t) + B * \sin(2\pi * f * t) + C \tag{2}$$

Ebben a felírásban már csak egyetlen nemlineáris tag szerepel, a jel frekvenciája. Az A és B paraméterek jelölik a koszinusz, illetve szinusz összetevők amplitúdóit, így ezekből az arcus tangens függvény segítségével a kezdőfázis kiszámolható.

A legkisebb négyzetek módszerét alkalmazó becslők a becsült paraméterekből előállított y, és a mintavételezett, zajjal terhelt bemeneti jel mintáit tartalmazó x vektor elemei közötti hiba minimalizálására törekednek. A becslés költségfüggvényének értelmezéséhez fel kell írjuk a minimalizálni kívánt hiba függvényét, ahol e jelöli a mintánként számolt hiba értékét:

$$\mathbf{e} = \mathbf{x} - A * \cos(2\pi * f * \mathbf{t}) + B * \sin(2\pi * f * \mathbf{t}) + C \tag{3}$$

A költségfüggvény felírásához a (3)-as egyenletet négyzetre kell emelni, és az így kapott kifejezést kell minimalizálni:

$$\sum_{n=1}^{M} [\, x_n - A * \cos(2 * \pi * f_0 * t_n) - B * \sin(2 * \pi * f_0 * t_n) - C]^2, \tag{4}$$

ahol $\mathbf{x_n}$ a bemeneti mintavételezett jel n-edik mintája, f0 a becsült frekvencia, tn pedig az n-edik minta mintavételezési időpontja. A költségfüggvény alapján felírható a rendszermátrix (D), amely a becslési paraméterek együtthatóit tartalmazza. Először a három paraméteres least squares becslőnél alkalmazott rendszermátrixot mutatom be, ami a négy paraméteres becslés egy speciális esete. Az ismertetés sorrendjét, a becslők alkalmazásának sorrendje indokolja. A négy paraméteres becslés elvégéséhez szükség van egy három paraméteres becslés eredményére. Ezek alapján a három paraméteres becslésnél alkalmazott rendszermátrix:

$$\mathbf{D} = \begin{bmatrix} \cos(2\pi f_0 t_1) & \sin(2\pi f_0 t_1) & 1 \\ \cos(2\pi f_0 t_2) & \sin(2\pi f_0 t_2) & 1 \\ \cdots & \cdots & \cdots \\ \cos(2\pi f_0 t_n) & \sin(2\pi f_0 t_n) & 1 \end{bmatrix} \tag{5}$$

A három becsült paramétert egy vektorba szervezve kapjuk s0-t:

$$s_0 = \begin{bmatrix} a_0 \\ b_0 \\ c_0 \end{bmatrix} \tag{6}$$

A D rendszermátrix és s_0 paramétervektor segítségével felírható a következő egyenlet a mintákat tartalmazó vektorra rendezve:

$$x = \mathbf{D} * s_0 \tag{7}$$

A (7)-es egyenlet s_0 -ra történő megoldásával megkaphatóak a becslési paraméterek, amelyeket a négyparaméteres becslés rendszermátrixához felhasználhatunk. A négyparaméteres becslés rendszermátrixa a következőképpen alakul:

$$\mathbf{D} = \begin{bmatrix} \cos(2\pi f_0 t_1) & \sin(2\pi f_0 t_1) & 1 & -a_0 t_1 \sin(2\pi f t_1) + b_0 t_1 \cos(2\pi f_i t_1) \\ \cos(2\pi f_0 t_2) & \sin(2\pi f_0 t_2) & 1 & -a_0 t_2 \sin(2\pi f t_2) + b_0 t_2 \cos(2\pi f_i t_2) \\ \cdots & \cdots & \cdots & \cdots \\ \cos(2\pi f_0 t_n) & \sin(2\pi f_0 t_n) & 1 & -a_0 t_M \sin(2\pi f t_n) + b_0 t_M \cos(2\pi f_i t_n) \end{bmatrix}, \tag{8}$$

ahol a_0 és b_0 a háromparaméteres becslés paramétervektorának első két eleme. Négyparaméteres becslés esetén a becsült paramétereket tartalmazó vektor összeállítása a következő:

$$s_0 = \begin{bmatrix} a_0 \\ b_0 \\ c_0 \\ \Delta f_0 \end{bmatrix}, \tag{9}$$

ahol Δf_0 a jelfrekvencia becslőjének megváltozása.

A következő fejezetekben ismertetem a különböző módszereket a paramétervektor kiszámítására.

2 Három paraméteres Least Squares becslő

A három paraméteres becslők alkalmazása esetén a hasznos jel frekvenciája ismert kell legyen és nem szabad időben változnia. A valóságban ez a követelmény ritkán teljesíthető (a jelgenerátorok pontatlansága miatt a jel frekvenciája elhangolódhat, illetve nem állítható be pontosan), de rövidebb mérések esetén közelítőleg helytálló a feltételezés.

A (7)-es egyenletet átrendezésével kiszámítható a paraméterek értéke. Mivel a **D** rendszermátrix nem négyzetes, azaz az egyenlet túlhatározott, ezért azt csak legkisebb négyzetes értelemben tudjuk megoldani, amihez szükség van a rendszermátrix pszeudó-inverzére. A (7)-es egyenletet s_0-ra rendezve, D_0 mátrix pszeudó-inverzének számítása után a következő egyenletet kapjuk:

$$s_0 = \left(\mathbf{D}^T \mathbf{D}\right)^{-1}\left(\mathbf{D}^T x\right) \tag{10}$$

Az (10)-es egyenletet úgy is megkaphatjuk, hogy a költségfüggvény s_0 szerinti deriváltját egyenlővé tesszük nullával (ez lesz a szélsőérték, azaz az optimum helye). A becsült jel amplitúdója (A), fázisa (ϕ) és egyen komponense (c) a következők szerint alakul:

$$A = \sqrt{a_0^2 + b_0^2} \tag{11}$$

$$\varphi = \tan^{-1}\left(\frac{-b_0}{a_0}\right) \tag{12}$$

$$c = c_0, \tag{13}$$

ahol a_0, b_0 és c_0 az s_0 becslő elemei.

3 Négy paraméteres Least Squares becslő

Az eljárás szintén a legkisebb négyzetek módszerét alkalmazza, azonban ismert frekvencia hiányában, a mintavételezett jelből állapítja meg a becsült jel frekvenciáját, amelynek értékét iteráció segítségével pontosítja.

Első lépésként meg kell határozzuk a frekvenciabecslő kezdeti értékét. Erre azért van szükség, mert az eljárás csak egy viszonylag szűk tartományban konvergál a paraméterek helyes értékéhez. Ennek oka, hogy a költségfüggvény nemlineáris a frekvencia paraméterben, továbbá az additív zaj is okozhat lokális minimumokat. Minél pontosabb a frekvencia kezdeti becslője, annál nagyobb eséllyel találja meg az eljárás a globális minimum helyét. A frekvencia kezdeti becslését végezhetjük például Fourier-transzformációval, amelynek eredményvektorából a maximális értéket vesszük. Ez az eljárás azonban kis frekvenciafelbontás esetén önmagában nem ad elég pontos kezdeti frekvencia értéket a becslőnk számára. Ha a mintavételezés a bemeneti jel frekvenciájának nem egész számú többszörösével történik, az egyes frekvenciakomponensek nem kerülnek DFT (Discrete Fourier Trasform) rácsra és a kezdeti frekvenciabecslőnk pontatlan lesz, ami a költségfüggvény minimalizálását egy lokális minimumba vezeti, ugyanis a minimumhely keresés gradiens alapon történik. Erre a problémára megoldást nyújt a 4. fejezetben ismertetésre kerülő IpFFT.

Második lépésként alkalmazzuk a három paraméteres eljárást és kiszámoljuk a becslők kezdeti értékét. Az s_0 vektor ebben az esetben kiegészül a frekvenciára vonatkozó paraméterrel is. A módosítással D_0 mátrix és s_0 vektor elemei a következőképp alakulnak:

$$\mathbf{D} = \begin{bmatrix} \cos(2\pi f_0 t_1) & \sin(2\pi f_0 t_1) & 1 & -a_0 t_1 \sin(2\pi f t_1) + b_0 t_1 \cos(2\pi f_i t_1) \\ \cos(2\pi f_0 t_2) & \sin(2\pi f_0 t_2) & 1 & -a_0 t_2 \sin(2\pi f t_2) + b_0 t_2 \cos(2\pi f_i t_2) \\ \cdots & \cdots & \cdots & \cdots \\ \cos(2\pi f_0 t_n) & \sin(2\pi f_0 t_n) & 1 & -a_0 t_M \sin(2\pi f t_n) + b_0 t_M \cos(2\pi f_i t_n) \end{bmatrix}, \tag{14}$$

$$s_0 = \begin{bmatrix} a_0 \\ b_0 \\ c_0 \\ \Delta f_0 \end{bmatrix} \tag{15}$$

Az s_0 vektor becslőjét a három paraméteres eljárásnál ismertetett módon számoljuk:

$$s_0 = \left(D^T D\right)^{-1} \left(D^T x\right) \tag{16}$$

A következő lépésben korrigáljuk a kezdeti frekvenciabecslő értékét az s_0 negyedik paraméterének megfelelően:

$$f_i = f_{i-1} + \Delta f_{i-1}, \tag{17}$$

ahol Δf_{i-1} az s_0 becslő vektor negyedik eleme.

Ezeket a lépéseket kell ismételnünk az iteráció során. Az iterációs lépések számát a kívánt pontosság függvényében szabadon megválaszthatjuk. A szabvány [2] ajánlása szerint hat iterációs lépéssel kellően pontos eredmény érhető el.

Az iteráció befejeztével a becsült jel amplitúdója (A) és fázisa (φ) a következő összefüggésekkel számítható:

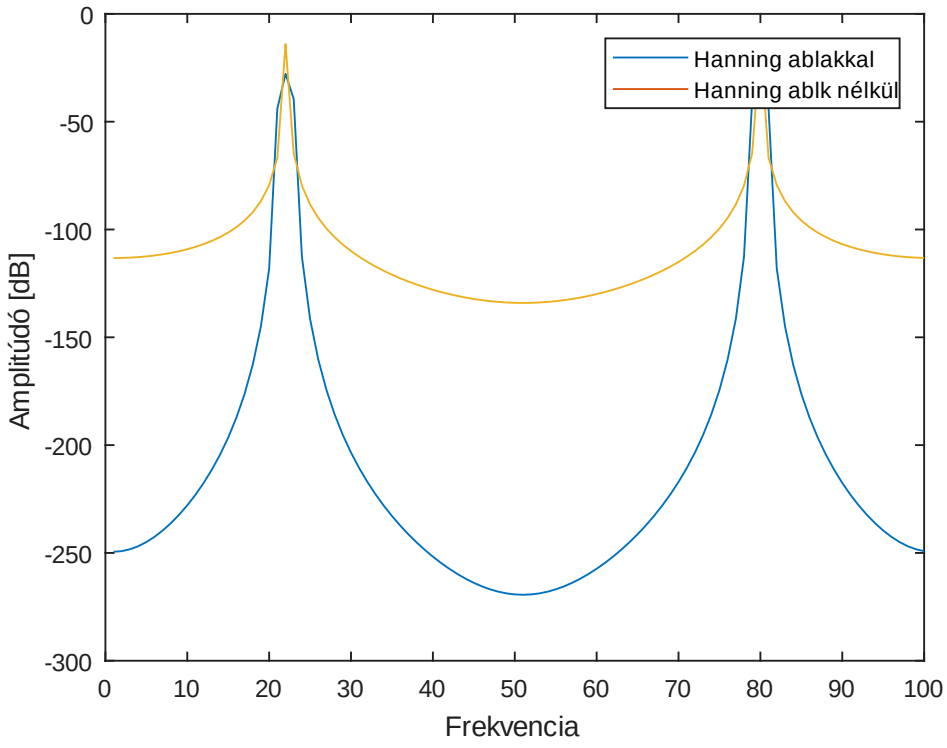
$$A = \sqrt{a_0^2 + b_0^2} \tag{18}$$

$$\varphi = \tan^{-1}\left(\frac{-b_0}{a_0}\right) \tag{19}$$

4 IpFFT használata

A négy paraméteres becslők kezdeti frekvenciájának megválasztását egyszerűen megtehetjük Fourier-transzformáció eredményvektorában végzett maximum hely kereséssel. Ez a módszer kellően pontos eredményt ad, ha a mintavételi frekvencia a forrás jel frekvenciájának egész számú többszöröse, azaz a vizsgálni kívánt jelet koherens mintavételi frekvenciával mintavételezzük. Ellenkező esetben a módszer nem ad kellően pontos kezdeti értéket, és a becslő a kezdeti hiba miatt a költségfüggvény egy lokális minimumába kerül [9]. A gradiens alapú minimum keresés miatt a becslés eredő hibája így sokkal nagyobb mértékű is lehet, mint a kezdeti frekvencia meghatározásának hibája.

A pontosabb kezdeti frekvenciaérték elérése érdekében interpolált FFT-t használunk, amelynek pontosságát ablakozó függvénnyel növeljük. Az ablakozó függvények közül a [2]-es forrás által említett Hanning ablakot használtam. A mintavételezett jel spektrumának alakulása Hanning ablak alkalmazásával a 1. ábra látható.



1. ábra - Mintavételezett jel spektruma Hanning ablakkal és nélküle

Az interpolációhoz a Hanning-ablakkal ellátott jel spektrumát fogjuk használni, amelyet xw vektor jelöl. Bevezetjük a k, iterációs feladatokat ellátó változót, amely jelöli a Hanning-ablakozott jel spektrumának maximumát.

$$[\mathrm{M}, \mathrm{I}] = \max(\mathrm{abs}(\mathrm{fft}(\mathrm{xw}))) \qquad \mathrm{k} = \mathrm{I} - 1, \tag{20}$$

ahol M és I két segédváltozó. M tartalmazza a maximum hely értékét, míg I a maximum hely indexét.

A spektrum abszolút értékének maximumához hozzá kell adnunk 1-et, hogy az algoritmus megfeleljen a Matlab által használt tömbindexelési konvenciónak. Külön figyelnünk kell arra, hogy a valós jelek Fourier-transzformáltjának komplex konjugált szimmetriája miatt az ablakozott jel spektruma abszolút értékének két maximuma van, amelyek közül nekünk csak az $f_s/2$ frekvenciaérték alatt lévőt kell figyelembe vennünk, ahol f_s a mintavételi frekvencia.

Bevezetjük az Y vektort, amely tartalmazza az ablakozott jel spektrumát és Δk korrekciós együtthatót, amely értékének kiszámításához feltételes elágazásra van szükségünk a következő módon:

ha Y(k+1) > Y(k-1):

$$\Delta \mathrm{k} = \frac{2 * \mathrm{Y}(\mathrm{k}+1) - \mathrm{Y}(\mathrm{k})}{\mathrm{Y}(\mathrm{k}+1) + \mathrm{Y}(\mathrm{k})} \tag{21}$$

ha Y(k+1) ≤ Y(k-1):

$$\Delta \mathrm{k} = \frac{\mathrm{Y}(\mathrm{k}) - 2 * \mathrm{Y}(\mathrm{k}-1)}{\mathrm{Y}(\mathrm{k}) + \mathrm{Y}(\mathrm{k}-1)} \tag{22}$$

Az interpolált frekvencia (f) és amplitúdó (A) értéke a következő egyenletekből adódik:

$$\mathrm{f} = (\mathrm{k} + \Delta \mathrm{k}) * \Delta \mathrm{f} \tag{23}$$

$$\mathrm{A} = \frac{\pi * \Delta \mathrm{k} * \mathrm{Y}(\mathrm{k})}{\sin(\pi * \Delta \mathrm{k})} \tag{24}$$

5 A LS becslő statisztikai tulajdonságai

A LS becslő paramétereinek varianciáját a bemeneti jel paramétereinek statisztikai tulajdonságaival alulról becsülhetjük. Alsó korlátot használva zárt alakban felírható a becslők varianciája az [5]-ös és [8]-as forrás alapján:

$$\mathrm{Var}(\hat{a}) \geq \mathrm{CRB}(a) \cong \frac{2\sigma^2}{N} \tag{25}$$

$$\mathrm{Var}(\hat{b}) \geq \mathrm{CRB}(b) \cong \frac{2\sigma^2}{N} \tag{26}$$

$$\mathrm{Var}(\hat{c}) \geq \mathrm{CRB}(c) \cong \frac{\sigma^2}{N} \tag{27}$$

A σ paraméter a bemeneten megjelenő, Gauss eloszlású zaj szórását, N pedig a minták hosszát jelöli.

