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

Áramkörszimulációs modellek transzformálása MATLAB számítási modellé

TDK dolgozat

Készítette:

Weisz Pál

Konzulens:

Dr. Orosz György

2023

Tartalomjegyzék

Áramkörszimulációs modellek transzformálása MATLAB számítási modellé

Biztonságkritikus rendszerek fejlesztési folyamata során fontos lépés a megtervezett termék analízise, mely során az elméletileg helyes működésről bizonyosodhatnak meg a fejlesztők. Egy ilyen módszer a worst-case áramköranalízis, mely az adott áramkör viselkedésének vizsgálatára szolgál az azt felépítő alkatrészek paramétereinek toleranciája és gyártási szórása, különböző környezeti hatások, hibajelenségek mellett.

Worst-case áramköranalízis tipikusan áramkörszimulációs szoftverek segítségével végezhető, azonban ezek a programok nem minden esetben nyújtanak gyors és hatékony megoldást bizonyos számítások elvégzésére. Előfordulhat például, hogy egy-egy probléma analitikus formában felírható, emiatt gyorsabban megoldható lenne, mint a numerikus algoritmusokat alkalmazó szimulátorokkal. Hasonlóképpen, a folyamat párhuzamosításával optimalizálható ugyanazon az áramkörön elvégzett szimuláció többszöri megismétlése eltérő paraméterek használatával. Felmerült továbbá az igény, hogy az áramkörmodell viselkedésének vizsgálatán túl haladó matematikai funkciókat is lehessen rajta alkalmazni egy arra optimalizált környezetben.

A folyamat felgyorsítása céljából egy olyan szoftveres keretrendszer kidolgozásával foglalkoztam, mely segítségével az LTspice áramkörszimulációs programban készített kapcsolási rajzok áramkörmodelljei komponensenként lebontva újraalkothatók MATLAB-ban, további hatékonyabb számítási feladatok elvégzéséhez, mintegy interfészt szolgáltatva a két program között. A technikai fejlesztéshez hozzátartozik az LTspice alkatrészmodellek feldolgozása, a modelleket tároló adatstruktúrák formátumának kialakítása, és magát a teljes áramkört leíró netlista fájl transzformálása MATLAB-ban hálózati egyenletek rendszerévé. Ezek aztán különböző számítási módszerekkel megoldhatók a szimuláció típusától, az alkalmazott analitikai eljárástól és a megoldó algoritmustól függően.

Transformation of circuit simulation models into MATLAB computational models

During the development process of safety-critical systems, the analysis of the designed product is an important step to ensure that the product is operating properly indeed. One such method is called worst-case circuit analysis, which is used to investigate the effects of component parameter tolerances and environmental conditions on the behaviour of the circuit.

Worst-case circuit analysis is typically performed using a circuit simulation software, but these programs do not always provide a fast and efficient solution to perform certain calculations. For example, it may be possible to describe a problem in analytical form and therefore solve it more quickly than with simulators using numerical algorithms. Similarly, by parallelising the process, it is possible to optimise the repetition of simulations done on the same circuit using different parameters. It has also become necessary to be able to apply advanced mathematical functions to the circuit model, beyond its behaviour, in an optimised environment.

In order to speed up the process, I developed a software framework to transform LTspice models of the circuit diagrams component by component in MATLAB, to perform further more efficient computational tasks, providing an interface between the two programs. The technical development involves processing the LTspice component models, formatting its data structure, and transforming the netlist file describing the complete circuit itself into a system of network equations in MATLAB. These equations can then be solved using different computational methods depending on the type of simulation, the analytical procedure used or the desired solver algorithm.

1. fejezet

Bevezetés

1.1. Áttekintő

Biztonságkritikus rendszerek esetén a hibázás, valamint az előírt specifikációk be nem tartása balesethez, környezeti kárhoz vezethet, ezért az ilyen alkalmazásokat fokozott körültekintéssel tervezik meg és verifikálják az iparban. Tervezéskor különböző szabványoknak, előírásoknak megfelelő módszereket alkalmaznak, verifikáció során pedig a rendszertervezéskor definiált teszteseteket állítanak elő, melyekkel a rendszer működését koponensenként, modulonként, majd az integrációt követően is vizsgálják.

Villamosmérnöki területen a modell-alapú tervezésre jellemző, hogy a megtervezett hardvert vagy hardver-modulokat különböző szintű teszteknek vetik alá. Költséghatékonysági szempontból célszerű, ha kezdetben nem rögtön a fizikailag elkészített áramkörrel próbálnak ki potenciálisan veszélyes helyzeteket, melyek a prototípus áramkör meghibásodását és cseréjét okozhatják. Helyette annak modelljével, számítógép segítségével egy szimulált környezetben végzik el a teszteket, és csak azt követően készül fizikailag megvalósított prototípus az áramkörről, amikor az összes előírt esetben hibátlanul működik a modell.

Többféle analízis módszer is használható az alkalmazástól és a tesztesetek típusától függően, dolgozatomban az áramkörök worst-case analízisével foglalkozom. A worst-case analízis használata esetén arra vagyunk kíváncsiak, hogy az adott áramkör működése hogyan függ az azt felépítő alkatrészek paramétereitől, különböző környezeti hatásoktól, és meghibásodási módoktól. Az ilyen jellegű tesztek során kifejezetten olyan szélsőséges eseményeket valamint azok okait keressük, ahol az áramkör működése a lehető legjobban eltér a specifikációnak megfelelő előírásoktól. Ezzel számszerűsíthető korlát adódik a körülményekhez tartozó legkevésbé kedvező forgatókönyvre, amivel később rendszerszinten számolhatnak a tervezők.

Az áramkör viselkedését befolyásoló tényezők szélsőérték-keresése nem triviális feladat, főleg komplexebb kapcsolások esetén, hiszen egyszerre több paraméter is hatással van az áramkör működésére.

1.2. Módszerek

Worst-case áramköranalízis feladatok megoldására számos lehetőség kínálkozik: akár szimulációk, akár rendszeregyenletek analitikus vagy numerikus megoldásával vizsgálhatjuk az áramkör viselkedését. Ezen módszerek egyike az extrémérték-elmélet (Extreme Value Analysis, EVA)[2] alkalmazásán alapul. Lényege, hogy a komplex felépítésű áramköröket komponensekre partícionálják, és a hozzájuk tartozó paraméterek lehetséges szélsőértékeit külön-külön egyesével behelyettesítve végzik el az áramkör szimulációját. Belátható, hogy N paraméterszám esetén ez $\mathcal{O}(2^N)$ számítást igényel, azaz exponenciálisan nő a szimuláció futásideje is. Bizonyos esetekben megsejthető, hogy az adott paraméter milyen értékénél fog legkedvezőtlenebbül működni az áramkör, de ez az esetek jelentős részében nem magától értetődő. Ráadásul ha a vizsgált komponens által produkált viselkedés, válasz a paramétertől nem monoton módon függ, a szélsőérték megállapítása szintén nehézségekbe ütközik.

Egy másik lehetséges módszer a Monte-Carlo szimuláció alkalmazása. Ez numerikus statisztikai eredményt ad az áramkör vizsgált válaszára vonatkozóan. A paramétereket ezúttal véletlenszerűen választja a szimulátor egy megadott toleranciatartományból, eredményként pedig eloszlásfüggvények adódnak. A módszer hátránya, hogy pont a szélsőértékeket nem deríti fel szisztematikusan, valamint a paraméterek egymástól való függetlenségét feltételezi.

Kitekintés gyanánt megemlítem, hogy létezik még intervallum-aritmetikán alapuló megközelítés, valamint a paraméterekre elvégezhető érzékenység-vizsgálatot követő analízis, ezeket azonban részletesen nem mutatom be, mivel nem ezen módszerek vizsgálata képezi dolgozatom fő tematikáját.

1.3. Motiváció

A szimulációk végrehajtására léteznek áramkörszimulációs programok. Ezek hatékonyan tudnak szimulációt végezni, rendelkeznek áramköri komponensek modelljeivel is, viszont amennyiben az eredményeket további feldolgozásra kell használni, abban az esetben az áramkör paraméterezése, a szimuláció eredményének exportálása és felhasználása nehézkes és lassú folyamat lehet.

A szimulátorprogramokkal végzett worst-case áramköranalízis másik hártánya, hogy alapvetően numerikus megoldó algoritmusokat használnak még akkor is, ha a modell analitikus kiértékelése hatékonyabb lenne. Sok esetben előfordul, hogy az áramkör modelljét jellemző egyenletek felíratók analitikusan, melyek megoldása kevésbé idő- és számításigényes, mintha numerikus módszereket alkalmaznánk, ráadásul a numerikus közelítésből adódó hiba sem jelentkezik a számítás során, hiszen analitikusan oldjuk meg az egyenleteket.

Az analitikus számítás további előnye, hogy mivel folytonos függvényekkel jellemezhető az áramkör, így tetszőleges érték azonnal behelyettesíthető az egyenletek felírása és szimbolikusan történő megoldása után. Nincs szükség a paraméter numerikus módszerekhez szükséges diszkrét idejű léptetésére, ezért rövidebb ideig tart a számítás. Lehetőség van ebben az esetben párhuzamos, egyszerre több értékre történő kiértékelésre is. Ez azért előnyös, mert a szimuláció újraindítása extra időbe telik sorosan végrehajtott számítási folyamatok esetén.

Egyre inkább jellemző, hogy az áramkörök tervezéséhez használt szoftverekben analízisre alkalmas funkciókat, esetleg szimulátort integrálnak[12], azonban továbbra is fennáll az igény, hogy az áramkörmodell viselkedésének vizsgálatán túl haladó matematikai funkciókat is lehessen rajta alkalmazni egy arra optimalizált környezetben, melyre viszont gyakran egy másik program alkalmasabb. A worst-case analízishez használt paramétereket jellemzően manuálisan kell definiálni, számítani, átmozgatni a különböző programcsomagok között, valamint a különböző szoftverkörnyezetben alkalmazott modellek sem feltétlenül kompatibilisek egymással.

Az alábbi táblázatban összefoglaltam azokat a felhasználó számára fontos szempontokat, melyek a kétféle megoldási módszert jellemzik.

	Szimulátorprogram	**Analitikus megoldás**
Modell áttekinthetősége, szerkeszthetősége	Egyszerű (kapcsolási rajz)	Nehézkes: rendszeregyenletek nehezen átláthatók
Alkatrészmodellek rendelkezésre állása	Nagyon jó	Fejleszteni kell hozzá
Integrálhatóság	Nehézkes, interfészek kialakítása szükséges	Jól illeszthető számítási folyamatokba
Speciális számítási feladatok végrehajtása	Nehézkes, interfészek kialakítása szükséges	Kihasználhatók speciális lehetőségek számítási feladatok gyorsítására (pl. zárt alakú megoldások)

1.1. táblázat. A szempontokat összefoglaló táblázat

Mindkét módszernek vannak előnyei és hátrányai, és ezt szerenténk ötvözni azáltal, hogy át tudjuk konvertálni a kapcsolási rajzot matematikai számítóprogramokban értelmezhető egyenletekké.

1.4. Létező megoldások

A cél, hogy különböző áramkör-reprezentációs leírások közötti konverziókat tudjak végezni, nevezetesen kapcsolási rajz alapján lehessen rendszeregyenleteket generálni, melyek adott esetben hatékonyabban kiértékelhetők a numerikus szimulációnál. Az áramköri modellek létrehozására illetve felépítésére az LTspice, matematikai modellezésre és a továbbis számítások elvégzésére pedig a MATLAB programot fogom használni.

Fontos megemlítenem, hogy léteznek egyébként hasonló törekvések áramköszimuláció-optimalizálásra. Azonban ezek a megoldások jellemzően vagy Python nyelven írt szimulátorok[14], vagy ugyancsak Python nyelvű interfész más szimulátorprogramok (Ngspice, Xyce) számára.[5]

A MATLAB is számos segédeszközt nyújt mérnöki megoldások támogatására. Ezek egyike a Simulink, mely általános célú jelfeldolgozási és folyamatszabályozási feladatok leírásán és megoldásán kívül kifejezetten villamosságtani alkalmazásokra specializált környezetet is tartalmaz, mely a SimScape névre hallgat.[8] Ebben a programban hasonlóképp felvehetők az LTspice-beliekhez hasonló általános kapcsolási rajzok, azonban véleményem szerint sokkal hosszadalmasabb összerakni és felparaméterezni az így elkészült áramköröket, valamint konfigurálni a szimulációs környezetet, mint az LTspice esetén. Előnye ennek a megoldási módszernek, hogy könnyebben exportálhatók, mozgathatók az adatok a SimScape és a MATLAB környezet változói között, és valamivel modernebb hatású a grafikus felület felhasználói élmény szempontjából.

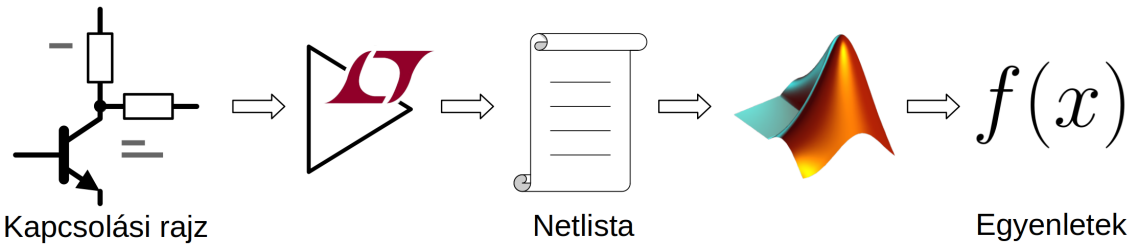
2. fejezet

Specifikáció és rendszertervezés

2.1. Követelmények

A keretrendszer legyen alkalmas arra, hogy egy áramköri modell alapján automatikusan rendszeregyenleteket generál, majd azokat megoldva szolgáltatja az áramkörhöz kapcsolódó szimulációs eredményeket. A keretrendszert elsődlegesen MATLAB-ban kell megírni szem előtt tartva a Python programnyelven történő implementáció lehetőségét is.

