

Neurális hálózatok házi feladat beszámoló

Pintér Bálint (I6QS0K), Szilágyi Gábor (NOMK01)

Koncentrált paraméterű RF szűrő optimalizációja aktív tanulással

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 esete 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 elemek, 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 szűrőnkhez.

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 áteresztetni. 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 történő 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 szélsőérték hely keresést ezután a *maximize()* tagfüggvény hajtja végre. Ez a folyamat kezdetben random helyeken kiértékel pontokból indul el, majd iteratíván halad a modellterben 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 feszültség hullámot, akkor az a rendszer 2-es portján milyen amplitúdójú és fázisú kimenő feszültség hullámot eredményez. Az S_{21} komplex fázisa a két

feszültség hullám 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 átereszt 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 $c(\mathbf{x})$ -t, ahol \mathbf{x} a behelyettesített elemértékek vektora, egy összeggel definiáljuk. Az összegben diszkrét frekvenciákon felvett hibaértékeket, $h(f, \mathbf{x})$ -ket adunk össze.

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

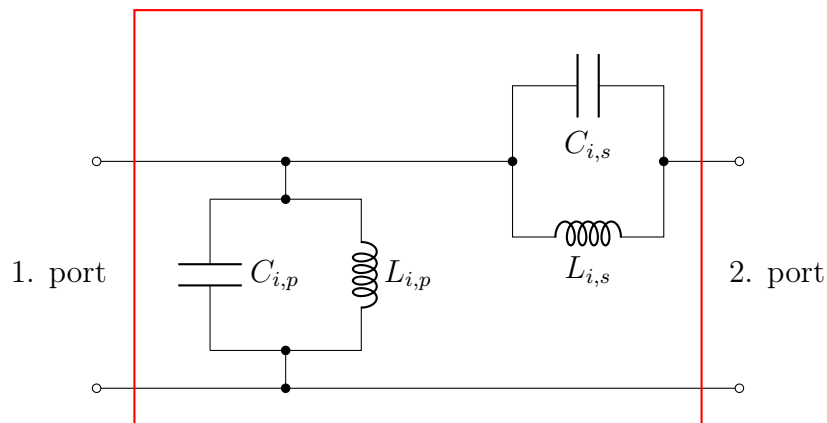
A vizsgált diszkrét f_i frekvenciákat logaritmikusan vettük fel a legkisebb és a legnagyobb specifikált frekvencia között, a kiértékelt frekvenciák száma, N_f megadható a specifikáció definiálásakor a programunkban. Ha egy adott frekvencián a kiszámolt $S(f) = |S_{21}|(f)$ kisebb, mint a megadott maximum, $S_{max}(f)$ 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 $h(f, \mathbf{x})$ 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 $h(f, \mathbf{x})$ pontosabban mit takar, később fejtjük ki a 3. fejezetben.

2.1. A hálózat kapcsolási rajza

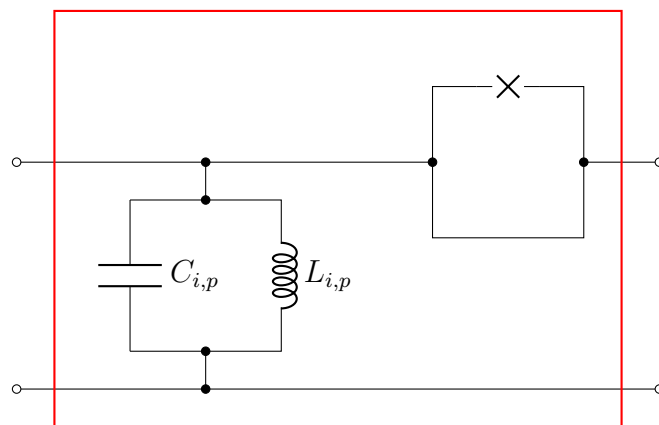
A hálózatnak gyakorlatilag a kapcsolási rajza adott L és C elemértékekkel, innen kell valahogyan eljutni az $|S_{21}|$ paraméterig. Olyan struktúrájú szűrőt választottunk, amivel szűrőkarakterisztikák széles köre megvalósítható. Alapvetően egy periodikus szerkezetű kapcsolásról van szó, de az egyes blokkokban, 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 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 egység hálózat egy kétkapú, 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