Ugyan így a négy paraméteres LS becslő esetén is felírható a paraméterek varianciája az [5]-ös forrás alapján:

$$\mathrm{Var}(\hat{a}) \geq \mathrm{CRB}(a) \cong \frac{2\sigma^2}{N}\left(1 + \frac{3a^2}{A^2}\right) \tag{28}$$

$$\mathrm{Var}(\hat{b}) \geq \mathrm{CRB}(b) \cong \frac{2\sigma^2}{N}\left(1 + \frac{3a^2}{A^2}\right) \tag{29}$$

$$\mathrm{Var}(\hat{c}) \geq \mathrm{CRB}(c) \cong \frac{\sigma^2}{N} \tag{30}$$

$$\mathrm{Var}(\hat{\omega}) \geq \mathrm{CRB}(\omega) \cong \frac{24\sigma^2}{A^2N^3} \tag{31}$$

Itt A a bemeneti jel amplitúdója, ami a és b paraméterekből a következőképp áll elő:

$$A = \sqrt{a^2 + b^2} \tag{32}$$

Itt az a paraméter a jel koszinuszos összetevője, míg b paraméter a jel szinuszos összetevője. Mivel a szimuláció során a jel amplitúdója egységnyi volt, ezek a komponensek a fázisszögből számíthatók:

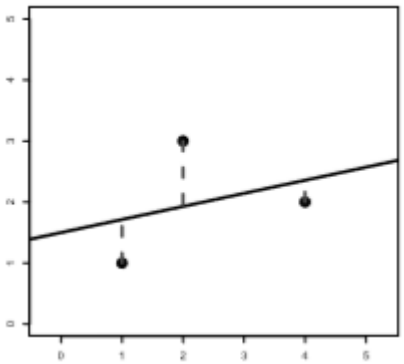
$$a = \cos(\phi) \tag{33}$$

$$b = \sin(\phi) \tag{34}$$

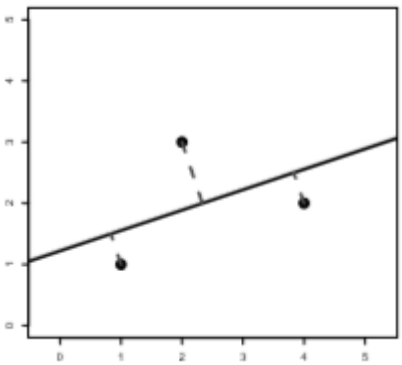
6 Négy paraméteres Total Least Squares becslő (TLS)

A Total Least Squares az eddigiekhez képest egy új megközelítést használ, amely görbeillesztés során nem csak az amplitúdó értékét, de az a mintavételi időpont bizonytalanságát is figyelembe veszi.

A 2. ábra és 3. ábra bemutatja az LS és TLS algoritmusok közötti különbséget. Az ábrákon egy egyenes illesztés példáját láthatjuk. Mind a LS és a TLS algoritmusok a hiba minimalizálására törekszik, azonban míg a LS algoritmus esetén a hibát a becsült egyenes és a mintavételi pontok közötti amplitúdó különbség alkotja, addig a TLS merőlegest állít a mintavételi pontokból a becsült egyenesre, amely merőleges hossza jelöli a becslési hibát. LS becslő esetén a becsült pont és a mintavételi pont időben azonos helyen van, míg a TLS becslés képes figyelembe venni a mintavételi időpont bizonytalanságát is.



2. ábra - LS algoritmus alkalmazása egyenes illesztésére



3. ábra - TLS algoritmus alkalmazása egyenes illesztésre

A TLS becslés szinguláris értékek szerinti felbontással kezdődik. A paraméterek becsült értékét a szinguláris felbontás eredményéből számítjuk. $\mathbf{U}, \mathbf{S}, \mathbf{V}$ mátrixok közül a $\mathbf{V}$ mátrix utolsó oszlopa tartalmazza a keresett paramétereket.

Az egyenes illesztés példáján bemutatott eljárás szinuszjelek paramétereinek becslésére is alkalmazható a [3]-as forrás leírása alapján. A (7)-es egyenletet TLS értelemben a következő módon írhatjuk fel:

$$x + \tilde{x} = \left[\mathbf{D} + \tilde{\mathbf{D}}\right] * s_0, \tag{35}$$

ahol x a zajmentes jel, $\tilde{x}$ a hasznos jelhez adott zaj, D a rendszermátrix, $\tilde{D}$ a mintavételezés hibájából eredő hiba a rendszermátrixban, s_0 vektor pedig a TLS becslés paramétereit tartalmazza, azaz a becslés megoldását.

Célunk, a négyzetes hiba minimalizálása. Ebben az esetben a hiba két komponensből áll. Írjunk fel egy hibamátrixot a rendszermátrix hibájából és a jelhez adódott zajból a következő módon:

$$\left[\tilde{\mathbf{D}}\ \tilde{x}\right] \tag{36}$$

Ezek után a megoldást a hibamátrix Frobenius normájának minimalizálásával keressük:

$$\min\left\|\left[\tilde{\mathbf{D}}\ \tilde{x}\right]\right\|_F^2 \tag{37}$$

Rendezzük a (35)-ös egyenletet 0-ra, így a következő egyenletet kapjuk:

$$\left[D + \tilde{D} \quad x + \tilde{x}\right]\begin{bmatrix} s_0 \\ -1 \end{bmatrix} = 0_{m\,x\,1} \tag{38}$$

Vezessük be Z mátrixot a jobb érthetőség és egyszerűbb leírás érdekében a következő módon:

$$Z = [D \quad x] \tag{39}$$

Keressük tehát azt a minimális hibamátrixot amelyre a (38)-as egyenlet egyértelműen megoldható s_0-ra. Vizsgáljuk meg az egyenletben szereplő mátrixok méretét. A rendszermátrix n*m méretű, ahol n a becslési paraméterek száma, m pedig a minták számát jelöli, amely minták az x vektorban szerepelnek, tehát x vektor egy m hosszúságú vektor. A Z mátrix ezek alapján egy n+1 dimenziós teret feszít ki, feltételezve természetesen, hogy a rendszermátrix lineárisan független vektorokból áll és a mintákat tartalmazó vektor is lineárisan független a rendszermátrixot alkotó összes vektorral. Az n hosszúságú α vektor csak akkor határozható meg egyértelműen, ha a Z mátrix dimenziója n. A hibamátrixnak tehát csökkentenie kell az n+1 dimenziós Z mátrix méretét. Használnunk kell az Eckart-Young tételt, ami szerint a legkisebb sajátérték elhagyásával érhetjük el a dimenziócsökkenést a legkevesebb információveszteséggel. Ehhez fel kell

bontanunk a mátrixot az SVD felbontás szerint. Az egyesitett mátrix a következő alakban irható fel:

$$[Z] = [U_p \quad u_q] \begin{bmatrix} \Sigma_p & \\ & \sigma_q \end{bmatrix} \begin{bmatrix} V_{pp} & v_{pq} \\ v_{qp} & v_{qq} \end{bmatrix}^T, \tag{40}$$

Itt Σ_p mátrix tartalmazza az n darab legnagyobb sajátértéket, míg σ_q a lekisebb sajátérték. Írjuk fel a (40)-es egyenletet, majd szorozzuk be mindkét oldalt V-vel és fejtsük ki a szorzatokat a mátrixok utolsó oszlopa szerint, vagyis az utolsó oszlopon kívüli elemeket tekintsük 0-nak:

$$[Z] \begin{bmatrix} V_{pp} & v_{pq} \\ v_{qp} & v_{qq} \end{bmatrix} = [U_p \quad u_q] \begin{bmatrix} \Sigma_p & \\ & \sigma_q \end{bmatrix} \begin{bmatrix} V_{pp} & v_{pq} \\ v_{qp} & v_{qq} \end{bmatrix}^T \begin{bmatrix} V_{pp} & v_{pq} \\ v_{qp} & v_{qq} \end{bmatrix} \tag{41}$$

$$[Z] \begin{bmatrix} V_{pp} & v_{pq} \\ v_{qp} & v_{qq} \end{bmatrix} = [U_p \quad u_q] \begin{bmatrix} \Sigma_p & \\ & \sigma_q \end{bmatrix} \begin{bmatrix} V_{pp} & v_{pq} \\ v_{qp} & v_{qq} \end{bmatrix}^T \tag{42}$$

$$U_p = 0 \qquad [Z] \begin{bmatrix} v_{pq} \\ v_{qq} \end{bmatrix} = u_q \sigma_q \tag{43}$$

Írjuk fel a zajjal terhelt Z mátrixot és töröljük a legkisebb sajátértékét:

$$[D + \tilde{D} \quad x + \tilde{x}] = [U_p \quad u_q] \begin{bmatrix} \Sigma_p & \\ & 0 \end{bmatrix} \begin{bmatrix} V_{pp} & v_{pq} \\ v_{qp} & v_{qq} \end{bmatrix}^T, \tag{44}$$

A keresett hibamátrix a következő módon fog alakulni:

$$[\tilde{D} \; \tilde{x}] = [D + \tilde{D} \quad x + \tilde{x}] - [D \quad x] \tag{45}$$

Helyettesítsük be a (45)-ös egyenletbe a (44)-es és (40)-es képletet:

$$[\tilde{D} \; \tilde{x}] = -[U_p \quad u_q] \begin{bmatrix} 0_{nxn} & \\ & \sigma_q \end{bmatrix} \begin{bmatrix} V_{pp} & v_{pq} \\ v_{qp} & v_{qq} \end{bmatrix}^T \tag{46}$$

Rendezés után a (46)-os képlet a következő alakra hozható:

$$[\tilde{D} \; \tilde{x}] = -u_q \sigma_q \begin{bmatrix} v_{pq} \\ v_{qq} \end{bmatrix}^T \tag{47}$$

Alkalmazzuk erre a (43)-as képletet:

$$[\tilde{D} \; \tilde{x}] = -[D \quad x] \begin{bmatrix} v_{pq} \\ v_{qq} \end{bmatrix} \begin{bmatrix} v_{pq} \\ v_{qq} \end{bmatrix}^T \tag{48}$$

A (45)-ös képlet rendezése után helyettesítsük be a (44)-es képletnél kapott összefüggést:

$$[D + \tilde{D} \quad x + \tilde{x}] = [D \quad x] - [D \quad x] \begin{bmatrix} v_{pq} \\ v_{qq} \end{bmatrix} \begin{bmatrix} v_{pq} \\ v_{qq} \end{bmatrix}^T \tag{49}$$

Szorozzuk be a (49)-es egyenlet mindkét oldalát a $\begin{bmatrix} V_{pq} \\ V_{qq} \end{bmatrix}$ vektorral. Ezzel a lépéssel a következő levezetéshez jutunk:

$$[D + \tilde{D} \quad x + \tilde{x}] \begin{bmatrix} V_{pq} \\ V_{qq} \end{bmatrix} = [D \quad x] \begin{bmatrix} V_{pq} \\ V_{qq} \end{bmatrix} - [D \quad x] \begin{bmatrix} V_{pq} \\ V_{qq} \end{bmatrix} \begin{bmatrix} V_{pq} \\ V_{qq} \end{bmatrix}^T \begin{bmatrix} V_{pq} \\ V_{qq} \end{bmatrix}$$

$$[D + \tilde{D} \quad x + \tilde{x}] \begin{bmatrix} V_{pq} \\ V_{qq} \end{bmatrix} = [D \quad x] \begin{bmatrix} V_{pq} \\ V_{qq} \end{bmatrix} - [D \quad x] \begin{bmatrix} V_{pq} \\ V_{qq} \end{bmatrix}$$

$$[D + \tilde{D} \quad x + \tilde{x}] \begin{bmatrix} V_{pq} \\ V_{qq} \end{bmatrix} = 0_{m\,x\,1} \tag{50}$$

A (50)-es egyenletet összevetve a (38)-as egyenlettel láthatjuk, hogy a lépések sorozata után a V mátrix elemeiből előáll s_0 a következő módon:

$$s_{0_{TLS}} = -(v_{pq} v_{qq})^{-1}, \tag{51}$$

ahol v_{pq} egy n elemű vektor a V mátrix n+1. oszlopából, v_{qq} pedig ugyan ennek az oszlopnak az utolsó eleme.

A becsült jel paraméterei a α vektorból a következők:

$$a_0 = s_0(2) \tag{52}$$

$$b_0 = s_0(1) \tag{53}$$

$$c_0 = s_0(3) \tag{54}$$

$$f = \frac{N * \omega}{2 * \pi}, \tag{55}$$

ahol $\omega = \cos^{-1}\left(\frac{b_0}{2}\right)$, N pedig a mintákat tartalmazó vektor hossza.

Láthatjuk, hogy a TLS algoritmus a rendszermátrix hibáját feltételezve képes a mintavételezési idő hibáját is figyelembe venni, tehát hogy a mintavételezés valójában nem teljesen egyenletesen történt. A becslési paraméterek egyértelmű kiszámításához szükséges dimenzió redukciót SVD felbontással, majd a legkisebb sajátérték elhagyásával éri el. A beágyazott rendszerekbe történő integrálását a szinguláris értékek szerinti felbontás lépése megnehezíti, ugyanis kiszámítása erőforrásigényes és nem készült még hatékony implementáció. A TLS becslő paramétereinek kiszámítása azonban felírható zárt alakban a [6]-os forrásban publikált megközelítéssel.

A (38)-as képletben látszik, hogy az [s_0 -1] vektor a $[D + \tilde{D} \quad x + \tilde{x}]$ mátrix egy sajátvektora, így $[D + \tilde{D} \quad x + \tilde{x}]^T [D + \tilde{D} \quad x + \tilde{x}]$ -nek is sajátvektora lesz. A (38)-as egyenletet ez alapján módosítva felírhatjuk a következőt:

$$[\mathrm{D} \quad \mathrm{x}]^{\mathrm{T}}[\mathrm{D} \quad \mathrm{x}]\begin{bmatrix} \mathrm{s}_0 \\ -1 \end{bmatrix} = 0_{\mathrm{m\,x\,1}} \tag{56}$$

Az (56)-os képletben az egyszerűbb leírás érdekében elhagytam a hibamátrix és hibavektor jelöléseit, így ettől a ponttól feltételezem, hogy **D** rendszermátrix és **x** vektor magába foglalja a hibát is. A műveleteket elvégezve a következő levezetésre jutunk:

$$[\mathrm{D} \quad \mathrm{x}]^{\mathrm{T}}[\mathrm{D} \quad \mathrm{x}]\begin{bmatrix} \mathrm{s}_0 \\ -1 \end{bmatrix} = \begin{bmatrix} \mathrm{D}^{\mathrm{T}}\mathrm{D} & \mathrm{D}^{\mathrm{T}}\mathrm{x} \\ \mathrm{x}^{\mathrm{T}}\mathrm{D} & \mathrm{x}^{\mathrm{T}}\mathrm{x} \end{bmatrix}\begin{bmatrix} \mathrm{s}_0 \\ -1 \end{bmatrix} = \sigma^2 I * \begin{bmatrix} \mathrm{s}_0 \\ -1 \end{bmatrix}, \tag{57}$$

ahol σ^2 a legkisebb sajátérték négyzetét jelöli. Mivel $\begin{bmatrix} \mathrm{s}_0 \\ -1 \end{bmatrix}$ vektor a $[\mathrm{D} \quad \mathrm{x}]^{\mathrm{T}}[\mathrm{D} \quad \mathrm{x}]$ mátrix sajátvektora, a kifejezés értéke nem változik, ha a mátrixot egy sajátértékével helyettesítjük. Az (57)-es kifejezésből a következő levezetésre jutunk, amit később rendezek s_0-ra:

$$\mathrm{D}^{\mathrm{T}}\mathrm{D}\mathrm{s}_0 - \mathrm{D}^{\mathrm{T}}\mathrm{x} = \sigma^2\mathrm{s}_0 \tag{58}$$

$$(\mathrm{D}^{\mathrm{T}}\mathrm{D} - \sigma^2\mathrm{I})\mathrm{s}_0 = \mathrm{D}^{\mathrm{T}}\mathrm{x} \tag{59}$$

$$\mathrm{s}_0 = (\mathrm{D}^{\mathrm{T}}\mathrm{D} - \sigma^2\mathrm{I})^{-1}\mathrm{D}^{\mathrm{T}}\mathrm{x} \tag{60}$$

A levezetés végén összefüggés adódik a becslőket tartalmazó vektorra, aminek köszönhetően a becslők értéke zárt alakban kiszámolható SVD felbontás elvégzése nélkül. A kihívást innentől már csak a sajátértékek meghatározása jelenti. A 10.3 fejezetben ez a kérdés részletesebb tárgyalásra kerül.

7 A TLS becslő egy módosított változata

A 6. fejezetben részletesen bemutatásra került a Total Least Squares algoritmus, amelyet a [3]-as és [7]-es forrásban megjelölt értekezés a rendszermátrix és a bemeneti mintákat tartalmazó vektor módosításával alkalmaz.