Az áramkör modellje LTspice-ban megrajzolt kapcsolási rajzként áll rendelkezésre, az ebből exportálható úgynevezett netlista fájlokból kell olyan hálózati egyenleteket generálni, melyek alapján később MATLAB segítségével elvégezhető a kívánt szimuláció. Követelmény továbbá, hogy a keretrendszer flexibilisen bővíthető és konfigurálható legyen a továbbfejleszthetőség végett.



2.1. ábra. Blokkdiagram a program működéséről

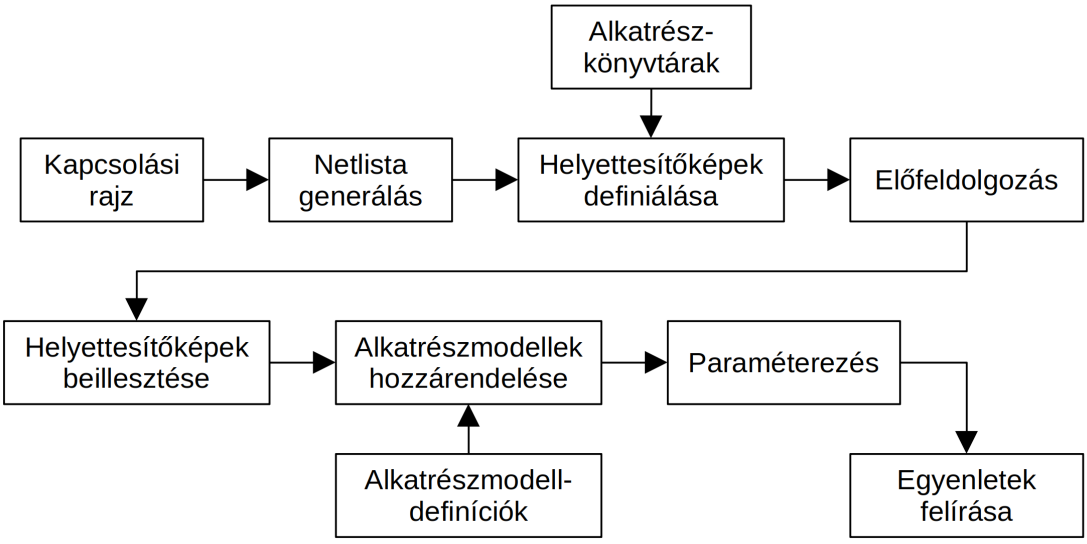
Tulajdonképpen olyan MATLAB-kód illetve modulárisan együttműködő programrészek összességét kell megvalósítani, melyek képesek hálózati egyenleteket generálni, mintegy interfészt szolgáltatva a két program között. A technikai fejlesztéshez hozzátartozik az LTspice alkatrészmodellek feldolgozása, a modelleket tároló adatstruktúrák formátumának kialakítása, és magát a teljes áramkört leíró netlista fájl transzformálása MATLAB-ban hálózati egyenletek rendszerévé. Ezek aztán különböző számítási módszerekkel megoldhatók a szimuláció típusától, az

alkalmazott analitikai eljárástól és a megoldó algoritmustól függően. Azt nem tekintem megkötésnek egyelőre, hogy milyen típusú szimulációt végeztessük el ezzel a keretrendszerrel, hiszen többféle matematikai módszer is kínálkozik egy-egy áramköri probléma megoldására.

2.2. Részfeladatok

A hosszú távú cél az áramkört mint hálózatot leíró függvények olyan reprezentációjának a kidolgozása, mely az alkalmazott matematikai módszertől függetlenül egyszerű és univerzális felületet biztosít bonyolult áramkörök további analíziséhez. Ehhez szükséges a modularitás, és hogy a keretrendszer egyenletek generálásáért felelős része nagymértékben konfigurálható legyen, adott esetben akár teljesen le is lehessen cserélni anélkül, hogy az egész szoftvert a netlisták előállításától kezdve újból implementálni kelljen. Hasonlóképp előfordul a gyakorlatban, hogy egy-egy áramköri elemet egyszerűbb vagy bonyolultabb modellekkel, helyettesítőképekkel jellemeznek, így maguknak az alkatrészmodelleknek is cserélhetőnek kell lenniük.

A feladatot az alábbi részekre bontottam:



2.2. ábra. A rendszerterv blokkvázlata

A kindulási pont egy rendelkezésre álló kapcsolási rajz, melyről az első lépésben netlista fájl készül. Mivel az áramkör tartalmazhat olyan nem elemi komponenseket is, melyek modelljét vagy helyettesítőképét egy külső fájl vagy alkatrészkönyvtár tartalmazza, így azoknak az alkatrészeknek a leírását is be kell illeszteni a netlistába. Ezt követi egy előfeldolgozás, mely során a így összeállított netlista eltérő formátumú részei egységesítésre kerülnek, és a feladat szempontjából irreleváns részeket pedig eltávolítjuk.

Mivel a helyettesítőképek kapcsolási rajzában szerepelhetnek ugyanolyan elnevezésű csomópontok és komponensek, mint egy másik helyettesítőképen vagy az eredeti kapcsolási rajzon, mindenképp szükséges arról gondoskodni, hogy az azonosítók továbbra is egyediek maradjanak. Előfordulhat, hogy egy összetett áramköri alkatrész többszintűen hierarchikus felépítésű, azaz maga is tartalmaz nem elemi komponenseket, ráadásul tetszőleges példányszámban és mélységben beágyazva. A helyettesítőképek beillesztésekor ezekkel a problémákkal foglalkozni kell.

Az alkatrészmodelleknek dinamikusan cserélhetőnek kell lennie, hiszen nemcsak a modell részletességétől, hanem az analízis típusától függően is változik a felírni kívánt egyenletek tartalma. Ez magával vonja azt a feladatot is, hogy olyan formátumban kell az alkatrészmodell-definíciókat tárolni, hogy egységes interfésszel lehessen mind az egyenletgeneráló, mind az alkatrészmodell-hozzárendelő modulhoz kapcsolni.

A modellek beillesztését követően történik annak meghatározása, hogy milyen paraméterekkel dolgozzon az egyenletgenerátor. Ezalatt nemcsak a modellek alkatrész-paramétereit kell érteni, hanem a szimuláció szempontjából meghatározó jellemzőket is itt lehet definiálni. Paraméterként nemcsak konkrét értékek, hanem aritmetikai kifejezések is megadhatók, akár úgy is, hogy azok más paraméterektől függnek. Ebben a lépésben lehetséges a rendszerszintű kényszereket, vagy a rendszer állapotváltozóinak kezdeti értékeit is megadni a szimulációhoz.

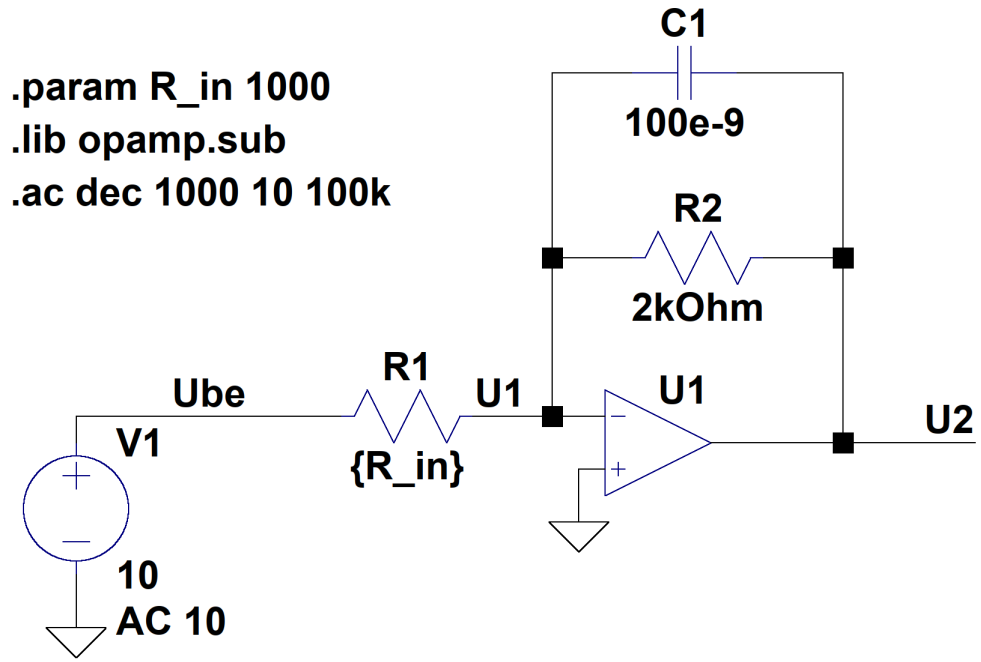
Végül az egyenletek felírása szisztematikusan, a csomópontok, komponensmodellek, paraméterek figyelembevételével történik. Fontosnak tartom megjegyezni, hogy bár dolgozatomban elsősorban analitikus egyenletek felírásával foglalkozom, az egyenletgeneráló algoritmus - mely a folyamat kis részét képezi csupán - a moduláris felépítésnek köszönhetően lecserélhető, és tetszőleges számításokat támogató egyenletek előállítására alkalmas.

3. fejezet

A rendelkezésre álló eszközök bemutatása

3.1. LTspice

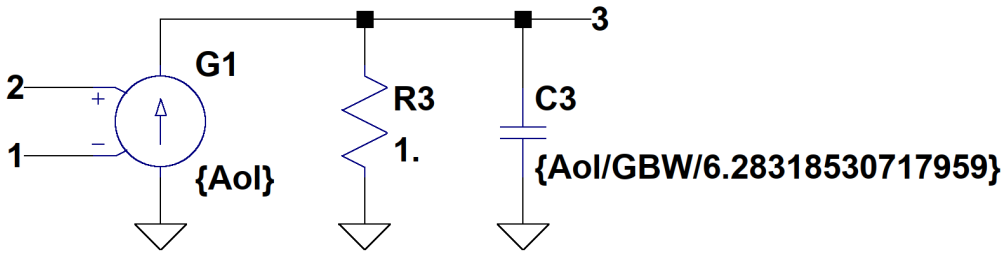
Az áramkörök kapcsolási rajzai grafikusan, az Analog Devices cég LTspice szimulátorprogramja segítségével állnak rendelkezésre.[6] Számos alkatrésztípus közül válogathatunk a tervezés során, valamint az egyes alkatrészekhez többféle áramköri modell is elérhető a programban, melyek rugalmasan paraméterezhetőek.



3.1. ábra. Példa egy LTspice-ban készített kapcsolási rajzra

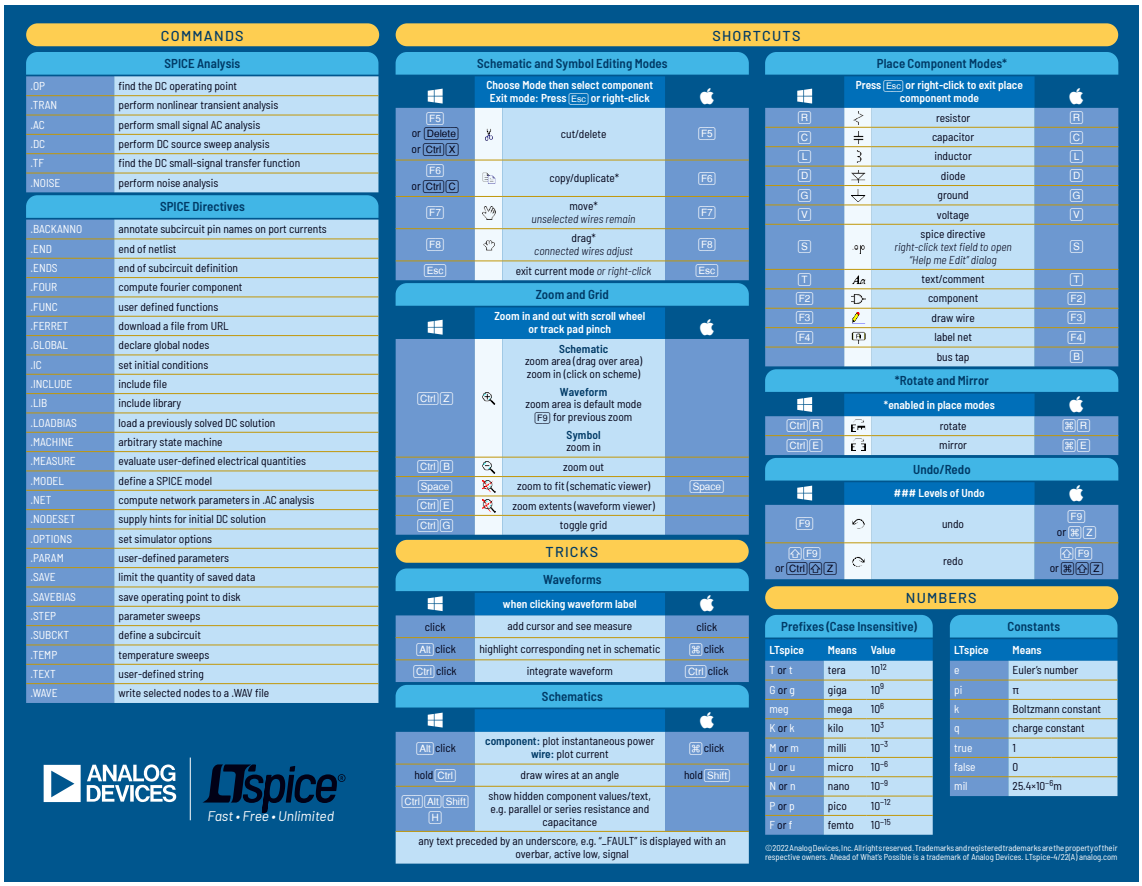
Összetett működésű komponenseknél előfordul, hogy azok modelljét is egy kapcsolási rajzzal vagy helyettesítőképpel írják le, mely szintén kisebb egységekből, további alkatrészekből áll. A könnyebb kezelhetőség és a terv olvashatósága végett

ezeket a helyettesítőképeket (subcircuit, .sub) külön fájlokba, alkatrészkönyvtárakba szervezik, és a hierarchiában legfelső szintű rajzon hivatkoznak rájuk az adott komponensnél.



