SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br.349

SIMBOLIČKO DERIVIRANJE UZ POMOĆ GENETSKOG PROGRAMIRANJA

Luka Donđivić

Zagreb, lipanj, 2008.

Sadržaj

Uvod

Metoda genetskog programiranja teorijski se počela razvijati pedesetih godina dvadesetog stoljeća, paralelno sa ostalim evolucijskim metodama, ali je zbog velikih zahtjeva za procesorskim vremenom za svoju pravu primjenu morala čekati sve do devedesetih godina. Pionir u istraživanju i razvoju te metode John R. Koza ju je uspješno primjenio na niz kompleksnih optimizacijskih i pretraživačkih problema. Do danas genetsko programiranje je dalo izvanredne rezultate na raznim područjima (elektronički dizajn, igranje igara, sortiranje, pretraživanje...)

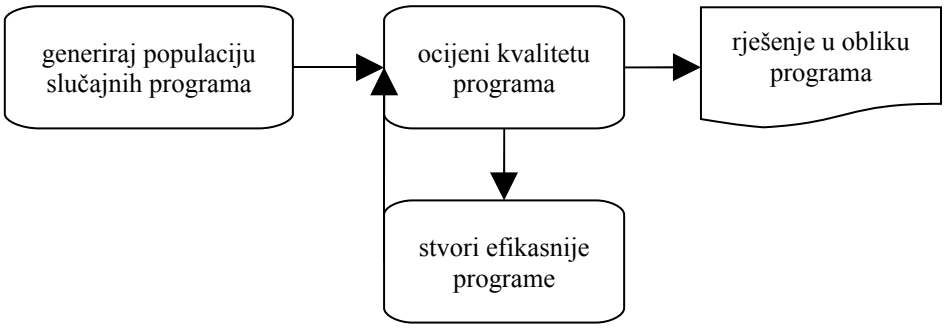
Razvoj teorije genetskog programiranja bio je vrlo težak i spor. Tek nakon 2000. godine došlo je do niza značajnijih koraka, tako da danas imamo uspješno izgrađen egzaktan probabilistički model GP-a.

Problem simboličke derivacije svodi se na poznati problem identifikacije funkcije iz zadanog uzorka, koji Koza obrađuje u svojoj knjizi „Genetic Programming: On the Programming of Computers by Means of Natural Selection“.

Genetsko programiranje

Genetsko programiranje je automatiziran optimizacijski postupak razvoja računalnih programa, čija je namjena rješavanje većinom složenih problema iz područja računarstva, ali i problema na koje svakodnevno nailazimo. Koncept je zasnovan na općim idejama proizašlima iz teorije genetskih algoritama (engl. *genetic algorithms*), kao i drugih evolucijskih metoda. Najjednostavnije rečeno, konačni cilj GP-a (kao produkt) je univerzalni računalni program koji nalazi rješenja problema kao ulaznih podataka.

Pojednostavljeni shematski prikaz genetskog programiranja dan je na Slici 2.1.



Slika 2.1. *Pojednostavljeni shematski prikaz genetskog programiranja*

1.1 GP kao poseban slučaj evolucijskog algoritma

Iterativni postupak na kojem se bazira genetsko programiranje, u sebi sadrži jasno vidljive konture općenitog pseudokoda evolucijskih algoritama, a to su:

1) generacija inicijalne populacije rješenja,
2) analiza svake jedinke populacije i populacije općenito, te
3) odabir genetskog operatora i provedba odgovarajućih akcija nad pojedinom jedinkom (jedinkama).

Inicijaliziraj populaciju
Evaluiraj inicijalnu populaciju
Radi
1. primijeni operacije odgovarajuće genetskim operatorima i izgeneriraj nova rješenja
2. evaluiraj rješenja u populaciji
3. selektiraj

Dok nije zadovoljen kriterij konvergencije

Slika 2.2. *Pseudokod evolucijskih algoritama*

1.2 Populacija

Biološka definicija populacije opisuje ju skupom jedinki iste vrste koje egzistiraju u istom prostoru. Razmnožavanjem unutar populacije (stvaranjem potomaka), ali i umiranjem, veličina populacije je konstantna kroz generativni proces. Što se tiče genetskog programiranja, jedinka predstavlja računalni program, a prostor egzistencije jedinki iz iste populacije je jedna iteracija u optimizacijskom algoritmu. Prema tome, govori se o populaciji računalnih programa jedne iteracije algoritma.

1.3 Funkcija dobrote

Poznato je da se među biološkom vrstom na određenom staništu neke jedinke više ističu, imaju duži životni vijek, veće mogućnosti za stvaranje potomstva i sl. Kao i prije, povlači se analogija sa GP-om. Naime, u cilju rješavanja problema evolucijskim algoritmom, jako je bitno da bolja rješenja ostaju u daljnjem razmatranju, a ona lošija da se odbace. Upravo dobro definirana funkcija dobrote obavlja tu ulogu. Na temelju raznih parametara, ona određuje dobrotu (engl. *fitness*) pojedine jedinke, o kojoj kasnije uvelike ovisi sudbina iste.

Definiranje funkcije dobrote je jedan od ključnih problema genetskog programiranja, jer je potrebno da bude što "bolja", a što jednostavnija; pošto je njeno evaluiranje prisutno u svakom trenutku generativnog procesa. U stvarnosti, odabir i definiranje funkcije dobrote se prilagođava samom problemu i njegovim karakteristikama, a zadovoljenost navedenih oprečnih zahtjeva se pokušava uravnotežiti.

