Neurális hálózatok házi feladat beszámoló

Pintér Bálint (I6QS0K), Szilágyi Gábor (NOMK01)

Koncentrált paraméterű RF szűrő optimalizációja
aktív tanulással

2023. június 9.

Tartalomjegyzék

1. Bevezetés **1**
1.1. Aktív tanulás 1
1.2. Az optimalizálandó probléma 1
1.3. A felhasznált könyvtár 2

2. A célfüggvény felírása **3**
2.1. A hálózat kapcsolási rajza 4
2.2. Impedanciák és admittanciák 6
2.3. Láncparaméterek 6
2.4. Szórási paraméterek 7

3. A célfüggvény javítása **7**
3.1. Az első verzió és a problémái 7
3.2. A paraméterek logaritmizálása 8
3.3. A függvényérték logaritmizálása 8
3.4. Büntetett margó 9

A. Az általunk írt specifikáció osztály **12**

1. Bevezetés

1.1. Aktív tanulás

A legtöbb neurális hálózatokat használó megoldás olyan problémára irányul, ahol sok rendelkezésre álló adat alapján kell a hálót betanítani egy feladat elvégzésére. Az a legtöbb esetben teljesül, hogy még több tanító adat felhasználásával jobb hálót lehetne tanítani, de ennek az extrém esetére tud megoldást nyújtani az aktív tanulás. A nem adathiányos problémáknál a rendelkezésre álló, címkézett adatpontok nagy részét felhasználva szokás tanítani a hálót, majd a fennmaradó adatpontokon ellenőrizni a háló teljesítőképességét olyan esetekre, amikkel nem találkozott a tanulás során. Az aktív tanulás folyamata ettől merőben eltér.

Aktív tanulás kiindulási helyzete, hogy nagyon sok címkézetlen adat áll rendelkezésre, de az egyes adatpontok címkézése rendkívül költséges. A címkézés költsége miatt végeredményben az a cél, hogy azt minél kevesebbszer kelljen elvégezni a tanulás során. A tanulási folyamat közben az eddig megkapott kevés címkézett adatpont alapján a háló jelöl ki következőnek címkézésre azt, amelyik várhatóan a leghasznosabb lesz számára. A hasznosság becslésére több megközelítés is létezik, erre a későbbiekben visszatérünk.

Az aktív tanulás egyik alesete a Bayes-optimalizáció. Itt nem egy osztályozót tanítunk minél kevesebb címkézett adat alapján, hanem egy „fekete doboz” függvény maximumát keressük a függvény minél kevesebb kiértékelése mellett. Ez a különbség már befolyásolni fogja a következőnek megcímkézendő adat választását, ami ebben az esetben a következő paraméterértékek megválasztását jelenti, ahol kiértékeljük a függvényt.

1.2. Az optimalizálandó probléma

Az optimalizálandó probléma egy koncentrált paraméterű rádiófrekvenciás (RF) szűrő elemeinek megadása, azaz méretezése. Egy koncentrált paraméterű elemekből felépülő szűrő

alapesetben ellenállásból, tekercsekből és kondenzátorokból áll, de léteznek aktív elemet, azaz erősítőt is tartalmazó szűrők – de ez már kívül esik a vizsgálatunkon, csak passzív eszközökkel foglalkozunk. Feltesszük továbbá, hogy a vizsgálatunk során csillapítással nem foglalkozunk, így ellenállást sem használunk a szűrőnkhöz.

Szűrőtervezéskor mindig egy előre megadott specifikációból indulunk ki, ebben elő van írva azon frekvenciatartományok csoportja, ahol a szűrni kell, illetve ahol csillapítás nélkül kell az RF teljesítményt áteresztení. Ennek megfelelően beszélhetünk záró és áteresztő sávokról, ahol a szűrő impedanciakarakterisztikájának rendre nagynak, illetve kicsinek kell lennie, ehhez a tekercsek és kondenzátorok soros és párhuzamos rezonanciáit használjuk ki. (S paraméter majd később).

A valóságban mind a tekercsek, mind a kondenzátorok 3D-s objektumok, induktivitásuk és kapacitásuk anyagparamétertől és geometriai méretektől függenek. Anyagparamétertől függés alatt a kondenzátorok dielektrikumában a relatív permittivitását, illetve a tekercsek belsejében a közeg relatív permeabilitását kell érteni – bár utóbbi nem jellemző RF szűrők esetében. Az anyagparaméter tehát egy fix érték, amit nem egyszerű változtatni, méretezés céljából ez nem járható út. Megoldást a geometriai méretek optimális megválasztása jelent, ami utat nyit az alapesetektől, az analitikus formulákkal egyszerűen kiszámolható geometriáktól való eltéréshez. Bár ezekben az alapesetekben, a síkkondenzátorban és az egyenes tekercsben, a szolenoidban is megjelennek geometriai jellemzők, a

$$C = \varepsilon_0 \varepsilon_r \frac{A}{d}$$
$$L = \mu_0 \mu_r \frac{N^2 A}{l}$$

formulák egyrészt csak közelítések, azaz nem írják le pontosan az L és a C értékeket, másrészt sokszor nem kivitelezhető az általuk leírt komponens.