A módosított vektor és rendszermátrix a szomszédos bemeneti minták különbségeit tartalmazzák a következő módon:

$$x' = [x_3 - x_2 \quad \dots \quad x_n - x_{n-1}] \tag{61}$$

$$D' = \begin{bmatrix} x_2 - x_1 & \dots & x_{n-1} - x_{n-2} \\ x_0 - x_1 & \dots & x_{n-3} - x_{n-2} \end{bmatrix} \tag{62}$$

Trigonometrikus azonosságok segítségével a minták különbségeire az alábbi összefüggés írható fel:

$$x_{n+1} - x_n = \cos\Omega \cdot (x_n - x_{n-1}) + y_{n-2} - y_{n-1}\,, \tag{63}$$

ahol

$$\Omega = \frac{2\pi f_x}{f_s}. \tag{64}$$

Ekkor x' és D' mátrixokra a következő egyenlet írható fel:

$$D' \cdot \cos\Omega 1 = x'. \tag{65}$$

A (65)-ös egyenletet az előző fejezetekben ismertetett LS vagy TLS módszerrel megoldva, $\cos\Omega$ becsülhető, amiből meghatározható a jelfrekvencia. A jelfrekvencia ismeretében a többi paraméter becslése elvégezhető háromparaméteres LS vagy TLS becsléssel. A [3]-as forrásban publikált megoldás során a szerzők mindkét esetben TLS becslést alkalmaztak.

8 Szimulációk

Az algoritmusok Matlabban történő implementálása után elkészítettem egy teszt környezetet, ami biztosította az algoritmusok számára a mintavételezett jelet.

A mintavételezett jelet véletlenszerű periódusszámmal, kezdőfázissal, valamint hozzáadott zajjal generáltam. Ezen felül a mintavételi időpontokhoz szintén zajt generáltam a jel létrehozásakor, ezzel modellezve a mintavételi időpont bizonytalanságát. Minden, ebben a fejezetben közölt eredmény száz teszt futtatás átlagolásával született. Az eredmények reprodukálhatósága érdekében a véletlen szám generátornak a 0-ás seedet állítottam be, aminek köszönhetően minden teszt azonos pszeudorandom véletlen számsorozattal fut le. A bemeneti jel paramétereinek generálása a következő megkötésekkel történt:

- A periódusok száma (J paraméter) 20 és 30 közötti egyenletes eloszlású véletlen értéket vehet fel
- A jel kezdőfázisa 0 és 2π közötti értéket, szintén egyenletes eloszlással
- A mintákat tartalmazó tömb a szimuláció során 1000 hosszúságú volt
- A jel egyenáramú komponensének értékét, azaz C paraméter értékét 0-nak választottam

A zajmentes teszt során a korábban ismertetett A, B és C paraméterek, valamint J hibájának középértéke és szórása egyaránt nulla. Itt J a periódusok számát jelöli és a következő összefüggés alapján következik a frekvencia:

$$\mathrm{f} = \frac{J}{N}, \tag{66}$$

ahol N a minták számát jelöli.

8.1 Szimuláció gauss eloszlású mintavételi bizonytalansággal

			Középérték			
			A	B	C	J
3 paraméteres	LS	б=0.1	0.000059	-0.000135	0.000032	
		б=0.05	-0.000047	0.00005	-0.000019	
		б=0.01	0.000015	0.000002	-0.000001	
	TLS	б=0.1	0.000081	-0.000137	0.000032	
		б=0.05	-0.000047	0.000048	-0.000019	
		б=0.01	0.000015	0.000002	-0.000001	
4 paraméteres	LS	б=0.1	0.000059	0.000075	0.00003	0.000009
		б=0.05	-0.000043	0.000039	-0.000019	-0.000008
		б=0.01	0.000002	-0.000005	-0.000001	0.000004
	TLS	б=0.1	0.000059	0.000075	0.00003	-0.000036
		б=0.05	-0.000043	0.000039	-0.000019	-0.000041
		б=0.01	0.000002	-0.000005	-0.000001	0
Módosított TLS		б=0.1	1.448473	-0.252595	-0.046716	-4.098
		б=0.05	-0.100308	-0.009231	0.000538	-0.910924
		б=0.01	-0.063349	-0.030613	0.000179	-0.043767

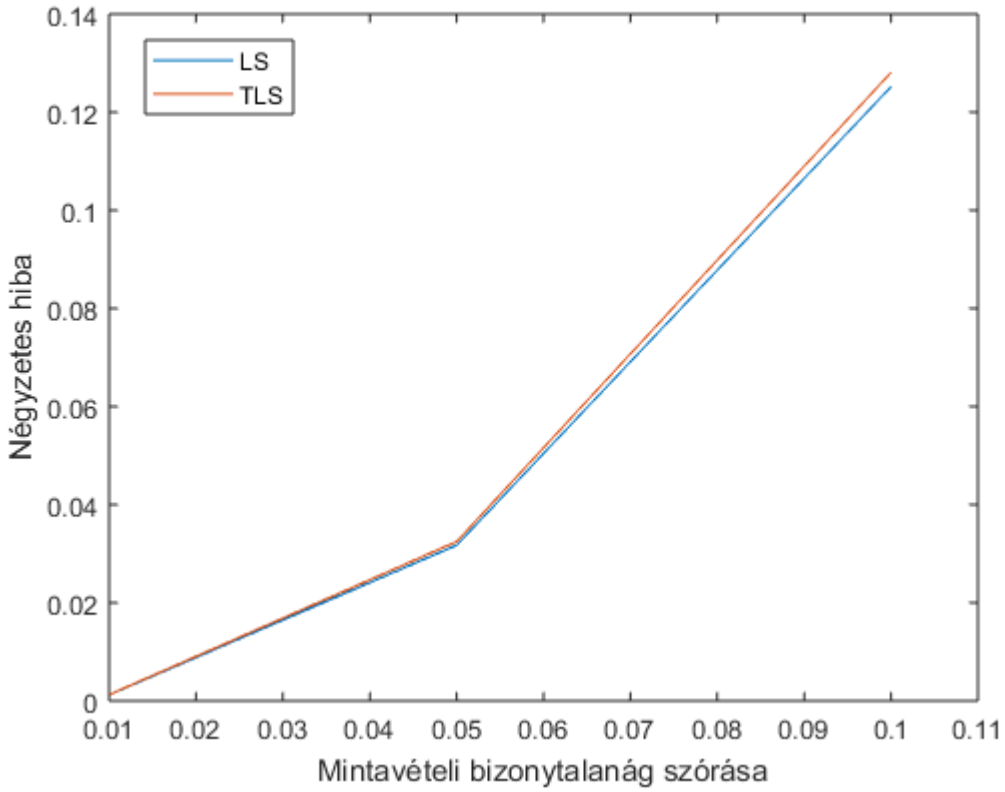
1. táblázat - A mintavételi bizonytalanság hatására bekövetkezett hiba középértéke (J=20±10)

			Szórás			
			A	B	C	J
3 paraméteres	LS	σ=0.1	0.001946	0.001797	0.00034	
		σ=0.05	0.000772	0.000961	0.000192	
		σ=0.01	0.00016	0.000177	0.000033	
	TLS	σ=0.1	0.00195	0.00179	0.00034	
		σ=0.05	0.000774	0.00096	0.000192	
		σ=0.01	0.00016	0.000177	0.000033	
4 paraméteres	LS	σ=0.1	0.000951	0.000924	0.000338	0.000349
		σ=0.05	0.000479	0.000501	0.000192	0.000187
		σ=0.01	0.000081	0.000094	0.000033	0.000034
	TLS	σ=0.1	0.000951	0.000924	0.000338	0.000825
		σ=0.05	0.000479	0.000501	0.000192	0.000379
		σ=0.01	0.000081	0.000094	0.000033	0.000074
Módosított TLS		σ=0.1	20.265187	19.793558	0.668208	4.047211
		σ=0.05	1.050579	1.034343	0.013609	0.711718
		σ=0.01	0.249858	0.233481	0.00206	0.104251

2. táblázat - A mintavételi bizonytalanság hatására bekövetkezett hiba szórása (J=20±10)

Az 1. táblázat összefoglalja a szimulációs eredményeket különböző mintavételi bizonytalansági mértéket használva. A perturbáció a generált mintákban a következő módon van jelen:

$$x(t) = A * \cos(\, 2\pi * f * (t + p) + \, \varphi_0 \,) + C, \tag{67}$$



4. ábra - Négyzetes hiba alakulása normális eloszlású mintavételi bizonytalanság hatására (J=20±10)

ahol p, a mintavételi bizonytalanság paramétere, egy normális eloszlású véletlen szám σ szórással.

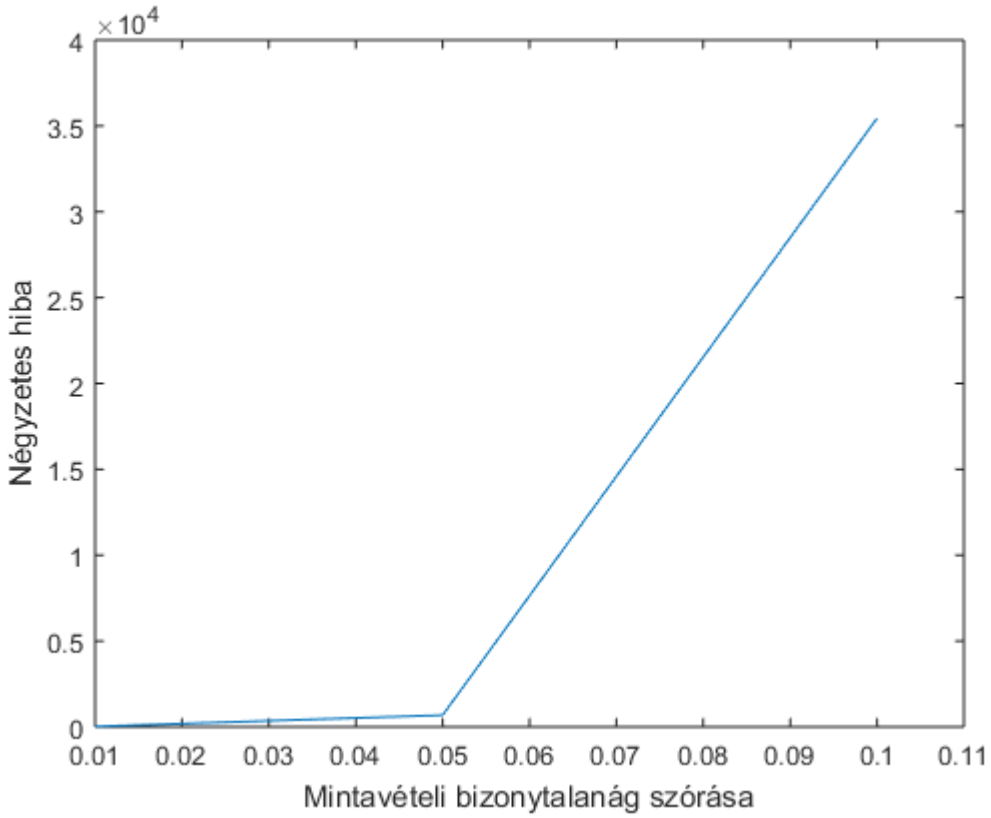
Az 4. ábrán a négyzetes hiba alakulása látható a mintavételi időpont bizonytalanságának mértékének függvényében. A négyzetes hiba a következő összefüggés alapján számítható:

$$\sum_{n=1}^{M} \left(\mathbf{x}(\mathbf{t_n}) - \left(\mathbf{a_0} * \cos\left(2 * \pi * \frac{J}{N} * \mathbf{t_n} \right) + \mathbf{b_0} * \sin\left(2 * \pi * \frac{J}{N} * \mathbf{t_n} \right) + \mathbf{c_0} \right) \right)^2 \tag{68}$$

Ez a legkisebb négyzetes költségfüggvény, amelyet az LS algoritmus minimalizál. Látható, hogy az LS becslő esetén az érték a TLS becslő értéke alatt marad, mivel TLS esetben nem ez a költségfüggvény kerül minimalizálásra.

A becslési paraméterekből előállított, kimeneti jelet y(t)-vel helyettesítve a (68)-as kifejezés a következő alakot ölti:

$$\sum_{t=1}^{M} (\,\mathbf{x(t)} - \mathbf{y(t)}\,)^2 \tag{69}$$



5. ábra – Módosított TLS algoritmus négyzetes hibájának alakulása normális eloszlású mintavételi bizonytalanság hatására (J=20±10)

Az 1. táblázatban található, paraméterekre bontott becslési hibákból is látszik, hogy a módosított TLS becslő hibája normális eloszlású perturbáció hatására sokkal nagyobb mértékű, mint a négy paraméteres LS vagy TLS esetben. A nagy mértékű paraméterhiba nagy mértékű négyzetes hibát von maga után, ezért a módosított TLS algoritmus négyzetes hibáját külön grafikonon kellett ábrázoljam, hogy ne vesszenek el az LS és TLS becslő részletei. A nagy mértékű hibát az okozza, hogy a módosított TLS algoritmus esetén a minták különbségét használjuk fel a számítás során. A jel mintái közötti eltérés alacsony frekvencia esetén túlságosan kicsi, ezért a rendszermátrix elemei nagyon érzékenyek lesznek a mintákon lévő zajokra. Ebből kifolyólag a szimulációk során célszerű a periódusok számát a mintaszám negyedének választani, mivel belátható, hogy ekkor lesz a legnagyobb a különbség két szomszédos minta által felvett értékben.

A szimulációt megismételtem 250±20 darab periódust használva. A periódusok számának eloszlása itt is egyenletes eloszlást követ a 230 – 270 intervallumon.

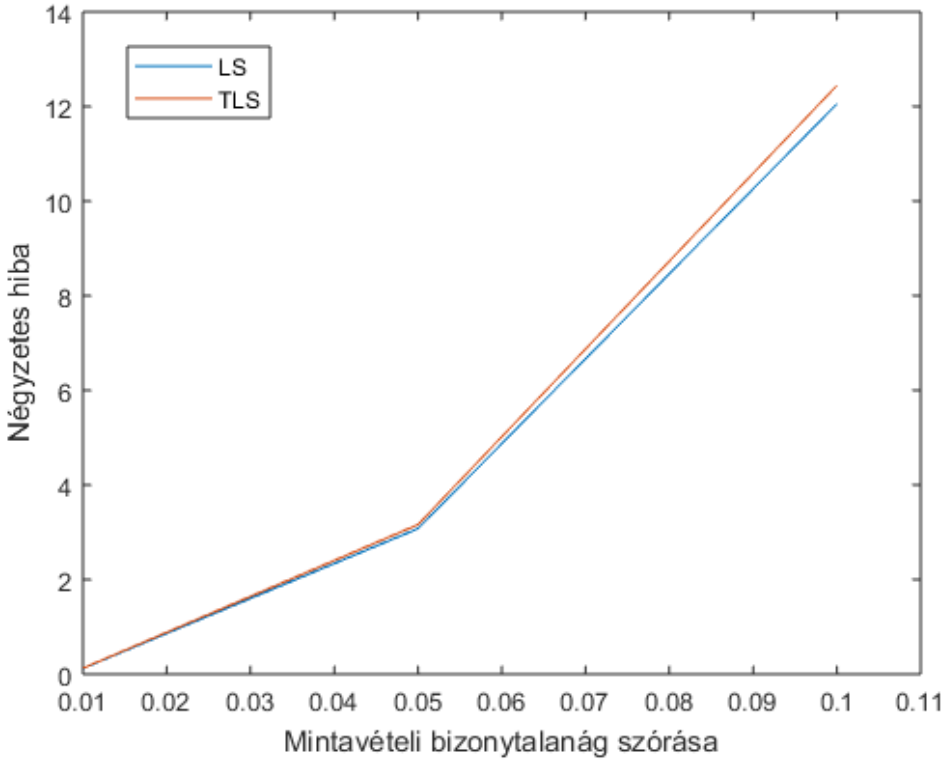
			Középérték			
			A	B	C	J
3 paraméteres	LS	σ=0.1	-0.002989	-0.002166	-0.000011	
		σ=0.05	-0.000524	-0.000872	0.000045	
		σ=0.01	0.000301	-0.000062	-0.000005	
	TLS	σ=0.1	-0.000816	-0.002599	-0.000012	
		σ=0.05	-0.000482	-0.000999	0.000044	
		σ=0.01	0.000309	-0.000074	-0.000005	
4 paraméteres	LS	σ=0.1	-0.002356	0.000217	-0.000013	-0.000212
		σ=0.05	-0.000189	0.000016	0.000044	-0.000005
		σ=0.01	0.000009	0.000035	-0.000005	0.000042
	TLS	σ=0.1	-0.002364	0.000213	-0.000013	-0.001309
		σ=0.05	-0.00019	0.000031	0.000044	-0.000473
		σ=0.01	0.000008	0.000035	-0.000005	-0.000032
Módosított TLS		σ=0.1	-0.099482	0.11176	0.000062	1.968157
		σ=0.05	0.038147	0.061981	0.000118	0.489261
		σ=0.01	0.006142	0.004281	-0.000008	0.019648

