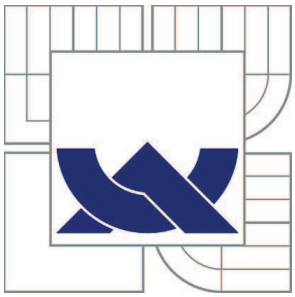
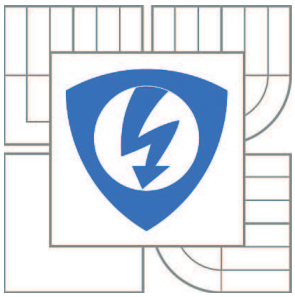


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

VÍCEKRITERIÁLNÍ NÁVRH POKRYTÍ ÚZEMÍ RÁDIOVÝM SIGNÁLEM

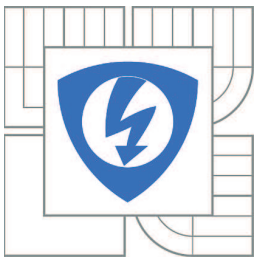
RADIO NETWORK MULTIOBJECTIVE DESIGN

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. PETR VÍTEČEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR KADLEC, Ph.D.
SUPERVISOR

BRNO 2014



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 Víteček	***ID:***	125699
Ročník:	2	***Akademický rok:***	2013/2014

NÁZEV TÉMATU:

Vícekriteriální návrh pokrytí území rádiovým signálem

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se se základními principy optimalizačních metod pro hledání tzv. Paretova čela u více-kriteriálních problémů. Implementujte vhodnou metodu vícekriteríální optimalizace.

Vytvořte model šíření EM vln, který bude schopen nad digitální mapou zvoleného území určit úroveň přijímaného signálu. Vytvořte program, který bude schopen pomocí vícekriteriální optimalizace navrhovat umístění a výkony jednotlivých vysílačů dle zadaných požadavků.

DOPORUČENÁ LITERATURA:

[1] DEB, K. Multi-Objective Optimization using Evolutionary Algorithms. Chichester: Wiley, 2001.

[2] NOVÁČEK, Z. Elektromagnetické vlny, antény a vedení. Elektronické skriptum. Brno: FEKT VUT v Brně, 2006.

Termín zadání:	10.2.2014	***Termín odevzdání:***	23.5.2014

Vedoucí práce: Ing. Petr Kadlec, Ph.D.
Konzultanti diplomové práce:

doc. Ing. Tomáš Kratochvíl, Ph.D.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Tato práce se zabývá návrhem pokrytí území radiovým signálem. Dané území je reprezentováno digitálním mapovým souborem, který vznikl v rámci projektu DEM (Digital Elevation Model). Základem je výpočet vzdáleností jednotlivých bodů ve vybraném mapovém podkladu. Následně je s využitím optimalizačního algoritmu vypočítána pozice vysílače v mapě, nastavení radiového systému a výsledné pokrytí, jež je reprezentováno intenzitou elektrického pole, popřípadě přijímaným výkonem v celé mapě. Optimalizační algoritmus v dané práci slouží k tomu, abychom nalezli nejlepší možné řešení z hlediska zadaných parametrů (např. výkon vysílače, výška vysílače) a výsledného pokrytí území.

KLÍČOVÁ SLOVA

Digitální elevační model, Digitální mapy, Matlab, Návrh radiového pokrytí, Vícekriteriální optimalizace, Optimalizace rojem částic

ABSTRACT

This thesis deals with radio network design for a chosen part of a map. Here map is represented by digital map file, which was created within the project DEM. First step is to calculate distances between points in chosen map. With help of optimization algorithms, appropriate position of transceiver in the map and parameters of radio systems are determined, also final coverage by radio signal, represented by intensity of electric field or received power in whole map. The optimization algorithm is used to find the best solution in terms of input parameters (e.g. power of transmitter, height of mast) and resulting coverage of land by radio signal.

KEYWORDS

Digital Elevation Model, Digital Maps, Matlab, Multiobjective Optimization, Radio Network Design, Particle swarm optimization

VÍTEČEK, P. Vícekriteriální návrh pokrytí území rádiovým signálem.
Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Ústav radioelektroniky, 2013. 52 s., 21 s. příloh. Diplomová práce. Vedoucí práce: Ing. Petr Kadlec Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma Vícekriteriální návrh pokrytí území rádiovým signálem 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/nebo majetkových a~jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Petru Kadlecovi Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

1 ÚVOD

V dané práci se budeme zabývat návrhem pokrytí území radiovým signálem. Úkolem je vytvořit program v prostředí Matlab, který bude schopen nad určitým územím vypočítat intenzitu elektrického pole či přijímaný výkon. Dané území bude reprezentováno pomocí *.hgt mapových soborů. S využitím optimalizačních procesů se dle zadaných parametrů nalezne nejvýhodnější konfigurace pozice a parametrů vysílače, pro dosažení co nejlepšího pokrytí území nad vybranou oblastí.

Druhá kapitola se zabývá popisem digitálního elevačního modelu, různých typů digitálních map a nakonec popisem SRTM(Shuttle Radar Topografy mission) map a jejich vytvářením.

Ve třetí kapitole se budeme věnovat šíření elektromagnetických vln, převážně jejich obecnému popisu pomocí Maxwellových rovnic. Následně šíření ve volném prostoru a nakonec jevům způsobujícím útlum vlnění, v našem případě převážně difrakci. V dané kapitole se také seznámíme s metodami výpočtu útlumu, k němuž dochází při difrakci vlny na překážce.

Čtvrtá kapitola se zabývá popisem optimalizačních algoritmů. Od popisu samotné teorie optimalizace, následované popisem námi zvolené metody a jejími úpravami a modifikacemi.

Poslední kapitola se zabývá samotným modelem. Od načítání map, přes výpočet doplňujících proměnných, až po výpočet intenzity elektrického pole, či jiných výsledných parametrů. Také zahrnuje seznámení se zjednodušeným kódem tak, abychom dosáhli co nejmenší výpočetní náročnosti. Nakonec je v ní probrán návrh optimalizačního algoritmu naší úlohy, její naprogramování a odzkoušení na vybraném mapovém podkladu či jeho části.

2 DIGITÁLNÍ MODEL TERÉNU

Digitální modely terénu (DTM) jsou souvislé plochy, které kromě hodnot nadmořských výšek (známé jako digitální výškový model – DEM) obsahují také další prvky, které popisují topografický povrch. Od svého vzniku, přibližně před padesáti lety, byly vyvinuty různé techniky pro tvorbu DTM. Až do konce 90. let byly vysoce kvalitní údaje získávány převážně pomocí leteckých snímků a ručního měření, nebo vektorizací vrstevnic z topografických map [1].

Kvalita DTM se výrazně zvýšila v posledním desetiletí v důsledku tří významných faktorů. První bylo zavedení nových metod pro sběr dat, zejména z družic a letadel. U malých měřítek (hrubší prostorové rozlišení) je použito „radar interferometric" techniky (ifSAR) k vytvoření globálního DTM. Pro větší měřítka je použito leteckého laserového skenování (A.L.S.) [1].

Druhým faktorem je zvyšující se dostupnost dalších zdrojů dat, které jsou užitečné pro posouzení či zdokonalení kvality DTM. Kromě leteckých snímků a vrstevnic lze použít další data obsahující výškové hodnoty, jako jsou základní body geodetické sítě, mezní body katastrálních map, databáze budov a další související data. Za třetí, aplikace využívající DTM jsou nyní součástí každodenního života, např. Google Earth3, NASA World Wind 5 [1].

Čím vyšší je rozlišení, tím obtížnější je hodnocení vstupních dat a sestavení DTM. DTM s vysokým rozlišením jsou tak více náchylné k chybám. Vizuální metody jsou velmi důležité pro vyhodnocení prostorových dat a mohou vyvážit některé nedostatky statistických metod [1].

V našem případě použijeme mapy formátu *.hgt. O těchto mapách a procesu jejich vytváření si povíme více v následující podkapitole.

2.1 SRTM

Shuttle radar topografy mission(SRTM) je společný projekt National Imagery and Mapping Agency (NIMA) a National Aeronautics and Space Administration (NASA). Cílem projektu je vytvořit digitální topografické údaje pro 80% zemského povrchu, zmapování pevniny mezi 60° severní a 57° jižní šířky, s datovými body umístěnými každou úhlovou vteřinu (cca 30 metrů) na zeměpisnou šířku. Absolutní vertikální přesnost bude 16 metrů, při 90% spolehlivosti. Tento radarový systém nasbíral data, která posloužila k vytvoření nejpřesnější a nejkompletnější topografické mapy zemského povrchu, která byla kdy vytvořena [2].

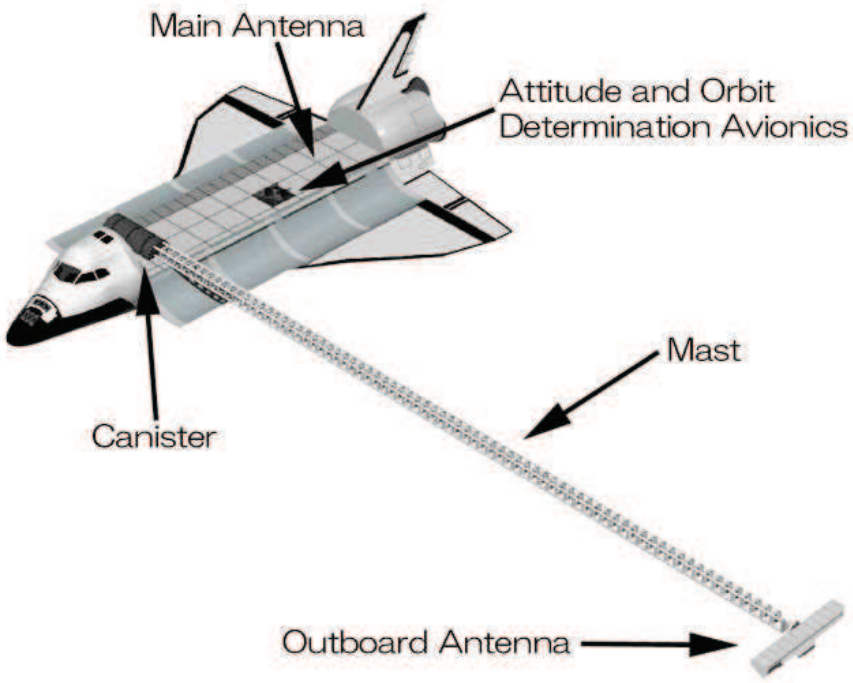
Tyto údaje mohou být přizpůsobeny, aby vyhovovaly potřebám armády, civilní a vědecké skupině uživatelů. Existují ale i jiná využití těchto dat, mezi něž patří např. realističtější letové simulátory, lepší umístění vysílačů signálu a zvýšení navigační bezpečnosti. Jakýkoliv projekt, který vyžaduje přesné znalosti o tvaru a výšce země může těžit z těchto údajů. Některými příklady jsou ochrana před povodněmi, zalesňování, monitorování sopek, výzkum zemětřesení, sledování pohybu ledovců [2].

SRTM data byla získána pomocí radarové interferometrie. Daná metoda je popsána v následující podkapitole.

2.1.1 Interferometrie

Interferometrie je velmi přesná metoda měření, v našem případě založená na interferenci radiových vln. Nutno podotknout, že tato metoda se geometricky odlišuje od metod optických, dle pozorovacího úhlu. Základem je vyslání elektromagnetické vlny o určité polarizaci směrem k zemi a její příjem ve stejné či jiné polarizační rovině [6].

SRTM byla „fixed-baseline“ interferometrickou misí. To znamená, že ve stejnou dobu byly shromážděny dvě sady radarových dat a antény sloužící ke sběru dat byly odděleny danou pevnou vzdáleností. Hlavní anténa, umístěná v nákladním prostoru raketoplánu „osvětluje“ danou část země. Když tyto vlny narazí do zemského povrchu, jsou rozptýleny do různých směrů. Tyto rozptýlené vlny jsou pak přijímány dvěma anténami, hlavní a vedlejší [3].



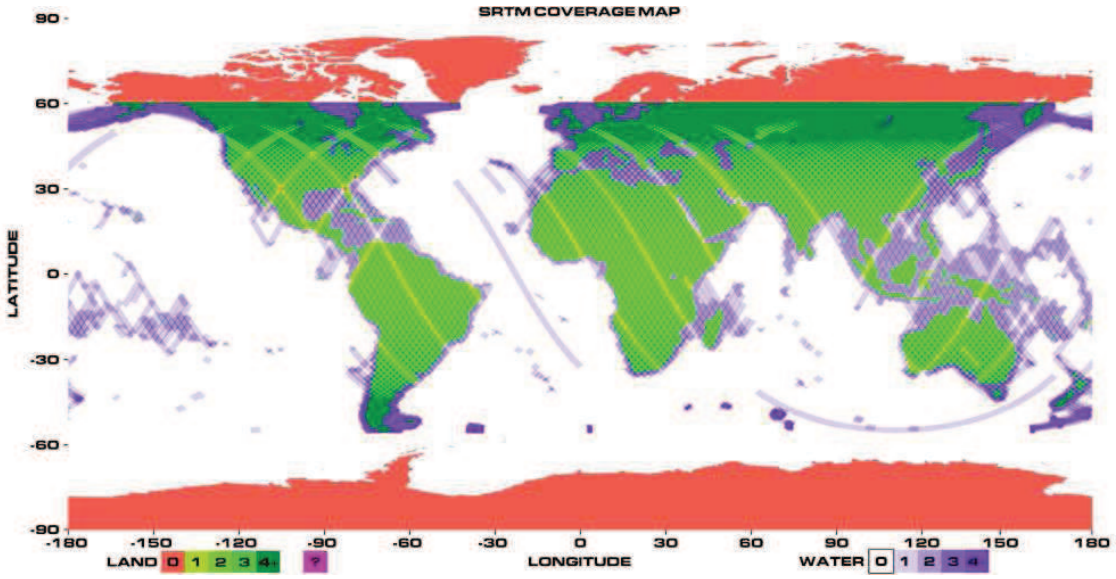
Obr. 2.1 Rozmístění měřících antén, při SRTM misi (převzato z [5])

Výchozí vzdálenost mezi hlavní a vedlejší anténou byla známa velmi přesně a zůstala konstantní. Během měření dat se měnila jen vzdálenost k zemskému povrchu, ve vztahu k oběma anténám. S odraženým radarovým paprskem byl bod, kde došlo k odrazu mírně odlišný pro hlavní a vedlejší anténu [3].

Pomocí informací o vzdálenosti mezi dvěma anténami a rozdíly v odražených radarových signálech mohla být vypočtena přesná nadmořská výška [3].

2.1.2 Výsledná mapa pokrytí

SRTM shromáždila údaje pro většinu zemského povrchu, který leží mezi 60° severní a 57° jižní šířky. To je asi 80% veškeré půdy na Zemi. V následující mapě barvy jednotlivých bodů indikují, kolikrát byla daná oblast zmapována pomocí SRTM. Pro zem, žluto-zelená barva znamená jedno měření, zelená dvě atd., jak je uvedeno v legendě v levém dolním rohu. Nad vodou je použit barevný kód v odstínech modré. Oblasti označené červenou barvou nemohly být zmapovány. SRTM byla topografická mise, takže data byla získána především na zemi. Malé množství dat bylo shromážděno nad vodou pro kalibrační účely [4].



Obr. 2.2 Zmapované území při SRTM (převzato z [4]).

Výsledná SRTM data lze nalézt na internetových stránkách uvedených v Tab. 2.1.

Tab. 2.1 Internetové stránky s SRTM daty ke stažení

Verze SRTM	URL
4	http://srtm.csi.cgiar.org/SELECTION/inputCoord.asp
2	http://dds.cr.usgs.gov/srtm/version2_1/
1 - 4	http://www.viewfinderpanoramas.org/dem3.html

3 ŠÍŘENÍ EM VLN

Elektromagnetické vlny jsou nositelem energie elektromagnetického pole. Na rozdíl od mechanických vln se mohou šířit i vakuem. Právě rychlost elektromagnetických vln ve vakuu je podle Einsteinovy teorie relativity horní hranicí, kterou látková tělesa nemohou překročit. Příkladem elektromagnetických vln je světlo, tepelné nebo ultrafialové záření, rozhlasové a televizní vlny, rentgenové záření a další [7].

Elektromagnetická vlna se skládá z elektrické a magnetické složky, které jsou na sebe navzájem kolmé a zároveň kolmé na směr šíření vlny. Elektromagnetické vlny lze popsat pomocí Maxwellových rovnic.

3.1 Maxwellovy rovnice

Maxwellovy rovnice vyjadřují souhrnně zákony elektromagnetického pole, tedy vzájemné obecné souvislosti mezi veličinami popisují toto pole v každém místě prostoru. Maxwellovy rovnice v integrálním tvaru je možno vyjádřit následující soustavou rovnic (3.1) - (3.4) [8]. První rovnice je formulována jako

$$\oint_l H \cdot dl = I_{zdroj} + I_{ind} + \frac{d\psi}{dt} \tag{3.1}$$

kde H je intenzita magnetického pole, dl je libovolně orientovaná uzavřená křivka, I_{zdroj} je proud vyvolaný zdrojem vlnění, I_{ind} je proud indukovaný elektrickým polem, $d\psi$ je tok elektrického pole určitou plochou. Daná rovnice říká, že změna elektrického toku a dané proudy vytvářejí magnetické pole. Další rovnice je formulována následovně

$$\oint_l E \cdot dl = -\frac{d\Phi}{dt} \tag{3.2}$$

kde E je intenzita elektrického pole, dl je libovolně orientovaná uzavřená křivka, Φ je magnetický indukční tok. Daná rovnice říká, že změna magnetického toku vytváří elektrické pole. Další rovnice je formulována následovně

$$\oint_S D \cdot dS = Q \tag{3.3}$$

kde D je elektrická indukce, S je libovolně orientovaná plocha, Q je elektrický náboj. Daná rovnice říká, že elektrický tok uzavřenou plochou, je roven náboji uzavřeném touto plochou. Poslední rovnice je formulována jako

$$\oint_S B \cdot dS = 0 \tag{3.4}$$

kde B je magnetická indukce, Sje libovolně orientovaná plocha. Daná rovnice říká, že magnetický tok uzavřenou plochou je nulový.

Vektory intenzit polí E, Ha indukcí D, B jsou vzájemně svázány materiálovými vztahy. V lineárním izotropním prostředí, kde parametry prostředí nezávisí na velikosti veličin elektromagnetického pole a jsou stejné ve všech směrech, platí [8].

$$D = \varepsilon \cdot E = \varepsilon_0 \cdot \varepsilon_r \cdot E \tag{3.5}$$

kde D je elektrická indukce,ε je permitivita prostředí, ε_r je relativní permitivita, ε_0 je permitivita vakua, E je intenzita elektrického pole.

$$B = \mu \cdot H = \mu_0 \cdot \mu_r \cdot H \tag{3.6}$$

kde B je magnetická indukce, μ je permeabilita prostředí, μ_r je relativní permeabilita, μ_o je permeabilita vakua, H je intenzita magnetického pole.

Konstantami úměrnosti jsou zde permitivita prostředí ε a jeho permeabilita μ. Ve vakuu mají tyto veličiny hodnoty $\varepsilon_0 = 10^{-9}/36$ (F/m) a $\mu_0 = 4\pi .10^{-7}$ (H/m). Relativní permitivita ε_r a relativní permeabilita μ_r jsou bezrozměrné veličiny, udávající kolikrát je permitivita či permeabilita daného prostředí větší než ve vakuu [8].

Elektrické pole působící ve vodivém prostředí vyvolává v tomto prostředí vodivý proud. Vektor plošné hustoty vodivého proudu J (jednotkou je A/m^2) je přímo úměrný vektoru intenzity elektrického pole E**,** dle vztahu

$$J = \gamma \cdot E \tag{3.7}$$

konstantou úměrnosti je zde měrná vodivost prostředí γ(jednotkou je S/m).[8]

Popis elektromagnetického pole integrálními rovnicemi (3.1)- (3.4) má obecnou platnost. Jejich řešení, nutné pro určení prostorového nebo časového rozložení intenzit polí nebo indukcí, je však velmi obtížné a v řadě situací zvládnutelné jen numerickými metodami. Maxwellovy rovnice (3.1) až (3.4) je možno převést na soustavu diferenciálních rovnic

$$rot\ H = J_{zdroj} + J_{ind} + \frac{\partial D}{\partial t} \tag{3.8}$$

kde *rot H* je rotace vektoru magnetického pole, J_{zdroj} proudová hustota zdroje, J_{ind} proudová hustota indukovaného pole, $\partial D/\partial t$ je hustota posuvného proudu.

$$rot\ E = -\frac{\partial B}{\partial t} \tag{3.9}$$

kde *rot H* je rotace vektoru elektrického pole, $\partial B/\partial t$ *je* derivace magnetické indukce.

$$div\ D = \rho \tag{3.10}$$

kde *div D* je divergence vektoru elektrické indukce, ρ je objemová hustota náboje.

$$div\ B = 0 \tag{3.11}$$

kde *div B* je divergence vektoru magnetické indukce.

Soustava diferenciálních rovnic (3.8) - (3.11)je snadněji řešitelná, ale popisuje jevy jen v oblastech, kde jsou vektory *E*, *H*, *D* a *B* spojité a diferencovatelné [8].

3.2 Šíření EM vln volným prostorem

Při šíření EM vln volným prostorem je jediným elementem způsobujícím útlum rostoucí vzdálenost od zdroje. Jelikož se nezahrnují do tohoto typu šíření další elementy, lze vypočítat výslednou intenzitu EM pole v místě příjmu jako

$$E = \frac{\sqrt{30 P_v G_v}}{d} \tag{3.12}$$

kde *E* je hodnota intenzity elektrického pole v bodě pozorování, P_v výkon na vstupu vysílací antény, G_v zisk vysílací antény v příslušném směru a *d* je vzdálenost vysílací antény a bodu příjmu [9].

Při šíření reálným prostorem bychom museli celou rovnici 3.12 vynásobit činitelem útlumu vůči volnému prostoru.

Namísto intenzity pole, v místě přijmu, můžeme vypočítat přijímaný výkon dle následujícího vzorce

$$P_{příj} = P_{vys} + G_{vys} + G_{příj} - FSL \tag{3.13}$$

kde $P_{příj}$ je přijímaný výkon v dBm, P_{vys} je výkon vysílače v dBm, G_{vys} je zisk vysílací antény v dBm, $G_{příj}$ je zisk přijímací antény v dBm a *FSL* je útlum volným prostorem.

Pokud známe intenzitu EM pole v místě příjmu a zároveň je v místě příjmu umístěna přijímací anténa o známém zisku G_p, potom lze také vypočítat výkon v místě příjmu dle následujícího vzorce

$$P_p = \frac{{E_{ef}}^2}{120\pi}\frac{G_p \lambda^2}{4\pi} = P_v G_v G_p \left(\frac{\lambda}{4\pi d}\right)^2 \tag{3.14}$$

Hodnotu intenzity EM pole lze také vyjádřit jako výkonovou úroveň přijatou přijímací anténou dle následujících vzorců

$$E_{dBu} = 20 \log \frac{E_{V/m}}{10^{-6}} \tag{3.15}$$

$$P_{dBm} = E_{dBu} - 77{,}2 - 20 \log f_{MHz} \tag{3.16}$$

kde P_{dBm} je výkon (jednotkou je dBm) přijatý přijímací anténou v bodě příjmu, E_{dBu} je intenzita elektrického pole (jednotka dBμV/m) v místě příjmu, f frekvence v MHz [9].

V některých případech mohou být intenzita EM pole či přijímaný výkon vyjádřeny jejich decibelovou hodnotou, vše záleží na tom, v jakých jednotkách požadujeme výslednou hodnotu. Nutno podotknout, že použití decibelových hodnot může v určitých případech vést ke zjednodušení výpočtu dané hodnoty.

3.3 Útlum EM vln

Pokud se jedná pouze o útlum vlny šířením volným prostorem (FSL), lze jej vypočítat pomocí následujícího vzorce

$$FSL = \left(\frac{4\pi d}{\lambda}\right)^2 = \left(\frac{4\pi df}{c}\right)^2 \tag{3.17}$$

kde d je vzdálenost přenosu signálu v metrech, λ je vlnová délka signálu v metrech, c je rychlost světla v metrech za sekundu a f je frekvence signálu v Hz.