A feladat innentől az, hogy olyan geometriai méreteket találjunk, amivel a komponensek pont az általunk kívánt L és C értékeket mutassák, ezzel a specifikációnak eleget tudjunk tenni. Esetünkben a nem analitikusan kiszámolható értékekhez numerikus megoldást kell keresni, amire egy jó megoldás, ha 3D véges elemes szimulációt készítünk. Egy-egy szimuláció futási ideje a megkövetelt precizitás és a rendelkezésünkre álló erőforrás függvényében változhat, de mindenképp több mint csupán egyetlen egyenlet kiértékelése. Éppen ezért szükséges az optimalizálást úgy végezni, hogy szempont legyen a minél kevesebbszer történű kiértékelés.

A fent leírtak az RF szűrőtervezés témakörében egy releváns problémát takarnak, munkánkban azonban nem 3D véges elemes szimulációkat használunk, nem geometriai paraméterek optimalizálását végezzük el. Tesszük mindezt egyrészt azért, mert a szimulációs szoftverek programozott vezérlése **kibaszott nehéz**, másrészt a neurális hálók szempontjából teljesen lényegtelen, hogy a kiértékelt függvényben mi történik. Azonban maga az optimalizálás amit csinálunk alkalmas egy ilyen feladatra, és érdemben tudja a tervezőt segíteni munkája során. Visszatérni tehát a koncentrált paraméterű, L és C elemeket tartalmazó hálózatra az optimalizáció szempontjából nem jelent kevesebb munkát, ebben az értelemben nem jár a feladat egyszerűsítésével.

A feladathoz már csak egy lépés hiányzik, amivel a problémát át tudjuk alakítani egy szélsőérték keresésre. Ennek az alapgondolata az, hogy az elvárt és a kapott kimenet különbségét vesszük figyelembe egy büntetőfüggvény segítségével. Magát az optimalizálást ezen keresztül tudjuk megtenni, ennek a függvénynek a kifejtésére a **??** részben kerül sor.

1.3. A felhasznált könyvtár

Bayes-optimalizáció megvalósításához már léteznek elkészített könyvtárak, így azt külön nem készítettük el, hanem felhasználtuk. Az általunk használt ingyenesen elérhető és letölthető *Bayesian Optimization* könyvtár más, neurális hálózatokra készített, ugyancsak szabadon hozzáférhető könyvtárakon alapszik, itt most ennek a bemutatását adjuk.

A *Bayesian Optimization* használata rendkívül egyszerű, összesen két dolgot kell megtennünk. Elsőként példányosítjuk a *BayesianOptimization* osztályt, ami bemeneti paraméterként megkapja a maximalizálandó költségfüggvényt, illetve a határait annak a tartománynak, ahol a maximumot keresnie kell. Fontos kiemelni, hogy esetünkben a kapott és az elvárt kimenetek közötti minimalizálás a feladat, így a költségfüggvényben át kell térni maximumhely keresésre.

A szélsőértékhely keresést ezután a *maximize( )* tagfüggvény hajtja végre. Ez a folyamat kezdetben random helyeken kiértékel pontokból indul el, majd iteratívan halad a modelltérben az optimum felé. Ennek megfelelően a *maximize( )* bemeneti paraméterei az inicializáló, véletlenszerű helyek száma és az iteráció száma. Ebben elsőként példányosul egy *UtilityFunction* nevű osztály, utána elindul az iteráció. Minden kör az előző lefutás utáni paraméterfrissítéssel, az *update_param( )*-mal kezdődik, ami a *UtilityFunction* egy tagfüggvénye. A függvénynek több felderítési típusa lehet, ezek a UCB (Upper Confidence Bounds method), az EI (Expected Improvement method) és a POI (Probability Of Improvement), alapesetben az UCB-t használjuk. Az *update_param( )*-nak három hiperparamétere van: χ, κ és κ_{decay}, az utóbbi a κ csökkentését végzi minden iterációban, de csak egy másik paraméter, κ_{delay} iterációtól kezdve. Frissítve a *UtilityFunction*-t, egy javaslattevő tagfüggvénynek, a *suggest( )*-nek adjuk át inputként, ami javasol egy új pontot, az *x_probe*-ot a következő kiértékelésre. Ebben a *suggest( )*-ben egy *argmax* vizsgálat történik egy akvizíciós függvényre, az *acq_max*-ra. Ez a vizsgálat két lépésből áll, elsőként egyenletes eloszlással felvesztünk mintavételi pontokat a *UtilityFunction*-ból a paramétertartomány határain belül, alapbeállításként 10^5 darabot. Második lépésként *L-BFGS-B* optimalizációs metódus futtatunk le minden mintavételezett pontra. Egy ciklusban kiválasztjuk a legnagyobbat, ezzel térünk vissza, ez lesz *x_probe*, a javasolt új kiértékelési hely. A *maximize( )* iterációs ciklusának harmadik lépése maga a kiértékelés, erre a *probe( )* tagfüggvény szolgál, ami az általunk megadott költségfüggvényt futtatja le a kiszámolt új *x_probe* helyen. Ezt a három iterációs lépést folytatja a kód a meghatározott iterációszámig.

Az *L-BFGS-B* optimalizáló solver nem ebben a könyvtárban van megírva, hanem a *scipy.optimizer* tartalmazza. Ez egy minimalizálási feladatokra készített másodrendű optimalizációs algoritmus, az *L-BFGS* kiterjesztése korlátok kezelésére. Feloldva a rövidítést a Limited-memory Broyden-Fletcher-Goldfarb-Shanno algoritmus egy kvázi-Newton módszer, másodrendű deriváltak közelítésére hasznos, ahol az közvetlenül nehéz kiszámolni. Konkretizálva ez azt jelenti, hogy nem számolja ki a Hesse-mátrixot, csak annak inverzét közelíti. A módszer nevében az L előtag a limitált memóriára utal, azaz mindig csak az elmúlt m lépés koordináta- és gradiensvektorát tárolja el. Ez a memóriagazdálkodás fontos, különösen az *acq_max* függvényben, ahol mind a 10^5 mintavételi pontra elvégezzük az *L-BFGS-B*-t.