3. táblázat - A mintavételi bizonytalanság hatására bekövetkezett hiba középértéke (J=250±20)

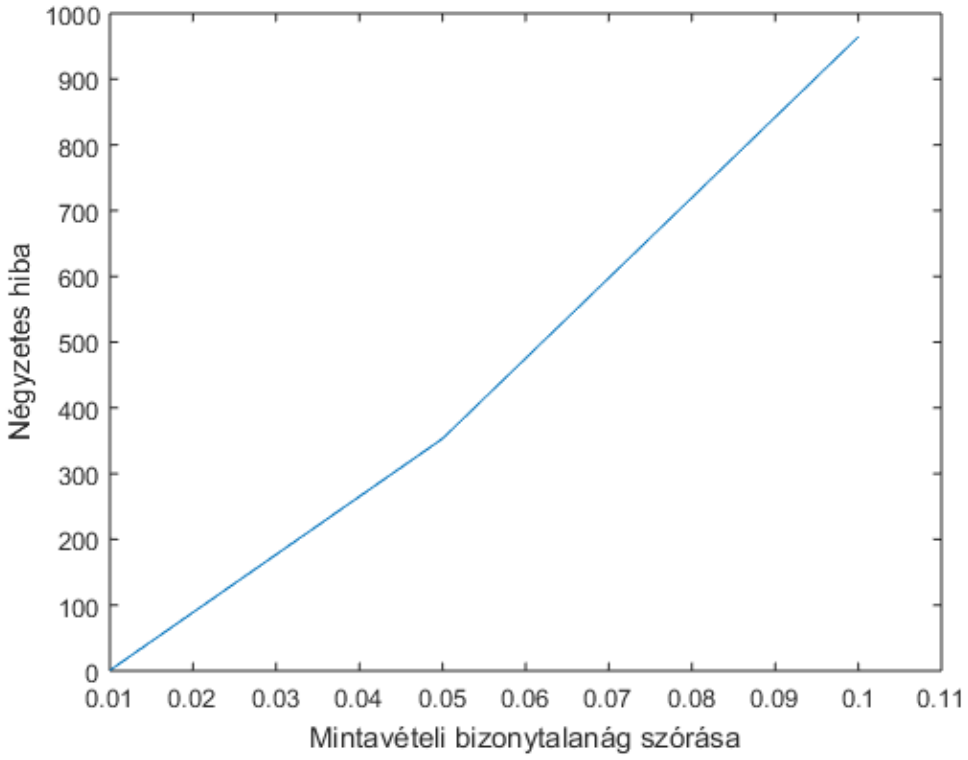
			Szórás			
			A	B	C	J
3 paraméteres	LS	σ=0.1	0.022739	0.021721	0.003743	
		σ=0.05	0.010167	0.009207	0.001615	
		σ=0.01	0.001938	0.002042	0.000348	
	TLS	σ=0.1	0.021248	0.019647	0.003768	
		σ=0.05	0.009965	0.008852	0.001618	
		σ=0.01	0.001934	0.00204	0.000348	
4 paraméteres	LS	σ=0.1	0.013618	0.012126	0.00374	0.003486
		σ=0.05	0.005728	0.005578	0.001616	0.001995
		σ=0.01	0.000873	0.000947	0.000348	0.000332
	TLS	σ=0.1	0.013829	0.012377	0.00374	0.008739
		σ=0.05	0.005745	0.005633	0.001616	0.004123
		σ=0.01	0.000875	0.000946	0.000348	0.000873
Módosított TLS		σ=0.1	1.050414	1.097196	0.006824	0.169971
		σ=0.05	1.047305	0.919942	0.002235	0.048227
		σ=0.01	0.040631	0.047441	0.000348	0.00334

4. táblázat - A mintavételi bizonytalanság hatására bekövetkezett hiba szórása (J=250±20)

Az 1-es és 3-as, valamint a 2-es és 4-es táblázatot összevetve látható, hogy a módosított TLS algoritmus hibája csökken, azonban még mindig lényegesen nagyobb az LS és TLS algoritmusok hibájánál. Sőt, a σ=0.1 esetben a becslés torzítottnak mondható, mivel a becslési hiba középértéke egy nagyságrenddel nagyobb a szórásnál. A táblázatokból az is látszik, hogy nagyobb frekvenciát, azaz több periódust használva az LS és TLS algoritmusok érzékenyebbé válnak a zajra.



6. ábra - Négyzetes hiba alakulása normális eloszlású mintavételi bizonytalanság hatására (J=250±20)



5. ábra – Módosított TLS algoritmus négyzetes hibájának alakulása normális eloszlású mintavételi bizonytalanság hatására (J=250±20)

Az algoritmusok korlátait ismerve célszerű a szimulációkat alacsony (20±10) és magasabb (250±20) periódusszámmal is elvégezni.

8.2 Szimuláció exponenciális eloszlású mintavételi bizonytalansággal

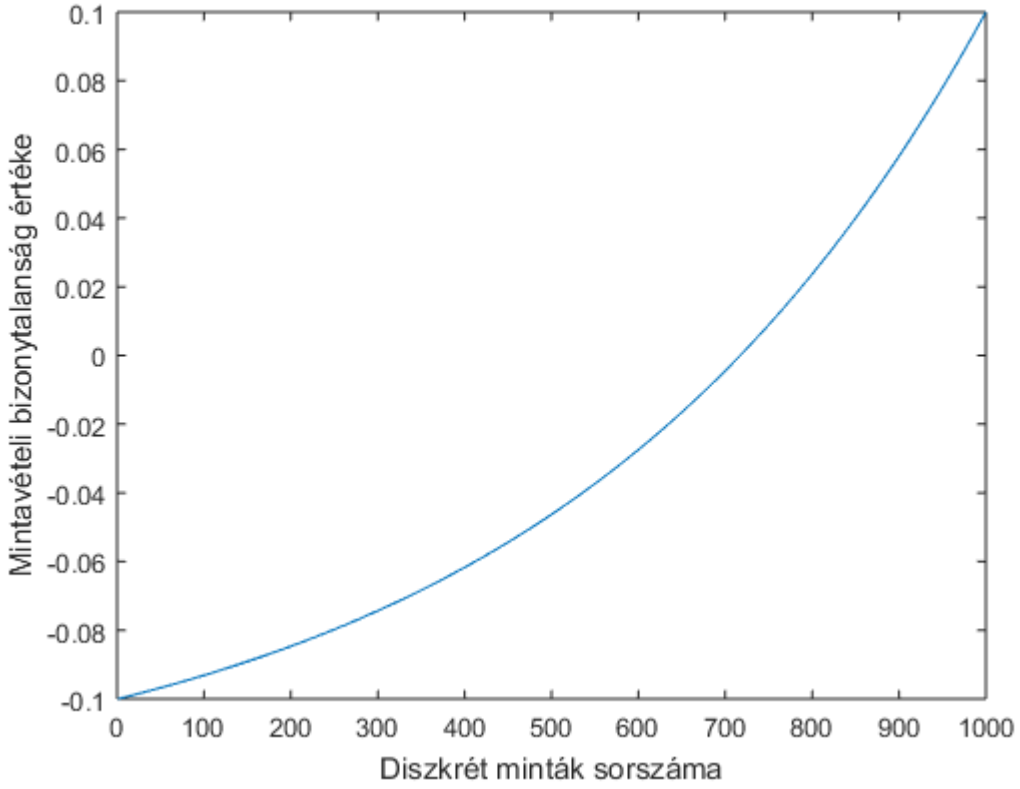
A következő szimulációban a (75)-ös kifejezésben szereplő, normális eloszlású mintavételi hibát exponenciális eloszlásúra cseréltem. A hiba értékének alakulása az idő függvényében a 7. ábrán látható. A szimulációt először 20±10 periódusszámmal végeztem.

			Középérték			
			A	B	C	J
3 paraméteres	LS	σ=0.2	0.002293	0.003864	-0.000007	
		σ=0.1	0.001441	-0.000903	0.000003	
		σ=0.01	-0.000094	-0.000066	0	
	TLS	σ=0.2	0.002294	0.003864	-0.000007	
		σ=0.1	0.001441	-0.000903	0.000003	
		σ=0.01	-0.000094	-0.000066	0	
4 paraméteres	LS	σ=0.2	0.002434	0.004055	-0.000008	0.009444
		σ=0.1	0.001511	-0.000933	0.000003	0.004688
		σ=0.01	-0.000103	-0.000068	0	0.000469
	TLS	σ=0.2	0.002433	0.004055	-0.000008	0.008877
		σ=0.1	0.001511	-0.000933	0.000003	0.00441
		σ=0.01	-0.000103	-0.000068	0	0.00044
Módosított TLS		σ=0.2	0.002305	0.003848	-0.000007	0.008852
		σ=0.1	0.001417	-0.000917	0.000003	0.004376
		σ=0.01	-0.0001	-0.000062	0	0.000437

5. táblázat - Hiba középértéke exponenciális eloszlású mintavételi bizonytalanság esetén (J=20±10)

			Szórás			
			A	B	C	J
3 paraméteres	LS	σ=0.2	0.026432	0.02715	0.000064	
		σ=0.1	0.013442	0.01326	0.000032	
		σ=0.01	0.001307	0.001358	0.000003	
	TLS	σ=0.2	0.026432	0.027151	0.000064	
		σ=0.1	0.013442	0.01326	0.000032	
		σ=0.01	0.001307	0.001358	0.000003	
4 paraméteres	LS	σ=0.2	0.02768	0.02842	0.000061	0.001067
		σ=0.1	0.014069	0.013875	0.00003	0.000537
		σ=0.01	0.001369	0.001424	0.000003	0.000045
	TLS	σ=0.2	0.02768	0.02842	0.000061	0.001008
		σ=0.1	0.014069	0.013875	0.00003	0.000508
		σ=0.01	0.001369	0.001424	0.000003	0.000042
Módosított TLS		σ=0.2	0.026625	0.026988	0.000065	0.001262
		σ=0.1	0.013474	0.013146	0.000033	0.000637
		σ=0.01	0.001306	0.001353	0.000004	0.000053

6. táblázat - Hiba szórása exponenciális eloszlású mintavételi bizonytalanság esetén (J=20±10)



7. ábra – A mintavételi bizonytalanság értéke diszkrét idő függvényében

			Középérték			
			A	B	C	J
3 paraméteres	LS	σ=0.2	0.016783	0.039546	0.000016	
		σ=0.1	0.017103	-0.00764	0.000001	
		σ=0.01	-0.00102	-0.0007	0	
	TLS	σ=0.2	0.016893	0.039526	0.000016	
		σ=0.1	0.017096	-0.00766	0.000001	
		σ=0.01	-0.00102	-0.0007	0	
4 paraméteres	LS	σ=0.2	0.016856	0.041133	0.000012	0.094102
		σ=0.1	0.017965	-0.00764	0.000001	0.046984
		σ=0.01	-0.00104	-0.0007	0	0.004698
	TLS	σ=0.2	0.016845	0.041127	0.000012	0.087924
		σ=0.1	0.017963	-0.00764	0.000001	0.044056
		σ=0.01	-0.00104	-0.0007	0	0.004421
Módosított TLS		σ=0.2	0.017354	0.04321	0.000008	0.100141
		σ=0.1	0.018851	-0.00795	0	0.05
		σ=0.01	-0.00109	-0.00073	0	0.004999

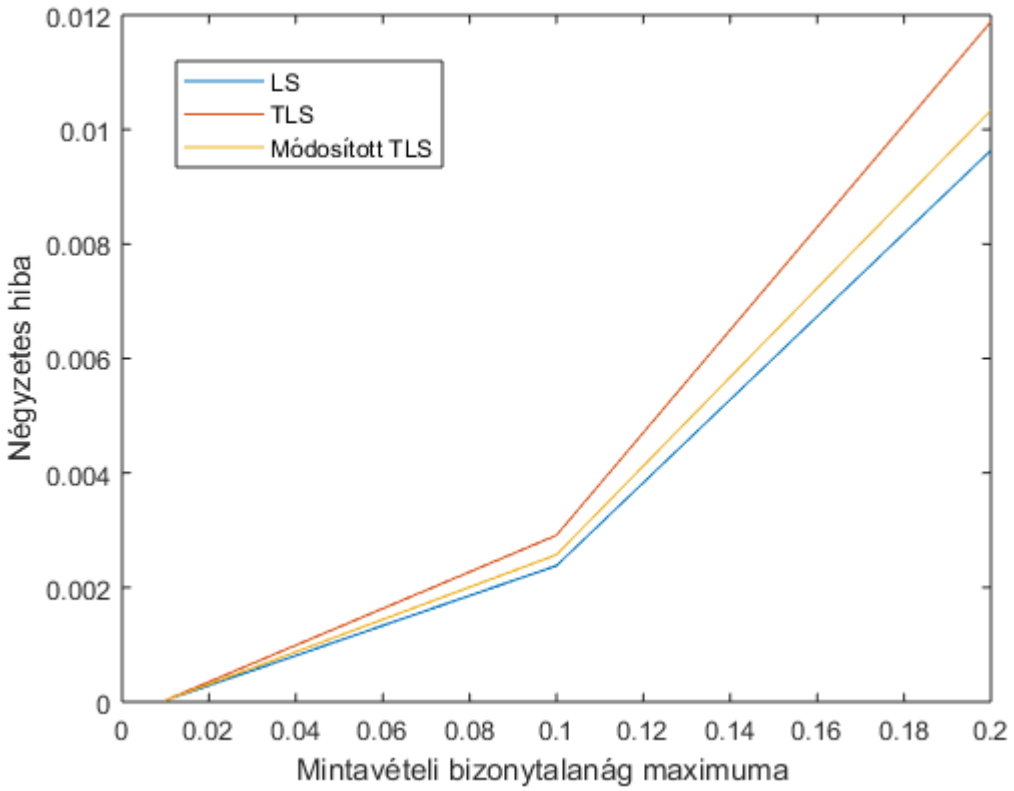
7. táblázat - Hiba középértéke exponenciális eloszlású mintavételi bizonytalanság esetén (J=250±20)

			Szórás			
			A	B	C	J
3 paraméteres	LS	σ=0.2	0.260472	0.264758	0.000071	
		σ=0.1	0.133182	0.13145	0.000037	
		σ=0.01	0.013243	0.013401	0.000004	
	TLS	σ=0.2	0.260579	0.264914	0.000071	
		σ=0.1	0.133207	0.131457	0.000037	
		σ=0.01	0.013243	0.013401	0.000004	
4 paraméteres	LS	σ=0.2	0.273855	0.278339	0.000067	0.002126
		σ=0.1	0.139559	0.138018	0.000035	0.001077
		σ=0.01	0.013857	0.014025	0.000004	0.00009
	TLS	σ=0.2	0.273799	0.278278	0.000067	0.002326
		σ=0.1	0.13955	0.138013	0.000035	0.001193
		σ=0.01	0.013857	0.014025	0.000004	0.000101
Módosított TLS		σ=0.2	0.287111	0.291577	0.000065	0.002277
		σ=0.1	0.146263	0.144665	0.000035	0.001141
		σ=0.01	0.014523	0.014696	0.000004	0.000095

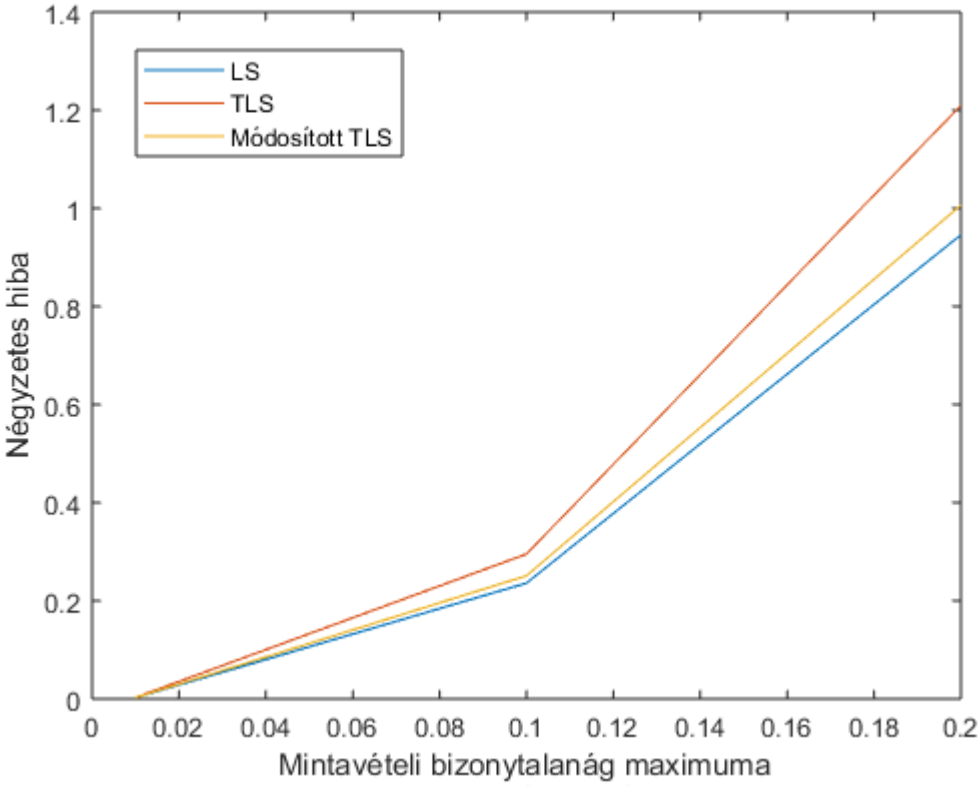
8. táblázat - Hiba szórása exponenciális eloszlású mintavételi bizonytalanság esetén (J=250±20)

A mintavételi bizonytalanság eloszlásának módosítása után összevetve az 1-es és az 5-ös, valamint a 2-es és a 6-os táblázatot az látható, hogy J paraméter kivételével a TLS és LS becslő algoritmusok becslési hibája megegyezik, a J paraméter tekintetében viszont a TLS jobbnak bizonyul. A módosított TLS algoritmus esetében is kisebb paraméterhibákat tapasztalhatunk exponenciális eloszlású mintavételi bizonytalanság hatására. A periódusok számának növelésével azonban a módosított TLS algoritmus hibája is növekszik ebben az esetben.

