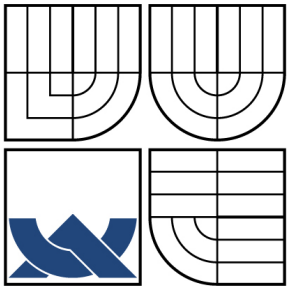
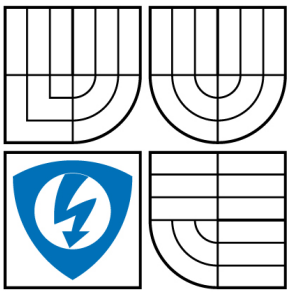


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF RADIO ELECTRONICS

TOOLBOX PRO SPOLUPRÁCI MATLABU S EXTERNÍMI SIMULAČNÍMI PROGRAMY

TOOLBOX FOR THE COOPERATION OF MATLAB AND EXTERNAL SIMULATION PROGRAMS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

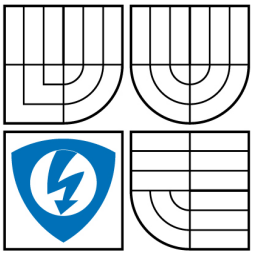
AUTOR PRÁCE — Bc. PETR MORAVEC

AUTHOR

VEDOUCÍ PRÁCE — Ing. LUKÁŠ OLIVA

SUPERVISOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav radioelektroniky

Diplomová práce

magisterský navazující studijní obor
Elektronika a sdělovací technika

Student:	Bc. Petr Moravec	***ID:***	77800
Ročník:	2	***Akademický rok:***	2008/2009

NÁZEV TÉMATU:

Toolbox pro spolupráci MATLABu s externími simulačními programy

POKYNY PRO VYPRACOVÁNÍ:

Analyzujte skriptovací rozhraní minimálně dvou programů pro vlnovou analýzu elektromagnetických struktur. Zaměřte se na možnosti ovládání těchto programů přes skriptovací jazyky a systémová rozhraní. Popište možnosti propojení programů s MATLABem pro Windows (vytváření geometrických objektů, spouštění analýzy, import výsledků). Popis ilustrujte komentovanými skripty pro úplnou analýzu vybraného problému, import a zpracování výsledků v MATLABu.

Na základě získaných poznatků navrhněte a implementujte datové struktury a funkce pro vytvoření propojovací vrstvy mezi MATLABem a simulačními programy; vrstva musí umožnit jednotný přístup k ovládání simulačních programů. Vrstva musí být podrobně popsána v programátorské a uživatelské příručce. Funkčnost vrstvy musí být ověřena optimalizací vybraných struktur vybranou globální optimalizační metodou.

Po dohodě s vedoucím implementujte další globální optimalizační metody. Navrhněte a implementujte postup pro porovnání výkonnosti optimalizačních algoritmů na jednoduchých elektromagnetických problémech. K jejich porovnání využijte základní statistické charakteristiky relevantní pro porovnání stochastických optimalizačních algoritmů.

DOPORUČENÁ LITERATURA:

[1] IVACHIV, P., KRAVAL, I., Základy komponentní technologie COM. Brno: Computer Press, 1999. 243 s. ISBN 80-7226-101-0.

Termín zadání:	9.2.2009	***Termín odevzdání:***	29.5.2009

Vedoucí práce: Ing. Lukáš Oliva

prof. Dr. Ing. Zbyněk Raida
předseda oborové rady

UPOZORNĚNÍ:

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení:	Bc. Petr Moravec
Bytem:	Konojedská 17,Praha, 100 00
Narozen/a (datum a místo):	19. března 1981 v Praze

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií
se sídlem Údolní 53, Brno, 602 00
jejímž jménem jedná na základě písemného pověření děkanem fakulty:
prof. Dr. Ing. Zbyněk Raida, předseda rady oboru Elektronika a sdělovací technika
(dále jen „nabyvatel“)

Čl. 1

Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):

   ☐ disertační práce
   ☒ diplomová práce
   ☐ bakalářská práce
   ☐ jiná práce, jejíž druh je specifikován jako .....
   (dále jen VŠKP nebo dílo)

Název VŠKP:	Toolbox pro spolupráci MATLABu s externími simulačními programy
Vedoucí/ školitel VŠKP:	Ing. Lukáš Oliva.
Ústav:	Ústav radioelektroniky
Datum obhajoby VŠKP:	

   VŠKP odevzdal autor nabyvateli*:

   ☒ v tištěné formě – počet exemplářů: 2
   ☒ v elektronické formě – počet exemplářů: 2

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.

3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.

4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

* hodící se zaškrtněte

Článek 2

Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.

2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.

3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti

   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)

4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3

Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.

2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.

3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.

4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: 29. května 2009

……..	……
Nabyvatel	Autor

Abstrakt

V diplomové práci je popsáno skriptovací rozhraní dvou programů CST a HFSS pro vlnovou analýzu elektromagnetických struktur. Práce je zaměřena především na možnosti ovládání těchto programu přes skriptovací jazyky a systémové rozhraní operačního systému Microsoft Windows. Popisuje postup propojení programů s MATLABem a ukazuje ho na komentovaných skriptech pro úplnou analýzu vybraného problému, import a zpracování výsledku v MATLABu.
Dále jsou navrženy a implementovány funkce tvořící propojovací vrstvu mezi MATLABem a simulačními programy. Vrstva umožňuje úplné ovládání simulačních programů publikované v dostupné dokumentaci k programům. Vrstva je podrobně popsána v referenční příručce a je využita k optimalizaci modelu planární antény metodami Particle swarm optimization (PSO). Dále je prezentováno další využití vrstvy pro porovnatelné srovnání implementací globálních optimalizačních metod – SOMA a Diferenciální evoluce včetně návrhu postupu pro porovnání výkonnosti optimalizačních algoritmů na jednoduchých elektromagnetických modelech.

Klíčová slova

CST, HFSS, MATLAB, VBA, PSO, DE, SOMA, TOOLBOX

Abstrakt

In this Master's thesis scripting interface of two programs CST Microwave studio and Ansoft HFSS for the purpose of analysis of electromagnetic structures is described. The work is focuses control of these programs with help of scripting languages and system's interface of MS Windows XP. Next the process of connecting programs with MATLAB is shown on commented scripts together with an example of complete analysis of a chosen problem, and the import and export of results results in MATLAB.

Further the functions which form programming interface between MATLAB and simulation programs are designed and implemented. The interconnection layer makes the complete control of simulating programs possible using the function description published in the official documentation of used simulation programs. The layer is described in reference manual in detail and it is used for optimization with use of Particle swarm optimalization (PSO) of planar antenna model. Then there is presented another usage of the layer for an implementation of global optimization methods - SOMA and DE including suggestion of process for comparison efficiency of optimization algorithms on simple electromagnetic models.

Key Words

CST, HFSS, MATLAB, VBA, PSO, DE, SOMA, TOOLBOX

Bibliografická citace

MORAVEC, P. *Toolbox pro spolupráci MATLABu s externími simulačními programy.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 76 s. Vedoucí diplomové práce Ing. Lukáš Oliva.

Prohlášení

Prohlašuji, že svou diplomovou práci na téma toolbox pro spolupráci MATLABu s externími simulačními programy jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 29. května 2009 .....

podpis autora

Poděkování

Děkuji vedoucímu diplomové práce Ing. Lukáši Olivovi za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne 29. května 2009 .....

podpis autora

Úvod

Diplomová práce se zaměřuje na sestavení Toolboxu pro práci s komerčními simulačními programy a na jeho využití pro globální optimalizace elektromagnetických struktur. Dále je využit pro porovnání tří optimalizačních metod na elektromagnetickém problému. Výsledky porovnání jsou srovnány s porovnáním metod pomocí optimalizačních funkcí.

V první kapitole jsou popsány rozdíly mezi způsoby softwaru a rozdíly mezi jednotlivými aplikacemi, pomocí nichž je možné následně skládat softwarové celky spolu s principem komponentních technologií a historii jejich vývoje.

Ve druhé kapitole je popsáno rozhraní mezi prostředím MATLAB a dvěma externími programy Ansoft HFSS a CST Microwave Studio, které jsou určené pro numerickou simulaci elektromagnetických struktur. Simulace elektromagnetických mikrovlnných struktur založených na Maxwellových rovnicích je důležitou součástí řešeného problému. Přesnost a správnost výsledků simulace je základem použití pro náhled před vlastním vyrobením skutečného simulovaných struktur.

Často využívaným skriptovacím jazykem v elektromagnetických simulačních programech dostupných na UREL FEKT VUT je Visual Basic, respektive jeho varianta Visual Basic for Applications (dále jen VBA). Je zde proto uveden návod k přístupu pomocí VBA přes rozhraní systému Microsoft Windows XP k simulačním programům obsahující konkrétně vytváření geometrických objektů, spouštění simulací, importu výsledků, a využití dokumentovaných funkcí dostupných z jazyka VBA. Popis je ilustrován komentovanými skripty na jednoduché ukázce pro oba externí simulační programy.

Jedním z podstatných důvodů pro vytvoření Toolboxu je malá rozšířenost schopností programování mezi uživateli elektromagnetických simulačních programů ve VBA v porovnání s MATLABem. Proto je předveden způsob, kterým lze konvertovat skript z VBA do MATLABu včetně komentované ukázky.

Ve třetí kapitole je popsán Toolbox tvořený propojující MATLAB a oba simulační programy. Funkce Toolboxu jsou předvedeny na krátkých ukázkách zdrojového kódu v MATLABu. Toolbox je použitelný především k optimalizacím.

Ve čtvrté kapitole jsou popsány globální optimalizační algoritmy implementované v Toolboxu a využité v závěrečném srovnání a v páté testovací funkce. Mezi použitými optimalizačními algoritmy jsou PSO (metoda roje částic), DE (diferenciální evoluce) a SOMA (samoorganizující se a migrační algoritmus). Tyto optimalizační algoritmy jsou použity k ověření činnosti Toolboxu. Dále je na statistických charakteristikách ukázána výkonnost těchto optimalizačních algoritmů na testovacích funkcích a na reálném modelu.

1. Historie vývoje softwaru

1.1 Historicky prvotní tvorba softwaru

Mezi první druhy programu patří tzv. Monolit. Monolit lze chápat jako jeden zdrojový text s jedním algoritmem, který byl psán programovacím jazykem assembler [1].

Dalším prvotním softwarem je strukturovanost programu. Strukturovanost programu je program vznikající jako skladba znovupoužitelných funkcí a procedur, vyjádřených pomocí blokových schémat, pomocí rozkladu struktur procesů a rozkladu struktur dat. V období vzniku této éry se začínají se objevovat moduly v daném jazyce nazývané module, unit apod. jako pokus o rozložení systému a jeho opětné složení, tj. první náznak analýzy a syntézy programu. Vznikají znovupoužitelné knihovny, ale jsou to pouze knihovny myšlenek před kompilací. Výsledný produkt po kompilaci je opět jeden EXE-soubor, takže o skládání výsledného softwaru nemůže být řeč [1].

1.2 Objektové programování

Oproti předchozí zmíněné strukturovatelnosti programu zde vznikají objektové knihovny. Oproti strukturálním knihovnám se již myšlenky neformulují v podobě funkcí, procedur a dat systému, ale vyjadřují se pomocí objektů. Modelování pomocí rozkladu struktur procesů a struktur dat je nahrazeno lépe analyticky vyjádřenou strukturou objektů při využití objektových modelů, scénářů spolupráce objektů, apod. Objektové knihovny také skládají systém, ale podobně jako u strukturálního přístupu se jedná o skládání v první myšlenkové fázi před vložením do jednoho celku. Po kompilaci vzniká opět jeden EXE-soubor [1].

1.3 Vznik dynamických knihoven

Zde se začínají objevovat první snahy poskládat softwarový produkt až po kompilaci, tj. dynamicky navázat jeho části i po zrodu spustitelných souborů. Vznikají knihovny DLL. Po kompilaci už není výsledný produkt reprezentován jedním souborem, ale několika soubory. Výměnou některého souboru lze bez následné kompilace změnit chování aplikace bez ztráty funkcionality celé aplikace. Není potřeba znovu kompilovat cely návrh, stačí pouze vyměnit soubory. Oproti předešlým postupům je zřejmý: Zatímco v předešlých aplikacích se systém musel poskládat před kompilací a vznikl jeden spustitelný soubor, nyní je složen po kompilaci z několika souborů, které si mezi sebou rozumějí na nejnižší strojové binární úrovni [1].

1.4 Komponentní technologie

Komponentní technologie je možno popsat jako software navržený pomocí objektového programování, kde se vytvářejí softwarové součástky, nichž se skládají větší a ještě větší kusy softwaru. Nakonec je výsledný softwarový produkt složen se součástek [1].

1.5 Porovnání typů aplikací

Obr. 1a-c Návrhy aplikací [1]

Na obr. 1a je vidět výsledný binární kód jako jeden spustitelný soubor. Ve vnitřní aplikaci nemá smysl hledat jakékoli její součásti. U tohoto typu aplikace nelze v žádném případě hovořit o tom, že by měla být složena y již hotových částí [1].

Na obr. 1b je vidět druhý možný způsob, kdy se aplikace skládá z definovaných částí, ale toto skládání nemá povahu skládání komponent, protože knihovna DLL zpřístupňuje pouze systémové funkce a proměnné globálního charakteru. Nevytvářejí se jednotlivé instance součástek, netvoří se součástky, ale skládáním se zpřístupňují pouze globální funkce a proměnné [1].

Na obr. 1c je podstatně rozdílná situace. Při použití komponentní technologie ve výrobě softwaru se výsledný produkt skládá ze součástek, což jsou nejen uzavřené a vyměnitelné části systému, ale jsou také identifikovatelné i ve svých verzích a použitelné v několika instancích. Můžeme použít tolik součástek daného typu, kolik potřebujeme [1].

1.6 Historie vývoje technologie COM a OLE

Každý vývoj, pokud je dělán dobře a správnými postupy, je spojen s neustálou snahou využít obecnějších řešení k řešení speciálních problémů. Správně učiněné zobecnění využívá dosažené řešení ke stále širšímu použití. Dodržování tohoto principu vede ke známému požadavku na vývojové práce: Pokud zjistíme, že problém je součástí obecnějšího řešení, je třeba provést nejprve zobecnění, vyřešit vše na této úrovni, a teprve pomocí tohoto výsledku vyřešit i náš speciální případ. Tímto postupem se neustále rozlišuje obzor našeho poznání a získáváme tak nástroje pro řešení dalších úkolů [1].

A právě k této situaci došlo při vývoji COMu. Při řešení určitých problémů se začala projevovat nutnost zobecnit řešení a postupně se tak historicky vyvíjely jednotlivé technologie. Při studia technologie COM dochází většinou k určitým zmatením nad pojmy. Jedna se nedorozumění ve vztahu a významu těchto pojmů:

- OLE (Object Linking and Embedding)
- COM

Technologie OLE byla stvořena v roce 1991. Původním záměrem bylo vyřešit problém s tzv. složeným dokumentem (angl. *compound document*). Když v reálném životě pracujeme s nějakými dokumenty, jedná se z 99 % o dokumenty složené z různých typů informace, jako jsou texty, obrázky, zvuky atd. Do té doby se pro práci s každým typem

dokumentu muselo pracovat svým zvláštním nástrojem, který si rozuměl pouze s daným typem dat. To se sice jevilo v programování jako normální postup, ale ve své podstatě se jedná o velmi nelogický. Proč ba dokument nemohl obsahovat několik typů informací? Problém však není v samotných nástrojích, které dokumenty zpracovávají, ale v „různosti“ dokumentů. Dokument složený z různých typů byl reprezentován skupinou souborů a každému z nich rozuměl určitý nástroj. Tento přístup je z hlediska uživatele nejen nepohodlný, ale i nepřirozený. Pro vyřešení tohoto problému vznikla technologie známá jako OLE, později označovaná jako OLE 1. Pomocí této technologie mohl uživatel například používat spreadsheet Excelu uvnitř dokumentu Wordu [1].

Technologie OLE 1 byla v té době postavena na využití způsobu komunikace mezi aplikacemi prostřednictvím rozhraní DDE (Dynamic Data Exchange). Toto řešení bylo velice složité pro programátory, kteří by jej chtěli realizovat ve svých aplikacích, na druhou stranu bylo také pomalé a nedokonalé. Však to byla pouze první verze OLE [1].

Nashromážděním těchto problémů vznikl požadavek na nový přístup k řešení a nejen na vylepšení technologie. Bylo nutné od základů vyvinout novou technologii, a proto se začali klást požadavky, které by tato nová technologie měla splňovat [1].

V první řadě musela nová technologie vyřešit nedostatky OLE 1 a zavést nové způsoby komunikace mezi aplikacemi – jiné než přes zatím používané rozhraní DDE. Dále se ukázalo, že není žádoucí vyřešit pouze složení dokumentů, ale je potřebné zavést spolupráci mezi aplikacemi různými způsoby. Vyvstaly tak další problémy k řešení, například možnost ovládání aplikací jako Word či Excel z jiných uživatelských aplikací [1].

Snaha o vyřešení všech těchto problémů „od základu“ vedla k zavedení nových technologií a ke vzniku OLE 2 jako návaznosti nové verze na OLE 1. Tedy „nová verze“ OLE označená jako OLE 2 je již postavena na nových technologiích a jednou z nich je technologie COM. Nová verze OLE 2 je o mnoho dokonalejší, než verze OLE 1 a mimo jiné již v sobě obsahuje technologii COM. Technologie OLE 2 nabídla potenciál k novému způsobu přístupu k řešení interakce mezi jednotlivými částmi softwaru. Tento potenciál se nakonec zhodnotil do nové technologie – COM. Tato technologie zastřešila všechny přístupy k řešení spolupráce mezi všemi druhy softwaru (dokumenty, knihovnami, objekty, aplikacemi, systémovým softwarem atd.) pod jedno paradigma [1].

Díky zavedení technologie COM se v té době také změnil pohled na pojem OLE- nejednalo se již pouze o řešení složených dokumentů, ale o mnohem obecnější řešení. Proto Microsoft tehdy rozhodl přívlastkem OLE (již bez verze) označit cokoli, co je postaveno na základech technologie COM. V této fázi nebylo OLE pouze názvem pro složené dokumenty, ale získalo „obecnější povahu“ [1].

Avšak i tento pohled se již stal historií. Protože se základní technologií propojování různých částí softwaru stala komponentní technologie COM, vrátil se opět název OLE jako označení pouze Object Linking and Embedding. Zastřešující technologií propojení různých kusů softwarů se stala technologie COM [1].

1.7 Technologie Active X

Dalo by se říci, že ActiveX je něco mezi DLL a samostatnou aplikací. DLL je knihovna funkci s přesně definovaným rozhraním, kterou lze včlenit do více aplikací a v paměti je jen jednou. Programátor ji může použít, aniž by znal její zdrojový kód - stačí mu vědět, jaké funkce implementuje a jak je zavolat. DLL se nahrává do paměti jen jednou a pak už jen čeká, kdo použije její funkce. ActiveX je autonomnější - může implementovat i uživatelské rozhraní, takže např. existuje ActiveX komponenta implementující jednoduchý tabulkový procesor či přehrávač medii. Ten je pak možno vložit do jiné aplikace, ale po svém spuštění funguje autonomně [2].

2. Simulační programy a skriptovací rozhraní

Jako programy pro elektromagnetickou simulaci byly vybrány Ansoft HFSS (HFSS) a CST Microwave studio (CST). Tyto dva programy umožňují trojrozměrnou simulaci elektromagnetických struktur. K simulaci využívají numerických metod postavených na Maxwellových rovnicích popisujících fyzikální jevy elektromagnetismu. Dalším důvodem výběru těchto simulačních programů byla možnost využití skriptovacího rozhraní. V obou simulačních programech je možnost skriptování v programovacím jazyce VISUAL BASIC. Oba programy mají funkce pro elektromagnetickou simulaci - vytváření geometrických objektů, nastavování okrajových podmínek, spouštění analýzy a mnoho dalších. Přehled všech implementovaných funkcí lze najít v dokumentaci programu [3][4].

Důležitou výhodou MATLABu proti VBA je možnost interaktivního zadávání příkazů (Read Evaluate Print Loop), které oproti VBA usnadňuje vývoj skriptů. Skriptem se rozumí soubor obsahující jednotlivé příkazy, prováděné interpreterem. MATLAB dále obsahuje podstatně lepší možnost grafického znázornění výsledků a statistického zpracování.

2.1 Skriptování v CST pomocí VBA

Pro používání skriptovacího rozhraní je nutné mít funkční program CST. Přístup ke skriptovacímu rozhraní je takový, že je nutné nejprve spustit program CST, kde v hlavním menu je záložka makra (MACROS). V této záložce je položka pro vytvoření makra (Make VBA macro…), při zvolení této položky se otevře okno, ve kterém je možné psát programovacím jazykem VISUAL BASIC. Toto okno je rozděleno čarou na dvě části, kde první část slouží k deklarování proměnných a druhá část je hlavní funkce programu (MAIN). Spuštění skriptu se provádí ikonkou spustit makro (RUN MACRO). Na obr. 2 je ukázka jednoduchého makra, které umí vykreslit geometrický objekt.

```
' patch
' Deklarování použitých proměnných jako objekt
Dim cst As Object
Dim mws As Object
Dim Brick As Object
' Hlavní cyklus
Sub Main ()
' Vytvoření objektu, objekt přistupující k aplikaci CST
Set cst = CreateObject("CSTStudio.Application")
' Vytvoření nového listu v programu CST
Set mws = cst.NewMWS
' Vytvoření geometrického objektu
With mws.brick
    .Reset
    ' Nastavení jména objektu
    .Name "KRYCHLE"
    ' Nastavení rozsahu ve směru X
    .Xrange "0", "2"
    ' Nastavení rozsahu ve směru Y
    .Yrange "0", "2"
    ' Nastavení rozsahu ve směru Z
    .Zrange "0", "2"
    ' Příkaz vykreslující zadaný objekt z předchozích
    ' nadefinovaných hodnot
    .Create
 End With

End Sub
```

Obr. 2 Ukázka zdrojového kódu skriptování v CST

2.2 Skriptování v HFSS pomocí VBA

Opět důležitým předpokladem k tomu, aby bylo možné skriptovat je, že musí být dostupný program HFSS. Způsob skriptování je zde trochu jiný než je u programu CST. Existují dva způsoby, jak použít skript k ovládání programu HFSS. Prvním způsobem je vytvoření nového textového souboru. Tento soubor včetně přípony se přejmenuje na žádaný název a koncovku „.vbs“, např: „antena.vbs“. Do tohoto nově vytvořeného souboru se vloží skript tvořený z VBA, viz obr. 3.

```
' Definování proměnných
Dim oHfssApp
Dim oDesktop
Dim oProject
Dim oDesign
Dim oEditor

' Vytvoření objektu přistupující k programu HFSS
' Nastavení výchozího okna v HFSS
Set oApp  = CreateObject("AnsoftHfss.HfssScriptInterface")
Set oDesktop = oApp.GetAppDesktop()
oDesktop.RestoreWindow
oDesktop.NewProject
Set oProject = oDesktop.GetActiveProject

' Nastavení pojmenování
oProject.InsertDesign "HFSS", "Example_antena", "DrivenModal", ""
Set oDesign = oProject.SetActiveDesign("Example_antena")
Set oEditor = oDesign.SetActiveEditor("3D Modeler")

' Vytvoření geometrického objektu, kde jsou zapsány
' informace o pojmenování objektu, počáteční pozice,
' velikosti stran od počáteční pozice ve směru X,Y a Z,
' nastavení barvy, průhlednost objektu, materiál a jaké
' souřadnice se mají použít.
oEditor.CreateBox _
Array("NAME:BoxParameters", _
"XPosition:=", "-13.5mm", _
"YPosition:=", "-15.0mm", _
"ZPosition:=", "-24.0mm", _
"XSize:=", "29.5mm", _
"YSize:=", "30.0mm", _
"ZSize:=", "48.0mm"), _
Array("NAME:Attributes", _
"Name:=", "AirBox", _
"Flags:=", "", _
"Color:=", "(132 132 193)", _
"Transparency:=", 0.75, _
"PartCoordinateSystem:=", "Global", _
"MaterialName:=", "vacuum", _
"SolveInside:=", true)
```

Obr. 3 Ukázka zdrojového kódu skriptování v HFSS.

Nově vytvořený a vyplněný soubor lze jednoduše spustit pouhým kliknutím v prostředí MS Windows. Druhým způsobem spuštění VBA skriptu je spuštění programu HFSS, kde v hlavním menu v záložce nástroje (TOOLS) je položka spustit skript (RUN SCRIPT). Program HFSS nabídne okno k nalezení cesty, pomocí niž lze vybrat požadovaný skript s koncovkou „.vbs“. Po vybrání se skript provede.

2.3 Rozhraní mezi programem MATLAB a simulačními programy

Rozhraní mezi programem MATLAB a externími simulačními programy je tvořeno komponentou Active X operačního systému MS Windows. Pro vytvoření komponenty Active X v programu MATLAB není nic jednoduššího, než použít implementovanou funkci „actxserver“. Funkce slouží k vytvoření komponenty Active X. Další funkcí, která je potřeba pro rozhraní mezi programem MATLAB a simulačními programy je funkce „invoke“, která je rovněž implementována v programu MATLAB a slouží k odesílání příkazů danému

simulačnímu programu. Nyní zde bude uvedena syntaxe a popis jednotlivých funkcí nutné k vytvoření skriptovacího rozhraní v programu MATLAB.

Funkce **ACTXSERVER**:
Syntaxe:

```
h = actxserver('progid')
```

Popis funkce:

`h = actxserver('progid')` vytváří místní OLE automatizační server, kde *`progid`* je programový identifikátor COM serveru, a *h* řídí vlastní interface pomocí serveru[4].

Funkce **INVOKE**:
Syntaxe:

```
S = h.invoke('methodname')
```

Popis funkce:

`S = h.invoke('methodname')` dovolává se metody specifikované v řetězci *`methodname`*, a vrátí výstupní hodnotu. Vrácená hodnota záleží na specifické metodě, která byla vyvolána, a je určena konkrétním ovládáním nebo serverem [4].

2.4 Skriptování v simulačních programech pomocí MATLABu

Nutnou podmínkou k tomu, aby bylo vůbec možné vytvořit skriptovací rozhraní a skriptovat v něm je funkční program MATLAB a také simulační program CST nebo HFSS. Skriptování v programu MATLAB se provádí tím způsobem, že je nejprve nutné vytvořit Active X server, čímž vznikne propojení mezi MATLABem a simulačním programem. Když se vytvoří toto spojení, lze už posílat jednotlivé příkazy do příslušného programu a daný program už vykonává jednotlivé příkazy přicházející přes toto vytvořené skriptovací rozhraní.