2. A célfüggvény felírása

A célfüggvényünket olyan formában definiáltuk, hogy az adott specifikációnak jobban megfelelő szűrőparaméterekhez nagyobb számot rendeljen. A specifikációban frekvenciasávok és a hozzájuk elvárt minimális vagy maximális átvitel van megadva. Az átvitel itt a teljes kétportú (vagyis négypólusú) rendszer szórási mátrixának (S-mátrixának) egyik elemét, név szerint az S_{21}-et jelenti. Mivel ez a reaktív elemek jelenléte miatt egy komplex szám, ennek az abszolút értékét vizsgáljuk, a specifikációbeli korlátok is az abszolút értékre vonatkoznak. Az S_{21} paraméter egy frekvenciafüggő mennyiség, ami azt az információt hordozza, hogy ha az 1-es porton (kapun) beküldünk a szűrőbe egy adott amplitúdójú és frekvenciájú szinuszos feszültséghullámot, akkor az a rendszer 2-es portján milyen amplitúdójú és fázisú kimenő feszültséghullámot eredményez. Az S_{21} komplex fázisa a két feszültséghullám relatív fázisát fejezi ki, ez számunkra nem érdekes, ezért vesszük az abszolút értéket. Ha $|S_{21}|(f) = 1$, akkor a szűrő az adott f frekvencián nem csillapít, de nem is erősít, tehát tökéletesen átereszti a rajta keresztülküldött jelet, legfeljebb más fázisban. Ez lenne a szűrő ideális viselkedése áteresztő sávban. Ha pedig $|S_{21}|(f) = 0$, akkor a beküldött jelből semmit nem enged át, azt általános esetben vagy elnyeli, vagy visszaveri, de mivel mi veszteségmentes, LC szűrőkkel foglalkoztunk, itt az elnyelés nem jöhet szóba. Ez utóbbi pedig az ideális viselkedés záró sávban. Természetesen véges frekvencián és véges elemértékekkel egyik szélső eset sem megvalósítható, még az ideális LC komponensekkel sem, amelyekkel mi számolunk. A szűrő tehát mindig valamilyen 0 és 1 közötti $|S_{21}|$-et fog mutatni bármilyen frekvencián és bármilyen behelyettesített elemértékek mellett.

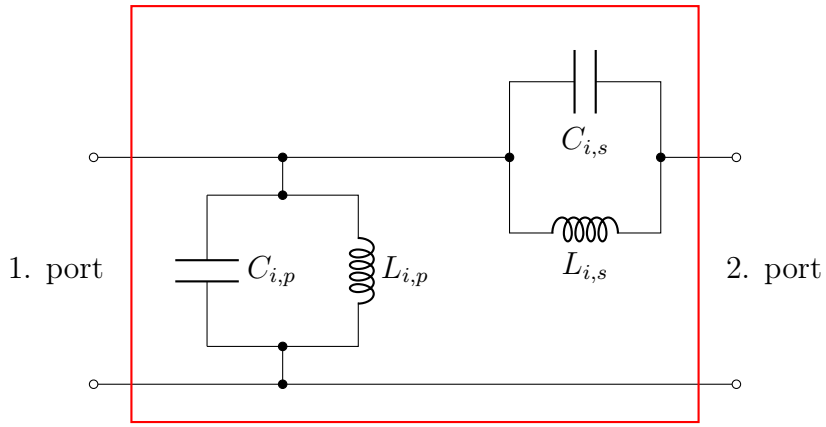
A célfüggvényünket, vagyis $c(\underline{x})$-t, ahol $\underline{x}$ a behelyettesített elemértékek vektora, egy összeggel definiáljuk. Az összegben diszkrét frekvenciákon felvett hibaértékeket, $h(f, \underline{x})$-ket adunk össze.

$$c(\underline{x}) = \sum_{i=1}^{N_f} h(f_i, \underline{x}) \tag{1}$$

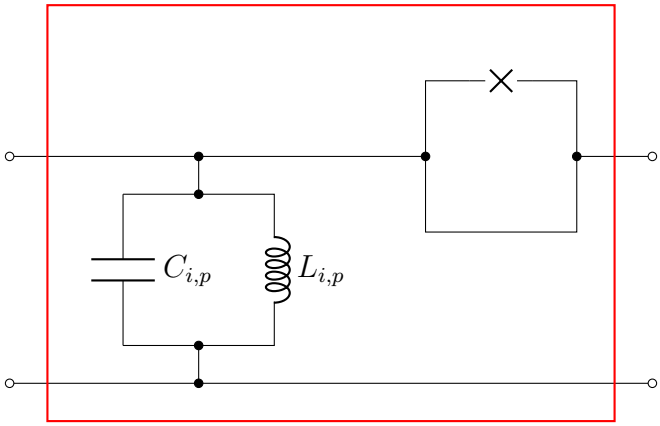
A vizsgált diszkrét f_i frekvenciákat logaritmikusan vettük fel a legkisebb és a legnagyobb specifikált frekvencia között, a kiértékelt frekvenciák száma, N_f megadható a specifikáció definiálásakor a programunkban. Ha egy adott frekvencián a kiszámolt $S(f) = |S_{21}|(f)$ kisebb, mint a megadott maximum, $S_{max}(f)$ vagy hasonlóan, ha nagyobb, mint a megadott minimum, $S_{min}(f)$, akkor a célfüggvény értékét nem változtatjuk az f_i frekvencián mutatott viselkedés alapján, itt nem büntetünk. Ez egyben azt is jelenti, hogy az adott diszkrét frekvencián megfelel a specifikációnak a szűrő. Ellenkező esetben minél jobban belóg az $S(f)$ a tiltott zónába, annál nagyobb abszolút értékű negatív szám lesz $h(f, \underline{x})$ értéke. A fent vázolt összegzős módszerrel egy olyan mennyiséget kapunk, ami az $S(f)$ átviteli függvény tiltott zónákba való belógásának integrálját közelíti. Hogy ez a belógás, vagyis a $h(f, \underline{x})$ pontosabban mit takar, később fejtjük ki a 3. fejezetben.