FSL je možné vyjádřit i v decibelové míře úpravami vztahu 3.17 následovně

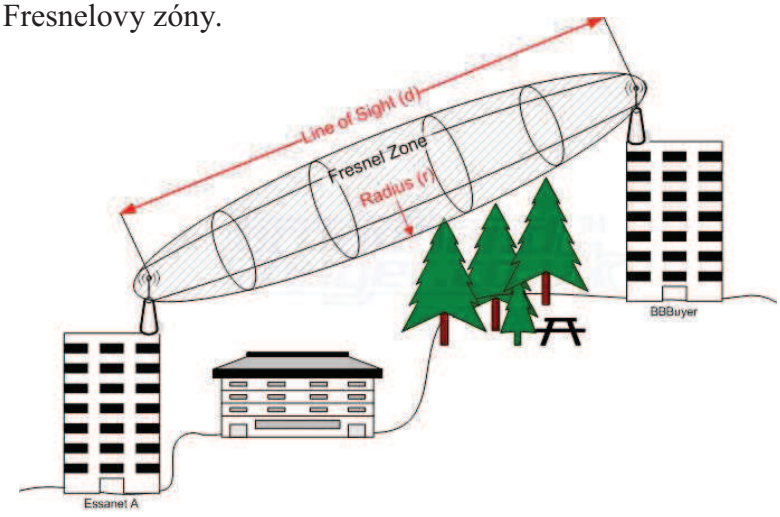
$$FSL = 10 \cdot \log_{10}\left(\frac{4\pi d}{\lambda}\right)^2 = 20 \cdot \log_{10}\left(\frac{4\pi d}{\lambda}\right) \tag{3.18}$$

což lze následně zapsat i jako

$$FSL = 32{,}44 + 20 \cdot \log_{10}(f_{MHz}) + 20 \cdot \log_{10}(d_{km}) \tag{3.19}$$

kde FSL je útlum šíření volným prostorem v dB, f_{MHz} je frekvence signálu v MHz a d_{km} je vzdálenost na kterou probíhá přenos v kilometrech.

V dané úloze se budeme věnovat pouze difrakci vlny na překážce. Zanedbáme tedy další možnosti, jako jsou vícecestné šíření, úniky signálu a vlastnosti prostředí, kterým se vlna šíří, jelikož jediným elementem způsobujícím útlum je terén, který může zasahovat do Fresnelovy zóny.



Obr. 3.1 Znázornění Fresnelovy zóny (převzato z [10])

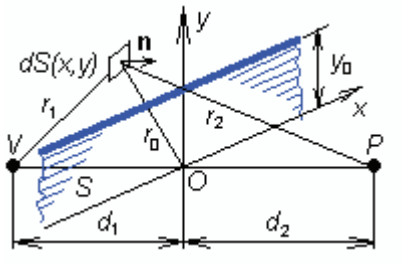
Fresnelovu zónu si lze představit jako rotační elipsoid mezi vysílačem a přijímačem, jak je znázorněno na Obr 3.1. Ideální podmínky pro šíření bez útlumu jsou „volná“ trasa vysílač-přijímač a „volná“ Fresnelova zóna.

Pokud je více než 60 % první Fresnelovy zóny volné, lze předpokládat, že signál se šíří jako při šíření ve volném prostoru. Poloměr Fresnelovy zóny v daném bodě lze vypočítat následovně

$$r_{ON} \cong \sqrt{n\lambda \frac{d_1 d_2}{d_1 + d_2}} \tag{3.20}$$

kde d_1 je vzdálenost od vysílače, d_2 vzdálenost od přijímače, λ je vlnová délka signálu a n značí číslo Fresnelovy zóny.

Nyní, pokud terén zasahuje do více, než 40 % Fresnelovy zóny, nahradíme ho rovinnou překážkou kolmou na rovinu vysílače a přijímače.



Obr. 3.2 Difrakce na překážce (převzato z [11])

Nejprve vypočteme intenzitu pole $E^{(S)}$, na rovině přepážky S tak, jakoby tam přepážka nebyla. Předpokládáme, že od zdroje V se šíří kulová vlna

$$E^{(S)} = C\frac{e^{-jkr_1}}{r_1} \tag{3.21}$$

zde C značí konstantu závislou na vyzařovaném výkonu zdroje a k je vlnové číslo [10].

S využitím Huygensova principu, kdy každý bod roviny S považujeme za Huygensův zdroj, o intenzitě $E^{(S)}$ (3.21), nyní můžeme výslednou intenzitu v bodě P vypočítat, jako součet záření jednotlivých elementů roviny S následovně. [11]

$$E^{(P)} = \frac{j}{\lambda}\int_{S_1} E^{(S)}\cos(n, r_2)\frac{e^{-jkr_2}}{r_2}\,dS \tag{3.22}$$

Další řešení problému difrakce tímto způsobem vede k výpočtu Fresnelových integrálů, což by vedlo ke zdlouhavým a náročným výpočtům.

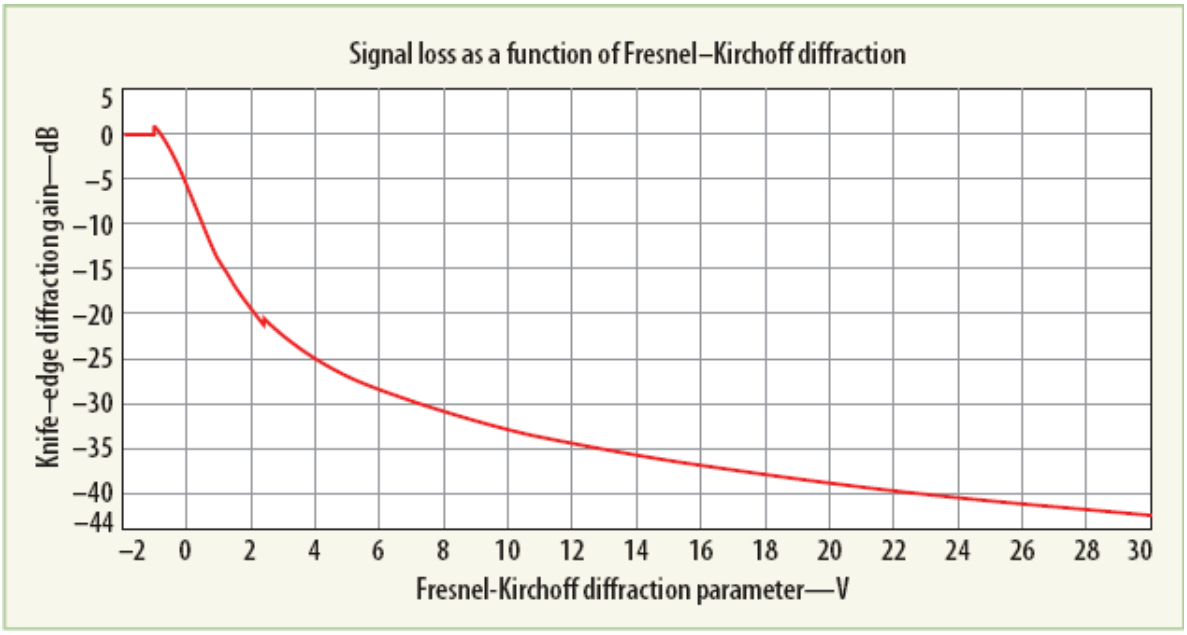
Namísto tohoto složitého řešení se rozhodneme využít řešení jednodušší, které si vystačí pouze s výškou překážky a jejími vzdálenostmi od vysílače a přijímače.

3.3.1 Aproximace překážky

Jelikož integrální výpočet útlumu překážky, je velmi složitým počinem, využijeme výpočtu pomocí difrakčního parametru. Ze znalosti výšky překážky *h*, vzdálenosti od přijímače d_1 a přijímače d_2, a vlnové délky signálu λ lze určit Fresnel-Kirchoffův difrakční parametr *v*, následovně:

$$v = h \cdot \sqrt{\frac{2 \cdot (d_1 + d_2)}{\lambda d_1 d_2}} \tag{3.23}$$

jak lze pozorovat, čím větší je výška překážky, tím větší je difrakční parametr *v*. Dále je možné z následujícího grafu určit, že s rostoucím difrakčním parametrem *v*, roste útlum překážky.



Obr. 3.3 Závislost útlumu překážky na difrakčním parametru v, převzato z [19]

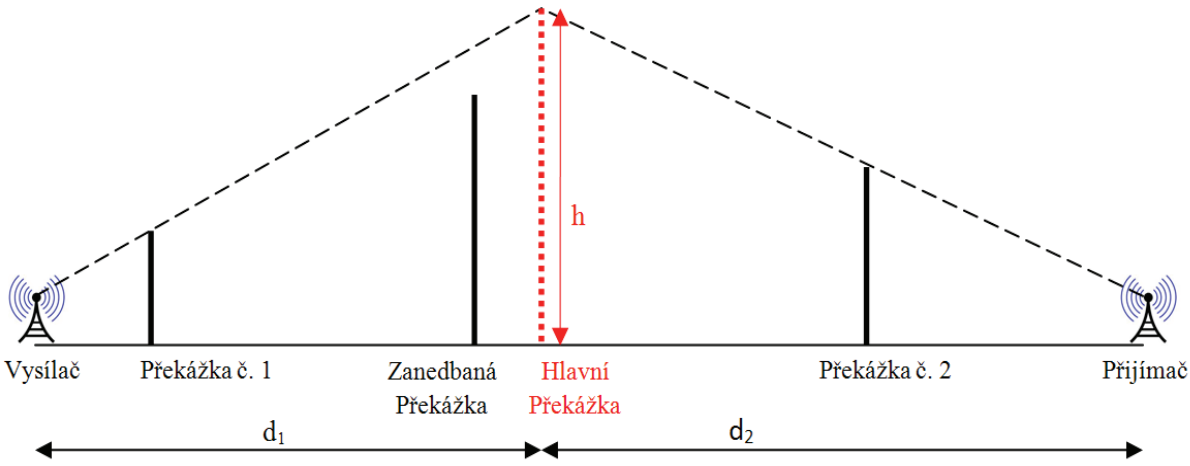
Dle Williama C. Leeho [20], lze vypočítat předpokládaný útlum překážky refrakcí, jako funkci Fresnel-Kirchoffova difrakčního parametru *v*, následovně:

$$G_d(dB) = \begin{cases} 0 & v \leq -1 \\ 20\ \log_{10}(0{,}5 - 0{,}62 \cdot v) & -1 \leq v \leq 0 \\ 20 \log_{10}\ (0{,}5 \cdot e^{-0{,}95 \cdot v}) & 0 \leq v \leq 1 \\ 20 \log_{10}\ (0{,}4 - \sqrt{0{,}1184 - (0{,}38 - 0{,}1 \cdot v)^2}) & 1 \leq v \leq 2{,}4 \\ 20 \log_{10}\left(\frac{0{,}225}{v}\right) & v > 2{,}4 \end{cases} \tag{3.24}$$

kde G_d (dB) je předpokládaný útlum překážky a *v* její ‚Fresnell-Kirchoffův difrakční parametr.

V případě výskytu více překážek na počítané trase, je možné použít některý z následujících modelů. Každý z těchto modelů přistupuje jiným způsobem k výpočtu difrakčního parametru v, jednotlivých překážek, což má za následek jiný výsledný útlum celé trasy.

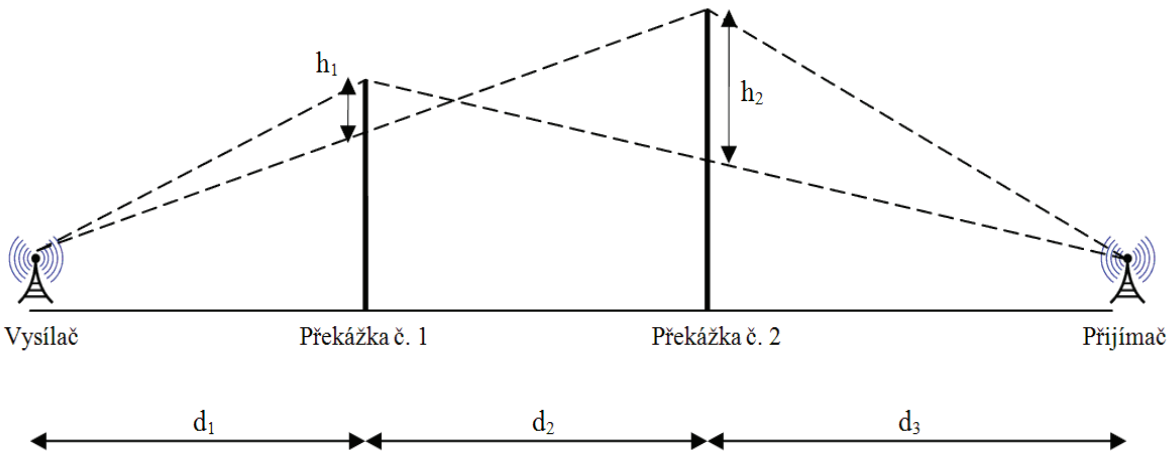
Bullingtonův model je nejjednodušším, ale také nejméně přesným případem. Redukuje celou trasu na jedinou, hlavní, překážku v místě, kde se protínají přímé viditelnosti vysílače i přijímače. Nevýhodou je, že tato překážka může být řešením pro více případů, také jsou zanedbány další překážky, které mohou mít na útlum dané trasy zásadní vliv.



Obr. 3.4 Příklad Bullingtonovy metody [autor]

Podrobnější vysvětlení provedeme na příkladu z Obr.3.4. Překážky č.1 a č.2 určují přímou viditelnost vysílače, respektive přijímače. V místě střetu přímých viditelností je vytvořena nová překážka, pro kterou vypočítáme difrakční parametr $v(h)$ dle vzorce 3.23. Dle jeho velikosti poté vypočítáme výsledný útlum danou překážkou $L(v(h))$, který odpovídá výslednému útlumu trasy. Lze si povšimnout zanedbané překážky, která ovšem způsobí největší útlum signálu na trase v porovnání s dalšími překážkami.

Epstein-Petersonův model uvažuje refrakci na každé signifikantní překážce na trase a výsledný útlum je součtem těchto útlumů.

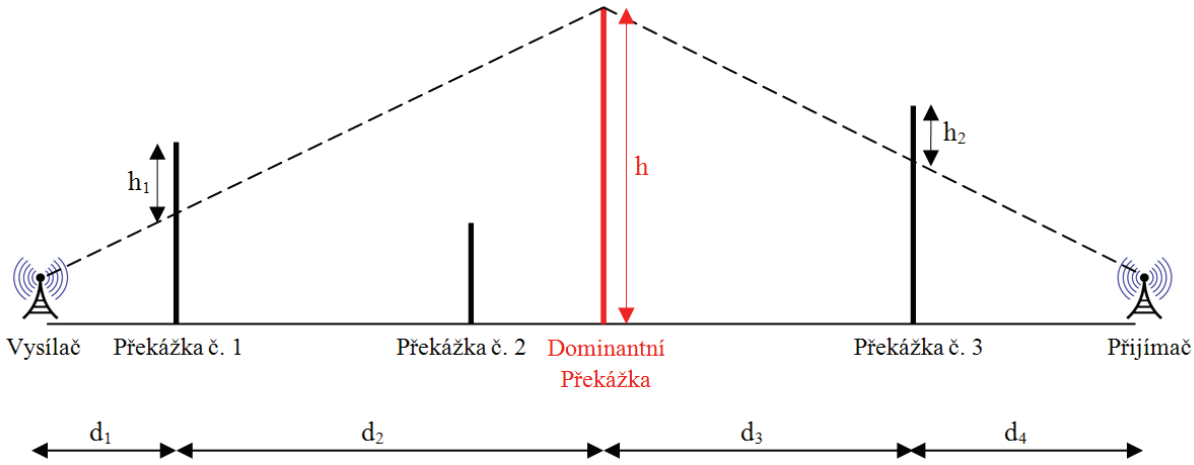


Obr. 3.5 Příklad Epstein-Petersonovy metody [autor]

Podrobnější vysvětlení provedeme na příkladu z Obr. 3.5. V tomto případě jsou na trase V-P dvě překážky. Pokud určíme přímé viditelnosti mezi překážkami, vysílačem a přijímačem, dostaneme výsledné výšky překážek vzhledem k pravidlům této metody. Díky této znalosti nyní můžeme vypočítat difrakční parametry jednotlivých překážek $v(h_1)$ a $v(h_2)$. Nyní lze určit výsledný útlum jako $L = L(v(h_1)) + L(v(h_2))$.

Na první pohled se tato metoda jeví jednoduše, ale s přibývajícím počtem překážek je složitější určit přímé viditelnosti mezi nimi, vysílačem a přijímačem. Což může vést ke složitějšímu určení výsledných výšek překážek a následně ke špatnému výpočtu.

Deygoutův model nalezne dominantní překážku na trase, například tu s největším $v(h)$, a vypočítá útlum dalších překážek vzhledem k této překážce. Celkový útlum je poté součtem útlumu dominantní překážky a druhotných překážek.



Obr. 3.6 Příklad Deygoutovy metody [autor]

Jako v předchozích případech provedeme podrobnější vysvětlení na příkladu z Obr.3.6 . Daný příklad už označil hlavní překážku, s největším difrakčním parametrem $v(h)$. Pro tuto překážku následovně provedeme výpočet jejího útlumu $L(v(h))$ dle vzorce 3.24. Nyní vytvoříme novou trasu, uvažující přímou viditelnost mezi vysílačem a dominantní překážkou, a také dominantní překážkou a přijímačem. Dále nalezneme všechny ostatní překážky, které tuto novou trasu narušují, což jsou překážky č.1 a č.3. Vypočteme jejich difrakční parametry $v(h_1)$ a $v(h_2)$. Z difrakčních parametrů dále vypočteme způsobený útlum $L(v(h_1))$ a $L(v(h_2))$. Výsledný útlum je poté dán jako $L = L(v(h)) + L(v(h_1)) + L(v(h_2))$.

Existují i další metody pro výpočet difrakce, ať už se jedná o modifikace předchozích metod, například Giovanelliho metoda [35], či jiné. Výpis nejznámějších metod lze nalézt v Tab. 3.1.

Ve většině případů se jedná o metody vycházející z některé již zmíněné metody. Tyto metody většinou upravují danou metodu tak, aby při výpočtu docházelo k co nejmenší chybě. Například výpočtem doplňujících faktorů, které se přičítají k výsledné hodnotě, či úpravou geometrie trasy, což může být například zavedení ekvivalentních výšek antén.

Tab. 3.1 Metody výpočtu difrakce

Metody pro výpočet difrakce na překážce
Bullington
Epstein-Peterson
Deygout
Giovanelli
Japanese
TheSlope UTD
Integral equation approach
The parabolic equation

V našem případě hledáme metodu co nejpřesnější, ale také nejméně náročnou na výpočetní operace. Z toho důvodu jsme se rozhodli využít Deygoutovu metodu. V rámci výpočetní náročnosti se jeví jako jedna z náročnějších, ovšem dosahuje dobrých výsledků z hlediska přesnosti i vzhledem k tomu, že Giovanelliho metoda, která dosahuje nejlepších výsledků, vychází právě z této metody.

Pro více informací si lze prostudovat literaturu [21] či jiná doporučení ITU, která se zabývá podrobnějším popisem tohoto problému. Dále také literaturu [35].

3.4 Výpočet výsledných hodnot

Naším úkolem je výpočet intenzity el. pole v místě příjmu, popřípadě přijímaného výkonu. V případě šíření signálu volným prostorem je možno použít některý ze vzorců 3.12 – 3.14. Pokud ovšem uvažujeme i ztráty refrakcí, modifikujeme vzorec 3.13 tak, aby tyto ztráty zahrnul

$$P_{přij} = P_{vys} + G_{vys} + G_{přij} - FSL - DL \tag{3.25}$$

kde *DL* označuje ztráty difrakcí v dB, které určíme pomocí vzorců 3.23 a 3.24.

V tomto bodě jsme tedy schopni vypočítat přijímaný výkon v dBm. Nyní se zaměříme na jeho vyjádření jako intenzitu pole. Toho docílíme úpravou vzorců 3.15 a 3.16. Nejprve vyjádříme ze vzorce 3.16 hodnotu E_{dBu} .

$$E_{dBu} = P_{dBm} + 77{,}2 + 20 \cdot \log_{10}(f_{MHz}) \tag{3.26}$$

V tomto případě už vlastníme hodnotu intenzity pole. Pokud danou intenzitu požadujeme v základních jednotkách, tedy V/m, upravíme vztah 3.15 k převodu na tuto hodnotu následovně

$$E_{V/m} = 10^{-6} \cdot 10^{\left(\frac{E_{dBu}}{20}\right)} \tag{3.27}$$

4 OPTIMALIZACE

Optimalizaci lze zjednodušeně popsat jako proces, při kterém hledáme nejlepší možné řešení daného problému, z určité skupiny řešení, na základě vstupních parametrů. Kvalitu daného řešení lze vyjádřit jeho kriteriální funkcí.

Kvalita řešení optimalizačního procesu je vyjádřena tzv. kriteriální funkcí, kterou formuluje uživatel s ohledem na požadovaný výstup. Kriteriální funkce je formulována tak, aby v případě ideálního řešení nabývala minimální, popřípadě maximální hodnoty. Optimalizací je pak myšleno hledání této minimální či maximální hodnoty a jejím výsledkem jsou vstupní stavové proměnné pro toto řešení. Optimalizační úlohu lze popsat pomocí rovnice

$$\begin{gathered} Minimalizuj\ \ f_m(x), \quad m = 1,2,\dots,M, \\ s\ ohledem\ na\ \ x_{min}^{(n)} \le x^{(n)} \le x_{max}^{(n)} \quad n = 1,2,\dots,N, \\ c_k(x) \ge 0, \quad k = 1,2,\dots,K. \end{gathered} \tag{4.1}$$

kde f_m označuje m-tou kriteriální funkci, M je počet kritérií, $x^{(n)}$ je n-tý prvek vektoru stavových proměnných, x_{min} a x_{max} vymezují přípustné hodnoty stavových proměnných, N je celkový počet stavových proměnných, c_k je k-tá omezující funkce, která doplňuje další závislosti mezi stavovými proměnnými nebo hodnotami kriteriální funkce [12].

V naší úloze využijeme metodu PSO, kterou podrobněji probereme v následující kapitole.

4.1 Particle swarm optimization (PSO)

Metoda představená Eberhartem a Kennedym v roce 1995 [14]. Původně inspirovaná migraci hejna ptáků při hledání potravy. Daná metoda dokáže konkurovat genetickým algoritmům, dokonce je předčí v řešení určitých úloh. Právě tuto metodu využijeme při výsledné optimalizační úloze.

Jednotlivé částice roje jsou zde reprezentovány jejich polohou a jejich rychlostí. Na začátku náhodně rozmístíme částice v prostoru a vygenerujeme jim náhodné rychlosti. Nyní označíme hodnotu každé částice jako pbest, tedy nejlepší možné řešení nalezené touto částicí. Hodnoty pbest porovnáme a tu, která dosahuje nejlepšího řešení kriteriální funkce, uložíme jako pbest, tedy nejlepší řešení nalezené všemi částicemi.

Původní algoritmus, z roku 1995, počítá nové hodnoty rychlostí částic dle následujícího vzorce

$$v_{t+1} = v_t + C_1 \cdot r_1 \cdot (pbest - x_t) + C_2 \cdot r_2 \cdot (gbest - x_t) \tag{4.2}$$

kde $\boldsymbol{v_{t+1}}$ je rychlost částice v následující iteraci, C_1 a C_2 jsou konstanty, většinou se udává $C_1 = C_2 = 2$, r_1 a r_2 jsou náhodná čísla z intervalu (0,1), ***gbest*** označuje polohu, v níž kriteriální funkce dosahuje nejlepšího řešení, nalezeného v rámci celého roje, ***pbest*** označuje polohu nejlepšího řešení nalezeného danou částicí, $\boldsymbol{x_t}$ je současná poloha dané částice v prostoru.

Následně vypočítáme nové polohy částic jako součet polohy částic z předchozí iterace a jejich nové rychlosti, určené v aktuální iteraci, *t+1*

$$x_{t+1} = x_t + v_{t+1} \tag{4.3}$$

kde $\boldsymbol{x_{t+1}}$ je nová poloha částice, $\boldsymbol{x_t}$ je poloha částice v předchozí iteraci a $\boldsymbol{v_{t+1}}$ je nová rychlost částice v této iteraci.

Dalším krokem je porovnání nově nalezených pozic částic s hodnotami ***pbest***. Pokud vykazuje nová poloha částice lepší výsledek kriteriální funkce než ***pbest*** dané částice, uložíme danou polohu jako ***pbest*** dané částice. Následně opět vybereme z ***pbest*** nejlepší řešení a uložíme ho jako ***pbest***.

Výpočet nových pozic opakujeme do té doby, dokud není splněna ukončovací podmínka. Celkový záměr PSO lze tedy popsat jako snahu o posouvání daných částic směrem k částici, která je nejblíže optimálnímu řešení.

4.1.1 Modifikace PSO

Většinou se jedná o modifikace výpočtu rychlosti částic, které mají například zajistit rychlejší konvergenci k optimálnímu řešení.