3.2. ábra. Az `U1` jelű műveleti erősítő helyettesítőképe

Az LTspice szimulátorprogram lévén alapvetően használható áramkör-analízishez, így lehetőség van magát a szimuláció folyamatát is definiálni, paraméterezni (hasonlóképpen az alkatrészekhez) a rajzon ponttal kezdődő szöveges utasítások, parancsok, úgynevezett direktívák segítségével.



3.3. ábra. Kivonat az LTspice kezelési útmutatójából

A létrehozott grafikus áramköri rajz szabványos SPICE netlistaként exportálható a programból. Ez lényegében egy szöveges fált, mely tartalmazza a kapcsolási rajzot mint hálózatot leíró csomópontokat, alkatrészeket, azok

paramétereit, valamint a szimuláció futtatásához szükséges egyéb beállításokat. Az LTspice egyébként a vektorgrafikus áramköri rajzot szintén szöveges formátumú (.asc kiterjesztésű) fájlban tárolja, viszont míg ez a fájl az egyes grafikus elemek elhelyezkedését írja le, addig a netlista az alkatrészmodellek kapcsolatának gráfját reprezentálja.

3.2. SPICE netlisták

SPICE netlistákat többféle áramkörszimulátor program is képes előállítani, azonban szigorúan kötött szintaxisa van ezeknek a leírófájloknak.[10][13] Tartalmazzák az alkatrészek típusát, paramétereit, a csomópontokat, valamint ezek kapcsolatait az alkatrészekkel. Szintén a netlistában jelenik meg az is, hogy az adott kapcsoláson milyen fajta szimulációt szeretnénk futtatni és annak milyen paraméterei vannak.

A netlista soronként értelmezendő, minden sorban egy komponens leírása vagy egy direktíva szerepel. Minden sor szóközökkel elválaszott mezőkre tagolható. Van lehetőség szöveges felhasználói üzenetek, megjegyzések elhelyezésére, ezek a sorok '*' vagy ';' karakterrel kezdődnek. Sortörés esetén a '+' karakterrel kell jelezni, hogy az így kezdődő sor az előző folytatása.

```
* pelda.asc
XU1 U1 0 U2 opamp Aol=100K GBW=10G
R1 Ube U1 {R_in}
R2 U1 U2 2kOhm
V1 Ube 0 10 AC 10
C1 U1 U2 100e-9
.ac dec 1000 10 100k
.lib opamp.sub
.param R_in 1000
.backanno
.end
```

3.4. ábra. A 3.1. ábra kapcsolási rajzáról generált netlista

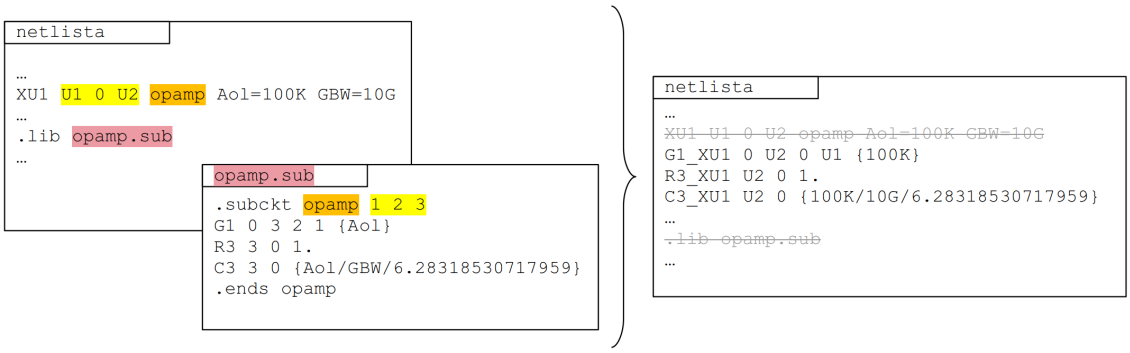
A SPICE modellekben minden alkatrész leírása az alkatrész típusát jelölő betűből és egy azt követő tetszőleges szöveges azonosítóból áll, ezt követi kettő vagy több csomópont azonosítója, melyekhez az alkatrész kapcsolódik.[11] A csomópontokat általában számozzák, de ezek is elláthatók egyedi szöveges jelöléssel, hogy például egy szimuláció esetén az adott csomóponton mért feszültségre lehessen hivatkozni a csomópont nevével, és ez segíti az adatrekord olvashatóságát. Az alkatrészek jellemző és parazita paramétereit számokkal és szöveges karaktert tartalmazó mértékegységekkel, prefixumokkal lehet megadni.[1]

Ha külső könyvtárból illesztünk be alkatrészeket a kapcsolásba, vagy hierarchikusan építjük fel több részből a teljes áramkört, az összetett komponensek legfelső hierarchikus szinten való kapcsolódási módját is a netlista fogja leírni.

```
.subckt opamp 1 2 3
G1 0 3 2 1 {Aol}
R3 3 0 1.
C3 3 0 {Aol/GBW/6.28318530717959}
.ends opamp
```

3.5. ábra. A műveleti erősítő netlistája

A példában az `XU1`-gyel jelölt műveleti erősítő helyettesítőképének netlistája a `.lib` direktíva alapján az `opamp.sub` fájlban található, vagyis annak tartalmát kell a megfelelő módon behelyettesíteni ahhoz, hogy a netlistából egyértelműen felírhatók legyenek a kívánt hálózategyenletek.



3.6. ábra. Helyettesítőkép beillesztésének elvi folyamata

3.3. MATLAB

A MATLAB a MathWorks Inc. cég által fejlesztett speciális programrendszer és programnyelv, melyet matematikai számítások elvégzésére optimalizáltak. A programhoz számos kiegészítő, toolbox és segédprogram tartozik, melyek egy-egy speciális műszaki szakterület támogatására hivatottak.

Bár a MATLAB-ot alapvetően numerikus számítások elvégzésére fejlesztették, a Symbolic Math Toolbox segítségével analitikusan lehet differenciálást, integrálást, különböző transzformációkat és egyenletek megoldását végezni.[9] A számítások analitikusan vagy változó pontosságú aritmetikával numerikusan elvégezhetők, az eredmények algebrai alakban jelennek meg.

```
eqs = [ ...
Ube-0 == V1, ...
G1_XU1 * (U1-0) + (U2-0) / R3_XU1 + (U2-0) * s * C3_XU1 + (U2-U1) / R2 + (U2-U1) * s * C1 == 0, ...
0 + (U1-Ube) / R1 + (U1-U2) / R2 + (U1-U2) * s * C1 == 0, ...
(Ube-U1) / R1 + I_V1 == 0 ...
];
```

3.7. ábra. A példa netlista alapján generált egyenletrendszer

Az egyenletrendszer megoldására különböző megoldó algoritmusok és eszközök használhatók, ezek vagy eredetileg is rendelkezésre állnak az MATLAB alap szoftvercsomagjában, vagy a Symbolic Toolboxhoz hasonlóan feltelepíthetők kiegészítőként.

Mivel a netlista szöveges fájl, az egyenletek előállításához szöveges adatok feldolgozására is szükség van. Erre a részfeladatra alapvetően nem ez a programnyelv a legalkalmasabb (ettől függetlenül megoldható benne), cserébe nagyfokú rugalmasságot tesz lehetővé.

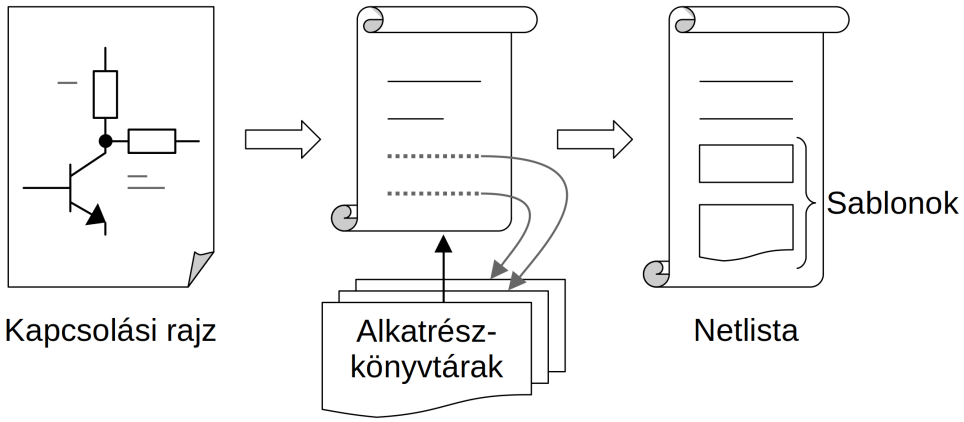
A feladat megoldása során a részfeladatok elvégzéséért felelős szoftvermodulokat, függvényeket külön fájlokba szerveztem, és az egyenletrendszer generálásánál például kihasználtam a lehetőséget, hogy a MATLAB képes olyan fájlokat előállítani, melyek azonnal futtathatóak is ugyanabban a környezetben.

4. fejezet

A megvalósítás részletei

4.1. SPICE netlisták előkészítése feldolgozásra

A SPICE netlista az áramkör kapcsolási rajzának szöveges formátumú reprezentációja. Az egyenletek felírásához szükséges lépések közé tartozik, hogy felismerjük a netlistában szereplő alkatrésztípusokat, valamint a köztük lévő kapcsolatokat. Mivel egy egyenletrendszert egy kapcsolási rajzra írunk fel, elemi szinten kell számításba venni az összes rajta szereplő alkatrészt, melyek a hálózat működésére hatással lehetnek. Az áramköri rajzok és az integrált komponensek helyettesítőképeinek összefésülése, valamint azok egységes formátumúra alakítása képezik a netlista előkészítését.



4.1. ábra. A netlista feldolgozásának előkészületi fázisa

4.1.1. Netlista létrehozása

Ahhoz, hogy először létrejöjjön a grafikus kapcsolási rajz alapján a netlista, az LTspice-t `-netlist` kapcsolóval MATLAB-ból meghívva és megadva az .asc fált paraméterként, le is lehet generáltatni azt. Ez a lépés automatizálható, hiszen a konverzióra ad támogatást a program. Ha hiba nékül lefutott az átalakítás azaz

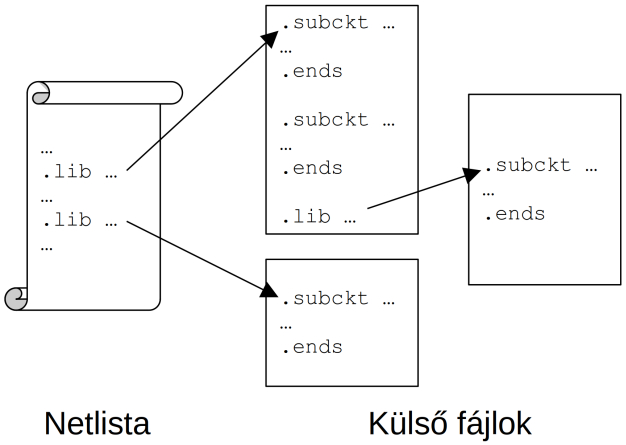
érvényes elektromos hálózatot ábrázol a kapcsolási rajz, létrejön a csomópontokat és az azokat összekapcsoló komponenseket mint elektromos hálózatot reprezentáló szöveges fájl a megadott kapcsolási rajzzal megegyező könyvtárban.

4.1.2. Helyettesítőképek importálása alkatrészkönyvtárakból

Rögtön a feldolgozási folyamat elején külön kell kezelni az integrált alkatrészeket, melyek további elemi vagy újabb összetett komponenseket tartalmaznak. A netlistát soronként dolgozom fel a programban, minden sor egy-egy alkatrészleírást vagy direktívát tartalmaz. Az integrált áramköri elemekhez tartozó sorok a netlistában definíció szerint `X`-szel kezdődnek, így könnyen felismerhető, hogy ezekhez a komponensekhez valamilyen külső fájlban találató helyettesítőkép (úgynevezett subcircuit) tartozik.[4]

A helyettesítőkép nevét az adott sorból ki lehet olvasni, míg azt, hogy az adott nevű subcircuit melyik alkatrészkönyvtár melyik külső fájljában található, a `.lib` direktívák definiálják. Többféle alapértelmezett alkatrészkönyvtár létezik az LTspice-hoz, a külső fájl keresésekor először ezeket a mappákat nézi végig a programom megfelelő prioritási sorrendben.[3]

Nehézséget jelent az importálás során, hogy egy ilyen külső fájl több helyettesítőképet is tartalmazhat `.subckt` és `.ends` direktívák közé foglalva, valamint az is, hogy ezek a külső fájlok szintén hivatkozhatnak másik fájlokra további `.lib` parancsokkal.