2.1. A hálózat kapcsolási rajza

A hálózatnak gyakorlatilag a kapcsolási rajza adott L és C elemértékekkel, innen kell valahogyan eljutni az $|S_{21}|$ paraméterig. Olyan struktúrájú szűrőt választottunk, amivel szűrőkarakterisztikák széles köre megvalósítható. Alapvetően egy periodikus szerkezetű kapcsolásról van szó, de az egyes blokkokban, amik egymás után vannak kapcsolva, külön meg lehet adni az elemértékeket, eleve azt is, hogy az adott pozícióban van-e kondenzátor vagy induktivitás. A programkódunk végleges verziójában legfeljebb 3 db ilyen blokk



1. ábra. Az optimalizálandó szűrő i-edik egysége

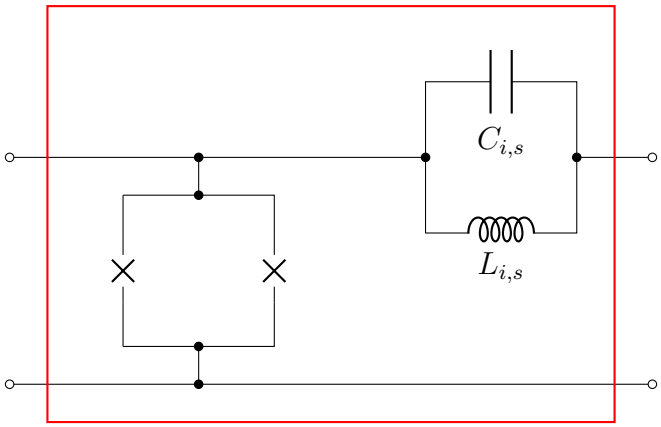


2. ábra. Az egységhálózat párhuzamosan kötött párhuzamos LC rezgőkörnek használva. Itt érdekesség, hogy a soros ágban egyik alkatrészt sem használjuk, mégis összesen egy rövidzárral vannak helyettesítve, mivel ennek a helyettesítésnek van értelme.

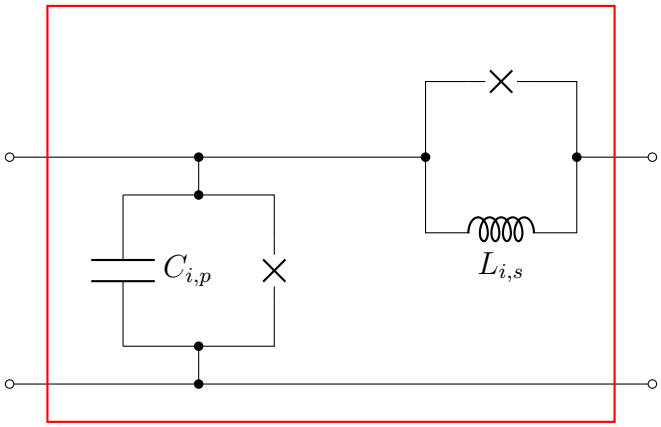
használható, de a kód minimális módosításával bővíthető még több blokkal szükség szerint. Egy ilyen blokk kapcsolási rajza az 1. ábrán látható. Ahogy az ábrán is látszik, az egységhálózat egy kétkapu, amiben egy párhuzamos és egy soros LC rezgőkör van. Ez csak abban az esetben néz ki pontosan így, ha mind a négy paraméterértéket beállítjuk. Ha egyes komponenseket nem használunk, akkor azokat szakadással, vagy rövidzárral helyettesíti a program, ahogy annak a teljes hálózat szempontjából értelme van. Erre láthatunk néhány példát a 2. 3. és 4. ábrán.

2.2. Impedanciák és admittanciák

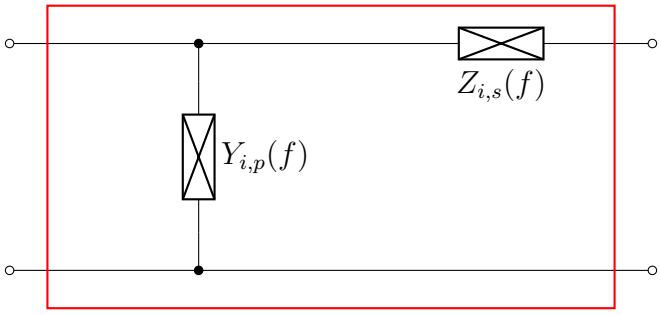
A célfüggvény számításának logikusan következő lépése, hogy az elemértékekből egy-egy diszkrét frekvencián helyettesítő impedanciát számolunk. A kódban egyébként az egyes párhuzamos rezgőkörökre direkte határozzuk meg annak az eredő impedanciát (vagy admittanciát) az *Impedance( )* és *Admittance( )* függvények segítségével. Ezek után a helyettesítőkép az 5. ábrán látható módon alakul. Itt a paraméterértékek az alábbi képletek alapján adódnak. Természetesen külön le vannak kezelve azok az esetek, amikor valamelyik impedancia vagy admittancia 0-ra vagy ∞-re adódik és nem lehet direkt behelyette-