1.4 Genetski operatori

Evolucijski aspekt genetskog programiranja se očituje u načinu optimiranja promatranog programa, gdje su, u ulogama genetskih operatora, prisutne metode reprodukcije i križanja (engl. *crossover*). U nekim slučajevima se javljaju i mutacija, permutiranje i sl.

1.4.1 Reprodukcija

Reprodukcija je jednostavno kopiranje odabranog čvora i njegovo umetanje u novu populaciju.

Parametri:

p_r – vjerojatnost odabira reprodukcije kao genetskog operatora,
p_n – vjerojatnost odabira pojedinog čvora.

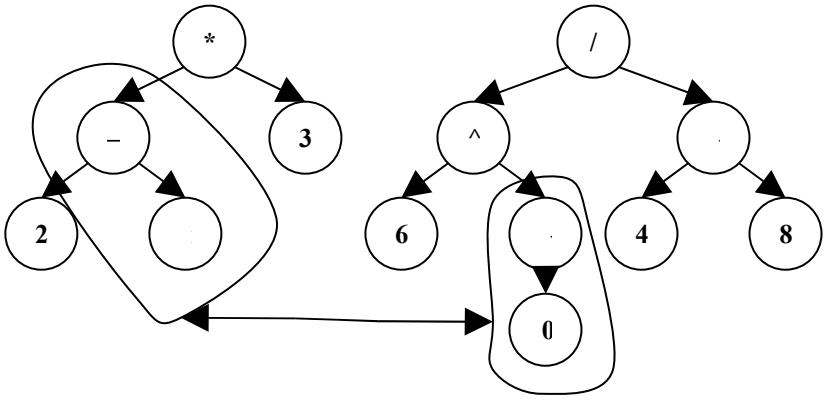
1.4.2 Križanje

Križanje je analogno biološkoj spolnoj reprodukciji. Naime, u tom postupku dolazi do zamjene podstabala odabrana dva čvora. Najučestaliji oblik je obavljanje opisane radnje na dva slučajno odabrana čvora.

Parametri:

p_c – vjerojatnost odabira križanja kao genetskog operatora,

p_n – vjerojatnost odabira pojedinog čvora.



Slika 2.3. *Križanje (crossover)*

1.5 Selekcija

Konačno, tijekom generiranja populacija odvija se selekcija, i to na temelju vjerojatnosti odabira genetskog operatora, te dobrote jedinki. Uobičajeno je da selekcija direktno (proporcionalno) ovisi o dobroti.

1.6 Kriterij prekidanja evolucijskog procesa

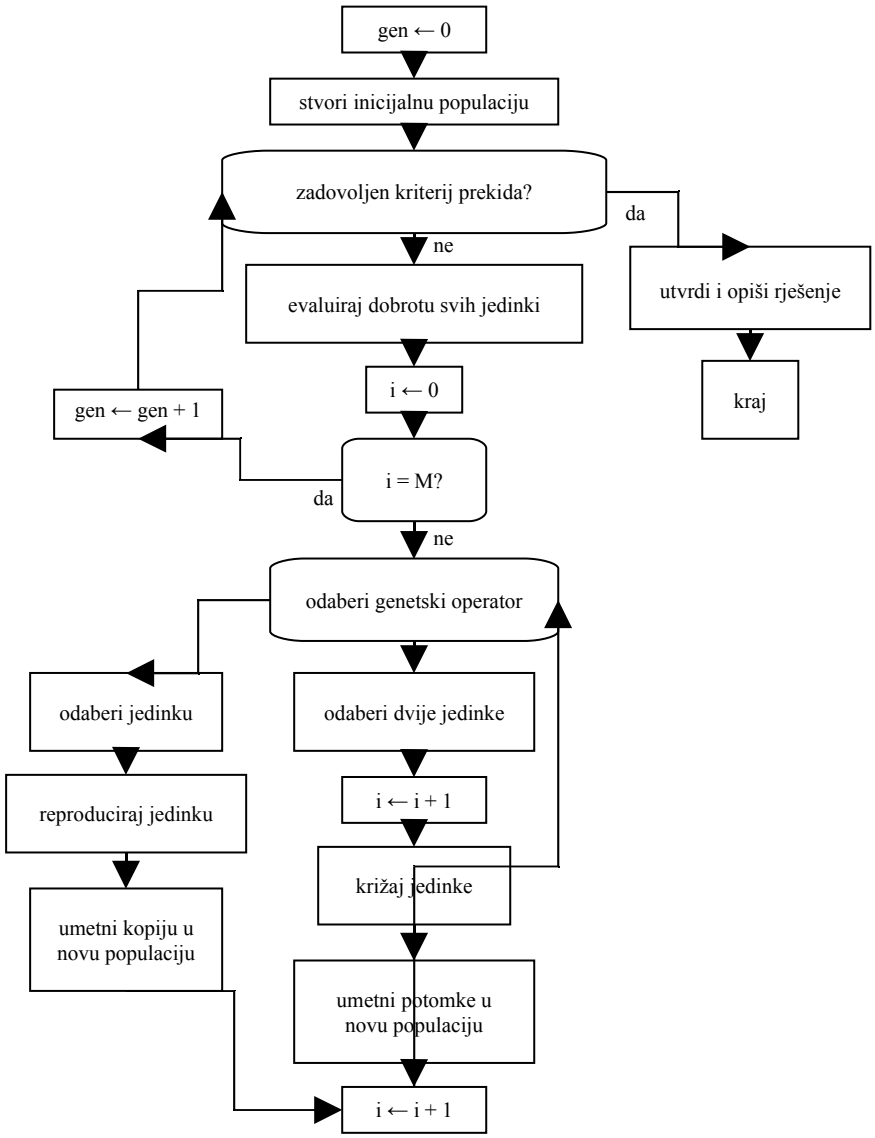
S obzirom na činjenicu da koncept genetskog programiranja s dobro definiranom funkcijom dobrote nad određenim problemom, i adekvatnim skupovima F i T, zadovoljavajuće brzo dolazi do rješenja problema, pa se u velikom broju slučajeva generativni postupak prekida pri ispunjenu određenog logičkog predikata. Dodatno se uzima i konstanta G, kao supremum broja iteracija, odnosno broja generiranih populacija. Također, zadaje se i supremum M broja jedinki u populaciji, te supremum D dubine generiranih stabala.

