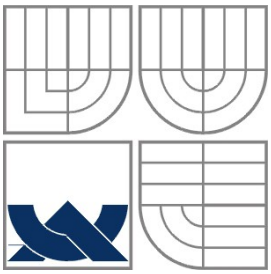


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

# GENETICKÉ PROGRAMOVÁNÍ IMPLEMENTACE JAVA

GENETIC PROGRAMMING - JAVA IMPLEMENTATION

DIPLOMOVÁ PRÁCE
DIPLOMA THESIS

**AUTOR PRÁCE** **BC. MAREK TOMAŠTÍK**
AUTHOR

**VEDOUCÍ PRÁCE** **DOC. ING. RADOMIL MATOUŠEK, PH.D.**
SUPERVISOR

BRNO 2013

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2012/2013

# ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Bc. Marek Tomaštík

který/která studuje v **magisterském navazujícím studijním programu**

obor: **Aplikovaná informatika a řízení (3902T001)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

**Genetické programování - Java implementace**

v anglickém jazyce:

**Genetic programming - Java implementation**

Stručná charakteristika problematiky úkolu:

Cílem práce bude návrh programové aplikace, která bude využitelná v úlohách automatického generování modelů, resp. úlohách tzv. symbolické regrese. Pro zvolené datové množiny nebo časové řady nelineárního charakteru bude generován analytický matematický model. Předpokládá se využití tzv. testovacích úloh a hledání optimální parametrizace řešiče, který bude založen na principech genetického programování (GP).

Cíle diplomové práce:

1. Implementace genetického programování (GP) v jazyce Java.
2.Implementace pokročilých operátorů GP (nedestruktivní operace, elitní přístup, zjednodušování výrazů).
3. Otestování navrženého řešiče na zvolených testovacích úlohách a porovnání s exaktním přístupem.
4. Popis GP, popis implementace a diskuze k dosaženým výsledkům.

Seznam odborné literatury:

Koza, J.R. (1990). Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems, Stanford University Computer Science Department technical report STAN-CS-90-1314. A thorough report, possibly used as a draft to his 1992 book.

Koza, J.R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press. ISBN 0-262-11170-5

Koza, J.R. (1994). Genetic Programming II: Automatic Discovery of Reusable Programs, MIT Press. ISBN 0-262-11189-6

Vedoucí diplomové práce: doc. Ing. Radomil Matoušek, Ph.D.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2012/2013.

V Brně, dne 20.11.2012

L.S.

| | |
|---|---|
| Ing. Jan Roupec, Ph.D.<br>Ředitel ústavu | prof. RNDr. Miroslav Doupovec, CSc., dr. h. c.<br>Děkan fakulty |

## ABSTRAKT

Tato diplomová práce má za cíl vytvoření programové aplikace v jazyce Java, dále využitelné v oblasti automatického generování modelů, speciálně v úlohách tzv. symbolické regrese. Práce zahrnuje stručný popis genetického programování (GP) a vlastní implementací GP s důrazem na využití pokročilých operátorů (nedestruktivní operace, elitní přistup, zjednodušování výrazů).

Pro zvolené datové množiny je technikou symbolické regrese generován matematický model. K ověření funkcionality je využito tzv. testovacích úloh. Pro vybranné parametry GP je hledáno optimální nastavení.



## ABSTRACT

This Master´s thesis implements computer program in Java, useful for automatic model generating, specially in symbolic regression problem. Thesis includes short description of genetic programming (GP) and own implementation with advanced GP operands (non-destructive operations, elitism, exptression reduction).

Mathematical model is generating by symbolic regression, exacly for choosen data set. For functioning check are used test tasks. Optimal settings is found for choosen GP parameters.



## KLÍČOVÁ SLOVA

EVT, genetické programování, křížení, mutace, selekce, Java, symoblická regrese, automatická tvorba modelů

## KEYWORDS

evolutionary computation techniques, genetic programming, mutation, crossover, selection, Java, symoblic regression, automatic model creation

# PROHLÁŠENÍ O ORIGINALITĚ

Prohlašuji, že předložená diplomová práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

Hustopeče, 23. května 2013

……...
Bc. Marek Tomaštík

# BIBLIOGRAFICKÁ CITACE

TOMAŠTÍK, M. *Genetické programování - Java implementace.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2013. 59 s. Vedoucí diplomové práce doc. Ing. Radomil Matoušek, Ph.D.

# PODĚKOVÁNÍ

Rád bych poděkoval doc. Ing. Radomilu Matouškovi, PhD. za užitečné rady, zajímavé podměty a věnovaný čas při tvorbě této práce. Rovněž velmi děkuji svým rodičům a dalším rodinným příslušníkům za všestrannou podporu během studia.

# Obsah:

# 1 ÚVOD

V živé přírode závisí přežití druhů, resp. jedinců daného druhu, na schopnosti adaptovat se na dané prostředí. Chování některých typů organismů nebo pochopení obecných biologických principů se stalo inspirací počítačovým algoritmům popř. metodám strojového učení. Podoblastí takto motivovaných metod jsou například algoritmy, hledající inspiraci v biologické evoluci. V tomto kontextu můžeme hovořit o Darwinově teorii popisující přírodní výběr nebo o zákonech dědičnosti popsaných Mendelem – takto inspirovane algoritmy se obecně označují jako evoluční výpočetní techniky (EVT). Tato práce mimojiné popisuje EVT s důrazem na konkrétní metodu označenou jako genetické programování (GP). Součástí práce je vlastní implementace GP a realizace ověřujících experimentů z oblasti tzv. symbolické regrese.

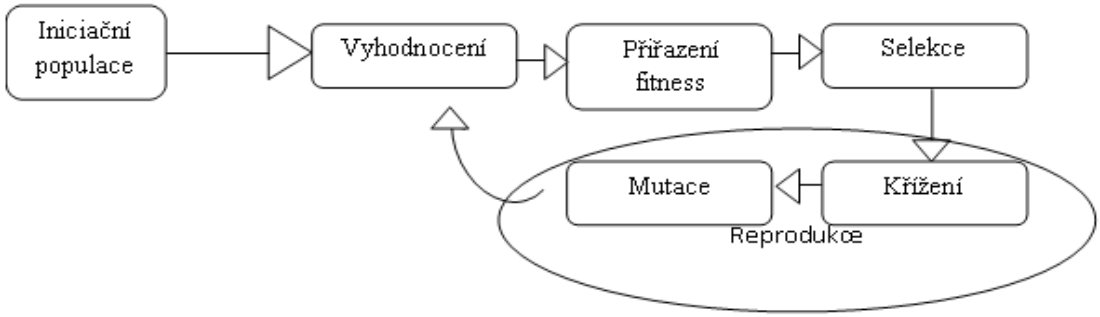
Práce je členěna do sedmi kapitol včetně úvodu a závěru. Druhá kapitola je koncipována jako „vsuvka“ užitečných numerických metod, týkajících se aproximace funkce. Kapitola třetí se podrobněji věnuje genetickému programování, strukturám reprezentujícím jedince a obecným typům metod ohodonocení (fitness). Obsahuje popis operací GP, konkrétně typu selekčního mechanismus, křížení, mutace, permutace, editace, zapozdření a dalších. Operátory obsažené v implementační aplikace – zejména nedestruktivní křížení a mutace, eltní přístup, editace (redukce) – obsahují popis způsobu implementace v rámci algoritmu. Dále uvádí popis principu automatického definování funkcí (ADF) a vybrané příklady použití GP. Následující kapitola je věnována problematice symbolické regrese (SR), včetně symbolicé regrese s tvorbou konstant. Obsahuje specifikace a principy funkcí ohodnocení (fitness), uživáných v rámci implementace. Jedná se o sumu absolutních odchylek (*„Sum Absolute Error“*), sumu kvadratických odchylek („Sum Square Error“) a „dynamické toleranční tuby (*„Sum Epsilon Tube Error“*). Implementační kapitola popisuje v první řadě grafické uživatelké prostředí a nastiňuje ovládání aplikace (kompletní instrukce k obsluze jsou součástí přílohy této práce). Následně podrobněji popisuje hlavní třídy aplikace, jejich funkci, dostupné metody a zpracovávané atributy. Mezi popisované třídy byl zahrnut řešič GP (`solver.java`), třída reprezentující jedince – binární strom (`TreeIFace.java`), třída hlavního vlákna aplikace (`SupremeThread.java`) a třída výpočtového vlákna (`WorkThread.java`). Kapitola šestá obsahuje soubor testovacích funkcí, testovaní vytvořené aplikace. Cílem první fáze testování je pomocí kombinací testů získat vhodné parametry – například. optimální množství jedinců mutovaných v každé generaci, počet jedinců populace atd. Získané informace po vyhodnocení určí možné optimální nastavení řešiče GP na uvažovaný problém, které je využito v druhé fázi měření tj. testování uvažovaných funkcí různými metodami ohodnocení (fitness). Závěr této práce obsahuje porovnání dosažených výsledků a návrh možných zlepšení celkové parametrizace řešiče.

# 2 POZNÁMKY K EVOLUČNÍM VÝPOČETNÍM TECHNIKÁM

## 2.1 Evoluční výpočetní techniky

Evoluční výpočetní techniky (EVT) [2], rovněž evoluční algoritmy (EA), jsou meta-heuristické techniky využívající populaci jedinců, přičemž jedinci jsou v podstatě potenciální řešení daného problému. EVT si berou za vzor biologické principy. V každé *iteraci*, zde označované jako *generace*, využívají původem přírodní mechanismy typu mutace, křížení, přírodní výběr, popř. další, spíše uměle zavedené operátory, např. operace garantovaného přežití nejschopnějších (elitních) jedinců. Po implementaci uvedených operátorů EVT je dosaženo nového řešení tj. nová generace, resp. populace jedinců. Mezi základní EVT patří například genetické algoritmy, hejnové algoritmy a genetické programování, které se dále uvedené problematiky týká a je tedy uvedeno podrobněji v kapitole 3. Poznamenejme, že EVT je obsahově velmi široké téma jak dokladuje například publikační přehled dle [7] .

### 2.1.1 Životní cyklus evolučních algoritmů (EA)




*Obr. 1 Základní životní cyklus EA*

Problematika evolučních algoritmů [2] nespočívá v jednom nebo dvou typech algoritmů. Průběh výpočtu neboli životní cyklus je však v zásadě totožný, jak je znázorněno na obr. 1. V rámci námi zkoumané problematiky – nezbytných parametrů a požadovaných specifikací – se vytvoří náhodně , resp. dle daných pravidel, počáteční populace z daného genofondu – přesná realizaci a typ genofondu přísluší danému typu evolučního algoritmu.

Takto získaná populace je zpracována algoritmem, každý jedinec je vyhodnocen a je mu přiřazena fitness funkce (obecně ohodnocení), závislá na dané problematice a míře vhodnosti jedince. Ohodnocená populace je podrobena selekci (prakticky je typů selekce mnoho) jedinců vhodných k reprodukci.

Termín reprodukce [2] je lehce zavádějící, k přímé reprodukci jedince může dojít v závislosti na parametrech. Nejběžnější průběh EA spočívá v pozměňování a kombinaci genotypu jedinců vybraných k reprodukci. Nově vzniklý jedinci se integrují do populace, která je podrobena kontrolní podmínce ukončení, v případě nesplnění – řešení se nejeví optimálním – algoritmus pokračuje hodnocením jedinců a přiřazením hodnoty fitness.

### 2.1.2 Genetické algoritmy

Genetické algoritmy (GA) patří do rodiny evolučních algoritmů **[2]**. Jak název napovídá inspirují se především v oblasti populační genetiky. Části prohledávaného prostoru reprezentují GA nejčastěji jako binární řetězce, které reprezentují genotyp a jednotlivé bity tak představují biologické geny. Reprezentace genotypu se nazývá chromozom, jehož délka je zpravidla fixní.

Cyklus výpočtu GA se shoduje s obecně platným schématem evoluční algoritmů, jak je uveden výše na obr. 1. V rámci reprodukce uvažujeme operátory typu křížení a mutace. Pokud je využita reprezentace jedince pomocí bitového řetězce je zpravidla mutace prováděna jako negace náhodně vybraného genu, křížením probíhá záměna polovin genotypů účastnících se jedinců.

### 2.1.3 Inteligence hejna (swarm intelligence)

Koncept inteligence hejna (SI) je dobře známý z přírodních společenství. Umělá inteligence hejna [3] je založena jako decentralizovaný, samoorganizující založený na chování jednotlivců, jejichž spoluprací vzniká právě chování (inteligence) hejna. Algoritmy SI náleží do evolučních algoritmů, lze je přiřadit také jako nástupce celulárních automatů, kde jednoduché lokální pravidlo každé buňky vytváří ve sledovaném prostoru chování globální.

Z principu sestavený algoritmus SI pracuje s populací jednoduchých agentů, kteří jsou ovlivňování prostředím a sebou navzájem. V úvahu připadají připadají právě ty příklady z reálné přírody, kde společenství postrádají centrální řízení a každý jedinec reaguje na nastalé změny prostředí ,případně chování jiného jedince podle „jednoduchého“ vzorce, čímž vzniká globální inteligence (chování). Obecně se pro tvorbu algoritmu uvažují ptačí, včelí a rybí hejna, kolonie mravenců nebo bakterií a některé další. Praktické využití lze nalézt například v robotice – krátkodobá předpověď problémů při hledání cesty robota popř. optimalizace samotné cesty [3].

Podíváme-li se na některé typy algoritmů blíže, budou se v implementaci algoritmu lišit v závislosti na použitém biologickém vzoru. Hledisko optimalizace funkce velmi věrně naplňuje mravenčí kolonie (ant colony optimalization [4]). Reálný přírodní systém je založen na náhodném rozdělení cest prvních mravenců od mraveniště. Putující zanechávající feromonovou stopu, na kterou reagují další jedinci. Síla stopy závisí na čase – s přibývajícím časem se rozplývá – a počtu jedinců, kteří stopu sledují a tím i obnovují resp. posilují. Logicky silnější feromonová stopa mezi mraveništěm a potenciálně objeveným zdrojem potravy, je užívána více jedinci a sílí, zatímco jiné slábnou. V implementaci je časová nestálost stopy užívána pro odbourání rizika uvíznutí v lokálním maximu.

Feromonová stopa je přímo určena vztahem:

$$\tau_{i,j} = \tau_{i,j} * \rho \quad , \tag{1}$$

kde $\rho$ označuje odpařování stopy.

Reprezentace vzniklého řešení se v případě SI liší od způsobu obecné reprezentace platné pro evoluční algoritmy. Jedinec hledá vhodnou cestu „za potravou“ dle svého uvážení ovlivňován prostředím (uvažujeme tak, že cesta ve smyslu „půdy“ předává informace od ostatních jedinců jako nositel feromonů, což samozřejmě neplatí u jiných typů hejn a zároveň obsahuje překážky). V každé iteraci je tedy „produktem“ jedince pozitivní zpětná vazba, kterou lze vyjádřit vztahem:

$$\tau_{i,j} = \tau_{i,j} + \frac{K}{f(x)} \quad , \tag{2}$$

kde: $K$ je kladná konstanta, která se dělí hodnotou účelové funkce v řešením nalezeném mravencem.

Implementace algoritmu je ve své podstatě podobná reálné předloze. Základní cyklus algoritmu pak koresponduje se schématem uvedeným výše na obr.1 s ohledem na optimalizační funkci je jistá pozornost věnována výběru cesty (hrany). Pro náš problém ovšem není rozhodující a proto se jí blíže zabývat nebudeme.