Příkladem může být zavedení inerciální váhy, představené Shi a Eberhartem [15].

$$v_{t+1} = w \cdot v_t + C_1 \cdot r_1 \cdot (pbest - x_t) + C_2 \cdot r_2 \cdot (gbest - x_t) \tag{4.4}$$

kde *w* je inerciální váha, jež je většinou vypočítanou hodnotou, která se v průběh výpočtu mění. Přesněji je snižována ze svého maxima do svého minima. Inerciální váha je obvykle vypočítána jako

$$w = w_{max} - \frac{[(w_{max} - w_{min}) \cdot N]}{N_{max}} \tag{4.5}$$

kde *w* je inerciální váha, w_{max} je maximální hodnota inerciální váhy, w_{min} je minimální hodnota inerciální váhy, *N* je počet iterací a N_{max} je maximální počet iterací.

Inerciální váha může být i fixní hodnotou, poté ovšem metoda PSO dosahuje horších výsledků, než při jejím postupném snižování. Další možnosti výpočtu inerciální váhy lze nalézt v literatuře [16].

Existují i další varianty PSO, pro diskrétní optimalizaci, obsahující komponenty z jiných přístupů, s jinými pravidly pro výpočet rychlosti částic atd. [17]

Právě modifikovanou variantu PSO (4.4), jak s proměnnou (4.5), tak s fixní inerciální váhou, využijeme při řešení optimalizační úlohy.

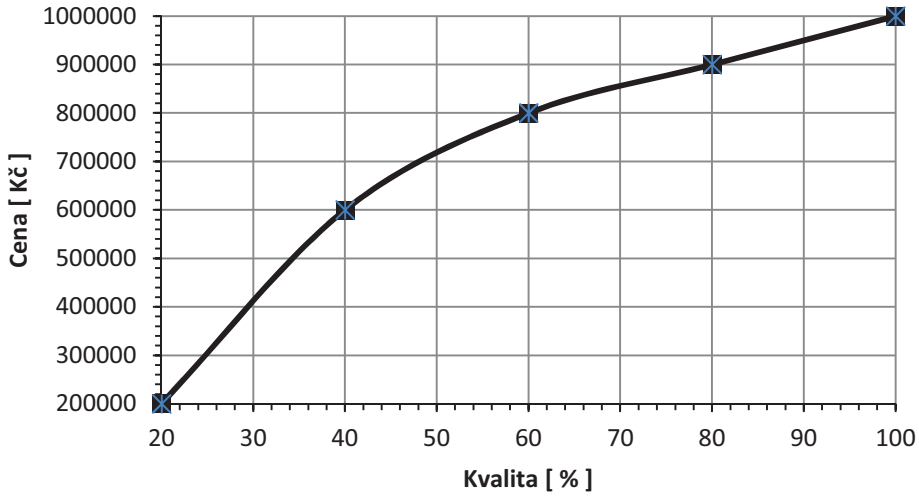
4.2 Vícekriteriální optimalizace

Pokud daný problém optimalizace zahrnuje více jak jednu kriteriální funkci, hovoříme o vícekriteriální optimalizaci. Poté nastává otázka, zda jsou některá z kritérií protichůdná či nikoli. Pokud daná kritéria protichůdná nejsou, měla by úloha směřovat k jedinému optimu.

V případě, že jsou daná kritéria protichůdná, je řešením vícekriteriálního problému tzv. Paretovo čelo.

Paretovo čelo je vlastně jakýsi soubor řešení s nejlepšími hodnotami kriteriálních funkcí vzhledem k ostatním kritériím. Pokud budeme chtít zlepšit dané řešení v rámci jednoho kritéria, zhorší se nám tím další kritéria.

Příkladem budiž následující příklad, sledujme kvalitu a cenu výrobku. Budeme chtít dosáhnout co nejvyšší kvality, ale zároveň co nejmenší ceny výrobku. V tomto případě tedy máme dvě protichůdná kritéria. Podívejme se nyní na Obr 4.1, který ilustruje daný případ.



Obr. 4.1 Příklad Paretova čela

Z daného obrázku je patrné, že existuje více optimálních řešení. Většinou nás zajímá řešení jediné, poté je nutné se rozhodnout, zda je pro nás výhodnější nejvyšší kvalita, přičemž zaplatíme nejvíce nebo se rozhodneme pro nejmenší cenu na úkor kvality. Další možností je zvolit kompromis mezi danými kritérii a vybrat si některé z řešení, jež leží uprostřed Paretova čela.

Dále se budeme zabývat problémem nalezení Paretova čela při vícekriteriální optimalizaci, čehož je dosaženo pomocí tzv. nedominatního třízení.

4.2.1 Nedominantní třízení

Většina vícekriteriálních optimalizačních algoritmů využívá principu dominance při hledání výsledných řešení. Princip dominance spočívá v jednoduchém porovnání, zda jedno řešení dominuje druhé či nikoli. Představme si, že porovnáváme dvě řešení problému, který má dvě kriteriální funkce, poté mohou nastat dva případy [18].

1) Dominance.

   V tomto případě jsou hodnoty kriteriálních funkcí prvního řešení lepší, než hodnoty kriteriálních funkcí řešení druhého, popřípadě naopak. Tedy jedno řešení je lepší v obou parametrech.

2) Nedominantní řešení.

   V tomto případě má první kriteriální funkce prvního řešení lepší hodnotu, než první kriteriální funkce řešení druhého. Zároveň má však druhá kriteriální funkce druhého řešení lepší hodnotu, než druhá kriteriální funkce prvního řešení. Ani jedno z řešení tedy není lepší než druhé v obou parametrech.

Jak již bylo zmíněno, řešením vícekriteriální optimalizace je tzv. Paretovo čelo, které obsahuje všechna nejlepší řešení daného problému z hlediska všech kriteriálních funkcí.

Všechna tato řešení jsou vůči sobě nedominovaná, ale dominují všechna ostatní řešení, která mohou nastat. Z tohoto důvodu je třeba provést nedominantní třízení.

K tomu můžeme využít následujících přístupů.

1) Slow & Naive [18]

   V tomto přístupu je každé řešení i porovnáno s ostatními řešeními pro dominanci. Pokud je řešení i dominováno alespoň jedním z ostatních řešení populace, označíme ho jako dominované. Nicméně pokud žádné z řešení nedominuje řešení i, poté řešení i patří do skupiny nedominovaných řešení.

   Daný přístup lze charakterizovat v několika krocích následovně, kde hledáme nedominovaný set řešení množiny P o velikosti N.

   I. Nastavíme čítač počtu řešení i = 1 a vytvoříme prázdnou množinu P‘, do které budeme ukládat nedominovaná řešení.
   II. Řešení j, z množiny P, takové, že $i \neq j$, porovnáme na dominanci s řešením i. Pokud řešení j dominuje řešení i, přejdeme ke kroku IV.
   III. Pokud nám v množině P zbývají další řešení, zvýšíme j o 1a vrátíme se do kroku II. Pokud nám již žádná řešení nezbývají, uložíme řešení i do množiny P‘, tzn. I je nedominovaným řešením.
   IV. Zvýšíme i o 1. Pokud je $i < N$, přejdeme ke kroku II. Jinak zastavíme hledání a označíme množinu P‘ za nedominovaná řešení.

2) Continuously updated [18]

   Tento přístup je podobný předchozímu, je zde však proveden lepší způsob uchování řešení, což vede k rychlejšímu běhu algoritmu. V tomto přístupu je každé řešení porovnáno s částečně naplněnou populací pro dominanci. Na začátku je první řešení z populace uloženo do prázdné množiny P‘. Poté je každé řešení i (od druhého po poslední) porovnáno se všemi řešeními v P‘.

   Pokud není řešení i dominováno žádným z řešení v množině P‘, je řešení i uloženo do této množiny. Pokud některé z řešení i dominuje nějaké řešení v množině P‘, pak je dominované řešení odstraněno z množiny P‘. Pokud nám nezbývají žádná řešení i k porovnání, jsou výsledné hodnoty uložené v množině P‘ nedominovaná řešení.

Daný přístup lze charakterizovat v několika krocích následovně

I. Inicializujeme P‘ = {1}. Nastavíme čítač počtu řešení i = 2
II. Nastavíme j = 1.
III. Porovnáme řešení i s řešením j z P‘ na dominanci.
IV. Pokud i dominuje j, vymažeme j-tý člen z množiny P‘. Pokud j < |P‘|, zvýšíme j o 1 a pokračujeme krokem III, jinak pokračujeme krokem V. Zároveň, pokud j-tý člen P‘ dominuje i, zvýšíme i o 1 a pokračujeme krokem II.
V. Uložíme i do množiny P‘. Pokud je i < N, zvýšíme i o 1 a přejdeme ke kroku II. Jinak zastavíme hledání a označíme množinu P‘ za nedominovaná řešení.

Popis a postup řešení daných přístupů byl převzat z [18], kde lze nalézt i další možnost přístupu.

4.2.2 Vícekriteriální PSO (MOPSO)

Vícekriteriální PSO je modifikací základní metody PSO tak, abychom mohli řešit problémy s více parametry.

Jelikož výsledkem vícekriterální optimalizace je více než jedno řešení, musíme upravit metodu PSO tak, aby nesměřovala pouze k jedinému optimu. Toho dosáhneme úpravou výběru hodnot gbest a pbest pro jednotlivé částice.

Nyní bude hodnota pbest každé částice vybrána, jako první nedominované řešení, jež bylo částicí nalezeno. Do doby, než částice nalezne dané řešení, porovnáváme novou hodnotu částice s jejím pbest. Pokud je pbest dominováno novou hodnotou částice, označíme tuto hodnotu jako nové pbest.

Každá částice také bude mít svou vlastní hodnotu gbest. Daná hodnota bude určena jako nedominované řešení, kterému je částice nejblíže v n-rozměrném prostoru.

Posledním krokem je zajištění rovnoměrného pokrytí Paretova čela. K tomu nám poslouží tzv. crowding distance (CD).

Při výpočtu crowding distance vzestupně seřadíme jednotlivé kriteriální funkce. Nejvyšší a nejmenší hodnotě kriteriální funkce přiřadíme nějakou vysokou hodnotu CD, například nekonečno. Pro ostatní hodnoty kriteriálních funkcí vypočítáme CD následovně

$$CD = \frac{f_{i+1} - f_{i-1}}{f_{max} - f_{min}} \tag{4.6}$$

kde f_{max} je maximální hodnota kriteriální funkce, f_{min} je minimální hodnota kriteriální funkce, f_{i+1} je následující hodnota kriteriální funkce, f_{i-1} je předchozí hodnota kriteriální funkce.

Výsledná CD, je poté součtem jednotlivých CD daných řešení, vypočítaných pro všechny kriteriální funkce. Řešení dosahující nejvyšší hodnoty CD jsou pro nás nejzajímavější, také nám zajistí rovnoměrné pokrytí Paretova čela.

5 MODEL POKRYTÍ

Jak již bylo zmíněno, daný model budeme vytvářet v programovém prostředí Matlab. Z důvodů snadnějšího zpracování a také díky schopnosti zpracovat větší objem dat než starší verze byl zvolen Matlab 2013a, popřípadě verze 2013b.

Základem je úprava skriptu pro načítání a vykreslení *.hgt souborů tak, abychom obdrželi matici obsahující hodnoty nadmořských výšek. Následně se budeme zabývat výpočtem pokrytí ve volném prostoru a také šířením s uvažováním difrakce na překážkách. Nakonec se budeme věnovat úpravám skriptu pro optimalizační algoritmy.

5.1 Načítání digitálních map

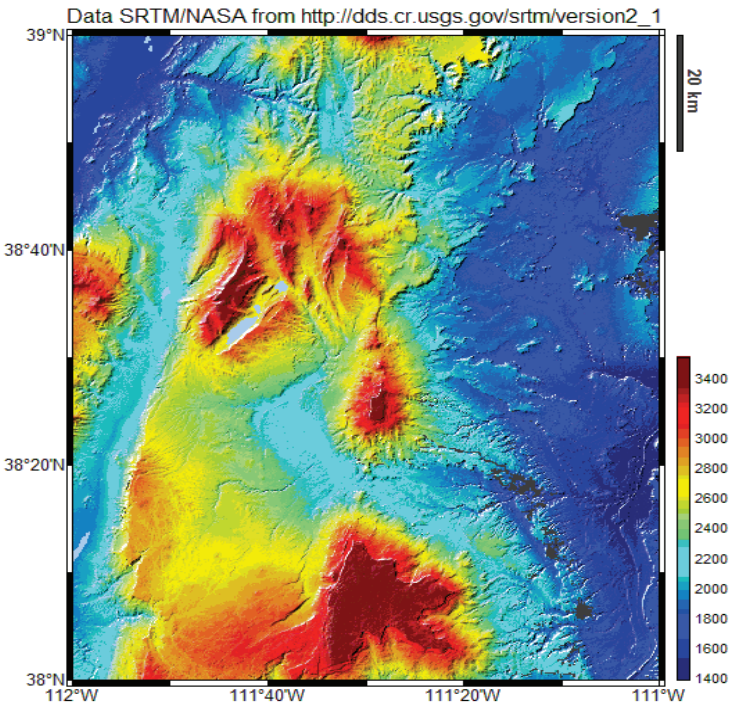
K načtení digitální mapy použijeme skript vytvořený François Beauducelem, který lze nalézt na www stránce viz. [12]. Daný skript upravíme použitím následujícího kódu

```
save('Hodnoty.mat','X')
```

což zajistí uložení hodnot nadmořské výšky do souboru `'Hodnoty.mat'`. V hlavním programu poté zavoláme již upravenou funkci pro načítání map. A následně použijeme funkci **`load`** k načtení potřebných dat z námi vytvořeného souboru. Nejdříve je však zapotřebí danou mapu načíst, tak aby se do daného souboru uložily potřebné hodnoty.

```
readght();
load('Hodnoty.mat','X')
```

Tento postup zajistí uložení dat do souboru `Hodnoty.mat`, následné načtení a také vykreslení mapy pomocí funkce readght.



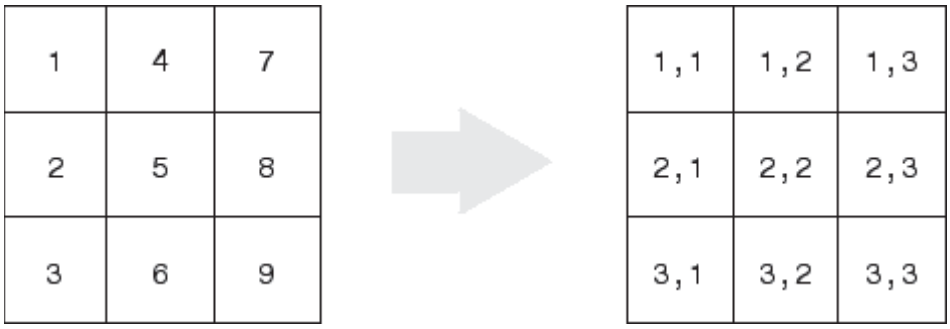
Obr. 5.1 Příklad vykreslené mapy, pro souřadnice N38W112

5.2 Geometrie mapy a tras

Před samostatnými výpočty intenzity elektromagnetického pole či přijímaného výkonu je zapotřebí znát vzdálenost mezi vysílačem a přijímačem. Dále je také zapotřebí znát hodnoty výšek terénu k určení, zda na dané trase terén zasahuje do Fresnelovy zóny či nikoli.

Načtený soubor digitálních dat je vlastně maticí, která obsahuje hodnoty nadmořských výšek. Hledáme tedy vzdálenost dvou bodů, vysílač a přijímač (dále jen V-P), v dané matici. Dále také hledáme počet bodů jednotlivých tras V-P a výškové hodnoty odpovídající těmto bodům na dané trase.

Předtím, než se budeme věnovat následujícím kapitolám, je třeba porozumět indexování pozic matice v programu Matlab. Jednotlivé pozice v matici můžeme reprezentovat jejich indexem, nebo jejich souřadnicemi, které označují řádek a sloupec, kde se pozice nachází. Příklad indexace je uveden na Obr 5.2



Obr. 5.2 Pozice v matici, dle indexů (vlevo) a souřadnic (vpravo), převzato z [32]

Z daného indexu v matici lze zjistit jeho souřadnice x a y. Popřípadě naopak, ze souřadnic x a y lze zjistit index dané pozice. K tomu využijeme následujících funkcí.

ind2sub()

sub2ind()

Indexaci v našem případě využijeme k nalezení kombinací všech možných pozic daných souřadnicemi x a y v mapě, a to pomocí následujícího kódu

```
[sx,sy] = ind2sub(size(Mapa),1:numel(Mapa));
```

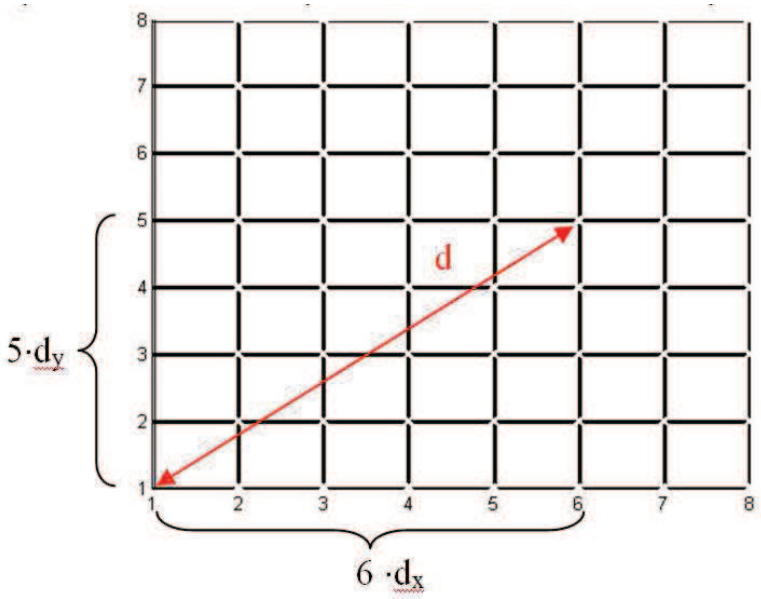
kde sx a sy jsou vektory, které udávají všechny možné umístění reprezentované souřadnicemi x a y, `size(Mapa)` je označení matice, ve které dané souřadnice hledáme a její velikosti. Nakonec `1:numel(Mapa)` je označení indexů jedna až celkový počet indexů v matici.

Tyto pozice jsou nadále použity k výpočtu vzdálenosti V-P a také při výpočtu výškových bodů trasy.

Poslední zmínkou je, že pokud se v dalším textu budeme bavit o souřadnici x, budeme se pohybovat po řádcích matice, pokud o souřadnici y, poté se budeme pohybovat po sloupcích matice.

5.2.1 Euklidovská vzdálenost

Je vzdálenost mezi dvěma určenými body v naší mapě, pokud neuvažujeme výškový rozdíl, pro lepší představu si lze prohlédnou Obr. 5.3.



Obr. 5.3 Příklad euklidovského prostoru, se znázorněním vzdáleností dx, dy, a d

K výpočtu vzdálenosti těchto bodů využijeme vztah pro výpočet vzdálenosti v euklidovském prostoru. Za předpokladu, že máme dva body o souřadnicích x_1,y_1 a x_2,y_2 můžeme danou vzdálenost vypočítat jako

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{5.1}$$

daný vztah platí, pokud jsou vzdálenosti d_x a d_y rovny jedné. V případě jiné hodnoty je daný vztah nutno upravit následujícím způsobem

$$d = \sqrt{d_x \cdot (x_1 - x_2)^2 + d_y \cdot (y_1 - y_2)^2} \tag{5.2}$$

Výsledný kód pro výpočet vzdálenosti vysílač-přijímač bude tedy vypadat následovně

```
for x = 1:Rozmer
  for y = 1:Rozmer
       Vzdalenost(x,y) = sqrt(dx*(x0-x)^2+dy*(y0-y)^2);
  end
end
```

kde x, y jsou souřadnice přijímače, x0,y0 jsou souřadnice přijímače a Rozmer je počet bodů dané mapy (v našem případě 3601 nebo 1201).

Jelikož známe pozici vysílače a pozice přijímačů, lze využít funkce pdist2, daný kód poté vypadá následovně

```
Euklid2 = pdist2([x_vys,y_vys],[sx',sy'],'euclidean');
Euklid = rozl*Euklid2;
```

kde Euklid2 je jednotková euklidovská vzdálenost, x_vys a y_vys jsou souřadnice vysílače v mapě, sx a sy jsou souřadnice všech ostatních bodů v mapě, ke kterým se daná vzdálenost počítá. Výslednou vzdálenost poté obdržíme vynásobením jednotkové vzdálenosti rozlišením mapy, v tomto případě je to možné, jelikož rozlišení je v obou osách stejné.

Použitím daného kódu jsme také dosáhli menší výpočetní náročnosti. Problému výpočetní náročnosti se budeme věnovat později v kapitole 1.5.

5.2.2 Čebyševova vzdálenost

Pokud k této vzdálenosti přičteme jedničku, pak v našem případě vyjadřuje počet bodů na trase V-P. Pro lepší představu si můžeme prohlédnout Obr. 5.4.

0	1	2	3
1	1	2	3
2	2	2	3

Obr. 5.4 Příklad Čebyševovy vzdálenosti v matici od bodu (1,1)

Pro výpočet Čebyševovy vzdálenosti opět využijeme funkci pdist2, kde změníme hledanou vzdálenost na čebyševovu, výsledný kód je poté

```
Chebyd = pdist2([x_vys,y_vys],[sx',sy'],'chebychev');
```

5.2.3 Výpočet výškových bodů trasy

Pokud si představíme trasu V-P jako úsečku definovanou jejími krajními body x_0,y_0 a x_1,y_1 , lze vypočítat další body na trase ze znalosti pozice bodu x či bodu y následovně

$$x = \frac{(x_1 - x_0)}{(y_1 - y_0)} \cdot (y - y_0) + x_0 \tag{5.3}$$

$$y = \frac{(y_1 - y_0)}{(x_1 - x_0)} \cdot (x - x_0) + y_0 \tag{5.4}$$

Nyní nastává otázka, který bod, x či y, můžeme do rovnice zadat, a který vypočítáme. Odpověď dostaneme ze vzdálenosti obou bodů na jednotlivých osách x a y.

Pokud je vzdálenost x větší jak vzdálenost y, zadáváme poté do rovnice hodnoty v intervalu (x_0,x_1) a dostáváme výsledné hodnoty y. Pokud je tomu naopak, zadáváme do rovnice hodnoty v intervalu (y_0,y_1). V případě, že jsou si vzdálenosti x a y rovny, můžeme si vybrat, který z intervalů hodnot využijeme k výpočtu.

Výslednou vypočítanou hodnotu pozice x či y poté zaokrouhlíme na nejbližší celé číslo. Tímto krokem se vyhneme hledání hodnoty na dané pozici, která by pravděpodobně byla určitým „průměrem“ hodnot okolních.

Výsledný kód pro danou funkci lze tedy charakterizovat v několika krocích takto

1) Ze znalosti krajních bodů určíme délky v jednotlivých směrech.

```
%Určení krajních bodů přímky
p1 = [x_vys,y_vys];      %Souřadnice vysílače
p2 = [sx,sy];            %Souřadnice přijímače

%Určení délkek v jednotlivých směrech
dx = p1(1):p2(1);
dy = p1(2):p2(2);

%Kontrola
if isempty(dx) == 1
      dx = fliplr(p2(1):p1(1));
else
end
if isempty(dy) == 1
      dy = fliplr(p2(2):p1(2));
else
end
```

Kontrola nám v daném případě slouží ke zjištění, zda jsou vektory délek prázdné či nikoli. Pokud provedeme v Matlabu zápis `a = 1:5`, obdržíme výsledný vektor a s hodnotami 1,2,3,4,5. Pokud ovšem provedeme zápis `a = 5:1`, obdržíme prázdnou matici o rozměrech 1x0.

Pro porovnání použijeme funkci **`isempty()`**, jejíž výsledek je jedna, pokud je testovaný vektor či matice prázdnou množinou. Pokud nastane tato situace, provedeme opačný zápis proměnné, tedy `a = 1:5`. Dále využijeme funkce **`fliplr()`**, která zajistí převrácení pořadí hodnot daného vektoru. Poté po zápisu **`fliplr(a)`** dostáváme výsledný vektor s hodnotami 5,4,3,2,1.

2) Pokud je vzdálenost ve směru x (řádky) větší než ve směru y (sloupce).

Nejprve provedeme porovnání vzdáleností podmínkou **`if`**, pokud je tato podmínka splněna, pokračuje v následujícím kódu. Pokud daná podmínka splněna není, přesouváme se do bodu 3 v rámci dané funkce. Dalším krokem je vytvoření vektoru s hodnotami souřadnice x, které budeme zadávat do rovnice 5.4 a jejich kontrola (viz. krok 1).

Nyní když máme připraveny souřadnice x, přejdeme k samotnému výpočtu souřadnice y dle vzorce 5.4. Vypočtené souřadnice zaokrouhlíme na celá čísla, jelikož souřadnice v matici jsou celá čísla.

Následně nalezneme indexy daných řešení. Posledním krokem je uložení hodnot o daných indexech do výsledného vektoru.