1.7 Utvrđivanje i opisivanje rješenja

Jednom kada se ispuni kriterij prekida generativnog procesa, iz populacije rješenja se odabire najkvalitetnije, te se opisuje, ako je ikako moguće. Moguće su i poželjne daljnje analize istog, kao i ponavljanje cijelog postupka zbog nezadovoljstva dobivenim rješenjem. Bolje rješenje se uglavnom pokušava naći redefiniranjem funkcije dobrote ili povećanjem supremuma broja iteracija.

1.8 Blok-dijagram GP-a

Nakon svega, genetsko programiranje se može predočiti sljedećim blok-dijagramom:



Slika 2.4. *Blok-dijagram genetskog programiranja*

Programsko ostvarenje

Poglavlje opisuje program koji implementira algoritam genetskog programiranja. Program je napisan u programskom jeziku C++.

1.9 Ulaz i izlaz

Program kao svoj ulaz prima:

1) Funkciju koju treba derivirati (u simboličkom obliku ili njen uzorak)
2) Granice intervala na kojem se vrši uzorkovanje funkcije
3) Parametre programa (opcionalno)

Ulazni podaci primaju se kroz datoteku „ulaz“ koja se treba nalaziti u tekućem direktoriju. U prvom retku trebaju se nalaziti granice intervala na kojem tražimo funkciju derivacije. Kao što je zadatkom određeno, funkcija se može zadati simbolički ili uzorkom točaka, pa tako nakon granica intervala može slijediti niz znakova: funkcija u simboličkom obliku ili uzorak točaka: više redaka sa parovima realnih vrijednosti (program sam prepoznaje o kojem se slučaju radi).

Parametri programa postavljaju se u datoteci „parametri“ koja se također treba nalaziti u tekućem direktoriju. Osim parametara, može se zadati skup dozvoljenih operatora i funkcija.

Ako neki parametar (ili skup operatora) nije postavljen, on poprima standardnu vrijednost (definiranu u poglavlju 4.)

Za parsiranje ulazne funkcije, te za evaluaciju funkcija u točkama koristi se besplatna vanjska biblioteka FunctionParser, autora Juhe Nieminena.

Izlaz programa:

1) Funkcija – aproksimacija derivacije zadane funkcije
2) Podaci o kvaliteti aproksimacije: apsolutna i relativna prosječna pogreška
3) Tekstualna datoteka “zaGnuplot” – naredbe programu Gnuplot za grafički prikaz svih međurješenja i konačnog rješenja

Program tijekom izvođenja na standardni izlaz ispisuje podatke o trenutno najboljoj pronađenoj aproksimaciji, te po završetku ispisuje najbolju jedinku koja se pojavila u evolucijskom procesu.

Osim toga, stvara u tekućem direktoriju datoteku “zaGnuplot” u koju zapisuje niz naredbi za iscrtavanje programu Gnuplot. Pokretanjem programa Gnuplot kojem kao parameter iz komandne linije dajemo ime datoteke “zaGnuplot” možemo grafički pratiti poboljšanje rješenja kroz process evolucije, te konačno možemo vidjeti i koliko dobro nađena funkcija aproksimira uzorak.

1.10 Svođenje problema pronalaska derivacije na problem identifikacije funkcije

Ukoliko je funcija zadana uzorkom točaka, potrebno je na početku pronaći njen simbolički oblik.

U slijedećem koraku zadani interval, na kojem tražimo funkciju derivacije, se podijeli na NTOC jednakih intervala (gdje je NTOC parametar programa), te se za sredinu svakog intervala izračuna aproksimativna vrijednost funkcije derivacije: $d_i=(y_{i+1} – y_i) / l$, gdje je y_i vrijednost originalne funkcije na početku intervala, y_{i+1} vrijednost originalne funkcije na kraju intervala, a l duljuna intervala. Aproksimacija je očito bolja što je vrijednost parametra NTOC veća, međutim linearno s parametrom NTOC raste i vrijeme izvršavanja programa.

Nakon što smo dobili uzorak točaka tražene funkcije, možemo pristupiti njenoj identifikaciji.

1.11 Implementacija genetskog programiranja u rješavanju problema identifikacije funkcije

1.11.1 Jedinka

Jedna jedinka predstavlja jednu funkciju. Funkcija je predstavljena stablom u kojem svaki unutrašnji čvor predstavlja jednu od dozvoljenih funkcija (sin, cos, exp, ln) ili matematičku operaciju (+, -, /, *, ^), te je broj čvorova djece jednak broju operanada funkcije/operacije. Vanjski čvorovi mogu biti ili parametar funkcije **x**, ili neka pseudo slučajno generirana realna konstanta.