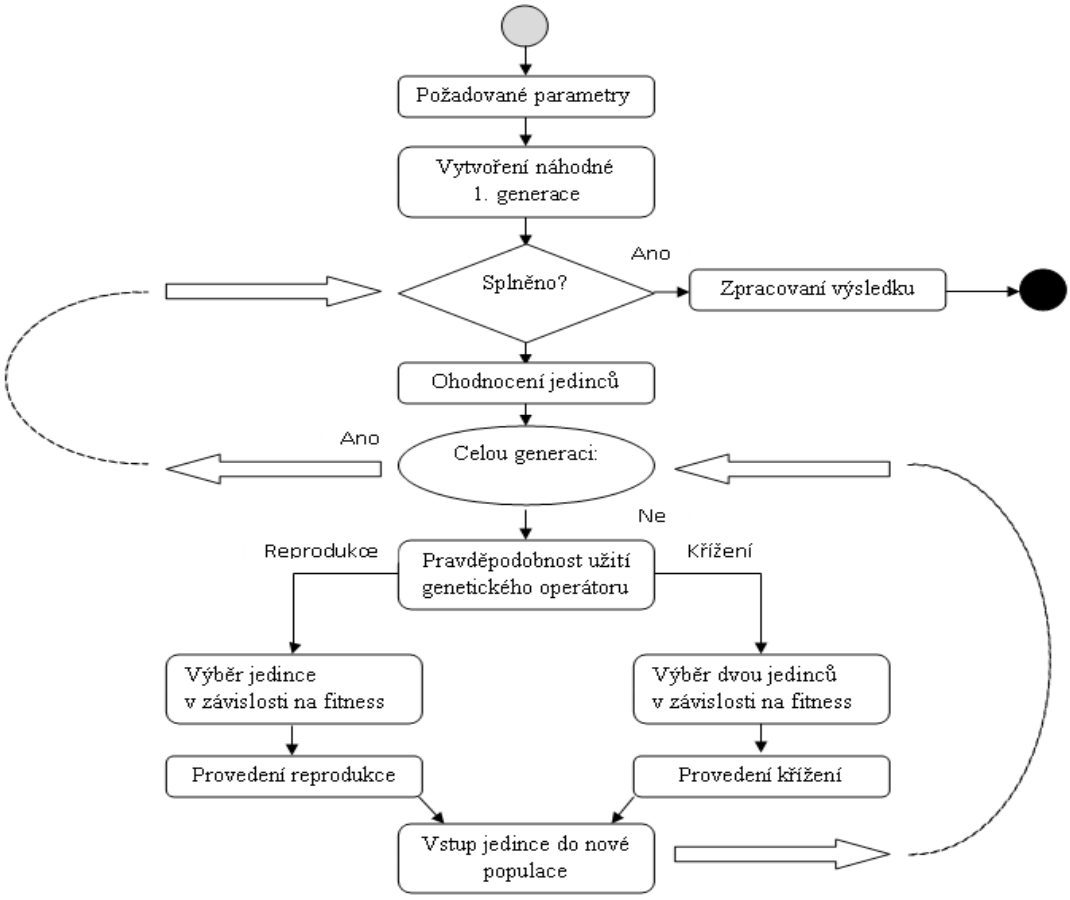
# 3 GENETICKÉ PROGRAMOVÁNÍ

Genetické programování (GP) [5] je odvětvím evolučních výpočetních technik, v podstatě využívající reprezentaci problému používanou v genetických algoritmech. Jedinec, jakožto potenciální řešení daného problému, může být definován jako velmi obecná struktura. Vezmeme-li v úvahu, že požadované struktury mají reprezentovat optimalizaci resp. aproximaci funkčních hodnot neznámé povahy, pak je reprezentace pojata jako generované rovnice funkcí na základě symbolické regrese. K využití a možné implementaci se dostaneme později.

Hlavní přínos v oblasti vývoje a definice GP má John R. Koza,. Jedná se o poměrně mladou disciplínu – první publikace týkající se přímo GP se řadí do 90. let minulého století, např. *Genetic Programming – on the programming of computers by means of natural selection* [5] popř. *Genetic programming II – Automatic Discovery of Reusable Programs* a mnohé další.

## 3.1 Obecný princip

V zásadě lze označit GP jako hledaní nejlépe ohodnoceného (hodícího se) programu ve skupině přípustných programů, kdy populace požadované velikosti – obvykle několika stovek jedinců – je geneticky křížena a mutována. Implementovaná reprodukce probíhá pomocí pravidel „přírodního“ výběru, která mají svůj základ v Darwinově teorii přežití nejschopnějších. Schématicky je proces GP popsán na obr. 2.




*Obr. 2 Model GP – John. R. Koza*

Jak ilustruje výše uvedený obr. 2, algoritmus GP lze rozdělit do několika kroků, které budou následně rozebrány v závislosti na způsobu realizace. Prvním krokem je vytvoření náhodné populace programů dle požadovaných parametrů, povoleného složení programů a řešeného problému.

Iniciační populace je ohodnocena fitness funkcí – užívané typy fitness funkce budou uvedeny níže.. Ohodnocená populace je následně předána genetickým operacím. Zde závisí na volbě parametru, jak velká část populace bude podrobena dané operaci.

První genetický operátor je zpravidla selekce, kdy je program vybrán na základě již zmíněné Darwinovy teorie přírodního výběru, v našem případě dle ohodnocení, které reprezentuje schopnost programu (jedince), přiblížit se řešení daného problému. Zpravidla dochází k výběru jednoho jedince pro reprodukci – zmutovanou, nebo beze změny – a výběru páru určenému ke křížení, kdy si programy vymění sekvence kódu. Vytvoření potomci sou zařazeni do nové populace. Celý proces probíhá pro celou populaci. Je-li podmínka splněna, dojde k prezentaci možných řešení.

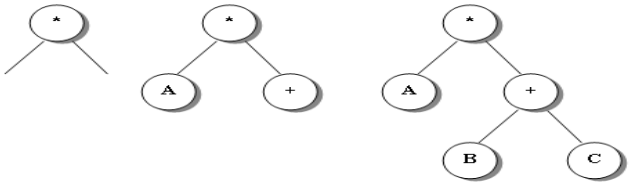
## 3.2 Počáteční a adaptaci podléhající struktury

Prohledávání stavového prostoru probíhá za pomoci populace individui reprezentujících body v daném prostoru,v rozsahu stovek popř. tisíců jedinců. Uvažujeme-li takové individuum, jako strukturu (tato představa vychází již z definice GP jako strukturovaného), potom jsou uvažované struktury tvořené dynamicky generovanými kombinacemi uvažovaných funkcí resp. kombinací zvolené „gramatiky“.

Reprezentace takové struktury je v zásadě v podobě stromu nebo podobné rekurzivně postavené struktury. Tento způsob byl v rámci GP přejat z jazyka LISP, který je ve své podstatě rekurzivní. Tvorba struktury tedy probíhá dynamicky a rekurzivně pomocí náhodných proměnných (atomů, analogicky atomických stavebních prvků) ze zvoleného intervalu, požadovaného typu – obecně nazývaných jako terminály - a přednastavených funkcí (f*unction set*). Function set může v zásadě obsahovat:

- aritmetické operátory (+, -, *, ^ ...)
- matematické funkce (sinus, cosinus, logaritmus …)
- logické operátory (AND, OR, NOT ...)
- řídící struktury a operátory (if, then, else ...)
- iterační funkce – cykly (for, do-while …)
- rekurzi uvozující a jiné dané problematiky se týkající funkce

Po sestavení tedy inicializační struktury seskupují řetězcové výrazy (*S-expression*) individuí seskupených v populaci. S výrazy jsou sestavovány jako náhodně generované stromy s orientovanými hranami, kdy generujeme od kořene (vrcholu) po listy stromu viz. obr. 3




*Obr. 3 Postupná tvorba S-výrazu charakterizovaného stromem*

## 3.3 Fitness – účelová funkce

Účelová funkce vychází z přírodního modelu selekce dle Darwinovy teorie, kdy míra adaptace jedince označuje pravděpodobnost jeho přežití a následné obohacení následujících generací svým genetickým kódem. Genetické programování využívá daný princip při určení takové funkce, která je vhodná pro kontrolu řešení daného problému. Můžeme ji aplikovat jako implicitní, explicitní popř. využít evolučního přístupu, kdy se účelová funkce vyvíjí v rámci herní strategie.

Posouzení kvality jedince v populaci je stěžejní částí celého algoritmu, jak v rámci zajištění konvergence řešení ( v globálním pohledu celého algoritmu), tak z pohledu problematiky samotné konstrukce funkce, tak aby docházelo k objektivnímu ohodnocení jedinců.

### 3.3.1 Čistá („Raw“) fitness

Čistá fitness (RF) je zpravidla postavena na množině tzv. *fitness cases*, tedy předem deklarovaných případů, ve kterých známe vstupní hodnotu a jí odpovídající hodnotu hodnotící funkce. Užití pouze části dané množiny je zpravidla omezena použitím rozdílných částí dané množiny v průběhu generací.

Nejčastěji je hodnocení vytvořené pomocí RF definováno jako chyba. RF každého jedince (S-výrazu), který charakterizuje jedince sestává ze součtu vzdáleností ve všech bodech S-výrazu a hodnot, odpovídajících danému prostoru. Výrazy mohou být mohou být ohodnoceny jako tvrzení (true/false), celé nebo reálné číslo, vektorem, symbolicky, příp. kombinací předchozího.

Uvažujeme-li ohodnocení pomocí číselné konstanty v libovolné tvaru, pak se u GP často užívá sumy absolutních odchylek (SAE), kdy můžeme účelovou funkci (též. fitness) definovat jako:

$$r(i,t) = \sum_{j=1}^{N} S(i,j) - C(i,j) \quad , \tag{3}$$

kde *S(i,j)* je návratová hodnota S-výrazu *i* pro *j-tý* případ z celé množiny případů fitness funkce a *C(i,j)* je správná (očekávaná) hodnota pro tentýž *j-tý* případ fitness funkce.

### 3.3.2 Standardizovaná fitness

Spočívá v přepočtu předchozího typu hodnotící funkce dle daného problému. Například pokud se uvažuje nižší hodnota jako vhodnější pro řešení úprava funkce je definována tvarem:

$$s(i,t) = r(i,t) \tag{4}$$

V opačném případě, kdy vyšší hodnota ukazuje bližší řešení a často je i maximální hodnota omezena, je přepočet funkce následující:

$$s(i,t) = r_{max} - r(i,t) \tag{5}$$

Pro představu, v případě, že máme omezeno hodnocení jedince na 100 – obecně může vyjadřovat 100% úspěšnost nalezeného řešení, hodnocení jedince, výsledná hodnota fitness bude při čistém ohodnocení například 31, rovna 69.

### 3.3.3 Normalizovaná fitness

Normalizovaný tvar fitness funkce zahrnuje proporcionální vyjádření s ohledem na celkový stav. Je definována vztahem:

$$n(i,t) = \frac{a(i,t)}{\sum_{k=1}^{M} a(k,t)} \tag{6}$$

kde:

- výsledná hodnota *n(i, t)* je reálné číslo <0,1>
- fitness je přímo-úměrná vhodnosti jedinců
- Σ *n(t,i)* je rovna 1

Užití této metody je vhodné zejména pokud je využita při selekci – selekce s proporcionálním ohodnocením – jako například turnajová (*tournament*) nebo „hodnostní“ (*rank*) selekce, v opačné případě není její užití doporučeno. Zvolený typ funkce je přímo spjat s technikou selekce jedinců.

## 3.4 Selekce (obecně reprodukce)

Operátor reprodukce si bere za cíl přežití nejschopnějších z Darwinovy teorie. Praktikuje se přímá vazba na hodnotu fitness funkce daného jedince a vždy pracuje pouze s jedním rodičem a potomkem.

Reprodukce probíhá ve dvou krocích. Základem je vybrání vhodných jedinců prostřednictvím selekce založené na metodě fitness – obecně Darwinova teorie uvádí lépe adaptované jedince jako schopnější se reprodukovat, čili mít potomky - v dalším kroku dojde k přidaní jedince do nové generace, která vznikne právě přidáním požadovaného počtu potomků.

Variant reprodukce je velké množství, obvykle se uvažuje v termínech *proporční* a *turnajové reprodukce* (selekce). Proporční reprodukce je založena na myšlence pravděpodobnosti definované jako:

$$p = \frac{f(s_i(t))}{\sum_{k=1}^{M} f(s_i(t))} \quad , \tag{7}$$

kde $f(s_i(t))$ zpravidla označuje normalizované podobu fitness funkce $n(s_i(t))$ jedince $i$ v populaci $M$.

Druhou významnou metodou reprodukce je turnajová selekce. Jak název napovídá, princip je založen na náhodném výběru jedinců (typicky dvou, ale i více) z populace, přičemž je reprodukován jedinec s vyšší fitness. K výběru jedince může dojít opakovaně – velmi dobré ohodnocení jedince znamená jeho mnohonásobnou účast při reprodukci.

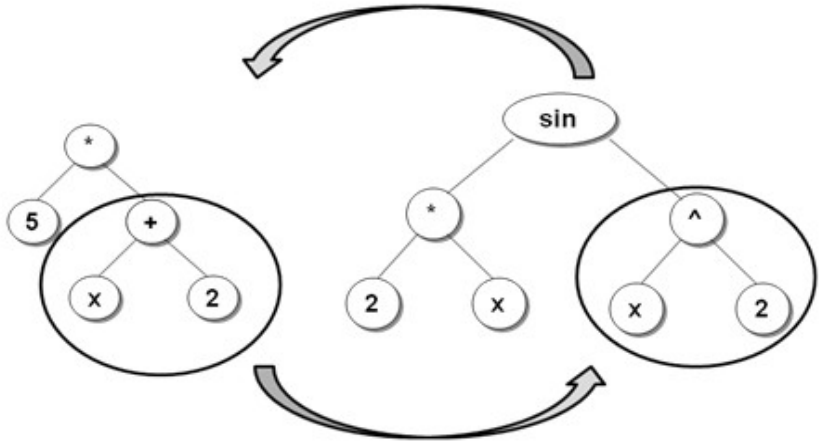
Tento typ selekce podporuje i zavedení elitního přístupu. Elitním přístupem, (kap. 3.4.7), je postup, kdy je porovnáním hodnoty funkce fitness nalezen nejlépe adaptovaný jedinec. Tento elitní jedinec pravděpodobně velmi přispěje svým kódem do další generace, přičemž ale může dojít ke ztracení jeho adaptace křížením s méně vhodným jedincem. Abychom této situaci předešli, vytváříme *kopii* elitního jedince. Toutou kopií je po vytvoření nové generace nahrazen její nejhůře ohodnocený jedinec, čímž zajistíme přežití nejlépe adaptovaného jedince v nezměněné formě.

Turnajová selekce je obecně výhodná pro svou schopnost zachování různorodosti (diverzity)

populace napříč generacemi, což zabraňuje předčasné konvergenci algoritmu, popřípadě uváznutí v lokálním extrému a je velice užitečná při tvorbě rozmanitých řešení daného problému.

## 3.5 Křížení

Křížení – v anglické literatuře označováno jako *„crossover“* – jinak také sexuální rekombinace je základním operátorem, vnášející do populace nové variace za použití informaci od dvou jedinců – rodičů. Určujícím prvkem jsou zde dva jedinci, jejichž kombinací vznikají v nové generaci právě dva potomci. Operátor je v zásadě podobný implementaci v GA, liší se hlavně v místě křížení. Zatímco v genetických algoritmech dochází k výměně polovin binárních řetězců, v rámci GP, kde jsou jedinci charakterizování zpravidla jako binární stromy (nebo jiné rekurzivně provázané struktury), dochází k náhodnému výběru místa křížení, čímž vzniknou podstromy, které se vzájemně vymění viz. obr 4.




*Obr. 4 Ukázka křížení („crossover“)*

V implementaci je následně nezbytné zajistit správnou funkčnost ve smyslu rozsahu náhodné pozice. Jednotlivé prvky číslujeme od kořene (označeného jako 0). Je nezbytné si uvědomit, že jedinci jsou již od první populace náhodně generované délky i tvaru, nemusí tedy nabývat stejné velikosti. V rámci iterací jsou následně aplikovány genetické operátory, které mohou pořadí zásadně pozměnit, proto je nezbytné, aby si algoritmus ohlídal interval pro bod křížení dle délky kratšího z jedinců.

Podrobnější příklad než je ilustrován na obr. 4 bude uveden v implementaci algoritmu na námi sledovaném problému. Nyní se dostaneme k neméně důležitým sekundárním operátorům.

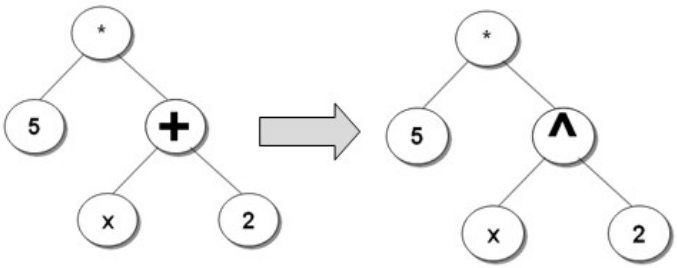
## 3.6 Mutace

Mutace je v literatuře označována jako sekundární operátor [5], v praxi ovšem patří k základním genetickým operátorům, je nepostradatelná.

Zatímco např. křížení provádí změny na úrovních části jedinců, mutaci naproti tomu provádí variace individuí – v případě reprezentace binárním stromem se jedná o změnu některého prvku stromu – a tyto změny vychází z existujících informací jedince, tzn. dle vlastností vybraného prvku.

Mutace je zpravidla náhodná, týká se vždy pouze samotného jedince a simuluje drobnou adaptaci na prostředí. Její vliv není velký, ale v rámci svého působení je nepostradatelná – drobné

změny v každém jedinci populace, jejíž velikost se pohybuje v několika stech až tisících jedinců jistě nemůže být pokládán za zanedbatelný.