Nyní zde bude uvedena ukázka krátkého zdrojového kódu, která je psána v programu MATLAB. V ukázce je ukázáno, jakým způsobem je možné vytvořit Active X server a odesílat příkazy pomocí tohoto serveru k danému simulačnímu programu. Nejprve bude krátká ukázka a popis zdrojového kódu ukázán pro program CST a poté pro program HFSS.

2.5 Ukázka zdrojového kódu pro ovládání programu CST z MATLABu

Připojení ke COM serveru pomocí Active X z programu MATLAB.

```
cst = actxserver('CSTSTUDIO.application');
```

Otevření nového projektu pomocí programu MATLAB.

```
mws = cst.invoke('NewMWS');
```

Definice geometrického objektu krychle pomocí programu MATLAB.

```
brick = mws.invoke('Brick');
    brick.invoke('Reset');
    brick.invoke('Name','KRYCHLE');
```

```
brick.invoke('Xrange',0,2);
brick.invoke('Yrange',0,2);
brick.invoke('Zrange',0,2);
brick.invoke('Create');
```

Kde proměnné *cst* a *mws* jsou objekty. Objekt *cst* je pro vytvoření spojení s COM serverem pomocí komponenty Active X a objekt *mws* je pro vytvoření nového projektu (okna) v programu CST. Dále objekt označený jako *brick* je potřeba k vytvoření geometrického objektu (krychle) v programu CST. Uvedený řetězec `('Brick')` říká programu CST, o který objekt půjde. Jméno *Name* je pojmenování objektu. Rozměry *Xrange*, *Yrange*, *Zrange* popisují velikost stran daného objektu. Vytvoření *Create* je posledním bodem těchto geometrických objektů, který provede vykreslení ze zadaných předchozích hodnot do grafického prostředí. Je zde nutné vědět, že při tomto sestavování zdrojového kódu se zde používají dva druhy proměnných. Prvním typem je řetězec *string*. Tento první typ proměnné se vždy vkládá mezi apostrofy, např: `('Reset')`. Druhým typem jsou číselné proměnné mezi které patří (Integer, Double, Float). Tento druh proměnných se ve skriptu nevkládá mezi žádné znaky, jsou pouze odděleny čárkou, např: `('Yrange',0,2)`.

2.6 Ukázka zdrojového kódu pro ovládání programu HFSS z MATLABu

Připojení ke COM serveru pomocí Active X z programu MATLAB.
Vytvoření objektu přistupující k programu HFSS
Nastavení výchozího okna v HFSS

```
App = actxserver('AnsoftHfss.HfssScriptInterface');
Desktop = App.GetAppDesktop();
Desktop.RestoreWindow;
Desktop.NewProject;
HProject = Desktop.GetActiveProject;
```

Nastavení pojmenování projektu

```
invoke(HProject,'InsertDesign','HFSS','Example_antena','DrivenModal','');
Design = HProject.invoke('SetActiveDesign','Example_antena');
Editor = Design.invoke('SetActiveEditor','3D Modeler');
```

Vytvoření geometrického objektu v programu HFSS, kde jsou zapsány informace o pojmenování objektu, počáteční pozice, velikosti stran od počáteční pozice ve směru X,Y a Z, nastavení barvy, průhlednost objektu, materiál a jaké souřadný systém se má použít.

```
 invoke(Editor,'CreateBox',...
{'NAME:BoxParameters',...
'XPosition:=','-37.7mm',...
'YPosition:=','-28.4mm',...
'ZPosition:=','0mm',...
'XSize:=','113.1mm',...
'YSize:=','85.2mm',...
'ZSize:=','1.6mm'},...
{'NAME:Attributes',...
'Name:=','Dielektrikum',...
'Flags:=','',...
'Color:=','(132 133 193)',...
'Transparency:=',0.75,...
'PartCoordinateSystem:=','Global',...
'MaterialName:=','FR4_epoxy',...
'SolveInside:=',true})
```

Kde proměnné *app*, *Hproject, Design a Editor* jsou objekty. Objekt *app* je pro vytvoření spojení s COM serverem pomocí komponenty Active X a objekt *Hproject, Design a Editor* slouží k vytvoření nového projektu (okna) v programu HFSS. Řádek kódu začínající (`invoke(Editor,'CreateBox',...)` slouží k vytvoření geometrického objektu. Toto skriptovací rozhraní není tak intuitivní, jako je tomu u programu CST. Program HFSS pozná že se jedná o geometrický objekt podle řetězce, který je nazván jako (`'CreateBox'`). V tomto vytvoření geometrického objektu jsou parametry popisující vytvářený geometrický objekt, jehož součásti je počáteční pozice (x,y,z), délka strany (Δx, Δy, Δz), název geometrického objektu, barvu objektu, průhlednost, materiál, atd.

Je zde nutné vědět, že při tomto sestavování zdrojového kódu se zde používají opět dva druhy proměnných plus jeden míšený typ. Prvním typem je řetězec *string*. Tento první typ proměnné se vždy vkládá mezi apostrofy, např: (`'Global'`). Druhým typem jsou číselné proměnné mezi které patří (Integer, Double, Float). Tento druh proměnných se ve skriptu nevkládá mezi žádné znaky, jsou pouze odděleny čárkou, např: `'Transparency:=',0.75,`. Míšený typ je zde takový, že se používá při definování velikostí a pozic geometrických objektů. Tyto míšené typy jsou vlastně typu string, ale jsou složeny ze znaménka, číslice a nakonec jednotky, např: `'-37.7mm'`.

3. Toolbox

Toolbox em se rozumí knihovna funkcí napsaná v programu MATLAB, jenž dokáže ovládat oba již zmíněné simulační programy. Vzhledem k tomu, že oba simulační programy nemají totožné funkce které přistupují k jednotlivým objektům, bylo potřeba vytvořit knihovnu funkcí zvlášť pro simulační program CST, tak i pro HFSS. V Toolboxu nebyly implementovány všechny dostupné funkce, které jsou jednotlivými programy podporovány. Všechny dostupné funkce ovládající jednotlivé programy a jsou popsány vždy v helpu daného simulačního programu pod názvem skriptování ve VBA. Vytvořené funkce, které byly vytvořeny jsou schopny ovládat jednotlivé simulační programy, kde například umí vytvořit skriptovací rozhraní pomocí Active X serveru, definovat geometrické objekty, nastavovat okrajové podmínky, spouštět analýzu, zpracovávat výsledky a mnoho dalšího. Vytvořené funkce jsou popsány v kapitole referenční příručka v příloze.

Toolbox ovládá simulační programy jedním ze dvou možných způsobů. Prvním způsobem ovládání je klasické, pomocí grafického prostředí. Uživatel vytváří posloupnost příkazů pomocí grafických nabídek, jenž lze vykreslovat například geometrické objekty, nastavovat okrajové podmínky, spouštět analýzu atd. Druhým způsobem je využití Toolboxu. Pro vykreslování není vůbec nutné, aby uživatel měl spuštěný simulační program. Stačí pouze v programu MATLAB vytvořit sadu příkazů, které jsou schopny vytvořit stejnou posloupnost příkazů, jako je tomu při ovládání přes grafické rozhraní, bez jediného kliknutí v grafickém rozhraní simulačních programů.

Jednotlivé funkce Toolboxu jsou vždy tvořeny tak, že se skládají ze dvou částí. V první části funkce je vlastní vykonání požadované funkce s požadovanými parametry v daném simulačním programu. Druhá část funkce slouží k vytváření historie z posloupnosti jednotlivých příkazů. V následujícím textu je uvedena ukázka zdrojového kódu funkce pro vytvoření geometrického objektu pro CST a poté pro HFSS.

3.1 Ukázka zdrojového kódu funkce pro program CST

Hlavička CST funkce pro vytvoření geometrického objektu:

```
function CstTorus (fid,CName,CComponent,CMaterial,CAxis,COutrad,...
CInrad,CXcenter,CYcenter,CZcenter,CSegments)
```

Použité globální proměnné:

```
global mws torus;
```

Přímé vykonání funkce v programu CST

```
torus = mws.invoke('Torus');
torus.invoke('Reset');
torus.invoke('Name',CName);
torus.invoke('Component',CComponent);
torus.invoke('Material',CMaterial);
torus.invoke('Axis',CAxis);
torus.invoke('Outerradius',COutrad);
torus.invoke('Innerradius',CInrad);
torus.invoke('Xcenter',CXcenter);
torus.invoke('Ycenter',CYcenter);
torus.invoke('Zcenter',CZcenter);
torus.invoke('Segments',CSegments);
torus.invoke('Create');
```

Zápis do dočasného souboru (ukládání do historie)

```
fprintf(fid, '%% Vytvoreni torus\n');
fprintf(fid, 'torus = mws.invoke(''Torus'');\n');
```

```
fprintf(fid, 'torus.invoke(''Reset'');\n');
fprintf(fid, ['torus.invoke(''Name'',''',CName,''');\n']);
fprintf(fid, ['torus.invoke(''Component'',''',CComponent,''');\n']);
fprintf(fid, ['torus.invoke(''Material'',''',CMaterial,''');\n']);
fprintf(fid, ['torus.invoke(''Axis'',''',CAxis,''');\n']);
fprintf(fid, ['torus.invoke(''Outerradius''',',',num2str(COutrad),');\n']);
fprintf(fid, ['torus.invoke(''Innerradius''',',',num2str(CInrad),');\n']);
fprintf(fid, ['torus.invoke(''Xcenter''',',',num2str(CXcenter),');\n']);
fprintf(fid, ['torus.invoke(''Ycenter''',',',num2str(CYcenter),');\n']);
fprintf(fid, ['torus.invoke(''Zcenter''',',',num2str(CZcenter),');\n']);
fprintf(fid, ['torus.invoke(''Segments''',',',num2str(CSegments),');\n']);
fprintf(fid, 'torus.invoke(''Create'');\n');
```

3.2 Ukázka zdrojového kódu funkce pro program HFSS

Hlavička HFSS funkce pro vytvoření geometrického objektu:

```
function HfssBox(fid, HName, HMaterial, HStart, HSize, HUnits)
```

Globální proměnné:

```
global Editor
```

Přímé vykonání funkce v programu HFSS

```
invoke(Editor,'CreateBox',...
    {'NAME:BoxParameters',...
    'XPosition:=',[num2str(HStart(1)),HUnits],...
    'YPosition:=',[num2str(HStart(2)),HUnits],...
    'ZPosition:=',[num2str(HStart(3)),HUnits],...
    'XSize:=',[num2str(HSize(1)),HUnits],...
    'YSize:=',[num2str(HSize(2)),HUnits],...
    'ZSize:=',[num2str(HSize(3)),HUnits]},...
    {'NAME:Attributes',...
    'Name:=',HName,...
    'Flags:=','',...
    'Color:=','(132 132 193)',...
    'Transparency:=',0,...
    'PartCoordinateSystem:=','Global',...
    'MaterialName:=',HMaterial,...
    'SolveInside:=',true})
```

Zápis do dočasného souboru (ukládání do historie)

```
fprintf(fid,'%% Vytvoreni kvadru\n');
fprintf(fid,'invoke(Editor,''CreateBox'',...\n');
fprintf(fid,'{''NAME:BoxParameters'',...\n');
fprintf(fid,['''XPosition:='',''',num2str(HStart(1)),HUnits,''',...\n']);
fprintf(fid,['''YPosition:='',''',[num2str(HStart(2)),HUnits],''',...\n']);
fprintf(fid,['''ZPosition:='',''',[num2str(HStart(3)),HUnits],''',...\n']);
fprintf(fid,['''XSize:='',''',[num2str(HSize(1)),HUnits],''',...\n']);
fprintf(fid,['''YSize:='',''',[num2str(HSize(2)),HUnits],''',...\n']);
fprintf(fid,['''ZSize:='',''',[num2str(HSize(3)),HUnits],'''},...\n']);
fprintf(fid,'{''NAME:Attributes'',...\n');
fprintf(fid,['''Name:='',''',HName,''',...\n']);
fprintf(fid,'''Flags:='','''',...\n');
fprintf(fid,'''Color:='',''(132 132 193)'',...\n');
fprintf(fid,'''Transparency:='',0,...\n');
fprintf(fid,'''PartCoordinateSystem:='',''Global'',...\n');
fprintf(fid,['''MaterialName:='',''',HMaterial,''',...\n']);
fprintf(fid,'''SolveInside:='',true})\n\n');
```

3.3 Použití Toolboxu

Použití Toolboxu je podmíněno tím, že uživatel musí umět ovládat alespoň jeden ze simulačních programů. aby věděl co je cílem simulace a musí vědět jak nastavit některé hodnoty, například jak správně nastavit okrajové podmínky u simulované struktury. Uživatel si musí uvědomit, jaká je posloupnost příkazů v grafickém prostředí, protože při

použití Toolboxu není žádná ochrana proti nesprávné posloupnosti použitých funkcí (například prohození funkce pro nastavení jednotek a funkce pro vytvoření Active X serveru).

Použití Toolboxu je dále představeno na funkčním příkladu, proto zde uvedu zdrojový kód opět pro oba simulační programy v pořadí CST a HFSS. Zdrojové kódy jsou psány v programu MATLAB, jak již bylo zmíněno. Ukázka zdrojového kódu je vždy doplněna komentářem.

3.4 Využití Toolboxu při modelování v CST

Funkce *global* definuje globální proměnná, která je potřeba ve všech volaných funkcích.

```
global cst mws fid;
```

Funkce *addpath* slouží k načtení cest k adresářům, ve kterých jsou uloženy všechny funkce, které se využívají při sestavování simulačního modelu pomocí skriptu.

```
addpath('../component/');
addpath('../global/');
addpath('../check3d/');
addpath('../Import_Export/');
addpath('../material/');
addpath('../monitors/');
addpath('../operation on solids/');
addpath('../ports/');
addpath('../results/');
addpath('../solids/');
addpath('../solver/');
```

Zde jsou nadefinovány absolutní cesty k souborům, které jsou potřeba ke správné funkčnosti Toolboxu. Soubor označen jako *tmpScriptFile* slouží k vytváření historie z posloupnosti napsané ve skriptu (napsané níže). Další soubor *tmpAdrSoub*, který je dán absolutní cestou, je určen funkci *CstSaveAsProject(tmpAdrSoub);*, která provede uložení vykresleného modelu do tohoto předem nadefinovaného souboru. Nutné je vzít v úvahu o jakou verzi programu CST se jedná, protože pro verzi 2006 je koncovka ukládaného souboru (.mod) a pro verzi 2008 je koncovka (.cst). Při záměně dojde k chybě při ukládání a samotný program CST se dotazuje, zda chce uživatel uložit projekt, což je nežádoucí jev například u optimalizace. Posledním souborem s nadefinovanou absolutní cestou (*CFileName*) slouží k ukládání výsledků po provedení analýzy.

```
tmpScriptFile = 'c:/tmp_cst/matlab/tmpData.m';
tmpAdrSoub = 'c:/tmp_cst/matlab/test/test.mod';
CFileName = 'c:/tmp_cst/matlab/ExportCst_S11.txt';
```

Zde se provede vytvoření a otevření dočasného souboru (*tmpData.m*) na zvolené absolutní adrese. Do tohoto souboru je zapisována historie posloupnosti příkazů po sobě jdoucích v tomto skriptu. Přístup k tomuto souboru je pomocí proměnné (*fid*), která je nastavena jako globální proměnná, protože do tohoto souboru se zapisují posloupnosti volaných příkazů a přístup je potřeba v každé volané funkci.

```
fid = fopen(tmpScriptFile, 'wt');
```

V tomto bloku jsou nadefinovány požadované parametry simulovaného modelu. Nejsou zde uvedeny jednotky, protože se definují jednou z funkcí z Toolboxu.

```
del_zar=6;
sir=37.7;
tl_sub=1.6;
```

```
tl_zar=1;
vys=28.4;
x_port=6;
y_port=7.1;
```

V bloku, který je uveden níže je vidět posloupnost několika funkcí, využívající funkce Toolboxu k sestavení simulačního modelu v programu CST. Název funkce se vždy skládá ze dvou částí. První částí názvu je typ programu (například *Cst* nebo *Hfss*) ve kterém je použitá funkce podporována a druhou částí názvu je vlastní název funkce (převzato z helpu programu). Všechny popíši jen stručně, protože detailní popis je uveden v referenční příručce v příloze včetně popisu jednotlivých proměnných a také vždy s použitelnou ukázkou k použití.

Funkce *CstNewProject* slouží k vytvoření Active X serveru a dále k otevření a navázání spojení programu CST, včetně otevření nového projektu.

```
CstNewProject(fid);
```

Funkce *CstUnits* slouží k nastavení jednotek, kde lze nastavit pouze tři základní jednotky a ostatní jednotky jsou automaticky nastaveny simulačním programem.

```
CstUnits(fid,'easy','mm','gHz','ns');
```

Funkce *CstMaterialExist* načte požadovaný materiál. Je-li tato funkce použita vícekrát po sobě, funkce nejprve zjistí, zda požadovaný materiál už nebyl někdy dříve definován.

```
CstMaterialExist('Copper');
```

Funkce *CstNewComponent* slouží k vytvoření nové komponenty.

```
CstNewComponent(fid,'P_ant');
```

Funkce *CstBrick* definuje ze zadaných parametrů krychli.

```
CstBrick(fid,'Antena','P_ant','Copper',-sir/2,sir/2,-vys/2,vys/2,
tl_sub,tl_sub+0.035);
CstBrick(fid,'zarez1','P_ant','Vacuum',-sir/2,-(sir/2)+del_zar,-tl_zar/2,
tl_zar/2,tl_sub,tl_sub+0.035);
```

Funkce *CstSolid* slouží k vytváření zářezů v dané struktuře.

```
CstSolid(fid,'Subtract','P_ant','Antena','zarez1');
CstBrick(fid,'zarez2','P_ant','Vacuum',(sir/2)-del_zar,sir/2,-tl_zar/2,
tl_zar/2,tl_sub,tl_sub+0.035);
CstSolid(fid,'Subtract','P_ant','Antena','zarez2');
CstBrick(fid,'zarez3','P_ant','Vacuum',-tl_zar/2,tl_zar/2,-(vys/2),
-(vys/2)+del_zar,tl_sub,tl_sub+0.035);
CstSolid(fid,'Subtract','P_ant','Antena','zarez3');
CstBrick(fid,'zarez4','P_ant','Vacuum',-tl_zar/2,tl_zar/2,(vys/2)-del_zar,
vys/2,tl_sub,tl_sub+0.035);
CstSolid(fid,'Subtract','P_ant','Antena','zarez4');
CstNewComponent(fid,'G_ant');
CstBrick(fid,'Ground','G_ant','Copper',-(sir/2)-7.3,(sir/2)+12.25,
-(vys/2)-13.05,((vys/2)+8),-0.035,0);
CstMaterialExist('FR4');
CstNewComponent(fid,'S_ant');
CstBrick(fid,'Substrat','S_ant','FR-4 (lossy)',-(sir/2)-7.3,(sir/2)+12.25,
-(vys/2)-13.05,(vys/2)+8,0,tl_sub);
CstNewComponent(fid,'P_port');
CstBrick(fid,'Port1','G_ant','Copper',-(12.7/2)+x_port,(12.7/2)+x_port,
-(12.7/2)+y_port,(12.7/2)+y_port,-1.635,-0.035);
```

Funkce *CstCylinder* slouží k vytvoření válce ze zadaných parametrů.

```
CstCylinder(fid,'Konektor','G_ant','Copper','Z',5.3/2,0,x_port,y_port,
-9.635,-1.635,0);
CstCylinder(fid,'Dira','S_ant','Vacuum','Z',1.5/2,0,x_port,y_port,0,
tl_sub,0);
CstSolid(fid,'Subtract','S_ant','Substrat','Dira');
CstCylinder(fid,'Dira1','G_ant','Vacuum','Z',2/2,0,x_port,y_port,
-0.035,0,0);
CstSolid(fid,'Subtract','G_ant','Ground','Dira1');
CstCylinder(fid,'Dira2','G_ant','Vacuum','Z',4.3/2,0,x_port,y_port,-1.635,
-0.035,0);
CstSolid(fid,'Subtract','G_ant','Port1','Dira2');
CstCylinder(fid,'Dira2','G_ant','Vacuum','Z',4.3/2,0,x_port,y_port,-9.635,
-1.635,0);
CstSolid(fid,'Subtract','G_ant','Konektor','Dira2');
CstCylinder(fid,'Coax','P_ant','Copper','Z',1.2/2,0,x_port,y_port,
-9.635,tl_sub,0);
```

Funkce *CstDeleteComponent* slouží k odebrání nepotřebné komponenty.

```
CstDeleteComponent(fid,'P_port');
```

Funkce *CstBackground* slouží k nastavení prostředí ve kterém je simulovaný model, opět pomocí zadaných parametrů do funkce.

```
CstBackground(fid,'Normal',1.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,'Normal',0.0,
'True');
```

Funkce *CstSetFrekvencyRange* potřebná pro nastavení rozsahu ve kterém bude provedena analýza.

```
CstSetFrekvencyRange(fid,1,3);
```

Funkce *CstWaveguidePort* pro definování použitého napájecího portu.

```
CstWaveguidePort(fid,1,1,'False',0.0,0,45,'Free','zmin','True','False',...
    -(5.3/2)+x_port,(5.3/2)+x_port,-(5.3/2)+y_port,(5.3/2)+y_port,-9.635,
-9.635,0.0,0.0,0.0,0.0,0.0,0.0);
```

Funkce *CstBoundary* sloužící k nastavení okrajových podmínek.

```
CstBoundary(fid,'expanded open','expanded open','expanded open',
'expanded open','open','expanded open','none','none','none','isothermal',
'isothermal','isothermal','isothermal','isothermal','isothermal','False',
'','','','','','','None','None','None','None','None','None');
```

Funkce *CstMonitor* sloužící k nastavení zobrazení vyzařovacího diagramu pro konkrétní frekvenci.

```
CstMonitor(fid,'farfield (f=2)','Frequency','Farfield',2);
CstMonitor(fid,'farfield (f=1.6433)','Frequency','Farfield',1.6433);
```

Funkce *CstTimeDomainSolverEasy* pro nastavení druhu a parametrů vybrané analýzy.

```
CstTimeDomainSolverEasy(fid,'TD-S','All','All',-30,'False','False',50,
'False','False','False','False','False');
```

Funkce *CstStartSolver* sloužící k vlastnímu spuštění dříve nastavené analýzy.

```
CstStartSolver(fid);
```

Funkce *CstGetfrequencyFromSavedFile* sloužící k načtení požadovaných vypočtených parametrů.

```
VectorFaS=CstGetfrequencyFromSavedFile(CFileName,0.5,'S',[1 1])
SF = VectorFaS(1,1);
```

Funkce *CstSaveAsProject* sloužící k uložení vytvořeného projektu.

```
CstSaveAsProject(tmpAdrSoub);
```

Funkce *CstCloseProject* k zavření projektu.

```
CstCloseProject;
```

3.5 Využití Toolboxu při modelování v HFSS

Funkce *global* definuje globální proměnná, která je potřeba ve všech volaných funkcích.

```
global fid Module Design Desktop;
```

Funkce *addpath* slouží k načtení cest k adresářům, ve kterých jsou uloženy všechny funkce, které se využívají při sestavování simulačního modelu pomocí skriptu.

```
addpath('../3D modeler/');
addpath('../analysis/');
addpath('../general/');
addpath('../boundary/');
addpath('../other/');
```

V tomto bloku jsou nadefinovány požadované parametry simulovaného modelu. Nejsou zde uvedeny jednotky, protože se definují přímo v požadované funkci.

```
sir = 37.7;
vys = 28.4;
tl_sub = 1.6;
tl_zar = 1;
del_zar = 6;
x_port = 6;
y_port = 7.1;
```

Zde jsou nadefinovány absolutní cesty k souborům, které jsou potřeba ke správné funkčnosti Toolboxu. Soubor označen jako *tempScriptFileHfss* slouží k vytváření historie z posloupnosti napsané ve skriptu (napsané níže). Další soubor *hfssSkriptTemp*, který je dán absolutní cestou, je určen funkci *HfssSaveProject(fid,hfssSkriptTemp);*, která provede uložení vykresleného modelu do tohoto předem nadefinovaného souboru. Posledním souborem s nadefinovanou absolutní cestou (*hfssExportFile*) slouží k ukládání výsledků po provedení analýzy.

```
tempScriptFileHfss = 'c:/tmp_hfss/matlab/tmpDataHfss.m';
hfssSkriptTemp = ' c:/tmp_hfss/matlab/tmpDataHfss.hfss';
hfssExportFile = ' c:/tmp_hfss/matlab/tempDataExportHFSS.m';
```

Zde se provede vytvoření a otevření dočasného souboru (*tmpDataHfss.m*) na zvolené absolutní adrese. Do tohoto souboru je zapisována historie posloupnosti příkazů po sobě jdoucích v tomto skriptu. Přístup k tomuto souboru je pomocí proměnné (*fid*), která je nastavena jako globální proměnná, protože do tohoto souboru se zapisují posloupnosti volaných příkazů a přístup je potřeba v každé volané funkci.

```
fid = fopen(tempScriptFileHfss, 'wt');
```

V bloku, který je uveden níže je vidět posloupnost několika funkcí, využívající funkce Toolboxu k sestavení simulačního modelu v programu HFSS.