Jedinka je realizirana kao struktura koja sadrži pokazivač na korijenski čvor stabla funkcije i realan broj – kaznu jedinke.

Čvor stabla je opisan kao struktura koja sadrži pokazivač na podatke o čvoru, te vektor pokazivača na čvorove djecu.

1.11.2 Funkcija kazne

Funkcija dobrote, ili u ovom slučaju funkcija kazne, uz genetske operatore ima presudan utjecaj na performanse programa. Kod problema identifikacije funkcije se pokazalo da je suma kvadrata razlika vrijednosti zadane funkcije i promatrane jedinke u točkama uzorka dobra mjera kazne.

$$Funkcija\ kazne = \sum_{i=1}^{NTOC} (f(x_i) - g(x_i))^2,$$

gdje su x_i i $f(x_i)$ parovi vrijednosti zadanog uzorka, a g(x) funkcija za koju računamo kaznu.

Problemi se javljaju kod funkcija koje se ne mogu evaluirati u nekim točkama uzorka. Za proces evolucije nije dobro takve funkcije odbaciti, nego se kazni doda fiksan kazneni iznos za svaku točku u kojoj funkcija nije definirana. Također, ako kvadrat razlike vrijednosti premašuje kazneni iznos, kazna se uvećava samo za kazneni iznos. Kazneni iznos je parametar programa (PENAL), te je bitno da ne bude ni premali ni preveliki. Dakle funkcija kazne se modificira ovako:

$$Funkcija\ kazne = \sum_{i=1}^{NTOC} pribrojnik_i,$$

$$pribrojnik_i = \begin{cases} (f(x_i) - g(x_i))^2, & za\ x_i\ u\ domeni\ od\ g\ i\ (f(x_i) - g(x_i))^2 < PENAL \\ PENAL, & inače \end{cases}$$

1.11.3 Populacija

Populacija je ostvarena kao vektor od NJED jedinki, gdje je NJED parametar programa. Na početku evolucije, jedinke populacije se generiraju slučajno, metodom „ramped half and half“: pola jedinki je generirano grow metodom (čvor na dubini manjoj od maksimalne se bira slučajno iz unije skupa konstanti i skupa operatora), a pola full metodom (čvor na dubini manjoj od maksimalne se bira slučajno samo iz skupa operatora).

Čvorovi na maksimalnoj dubini u obje metode se slučajno biraju iz skupa konstanti (član tog skupa je i varijabla x).

Maksimalna dubina stabla određena je parametrom MAXD. Njegovim porastom se povećava prostor rješenja koji se pretražuje, ali se i eksponencijalno povećava vrijeme izvršavanja programa.

Ostali parametri koji utječu na generiranje slučajne jedinke su:

PKONSTX – vjerojatnost da će se iz skupa konstanti odabrati varijabla x.

MAXKONST – maksimalna vrijednost slučajno generirane konstante

MINKONST – minimalna vrijednost slučajno generirane konstante

POPERATOR – u grow metodi, vjerojatnost da će se čvor odabrati iz skupa operatora

1.11.4 Selekcija

Program implementira tri turnirsku selekciju. Dok ne nastupi uvjet za prestanak evolucijskog procesa ponavlja se slijedeći postupak:

1) Slučajno se odaberu se tri jedinke
2) Najlošija jedinka (ona s najvećom vrijednosti funkcije kazne) se briše iz populacije
3) Križanjem preostalih dviju jedinki nastaje nova jedinka, koja se dodaje u populaciju

Tri turnirska selekcija ima inherentno ugrađen elitizam – dvije najbolje jedinke populacije uvijek ostaju sačuvane.

1.11.5 Križanje

Operator križanja je standardni za jedinke prikazane stablom: slučajno odabrani čvor i njegovo podstablo se obrišu iz jedinke majke, te se zamjene slučajno odabranim čvorom i njegovim podstablom iz jedinke oca. (2.4.2)

Pri tome je potrebno paziti da dubina stabla novonastale jedinke ne prijeđe maksimalnu dubinu MAXD.

1.11.6 Mutacija

Kako bi se smanjila vjerojatnost preuranjene konvergencije populacije prema lokalnom minimumu uveden je operator mutacije. Njegova uloga je uvođenje novog genetskog materijala tijekom cijelog evolucijskog procesa. Mutacija se primjenjuje samo na jedinke nastale križanjem, s vjerojatnošću pojavljivanja određenom parametrom PMUTACIJE. (detaljnije u poglavlju 4.3)

Operator mutacije izveden je tako da se slučajno odabrani čvor i njegovo podstablo zamijene slučajno generiranim podstablom. Slučajno podstablo se kao i slučajne jedinke generira „ramped half and half“ metodom.

1.11.7 Uvjet za prekid evolucijskog procesa

Program u jednom pokretanju pokreće evoluciju NRUNS puta, gdje je NRUNS parametar programa.

Svako pokretanje se završava na jedan od tri načina:

1) Ako se u određenom vremenu (parametar MAXTIME1) kazna najbolje jedinke ne popravi za barem neki postotak (parametar POBOLJSANJE)
2) Ako kazna najbolje jedinke dosegne zanemarivo malu vrijednost – pronađeno je najbolje moguće rješenje
3) Ako korisnik pritiskom na tipke Ctrl+Break pošalje signal SIGINT (maskiran u programu)
4) Ako je isteklo maksimalno vrijeme za izvršavanje jednog pokretanja (određeno parametrom MAXTIME u minutama)

Eksperimentalni rezultati

Cilj ovog poglavlja je empirijski odrediti utjecaj pojedinih parametara na performanse programa, te prema rezultatima odabrati njihove optimalne vrijednosti.