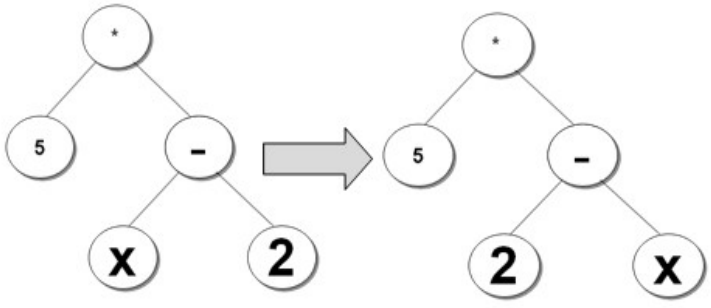



*Obr. 5 Ukázka mutace*

Na obr. 5 můžeme vidět příklad mutace, provedení ovšem závisí na dané problematice a je tedy velice flexibilní v rámci dané implementace. Otázky implementace a specifické vlastnosti – jako například *nedestruktivní* přístup, který dovoluje provést pouze takovou mutaci, která nepoškodí daného jedince – např. Binární operátor pouze za binární, unární za unární, proměnou za konstantu nebo opačně.

## 3.7 Permutace

Permutace v GP je definována jako změna pořadí jednotlivých prvků v S-výrazu mezi dvěma body. Aplikováním na jedince s vysokou mírou adaptace ( příznivá fitness) může přispět k přiblížení se ideálnímu řešení. Vždy se aplikuje na každého jedince v populaci samostatně, způsob výběru je řízen stejnými pravidly reprodukce jako jiné operátory. Uvažujeme-li jedince jako binární strom, pak permutace vždy probíhá pouze na listech stromu, dovolený prvek k možné permutaci je tedy přímo nadřazen koncovým prvkům stromu. Bližší pohled na možný způsob permutace je uveden na obr. 6,




*Obr. 6 Ukázka permutace*

jako ostatní operátory ovšem záleží na zkoumaném problému, kterým se výsledná implementace řídí.

## 3.8 Editace

Operátor editace provádí úpravu a zjednodušení daného jedince (resp. S-výrazu). Aplikován je vždy na jednoho rodiče, výstupem je pouze jeden potomek. Obecně pracuje s množinou „editačních pravidel“, z literatury plyne [5], že funkce obecně je vhodná k editaci v případě, že nemá v žádné vedlejší důsledky a její výsledek nezávisí na proměnných – je tvořen pouze konstantními atomickými prvky. Například:

$$(2 + 1) * 5 = 15 \tag{8}$$

$$true \quad \& \quad true = true \tag{9}$$

Tyto úpravy jsou na první pohled triviální. Uvažujeme-li relativně rozsáhlou délku jedince (S-výrazu), reprezentovaného například binárním stromem, pak užitím editace dojde v rámci populace k nezanedbatelnému zvýšení výkonu GP.

Další možnost využití editace spočívá zejména v rámci „kosmetických úprav“ řešení. V rámci GP možná složitost výstupního výrazu nijak nepřekáží, spíše přispívá k různorodosti populace. Při shledání optimálního řešení (splnění vstupní podmínky), dostaneme například tento S-výraz:

$$\sin(x + 2) + \cos(\sin(0)) * (((2 + 1)\hat{}2)\hat{}(-3)) \tag{10}$$

Tento výraz už pro obsluhu příliš přehledný není, aplikace editačního operátoru je v pořádku a prospěšná, využitím základního pravidla editace dostaneme:

$$\sin(x + 2) + C \quad , \tag{11}$$

kde $C$ je konstanta čísla $6^3$.

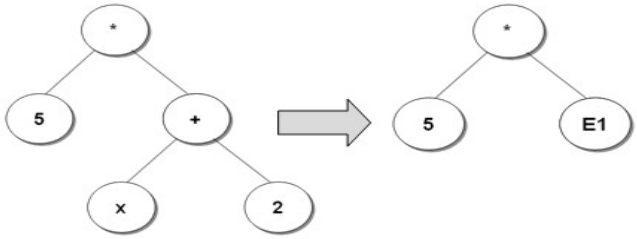
Tato varianta působí na lidského pozorovatele jistě přehledněji a jasněji. V globálním měřítku, ale takto upravený výraz ztratil významnou část svého *variačního potenciálu* v populaci. Aplikace rekombinace (uvažujme základní operátory křížení a mutace viz. výše) v další generaci může původní S-výraz změnit například na:

$$\sin(x + 2) + \cos(\sin(2x) * (((x + 1)\hat{}2)\hat{}(-3)) \tag{12}$$

Této variace již není možno dosáhnout. Námi editovaný výraz této variace jistě nedosáhne a uvážíme-li, že editace je zpravidla implementována na každého jedince populace, je populace značně ochuzena a může snadno zdegenerovat. Při implementaci tohoto operátoru je tedy na místě jistá opatrnost a soubor editačních pravidel sestavován „s rozmyslem“.

## 3.9 Zapouzdření („Encapsulation“)

Tzv. zapouzdření má za úkol automatické rozpoznání potenciálně užitečných ( nadále využitelných) podstromů resp. podvýrazů, jejich pojmenování a „uschování“ pro další použití. V praxi si tento operátor můžeme představit dle obrázku 7.




*Obr. 7 Ukázka zapouzdření*

## 3.10 Elitismus

Elitismus je proces, při kterém dochází k vyhodnocení nejlépe adaptovaného jedince v populaci (dle hodnoty fitness) a jeho nepozměněné reprodukci do další populace. V zásadě se nejedná o operátor rekombinace, spíše o speciální případ selekčního operátoru (způsobu selekce jedinců).

Implementace probíhá uschováním kopie (*„deep copy“*) – uvažujeme-li vyšší programovací jazyk jako například Javu – elitního jedince mimo reprodukci. Průběh rekombinací je stejný, elitní jedinec může ve své podstatě do nové populace zásadně přispět. Po jejich skončení je v populaci vybrán jedinec s nejhorším hodnocením, který je elitním v nezměněném tvaru nahrazen.

## 3.11 Automatické definování funkcí (ADF)

Myšlenka ADF (v literatuře [6] uváděno jako *„Automatic function definition“*) je reakcí na vlastnost GP, která předpokládá předem definovaný soubor povolených funkcí a terminálů (obecně konstant).

Tento soubor „pravidel“ zpravidla obsahuje pouze základní funkce a operátory. Není reálně možné při sestavení těchto parametrů nadefinovat všechny vzorové kombinace (resp. I-kombinaci z reálného oboru). Tento nedostatek zamezuje vytvoření vzorů, které by mohly být hierarchicky nižšími celky hledaného problému.

Pojmem ADF není myšleno generovaní přesných funkcí nebo jejich částí s fixními proměnnými. Dochází k vytváření pomocných funkcí – resp stromů nebo S-výrazů – které mají pevnou strukturu (např. pevná délka a tvar reprezentace jedince), a jejichž výskyt je v rámci daného problému očekáván.

Tyto „předlohy“ nemohou být přímo definovány do množiny a jejich implementace probíhá jako implementace části kódu, kdy hlavní funkce při tvorbě jedince odskakuje do pomocných funkcí.

Automatická definice funkcí může, při nastavení vzorů vhodných pro daný problém, velmi napomoci k zvýšení efektivity GP,

## 3.12 Příklady aplikace GP

Možností aplikace je celá řada, zasahující do různých oblastí. Zde budou uvedeny pouze některé základní resp. často realizované případy, které zdaleka nezahrnují ani zlomek zkoumaných oblastí. Jsou to například:

- *symbolická regrese:* proces, kdy je pomocí GP hledán symbolický předpis funkce, při dané množině funkcí a operátorů. V zásadě je vytvářen program pro aproximaci dané funkce. Tento způsob aplikace je pro tuto práci stěžejní a bude proto níže popsán důkladněji.

- *plánování cesty robota:* GP stejně jako algoritmy užívající princip inteligence hejna nebo jiné typy EVT bylo úspěšně použito při plánování cesty robota – optimalizace cesty, příp. vypořádání se s překážkami při pohybu.

- *Generátory náhodných čísel:* Optimalizace stávajících generátorů pomocí GP – hledání funkčního předpisu generátoru s kontinuálním vstupem.

- *Využití v ekonomii:* V této oblasti byl důraz převážně na simulaci trhu a ekonomických procesů, ověřování hypotéz – např. ověření Keplerova 3. zákona [5] .

Tento výčet je pouhý „zlomek“ úspěšných aplikací využivající genetické programování.[7]

# 4 SYMBOLICKÁ REGRESE

Symbolická regrese (SR) je inverzní postup pro získání symbolického tvaru funkce z testovací množiny dat – souřadnice $x$ a jim odpovídající funkční hodnoty $y$, kdy $y=f(x)$. Problematika SR sestává z tvorby *modelu formy funkce* a z nastavení vhodné množiny jejich *proměnných* [8].

## 4.1 Symbolická regrese s tvorbou konstant

Jedná se o rozšíření symbolické regrese (SR) v předchozím případě závisející pouze na $x$, kdy je problém terminálu (prakticky hodnota symoblického vyjádření funkce rozšířena o element náhodné konstanty (číslo, true/false,null) – tzn. $T \in \{X,R\}$.

V návaznosti na toto rozšíření se v rámci inicializační (náhodně generované) populace objevuje variace konstant resp. parametrů funkčních operátorů. Zpravidla je rozsah možných konstat dán intervalem, jak uvidíme na níže uvedeném příkladu.

Pro představu, uvažujme 20 číselných páru souřadnic $[x,y]$, kde $y=f(x)$, $x \in <-1,1>$. Snahou je najít křivku (či její dobrou aproximiaci) dle dané množiny hodnot. Pro tento příklad je hledaná křívka ve tvaru:

$$y(x) = x^4 + x^3 + x^2 + x \tag{13}$$

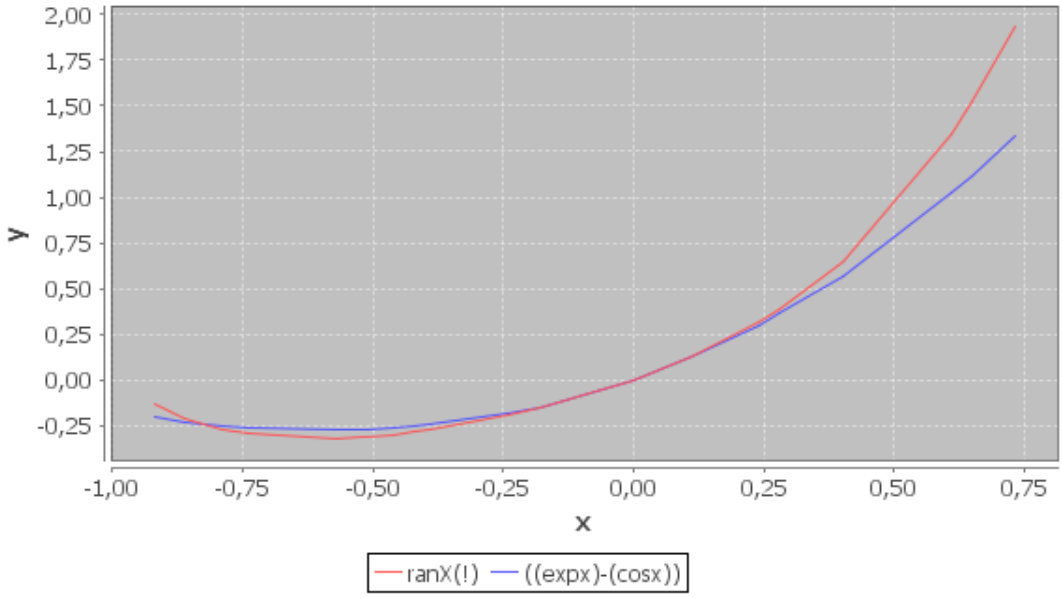
Množina uvažovaných funkcí bude obsahovat: sčítání (+), odčítání (-), násobení (*), celočíselné dělení (/), obecnou mocninu (^), druhou mocninu (^2), funkce cosinus (cos) a exponenciálu (exp). Správné nastavení funkcí, které mohou být nápomocny při řešení. Množina může velmi zeefektivnit rychlost řešení nebo naopak úplně znemožnit nalezení vyhovující formy dané funkce. Součástí příkladu je implementace tvorby konstant z intervalu <-4,4>.

Symbolický předpis funkce a grafické porovnání hledané a regresí nalezené funkce vidíme na obr.10 na následující straně.

Nalezená funkce je tedy:

$$\exp(x) - \cos(x) \tag{14}$$

- interval funkčních hodnot: <-1,1>
- interval variace konstant: <-4,4>
- SAE fitness: ~1,8451
- pořádí populace 88




*Obr. 8 Symbolická regrese funkce $x^4+x^3+x^2+x$*

Tento příklad kombinuje *získání formy funkce* pomocí symoblické regrese a *hledaní vhodných koeficientů* nepolynomiálních funkcí (sinus, cosinus ..), což obvykle zahrnuje pod celkový problém *„symbolická regrese“*. Příklad byl vytvořen v implementaci SR problému, která je součástí této práce – viz. kapitola 5.

## 4.2 Vhodné fitness funkce

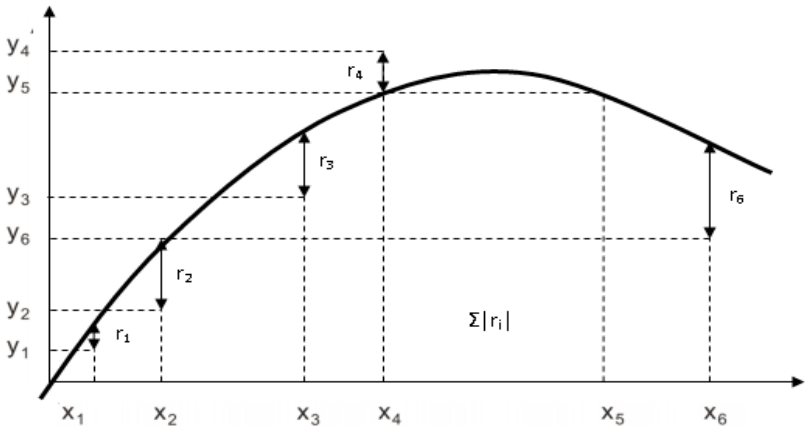
GP je z hlediská cílů optimalizace nutné přizpůsobit konkrétní řešené problematice. Uvažujeme-li aproximaci funkcí pomocí symbolické regrese setkáme se zpravidla s následujícími typy účelových funkcí označených v EVT jako tzv. fitness.

### 4.2.1 SAE („Sum Absolute Error“)

Součet absolutních odchylek (SAE) je jedná ze základních metod posuzování kvality u aproximace funkce. Ohodnocení totou metodou získame dle:

$$\sum |y - y_{GP}| \quad , \tag{15}$$

kde $y$ označuje vysvětlovanou funkční hodnotu $y=f(x)$ a $y_{GP}$ hodnotu funkce získanou pomocí symbolické regrese (tedy výpočtem pomocí GP).