4.2. ábra. Alkatrészkönyvtár-hivatkozások

Ez a hivatkozási lánc mely tulajdonképpen egy fagráf, tetszőlegesen mély lehet, viszont futási időben fel kell tudni építeni a helyettesítőképek elemi komponensekre való bontásához. Az implementációban iteratív módon bejártam minden utat, mint a `.lib` direktívák által kijelölt élsorozatot a gráfban, és a netlistához sablonként hozzáfűztem a bejárás során megtalált subcircuit-eket definiáló blokkokat.

Azért fontos kiemelni, hogy a hozzáfűzés csak sablonként történt, mert az adott típusú integrált áramkörből több példány is szerepelhet eredetileg a netlistában, melyeknek ugyan a belső felépítése megegyezik, viszont a bennük szereplő elemi komponensek nem azonosak a különböző példányok esetében. Ezt a problémát a 4.2. alfejezetben járom körül részletesen.

Sajnos nemcsak szöveges tartalmú külső fájlok léteznek a subcircuit-ek, helyettesítőképek definiálására, hanem bináris állományok is. Ilyen formátumú fájlokat az adott integrált áramkör gyártója teszi közzé azzal a megfontolással, hogy áramkör-szimulációhoz használhatóak legyenek, viszont ne lehessen belőlük egyszerű módszerekkel visszafejteni az alkatrész helyettesítőképét. Ennek megfelelően az LTspice képes ilyen fájlokkal dolgozni a szimulációk során, azonban az én megközelítésem szempontjából ezek a fájlok használhatatlanok, vagyis megkötést jelentenek a megoldásom alkalmazhatósági területe számára.

4.1.3. Előfeldolgozás

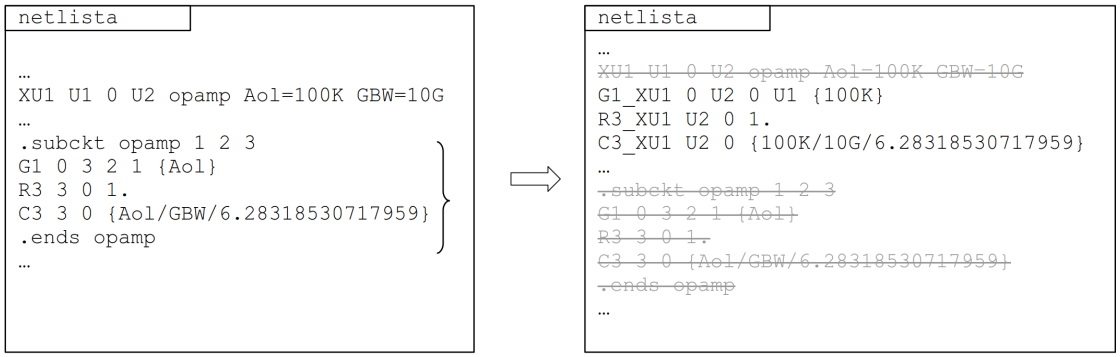
Az eredeti netlista csupán azokat az információkat tartalmazza, melyeket a kapcsolási rajzon megadott a tervező. Ilyenek az alkatrészmodelleken és a csomópontokon kívül a komponensek jellemző tulajdonságainak értékei, a hozzájuk kapcsolható modellparaméterek, integrált áramkörök neve, különböző megjegyzések. Direktívák formájában szerepelnek benne még egyéb változó paraméterek, függvények, szimulációs vagy mérési eljárások definíciói, melyek azonban nem feltétlenül szükségesek az egyenletgenerálás vagy az analitikus számítási módszer szempontjából.

A helyettesítőkép-sablonok beillesztése során is ilyen jellegű adatokkal bővült a netlista, hiszen a helyettesítőképek SPICE netlistáját használjuk fel, melyeket különböző forrásokból importáltunk az előbb. Ezek a fájlrészletek annak ellenére, hogy szabványos formátumúak, a szöveges adatfeldolgozás szempontjából mutatnak némi eltérést egymáshoz képest, ezért célszerű előfeldolgozást végezni rajtuk.

Az előfeldolgozás során a felesleges whitespace karakterek, nem használt direktívák és megjegyzések törlésre kerülnek. Mivel a netlista soraiban az egyes mezőket szóközök választják el, figyelemmel kellett lennem arra is, hogy például aritmetikai kifejezésekben a műveleti jelek környezetében, vagy felsorolt értékek esetén ne maradjon felesleges szóköz (adott esetben egyenlőségjel vagy zárójel se), mivel azok megzavarják a mezőhatárok kijelölését a további szöveges adatfeldolgozás során. A netlista jobb áttekinthetőségének kedvéért különválasztottam a ponttal kezdődő direktívákat az alkatrészeket leíró soroktól.

4.2. Helyettesítőképek beillesztése

A helyettesítőképek netlistáiból sablonok készültek az előző lépések eredményeként. A sablonok beillesztése majd példányosítása során áll elő a végleges netlista, amely már nem tartalmaz integrált áramköröket, így a hálózategyenletek felírására közvetlenül alkalmas. A következő alfejezetben foglalom össze a beillesztés folyamatát.



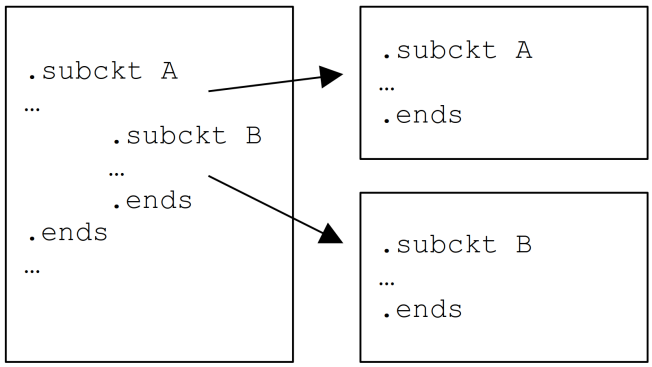
4.3. ábra. Helyettesítőkép beillesztése

4.2.1. Hierarchikus sablonok

A netlistához hozzáfűzött sablonok általános leírást nyújtanak egy-egy integrált áramkör működéséről, ahogy azt eredetileg az alkatrészkönyvtárakban is tették. Azért volt célszerű mégis beilleszteni őket a netlistába, hogy a feldolgozás műveleteit a helyettesítőképeken is ugyanabban a lépésben lehessen végrehajtani, mint a netlista többi részén.

Előfordulnak olyan helyettesítőképek, melyek egymásba ágyazott `.subckt` blokkokat tartalmaznak. Ez megnehezíti a sablon használatát, hiszen a helyettesítőkép netlistába illesztését nem lehet egylépésben elvégezni beágyazott subcircuit-ek esetén. A szoftver modularitása szempontjából lényeges, hogy ne kelljen utólag olyan lépéseket végrehajtani, amiket már egyszer elvégeztünk. Például egy sablon alapján beillesztett subcircuit ne tartalmazzon másik sablont, hiszen bonyolítja a feldolgozást, ha az a netlista példányosított komponenseket tartalmazó részére kerül.

A sablonok importálás után a netlistában közvetlenül egymás után szerepelnek. Az aktuális sablon beágyazottságának szintje megállapítható, ha veremként tekintünk rá: amikor `.subckt` sorral találkozik a feldolgozóprogram, eggyel növeli, amikor `.ends` sorral eggyel csökkenti az adott sorhoz (vagyis a sablon kezdetéhez) tartozó beágyazottsági szintet. Miden iterációban a legmélyebben beágyazott sablont keresi az algoritmus, majd kiveszi a helyéről és a netlista végére írja egy szintre a többi sablonnal. Ezt a lépéssorozatot addig ismétli, amíg talál beágyazott sablont.



4.4. ábra. Hierarchikus sablon felbontása

4.2.2. Struktúrálás

Idáig szöveges adatok előkészítése zajlott, és a sablonokat, helyettesítőképeket kizárólag szöveges adatként kezeltük. A további lépésekhez azonban fontos olyan adastruktúrák kialakítása, mely objektumorientált szemléletet követve használható az alkatrészek áramkörbe illesztéséhez.

Ezeknek az áramkör-prototípusoknak van nevük, csatlakozási pontjai, és tartalmazzák az áramkör helyettesítőképének netlista-részletét. A prototípus objektumok tulajdonságait a sablon sorainak megfelelő mezőiből állítom elő a programban. A prototípusok példányosíthatók, hiszen egy kapcsolási rajzon több ugyanolyan típusú integrált áramkör is szerepelhet.

A programban egy gyűjteménybe szerveztem a prototípusokat. A sablonokhoz hasonlóan mindegyik subcircuit prototípusa is csak egyszer szerepel a struktúrák gyűjteményében. A példányosítás során másolat készül a prototípusról, mely egyedi azonosítóval látja el a helyettesítőkép komponenseit, és ezt követi az áramkörbe illesztés.

4.2.3. Példányosítás

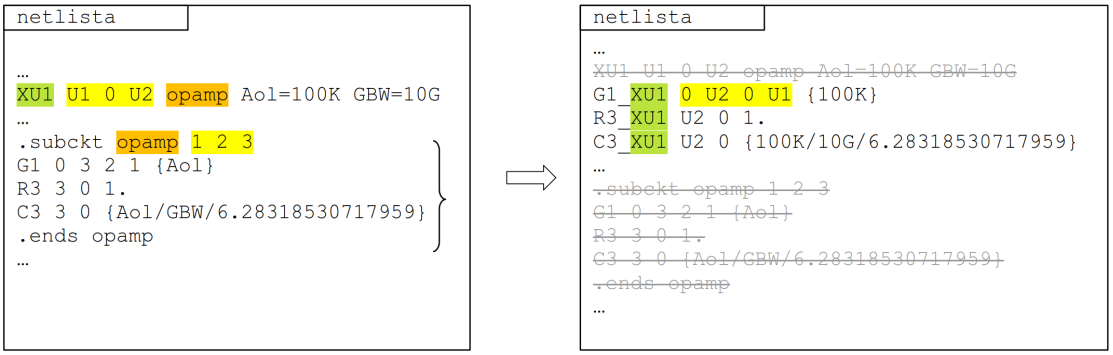
A 4.5. ábrán látható egy integrált komponens általános leírása a netlistában.

```
Xxxx n1 n2 n3... <subckt name> [<parameter>=<expression>]
```

4.5. ábra. Integrált áramkör típusú alkatrész formátuma

Megfigyelhető, hogy az `n1 n2 n3 ...` csomópontok száma a többi szabványos alkatrésztípussal ellentétben nem definiált, hiszen függ az integrált áramkör fajtájától. A csatlakozási pontok azonosítóit az alkatrészkönyvtárakból sablonként importált `.subckt` blokkok tartalmazzák csakúgy, mint a helyettesítőkép nevét.

Visszautalva a 3. fejezetben bemutatott példára, egy integrált áramkör helyettesítőképének netlistába való beillesztése optimális esetben az alábbi módon történik:



4.6. ábra. Sablon példányosítása

Példányosítás során a helyettesítőkép elemi komponenseinek nevét az integrált áramkör azonosítójával egészítem ki (a 4.6. ábrán zöld kiemeléssel jelölve), ezzel biztosítva egyediségüket a végleges netlistában. Erre azért van szükség, mert egy sablonról több példány is készülhet, ha a kapcsolás több ugyanolyan típusú integrált áramkört tartalmaz. A netlista csomópontjait a helyettesítőkép csatlakozási pontjainak megfelelő sorrendjében helyettesítem be. Az integrált áramkörök példányosítását követően a netlistából eltávolításra kerülnek a felhasznált sablonok.

Pusztán a netlista ismeretében nem triviális annak megállapítása, hány csatlakozási pont tartozik az adott subcircuit-hez, ebből következően pedig, hogy a netlista sor hányadik mezője tartalmazza az integrált áramkör nevét, mely szintén szükséges az azonosításához. Mivel előfordulhat olyan eset, hogy valamely csomópont neve megegyezik a subcircuit nevével, nem volt elegendő csak a névre keresni a mezők közt és az alapján elvégezni a hozzárendelést, hanem a csatlakozási pontok számát is figyelembe kellett venni. A prototípus-gyűjteményből mintaillesztéssel választom ki, hogy melyik integrált áramkörhöz melyik prototípus tartozik.

Az integrált áramkörök egymást is tartalmazhatják, vagyis magában a sablonban is szerepelhet integrált áramkör. Ilyen esetben nemcsak az elemi komponensek egyedi azonosítójáról kell gondoskodni, hanem arról is, hogy a helyettesítőkép belső csomópontjai is egyediek legyenek, viszont a magasabb szinten definiált csatlakozási pontok azonosítói ne változzanak. Hasonlóképp érkezhetnek paraméterek is magasabb szintről, mely a helyettesítőkép működését befolyásolják.