Sva mjerenja su obavljena na prijenosnom računalu s Intel Pentium procesorom frekvencije takta 1.4 GHz.

Program se testira sa osam različitih ispitnih funkcija, tablica 4.1.

Tablica 4.1 Ispitne funkcije

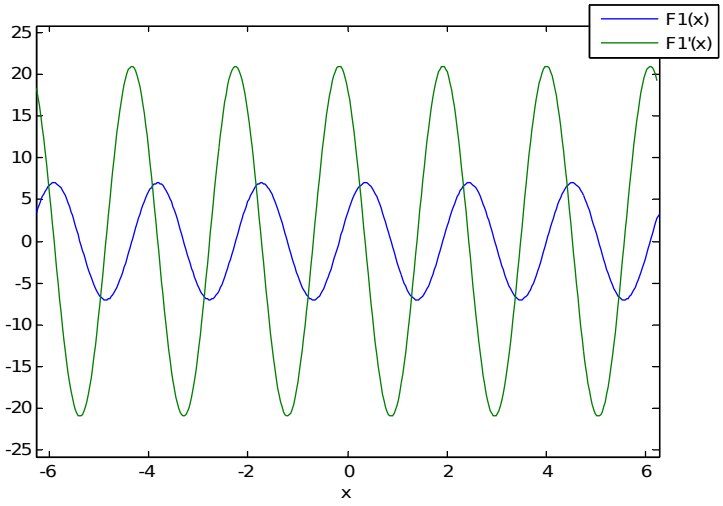
Ime	Funkcija	Interval	Derivacija
F1	7*sin(3*x+0.5)	[-6.28,6.28]	21*cos(3*x+0.5)
F2	x^3-4*x^2+9*x-3	[-6.28,6.28]	3*x^2-8*x+9
F3	x^4-2*x^3-x^2+2*x	[-6.28,6.28]	4*x^3-6x^2-2*x+2
F4	sin(x)*cos(x)	[-6.28,6.28]	2*cos(x)^2-1
F5	sin(x^2)	[-6.28,6.28]	2*cos(x^2)*x
F6	x^2*sin(x)*cos(x)	[-6.28,6.28]	x*(2*cos(x)^2*x+2*sin(x)*cos(x)-x)
F7	ln((2+sin(x))^x)	[-6.28,6.28]	ln(2+sin(x))+x*cos(x)/(2+sin(x))
F8	exp(sin(x))	[-6.28,6.28]	cos(x)*exp(sin(x))

U nastavku se za svaku ispitnu funkciju nalazi njen graf i graf njene derivacije, nacrtan u programskom paketu MATLAB.

Za svaku funkciju nalazi se i po jedan primjer aproksimacije derivacije pronađene programom i iscrtane pomoću programa Gnuplot zajedno s uzorkom točaka koji funkcija aproksimira.

Pronađene funkcije zbog velike duljine zapisa nisu navedene onako kako ih je program ispisao, nego su pojednostavljene koristeći funkciju simplify u programskom paketu MATLAB.

Funkcija F1



Slika 4.1 Ispitna funkcija F1(x) i njena derivacija F1'(x)

Pronađena funkcija:

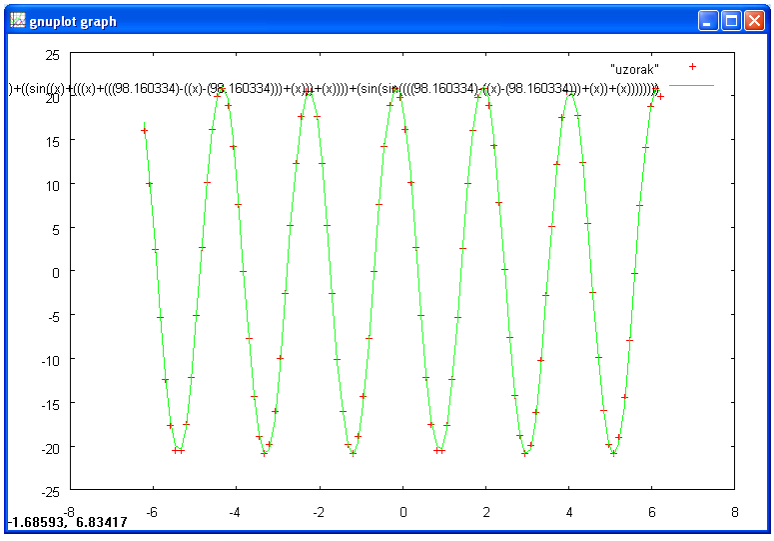
f1(x) = 9.*sin(196.32066800000001194348442368209+3.*x)+9.*sin(sin(3.*x-98.160334000000005971742211841047))

+6.*sin(3.*x-98.160334000000005971742211841047)+sin(x) + sin(x+98.160334000000005971742211841047)

+sin(sin(x+196.32066800000001194348442368209))