A mintavételi bizonytalanság szórásának függvényében felrajzolható a négyzetes hiba alakulása(7. és 8. ábra). A grafikonokon az látszik, hogy a TLS és a módosított TLS becslők négyzetes hibája nagyobb, mint az LS becslő négyzetes hibája. Ennek az az oka, hogy a TLS becslő paramétereit a legkisebb négyzeték módszere által használt költségfüggvénybe helyettesítettem vissza, ami nincs tekintettel a mintavételi bizonytalanságra. Ennek ellenére a négyzetes hiba különbsége az LS és TLS becslők között csupán 10^{-3} nagyságrendű 20±10 periódusszám használatával.



6. ábra - Négyzetes hiba alakulása különböző mértékű, exponenciális eloszlású mintavételi bizonytalanság hatására (J=20±10)



7. ábra - Négyzetes hiba alakulása különböző mértékű, exponenciális eloszlású mintavételi bizonytalanság hatására (J=250±20)

A négyzetes hiba 250±20 periódus használatával ebben az esetben az LS, a TLS és a módosított TLS becslők esetén is megnövekedett a 20±10 periódusszám esetén tapasztalt eredményekhez képest. Itt is látható, hogy exponenciális eloszlású mintavételi bizonytalanság esetén a módosított TLS hibája a periódusszám növelésével nem csökken, ahogyan azt a Gauss eloszlású bizonytalanságnál láthattuk.

8.3 Szimuláció exponenciális eloszlású mintavételi bizonytalansággal és additív, normál eloszlású zajjal

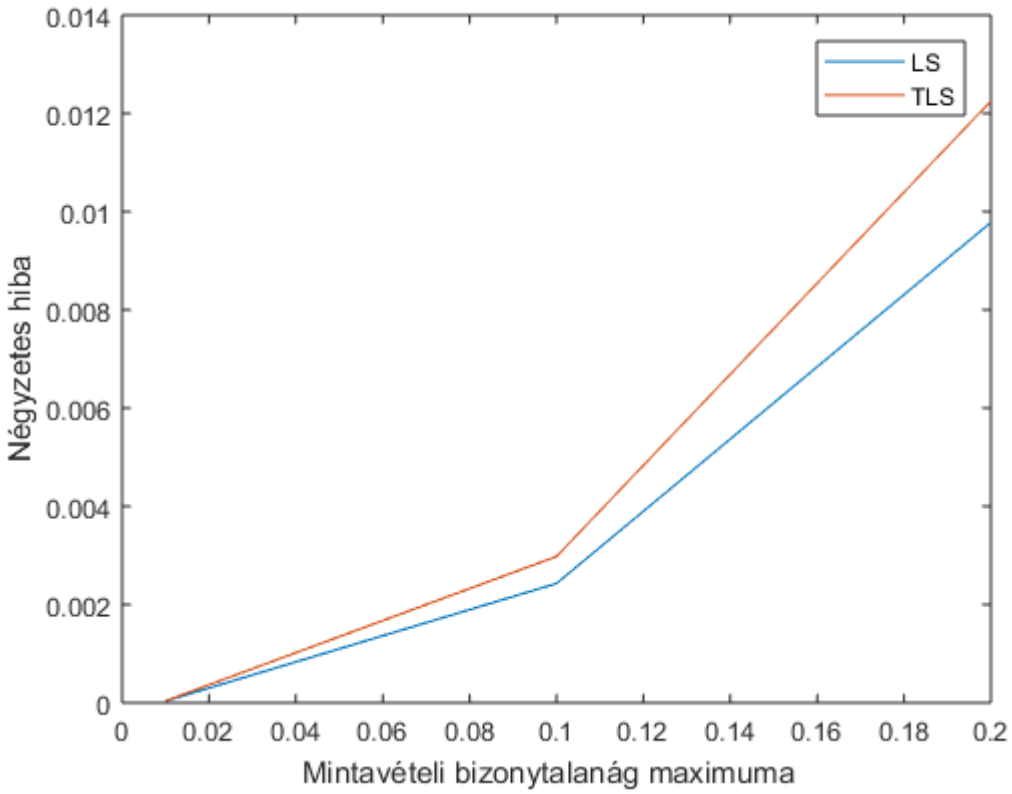
A mintavételi bizonytalanságból eredő zajon kívül normális eloszlású, additív zajjal is vizsgáltam az algoritmusokat. Ez a szimulációs eset vegyíti a fejezeben látott eddigi szimulációs példákat, hiszen egyszerre van benne jelen exponenciális és normális eloszlású zaj. Az exponenciális eloszlású mintavételi bizonytalanság mellett 80 dB jel/zaj viszony értékű normális eloszlású zajt adtam a jelhez. A vizsgálatokat elvégeztem 20±10 periódusszámmal és 250±20 periódusszámmal is.

			Középérték			
			A	B	C	J
3 paraméteres	LS	σ=0.2	0.000889	0.006428	-0.00001	
		σ=0.1	0.000786	0.000361	-0.000004	
		σ=0.01	0.000197	0.000112	0	
	TLS	σ=0.2	0.000891	0.006428	-0.00001	
		σ=0.1	0.000786	0.000361	-0.000004	
		σ=0.01	0.000197	0.000112	0	
4 paraméteres	LS	σ=0.2	0.000939	0.006749	-0.000011	0.009487
		σ=0.1	0.000838	0.000374	-0.000004	0.004738
		σ=0.01	0.000208	0.000118	0	0.000466
	TLS	σ=0.2	0.000939	0.006749	-0.000011	0.008888
		σ=0.1	0.000838	0.000374	-0.000004	0.004456
		σ=0.01	0.000208	0.000118	0	0.000438
Módosított TLS		σ=0.2	0.000181	0.010277	-0.000028	0.009446
		σ=0.1	0.00785	0.000993	-0.000004	0.005284
		σ=0.01	-0.000052	-0.00034	0.000025	0.000232

9. táblázat - Hiba középértéke exponenciális eloszlású mintavételi bizonytalanság és 80dB jel/zaj viszony értékű, normál eloszlású additív zaj esetén (J=20±10)

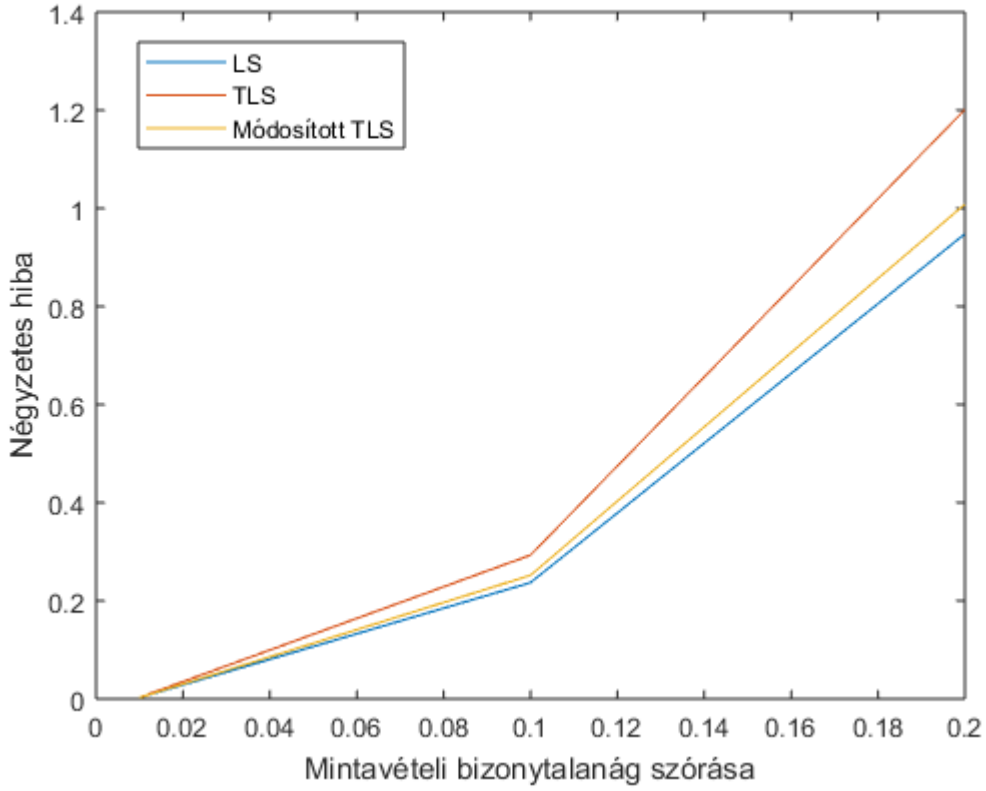
			Szórás			
			A	B	C	J
3 paraméteres	LS	σ=0.2	0.027572	0.025683	0.000067	
		σ=0.1	0.01334	0.013685	0.000035	
		σ=0.01	0.001231	0.001408	0.000005	
	TLS	σ=0.2	0.027572	0.025684	0.000067	
		σ=0.1	0.01334	0.013685	0.000035	
		σ=0.01	0.001231	0.001408	0.000005	
4 paraméteres	LS	σ=0.2	0.028951	0.026954	0.000063	0.001113
		σ=0.1	0.013954	0.014333	0.000034	0.00049
		σ=0.01	0.001289	0.001475	0.000005	0.000052
	TLS	σ=0.2	0.028951	0.026954	0.000063	0.001046
		σ=0.1	0.013955	0.014333	0.000034	0.000462
		σ=0.01	0.001289	0.001475	0.000005	0.000049
Módosított TLS		σ=0.2	0.037856	0.04059	0.000291	0.012779
		σ=0.1	0.023321	0.034014	0.000232	0.011481
		σ=0.01	0.024731	0.030034	0.000236	0.012398

10. táblázat - Hiba szórása exponenciális eloszlású mintavételi bizonytalanság és 80dB jel/zaj viszony értékű, normál eloszlású additív zaj esetén (J=20±10)



8. ábra - Négyzetes hiba alakulása különböző mértékű, exponenciális eloszlású mintavételi bizonytalanság és 80dB jel/zaj viszony értékű additív zaj hatására (J=20±10)

A szimulációt 250±20 periódusszámmal is lefuttattam. Ebben az esetben a módosított TLS hibájának csökkenni, míg az LS és TLS algoritmusok hibájának növekedni kell az elvárás alapján.



9. ábra - Négyzetes hiba alakulása különböző mértékű, exponenciális eloszlású mintavételi bizonytalanság és 80dB jel/zaj viszony értékű additív zaj hatására (J=250±20)

			Középérték			
			A	B	C	J
3 paraméteres	LS	б=0.2	0.001677	0.065881	-0.000002	
		б=0.1	0.008122	0.003975	-0.000003	
		б=0.01	0.001771	0.001222	0	
	TLS	б=0.2	0.001846	0.065916	-0.000002	
		б=0.1	0.008129	0.003963	-0.000003	
		б=0.01	0.001771	0.001222	0	
4 paraméteres	LS	б=0.2	0.000716	0.068579	-0.000004	0.094175
		б=0.1	0.008768	0.004277	-0.000003	0.047076
		б=0.01	0.001855	0.001288	0	0.004694
	TLS	б=0.2	0.000692	0.068553	-0.000004	0.088139
		б=0.1	0.008765	0.004283	-0.000003	0.044209
		б=0.01	0.001855	0.001288	0	0.00441
Módosított TLS		б=0.2	0.000239	0.071874	-0.000005	0.100218
		б=0.1	0.009175	0.004513	-0.000003	0.050094
		б=0.01	0.001948	0.001352	0	0.004999

11. táblázat - Hiba középértéke exponenciális eloszlású mintavételi bizonytalanság és 80dB jel/zaj viszony értékű, normál eloszlású additív zaj esetén (J=250±20)

			Szórás			
			A	B	C	J
3 paraméteres	LS	σ=0.2	0.270925	0.250154	0.000084	
		σ=0.1	0.133686	0.132784	0.000038	
		σ=0.01	0.012454	0.013977	0.000005	
	TLS	σ=0.2	0.271051	0.25029	0.000084	
		σ=0.1	0.133679	0.132825	0.000038	
		σ=0.01	0.012454	0.013977	0.000005	
4 paraméteres	LS	σ=0.2	0.284965	0.2622	0.00008	0.002223
		σ=0.1	0.140206	0.138944	0.000036	0.000987
		σ=0.01	0.013046	0.014641	0.000005	0.000103
	TLS	σ=0.2	0.284907	0.26215	0.00008	0.002458
		σ=0.1	0.140206	0.138931	0.000036	0.001145
		σ=0.01	0.013046	0.014641	0.000005	0.000115
Módosított TLS		σ=0.2	0.298694	0.274707	0.000078	0.002365
		σ=0.1	0.147024	0.145535	0.000034	0.001043
		σ=0.01	0.01369	0.015346	0.000005	0.000112

12. táblázat - Hiba szórása exponenciális eloszlású mintavételi bizonytalanság és 80dB jel/zaj viszony értékű, normál eloszlású additív zaj esetén (J=250±20)

A periódusszám növelésével az LS és TLS algoritmusok hibája növekszik, míg a módosított TLS hibája csökken, ahogy azt a normális eloszlású mintavételi bizonytalanság esetén is láthattuk.

9 Elméleti eredmények értékelése

A szimulációk elvégzése során célom a különböző becslési algoritmusok alkalmazhatóságának vizsgálata volt. Az alkalmazhatóságot jelen esetben a bemeneti mintákra felírt egyenlet megoldásának hibája dönti el. A hiba mértékének függvényében döntést kell hozni, hogy az adott algoritmust milyen jeltípusok esetén célszerű alkalmazni. A jeltípusokat az elvégzett vizsgálatok alapján három paraméterrel lehet jellemezni: mintavételi bizonytalanság eloszlása, additív zaj és a jelperiódusok száma.

Az LS becslő algoritmust normális eloszlású mintavételi bizonytalanság esetén éri meg használni, ugyanis a bizonytalanság szórásának növekedésével ez a becslő érte el a legkisebb hibát a TLS és a módosított TLS algoritmussal szemben. A TLS algoritmus használata az LS-el szemben exponenciális eloszlású mintavételi bizonytalanság esetén indokolt, mivel a bizonytalanság szórásának növekedésével a TLS algoritmus mindvégig kisebb becslési hibát ért el. A módosított TLS algoritmus a többi algoritmussal szemben szinte minden szimuláció alkalmával nagyobb becslési hibát mutatott. A vizsgált jelek közül egyedül az exponenciális eloszlású mintavételi bizonytalansággal rendelkező, additív zaj nélküli, 20±10 periódusszámú jeltípusra mutatott kiemelkedően kicsi becslési hibát.