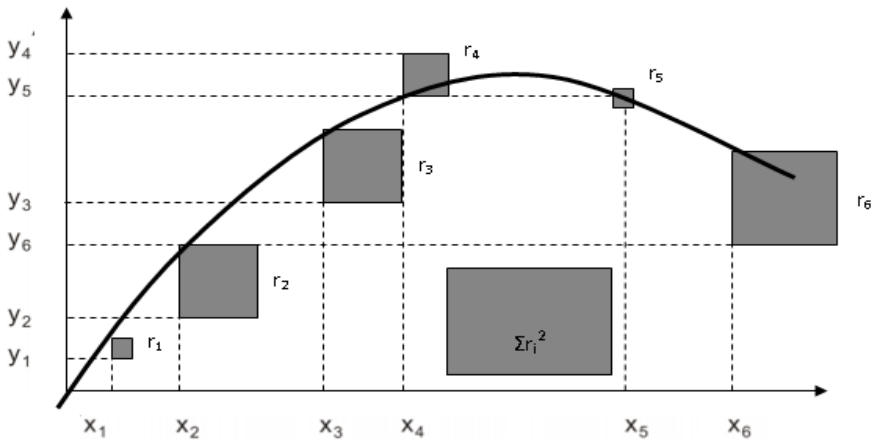



*Obr. 9 SAE fitness funkce*

### 4.2.2 SSE(„ Sum Square Error“)

Součet kvadratických odchylek (SSE) je další ze základních metod posuzování kvality u aproximace funkce. Ohodnocení totou metodou získame dle:

$$\sum (y - y_{GP})^2 \tag{16}$$




*Obr. 10 SEE fitness funkce*

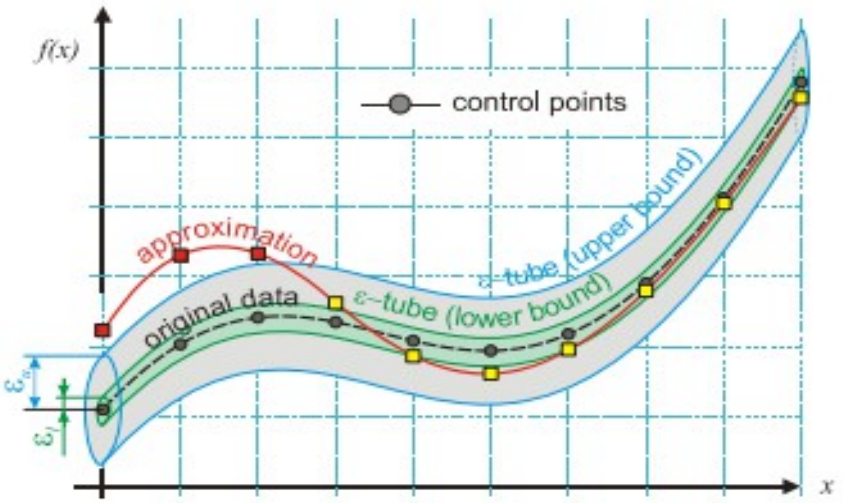
### 4.2.3 STE(„Sum Epsilon-Tube Error“ )

Metoda spočívá v obalení námi hledané funkce tolerančním pásmem, jehož šířka se dynamicky mění. U každého jedince ( generované fukce) posuzujeme, zda se funkční hodnoty *y* nachází v toleranci nebo nikoli. [9]

$$STE_{\varepsilon} = \sum_{i} e_{\varepsilon}(r_i) \quad e_{\varepsilon}(r_i) = \begin{cases} 0 & r_i \notin [-\varepsilon, \varepsilon] \\ 1 & r_i \in [-\varepsilon, \varepsilon] \end{cases}, \tag{17}$$

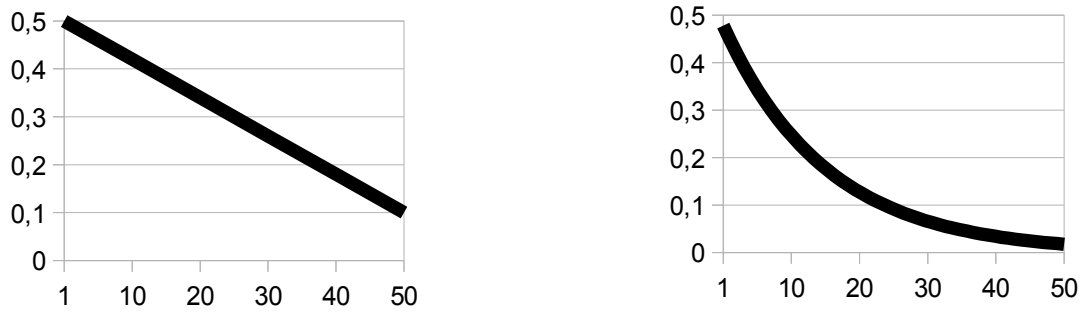
kde:

- $\varepsilon$ označuje poloměr tolerančního pásma (tuby)
- $i$ označuje uvažovaný bod z množiny M
- $e$ je závislost hodnocení na existenci bodu v daném tolerančním pásmu




*Obr. 11 STE fitness funkce* [9]

Vlastní implementace STE počítá zpravidla s nastavením šířky pásma (uvažujme pro inicializaci dostatečně velkou, ovlivnitelnou uživatelem). Pokud část bodů (funkčních hodnot $y_{GP}$), námi zkoumaného jedince, leží v tolerančním pásmu, změníme toleranční pásmo. V praxi bývá použito lineárního popř. procentuálního modelu. STE je poměrně novou metodou ohodnocení a díky vytvoření selekčního tlaku přináší velmi zajímavé výsledky. [9]



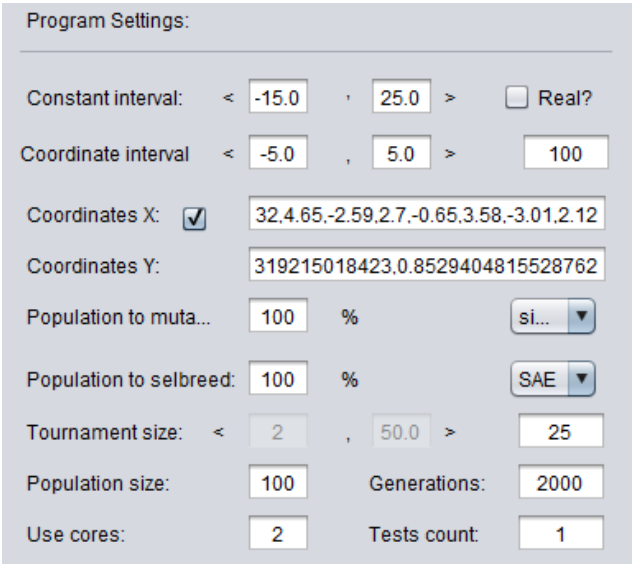

*Obr. 12 Lineární (vlevo) a procentuální model „úbytku“ tolerančního pásma*

# 5 IMPLEMENTACE

Tato část je věnována implementaci GP v jazyce java. Program je navržen jako více-vláknová aplikace s možností parametrizace atributů výpočtu. Výsledná aplikace zahrnuje několik tříd, blíže si popíšeme třídy reprezentující jedince (`TreeIFace.java`), řešiče GP (`Solver.java`), hlavního vlákna, které řídí výpočet a provádí distribuci dat (`SupremeThread.java`) a třídu vlákna, provádějícího výpočty (`WorkThread.java`). Ostatní třídy, na které jsou v následujícím textu případné odkazy jsou popsány v `Javadoc`, který je společně s programem a instrukcemi přílohou této práce.

## 5.1 Grafické uživatelské prostředí (GUI)

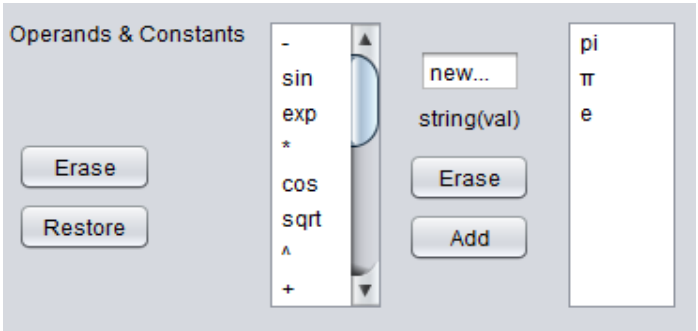
### 5.1.1 Panel nastavení („Program Settings“)




*Obr. 13 Programová nastavení*

Základní panel sloužící k nastavení parametrů řešiče. Podrobné instrukce ohledně konkrétního nastavení jsou uvedené v instrukcích pro obsluhu – příloha práce.

### 5.1.2 Panel operátorů a konstant



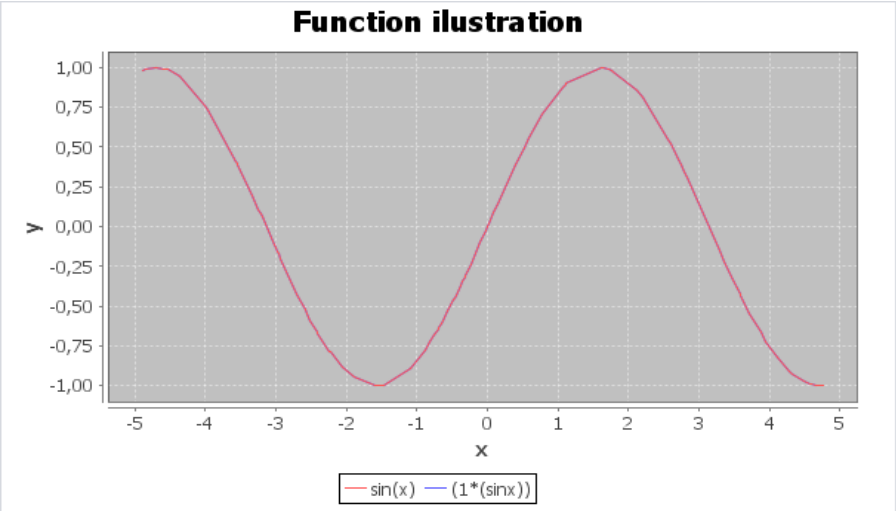

*Obr. 14 Operátory a konstanty*

Na tomto panelu nastavujeme povolené operátory, tedy takové, které mohou být v rámci regrese prospěšné. Výběr zahrnuje operátory, které zastupují námi využitelné funkce `Java.lang.Math`. V rámci nastavení konstant (mimo základních), lze definovat jakékoli další konstanty pomocí tlačítka Add a připojeného textového pole. Konstanty se definují jako číslo (pokud chceme jako konstantu dále nejmenované číslo např. -4) nebo ve tvaru „*JménoKonstanty(HodnotaKonstanty)*“, například κ(1.14). Hodnoty konstant jsou brány v tvaru podporovaném jazykem Java, tedy reálná část čísla musí být oddělena tečkou. Je aplikována kontrola odstranění mezer ze zadaného řetězce, zamezení redudance při vkládání a možnost mazání konstant. Takto nastavené konstanty a jejich, pokud možno jednoduché, matematické úpravy (např. 2-π, 2π..), lze využít jako parametry intervalů pro konstanty nebo souřadnice.

### 5.1.3 Panely testování

Tato část grafického rozhraní byla přidána pro „zpřijemnění“ měření. Skládá se z tabulky pro ukládání výsledků elitních jedinců v jednotlivých testech (počet je závislý na parametru *Tests count* a grafu na ilustraci regrese. Uloženo je ohodnocení fitness nejlépe padnoucí regresivní funkce, její symoblický tvar a údaj o tom, ve které generaci se objevila.

Grafické vyjádření srovnává hledanou funkci danou pomocí funkčních hodnot a námi testovaných případ i symbolicky známou a regresivní funkci získanou pomocí GP. Pro větší testovací sestavy je do tabulky připojen řádek, který je spojen s grafem vyjadřující srovnání 20 nejlepších měření s hledanou funkcí z testovací množiny (její grafická i symbolická reprezentacejenámznáma). Samotný graf v aplikaci vychází z knihoven `jfreeChart-1.0.14.java` a `JCommon-1.0.18.java` [10] [11].




*Obr. 15 Grafické porovnání hledané funkce a její symoblické regrese*

### 5.1.4 Panel ovládání/průběhu

Mimo ovládacího prvku (tlačítko Start), obsahuje průběžně aktualizované informace o aktuální generaci a elitního jedince *aktuální* populace.

## 5.2 Třída TreeIFace

Implementuje reprezentaci jedince jako binární strom, obsahuje metody pro správu jedince, základní úpravy, sestavení S-výrazu a výpočet funkční hodnoty y=f(x) pro dané souřadnice x. Hlavními atributy jsou:

- instance třídy `Constants` – definované konstanty a metody generující náhodná čísla dle parametrů
- instance třídy `Operands` – povolené operátory a související metody
- instance `java.util.linkedlist<Node>` - oboustranný spojový seznam, obsahující rekurzivně propojené prvky stromu (jedince)
- double `Rating` (reálné číslo) fitness ohodnocení stromu
- několik instancí zásobníků použitých při výpočtech (viz. metody `evaluate` a jim podřízené) a další private atributy

Blížší popis metod, definovaných v této třídě (pouze `public`, tedy přístupné mimo třídu, její instance a potomky) se nachází v tab.1.

| | |
|---|---|
| Metoda | `public TreeIFace(Constants c,Operands o)` |
| Parametry | `c` - databáze konstant @see Constants<br>o - databáze operátorů @see Operands |
| Popis | Konstruktor – vytváří instanci třídy s požadovanými parametry. |
| Metoda | `public void add(java.lang.String aValue)` |
| Parametry | `aValue` - datový řetězec prvku |
| Popis | Vytvoří kořen (prvek stromu s indexem 0, který nemá předchůdce) s hodnotou odpovídající @param aValue. |
| Metoda | `public void add(java.lang.String aValue,int aPrev)` |
| Parametry | aValue - datový řetězec pro prvek<br>aPrev - pozice předchůdce v listu @see Nodes |
| Popis | Vytvoří prvek stromu s hodnotou odpovídající @param aValue jako následovníka @param aPrev. Dle stavu a typu dat v předchůdci zvolí pozici (levý [0], pravý[1]. Binární operátory jsou plněny zprava. V případě existence 2 následovníků (bin. strom), nahradí nahodně vybraného. |
| Metoda | `public void change(int aIndex,java.lang.String aValue)` |
| Parametry | aValue - datový řetězec pro prvek<br>aIndex - pozice měněného v listu @see Nodes |
| Popis | Změní hodnotu prvku na pozici @param aIndex na hodnotu @param aValue. V rámci provázání jednotlivých prvků (uzlů) resp. podstromů je změna aplikována i v odkazech na předchůdce v následovnících. |
| Metoda | `public java.lang.String getValue(int aIndex)` |
| Parametry | `aIndex` - pozice prvku<br>**Returns:** hodnota prvku (String) |
| Popis | Vyžádá hodnotu daného prvku @param aIndex. |

| | |
|---|---|
| Metoda | `public Node getNode(int aIndex)` |
| Parametry | `aIndex` - pozice prvku v listu @see Nodes<br>**Returns:** prvek (Node) |
| Popis | Vyžádá daný prvek stromu @param aIndex. |
| Metoda | `public java.lang.String getNodetoStr(int aIndex)` |
| Parametry | `aIndex` - pozice prvku v listu @see Nodes |
| Popis | Jedná se o pomocnou metodu, využitelná při ladění – přepis hodnoty prvku a jeho následovníků do přehledného stringu |
| Metoda | `public int size()` |
| Parametry | **Returns:** int velikost |
| Popis | Vrací velikost stromu reprezentovaného třídou @see TreeIFace. |
| Metoda | `public boolean isOperand(java.lang.String aString)` |
| Parametry | aString - hodnota prvku<br>**Returns:** boolean true jedná-li se o operátor, jinak else. |
| Popis | Kontrola zda je hodnota daného prvku operátor z @see operands. |
| Metoda | `public boolean isConst(java.lang.String aString)` |
| Parametry | aString - hodnota prvku<br>**Returns:** boolean true jedná-li se o konstantu z výčtu, jinak else. |
| Popis | Kontrola zda je hodnota daného prvku konstanta z @see constants. |
| Metoda | `public int getOperandSide(java.lang.String aOperand)` |
| Parametry | aOperand - hodnota operátoru<br>**Returns:** int 0-levostranný, 1-pravostranný, 2-binární |
| Popis | Vrací typ operátoru - binární, unární (levo-pravostranný). |
| Metoda | `public boolean isVariable(java.lang.String aString)` |
| Parametry | aString - hodnota prvku<br>**Returns:** boolean true jedná-li se o x, jinak else. |
| Popis | Kontrola zda je hodnota daného prvku proměnnou x. |
| Metoda | `public boolean isAvailable(Node aNode)` |
| Parametry | aNode - hodnota prvku<br>**Returns:** boolean true, pokud @param aNode je nenulový a nebyl kontrolován(vyhodnocen), jinak false. |
| Popis | Dílčí metoda podílející se na vyjádření stromu. |
| Metoda | `public boolean isNoLongerAvailable(Node aNode)` |
| Parametry | aNode - zkoumaný prvek<br>**Returns:**boolean true, pokud @param aNode je nenulový a byl kontrolován, jinak false. |
| Popis | Dílčí metoda podílející se na vyjádření stromu. Při výpočtu hodnoty/S-výrazu stromu je prvek, který byl kontrolován označen jako checked=true. |
| Metoda | `public boolean isNegative(java.lang.String aValue)` |
| Parametry | aValue - hodnota prvku(číslo ve formě řetězce)<br>**Returns:**boolean true, pokud substring @param aValue obsahuje mínus. |
| Popis | Kontrola signatury číselné hodnoty prvku. |

| | |
|---|---|
| Metoda | `public double evaluate(double aX)` |
| Parametry | aX - hodnota souřadnice x<br>**Returns:** double číselnou hodnotu stromu pro dané x. |
| Popis | Číselné vyjádření stromu pro danou hodnotu x. Zastřešení výpočtu jednotlivých prvků. Pro každý volá podfunkci uvedenou níže. |
| Metoda | `public java.lang.String evaluate()` |
| Parametry | **Returns:** String stromové vyjádření převedené na řetezec. |
| Popis | Symbolické vyjádření stromu.. Zastřešení vyjádření jednotlivých prvků. Pro každý volá podfunkci uvedenou níže. |
| Metoda | `public double evalOperation(java.lang.String aOperand, java.lang.String aValue1,java.lang.String aValue2)` |
| Parametry | aOperand - operátor výpočtu<br>aValue1 - hodnota levé strany (levý následovník)<br>aValue1 - hodnota pravé strany (pravý následovník)<br>**Returns:** void (beznávratový typ) |
| Popis | Dle vstupních parametrů provede odpovídající výpočet. Nezávislá na typu operátoru (bin., unární). |
| Metoda | `public void bubbleSort()` |
| Parametry | - |
| Popis | BubleSort algoritmus, využívá se v rámci křížení pro utřídení prvků v listu @see Nodes - snažší manipulace. |
| Metoda | `public void resetPosition()` |
| Parametry | - |
| Popis | Opravuje hashposition (klíč udávají pozici prvku v listu @see Nodes na odpovídající hodnotu po třídění předchozí metodou. Prvky, které odpovídajíc @param null, jsou odstraněny. |
| Metoda | `public double getX()` |
| Parametry | **Returns:** x |
| Popis | Vrací hodnotu x. |
| Metoda | `public void setX(double x)` |
| Parametry | x - nastavované x |
| Popis | Nastaví hodnotu x. |
| Metoda | `public java.lang.String getName()` |
| Parametry | **Returns:**the name |
| Popis | Vrací jméno stromu (jedince). Využívá se pouze při křížení. @see Solver.gpSelBreeding |
| Metoda | `public void setName(java.lang.String name)` |
| Parametry | `name` - the name to set |
| Popis | Nastaví jméno stromu (jedince). Využívá se pouze při kříženi @see Solver.gpSelBreeding |

| | |
|---|---|
| Metoda | `public double getRating()` |
| Parametry | **Returns:** the rating |
| Popis | Vrací hodnocení stromu (fitness) |
| Metoda | `public void setRating(double rating)` |
| Parametry | `rating` - the rating to set |
| Popis | Nastaví hodnocení stromu (fitness) |
| Metoda | `public int getHashposition()` |
| Parametry | **Returns:** the hashposition |
| Popis | Vrací pozici jedince v populaci. @see Solver.firstGeneration() |
| Metoda | `public void setHashposition(int hashposition)` |
| Parametry | `hashposition` - the hashposition to set |
| Popis | Nastaví pozici jedince v populaci. @see Solver.firstGeneration() |