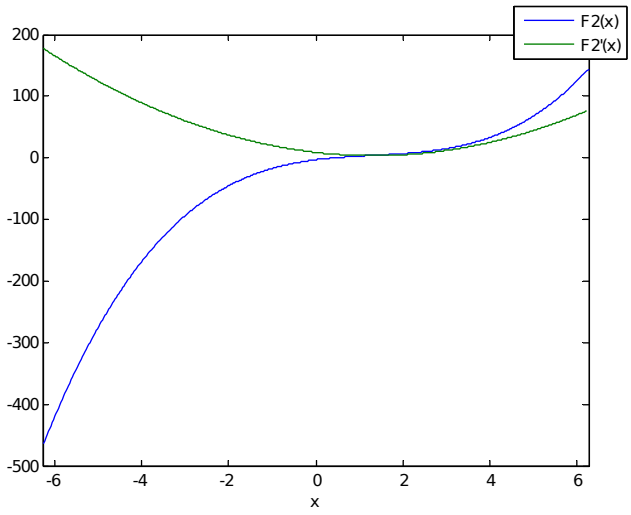
Iznos funkcije kazne: 22.218882

Prosječna relativna pogreška: 0.53%



Slika 4.2 Pronađena funkcija f1(x)

Funkcija F2



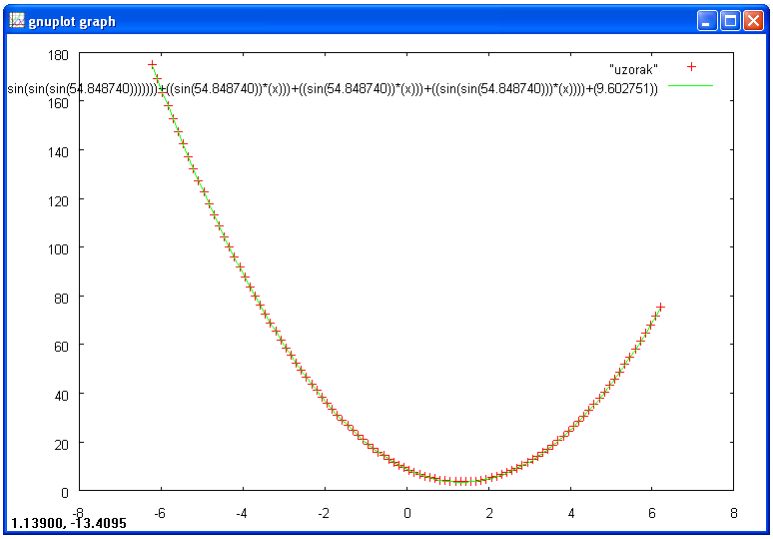
Slika 4.3 Ispitna funkcija F2(x) i njena derivacija F2'(x)

Pronađena funkcija:

f2(x) = -8.00483211729358678111467434856 03*x+3.*x^2+8.926545747694565990926207632 5831

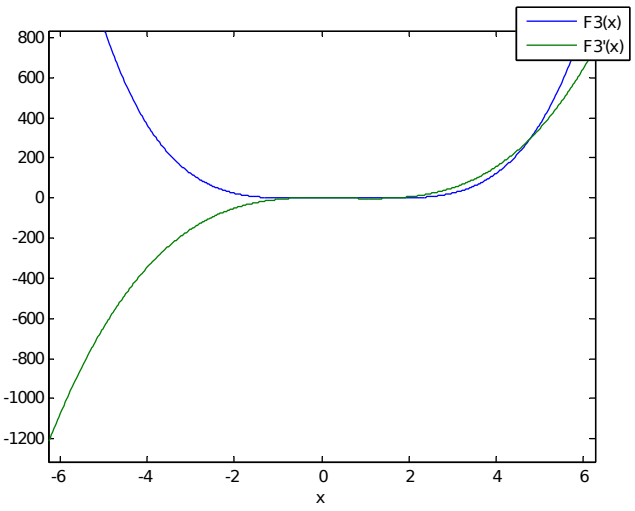
Iznos funkcije kazne: 0.62973

Prosječna relativna pogreška: 0.01%



Slika 4.4 Pronađena funkcija f2(x)

Funkcija F3



Slika 4.5 Ispitna funkcija F3(x) i njena derivacija F3'(x)

Pronađena funkcija:

f3(x) =.22204460492503130808472633361816e-15*x*(18014398509481984.*x^2-26783102720392673.*x

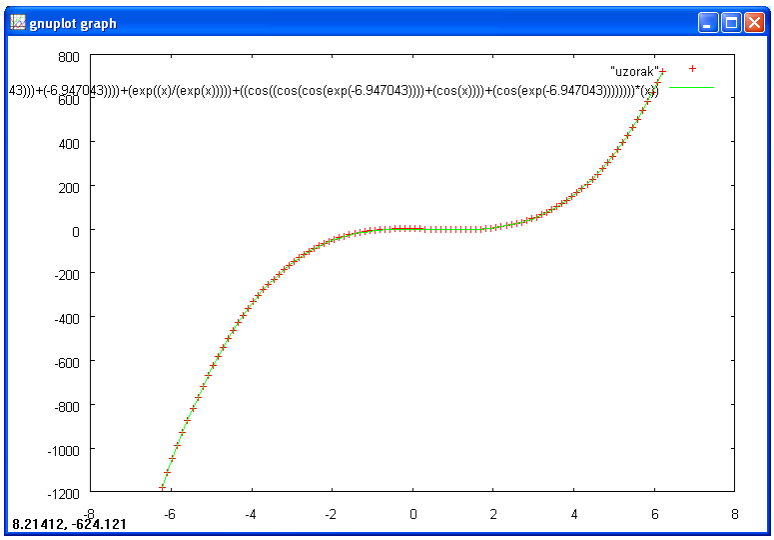
+4503599627370496.*x*exp(-4.74888191001643455990688380552 46+cos(exp(x)))-4503599627370496.