A tapasztalt becslési hibák alapján az algoritmusok alkalmazását a következő táblázattal lehet jellemezni:

		Mintavételi bizonytalanság	
Mintavételi frekvencia	Additív zaj	exponenciális eloszlású	normális eloszlású
J=20±10	nincs additív zaj	Módosított TLS	LS
	normális eloszlású additív zaj	TLS	LS
J=250±20	nincs additív zaj	TLS	LS
	normális eloszlású additív zaj	TLS	LS

10 Becslési eljárások implementálása C nyelven

Az algoritmusok működését egy Cortex M4 alapú mikrokontroller segítségével igazoltam. A mikrokontroller programozására számos lehetséges módszer létezik.

Az arm mbed online C++ nyelvű fejlesztői környezetét használva lehetőségünk van előre megírt könyvtárak, programok használatára, azonban személyes preferenciák miatt inkább a C nyelvet választottam az STM32 CubeMX és Attollic TrueStudio fejlesztést támogató szoftverekkel. Az STM32 CubeMX segítségével generáltam az alap C kódot, amiben inicializálásra kerültek a használt perifériák, megszakítások és beállításra került a kiválasztott órajel forrás. Ezzel az alkalmazással hatékonyan sikerült redukálni a fejlesztéshez szükséges előkészületek idejét. Az Attollic TrueStudio egy C fejlesztői és hibakeresési környezetet biztosít.

Az eddig ismertetett algoritmusok közül a három paraméteres Least Squares és a három paraméteres Total Least Squares került implementálásra. Ennek oka, hogy a Total Least Squares becslő paramétereinek kiszámításához szükségünk van a rendszermátrix sajátértékeire. A rendszermátrix karakterisztikus egyenletéből ez egyszerűen kiszámítható. Háromparaméteres esetben ez egy negyedfokú egyenlet megoldását jelenti. Az egyenlet megoldásai egyszerű műveletek segítségével felírható a rendszermátrix elemeiből. Négy paraméteres esetben a sajátérték számítás egy ötödfokú egyenlet megoldásával lehetséges, ami egy erőforrásigényes gyökhelykeresési algoritmus implementációját igényelné, ami a diplomatervem céljai között nem szerepel.

A következő fejezetekben a háromparaméteres becslő algoritmusok C nyelvű implementációt ismerhetjük meg részletesen. Mivel a rendszermátrix mindkét algoritmus esetén megegyezik, a függvényhívások előtt inicializálni kell. Az inicializációhoz ismerni kell a CMSIS (Cortex Microcontroller Software Interface Standard) könyvtárban definiált mátrix struktúra felépítését. Az implementáció során az „arm_matrix_instance_f32” mátrix definíciót használtam. Ez a struktúra tartalmazza a mátrix sorainak és oszlopainak számát, valamint egy 32 bites float értékekből álló tömböt, amelyben sorrend szerint szerepelnek a mátrix elemei.

A mintákat tartalmazó tömb méretét a kód elején, fordítás előtt meghatározott makró segítségével állítom, amire N-nel hivatkozok. Fordítás során a C fordító a kódban

talált N karaktereket a makróban meghatározott számra cseréli. A rendszermátrixot tartalmazó vektor egy for ciklus segítségével inicializálható:

```
for(int ii=0;ii<(N*3);ii=ii+3)
{
 x_f32[ii]=arm_cos_f32(2*3.14159265358979*J_for_D/N*iii);
 x_f32[ii+1]=arm_sin_f32(2*3.14159265358979*J_for_D/N*iii);
 x_f32[ii+2]=1;
 iii++;
}
```

A cikluson belül található J_for_D, float32_t típusú változó tartalmazza az IpFFT által meghatározott kezdeti frekvenciát. A rendszermátrix használatához már csak hozzá kell rendelnünk ezt a vektort a megfelelő struktúrához. A hozzárendelést az algoritmusokat megvalósító C függvényekben végeztem el.

10.1 IpFFT C nyelvű megvalósítása

A rendszermátrix inicializálásához meg kell határozni a bemeneti mintákból a frekvencia értékét. Ez megoldható lenne diszkrét Fourier-transzformációval, de ez nem adna a becsléshez elég pontos kezdeti értéket, ezért szükséges az interpoláció alkalmazása.

Első lépésként létrehozom a függvényen belül használt lokális változókat:

```
 float32_t x_temp[N];
 float32_t x_temp2[2*N];
 float32_t max_val;
 uint32_t max_ind;
```

x_temp tartalmazza a bemeneti mintákat, a tömb értékei a függvény paramétereként kerül átadásra. x_temp2 változó a diszkrét Fourier-transzformációhoz szükséges ideiglenes mátrix, amelynek hossza a bemeneti mintákat tartalmazó tömb kétszerese, mivel a transzformáció végrehajtása a minták komplex alakján történik. A minták komplex alakja egy float32_t típusú tömbben kerül tárolásra, amelynek elemei felváltva valós és komplex együtthatók. A max_val és max_ind változók tárolják a Fourier-transzformáció után a maximális aplitúdó értékét, valamint a maximális amplitúdó tömbben elfoglalt helyét.

A változók létrehozása után Hanning ablakot használva ablakozom a mintákat tartalmazó tömböt:

```
for(int iterator=0;iterator<N;iterator++)
 {
  float32_t cos=arm_cos_f32(2*3.14159265358979*iterator/N);
  w_f32[iterator]=0.5 – 0.5 * cos;
  xw_f32[iterator]=x_temp[iterator]*w_f32[iterator];
 }
```

Ablakozás után feltöltöm a minták értékével az x_temp2 tömböt. A tömb minden második eleme 0 értékű, mivel tisztán valós jelre használom a diszkrét FFT-t.

```
for (int n = 0; n < N; n++)
 {
  x_temp2[2*n] = xw_f32[n];
  x_temp2[2*n + 1] = 0;
 }
```

A diszkrét FFT bemeneti tömbjének feltöltése után végrehajtom a Fourier-transzformációt, veszem az eredmény abszolút értékét és megkeresem a maximális értékű elemét és a hozzá tartozó tömbindexet.

```
arm_cfft_f32(&arm_cfft_sR_f32_len128, x_temp2, 0, 1);
arm_cmplx_mag_f32(x_temp2, x_temp, N);
arm_max_f32((x_temp+1), N - 1, &max_val, &max_ind);
```

Az arm_cfft_f32 függvény bemeneti paraméterként vár egy konfigurációs struktúrát, ami többek között meghatározza a bemeneti minták számát is. Második paraméterként át kell adni neki a komplex értékeket tartalmazó tömböt, amelybe a transzformáció eredménye is kerül. Az utolsó két paraméter két bináris érték. Az első meghatározza, hogy az alkalmazott transzformáció FFT vagy inverz FFT, a második pedig a bitfordítás engedélyezése. Ha a bitfordítást nem engedélyezzük, az eredmény bitjeit fordított sorrendben kapjuk meg.

A transzformáció után az arm_cmplx_mag_f32 függvény segítségével az eredmény abszolút értékét veszem. A függvény bemeneti paraméterként várja a forrás és az eredmény tömbjét, valamint a minták számát. Az eredmények tárolásához elég egy N hosszú tömb, mivel nincs szükség komplex értékek tárolására, így felhasználhatjuk a korábban létrehozott x_temp tömböt.

Az abszolútérték képzés után az eredménytömbben keresem meg a maximális értékkel rendelkező indexet, amely kezdeti értéket jelent az interpolációhoz. Hibakeresési céllal külön változóba a tömb indexének értékét is elmentem. Normál esetben az index csak a maximum pozícióját jelöli és nem következik belőle a frekvencia értéke. Mivel a

teszt során mértékegység nélküli frekvencia értékeket használok, nem szükséges további átalakítás. Ellenkező esetben szükség lenne beszorozni a Fourier-transzformáció alapegységével. A bemeneti tömb pointerének értékét eggyel megnöveltem, ezzel kihagyva a keresésből az egyen szintet, azaz a 0-ás tömbindexet.

A következő lépésben elvégzem a 4. fejezetben ismertetett interpolációt. Az interpoláció eredménye a float32_t típusú dJ változó értéke, ami a maximális indexhez szükséges kompenzációt határozza meg.

```
if(x_temp2[max_ind-1]<x_temp2[max_ind+1])
 {
  dJ=(2*x_temp2[max_ind+1]-
x_temp2[max_ind])/(x_temp2[max_ind+1]+x_temp2[max_ind]);
 }else
 {
  dJ=(x_temp2[max_ind]-2*x_temp2[max_ind-
1])/(x_temp2[max_ind]+x_temp2[max_ind-1]);
 }
```

Az interpoláció után a max_index változó értékét korrigálom a kiszámolt dJ-vel, az eredményt elmentem Jk változóba, ami az IpFFT függvény visszatérési értéke.

```
Jk =(max_ind+dJ);
return Jk;
```

10.2 Három paraméteres Least Squares

A becslő algoritmusok első lépéseként inicializálni kell az alkalmazott mátrixokat. Az inicializálás során a korábban említett mátrix struktúra paraméterei kerülnek beállításra.

Az inicializáció az „arm_mat_init_f32” függvénnyel végezhető el. A függvény paraméterként várja sorban az inicializálni kívánt mátrix példányát, sorainak, oszlopainak számát, valamint a mátrix elemeit tartalmazó tömböt. Fontos figyelni rá, hogy a mátrix mérete megegyezzen az elemeket tartalmazó tömb méretével, azaz a tömb mérete meg kell egyezzen a sorok számának és az oszlopok számának szorzatával. Az inicializálás során a sorok méretét egy N paraméterrel állítottam be, aminek köszönhetően a bemeneti paraméterek száma az N makró értékének átírásával és a kód újrafordításával módosítható.

Először inicializálom a rendszermátrixot:

```
arm_mat_init_f32(&x,N,3,(float32_t *)x_f32);
```

Ezután a forrás jel mintáit tartalmazó vektort helyezem mátrix struktúrába:

```
arm_mat_init_f32(&y,N,1,(float32_t *)f_array);
```

Ezután pedig az összes többi, számítás során használt mátrixot:

```
arm_mat_init_f32(&xt,3,N,(float32_t *)xt_f32);
arm_mat_init_f32(&tempm1,3,3,(float32_t *)tempm1_f32);
arm_mat_init_f32(&tempm2,3,3,(float32_t *)tempm2_f32);
arm_mat_init_f32(&tempm3,3,1,(float32_t *)tempm3_f32);
arm_mat_init_f32(&tempm4,3,1,(float32_t *)tempm4_f32);
```

Inicializálás után elvégezzük az LS becslést mátrix műveletek segítségével. A becsléshez az (10)-es egyenletet kell megoldanom, amihez felhasználtam a mátrix transzponálás, szorzás és invertálás függvényeit.

```
arm_mat_trans_f32(&x,&xt);
arm_mat_mult_f32(&xt, &x, &tempm1);
arm_mat_inverse_f32 (&tempm1, &tempm2);
arm_mat_mult_f32 (&xt, &y, &tempm3);
arm_mat_mult_f32 (&tempm2, &tempm3, &tempm4);
ls_A=tempm4.pData[0];
ls_B=tempm4.pData[1];
ls_C=tempm4.pData[2];
fi = atan2f(-ls_B,ls_A);
```

A transzponálás függvényének használatához meg kell adni a transzponálandó mátrix referenciáját, valamint az eredménymátrix referenciáját, a szorzáshoz a két szorzó referenciáját, valamint az eredménymátrix referenciáját, invertáláshoz pedig az invertálandó mátrix referenciáját és az eredménymátrix referenciáját. A három paraméter a tempm4 mátrix elemei közül kerül ki, sorban A, B és C, valamint a (12)-es egyenletnek megfelelően kiszámolható a kezdőfázis.

10.3 Három paraméteres Total Least Squares

A 6. fejezetben már említésre került a Total Least Saquares becslő megvalósíthatóságának kérdése. Asztali számítógépes környezetben nem jelenthet problémát az SVD felbontás elvégzése Matlab segítségével, azonban ennek C nyelvű implementációja rendkívül nehéz. A gyakorlati megvalósítást ezért a TLS becslés zárt alakú felírásával végeztem, amit a (61)-es kifejezés ír le.

A kihívást ebben az esetben a $[D \quad x]^T[D \quad x]$ mátrix sajátértékeinek számítása jelenti. Matlab segítségével lehetőségünk van bármilyen kifejezést C nyelvű kóddá alakítani a „ccode()” függvény használatával. Mivel ismert a mátrix pontos mérete (4x4), kézenfekvő megoldás lehet a mátrix elemeit szimbólumokkal felírni és a szimbólumok

segítségével kiszámítani a sajátértékeket. Ezt a következő Matlab utasítások sorozatával tehetjük meg:

```
syms a b c d e f g h i j k l m n o p;
M=[a b c d; e f g h; i j k l; m n o p];
ccode(eig(M))
```

Az első sorban létrehozzuk a szimbólumokat, amelyek a mátrix elemeit alkotják. A második sorban elhelyezzük ezeket a szimbólumokat a mátrixban, majd a negyedik sorban kiszámítjuk a sajátértékeket és ebből C nyelvű kódot generálunk. Ez a megközelítés a lehető legegyszerűbb módja a feladat megoldásának, azonban túlságosan nagy méretű kódot eredményez, ami az általam használt Cortex M4 alapú mikrokontroller memóriájába nem fér bele.

Egy másik megoldási lehetőség, ha felírom a karakterisztikus egyenletet, amelynek gyökei a sajátértékek, majd megoldóképletet alkalmazva kiszámolom. A karakterisztikus egyenlet a következő kifejezéssel kapható meg:

$$\det([\mathrm{D}\quad \mathrm{x}]^{\mathrm{T}}[\mathrm{D}\quad \mathrm{x}] - \lambda * \mathrm{I}) = 0 \tag{70}$$

Az így kapott, λ hatványainak együtthatóira alkalmazni kell a negyedfokú egyenlet megoldóképletét:

$$ax^4 + bx^3 + cx^2 + dx + c = 0$$

$$\sigma_{1,2} = -\frac{b}{4a} - S \pm \frac{1}{2}\sqrt{-4S^2 - 2p + \frac{q}{S}} \tag{71}$$

$$\sigma_{3,4} = -\frac{b}{4a} + S \pm \frac{1}{2}\sqrt{-4S^2 - 2p + \frac{q}{S}},$$

ahol

$$p = \frac{8ac - 3b^2}{8a^2}\ ,\quad q = \frac{b^3 - 4abc + 8a^2d}{8a^3}\ ,$$

$$S = \frac{1}{2}\sqrt{-\frac{2}{3}p + \frac{1}{3a}\left(Q + \frac{\Delta_0}{Q}\right)}\ ,Q = \sqrt[3]{\frac{\Delta_1 + \sqrt{\Delta_1^2 - 4\Delta_0^3}}{2}}\ ,$$

$\Delta_0 = c^2 - 3bd + 12ae$,

$\Delta_1 = 2c^3 - 9bcd + 27b^2e + 27ad^2 - 72ace$

a, b, c, d és e paraméterek pedig rendre a karakterisztikus polinomban szereplő negyed, harmad, másod, első és konstans együtthatók.

Az implementáció során oda kell figyelni a gyökvonások argumentumára, mivel a lebegőpontos számábrázolás miatt a változók pontatlansága az értékük növekedésével egyre nagyobb. Lehetséges tehát, hogy két nagy értékű változó kivonása a számábrázolási hiba miatt negatív értéket, a számítás során ez pedig komplex gyököket eredményez, holott tisztán valós gyökökre számítunk.