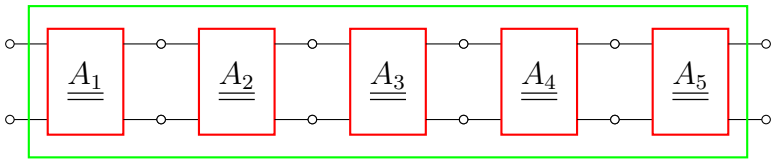
3. ábra. Az egységhálózat sorosan kötött párhuzamos LC rezgőkörnek használva.



4. ábra. Az egységhálózat elsőrendű LC aluláteresztő szűrőnek használva.



5. ábra. Helyettesítő impedancia és admittancia



6. ábra. A hálózat egységeinek kaszkád kapcsolása.

síteni a képletekbe.

$$Z_{i,p,L} = i2\pi f \cdot L_{i,p} \tag{2}$$

$$Z_{i,p,C} = \frac{1}{i2\pi f \cdot C_{i,p}} \tag{3}$$

$$Z_{i,s,L} = i2\pi f \cdot L_{i,s} \tag{4}$$

$$Z_{i,s,C} = \frac{1}{i2\pi f \cdot C_{i,s}} \tag{5}$$

$$Z_{i,s} = \frac{Z_{i,s,C} \cdot Z_{i,s,L}}{Z_{i,s,C} + Z_{i,s,L}} \tag{6}$$

$$Y_{i,p} = \frac{Z_{i,p,C} + Z_{i,p,L}}{Z_{i,p,C} \cdot Z_{i,p,L}} \tag{7}$$

2.3. Láncparaméterek

A következő lépés egy köztes számítás, ami az egyes egységblokkok kaszkádkapcsolásának számítását könnyíti meg. Ehhez a láncparamétereket hívjuk segítségül, amelyek az angol szakirodalomban ABCD-paraméterekként szerepelnek. A láncparaméterek mátrixa, vagyis $\underline{\underline{A}} \in \mathbb{C}^{2\times 2}$ lényegében a leírt rendszer két portján felvett feszültségek és áramok között teremt kapcsolatot, az impedanciaparaméterekhez (Z-paraméterek) nagyon hasonló módon. Egy adott frekvencián ezzel a 2×2-es mátrixszal teljesen leírható a hálózat. A feszültségek és áramok referenciairányai úgy vannak megválasztva a láncparaméterek felírásakor, hogy a kaszkádba kapcsolt kétportú hálózatok eredő láncparamétereit az összekapcsolt alegységek $\underline{\underline{A}}$ mátrixainak szorzataként kaphatjuk meg, ahogy az a 6. ábrán látható.

$$\underline{\underline{A}} = \prod_i \underline{\underline{A_i}} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \tag{8}$$

2.4. Szórási paraméterek

Végül elérkeztünk a szórási paraméterek számításához. A 2.3. pontban tárgyalt láncparaméterek használatát az is indokolja, hogy létezik egyszerű képlet az $\underline{\underline{A}}$ mátrixból $\underline{\underline{S}}$ mátrixba való áttérésre. Az $\underline{\underline{S}}$ mátrix minden elemére külön analitikus képlet van [1], de mivel nekünk csak az S_{21}-re van szükségünk, így csak az erre a konkrét paraméterre

vonatkozót kellett használnunk. Az S-paraméterek használatához minden portra definiálni kell egy normalizáló impedanciát, Z_0-t. Ezt mi a konvencionálisnak számító $50\,\Omega$-ra választottuk.

$$S_{21} = \frac{2}{A + B/Z_0 + CZ_0 + D} \tag{9}$$

3. A célfüggvény javítása

A $c(\underline{x})$ célfüggvény a félév során átesett néhány változtatáson. Ezt az indokolta, hogy a naívan, első próbálkozásként felírt célfüggvény több szempontból sem viselkedett jól az optimalizálás szempontjából. Ebben a fejezetben ezeket a fejlesztési lépéseket írjuk le.

3.1. Az első verzió és a problémái

Először a következőképpen definiáltuk a célfüggvényt felépítő $h(f, \underline{x})$ hibafüggvényt.

$$h(f, \underline{x}) = \begin{cases} \dfrac{S}{S_{min}} - 1 & \text{ha } S < S_{min} \\ 1 - \dfrac{S}{S_{max}} & \text{ha } S > S_{max} \\ 0 & \text{egyébként} \end{cases} \tag{10}$$