Funkce *HfssNewProject* slouží k vytvoření Active X serveru a dále k otevření a navázání spojení programu HFSS.

```
HfssNewProject(fid);
```

Funkce *HfssInsertDesign* je určena k vytvoření nového projektu se zvoleným pojmenováním.

```
HfssInsertDesign(fid,'Example_antena');
```

Funkce *HfssBox* definuje ze zadaných parametrů krychli.

```
HfssBox(fid,'Dielektrikum','FR4_epoxy',[-sir,-vys,0],
[sir+(2*sir),vys+(2*vys),tl_sub],'mm');
```

Funkce *HfssRectangle* pro vytvoření 2D plochy (čtverce). Tato funkce je ve skriptu použita vícekrát, proto ji zde už nebudu popisovat.

```
HfssRectangle(fid,'Antena','Z',[0,0,tl_sub],sir,vys,'mm');
```

Funkce *HfssSetColor* pro nastavení barvy vytvořeného objektu. Tato funkce je ve skriptu použita vícekrát, proto ji zde už nebudu popisovat.

```
HfssSetColor(fid,'Antena',[202 184 123]);
```

Funkce *HfssSetTransparency* kterou se nastavuje transparentnost (průhlednost) objektu. Tato funkce je ve skriptu použita vícekrát, proto ji zde už nebudu popisovat.

```
HfssSetTransparency(fid,'Antena',0);
HfssRectangle(fid,'Zarez1','Z',[0,((vys/2)-(tl_zar/2)),
tl_sub],del_zar,tl_zar,'mm');
```

Funkce *HfssSubtract* slouží k vytvoření zářezu na daném objektu. Tato funkce je ve skriptu použita vícekrát, proto ji zde už nebudu popisovat.

```
HfssSubtract(fid,'Antena','Zarez1');
HfssRectangle(fid,'Zarez2','Z',[((sir/2)-(tl_zar/2)),0,tl_sub],
tl_zar,del_zar,'mm');
HfssSubtract(fid,'Antena','Zarez2');
HfssRectangle(fid,'Zarez3','Z',[(sir-del_zar),((vys/2)-(tl_zar/2)),tl_sub],
del_zar,tl_zar,'mm');
HfssSubtract(fid,'Antena','Zarez3');
HfssRectangle(fid,'Zarez4','Z',[((sir/2)-(tl_zar/2)),(vys-del_zar),tl_sub],
tl_zar,del_zar,'mm');
HfssSubtract(fid,'Antena','Zarez4');
HfssRectangle(fid,'Ground','Z',[(-sir),(-vys),0],
sir+(2*sir),vys+(2*vys),'mm');
HfssSetColor(fid,'Ground',[202 184 123]);
HfssSetTransparency(fid,'Ground',0);
```

Funkce *HfssCircle* slouží k vytvoření 2D objektu (kruhu).

```
HfssCircle(fid,'G_Dira','Z',[((sir/2)+x_port),((vys/2)+y_port),0],
2.7,'mm');
HfssSubtract(fid,'Ground','G_Dira');
```

Funkce *HfssCylinder* ro definování 3D objektu (válce). Tato funkce je ve skriptu použita vícekrát, proto ji zde už nebudu popisovat.

```
HfssCylinder(fid,'Koax','Z',[((sir/2)+x_port),((vys/2)+y_port),0],2.7,
-8,'mm','vacuum');
HfssCylinder(fid,'Port1','Z',[((sir/2)+x_port),((vys/2)+y_port),0],0.5,
-8,'mm','vacuum');
```

Funkce *HfssAssignMaterial* která určuje typ použitého materiálu u zvoleného objektu. Tato funkce je ve skriptu použita vícekrát, proto ji zde už nebudu popisovat.

```
HfssAssignMaterial(fid,'Port1','pec');
HfssSetColor(fid,'Port1',[202 184 123]);
HfssCylinder(fid,'Pin','Z',[((sir/2)+x_port),((vys/2)+y_port),0],0.5,
tl_sub,'mm','vacuum');
HfssAssignMaterial(fid,'Pin','pec');
HfssSetColor(fid,'Pin',[202 184 123]);
HfssBox(fid,'airvol','air',[(-sir),
(-vys),0],[sir+(2*sir),vys+(2*vys),(20*tl_sub)],'mm');
HfssSetTransparency(fid,'airvol',0.9);
```

Funkce *HfssAssignRadiationFaces* slouží k nastavení okrajových podmínek pomocí vybrané plochy, která je určena číslem plochy. Tato funkce je ve skriptu použita vícekrát, proto ji zde už nebudu popisovat.

```
HfssAssignRadiationFaces(fid,'AirvolRad1',183);
HfssAssignRadiationFaces(fid,'AirvolRad2',185);
HfssAssignRadiationFaces(fid,'AirvolRad3',186);
HfssAssignRadiationFaces(fid,'AirvolRad4',187);
HfssAssignRadiationFaces(fid,'AirvolRad5',188);
```

Funkce *HfssAssignPE* slouží k nastavení také okrajových podmínek pomocí vybrané plochy, která je určena názvem plochy. Tato funkce je ve skriptu použita vícekrát, proto ji zde už nebudu popisovat.

```
HfssAssignPE(fid,'PeE1','Antena',false);
HfssAssignPE(fid,'PeE2','Ground',true);
HfssCircle(fid,'WPort','Z',[((sir/2)+x_port),((vys/2)+y_port),-8],
2.7,'mm');
HfssSetTransparency(fid,'WPort',1);
```

Funkce *HfssAssignWavePort* je určena k nadefinování konkrétního druhu napájecího portu.

```
HfssAssignWavePort(fid,'WavePort','WPort',[((sir/2)+x_port),
((vys/2)+y_port),-8],[((sir/2)+x_port+2.7),((vys/2)+y_port),-8],'mm');
```

Funkce *HfssInsertSolution* sloužící k nastavení parametrů analýzy.

```
HfssInsertSolution(fid,'Setup3GHz',3,0.02,25);
```

Funkce *HfssFastSweep* k nastavení druhu krokování v analýze.

```
HfssFastSweep(fid,'sweep1to3GHz','Setup3GHz',1,3,1000);
```

Funkce *HfssSolveSetup* sloužící ke spuštění dříve nastavené analýzy.

```
HfssSolveSetup(fid, 'Setup3GHz');
```

Funkce *HfssSaveProject* potřebná pro uložení vytvořeného projektu.

```
HfssSaveProject(fid,hfssSkriptTemp);
```

Funkce *HfssExportNetworkData* potřebná pro exportování výsledků analýzy do dočasného souboru.

```
HfssExportNetworkData(fid,hfssExportFile,'Setup3GHz','sweep1to3GHz','m',50)
```

Funkce *GetFrequencyFromDiagram* sloužící k načtení analyzovaných dat, které byly uloženy do dočasného souboru.

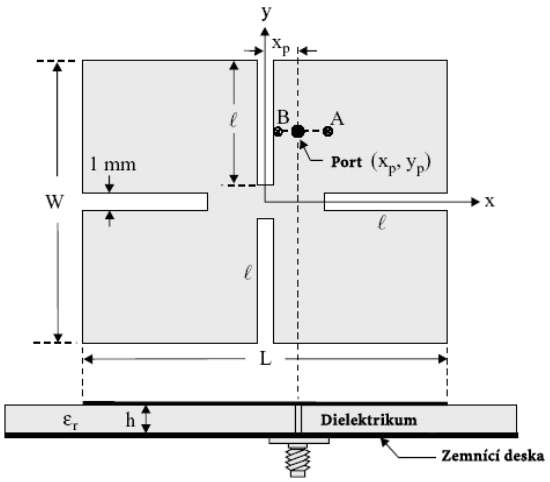
```
SF = GetFrequencyFromDiagram(0.5,hfssExportFile);
SF = SF(1,1);
```

Funkce *HfssCloseProject* sloužící k uzavření vytvořeného projektu.

```
HfssCloseProject;
```

3.6 Model planární antény se čtyřmi poruchovými zářezy

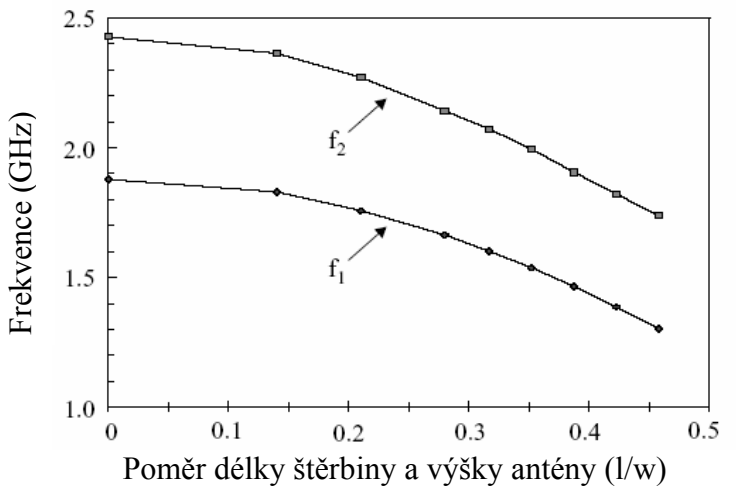
Planární anténa je na obr. 4. Čtyři poruchové zářezy jsou o stejné délce *l* a šířka štěrbiny je 1 mm [5].



Obr. 4 Model dvoupásmové mikropáskové antény se čtyřmi poruchovými zářezy, *W* – výška antény, *L* – šířka antény, *l* – délka zářezu, *h* – tloušťka substrátu, ε_r – permitivita substrátu [5].

Bod A značí pozici napájení antény pro dvou-frekvenční chod pro jednoduché pravoúhlé pole bez zářezů, který určuje optimální pozici napájecího portu pro vybuzení vidů TM_{01} a TM_{10}. V přítomnosti zářezů dochází k jevu, kdy optimální napájecí pozice prvních dvou rezonančních frekvencí může být dosažena přiblížením se horizontálně směrem k bodu B. Když délka zářezu vzroste, napájecí pozice se o to více přiblíží k bodu B. Čárkovaná osa X_p znázorňuje vzdálenost napájecího portu od středu antény a dá se vyjádřit hodnotami X_p a Y_p [5].

Se změnou velikosti štěrbin souvisí také hodnota rezonanční frekvence. Když se délka zářezů prodlouží, tak se první dvě rezonanční frekvence sníží. Obr. 5 a tab. 1 tuto skutečnost znázorňuje. Podobně je tomu v případě, když se štěrbiny zkříží nebo v páru ohnou [5].



Obr. 5 Závislost prvních dvou rezonančních frekvencí planární antény se čtyřmi poruchovými zářezy v poměru délky zářezu k výšce planární antény (*l*/*W*). Rezonanční frekvence se snižuje s rostoucí délkou zářezu *l* [5].

Parametry antény: permitivita dielektrika ε_r = 4,4; tloušťka dielektrika h = 1,6 mm; šířka antény L = 37,7 mm, výška antény W = 28,4 mm tloušťka zářezu l = 1 mm [5].

Tab. 1 Zadané délky štěrbin *l* a k nim odpovídající rezonanční frekvence a jejich poměry dvou prvních frekvencí (f_2/f_1). Vzdálenost X_p značí pozici umístění napájecího portu antény, jejichž velikost je brána od středu antény [5].

***l* [mm]**	**X_p [mm]**	**f_1,[MHz]**	**f_2,[MHz]**	**f_2/f_1[-]**
0	6,5	1878	2427	1.292
4	6,5	1830	2364	1.291
6	6,0	1758	2271	1.292
8	5,5	1664	2142	1.287
9	4,7	1601	2072	1.294
10	4,0	1538	1995	1.297
11	3,5	1466	1906	1.300
12	2,0	1386	1822	1.315

4. Optimalizace

Optimalizační algoritmy slouží k nalezení minima dané účelové funkce tak, že hledají optimální numerickou kombinaci jejich argumentů. Optimalizační algoritmy lze rozdělit podle principu jejich činnosti. Algoritmy se dělí do čtyř skupin a to na enumerativní, deterministické, stochastické a smíšené [6].

V této práci budou použity optimalizační algoritmy, které spadají do skupiny smíšené, proto zde bude uvedeno více o této skupině [6].

4.1 Smíšené optimalizační algoritmy

Tato třída algoritmů představuje směs metod deterministických a stochastických, které ve vzájemné spolupráci dosahují překvapivě dobrých výsledků. Poměrně silnou podmnožinou těchto algoritmů jsou evoluční algoritmy. Algoritmy smíšeného charakteru jsou [6].

- Robustní, což znamená, že nezávisle na počátečních podmínkách poměrně často naleznou kvalitní řešení, jenž je reprezentováno obvykle jedním či více globálními extrémy [6].
- Efektivní a výkonné. Tyto termíny znamenají, že jsou schopny nalézat kvalitní řešení během relativně malého počtu ohodnocení účelové funkce [6].
- Jsou odlišné od čistě stochastických metod (díky přítomnosti podmnožiny deterministických metod) [6].
- Mají minimální nebo žádné požadavky na předběžné informace [6].
- Jsou schopné pracovat s problémy typu „černá skříňka“, tzn. že nepotřebují ke své činnosti analytický popis problému [6].
- Jsou schopny nalézt více řešení jednoho problému [6].

Dalo by se tedy říci, že smíšená optimalizace je vhodná na problémy „bez omezení“ velikosti jejich prostoru možných řešení [6].

4.2 PSO (Particle swarm optimization)

Algoritmus je založen na práci s populací jedinců. Jejichž pozice v prostoru možných řešení je měněná pomocí tzv. rychlostního vektoru [6]. V následujícím textu jsou popsány používané termíny v optimalizaci PSO.

1) *Částečka* nebo *Agent:* Každý jednotlivec v roji (každého jednotlivce si lze představit jako včelu, která se pohybuje v poli) je platná částečka nebo agent. Všechny částečky v roji, rozmístěné jednotlivě se řídí stejným pravidlem, přibližování se k nejlepšímu osobní a nejlepšímu celkovému umístění, zatímco stále kontrolují hodnotu aktuálního umístění [7].

2) *Pozice:* Pozice prostoru včely reprezentující sadu vstupních parametrů pro jedno řešení optimalizovaného problému. Toto je reprezentované souřadnicemi na osách prohledávaného prostoru parametrů optimalizovaného problému. Velikost ani počet rozměrů prohledávaného prostoru není principiálně omezen [7].

3) *Fitness, (kriteriální funkce):* Je funkce , která ohodnotí vhodnost pozice – kvalitu řešení daného pozicí v prohledávaném prostoru. Funkce Fitness musí vzít pozici v prostoru řešení

a vrátit jedno číslo pozice reprezentující hodnotu. Analogií Fitness v příkladu se včelami je hodnota hustoty květin. Vyšší hustota květin se rovná zlepšení umístění. Může se například jednat o zisk antény, hmotnost, špičkovou křížovou polarizaci, nebo nějaký druh zatížené sumy všech těchto faktorů. Funkce Fitness poskytuje rozhraní mezi fyzickým problémem a optimalizačním algoritmem [7].

4) *pbest:* Každá včela si pamatuje umístění kde se setkala s větším počtem květin. Toto umístění s nejvyšší Fitness hodnotou, které objevila včela je známo jako osobní nejlepší místo označované jako *pbest*. Každá včela má vlastní *pbest, která je dána cestou včely*. Každý bod podél cesty včela srovnává Fitness hodnotu aktuálního umístění s hodnotou *pbest*. Jestliže aktuální umístění má vyšší Fitness hodnotu, pak je *pbest* nahrazen jeho aktuálním umístěním [7].

5) *gbest:* Každá včela zároveň zná umístění nejvyšší koncentrace květin, objevený celým rojem. Toto umístění nejvyšší koncentrace květin je známo jako globálně nejlepší a je označeno jako *gbest*. Pro celý roj je hodnota *gbest* ke kterému je každá včela přitahována. Každý bod podél jejich cesty a každá včela srovnává Fitness hodnotu jejich aktuálního umístění s *gbest*. Jestliže nějaká včela je v místě, kde je vyšší Fitness hodnota, pak *gbest* je nahrazena tou včelou, která má lepší hodnotu než je *gbest* [7].

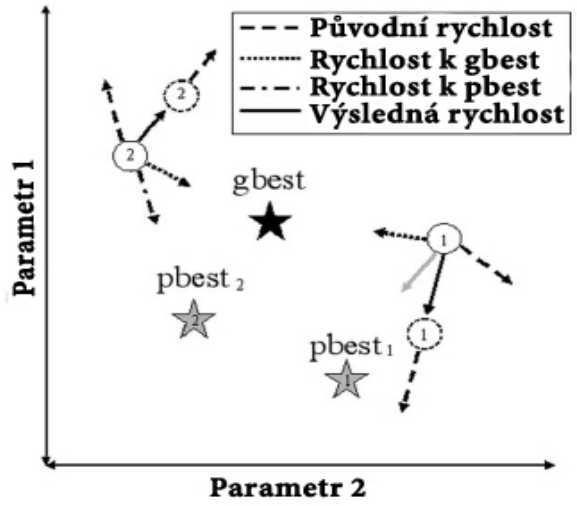
4.2.1 PSO algoritmus

1) *Definování řešeného prostoru:* První krok k realizaci PSO je výběr parametrů, které potřebují být optimalizovány a dát jim rozumný rozsah ve kterém se budou hledat optimální řešení. Toto vyžaduje specifikování minimální a nejvyšší hodnoty pro každý rozměr v N-rozměrné optimalizaci. Toto je nadefinováno jako Xmin_n a Xmax_n, kde n jsou rozsahy od 1 do N [7].

2) *Definování funkce Fitness:* Je to důležitý krok, který stanoví spojení mezi algoritmem optimalizace a fyzickým prostorem. Funkce Fitness by měla mít funkční závislost, která je dána optimalizační charakteristikou. Funkce Fitness a prostor řešení musí být specificky vyvinut pro každou optimalizace [7].

3) *Inicializování náhodného umístění a rychlosti roje:* Každá částečka začne ve vlastním náhodném umístění s náhodnou rychlostí hledat optimální pozici v řešeném prostoru. Její výchozí pozice je jediná lokace, s kterou se každá částečka během startu setkala. Tato pozice se stane pro každou částečku nejlepším nalezeným bodem *pbest*. První *gbest* je pak vybraný ze mezi těmito výchozími postaveními [7].

4) *Systematicky přelet částečky přes řešený prostor:* Každá částečka musí být přesunuta přes řešený prostor řešení jako by to byla včela v roji. Algoritmus jedná v závislosti na funkci *Fitness* podle každé částečky (včely) a podle potřeby s nimi pohybuje [7].



Obr.6 Jednotlivé částečky (1 a 2) zrychlují k umístění nejlepšího řešení *gbest*, a umístění jejich vlastního osobní nejlepší *pbest*, ve 2-D prostor [7].

a) *Ohodnocení částečky funkcí Fitness, a porovnání s gbest,pbest:* Funkce *Fitness*, používá osy částečky v prostoru řešení a vrací hodnotu funkce *Fitness*, která má být přidělena do aktuálního umístění. Jestliže je hodnota *Fitness* větší než hodnota *pbest* nebo globální *gbest,*pro danou částečku, pak jsou vhodná umístění nahrazená [7].

b) *Aktualizace rychlosti částečky:* Rychlost částečky je element celé optimalizace. Klíč k porozumění udává rovnice (1). Změna rychlosti částečky je měněna podle umístění příbuzného *pbest* a *gbest.* To je zrychlené ve směrech, kde umístění má největší hodnotu *Fitness* funkce podle následující rovnice:

$$v_n = \omega * v_n + c1.rand() * (P_{best,n} - X_n) + c2.rand() * (g_{best,n} - X_n) \quad (1)$$

kde v_n je rychlost částice v n-rozměrném prostoru a X_n je částečka pohybující se v n-rozměrném prostoru [7].

c) *Pohyb částečky:* Rychlost byla určena jednoduše přesunutím částečky na její následující pozici. Rychlost je použita při dalším kroku Δt, obvykle je vybrán jeden a nová osa X_n je počítána pro každý z N-rozměrů a je popsán následující rovnicí:

$$X_n = X_n + \Delta t * v_v \quad (2)$$

5) *Opakování:* Poté co tento proces je uskutečněn pro každou částečku v roji, je proces opakovaně spouštěn u kroku 4) [7].

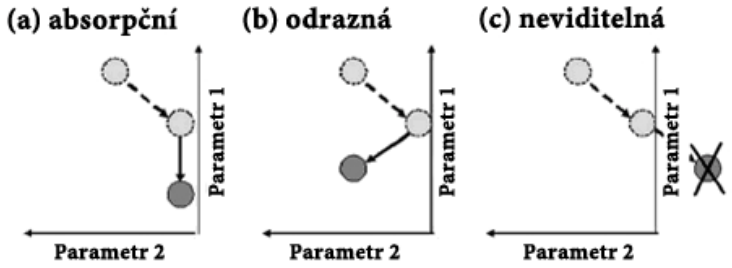
4.2.2 Typy hranic

Z literatury jsou známy následující typy hranic [7].

1) *Absorpční hranice:* Když částečka narazí do definované hranice v jednom z rozměrů, je její rychlost pohlcena a částečka zůstane umístěna na hranici [7].

2) *Odrážení od hranice:* Když částečka narazí do definované hranice v jednom z rozměrů, pak je rychlost stejná, ale částečka je vrácena zpět do prohledávaného prostoru [7].

3) *Neviditelné hranice:* Částečky mají dovoleno letět bez nějakého fyzického omezení. Nicméně, částečky, které se toulají venku v dovoleném prostoru nejsou ohodnoceny funkcí *Fitness* [7].



Obr. 7 Typy hranic [7]

4.3 SOMA (Samo-organizující se Migrační algoritmus)

Činnost algoritmu SOMA je založena na genetickém principu. Algoritmus SOMA se řadí mezi evoluční algoritmy, s tím rozdílem, že zde se nevytvářejí noví potomci. Tento algoritmus pracuje stejně jako ostatní evoluční algoritmy s populací jedinců, kde jedinci kooperují na řešení společného úkolu. Jedinci zde nejsou kříženi z rodičů, ale prohledávají prostor možných řešení. Algoritmus SOMA není klasifikován jako algoritmu evoluční, ale jako tzv. mimetický [6].

Vlastnost samo-organizace u SOMA algoritmu plyne z faktu, že se jedinci ovlivňují navzájem během lepšího řešení, což mnohokrát vede k tomu, že v prostoru možných řešení vznikají skupiny jedinců, které se rozpadají či spojují, putují přes prohledávaný prostor. Lze říci, že skupina jedinců (populace) sama organizuje vzájemný pohyb jedinců [6].

4.3.1 Parametry algoritmu SOMA

Tab. 2 Význam parametrů SOMA [6]

Parametr	Doporučený rozsah	Poznámka
Mass	<1.1,5>	Řídící parametr
Step	<0.11,Mass>	Řídící parametr
PRT	<0,1>	Řídící parametr
D	dáno problémem	Počet argumentů účelové funkce
NP	<10,definuje uživatel >	Řídící parametr
Migrace	<10,definuje uživatel >	Ukončovací parametr
AccteptedError	<libovolný,definuje uživatel >	Ukončovací parametr

Kvalita běhu algoritmu je závislá na zmíněných parametrech. Při nesprávném nastavení muže docházet k necitlivosti na nastavení parametrů [6].

Mass <1.1,3>. Tento parametr určuje jak daleko se aktivní jedinec zastaví od tzv. vedoucího jedince. Při Mass = 1 se aktivní jedinec zastaví na pozici vedoucího jedince, při Mass = 2 za ním ve stejné vzdálenosti z jaké startoval, atd. Jestliže je Mass < 1 pak dojde k tomu, že se zastaví před vedoucím jedincem, což povede k degradaci migračního procesu, tzn. Že budou nalezeny jen lokální extrémy. Z tohoto důvodu je doporučeno mít Mass > 1. Konečná hranice kdy Mass = 3 byla zjištěna jako dostačující prakticky ve všech simulacích na všech problémech [6].

Step <1.1,Mass>. Parametr Step určuje „zrnitost“, s jakou bude mapována cesta aktivního jedince. V případě jednoduché unimodální účelové funkce (konvexní, pár lokálních extrémů, atd.), je možné použít velkou hodnotu pro Step pro urychlení chodu algoritmu. Jestliže není známo ani přibližně jaká geometrie reprezentuje účelovou funkci, pak je doporučeno nastavit Step na nízkou hodnotu. Prostor možných řešení pak bude prohledán podrobněji, čímž se zvýší pravděpodobnost nalezení globálního extrému. Je rovněž důležité nastavit Step tak, aby vzdálenost mezi vedoucím a aktivním jedincem nebyla celočíselným násobkem parametru Step. Pokud by se tak stalo, diverzibilita populace by klesla, protože každý jedinec by mohl být finálně přitažen vedoucím jedincem a proces by tak mohl rychleji skončit v lokálním extrému. Proto Step o velikosti 0.11 je lepší než 0.1[6].

PRT <0,1>. PRT znamená perturbaci. Podle tohoto řídícího parametru se tvoří perturbační vektor (PRTVector), který ovlivňuje to zda se aktivní jedinec bude pohybovat přímo či ne. Je to jeden z nejdůležitějších parametrů s největší citlivostí. Jeho optimální hodnota je okolo 0.1. Pokud hodnota PRT narůstá, pak konvergence SOMA k lokálním extrémům silně vzrůstá. V případě nízkodimenzionální funkce a vysokého počtu jedinců je možné nastavit PRT na 0,7-1,0. Jestliže je PRT = 1 pak stochastická složka chování SOMA zaniká a algoritmus se chová jen pouze podle deterministických pravidel, což znamená, že v případě multimodální účelové funkce se hodí pouze k lokální optimalizaci[6].

