VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV VÝKONOVÉ ELEKTROTECHNIKY A ELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

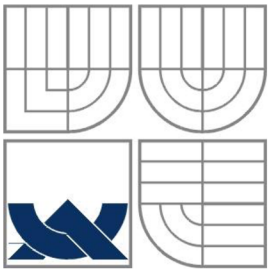
DEPARTMENT OF POWER ELECTRICAL AND ELECTRONIC ENGINEERING

GLOBÁLNÍ OPTIMALIZACE FUNKCÍ VÍCE PROMĚNNÝCH

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

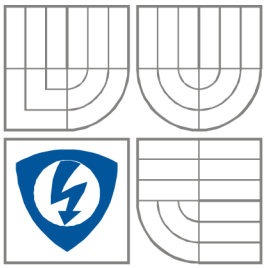
AUTOR PRÁCE Bc. STANISLAV VLÁČIL
AUTHOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV VÝKONOVÉ ELEKTROTECHNIKY A ELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF POWER ELECTRICAL AND ELECTRONIC ENGINEERING

GLOBÁLNÍ OPTIMALIZACE FUNKCÍ VÍCE PROMĚNNÝCH

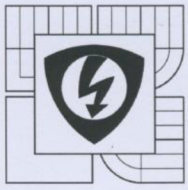
GLOBAL OPTIMIZATION OF MULTIPLE VARIABLES FUNCTIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. STANISLAV VLÁČIL
AUTHOR

VEDOUCÍ PRÁCE Mgr. PETR KLOC, Ph.D.
SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ
Fakulta elektrotechniky
a komunikačních technologií
Ústav výkonové elektrotechniky a elektroniky

Diplomová práce

magisterský navazující studijní obor
Silnoproudá elektrotechnika a výkonová elektronika

Student: Bc. Stanislav Vláčil ***ID:*** 162249
Ročník: 2 ***Akademický rok:*** 2014/15

NÁZEV TÉMATU:

Globální optimalizace funkcí více proměnných

POKYNY PRO VYPRACOVÁNÍ:

1.Na základě rešerše literatury vyberte algoritmus pro hledání globálního minima funkce více proměnných.
2.Vytvořte počítačový kód využívající tento algoritmus, posuďte jeho časovou náročnost a pravděpodobnost nalezení globálního minima.
3. Aplikujte zvolenou metodu na problematiku zjednodušení výpočtů přenosu záření v plazmatu

DOPORUČENÁ LITERATURA:

[1] PRESS, William H, Saul A TEUKOLSKY, William T VETTELING a Brian P FLANNERY. Numerical Recipies: The art of scientific computing. 3rd ed. Cambridge: Cambridge University Press, 2007, xxi, 1235 s. ISBN 978-0-521-88068-8.
[2] Journal of Global Optimization, Springer, ISSN 0925-5001
[3] MODEST, M. Radiative heat transfer. 2nd ed. Amsterdam: Academic Press, c2003, xxii, 822 s. ISBN 01-250-3163-7.

Termín zadání: 22. 9. 2014 ***Termín odevzdání:*** 26.5.2015

Vedoucí práce: Mgr. Petr Kloc, Ph.D.
Konzultanti diplomové práce:

Ing. Ondřej Vítek, Ph.D.
předseda oborové rady

UPOZORNĚNÍ:

Abstrakt

Diplomová práce se zabývá problematikou numerického hledání globální minima. Teoretická část obsahuje seznámení s problémem globální optimalizace, jejím základním rozdělením a ukázkou některých konkrétních úloh optimalizace. Dále seznámí čtenáře se základy některých numerických metod a popisem jejich slabých a silných stránek. Praktická část ukazuje konkrétní aplikaci algoritmu globální optimalizace a jeho užití na data zadané funkce, která se vztahují k problematice přenosu záření v elektrickém oblouku.

Abstract

Master's thesis deals with numerical finding the global minimum. A theoretical part of project presents a problem of global optimization, her basic division and shows basic examples the specific problems of global optimization. Further, the reader is familiar with the basics of some numerical methods and with description of their strengths and weaknesses. A practical part shows concrete algorithm of global optimization, its use on data of specified function. Data refer to the issue of radiative heat transfer.

Klíčová slova

Globální minimum, optimalizace, numerická metoda, programovací jazyk, optimalizační program, přenos záření v elektrickém oblouku, EGO

Keywords

Global minimum, optimization, numerical method, programming language, optimization program, radiative heat transfer, EGO

Bibliografická citace

VLÁČIL, S. *Globální optimalizace funkcí více proměnných.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2015. 57 s. Vedoucí diplomové práce Mgr. Petr Kloc, Ph.D..

Prohlášení

Prohlašuji, že svou diplomovou práci na téma Globální optimalizace funkcí více proměnných jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

V Brně dne …… Podpis autora ……..

Poděkování

Děkuji vedoucímu diplomové práce Mgr. Petru Klocovi, 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

SEZNAM SYMBOLŮ A ZKRATEK

D	Definiční obor
MILP	Mixed-Integer Linear Programming
2D	Two Dimensional
3D	Three Dimensional

ÚVOD

Optimalizace je oborem aplikované matematiky zabývající se hledáním extrémů funkcí na jejich definičním oboru, které podléhají různým omezením svých proměnných. V historii se optimalizace poprvé objevila ve spojení s logistickými a transportními problémy. Typické byly snahy nalézt takové řešení těchto problémů, aby cena byla minimální a zároveň byla uspokojena všechna omezení, kde cena i omezující podmínky byly lineárními funkcemi rozhodujících proměnných. Všechny tyto problémy spadají do kategorie lineárního programování. Nejslavnější algoritmus pro řešení těchto problémů se nazývá Simplex algoritmus. Tento vznikl již ve 40. až 50. letech 20. století. Jeho složitost stoupá exponenciálně s počtem proměnných, ale v mnoha praktických případech je velmi efektivní. Spousta současných algoritmů je založena právě na tomto algoritmu. Ty jsou pak schopné řešit problémy, které osahují až miliony proměnných a omezujících podmínek.

V mnoha případech vyžaduje podstata problému jednoznačně definované rozhodovací proměnné ať už v celočíselné nebo binární podobě i poté bývá řešení značně složité. Takovéto problémy nazýváme MILP, neboli smíšené celočíselné lineární programování. Zde jsou některé nebo všechny proměnné omezeny na celočíselné hodnoty a účelové funkce a omezující podmínky jsou lineárními funkcemi. Pro řešení těchto problémů se využívají nejčastěji algoritmy Cutting Plane nebo Branch-and-Bound algoritmus. Největší složitost těchto algoritmů je exponenciální a přitom záleží na velikosti instance.

Skutečné rozdělení na problémy složité a jednoduché spočívá v tom, zda je problém konvexní nebo nekonvexní. Problém je konvexní, pokud hledáme minimum konvexní funkce nebo maximum konkávní funkce. Pokud je problém konvexní můžeme zaručit nalezení extrému. Na druhou stranu nekonvexní problémy mají mnoho lokálních extrémů a výběr toho nejlepšího může být složitým úkolem. Problém se ještě složitější, když máme diskrétní hodnoty a to i tehdy, když matematické vyjádření takových funkcí je lineární. Poté při optimalizaci hledáme spíše prostor, kde se globální extrém nachází.

Globální optimalizace je úloha, kterou je nutno řešit v mnoha praktických problémech, takže je nutné hledat algoritmy, které jsou pro řešení konkrétních problému použitelné.[3][4]

1 GLOBÁLNÍ OPTIMALIZACE

Jak již bylo řečeno v úvodu, optimalizace je oborem aplikované matematiky zabývající se hledáním extrémů funkcí na jejich definičním oboru, které podléhají různým omezením svých proměnných.[3]

1.1 Formulace problému globální optimalizace

Úlohu nalezení globálního minima můžeme formulovat takto:

Mějme účelovou funkci $f : D \to R, D \subseteq R^d$. (1)

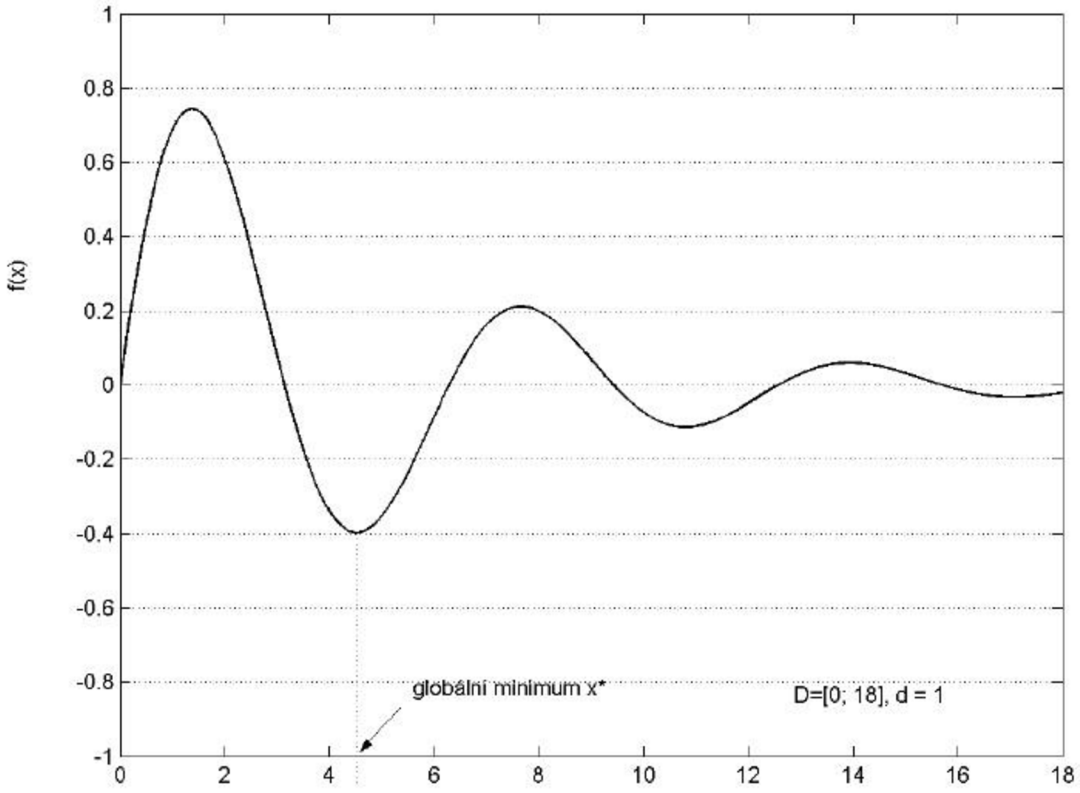
Pak $x^* = \arg\min_{x \in D} f(x)$ je globální minimum. (2)

Globální minimum je tedy jeden nebo více bodu z D s nejmenší funkční hodnotou, formálně

$$x^* = \arg\min_{x \in D} f(x) = \{x^* \in D : f(x^*) \leq f(x) pro \forall x \in D\}. \quad (3)$$

Chceme-li nalézt globální maximum, pak jej nalezneme jako globální minimum funkce $g(x) = -f(x)$. (4)

Problém nalezení globálního minima můžeme ilustrovat jednoduchým obrázkem:



Obr. 1. Globální minimu[4]

Z matematické analýzy známe postup, jak nalézt extrémy funkcí, u kterých existuje první a druhá derivace. Může se zdát, že úloha nalezení globálního minima je velmi jednoduchá. Bohužel tomu tak není. Nalézt obecné řešení takto jednoduše formulovaného problému je obtížné, zvláště když je účelová funkce multimodální, není diferencovatelná, případně má další nepříjemné vlastnosti. Analýza problému globální optimalizace ukazuje, že neexistuje deterministický algoritmus řešící obecnou úlohu globální optimalizace (tj. nalezení dostatečně přesné aproximace x*) v polynomiálním čase, tzn. problém globální optimalizace je NP-obtížný.

Přitom globální optimalizace je úloha, kterou je nutno řešit v mnoha praktických problémech, mnohdy s velmi významným ekonomickým efektem, takže je nutné hledat algoritmy, které jsou pro řešení konkrétních problému použitelné.[4]

1.2 Rozdělení algoritmů globální optimalizace

Algoritmy, které řeší úlohy globální optimalizace MILP(Smíšené celočíselné lineární programování) dělíme na dvě velké skupiny:

a)Deterministické algoritmy

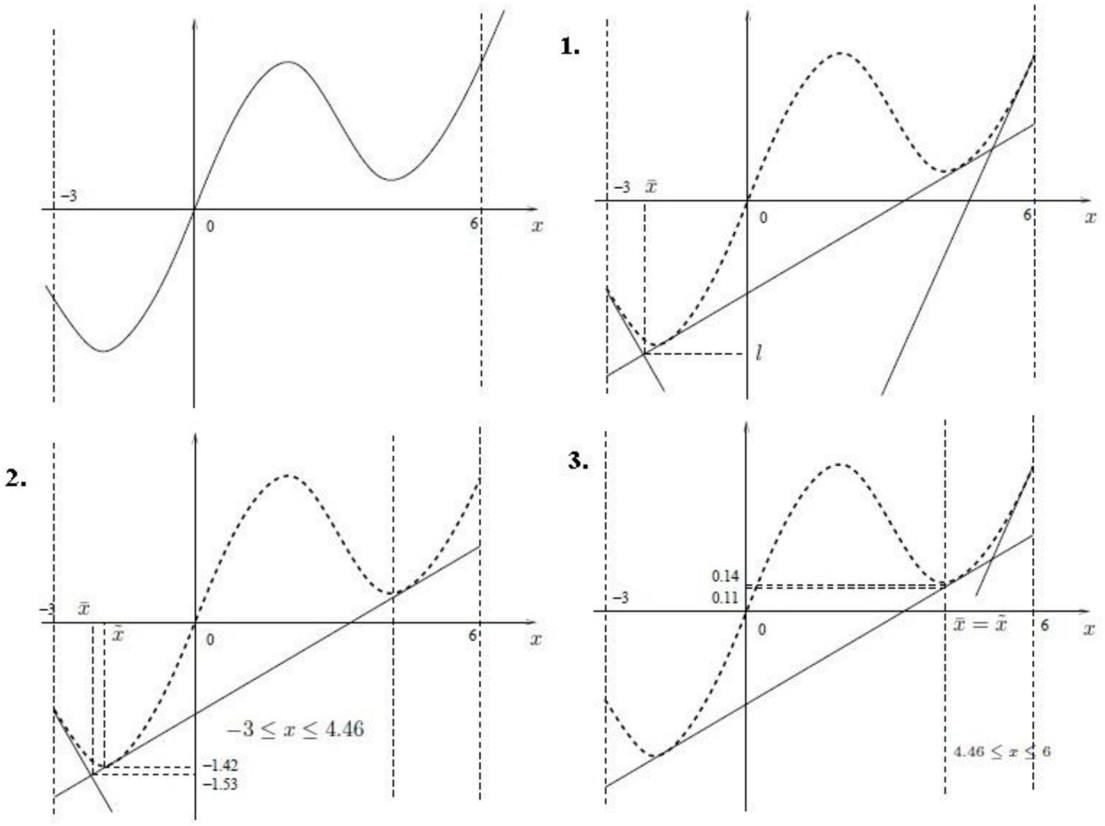
b)Stochastické algoritmy

1.2.1 Deterministické algoritmy

Deterministický je algoritmus, který může být simulován deterministickým konečným automatem. Jelikož algoritmus je posloupností instrukcí, které zajišťují, že počítač vykoná určité kroky, tak je počítač teoreticky deterministickým konečným automatem. To ale neznamená, že všechny algoritmy musí být neodmyslitelně deterministické. Navíc může být dokázáno, že jakýkoliv nedeterministický konečný automat může být simulován pomocí deterministického konečného automatu.

Do této skupiny algoritmů patří především algoritmy třídy „ Rozděl a panuj ‘‘. Nejsou sice jediné, které se z deterministických algoritmů používají, nicméně jsou nejrozšířenější. Jejich techniky jsou velice známé a z hlediska deterministického řešení problémů globální optimalizace jsou nejefektivnější. Algoritmy tohoto typu mohou být použity na vyřešení široké škály problémů optimalizace, i když účelové funkce a omezující podmínky jsou neznámé. Síla těchto algoritmů velice záleží na formulaci problému.[3]

Příklad použití deterministického algoritmu:



Obr. 2. Příklad deterministického algoritmu[3]

Problémem je určit minimum funkce $\min\left\{f(x)=\frac{1}{4}x+\sin(x)\,|\,x\geq -3, x\leq 6\right\}$. Tento problém je velice jednoduchým příkladem nelineární funkce se dvěma minimy a jen jedno z nich je minimem globálním. Na obrázku je znázorněno grafické řešení nalezení jejího minima. Optimalizace je provedena metodou „Rozděl a omez (Branch-and-Bound)“. V každé iteraci zvolíme horní a dolní hranici intervalu a za použití vždy stejného postupu zjistíme, zda jsme dostatečně blízko hledaného optima. Pokud nedosáhneme požadované přesnosti, rozdělíme celý interval na dva a opakujeme proces s každým z těchto regionů zvlášť, dokud není celý interval prověřen na přítomnost extrému.

V první iteraci využijeme hranice celého intervalu. Pro jednoduchost zvolíme přesnost jen $\varepsilon = 0{,}15$, v praxi se tato hodnota pohybuje kolem $\varepsilon = 1\cdot 10^{-6} - 1\cdot 10^{-3}$. Najdeme přímku, která prochází těsně pod naší účelovou funkcí a vytvoříme tečny k naší funkci s koncovými body x=-3 a x=-6, abychom tím naši přímku zkrátili. Minimum nalezneme v průsečíku tečny a vytvořené přímky $\bar{x} = -2{,}13$ a $l = -1{,}53$. Skutečné minimum je v bodě $l' = -1{,}53$. Dále v tomto intervalu aplikujeme Newtonovu metodu, která nám dá řešení v bodě $\tilde{x} = 4{,}46$. Porovnáme funkční hodnoty, které se v tomto případě značně liší 0,147 a -1,53. Takže si nemůžeme být jisti, že jsme nalezli minimum. Tím pádem využijeme jako horní hranici intervalu bod nalezený pomocí Newtonovy metody $\tilde{x} = 4{,}46$ a postup opakujeme. Poté nám průsečík zůstane ve stejném místě, ale Newtonova metoda nalezne optimum v bodě $\tilde{x} = -1{,}823$. Jelikož po výpočtu přesnosti nám

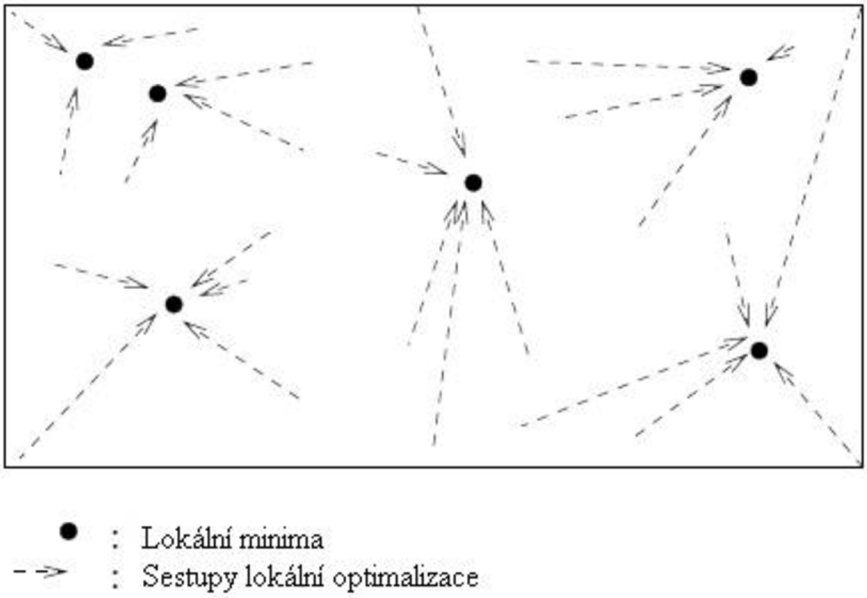
vyjde $0{,}11 < \varepsilon$, tak nalezený bod je naším hledaným bodem. Poté už jen aplikujeme metodu na zbývající interval $4{,}46 \leq x \leq 6$ ve které ovšem nebude lepší řešení nalezeno a proto bod nalezený v 2. iteraci zůstane nejlepším kandidátem na minimum.[3]

1.2.2 Stochastické algoritmy

Nemožnost nalézt deterministický algoritmus obecně řešící úlohu globální optimalizace vedla k využití algoritmů stochastických, které sice nemohou garantovat nalezení řešení v konečném počtu kroků, ale často pomohou nalézt v přijatelném čase řešení prakticky použitelné. Stochastické algoritmy pro globální optimalizaci heuristicky prohledávají prostor definičního oboru. Heuristikou rozumíme postup, ve kterém se využívá náhoda, intuice, analogie a zkušenost. Rozdíl mezi heuristikou a deterministickým algoritmem je v tom, že na rozdíl od deterministického algoritmu heuristika nezajišťuje nalezení řešení. Heuristiky jsou v praktickém životě zcela samozřejmě užívané postupy.