V rámci posledního kroku také provedeme kontrolu, zda některá z hodnot indexu není číslem, a to pomocí funkce **isnan()**. Výsledkem této funkce je jedna, pokud je vstupní hodnota NaN. Tato situace nastává pouze v případě, že souřadnice vysílače i přijímače jsou ve stejném bodě.

Pokud tato situace nastane, uložíme do dané pozice na trase hodnotu odpovídající hodnotě na pozici vysílače.

```
%Podmínka velikosti
if length(dx) > length(dy)

%Vytvoření vektoru s hodnotami souřadnice x a jeho kontrola
x = p1(1):p2(1);
if isempty(x) ==1
   x = fliplr(p2(1):p1(1));
else
end

%Výpočet hodnot y
y = round((x-p1(1))* (p2(2)-p1(2))/ (p2(1)-p1(1))+p1(2));

%Nalezení indexů vypočítaných souřadnic
pozice = sub2ind(size(Mapa),x',y');

%Kontrola a vytvoření výsledného vektoru trasy
if isnan(pozice) == 1
   Trasa=Mapa(x_vys,y_vys);
else
   Trasa=Mapa(pozice);
end
```

3) Pokud je vzdálenost ve směru y (sloupce) větší než ve směru x (řádky).

Výsledný kód je v tomto případě téměř totožný s krokem 2. Pokud jsme se dostali do tohoto kroku, znamená to, že podmínka **if** v prvním bodě nebyla splněna. Budeme tedy pokračovat v dané podmínce zápisem podmínky takto

```
%Podmínka velikosti
elseif length(dy) > length(dx)
```

kde použitý zápis podmínky **elseif** odkazuje na další podmínku. Pokud tato podmínka splněna není, přecházíme ke kroku 4). Pokud daná podmínka splněna je, pokračujeme ve výpočtu, nyní však přizpůsobíme výpočet podmínce.

```
%Vytvoření vektoru s hodnotami souřadnice y a jeho kontrola
y = p1(2):p2(2);
if isempty(y) ==1
   x = fliplr(p2(2):p1(2));
else
end
```

```
%Výpočet hodnot x
x = round((y-p1(2))* (p2(1)-p1(1))/ (p2(2)-p1(2))+p1(1));
```

Nalezení indexů daných souřadnic, jejich kontrola a vytvoření výsledného vektoru probíhá pomocí totožného kódu, jako v kroku 2).

4) Pokud jsou obě vzdálenosti stejné.

V případě, že není ani jedna z podmínek v předchozích krocích splněn, dostáváme se do posledního kroku. Jelikož je tento krok poslední možností, podmínku **`if`** zde nahradíme zápisem **`else`**, který nám odkazuje na všechna ostatní možná porovnání. Také nesmíme zapomenout ukončit cyklus podmínek klíčovým slovem **`end`**.

Výsledný kód posledního kroku vznikne vložením kódu některého z předchozích kroků, ve kterém vynecháme podmínku velikosti, mezi podmínku **`else`** a ukončení **`end`**.

Nyní si předvedeme celkový kód na následujícím příkladu. Mějme matici **A** o rozměrech 4x4, kterou náhodně naplníme čísly 1 až 16. Poté určíme krajní body, pro které budeme danou trasu počítat, např. `p1 = [1,1]` a `p2 = [4,3]`.

$$A = \begin{matrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{matrix} \qquad dx = 4 \qquad dy = 3$$

$$x = [1, 2, 3, 4]$$

$$y = [1, 2, 2, 3]$$

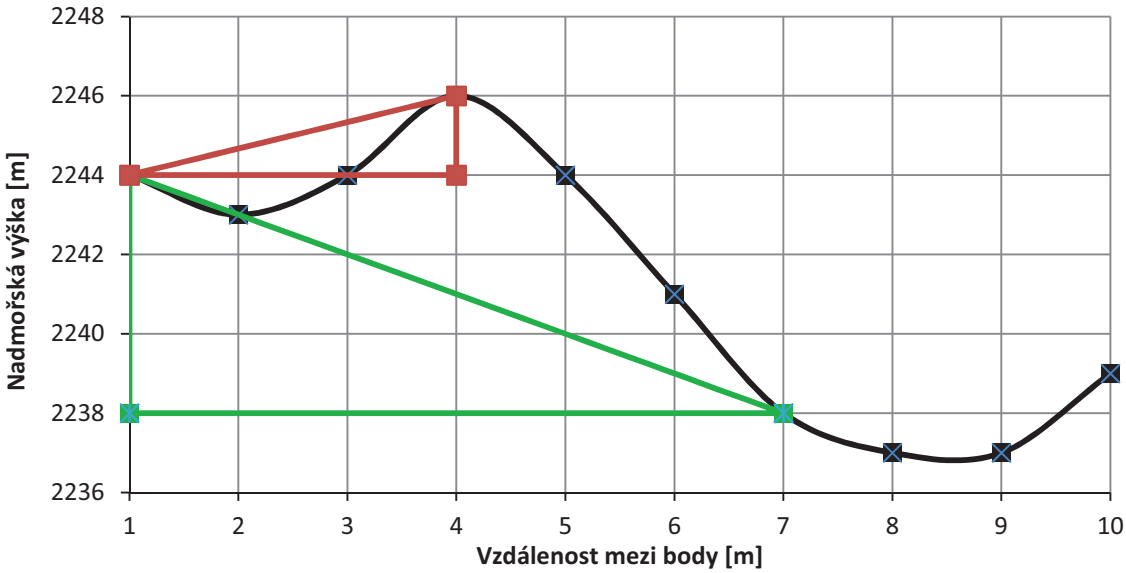
$$\begin{matrix} x & y \\ 1 & 1 \\ 2 & 2 \\ 3 & 2 \\ 4 & 3 \end{matrix} \quad => \quad \begin{matrix} Index \\ 1 \\ 6 \\ 7 \\ 12 \end{matrix}$$

$$A(1) = 16, A(6) = 11, A(7) = 7, A(12) = 15$$

$$výsledek = [16 \quad 11 \quad 7 \quad 15]$$

5.3 Výpočet útlumu signálu

Dalším krokem je výpočet útlumu signálu. V našem případě je jediným elementem způsobujícím útlum, včetně útlumu volným prostorem (FSL), terén, který může zasahovat do Fresnelovy zóny. Výpočet výsledného pokrytí lze považovat za výpočet velkého počtu spojů bod-bod a právě tento spoj bude následujícím ukázkovým příkladem.

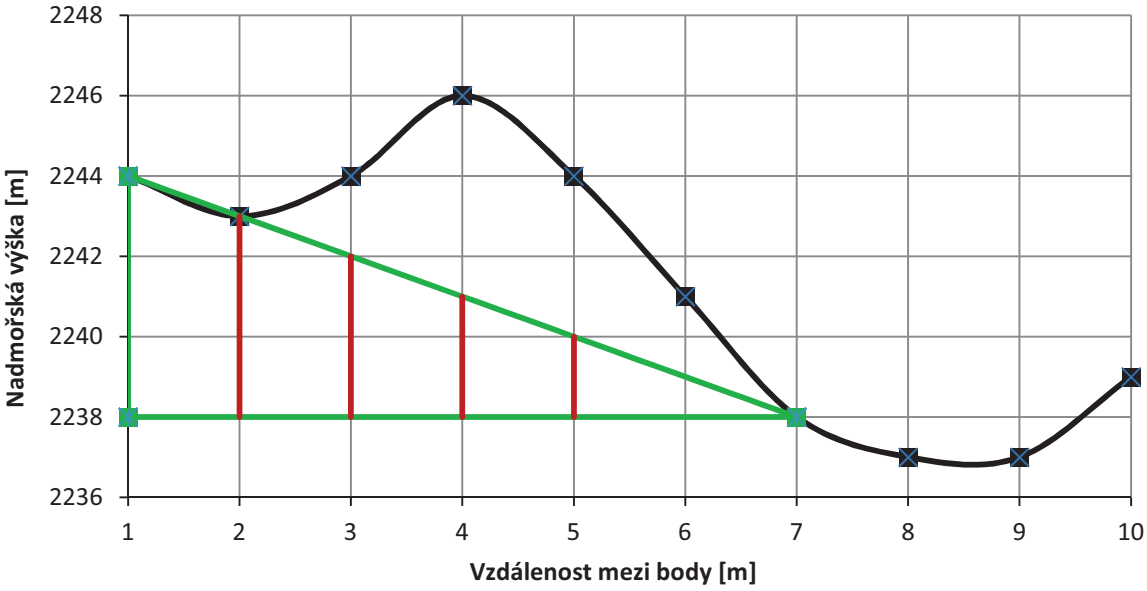


Obr. 5.5 Příklad spoje bod-bod pro výpočet útlumu

Ukázková mapa obsahuje 10 bodů o různých nadmořských výškách. Nejprve se zaměříme na výpočet přímé trasy mezi vysílačem a přijímačem (dále jen vzdálenost V-P), kterou můžeme vypočítat, pokud známe nadmořské výšky a euklidovskou vzdálenost V-P. Pokud je rozdíl v nadmořské výšce vysílače a přijímače nulový, přímá trasa má stejnou délku jako euklidovská vzdálenost. Pokud se dané výšky liší, výslednou trasu počítáme jako přeponu pravoúhlého trojúhelníku, kde jsou odvěsny reprezentovány rozdílem výšek a euklidovskou vzdáleností. Příklady těchto tras jsou znázorněny v Obr 5.5 pomocí barevných trojúhelníků.

Nyní, když známe přesnou vzdálenost V-P, lze vypočítat poloměry Fresnelových zón na trase (3.16). Pro zjednodušení uvažujeme, že v místě, kde daný terén porovnáváme, je Fresnelova zóna kolmá na trasu (formulovat).

Dále vypočítáme mezní výšky, které na dané trase reprezentují výšku, v níž je hodnota výšky překážky nulová. V tomto případě mohou nastat tři případy. Za prvé, když je rozdíl nadmořských výšek vysílače a přijímače nulový. Potom vypočteme mezní výšku jako výšku místa vysílače. Následně porovnáme výškové hodnoty na trase V-P s danou mezní výškou. Rozdíl mezi mezní výškou a daným výškovým bodem poté uvažujeme za výšku překážky. Nutno podotknout, že výška překážky může dosahovat i záporných hodnot



Obr. 5.6 Znázornění počítaných mezních výšek červenou barvou

Ve druhém případě je přijímač položen níže jak vysílač. Mezní výšky jsou tedy vypočítány jako imaginární strany pravoúhlého, červeně znázorněné, trojúhelníku, které jsou kolmé na osu vzdálenosti. Tyto strany jsme schopni spočítat pomocí Sinové věty a pravidla, že vnitřní součet úhlů trojúhelníku je 180°. Pokud známe tyto hodnoty, opět k nim přičteme výšku přijímače.

V posledním případě, přijímač výše jak vysílač, je postup téměř totožný jako ve druhém, namísto výšky přijímače však přičítáme nadmořskou výšku vysílače.

Nyní lze vypočítat výslednou hodnotu intenzity EM pole (3.17, 3.18) s tím, že pro každý bod na dané trase, který je reprezentován jeho nadmořskou výškou, je vypočítána výška překážky a následně její útlum.

Nutno podotknout, že pokud počítáme intenzitu EM pole, nejbližší body kolem vysílače jsou počítány bez útlumu. Ať už je rozdíl výšek v tomto případě kladný či záporný, na takto krátkou vzdálenost se jedná o komunikaci na přímou viditelnost, při které se uplatní jen útlum volným prostorem.

5.3.1 Aplikace Deygoutovy metody

Nyní, když máme mapový podklad načtený a rozdělený na všechny možné kombinace tras V-P. A také víme, jak určit mezní výšky a z nich výšky překážek, je nutné vypočítat výsledný útlum trasy.

K tomu využijeme aplikaci Deygoutovy metody popsané v kapitole 3.3.1. Ze známých výšek překážek a jejich útlumů vybereme hlavní překážku s největším útlumem. Poté rozdělíme původní trasu na dvě nové trasy, trasu vysílač – hlavní překážka a trasu hlavní překážka – přijímač. Od tohoto bodu pokračujeme, jakoby od začátku. Pro všechny body na daných trasách vypočítáme výšku překážky a její následný útlum. Nyní však nehledáme hlavní překážku na dané trase, nýbrž sečteme útlum způsobený jednotlivými překážkami.

Čímž dostáváme dvě proměnné, první reprezentuje útlum na trase vysílač – hlavní překážka, druhá na trase hlavní překážka – přijímač. Výslednou hodnotu útlumu trasy poté obdržíme jako součet útlumu hlavní překážky a těchto dvou proměnných.

V našem případě je v daném modelu nejdříve nalezena dominantní překážka, určena její pozice a útlum, a to následujícím způsobem a pomocí následujících úseků kódu.

1) Nejdříve jsou určeny parametry, které jsou potřebné pro výpočet.

```
Cesta          %Soubor obsahující výškové hodnoty bodů na trase
Vzdalenost     %délka trasy
lambda         %lambda = c/f
h_vysilac      %výška vysílače
```

2) Vypočítáme rozdíl výšek vysílače a přijímače, z něj vypočítáme reálnou délku trasy V-P a také úhly v daném trojúhelníku.

```
Rozdil = Cesta(1,end)-(Cesta(1,1)+h_vysilac);
%Cesta(1,end)je pozice přijímače,Cesta(1,1) pozice vysílače

if Rozdil > 0;           %Přijímač je výše jak vysílač
   Trasa = sqrt(Vzdalenost^2+Rozdil^2);
elseif Rozdil < 0;       %Vysílač je výše jak přijímač
   Trasa = sqrt(Vzdalenost^2+Rozdil^2);
else Rozdil == 0;        %Vysílač i přijímač ve stejné výšce
   Trasa = Vzdalenost;
end

Uhel_TR = asind(Vzdalenost/Trasa) ;
Uhel_TN = 90-Uhel_TR;
```

3) Vypočítáme vzdálenosti d_1 a d_2 s uvažováním výjimky o nejbližších bodech.

```
Map_l = length(Cesta);  %Zjištění délky cesty
if Map_l ==1 || Map_l==2
   d1 = 0;
   d2 = 0;
 else
   d1 = (Trasa/(Map_l-1)):(Trasa/(Map_l-1))…
        :(Trasa-(Trasa/(Map_l-1)));
   d2 = fliplr(d1);
end
```

Kde podmínka **if** označuje pozice při kterých je délka cesty V-P rovna jedné, nacházíme se tedy na pozici vysílače, popřípadě kdy je daná délka rovna dvěma, poté se nacházíme v některém z nejbližších bodů v mapě a uvažujeme komunikaci na přímou viditelnost, tj. bez útlumu a vzdálenosti d_1 a d_2 nejsou zapotřebí.

Funkce **fliplr()** nám v tomto případě slouží k vytvoření druhého vektoru vzdáleností d_2. Jelikož víme, že součet vzdáleností d_1 a d_2 k danému bodu je roven celkové vzdálenosti, poté stačí jen přeřadit vzdálenosti d_1 do obráceného pořadí.

Je také nutné si povšimnout, že počítáme menší počet vzdáleností d_1 a d_2 než je počet bodů na trase, což je způsobeno tím, že pro krajní pozice, ve kterých je umístěn vysílač, respektive přijímač, nejsou tyto vzdálenosti zapotřebí.

4) Vypočítáme hodnoty mezních výšek. Odebereme krajní body.

```
if Rozdil > 0;              %Přijímač je výše jak vysílač
   pom_d_1 = 0:Vzdalenost/(Map_l-1):Vzdalenost;
   Mez_h = Cesta(1,1) + (pom_d_1.*(sin((Uhel_TN/180)*pi)))…
   ./(sin((Uhel_TR/180)*pi));

elseif Rozdil < 0;          %Vysílač je výše jak přijímač
   pom_d_1 = 0:Vzdalenost/(Map_l-1):Vzdalenost;
   pom_d_2 = fliplr(pom_d_1);
   Mez_h = Cesta(1,end) + (pom_d_2.*(sin((Uhel_TN/180)*pi)))…
   ./(sin((Uhel_TR/180)*pi));

else Rozdil == 0;           %Vysílač i přijímač ve stejné výšce
   Mez_h(1:Map_l) = Cesta(1,end);
end
```

V daném kódu si lze všimnou pomocné vzdálenosti pom_d_1 popřípadě pom_d_2. Daná vzdálenost nám slouží k výpočtu mezní výšky dle Sinové věty. V podstatě se jedná o stranu dané trojúhelníku v rovině vysílače či přijímače, dle toho, který se nachází výše.

```
if Map_l ==1 || Map_l==2
   Mez_h_2 = 0;
else
   Mez_h_2 = Mez_h(2:end-1);
end
```

Pokud je počet bodů trasy 1 či 2, uvažujeme nulovou mezní výšku překážky. Tuto podmínku zavádíme pouze z důvodu následujícího zápisu kódu, jelikož výška překážky nemá vliv na útlum trasy při délce 1 či 2 body. V dalším zápisu poté vynecháme z mezní výšky krajní body, tedy pozici vysílače a přijímače.

5) Vypočítáme hodnoty výšek překážek.

```
if Map_l ==1 || Map_l==2
   h_prk = 0;
else
   h_prk = Cesta(2:end-1)-Mez_h_2;
end
```

Pokud je počet bodů trasy 1 či 2, uvažujeme nulovou překážku trasy. Tuto podmínku opět zavádíme pouze z důvodu následujícího zápisu kódu, jelikož výška překážky nemá vliv na útlum trasy při délce 1 či 2 body. Výsledná hodnota výšky překážky je pouhým odečtem mezních výšek od výškových bodů trasy, kdy u výškových bodů trasy vynecháme krajní hodnoty, tedy výšku vysílače respektive přijímače.

6) Vypočítáme hodnoty difrakčního parametru *v*.

```
v = h_prk.*sqrt((2*(d1+d2))/(lambda.*d1.*d2));
```

Nyní když známe výšky jednotlivých překážek na trase, vypočítáme difrakční parametr, dle vzorce 3.23, pro každou překážku na trase

7) Zaznamenáme si hodnotu největšího parametru *v* a jeho pozici.

```
if Map_l ==1 || Map_l==2
   v_max = -2;
   pozice_v_max = 0;
else
   v_max = max(v);
   pozice_v_max = find(v==max(v))+1;
   pom_1 = length(pozice_v_max);
%Podmínka jediné hlavní překážky
if pom_1 > 1
   pozice_v_max = pozice_v_max(1);
end
end
```

Pro délku trasy rovnu jedné či dvěma uvažujeme nulový útlumu, z tohoto důvodu položíme hodnotu v_max = -2, zároveň pozici hlavní překážky položíme rovnu 0. Pro ostatní délky trasy nalezneme hodnotu největšího difrakčního parametru v pomocí funkce **`max()`**.

Dále nalezneme pozici daného maxima pomocí funkce **`find()`**. Podmínka pro jediné maximum nám poslouží v případě, že je více překážek reprezentováno stejným difrakčním parametrem *v*. Poté označíme za hlavní překážku tu, která je nejblíže vysílači.

8) Vypočítáme útlum hlavní překážky ze znalosti největšího *v*.

Nyní dle vzorce 3.24 vypočítáme útlum překážky odpovídající danému difrakčnímu parametru. Daný kód realizující výběr a výpočet vypadá následovně.

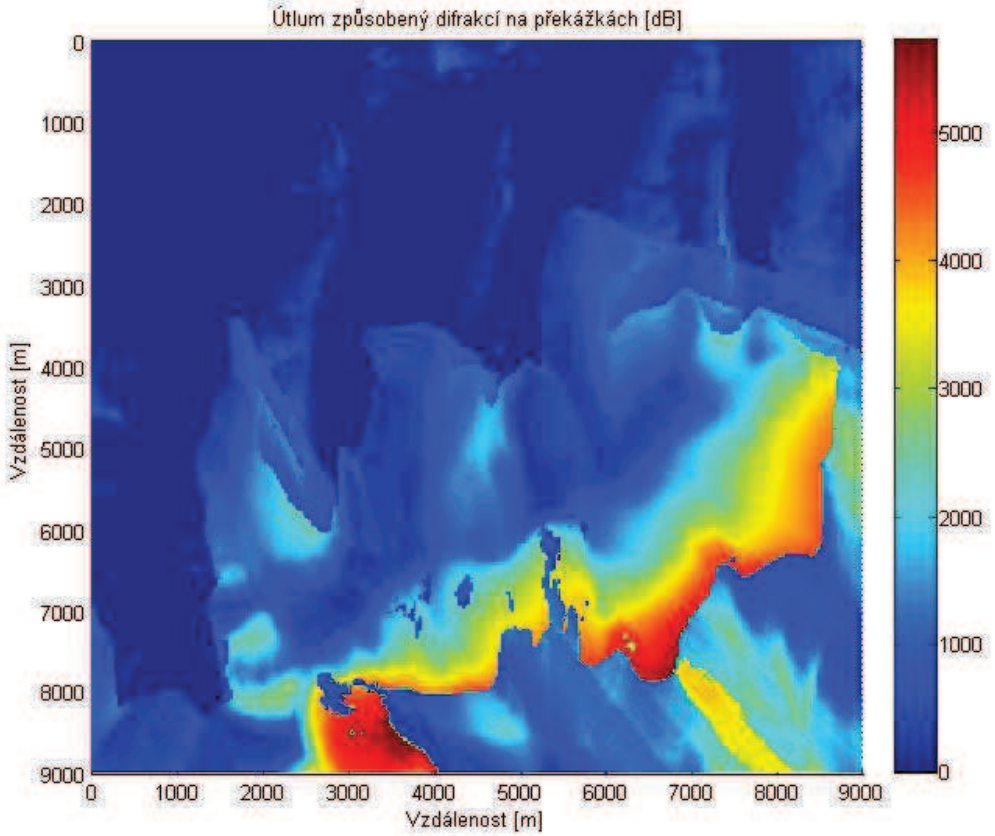
```
if v_max <= -1
   GAX = 0;
elseif v_max > -1 && v_max <= 0
   GAX = 20 * log10(0.5 - 0.62*v_max);
elseif v_max > 0  && v_max <= 1
   GAX = 20 * log10(0.5 * exp(-0.95 * v_max));
elseif v_max > 1  && v_max <= 2.4
   GAX = 20 * log10(0.4-sqrt(0.1184-(0.38-0.1*v_max)^2));
elseif v_max > 2.4
   GAX = 20 * log10(0.225/v_max);
end

Utlum_v_max = GAX;
```

Nyní, když známe pozici hlavní překážky, je možné trasu rozdělit. Rozdělení provedeme jednoduchým vytvořením dvou nových vektorů ze stávajícího vektoru trasy následovně.

```
%Trasa vysílač - hlavní překážka
Cesta_1 = Cesta(1:pozice_v_max)
%Trasa hlavní překážka - přijímač
Cesta_2 = Cesta(pozice_v_max:end)
```

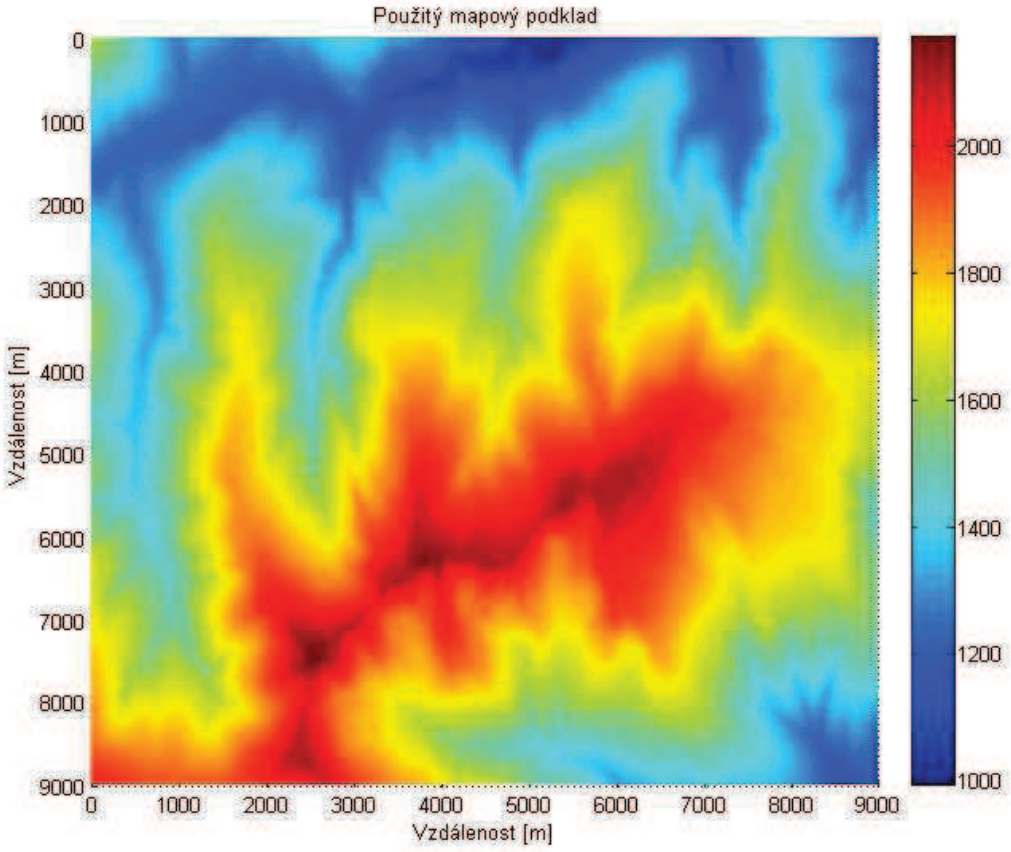
Poté pro každou dílčí trasu opakujeme předešlý postup v krocích 1) – 6). V sedmém kroku, ze všech nalezených difrakčních parametrů *v*, vypočítáme útlum jednotlivých překážek na dílčí trase. Posledním krokem bude výpočet útlumu trasy. Výsledný útlum této trasy je poté dán, jako součet útlumu jednotlivých překážek dílčí trasy.