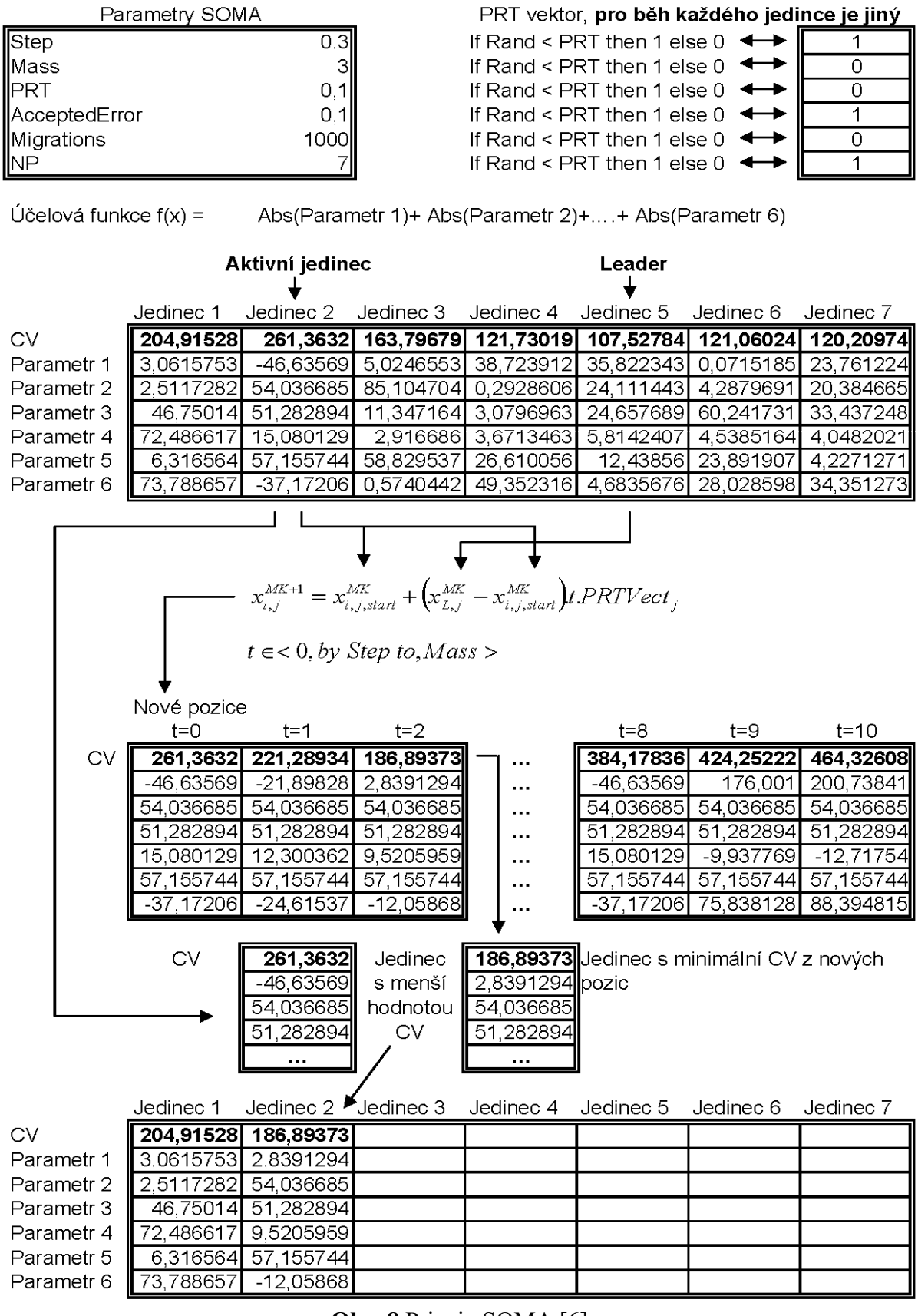
D je parametr, který udává počet optimalizovaných proměnných daného problému neboli počet argumentů účelové funkce. Je dán samotným problémem a může být změněn pouze tehdy, když je předefinován celý problém[6].

NP <10, dáno uživatelem>. Tento řídící parametr určuje kolik jedinců bude tvořit populaci. V podstatě může být nastaven už od 2, nutno však poznamenat, že v takovém případě by byl SOMA roven klasickým deterministickým metodám. Obecně může být nastaven v rozsahu 0.2 až 0,5 parametru D v případě, že tento je vysoký. Například když účelová funkce má 100 argumentů, pak se populace může skládat z cca 30-50 jedinců. V případě jednoduché funkce může být zvolen nízký počet jedinců. V opačném případě není problém nastavit např. NP = D. Obvykle by se nemělo klesnout pod NP = 10 z důvodu výkonnosti algoritmu[6].

Migrace <10,dáno uživatelem>. V podstatě udává, kolikrát se populace jedinců přeorganizuje. Je to ukončovací parametr[6].

AcceptedError <± libovolný,dáno uživatelem >. Je to ukončovací parametr. Definuje jaký je maximální rozdíl mezi nejhorším a nejlepším jedincem v aktuální populaci je povolen[6].

4.3.2 Princip algoritmu SOMA



Obr. 8 Princip SOMA [6]

4.4 Diferenciální evoluce

Diferenciální evoluce je poměrně nový typ evolučního algoritmu (od r.1995), který vyvinuli a poprvé použili Ken Price a Rainer Storm. Jeho schéma je dost podobné algoritmům genetickým, s nimiž má několik společných rysů, jako je například tvorba potomků (zde však pomocí 4 rodičů a ne 2 jak je tomu u genetických algoritmů), používání tzv. generací apod.

Funkce diferenciální evoluce spočívá v tom, že generuje tzv. zkušební řešení přičtením diference dvou náhodně vybraných vektorů (jedinců) ke třetímu jedinci z populace. Poté byla zkombinována diferenciální mutace a metody selekce z genetického žíhání [6].

4.4.1 Parametry a terminologie

Činnost a kvalita diferenciální evoluce je ovlivněna jejími řídícími parametry stejně jako u ostatních evolučních algoritmů[6].

Tab. 3 Hodnoty řídících parametrů diferenciální evoluce

Řídící parametr	Interval	Optimum	Poznámka
NP	<2D,100D>	10D	Velikost populace, 100D jestli je funkce vysoce multimodální
F	<0,2>	.3-.9	Mutační konstanta
CR	<0,1>	.8-.9	Práh křížení, CR <<1 jestli je funkce neseparabilní (epistatická)
Generations	Uživatel		Počet kol šlechtění populace.

CR <0,1>. Jde o tzv. Práh křížení. V případě že se jedná o funkci, která je separabilní, pak je doporučeno nastavit tento parametr na hodnoty blízké 0. V opačném případě jsou výhodné hodnoty, které se blíží 1. V případě, že se CR nastaví na 0 dojde k tomu, že se mutace nedostane do zkušebního jedince, který díky tomu bude kopií aktuálního (čtvrtého rodiče). Vývoj evoluce se pak zastaví. V případě že bude CR nastavena na 1, bude zkušební jedinec tvořen pouze ze tří náhodně vybraných rodičů – jedinců z populace a diferenciální evoluce se bude spíše podobat náhodnému hledání nežli evolučnímu algoritmu (všichni tři jedinci, byť evolučně vytvořené, jsou náhodně vybráni bez ohledu na jejich kvalitu). Je proto vhodné, aby CR nikdy nenabývalo těchto hodnot[6].

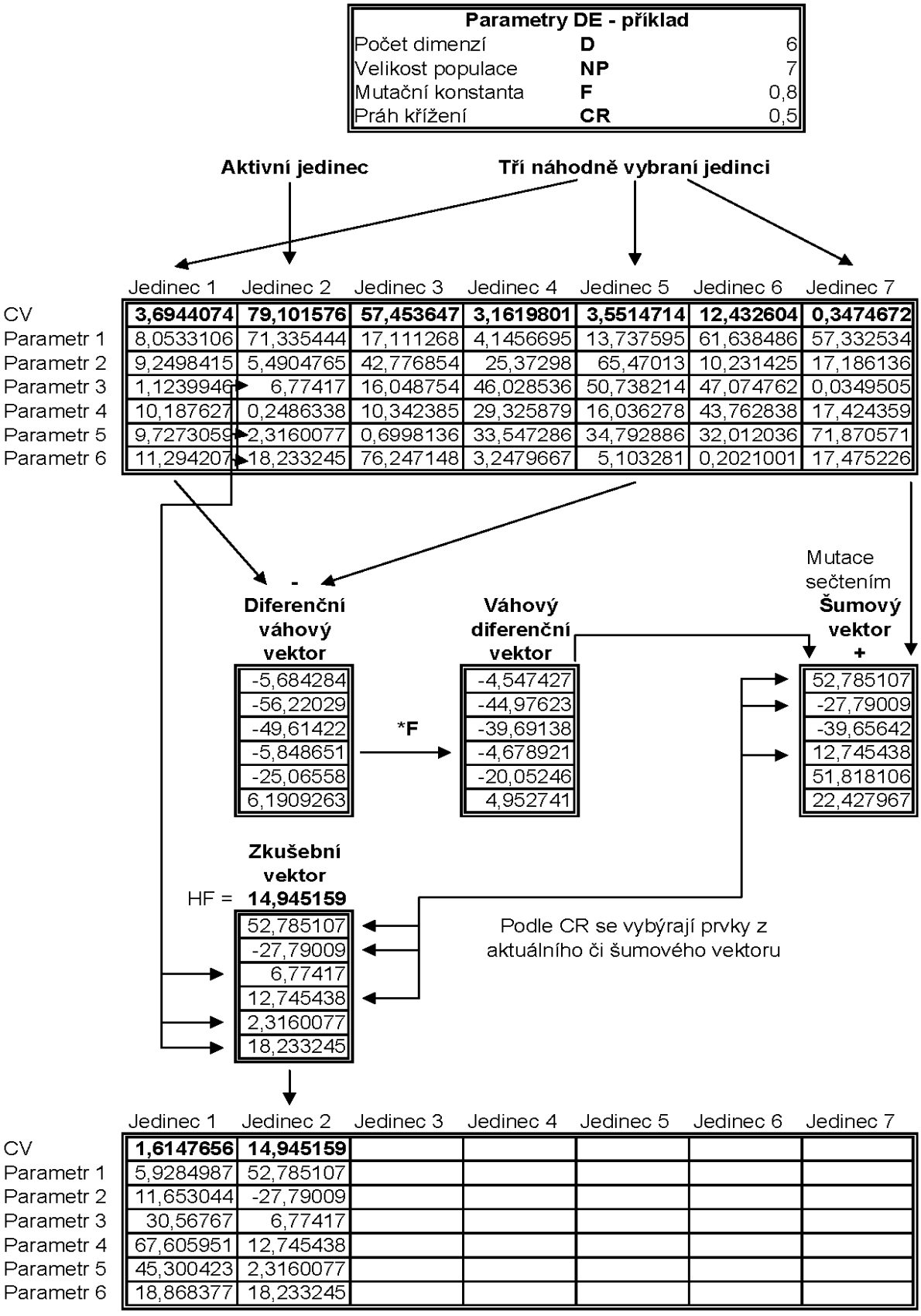
NP <2D,100D>. Jedná se o parametr udávající velikost populace. Hodnoty tohoto parametru jsou získány pouze zkušeností s tímto algoritmem. Jediné, co se dá s určitostí říci je, že NP by neměl být menší než 4. To je minimální velikost populace, při které diferenciální evoluce ještě pracuje[6].

F <0,2>. Mutační konstanta je poslední řídící parametr diferenciální evoluce a její hodnota se pohybuje v intervalu <0,2>[6].

Generations > 0. Udává počet evolučních cyklů, tzv. generací, během nichž se celá populace vyvíjí[6].

D – dimenze problému. Jde o počet argumentů účelové funkce. Parametr „D“ je tedy dán řešeným problémem a lze ho změnit pouze reformulací problému[6].

4.4.2 Princip diferenciální evoluce



Obr. 9 Princip diferenciální evoluce [6].

4.5 Kriteriální funkce

Kriteriální funkce slouží k ohodnocení nalezeného výsledku pomocí optimalizace. Funkcí sloužících k ohodnocení je velké množství. Zde uvedu jednu kriteriální funkci, která je implementována do všech tří popsaných optimalizačních algoritmů. Je to funkce, která ohodnocuje pouze jeden hledaný parametr. Hledaným parametrem je v tomto případě rezonanční frekvence.

$$F = \sqrt{(f_b - f_{hl})^2 + (f_b - f_{hl})^2} \qquad (3)$$

kde f_b je hodnota rezonanční frekvence z optimalizovaného objektu a f_{hl} je požadovaná hodnota. Ve vzorci (3) je první a druhá část totožná a to proto kdyby bylo potřeba hledat současně více požadovaných parametrů (například více požadovaných rezonančních frekvencí po daném optimalizovaném modelu).

4.6 Ověření funkčnosti propojovací vrstvy

Ověření funkčnosti propojovací vrstvy patří ke kapitole Toolbox. Je potřeba ověřit, zda propojovací vrstva je schopna nalézt požadované řešení. K tomuto ověření byla použita globální optimalizace. Ověřovací globální optimalizace byla PSO. Optimalizace byla provedena na popsaném modelu planární antény, viz obr. 4. Na modelu planární antény byly optimalizované parametry (šířka a výška antény a také délka a tloušťka zářezu) a pevně nastavené parametry, jenž byly tloušťka dielektrika a poloha napájecího portu, jak ve vodorovném, tak i ve svislém směru. Ověření je provedeno pro oba simulační programy, kde výsledkem je porovnání od skutečné hodnoty převzaté z literatury [5].

Parametry optimalizace byly nastaveny:

- počet generací je 5
- počet jedinců je 30

Vstupní parametry antény:

- šířka antény je 37,7 mm
- výška antény je 28,4 mm
- délka zářezu je 9 mm
- tloušťka zářezu je 1 mm
- tloušťka dielektrika je 1,6 mm
- poloha napájecího portu ve směru X je 4,7 mm
- poloha napájecího portu ve směru Y je 7,1 mm
- permitivita dielektrika je 4,4

Požadovaný kriteriální parametr:

- rezonanční frekvence je 1,6 GHz

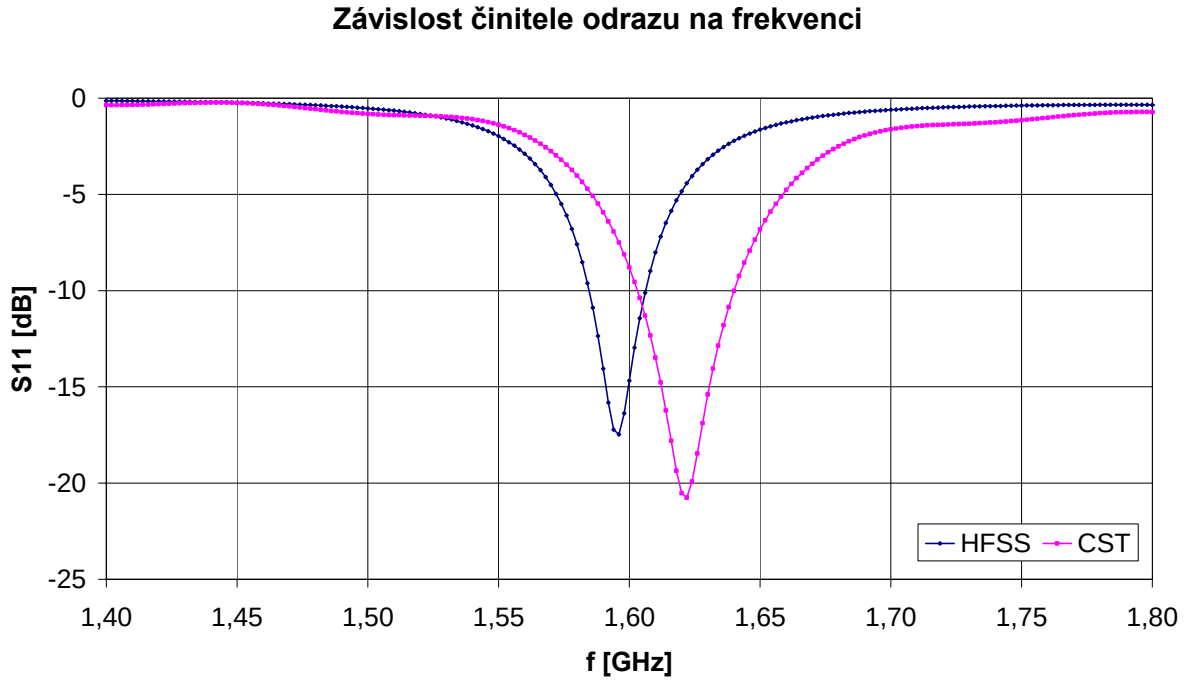
Optimalizace využívala kriteriální funkce nastavenou na první rezonanční frekvenci v počítaném pásmu od 1 GHz do 3 GHz. V Případě, že bylo vypočítáno více rezonančních frekvencí v počítaném pásmu, tak byla vždy vzata jen první rezonanční frekvence a ta byla

vrácena optimalizačnímu skriptu, jako kriterium. Optimalizační skript vrátil hodnoty, jež byly nejblíže požadovanému řešení. Vrácené parametry jsou uvedeny v tab. 4 pro oba externí programy.

Tab. 4 Přehled výsledků z optimalizace mezi programy CST a HFSS.

	Šířka antény [mm]	**Výška antény [mm]**	**Šířka zářezu [mm]**	**Délka zářezu [mm]**	**Frekvence [GHz]**
CST	37,7123	28,413	1	9	1,622
HFSS	38,1472	28,5302	0,5	9	1,596

Na následujícím obr. 10 je zobrazena závislost činitele odrazu S_{11} na frekvenci. Obrázek popisuje výsledek optimalizace v grafické podobě.



Obr. 10 Závislost činitele odrazu na frekvenci pro program CST a HFSS.

5. Testovací funkce

Důležitým prvkem testování optimalizačních algoritmů jsou testovací funkce [8]. Funkce pro testování optimalizačních algoritmů mohou být dvojího typu. Mezi první patří reálné funkce, které jsou následně ověřeny měřením nebo simulací. Mezi druhé testovací funkce patří uměle vytvořené funkce od relativně jednoduchých až po složitější, kde dané vytvořené funkce mohou mít více globálních minim i maxim. Funkce mohou mít mnoho nelinearit nebo různé roviny kolem hledaného globálního minima. Tyto funkce se používají k testování optimalizací, kde se zjišťují různé parametry testované optimalizace. Mezi parametry testované optimalizace patří zda optimalizace jsou schopni v různorodém prostoru najít globální minimum, neuváznou pouze v lokálním minimum a dále také rychlost nalezení globálního minima. Uměle vytvořené testovací funkce však nejsou směrodatným předpokladem toho, že testovaná optimalizace bude schopna vyhledat globální minimum stejně dobře jak u uměle vytvořené funkce, tak i u reálného problému. V reálném problému mohou být některé parametry velmi citlivé v některých oblastech a naopak někde nejsou citlivé vůbec[6][8].

Dále jsou zde uvedeny všechny použité funkce, kterých se v této práci využívá k testování. Funkce zde budou řazeny podle složitosti a budou vždy vyobrazeny. Průběhy byly zobrazovány v programu MATLAB a umístěny zde. U každé funkce bude uvedeno globální minimum, matematický popis a použitý interval funkce.

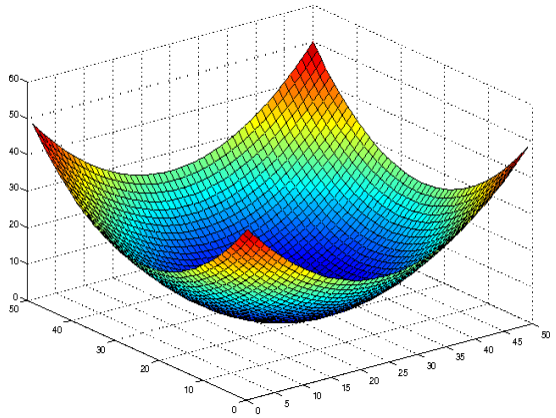
De Jong's function

Matematicky popis: $f(x) = \sum_{i=1}^{n} x^2$

Globální minimum: $f(x, y) = 0$

Umístění globálního minima: $x = 0, y = 0$

Interval použití: $-5.12 \le x_i \le 5.12, i = 1..n$

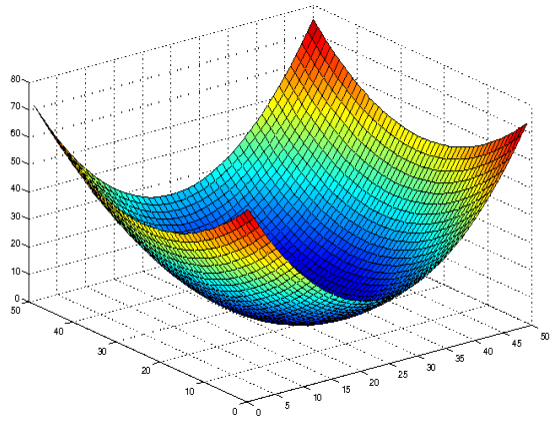


Obr. 11 De Jong's function [8] ve 2D $f(x, y) = x^2 + y^2$

Axis parallel hyper-elipsoid function

Matematicky popis: $f(x) = \sum_{i=1}^{n} (i.x_i^2)$

Globální minimum: $f(x, y) = 0$
Umístění globálního minima: $x = 0, y = 0$
Interval použití: $-5.12 \leq x_i \leq 5.12, i = 1..n$

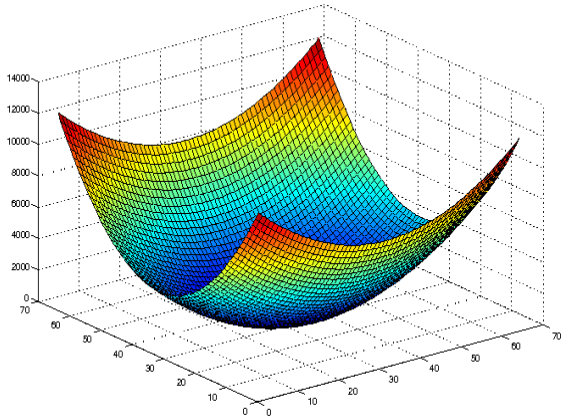


Obr. 12 Axis parallel hyper-elipsoid function [8] ve 2D $f(x, y) = x^2 + 2.y^2$

Rotated hyper-elipsoid function

Matematicky popis: $f(x) = \sum_{i=1}^{n} \sum_{j=1}^{i} x_j^2$

Globální minimum: $f(x, y) = 0$
Umístění globálního minima: $x = 0, y = 0$
Interval použití: $-65.536 \leq x_i \leq 65.536, i = 1..n$

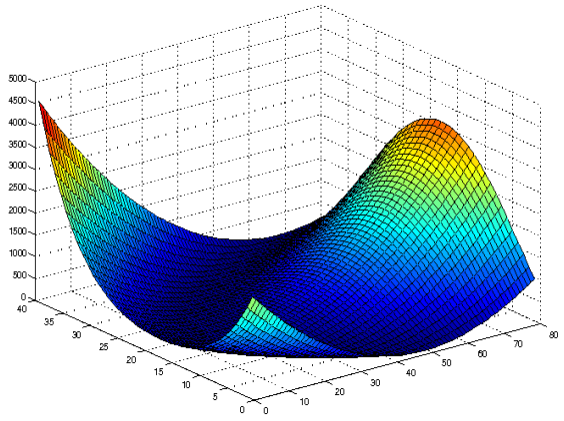


Obr. 13 Rotated hyper-elipsoid function [8] ve 2D $f(x, y) = x^2 + (x^2 + y^2)$

Rosenbrock's Valley function

Matematicky popis: $f(x) = \sum_{i=1}^{n-1} \left[100.\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)^2 \right]$

Globální minimum: $f(x, y) = 0$
Umístění globálního minima: $x = 1, y = 1$
Interval použití: $-2.048 \le x_i \le 2.048, i = 1..n$

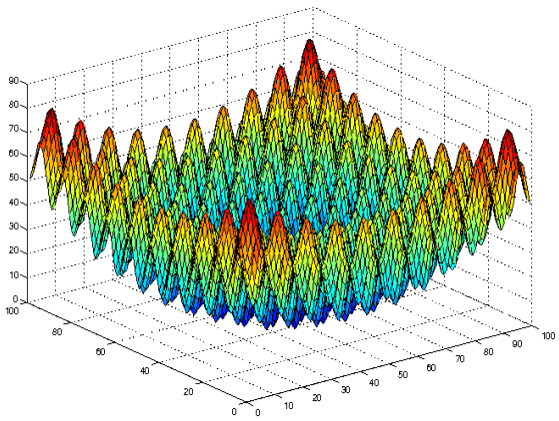


Obr. 14 Rosenbrock's Valley function [8] ve 2D $f(x, y) = 100.\left(y - x^2\right)^2 + \left(1 - x\right)^2$

Rastrigin's function

Matematicky popis: $f(x) = 10n + \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi\pi_i) \right]$

Globální minimum: $f(x, y) = 0$
Umístění globálního minima: $x = 0, y = 0$
Interval použití: $-5.12 \le x_i \le 5.12, i = 1..n$

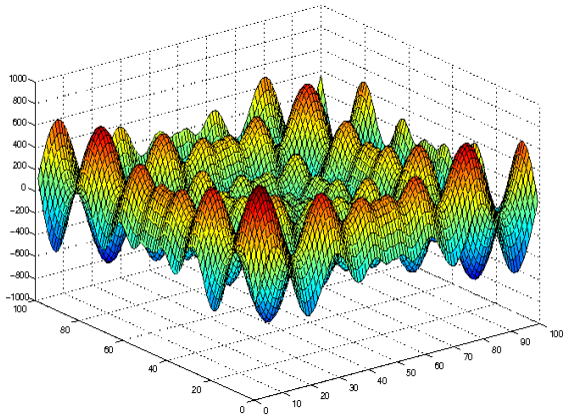


Obr. 15 Rastrigin's function [8] ve 2D $f(x, y) = 10.2 + \left[x^2 - 10\cos(2\pi\pi) \right] + \left[y^2 - 10\cos(2\pi\pi) \right]$

Schwefel's function

Matematicky popis: $f(x) = \sum_{i=1}^{n} \left[- x_i \sin\left(\sqrt{|x_i|}\right) \right]$

Globální minimum: $f(x,y) = -418{,}9829.n$
Umístění globálního minima: $x = 0, y = 0$
Interval použití: $-500 \le x_i \le 500, i = 1..n$

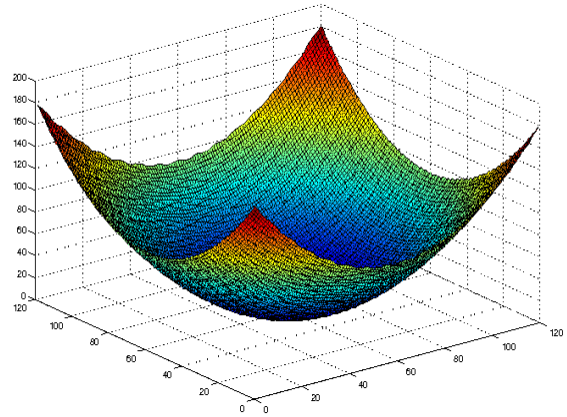


Obr. 16 Schwefel's function [8] ve 2D $f(x,y) = -x.\sin\left(\sqrt{|x|}\right) - y.\sin\left(\sqrt{|y|}\right)$

Griewangk's function

Matematicky popis: $f(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$

Globální minimum: $f(x,y) = 0$
Umístění globálního minima: $x = 0, y = 0$
Interval použití: $-600 \le x_i \le 600, i = 1..n$

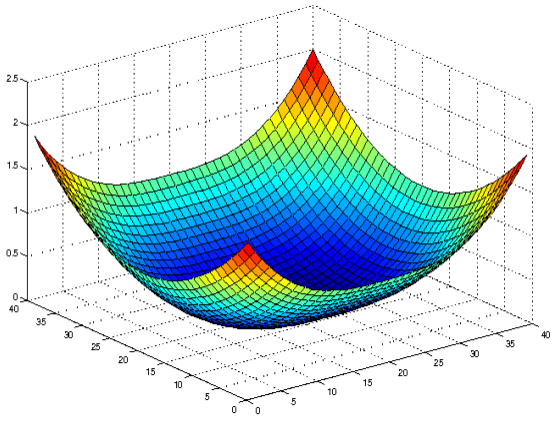


Obr. 17 Griewangk's function [8] ve 2D $f(x,y) = \frac{x^2 + y^2}{4000} - \cos(x)\cos\left(\frac{y}{\sqrt{2}}\right) + 1$