*Tab. 1 Public metody třídy `GPJava.TreeIFace.java`*

## 5.3 Třída Solver

Charakterizuje samotný řešič. Implementuje pokročílé operátory GP (nedestruktivní operace – mutace, křížení – elitní přístup, zjednodušování výrazů – operátor redukce. Aplikuje symbolickou regresi ve smyslu *nalezení symoblické formy* hledané funkce *zároveň s nalezením koeficientů* nepolynomiálních funkcí. Hlavními atributy jsou:

- instance třídy `Constants` – definované konstanty a metody generující náhodná čísla dle parametrů
- instance třídy `Operands` – povolené operátory a související metody
- instance `java.util.linkedlist<Node>` - oboustranný spojový seznam, obsahující rekurzivně propojené prvky stromu (jedince)
- instance `GPJava.TreeIFace` obsahující „hlubokou kopii“ elitního jedince populace v každé generaci
- pole instancí `GPJava.TreeIFace` reprezentující populaci
- pole instancí `GPJava.TreeIFace` reprezentující populaci po turnajové selekci
- atributy intervalů a typů (celé/reálné) souřadnic a konstant souřadnice x a y, instance podstromů atd. obecně všechny nezbytná nastavení z GUI a další pomocné privátní prvky (`private`).

Blížší popis metod, definovaných v této třídě (pouze `public`, tedy přístupné mimo třídu, její instance a potomky) se nachází v tab.2.

| | |
|---|---|
| Metoda | `public java.lang.String writeX()` |
| Parametry | **Returns:** řetezec ve tvaru: 5,-2,.. |
| Popis | Vrací řetezec souřadnic x, automaticky generované pro vybranou funkci. |
| Metoda | `public java.lang.String writeY()` |
| Parametry | **Returns:** řetezec ve tvaru: 5,-2,... |
| Popis | Vrací řetezec souřadnic y, funkčních hodnot x vybrané funkce. |
| Metoda | `public void automaticCoordinates(int aWhat, int aNumber)` |
| Parametry | aWhat - pozice v comboboxu - určuje předefinovanou funkci<br>aNumber - požadovaný počet bodů |
| Popis | Vyhrazuje paměť pro pole souřadnice, volá podfunkci generující souřadnice. |
| Metoda | `public void generateFunction (java.lang.String aFunction, int aNumber)` |
| Parametry | aFunction - název zvolené funkce<br>aNumber - požadovaný počet bodů |
| Popis | Generuje souřadnice x zvolené funkce dle intervalu a typu (reálné/celé) zvoleného v GUI a provádí výpočet funkčních hodnot y. Zapisuje do polí v Solveru. |
| Metoda | `public void readX(java.lang.String aX)` |
| Parametry | aX - řetezec souřadnic x ve tvaru: -5,0,2... |
| Popis | Čte souřadnice x z GUI a načítá do pole v Solveru. Aplikována kontrola mezer mezi souřadnicemi a jejich odstranění |
| Metoda | `public void readY(java.lang.String aY)` |
| Parametry | `aY` - řetezec souřadnic y ve tvaru: -5,0,2... |
| Popis | Čte souřadnice y z GUI a načítá do pole v Solveru. Aplikována kontrola mezer mezi souřadnicemi a jejich odstranění. |
| Metoda | `public void secureElite()` |
| Parametry | - |
| Popis | Vytváří hlubokou kopii (deep copy) elitního stromu populace, která je uschována v atributu @param elite v Solveru |
| Metoda | `public int getWorst(TreeIFace[] aPopulation)` |
| Parametry | aPopulation - pole obsahující populaci jedinců<br>**Returns:** int pozice v populaci |
| Popis | Vrací pozici jedince (stromu) s nejhorším ohodnocením |
| Metoda | `public int getElite(TreeIFace[] aPopulation)` |
| Parametry | aPopulation - pole obsahující populaci jedinců<br>**Returns:** int pozice v populaci |
| Popis | Vrací pozici jedince (stromu) s nejlepším (elitním) ohodnocením. |
| Metoda | `public void isExisting (TreeIFace curr, int aFrom,int aTo,int beyond)` |
| Parametry | curr - kontrolovaný jedinec; aFrom - začátek oblasti pro kontrolu<br>aTo - konec kontrolované oblasti |
| Popis | Součást konstroly populace vhodné pro reprodukci. Zabraňuje vícenásobné mutaci. |

| | |
|---|---|
| Metoda | `public void protectRedu(TreeIFace[] aPopulation)` |
| Parametry | `aPopulation` - pole obsahující populaci jedinců |
| Popis | Kontrola v rámci populace vhodné pro reprodukci. Populace vytvořená selekcí může obsahovat stejného jedince několikrát, mutace například by měla být na daném jedinci provedena pouze jednou. O zamezení opakované mutace jedince v jedné generaci se stará podmetoda isExisting. |
| Metoda | `public void selection(int aFrom, int aTo)` |
| Parametry | aFrom - začátek oblasti selekce; aTo - konec oblasti selekce |
| Popis | Turnajová selekce po částech populace. |
| Metoda | `public void rewritePop()` |
| Parametry | - |
| Popis | V rámci reprodukčních operátorů se jedinci pracuje v rámci odkazů. V případě změny dochází okamžitě ke změně jedince, tato metoda provádí sekundární zabezpečení změny jedinců a zajišťuje znovupoužitelnost atributů. |
| Metoda | `public void fitness(TreeIFace aTree)` |
| Parametry | aTree - hodnocený strom (jedinec)<br>aTo - konec oblasti selekce |
| Popis | Nastavuje hodnotu fitness daného jedince pro všechna x. Typ fitness funkce je určen atributem, nastavitelným z GUI. |
| Metoda | `public void firstGeneration()` |
| Parametry | - |
| Popis | Generuje náhodnou inicializační populaci Využití ADF - obsahuje předpokládané využitelné podstromy. |
| Metoda | `public byte randomChance()` |
| Parametry | **Returns:** byte s pravděpodobnosti: 5%: 0; 85%: 1; 10%: -1 |
| Popis | Generovaná pravděpodobnost - užití v mutaci |
| Metoda | `public int randomNode(int aSize)` |
| Parametry | aSize - velikost jedince (stromu)<br>**Returns:** id |
| Popis | Vrací id náhodného prvku stromu Využití v mutaci. |
| Metoda | `public int mutation(Solver curr,int[] partSize,int mutated)` |
| Parametry | curr - řešič @see Solver, obsahující populaci na niž má být metoda aplikována.<br>partSize - oblast aplikace - paralelní výpočet více vlákny<br>mutated - počet již zmutovaných jedinců<br>**Returns:** mutated++ - počet nově zmutovaných jedinců |
| Popis | Reprodukční metoda mutace - NEDESTRUKTIVNÍ náhodná změna prvku v jedinci. |
| Metoda | `public int selbreading (Solver curr,int[] partSize,int selBredeed)` |
| Parametry | curr - řešič @see Solver, obsahující populaci na niž má být metoda aplikována.<br>partSize - oblast aplikace - paralelní výpočet více vlákny<br>selBredeed- počet již křížených jedinců<br>**Returns:**selBredeed- počet nově zkřížených dvojic |
| Popis | Reprodukční metoda křížení - NEDESTRUKTIVNÍ výměna podstromů. Volá podm. |

| | |
|---|---|
| Metoda | `public void reduction(Solver curr,int[] partSize)` |
| Parametry | curr - řešič @see Solver, obsahující populaci na niž má být metoda aplikována.<br>partSize - oblast aplikace - paralelní výpočet více vlákny |
| Popis | Reprodukční metoda redukce - zjednodušení jedinců (S-výrazů). Volá podmetodu gpSelfBreeding. |
| Metoda | `public void gpMutation(TreeIFace aTree)` |
| Parametry | `aTree` - mutovaný jedinec |
| Popis | Mutace - výkonná metoda pro daného jedince Implementace nedestruktivní mutace. Provede změnu náhodného prvku binárního stromu reprezentujícího jedince. |
| Metoda | `public TreeIFace gpReduction(TreeIFace aTree)` |
| Parametry | aTree - redukovaný jedinec<br>**Returns:** void (beznávratový typ) |
| Popis | Redukce - nadřazená metoda - provádí "denullizaci" prvků jedince Volá podmetodu @see gpReduPart, dále odstraňuje null prvky vzniklé úpravou a aktualizuje pozice @see TreeIFace.resetPosition() |
| Metoda | `public void gpReduPart(Node aNode, TreeIFace aTree)` |
| Parametry | aTree - redukovaný jedinec<br>aNode - kontrolovaný prvek stromu (jedince) |
| Popis | Redukce - rekurzivní metoda kontroly stromu a redukce V případě možnosti redukce volá podmetodu @see reduAction. |
| Metoda | `public void reduAction(Node prev,TreeIFace aTree)` |
| Parametry | aTree - redukovaný jedinec<br>prev - předchůdce redukovaných prvků, který bude modifikován |
| Popis | Redukce - výkonná metoda Volána z nadřazené metody @see gpRedupart. |
| Metoda | `public void gpSelBrdPart(Node aNode)` |
| Parametry | aTree - redukovaný jedinec<br>prev - předchůdce redukovaných prvků, který bude modifikován |
| Popis | Křížení - rekurzivní metoda, připravující stromy na samotné křížení @see gpSelfBreeding Volána z nadřazené metody @see gpSelfBreeding. |
| Metoda | `public void gpSelBreeding(TreeIFace aFirst, TreeIFace aSecond)` |
| Parametry | aFrist - první rodič<br>aSecond - druhý rodič |
| Popis | Křížení - výkonná metoda Volána z nadřazené metody @see selfBreeding. Pomocí podmetody gpSelBreeding převádí v případě rozsahlejších stromů, z odkazů na podstromy hodnoty typu označené klíčem (@see Result) do zásobníků (@see Stack). Nejprve prohodí hodnoty @see swapValues na počátečním místě kříženi a vytvoří pár, který obsahuje informaci původních pozic prohozených hodnot. Existují-li u následujících prvků následovnící v potřebném počtu jsou hodnoty prohozeny, v opačném případě je vytvořen nový prvek.Dodržení pořadí vytvářených prvků - @see order. Přebývající prvky stromů, jejichž hodnota je nastavena na null změní na null a odstraní @see TreeIFace.resetPosition |
| Metoda | `public int swapValues (TreeIFace aFirst, TreeIFace aSecond,int index,int pointer)` |
| Parametry | aFirst - první rodič<br>aSecond - druhý rodič |

| | |
|---|---|
| | index - index pozice v podstromech,která se má měnit - stejný pro obsa<br>pointer - ukazatel na vhodný indexaci obsahující pár @see this.pairs<br>**Returns:** ukazatel @param pointer posunutý o 1 |
| Popis | Provádí výměnu hodnot jednotlivých existujících prvků. Volána nadřízenou metodou gpSelBreeding. Pokud oba rodiče mají prvku se stejnou pozicí, prochází zásobníky hodnot a vytváří páry hodnot,které prohodí. Toto prohození zaznamená a přídá do seznamu změn, obecně seznamu výměněných dvojic @see this.pairs |
| Metoda | `public void order(TreeIFace aAdd, TreeIFace aFrom)` |
| Parametry | aAdd - jedinec do kterého je přídáváno<br>aFrom - jedinec ze kterého je přídáváno<br>index - index pozice v podstromech,která se má měnit - stejný pro oba |
| Popis | Obstarává vkládání v daném rodiči neznámých prvků. Volána nadřízenou metodou gpSelBreeding. V případě, že se liší struktura rodičů, je třeba přidávat nové prvky ve správném pořadí, v závislosti na principu metody přidání @see TreeIFace.add(value,prev). Postupně kontroluje a přídává všechny hodnoty v zásobníku daného podstromu. V případě, že předchůdce je binárni operátor resp. má 2 následovníky volá podmetodu findSecond, pro nalezení dvojce. V rámci vytvoření nových prvků opět přidá indexový pár do dokumentace @see this.pairs. |
| Metoda | `public double[] getX()` |
| Parametry | **Returns:** the x |
| Popis | Vrací pole souřadnic x. |
| Metoda | `public void setX(double[] x)` |
| Parametry | `x` - the x to set |
| Popis | Nastaví pole souřadnic x. |
| Metoda | `public double[] getY()` |
| Parametry | **Returns:** the y |
| Popis | Vrací pole souřadnic y. |
| Metoda | `public void setY(double[] y)` |
| Parametry | `y` - the y to set |
| Popis | Nastaví pole souřadnic y. |
| Metoda | `public double getMutateParam()` |
| Parametry | **Returns:** the mutateParam |
| Popis | Vrací počet mutovatelných jedinců v závislosti na velikosti populace. |
| Metoda | `public void setMutateParam(double mutateParam)` |
| Parametry | `mutateParam` - the mutateParam to set |
| Popis | Nastaví počet mutovatelných jedinců v závislosti na velikosti populace. |
| Metoda | `public double getSelBrdParam()` |
| Parametry | **Returns:** the selBrdParam |
| Popis | Vrací počet páru pro křížení v závislosti na velikosti populace. |
| Metoda | `public void setSelBrdParam(double selBrdParam)` |
| Parametry | `selBrdParam` - the selBrdParam to set |
| Popis | Nastaví počet páru pro křížení v závislosti na velikosti populace. |

| | |
|---|---|
| Metoda | `public int getTournamentMembers()` |
| Parametry | **Returns:** the tournamentMembers |
| Popis | Vrací počet účastníků selekčního turnaje. |
| Metoda | `public void setTournamentMembers(int tournamentMembers)` |
| Parametry | `tournamentMembers` - the tournamentMembers to set |
| Popis | Nastaví počet účastníků selekčního turnaje. |
| Metoda | `public int getPopulationSize()` |
| Parametry | **Returns:** the populationSize |
| Popis | Vrací velikost populace. |
| Metoda | `public void setPopulationSize(int populationSize)` |
| Parametry | `populationSize` - the populationSize to set |
| Popis | Nastaví velikost populace. |
| Metoda | `public java.lang.String getFitnessStrategy()` |
| Parametry | **Returns:** the fitnessStrategy |
| Popis | Vrací String-označní zvoleného typu fitness funkce. |
| Metoda | `public void setFitnessStrategy (java.lang.String fitnessStrategy)` |
| Parametry | `fitnessStrategy` - the fitnessStrategy to set |
| Popis | Nastaví String-označní zvoleného typu fitness funkce. |
| Metoda | `public double[] getEvalY()` |
| Parametry | **Returns:** the evalY |
| Popis | Vrací pole funkčních hodnot vypočítaných z generovaného jedince. |

*Tab. 2 Public metody třídy `GPJava.Solver.java`*

## 5.4 Třída SupremeThread

Jedná se o upravené vlákno, které inicializuje program, přiděluje zdroje a spouští vlákna vykonávající výpočet dle diagramu GP (viz obr.4 ). Hlavními atributy jsou:

- instance třídy `GPJava.TreeIFace` – elitní jedinec v případě nalezení ideálního řešení
- instance třídy `GPJava.TreeIFace` – elitní jedinec v případě nalezení přípustého řešení
- řetězce `java.lang.String` – symoblický předpis (rovnice) elitního jedince (za běhu/po skončení)
- další pomocné atributy (`private`)

Blížší popis metod, definovaných v této třídě (pouze `public`, tedy přístupné mimo třídu, její instance a potomky) se nachází v tab.4.

| | |
|---|---|
| Metoda | `public void initThread (int cores,int jMax,gpjava.Frame.Counter t)` |
| Parametry | cores - dovolený počet podvláken<br>jMax - maximální počet iterací<br>t - instance vlákna counter (obnovení částí GUI v rámci výpočtu) |
| Popis | Předává řídícímu vláknu potřebné informace. O předání řešice se stará Konstruktor `public SupremeThread(Solver solver)` |
| Metoda | `public void start()` |
| Parametry | **Overrides:** start in class java.lang.Thread |
| Popis | Běh hlavního vlákna Průběh je cyklický dle počtu iterací, proto dojde k vyčištění potřebných atributů, následuje volání řešící metody. |
| Metoda | `public TreeIFace secureElite()` |
| Parametry | **Returns:** strom reprezentující elitního jedince |
| Popis | Vytvoření hluboké kopie elitního jedince |
| Metoda | `public double getEliteRating` |
| Parametry | **Returns:**the eliteRating |
| Popis | Vrací ohodnocení (fitness) elitního jedince |
| Metoda | `public void setEliteRating(double eliteRating)` |
| Parametry | **Parameters:** eliteRating - the eliteRating to set |
| Popis | Nastaví ohodnocení (fitness) elitního jedince |
| Metoda | `public int getEliteJ()` |
| Parametry | **Returns:** the eliteJ |
| Popis | Vrací počet generací. |

*Tab. 3 Public metody třídy `GPJava.SupremeThread.java`*

## 5.5 Třída WorkThread

Jedná se o upravené vlákno, provádějící samotný výpočet - volání operací GP, v rámci přiděleného prostoru. Aplikace podporuje užití více vláken, tzn. zrychlení výpočtu při spuštění na vícejádrovém procesoru. V praxi dochází k rozdělení populace dle počtu vláken (parametr *Cores* v GUI). Jednotlivé operace jsou prováděny současně několika vlákny, přičemž každé má na starosti část populace.