Pokud tento postup zopakujeme pro všechny dostupné trasy V-P, obdržíme výslednou mapu útlumu daného překážkami. Příklad takové mapy lze vidět na Obr. 5.7.



Obr. 5.7 Útlum způsobený difrakcí signálu na překážkách, souřadnice vysílače 1,1

Na Obr. 5.8 uvádíme mapový podklad použitý při výpočtu pokrytí, aby čtenáři získali lepší představu o vlivu jednotlivých překážek na výsledný útlum.



Obr. 5.8 Mapový podklad použitý při výpočtu (výškové hodnoty v metrech)

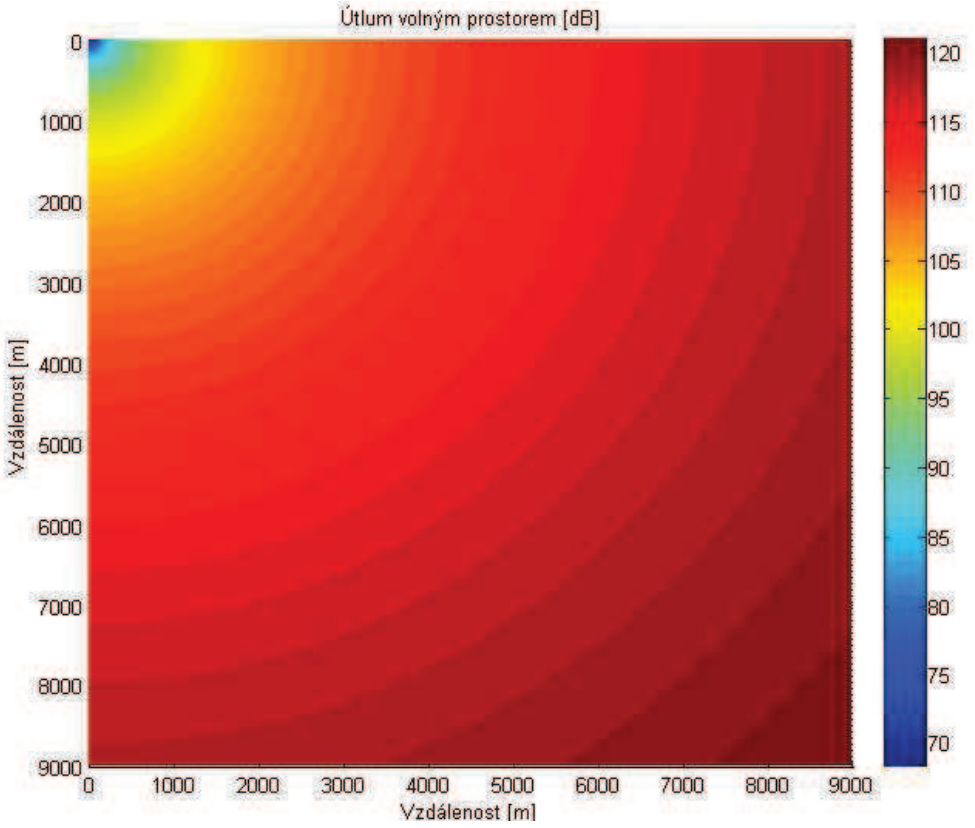
V tomto případě jsme zvolili následující nastavení systému, zisk vysílací antény G_{vys} = 15 dB, zisk přijímací antény $G_{příj}$ = 15 dB, výkon vysílače P_T = 60 dBm, výška vysílače h = 80 m.

Z Obr. 5.7 lze pozorovat, že horský hřeben způsobuje útlum signálu, především směřujícího k jeho odvrácené straně. Díky vysoké výšce vysílače a vysokému výkonu prochází signál určitými místy hřebene, což vede k menšímu útlumu trasy do některých míst za ním.

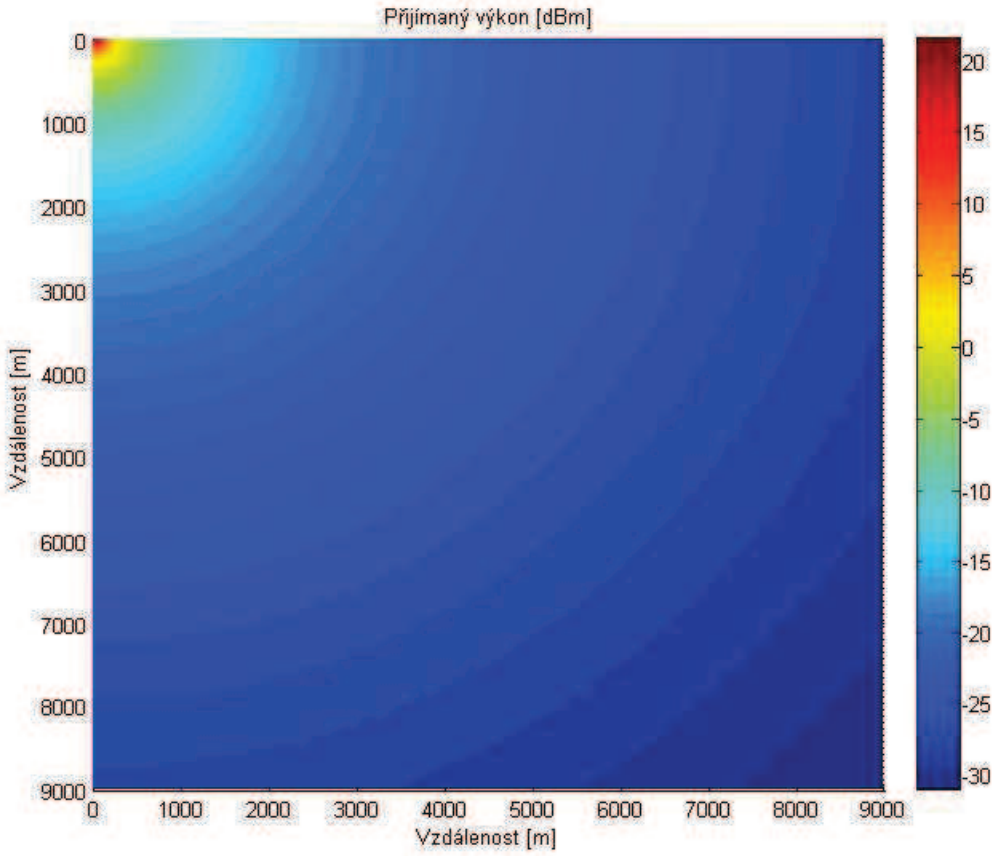
Lze si také povšimnou, že daný výpočet útlumu není ideální, což je způsobeno rozlišením mapy a jejím rozkladem na jednotlivé trasy, jejichž výškové hodnoty nemusí odpovídat skutečné výšce překážky na přímé trase V-P, což vede k částečnému zkreslení. Řešením tohoto problému by mohlo být použití mapy o vyšším rozlišení.

5.4 Výpočet šíření signálu ve volném prostoru

V předchozích bodech jsme popsali výpočet euklidovské vzdálenosti. Pokud známe euklidovskou vzdálenost, můžeme vypočítat hodnotu útlumu šíření volným prostorem dle vzorce 3.18. Tuto hodnotu následně využijeme k výpočtu přijímaného výkonu dle vzorce 3.13. Vypočítané hodnoty lze nalézt na Obr. 5.9 a 5.10.



Obr. 5.9 Útlum signálu volným prostorem, souřadnice vysílače 1,1



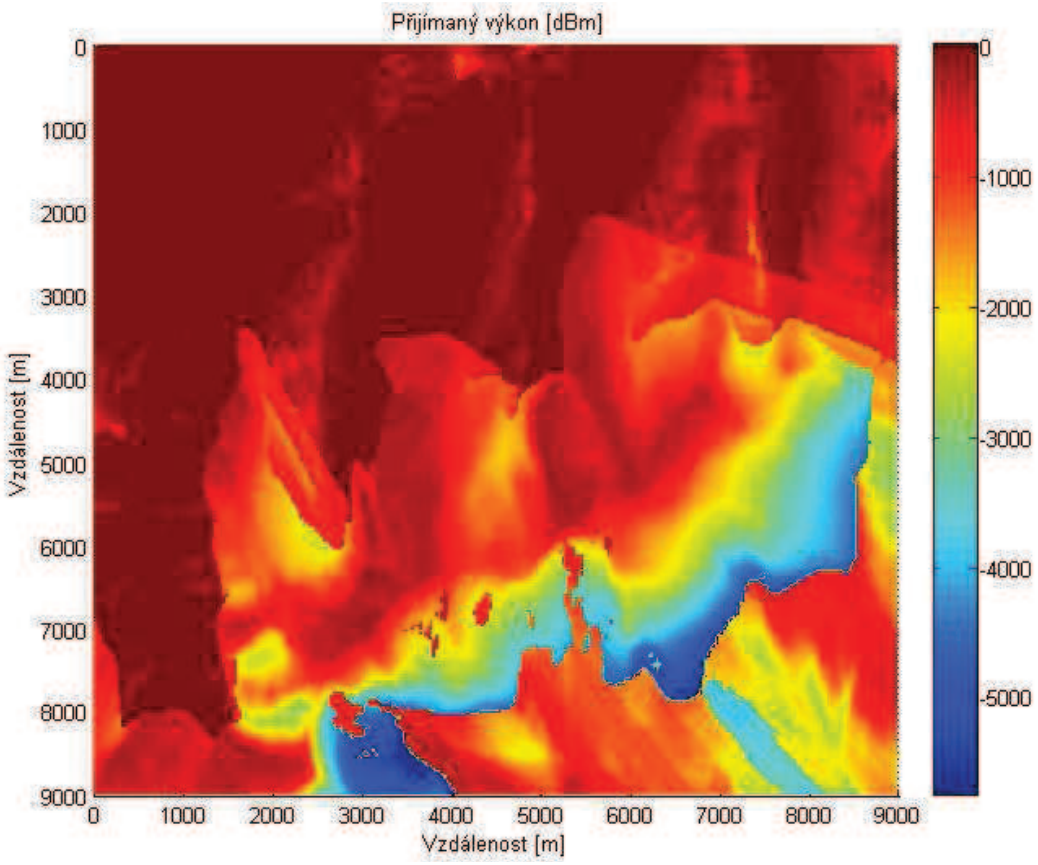
Obr. 5.10 Přijímaný výkon při šíření volným prostorem, souřadnice vysílače 1,1

5.5 Výsledný model výpočtu pokrytí

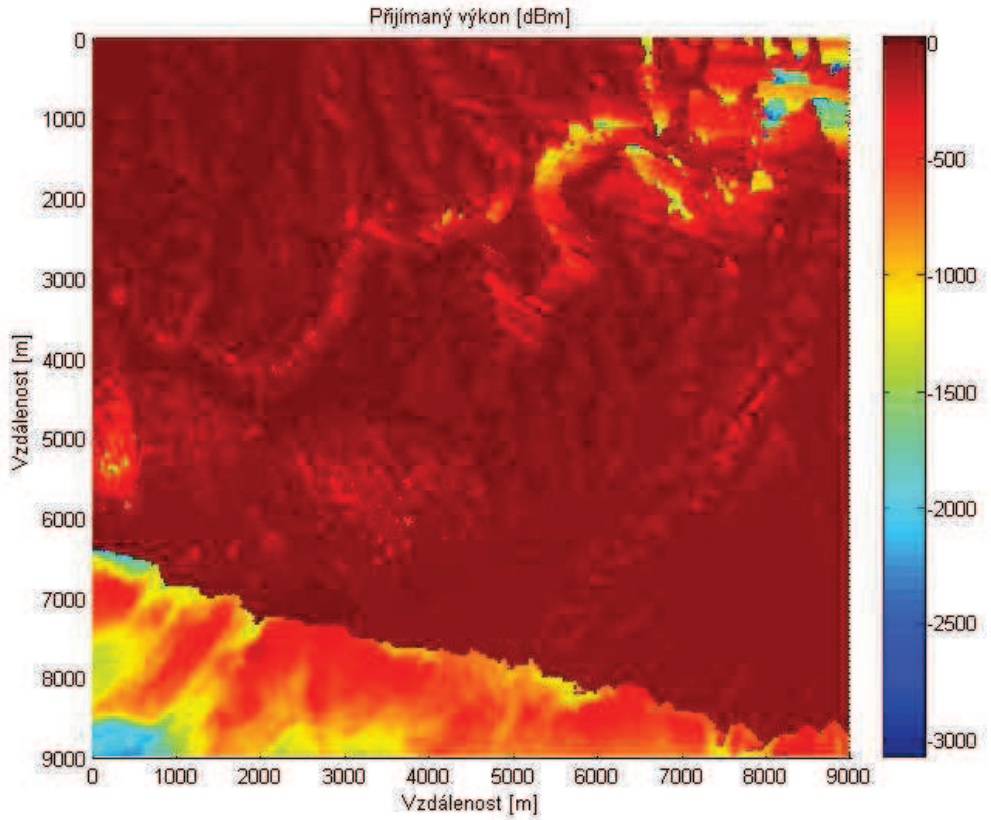
Výsledný model výpočtu pokrytí je složen z načtení mapového podkladu, výpočtů vzdáleností jednotlivých bodů, určení bodů tras, výpočtu útlumu těchto tras, výpočtu šíření signálu volným prostorem a následného konečného výpočtu finální hodnoty přijímaného výkonu, kterému se budeme věnovat v této kapitole.

Posledním krokem je v našem případě výpočet výsledného pokrytí, tedy přijímaného výkonu, který zahrnuje útlum volným prostorem a útlum difrakcí. Daný výkon vypočteme pomocí vzorce 3.25.

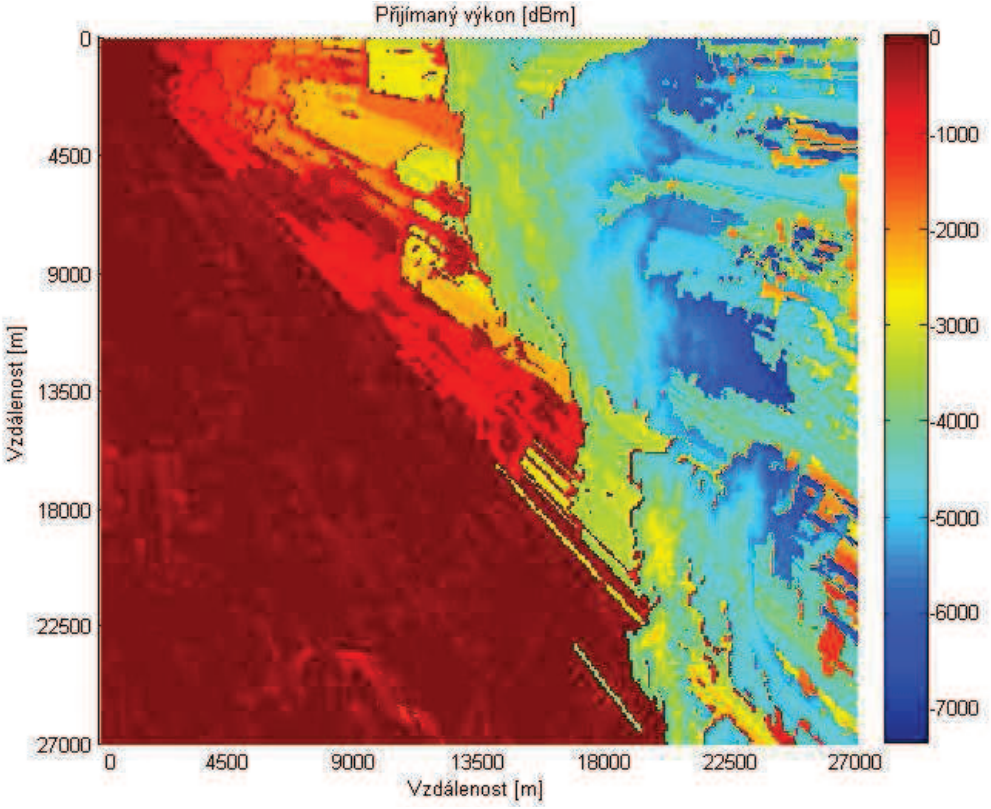
Výslednou mapu pokrytí si lze prohlédnou na Obr 5.11. Dále jsme celý model odzkoušeli pro dvě různé mapy, jednu s rozlišením 30m a druhou s rozlišením 90m. Výsledky pokrytí pro dané mapy jsou na Obr. 5.12 a 5.13. V obou případech jsme umístili vysílač na pozici mapy 1,1.



Obr. 5.11 Výsledná mapa pokrytí, souřadnice vysílače 1,1



Obr. 5.12 Výsledná mapa pokrytí, jako mapový podklad použita část souboru N38W112.hgt, souřadnice vysílače 1,1



Obr. 5.13 Výsledná mapa pokrytí, jako mapový podklad použita část souboru N48E012.hgt, souřadnice vysílače 1,1

5.6 Optimalizační úloha

Jak bylo zmíněno v kapitole 4.1, využili jsme techniku MOPSO. Prvním krokem tedy bylo určení kriteriálních funkcí. V našem případě se jedná o funkce dvě. První charakterizuje výsledné pokrytí. Druhá charakterizuje parametry radiového systému při výpočtu daného pokrytí. Výsledné kriteriální funkce jsme formulovali následovně

$$f_{krit\_1} = \sum_{i=1}^{N} P_{rec_i} \tag{5.5}$$

kde f_{krit_1} je první kriteriální funkce, N je počet bodů mapy, $P_{rec\text{-}i}$ je přijímaný výkon v jednotlivých bodech mapy. Dále

$$f_{krit\_2} = \left(\frac{h_{vysílač}}{h_{vysílač\_max}} + \frac{P_{vysílač}}{P_{vysílač\_max}} \right) \tag{5.6}$$

kde f_{krit_2} je druhá kriteriální funkce, $h_{vysílač}$ je nastavená výška vysílače, $h_{vysílač\_max}$ je maximální nastavitelná hodnota výšky vysílačem, $P_{vysílač}$ je nastavený výkon vysílače, $P_{vysílač\_max}$ je maximální nastavitelná hodnota výkonu vysílače.

Pokud se na daný problém podíváme podrobněji, zjistíme, že první kriteriální funkce odpovídá součtu hodnot vypočítanému námi vytvořeným skriptem na výpočet pokrytí.

Nyní se podíváme na samotnou tvorbu optimalizačního procesu, při kterém provedeme i několik změn ve funkci na výpočet pokrytí.

5.6.1 Změny vzhledem k optimalizaci

První změnou je přesunutí načítání mapy do hlavní funkce optimalizace. Mapu je potřeba načíst jen jednou, další načítání by bylo jen nadbytečným časem, který by program potřeboval k vykonání.

Další změnou je vytvoření funkce z vytvořeného modelu pokrytí tak, aby mohl sloužit k výpočtu hodnot v rámci optimalizačního skriptu.

5.6.2 Hlavní program optimalizace

Prvním krokem je v našem případě nastavení parametrů radiového systému a dané optimalizace. V případě radiového systému zvolíme minimální a maximální přípustné hodnoty zde zmíněných parametrů. Maximální přípustné souřadnice vysílače jsou automaticky dopočítány, pomocí funkce **`length()`**, na základě počtu bodů mapy.

Dále náhodně vygenerujeme první roj částic a jejich rychlosti. K tomu nám poslouží funkce **`randi()`**. Zápis funkce upravíme tak, aby generoval hodnotu v rozmezí minimální a maximální hodnoty parametru systému. Příkladem budiž následující zápis, jež generuje hodnotu výkonu vysílače.

```
P_T = randi([P_T_min,P_T_max]);
```

Poté co vygenerujeme první roj částic, vypočítáme první hodnoty kriteriálních funkcí. Dále následuje výběr ***pbest*** a ***gbest*** jednotlivých částic.

Jako hodnotu ***pbest*** uložíme vygenerované hodnoty, jelikož jiné hodnoty zatím částice nenašla.

Dalším krokem je nalezení nedominovaných řešení prvního roje a jejich uložení do výsledného archivu. Nedominované hodnoty nalezneme pomocí námi vytvořené funkce NDsort. Tato funkce využívá principu metody slow & naive (viz. kapitola 4.2.1). V našem případě zjistíme velikost vstupního archivu pomocí funkce **`size()`**. Poté každou částici tohoto archivu porovnáme s ostatními částicemi pro dominanci. Pokud je některá z částic archivu dominovaná, poznačíme si její umístění v archivu. Nakonec všechna zaznamenaná umístění z archivu vymažeme.

```
N = size(vstup,1);      %Počet řešení v archivu

for pom_1 = 1:N
  for pom_2 = 1:N;

  %Podmínka stejného řešení
  if pom_2 == pom_1;
  else
  %Hledám řešení která jsou dominovaná - označím je
       if fce_1(pom_1) <= fce_1(pom_2) &&...
          fce_2(pom_1) >= fce_2(pom_2)
             [del] = [del pom_1];
       else
       end
  end
  end
end

%Vymažeme řešení označená pro dominanci
vstup(del,:) = [];
vystup =  vstup;
```

Nyní vlastníme první nedominovaná řešení. Tyto řešení uložíme do výsledného archivu. K tomu využijeme funkci **`vertcat()`**, která slouží k vertikálnímu skládání vektorů.

```
archiv = vertcat(archiv,vystup)
```

Z těchto řešení vybereme hodnotu ***gbest*** pro každou částici. Jelikož řešíme dvojkriteriální problém, využili jsme k výpočtu vzdálenosti jednotlivých nedominovaných řešení od částice funkci **`pdist2()`**.

```
dxy = pdist2([PARTICLES(:,5),PARTICLES(:,6)]...
             ,[archiv(:,5),archiv(:,6)],'euclidean');
```

Nyní, pomocí funkce **`min()`**, nalezneme nejbližší řešení dané částici a toto řešení uložíme jako ***gbest*** dané částice.

```
%Nalezení nejmenší vzdálenosti a jejího indexu
 [~,ind] = min(dxy,[],2);
%Výběr řešení s nejmenší vzdáleností od částice jako gbest
 GBEST = vystup(ind,:);
```

Následně se dostáváme do iterační smyčky optimalizačního procesu. Prvním krokem je výpočet nových rychlostí částic v této iteraci, poté je na řadě výpočet nových poloh částic v této iteraci. Po výpočtu nových poloh je nutné provést kontrolu, zda se částice nachází ve vyhrazeném prostoru. V našem případě to řešíme jednoduchou podmínkou. Pokud poloha částice překračuje hranice prostoru, je nahrazena právě hraniční polohou v daném směru.

Nové hodnoty částic uložíme do archivu a provedeme jeho třízení.

Dále zkontrolujeme, zda je ***pbest*** částic již první nedominované řešení či nikoli. K tomuto účelu jsme vytvořili vektor nulových hodnot a sadu podmínek, jenž slouží ke kontrole, zdali je už dané řešení první nedominované. Pokud ne, porovnáme novou hodnotu částice s jejím ***pbest*** na dominanci a určíme, zda bude ***pbest*** nahrazeno či nikoli.