Sum of different power function

Matematicky popis: $f(x) = \sum_{i=1}^{n} |x_i|^{i+1}$

Globální minimum: $f(x, y) = 0$
Umístění globálního minima: $x = 0, y = 0$
Interval použití: $-1 \le x_i \le 1, i = 1..n$

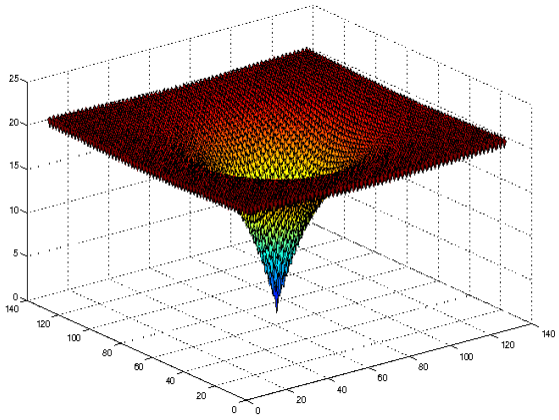


Obr. 18 Sum of different power function [8] ve 2D $f(x, y) = |x|^2 + |y|^3$

Ackley function

Matematicky popis: $f(x) = -a.\exp\left(-b.\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(c.x_i)\right) + a + \exp(1)$

Globální minimum: $f(x, y) = 0$
Umístění globálního minima: $x = 0, y = 0$
Interval použití: $-32.768 \le x_i \le 32.768, i = 1..n$
Parametry funkce: a = 20, b = 0,2, c = 2π



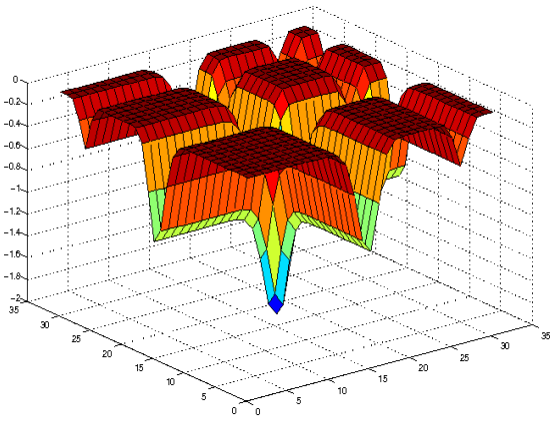
Obr. 19 Ackley function [8] ve 2D

$$f(x,y) = -a.\exp\left(-b.\sqrt{\frac{1}{n}.x^2.y^2}\right) - \exp\left(\frac{1}{n}.\cos(c.x).\cos(c.y)\right) + a + \exp(1)$$

Michalewicz function

Matematicky popis: $f(x) = -\sum_{i=1}^{n} \sin(x_i)\left[\sin\left(\frac{i.x_i^2}{\pi}\right)\right]^{2m}$

Globální minimum: $f(x,y) = 0$
Umístění globálního minima: $x = 0, y = 0$
Interval použití: $0 \le x_i \le \pi, i = 1..n$
Parametry funkce: m = 10, n = 2



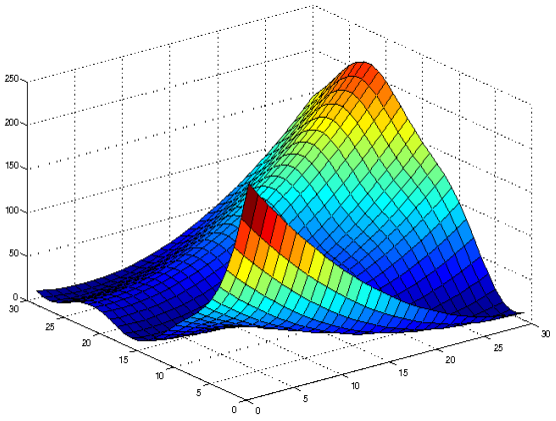
Obr. 20 Michalewicz function [8] ve 2D $f(x,y) = -\sin(x).\left(\sin\left(\frac{i.x^2}{\pi}\right)\right)^{2m} - \sin(y).\left(\sin\left(\frac{i.y^2}{\pi}\right)\right)^{2m}$

Branins function

Matematicky popis: $f(x) = a.\left(x_2 - b.x_1^2 + c.x_1 - d\right)^2 + e.(1-f).\cos(x_1) + e$

Tří globální minima: $f(x,y)=0{,}397887$
Umístění globálních minim: $(x,y)=(-\pi,12.275),(\pi,2.275),(9.42478,2.475)$
Interval použití: $0 \le x_i \le \pi, i=1..n$

Parametry funkce: a = 1, $b=\frac{5.1}{4\pi^2}$, $c=\frac{5}{\pi}$, d = 6, e = 10, $f=\frac{1}{8.\pi}$

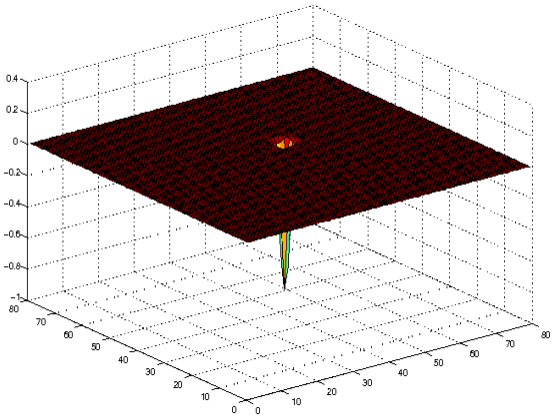


Obr. 21 Branins function [8] ve 2D $f(x,y)=a.\left(y-b.x^2+c.x-d\right)^2+e.(1-f).\cos(x)+e$

Easom function

Matematicky popis: $f(x_1,x_2)=-\cos(x_1)\cos(x_2)\exp\left(-(x_1-\pi)^2-(x_2-\pi)^2\right)$

Globální minimum: $f(x,y)=-1$
Umístění globálního minima: $(x,y)=(\pi,\pi)$
Interval použití: $-100 \le x_i \le 100, i=1..n$

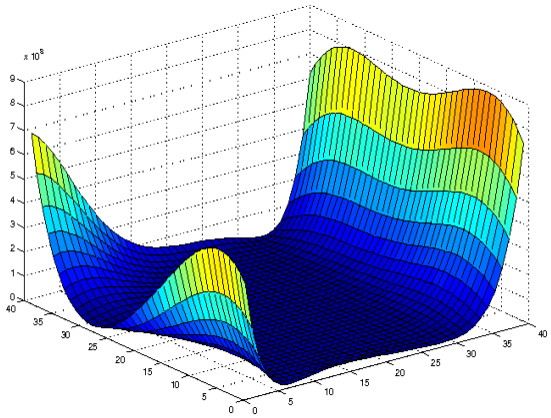


Obr. 22 Easom function [8] ve 2D $f(x,y)=-\cos(x)\cos(y)\exp\left(-(x-\pi)^2-(y-\pi)^2\right)$

Goldstein-Price function

Matematicky popis: $f(x_1,x_2)=\left[1+(x_1+x_2+1)^2\left(19+14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2\right)\right]$
$\left[30+(2x_1-3x_2)^2\left(18-32x_1+12x_1^2-48x_2+36x_1x_2+27x_2^2\right)\right]$

Globální minimum: $f(x,y)=3$
Umístění globálního minima: $(x,y)=(0,-1)$
Interval použití: $-2\le x_i\le 2, i=1..n$



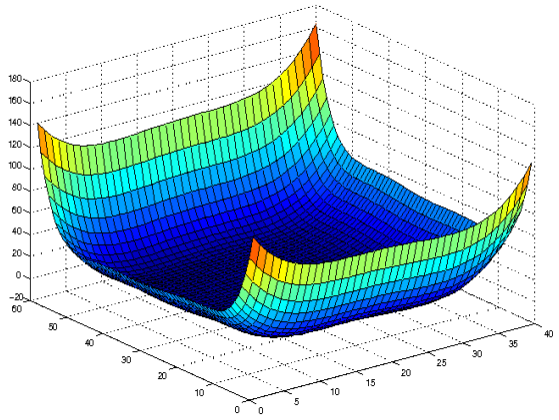
Obr. 23 Goldstein-Price function [8] ve 2D

$$f(x,y)=\left[1+(x+y+1)^2\left(19+14.x+3.x^2-14.y+6.x.y+3.y^2\right)\right]$$
$$\left[30+(2.x-3.y)^2\left(18-32.x+12.x^2-48.y+36.x.y+27.y^2\right)\right]$$

Six-hump camel back function

Matematicky popis: $f(x_1,x_2)=\left(4-2.1x_1^2+\frac{x_1^4}{3}\right).x_1^2+x_1.x_2+\left(-4+4.x_2^2\right).x_2^2$

Dvě globální minima: $f(x,y)=-1.0316$
Umístění globálních minim: $(x,y)=(-0.0898,0.7126),(0.0898,-0.7126)$
Interval použití: $-3\le x_1\le 3,-2\le x_2\le 2, i=1..n$



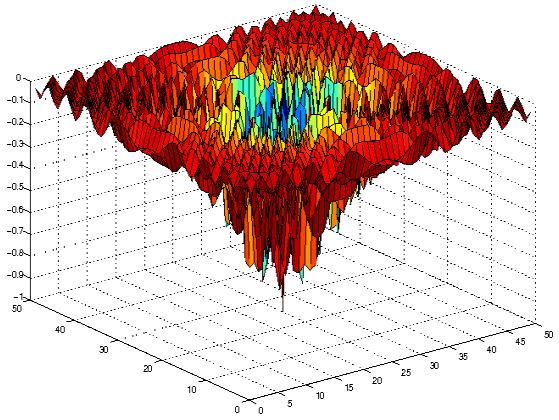
Obr. 24 Six-hump camel back function [8] ve 2D

$$f(x,y)=\left(4-2.1.x^2+\frac{x^4}{3}\right).x^2+x.y+\left(-4+4.y^2\right).y^2$$

Drop wave function

Matematicky popis: $f(x_1,x_2)=-\dfrac{1+\cos\left(12\sqrt{x_1^2+x_2^2}\right)}{\frac{1}{2}\left(x_1^2+x_2^2\right)+2}$

Globální minimum: $f(x,y)=-1$
Umístění globálních minim: $(x,y)=(0,0)$
Interval použití: $-5.12\le x_1\le 5.12,-5.12\le x_2\le 5.12, i=1..n$

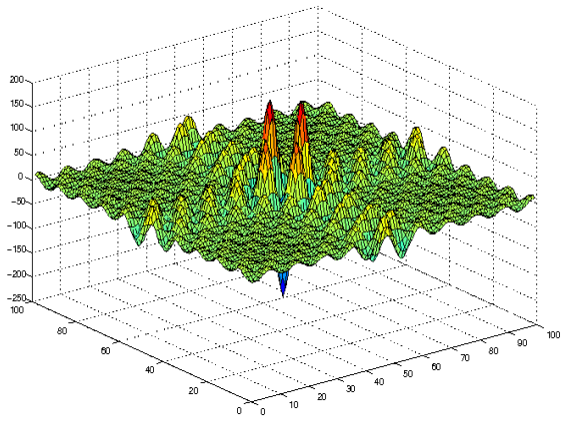


Obr.25 Drop wave function [8] ve 2D $f(x,y)=-\dfrac{1+\cos\left(12\sqrt{x^2+y^2}\right)}{\frac{1}{2}\left(x^2+y^2\right)+2}$

Shubert function

Matematicky popis: $f(x_1, x_2) = -\sum_{i=1}^{5} i.\cos((i+1).x_1 + 1).\sum_{i=1}^{5} i.\cos((i+1).x_2 + 1)$

Globální minimum: $f(x, y) = -1$
Umístění globálních minim: $(x, y) = (0, 0)$
Interval použití: $-5.12 \le x_1 \le 5.12, -5.12 \le x_2 \le 5.12, i = 1..n$



Obr. 26 Shubert function [8] ve 2D

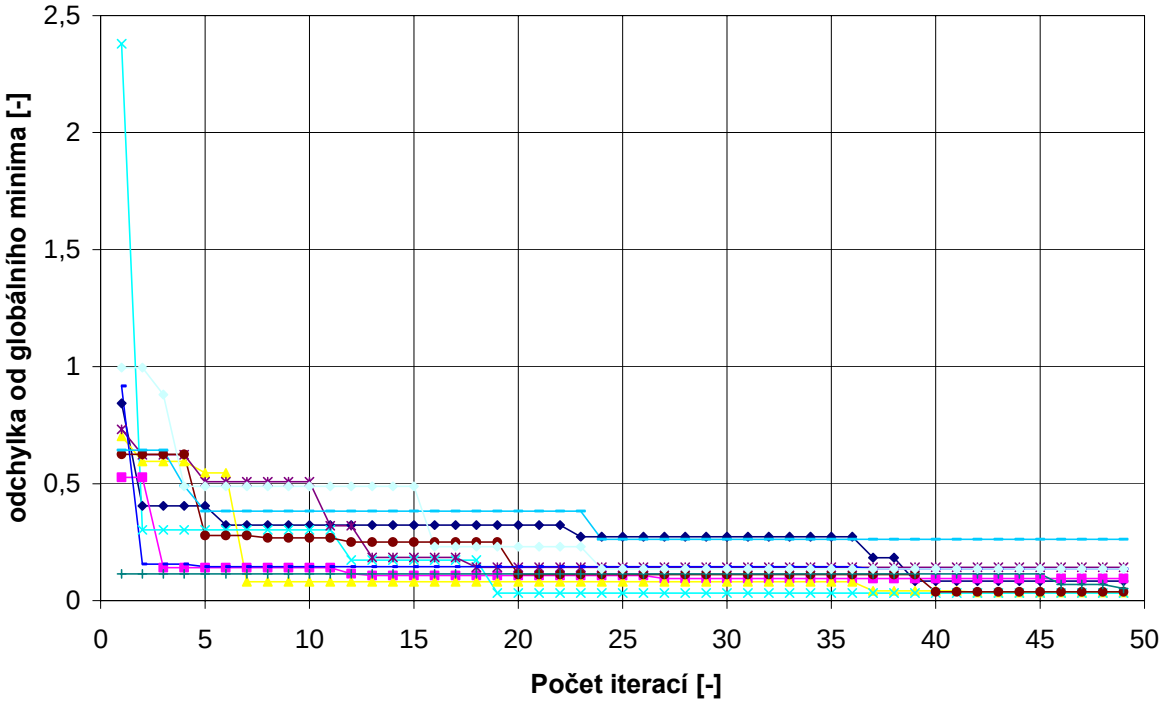
$$f(x, y) = -(\left(1.\cos((1+1).x + 1)\right) + \left(2.\cos((2+1).x + 1)\right) + \left(3.\cos((3+1).x + 1)\right) +$$
$$+ \left(4.\cos((4+1).x + 1)\right) + \left(5.\cos((5+1).x + 1)\right)) * (\left(1.\cos((1+1).y + 1)\right) + \left(2.\cos((2+1).y + 1)\right) +$$
$$\left(3.\cos((3+1).y + 1)\right) + \left(4.\cos((4+1).x + 1)\right) + \left(5.\cos((5+1).y + 1)\right))$$

6. Porovnání optimalizačních algoritmů

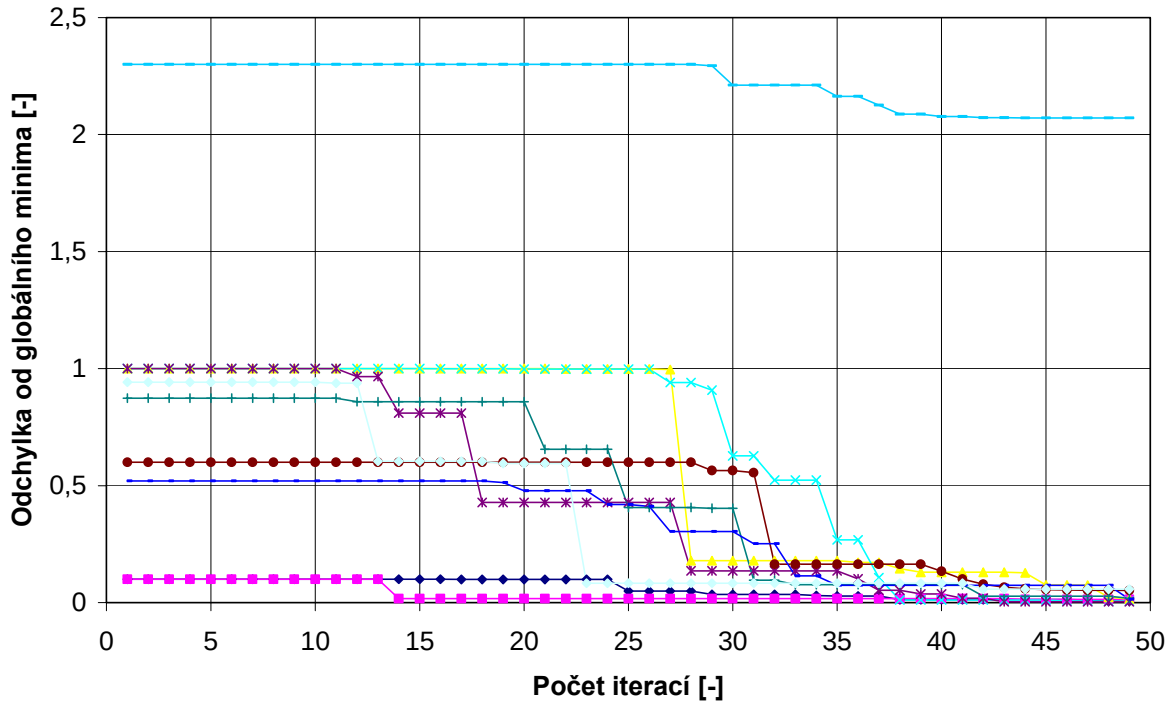
Kapitola porovnání optimalizačních algoritmu je zde uvedena z důvodu ohodnocení globálních optimalizačních algoritmů. Ohodnocení algoritmů je zde uvedeno na statistických charakteristikách, které jsou určeny pro porovnání stochastických optimalizačních algoritmů. Porovnání optimalizačních algoritmů je rozděleno na dvě části. První částí je porovnání optimalizačních algoritmů na popsaných testovacích funkcích [8] a druhou částí porovnání je porovnání optimalizačních algoritmů na reálném modelu [5], viz obr. 4.

6.1 Testovací funkce k určení výkonnosti optimalizačních algoritmů

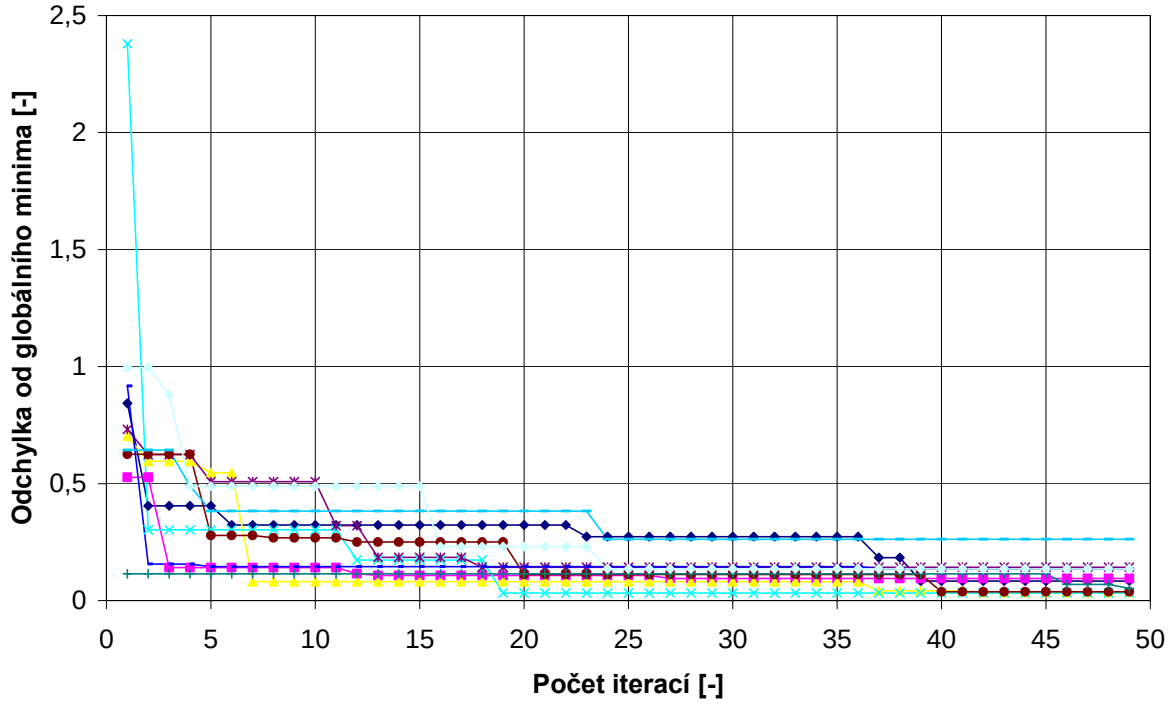
Pro určení výkonnosti optimalizačních algoritmů se používají testovací funkce [8]. Tyto funkce ukazují jakým způsobem je optimalizace schopna či neschopna nalézt globální minimum a také v kolika krocích. Pro statické charakteristiky jsou vybrány tři testovací funkce, seřazené podle obtížnosti. Vybrané testovací funkce jsou De Jong, Easom a Shubert. Každá z těchto vybraných testovacích funkcí je optimalizována popsanými optimalizacemi. Aby bylo možné vytvořit statické charakteristiky, je vždy na jedné vybrané testovací funkci spuštěna vybraná optimalizace alespoň 10 krát a výsledky jsou vyneseny do grafu.



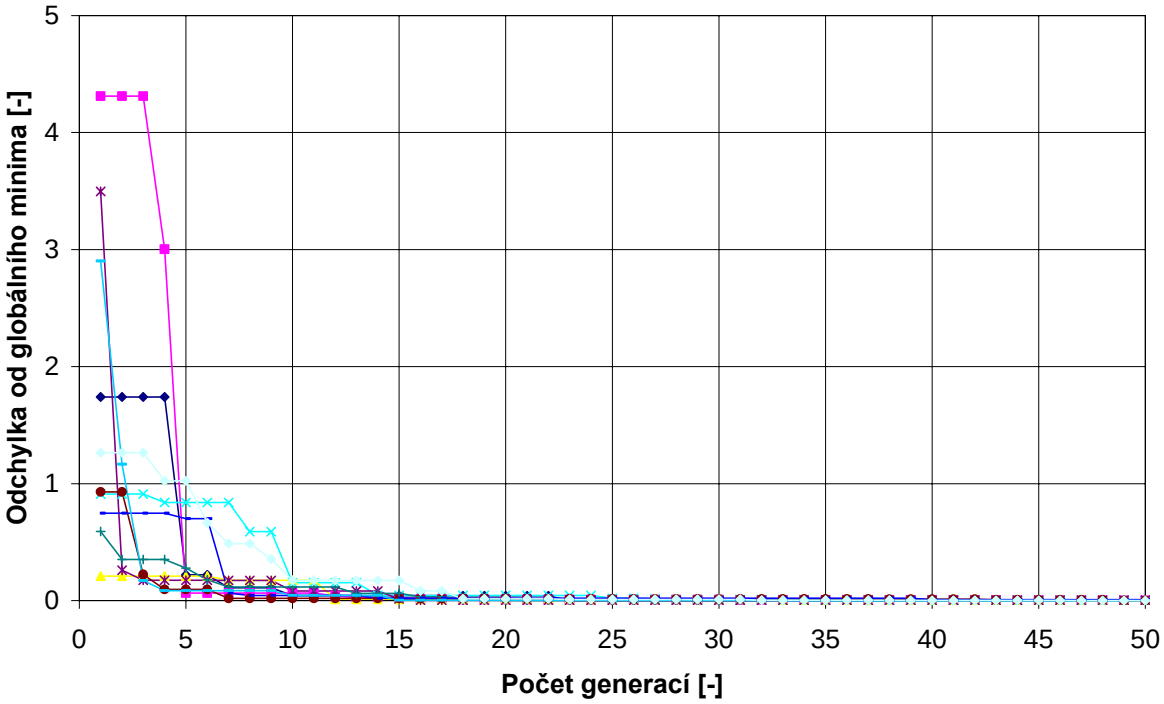
Obr. 27 Statistická charakteristika při použití optimalizace PSO na testovací funkci De Jong pro 10 opakování.



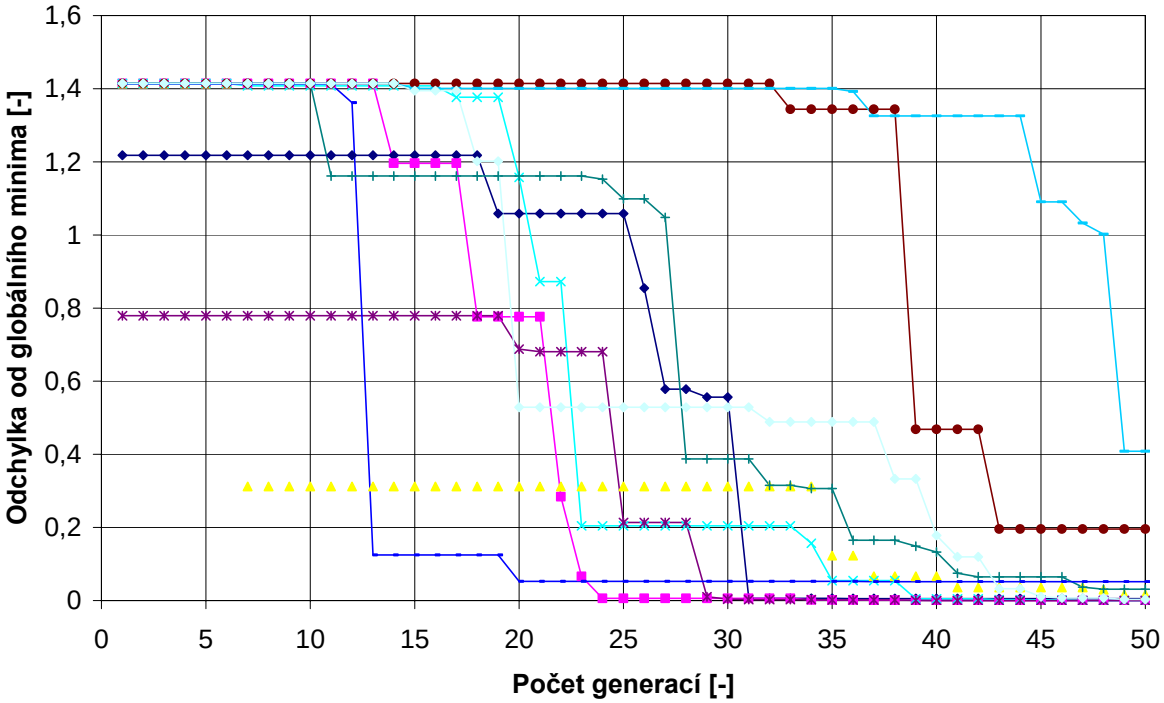
Obr. 28 Statistická charakteristika při použití optimalizace PSO na testovací funkci Easom pro 10 opakování.