4.2.4. Rekurzív behelyettesítés

Mivel a helyettesítőképek beillesztését követően a netlista csak elemi komponenseket tartalmazhat, a példányosítást rekurzív módon végeztem el. A rekurzív megoldást indokolja, hogy az integrált áramkörök egymásba ágyazhatósága miatt nem ismert előre a szükséges példányok száma, ezért nem lehet előzetesen létrehozni az sablonokból az egyedi azonosítójú elemi komponenseket tartalmazó netlista-részleteket. Erre a problémára elvileg megoldást jelenthetne a hierarchikus sablonoknál alkalmazott iteratív módszer, melyet azonban a globális paraméterek és csomópontnevek beágyazott integrált áramkörök felé való továbbterjesztése miatt nem lehet alkalmazni.

Az integrált áramkör egyes példányai esetében figyelemmel kell lenni arra, melyek azok a csomópontok a helyettesítőképen, melyek egyúttal az integrált áramkör csatlakozási pontjai is, és melyek azok, amelyek csak az integrált áramkör belső komponenseit kapcsolják össze. Ez utóbbiakat az elemi komponensekhez hasonlóan szintén egyedi azonosítóval kell ellátni példányosításkor, a csatlakozási pontok neveinek viszont meg kell egyezniük a netlistában definiált csomópontokéval. Mindkét esetben nyilván kell tartani az aktuális integrált áramkörrel kapcsolatba hozható összes csomópontot.

A csomópontok neve az elemi komponenseket leíró netlista sorokból nyerhető ki, azonban alkatrésztípustól függően eltérő számú csomópontot tartalmaz egy-egy ilyen sor annak megfelelően, hogy az adott komponensnek fizikailag hány kivezetése van. Hasonlóképp előfordulhat, hogy az integrált áramkör bemeneti paramétereit tovább kell adni egy másik integrált áramkörnek, melyek azonosítóit szintén meg kell őrizni eredeti formájukban a hívások során.

Végső soron tehát a behelyettesítés folyamán nemcsak az integrált áramköröket kell a sablonok alapján feldolgozni, hanem minden egyes komponensen el kell végezni legalább egy ellenőrzési műveletet. A rekurzív módszer erre a két feladatra egyszerre nyújt megoldást, melyet a megvalósított programban egységbe zárt modulként implementáltam. A rekurzívan hívható szoftvermodul a példányosított integrált áramkör netlistarészletével tér vissza, ami, ha minden subcircuit-et kifejtettünk, teljessé teszi a leírást.

4.3. Egyenletek felírása

A folyamat ezen pontján rendelkezésünkre áll egy olyan netlista, mely alapján jól algoritmizálható módon készíthetők hálózategyenletek. Dolgozatomban csomóponti analízishez használható, lineáris alkatrészek viselkedésének komplex frekvenciatartománybeli leírására alkalmas egyenleteket állítok elő.

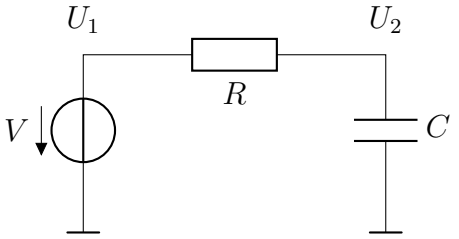
4.3.1. Csomóponti egyenletek általános alakja

A hálózategyenletek teljes rendszere a Kirchhoff-féle csomóponti és huroktörvények alapján felírható. Minden csomópontra és hurokra képezhető ilyen módon egy-egy egyenlet. Az áramkör viselkedésének modellezéséhez ezeken kívül szükséges a hálózatot alkotó komponensek karakterisztikáit is felírni, melyek a komponens kapcsain mérhető feszültségek és a rajtuk folyó áramok közti kapcsolatot definiálják.

Az egyenletrendszer egyszerűsíthető, ha a csomópontok feszültségét egy rögzített referenciaponthoz képest mérve tekintjük ismeretlenként, az alkatrészeken folyó áramokat pedig a karakterisztikák segítségével számítjuk. Mivel minden csomópontra igaz, hogy a befolyó és kifolyó áramok előjeles összege zérus, huroktörvény a karakterisztikák miatt automatikusan teljesül, ráadásul a referenciapont feszültségét 0-nak választva eggyel kevesebb csomóponti egyenletet elegendő felírni a hálózatra.

A csomópontok áramaira vonatkozó egyenleteket olyan tagok segítségével írjuk fel, melyek kifejezik, hogy egy-egy alkatrész hogyan járul hozzá a csomópont áramához a szomszédos csomópontok potenciáljának függvényében.

Íme egy egyszerű példa a hálózati egyeletek felírására:



$$\begin{cases} U_1 : \dfrac{U_1 - U_2}{R} + I_V = 0 \\ U_2 : \dfrac{U_2 - U_1}{R} + (U_2 - 0)sC = 0 \\ V = U_1 - 0 \end{cases}$$

4.7. ábra. Hálózat és a hozzá tartozó egyenletek

A példában az U_1 és U_2 csomópontokra írtuk fel a be-és kifolyó áramokat az alkatrészek karakterisztikájának felhasználásával. Az egyenletrendszerben a csomóponti potenciálok valamint az I_V forrás árama az ismeretlenek, C, s és R paraméterek, valamint a V jelű forrás feszültségkényszert ír elő az U_1 csomóponra. A 0 referencia csomópontot az áramköri rajzon "föld" szimbólum jelöli, mint ahogy egyébként LTspice-ban is.

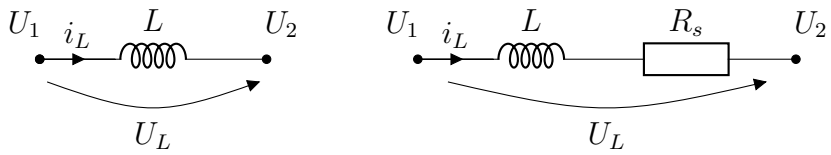
4.3.2. Alkatrésztípusok, alkatrészmodellek

A rendszertervezés során felmerült, hogy mivel mindenképp szükséges az alkatrészek matematikai modelljének kidolgozása, érdemes lenne valamiféle általános alkatrészkönyvtárat létrehozni a matematikai alkatrészmodellek számára. Ebben a könyvtárban egy-egy fájl tartalmazza az adott alkatrész karakterisztikáját leíró egyenletrészletet, mely a netlista komponenseinek külön-külön megfeleltethető,

azok mezői alapján pedig felparaméterezhető. Ez a fajta megoldás amiatt előnyös, mert az alkatrészek modellje igény szerint módosítható, lecserélhető anélkül, hogy a szoftver többi részét módosítani kellene.

Egy alkatrészhez akár több egyenlet is tartozhat a komponens kivezetéseinek számától függően, valamint aszerint, hogy a hálózatra vonatkozóan tartalmaz-e extra állapotváltozót vagy okoz-e rendszerszintű kényszert az adott komponens, melyet esetleg külön egyenletként hozzá kell venni a hálózategyenletek rendszeréhez. Az alkatrész karakterisztikáját leíró egyenletrészleteket ezekben a fájlokban olyan formátumban állítom elő, hogy a MATLAB számára közvetlenül felhasználható legyen. Ezek az alkatrészmodell-fájlok tulajdonképpen MATLAB-függvények, melyek az adott alkatrészhez tartozó netlista sor mezőivel felparaméterezett szimbolikus egyenletrészletet adják vissza.

Az alábbi példa egy tekercs ideális, valamint ekvivalens soros ellenállással rendelkező modelljének leírását szemlélteti.



4.8. ábra. Induktivitás két lehetséges áramköri modellje

A modellek paraméterei az L induktivitás és az R_s ekvivalens soros ellenállás. Ezeknek az értékeit a netlistából paraméterként helyettesíthetjük be a matematikai modellbe. A csomópontok potenciálja mindkét esetben U_1 és U_2. Az analízis során felhasználói szempontból fontos, hogy ugyanazokat az elnevezéseket használjuk a programban, mint amelyek a netlistában is szerepeltek, vagyis az egyenletrendszerben U_1 nevű változó fogja tartalmazni a netlista U_1 azonosítójú csomópontjának feszültségértékét.

Az áramköri modelleket jellemző egyenletek (komplex frekvenciatartományban felírva):

$$\begin{cases} i_L = \dfrac{U_1 - U_2}{sL} \\ U_L = U_1 - U_2 \end{cases} \qquad \begin{cases} i_L = \dfrac{U_1 - U_2}{R_s + sL} \\ U_L = U_1 - U_2 \end{cases}$$

A modellt megvalósító MATLAB-függvény paraméterként egy induktivitáshoz tartozó netlista sort kap meg, mely a szükséges paramétereket tartalmazza. Mivel az algoritmus az áramkör minden csomópontjára szekvenciálisan ír fel egy-egy egyenletet, fontos tudni, hogy az adott iterációban a komponens melyik kivezetése csatlakozik a szóban fogró csomópontra. Ennek az áram- és feszültség-referenciairányok miatt van jelentősége, hiszen míg például az i_L áram U_1

jelű csomópontból "kifelé", addig U_2 jelű csomópontba "befelé" folyik, az egyenletek szempontjából pedig az áram iránya (előjele) nem mindegy.

A komponens matematikai modelljét reprezentáló MATLAB függvényben az alkatrész minden csatlakozási pontjára külön írtam fel a karakterisztikát jellemző áramegyenletet, mely a referenciairányokat helyesen veszi figyelembe:

$$\begin{cases} U_1 : i_L = \dfrac{U_1 - U_2}{sL} \\ U_2 : i_L = \dfrac{U_2 - U_1}{sL} \end{cases} \qquad \begin{cases} U_1 : i_L = \dfrac{U_1 - U_2}{R_s + sL} \\ U_2 : i_L = \dfrac{U_2 - U_1}{R_s + sL} \end{cases}$$

4.3.3. Paraméterezés

A hálózati egyenletek felírására kidolgoztam néhány ökölszabályt, például hogy a netlistából mely mezőket tekintsük paramétereknek, milyen állapotváltozókat kell bevezetni bizonyos típusú alkatrészek esetén és milyen alakúak legyenek a szimbolikus függvények.

Paraméterként komponensek jellemző értékei adhatók meg tipikusan, de előforduhatnak aritmetikai kifejezéssel, esetleg táblázatban vagy listában felsorolt értékekkel megadott paramétervektor is, melyek több egymást követő szimuláció alkalmával kerülnek behelyettesítésre és kiértékelésre.

A SPICE modellekben megadott aritmetikai kifejezések és mértékegység prefixummal rendelkező mérőszámok formátuma alapvetően nem teljes mértékben kompatibilis a MATLAB-ban használt operátorokkal és számformátumokkal, így a paraméterek behelyettesítésekor a szükséges átalakításokat el kell végezni, ezt az átalakítást természetesen egy külön programmodul végzi.

Mivel szimbolikus egyenleteket írunk fel, a paraméterek behelyettesítése történhet közvetlenül azok megoldását megelőzően, így nem szükséges egy-egy paraméter megváltozása esetén ismét felírni az elejétől kezdve a megoldani kívánt egyenletrendszert.

4.3.4. Egyenletgenerálás

Az egyenletrendszer szisztematikusan generálható köszönhetően a netlista megfelelő előkészítésének. A csomóponti áramokra felírható egyenletekhez az alkatrészek csomóponthoz kapcsolódó megfelelő karakterisztikus egyenletét tagonként összeadjuk. Ezt minden csomópontra elvégezzük, majd az alkatrészek által definiált rendszerszintű kényszereket is hozzávesszük a csomóponti egyenletekhez. A csomópontok potenciálja szimbolikus változóként szerepel az egyenletekben, az alkatrészek értékei szintén. Ezek értéke később az analízis során paraméterként megadható. Az egyenlet megoldása elsősorban a csomóponti potenciálokra vagy

az állapotváltozókra történik, viszont a karakterisztikák ismeretében kifejezhető bármelyik komponens árama is, vagyis ez nem jelent megkötést a megoldás szempontjából.

A szimbolikus egyenleteket MATLAB-ban alapvetően kézzel lehet megadni, nem igazán létezik arra az esetre szoftveres támogatás, hogy az egyenletrendszer struktúráját a benne szereplő változók neveivel és paramétereivel együtt dinamikusan lehessen konfigurálni. Léteznek ugyan ennek megvalósítására alkalmas megoldások, viszont ezek alkamazását nem javasolja a MATLAB kézikönyve, ugyanis rontja a programkód áttekinthetőségét és a fejlesztés során számos hibalehetőséget rejt magában.[7]

Végülis az egyenletek MATLAB-beli futásidejű felírásával az a feladat, hogy "önmagát létrehozó" kódot készítsek, mely valóban hordoz magában kockázatos lehetőségeket. Az egyenletrendszert az alkatrészmodellek fájljaihoz hasonlóan egy különálló állományban hozom létre, mely szintén egy hívható és futtatható MATLAB függvény. Ennek egyik oka, hogy a generált egyenletrendszer helyességéről még annak megoldása előtt meg lehessen bizonyosodni, másrészt a további fejlesztéseknek fenntartott lehetőségek kedvéért célszerű, ha az egyenletgenerátor programmodul nagyon széles skálán testreszabható. Ez a konfigurálhatóság annak köszönhető, hogy az alkatrészmodellek által visszaadott szimbolikus függvényrészletek szöveges formátumúak, tehát a végső egyenletrendszert tároló fájlba közvetlenül beírhatók.

A létrehozott egyenleteken kívül természetesen további műveletek, analízis módszerekhez kacsolódó programrészletek is illeszthető automatizált módon az egész folyamat eredményeként előállt MATLAB-függvénybe.