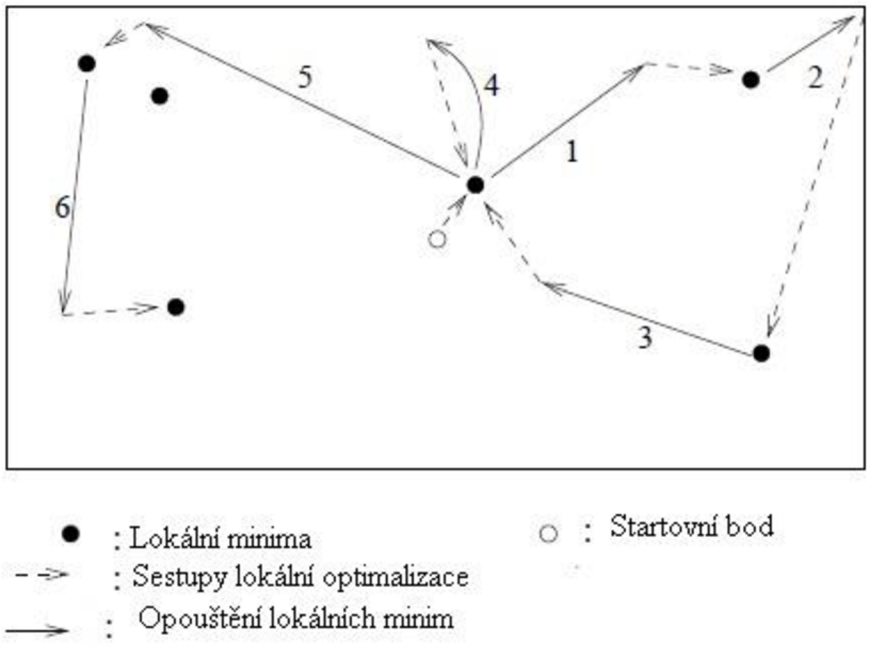
Většina stochastických algoritmů pro hledání globálního minima v sobě obsahuje zjevně či skrytě proces učení. Inspirace k užití heuristik jsou často odvozeny ze znalostí přírodních nebo sociálních procesu. Postupy učení najdeme snad ve všech známých stochastických algoritmech s výjimkou slepého náhodného prohledávání. [4]

Příklad principů stochastických algoritmů:



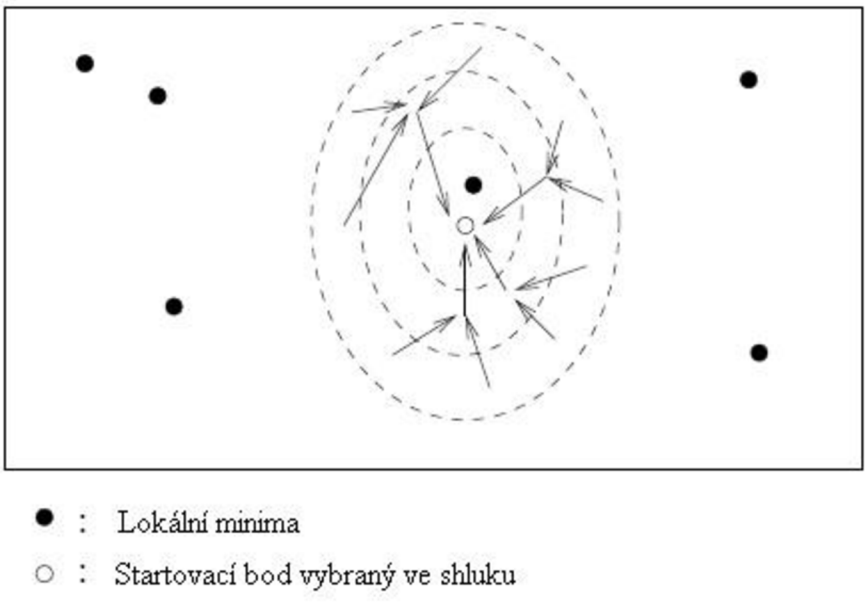
Obr. 3. Stochastické prohledávání 1[3]

Příklad, kde se startuje z více startovních pozic v zadaném intervalu. Tyto pozice se volí podle určitých pravidel. Poté se pomocí heuristiky sestupuje ze startovních pozic do lokálních minim. Takovéto metody zajišťují konvergenci teoreticky až v nekonečnu. Nemůžeme tedy zaručit nalezení minima. Takovéto algoritmy jsou většinou dobré pro použití u malých až středně rozsáhlých problémů.



Obr. 4. Stochastické prohledávání 2[3]

Další příklad stochastického prohledávání, které využívá tzv. opouštění lokálních minim. Startovní bod se volí tam, kde je minimum nejpravděpodobnější. Poté se spustí hledání minima a po nalezení se opustí podle nějakého pravidla (heuristiky) a hledací proces se opět spustí. Nalezené body se zapisují do tabulky a poté se z nich vybere nejvhodnější bod. Zde se doba hledání omezuje např. počtem opuštění minim.



Obr. 5. Stochastické prohledávání 3[3]

Taková optimalizace může být náročná z hlediska výpočetního výkonu, protože více spuštěných optimalizačních procedur může vést ke stejnému výsledku. To je z hlediska

výpočetních zdrojů plýtváním. Ideální je, aby každý bod byl nalezen jen jednou. Z tohoto důvodu je možné využít tzv. shlukování (clustering). To spočívá ve vzorkování celé oblasti a vytvoření shluků kolem podobných bodů. Poté se hledací procedura spustí v každém shluku pouze jednou. Využití shluku je znázorněno na obrázku 5. [3]

1.2.3 Evoluční algoritmy

V posledních desetiletích se s poměrným úspěchem pro hledání globálního minima funkcí užívají stochastické algoritmy zejména evolučního typu. Rozvoj evolučních algoritmů je záležitostí posledních desetiletí a je podmíněn rozvojem počítačů a pokroky v informatice. Charakteristické pro ně je to, že pracují s populací a využívají tzv. evoluční operátory, zejména tyto:

- selekce – nejsilnější jedinci z populace mají vetší pravděpodobnost přežití a předání svých vlastností
- křížení (rekombinace) – dva nebo více jedinců z populace si vymění informace a vzniknou tak noví jedinci kombinující vlastnosti rodičů
- mutace – informace zakódovaná v jedinci může být náhodně modifikována

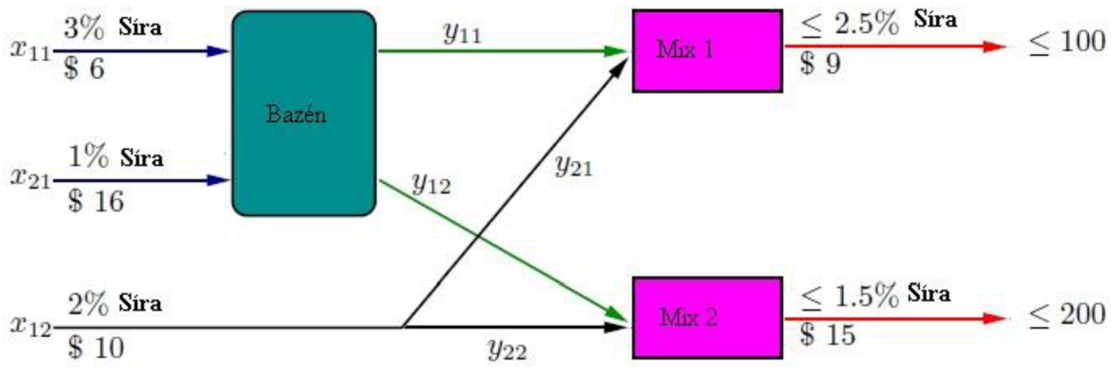
Evoluční algoritmy jsou heuristiky, které nějakým způsobem modifikují populaci tak, aby se její vlastnosti zlepšovaly. O některých třídách evolučních algoritmů je dokázáno, že nejlepší jedinci populace se skutečně přibližují ke globálnímu optimu. Evoluční algoritmy byly a jsou předmětem intenzivního výzkumu, počet a rozsah publikací z této oblasti je ohromující. Jedním z hlavních motivů jsou především aplikace v praktických problémech, které jinými metodami nejsou řešitelné.[4]

1.3 Některé problémy vyžadující globální optimalizaci

Zde uvedu příklady problémů, které vyžadují užití globální optimalizace, aby mohly být vyřešeny.

1.3.1 Problém slučování roztoků

Problém spočívá v tom, že máme 3 různé roztoky o známé koncentraci. 2 z těchto roztoků vstupují do bazénu, kde se promíchají a jejich koncentrace se srovná. Poté odcházejí ve 2 proudech pryč z bazénu. Tyto dva proudy jsou se třetím namíchány v takovém poměru, aby výsledné 2 produkty měly požadovanou koncentraci. Jde o to, že každý z roztoků má různou cenu a my potřebujeme zjistit cenu výsledných produktů a množství jednotlivých vstupů potřebných k namíchání.



Obr. 6. Roztoky[3]

Tento problém zle vyřešit optimalizací následujících rovnic:

$\min_{x,y,p}$ $6x_{11}+16x_{21}+10x_{12}-9(y_{11}+y_{21})-15(y_{12}+y_{22})$ - cena (6)

$x_{11}+x_{21}-y_{11}-y_{12}=0$ - rovnováha hmoty

$x_{12}-y_{21}-y_{22}=0$ - rovnováha hmoty

$y_{11}+y_{21}\le 100$ - náklady

$y_{12}+y_{22}\le 200$ - náklady

$3x_{11}+x_{21}-p(y_{11}+y_{12})=0$ - rovnováha síry

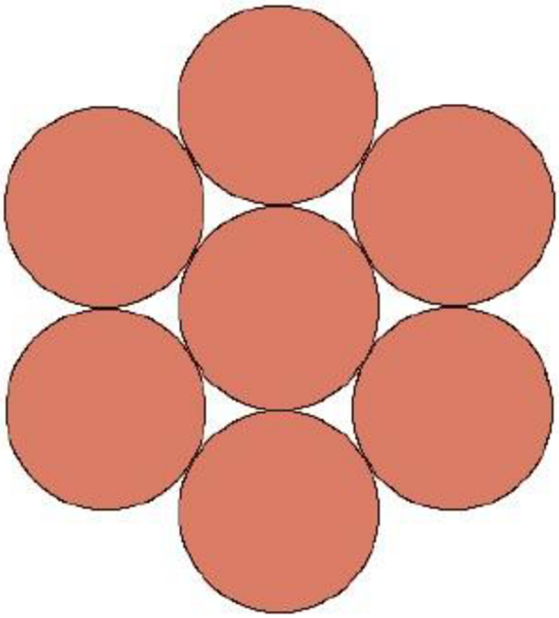
$py_{11}+2y_{21}\le 2.5(y_{11}+y_{21})$ - potřebné množství

$py_{12}+2y_{22}\le 1.5(y_{12}+y_{22})$ - potřebné množství

V této rovnici jsou x a y vstupní proudy a p je procento koncentrace síry.[3]

1.3.2 Problém počtu dotyků

Tento problém spočívá ve zjištění maximálního počtu dotyků kružnic soustředěných kolem středové kružnice ve 2D nebo koulí ve 3D bez toho, aby došlo k překrytí. Tento problém je velmi dobře znám z kombinatorické geometrie. Ve 2D se tento problém může jevit jako triviální, což vidíme na obrázku, ale ve 3D problém již tak triviální není. Tímto problémem se zabývala spousta známých matematiků a fyziků.



Obr. 7. Problém dotyku[3]

Až J. Leech dokázal, že počet dotyků ve 3D je 12. Tento problém byl již vyřešen i rozměru 4D. 2D problém popisují následující rovnice:

$$\begin{aligned} &\max && \alpha && (7)\\ &\forall i \le N && \left\|x^i\right\|_2 = 2R \\ &\forall i < j \le N && \left\|x^i - x^j\right\|_2 \ge 2R\alpha \\ & && \alpha \ge 0 \\ &\forall i \le N && x^i \in R^D \end{aligned}$$

Kde se snažíme maximalizovat α, N je počet sfér a proměnná x^i pozici i-tého kruhu kolem centrálního kruhu.[3]

1.4 Funkce Black-Box

Za Black-Box považujeme takovou funkci $f(x): R^n \to R$, u které neznáme její analytickou podobu. Black-Box funkce můžeme hodnotit podle toho, jak snadno lze určit:

- funkční hodnotu
- určitost (jednoznačnost)

- (přibližně) gradient

Black-Box funkce nemusí být za každou cenu složitá. Může být i hladká, definovaná všude a dokonce konvexní. Reálné příklady Black-Box funkcí:

- referenční kódy, kde nevíme co je uvnitř knihoven nebo spustitelných kódů
- numerické kódy obsahující parciální derivace a integrály
- experimenty reálného života:cash testy, chemické reakce…

Tyto funkce dělíme z hlediska, zda se dá jednoduše ověřit jejich funkční hodnota na:

- „cheap'' funkce, které mohou být ověřeny tisíckrát a více
- „costly'' nebo „expensive'' mohou být ověřeny jen párkrát typicky << 200

Právě pro funkce „cheap'' lze využít algoritmy založené na náhodném hledání, experimentování a deterministických algoritmech. Jako jsou kvazi-newtonovské metody, metoda kompasového hledání, simplexové metody a některé genetické algoritmy.

U optimalizace „expensive'' funkcí je třeba omezit co nejvíce snížit počet ověření hodnot. Je tedy třeba pečlivě volit další bod, který se bude ověřovat. Optimalizace samotná může být poměrně rychlá oproti samotnému ověření hodnot funkcí. Touto optimalizací se zabývají algoritmy jako EGO D. Jones, Kriging, RBF, SVM, práce Guttmana 2001, práce Holmstroma 2008. Algoritmy sloužící k optimalizaci takových funkcí se stále vyvíjejí.[5]

2 NUMERICKÉ METODY

Numerické metody jsou metody, které jsou využívané k nalezení minima nebo maxima pomocí počítačů. Je u nich kladen důraz na to, aby minimum bylo nalezeno co nejrychleji s malými nároky na výpočetní výkon a na paměťovou náročnost. Jak bylo popsáno v předchozí kapitole, tak metody mohou využívat různých principů hledání extrémů. Vždy je třeba zvolit takový konkrétní algoritmus, který nám nejlépe zaručí, že extrém bude co nejpřesněji v určitém čase nalezen. V této části budou uvedeny některé numerické optimalizační metody, jejich stručný popis a soupis jejích využití, silných a slabých stránek.[1]

2.1 Náhodné prohledávání

Náhodné prohledávání se také někdy nazývá slepý algoritmus, neboť v něm se opakovaně generuje náhodné řešení (nový bod y v souvislé oblasti D z rovnoměrného spojitého rozdělení) a zapamatovává se tehdy, když je lepší než předtím nalezené řešení. Náhodné prohledávání je příklad velmi jednoduchého stochastického algoritmu pro hledání globálního minima využívající velmi prostou heuristiku, kterou bychom mohli slovně vyjádřit jako „zkus bez rozmyslu cokoliv. Slepý algoritmus lze zapsat v jakémkoliv jazyce napsat na pár řádcích. Výhodou tohoto algoritmu je tedy jeho neobyčejná jednoduchost. Nevýhoda pak spočívá v tom, že konverguje v nekonečnu. Pokud tedy ukončíme hledaní v reálném čase, tak jeho výsledek nemusí odpovídat hodnotě minima. To je třeba po skončení nejlépe experimentálně ověřit.[4]

2.2 Simplexová metoda

Simplexová metoda je velmi jednoduchý a populární algoritmus pro hledání globálního minima. Původní verzi algoritmu publikovali Nelder a Mead v roce 1965. Od té doby bylo navrženo několik dalších modifikací simplexové metody. Klíčovým pojmem algoritmu je tzv. simplex, což je množina nekomplanárních bodů v definičním oboru funkce. V simplexu nalezneme body s nejvyšší a nejnižší funkční hodnotou a dále spočítáme těžiště. Pomocí tohoto simplexu hledáme nový bod y, který v případe, že je menší nahradí v simplexu dosud nejhorší bod a pokračuje se znovu. Pro hledání nového bodu se užívá reflexe. Reflexe znamená překlopení nejhoršího bodu přes těžiště. Pokud nový bod získaný reflexí nesplňuje podmínku, že je lepší než nejhorší bod užije se redukce, což je vlastně takové „smrštění simplexu směrem k nejlepšímu bodu, tak že vzdálenosti bodu simplexu budou poloviční. Verbálně můžeme simplexovou metodu popsat jako jakýsi pohyb simplexu v prohledávaném prostoru směrem k oblasti s nižšími funkčními hodnotami. Tento algoritmus bude konvergovat rychleji než náhodné prohledávání a bude záležet na tom, kde bude na počátku simplex umístěn.

2.3 Řízené náhodné prohledávání

Řízené náhodné prohledávání (controlled random search, CRS) je příkladem velmi jednoduchého a přitom efektivního stochastického algoritmu pro hledání minima v souvislé oblasti. Algoritmus CRS navrhl Price v sedmdesátých létech minulého století. Pracuje se s populací N bodů v prohledávaném prostoru D a z nich se heuristicky generuje nový bod y, který muže být zařazen do populace místo dosud nejhoršího bodu. Počet vygenerovaných bodu populace N je vetší než dimenze d prohledávaného prostoru D. Jako heuristiku pro generování nového bodu y užíval Price reflexi simplexu, která je známa ze simplexové metody. Z populace se vybere náhodně d + 1 nekomplanárních bodů tvořících simplex. Pokud v novém bodu y je funkční hodnota f(y) menší než je v nejhorším bodu populace, pak je tento nejhorší bod nahrazen bodem y.[4]

2.4 Genetické algoritmy

Tyto algoritmy využívají principů popsaných algoritmů (simplex, prohledávání) a k tomu využívají prvků evolučních algoritmů jako selekce, křížení a mutace za účelem nalezení optima. Využitím těchto prvků se zlepší rychlost konvergence. Dnes je takových algoritmů velké množství jako: Evoluční strategie, Diferenciální evoluce, Algoritmus SOMA, Evoluční prohledávání, Algoritmy se soutěžícími heuristikami.