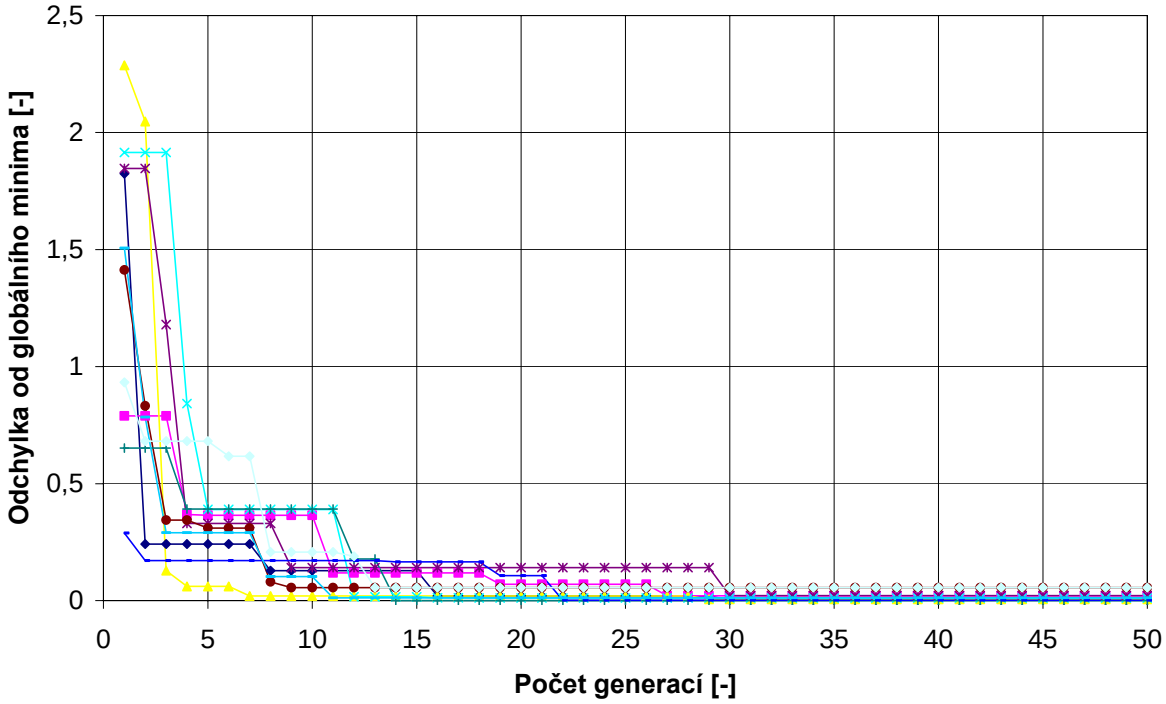
Obr. 29 Statistická charakteristika při použití optimalizace PSO na testovací funkci Shubert pro 10 opakování.



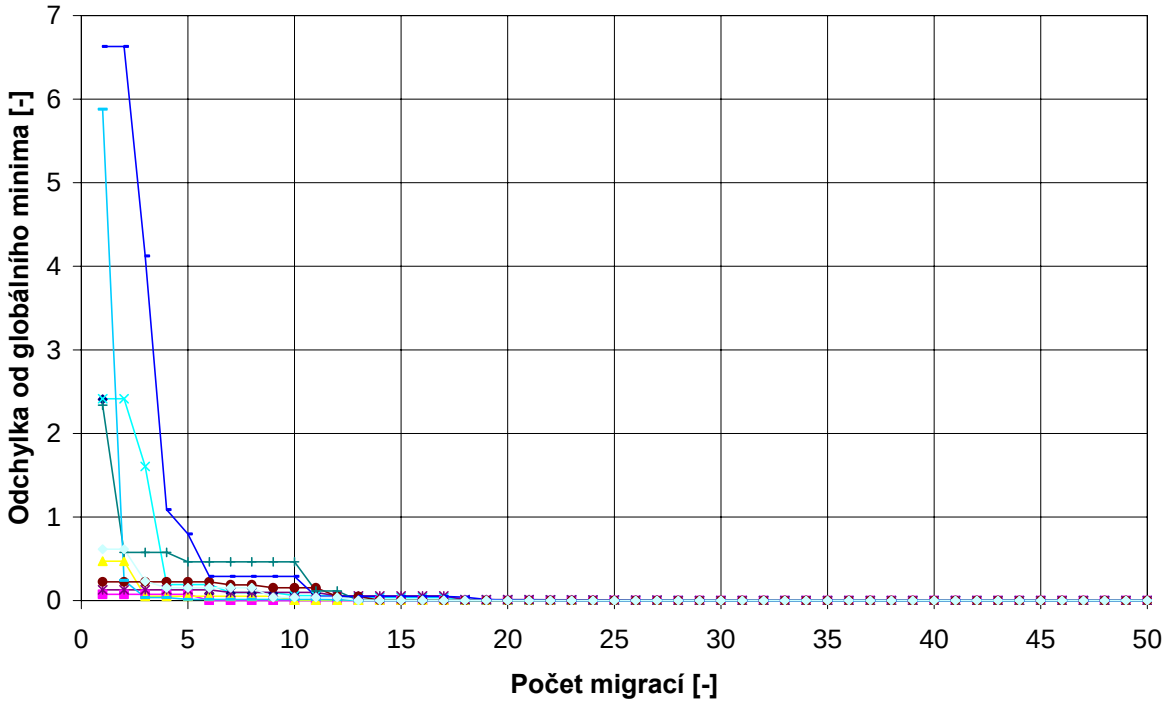
Obr. 30 Statistická charakteristika při použití optimalizace DE na testovací funkci De Jong pro 10 opakování.



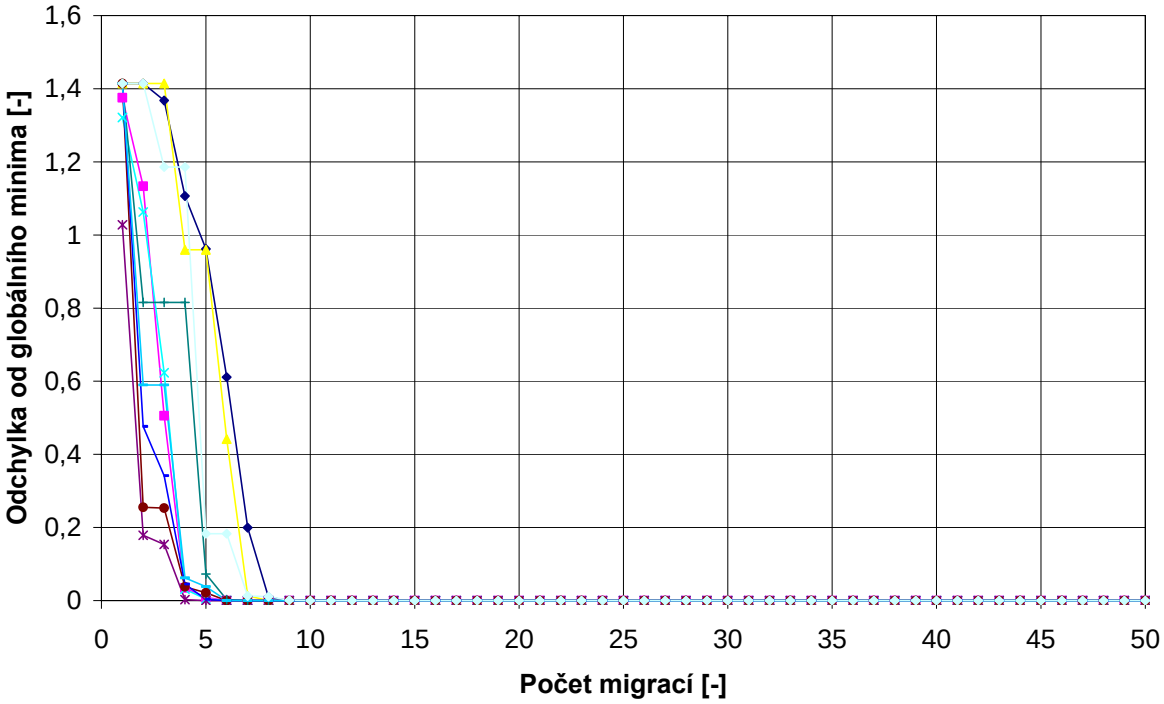
Obr. 31 Statistická charakteristika při použití optimalizace DE na testovací funkci Easom pro 10 opakování.



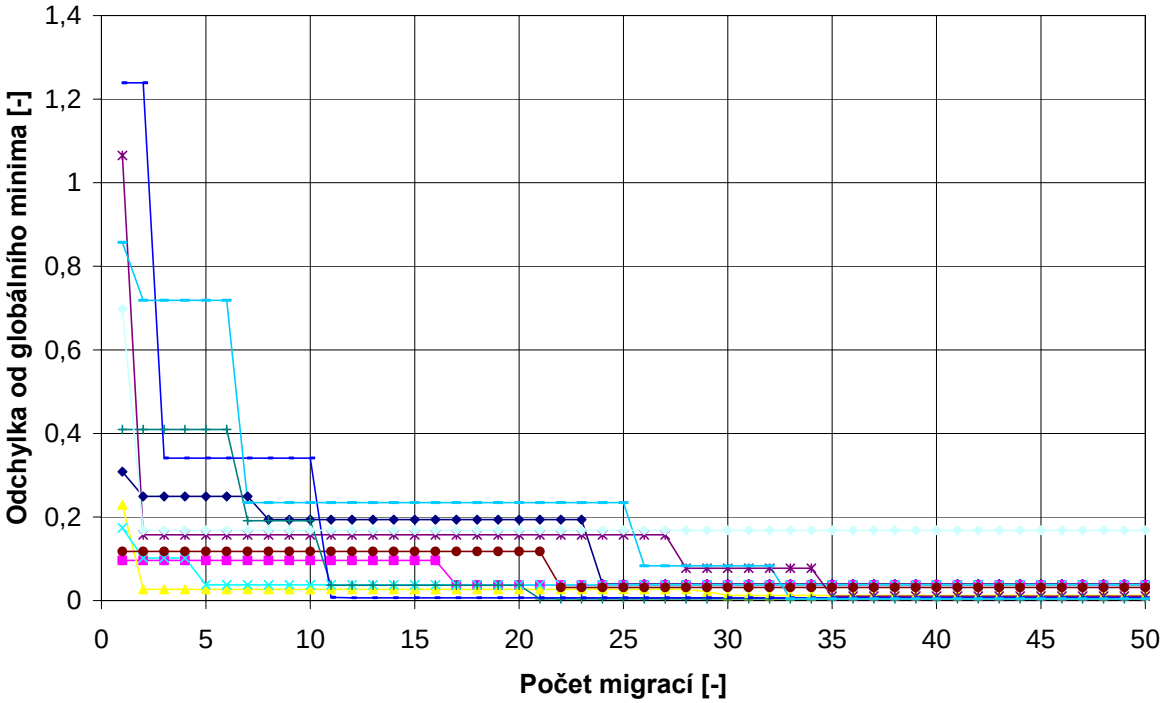
Obr. 32 Statistická charakteristika při použití optimalizace DE na testovací funkci Shubert pro 10 opakování.



Obr. 33 Statistická charakteristika při použití optimalizace SOMA na testovací funkci DE Jong pro 10 opakování.



Obr. 34 Statistická charakteristika při použití optimalizace SOMA na testovací funkci Easom pro 10 opakování.



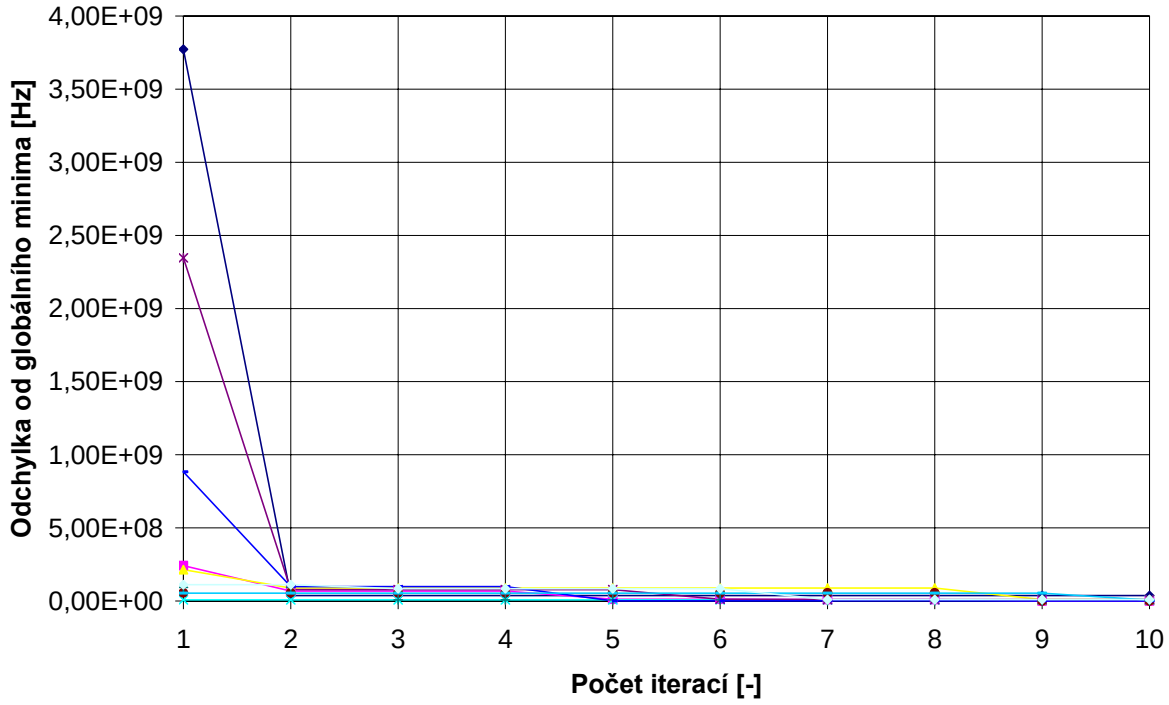
Obr. 35 Statistická charakteristika při použití optimalizace SOMA na testovací funkci Shubert pro 10 opakování.

Na vybraných testovacích funkcích byla testována výkonnost optimalizačních algoritmů. Optimalizačními algoritmy byly PSO, DE, SOMA. Vybrané testovací funkce byly pro optimalizaci nastaveny podle literatury [8], kde nastavované parametry byly velikosti hranic

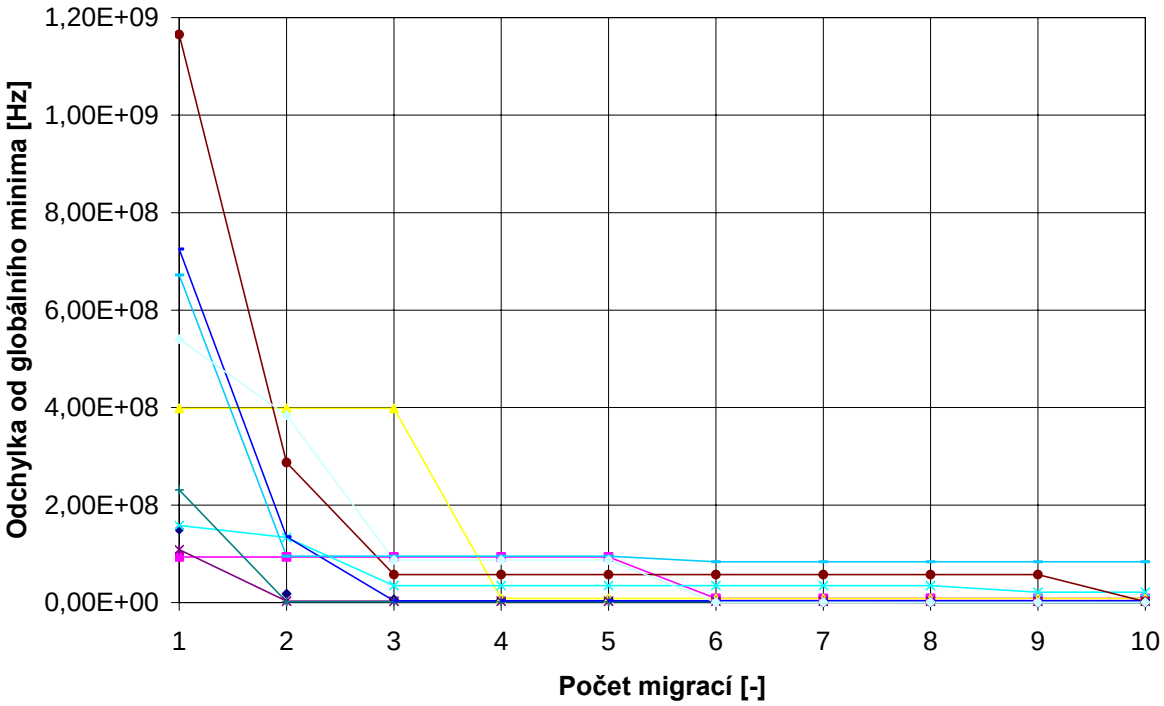
a hledaný parametr bylo globální minimum v tomto intervalu hranic. Globální minimum je v těchto případech nula, protože byla použita popsaná kriteriální funkce, kde zadaná hodnota je vždy globální minimum pro jednotlivě vybrané testovací funkce. Z grafů je vidět, nejlepší optimalizace se zdá být SOMA následně DE a poslední PSO. Toto pořadí lze určit podle schopnosti nalézt globální minimum u testovacích funkcí relativně málo (migracích, generacích, iteracích).

6.2 Reálná funkce k určení výkonnosti optimalizačních algoritmů

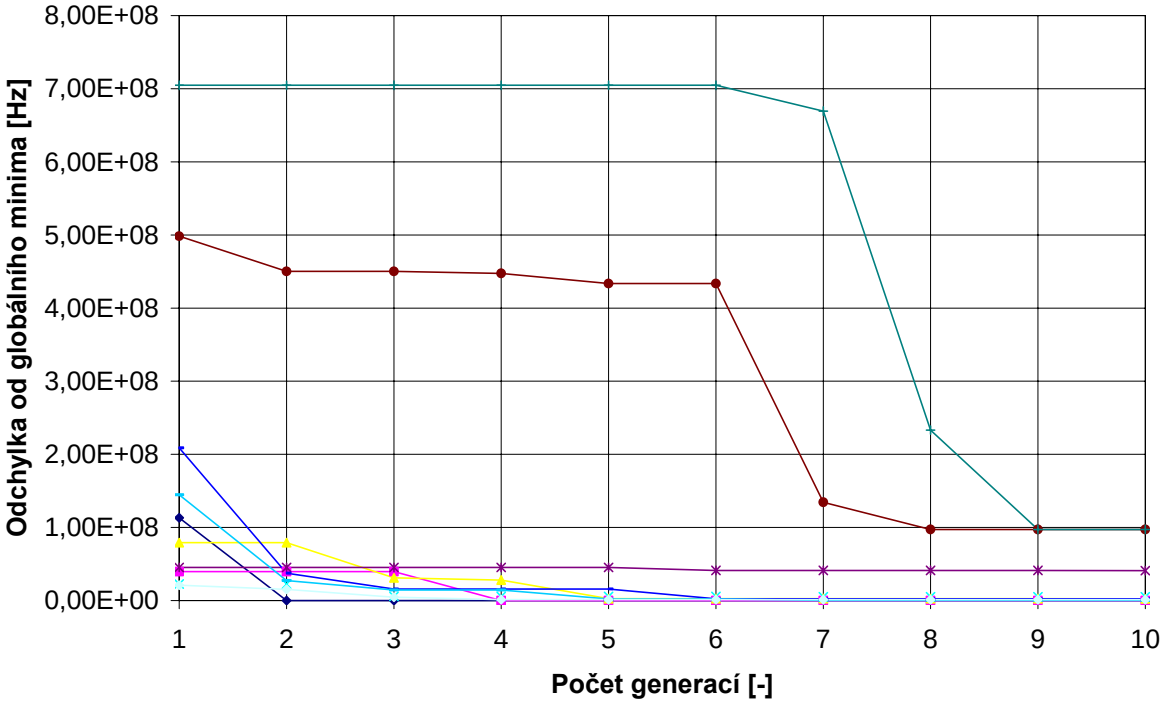
Nyní testovací funkce jsou nahrazeny reálným modelem, viz obr .č.4. Model je sestaven vždy pomocí Toolboxu, kde sledovaný parametr je rezonanční frekvence. Požadovaná rezonanční frekvence je 1,6 GHz. Ve statistických grafech jsou sledovány odchylky od globálního minima, tj. 1,6 GHz. Statistické charakteristiky jsou testovány pro oba simulační programy, tak i pro popsané optimalizační metody na popsaném reálném modelu, viz obr .č.4. Opět aby bylo možné tyto charakteristiky označit jako statistické jsou optimalizace spuštěny alespoň 10 krát. Z časových důvodů jsou počty (migrací,iterací nebo generací) velmi omezeny, však mají dostatečnou vypovídací hodnotu.



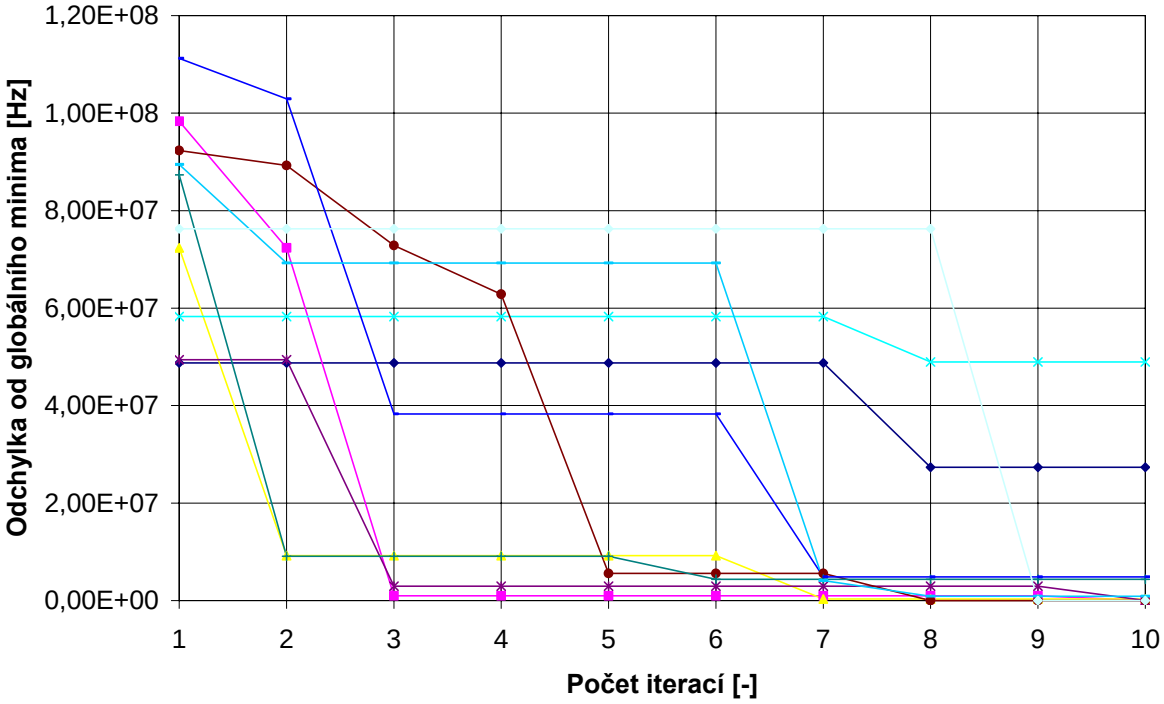
Obr. 36 Statistická charakteristika při použití optimalizace PSO na reálném modelu v programu CST pro 10 opakování.



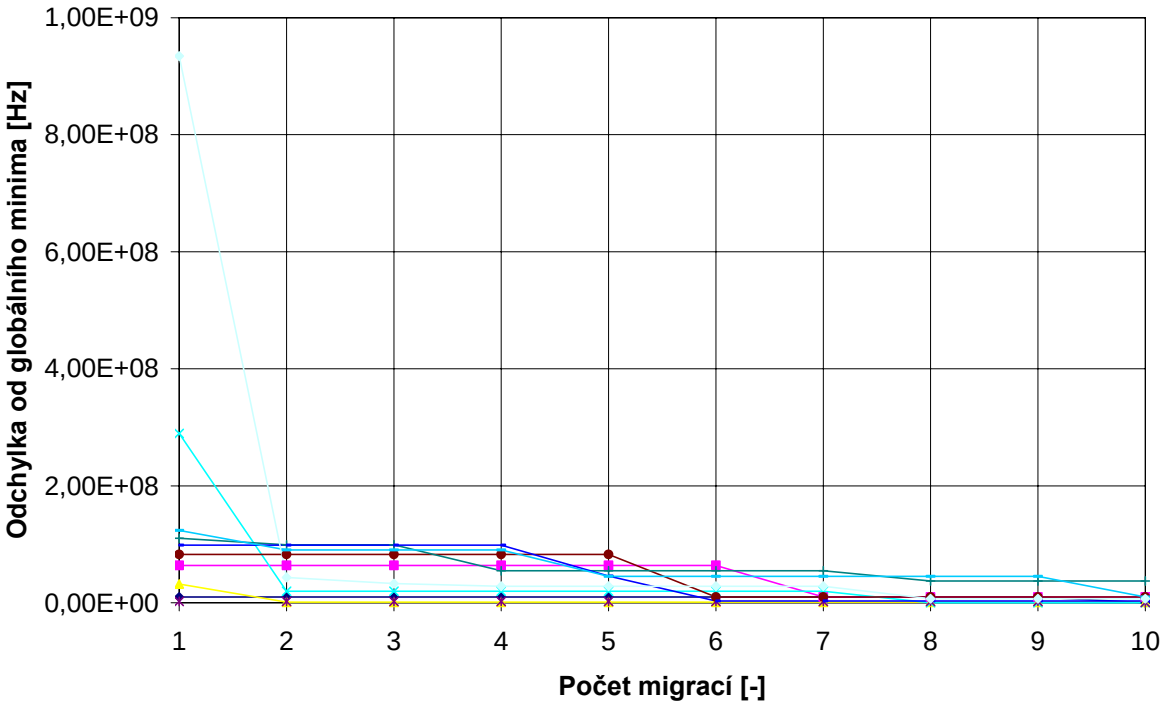
Obr. 37 Statistická charakteristika při použití optimalizace SOMA na reálném modelu v programu CST pro 10 opakování.