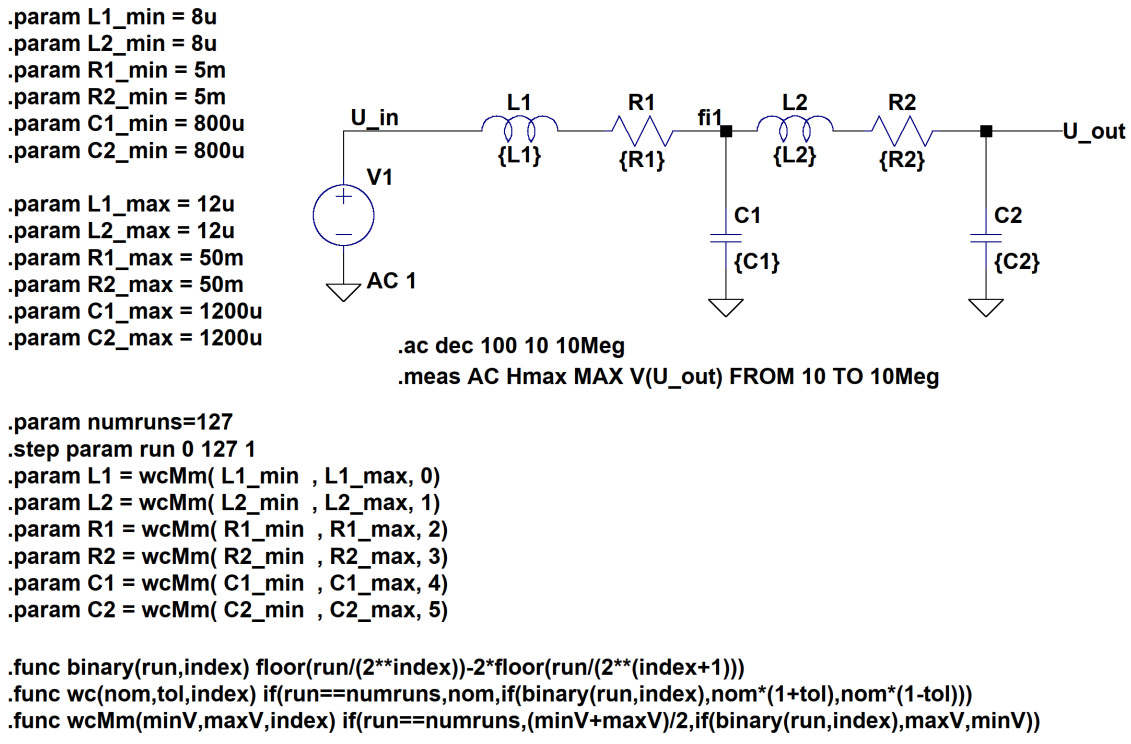
5. fejezet

Eredmények

A továbbiakban bemutatom az elkészült program helyes működését egy gyakorlati példán keresztül. Az alkalmazási példában extrémérték-analízist végzek analitikus módon. Összehasonlítom a megoldásom által szolgáltatott eredményt a szimulátorprogram segítségével meghatározottal nemcsak annak helyessége, hanem az alkalmazott módszerek futásidejének szempontjából is.

5.1. Demonstrációs áramkör

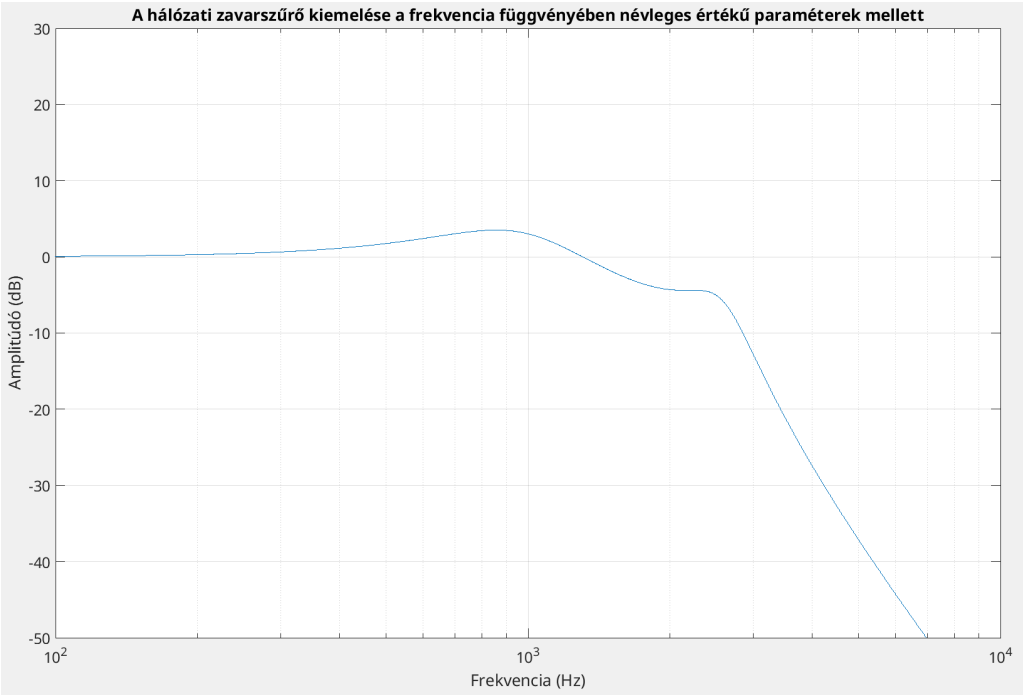
A megvalósított program működését az 5.1. ábrán látható áramkörön mutatom be.



5.1. ábra. A demonstrációs áramkör rajza LTspice-ban

Ez valamilyen elektronikus eszköz hálózati zavarszűrőjének a modellje, ilyen előfordul a gyakorlatban is. Szerenténk megnézni a maximális kiemelését különböző paraméter-együttállások esetén, mivel fontos számunkra hogy a szűrő rezonanciafrekvenciáján mekkora lehet maximálisan zavarjel amplitúdója egységnyi amplitúdójú gerjesztés estén ami az elektronikus eszköz működését befolyásolhatja.

Az szűrő kiemelése névleges értékű paraméterek mellett az 5.2. ábrán látható módon alakul. A szűrőáramkört felépítő komponensek névleges értékei: $L_1 = L_2 = 10$ µH, $R_1 = R_2 = 50$ mΩ, $C_1 = C_2 = 1$ mF.



5.2. ábra. A szűrő átvitele névleges értékű paraméterek esetén

Extrémérték-analízis során minden paraméterkombinációra felveszünk egy ilyen görbét, és az így kapott görbesereg maximumát (és a hozzá tartozó paraméter-kombinációt) tekintjük legrosszabb esetnek, amit a worst-case analízissel keresünk. A mintapéldában az alkatrészértékek szélső értékei az 5.1 ábrán lévő szimulációs modellen vannak definiálva `.param` direktívák segítségével.

A megoldásom eredményeként az LTspice modellből először az 5.3. ábrán látható netlista, majd pedig az 5.4. ábrán látható MATLAB állomány készült.

```
V1 U_in 0 AC 1
L1 U_in N001 {wcMm(L1_min,L1_max,0)} Rser=0
R1 fi1 N001 {wcMm(R1_min,R1_max,2)}
C1 fi1 0 {wcMm(C1_min,C1_max,4)}
C2 U_out 0 {wcMm(C2_min,C2_max,5)}
L2 fi1 N002 {wcMm(L2_min,L2_max,1)} Rser=0
R2 U_out N002 {wcMm(R2_min,R2_max,3)}
.step param run 0 127 1
.ac dec 100 10 10Meg
.func binary(run,index)floor(run/(2**index))-2*floor(run/(2**(index+1)))
.func wc(nom,tol,index)if(run==numruns,nom,if(binary(run,index),nom*(1+tol),nom*(1-tol)))
.func wcMm(minV,maxV,index)if(run==numruns,(minV+maxV)/2,if(binary(run,index),maxV,minV))
.meas AC Hmax MAX V(U_out)FROM 10 TO 10Meg
.backanno
```

5.3. ábra. Netlista

```
syms U_in N001 fi1 U_out N002 ; % nodes
syms V1 L1 R1 C1 C2 L2 R2 ; % symbols
syms I_V1 ; % variables
syms s;

eqs = [...
... % General consraints
U_in - 0 == V1, ...
... % Nodal equations
I_V1 + (U_in - N001) / s / L1 == 0, ...
(N001 - U_in) / s / L1 + (N001 - fi1) / R1 == 0, ...
(fi1 - N001) / R1 + (fi1 - 0) * s * C1 + (fi1 - N002) / s / L2 == 0, ...
(U_out - 0) * s * C2 + (U_out - N002) / R2 == 0, ...
(N002 - fi1) / s / L2 + (N002 - U_out) / R2 == 0 ...
];

UoutLC = solve(eqs, [U_in, N001, fi1, U_out, N002, I_V1]);
```

```
fnLC = matlabFunction(UoutLC.U_out, 'File', 'LC') ;
x_name = {'L1', 'L2', 'R1', 'R2', 'C1', 'C2'};
x_min = [ 8e-6,  8e-6,  5e-3,  5e-3,  800e-6,  800e-6];
x_max = [12e-6, 12e-6, 50e-3, 50e-3, 1200e-6, 1200e-6];
x = x_min;
fm = 10;
fM = 10e6;
NperD = 1000;
fV = logspace(log10(fm), log10(fM), NperD*round(log10(fM/fm)));

sV = 1j*2*pi*fV;
Xs = [x_min; x_max];
Ha = zeros(128, length(fV));

for ii = 0:127
  xs = dec2bin(ii,6);
  xsn = (xs+0)-48+1;
  H = abs(fnLC(Xs(xsn(5),5),Xs(xsn(6),6),Xs(xsn(1),1),Xs(xsn(2),2),Xs(xsn(3),3), Xs(xsn(4),4),1,sV));
  Ha(ii+1,:) = H;
end

20*log10(max(max(Ha)))
```

5.4. ábra. Az automatikusan generált MATLAB-modell és EVA-számítás

A generált fájl két részre bontható: első felében maga a számítási modell látszik, a netlista alapján előállított szimbolikus változókkal és egyenletekkel, valamint a szimbolikus megoldással. Az állomány második felében az alkalmazott analízis módszertől függő számítások végezhetők el. Itt történik a paraméterek behelyettesítése, majd a mérési eredmények kiértékelése is.

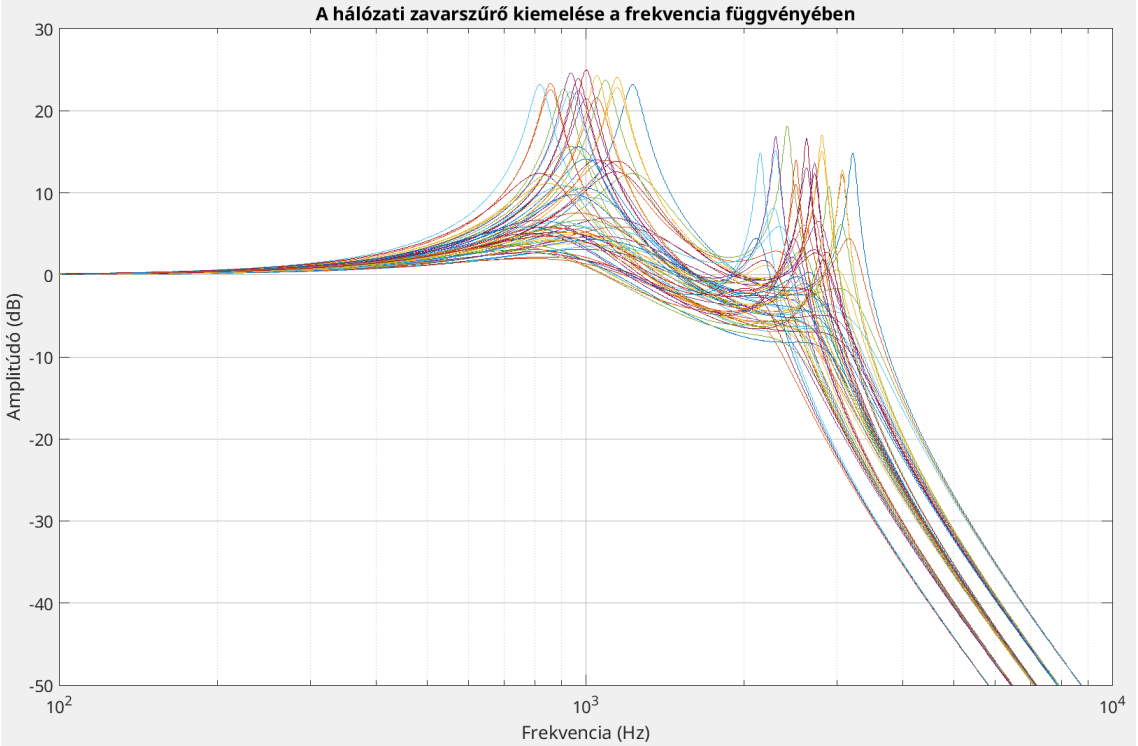
5.2. Az eredmények validálása

Elvégeztem az analízist a MATLAB-ban megvalósított programommal analitikusan, valamint leszimuláltam az áramkör működését LTspice-ban is.