Erre a felírásra teljesül, amit fentebb célként vázoltunk, vagyis, hogy minél jobban átlóg valamelyik határon a függvény görbéje, annál jóbban büntetjük és az egyáltalán nem belógó függvényt nem büntetjük. A hibafüggvényt is változtattuk később, de tapasztalatunk szerint már akár ilyen formában is használható lenne. Ebben a formában a $c(\underline{x})$ függvénnyel a fő probléma, hogy az $\underline{x}$ vektorban egyszerűen a tekercsek induktivitásai és a kondenzátorok kapacitásai szerepeltek H-ben és F-ban. Ez magában még nem hangzik problémásnak, de ehhez azt is figyelembe kell vennünk, hogy az eredményes optimalizáláshoz *nagyságrendileg* elég széles tartományon belül kell megválasztania az optimalizáló algoritmusnak a következő kipróbálandó paraméterértéket. Az *Optimizer* objektum deklarálásakor meg kell adnunk a változtatható függvényparaméterek neveit és mindegyikhez azokat a határokat, amelyek közötti értékeket behelyettesíthet az optimalizálás során. Ezt a következőképpen lehet megadni a 4. ábrán vázolt elsőrendű aluláeresztő szűrő esetén, ha a kapacitás értékét 10^{-6}–$10^{-3}\,\mathrm{F}$ között, az induktivitás értékét pedig 10^{-5}–$10^{-2}\,\mathrm{H}$ között engedjük meg.

```
bounds = {'par1C': (1e-6, 1e-3), 'ser1L': (1e-5, 1e-2)}
optimizer = BayesianOptimization(
    pbounds=bounds,
    ...
)
```

A fent leírt alsó és felső határok egészen realisztikusak lehetnek, az optimalizációnak mégis nehezen tudja kezelni az ilyen eseteket. Ezt az okozza, hogy az intervallumok 0-hoz közeli végénél kis paraméterérték-változtatásra is nagyon jelentősen változik az átviteli karakterisztika, ez által a hibafüggvény és a célfüggvény értéke is. Más szóval a 0-közeli paraméterértékeknél nem tud az optimalizáló rátanulni a hirtelen nagyon megnövekvő parciális deriváltak miatt a célfüggvényre, pedig a feladat szempontjából ezek a kis abszolútértékű paraméterértékek nagyon is fontosak lennének.

Ez a probléma a kód futása során egyszerűen úgy jelentkezett, hogy ha mondjuk egy elemérték az 10^{-6}–10^{-2} tartományban változhatott, akkor az aptimalizáló egyáltalán nem is próbált ki olyan értékeket, amelyek körülbelül a 10^{-6}–10^{-4} tartományba esnek.

3.2. A paraméterek logaritmizálása

Az előző pontban vázolt, sok nagyságrendet átöltlő intervallumok problémájára találtunk megoldást. Ez gyakorlatilag annyi, hogy nem az elemértékeket adjuk meg a célfüggvény paramétereiként, hanem ezek (természetes alapú) logaritmusait. Ezek alapján majd a célfüggvény belül egyből visszaszámolja az eredeti elemértékeket és változatlanul számol tovább, csak az változik, hogy a célfüggvény összességében sokkal simább lesz és az optimalizáló jobban tudja kezelni. Ekkor a paraméter-határértékek megadása a következőképpen néz ki.

```
1 import math
2 bounds = {  'par1C': (math.log(1e-6), math.log(1e-3)),
3             'ser1L': (math.log(1e-5), math.log(1e-2))}
```

Ezzel az említett mintavételezési probléma megoldódott.

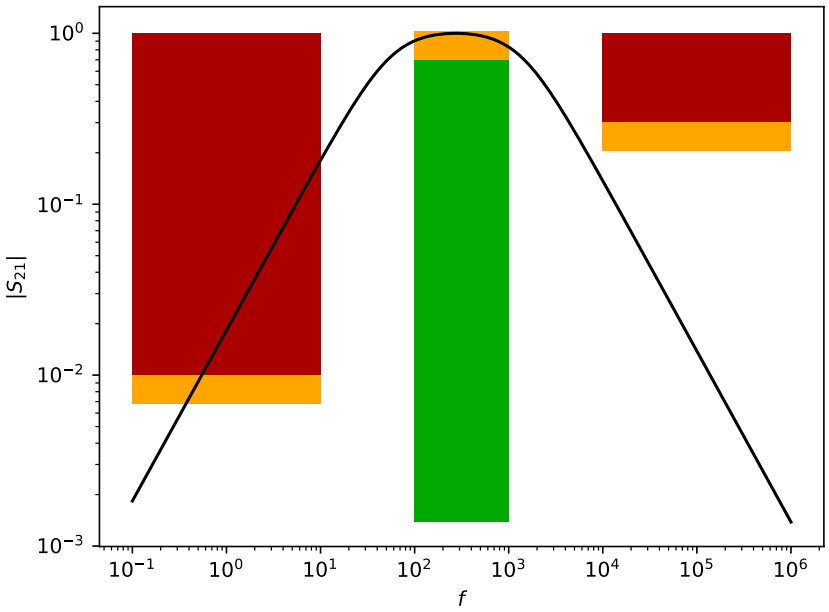
3.3. A függvényérték logaritmizálása

Egy másik, kevésbé jelentős problémára nyújtott megoldást a függvény kimenetének logaritmizálása. Ez egy konstans szorzó erejéig ekvivalens azzal, hogy dB-ben számoljuk ki az eltérést S_{max} és S_{min}, illetve az aktuális $S = |S_{21}|$ között. Ez a konstans szorzó a különböző logaritmus alapokból származik a dB skála és az általunk használt természetes alapú logaritmus között, de nincs jelentőssége, mivel ilyen jellegű változtatásra érzéketlen az optimalizálónk. A függvényérték logaritmizálásával azt értük el, hogy loglog skálán ábrázolva az átviteli függvényt és a tiltott régiókat ténylegesen görbe alatti terület jellegű lesz a célfüggvény.