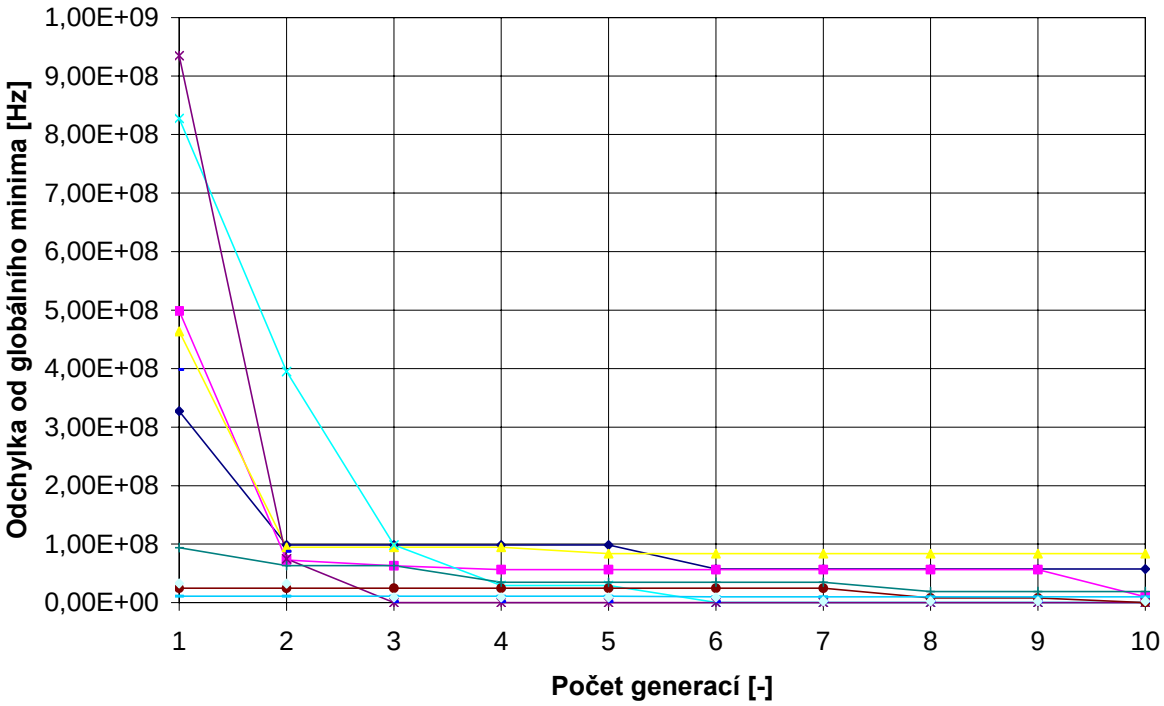
Obr. 38 Statistická charakteristika při použití optimalizace DE na reálném modelu v programu CST pro 10 opakování.



Obr. 39 Statistická charakteristika při použití optimalizace PSO na reálném modelu v programu HFSS pro 10 opakování.



Obr. 40 Statistická charakteristika při použití optimalizace SOMA na reálném modelu v programu HFSS pro 10 opakování.



Obr. 41 Statistická charakteristika při použití optimalizace DE na reálném modelu v programu HFSS pro 10 opakování.

K ověření výkonnosti byl použit reálný model, viz obr. 4. K ověření byly použity globální optimalizační algoritmy, mezi které patří PSO, DE a SOMA. U těchto algoritmů byly nastaveny (migrace, generace, iterace) na 10 a počet jedinců populace je 8. Optimalizovaný model měl nastaveny všechny parametry stejně pro všechny optimalizační techniky, tj. okrajové podmínky. Z uvedených grafických údajů, lze říci, že všechny použité optimalizační metody jsou schopné nalézt globální minimum pro jednoduchý reálný model. Nejlepší nalezená hodnota u PSO, DE a SOMA pomocí programu CST je nula, což znamená, že bylo nalezeno přesné umístění globálního minima a nejlepší nalezená hodnota u PSO a DE pomocí programu HFSS, jenž je také nula. Z uvedeného lze říci, že optimalizovaný model je též velice jednoduchý k ohodnocení výkonnosti optimalizačních algoritmů.

7. Závěr

Diplomová práce popisuje rozhraní mezi dvěma externími simulačními programy a programem MATLAB. Mezi vybrané simulační programy byly vybrány CST a HFSS. Tyto programy splňují dvě hlavní důležité podmínky, které jsou simulace elektromagnetických struktur ve 3D rozměru a druhou podmínkou bylo skriptovací rozhraní pomocí VBA. Součástí bylo vytvořit propojovací vrstvu mezi simulačními programy a programem MATLAB. Aby bylo možné využívat propojovací vrstvu z MATLABu bylo potřeba funkce tvořící propojovací vrstvu konvertovat. Hlavním důvodem konverze je velmi malá rozšířenost schopností programování ve VBA v porovnání s MATLABem mezi uživateli elektromagnetických simulačních programů.

Následně byl popsán Toolbox tvořený z jednotlivých datových struktur a funkcí propojující MATLAB a simulační programy. Byl rozebrán způsob tvorby jednotlivých funkcí implementovaných v Toolbox u včetně krátkých ukázek zdrojového kódu pro oba simulační programy. Dále byl ukázán způsob ovládání Toolbox u na funkční ukázce, včetně popisu. Součástí diplomové práce byla implementace globálních optimalizačních algoritmů, jenž jsou PSO, DE, SOMA. Pomocí optimalizace PSO byla ověřena funkčnost propojovací vrstvy na jednoduchém elektromagnetickém problému. U zkoumaného modelu byla požadována rezonanční frekvence, která byla 1,6 GHz. Model byl vždy sestaven pomocí Toolboxu. Vrácené hodnoty z optimalizačního algoritmu byla f =1,622 GHz pro CST a 1,596 GHz pro HFSS.

V poslední části byl proveden rozbor výkonnosti jednotlivých optimalizačních algoritmů pomocí statistických charakteristik. Rozbor byl proveden na testovacích funkcích ale také na elektromagnetickém problému. Z testovacích funkcí byly vybrány tři, podle jejich složitosti. Tyto funkce byly spuštěny vždy desetkrát v jednotlivých optimalizacích. Výsledkem lze říci, že jednotlivé optimalizace konvergují do globálního minima už do prvních deseti (migracích, generacích nebo iteracích). Dá se říci, že konvergence u optimalizací má náhodný charakter a lze říci, že optimalizace jsou schopny nalézt globální minimum bez větších problému u vybraných testovacích funkcí. Poté byly optimalizace opět použity stejným způsobem, ale místo testovacích funkcí byl použit reálný model. U reálného modelu byla zkoumána konvergence k požadované rezonanční frekvenci, která byla 1,6 GHz. Reálný model byl tvořen Toolboxem pro oba simulační programy. Konvergence u reálného modelu má také náhodný charakter. Lze říci, že dané optimalizace jsou schopny nalézt globální minimum bez větších problémů u tohoto použitého jednoduchého modelu.

Seznam literatury

[1] IVACHIV, P., KRAVAL, I., Základy komponentní technologie COM. Brno: Computer Press, 1999. 243s. IBSN 80-7226-101-0.

[2] Technologie ACTIVEX [online]. 2009. Dostupné z : <http://tereza.fjfi.cvut.cz/pipermail/fanda/2000-April/002507.html >.

[3] CST Microwave studio. [online].2009. Dostupné z : < http://www.cst.com>.

[4] ANSOFT HFSS. [online].2009. Dostupné z : < http://www.ansoft.com>.

[5] KIN –LU, W. Compact and Boardband Microstrip Antennas. New York: John Wiley & Sonc, Inc., 2002

[6] ZELINKA, I., Umělá intelligence v problémech globální optimalizace. Praha: Ben, 2002. 192s. IBSN 80-7300-069-5.

[7] ROBINSON, J., RAHMAT-SAMII, Y.Particle Swarm Optimization in Electromagnetics. IEEE Transactions on Antennas and Propagation, February 2004, vol. 52, no. 2

[8] MOLGA, M., SMUTNICKI, C., Test functions for optimization needs. Poland: 2005

Seznamy zkratek

2D – dva rozměry
3D – tři rozměry
CST – Program CST Microwave Studio
DE - Diferenciální evoluce
HFSS – Program Ansoft HFSS
MATLAB – Program MATLAB
PSO – Metoda roje částic
SOMA – Samoorganizující se a migrační algoritmus
VBA – VISUAL BASIC

Seznam příloh

Tištěná příloha:

- Referenční příručka

Elektronická příloha na CD obsahuje soubory:

- text
- TOOLBOX
- výsledky

Příloha

Referenční příručka

Component

CstDeleteComponent

Funkce `CstDeleteComponent(fid,CComponent)` slouží k vymazání již existující komponenty. Nemá návratovou hodnotu.
Parametry funkce:

`fid` - soubor identifikující historii CST skriptu.
`CComponent` - jméno požadované komponenty.

Příklad použití funkce:

```
fid = fopen('pokus.m','wt');
...
CstDeleteComponent(fid,'ground');
```

CstChangeComponent

Funkce `CstChangeComponent(fid,CNew,COld)` je vhodná ke změnění typu komponenty. Požadovaná stará komponenta musí být již vytvořena před zavoláním této funkce. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii CST skriptu.
`CNew` - jméno nové komponenty
`COld` - jméno staré komponenty.

Příklad použití funkce:

```
fid = fopen('pokus.m','wt');
...
CstChangeComponent(fid,'nova_komponenta','stara_komponenta');
```

CstNewCompnent

Funkce `CstNewComponent(fid,CComponent)` slouží k vytvoření nové komponenty. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii CST skriptu.
`CComponent` – jméno nové komponenty, která bude vytvořena.

Příklad použití funkce:

```
fid = fopen('pokus.m','wt');
...
CstNewComponent(fid,'ground');
```

Global

CstNewProject

Funkce `CstNewProject(fid)` slouží k vytvoření COM serveru pomocí Active X mezi programem CST a MATLABem. Funkce je základem pro práci se skripty a stojí vždy na začátku skriptovacího souboru. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii CST skriptu.

Příklad použití funkce:

```
TmpMyFile = 'C:\temp\myfile.m';
fid = fopen(TmpMyFile, 'wt');
CstNewProject(fid);
```

CstUnits

Funkce `CstUnits (fid,set,UGeometry,UFrequency,UTime,UTemperature,UVoltage, UCurrent,UResistance,UConductance,UCapacitance,UInductance)` slouží k nastavení jednotek v programu CST. Funkce nastavuje jednotky globálně. Nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii CST skriptu.
`set` – Ve funkci je možné volit mezi dvěmi variantami:
1) Varianta "full", kde je možné nastavovat všechny jednotky používané v CST Studiu jako jsou jednotky rozměru, frekvence, času, teploty, napětí, proudu, odporu,vodivosti,kapacity a indukčnosti. U varianty "full" je potřeba zadávat všechny zmíněné jednotky

2) Varianta "easy", je zjednodušena a jsou v ni potřeba zadávat pouze nejčastěji používané jednotky, mezi které patří jednotky rozměru, frekvence a času.

`UGeometry` – jednotky geometrie ('m','mm','cm' a další definované v CST).
`UFrequency` – jednotky frekvence ('Hz','GHz','MHz' a další definované v CST).
`UTime` – jednotky času ('s','ms','ns' a další definované v CST).
`UTemperature` – jednotka teploty.
`UVoltage` – jednotka napětí.
`UCurrent` – jednotka proudu.
`UResistance` – jednotka odporu.
`UConductance` – jednotka konduktivity.

`UCapacitance` – jednotka kapacity.
`UInductance` – jednotka indukce.

Příklad použití funkce:

```
fid = fopen('pokus.m','wt');
 ...
```

Varianta 1("full"):
```
CstUnits(fid,'full','m','Hz','s','Celsius','V','A','Ohm','S','pF','nH');
```

Varianta 2("easy"):
```
CstUnits(fid,'easy','m','Hz','s','','','','','','','');
```

Import_Export

GetFrequencyFromDiagram

Funkce `SF = GetFrequencyFromDiagram(CUroven,CFile)` slouží k vyčíslení frekvence a činitele odrazu z vyexportovaných hodnot z programu CST. Funkce vrací hodnoty frekvence, činitele odrazu a také šířku laloku frekvence.

Parametry funkce:

`CUroven` - rozsah hodnot od 1 do 0 (hodnota 1 už je bez rozlišení!).
`CFile` – cesta vyexportovaného souboru.

Příklad použiti funkce:

```
fid = fopen('pokus.m', 'wt');
...
SF = GetFrequencyFromDiagram(0.5,'C:/Temp/tempDataExportCST.txt');
```

Materials

CstDefineMaterialDefault

Funkce `CstDefineMaterialDefault (fid,CName,CType,CEpsilon,CMue,CKappa,CTanD, CTanDFreq, CTanDGiven, CTanDModel, CKappaM, CTanDM, CTanDMFreq, CTanDMGiven, CDispModelEps, CDispModelMue, CRho, CThermal, CColA, CColB, CColC, CWireframe, CTransparency)` slouží k nadefinování vlastního nového materiálu, který není součástí programu CST.

Parametry funkce:

`fid` – soubor identifikující historii CST skriptu.
`CName` – jméno definovaného materiálu.
`CType` – typ materiálu (normál nebo PEC)
`CEpsilon` – jednotka permitivity

CMue - jednotka permeabillity
CKappa – jednotka vodivosti.
CTanD – jednotka elektrický ztrátový úhel.
CTanDFreq – jestliže je CTanDGiven „true“, pak specifikuje frekvenci u které byl určen ztrátový činitel.
CTanDGiven – definuje zda se jedná o ztrátový materiál („true“,“false“).
CTanDModel – nastaví ztrátový činitel u materiálu.
CKappaM – jednotka magnetické vodivosti.
CTanDM – nastaví magnetický ztrátový úhel.
CTanDMFreq – jestliže je CTanDMGiven „true“, pak specifikuje frekvenci u které byl určen ztrátový činitel.
CTanDMGiven – definuje zda se jedná o ztrátový materiál („true“,“false“).
CDispModelEps – definuje elektrický rozptyl v materiálu.
CDispModelMue – definuje magnetický rozptyl v materiálu.
CRho – jednotka hustoty materiálu (kg/m³).
CThermal – definuje teplotní vodivost materiálu.
CColA – nastavení barevné složky „R“ –červené.
CColB – nastavení barevné složky „G“ –zelené.
CColC – nastavení barevné složky „B“ –modré.
CWireframe – jestliže je nastaveno „true“ pak všechny pevné látky spojené s tímto materiálem budou zobrazeny.
CTransparency – nastavení průhlednosti materiálu.

Příklad použití funkce:

```
fid = fopen('pokus.m', 'wt');
...
CstDefineMaterialDefault(fid,'Copper','Normal',1,1,5.8e+007,0,0,0,
False,ConstTanD,None,None,1,'ConstTanD',0,0,0,'False','None','None');
```

Monitors

CstMonitor

Funkce CstMonitor(fid,CName,CDomain,CFtype,CFrequency) je vhodná pro vytvoření monitorování na zvolené frekvencí. Funkce nemá návratovou hodnotu.

Parametry funkce:

fid - soubor identifikující historii CST skriptu.
CName – jméno monitoru.
CDomain – typ domény(frekvenční, časová).
CFtype – nastavení co bude monitorováno („Efield“, „Hfield“, „Farfield“, atd.).
CFrequency – monitorovaná frekvence.

Příklad použití funkce:

```
fid = fopen('pokus.m', 'wt');
...
CstMonitor(fid,'farfield (f=1.6433)','Frequency','Farfield',1.6433);
```

Operation on solid

CstSolid

Funkce `CstSolid(fid,CProces,CComponent,CName_a,CName_b)` je potřeba k vyříznutí jednoho objektu do druhého objektu. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii CST skriptu.
`CProces` – proces, který se provede (může být vyříznutí, přidání, vložení, atd).
`CComponent` – komponenta, ve kterém jsou dané objekty.
`CName_a` – jméno prvního objektu.
`CName_b` – jméno druhého objektu.

Příklad použití funkce:

```
fid = fopen('pokus.m', 'wt');
...
CstSolid(fid,'Subtract','P_ant','Antena','zarez3');
```

Ports

CstWaveguidePort

Funkce `CstWaveguidePort(fid, CPnumber, CNmode, CAdPolarization, CPolAngle, CRefDist, CTextSize, CCoordinates, COrientation, CPortOnBound, CClipPickedPortToBound, CXrangeMin, CXrangeMax, CYrangeMin, CYrangeMax, CZrangeMin, CZrangeMax, CXrangeAddMin, CXrangeAddMax, CYrangeAddMin, CYrangeAddMax, CZrangeAddMin, CZrangeAddMax)` nastavuje vlnový port v programu CST. Nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii CST skriptu.
`CPnumber` – nastaví číslo vlnového portu.
`CNmode` – nastaví mód vlnového portu.
`CAdPolarization` – rozhodující prvek zda polarizace elektrického pole by měla být nastavená nebo ne, tato metoda pracuje spolu s metodou `CPolAngle`.
`CPolAngle` – jestliže je aktivována metoda `CAdPolarization` může být polarizační úhel definován.
`CRefDist` – nastaví odstup referenční roviny a parametr S bude vypočítán na referenční úrovní.
`CTextSize` – velikost písma.
`CCoordinates` – určuje jak je definována transverzální expanze vlnového portu.
`COrientation` – definuje orientaci, směr napájení, vlnového portu.
`CPortOnBound` – rozhodne jestliže port je umístěn na hranicí výpočetní domény „true“ nebo může být umístěn v doméně výpočtu kvůli jeho definici normálního uložení (Xrange, Yrange nebo Zrange metoda) „false“.

`CClipPickedPortToBound` – u vybraného portu se tato metoda rozhodne jestliže port by měl být umístěn na hranicích výpočetní domény „ture“ nebo by měl být stanovený korespondent k informacím výběru „false“.
`CXrangeMin` – | průsečné osy portu vlnovodu jsou definovány jako "Free", tyto metody určí
`CXrangeMax` – | rozsah v každém směru, (x, y, z)
`CYrangeMin` – |
`CYrangeMax` – |
`CZrangeMin` – |
`CZrangeMax` – |
`CXrangeAddMin` – | doplňkové rozměry (x,y,z), běžně jsou nastaveny na (0.0).
`CXrangeAddMax` – |
`CYrangeAddMin` – |
`CYrangeAddMax` – |
`CZrangeAddMin` – |
`CZrangeAddMax` – |

Příklad použití funkce:

```
fid = fopen('pokus.m', 'wt');
...
CstWaveguidePort(fid,1,1,'False',0.0,0,45,'Free','zmin','True',...
'False',-5,5,-2,2,-9.635,-9.635,0.0,0.0,0.0,0.0,0.0,0.0);
```

Results

CstGetFrequencyResults

Funkce `CstGetFrequencyResults` slouží ke získání výsledků z programu CST. Získané výsledky jsou frekvence a činitel odrazu, jak v míře bezrozměrné, tak i v míře decibelové. Funkce je převážně podobná jako funkce `GetFrequencyFromDiagram`, ale tato funkce nemá nastaveny rozhodovací funkce, pro zjištění zda se jedná o širokopásmovou anténu či nikoli nebo když dojde k situaci, kdy budou dvě frekvence blízko sebe, program CST nedokáže rozhodnout, zda se tak jedná. Funkce nemá žádné měnitelné parametry, ve funkci je pevně nastavena odečítací úroveň, což v decibelové míře je asi okolo -6dB. Funkce vrací hodnoty frekvence a činitele odrazu a šířku laloku jednotlivé frekvence.

Příklad použití funkce:

```
fid = fopen('pokus.m', 'wt');
...
VectorFaS = CstGetFrequencyResults;
```

Solid

CstBrick

Funkce `CstBrick (fid,CName,CComponent,CMaterial,CXrangeA,CXrangeB,CYrangeA, CYrangeB,CZrangeA,CZrangeB)` je k vytvoření nové krychle v programu CST. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` – soubor identifikující historii CST skriptu.
`CName` – jméno krychle.
`CComponent` – jméno komponenty.
`CMaterial` – materiál objektu.

`CXrangeA` – |
`CYrangeA` – rozměr krychle od x,y,z.
`CZrangeA` – |

`CXrangeB` – |
`CYrangeB` – rozměr krychle do x,y,z.
`CZrangeB` - |

Příklad použití funkce:

```
fid = fopen('pokus.m', 'wt');
...
CstBrick(fid,'brick1','component1','PEC',-1,1,-2,2,-3,3);
```

CstCone

Funkce `CstCone(fid, CName, CComponent, CMaterial, CAxis, CTopradius, CBottomradius ,CXcenter, CYcenter, CZcenter, CRangeA, CRangeB, CSegments)` je k vytvoření kónusu programu CST. Nemá návratovou hodnotu.

Parametry funkce:

`fid` – soubor identifikující historii CST skriptu.
`CName` – jméno kónusu.
`CComponent` – jméno komponenty.
`CMaterial` – materiál objektu.
`CAxis` – osa otočení kónusu může být 'X','Y' nebo 'Z'.
`CTopradius` – poloměr vrcholu kónusu.
`CBottomradius` – poloměr základny kónusu.
`CXcenter` – poloha kónusu ve směru X.
`CYcenter` – poloha kónusu ve směru Y.
`CZcenter` – poloha kónusu ve směru Z.
`CRangeA` – je rozměr min, je-li například CAxis = 'Z', pak CRangeA bude Zmin.
`CRangeB` – je rozměr max, je-li například CAxis = 'Z', pak CRangeB bude Zmax.
`CSegments` – určuje kolik bude mít stran, pro hodnotu 0-2 je kónus rotační a pro hodnoty segmentu větší než 2 určuje tento parametr kolik bude mít stran.

Příklad použití funkce:

```
fid = fopen('pokus.m', 'wt');
...
CstCone(fid,'cone1','component1','PEC','Z',0.5,2.0,2,1,0,0,'a+3',0);
```

CstCylinder

Funkce `CstCylinder(fid, CName, CComponent, CMaterial, CAxis, COuterradius, CInnerradius, CAcenter, CBcenter, CRangeA, CRangeB, CSegments)` slouží k vytvoření válce v programu CST. Nemá návratovou hodnotu.

Parametry funkce:

`fid` – soubor identifikující historii CST skriptu.
`CName` – jméno válce.
`CComponent` – jméno komponenty.
`CMaterial` – materiál objektu.
`CAxis` – osa otočení válce může být 'X','Y' nebo 'Z'.
`COuterradius` – poloměr venkovního rozměru.
`CInnerradius` – poloměr vnitřního rozměru.
`CAcenter` – umístění válce v ose X, jestli-že CAxis bude 'Z'.
`CBcenter` – umístění válce v ose Y, jestli-že CAxis bude 'Z'.
`CRangeA` – je rozměr min, je-li například CAxis = 'Z', pak CRangeA bude Zmin.
`CRangeB` – je rozměr max, je-li například CAxis = 'Z', pak CRangeB bude Zmax.
`CSegments` – určuje kolik bude mít stran, pro hodnotu 0-2 je kónus rotační a pro hodnoty segmentu větší než 2 určuje tento parametr kolik bude mít stran.

Příklad použití funkce:

```
fid = fopen('pokus.m', 'wt');
...
CstCylinder(fid,'Dira','S_ant','Vacuum','Z',0.5,0,11,12,0,1.6,0);
```

CstECylinder

Funkce `CstECylinder(fid, CName, CComponent, CMaterial, CAxis, CRadiusA, CRadiusB, CXcenter, CYcenter, CZcenter, CRangeA, CRangeB, CSegments)` slouží k vytvoření eliptického válce. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` – soubor identifikující historii CST skriptu.
`CName` – jméno eliptického válce.
`CComponent` – jméno komponenty.
`CMaterial` – materiál objektu.
`CAxis` – osa otočení eliptického válce může být 'X','Y' nebo 'Z'.
`CRadiusA` – poloměr v ose X, jestli-že CAxis bude 'Z'.
`CRadiusB` – poloměr v ose Y, jestli-že CAxis bude 'Z'.
`CXcenter` – umístění válce v ose X.
`CYcenter` – umístění válce v ose Y.
`CZcenter` – umístění válce v ose Z.
`CRangeA` – je rozměr min, je-li například CAxis = 'Z', pak CRangeA bude Zmin.
`CRangeB` – je rozměr max, je-li například CAxis = 'Z', pak CRangeB bude Zmax.
`CSegments` – určuje kolik bude mít stran, pro hodnotu 0-2 je eliptický válec rotační a pro hodnoty segmentu větší než 2 určuje tento parametr kolik bude mít stran.

Příklad použití funkce:

```
fid = fopen('pokus.m', 'wt');
...
CstECylinder(fid,'Ecyl1','component1','PEC','Z',1.5,0.5,2,1,0,0,3,0);
```

CstSphere

Funkce `CstSphere(fid, CName, CComponent, CMaterial, CAxis, CCenterRadius, CTopRadius, CBottomRadius, CCenterX, CCenterY, CCenterZ, CSegments)` slouží k vytvoření koule v programu CST. Nemá návratovou hodnotu.