-4503599627370496.*cos(exp(-3.30176255795240913837318430523 74

+2.*exp(x*exp(-1.*x))+cos(.54030269480953774241527298727306+cos(x)))))

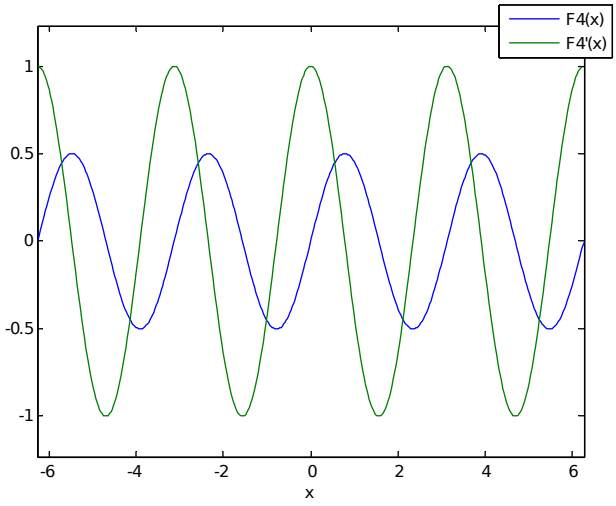
Iznos funkcije kazne: 123.885515

Prosječna relativna pogreška: 0.06%



Slika 4.6 Pronađena funkcija f3(x)

Funkcija F4



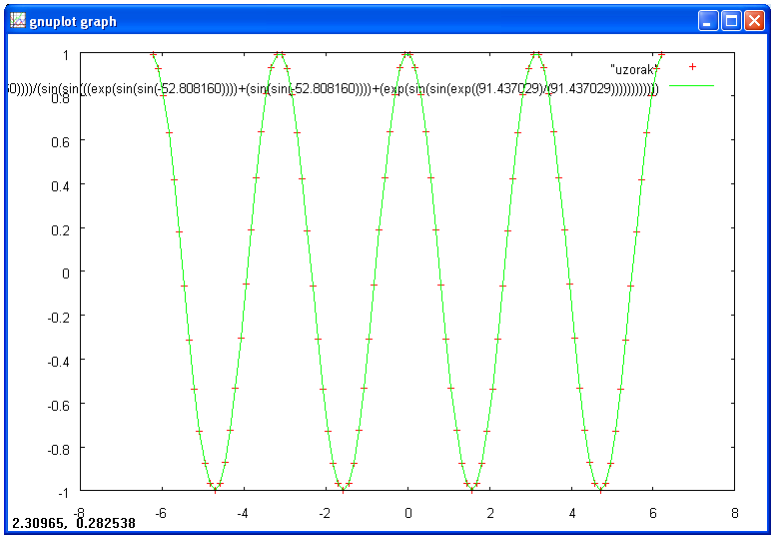
Slika 4.7 Ispitna funkcija F4(x) i njena derivacija F4'(x)

Pronađena funkcija:

F4(x) = cos(2*x)

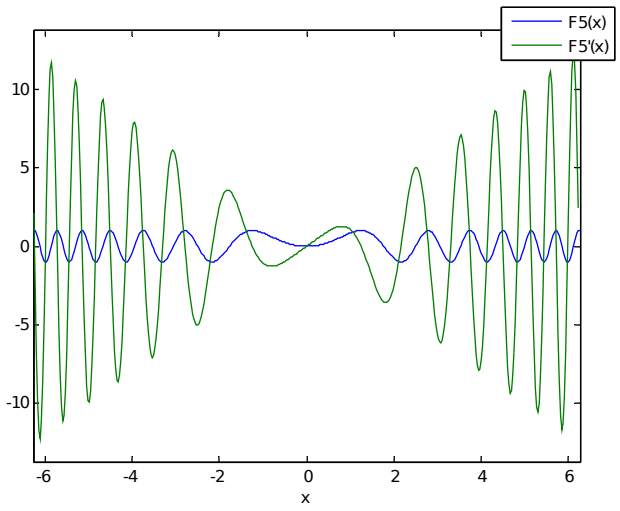
Iznos funkcije kazne: 0.00004

Prosječna relativna pogreška: 0.00%



Slika 4.8 Pronađena funkcija f4(x)

Funkcija F5



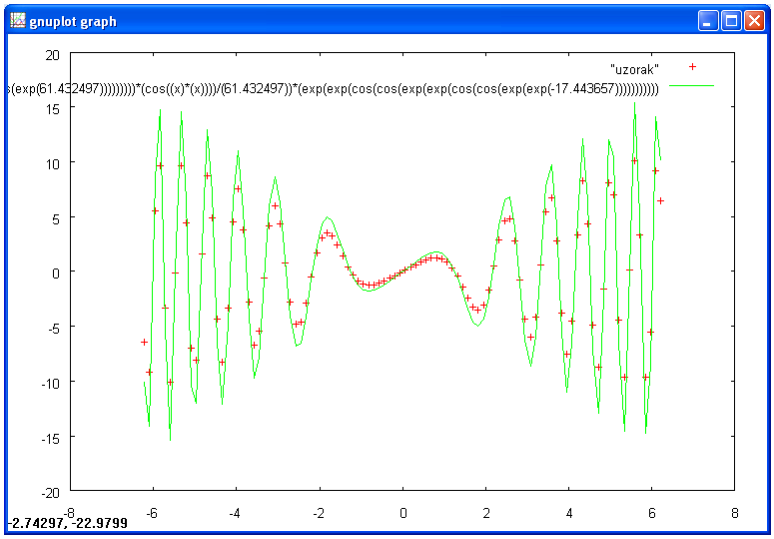
Slika 4.9 Ispitna funkcija F5(x) i njena derivacija F5'(x)