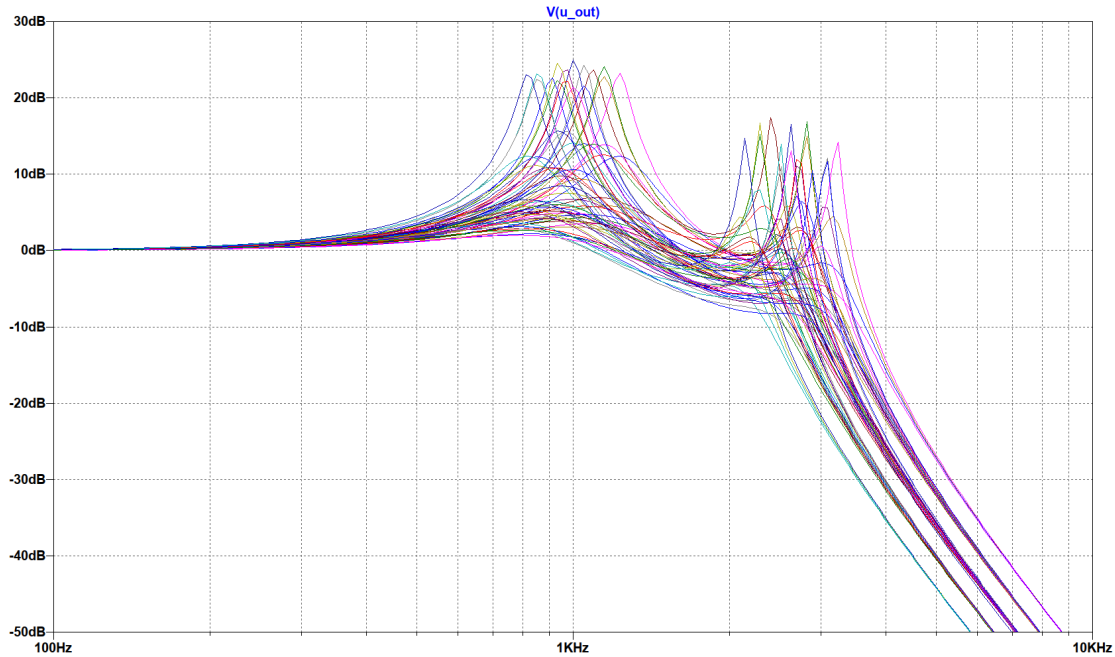
Az analízis végeztével előállt görbesereg a 5.5. és a 5.6. ábrákon figyelhető meg. A görbék jellegüket tekintve nagyon hasonlóak egymáshoz a két esetben, ebből arra lehet következtetni, hogy helyesen modellezik az áramkört a felírt egyenletek.

A maximális kiemelés egyébként az LTspice-beli szimulációnál 24.9331 dB-re, MATLAB-ban 24.9730 dB-re adódott. Látható a görbesereg alapján hogy a MATLAB finomabb felosztással dolgozott az LTspice-hoz képest, emiatt a maximumot jobban közelíti a mért eredmény, mint a durvább felbontású lépésközzel végzett numerikus szimuláció.

A worst-case analízis során kiderült, hogy a névleges 3.5199 dB helyett 24.973 dB-es a szűrő kiemelése (és ez a maximumhely 855.7 Hz-en található). Hogy ez a gyakorlatban milyen súlyos problémát okoz a védendő elektromos berendezés működésében (vagy egyáltalán hatással van-e rá), azt a fejlesztőmérnökök tudják meghatározni.



5.5. ábra. Az analitikus számítás eredménye



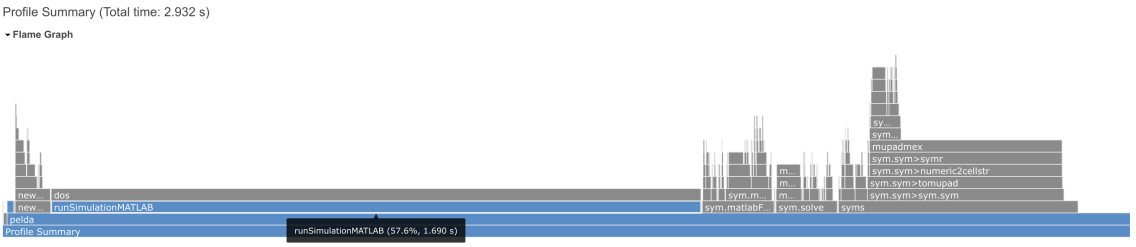
5.6. ábra. LTspice-beli szimuláció eredménye

5.3. Futásidő-analízis

A kitűzött cél az analízis elvégzésének felgyorsítása volt, így azt is leellenőriztem, hogy az általam megvalósított megoldás amellett, hogy helyes eredményt szolgáltat valóban jobban teljesít-e.

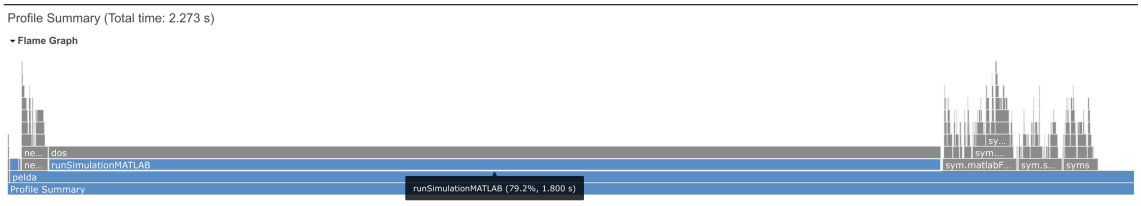
A program futásideje MATLAB segítségével mérhető. Ahhoz, hogy az LTspice szimuláció idejével össze tudjam hasonlítani az én megoldásom hatékonyságát, készült egy tesztkörnyezet MATLAB-ban. A tesztkörnyezetben lefut az én megoldásom netlista-átalakítással egyenletgenerálással és analitikus megoldással együtt. Másik esetben a MATLAB-kód az LTpice programot indítja el, hogy végezze el a szimulációt, és szolgáltasson kiértékelt eredményeket. Készült egy harmadik lehetőség is, mely során csak a MATLAB funkcióit használjuk, az egyenleteket kézzel felírtuk és átadtuk az analitikus megoldó számára. Ezzel annak hatását tudjuk vizsgálni, mennyiben múlik az LTspice programhíváson a kiértékelés gyorsasága. A program futásának elemzéséhez a MATLAB Profiler funkcióját használtam. Ezzel az eszközzel nyomon követhető, hogy a futás során melyik részfeladat elvégzése mennyi ideig tart.

Referenciaként megvizsgáltam, mennyi ideig tart egy MATLAB-bal elindított LTspice-szimulációt lefuttatni, majd pedig manuálisan felírt rendszeregyenleteket kiértékelni szimbolikusan. Ezt szemlélteti a 5.7 ábra.



Generated 28-okt.-2023 23:34:50 using performance time.

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
pelda	1	2.444	0.136	
runSimulationMATLAB	1	1.690	0.001	
dos	1	1.689	1.689	
mupadmex (MEX-file)	831	0.673	0.643	
sym.matlabFunction	1	0.187	0.006	
sym.solve	1	0.159	0.011	
syms	1	0.147	0.036	



Generated 28-okt.-2023 23:34:04 using performance time.

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
pelda	1	2.267	0.073	
runSimulationMATLAB	1	1.800	0.000	
dos	1	1.800	1.800	
sym.matlabFunction	1	0.149	0.006	
sym.solve	1	0.088	0.008	
sym.matlabFunction>writeMATLAB	1	0.083	0.002	
mupadmex (MEX-file)	716	0.078	0.075	

5.7. ábra. Futásidő-analízis LTspice-szimulációra

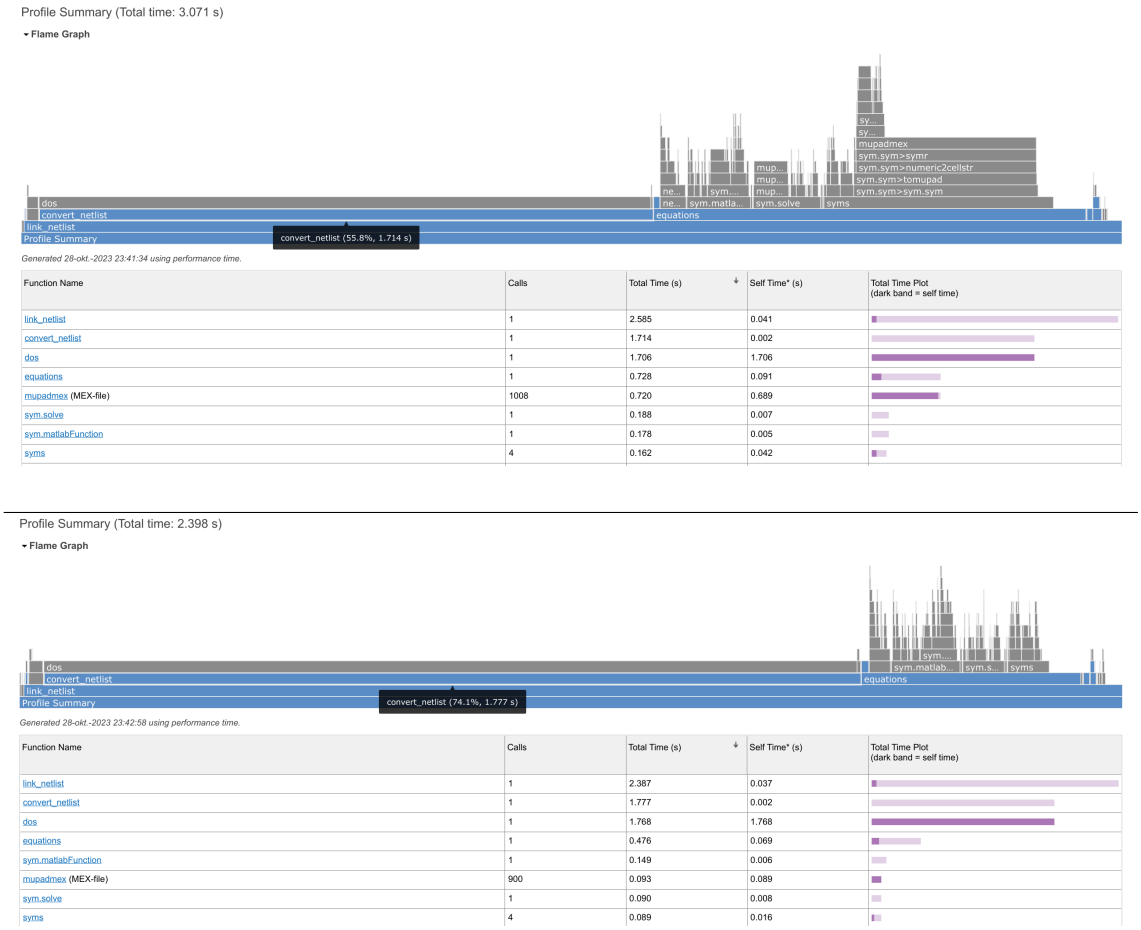
A mérési eredményeket az alábbi táblázatban foglaltam össze:

	Teljes futásidő	**Ebből az LTspice futásideje**	**MATLAB további futásideje**
Első futtatás alkalmával	2.932 s	1.690 s	1.242 s
Korábbi részeredmények felhasználása esetén	2.273 s	1.800 s	0.473 s

A felső és alsó grafikon két egymást követő futtatást ábrázol. A különbség a két eset között annyi, hogy a felső diagram alapján tovább tart a program futása, és a szimbolikus egyenletekkel végzett MATLAB-műveletek összességében több időt vesznek igénybe, mint a második futtatás alkalmával.

Ez annak köszönhető, hogy az első futtatáskor a változókat tartalmazó MATLAB workspace üres volt, és valószínűleg az inicializálás és a szimbolikus egyenletek műveletei miatt tovább tartott a számítás. Másodjára úgy indítottam el a programot, hogy a workspace-ben szerepeltek az előző futtatás részeredményei, ezeket pedig fel tudta használni a MATLAB a számítás elvégzéséhez.

Előnyös tulajdonság, hiszen az analízis ismételt elvégzésére a gyakorlatban szükség lehet. Például amikor csak a paraméterek szélsőértékeit változtatjuk meg, viszont a rendszeregyenlethez nem nyúlunk, így azokat nem szükséges újraszámolni. Megfigyelhető továbbá, hogy a szimuláció futásidejére nem volt hatással a MATLAB belső állapota, hiszen az LTspice programhívás ettől függetlenül történt mindkét alkalommal. Hasonlóképp alakult a futásidő akkor is, amikor az én megoldásom hatékonyságát vizsgáltam.