2.5 Úvod do optimalizace Black-Box funkcí

V polovodičovém, automobilovém průmyslu a v mnoha dalších je kladen stále větší důraz na tvorbu a návrh produktů s využitím matematických počítačových modelů. Tyto modely umožnují využít postupy, díky kterým není třeba využívat drahých prototypů hardwaru. Tento přístup je obvykle obtížný a bývá většinou náročný na výpočetní výkon a čas. Například simulace

automobilových crash testů mohou trvat desítky hodin i déle. Návrh optimalizačních algoritmů, které si dokážou s takovými obtížnými funkcemi poradit, může být velkou výzvou pro ty největší odborníky z oboru optimalizace. Práce se bude zabývat tím, jak jen pomocí několika známých bodů ze zadaného intervalu náročné funkce sestavit model problému a následně odhadnout možné umístění optima funkce. Prezentovaná metoda využívá metodiku odezvy na průběh optimalizované funkce, založenou na modelování průběhu a možné odchylky od tohoto modelovaného průběhu. Funkce pracuje se stochastickými(náhodně vybranými) daty. V postatě je nutné nejprve jak moc se funkce mění v závislosti na vzdálenosti a směru od vybraných bodů a na základě těchto změn předpovědět nový bod, který nejlépe doplní vybrané body tak, aby chyba odhadovaného průběhu byla co nejmenší. K optimalizaci takové funkce bude nutné využít znalosti z matematiky, hlavně odvětví statistiky, znalosti tvorby algoritmů a programování.[3]

3 OPTIMALIZACE ZADANÉ FUNKCE

Nejdříve je třeba shrnout vlastnosti zadané funkce, pro kterou má být nalezeno globální minimum. Budou uvedeny a popsány zvolené optimalizační metody a zhodnocena jejich konvergence. Zvláště se bude práce věnovat stochastickému modelu, který je představen v části 2.5.

3.1 Vlastnosti zadané funkce

Zadaná funkce má následující vlastnosti:

- Funkce je typu Black-Box, to znamená, že neexistuje její funkční předpis ani není možné určit gradient nebo by byl nepoužitelný. To lze popsat rovnicí $f : X \subseteq R^n \rightarrow R, x \rightarrow f(x)$. (8)

Obr. 8. Problém funkce Black-Box[6]

- Funkce je z hlediska ověření funkční hodnoty náročná tedy „expensive''. Z tohoto důvodu je tedy nutné najít takovou optimalizační metodu, která zajistí rychlou konvergenci alespoň k uspokojivému optimu a zároveň s minimálním počtem ověření.
- Funkce je vždy zadána na intervalu $L < x_1 \leq x_2 < H$. Mohu předpokládat vlastnost $f(x_1, x_2) = f(x_2, x_1)$. (9)
- Funkce je na intervalu spojitá, ale její funkční hodnoty se rychle mění, takže z hlediska optimalizace by ji bylo možné považovat za nespojitou.

Z uvedených vlastností zle předpokládat, že velké množství optimalizačních metod nebude vhodných k nalezení optima (v řešeném případě jejího minima) hodnoty této funkce.

3.2 Stochastický model

Metoda předpokládá existenci ověřených bodů deterministické funkce *k* proměnných v *n* bodech. Ke každému bodu $\mathbf{x}^{(i)}$ se předpokládá jeho funkční hodnota $y^{(i)} = y(\mathbf{x}^{(i)})$, kde i = 1….n. Tato optimalizační technika předpokládá, že optimalizovaná funkce odpovídá následujícímu modelu:

$$y\left(x^{(i)}\right) = \sum_{h} \beta_h f_h\left(x^{(i)}\right) + \varepsilon^{(i)} \tag{10}$$

Kde f_h…lineární nebo nelineární funkce, β_h…neznámý koeficient, který má být odhadnut a $\varepsilon^{(i)}$… je normálně distribuovaná nezávislá chyba.

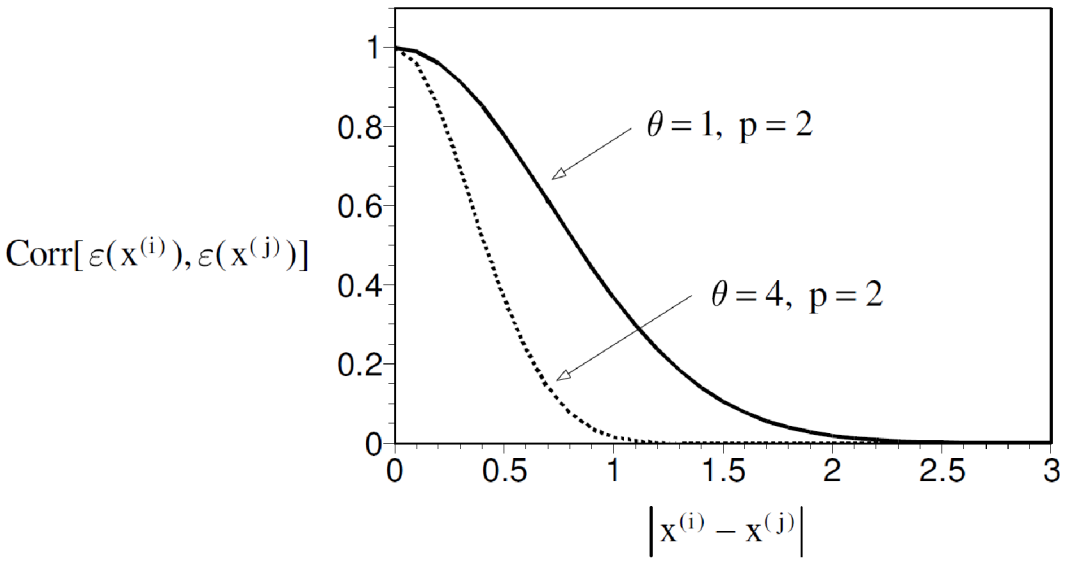
Tato metoda předpokládá, že chyba není až tak náhodná ale souvisí se vzdáleností mezi sousedícími body. V modelu se nevyužívá euklidovská metrika ale metrika speciální, znázorněná následujícím předpisem:

$$d\left(x^{(i)}, x^{(j)}\right) = \sum_{h=1}^{k} \theta_h \left| x_h^{(i)} - x_h^{(j)} \right|^{ph} \qquad \left(\theta_h \geq 0, ph \in [1,2]\right) \tag{11}$$

S použitím této metriky je následná korelace mezi chybou v bodě $\mathbf{x}^{(i)}$ a $\mathbf{x}^{(j)}$ následující:

$$Corr\left(\varepsilon\left(x^{(i)}\right), \varepsilon\left(x^{(j)}\right)\right) = \exp\left[-d\left(x^{(i)}, x^{(j)}\right)\right] \tag{12}$$

Pokud je vzdálenost mezi body malá, tak se korelace blíží 1 a pokud je velká směřuje k 0. Parametr θ_h v předpisu metriky reprezentuje význam proměnné. Exponent ph souvisí s tím jak je funkce hladká. Blíže k 1 je lépe používat u méně hladkých funkcí. Následující graf znázorňuje vliv parametrů θ_h a ph na velikost korelace v závislosti na vzdálenosti bodů.[2]



Obr. 9 Vliv parametrů θ_h a ph na hodnotu korelace[3]

S využitím předchozích rovnic lze sestavit korelační matici **R**, která obsahuje hodnotu všech korelací mezi všemi body s ověřenou funkční hodnotou. S využitím této matice lze předpokládat průběh funkce podle následujícího předpisu:

$$\hat{y}(x^*) = \hat{u} + r' R^{-1} (y - 1\hat{u}) \tag{13}$$

x*…vektor souřadnic odhadovaného bodu

$\hat{\mu}$ …střední hodnota funkce

r´…vektor korelací mezi odhadovaným bodem a body s ověřenou funkční hodnotou

R^{-1}... inverzní korelační matice

y...vektor funkčních hodnot v ověřených bodech

1… vektor hodnot 1

Střední hodnotu lze určit ze vztahu:

$$\hat{\mu} = \frac{1' R^{-1} y}{1' R^{-1} 1} \tag{14}$$

R^{-1}... inverzní korelační matice

y...vektor funkčních hodnot v ověřených bodech

1… vektor hodnot 1

Z předpisu pro předpokládaný průběh(prediktor) je jasné, že předpokládaná funkční hodnota je dána součtem střední hodnoty funkce a odhadu založeném na korelaci ověřených bodů s odhadovaným bodem. Pokud bude korelace 0, tak se zkrátka předpokládá, že $\hat{y} = \hat{u}$. Korelace ale ovlivňuje přesnost prediktoru. Je zřejmé, že chyba předpokládaného průběhu bude tím větší, čím bude větší vzdálenost mezi ověřenými body a odhadovaným bodem. Velikost této chyby odráží následující předpis průměrné kvadratické chyby prediktoru:

$$s^2(x^*) = \sigma^2 \left[1 - r' R^{-1} r + \frac{(1 - 1' R^{-1} r)^2}{1' R^{-1} 1} \right] \tag{15}$$

x*…vektor souřadnic odhadovaného bodu

σ^2 …rozptyl směrodatné odchylky

r, r´…vektor korelací mezi odhadovaným bodem a body s ověřenou funkční hodnotou

R^{-1}... inverzní korelační matice

1, 1´… vektor hodnot 1

Rozptyl směrodatné odchylky lze určit ze vztahu:

$$\hat{\sigma}^2 = \frac{(y - 1\hat{\mu})' R^{-1} (y - 1\hat{\mu})}{n} \tag{16}$$

y...vektor funkčních hodnot v ověřených bodech

$\hat{\mu}$ …střední hodnota funkce

$\mathbf{R}^{-1}$... inverzní korelační matice

n…počet ověřených bodů

Následující předpis, který je velmi důležitý pro následnou optimalizaci se nazývá očekávané zlepšení. Hodnota této funkce udává, kde nejpravděpodobněji by se mohlo nacházet minimum funkce. Čím vyšší hodnota této funkce tím vyšší pravděpodobnost, že se v tomto bodě bude nacházet hodnota nižší než nejnižší dosud ověřená funkční hodnota. Následující předpis je předpisem předpokládaného zlepšení:

$$E[I(x)] = (f_{\min} - \hat{y})\Phi\left(\frac{f_{\min} - \hat{y}}{s}\right) + s\phi\left(\frac{f_{\min} - \hat{y}}{s}\right) \tag{17}$$

f_{min}…nejnižší ověřená funkční hodnota

$\hat{y}$ …hodnota prediktoru v bodě **x**

s…směrodatná odchylka(odmocnina z kvadratické chyby prediktoru)

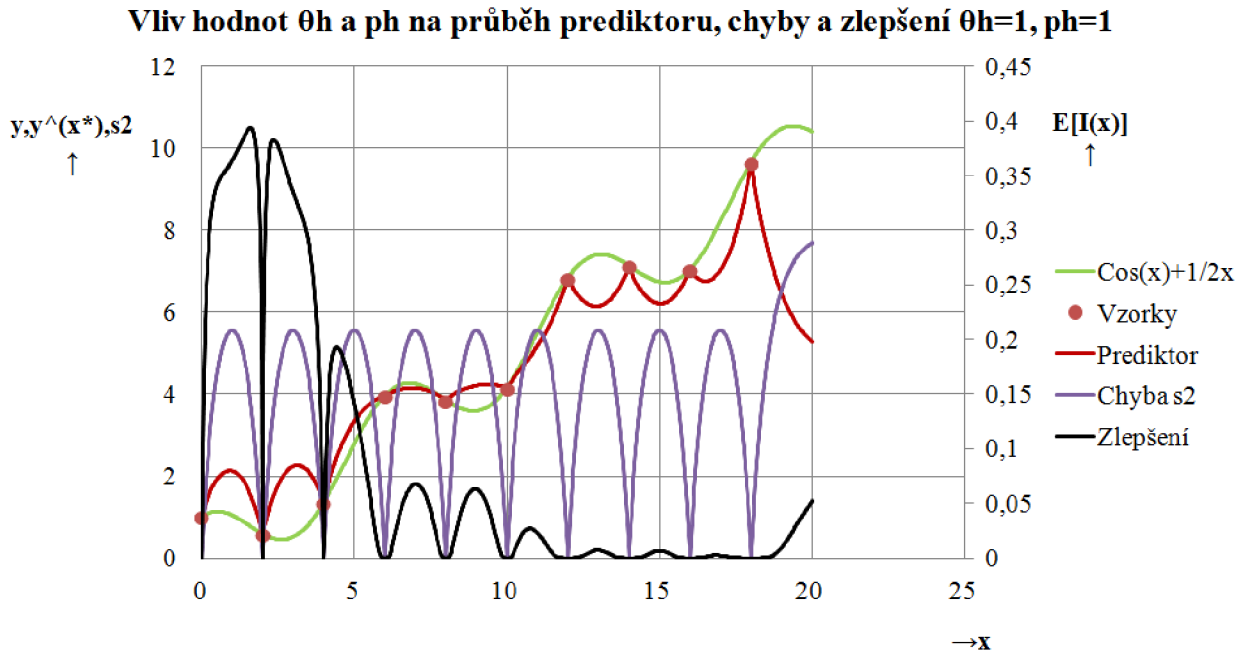
Φ…normální distribuční funkce

ϕ…funkce normálního rozdělení

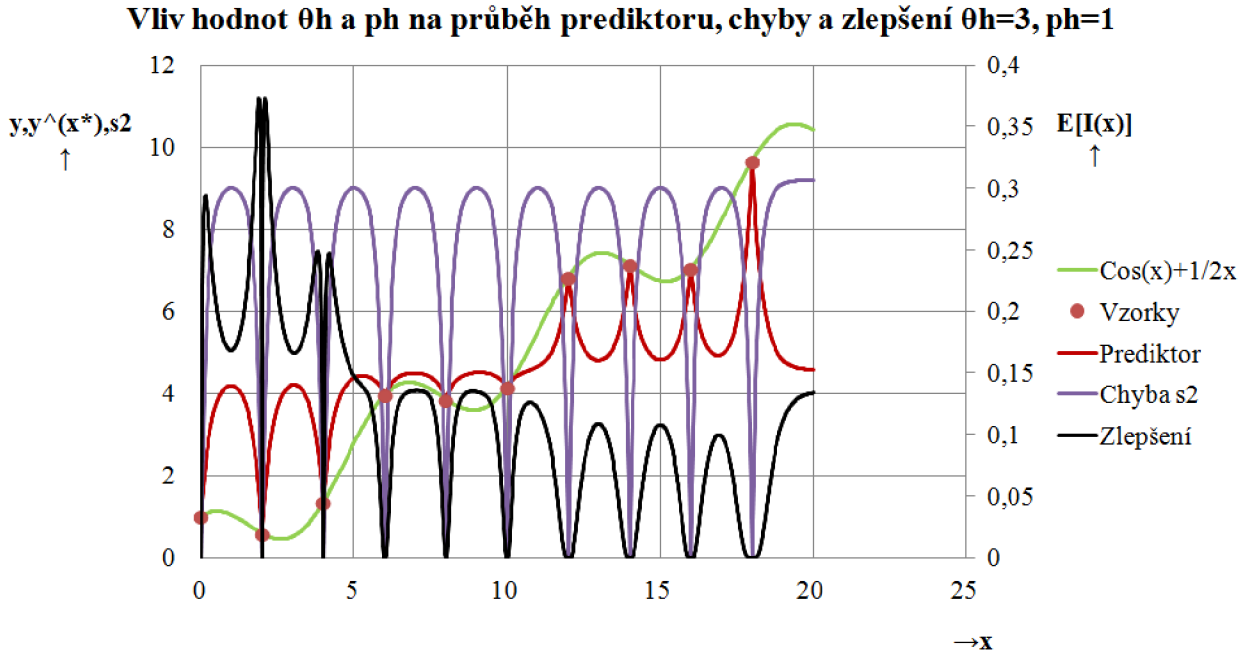
3.3 Názorné průběhy jednotlivých implementovaných funkcí

V následujících grafech je vidět vliv volitelných parametrů θh a ph. Funkčnost je testována na funkci $y(x) = \cos(x) + \frac{1}{2}x$. Tato byla zvolena, protože obsahuje velké množství lokálních minim. V grafech jsou uvedeny průběhy prediktoru, jeho chyby a průběh očekávaného zlepšení funkce. Funkce je vykreslena na intervalu <0,20>. Vzorky pro výpočet prediktoru, jeho chyby a zlepšení jsou zde zvoleny pravidelně vždy s krokem 2. Parametr θh by měl ovlivňovat průběh funkcí z hlediska vzdáleností. Pokud je tento parametr velký, tak i při malé vzdálenosti mezi body bude výsledná korelace malá. Pomocí tohoto parametru tedy lze uměle měnit vzdálenost mezi body. Druhý parametr ph souvisí s tím jak hladký průběh má vyšetřovaná funkce. Tento parametr je vhodné nastavit blíže 2, pokud má funkce hladký průběh. Pokud je funkce méně hladká je vhodné tento parametr volit blíže 1.

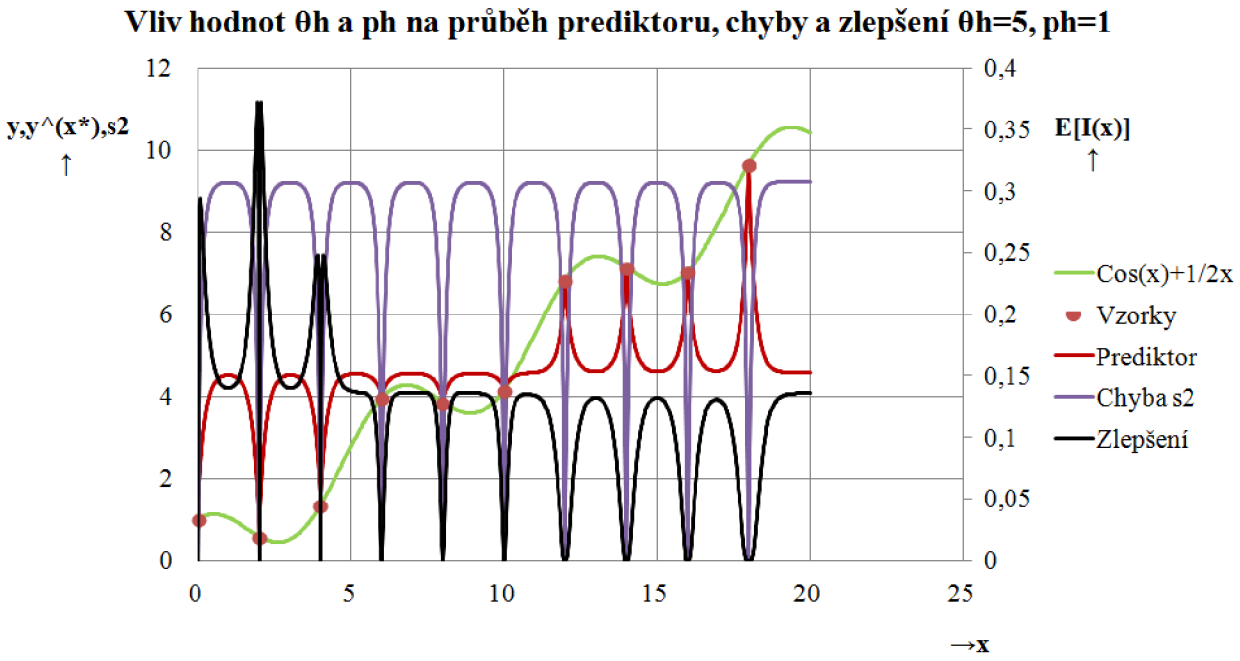
V grafech má vyšetřovaná funkce průběh světle zeleně. Vzorky ze kterých jsou určeny průběhy prediktoru, chyby a zlepšení jsou červené body na tomto průběhu. Předpokládaný průběh funkce, tedy prediktor, je červený. Chyba prediktoru má v grafech fialovou barvu a konečně předpokládané zlepšení je černou barvou. Předpokládané zlepšení má osu y odlišnou od ostatních průběhů(osa vpravo).