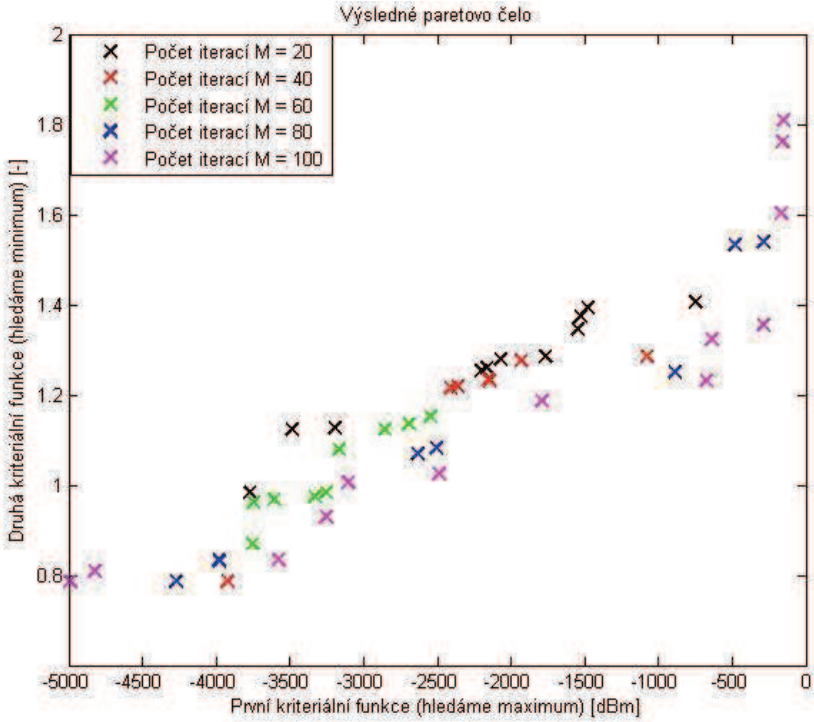
```
% Vektor pro kontrolu
Lock = zeros(N,1);
for pom_1 = 1:N    %Pro všechny částice
if Lock(pom_1) == 1
%Pbest je první nedominované - navždy
else
%Porovnáme pbest s novou částicí pro dominanci

%Nové řešení dominuje staré - uložíme ho jako pbest
  if PARTICLES(pom_1,5) >= archiv_1(pom_1,5) && ...
     PARTICLES(pom_1,6) <= archiv_1(pom_1,6)
       PBEST(pom_1,:) = PARTICLES(pom_1);

%Staré řešení dominuje nové - ponecháme ho jako pbest
  elseif PARTICLES(pom_1,5) <= archiv_1(pom_1,5) && ...
         PARTICLES(pom_1,6) >= archiv_1(pom_1,6)
            PBEST(pom_1,:) = PARTICLES(pom_1);
%Nedominantní vůči sobě
else
  Lock(pom_1) = 1;
  PBEST(pom_1,:) = PARTICLES(pom_1,:);
end
end
end
```

Předposledním krokem iterační smyčky je určení hodnot ***gbest*** pro jednotlivé částice. Tento proces je opět řešen nalezením nejbližšího řešení, z archivu nedominovaných řešení, dané částici.

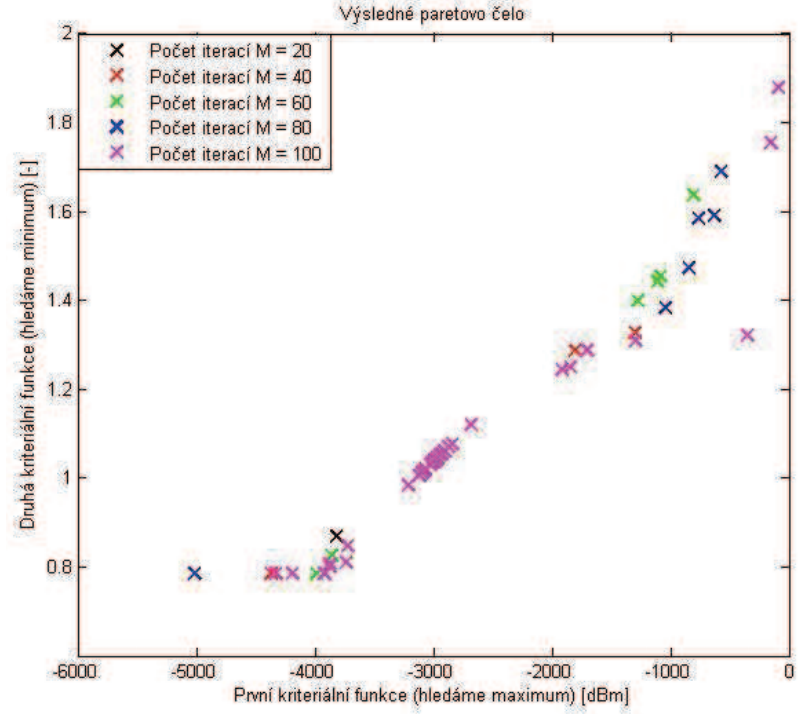
Na úplný konec uložíme výsledné hodnoty z této iterace do pomocného archivu, pro účely porovnání v následující iteraci.

5.6.3 Výsledek optimalizačního procesu

Jak bylo řečeno v kapitole 4.2, výsledkem optimalizačního procesu je soubor nedominovaných řešení, tzv. Paretovo čelo. Příklad zobrazení Paretova čela pro námi vytvořený algoritmus lze nalézt na Obr. 5.14 a 5.15.



Obr. 5.14 Výsledné Paretovo čelo, optimalizace s fixní inerciální váhou



Obr. 5.15 Výsledné Paretovo čelo, optimalizace s měnící se inerciální váhou

5.7 Úpravy kódu pro snížení výpočetní náročnosti

Jelikož optimalizace je dosti náročným výpočetním úkonem, je zapotřebí, aby daný kód pro výpočet hodnoty elektrické intenzity, výkonu a dalších hodnot, mohl být vykonán v co nejkratším čase. To zajistíme nahrazením časově náročných úkonů, jako jsou cyklické operace, za jednodušší kód – tzv. vektorizací, dále také použitím optimalizovaných funkcí programu. Příkladem budiž následující kód, u kterého použijeme funkce **tic** a **toc**, ke zjištění jeho časové náročnosti.

```
% Zadané parametry pro výpočet
Rozmer = 3601;
dx = 1;
dy = 1;
Vzdalenost = zeros(Rozmer);
x0 = 1;
y0 = 1;

% Porovnávaný kód č.1
tic
for x = 1:Rozmer
  for y = 1:Rozmer
       Vzdalenost(x,y) = sqrt(dx*(x0-x)^2+dy*(y0-y)^2);
  end
end
toc
```

Výsledný čas potřebný k vykonání kódu je $t = 0{,}884394$ s . Daný kód nyní porovnáme s jeho náhradou, která nevyužívá funkcí **for**.

```
% Porovnávaný kód č.2
tic
Vzdalenost2 = sqrt(repmat((1:Rozmer),Rozmer,1)-x0).^2 +…
               repmat((1:Rozmer),Rozmer,1)-y0).^2);
toc
```

Výsledný čas k vykonání náhradního kódu je $t = 0{,}845814$ s. Rozdíl v daných zápisech je tedy přibližně $\Delta t = 0{,}04$ s. Tento docela malý rozdíl ovšem může znamenat velký rozdíl ve výsledném kódu. Například pro sto běhů optimalizačního algoritmu dosahuje rozdíl $\Delta t = 4$ s.

Pro přesnější porovnání nyní vyzkoušíme běh obou kódů pro různé hodnoty rozměru mapy. V prvním případě pro rozměr velikosti 1201, který je jedním z rozměrů námi používaných map. V tomto případě je $t_{kod} = 0{,}112959\ s$ a $t_{nahrada} = 0{,}096168\ s$.

Druhým případem bude rozměr velikosti 100, například kdybychom chtěli vybrat jen určitou část mapy. V tomto případě jsou výsledné časy $t_{kod} = 795\ \mu s$ a $t_{nahrada} = 1{,}087\ ms$.

Lze tedy usoudit, že u méně náročných operací je vhodné využít cyklické operace.

Nyní se zaměříme na použití „vlastních“ funkcí matlabu. Využijeme opět příkladu s výpočtem Euklidovské vzdálenosti. Nyní ovšem využijeme funkce pdist2

```
tic
Euklid = pdist2([x_vys,y_vys],[sx',sy'],'euclidean');
toc
```

kde pro rozměr velikosti 1201 dostáváme $t_{kod} = 0{,}181976\ s$, *pro* rozměr velikosti 100 je $t_{kod} = 75{,}456\ ms$. *Z* daných hodnot lze zjistit, že vnitřní funkce nemusí mít vždy dosahovat nejlepších možných výsledků. Navíc je u ní zapotřebí výpočtu dalších hodnot, což by výslednou dobu výpočtu prodloužilo. Výhodou snad může být, že zápis dané funkce je jednodušším a již odzkoušeným řešením.

V následujících kapitolách 5.7.1 – 5.7.3 jsou popsány nečastější způsoby zrychlení kódu v programu Matlab. Podrobnější popis daných způsobů a jiných řešení viz. literatura [22],[23],[24].

5.7.1 Alokace paměti

Jedním z nejjednodušších způsobů zrychlení kódu je alokace paměťového prostoru. Tím zajistíme, aby daná proměnná měla v paměti vyhrazenou určitou část prostoru pro svoji potřebu. Pokud zajistíme, aby program zapisoval data do alokované paměti, zkrátíme tím pracovní proces o dobu, kterou by strávil hledáním volného místa v paměti a kopírováním dat na toto místo.

Příkladem budiž následující kód, smyčka, ve které probíhá jednoduchá matematická operace. Budeme uvažovat dvě možnosti.

1) Pro výsledné hodnoty nebyl alokován paměťový prostor.

```
tic
for l = 1:N
archiv(l) = 5*l;
end
t1 = toc                          %Čas smyčky
```

2) Pro výsledné hodnoty byl alokován paměťový prostor.

```
tic
archiv_2 = zeros(N,1);            %Alokace
t2 = toc                          %Čas potřebný k alokaci
tic
for l = 1:N
archiv_2(l) = 5*l;
end
t3 = toc                          %Čas smyčky
t4 = t2 + t3;                     %Celkový čas kódu
```

Výsledné hodnoty časů t1, t2, t3, t4 si lze prohlédnout v Tabulce 1, kdy jsme obě možnosti vyzkoušeli pro různou velikost N.

Tab. 5.1 Porovnání výpočetní časové náročnosti

N [-]	100	1000	10 000	100 000
t1 [s]	$127{,}39 \cdot 10^{-6}$	$578{,}29 \cdot 10^{-6}$	$5{,}10 \cdot 10^{-3}$	$48{,}60 \cdot 10^{-3}$
t2 [s]	$12{,}29 \cdot 10^{-6}$	$13{,}54 \cdot 10^{-6}$	$58{,}45 \cdot 10^{-6}$	$628{,}71 \cdot 10^{-6}$
t3 [s]	$3{,}14 \cdot 10^{-6}$	$10{,}12 \cdot 10^{-6}$	$79{,}61 \cdot 10^{-6}$	$770{,}00 \cdot 10^{-6}$
t4 [s]	$15{,}43 \cdot 10^{-6}$	$23{,}67 \cdot 10^{-6}$	$138{,}08 \cdot 10^{-6}$	$1{,}40 \cdot 10^{-3}$

Lze si povšimnout, že samotná alokace může zabrat více času, než řešení dané úlohy. Na druhou stranu, celkový čas alokace a řešení je výrazně menší, než čas potřebný k vyřešení úlohy bez alokace.

5.7.2 Vektorizace

Vektorizace je úprava kódu, při kterém se snažíme náš kód upravit tak, aby pracoval paralelně s daty v určené struktuře, namísto použití smyček. Vektorizace daného kódu je výhodná z několika důvodů. Vektorizovaný kód má častěji lepší performanci a je snadnější se v něm vyznat. Ve většině případů je také kratší, což znamená menší možnost výskytu chyb.

Příkladem budiž dva následující kódy, první využívající cyklické operace, druhý vektorizovaný.

1) Cyklická operace

```
tic
for i = 1:100
    x(i) = ((i+1)/i)^2;
end
tcykl = toc
```

2) Vektorizovaný kód

```
tic
i = 1:100;
x = ((i+1)./i).^2;
tvekt = toc
```

Výsledné časy *tcykl = 1,11 ms* a *tvekt = 38,34 μs*. Lze tedy pozorovat, že vektorizovaný kód potřebuje mnohem menší čas ke svému vykonání.

Umístění tečky před daný operátor (operátory /,*,^) nám zajistí, že daná operace je prováděna s prvky se stejným umístěním v dané matici či vektoru.

5.7.3 Použití vlastních funkcí Matlabu

Většina vlastních funkcí je vektorizovaná, což vede k jejich lepší performanci. Navíc jsou tyto funkce odzkoušeny, což znamená, že by nemělo docházet k chybám při jejich správném zápisu.

5.8 Výpočetní náročnost výsledného kódu

V této podkapitole se zaměříme na testování výpočetní náročnosti vytvořených programů, jelikož mnohé jistě zajímá čas běhu vytvořených skriptů.

Prvně se zaměříme na vytvořený model výpočtu pokrytí území a otestujeme jeho náročnost při výpočtu několika map o různých rozměrech. Výsledné časy simulací lze nalézt v Tab. 5.2.

Tab. 5.2 Výsledné časy simulací výpočtu pokrytí pro různé velikosti mapy

Číslo simulace	Rozměr mapy	Rozlišení mapy	Celkový čas simulace
1	10 x 10 bodů	1 arc sekunda (30 metrů)	12,172 s
2	50 x 50 bodů		12,823 s
3	100 x 100 bodů		15,464 s
4	200 x 200 bodů		36,884 s
5	300 x 300 bodů		95,477 s
6	10 x 10 bodů	3 arc sekundy (90 metrů)	6,424 s
7	50 x 50 bodů		7,773 s
8	100 x 100 bodů		13,211 s
9	200 x 200 bodů		35,329 s
10	300 x 300 bodů		77,355 s

Poté se zaměříme na danou úlohu optimalizace, kterou vyzkoušíme pro různý počet iterací, různý počet částic a různou velikost vybrané mapy. Výsledné časy simulací lze nalézt v Tab. 5.3, 5.4 a 5.5.

Tab. 5.3 Výsledné časy běhu optimalizačního skriptu pro různý počet částic

Číslo simulace	Rozměr mapy	Počet iterací	Počet částic	Celkový čas simulace
1	11 x 11 bodů	100	10	96,795 s
2			20	188,585 s
3		10	30	36,703 s
4			40	41,447 s
5			50	49,324 s

Tab. 5.4 Výsledné časy běhu optimalizačního skriptu pro různý počet iterací

Číslo simulace	Rozměr mapy	Počet iterací	Počet částic	Celkový čas simulace
1	11 x 11 bodů	10	10	15,186 s
2		20		23,545 s
3		30		34,758 s
4		40		40,447 s
5		50		49,204 s

Tab. 5.5 Výsledné časy běhu optimalizačního skriptu pro různé rozměry mapy

Číslo simulace	Rozměr mapy	Počet iterací	Počet částic	Celkový čas simulace
1	10 x 10 bodů	10	10	13,562 s
2	50 x 50 bodů			229,865 s
3	100 x 100 bodů			1 004,949 s
4	150 x 150 bodů			2 199,498 s
5	200 x 200 bodů			4 832,133 s

Pro úplnost ještě uvedeme technické specifikace počítače použitého k simulacím. Ty jsou následující

- Procesor: Intel core 2 duo T5870, 2.00 GHz
- Paměť: 4 GB DDR2 RAM, 800 MHz
- Systém: Windows Vista Home Premium, 32 bit

nutné je přitom podotknout, že se jedná o notebook. Většina stolních počítačů dnes disponuje vyšším výpočetním výkonem a pravděpodobně by řešení daných úloh zvládli v mnohem kratším čase.

Všechny tyto simulace slouží pouze k přibližnému přehledu. Celkový čas se může lišit podle rozsahu hodnot, kterých dosahují kriteriální funkce či členitosti terénu vybrané mapy. Také je to způsobeno tím, že optimalizace je náhodný proces.

5.9 Příklad

Nastavení radiového systému je následující

$f = 2100$ MHz

$G_T = 15$ dB

$G_R = 15$ dB

$P_{T\_min} = 30$ dBm

$P_{T\_max} = 60$ dBm

$h_{vysilac\_min} = 10$ m

$h_{vysilac\_max} = 35$ m

Nastavení optimalizace je následující

$N = 10$

$M = 20$

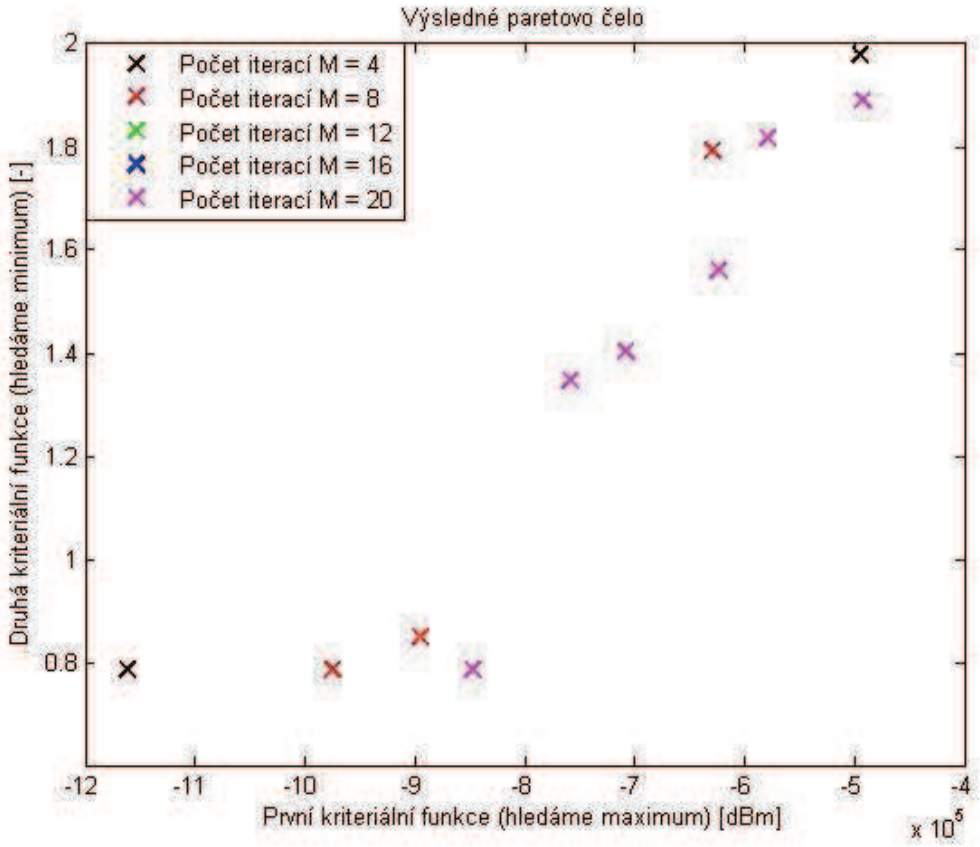
$MM = 50$

w_min = 0,4

w_max = 0,7

inerciální váha – postupně se snižující

Použita je mapa 100x100 bodů, o rozlišení 90 m



Obr. 5.16 Výsledné paretovo čelo pro dané nastavení

Vybrali jsme tři body, na Paretově čele, dosahující následujících parametrů

1) Pro kraj Paretova čela, kdy obě kriteriální funkce dosahují svého minima

   x = 54, y = 31, h_{vys} = 10 m, P = 30 dBm

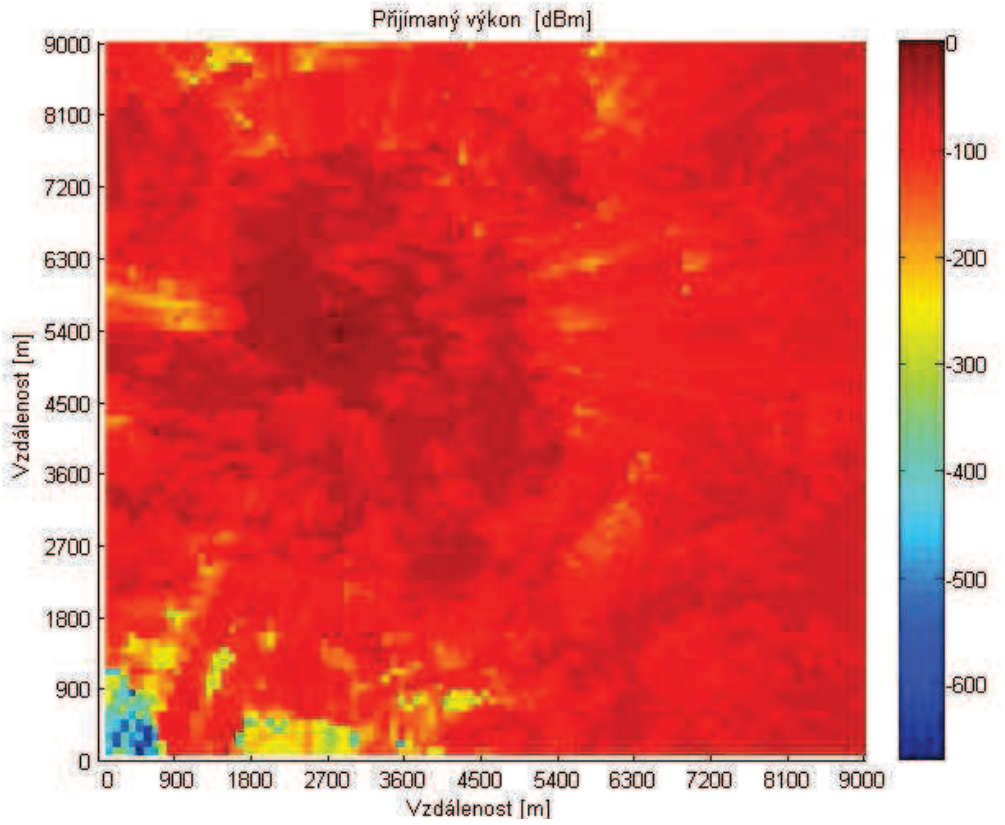
2) Pro střed Paretova čela

   x = 33, y = 18, h_{vys} = 15,3612 m, P = 57,7515 dBm

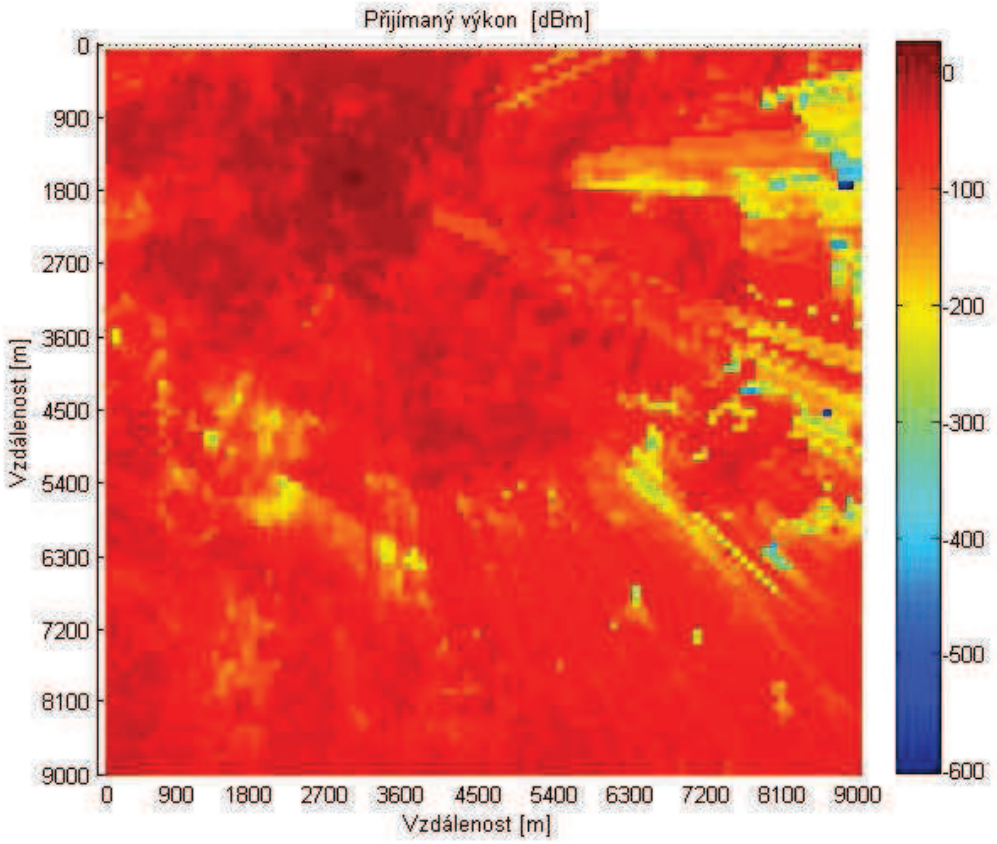
3) Pro kraj Paretova čela, kdy obě kriteriální funkce dosahují svého maxima

   x = 40, y = 18, h_{vys} = 35 m, P = 53,4589 dBm

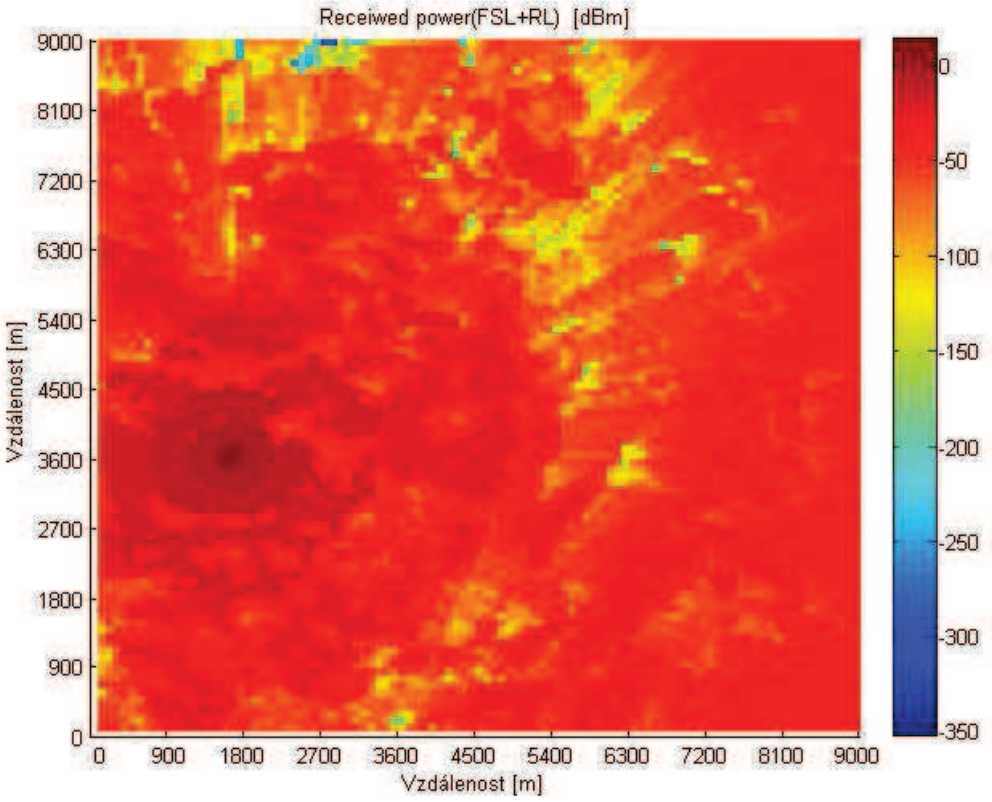
Výsledné mapy pokrytí pro tyto případy lze nalézt na Obr. 5.17, 5.18 a 5.19.