A C nyelvű implementáció első lépésekén inicializálom az algoritmus futása során használt mátrixokat:

```
arm_mat_init_f32(&ident,3,3,(float32_t *)ident_f32);

arm_mat_init_f32(&tls_svd_trans,4,N,(float32_t *)tls_svd_trans_f32);
arm_mat_init_f32(&tls_svd_temp1,4,4,(float32_t *)tls_svd_temp1_f32);
arm_mat_init_f32(&beta,3,1,(float32_t *)beta_f32);
arm_mat_init_f32(&tls_svd,N,4,(float32_t *)tls_svd_f32);
arm_mat_init_f32(&tls_temp1,3,3,(float32_t *)tls_temp1_f32);
arm_mat_init_f32(&tls_temp2,3,3,(float32_t *)tls_temp2_f32);
arm_mat_init_f32(&tls_temp3,3,3,(float32_t *)tls_temp3_f32);
arm_mat_init_f32(&tls_temp4,3,N,(float32_t *)tls_temp1_f32);
```

Az ident mátrixhoz hozzárendelem az ident_f32 float32_t típusú tömböt, amely egy 3x3-as identitásmátrixot tartalmaz. Következő lépésként egy for cikluson belül beletöltöm $[D \quad x]$ mátrix elemeit tls_svd mátrix struktúrába.

```
for(int iterator=0;iterator<N*4;iterator++)
 {
  if(iterator%4 == 0)
  {
   tls_svd_f32[iterator]=f_array[iterator_y];
   iterator_y++;
  } else
  {
   tls_svd_f32[iterator]=x_f32[iterator_x];
   iterator_x++;
  }
 }
```

Ezután kiszámolom $[\mathrm{D} \quad \mathrm{x}]^{\mathrm{T}}[\mathrm{D} \quad \mathrm{x}]$ mátrix elemeit. Először transzponálom tls_svd mátrixot és a transzponálás eredményét tls_svd_trans mátrixba mentem, majd a tls_svd és a tls_svd_trans mátrixokat összeszorzom.

```
arm_mat_trans_f32(&tls_svd,&tls_svd_trans);
arm_mat_mult_f32 (&tls_svd_trans, &tls_svd, &tls_svd_temp1);
```

Ezután az eredmény elemeit float32_t típusú változókba mentem, hogy egyszerűsítsem a megoldóképlettel történő felírást:

```
a=tls_svd_temp1_f32[0];
b=tls_svd_temp1_f32[1];
c=tls_svd_temp1_f32[2];
d=tls_svd_temp1_f32[3];
e=tls_svd_temp1_f32[4];
f=tls_svd_temp1_f32[5];
g=tls_svd_temp1_f32[6];
h=tls_svd_temp1_f32[7];
i=tls_svd_temp1_f32[8];
j=tls_svd_temp1_f32[9];
k=tls_svd_temp1_f32[10];
l=tls_svd_temp1_f32[11];
m=tls_svd_temp1_f32[12];
n=tls_svd_temp1_f32[13];
o=tls_svd_temp1_f32[14];
p=tls_svd_temp1_f32[15];
```

A Matlabbal kiszámolt együtthatók értékét erre a célra létrehozott float32_t típusú változókba mentem:

```
A = 1;

B = (0 - a - f - k - p);

C = (a*f - b*e + a*k - c*i + a*p - d*m + f*k - g*j + f*p - h*n + k*p - l*o);

D = (a*g*j - a*f*k + b*e*k - b*g*i - c*e*j + c*f*i - a*f*p + a*h*n + b*e*p -
b*h*m - d*e*n + d*f*m - a*k*p + a*l*o + c*i*p - c*l*m - d*i*o + d*k*m - f*k*p
+ f*l*o + g*j*p - g*l*n - h*j*o + h*k*n);

E = a*f*k*p - a*f*l*o - a*g*j*p + a*g*l*n + a*h*j*o - a*h*k*n - b*e*k*p +
b*e*l*o + b*g*i*p - b*g*l*m - b*h*i*o + b*h*k*m + c*e*j*p - c*e*l*n - c*f*i*p
+ c*f*l*m + c*h*i*n - c*h*j*m - d*e*j*o + d*e*k*n + d*f*i*o - d*f*k*m -
d*g*i*n + d*g*j*m;

Q=powf(A*(D*D)*(2.7E1/2.0)+(B*B)*E*(2.7E1/2.0)+C*C*C+sqrtf(powf(C*C+A*E*1.2E1
-B*D*3.0,3.0)*-4.0+powf(A*(D*D)*2.7E1+(B*B)*E*2.7E1+(C*C*C)*2.0-A*C*E*7.2E1-
B*C*D*9.0,2.0))*(1.0/2.0)-A*C*E*3.6E1-B*C*D*(9.0/2.0),1.0/3.0);

S=sqrtf(1.0/(A*A)*((B*B)*3.0-
A*C*8.0)*(1.0/1.2E1)+(Q*(1.0/3.0)+((C*C)*(1.0/3.0)+A*E*4.0-
B*D)/Q)/A)*(1.0/2.0);
```

Ezután megvizsgálom a gyökvonások argumentumait és ha a számábrázolási hiba miatt negatív ezek értéke, nullával helyettesítem a gyökvonás eredményét. Ezzel a helyettesítéssel növelem az algoritmus pontatlanságát, azonban a komplex számokkal történő számítás a struktúrából adódóan kifejezetten nehéz, ésszerű elhanyagolást jelent a helyettesítés.

```
if(powf(C*C+A*E*1.2E1-B*D*3.0,3.0)*-
4.0+powf(A*(D*D)*2.7E1+(B*B)*E*2.7E1+(C*C*C)*2.0-A*C*E*7.2E1-
B*C*D*9.0,2.0)<0)
 {
  Q=pow(A*(D*D)*(2.7E1/2.0)+(B*B)*E*(2.7E1/2.0)+C*C*C-A*C*E*3.6E1-
B*C*D*(9.0/2.0),1.0/3.0);
 }
debug1=((1.0/(A*A)*((B*B)*3.0-A*C*8.0)*(1.0/4.0)-(S*S)*4.0-
(1.0/(A*A*A)*((A*A)*D*8.0+B*B*B-A*B*C*4.0)*(1.0/8.0))/S);
if(debug1<0)
{
 debug1=0;
}
```

A debug1 változó hibakeresés céljából lett létrehozva. A tesztek során több helyen is előfordultak negatív számok a gyökvonásokon belül, ezért a hibakeresés céljából létrehozott változót felhasználtam a negatív értékek korrigálása során.

A sajátértékeket ezután a szig nevű tömb elemeibe mentettem.

```
szig[0] = -S+sqrtf(1.0/(A*A)*((B*B)*3.0-A*C*8.0)*(1.0/4.0)-
(S*S)*4.0+(1.0/(A*A*A)*((A*A)*D*8.0+B*B*B-A*B*C*4.0)*(1.0/8.0))/S)*(1.0/2.0)-
(B*(1.0/4.0))/A;

szig[1] = -S-sqrtf(1.0/(A*A)*((B*B)*3.0-A*C*8.0)*(1.0/4.0)-
(S*S)*4.0+(1.0/(A*A*A)*((A*A)*D*8.0+B*B*B-A*B*C*4.0)*(1.0/8.0))/S)*(1.0/2.0)-
(B*(1.0/4.0))/A;

szig[2] = S+debug1*(1.0/2.0)-(B*(1.0/4.0))/A;

szig[3] = S-debug1*(1.0/2.0)-(B*(1.0/4.0))/A;
```

Következő lépésben buborék rendezéssel sorba rendezem a sajátértékeket a legnagyobbtól a legkisebbig.

```
for(int iii=0;iii<4;iii++)
 {
  for(int ii=0;ii<4;ii++)
   {
    if(szig[ii]<szig[ii+1])
     {
      sort_temp=szig[ii];
      szig[ii]=szig[ii+1];
      szig[ii+1]=sort_temp;
     }
   }
 }
```

A következő lépésben beírom az identitásmátrixba a legkisebb sajátérték négyzetét. Mátrixszorzás helyett inkább a tömb értékeit módosítottam az átláthatóság érdekében.

```
ident_f32[0]=powf(szig[3],2);
ident_f32[1]=0;
ident_f32[2]=0;
ident_f32[3]=0;
ident_f32[4]=powf(szig[3],2);
ident_f32[5]=0;
ident_f32[6]=0;
ident_f32[7]=0;
ident_f32[8]=powf(szig[3],2);
```

Ezután végrehajtom a (61)-es kifejezésben szereplő mátrixműveleteket.

```
arm_mat_trans_f32(&x,&xt);
arm_mat_mult_f32(&xt, &x, &tls_temp1);
arm_mat_sub_f32(&tls_temp1,&ident,&tls_temp2);
arm_mat_inverse_f32 (&tls_temp2, &tls_temp3);
arm_mat_mult_f32 (&tls_temp3, &xt, &tls_temp4);
arm_mat_mult_f32 (&tls_temp4, &y, &beta);
```

Végül kigyűjtöm a becslési paramétereket beta mátrixból és kiszámolom a kezdőfázist.

```
tls_A=beta.pData[0];
tls_B=beta.pData[1];
tls_C=beta.pData[2];
fi_tls = atan2f(-tls_B,tls_A);
```

10.4 Eredmények értékelése

A szimulációs eredmények értékelése után az algoritmusok C nyelvű implementációját is elvégeztem, amelynek célja az algoritmusok beágyazott rendszereken történő futtatásának vizsgálata volt.

Amíg a szimuláció során az algoritmusok becslési hibájá, addig ebben a fejezetben a megvalósítás során fellépő nehézségek, megkötések és elhanyagolások kerültek előtérbe. Az implementáció során felhasználtam a fejlesztői környezet által tartalmazott CMSIS könyvtárat, amely kész megoldásokat ad a különböző mátrix műveletek elvégzéséhez. A könyvtár használatával a Least Squares algoritmus implementációja nem jelentett problémát, hiszen a becslő értéke kiszámítható egyszerű mátrixműveletek elvégzésével. A Total Least Squares algoritmusra ez a megállapítás nem érvényes. Az implementáció során meg kellett kerüljem a szinguláris érték szerinti felbontás alkalmazását, mivel beágyazott környezetben ez nem triviális, nehéz feladat. A megvalósításhoz ezért felhasználtam a Total Least Squares becslő zárt alakú megoldását, aminek segítségével a sajátértékek ismeretében a becslés elvégezhető. A sajátértékek kiszámításához három paraméteres Total Least Squares becslő esetén a negyedfokú karakterisztikus egyenlet gyökeivel juthatunk. A számítást nehezítette a számábrázolás hibája, amelynek kiküszöböléséhez megkötéseket kellett tennem.

Ezek a megkötések növelik a becslő hibáját, így beágyazott környezetben a Total Least Squares algoritmus használata nagyobb becslési hibát okoz, mintha Least Squares becslőt alkalmaztunk volna.

11 Összefoglalás

Dolgozatom elkészítése során a különböző szinuszbecslő eljárások érzékenységét vizsgáltam. A vizsgálat célja volt, hogy megállapítsam a mintavételi időpont bizonytalanság hatását a becslési algoritmusok statisztikai tulajdonságaira.

A mintavételi bizonytalanság kapcsán előtérbe került a Total Least Squares algoritmus használata, mivel ellentétben a Least Squares algoritmussal, képes a minták időbeli eltolódását figyelembe venni. Az irodalomkutatást Jian Qiu Zhang, Zhao Xinmin, Hu Xiao, és Sun Jinwei 1997-es publikációjával kezdtem [3]. A cikk szerzői a Total Least Squares algoritmust felhasználva publikáltak egy lehetséges megoldást a mintavételi idő bizonytalanságából eredő hiba korrigálására. A publikációban közölt eljáráson kívül megvizsgáltam a szabványos, Least Squares algoritmussal elért megoldások mellett ugyanazon egyenletek Total Least Squares algoritmussal történő megoldását. A vizsgálat során elvégeztem az algoritmusok szimulációját különböző hibaforrások alkalmazásával, olyan jeltípusok keresése céljából, amelyek esetén a Total Least Squares becslő használatával kisebb becslési hiba érhető el. A szimuláció eredményeit felhasználva levontam a következtetéseket és meghatároztam azokat a jeltípusokat, amelyek esetén a jelparamétereket a Total Least Squares algoritmus kisebb hibával becsüli, mint a Least Squares algoritmus. Az elméleti vizsgálat után elkészítettem a három paraméteres becslő algoritmusok C nyelvű implementációit. Az implementáció során bemutatásra került több lehetséges módszer, amelyek alkalmazásával a Total Least Squares algoritmus szinguláris érték szerinti felbontás nélkül megvalósítható. Ezek a módszerek kiemelten fontos szerepet játszanak az algoritmusok alkalmazhatóságának vizsgálatában, mivel a szinguláris érték szerinti felbontás problémájának megoldása mikrovezérlőn nem triviális feladat. Az integrálhatóságot eldönti, hogy az adott alternatív megközelítés használatával milyen pontos végeredményt kapunk és ennek elérése érdekében milyen megkötéseket kell tennünk.

Az algoritmusok mikrovezérlők által nyújtott környezetbe való integrálásával elérhetjük, hogy a becslést alacsony fogyasztású eszközökön is el tudjuk végezni, ezzel elősegítve az energia és költséghatékonyságot.

12 Irodalomjegyzék

[1] IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters," in *IEEE Std 1241-2010 (Revision of IEEE Std 1241-2000)* , vol., no., pp.1-139, 14 Jan. 2011, doi: 10.1109/IEEESTD.2011.5692956

[2] Ming, X. and Kang, D., 1996. Corrections for frequency, amplitude and phase in a fast Fourier transform of a harmonic signal. *Mechanical Systems and Signal Processing*, *10*(2), pp.211-221.

[3] Zhang, J.Q., Xinmin, Z., Xiao, H. and Jinwei, S., 1997. Sinewave fit algorithm based on total least-squares method with application to ADC effective bits measurement. *IEEE transactions on Instrumentation and Measurement*, *46*(4), pp.1026-1030.

[4] Golub, G.H. and Van Loan, C.F., 1980. An analysis of the total least squares problem. *SIAM journal on numerical analysis*, *17*(6), pp.883-893.

[5] Andersson, T. and Handel, P., 2006. Cramer-Rao bound and the parsimony principle. Instrumentation and Measurement. *IEEE Transactions*, *55*(1), pp.44-53.

[6] Pešta, M., 2008. Total least squares approach in regression methods. *WDS'08 Proceedings of Contributed Papers: Part I–Mathematics and Computer Sciences*, pp.88-93.

[7] Wang, X. and Xu, Y., 2010, August. Total Least Squares method for sine fitting. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on* (Vol. 6, pp. V6-590). IEEE.

[8] Handel, P., 2000. Properties of the IEEE-STD-1057 four-parameter sine wave fit algorithm. *IEEE Transactions on Instrumentation and Measurement*, *49*(6), pp.1189-1193.

[9] Bilau, T.Z., Megyeri, T., Sárhegyi, A., Márkus, J. and Kollár, I., 2004. Four-parameter fitting of sine wave testing result: iteration and convergence. *Computer Standards & Interfaces*, *26*(1), pp.51-56.

[10] CMSIS DSP Software Library, http://www.keil.com/pack/doc/CMSIS/DSP/html/index.html, 2018.12.16