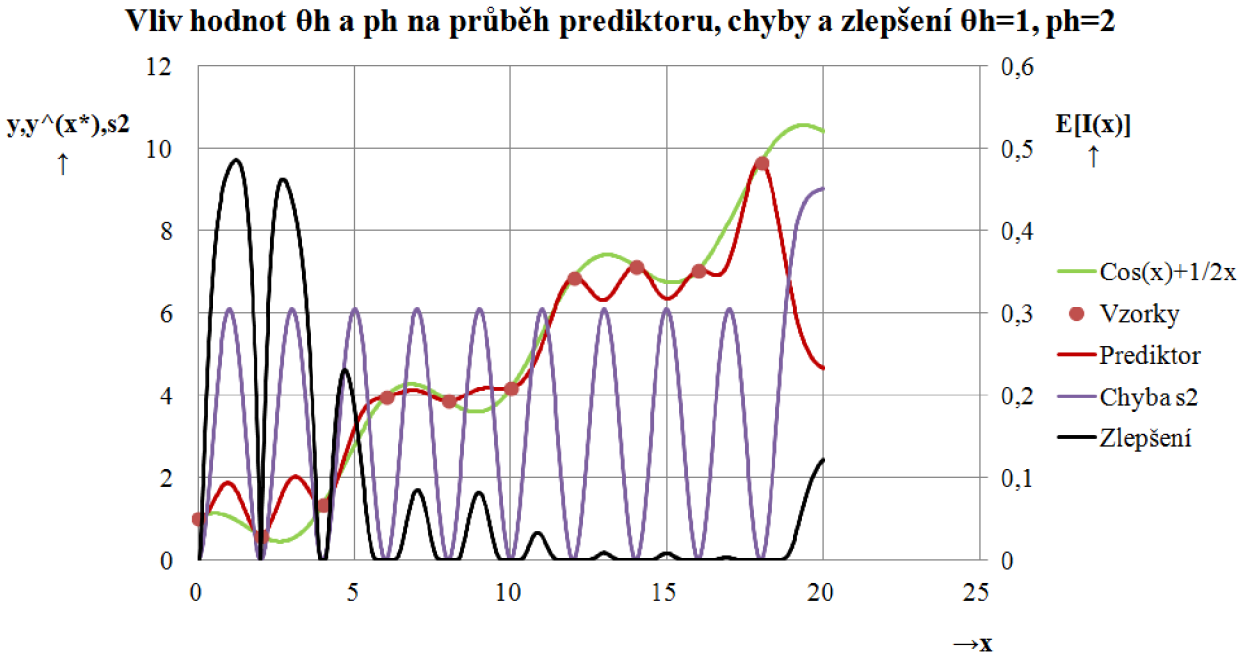
Obr. 10 Vliv parametrů θh a ph na průběh prediktoru, jeho chyby a zlepšení. θh = 1, ph = 1



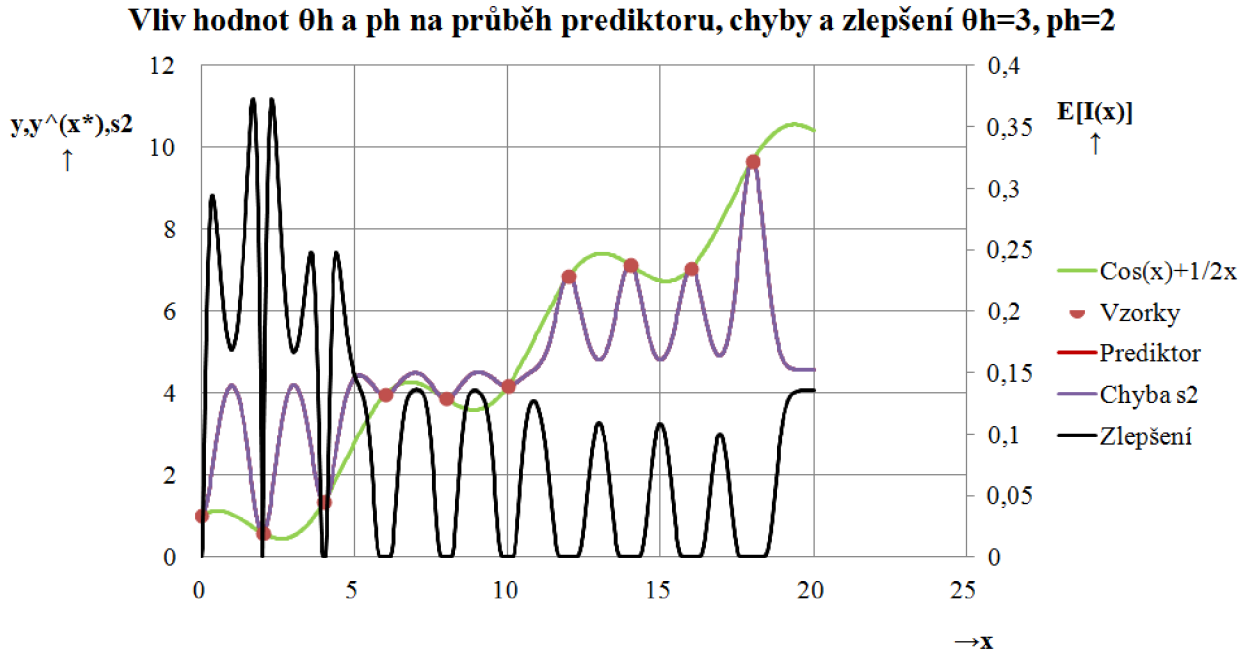
Obr. 11 Vliv parametrů θh a ph na průběh prediktoru, jeho chyby a zlepšení. θh = 2, ph = 1



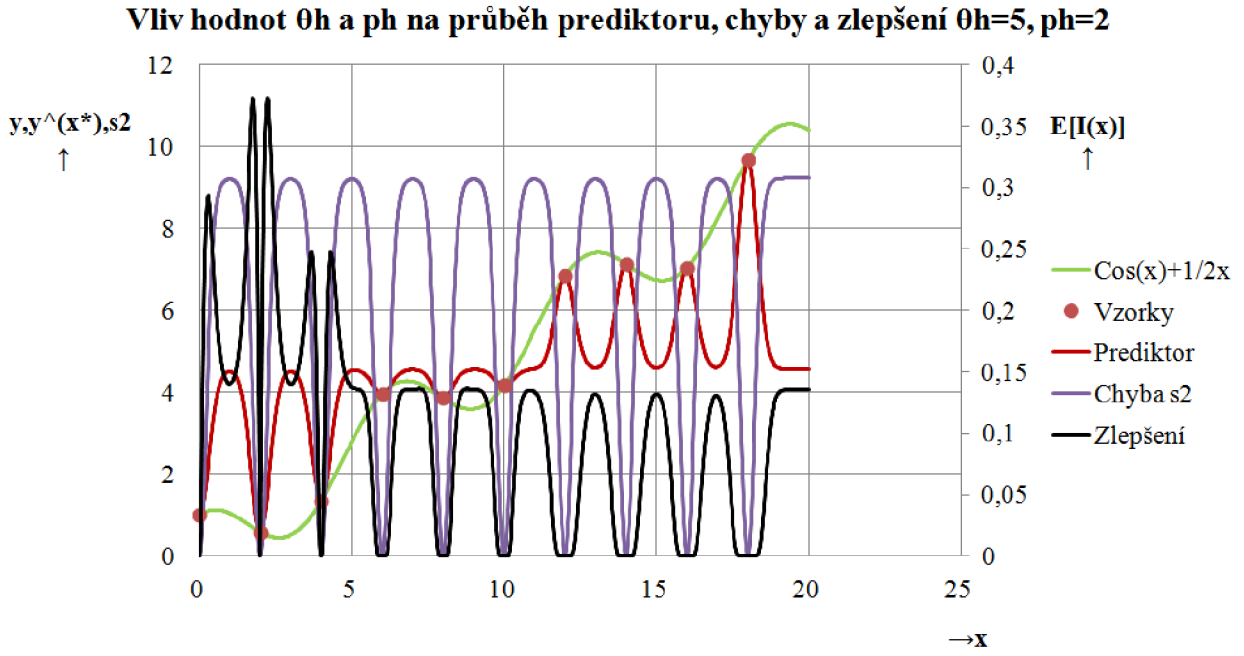
Obr. 12 Vliv parametrů θh a ph na průběh prediktoru, jeho chyby a zlepšení. θh = 5, ph = 1



Obr. 13 Vliv parametrů θh a ph na průběh prediktoru, jeho chyby a zlepšení. θh = 1, ph = 2



Obr. 14 Vliv parametrů θh a ph na průběh prediktoru, jeho chyby a zlepšení. θh = 3, ph = 2



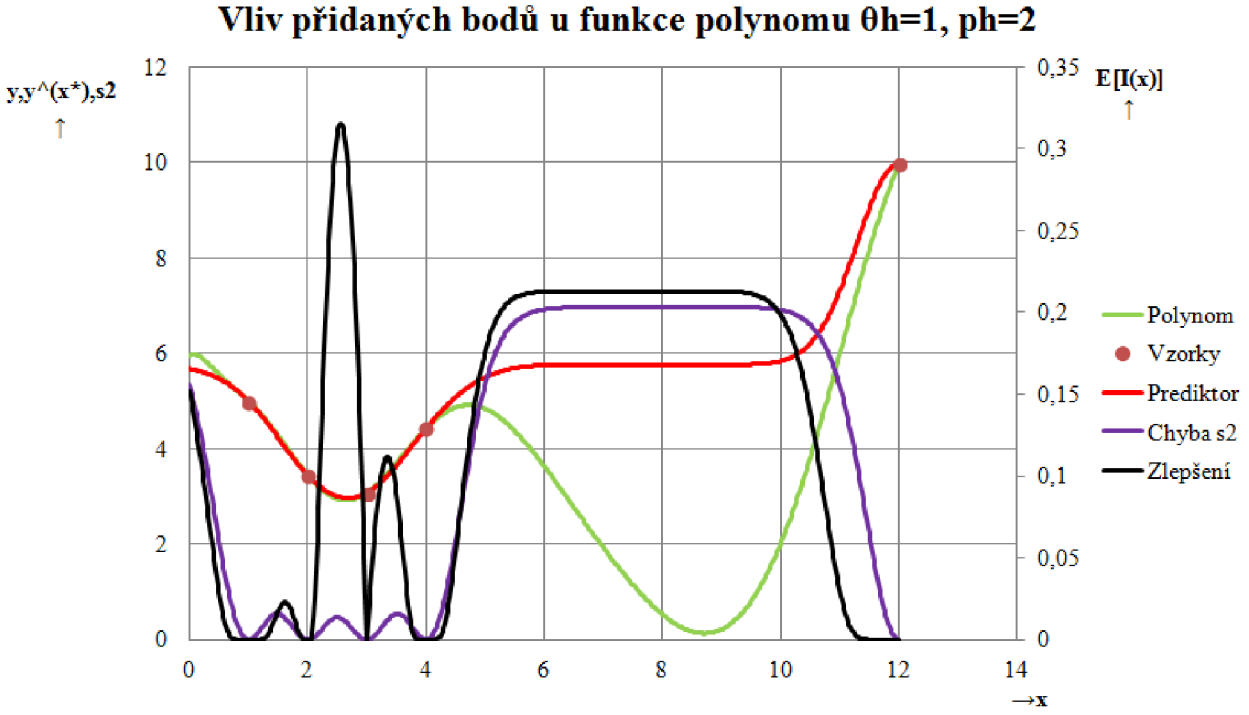
Obr. 15 Vliv parametrů θh a ph na průběh prediktoru, jeho chyby a zlepšení. θh = 5, ph = 2

Z grafů na obrázcích 10-15 je vidět jak parametry θh a ph ovlivňují jednotlivé průběhy. Je tedy důležité při následné optimalizaci tyto parametry vhodně zvolit. Zde by bylo vhodné

3.3.1 Vliv přidaných bodů na průběh funkcí

Zde bude demonstrován vliv přidání dalšího bodu(vzorku) do místa, kde je očekávané zlepšení funkce největší. V následujících grafech je vyšetřovanou funkcí polynom 15 stupně z důvodu srovnání správnosti průběhu s průběhem v práci [9]. Parametry θh a ph byly zvoleny po otestování jako θh = 1 a ph = 2. Polynom je ve tvaru $f(x)=a+bx+cx^2+dx^3+...+px^{15}$. Koeficienty jsou následující:

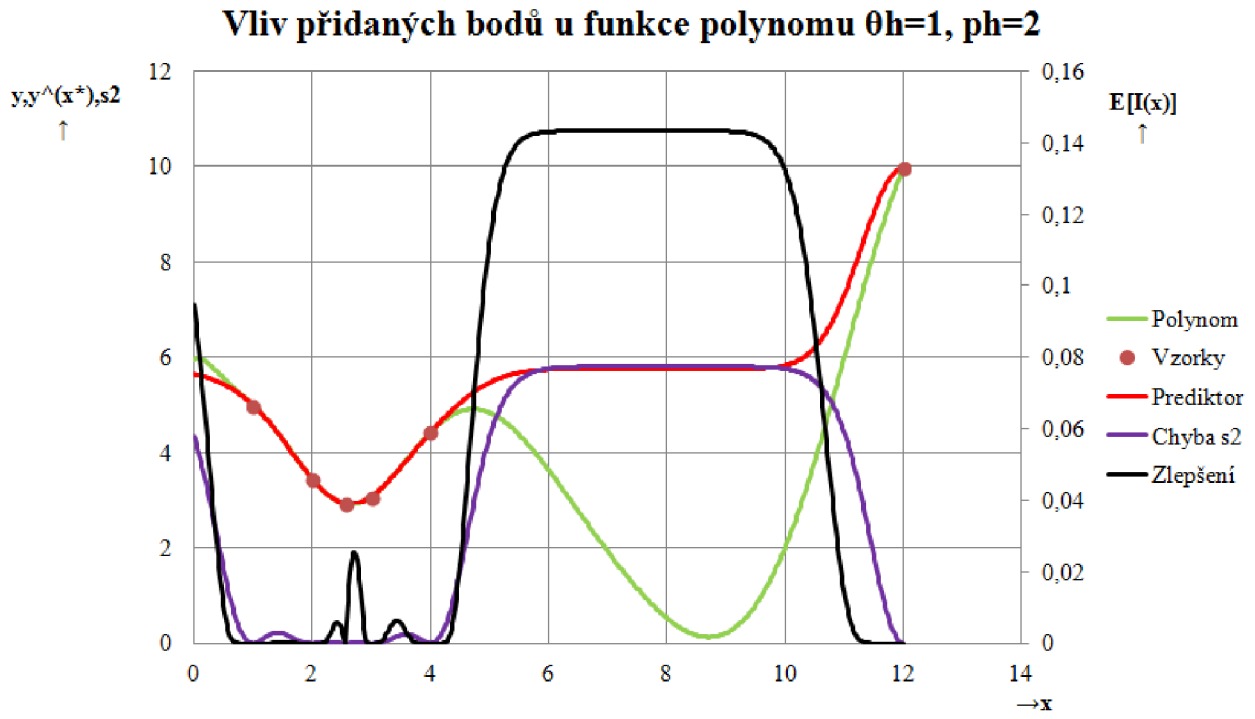
a = 5.95121023505635
b = 0.887754463839314
c = -7.70721913502845
d = 13.8332014875226
e = -13.3092977965037
f = 7.36278255763972
g = -2.49262822525486
h = 0.537433122512059
i = -0.0737094124205644
j = 0.00586951381472542
k = -0.000136327435548531
l = -2.38489156809096 x 10^{-5}
m = 2.96118529300681 x 10^{-6}
n = -1.60787297342575 x 10^{-7}
o = 4.51685442306565 x 10^{-9}
p = -5.33679890011691 x 10^{-11}



Obr. 16 Vliv přidaných bodů na průběhy funkcí θh = 1,ph =2. Počet vzorků 5

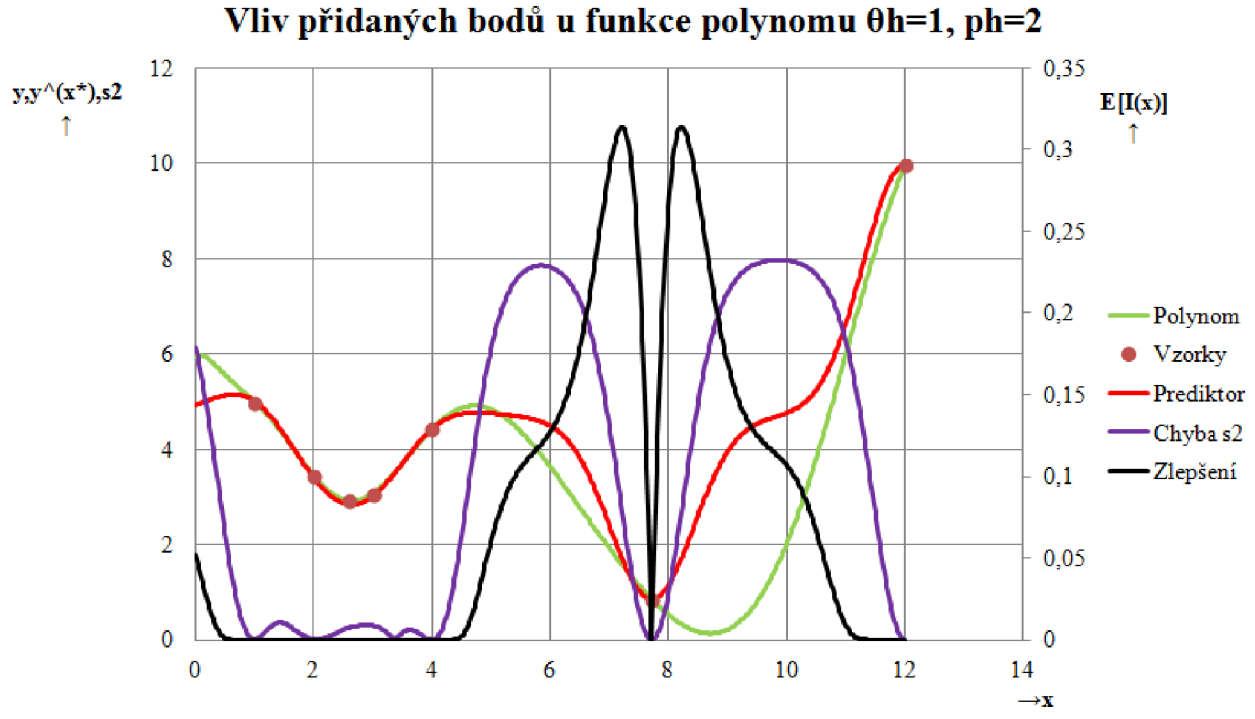
Z průběhu očekávaného zlepšení je patrné, že k největšímu zlepšení by mělo dojít po přidání bodu do souřadnic, kde x = 2,56. V tomto bodě se očekává minimum funkce polynomu.

V následujícím obrázku bude znázorněno, jaký bude mít vliv přidání tohoto bodu na průběhy funkcí.