Obr. 5.17 Zobrazení mapy pokrytí, případ 1)



Obr. 5.18 Zobrazení mapy pokrytí, případ 2)



Obr. 5.19 Zobrazení mapy pokrytí, případ 3)

Pozice zobrazeneé v mapě mohou být graficky odlišné, což je způsobeno vytvořením výsledných obrázků pomocí funkce **`surfc()`** .

6 ZÁVĚR

Úkolem této diplomové práce bylo vytvoření modelu pokrytí území v programu Matlab, jež bude využívat optimalizačního algoritmu k nalezení nejlepšího řešení z hlediska výsledného pokrytí a zadaných parametrů systému. Konečným výstupem této diplomové práce jsou nakonec dvě sady skriptů.

První, sloužící k výpočtu pokrytí nad daným územím. V této sadě skriptů rozhoduje o nastavení radiového systému a umístění vysílače sám uživatel. Při tvorbě těchto skriptů jsme se inspirovali Leeho modelem šíření, který uvažuje tři způsoby útlumu signálu. A to útlum způsobený rozdílem efektivní výšky vysílače, útlum způsobený difrakcí a útlum šířením volným prostorem. Celkově je tento model šíření založen na dvou z předchozích způsobů. Těmi jsou šíření signálu volným prostorem a útlum signálu způsobený difrakcí na překážce.

Pro výpočet difrakce jsme použili metodu využívající výpočtu Fresnell-Kirchoffova difrakčního parametru, ze kterého následně určí útlum překážky. Výsledná hodnota přijímaného výkonu či intenzity EM pole je poté určena jako hodnota při šíření volným prostorem, kterou zmenšíme o daný útlum způsobený difrakcí překážek. Konečným výstupem těchto skriptů je zobrazení mapy pokrytí nad vybranou mapou. Je nutné podotknout, že komerčně dostupné modely obvykle uvažují další jevy, jež mají vliv na hodnotu přijímané hodnoty. Z tohoto důvodu s nimi nelze námi vytvořený model pokrytí srovnávat.

Druhá sada skriptů využívá optimalizačních algoritmů k nalezení optimálního řešení z hlediska umístění vysílače a nastavení radiového systému. V tomto případě nastavuje uživatel pouze parametry optimalizační úlohy, minimální a maximální přípustné hodnoty, jichž mohou parametry radiového systému dosahovat. Pro výpočet optimalizační úlohy jsme využili metodu MOPSO vzhledem k jejímu jednoduchému přizpůsobení danému problému.

V případě obou sad je skript Main.m hlavním skriptem, který slouží k nastavení daných simulací a jejich spuštění. Dané skripty lze nalézt na přiloženém kompaktním disku na vnitřní straně desek práce. Zároveň lze nalézt kód druhé sady skriptů v přílohách této práce. Kód první sady neuvádíme, jelikož je obsažen ve druhé sadě jako funkce, sloužící k výpočtu hodnot jedné z kriteriálních funkcí. V přílohách této práce lze také nalézt všechny užité funkce programu Matlab, kterých bylo využito při tvorbě výsledných skriptů.

V úplném závěru bych rád zmínil několik návrhů, jenž by mohly přispět k vylepšení daného modelu. Prvním je aplikace některé ze složitějších metod pro výpočet difrakce, čímž bychom dosáhli lepších výsledků z hlediska výsledného pokrytí. Dále se pokusit daný model obohatit o vlivy, ke kterým dochází při šíření signálu v reálném světě, např. odrazy signálu od objektů. Tímto bychom se přiblížili výsledku, jaký by nastal při šíření signálu v podmínkách reálného světa. Nakonec úprava tzv. nedominatního třízení na „rychlejší“ metodu a optimalizačního algoritmu tak, aby byl schopen řešit n-kriteriální problémy.

LITERATURA

[1] PODOBNIKAR, Tomaz. *Methods for visual quality assessment of a digital terrain model*. Dostupné z: http://sapiens.revues.org/738

[2] Shuttle Radar Topography Mission: Mission Summary. USGS. [online]. [cit. 2013-12-18]. Dostupné z: http://srtm.usgs.gov/Mission/missionsummary.php

[3] Shuttle Radar Topography Mission: Interferometry and SRTM - An Overview. USGS. [online]. [cit. 2013-12-18]. Dostupné z: http://srtm.usgs.gov/data/interferometry.php

[4] Shuttle Radar Topography Mission: Final Data Coverage Maps. USGS. [online]. [cit. 2013-12-18]. Dostupné z: http://srtm.usgs.gov/data/coveragemaps.php

[5] Spaceoffice.nl: InSAR en SRTM. USGS. [online]. [cit. 2013-12-18]. Dostupné z: http://www.spaceoffice.nl/nl/Satelliettoepassingen/Technologie/RADAR/InSAR%20en%20SRTM/

[6] Gisat: Radarová interferometrie. GISAT. [online]. [cit. 2013-12-18]. Dostupné z: http://www.gisat.cz/content/cz/dpz/zpracovani-dat/radarova-interferometrie

[7] ZDRAŽIL, Vladimír RNDR. PH.D. SKRIPTUM *FYZIKA 2: Vlnění*. 1. vyd. Brno: VUTIUM, 2000.

[8] NOVÁČEK, Zdeněk Doc. Ing. CSc. *SKRIPTA: Elektromagnetické vlny, antény a vedení*. 143 s.

[9] PECHAČ, Pavel. *Šíření vln v zástavbě: modely pro plánování mobilních radiových systémů*. Praha, 2005, 112 s. ISBN 80-7300-186-1.

[10] Calculate Fresnel Zone. YOUNG, Stuart. *Broadband.buyer.co.uk* [online]. [cit. 2013-12-18]. Dostupné z: http://www.broadbandbuyer.co.uk/Advice/Article.asp?TextID=687

[11] RAIDA, Zbyněk et al. *Multimediální učebnice: Elektromagnetické vlny, Mikrovlnná technika*. Dostupné z: http://www.urel.feec.vutbr.cz/~raida/multimedia/index.php

[12] RAIDA, Zbyněk et al. *Mikrovlnné struktury z netradičních materiálů*. Vysoké učení technické v Brně, 2011. ISBN 978-80-214-4419-5.

[13] READGHT: Import/download NASA SRTM data files (.hgt). BEAUDUCEL, François. *MatlabCentral* [online]. [cit. 2013-12-18]. Dostupné z: http://www.mathworks.com/matlabcentral/fileexchange/36379-readhgt-importdownload-nasa-srtm-data-files-hgt

[14] J. Kennedy and R. C. Eberhart, Particle swarm optimization, Proc. of the 1995 IEEE Int. Conf. Neural Networks, vol.4, pp.1942-1948, 1995.

[15] Y. Shi and R. Eberhart., "A modified particle swarm optimizer", In Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on, pages 69–73. IEEE, 2002.

[16] Inertia Weight Strategies in Particle Swarm Optimization. 640 - 646. Dostupné z: http://www.softcomputing.net/nabic11_7.pdf

[17] MONTES DE OCA, Marco A. IRIDIA-CODE, Universit´e Libre de Bruxelles (U.L.B.). *Particle Swarm Optimization: Introduction*. Dostupné z: http://iridia.ulb.ac.be/~mmontes/slidesCIL/slides.pdf

[18] DEB, K. Multi-Objective Optimization using Evolutionary Algorithms. Chichester: Wiley, 2001.

[19] FORTIN, Dominique a David HALL. *Test Radio Receivers With Recorded Signals*. 2008. Dostupné z: http://mwrf.com/test-and-measurement/test-radio-receivers-recorded-signals

[20] LEE, William C.Y. *Wireless & Cellular Telecommunications*. Third edistion. ISBN 0-07-143686-3.

[21] THE ITU RADIOCOMMUNICATION ASSEMBLY. *RECOMMENDATION ITU-R P.526-6: PROPAGATION BY DIFFRACTION*.

[22] *Speedup tricks*. Dostupné z: http://yagtom.googlecode.com/svn/trunk/html/speedup.html#1

[23] ACKLAM, Peter J. *M A T L A B array manipulation tips and tricks*. 2003. Dostupné z: http://home.online.no/~pjacklam/matlab/doc/mtt/doc/mtt.pdf

[24] MATHWORKS. *Code performance: MATLAB & Simulink*. Dostupné z: http://www.mathworks.com/help/matlab/code-performance.html

[25] VAN DEN BERGH, Frans. *An Analysis of Particle Swarm optimizers*. University of Pretoria, 2006.

[26] YANG, Chunming a Dan SIMON. *A New Particle Swarm Optimization Technique*. Electrical and Computer Engineering Department Cleveland State University. ISBN 0-7695-2359-5. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.399.5004&rep=rep1&type=pdf

[27] WEISE, Thomas. *Global Optimization Algorithms: Theory and Application*. 2009. Dostupné z: http://www.it-weise.de/projects/book.pdf

[28] ABIDO, M. A. *Two-Level of Nondominated Solutions Approach to Multiobjective Particle Swarm Optimization*. Dhahran 31261, Saudi Arabia: Electrical Engineering Department King Fahd University of Petroleum and Minerals. Dostupné z: http://www.genetic-programming.org/hc2007/05-Abido/Abido-Paper3.pdf

[29] Particle Swarm Optimization. [online]. [cit. 2014-05-20]. Dostupné z: http://www.swarmintelligence.org/

[30] Artificial Intelligence: Particle Swarm Optimization. MCCAFFREY, James. [online]. [cit. 2014-05-20]. Dostupné z: http://msdn.microsoft.com/en-us/magazine/hh335067.aspx

[31] SHEIKH, Asrar U.H. *Wireless Communications: Theory and Techniques*. Kluwer Academic Publishers, 2004. ISBN 1-4020-7621-5.

[32] Subscript from linear index: MATLAB ind2sub. MATHWORKS. [online]. [cit. 2014-05-20]. Dostupné z: http://www.mathworks.com/help/matlab/ref/ind2sub.html

[33] EBRAHIMIPOUR, V., A. HAERI, M. SHEIKHALISHAHI a S.M. ASADZADEH. Application of Multi-Objective Particle Swarm Optimization to Solve a Fuzzy Multi-Objective Reliability Redundancy Allocation Problem. [online]. 2012 [cit. 2014-05-20]. DOI: 10.5923/j.safety.20120102.02. Dostupné z: http://article.sapub.org/10.5923.j.safety.20120102.02.html

[34] ANDERSON, Christopher R., Jeffrey H. REED ,Michael BUEHRER et al. *Introduction to Ultra Wideband Communication Systems*. ISBN 0-13-148103-7.

[35] AKKAŞH, Cem. *Methods for Path loss Prediction*. Växjö University, 2009. Dostupné z: http://lnu.diva-portal.org/smash/get/diva2:273839/FULLTEXT01.pdf

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

A.L.S.	Airborne Laser Scanning
B	Magnetická indukce
c	Rychlost světla
C_1	Konstanta
C_2	Konstanta
CD	Crowding Distance
d	Vzdálenost
D	Elektrická indukce
DEM	Digitální elevační model
DTM	Digitální model terénu
λ	Vlnová délka
e	Eulerovo číslo
E	Intenzita elektrického pole
EM	Elektro-magnetická/ý
ε	Permitivita
ε_0	Permitivita vakua
ε_r	Relativní permitivita
f	Frekvence
FSL	Free Space Loss (Útlum volným prostorem)
Φ	Magnetický indukční tok
γ	Měrná elektrická vodivost (Konduktivita)
G_P	Zisk přijímací antény
G_V	Zisk vysílací antény
gbest	Nejlepší řešení nalezené celým rojem
h	Výška překážky
H	Intenzita magnetického pole
j	Imaginární jednotka
l	Délka
μ	Permeabilita
μ_0	Permeabilita vakua
μ_r	Relativní permeabilita

n	Číslo Fresnelovy zóny
ψ	Elektrický indukční tok
P_V	Výkon vysílače
pbest	Nejlepší řešení nalezené danou částicí
Q	Elektrický náboj
r_1	Náhodné číslo (0,1)
r_2	Náhodné číslo (0,1)
ρ	Objemová hustota elektrického náboje
r_{ON}	Poloměr Fresnelovy zóny
s	Sekunda
S	Plocha
SRTM	Shuttle Radar topograpfy mission
t	Čas
v	Fresnell-Kirchoffův difrakční parametr
v_t	Rychlost částice v této iteraci
v_{t+1}	Rychlost částice v následující iteraci
V-P	Vysílač – Přijímač
w	Inerciální váha
x_t	Poloha částice v předchozí iteraci
x_{t+1}	Poloha částice v následující iteraci

SEZNAM PŘÍLOH

A POUŽITÉ FUNKCE PROGRAMU MATLAB

asind
cell2mat
clc
clear
close
double
else
elseif
end
exp
find
fliplr
for
if
ind2sub
isempty
isnan
legend
length
linspace
load
max
min
open
pdist2
plot
rand
randi
save
size
sortrows
sqrt
sub2ind
sum
surfc
title
vertcat
xlabel
ylabel
zeros

B VÝSLEDNÝ KÓD VYTVOŘENÝCH FUNKCÍ

B.1 Main.m

```
    clear all;
    close all;
    clc;

%% ================================================================
%                      Zobrazení souvisejících funkcí
%================================================================

   %Související funkce s optimalizací
    open('NDsort.m')
    open('crowding_dist.m')

    %Související funkce s výpočtem pokrytí
    open('Pokryti.m');

%% ================================================================
%                        Načtení mapového podkladu
%================================================================

readhgt();       %Fce pro extrahování hgt hodnot do m.filu
load('hodnoty.mat','X');    %Funkce pro načtení struktury s
informacemi
velikost = size(X.z); %Zjištění velikosti, tedy rozlišení mapy
Mapa = double(X.z);   %Uložení výškových hodnot do proměnné

%Výběr části mapového podkladu, která poslouží jako mapa,
%zároveň nastavení vzdálenosti bodů v mapě(rozlišení), velikost
%mapy zde volí sám uživatel

%Rozměry v obou směrech musí být stejné - čtvercová mřížka
        if velikost == 3601
            Mapa = Mapa(1:7,1:7);
            rozl = 30;
        else
            Mapa = Mapa(1:33,1:33);
            rozl = 90;
        end

        %Uvolnění paměti
        clear X;
        clear velikost;

%% Nastavení parametrů radiového systému
```

```
        f = 2100e6;             %Frekvence signálu       [Hz]
        c = 3e8;                %Rychlost světla         [m/s]
        lambda = c/f;           %Vlnová délka            [m]
        k = (2*pi)/lambda;      %Vlnové číslo            [1/m]

        G_T = 15;               %Zisk vysílací antény    [dBi]
        G_R = 15;               %Zisk přijímací antény   [dBi]

        %Dále zmíněné parametry jsou zvoleny náhodně
        % P_T       - výkon vysílače                     [dBm]
        % h_vysilac - výška antény                       [m]
        % x_vys     - x-ová souřadnice vysílače          [-]
        % y_vys     - y-ová souřadnice vysílače          [-]

%Určení rozsahu náhodně zvolených veličin - nastavitelné hodnoty
P_T_min = 30;  %Minimální hodnota vysílaného výkonu     [dBm]
P_T_max = 60;  %Maximální hodnota vysílaného výkonu     [dBm]
h_vysilac_min = 10; %Minimální výška vysílače               [m]
h_vysilac_max = 35; %Maximální výška vysílače               [m]

%Určení rozsahu náhodně zvolených veličin - neměnné hodnoty
x_vys_min = 1;   %Min. hodnota x-souřadnice vysílače [-]
x_vys_max = length(Mapa); %Max. hodnota x-souřadnice vysílače[-]
y_vys_min = 1;   %Min. hodnota y-souřadnice vysílače [-]
y_vys_max = length(Mapa); %Max. hodnota y-souřadnice vysílače[-]

%% Nastavení parametrů optimalizace

        N = 20;     % Počet částic
        M = 100;    % Počet iterací
        MM = 50;    % Maximální počet řešení

        %Nastavení typu inerciální váhy
        Inertial_w = 1;
        %   0    -   inerciální váha je fixní hodnotou
        %   1    -   inerciální váha se mění v každé iteraci

        %Nastavení hodnot pro výpočet proměnné Inerciální váhy
        w_min = 0.4;    %Minimální inerciální váha
        w_max = 0.7;    %Maximální inerciální váha

%Inicializace paměti pro částice, dle proměnných následovně
        % PARTICLES = [x_vys,y_vys,h_vysilac,P_T,fce_1,fce_2]
            PARTICLES = zeros(N,6);
        %Inicializace paměti, pro rychlosti částic následovně
        % velocity = [r_x_vys,r_y_vys,r_h_vysilac,r_P_T]
            velocity = zeros(N,4);
        %Pomocný archiv
            archiv_2 = [];
        %Externí archiv pro ukládání všech řešení
            repository = [];
        %Konstanty pro výpočet rychlosti
            Inertial_weight = 1;
```

```
            C1 = 2;
            C2 = 2;
%Vytvoření proměnné, pro kontrolu, zdali je pbest již první
%nedominované či nikoli
            Lock = zeros(N,1);

%% Vytvoření prvního roje částic

    for opt_1 = 1:N

    %Randomizace hodnoty x_vys
    x_vys = randi([x_vys_min,x_vys_max]);
    %Randomizace hodnoty y_vys
    y_vys = randi([y_vys_min,y_vys_max]);
    %Randomizace hodnoty h_vysilac
    h_vysilac = randi([h_vysilac_min,h_vysilac_max]);
    %Randomizace hodnoty P_T
    P_T = randi([P_T_min,P_T_max]);
    %Výpočet hodnoty fce_1
    fce_1 = sum(Pokryti(Mapa,h_vysilac,x_vys…
     ,y_vys,rozl,lambda,f,P_T,G_T,G_R));
    %Výpočet hodnoty fce_2
    fce_2 = ((h_vysilac/h_vysilac_max)+(P_T/P_T_max));
    %Uložení proměnných do struktury
    PARTICLES(opt_1,:) = …
     [x_vys,y_vys,h_vysilac,P_T,fce_1,fce_2];
    %Vymazání proměnných pro další použití
    clear x_vys; clear y_vys; clear h_vysilac;
    clear P_T;  clear fce_1; clear fce_2;

    end

%Uložení prního roje do archivu pro další porovnání, tak abychom
%nalezli a mohli updatovat PBEST jednotlivých řešení
    archiv_1 = PARTICLES;

%% Vytvoření prvních rychlostí částic

    for pom_1 = 1:N

    %Nalezení rychlosti pro souřadnici x
    velocity(pom_1,1) = randi([x_vys_min,x_vys_max])/10;
    %Nalezení rychlosti pro souřadnici y
    velocity(pom_1,2) = randi([y_vys_min,y_vys_max])/10;
    %Nalezení rychlosti pro výšku antény
    velocity(pom_1,3) = randi([h_vysilac_min,h_vysilac_max])/10;
    %Nalezení rychlosti pro výkon vysílače
    velocity(pom_1,4) = randi([P_T_min,P_T_max])/10;

    end

%% Nalezení prvních PBEST
```

```
    % Pro prví iteraci odpovídá PBEST každé částice této částici
    PBEST = PARTICLES;

%% Nalezení prvních GBEST

    % Pro první iteraci nalezneme nedominovaná řešení
    vystup = NDsort(PARTICLES,archiv_2);
    %Uložení prvních nedominovaných řešení do archivu řešení
    repository = vertcat(repository,vystup);

% Nejbližší z nich k dané částici vybereme jako GBEST pro tuto
% částici
% Jelikož jsme nyní ve dvourozměrném prostoru, počítáme onu
% vzdálenostjako euklidovskou vzdálenost od částice k různým
% řešením
    dxy = pdist2([PARTICLES(:,5),PARTICLES(:,6)]...
                ,[vystup(:,5),vystup(:,6)],'euclidean');
    [~,ind] = min(dxy,[],2);
    GBEST = vystup(ind,:);

    %Vymazání proměnných pro další použití
    clear dxy;
    clear ind;

%% Iterační smyčka

    %Prealokace paměti pro pozice částic
    position = zeros(N,4);

 for opt_3 = 1:M

%% Podmínka a výpočet inerciální váhy

    if Inertial_w == 0;

    elseif Inertial_w == 1;
        Inertial_weight = w_max -(((w_max-w_min)*opt_3)/M);
    else
    end

%% Výpočet nových rychlostí a částic

  for opt_4 = 1:N

     r1 = rand(1);                  %Random Factor 0-1
     r2 = rand(1);                  %Random Factor 0-1




%Výpočet rychlosti - souřadnice x, souřadnice y, výška vysílače,
%výkon vysílače
```

```
velocity(opt_4,1:4) = Inertial_weight.*velocity(opt_4,1:4) + ...
         C1.*r1.*(PBEST(opt_4,1:4) - PARTICLES(opt_4,1:4)) +...
         C2.*r2.*(GBEST(opt_4,1:4) - PARTICLES(opt_4,1:4));


%Výpočet nové hodnoty - souřadnice x
position(opt_4,1) = round(PARTICLES(opt_4,1) +…
velocity(opt_4,1));

     if position(opt_4,1) > x_vys_max
         position(opt_4,1) = x_vys_max;
     elseif position(opt_4,1) < x_vys_min
         position(opt_4,1) = x_vys_min;
     else
     end

%Výpočet nové hodnoty - souřadnice y
position(opt_4,2) = round(PARTICLES(opt_4,2) +…
velocity(opt_4,2));

     if position(opt_4,2) > y_vys_max
         position(opt_4,2) = y_vys_max;
     elseif position(opt_4,2) < y_vys_min
         position(opt_4,2) = y_vys_min;
     else
     end

%Výpočet nové hodnoty - výška vysílače
position(opt_4,3) = PARTICLES(opt_4,3) + velocity(opt_4,3);
     if position(opt_4,3) > h_vysilac_max
         position(opt_4,3) = h_vysilac_max;
     elseif position(opt_4,3) < h_vysilac_min
         position(opt_4,3) = h_vysilac_min;
     else
     end

%Výpočet nové honoty - výkon vysílače
position(opt_4,4) = PARTICLES(opt_4,4) + velocity(opt_4,4);
     if position(opt_4,4) > P_T_max
         position(opt_4,4) = P_T_max;
     elseif position(opt_4,4) < P_T_min
         position(opt_4,4) = P_T_min;
     else
     end

%Výpočet nové hodnoty fce_1
fce_1 = sum(Pokryti(Mapa,position(opt_4,3),position(opt_4,1)...
                        ,position(opt_4,2),rozl,lambda,f...
                        ,position(opt_4,4),G_T,G_R));


%Výpočet nové hodnoty fce_2
 fce_2 = (position(opt_4,3)/h_vysilac_max)+…
```

```
          (position(opt_4,4)/P_T_max);

%Uložení hodnot - vytvoření nové částice
     PARTICLES(opt_4,:) = [position(opt_4,:),fce_1,fce_2];

      %Vymazání proměnných pro další použití
     clear fce_1;   clear fce_2;

%Konec výpočtu nových částic
  end

%Nové hodnoty částic uložíme do archivu řešení
      [repository] = vertcat(repository,PARTICLES);

%Nedominantní třízení archivu
      [repository] = NDsort(repository,archiv_2);

%Podmínka maximálního počtu řešení - šetříme paměť
    if size(repository,1) > MM
        repository = crowding_dist(repository,MM);
    else
    end

%% Určení nových PBEST  pro každou částici

    for pom_1 = 1:N
    if Lock(pom_1) == 1 %Pbest je první nedominované - navždy
    else                %Porovnáme pbest s novou částicí pro
                         % dominanci

      %Nové řešení dominuje staré - uložíme ho jako pbest
      if PARTICLES(pom_1,5) >= archiv_1(pom_1,5) && ...
         PARTICLES(pom_1,6) <= archiv_1(pom_1,6)
           PBEST(pom_1,:) = PARTICLES(pom_1);

%Staré řešení dominuje nové - ponecháme ho jako pbest
      elseif PARTICLES(pom_1,5) <= archiv_1(pom_1,5) && ...
             PARTICLES(pom_1,6) >= archiv_1(pom_1,6)
                PBEST(pom_1,:) = PARTICLES(pom_1);

%Nedominantní vůči sobě
      else
            Lock(pom_1) = 1;
            PBEST(pom_1,:) = PARTICLES(pom_1,:);
      end
    end
    end




%% Určení nových GBEST  pro každou částici
```

```
    %Nalezení vzdálenosti k jednotlivým řešením
    dxy = pdist2([PARTICLES(:,5),PARTICLES(:,6)]...
                ,[repository(:,5),repository(:,6)],'euclidean');
    [~,ind] = min(dxy,[],2);
    GBEST = repository(ind,:);
    %Vymazání proměnných pro další použití
    clear dxy;
    clear ind;

%% Uložení současných hodnot částic do archivu
%  Pro porovnání v následující iteraci
    archiv_1 = PARTICLES;

%Konec iterační smyčky
 end
```