Függelék

A megvalósított main.c fájl tartalma:

```
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  ** This notice applies to any and all portions of this file
  * that are not between comment pairs USER CODE BEGIN and
  * USER CODE END. Other portions of this file, whether
  * inserted by the user or by software development tools
  * are owned by their respective copyright owners.
  *
  * COPYRIGHT(c) 2018 STMicroelectronics
  *
  * Redistribution and use in source and binary forms, with or without modification,
  * are permitted provided that the following conditions are met:
  *   1. Redistributions of source code must retain the above copyright notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above copyright notice,
  *      this list of conditions and the following disclaimer in the documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its contributors
  *      may be used to endorse or promote products derived from this software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
  *
  ******************************************************************************
  */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include <stm32f4xx.h>
#include "stm32f4xx_hal.h"

/* USER CODE BEGIN Includes */
#include "arm_math.h"
#include "math.h"
#include "arm_const_structs.h"
/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
```

```
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_ADC1_Init(void);
static void MX_NVIC_Init(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/

/* USER CODE END PFP */

/* USER CODE BEGIN 0 */
#define N 128        //Mintavételi pontok száma, bemenet mérete

// UART fogadással kapcsolatos változók
static uint8_t UART_data, data_ready=0;

// UART buffer változók
static char buffer[100]="", TXbuffer[100]="";

// UART küldés flag
static volatile uint8_t Tx_ready=0;


float32_t f_array[N]; // UART felől kapott számok tömbje

float32_t x_f32[N*3];  //D rendszermátrix elemeit tartalmazó tömb
float32_t xt_f32[N*3]; //x transzponáltja
float32_t y_f32[N*3];


// Számításokhoz szükséges ideiglenes mátrixot (részeredmények tárolása)
float32_t tempm1_f32[9];
float32_t tempm2_f32[9];
float32_t tempm3_f32[3];
float32_t tempm4_f32[3];
float32_t A;
float32_t B;
float32_t C;
float32_t D;
float32_t E;
float32_t Q;
float32_t S;
float32_t debug1;


float32_t delta;


float32_t tls_svd_f32[N*4];                  // [y,x] mátrixot tartalmazza y: bemenet, x: rendszermátrix
float32_t beta_f32[3];                       //TLS algoritmus becslőket tartalmazó vektora
float32_t szig[4];                           //Sajátértékeket tartalmazó vektor
float32_t   subs1,subs2,subs3,subs4,subs5,subs6,subs7;            //Negyedfokú egyenlet megoldásához szükséges behelyettesítés (optimalizáció céljával)
float32_t ident_f32[9]={    1,0,0, ,1,0, 0,0,1};       //Identitásmátrix 4x4
float32_t tls_svd_trans_f32[N*4];            //tls_svd transzponáltja
float32_t tls_svd_temp1_f32[16];             //ideiglenes mátrix (részeredmények tárolása)
float32_t  a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p;     //Mátrix elemeit tároljuk egyéni változókban a sajátértékek kiszámításának megkönnyítésére

//ideiglenes mátrix (részeredmények tárolása)
float32_t sort_temp;
float32_t tls_temp1_f32[9];
float32_t tls_temp2_f32[9];
float32_t tls_temp3_f32[9];
float32_t tls_temp4_f32[N*3];

// iterációs céllal létrehozott változók
```

```
uint32_t ind;
uint32_t index_value;
uint32_t f_array_size=0;
uint32_t decimal;
uint32_t iii=1;
uint32_t iterator_y=0;
uint32_t iterator_x=0;
uint32_t uart_i=0;
uint32_t uart_ii=0;
uint32_t sign_flag=0;



//float tömbökhöz tartozó mátrixok
arm_matrix_instance_f32      x,y,xt,tempm1,tempm2,tempm3,tempm4,tls_svd,beta,ident,tls_svd_trans,tls_svd_temp1,
tls_temp1, tls_temp2, tls_temp3, tls_temp4;

arm_status status;

//ipfft által használt tömbök
float32_t dJ, xw_f32[N],w_f32[N],Y_f32[N],Y_f32_real[N/2];
arm_matrix_instance_f32 xw;
arm_rfft_fast_instance_f32 fft_instance;

float32_t ls_A, ls_B, ls_C, Jk, fi;
float32_t tls_A, tls_B, tls_C, fi_tls;

float32_t ipfft(float32_t *input_array)
{
   // Függvény bemeneti tömb elemeinek másolása lokális tömb változójába
   float32_t x_temp[N];
   for(int it=0;it<N;it++){
      x_temp[it]=input_array[it];
   }

   float32_t x_temp2[N];        //Lokális tömb az FFT utáni komplex komponensek abszolútértékének kiszámításához
   float32_t x_temp3[2*N];      //Lokális tömb az FFt elvégzéséhez
   float32_t max_val;           //x_temp tömb maximális indexének értéke
   uint32_t max_ind;            //x_temp tömb maximális indexe

   //Hanning ablak inicializálása
   for(int iterator=0;iterator<N;iterator++)
   {
      float32_t cos=arm_cos_f32(2*3.14159265358979*iterator/N);
      w_f32[iterator]=0.5 - 0.5 * cos;
      xw_f32[iterator]=x_temp[iterator]*w_f32[iterator];
   }

   // x_temp3 tömb inicializálása FFT-hez (a komplex együtthatók valós jel lévén 0)
   for (int n = 0; n < N; n++)
      {
         x_temp3[2*n] = xw_f32[n];
         x_temp3[2*n + 1] = 0;
      }

   arm_cfft_f32(&arm_cfft_sR_f32_len128, x_temp3, 0, 1);    //128 elem hosszú FFT
   arm_cmplx_mag_f32(x_temp3, x_temp2, N);                  //FFT kimenetének abszolút értéke
   arm_max_f32((x_temp2+1), N - 1, &max_val, &max_ind);    //maximum keresés

   //Interpoláció
   if(x_temp2[max_ind-1]<x_temp2[max_ind+1])
   {
      dJ=(2*x_temp2[max_ind+1]-x_temp2[max_ind])/(x_temp2[max_ind+1]+x_temp2[max_ind]);
   }else
   {
      dJ=(x_temp2[max_ind]-2*x_temp2[max_ind-1])/(x_temp2[max_ind]+x_temp2[max_ind-1]);
   }
```

```
    Jk=(max_ind+dJ);     //FFT eredményének kompenzálása
    return Jk;
}


void ls()
{
    //x_f32 D mátrix elemeit tartalmazza
    arm_mat_init_f32(&x,N,3,(float32_t *)x_f32);

    //bemeneti jel
    arm_mat_init_f32(&y,N,1,(float32_t *)f_array);

    //x transzponáltja lesz
    arm_mat_init_f32(&xt,3,N,(float32_t *)xt_f32);

    //ideiglenes mátrixok a részeredmények tárolására
    arm_mat_init_f32(&tempm1,3,3,(float32_t *)tempm1_f32);
    arm_mat_init_f32(&tempm2,3,3,(float32_t *)tempm2_f32);
    arm_mat_init_f32(&tempm3,3,1,(float32_t *)tempm3_f32);
    arm_mat_init_f32(&tempm4,3,1,(float32_t *)tempm4_f32);

    //Pszeudoinverz kiszámítása
    status=arm_mat_trans_f32(&x,&xt);
    status=arm_mat_mult_f32(&xt, &x, &tempm1);
    status=arm_mat_inverse_f32 (&tempm1, &tempm2);

    //Rendszermátrix szorzása a mintavételezett pontok tömbjével
    status=arm_mat_mult_f32 (&xt, &y, &tempm3);
    status=arm_mat_mult_f32 (&tempm2, &tempm3, &tempm4);

    //Paraméterek gyűjtése egyéni változókba
    ls_A=tempm4.pData[0];
    ls_B=tempm4.pData[1];
    ls_C=tempm4.pData[2];
    fi = atan2f(-ls_B,ls_A);
    return;
}



void tls()
{
    /*Mátrix inicializáció*/
    arm_mat_init_f32(&ident,3,3,(float32_t *)ident_f32);
    arm_mat_init_f32(&tls_svd_trans,4,N,(float32_t *)tls_svd_trans_f32);
    arm_mat_init_f32(&tls_svd_temp1,4,4,(float32_t *)tls_svd_temp1_f32);
    arm_mat_init_f32(&beta,3,1,(float32_t *)beta_f32);
    arm_mat_init_f32(&tls_svd,N,4,(float32_t *)tls_svd_f32);
    arm_mat_init_f32(&tls_temp1,3,3,(float32_t *)tls_temp1_f32);
    arm_mat_init_f32(&tls_temp2,3,3,(float32_t *)tls_temp2_f32);
    arm_mat_init_f32(&tls_temp3,3,3,(float32_t *)tls_temp3_f32);
    arm_mat_init_f32(&tls_temp4,3,N,(float32_t *)tls_temp1_f32);


    /*[x,D] mátrix előállítása*/
    for(int iterator=0;iterator<N*4;iterator++)
    {
        if(iterator%4 == 0)
        {
            tls_svd_f32[iterator]=f_array[iterator_y];
            iterator_y++;
        } else
        {
            tls_svd_f32[iterator]=x_f32[iterator_x];
            iterator_x++;
```

```
    }
  }


  /*Kiszámolom ([x,D]'*[x,D]) mátrixot*/
  arm_mat_trans_f32(&tls_svd,&tls_svd_trans);
  arm_mat_mult_f32 (&tls_svd_trans, &tls_svd, &tls_svd_temp1);

  /*1. Mátrix elemeinek azonosítása az abc betűivel (ezzel csökkentem a leírt karakterek számát)*/
  a=tls_svd_temp1_f32[0];
  b=tls_svd_temp1_f32[1];
  c=tls_svd_temp1_f32[2];
  d=tls_svd_temp1_f32[3];
  e=tls_svd_temp1_f32[4];
  f=tls_svd_temp1_f32[5];
  g=tls_svd_temp1_f32[6];
  h=tls_svd_temp1_f32[7];
  i=tls_svd_temp1_f32[8];
  j=tls_svd_temp1_f32[9];
  k=tls_svd_temp1_f32[10];
  l=tls_svd_temp1_f32[11];
  m=tls_svd_temp1_f32[12];
  n=tls_svd_temp1_f32[13];
  o=tls_svd_temp1_f32[14];
  p=tls_svd_temp1_f32[15];

  A  = 1;
  B  = (0 - a - f - k - p);
  C  = (a*f - b*e + a*k - c*i + a*p - d*m + f*k - g*j + f*p - h*n + k*p - l*o);
  D  = (a*g*j - a*f*k + b*e*k - b*g*i - c*e*j + c*f*i - a*f*p + a*h*n + b*e*p - b*h*m - d*e*n + d*f*m - a*k*p +
a*l*o + c*i*p - c*l*m - d*i*o + d*k*m - f*k*p + f*l*o + g*j*p - g*l*n - h*j*o + h*k*n);
  E  = a*f*k*p - a*f*l*o - a*g*j*p + a*g*l*n + a*h*j*o - a*h*k*n - b*e*k*p + b*e*l*o + b*g*i*p - b*g*l*m -
b*h*i*o + b*h*k*m + c*e*j*p - c*e*l*n - c*f*i*p + c*f*l*m + c*h*i*n - c*h*j*m - d*e*j*o + d*e*k*n + d*f*i*o -
d*f*k*m - d*g*i*n + d*g*j*m;
  Q    = powf(A*(D*D)*(2.7E1/2.0)+(B*B)*E*(2.7E1/2.0)+C*C*C+sqrtf(powf(C*C+A*E*1.2E1-B*D*3.0,3.0)*-
4.0+powf(A*(D*D)*2.7E1+(B*B)*E*2.7E1+(C*C*C)*2.0-A*C*E*7.2E1-B*C*D*9.0,2.0))*(1.0/2.0)-
A*C*E*3.6E1-B*C*D*(9.0/2.0),1.0/3.0);
  if(powf(C*C+A*E*1.2E1-B*D*3.0,3.0)*-4.0+powf(A*(D*D)*2.7E1+(B*B)*E*2.7E1+(C*C*C)*2.0-
A*C*E*7.2E1-B*C*D*9.0,2.0)<0)
  {
    Q=pow(A*(D*D)*(2.7E1/2.0)+(B*B)*E*(2.7E1/2.0)+C*C*C-A*C*E*3.6E1-B*C*D*(9.0/2.0),1.0/3.0);
  }
  S=sqrtf(1.0/(A*A)*((B*B)*3.0-A*C*8.0)*(1.0/1.2E1)+(Q*(1.0/3.0)+((C*C)*(1.0/3.0)+A*E*4.0-
B*D)/Q)/A)*(1.0/2.0);

  /*A feltételes elágazásban szereplő kifejezés a számábrázolási pontatlanság miatt negatív, ezért korrigálni kellett*/
  if((1.0/(A*A)*((B*B)*3.0-A*C*8.0)*(1.0/4.0)-(S*S)*4.0-(1.0/(A*A*A)*((A*A)*D*8.0+B*B*B-
A*B*C*4.0)*(1.0/8.0))/S) <0)
  {
    debug1=0; //A gyökvonás eredményét egyenlőve teszem 0-val, a komplex sajátértékek elkerülése érdekében
  }
  /*Sajátértékek kiszámítása megoldóképlet segítségével*/
  szig[0]  =  -S+sqrtf(1.0/(A*A)*((B*B)*3.0-A*C*8.0)*(1.0/4.0)-(S*S)*4.0+(1.0/(A*A*A)*((A*A)*D*8.0+B*B*B-
A*B*C*4.0)*(1.0/8.0))/S)*(1.0/2.0)-(B*(1.0/4.0))/A;
  szig[1]  =  -S-sqrtf(1.0/(A*A)*((B*B)*3.0-A*C*8.0)*(1.0/4.0)-(S*S)*4.0+(1.0/(A*A*A)*((A*A)*D*8.0+B*B*B-
A*B*C*4.0)*(1.0/8.0))/S)*(1.0/2.0)-(B*(1.0/4.0))/A;
  szig[2] = S+debug1*(1.0/2.0)-(B*(1.0/4.0))/A;
  szig[3] = S-debug1*(1.0/2.0)-(B*(1.0/4.0))/A;

  /*Sajátértékek sorba rendezése*/

  for(int iii=0;iii<4;iii++)
  {
    for(int ii=0;ii<4;ii++)
    {
      if(szig[ii]<szig[ii+1])
      {
```

```
            sort_temp=szig[ii];
            szig[ii]=szig[ii+1];
            szig[ii+1]=sort_temp;
         }
      }
   }

   /*Mátrixszorzás helyett beírom szigma négyzet érékeit az identitás mátrixba*/
   ident_f32[0]=powf(szig[3],2);
   ident_f32[1]=0;
   ident_f32[2]=0;
   ident_f32[3]=0;
   ident_f32[4]=powf(szig[3],2);
   ident_f32[5]=0;
   ident_f32[6]=0;
   ident_f32[7]=0;
   ident_f32[8]=powf(szig[3],2);

   arm_mat_trans_f32(&x,&xt);
   arm_mat_mult_f32(&xt, &x, &tls_temp1);
   arm_mat_sub_f32(&tls_temp1,&ident,&tls_temp2);
   arm_mat_inverse_f32 (&tls_temp2, &tls_temp3);
   arm_mat_mult_f32 (&tls_temp3, &xt, &tls_temp4);
   arm_mat_mult_f32 (&tls_temp4, &y, &beta);

   tls_A=beta.pData[0];
   tls_B=beta.pData[1];
   tls_C=beta.pData[2];
   fi_tls = atan2f(-tls_B,tls_A);
   return;
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
   if(UART_data != 13 && UART_data != 10 && UART_data != 45)
   {
      buffer[uart_i++]=UART_data;
      if(UART_data == 46)
      {
         decimal = uart_i-1;
      }
   }else if(UART_data == 13)
   {
      for(int j=0;j<(decimal+4);j++)
      {
         if(j < decimal)
         {
            f_array[f_array_size] += ((float)buffer[j] - '0') * powf(10,(decimal - j)-1);
         }
         if(j > decimal)
         {
            f_array[f_array_size] += ((float)(buffer[j] - '0')) /( powf(10,(j-decimal)));
         }

      }
      f_array_size++;
      uart_i=0;
      if(sign_flag){
         f_array[f_array_size-1] =  (-1) * f_array[f_array_size-1];
      }
      sign_flag=0;
   } else if(UART_data == 45)
   {
      sign_flag=1;
   }else if(UART_data == 10)
   {
      data_ready = 1;
```

```
   }
HAL_UART_Receive_IT(&huart2,&UART_data,1);

}

// D mátrix előállítása
// pi=3.14159265358979
void init_D()
{
   iii=0;
   float32_t J_for_D;
   J_for_D=ipfft(f_array);
        for(int ii=0;ii<(N*3);ii=ii+3)
        {
           x_f32[ii]=arm_cos_f32(2*3.14159265358979*J_for_D/N*iii);
           x_f32[ii+1]=arm_sin_f32(2*3.14159265358979*J_for_D/N*iii);
           x_f32[ii+2]=1;
           iii++;
        }
}


/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  *
  * @retval None
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_ADC1_Init();

  /* Initialize interrupts */
  MX_NVIC_Init();
  /* USER CODE BEGIN 2 */
  HAL_UART_Receive_IT(&huart2,&UART_data,1);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
```

```
  if(data_ready==1)
        {
        init_D();

        ls();

        tls();
        }

    /*        data_ready=0;
           snprintf(TXbuffer, sizeof TXbuffer, "%2.10f \n %2.10f",m,b);
           HAL_UART_Transmit(&huart2, TXbuffer, sizeof TXbuffer,100);
        }
        */
 /* USER CODE END WHILE */

 /* USER CODE BEGIN 3 */

 }
 /* USER CODE END 3 */

}

/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{

 RCC_OscInitTypeDef RCC_OscInitStruct;
 RCC_ClkInitTypeDef RCC_ClkInitStruct;

  /**Configure the main internal regulator output voltage
  */
 __HAL_RCC_PWR_CLK_ENABLE();

 __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /**Initializes the CPU, AHB and APB busses clocks
  */
 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
 RCC_OscInitStruct.HSEState = RCC_HSE_ON;
 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
 RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
 RCC_OscInitStruct.PLL.PLLM = 6;
 RCC_OscInitStruct.PLL.PLLN = 180;
 RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
 RCC_OscInitStruct.PLL.PLLQ = 2;
 RCC_OscInitStruct.PLL.PLLR = 2;
 if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
 {
  _Error_Handler(__FILE__, __LINE__);
 }

  /**Activate the Over-Drive mode
  */
 if (HAL_PWREx_EnableOverDrive() != HAL_OK)
 {
  _Error_Handler(__FILE__, __LINE__);
 }

  /**Initializes the CPU, AHB and APB busses clocks
  */
 RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                             |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
 RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
```

```
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/**
  * @brief NVIC Configuration.
  * @retval None
  */
static void MX_NVIC_Init(void)
{
  /* USART2_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(USART2_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(USART2_IRQn);
}

/* ADC1 init function */
static void MX_ADC1_Init(void)
{

  ADC_ChannelConfTypeDef sConfig;

    /**Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)
    */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }
```

```
}

/* USART2 init function */
static void MX_USART2_UART_Init(void)
{

  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  file: The file name as string.
  * @param  line: The line in file as a number.
  * @retval None
  */
void _Error_Handler(char *file, int line)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  while(1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
```

```
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/**
  * @}
  */

/**
  * @}
  */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```