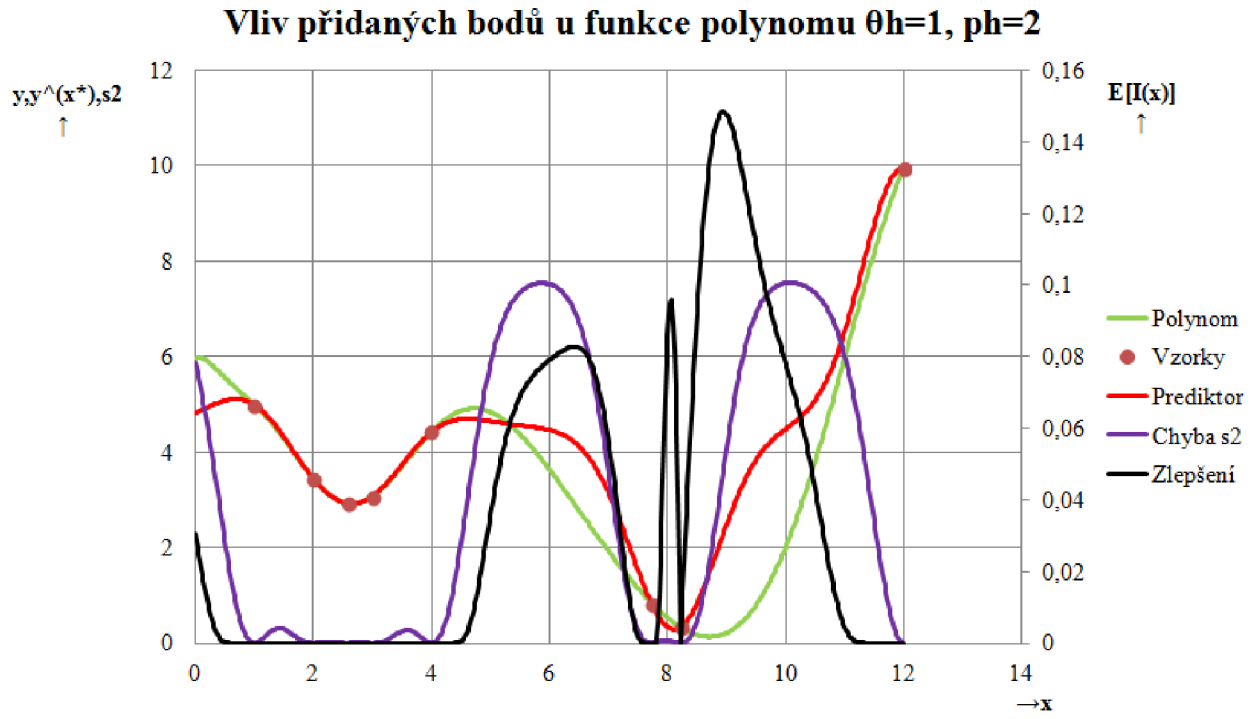
Obr. 17 Vliv přidaných bodů na průběhy funkcí θh = 1,ph =2. Počet vzorků 6

Z grafu lze odečíst, že největší změna nastala v průběhu očekávaného zlepšení. Je třeba si všimnout, že rozlišení osy pro zlepšení značně kleslo. Nyní tento průběh naznačuje, že pro nalezení minima by měl být bod přidán do souřadnic kde x = 7,72. V dalším grafu bude opět tento bod přidán.



Obr. 18 Vliv přidaných bodů na průběhy funkcí θh = 1,ph =2. Počet vzorků 7

V grafu je patrné, že se opět nejvíce změnil průběh očekávaného zlepšení. Průběh prediktoru a jeho chyby se také značně změnil. Rozlišení osy pro zlepšení se opět změnilo. V posledním grafu bude přidán poslední bod, kde má zlepšení maximální hodnotu a to v x = 8,22.



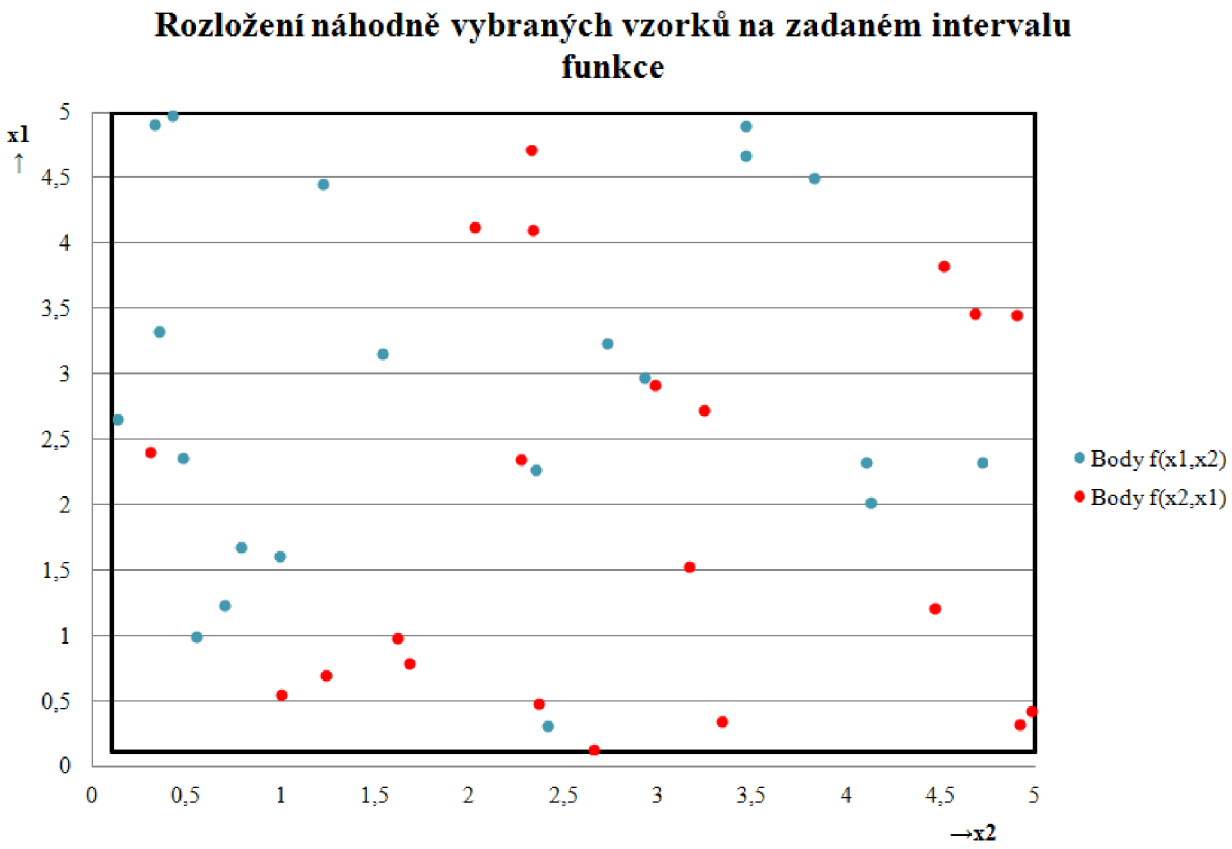
Obr. 19 Vliv přidaných bodů na průběhy funkcí θh = 1,ph =2. Počet vzorků 8

Na posledním průběhu jde vidět, že opět největší změny dostal průběh očekávaného zlepšení. Poslední přidaný bod měl funkční hodnotu f(8,22) = 0,330082. To vhledem k minimu funkce, které je v bodě f(8,72) = 0,128582 je chyba(vzhledem k maximu a skutečnému minimu na intervalu<0,12>) δ = 2.023%.

Účelem optimalizace Black-Box funkce je co nejmenší počet ověření skutečné hodnoty funkce. Zde bylo třeba ověřit hodnotu ve 3 bodech, abychom nalezli minimum s přesností δ = 2.023%. V další části práce bude znázorněno jak prediktor, chyba a zlepšení funguje se zadanou funkcí k optimalizaci. Poté bude přistoupeno k hlavní optimalizaci, kdy bude vybrán nejvhodnější interval pro minimum funkce pouze z průběhu chyby a prediktoru a očekávaného zlepšení. Tím by mělo dojít k minimalizaci bez nutnosti ověřování funkčních hodnot.

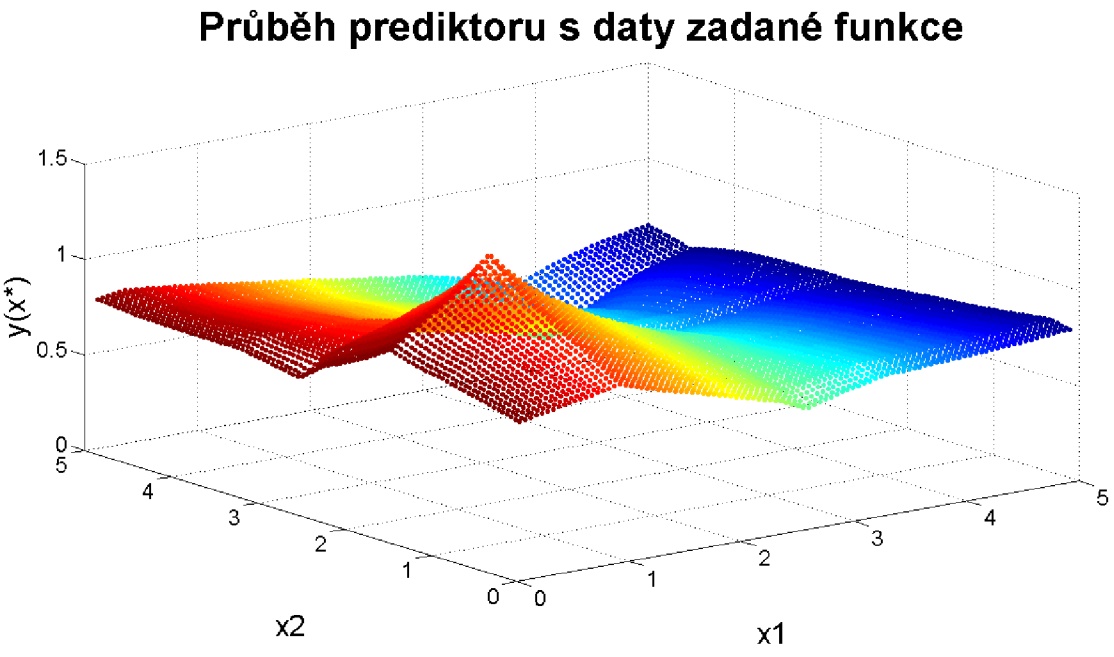
3.4 Aplikace metody na zadanou funkci

Vzorky pro zadanou funkci byly na rozdíl od předchozích případů vybrány náhodně na zadaném intervalu funkce. Náhodné rozložení je patrné na následujícím obrázku 20. V dalších grafech bude znázorněn průběh prediktoru, jeho chyby a očekávaného zlepšení funkce. Poté co budou získány tyto průběhy, bude možné přistoupit k vlastní optimalizaci. Vzhledem k tomu, že lze přepokládat vlastnost $f(x_1, x_2) = f(x_2, x_1)$, tak mohu sadu vzorků o tyto body rozšířit. To by mělo přispět k větší přesnosti průběhů. Tyto vzorky jsou na obrázku **červeně**.

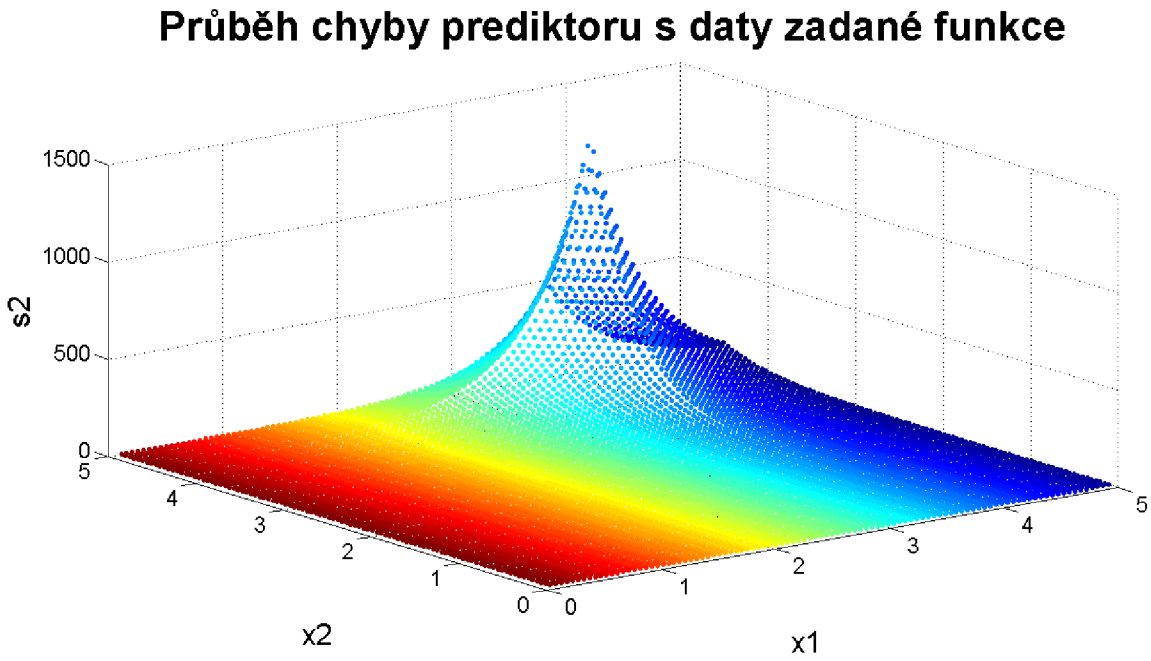


Obr. 20 Rozložení náhodných vzorků na intervalu funkce zadané k optimalizaci

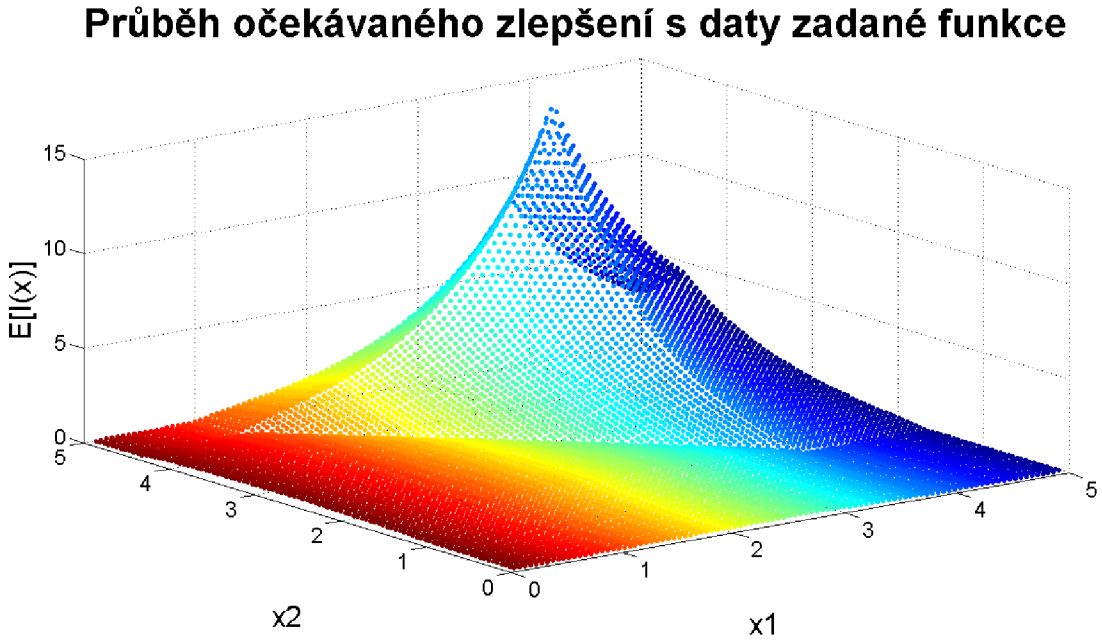
3.4.1 Průběh prediktoru, chyby a zlepšení s daty zadané funkce θh = 1,ph =1



Obr. 21. Průběh prediktoru s daty funkce zadané k optimalizaci θh=1 ph=1

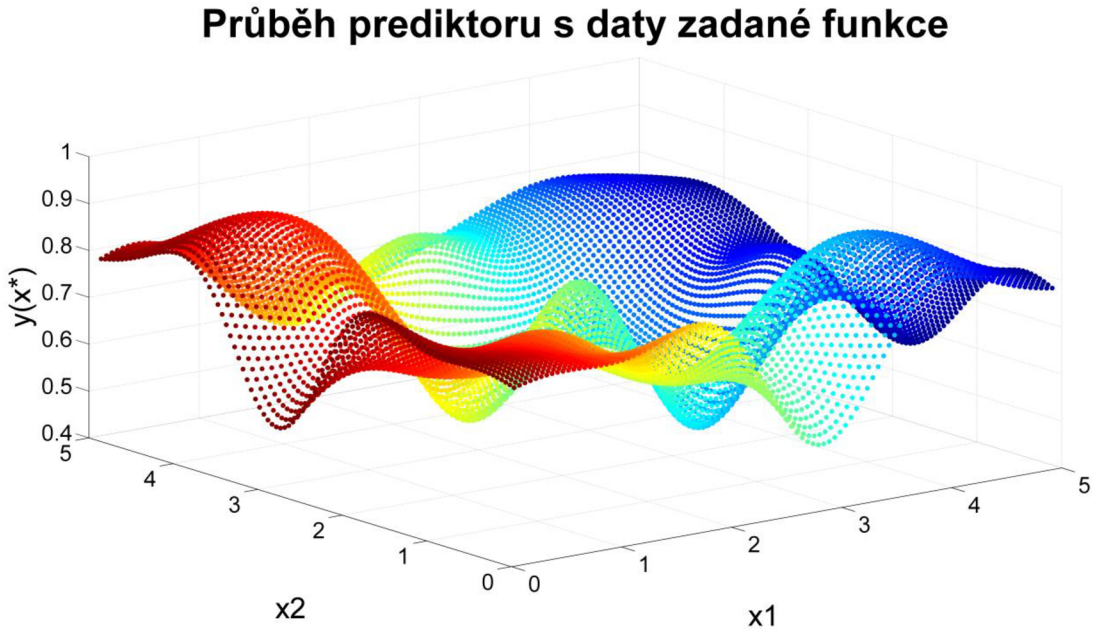


Obr. 22. Průběh chyby prediktoru s daty funkce zadané k optimalizaci θh=1 ph=1

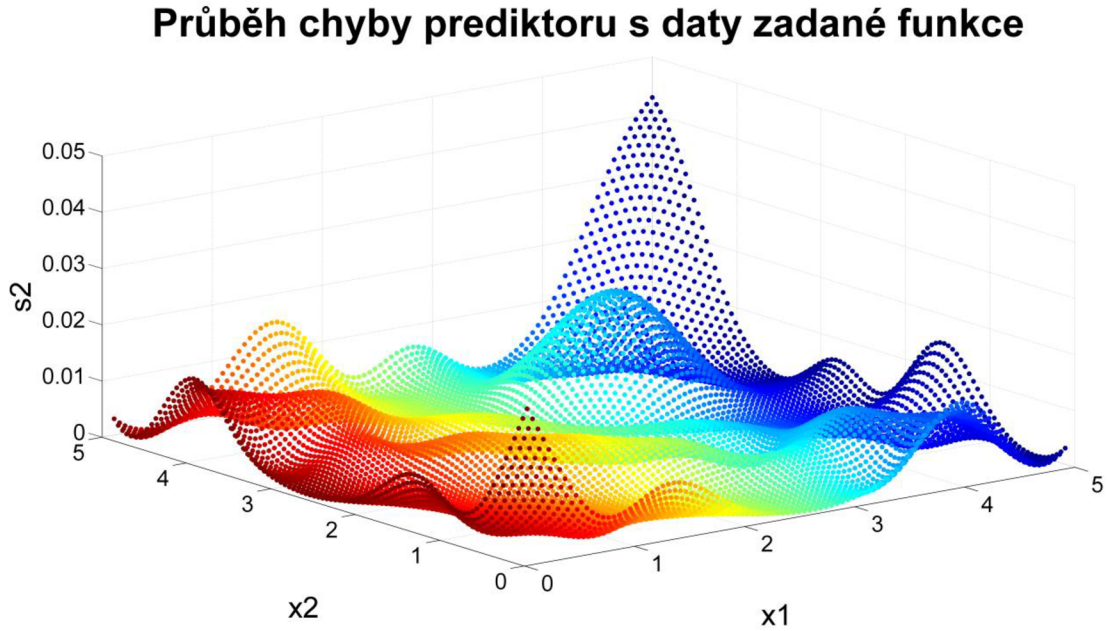


Obr. 23. Průběh očekávaného zlepšení s daty funkce zadané k optimalizaci θh=1 ph=1