B.2 NDsort.m

```
%%=================================================================
%      Funkce pro nedominantí třízení - Metoda Slow & Naive
%=================================================================

function [vystup] = NDsort(vstup,del)

    %Označení pro snadnější zápis
    fce_1 = vstup(:,5); %find max
    fce_2 = vstup(:,6); %find min
    %ZJištění velikosti archivu
    N = size(vstup,1);

for pom_1 = 1:N
    for pom_2 = 1:N;
        %Podmínka stejného řešení
        if pom_2 == pom_1;
        else
            %Hledám řešení která jsou dominovaná - označím je
            if fce_1(pom_1) <= fce_1(pom_2) &&…
               fce_2(pom_1) >= fce_2(pom_2)
                [del] = [del pom_1];
            else
            end
        end
    end
end

    %Vymažeme řešení označená pro dominanci
    vstup(del,:) = [];
    vystup =  vstup;
end
```

B.3 crowding_dist.m

```
%%=========================================================
%        Funkce pro výpočet crowding distance
%=========================================================

function [repository] = crowding_dist(repository,MM)

    %Přepis pro jednodušší zápis
    fce_1 = repository(:,5);
    fce_2 = repository(:,6);

    CD = zeros(size(repository,1),6);
    CD(:,1) = fce_1;    %Do prvního sloupce uložíme hodnoty
fce_1
    CD(:,2) = fce_2;    %Do druhého sloupce uložíme hodnoty
fce_2
    CD(:,6) = 1:1:size(repository,1);   %vytvoříme index pro
řešení

    %seřazení dle fce_1
    CD = sortrows(CD,1);
    %Pro minimální a maximální hodnotu považujeme CD = Inf
    CD([1,end],3) = Inf;
    %Výpočet CD dle fce_1
    for pom_1 = 2:size(repository,1)-1
        CD(pom_1,3) = (CD(pom_1+1,1) - CD(pom_1-1,1))/...
                      (CD(end,1) - CD(1,1));
    end

    %seřazení dle fce_2
    CD = sortrows(CD,2);
    %Pro minimální a maximální hodnotu považujeme CD = Inf
    CD([1,end],4) = Inf;
    %Výpočet CD dle fce_2
    for pom_2 = 2:size(repository,1)-1
            CD(pom_2,4) = (CD(pom_1+1,2) - CD(pom_1-1,2))/...
                      (CD(end,2) - CD(1,2));
    end

    %Výsledná crowding distance je součtem cd pro jednotlivé
funkce
    CD(:,5) = CD(:,3) + CD(:,4);
    %Seřazení dle výsledné crowding distance, řešení s nejvyšší
hodnotou
    %ponecháme
    CD = sortrows(CD,5);
    %Najdeme indexy řešení, které ponecháme
    index = CD(end-MM:end,6);
    repository = repository(index,:);
end
```

B.4 Pokryti.m

```
%% ============================================================
%          Funkce pro výpočet pokrytí signálem
%============================================================

function [P_2]= Pokryti(Mapa,h_vysilac,x_vys,y_vys…
                            ,rozl,lambda,f,P_T,G_T,G_R)


%% ============================================================
%         Výpočet vzdálenosti jednotlivých bodů
%============================================================

[sx,sy,Euklid,Chebyd] = Vzdalenosti(Mapa,x_vys,y_vys,rozl);
% sx - souřadnice x jednotlivých bodů
% sy - souřadnice y jednotlivých bodů
% Euklid - euklidovská vzdálenost bodů od pozice vysílače
% Chebyd - čebyševova vzdálenost bodů od pozice vysílače

%% ============================================================
%                        Rozložení na jednotlivé trasy
%                    Výpočet výškových bodů jednotlivých tras
%============================================================

 [Trasy] = Rozloz(x_vys,y_vys,Mapa,sx,sy);
% Trasy - struktura obsahující jednotlivé trasy (výškové
hodnoty),od vysílače směrem k přijímači

%% ============================================================
%  Výpočet útlumu jednotlivých tras v závislosti na refrakci
%                        Výpočet útlumu volným prostorem
%============================================================

 [Celkovy_utlum_refr,pom_matice] =
Utlum_tras(Trasy,Euklid,h_vysilac,lambda,Chebyd);
% Útlum, pro jednotlivé trasy, způsobený refrakcí signálu na
překážkách

FSL = 32.44 + 20*log10(f/1e6)+20*log10(pom_matice(:,4)/1e3);
        % FSL - útlum volným prostorem
        % Frekvence podělena 1e6 - vzorec počítá s f [MHz]
        % Vzdálenost podělena 1e3 - vzorec počítá s d [km]
FSL2 = 10*log10(((4*pi*Euklid)./lambda).^2);

%% ============================================================
% Úprava parametrů v bodě vysílače - nahrazení Inf hodnot, atd.
%============================================================

        % Nalezení indexu pozice vysílače
        index = sub2ind(size(Mapa),x_vys,y_vys);

     % Nahrazení daných hodnot odpovídajícími
```

```
        Euklid(index) = 0.1;
     %Jelikož neumíme dělit 0, nahradíme nulovou vzdálenost
 pom_matice(index,4) = h_vysilac;
%Skutečná vzdálenost v místě vysílače je výška vysílače,
popřípadě opět nahradíme "0" (nenulovou hodnotou blízkou 0)

%% ============================================================
%                   Výpočet přijímaného výkonu
%      Výpočet intenzity elektrického pole v dané oblasti
% ============================================================

% Výpočet intenzity pole - s euklidovskou vzdáleností
Eef_1 = sqrt(30*P_T*G_T)./Euklid;

% Výpočet intenzity pole - se skutečnou vzdáleností
Eef_2 = sqrt(30*P_T*G_T)./pom_matice(:,4);

% Výpočet výkonu signálu pří šíření volným prostorem [dBm]
P = G_T + G_R + P_T - FSL ;

% Výpočet výkonu signálu pří šíření s refrakcí [dBm]
P_2 = P + Celkovy_utlum_refr;



%         figure(1);
%         subplot(2,2,1);
%         surfc(reshape(FSL,7,7),'EdgeAlpha',0);
%         title('Free-space loss  [dB]');
%         colorbar;
%         subplot(2,2,2);
%         surfc(reshape(P,7,7),'EdgeAlpha',0);
%         title('Received power(FSL)  [dBm]');
%         colorbar;
%         subplot(2,2,3);
%         surfc(reshape(-
1*Celkovy_utlum_refr,7,7),'EdgeAlpha',0);
%         title('Refraction loss  [dB]');
%         colorbar;
%         subplot(2,2,4);
%         surfc(reshape(P_2,7,7),'EdgeAlpha',0);
%         title('Received power(FSL+RL)  [dBm]');
%         colorbar;


end
```

B.5 Vzdalenosti.m

```
%% ================================================================
% Funkce pro výpočet Euklidovy, Chebychevovy vzdálenosti,
% souřadnic x a y
%=================================================================

function [sx,sy,Euklid,Chebyd] =
Vzdalenosti(Mapa,x_vys,y_vys,rozl)

% Zjistění souřadnic x, y všech bodů v mapě
    [sx,sy] = ind2sub(size(Mapa),1:numel(Mapa));

% Výpočet euklidovské vzdálenosti mezi základním bodem a body
ostatními
    Euklid2 = pdist2([x_vys,y_vys],[sx',sy'],'euclidean');
    Euklid = rozl*Euklid2;

% Výpočet čebyševovy vzdálenosti mezi základním bodem a body
ostatními
    Chebyd = pdist2([x_vys,y_vys],[sx',sy'],'chebychev');

end
```

B.6 Rozloz.m

```
%% ================================================================
% Funkce na rozložení mapového podkladu na jednotlivé trasy
%=================================================================

function [Trasy] = Rozloz(x_vys,y_vys,Mapa,sx,sy)

% Vytvoření struktury pro uložení jednotlivých tras
Trasy = cell(numel(Mapa),1);
% Cyklus pro výpočet jednotlivých tras
for k = 1:numel(Mapa);
    %Určení krajních bodů přímky
    p1 = [x_vys,y_vys];
    p2 = [sx(k),sy(k)];
        %Určení délkek v jednotlivých směrech
    dx = p1(1):p2(1);
    dy = p1(2):p2(2);
    if isempty(dx) ==1
        dx = fliplr(p2(1):p1(1));
    else
    end
    if isempty(dy) ==1
        dy = fliplr(p2(2):p1(2));
    else
    end
```

```
% Výpočet ostatních bodů úsečky, v závislosti na vzdálenosti
daných bodů
    if length(dx) > length(dy)
        x = p1(1):p2(1);
        if isempty(x) ==1
            x = fliplr(p2(1):p1(1));
        else
        end
y = round((x-p1(1))*(p2(2)-p1(2))/(p2(1)-p1(1))+ p1(2));

        pozice = sub2ind(size(Mapa),x',y');
        if isnan(pozice) == 1
            Trasy{k}=Mapa(x_vys,y_vys);
        else
            Trasy{k}=Mapa(pozice);
        end
    elseif length(dy) > length(dx)
        y = p1(2):p2(2);
        if isempty(y) ==1
            y = fliplr(p2(2):p1(2));
        else
        end
x = round((y-p1(2))*(p2(1)-p1(1))/(p2(2) - p1(2)) + p1(1));

        pozice = sub2ind(size(Mapa),x',y');
        if isnan(pozice) == 1
            Trasy{k}=Mapa(x_vys,y_vys);
        else
            Trasy{k}=Mapa(pozice);
        end
    else
        x = p1(1):p2(1);
        if isempty(x) ==1
            x = fliplr(p2(1):p1(1));
        else
        end
y = round((x-p1(1))*(p2(2)-p1(2))/(p2(1)-p1(1)) + p1(2));

        pozice = sub2ind(size(Mapa),x',y');
        if isnan(pozice) == 1
            Trasy{k}=Mapa(x_vys,y_vys);
        else
            Trasy{k}=Mapa(pozice);
        end
    end

    % Konec cyklu pro výpočet
end
% Konec funkce
end
```

B.7 Utlum_tras.m

```
%% ================================================================
% Funkce pro výpočet útlumu jednotlivých tras, za pomoci funkcí
% Najdi,Deygout1, Deygout2
%================================================================

function [Celkovy_utlum_refr,pom_matice] =
Utlum_tras(Trasy,Euklid,h_vysilac,lambda,Chebyd)

pom_1 = length(Trasy);
pom_matice = zeros(pom_1,4);
Celkovy_utlum_refr = zeros(numel(Trasy),1);

%Určení zda jsou na trase překážky, či nikoliv, nalezení pozice
%překážky a zjištění jejího útlumu
for k=1:pom_1
    Cesta = cell2mat(Trasy(k));
    Vzdalenost = Euklid(k);

    [v_max,pozice_v_max,Utlum_v_max,Trasa] = …
Najdi(Cesta,Vzdalenost,lambda,h_vysilac);

    pom_matice(k,1) = v_max;
    pom_matice(k,2) = pozice_v_max;
    pom_matice(k,3) = Utlum_v_max;
    pom_matice(k,4) = Trasa;
    %Dopočítá útlum trasy, pokud jsou na ní překážky
    if pom_matice(k,2) == 0
        Celkovy_utlum_refr(k) = 0;
    else
        Celkovy_utlum_refr(k) = pom_matice(k,3);
% Pomocná pro zjištění délek Vys. - Hl.prk a Hl.prk - Příj.
        pom_2 = linspace(0,Euklid(k),Chebyd(k)+1);
        pom_3 = Cesta';
        % Rozdělení dle hlavní překážky
        Cesta_1 = pom_3(1:pom_matice(k,2));
        Cesta_2 = pom_3(pom_matice(k,2):end);

        Vzdalenost_1 = pom_2(pom_matice(k,2));
        Vzdalenost_2 = pom_2(end-pom_matice(k,2));

% Výpočet útlumu trasy, při znalosti hlavní překážky

Utlum_trasy_1=Deygout_1(Cesta_1,Vzdalenost_1,lambda,h_vysilac);
        Utlum_trasy_2=Deygout_2(Cesta_2,Vzdalenost_2,lambda);

        Celkovy_utlum_refr(k) = Utlum_trasy_1 + Utlum_trasy_2 +…
pom_matice(k,3);
    end
end
end
```

B.8 Najdi.m

```
function [v_max,pozice_v_max,Utlum_v_max,Trasa]=…
      Najdi(Cesta,Vzdalenost,lambda,h_vysilac)

Map_l = length(Cesta);
Cesta = Cesta';
%% Z rozdílu výšek vypočítá reálnou délku trasy
Rozdil = Cesta(1,end)-(Cesta(1,1)+h_vysilac);

if Rozdil > 0;
    Trasa = sqrt(Vzdalenost^2+Rozdil^2);
elseif Rozdil < 0;
    Trasa = sqrt(Vzdalenost^2+Rozdil^2);
elseif Rozdil == 0;
    Trasa = Vzdalenost;
end

Uhel_TR = asind(Vzdalenost/Trasa) ;
Uhel_TN = 90-Uhel_TR;

%% Výpočet vzdáleností d1 a d2 pro výpočet R_fres
if Map_l ==1 || Map_l==2
    d1 = 0;
    d2 = 0;
else
    d1 = (Trasa/(Map_l-1)):(Trasa/(Map_l-1)):(Trasa-
(Trasa/(Map_l-1)));
    d2 = fliplr(d1);
end

%% Výpočet poloměru první Fresnelovy zóny
R_fres = sqrt(1)*sqrt((d1.*d2)./(d1+d2));

%% Výpočet mezních hodnot výšek

if Rozdil > 0;
    pom_d_1 = 0:Vzdalenost/(Map_l-1):Vzdalenost;
    Mez_h = Cesta(1,1) +
(pom_d_1.*(sin((Uhel_TN/180)*pi)))./(sin((Uhel_TR/180)*pi));

elseif Rozdil < 0;
    pom_d_1 = 0:Vzdalenost/(Map_l-1):Vzdalenost;
    pom_d_2 = fliplr(pom_d_1);
    Mez_h = Cesta(1,end) +
(pom_d_2.*(sin((Uhel_TN/180)*pi)))./(sin((Uhel_TR/180)*pi));

elseif Rozdil == 0;
    Mez_h(1:Map_l) = Cesta(1,end);   % výška vysílače
end
```

```
%% Zkrácení
if Map_l ==1 || Map_l==2
    Mez_h_2 = 0;
else
    Mez_h_2 = Mez_h(2:end-1);
End

 %% Výpočet výšky překážek na trase
if Map_l ==1 || Map_l==2
    h_prk = 0;
else
    h_prk = Cesta(2:end-1)-Mez_h_2;
end

%% Výpočet Fresnel-Kirchoffova difrakčního parametru
v = h_prk*sqrt((2*(d1+d2))/(lambda.*d1.*d2));
%% Záznam největšího v a pozice překážky
if Map_l ==1 || Map_l==2
    v_max = -2;
    pozice_v_max = 0;
else
    v_max = max(v);
    pozice_v_max = find(v==max(v))+1;    %+1, jelikož nepočítáme
krajní body
    pom_1 = length(pozice_v_max);
    if pom_1 > 1
        pozice_v_max = pozice_v_max(1);
    end

end

%% Výpočet a záznam útlumu hlavní překážky


if v_max <= -1
    GAX = 0;
elseif v_max > -1 && v_max <= 0
    GAX = 20 * log10(0.5 - 0.62*v_max);
elseif v_max > 0  && v_max <= 1
    GAX = 20 * log10(0.5 * exp(-0.95 * v_max));
elseif v_max > 1  && v_max <= 2.4
    GAX = 20 * log10(0.4 - sqrt(0.1184 - (0.38 - 0.1 *
v_max)^2));
elseif v_max > 2.4
    GAX = 20 * log10(0.225/v_max);
end

Utlum_v_max = GAX;

end
```

B.9 Deygout_1.m

```
Fiction [Utlum_trasy_1]=…
     Deygout_1(Cesta_1,Vzdalenost_1,lambda,h_vysilac)

    Map_l = length(Cesta_1);

%% Výpočet parametrů první trasy
    Rozdil_1 = Cesta_1(1,end)-(Cesta_1(1,1)+h_vysilac);
%% Výpočet reálné délky trasy č.1
     if Rozdil_1 > 0;
        Trasa_1 = sqrt(Vzdalenost_1^2+Rozdil_1^2);
     elseif Rozdil_1 < 0;
        Trasa_1 = sqrt(Vzdalenost_1^2+Rozdil_1^2);
     elseif Rozdil_1 == 0;
        Trasa_1 = Vzdalenost_1;
     end

     Uhel_TR_1 = asind(Vzdalenost_1/Trasa_1) ;
     Uhel_TN_1 = 90-Uhel_TR_1;
%% výpočet R_fres
     if Map_l ==1 || Map_l==2
         d1 = 0;
         d2 = 0;
     else
         d1 = (Trasa_1/(Map_l-1)):(Trasa_1/(Map_l-1))…
                :(Trasa_1-(Trasa_1/(Map_l-1)));
         d2 = fliplr(d1);
     end
%% Výpočet poloměru první Fresnelovy zóny
     R_fres = sqrt(1)*sqrt((d1.*d2)./(d1+d2));
%% Výpočet mezních hodnot výšek
    if Map_l ==1 || Map_l ==2
       Mez_h = 0;
    else
         if Rozdil_1 > 0;
            pom_d_1 = linspace(0,Vzdalenost_1,Map_l);
            Mez_h = Cesta_1(1,1) +…
(pom_d_1.*(sin((Uhel_TN_1/180)*pi))./(sin((Uhel_TR_1/180)*pi));
         elseif Rozdil_1 < 0;
            pom_d_1 = linspace(0,Vzdalenost_1,Map_l);
            pom_d_2 = fliplr(pom_d_1);
            Mez_h = Cesta_1(1,end) +…
(pom_d_2.*(sin((Uhel_TN_1/180)*pi))./(sin((Uhel_TR_1/180)*pi));
         elseif Rozdil_1 == 0;
            Mez_h(1:Map_l) = Cesta_1(1,end);   % výška vysílače
         end
    end
```

```
%% Zkrácení
if Map_l ==1 || Map_l==2
    Mez_h_2 = 0;
else
    Mez_h_2 = Mez_h(2:end-1);
end
%% Výpočet výšky překážek na trase
       if Map_l ==1 || Map_l==2
            h_prk = 0;
       else
            h_prk = Cesta_1(2:end-1)-Mez_h_2;
       end
%% Výpočet Fresnel-Kirchoffova difrakčního parametru
       if Map_l ==1 || Map_l==2
            v = -2;
       else
            v = h_prk*sqrt((2*(d1+d2))/(lambda.*d1.*d2));
       end
%% Přepočet v na útlum

        if Map_l ==1 || Map_l ==2
            GAX = 0;
        else

            for h = 1:(Map_l-2)
                if v(h) <= -1
                    GAX(h) = 0;
                elseif v(h) > -1 && v(h) <= 0
                    GAX(h) = 20 * log10(0.5 - 0.62*v(h));
                elseif v(h) > 0  && v(h) <= 1
                GAX(h) = 20 * log10(0.5 * exp(-0.95 * v(h)));
                elseif v(h) > 1  && v(h) <= 2.4
GAX(h) = 20 * log10(0.4 - sqrt(0.1184 - (0.38 - 0.1 * v(h))^2));
                elseif v(h) > 2.4
                    GAX(h) = 20 * log10(0.225/v(h));
                end
            end
        end

               OL_1 = sum(GAX);


       Utlum_trasy_1 = OL_1;

end
```

B.10 Deygout_2.m

```
function [Utlum_trasy_2]=Deygout_2(Cesta_2,Vzdalenost_2,lambda)

    Map_l = length(Cesta_2);

%% Výpočet parametrů druhé trasy
    Rozdil_2 = Cesta_2(1,end)-(Cesta_2(1,1));
%% Výpočet reálné délky trasy č.2
     if Rozdil_2 > 0;
        Trasa_2 = sqrt(Vzdalenost_2^2+Rozdil_2^2);
     elseif Rozdil_2 < 0;
        Trasa_2 = sqrt(Vzdalenost_2^2+Rozdil_2^2);
     elseif Rozdil_2 == 0;
        Trasa_2 = Vzdalenost_2;
     end

     Uhel_TR_2 = asind(Vzdalenost_2/Trasa_2) ;
     Uhel_TN_2 = 90-Uhel_TR_2;
%% výpočet R_fres
     if Map_l ==1 || Map_l==2
         d1 = 0;
         d2 = 0;
     else
         d1 = (Trasa_2/(Map_l-1)):(Trasa_2/(Map_l-1))…
           :(Trasa_2-(Trasa_2/(Map_l-1)));
         d2 = fliplr(d1);
     end
%% Výpočet poloměru první Fresnelovy zóny
     R_fres = sqrt(1)*sqrt((d1.*d2)./(d1+d2));
%% Výpočet mezních hodnot výšek
    if Map_l ==1 || Map_l ==2
       Mez_h = 0;
    else
         if Rozdil_2 > 0;
            pom_d_1 = linspace(0,Vzdalenost_2,Map_l);
            Mez_h = Cesta_2(1,1) +…
(pom_d_1.*(sin((Uhel_TN_2/180)*pi)))./(sin((Uhel_TR_2/180)*pi));
         elseif Rozdil_2 < 0;
            pom_d_1 = linspace(0,Vzdalenost_2,Map_l);
            pom_d_2 = fliplr(pom_d_1);
            Mez_h = Cesta_2(1,end) +…
(pom_d_2.*(sin((Uhel_TN_2/180)*pi)))./(sin((Uhel_TR_2/180)*pi));
         elseif Rozdil_2 == 0;
            Mez_h(1:Map_l) = Cesta_2(1,end);   % výška vysílače
         end
    end
%% Zkrácení
if Map_l ==1 || Map_l==2
    Mez_h_2 = 0;
else
    Mez_h_2 = Mez_h(2:end-1);
end
```

```
%% Výpočet výšky překážek na trase
       if Map_l ==1 || Map_l==2
            h_prk = 0;
       else
            h_prk = Cesta_2(2:end-1)-Mez_h_2;
       end
%% Výpočet Fresnel-Kirchoffova difrakčního parametru
       if Map_l ==1 || Map_l==2
            v = -2;
       else
            v = h_prk*sqrt((2*(d1+d2))/(lambda.*d1.*d2));
       end
%% Přepočet v na útlum


        if Map_l ==1 || Map_l ==2
            GAX = 0;
        else
            for h = 1:(Map_l-2)
                if v(h) <= -1
                    GAX(h) = 0;
                elseif v(h) > -1 && v(h) <= 0
                    GAX(h) = 20 * log10(0.5 - 0.62*v(h));
                elseif v(h) > 0  && v(h) <= 1
                GAX(h) = 20 * log10(0.5 * exp(-0.95 * v(h)));
                elseif v(h) > 1  && v(h) <= 2.4
GAX(h) = 20 * log10(0.4 - sqrt(0.1184 - (0.38 - 0.1 * v(h))^2));
                elseif v(h) > 2.4
                    GAX(h) = 20 * log10(0.225/v(h));
                end
            end
        end

               OL_2 = sum(GAX);


       Utlum_trasy_2 = OL_2;

end
```