3.4. Büntetett margó

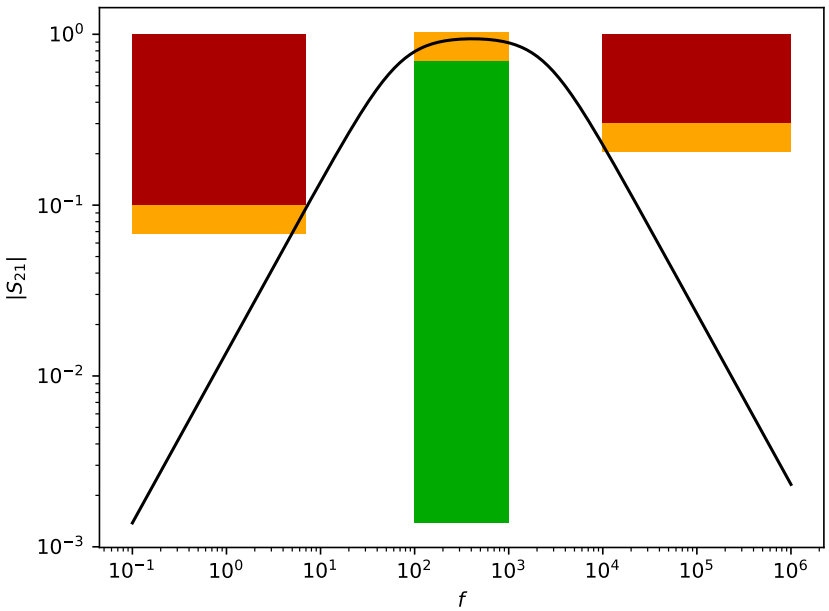
Az optimalizálás segítése érdekében bevezettünk egy opcionális `margin` nevű paramétert, amit a `Spec` objektum példányosításakor kell megadni. Ez egy további büntetett régiót ad az áteresztő sávok tiltott régiója fölé, illetve a záró sávok tiltott régiója alá. Ezzel effektíve azt érjük el, mintha ennyivel szigorúbbak lennének a határértékeink. A büntetett margót a 7. ábra szemlélteti. Az ábrán a bal oldali (kisebb frekvenciákhoz tartozó) záró sávban jól láthatóan az eredeti specifikáció által definiált tiltott régióba is belelóg az átviteli függvény. Emiatt természetesen ennek a sávnak nemnulla hozzájárulása van a célfüggvényhez. Az áteresztő sávban az eredeti specifikációnak ugyan megfelel az átviteli függvény, de a margóba még belóg, így itt is nemnulla büntetés adódik. A jobb oldali záró sávnak még a margóját is elkerüli az átviteli függvény, így ennek a sávnak 0 a célfüggvényhez a hozzájárulása.

A margó bevezetését az indokolta, hogy jobban fel lehessen oldani az olyan helyzeteket, amikor az átviteli függvény ugyan teljesíti az eredeti specifikációt, de egyes frekvenciákon csak éppen, mivel ilyenkor valós alkalmazásoknál érdemes valamennyi tartalékot rászámolni a tervezésnél. Ilyenkor a margó lehetővé teszi, hogy nagyjából egyforma mértékben közelítse meg a határértékeket az átviteli függvény, ne pedig az egyiknél nagyon nagy tartalékkal, a másiknál pedig kis tartalékkal haladjon el. A margó célszerű használatakor olyan megoldás születik, amikor a megfelel a hálózat a specifikációnak, de a legtöbb



7. ábra. Három specifikált frekvenciasáv, két záró (piros) és egy áteresztő (zöld) sáv, valamint a büntetett margó használata.

sávhoz tartozó margóba kicsivel belelóg. Erre mutat példát a 8. ábra, még ha csak egy nagyon egyszerű esetre is, amit az optimalizáló algoritmus talált meg.



8. ábra. A margó ideális használata.

Hivatkozások

[1] Láncparaméter – S-paraméter konverzió (elérve 2023.06.07.): https://www.rfwireless-world.com/Terminology/abcd-matrix-vs-s-matrix.html

[2] BayesianOptimization python könyvtár GitHub oldala. (Elérve: 2023.06.07.) https://github.com/bayesian-optimization/BayesianOptimization