Hlavními atributy třídy jsou:

- instance třídy `volatile GPJava.Solver` – nastavený řešič předaný řídícím vláken, uskladněný v paměti RAM
- číslo typu `volatile java.lang.Integer` – ID operace, která má být provedená, předané řídícím vláken a uskladněné v paměti RAM
- pole typu `volatile java.lang.Integer` – část populace, značená jako <od,do>, předaný řídícím vláken a uskladněné v paměti RAM

# 6 TESTOVÁNÍ VYBRANÝCH FUNKCÍ

V předchozích kapitolách byly stručně popsány, jak teoretické základy GP a souvisejících výpočetních technik, tak vytvořená aplikace, kterou nyní budeme testovat na množině zadaných kontrolních funkcí.

## 6.1 Testovací funkce

V rámci testovaní se setkáme s těmito funkcemi („*FormatNazvu*“, korespondující s GUI):

- „Quintic“ - polynom 5. řádu v daném intervalu

$$x^5 - 2x^3 + x \quad , \quad x \in \langle -1,1 \rangle \tag{18}$$

- „Sextic“ - polynom 6. řádu v daném intervalu

$$x^6 - 2x^4 + x^2 \quad , \quad x \in \langle -1,1 \rangle \tag{19}$$

- „Sin1“ - součet funkcí sinus s rostoucími koeficienty v daném intervalu

$$\sin x + \sin 2x + \sin 3x + \sin 4x \quad , \quad x \in \langle -\pi, \pi \rangle \tag{20}$$

- „Sin2“ - varianta funkce sinus

$$x \sin \frac{x}{2} \quad x \in \langle -\pi, \pi \rangle \tag{21}$$

## 6.2 Nastavení parametrů řešiče

Nastavení vhodných parametrů pro řešění může být poněkud ošemetné a již víme, že algoritmy GP je v rámci užití nezbytné *nastavit pro konkrétní problém*. V následující části bude navržen princip optimalizace parametrů GP pro námi uvažovaný problém (symbolická regrese testovacích funkcí, viz. kap. [6.1]).

V zásadě uvažujeme se:

- velikosti populace – 100 jedinců, turnaj 2 jedinců
- intervaly souřadnic
- intervaly pro tvorbu konstant <1,9>
- % křížení 75 a mutace 1,2 a 5
- 100 běhů pro danou kombinaci
- povolené operátory: +, -, *, /, ^, ^2, abs, sinus, cosinus, log, exp, tgh
- v rámci této parametrizace je maximální počet generací roven 1000

Výsledné hodnoty 100 testování, každé kombinace % parametrů rekombinačních operátorů (křížení, mutace), srovnáme v tabulce, popř. pomocí boxového grafu, kdy nejlepších 25 % generovaných funkcí, porovnáme dle fitness ohodnocení pomocí 3D grafu. Získané optimální nastavení na daný problém, aplikujeme v rámci 100 testů každé funkce, při použití každé metody ohodnocení (fitness).

### 6.2.1 Funkce „Quintic“ - $x^5-2x^3+x$ - parametr mutace na 1, 2 a 5%, metoda SAE

(značení: Q1,Q3 – první a třetí kvartál, 25-75Pct – rozložení hodnot v rámci testů)

| | mutace 1% | mutace 2% | mutace 5% |
|---|---|---|---|
| Průměr | 11,44754 | 7,399382 | 8,391276 |
| Min | 9,44E-16 | 7,22E-16 | 9,26E-16 |
| Q1 | 11,95076 | 5,048062 | 7,981741 |
| Medián | 11,96089 | 7,82362 | 9,016115 |
| Q3 | 11,98743 | 9,772188 | 9,187036 |
| Max | 15,7526 | 11,67351 | 17,36475 |
| 25Pct | 11,95076 | 5,048062 | 7,981741 |
| 50Pct | 0,010123 | 2,775558 | 1,034375 |
| 75Pct | 0,026547 | 1,948567 | 0,170921 |
| min | 0,010123 | 2,775558 | 1,034375 |
| max | 3,765165 | 1,901326 | 8,177714 |

*Tab. 4 Popisné statistiky – hodnoty účelových funkcí pro GP: 75% křížení, metoda SAE*

Při užití křížení *75%* populace jedinců o velikosti *S=100* a využití fitness metody *SAE*, vychází nejlepší ohodnocení při mutaci *2%* jedinců v populaci.

### 6.2.2 Funkce „Sextic“ - $x^6-2x^4+x^2$ - parametr mutace na 1, 2 a 5%, metoda SAE

(značení: Q1,Q3 – první a třetí kvartál, 25-75Pct – rozložení hodnot v rámci testů)

| | mutace 1% | mutace 2% | mutace 5% |
|---|---|---|---|
| Průměr | 14,68816 | 4,496703 | 12641,91 |
| Min | 6,83E-16 | 6,42E-16 | 6,13E-16 |
| Q1 | 7,854367 | 4,537657 | 1011,752 |
| Medián | 14,72641 | 4,600709 | 8649,257 |
| Q3 | 18,13963 | 4,640634 | 25174,42 |
| Max | 40,30563 | 4,963826 | 59536,68 |
| 25Pct | 7,854367 | 4,537657 | 1011,752 |
| 50Pct | 6,872039 | 0,063052 | 7637,505 |
| 75Pct | 3,413228 | 0,039925 | 16525,17 |
| min | 6,872039 | 0,063052 | 7637,505 |
| max | 22,166 | 0,323192 | 34362,26 |

*Tab. 5 Popisné statistiky – hodnoty účelových funkcí pro GP: 75% křížení, metoda SAE*

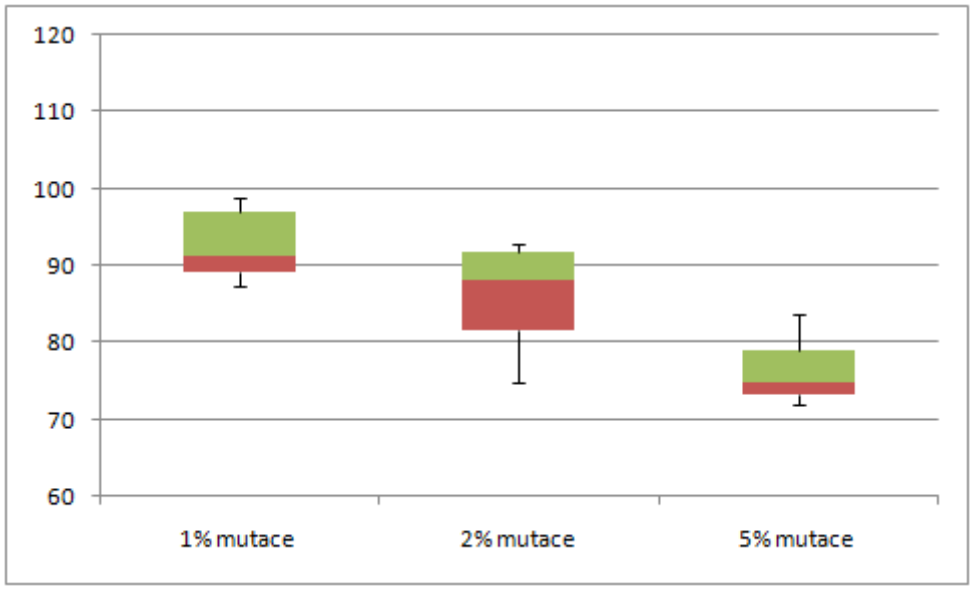
Při užití křížení *75%* populace jedinců o velikosti *S=100* a využití fitness metody *SAE*, vychází nejlepší ohodnocení při mutaci *2%* jedinců v populaci.

### 6.2.3 Funkce „Sin1“ - sin(x)+sin(2x)+sin(3x) - parametr mutace na 1, 2 a 5%, metoda SAE

(značení: Q1,Q3 – první a třetí kvartál, 25-75Pct – rozložení hodnot v rámci testů)

| | mutace 1% | mutace 2% | mutace 5% |
|---|---|---|---|
| Průměr | 88,62308 | 83,78815 | 75,35573 |
| Min | 69,30665 | 60,62291 | 62,22211 |
| Q1 | 89,24477 | 81,49854 | 73,34412 |
| Medián | 91,14553 | 88,14678 | 74,71884 |
| Q3 | 97,04527 | 91,70459 | 78,90776 |
| Max | 98,79284 | 92,87437 | 83,5492 |
| 25Pct | 89,24477 | 81,49854 | 73,34412 |
| 50Pct | 1,900764 | 6,648235 | 1,37472 |
| 75Pct | 5,899739 | 3,557811 | 4,188914 |
| min | 1,900764 | 6,648235 | 1,37472 |
| max | 1,747574 | 1,169783 | 4,641444 |

*Tab. 6 Popisné statistiky – hodnoty účelových funkcí pro GP: 75% křížení, metoda SAE*




*Obr. 16 Ilustrace hodnot popisných statistk účelových funkcí pro GP: 75% křížení, metoda SAE*

Při užití křížení *75%* populace jedinců o velikosti *S=100* a využití fitness metody *SAE*, vychází nejlepší ohodnocení při mutaci *5%* jedinců v populaci.

### 6.2.4 Funkce „Sin2“ - xsin(x/2) - parametr mutace na 1, 2 a 5%, metoda SAE

(značení: Q1,Q3 – první a třetí kvartál, 25-75Pct – rozložení hodnot v rámci testů)

| | mutace 1% | mutace 2% | mutace 5% |
|---|---|---|---|
| Průměr | 53,29372 | 15,93785 | 75,35573 |
| Min | 30,27851 | 5,894205 | 62,22211 |
| Q1 | 30,27851 | 14,89661 | 73,34412 |
| Medián | 30,27851 | 16,87019 | 74,71884 |
| Q3 | 80,64172 | 17,20034 | 78,90776 |
| Max | 113,1261 | 17,20034 | 83,5492 |
| 25Pct | 30,27851 | 14,89661 | 73,34412 |
| 50Pct | 0 | 1,973585 | 1,37472 |
| 75Pct | 50,36321 | 0,330146 | 4,188914 |
| min | 0 | 1,973585 | 1,37472 |
| max | 32,48435 | 0 | 4,641444 |

*Tab. 7 Popisné statistiky – hodnoty účelových funkcí pro GP: 75% křížení, metoda SAE*

Při užití křížení *75%* populace jedinců o velikosti *S=100* a využití fitness metody *SAE*, vychází nejlepší ohodnocení při mutaci *2%* jedinců v populaci.

### 6.2.5 Funkce „Quintic“ - $x^5-2x^3+x$ - parametr mutace na 1, 2 a 5%, metoda SSE

(značení: Q1,Q3 – první a třetí kvartál, 25-75Pct – rozložení hodnot v rámci testů)

| | mutace 1% | mutace 2% | mutace 5% |
|---|---|---|---|
| Průměr | 6,227538 | 0,931873 | 1,305225 |
| Min | 3,39E-32 | 3,31E-32 | 0,249122 |
| Q1 | 3,525975 | 0,368736 | 1,200801 |
| Medián | 4,407324 | 0,907207 | 1,335337 |
| Q3 | 8,840015 | 1,37172 | 1,578243 |
| Max | 25,51178 | 2,257329 | 1,816806 |
| 25Pct | 3,525975 | 0,368736 | 1,200801 |
| 50Pct | 0,881349 | 0,538471 | 0,134537 |
| 75Pct | 4,432691 | 0,464513 | 0,242906 |
| min | 0,881349 | 0,538471 | 0,134537 |
| max | 16,67176 | 0,885609 | 0,238563 |

*Tab. 8 Popisné statistiky – hodnoty účelových funkcí pro GP: 75% křížení, metoda SSE*

Při užití křížení na *75%* populace jedinců o velikosti *S=100* a využití fitness metody *SSE*, vychází nejlepší ohodnocení při mutaci *2%* jedinců v populaci.

### 6.2.6 Funkce „Sextic“ - $x^6-2x^4+x^2$ - parametr mutace na 1, 2 a 5%, metoda SSE

(značení: Q1,Q3 – první a třetí kvartál, 25-75Pct – rozložení hodnot v rámci testů)

| | mutace 1% | mutace 2% | mutace 5% |
|---|---|---|---|
| Průměr | 1,16365 | 9,650532 | 10,07592 |
| Min | 3,08E-32 | 4,72E-16 | 6,766239 |
| Q1 | 0,836694 | 9,454458 | 9,794685 |
| Medián | 1,235281 | 10,17969 | 10,29793 |
| Q3 | 1,352648 | 10,91997 | 10,88898 |
| Max | 3,120297 | 11,965 | 11,88124 |
| 25Pct | 0,836694 | 9,454458 | 9,794685 |
| 50Pct | 0,398587 | 0,725229 | 0,50324 |
| 75Pct | 0,117367 | 0,740279 | 0,591051 |
| min | 0,398587 | 0,725229 | 0,50324 |
| max | 1,767649 | 1,045037 | 0,992262 |

*Tab. 9 Popisné statistiky – hodnoty účelových funkcí pro GP: 75% křížení, metoda SSE*

Při užití křížení na *75%* populace jedinců o velikosti *S=100* a využití fitness metody *SSE*, vychází nejlepší ohodnocení při mutaci 1*%* jedinců v populaci.

### 6.2.7 Funkce „Sin1“ - sin(x)+sin(2x)+sin(3x) - parametr mutace na 1, 2 a 5%, metoda SSE

(značení: Q1,Q3 – první a třetí kvartál, 25-75Pct – rozložení hodnot v rámci testů)

| | mutace 1% | mutace 2% | mutace 5% |
|---|---|---|---|
| Průměr | 137,8731 | 99,28097 | 111,0312 |
| Min | 79,45287 | 46,95821 | 54,88973 |
| Q1 | 139,0664 | 92,35549 | 110,6839 |
| Medián | 143,2195 | 98,95935 | 110,6839 |
| Q3 | 159,4293 | 119,9885 | 110,6839 |
| Max | 169,8609 | 135,7814 | 175,4465 |
| 25Pct | 139,0664 | 92,35549 | 110,6839 |
| 50Pct | 4,15315 | 6,603865 | 0 |
| 75Pct | 16,20975 | 21,02917 | 0 |
| min | 4,15315 | 6,603865 | 0 |
| max | 10,43159 | 15,79284 | 64,76269 |

*Tab. 10 Popisné statistiky – hodnoty účelových funkcí pro GP: 75% křížení, metoda SSE*

Při užití křížení na *75%* populace jedinců o velikosti *S=100* a využití fitness metody *SSE*, vychází nejlepší ohodnocení při mutaci *2%* jedinců v populaci.