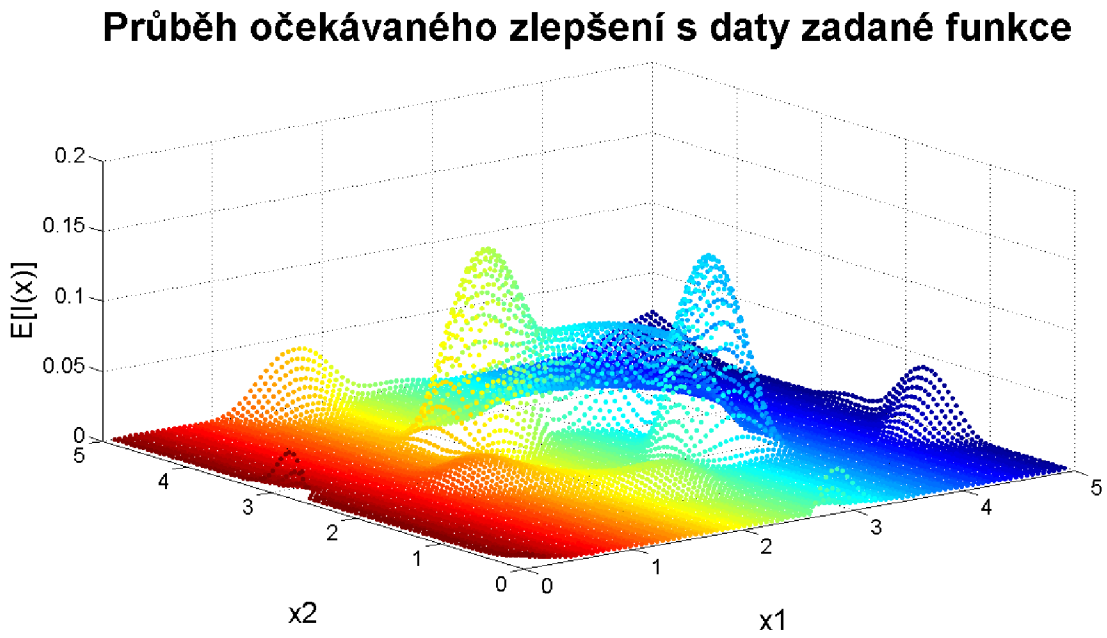
3.4.2 Průběh prediktoru, chyby a zlepšení s daty zadané funkce θh = 1,ph =2



Obr. 24. Průběh prediktoru s daty funkce zadané k optimalizaci θh=1 ph=2



Obr. 25. Průběh chyby prediktoru s daty funkce zadané k optimalizaci θh=1 ph=2



Obr. 26 Průběh očekávaného zlepšení s daty funkce zadané k optimalizaci θh=1 ph=2

Je vidět, že průběhy s hodnotami parametrů θh=1 a ph=2 vypadají lépe než s hodnotami θh=1 a ph=1. Funkce se však bude optimalizovat v obou případech, poté se oba porovnají z hlediska času a konvergence.

3.5 Optimalizační algoritmus

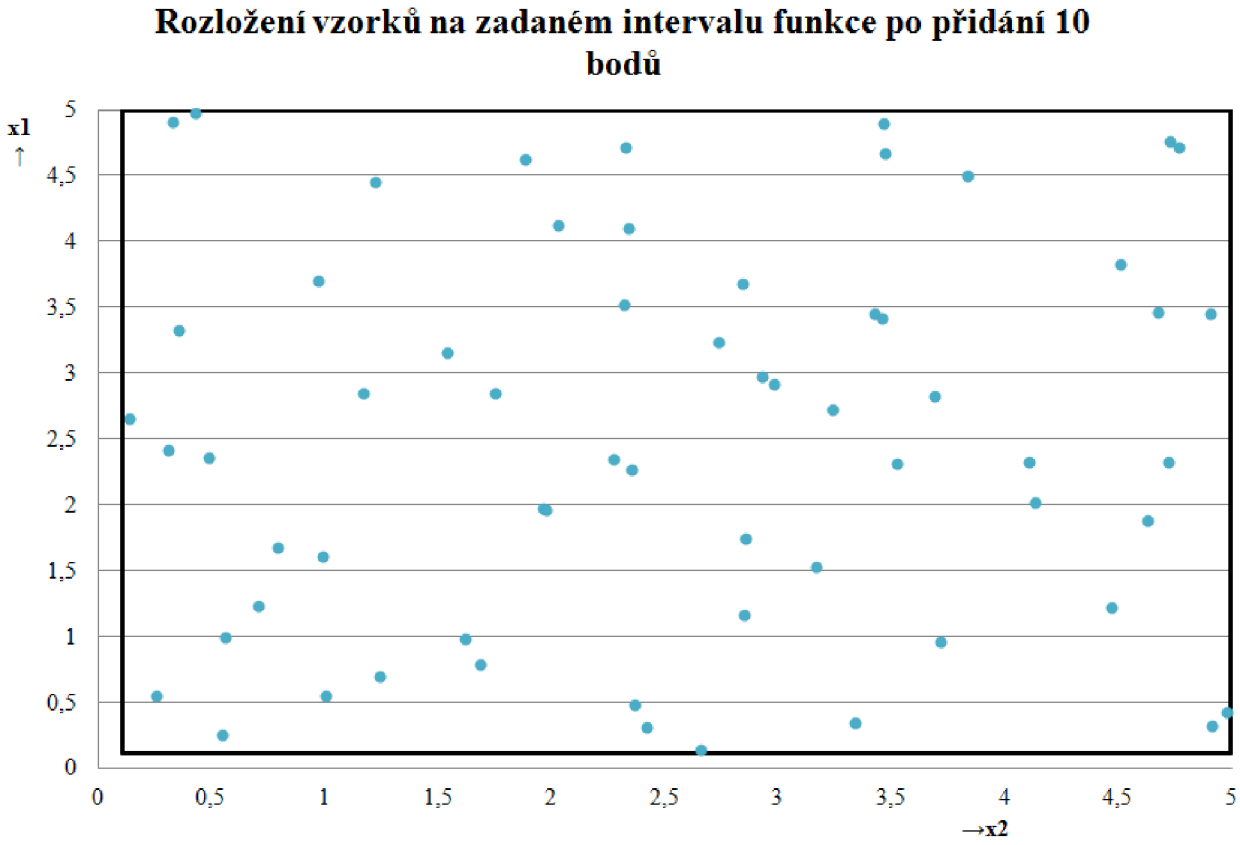
Implementovaný optimalizační algoritmus je založený na metodě „Branch and Bound´´. Vstupními parametry optimalizační funkce je interval, na kterém bude probíhat optimalizace, pole ověřených vzorků zadané funkce, počet těchto bodů a řád problému.

Princip funkce je takový, že nejprve je interval rozdělen na menší intervaly. Ve 2D případě to jsou 4 intervaly. Poměr rozdělení těchto intervalů jsem zvolil v poměru zlatého řezu. V každém tomto intervalu se vypočítá prediktor a jeho chyba s možností nastavení hustoty vzorkování. Z každého intervalu je vybrána nejnižší hodnota prediktoru a nejvyšší hodnota jeho chyby. Tyto hodnoty jsou dosazeny do předpisu pro předpokládané zlepšení. Vypočítaná hodnota nám tedy musí udávat, k jakému největšímu zlepšení může v tomto intervalu dojít. Interval, který má hodnotu zlepšení největší je pak vybrán do další interace funkce a děj se opakuje. Tímto způsobem si určíme velice malý interval, kde by po přidání bodu mělo dojít k nalezení minima funkce. Do tohoto intervalu je přidán bod.

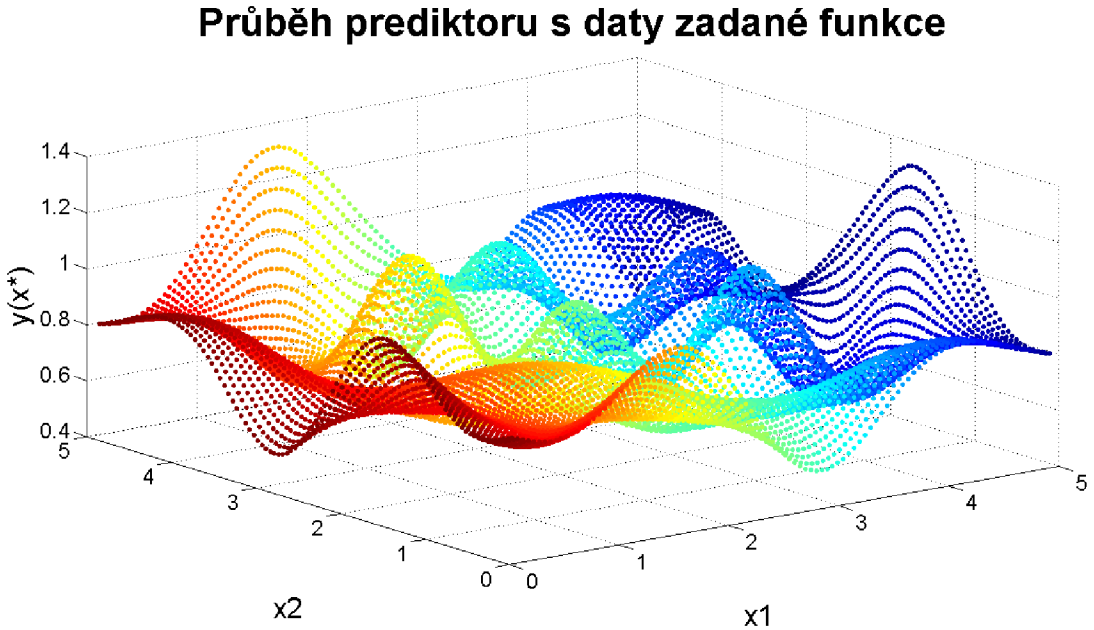
Celý předchozí princip se opakuje, dokud není přidán požadovaný počet bodů nebo by bylo možné ukončit algoritmus po dosažení požadované hodnoty chyby. V následujících bude znázorněno, kam byly body přidány a jak se měnily průběhy jednotlivých funkcí s množstvím přidaných bodů. Algoritmus opět využívá symetrie $f(x_1, x_2) = f(x_2, x_1)$, tzn. přidáním jednoho bodu se získají vlastně body dva podle osy symetrie.

3.5.1 Průběh funkcí při optimalizaci

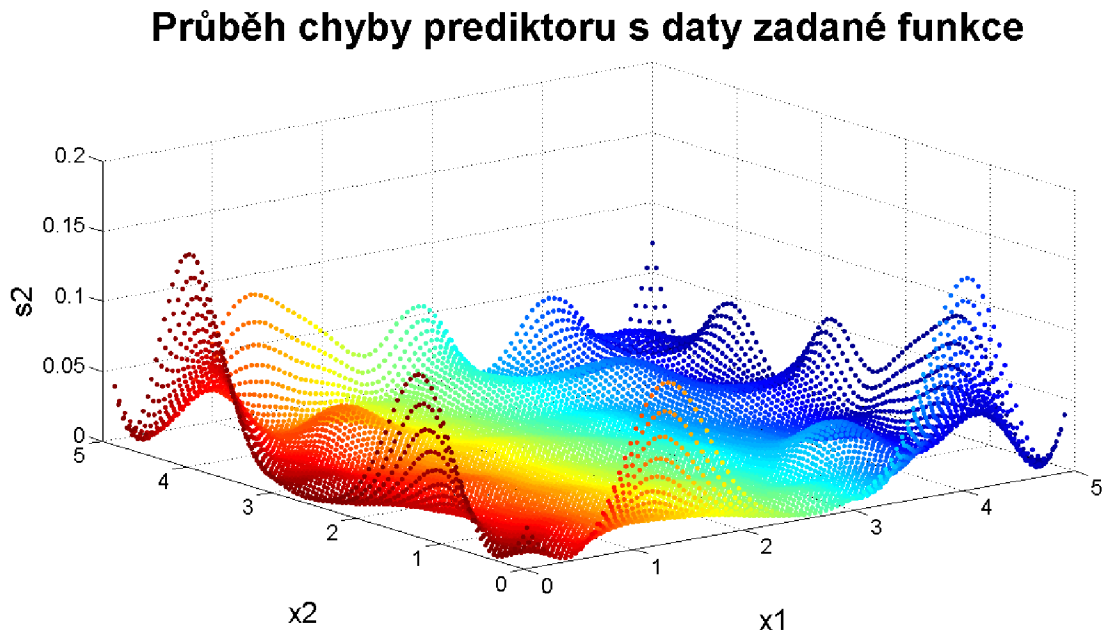
Výchozí stav navzorkování zadaného intervalu lze vidět na obrázku 20. Na dalším obrázku bude znázorněno, kam byly přidány body optimalizačním algoritmem a také bude znázorněno, jak se změnily průběhy prediktoru, jeho chyby a zlepšení na zadaném intervalu.



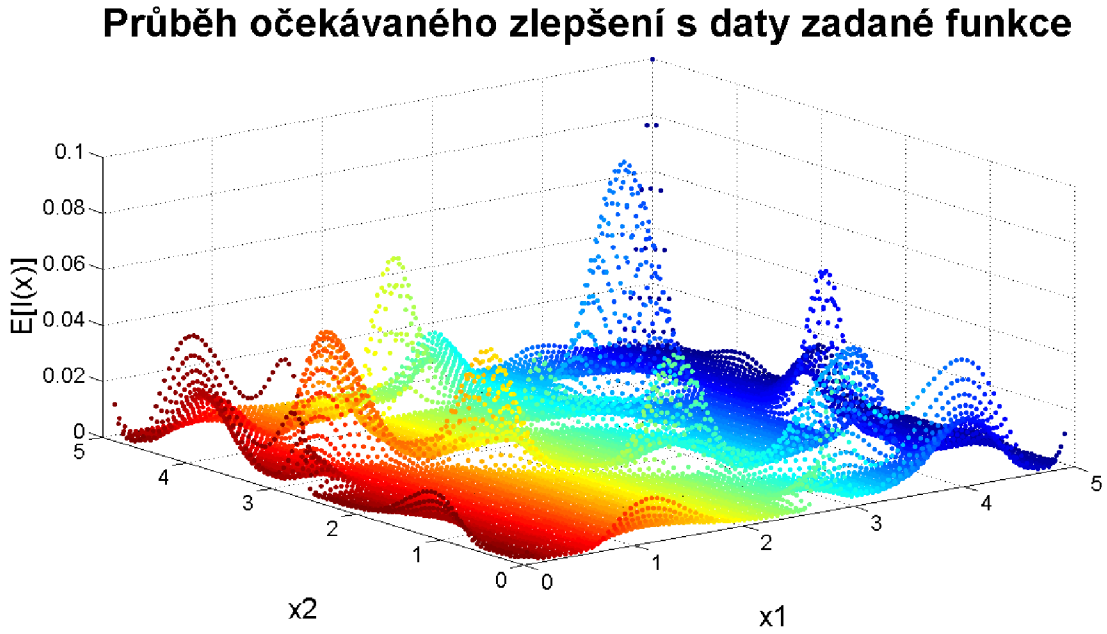
Obr. 27. Rozložení vzorků na intervalu po přidání 10 bodů