A. Az általunk írt specifikáció osztály

```
import numpy as np
import math
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from matplotlib.backends.backend_pdf import PdfPages

pi = math.pi
inf = math.inf
nan = math.nan

#ipmedance of the series LC subcircuit
def Impedance(f, lnC, lnL):
    match [math.isnan(lnC), math.isnan(lnL)]:
        case [True, True]:
            return 0.0
        case [True, False]:
            L = math.exp(lnL)
            return f*1j*2*pi*L
        case [False, True]:
            C = math.exp(lnC)
            return -1j*1/(f*2*pi*C)
        case _:
            C = math.exp(lnC)
            L = math.exp(lnL)
            ZC = -1j*1/(f*2*pi*C)
            ZL = 0+f*1j*2*pi*L
            return ZL*ZC/(ZL+ZC)

def Admittance(f, lnC, lnL):
    match [math.isnan(lnC), math.isnan(lnL)]:
        case [True, True]:
            return 0.0
        case [True, False]:
            L = math.exp(lnL)
            return 1/(f*1j*2*pi*L)
        case [False, True]:
            C = math.exp(lnC)
            return f*2*pi*C*1j
        case _:
            C = math.exp(lnC)
            L = math.exp(lnL)
            ZC = -1j*1/(f*2*pi*C)
            ZL = 0+f*1j*2*pi*L
            return (ZL+ZC)/ZL*ZC

class Spec:
    def __init__(self, starts, ends, limits, directions, margin, n):
        # starts of pass- or stopbands in Hz
        self.starts = starts
        # ends of start- or stopbands in Hz
        self.ends = ends
        # pass- or stopband threshold values
        self.limits = limits
        # pass- or stopbands? possible values: "pass" or "stop" strings
        self.directions = directions
        # number of frequency points
        self.n = n
        # margin to still punish solution that only barely satisfies the specification
        # on stopbands, with limit l and margin m, the margin range is from l*(1-m) to l
        # on passbands, with limit l and margin m, the margin range is from l to l*(1+m)
        self.margin = margin

    #def cost(self, types, values):
    def cost(self, plot=False, par1C=nan, par1L=nan, ser1C=nan, ser1L=nan, par2C=nan,
    par2L=nan, ser2C=nan, ser2L=nan, par3C=nan, par3L=nan, ser3C=nan, ser3L=nan):
        """Parameters define a ladder structure,
        where each series or parallel element consists of
        two discrete, ideal L or C components in parallel
        with each other. Default values define a perfect
        all-pass filter. In the function parameters, "parXY" means
```

```
        the value of the Y-th sub-component of the X-th parallel
        element. Similarly, "serXY" means the value of the Y-th
        sub-component of the X-th series element. The ladder starts
        with a parallel element."""
        values = [par1C, par1L, ser1C, ser1L, par2C, par2L, ser2C, ser2L, par3C, par3L,
    ser3C, ser3L]
        nval = len(values)
        nladder = int(nval/4)
        minf = (min(self.starts))
        maxf = (max(self.ends))
        # frequency axis sample points (log spacing)
        faxis = []
        # no. of frequency sample points
        #n = self.n
        factor = (maxf/minf)**((1/self.n))
        for i in range(self.n+1):
            faxis.append(minf*factor**(i))
        # reference impedance in Ohm, on both ports
        Z0 = 50
        Y0 = 1/Z0
        # array of overall S21 values at the frequency sample points
        S21 = []
        #process 1 parallel and 1 series element:
        for f in faxis:
            # ABCD parameter matrix of the whole system
            ABCD = np.matrix([[1,0],[0,1]])
            for l in range(nladder):
                # admittance of the two parallel components together from the current
    step of the ladder
                Y = Admittance(f, values[4*l+0], values[4*l+1])
                mPar = np.matrix([[1, 0],[Y, 1]])
                Z = Impedance(f, values[4*l+2], values[4*l+3])
                mSer = np.matrix([[1, Z],[0, 1]])
                ABCD = ABCD*mPar*mSer
            # 2/(A+B/Z0+C*Z0+D)
            A = ABCD.item(0,0)
            B = ABCD.item(0,1)
            C = ABCD.item(1,0)
            D = ABCD.item(1,1)
            S21.append(abs(2/(A+B/Z0+C*Z0+D)))
        # natural log of margin+1
        lnmargin=math.log(self.margin+1)
        if(plot):
            # green: passband; red: stopband; orange: ok, but close to not ok, still
    punished
            fig, ax = plt.subplots()
            minS21 = min(S21)
            for i in range(len(self.starts)):
                if(self.directions[i] == "pass"):
                    # region forbidden by the original specification
                    ax.add_patch(Rectangle((self.starts[i], minS21),
                                           self.ends[i]-self.starts[i],
                                           self.limits[i]-minS21,
                                           facecolor='#00aa00'))
                    # region close to original limit, but satisfying it, additional
    penalty region
                    ax.add_patch(Rectangle((self.starts[i], self.limits[i]),
                                           (self.ends[i]-self.starts[i]),
                                           self.limits[i]*self.margin,
                                           facecolor='orange'))
                else: # "stop"
                    # region forbidden by the original specification
                    ax.add_patch(Rectangle((self.starts[i], self.limits[i]),
                                           self.ends[i]-self.starts[i],
                                           1.0-self.limits[i],
                                           facecolor='#aa0000'))
                    # region close to original limit, but satisfying it, additional
    penalty region
                    ax.add_patch(Rectangle((self.starts[i], self.limits[i]*(1/(1+self.
    margin))),
                                           (self.ends[i]-self.starts[i]),
                                           self.limits[i]*(1-1/(1+self.margin)),
                                           facecolor='orange'))
```

```
            ax.loglog(faxis, S21, 'k-')
            plt.ylabel(r'$|S_{21}|$')
            plt.xlabel(r'$f$')
            plt.savefig("plot.pdf", dpi=120, format='pdf', bbox_inches='tight')
            plt.show()
        cost = 0
        # number of freq points in the regions where the S21 is specified
        ncost = 0
        for i in range(len(self.starts)):
            for j in range(len(faxis)):
                if faxis[j]>self.starts[i] and faxis[j]<self.ends[i]:
                    ncost = ncost + 1
                    lnlimit = math.log(self.limits[i])
                    lnS21 = math.log(S21[j])
                    if self.directions[i] == "pass":
                        cost += max(0, min(-lnS21, lnlimit+lnmargin-lnS21))
                    else: # "stop"
                        cost += max(0, lnS21-lnlimit+lnmargin)
    # negative of average cost, for function maximizing
        return -cost/ncost
```