5.8. ábra. Futásidő-analízis egyenletgenerálással

A futásidőt ekkor is a netlista létrehozásához szükséges LTspice programhívás befolyásolja legjobban, ismételt futtatásokra pedig a szimbolikus egyenletek kezelése rövidebb ideig tart:

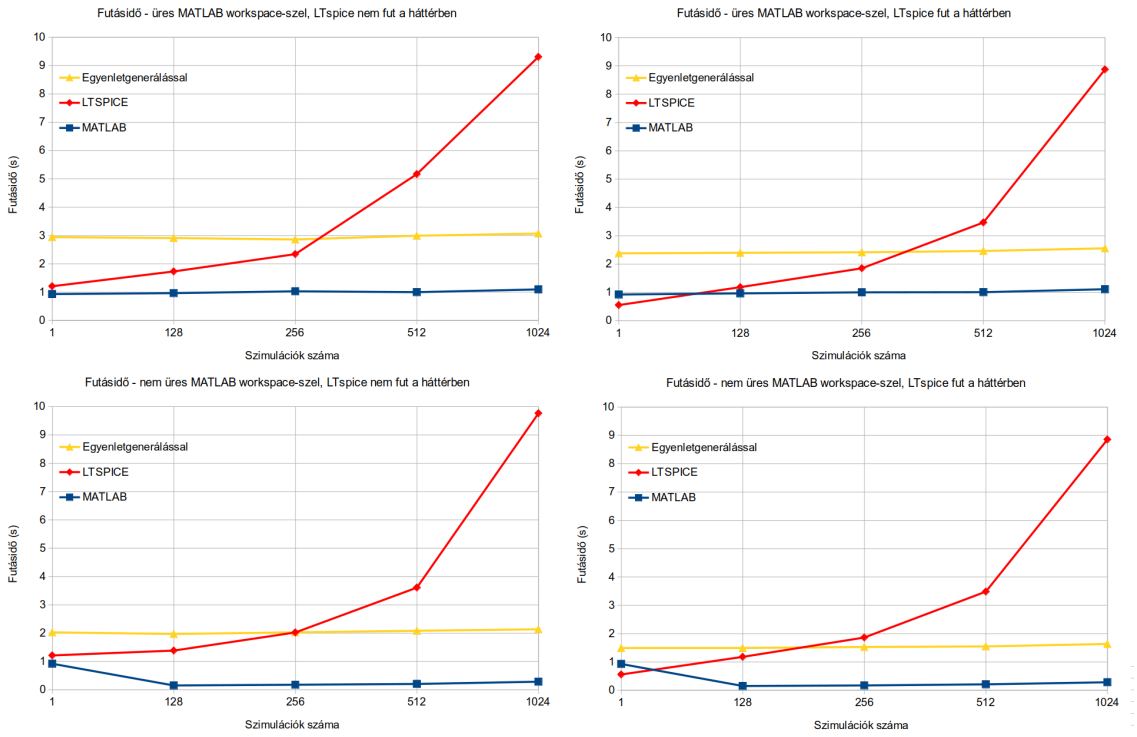
	Teljes futásidő	**Ebből az LTspice futásideje**	**MATLAB további futásideje**
Első futtatás alkalmával	3.071 s	1.714 s	1.357 s
Korábbi részeredmények felhasználása esetén	2.398 s	1.777 s	0.621 s

Feltűnhet, hogy a program teljes futásideje több, mint amikor az LTspice-szal végeztettük a szimulációt. Ez azt jelentené, hogy valójában nem hatékony a megoldásom, mert a netlista előállítása összemérhető idejű egy teljes LTspice-szimuláció futásával, melyre viszont egyáltalán nincs hatással, mit programozok MATLAB-ban.

A mérést több szimulációra (különböző számú paraméterkombinációval) elvégeztem, hogy kiderüljön, hogyan függ a paraméterek számától az egyes módszerek futásideje. Mivel a kapcsolási rajzban rögzített számú paraméter van, a szimulációszámot úgy növeltem meg, hogy egy paraméterkombinációhoz tartozó szimulációt egyszerűen több alkalommal (128, 256, 512, 1024) is elvégeztettem.

Azt tapasztaltam, hogy mivel a programot nem valósidejű operációs rendszer környezetben használom, a futásidő nem determinisztikus. Adott szimulációszám mellett ezért többször is megmértem a futásidőt, és ezek átlagát ábrázoltam a három tesztesetben:

- Sárga: az általam fejlesztett, LTSpice kapcsolási rajz alapján MATLAB egyenleteket generáló programmal
- Piros: LTSpice szimulátor segítségével (MATLAB programból indítva)
- Kék: a rendszeregyenleteket manuálisan felírva és szimbolikus toolbox segítségével MATLAB-ban megoldva



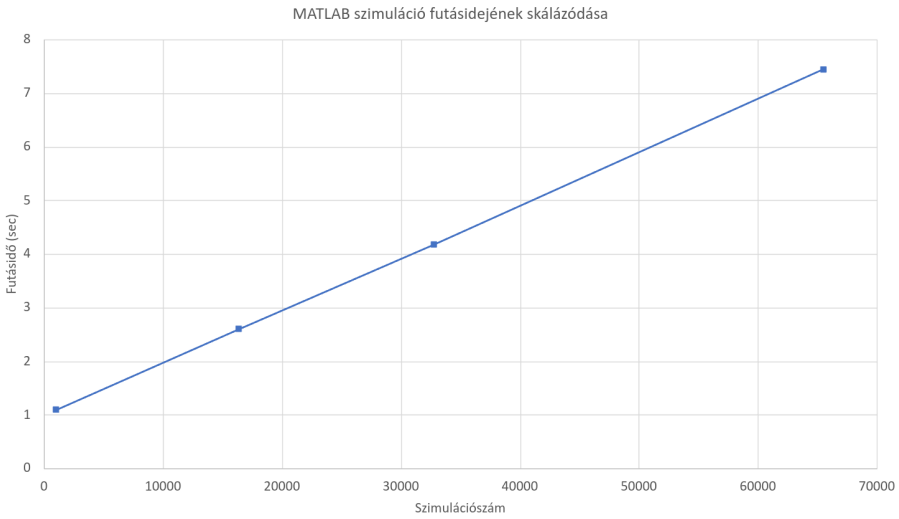
5.9. ábra. Futásidő a szimulációszám függvényében

Egyértelműen látszik, hogy az LTspice-ban végzett szimulációk futásideje azok számával exponenciálisan nő. Az is megfigyelhető, hogy kevés számú szimulációra az LTspice valóban gyorsabban lefut az én megoldásomnál, azonban a szimulációk számának növelésével látványosan jobb eredményt nyújt a programom. Ez pozitív eredmény, hiszen épp az ilyen esetekre szerettünk volna gyorsabb működést biztosítani.

Az 5.9. ábráról kiderül az is, hogy pusztán az LTspice szimuláció, a MATLAB-ban manuálisan felírt szimbolikus egyenletek megoldásának, valamint az én módszerem futásideje milyen összefüggésben áll egymással. A MATLAB előnye egyértelmű: a szimulációszám növelésével a futásidő jóval kisebb mértékben

növekszik az LTspice-énál. Mivel ugyanezt a számítási módszert tartalmazza az én megoldásom is, így az is hasonlóan jól teljesít ebben a tekintetben.

Ha a MATLAB-ban rendelkezésre állnak a szimbolikus egyenletek, a számítás nagyon rövid idő alatt elvégezhető. A netlista előállítása és az egyenletek felírása miatt az én módszerem valamivel lassabb. Érdekesség, hogy nemcsak a MATLAB-ban megírt program futásideje függ attól, hogy üres volt-e induláskor a workspace, hanem az is, hogy az LTspice programhívás előtt esetleg futott-e már korábban a számítógépen. Azt tapasztaltam, hogy ha az LTspice-ot azelőtt elindítjuk, hogy a MATLAB-ból is meghívnánk, a programhíváskor sokkal gyorsabban lefut. Az 5.9. ábra bal oszlopához képest a jobb oldali diagrammokon az LTspice programhívást használó módszerek futásideje nagyjából fél másodperccel szisztematikusan kevesebb. Hasonlóképp megfigyelhető a workspace futtatásra gyakorolt hatása. Amikor üres volt, majdnem egy másodperccel tovább tartott a program futása, mint amikor korábbi részeredmények megtartásával futtattam a szimulációt. Ez a hatás azoknál a módszereknél jelentkezik, ahol a MATLAB nagyobb mennyiségű számítást végez. Az 5.9. ábrán a felsőkhöz képest az alsó diagrammokon a MATLAB-ot használó számítások lettek gyorsabbak.



5.10. ábra. MATLAB szimuláció futásidejének skálázódása

A szimulációszám további növelése esetén a MATLAB segítségével végzett számítások futásideje jó közelítéssel a 5.10. ábrán látható módon lineárisan skálázódik szemben az LTspice exponenciális függésével, tehát az általam kidolgozott módszer nagyszámú paraméterek esetén is hatékonynak mondható.

6. fejezet

Összefoglaló, kitekintés

Dolgozatomban bemutattam, hogyan készítettem LTpice-beli áramköri modellekből MATLAB-ban analitikusan megoldható szimbolikus hálózategyenleteket, majd a módszer alkalmazásával egy gyakorlati példán keresztül szemléltettem, hogyan lehet áramköranalízis-feladatok elvégzésének hatékonyságát növelni.

- Áttekintést adtam a worst-case analízis alkalmazási területeiről, és néhány analízismódszert is megemlítettem.
- Felvázoltam a kitűzött feladat motivációit, összevetettem a numerikus szimuláció és az analitikus módszerek előnyeit, hátrányait.
- Bemutattam már létező megoldásokat, értekeztem ezek előnyeiről és hátrányairól.
- Az előírt követelményeknek megfelelően specifikáltam és megterveztem a szoftverrendszert.
- Elvégeztem a feladat részekre bontását, definiáltam a programkomponensek részfeladatait, működését.
- Bemutattam a feladat megoldásához felhasznált eszközöket, ezek főbb tulajdonságait egy áramköri példa segítségével.
- A feladat megoldásának főbb lépéseit ismertettem, bemutattam a részfeladatokhoz kapcsolódó problémákat, ezekre megoldásokat javasoltam.
- Az elkészült program működését ellenőriztem, és számszerűsíthető eredményekkel támasztottam alá a megoldásom helyességét.

6.1. Továbbfejlesztési lehetőségek

A program a dolgozat írásakor lineáris áramkörök számítását támogatja. Fejlesztési lehetőségek közé tartozik nemlineáris komponensek, például félvezetők karakterisztikáinak analitikus leírása, vagy amennyiben ez valami miatt nem lehetséges, az áramkört munkaponti linerizálást követően modellezni MATLAB-ban a félvezetők linearizált helyettesítőképeinek felhasználásával.

A példában frekvenciatartománybeli analízisre való alkalmazást mutattam be. Ha más típusú analízis módszert szeretnénk alkalmazni, másféle egyenleteket kell felírni ezekhez pedig adott esetben más alkatrészmodellek és megoldó algoritmusok szükségesek. (Jellemzően időtartománybeli vagy tranziens analízis viselkedést szokás még vizsgálni.)

Dolgozatomban az analitikus számítási módszer hangsúlyos szerepet kapott. Érdemes a továbbiakan utánajárni, hogy MATLAB-ban milyen numerikus módszerek állnak rendelkezésre, és lehet-e a segítségükkel áramköranalízis-feladatokat hatékonyabban elvégezni, mint LTspice-ban.

Említettem a párhuzamos (tömbös) kiértékelés lehetőségét, ezt a kérdéskört érdemes lehet jobban körüljárni, hiszen még jobban lerövidítené az analízis elvégzésének idejét.

Köszönetnyilvánítás

Szeretnék köszönetet mondani konzulensemnek, Dr. Orosz Györgynek, valamint doktoranduszhallgató ismerőseimnek. Hálás vagyok nekik a felkészülésben nyújtott segítségükért.

A 2019-1.3.1-KK-2019-00004 számú projekt a Nemzeti Kutatási Fejlesztési és Innovációs Alapból biztosított támogatással, a 2019-1.3.1-KK pályázati program finanszírozásában valósult meg.

Irodalomjegyzék

[1] Joshua Cantrell: Writing Simple Spice Netlists, 2001. 08.
URL https://eee.guc.edu.eg/Courses/Electronics/ELCT503%20Semiconductors/Lab/spicehowto.pdf.

[2] Myriam Charras-Garrido1 – Pascal Lezaud: Extreme Value Analysis: an Introduction. *Journal de la Société Française de Statistique*, 154. évf. (2013) 2. sz., 72–76. p. ISSN: 2102-6238.

[3] Analog Devices Corporation: .LIB – Include a Library, 1998-2018. URL https://ltwiki.org/LTspiceHelp/LTspiceHelp/_LIB_Include_a_library.htm.

[4] Analog Devices Corporation: LTspice XVII Help, 1998-2018.
URL https://ltwiki.org/files/LTspiceHelp.chm.html.

[5] Fabrice Salvaire et. al.: PySpice : Simulate Electronic Circuit using Python and the Ngspice / Xyce Simulators.
URL https://github.com/PySpice-org/PySpice. hozzáférés dátuma 2023. 03.

[6] Analog Devices Inc.: Ltspice. URL https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html. Hozzáférés dátuma 2023. 01.

[7] The MathWorks Inc.: Evaluate matlab expression.
URL https://www.mathworks.com/help/matlab/ref/eval.html.

[8] The MathWorks Inc.: Simscape.
URL https://www.mathworks.com/products/simscape.html.

[9] The MathWorks Inc.: Symbolic math toolbox.
URL https://www.mathworks.com/products/symbolic.html.

[10] Laurence W. Nagel – D.O. Pederson: SPICE (Simulation Program with Integrated Circuit Emphasis). UCB/ERL M382. Jelentés, 1973. Apr, EECS

Department, University of California, Berkeley. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html.

[11] T. Quarles – D. Pederson nad R. Newton – A. Sangiovanni-Vincentelli – Christopher Wayne.: CIRCUIT ELEMENTS AND MODELS. URL http://bwrcs.eecs.berkeley.edu/Classes/IcBook/SPICE/UserGuide/elements_fr.html. Hozzáférés dátuma 2023. 06.

[12] KiCad Development Team: Spice simulation.
URL https://www.kicad.org/discover/spice/. Hozzáférés dátuma 2023. 10.

[13] Stanford University: SPICE 'Quick' Reference Sheet, 2001. 01. URL https://web.stanford.edu/class/ee133/handouts/general/spice_ref.pdf.

[14] Giuseppe Venturini – Ian Daniher – Rob Crowther és mások: Ahkab: A SPICE-like electronic circuit simulator written in Python.
URL https://ahkab.github.io/ahkab/. Hozzáférés dátuma 2023. 03.