### 6.2.8 Funkce „Sin2“ - xsin(x/2) - parametr mutace na 1, 2 a 5%, metoda SSE

(značení: Q1,Q3 – první a třetí kvartál, 25-75Pct – rozložení hodnot v rámci testů)

| | mutace 1% | mutace 2% | mutace 5% |
|---|---|---|---|
| Průměr | 41,94932 | 3,792871 | 3,496841 |
| Min | 7,455876 | 0,520605 | 1,169907 |
| Q1 | 11,05801 | 3,758141 | 2,764505 |
| Medián | 11,05801 | 3,894516 | 4,147459 |
| Q3 | 75,19896 | 4,689463 | 4,147459 |
| Max | 160,7412 | 4,715232 | 11,4674 |
| 25Pct | 11,05801 | 3,758141 | 2,764505 |
| 50Pct | 0 | 0,136374 | 1,382954 |
| 75Pct | 64,14095 | 0,794947 | 0 |
| min | 0 | 0,136374 | 1,382954 |
| max | 85,54229 | 0,025769 | 7,319945 |

*Tab. 11 Popisné statistiky – hodnoty účelových funkcí pro GP: 75% křížení, metoda SSE*

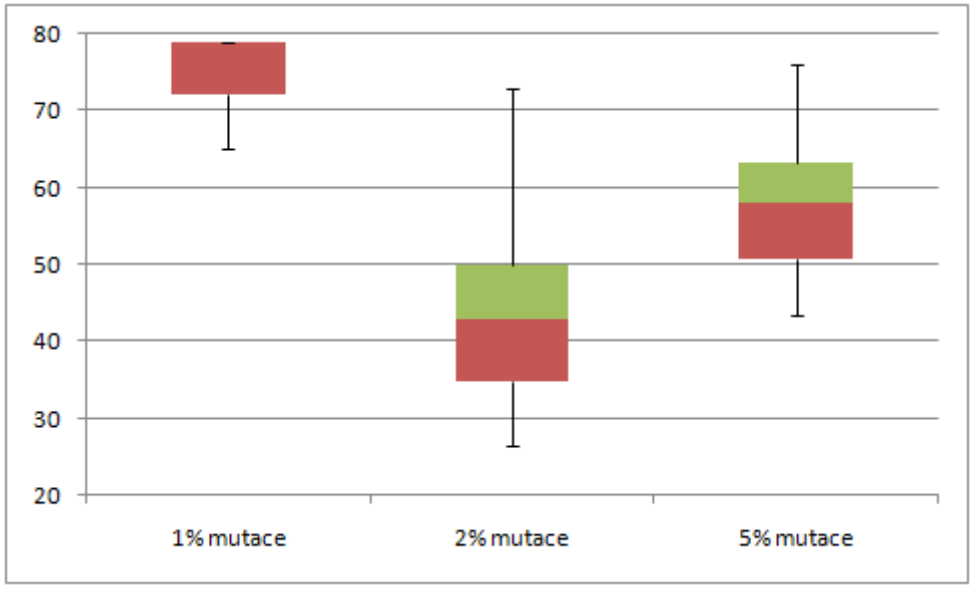
Při užití křížení na *75%* populace jedinců o velikosti *S=100* a využití fitness metody *SSE*, vychází nejlepší ohodnocení při mutaci *2%* jedinců v populaci.

### 6.2.9 Funkce „Quintic“ - $x^5-2x^3+x$ - parametr mutace na 1, 2 a 5%, metoda STE

(značení: Q1,Q3 – první a třetí kvartál, 25-75Pct – rozložení hodnot v rámci testů)

| | mutace 1% | mutace 2% | mutace 5% |
|---|---|---|---|
| Průměr | 75,49 | 40,38 | 56,33 |
| Min | 54 | 5 | 15 |
| Q1 | 72 | 34,75 | 50,75 |
| Medián | 79 | 43 | 58 |
| Q3 | 79 | 50 | 63,25 |
| Max | 79 | 73 | 76 |
| 25Pct | 72 | 34,75 | 50,75 |
| 50Pct | 7 | 8,25 | 7,25 |
| 75Pct | 0 | 7 | 5,25 |
| min | 7 | 8,25 | 7,25 |
| max | 0 | 23 | 12,75 |

*Tab. 12 Popisné statistiky – hodnoty účelových funkcí pro GP: 75% křížení, metoda STE*




*Obr. 17 Ilustrace hodnot popisných statistk účelových funkcí pro GP: 75% křížení, metoda STE*

Při užití křížení na *75%* populace jedinců o velikosti *S=100* a využití fitness metody *STE*, vychází nejlepší ohodnocení při mutaci *2%* jedinců v populaci.

### 6.2.10 Funkce „Sextic“ - $x^6-2x^4+x^2$ - parametr mutace na 1, 2 a 5%, metoda STE

(značení: Q1,Q3 – první a třetí kvartál, 25-75Pct – rozložení hodnot v rámci testů)

| | mutace 1% | mutace 2% | mutace 5% |
|---|---|---|---|
| Průměr | 76,4 | 87,5 | 89,25 |
| Min | 59 | 80 | 56 |
| Q1 | 77 | 81 | 84 |
| Medián | 78 | 88 | 86 |
| Q3 | 78 | 91 | 97 |
| Max | 78 | 100 | 100 |
| 25Pct | 1 | 7 | 2 |
| 50Pct | 0 | 9 | 3 |
| 75Pct | 1 | 7 | 2 |
| min | 1 | 7 | 2 |
| max | 0 | 9 | 3 |

*Tab. 13 Popisné statistiky – hodnoty účelových funkcí pro GP: 75% křížení, metoda STE*

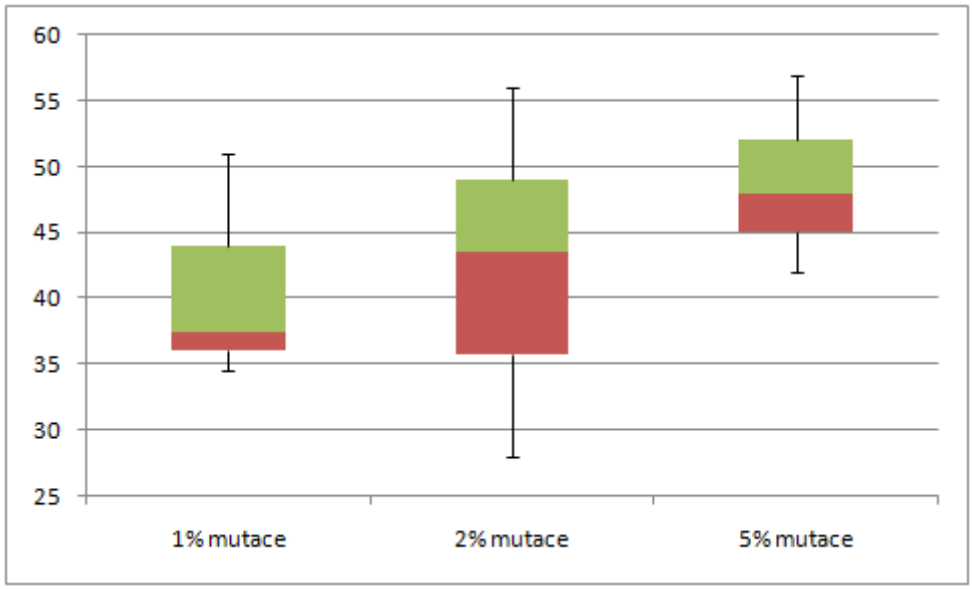
Při užití křížení na *75%* populace jedinců o velikosti *S=100* a využití fitness metody *STE*, vychází nejlepší ohodnocení při mutaci *1%* jedinců v populaci.

### 6.2.11 Funkce „Sin1“ - sin(x)+sin(2x)+sin(3x) - parametr mutace na 1, 2 a 5%, metoda STE

(značení: Q1,Q3 – první a třetí kvartál, 25-75Pct – rozložení hodnot v rámci testů)

| | mutace 1% | mutace 2% | mutace 5% |
|---|---|---|---|
| Průměr | 39,28 | 42,19 | 47,67 |
| Min | 32 | 22 | 32 |
| Q1 | 36 | 35,75 | 45 |
| Medián | 37,5 | 43,5 | 48 |
| Q3 | 44 | 49 | 52 |
| Max | 51 | 56 | 57 |
| 25Pct | 36 | 35,75 | 45 |
| 50Pct | 1,5 | 7,75 | 3 |
| 75Pct | 6,5 | 5,5 | 4 |
| min | 1,5 | 7,75 | 3 |
| max | 7 | 7 | 5 |

*Tab. 14 Popisné statistiky – hodnoty účelových funkcí pro GP: 75% křížení, metoda STE*




*Obr. 18 Ilustrace hodnot popisných statistk účelových funkcí pro GP: 75% křížení, metoda STE*

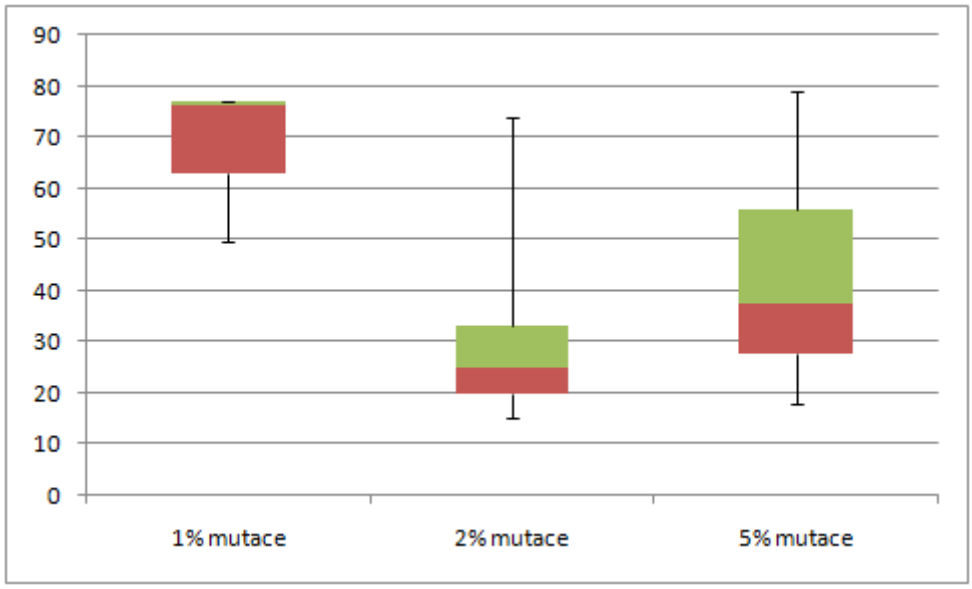
Při užití křížení na *75%* populace jedinců o velikosti *S=100* a využití fitness metody *STE*, vychází nejlepší ohodnocení při mutaci *2%* jedinců v populaci.

### 6.2.12 Funkce „Sin2“ - xsin(x/2) - parametr mutace na 1, 2 a 5%, metoda STE

(značení: Q1,Q3 – první a třetí kvartál, 25-75Pct – rozložení hodnot v rámci testů)

| | mutace 1% | mutace 2% | mutace 5% |
|---|---|---|---|
| Průměr | 67,96 | 29,48 | 40,81 |
| Min | 42 | 9 | 16 |
| Q1 | 63 | 20 | 27,75 |
| Medián | 76,5 | 25 | 37,5 |
| Q3 | 77 | 33,25 | 56 |
| Max | 77 | 74 | 79 |
| 25Pct | 63 | 20 | 27,75 |
| 50Pct | 13,5 | 5 | 9,75 |
| 75Pct | 0,5 | 8,25 | 18,5 |
| min | 13,5 | 5 | 9,75 |
| max | 0 | 40,75 | 23 |

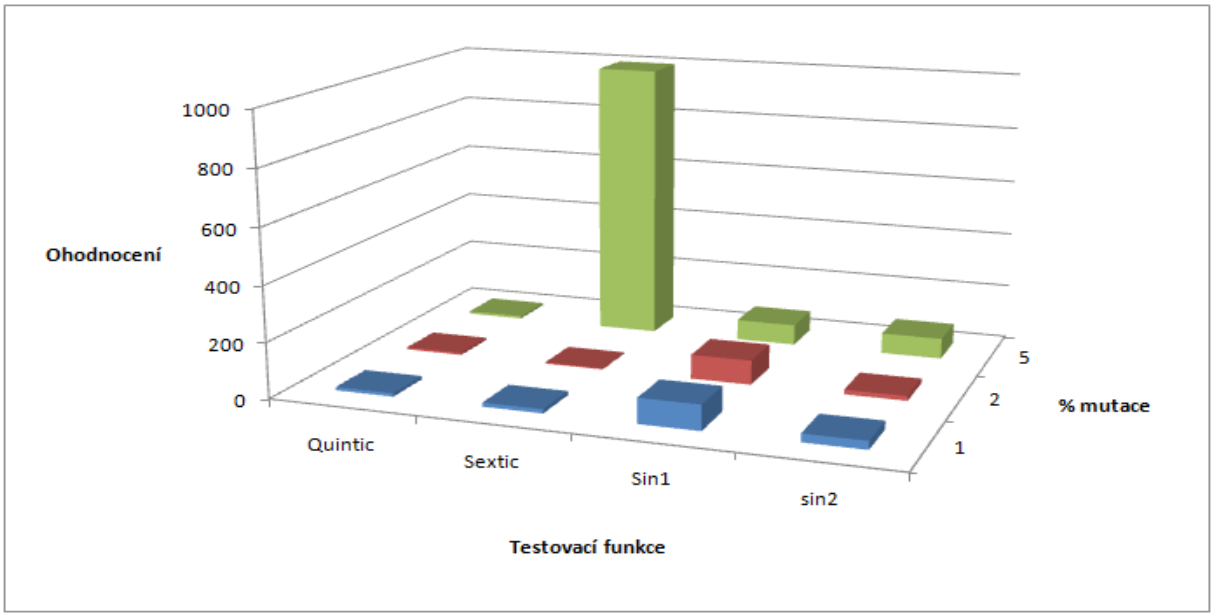
*Tab. 15 Popisné statistiky – hodnoty účelových funkcí pro GP: 75% křížení, metoda STE*




*Obr. 19 Ilustrace hodnot popisných statistk účelových funkcí pro GP: 75% křížení, metoda STE*

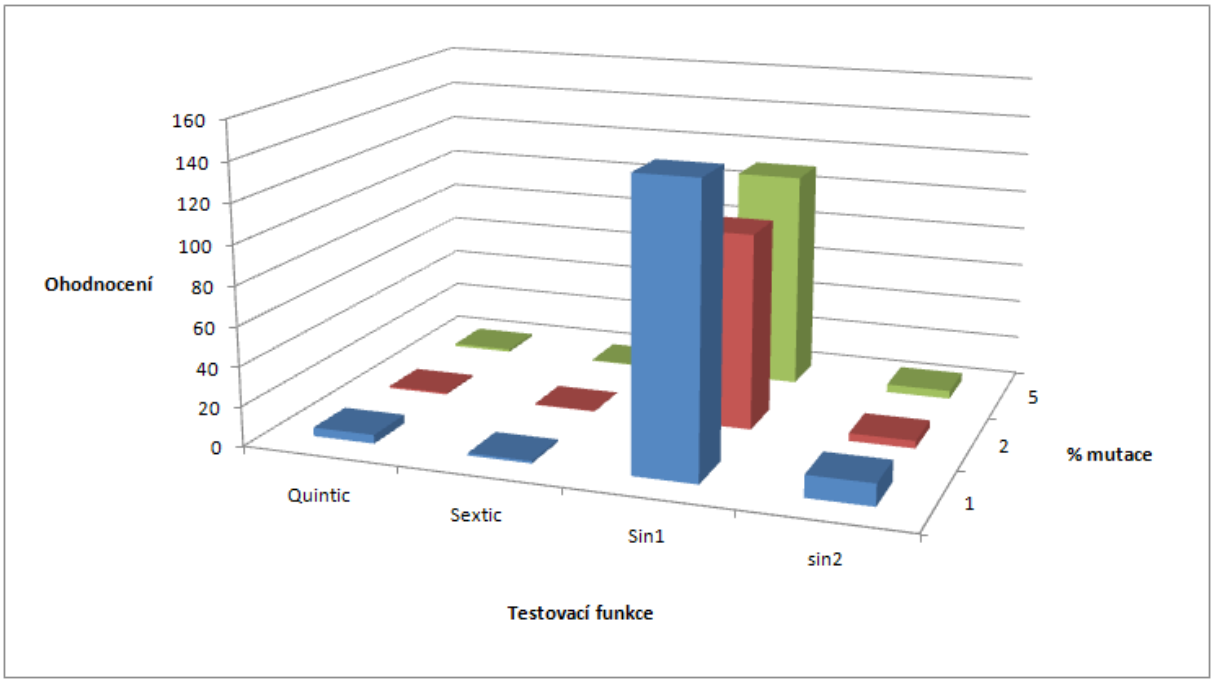
Při užití křížení na *75%* populace jedinců o velikosti *S=50* a využití fitness metody *SSE*, vychází nejlepší ohodnocení při mutaci *2%* jedinců v populaci.