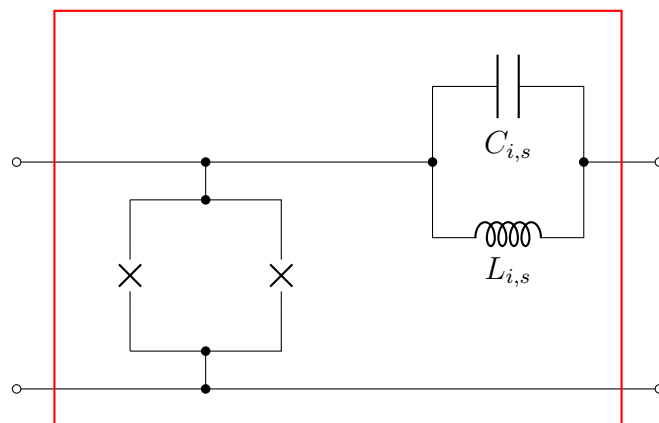
A célfüggvény számításának logikusan következő lépése, hogy az elemértékekből egy-egy diszkrét frekvencián helyettesítő impedanciát számolunk.



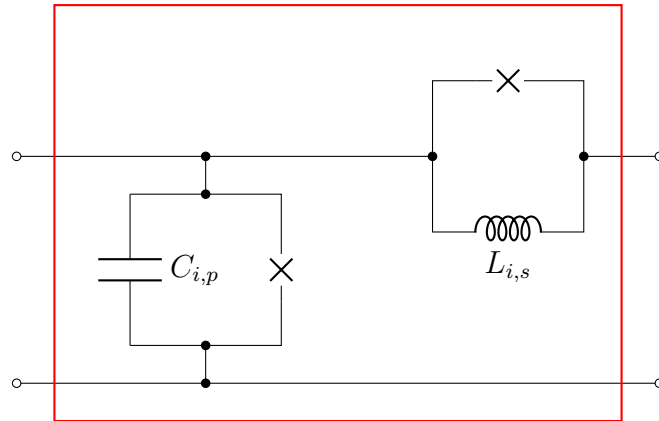
1. ábra. Az optimalizálható szűrő i -edik egysége