Obr. 28. Průběh prediktoru po přidání 10 bodů

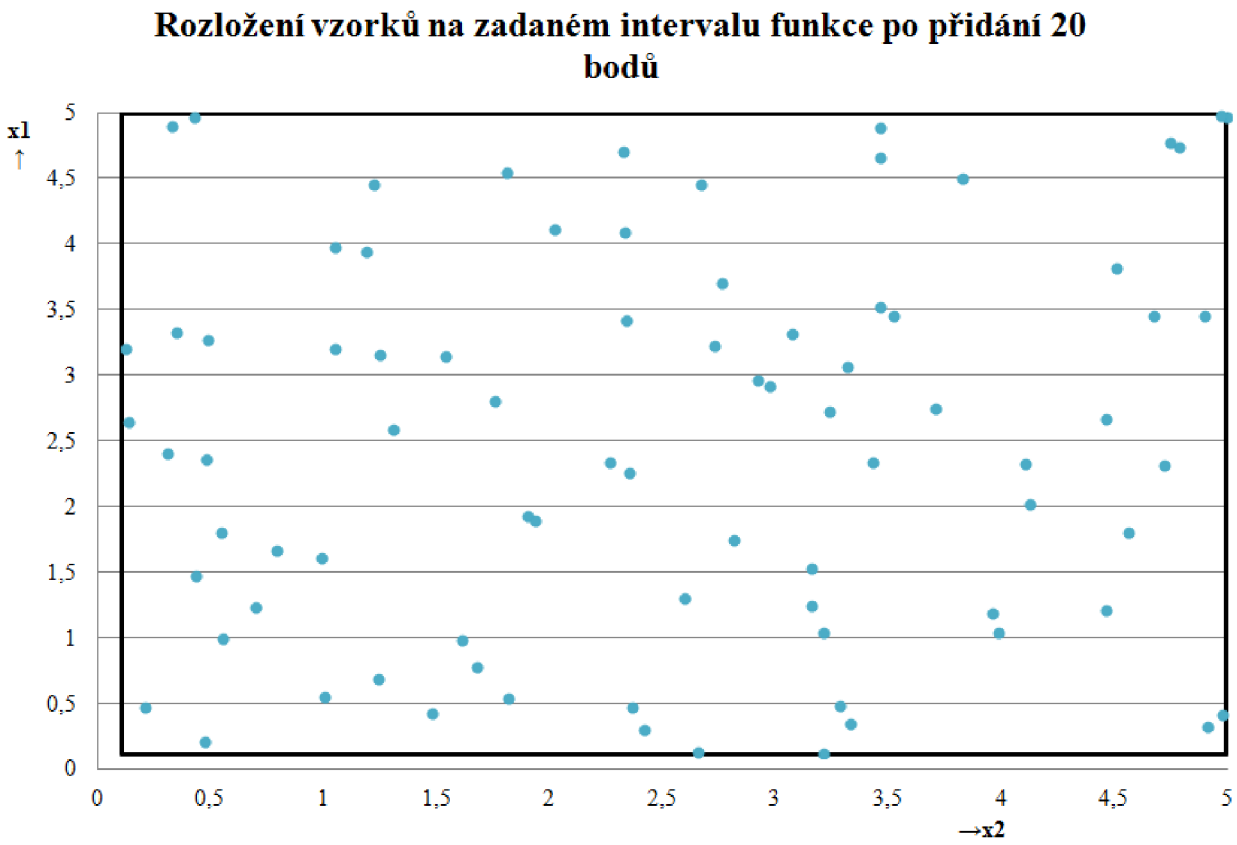


Obr. 29. Průběh chyby prediktoru po přidání 10 bodů

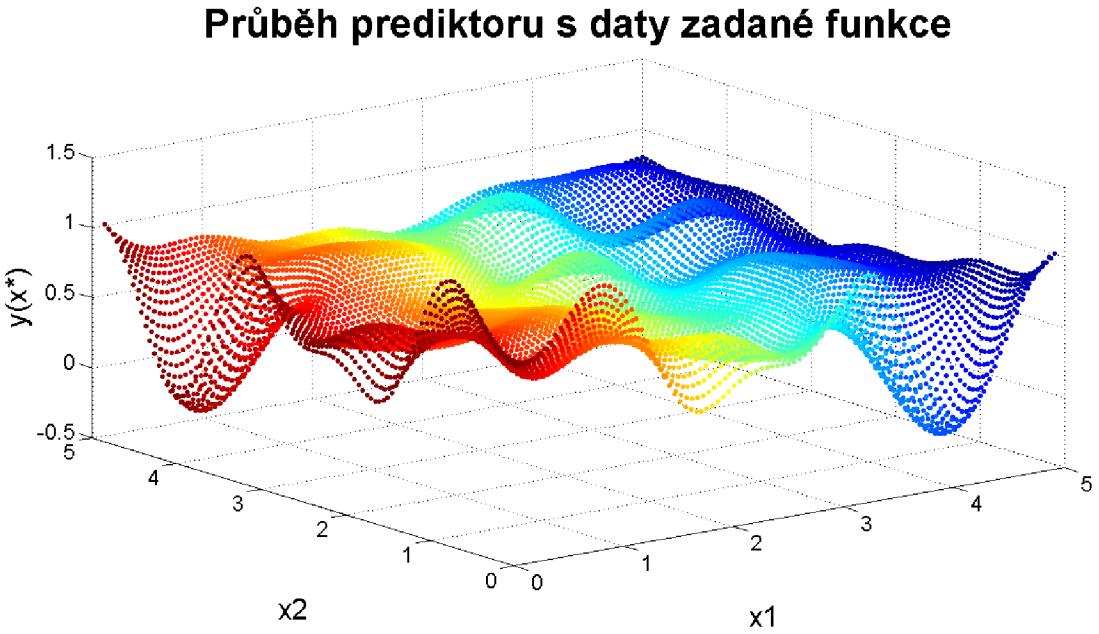


Obr. 30. Průběh očekávaného zlepšení po přidání 10 bodů

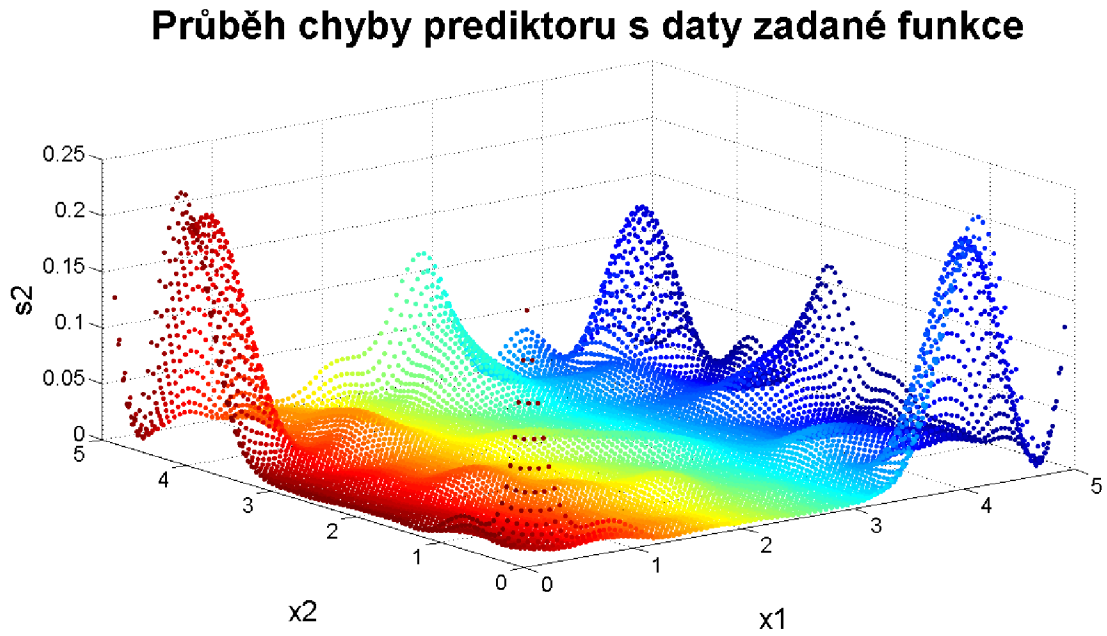
Na obrázku 27. bylo tedy přidáno vlastně 20 vzorků kvůli ose symetrie. Dále následují průběhy funkcí na obrázcích 28,29 a 30. Na dalším setu obrázků budou uvedeny průběhy po přidání 20 bodů.



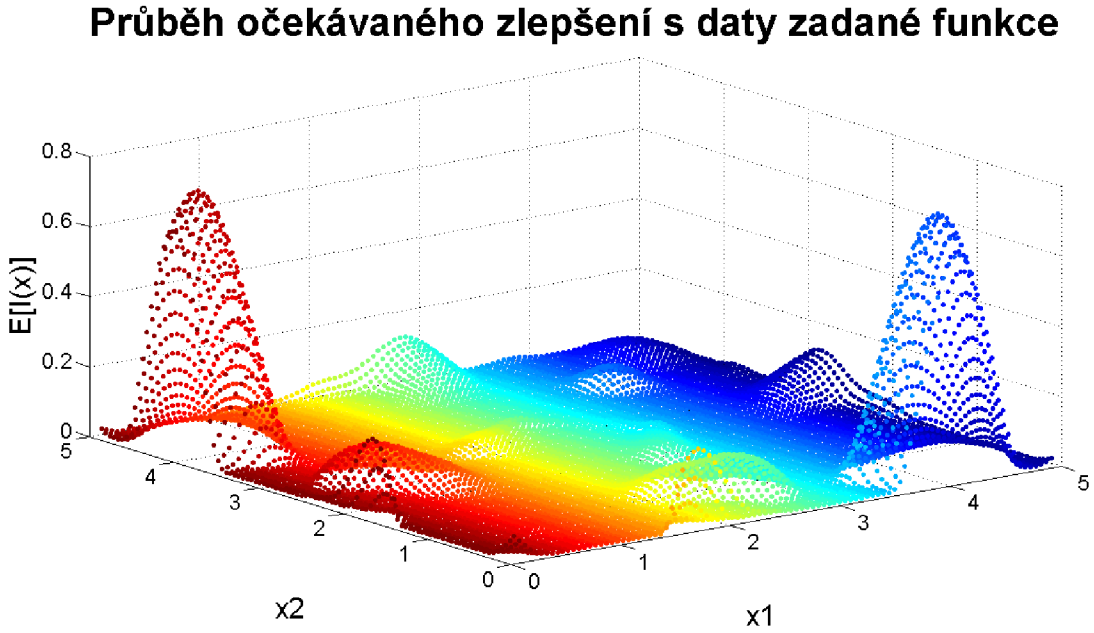
Obr. 31. Rozložení vzorků na intervalu po přidání 20 bodů



Obr. 32. Průběh prediktoru po přidání 20 bodů



Obr. 33. Průběh chyby prediktoru po přidání 20 bodů



Obr. 34. Průběh očekávaného zlepšení po přidání 20 bodů

Podle chyby a očekávaného zlepšení je patrné, že funkce se uvnitř intervalu mnohem lépe zmapovala. Větší nejistoty jsou už jen na okrajích zadaného intervalu. V další kapitole bude uvedena tabulka, která bude shrnovat konvergenci algoritmu a časovou náročnost algoritmu. Výpočty budou provedeny pro parametry θh=1 a ph=1. Tyto už však nebudou znázorněny v grafech ale výsledky pouze shrnuty tabulkou.

3.6 Shrnutí výsledků a posouzení kódu

Tabulka shrnující funkčnost algoritmu						
θh = 1, ph = 1, Počet ověření = 10						
		Nalezené minimum			Skutečné min. v	
		x_1	x_2	y	tomto bodě	
Čas[s]	114	3,8095	4,5011	-1,2227	0,7388	
Maximální chyba	473					
Maximální zlepšení	9,6					
θh = 1, ph = 1, Počet ověření = 20						
		Nalezené minimum				
		x_1	x_2	y		
Čas[s]	605	2,7222	3,2168	0,5728	0,5637	
Maximální chyba	44,21					
Maximální zlepšení	0,9548					
θh = 1, ph = 1, Počet ověření = 25						
		Nalezené minimum				
		x_1	x_2	y		
Čas[s]	1191	2,7228	3,2168	0,5722	0,5637	
Maximální chyba	103					
Maximální zlepšení	3,994					
θh = 1, ph = 2, Počet ověření = 10						
		Nalezené minimum				
		x_1	x_2	y		
Čas[s]	116	4,4023	2,2289	0,4834	0,6789	
Maximální chyba	0,148					
Maximální zlepšení	0,1					
θh = 1, ph = 2, Počet ověření = 20						
		Nalezené minimum				
		x_1	x_2	y		
Čas[s]	613	0,3024	4,105	-0,1835	0,7598	
Maximální chyba	0,237					
Maximální zlepšení	0,75					
θh = 1, ph = 2, Počet ověření = 25						
		Nalezené minimum				
		x_1	x_2	y		
Čas[s]	1205	2,1301	1,0928	0,36236	0,7029	
Maximální chyba	1492					
Maximální zlepšení	14,82					

Tab. 1. Tabulka shrnující časovou náročnost a konvergenci algoritmu

ZÁVĚR

Práce čtenáře seznámí s globální optimalizací funkcí. Při rešerši literatury bylo zjištěno, že optimalizace některých funkcí může být skutečný problém. Nejspíše proto na poli optimalizace probíhá stále intenzivní výzkum.

Metod, které bývají užity k optimalizaci, je opravdu ohromné množství. Existují metody víceméně univerzální nebo metody na konkrétní typ funkcí. Pak už záleží jen na subjektivním výběru programátora, kterou funkci zvolí na základě svých požadavků a vlastností konkrétní numerické metody. Pokud existuje deterministická metoda, která řeší daný typ problému, bývá lepší zvolit tuto metodu. Najít ale deterministickou metodu na některé typy funkcí bývá nemožné. Pak je nutné využít metod stochastických, které používají heuristiky k prohledávání definičního oboru funkce. Ze stochastických metod jsou nejlepší metody založené na evolučních algoritmech. Pak už jen záleží na tom najít algoritmus, který s naší funkcí nejlépe konverguje. Specifickou skupinou jsou funkce Black-Box, které mohou být na optimalizaci opravdu náročné, zvláště pokud je jejich průběh velmi nestandardní, jako v případě zadané funkce. Jelikož je zadaná funkce také „expensive'' je nutné myslet na počet ověření funkce.

Pro práci se zadanými daty byly naprogramovány funkce, které umožňují efektivní práci se souborem zadaných hodnot. Funkce byly programovány tak, aby byla možná jejich modifikace na problém s větším množstvím proměnných.

Algoritmus, který byl vybrán a implementován se nazývá algoritmus EGO(Efficient Global Optimization). Jeho principem je, že se z několika známých bodů mapuje průběh funkce pomocí tzv. prediktoru, což je vlastně odhad průběhu funkce. Pro každý odhad je pak také určena chyba tohoto odhadu. Když znám přibližný odhad průběhu funkce a jeho chybu, lze pak určit, na kterém místě je nejvýhodnější přidat další bod, abychom nalezli požadované minimum.

V práci bylo znázorněno, jak algoritmus pracuje na různých funkcích a znázorněny a vysvětleny jeho základní principy. Poté byl algoritmus aplikován na zadanou funkci, týkající se problematiky záření v plazmatu. To jak na této funkci funguje, znázorňují grafy uvedené v práci. Na základě rešerše literatury byl zvolen optimalizační algoritmus založený na metodě Branch and Bound. Pomocí tohoto algoritmu byly přidány body tam, kde by se mělo podle analýzy průběhu funkce nacházen její minimum. Výsledné určení minima je vidět v závěrečné tabulce.

Jelikož skutečné minimum funkce je následující minf(2.7607, 2.7705) = 0.28058, tak lze usoudit, že ani pomocí této sofistikované metody nebylo nalezeno. Nutno podotknout, že okolní body ve vzdálenosti jen 0,005 od tohoto bodu jsou svými funkčními hodnotami tomuto minimu značně vzdáleny. Nejspíše z tohoto důvodu má i tato metoda s nalezením minima problém. Nejblíže k nalezení tohoto minima byl algoritmus v případě kdy $\theta h=1$ a $ph=1$ a po přidání 20 a 25 bodů.

Ke zlepšení by mohlo dojít při jiném nastavení parametrů θh a ph a nebo pokud by optimalizační algoritmus na přidávání bodů byl nahrazen sofistikovanějším.

Tato práce byla velmi poučná i zábavná a velmi jsem si v ní rozšířil své znalosti na poli optimalizace, statistiky a neposlední řadě programování v jazyce C.

LITERATURA

[1] PRESS, William H, Saul A TEUKOLSKY, William T VETTELING a Brian P FLANNERY. Numerical Recipies: The art of scientific computing. 3rd ed. Cambridge: Cambridge University Press, 2007, xxi, 1235 s. ISBN 978-0-521-88068-8.

[2] Journal of Global Optimization, Springer, ISSN 0925-5001

[3] LIBERTI, Leo. *Introduction to Global Optimization*. LIX, Ecole Polytechnique, Palaiseau F-91128, France, 2008.

[4] TVRDÍK, Josef. *Evoluční algoritmy*. Ostrava, 2004. Učební text. Ostravská univerzita.

[5] CASSIOLI, Andrea. *A Tutorial on Black–Box Optimization* [online]. LIX-ECOLE POLYTECHNIQUE, 24/04/2013 [cit. 2014-12-08]. Dostupné z:http://www.lix.polytechnique.fr/~dambrosio/blackbox_material/Cassioli_1.pdf. Prezentace. École polytechnique.

[6] AUGER, Anne a Nikolaus HANSEN. *Introduction to Black-Box Optimization in Continuous Search Spaces: Definitions, Examples, Difficulties*[online]. [cit. 2015-05-25]. Dostupné z: https://www.lri.fr/~hansen/copenhagen-optimization-intro.pdf

[7] *Sallyx.org* [online]. 11.9.2005 [cit. 2014-12-11]. Dostupné z: http://www.sallyx.org/

[8]LISAL,Martin. *Bilineární interpolace* [online]. Ústí n L., 1998 [cit. 2014-12-11]. Dostupné z: http://physics.ujep.cz/~mlisal/nm_1/jskvor/PDF/BilinInterpol.pdf. Učební text. Univerzita J.E. Purkyně.

[9]JONES,Donald, Matthias SCHONLAU a William J.WELCH *Journal of Global Optimization - Springer: Efficient Global Optimization of Expensive Black-Box Functions* [online]. 1998 [cit. 2015-05-18]. ISSN 1573-2916. Dostupné z: http://link.springer.com/article/10.1023%2FA%3A1008306431147

PŘÍLOHY

PI Kód v jazyce C – hlavičkový soubor diplomka_optimalizace.h

PII Kód v jazyce C – soubor main.c

PIII CD-ROM: Kód v jazyce C – soubor diplomka_optimalizace.c, soubor main.c, hlavičkový soubor diplomka_optimalizace.h, datový soubor 2D.dat

PŘÍLOHA PI:

KÓD V JAZYCE C – HLAVIČKOVÝ SOUBOR DIPLOMKA_OPTIMALIZACE.H

int compare_ints(const void* a, const void* b);

int nacti (double **pole1, double **pole2, double **pole3,int *velikost);

double interpoluj(double x1, double x2, double *polex1, double *polex2, double *poley, int size);

double vzdalenost(double *boda, double *bodb, double theta, double ph, int size);

int maticeR(double *polebodu, double matrixR[100][100], int pocetbodu, int rad);

double LUdeterminant(double a[100][100], int rad);

double determinant(double a[100][100], int k);

void trans(double num[100][100], double fac[100][100], double invMatrix[100][100],int r);

void cofactors(double num[100][100], double invMatrix[100][100],int f);

void multiply (double matice1[100][100], double matice2[100][100], double maticevys[100][100], int pocr1, int pocs1, int pocr2, int pocs2);

void fjednotkovaMatice(double matrixR[100][100], int rad);

double mi(double *poleznamychbodu, double invMatrixR[100][100], int size, int rad);

double predictor(double *odhadovanybod, double *poleznamychbodu, double invMatrixR[100][100], int size, int rad);

double sigma(double *poleznamychbodu, double invMatrixR[100][100], int size, int rad);

double meansquarederror(double *odhadovanybod, double *poleznamychbodu, double invMatrixR[100][100], int size, int rad);

double expectedimprovement(double *odhadovanybod, double *poleznamychbodu, double invMatrixR[100][100], int size, int rad);

double expectedimprovementBB(double *poleznamychbodu, double invMatrixR[100][100], double yprediktor, double s2chyba, int size, int rad);

double polynom(double x);

void BBoptimalizace2D(double *vzorky,double *interval, int size, int rad);

PŘÍLOHA PII:

KÓD V JAZYCE C – SOUBOR MAIN.H

```
#include <stdio.h>
#include <windows.h>
#include <math.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include "diplomka_optimalizace.h"

#define NAZEV "C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\2D.dat"
#define NAZEVVYSTUPU
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\CosxplusxVystup1.txt"
#define NAZEVVYSTUPU2
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\CosxplusxVystup2.txt"
#define NAZEVVYSTUPU3
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\CosxplusxVystup3.txt"
#define NAZEVVYSTUPU4
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\CosxplusxVystup4.txt"
#define NAZEVVYSTUPU5
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\CosxplusxVystup5.txt"
#define NAZEVVYSTUPU6
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\CosxplusxVystup6.txt"
//Polynom
#define NAZEVVYSTUPU7
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\PolynomVystup1.txt"
#define NAZEVVYSTUPU8
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\PolynomVystup2.txt"
#define NAZEVVYSTUPU9
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\PolynomVystup3.txt"
#define NAZEVVYSTUPU10
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\PolynomVystup4.txt"
#define NAZEVVYSTUPU11
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\PolynomVystup5.txt"
#define NAZEVVYSTUPU12
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\PolynomVystup6.txt"
//zadana funkce
#define NAZEVVYSTUPU13
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\ZadanaFunkcex1.txt"
#define NAZEVVYSTUPU14
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\ZadanaFunkcex2.txt"
#define NAZEVVYSTUPU15
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\ZadanaFunkcey.txt"
#define NAZEVVYSTUPU16
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\ZadanaFunkceR.txt"
#define NAZEVVYSTUPU17
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\ZadanaFunkceR-1.txt"
```

```
#define NAZEVVYSTUPU18
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\ZadanaFunkcePrediktor.txt"
#define NAZEVVYSTUPU19
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\ZadanaFunkceChyba.txt"
#define NAZEVVYSTUPU20
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\ZadanaFunkceZlepseni.txt"
#define NAZEVVYSTUPU21 "C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\x1.txt"
#define NAZEVVYSTUPU22 "C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\x2.txt"
#define NAZEVVYSTUPU23
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\ZadanaFunkcex1BB.txt"
#define NAZEVVYSTUPU24
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\ZadanaFunkcex2BB.txt"
#define NAZEVVYSTUPU25
"C:\\Users\\Omajo\\Desktop\\VUT\\SP_code\\ZadanaFunkceyBB.txt"

#define iterace 1000
#define dtheta 1
#define dph 1
#define npoints 1000
#define step 0.02


int main(int argc, char **argv)
{
    FILE *vystup;
    FILE *vystup2;
    FILE *vystup3;
    srand((unsigned int) time(NULL));

    int size,i,j,pocetpridanychbodu=20;
    double *polea,*poleb,*polec,cas;
    //double truemin[3];
    double proMaticiR[300], detmatR, mojeMaticeR[100][100], mojeInvMaticeR[100][100],
    jednotkovaMatice[100][100], k, **detMatice, pozbod[3], odhady, x,
    interval[100],nalezeneminimum[3];
    __int64 frekvence, start, stop;

    if (!QueryPerformanceFrequency((LARGE_INTEGER*)&frekvence)) // Zjistíme
    frekvenci performance counteru
    {
    printf("Vas hardware nepodporuje Performance Counter\n");
    return 0; // Hardware nepodporuje Performance Counter proto skončíme bez měření
    }
    /*
    vystup = fopen(NAZEVVYSTUPU13, "w");                    //Nahodne vzorky
    zadane funkce x1

    for(i=0; i<21; i++){
            fprintf(vystup, "%20.15lf %s", 0.1049+((double)rand() /
            (double)RAND_MAX)*(4.9951-0.1049), "\n");
    }
```

```
fclose(vystup);

vystup = fopen(NAZEVVYSTUPU14, "w");                    //Nahodne vyorky
zadane funkce x2

for(i=0; i<21; i++){
        fprintf(vystup, "%20.15lf %s", 0.1049+((double)rand() /
        (double)RAND_MAX)*(4.9951-0.1049), "\n");
}
fclose(vystup);
*/
//double bod1[3] = {1,1,1};
//double bod2[3] = {2,2,2};
if(nacti(&polea,&poleb,&polec,&size) == 1){
        printf("Data nebyla nactena !!!\n Soubor se nepodarilo otevrit nebo chybi
        pamet.\n");
}
else{
        printf("Data se nacetla v poradku.\n");
}
k=size;//uchovani velikosti pole
/*
for(i=0; i<10; i++){                            //testovaci data cos(x)+x
        proMaticiR[i*2]=(double)(i*2);
        proMaticiR[(i*2)+1]=cos((double)(i*2))+((double)(i*2))/2;
}
vystup = fopen(NAZEVVYSTUPU, "w");

for(i=0; i<10; i++){                            //ulozeni dat cos(x)+x do souboru
        fprintf(vystup, "%20.15lf %s %20.15lf %s", proMaticiR[i*2], "\t",
        proMaticiR[(i*2)+1], "\n");
}
fclose(vystup);

detMatice = (double **) malloc((size_t)10 * (sizeof(double *)));        //dynamicka
alokace matice
if (detMatice == NULL)
   return 1;

for (i = 0; i < 10; i++) {
 detMatice[i] = (double *) malloc((size_t)20 * (sizeof(double)));
  if (detMatice[i] == NULL)
       return 1;
}

maticeR(&proMaticiR[0],mojeMaticeR,10,1);

vystup = fopen(NAZEVVYSTUPU2, "w");          //Matice R do souboru
for (i = 0; i < 10; i++) {
        fprintf(vystup, "%s", "\n");
        for (j = 0; j < 10; j++){
```

```
                fprintf(vystup, "%20.15lf %s", mojeMaticeR[i][j], "  ");
                detMatice[i][j] = mojeMaticeR[i][j];
        }
}
fclose(vystup);

//detmatR = determinant(mojeMaticeR,11);
//detmatR = Determinant(detMatice,10);
//printf("Determinant = %20.15lf\n", detmatR);
detmatR = LUdeterminant(mojeMaticeR,10);
printf("DeterminantLU = %20.15lf\n", detmatR);
cofactors(mojeMaticeR, mojeInvMaticeR ,10);

//multiply(mojeMaticeR,mojeInvMaticeR,jednotkovaMatice,6,6,6,6);

vystup = fopen(NAZEVVYSTUPU3, "w");          //inverzni matice R do souboru
for (i = 0; i < 10; i++) {
        fprintf(vystup, "%s", "\n");
        for (j = 0; j < 10; j++){
                fprintf(vystup, "%20.15lf %s", mojeInvMaticeR[i][j], "  ");
        }
}
fclose(vystup);

vystup = fopen(NAZEVVYSTUPU4, "w");
for (i = 0; i <= npoints; i++) {
        pozbod[0] = step*i;
        odhady = predictor(&pozbod[0],&proMaticiR[0],mojeInvMaticeR,10,1);
        fprintf(vystup, "%20.15lf %s", odhady, "\n");
}
fclose(vystup);

vystup = fopen(NAZEVVYSTUPU5, "w");
for (i = 0; i <= npoints; i++) {
        pozbod[0] = step*i;
odhady =meansquarederror(&pozbod[0],&proMaticiR[0],mojeInvMaticeR,10,1);
fprintf(vystup, "%20.15lf %s", odhady, "\n");
}
fclose(vystup);

vystup = fopen(NAZEVVYSTUPU6, "w");
for (i = 0; i <= npoints; i++) {
        pozbod[0] = step*i;
odhady = expectedimprovement(&pozbod[0],&proMaticiR[0],mojeInvMaticeR,10,1);
fprintf(vystup, "%20.15lf %s", odhady, "\n");
}
fclose(vystup);

//polynom
//testovaci data polynom f(x) = a + b*x + c*x^2 + d*x^3 + ... + p*x^15
```

```
                /*proMaticiR[0]= 1;
                proMaticiR[1]=polynom(1);
                proMaticiR[2]= 2;
                proMaticiR[3]=polynom(2);
                proMaticiR[4]= 3;
                proMaticiR[5]=polynom(3);
                proMaticiR[6]= 4;
                proMaticiR[7]=polynom(4);
                proMaticiR[8]= 12;
                proMaticiR[9]=polynom(12);
                proMaticiR[10]= 2.56;
                proMaticiR[11]=polynom(2.56);
                proMaticiR[12]= 7.72;
                proMaticiR[13]=polynom(7.72);
                proMaticiR[14]= 8.22;
                proMaticiR[15]=polynom(8.22);

                proMaticiR[0]= 1;
                proMaticiR[1]=polynom(1);
                proMaticiR[2]= 2;
                proMaticiR[3]=polynom(2);
                proMaticiR[4]= 2.56;
                proMaticiR[5]=polynom(2.56);
                proMaticiR[6]= 3;
                proMaticiR[7]=polynom(3);
                proMaticiR[8]= 4;
                proMaticiR[9]=polynom(4);
                proMaticiR[10]= 7.72;
                proMaticiR[11]=polynom(7.72);
                proMaticiR[12]= 8.22;
                proMaticiR[13]=polynom(8.22);
                proMaticiR[14]= 12;
                proMaticiR[15]=polynom(12);

        vystup = fopen(NAZEVVYSTUPU7, "w");

        for(i=0; i<8; i++){                             //ulozeni dat polynom f(x) = a + b*x + c*x^2
+ d*x^3 + ... + p*x^15
        fprintf(vystup, "%20.15lf %s %20.15lf %s", proMaticiR[i*2], "\t", proMaticiR[(i*2)+1],
"\n");
        }
        fclose(vystup);

        maticeR(&proMaticiR[0],mojeMaticeR,8,1);

        vystup = fopen(NAZEVVYSTUPU8, "w");          //Matice R do souboru
        for (i = 0; i < 8; i++) {
                fprintf(vystup, "%s", "\n");
                for (j = 0; j < 8; j++){
                        fprintf(vystup, "%20.15lf %s", mojeMaticeR[i][j], "  ");
                        //detMatice[i][j] = mojeMaticeR[i][j];
```

```
            }
        }
        fclose(vystup);

        detmatR = LUdeterminant(mojeMaticeR,8);
        printf("DeterminantLU = %20.15lf\n", detmatR);
        cofactors(mojeMaticeR,mojeInvMaticeR,8);

        //multiply(mojeMaticeR,mojeInvMaticeR,jednotkovaMatice,8,8,8,8);

        vystup = fopen(NAZEVVYSTUPU9, "w");          //inverzni matice R do souboru
        for (i = 0; i < 8; i++) {
                fprintf(vystup, "%s", "\n");
                for (j = 0; j < 8; j++){
                        fprintf(vystup, "%20.15lf %s", mojeInvMaticeR[i][j], "  ");
                }
        }
        fclose(vystup);

        vystup = fopen(NAZEVVYSTUPU10, "w");
        for (i = 0; i <= 600; i++) {
                pozbod[0] = step*i;
                odhady = predictor(&pozbod[0],&proMaticiR[0],mojeInvMaticeR,8,1);
                fprintf(vystup, "%20.15lf %s", odhady, "\n");
        }
        fclose(vystup);

        vystup = fopen(NAZEVVYSTUPU11, "w");
        for (i = 0; i <= 600; i++) {
                pozbod[0] = step*i;
                odhady = meansquarederror(&pozbod[0],&proMaticiR[0],mojeInvMaticeR,8,1);

                fprintf(vystup, "%20.15lf %s", odhady, "\n");
        }
        fclose(vystup);

        vystup = fopen(NAZEVVYSTUPU12, "w");
        for (i = 0; i <= 600; i++) {
                pozbod[0] = step*i;
        odhady =expectedimprovement(&pozbod[0],&proMaticiR[0],mojeInvMaticeR,8,1);
        fprintf(vystup, "%20.15lf %s", odhady, "\n");
        }
        fclose(vystup);
        */
        //Funkce zadaná k optimalizaci
        if ((vystup = fopen(NAZEVVYSTUPU13, "r")) == NULL) {
          fprintf(stderr, "Chyba otevreni souboru %s pro cteni\n", NAZEVVYSTUPU13);
          return 1;
        }
        if ((vystup2 = fopen(NAZEVVYSTUPU14, "r")) == NULL) {
          fprintf(stderr, "Chyba otevreni souboru %s pro cteni\n", NAZEVVYSTUPU14);
```

```
   return 1;
}

i=0;
while(fscanf(vystup, "%lf", &x) != EOF){
i++;
}
rewind (vystup);
size=i;

for (i = 0; i < size; i++) {
   fscanf(vystup, "%lf", &proMaticiR[i*3]);
         fscanf(vystup2, "%lf", &proMaticiR[(i*3)+1]);
                  proMaticiR[(i*3)+2] =
interpoluj(proMaticiR[i*3],proMaticiR[(i*3)+1],polea,poleb,polec,k);
}
fclose(vystup);
fclose(vystup2);

vystup = fopen(NAZEVVYSTUPU15, "w");
for (i = 0; i < size; i++) {
fprintf(vystup, "%20.15lf %s",
interpoluj(proMaticiR[i*3],proMaticiR[(i*3)+1],polea,poleb,polec,k), "\n");
}
fclose(vystup);

maticeR(&proMaticiR[0],mojeMaticeR,size,2);

vystup = fopen(NAZEVVYSTUPU16, "w");                    //Matice R do souboru
for (i = 0; i < size; i++) {
         fprintf(vystup, "%s", "\n");
         for (j = 0; j < size; j++){
                  fprintf(vystup, "%20.15lf %s", mojeMaticeR[i][j], "  ");
         }
}
fclose(vystup);

detmatR = LUdeterminant(mojeMaticeR,size);
printf("DeterminantLU = %20.15lf\n",detmatR);
cofactors(mojeMaticeR,mojeInvMaticeR,size);

//multiply(mojeMaticeR,mojeInvMaticeR,jednotkovaMatice,size,size,size,size);

vystup = fopen(NAZEVVYSTUPU17, "w");                    //inverzni matice R do souboru
for (i = 0; i < size; i++) {
         fprintf(vystup, "%s", "\n");
         for (j = 0; j < size; j++){
                  fprintf(vystup, "%20.15lf %s", mojeInvMaticeR[i][j], "  ");
         }
}
fclose(vystup);
```

```
vystup = fopen(NAZEVVYSTUPU18, "w");
for (i = 0; i < 100; i++) {
        for (j = 0; j < 100; j++) {
                pozbod[0] = 0.1049+(((double)i / 99)*(4.9951-0.1049));
                pozbod[1] = 0.1049+(((double)j / 99)*(4.9951-0.1049));
                odhady = predictor(&pozbod[0],&proMaticiR[0],mojeInvMaticeR,size,2);

                fprintf(vystup, "%20.15lf %s", odhady, "\n");
}
}
fclose(vystup);

vystup = fopen(NAZEVVYSTUPU19, "w");
for (i = 0; i < 100; i++) {
        for (j = 0; j < 100; j++) {
pozbod[0] = 0.1049+(((double)i / 99)*(4.9951-0.1049));
pozbod[1] = 0.1049+(((double)j / 99)*(4.9951-0.1049));
odhady =meansquarederror(&pozbod[0],&proMaticiR[0],mojeInvMaticeR,size,2);
fprintf(vystup, "%20.15lf %s", odhady, "\n");
}
}
fclose(vystup);

vystup2 = fopen(NAZEVVYSTUPU21, "w");
vystup3 = fopen(NAZEVVYSTUPU22, "w");
vystup = fopen(NAZEVVYSTUPU20, "w");
for (i = 0; i < 100; i++) {
        for (j = 0; j < 100; j++) {
pozbod[0] = 0.1049+(((double)i / 99)*(4.9951-0.1049));
pozbod[1] = 0.1049+(((double)j / 99)*(4.9951-0.1049));
odhady = expectedimprovement(&pozbod[0],&proMaticiR[0],mojeInvMaticeR,size,2);
fprintf(vystup, "%20.15lf %s", odhady, "\n");
fprintf(vystup2, "%20.15lf %s", pozbod[0], "\n");
fprintf(vystup3, "%20.15lf %s", pozbod[1], "\n");
}
}
fclose(vystup);
fclose(vystup2);
fclose(vystup3);

//optimalizace zadane funkce
interval[0]=0.1049;
interval[1]=4.9951;
interval[2]=0.1049;
interval[3]=4.9951;

QueryPerformanceCounter((LARGE_INTEGER*)&start);//zacatek mereni casu

for (j = 0; j < pocetpridanychbodu; j++){
        for (i = 0; i < 5; i++){
```

```
                    BBoptimalizace2D(&proMaticiR[0],&interval[0],size,2);
                }
                proMaticiR[size*3]=interval[0]+((double)rand() /
(double)RAND_MAX)*(interval[1]-interval[0]);
                proMaticiR[size*3+1]=interval[2]+((double)rand() /
(double)RAND_MAX)*(interval[3]-interval[2]);

        proMaticiR[size*3+2]=interpoluj(proMaticiR[size*3],proMaticiR[(size*3)+1],polea,pole
b,polec,k);
                proMaticiR[size*3+3]=proMaticiR[size*3+1];
                proMaticiR[size*3+4]=proMaticiR[size*3];
                proMaticiR[size*3+5]=proMaticiR[size*3+2];
                size=size+2;
                interval[0]=0.1049;
                interval[1]=4.9951;
                interval[2]=0.1049;
                interval[3]=4.9951;
        }

        QueryPerformanceCounter((LARGE_INTEGER*)&stop);        //konec mereni casu
        cas = ((double)(stop-start))/(double)frekvence;        //vypocet casu v sekundach
        printf("Namereny cas: %f s\n",cas);

        vystup = fopen(NAZEVVYSTUPU23, "w");                   //x1 po pridani bodu

        for(i=0; i<size; i++){
                fprintf(vystup, "%20.15lf %s", proMaticiR[i*3], "\n");
        }
        fclose(vystup);

        vystup = fopen(NAZEVVYSTUPU24, "w");                   //x2 po pridani bodu

        for(i=0; i<size; i++){
                fprintf(vystup, "%20.15lf %s", proMaticiR[i*3+1], "\n");
        }
        fclose(vystup);
        vystup = fopen(NAZEVVYSTUPU25, "w");                   //y po pridani bodu

        for(i=0; i<size; i++){
                fprintf(vystup, "%20.15lf %s", proMaticiR[i*3+2], "\n");
        }
        fclose(vystup);

        maticeR(&proMaticiR[0],mojeMaticeR,size,2);
        detmatR = LUdeterminant(mojeMaticeR,size);
        printf("DeterminantLU = %20.15lf\n",detmatR);
        cofactors(mojeMaticeR,mojeInvMaticeR,size);

        vystup = fopen(NAZEVVYSTUPU18, "w");
        for (i = 0; i < 100; i++) {
                for (j = 0; j < 100; j++) {
```

```
                        pozbod[0] = 0.1049+(((double)i / 99)*(4.9951-0.1049));
                        pozbod[1] = 0.1049+(((double)j / 99)*(4.9951-0.1049));
                        odhady = predictor(&pozbod[0],&proMaticiR[0],mojeInvMaticeR,size,2);

                        fprintf(vystup, "%20.15lf %s", odhady, "\n");
                        if(i==0 && j==0){
                                nalezeneminimum[0]=pozbod[0];
                                nalezeneminimum[1]=pozbod[1];
                                nalezeneminimum[2]=odhady;
                        }
                        if(nalezeneminimum[2]>odhady){
                                nalezeneminimum[0]=pozbod[0];
                                nalezeneminimum[1]=pozbod[1];
                                nalezeneminimum[2]=odhady;
                        }
        }
        }
        printf("Nalezene minimum (x1, x2, y)(%20.15lf, %20.15lf,
%20.15lf)\n",nalezeneminimum[0],nalezeneminimum[1],nalezeneminimum[2]);
        fclose(vystup);

        vystup = fopen(NAZEVVYSTUPU19, "w");
        for (i = 0; i < 100; i++) {
                for (j = 0; j < 100; j++) {
        pozbod[0] = 0.1049+(((double)i / 99)*(4.9951-0.1049));
        pozbod[1] = 0.1049+(((double)j / 99)*(4.9951-0.1049));
        odhady = meansquarederror(&pozbod[0],&proMaticiR[0],mojeInvMaticeR,size,2);
        fprintf(vystup, "%20.15lf %s", odhady, "\n");
        }
        }
        fclose(vystup);

        vystup = fopen(NAZEVVYSTUPU20, "w");
        for (i = 0; i < 100; i++) {
                for (j = 0; j < 100; j++) {
        pozbod[0] = 0.1049+(((double)i / 99)*(4.9951-0.1049));
        pozbod[1] = 0.1049+(((double)j / 99)*(4.9951-0.1049));
        odhady = expectedimprovement(&pozbod[0],&proMaticiR[0],mojeInvMaticeR,size,2);
        fprintf(vystup, "%20.15lf %s", odhady, "\n");
        }
        }
        fclose(vystup);

        //vzdalenost(&bod1[0], &bod2[0], 1, 2, 3);
        //interpoluj(3 , 2.75 ,polea ,poleb ,polec ,size);
        //printf("%lf\n", *polec);

        //qsort(polec, size, sizeof(double), compare_ints);

        //printf("Skutecne minimum min=[x, x, %lf]\n", *(polec+size-1));
        /*for(i=0; i<=2; i++){
```

```
    truemin[2]=nahodnehledani(polea ,poleb ,polec , &truemin[0], size);
    printf("Minimum nalezene nahodnym prohledavanim min=[%lf, %lf, %lf]\n",
    truemin[0], truemin[1], truemin[2]);
    truemin[2]=evolucnistrategie(polea ,poleb ,polec , &truemin[0], size);
    printf("Minimum nalezene evolucni strategii min=[%lf, %lf, %lf]\n", truemin[0],
    truemin[1], truemin[2]);
    }
    for (i = 0; i < 20; i++) {
      free(mojeMaticeR[i]);
    }
    free(mojeMaticeR);*/
    //free(polea);
    //free(poleb);
    //free(polec);

    getch();
    return 0;
}
```