## 6.3 Srovnání parametrizace GP dle hodnotící funkce (fitness)



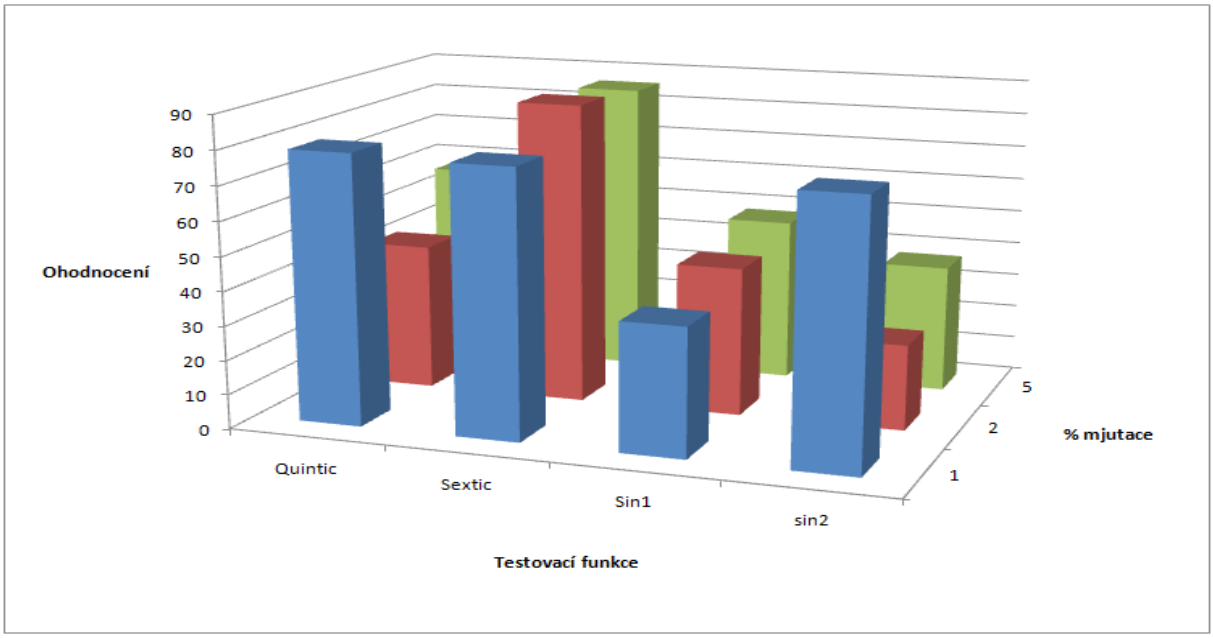

*Obr. 20 Srovnání testování parametrů SAE*

Metodou ohodnocení SAE (fitness) byla v rámic testů nejlépe ohodnocena funkce „sextic“. tj. $y(x)=x^6-2x^4+x^2$ při mutaci 2% populace. Osa ohodnocení uvedněná na obr. 43 není přesná, hodnota SAE (sumy absolutních odchylek) je přibližně 6x vyšší.




*Obr. 21 Srovnání testování parametrů SSE*

Metodou ohodnocení SSE (fitness) byla v rámci testů nejlépe ohodnocena funkce „sextic“. tj. $y(x)=x^6-2x^4+x^2$ při mutaci 2% populace, stejně jako v případě SAE metody fitnes.




*Obr. 22 Srovnání testování parametrů STE*

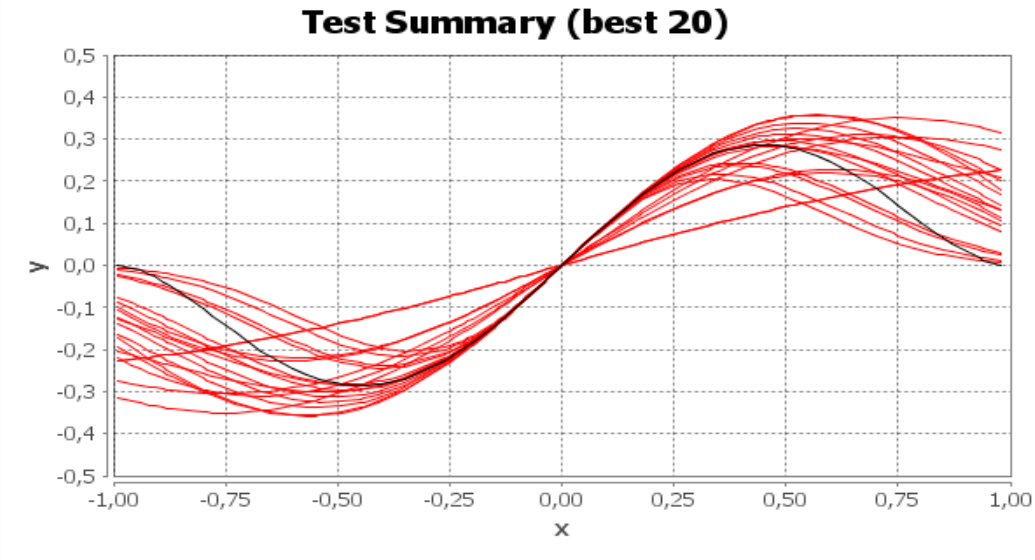
V rámci *„Sum Epsilon-tube Error“* hodnota spočtená STE vyjadřuje existenci bodů generované křívky v tolerančím pásmu. Vyšší ohodnocení je tedy příznivější, na rozdíl od předchozích dvou metod. Nejlépe ohodnocená je i v tomto srovnání stejná funkce a parametry, tedy.„sextic“- $y(x)=x^6-2x^4+x^2$ při mutaci 2% populace

## 6.4 Ukázka výsledků jednotlivých testů – metody SAE a SSE

Tato podkapitola obsahuje kompletní výsledky testů vybraných funkcí, z testovací množiny , které přesvědčivě ukazují funkcionalitu programové aplikace a schopnost GP, řešit problematiku symoblické regrese, resp. automatického generování modelů.

Implementovaná aplikace byla navržena, pro co možná nejvyšší trasparentnost získaného řešení. Výsledná řešení jsou reprezentována tabulkou obsahující jednotlivá řešení každého testu ( v rámci předchozích zkoušek aplikace jsem za test považovali celkem 100 hledání řešení, tj. funkční předpis hledané funkce dle zadaných souřadnic). Interpretace námi hledaného řešení je zakomponována přímo v grafickém uživatelském prostředí – tabulka pro uchování dosažených řešení, která obsahuje hodnotu použité fitness funkce, funkční předpis funkce generované a generaci ve, které bylo řešení nalezeno.

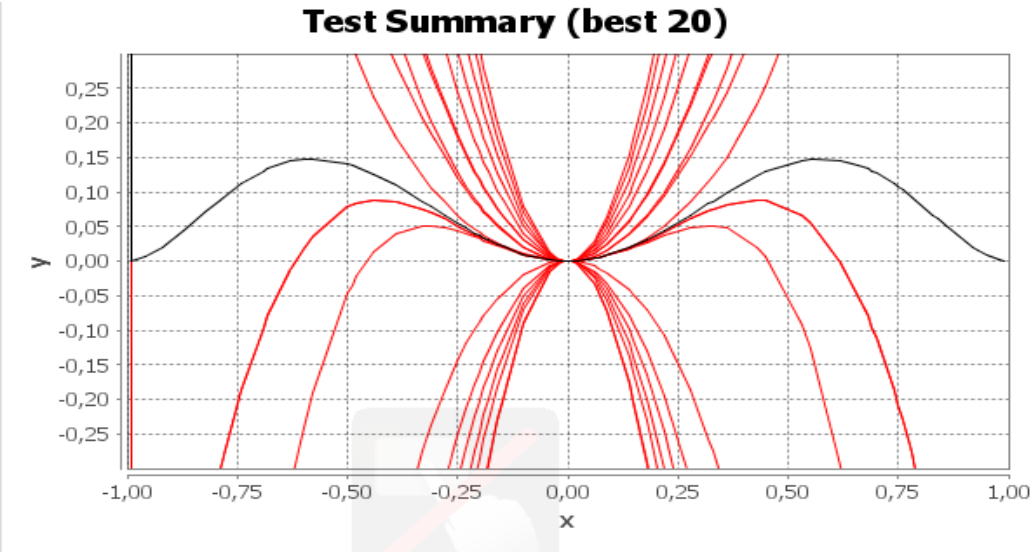
V případě 20 a více testů je vygenerován souhrný graf, který obsahuje ilustraci hledané funkce (označena *černou barvou*) a 20 nejlepších nalezených řešení (označeny *červenou barvou*).




*Obr. 23 Funkce „Quintic" - $x^5-2x^3+x$ , při křížení 75% a mutaci 2% populace, metoda SAE*

Nejlepší řešení :

| ((1*(x^5.0))-((2.0*(x^3.0))-(1.0*(x^1.0)))) | (21) |
|---|---|

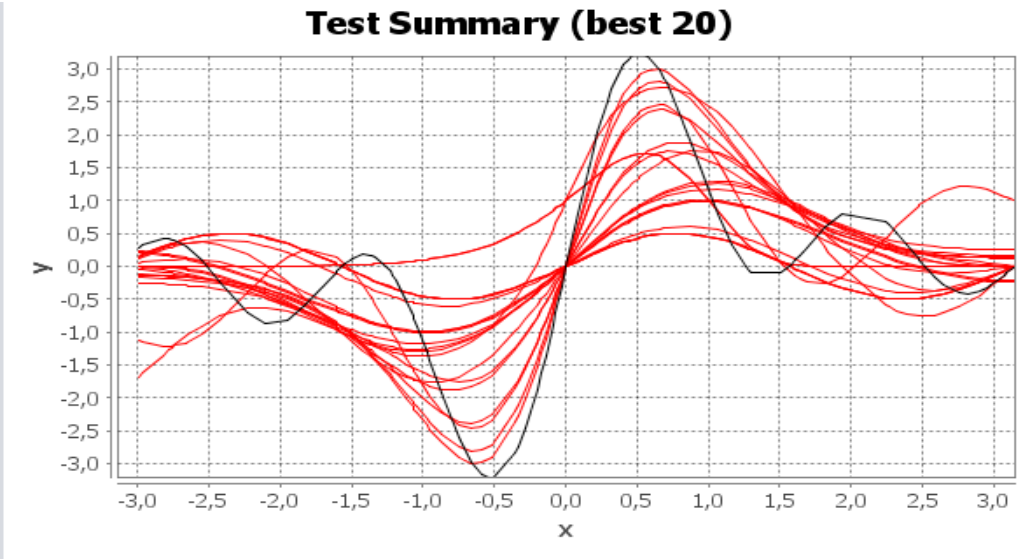



*Obr. 24 Funkce „Sextic" - $x^6-2x^4+x^2$ , při křížení 75% a mutaci 5% populace, metoda SAE*

Nejlepší řešení :

| ((1*(x^6.0))-((2.0*(x^4.0))+(7.0*(x^2.0)))) |
|---|

(22)

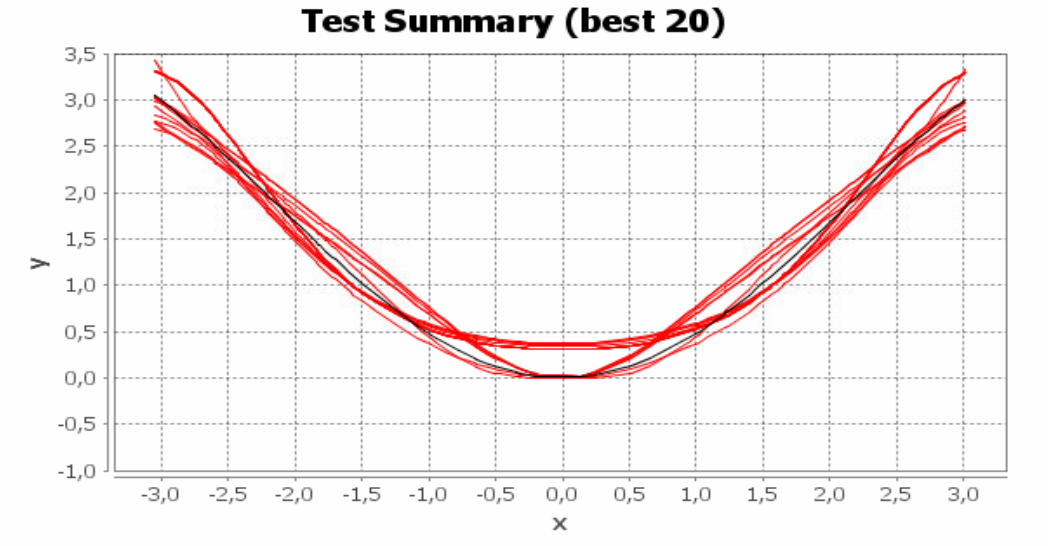



*Obr. 25 Funkce „Sin1" - sin (x) + sin (2x) + sin(3x) + sin(4x): křížení 75%, mutace 2% z populace, metoda SAE*

Nejlepší řešení :

| ((tghx)*(9^(cosx))) |
|---|

(23)




*Obr. 26 Funkce „Sin2" - xsin (x) křížení 75%, a mutaci 2% z populace, metoda SSE*

Nejlepší řešení :

| ((x*((tghx)^3)) |
|---|

(24)

# 7 ZÁVĚR

Cílem této práce bylo vytvoření aplikace, využitelné v úlohách automatického generování modelů resp. symbolické regrese. V impementaci byly aplikovány pokročilé operátory GP (nedestruktivní mutace a křížení, elitního přístupu, spec. formy turnajové selekce a redukce jedinců).

Práce obsahuje stučný popis genetického programování (GP), základní charakteristiky evolučních výpočtetních technik (EVT) a symoblické regrese.

Aplikace je implementována v jazyce Java, podporuje více-vláknový režim pro rychlejší výpočet. Grafické uživatelské prostředí je doplněno tabulkou pro výsledky a grafem nalezeného řešení, příp. souhrným grafem, při počtu testů větších než 19. Řešič GP je vysoce parametrizovatelný, tedy může být využit pro širší oblast problémů. Vzhledem k nezbytnosti přesného nastavení GP algoritmů obecně (řešič této aplikace není vyjímkou) je přesné nastavení získat vyšším počtem kontrolních měření. Možných kombinací je vzhledem k úrovni parametrizace aplikace nespočet a nebylo tedy možné všechny otestovat.

Testování funkcionality aplikace (kap. 6) resp. řešiče GP spočívá v protestování množiny testovacích funkcí v rámci zvolených parametrů řešiče. Byly testovány kombinace nastavení parametrů – v tomto případě převážně procentuální parametry, udávající použití mutace a křížení v rámci reprodukce. V rámci statistického rozboru byl počet testování jednotlivých možností nastavení parametru roven stu., s daným maximálním počtem generací populace (přip. cyklů programu),

Cíle této práce byly splněny. Aplikace prokázala svoji funkčnost a schopnost nalézt řešení hledaného problému. . Domnívám se, že případné pokračování vývoje, příp. testování aplikace, dále přispěje k možnostem řešení problematiky generování modelů, konkrétně symbolické regrese.

# SEZNAM POUŽITÉ LITERATURY


[1] Hasík, Karel, Numerické metody.pdf[online], [cit. 11. 5. 2013], Dostupné z :http://www.slu.cz/math/cz/knihovna/ucebni-texty/Numericke-metody/Numericke-metody.pdf

[2] Wiese, Tomas, Global Optimization Algorithms.pdf[online] – Theory and Application – 2nd edition, 26.6.2009,[cit. 12. 5. 2013], Dostupné z:http://www.it-weise.de/projects/book.pdf

[3] Wikipedia – Swarm Inteligence http://en.wikipedia.org/wiki/Swarm_intelligence#Ant_colony_optimization

[4] Wikipedia – Ant colony optimization http://en.wikipedia.org/wiki/Ant_colony_optimization

[5] John R. Koza: Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 1992, 819p., ISBN: 0-262-11170-5

[6] KOZA, John R.,Hierarchical Automatic Function Definition in Genetic Programming.pdf [online], 1992[cit. 15. 5. 2013],, Dostupné z: http://www.cs.bham.ac.uk/~wbl/biblio/cache/http___www.genetic-programming.com_jkpdf_foga1992.pdf

[7] Genetic Programming Bibliography entries for John Koza http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/JohnKoza.html

[8] Koza, John R.,Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems.pdf[online], June 1990,[cit. 16. 5. 2013], Dostupné z: ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/90/1314/CS-TR-90-1314.pdf

[9] Matoušek, Radomil, Grammatical Evolution: STE criterion in Symbolic Regression Task.pdf [online],October 20-22, 2009, San Francisco, USA, [cit. 18. 5. 2013], Dostupné z: http://www.iaeng.org/publication/WCECS2009/WCECS2009_pp1050-1054.pdf

[10] XYLine Chart example (JFreeChart) http://www.roseindia.net/chartgraphs/xyline-chart.shtml

[11] JfreeChart Forum-General http://www.jfree.org/phpBB2/viewforum.php?f=3

[12] MAROŠ, Bohumil, Marošová, Marie: Numerické metody I, Akademické nakladatelství CERM, s. r. o., Brno, 2003, 144s, ISBN 80-214-2388-9