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



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



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

2.3. Láncparaméterek

2.4. Szórási paraméterek

3. A célfüggvény javítása

3.1. Az első verzió és a problémái

3.2. A paraméterek logaritmizálása

3.3. A függvényérték logaritmizálása

A. Az általunk írt specifikáció osztály

```
1 import numpy as np
2 import math
3 import matplotlib.pyplot as plt
4 from matplotlib.patches import Rectangle
5 from matplotlib.backends.backend_pdf import PdfPages
6
7 pi = math.pi
8 inf = math.inf
9 nan = math.nan
10
11 #impedance of the series LC subcircuit
12 def Impedance(f, lnC, lnL):
13     match [math.isnan(lnC), math.isnan(lnL)]:
14         case [True, True]:
15             return 0.0
16         case [True, False]:
17             L = math.exp(lnL)
18             return f*1j*2*pi*L
19         case [False, True]:
20             C = math.exp(lnC)
21             return -1j*1/(f*2*pi*C)
22         case _:
23             C = math.exp(lnC)
24             L = math.exp(lnL)
25             ZC = -1j*1/(f*2*pi*C)
26             ZL = 0+f*1j*2*pi*L
27             return ZL*ZC/(ZL+ZC)
28
29 def Admittance(f, lnC, lnL):
30     match [math.isnan(lnC), math.isnan(lnL)]:
31         case [True, True]:
32             return 0.0
33         case [True, False]:
34             L = math.exp(lnL)
35             return 1/(f*1j*2*pi*L)
36         case [False, True]:
37             C = math.exp(lnC)
38             return f*2*pi*C*1j
39         case _:
40             C = math.exp(lnC)
41             L = math.exp(lnL)
42             ZC = -1j*1/(f*2*pi*C)
43             ZL = 0+f*1j*2*pi*L
44             return (ZL+ZC)/ZL*ZC
45
46 class Spec:
47     def __init__(self, starts, ends, limits, directions, margin, n):
48         # starts of pass- or stopbands in Hz
49         self.starts = starts
50         # ends of start- or stopbands in Hz
51         self.ends = ends
52         # pass- or stopband threshold values
53         self.limits = limits
54         # pass- or stopbands? possible values: "pass" or "stop" strings
55         self.directions = directions
56         # number of frequency points
57         self.n = n
58         # margin to still punish solution that only barely satisfies the specification
59         # on stopbands, with limit l and margin m, the margin range is from l*(1-m) to l
60         # on passbands, with limit l and margin m, the margin range is from l to l*(1+m)
61         self.margin = margin
62
63     #def cost(self, types, values):
64     def cost(self, plot=False, par1C=nan, par1L=nan, ser1C=nan, ser1L=nan, par2C=nan,
65             par2L=nan, ser2C=nan, ser2L=nan, par3C=nan, par3L=nan, ser3C=nan, ser3L=nan):
66         """Parameters define a ladder structure,
67         where each series or parallel element consists of
68         two discrete, ideal L or C components in parallel
69         with each other. Default values define a perfect
70         all-pass filter. In the function parameters, "parXY" means
```

```

70     the value of the Y-th sub-component of the X-th parallel
71     element. Similarly, "serXY" means the value of the Y-th
72     sub-component of the X-th series element. The ladder starts
73     with a parallel element.""
74     values = [par1C, par1L, ser1C, ser1L, par2C, par2L, ser2C, ser2L, par3C, par3L,
ser3C, ser3L]
75     nval = len(values)
76     nladder = int(nval/4)
77     minf = (min(self.starts))
78     maxf = (max(self.ends))
79     # frequency axis sample points (log spacing)
80     faxis = []
81     # no. of frequency sample points
82     #n = self.n
83     factor = (maxf/minf)**((1/self.n))
84     for i in range(self.n+1):
85         faxis.append(minf*factor**(i))
86     # reference impedance in Ohm, on both ports
87     Z0 = 50
88     Y0 = 1/Z0
89     # array of overall S21 values at the frequency sample points
90     S21 = []
91     #process 1 parallel and 1 series element:
92     for f in faxis:
93         # ABCD parameter matrix of the whole system
94         ABCD = np.matrix([[1,0],[0,1]])
95         for l in range(nladder):
96             # admittance of the two parallel components together from the current
step of the ladder
97             Y = Admittance(f, values[4*l+0], values[4*l+1])
98             mPar = np.matrix([[1, 0],[Y, 1]])
99             Z = Impedance(f, values[4*l+2], values[4*l+3])
100             mSer = np.matrix([[1, Z],[0, 1]])
101             ABCD = ABCD*mPar*mSer
102             # 2/(A+B/Z0+C*Z0+D)
103             A = ABCD.item(0,0)
104             B = ABCD.item(0,1)
105             C = ABCD.item(1,0)
106             D = ABCD.item(1,1)
107             S21.append(abs(2/(A+B/Z0+C*Z0+D)))
108     # natural log of margin+1
109     lnmargin=math.log(self.margin+1)
110     if(plot):
111         # green: passband; red: stopband; orange: ok, but close to not ok, still
punished
112         fig, ax = plt.subplots()
113         minS21 = min(S21)
114         for i in range(len(self.starts)):
115             if(self.directions[i] == "pass"):
116                 # region forbidden by the original specification
117                 ax.add_patch(Rectangle((self.starts[i], minS21),
118                                         self.ends[i]-self.starts[i],
119                                         self.limits[i]-minS21,
120                                         facecolor='#00aa00'))
121                 # region close to original limit, but satisfying it, additional
penalty region
122                 ax.add_patch(Rectangle((self.starts[i], self.limits[i]),
123                                         (self.ends[i]-self.starts[i]),
124                                         self.limits[i]*self.margin,
125                                         facecolor='orange'))
126             else: # "stop"
127                 # region forbidden by the original specification
128                 ax.add_patch(Rectangle((self.starts[i], self.limits[i]),
129                                         self.ends[i]-self.starts[i],
130                                         1.0-self.limits[i],
131                                         facecolor='#aa0000'))
132                 # region close to original limit, but satisfying it, additional
penalty region
133                 ax.add_patch(Rectangle((self.starts[i], self.limits[i]*(1/(1+self.
margin))),
134                                     (self.ends[i]-self.starts[i]),
135                                     self.limits[i]*(1-1/(1+self.margin)),
136                                     facecolor='orange'))

```



```

137         ax.loglog(faxis, S21, 'k-')
138         plt.ylabel(r'$|S_{21}|$')
139         plt.xlabel(r'$f$')
140         plt.savefig("plot.pdf", dpi=120, format='pdf', bbox_inches='tight')
141         plt.show()
142     cost = 0
143     # number of freq points in the regions where the S21 is specified
144     ncost = 0
145     for i in range(len(self.starts)):
146         for j in range(len(faxis)):
147             if faxis[j]>self.starts[i] and faxis[j]<self.ends[i]:
148                 ncost = ncost + 1
149                 lnlimit = math.log(self.limits[i])
150                 lnS21 = math.log(S21[j])
151                 if self.directions[i] == "pass":
152                     cost += max(0, min(-lnS21, lnlimit+lnmargin-lnS21))
153                 else: # "stop"
154                     cost += max(0, lnS21-lnlimit+lnmargin)
155     # negative of average cost, for function maximizing
156     return -cost/ncost

```