Parametry funkce:

`fid` – soubor identifikující historii CST skriptu.
`CName` – jméno eliptického válce.
`CComponent` – jméno komponenty.
`CMaterial` – materiál objektu.
`CAxis` – osa otočení eliptického válce může být 'X','Y' nebo 'Z'.
`CCenterRadius` – poloměr koule.
`CTopRadius` – poloměr na vrcholu koule.
`CBottomRadius` – poloměr na základně koule.
`CCenterX` – umístění koule v ose X.
`CCenterY` – umístění koule v ose Y.
`CCenterZ` – umístění koule v ose Z.
`CSegments` – určuje kolik bude mít stran, pro hodnotu 0-2 je koule rotační a pro hodnoty segmentu větší než 2 určuje tento parametr kolik bude mít stran.

Příklad použití funkce:

```
fid = fopen('pokus.m', 'wt');
...
CstSphere(fid,'sphere1','component1','PEC','Z',1,0,0,2,1,3,0);
```

CstTorus

Funkce `CstTorus (fid, CName, CComponent, CMaterial, CAxis, COuterradius, CInnerradius, CXcenter, CYcenter, CZcenter, CSegments)` slouží k vytvoření kruhového prstence. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` – soubor identifikující historii CST skriptu.
`CName` – jméno kruhového prstence.
`CComponent` – jméno komponenty.
`CMaterial` – materiál objektu.
`CAxis` – osa otočení kruhového prstence může být 'X','Y' nebo 'Z'.
`COuterradius` – vnější poloměr kruhového prstence.
`CInnerradius` – vnitřní poloměr kruhového prstence.
`CXcenter` – umístění kruhového prstence v ose X.
`CYcenter` – umístění kruhového prstence v ose Y.

`CZcenter` – umístění kruhového prstence v ose Z.
`CSegments` – určuje kolik bude mít stran, pro hodnotu 0-2 je koule rotační a pro hodnoty segmentu větší než 2 určuje tento parametr kolik bude mít stran.

Příklad použití funkce:

```
fid = fopen('pokus.m', 'wt');
...
CstTorus(fid,'torus1','component1','PEC','Z',1.5,0.5,2,1,0,0);
```

Solver

CstBackground

Funkce `CstBackground(fid, CType, CEps, CMue, CXminSpace, CXmaxSpace, CYminSpace, CYmaxSpace, CZminSpace, CZmaxSpace, CThermalType, CThermalConductivity, CApplyInAllDirections)` slouží k nastavení prostředí, ve kterém se modelovaný objekt nachází. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` – soubor identifikující historii CST skriptu.
`CType` – typ prostředí.
`CEps` – definování permitivity.
`CMue` - definování permeability.
`CXminSpace` – rozměr prostoru ve směru X.
`CXmaxSpace` – rozměr prostoru ve směru X.
`CYminSpace` – rozměr prostoru ve směru Y.
`CYmaxSpace` – rozměr prostoru ve směru Y.
`CZminSpace` – rozměr prostoru ve směru Z.
`CZmaxSpace` – rozměr prostoru ve směru Z.
`CThermalType` – typ teplotně závislého materiálu (normal nebo PTC).
`CThermalConductivity` – teplotní vodivost prostředí.
`CApplyInAllDirections` – použití rozměru prostoru ve všech směrech.

Příklad použití funkce:

```
fid = fopen('pokus.m', 'wt');
...
CstBackground(fid,'Normal',1,1,0,0,0,0,0,0,'Normal',0,'True');
```

CstBoundary

Funkce `CstBoundary(fid, CXmin, CXmax, CYmin, CYmax, CZmin, CZmax, CXsymmetry, CYsymmetry, CZsymmetry, CXminT, CXmaxT, CYminT, CYmaxT, CZminT, CZmaxT, CApplyInAllDirections, CXminTE, CXmaxTE, CYminTE, CYmaxTE, CZminTE, CZmaxTE, CXminTT, CXmaxTT, CYminTT, CYmaxTT, CZminTT, CZmaxTT)` slouží k nastavení okrajových podmínek namodelovaného objektu.

Parametry funkce:

`fid` – soubor identifikující historii CST skriptu.
`CXmin` – | typ okrajové podmínky.
`CXmax` – | může být např. „electric“,“magnetic“,“open“,
`CYmin` – | a další implementované v programu CST.
`CYmax` – |
`CZmin` – |
`CZmax` – |
`CXsymmetry` – | definuje namodelovanou strukturu, zda je symetrická vůči některé z os X,Y
`CYsymmetry` – | nebo Z, parametry jsou elektrická, magnetická a nebo žádná symetrie.
`CZsymmetry` – |
`CXminT` – | definuje teplotní typ okrajové podmínky.
`CXmaxT` – |
`CYminT` – |
`CYmaxT` – |
`CZminT` – |
`CZmaxT` – |
`CApplyInAllDirections` – jestli-že bude „true“ bude typ okrajové podmínky nastaven ve všech směrech, hodnota ostatních bude stejná jako CXmin.
`CXminTE` – | specifikuje teplotní typ.
`CXmaxTE` – |
`CyminTE` - |
`CYmaxTE` – |
`CZminTE` – |
`CZmaxTE` – |
`CXminTT` – | specifikuje teplotní hodnotu okrajové podmínky.
`CXmaxTT` – | hodnoty mohou být („none“, „fixed“ a „floating“.
`CYminTT` – |
`CYmaxTT` – |
`CZminTT` – |
`CZmaxTT` - |

Příklad použití funkce:

```
      fid = fopen('pokus.m', 'wt');
      ...
CstBoundary(fid,'expanded open','expanded open','expanded open','expanded
open', 'open', 'expanded open', 'none',     'none', 'none', 'isothermal',
'isothermal',  'isothermal',  'isothermal',  'isothermal',  'isothermal',
'False', '', '', '', '', '', '', 'None', 'None', 'None', 'None', 'None',
'None');
```

CstSetFrequencyRange

Funkce `CstSetFrekvencyRange(fid,CFmin,CFmax)` nastaví frekvenční rozsah ve kterém bude provedena analýza. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` – soubor identifikující historii CST skriptu.
`CFmin` – minimální frekvence.
`CFmax` – maximální frekvence.

Příklad požití funkce:

```
fid = fopen('pokus.m', 'wt');
...
CstSetFrekvencyRange(fid,1,3);
```

CstStartSolver

Funkce `CstStartSolver(fid)` slouží ke spuštění analýzy. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` – soubor identifikující historii CST skriptu.

Příklad použití funkce:

```
fid = fopen('pokus.m', 'wt');
...
CstStartSolver(fid);
```

CstTimeDomainSolverEasy

Funkce `CstTimeDomainSolverEasy(fid, CCalculationType, CStimulationPort, CStimulationMode, CSteadyStateLimit, CMeshAdaption, CAutoNormImpedance, CNormingImpedance, CCalculateModesOnly, CSParaSymmetry, CStoreTDResultsInCache, CFullDeembedding, CUseDistributedComputing)` slouží k nastavení časové analýzy. Nemá návratovou hodnotu.

Parametry funkce:

`fid` – soubor identifikující historii CST skriptu.
`CCalculationType` – definuje výpočetní analýzu.
`CStimulationPort` – výběr portů, které budou použity pro vybuzení.
`CStimulationMode` – výběr módu, který bude použit pro vybuzení.
`CSteadyStateLimit` – definuje ustálený stav monitoru a ovlivňuje trvání simulace. Je to hodnota přesnosti frekvence signálu, které jsou vypočítány fourierovou transformací časového signálu.
`CMeshAdaption` – jestli-že MeshAdaption je povoleno a rovná se „true“, pak proběhne několik simulací k nalezení optimálního namešování.
`CAutoNormImpedance` – s-parametry jsou vždy normovány k referenci impedance.
`CNormingImpedance` – reference impedance, která bude použita.
`CCalculateModesOnly` – jestli-že *flag* je „true“, pak se při analýze počítají pouze módy portu.
`CSParaSymmetry` – použití s-parametrové symetrie.

`CStoreTDResultsInCache` – jestli-že *flag* je nastaven na „true“ uloží metoda výsledky časové analýzy do paměti.
`CFullDeembedding` – typ zdroje je dán automaticky ke všem portům/ módům, jestli-že je nastaven „true“.
`CUseDistributedComputing` – jestli-že je nastaven „true“, pak metoda umožňuje distribuovaný výpočet přes sít.

Příklad použití funkce:

```
fid = fopen('pokus.m', 'wt');
 ...
CstTimeDomainSolverEasy(fid,'TD-S','All','All',-30,'False','False',
50,'False','False','False','False','False');
```

Funkce HFSS:

General

HfssExitApplication

Funkce `HfssExitApplication` slouží k ukončení programu HFSS. Funkce nemá parametry a ani nevrací žádnou hodnotu.

Poznámka:

Funkci je lze použít v případě, že je program HFSS jíž otevřen.

Příklad použití funkce :

```
HfssExitApplication;
```

HfssInsertDesign

Funkce `HfssInsertDesign(fid,HdesignName,HdesignType)` slouží k vytvoření nového projektu v programu HFSS. Funkce nevrací žádnou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HdesignName` - jméno nového vloženého designu.
`HdesignType` - výběr z následujícího seznamu:
1. driven modal (vlastni)
2. driven terminal
3. eigenmode

, nebude-li proměnná typu `HdesignType` zadána bude automaticky nastavena na „driven modal“.

Poznámka :

Tato funkce lze použít nejdříve po funkci `HfssNewProject()`.

Příklad použití funkce :

```
fid = fopen('pokus.m','wt');
...
HfssInsertDesign(fid,'Muj_Design','driven modal');
```

HfssNewProject

Funkce `HfssNewProject(fid)` slouží k otevření nového pracovního okna programu HFSS. Vytvoří se aktivní spojeni pomocí Active X s programem HFSS. Funkce nevrací žádnou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.

Příklad použití funkce:

```
fid = fopen('pokus.m','wt');
...
HfssNewProject(fid);
```

HfssSaveAsProject

Funkce `HfssSaveProject(fid, HprojectFile, HOverwrite)` slouží k uložení aktivního vytvořeného projektu v programu HFSS na zvolené místo na disku a se zvoleným jménem souboru.

Parametry funkce :

`fid` - soubor identifikující historii HFSS skriptu.
`HprojectFile` – celé jméno aktivního projektu, pod kterým bude soubor uložen.
`HOverwriteH` – jestli-že bude nastaven na „true“ bude existující soubor se shodným jménem přepsán. V případě, že je proměnná typu `HOverwriteH` prázdná bude nastavena na hodnotu „true“.

Příklad použití funkce:

```
fid = fopen('pokus.m', 'wt');
 ...
HfssSaveAsProject(fid,'C:/Temp/tmpData.m');
```

3D modeler

HfssAssignMaterial

Funkce `HfssAssignMaterial(fid,'FR4Mount','FR4epoxy')` u zadaného objektu nastaví požadovaný materiál. Požadovaný materiál musí být jíž implementován v programu HFSS

nebo lze funkci použít po přidání vlastního materiálu, k čemuž sloučí funkce `HfssAddMaterial(...)`. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HObject` - jméno nastavovaného objektu.
`HMaterial` - materiál nastavovaného objektu, musí být shodný s programem HFSS.

Příklad použití funkce :

```
fid = fopen('antenna.m', 'wt');
 ...
hfssAssignMaterial(fid, 'FR4Mount', 'FR4epoxy');
```

HfssBox

Funkce `HfssBox(fid,HName,HMaterial,HStart,HSize,HUnits)` slouží k vytvoření kvádru. Funkce nemá návratovou hodnotu.

Parametry funkce :

`fid` - soubor identifikující historii HFSS skriptu.
`HName` - jméno krychle .
`HStart` - startovní pozice krychle, specifikované jako [x, y, z].
`HSize` - velikosti stran krychle, specifikované jako [Δx, Δy, Δz].
`HUnits` - použité jednotky jako (např: 'mm', 'meter', a ostatní používané v HFSS).
`HMaterial` – požadovaný materiál krychle

Příklad použití funkce :

```
fid = fopen('antenna.m', 'wt');
...
HfssBox(fid,'zarez2','vacuum',[18.35,0,0],[1,6,0.035],'mm');
```

HfssCircle

Funkce `HfssCircle(fid,HName,HAxis,HCenter,HRadius,HUnits)` slouží k vytvoření kruhu v programu HFSS. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HName` - pojmenování kruhu přímo (v HFSS).
`HAxis` - výběr mezi 'X', 'Y', nebo 'Z' reprezentující osu kruhu.
`HCenter` - střed kruhu (použitý [x, y, z] formát).
`HRadius` - poloměr kruhu.
`HUnits` - použité jednotky (jako např. 'in', 'mm', 'meter' nebo jednotky definované (v HFSS).

Příklad použití funkce :

```
fid = fopen('antenna.m', 'wt');
 ...
HfssCircle(fid, 'C_Patch', 'Z', [10, 11, 12], 13, 'mm');
```

HfssCylinder

Funkce `HfssCylinder(fid,HName,HAxis,HCenter,HRadius,HHeight,HUnits)` slouží k vytvoření válce v HFSS. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HName` - jméno válce (v HFSS).
`HCenter` - umístění středu válce (specifikované jako [x, y, z]).
`HAxis` - osa otočení válce (specifikované jako 'X', 'Y', nebo 'Z').
`HRadius` - poloměr válce.
`HHeight` - výška válce (od bodu specifikovaného jako `HCenter`).
`HUnits` - použité jako 'in', 'mm', 'meter' nebo definované v HFSS.

Příklad použití funkce:

```
fid = fopen('projekt.m', 'wt');
...
hfssCylinder(fid,'Cyl1','Z',[0, 0, 0],0.1,10,'mm');
```

HfssGetFaceByPosition

Funkce `HfssGetFaceByPosition(fid,HObjectName,HPosition,HUnits)` je potřebná ke zjištění čísla stran(face) vybraného objektu. Funkce je potřebná při vytváření okrajových podmínek. Funkce vrací číslo strany.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HObjectName` - objekt u kterého jsou zjišťovány čísla stran.
`HPosition` - bod ze kterého se zjišťuje, o jakou stranu jde, srovnáno [x,y,z].
`HUnits` - jednotky které jsou standardem v HFSS.
`FaceId` - Vracena hodnota čísla strany.

Příklad použití funkce :

```
fid = fopen('antenna.m', 'wt');
...
HfssGetFaceByPosition(fid,'airvol',[0,0,0],'mm');
```

HfssRectangle

Funkce `HfssRectangle(fid,HName,HAxis,HStart,HWidth,HHeight,HUnits)` je potřeba k vytvoření čtverce v HFSS. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HName` - jméno čtverce (v HFSS).
`HAxis` - osa natočení čtverce.
`HStart` - pozice startu čtverce (spodní roh čtverce). Specifikované jako[sx, sy, sz].
`HWidth` - sirka čtverce. Jestli ze je osa 'X', pak to reprezentuje velikost osy Y ve čtverci.
`HHeight` - výška čtverce. Jestli ze je osa 'X', pak to reprezentuje velikost osy Z ve čtverci.
`HUnits` - specifikováno jako 'in', 'meter', 'mm', ... nebo jiné definované jednotky v HFSS.

Příklad použití funkce :

```
fid = fopen('antenna.m', 'wt');
...
HfssRectangle(fid, 'Rect1', 'X', [0,0,0], 10, 20, 'in');
```

HfssRename

Funkce `HfssRename(fid, HoldName, HnewName)` slouží k přejmenování objektu v HFSS. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HoldName` - staré jméno objektu.
`HnewName` - nové jméno objektu.

Příklad použití funkce :

```
fid = fopen('antenna.m', 'wt');
 ...
HfssRename(fid, 'dipole1', 'dipole_one');
```

HfssSetColor

Funkce `HfssSetColor(fid,HObject,HColor)` slouží k nastavení barev u vybraného objektu v HFSS. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HObject` - jméno objektu pro nastavení barvy.
`HColor` - [3x1 vektor] prezentující [R, G, B] komponenty barev.

Priklad pouziti funkce :

```
fid = fopen('antenna.m', 'wt');
...
HfssSetColor(fid, 'Substrate', [0, 64, 0]);
```

HfssSetTransparency

Funkce `HfssSetTransparency(fid,HObjectList,HValue)` slouží k nastavení průhlednosti vybraného objektu. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HObjectList` – jméno objektu.
`HValue` – žádaná hodnota průhlednosti (může byt od 0 do 1).

Příklad použití funkce:

```
fid = fopen('antenna.m', 'wt');
...
HfssSetTransparency(fid, 'AirBox', 0.95);
```

HfssSphere

Funkce `HfssSphere(fid,HName,HCenter,HRadius,HUnits)` potřebná k vytvoření koule v programu HFSS. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HName` - jméno koule (použité v HFSS).
`HCenter` - střed koule (specifikované jako [cx, cy, cz]).
`HRadius` - poloměr koule.
`HUnits` - jednotky jako 'in', 'm', 'meter' nebo ostatní definované v HFSS.

Příklad použiti funkce :

```
fid = fopen('antenna.m', 'wt');
...
HfssSphere(fid, 'Sphere1', [0,0,0], 1, 'mm');
```

HfssSubtract

Funkce `HfssSubtract(fid,HblankParts,HtoolParts)` je potřebná funkce k vyříznutí jednoho objektu od druhého. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HblankParts` – objekt, který bude vyříznut.
`HtoolParts` - objekt ze kterého bude vyříznut objekt `HblankParts`.

Příklad použití funkce :

```
fid = fopen('antenna.m', 'wt');
 ...
```

```
HfssSubtract(fid,'Antena','zarez2');
```

Boundary

HfssAssignLumpedPort

Funkce `HfssAssignLumpedPort(fid, HName, HObjectName, HStart, HEnd, HUnits, HResistance, HReactance)` potřebná k vytvoření portu (Lumped Port) na vybraném objektu. Jestli-že nebudou zadány parametry `HResistance, HReactance` budou automaticky přednastaveny.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HPortName` - jméno vlnového portu.
`HObjectName` - jméno objektu na kterém bude umístěn port.
`HStart` - vektor umístění startovního místa portu [x,y,z]
`HEnd` - vektor umístění koncového místa portu [x,y,z]
`HUnits` - specifikované jednotky 'in', 'meter', 'mm' nebo další definované jednotky v HFSS.
`HResistance` - skalární hodnota odporu, je přednastavena na 50 Ohmu
`HReactance` -skalární hodnota reaktance, je přednastavena na 0 Ohmu

Poznámka :

1. Port (LumpedPort) je dobré například umístit na objekt (kruh,krychle).

Příklad použiti funkce :

```
fid = fopen('antenna.m', 'wt');
 ...
HfssAssignLumpedPort(fid,'WavePort','WPort',[10,20,-8],…
[2,3,-8],'mm');
```

HfssAssignPE

Funkce `HfssAssignPE(fid, HName, HObjectList, HinfGND)` potřebná k nastavení okrajových podmínek. Funkce nastaví vybraný objekt jako perfektní elektrický vodič. Nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HName` - jméno objektu , který bude nastaven jako PEC.
`HObjList` - pojmenování v seznamu okrajových podmínek
`HinfGND` - jestli-že se jedna o zemnici plochu (infinite ground plane)
pak true jinak je přednastaveno false.

Příklad použití funkce :

```
fid = fopen('antenna.m', 'wt');
```

```
...
HfssAssignPE(fid, 'GNDplane', 'AntennaGND', true);
```

HfssAssignRadiationFaces

Funkce `HfssAssignRadiationFaces(fid, HName, HObject)` potřebná k označení okrajové podmínky vyzařování (Radiation) podle čísla stany objektu. Nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HName` - pojmenování v seznamu okrajových podmínek.
`HFaces` - plocha u které jsou nastaveny okrajové podmínky.

Příklad použití funkce :

```
fid = fopen('antenna.m', 'wt');
...
HfssAssignRadiationFaces(fid, 'AirVolRad1', 183);
```

HfssAssignRadiationObject

Funkce `HfssAssignRadiationObject(fid, HName, HObject)` je potřebná k označení okrajové podmínky vyzařování (Radiation) vybraného objektu. Nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HName` - pojmenování v seznamu okrajových podmínek.
`HObject` - objekt u kterého jsou nastaveny okrajové podmínky.

Příklad použiti funkce :

```
fid = fopen('antenna.m', 'wt');
...
HfssAssignRadiationObject(fid, 'AirvolRad', 'airvol');
```

HfssAssignWavePort

Funkce `HfssAssignWavePort(fid, HPortName, HObject, HStart, HEnd, HUnits)` je potřebná k vytvoření vlnového portu. Funkce nemá návratovou hodnotu.

Parametry funkce:
`fid` - soubor identifikující historii HFSS skriptu.
`HPortName` - jméno portu.
`HObject` - jméno objektu na kterém bude umístěn port.
`HStart` - vektor umístěni startovního místa portu [x,y,z]
`HEnd` - vektor umístěni koncového místa portu [x,y,z]
`HUnits` - jednotky 'in', 'meter', 'mm' nebo další definované jednotky v HFSS.

Poznámka :

Port je vhodné například umístit na objekt (kruh).

Příklad použití funkce :

```
fid = fopen('antenna.m', 'wt');
 ...
HfssAssignWavePort(fid,'WavePort','WPort',[10,11,-8],[1,2,-8],'mm');
```

Analysis

HfssExportNetworkData

Funkce `HfssExportNetworkData(fid, HfileName, HsetupName, HsweepName, HexpFileType, HrenormZ)` slouží k vyexportování výsledků z programu HFSS do předem definovaného souboru. Exportované parametry jsou činitel odrazu, impedance a frekvence. Nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.

`HfileName` - výstupní datový soubor (do kterého budou uloženy výsledky).
`HsetupName` - jméno analýzy, která bude vyexportována.
`HsweepName` - jméno frekvenčního rozsahu krokováni, která bude vyexportována.
`HexpFileType` - výstupní datový formát (přednastaven je m-file).
Definované jako:
'h' - HFSS 8.x formát (.szg)
't' - formát (.tab)
's' - formát (.sNp)
'c' - formát (.cit)
'm' - MATLAB soubor (.m) - přednastaveno.
'z' - terminal Z0.
`HrenormZ` - normalizační impedance (jestli-že je odlišná od 50 Ohmů). Pak bude přednastavena na hodnotu 50 ohmů.

Příklad použiti funkce :

```
fid = fopen('antenna.m', 'wt');
...
HfssExportNetworkData(fid, 'C:\temp\tempData.m', 'Setup150MHz', ...
                'Sweep100to200MHz', 'm', 75);
```

HfssFastSweep

Funkce `HfssFastSweep(fid,HName,HSolutionName,HfStartGHz,HfStopGHz,HnPoints)` je potřeba k vytvoření rozsahu krokování v analýze.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HName` - jméno rozsahu krokování.
`HSolutionName` – jméno řešení, ke kterému bude rozsah krokování přidán.
`HfStartGHz` - startovní frekvence rozsahu v GHz.
`HfStopGHz` - cílová frekvence rozsahu v GHz.
`HnPoints` - počet kroků v rozsahu (přednastaveno je 1000).

Příklad použiti funkce :

```
fid = fopen('antenna.m', 'wt');
...
HfssFastSweep(fid,'Interp600to900MHz','Solve750MHz',0.6,0.9,1000);
```

HfssInsertSolution

Funkce `HfssInsertSolution(fid, HName, HfGHz, HmaxDeltaS, HmaxPass)` je potřeba k vytvoření nastavení analýzy v HFSS. Funkce nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HName` - jméno nové analýzy.
`HfGHz` - použitá frekvence (v GHz).
`HmaxError` - maximum chyb, (může byt od 0 a 1, přednastaveno je 0.02).
`HmaxPasses` – maximum přizpůsobení před simulací, je deklarováno jako nekonvergovat a je přednastaveno 25.

Příklad použiti funkce :

```
fid = fopen('antenna.m', 'wt');
 ...
HfssInsertSolution(fid, 'Setup150MHz', 0.15, 0.02, 25);
```

HfssInterpolatingSweep

Funkce `HfssInterpolatingSweep(fid, HName, HSolutionName, HfStart, HfStop, HnPoints, HnMaxSols, HiTol )` potřebná k vytvoření interpolačnímu rozsahu krokování v analýze.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HName` - jméno interpolačního rozsahu.
`HSolutionName` – jméno analýzy, jako interpolačního rozsahu krokování, který bude přidán.
`HfStart` - startovní frekvence rozsahu krokování v GHz.
`HfStop` - cílová frekvence rozsahu krokování v GHz.
`HnPoints` - počet kroků v rozsahu (přednastaveno je 1000).
`HnMaxSols` - maximum interpolačních bodů, které jsou potřeba k analyzování (více bodů bude záruka konvergence) – přednastaveno je 101.
`HiTol` - interpolační tolerance (přednastavená na 0.5).

Příklad použití funkce :

```
fid = fopen('antenna.m', 'wt');
...
HfssInterpolatingSweep(fid, 'Interp600to900MHz', 'Solve750MHz', ...
 0.6, 0.9, 1000, 101, 0.5);
```

HfssRenameSetup

Funkce `HfssRenameSetup(fid,HoldName,HNewName)` je vhodná pro přejmenovaní analýzy v HFSS. Nemá návratovou hodnotu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HoldName` - staré jméno objektu.
`HnewName` - nové jméno objektu.

Příklad použití funkce :

```
fid = fopen('antenna.m', 'wt');
 ...
HfssRenameSetup(fid, 'setup1', 'setup_one');
```

HfssSolveSetup

Funkce `HfssSolveSetup(fid, HSetupName)` pozastavení všech potřebných objektů a parametrů spouští analýzu.

Parametry funkce:

`fid` - soubor identifikující historii HFSS skriptu.
`HSetupName` - jméno vytvořené analýzy, kterou funkce spustí.

Příklad použiti funkce :

```
fid = fopen('antenna.m', 'wt');
 ...
HfssSolveSetup(fid, 'Setup750MHz');
```

Other

GetFrequencyFromDiagram

Funkce `SF = GetFrequencyFromDiagram(HUroven,HFile)` je potřebná ke získání parametru (S11 a frekvence) z výsledného grafu, který byl vyexportován z HFSS funkcí `HfssExportNetworkData`. Funkce má návratovou hodnotu složenou z údajů o frekvenci, o činiteli odrazu míře decibelové, tak i v míře bezrozměrné a o počtu vzorků zabírající každý lalok vypočtené frekvence.

Parametry funkce:

`HUroven` – rozhodovací úroveň od 1 do 0 (hodnota 1 už je bez rozlišení!)
`HFile` - Exportovaný soubor 'C:/Temp/tempDataExportHFSS.m'

Příklad použití funkce:

```
      fid = fopen('pokus.m', 'wt');
      ...
SF = GetFrequencyFromDiagram(0.5,'C:/Temp/tempDataExportHFSS.m');
```