Pronađena funkcija:

f5(x) = 1.8773046606576371253538582095644*cos(x^2)*x

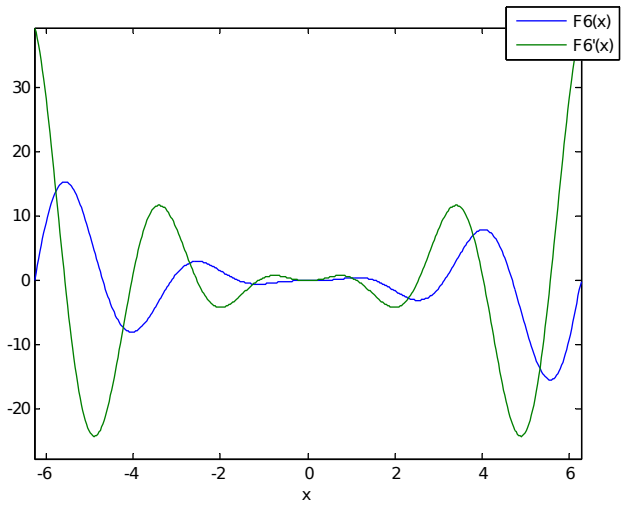
Iznos funkcije kazne: 1.81112

Prosječna relativna pogreška: 0.66%



Slika 4.10 Pronađena funkcija f5(x)

Funkcija F6



Slika 4.11 Ispitna funkcija F6(x) i njena derivacija F6'(x)

Pronađena funkcija:

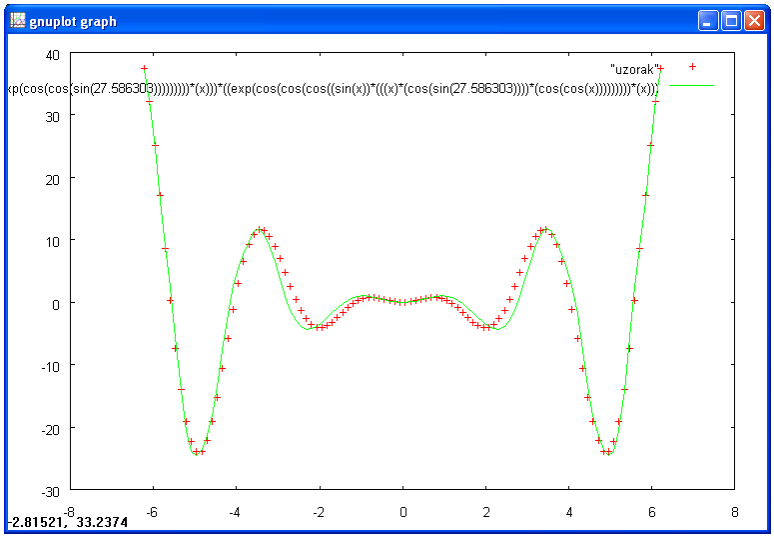
f6(x) = sin(2.264167909314167115297777854721 1*x)*x*exp(cos(exp(cos(.400697687889007669870267136502 66*x))

*cos(sin(sin(sin(sin(.805061449230488501704883219645 13*x))))))

+cos(cos(cos(.805061449230488501704883219645 13*sin(x)*x*cos(cos(x))))))

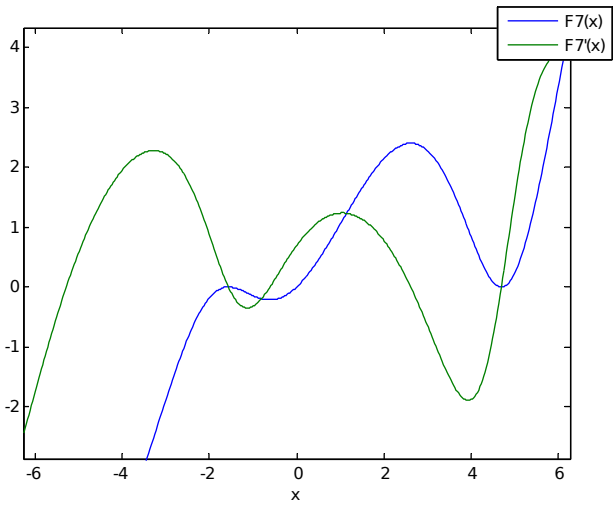
Iznos funkcije kazne: 297.59875

Prosječna relativna pogreška: 2.81%



Slika 4.12 Pronađena funkcija f6(x)

Funkcija F7



Slika 4.13 Ispitna funkcija F7(x) i njena derivacija F7'(x)

Pronađena funkcija:

f7(x) = (cos(sin(sin((.98938970984695862043167835508939*x+sin((1.549502939439002080490581647609 4*x

+.774751469719501040245290823804 68)/x))/x))/cos(sin(1.6330582870622680768946111129480

*sin(1.188864181054313240437636522983 7*x))))+sin(sin(1.2907364994893784082563570336788*x)

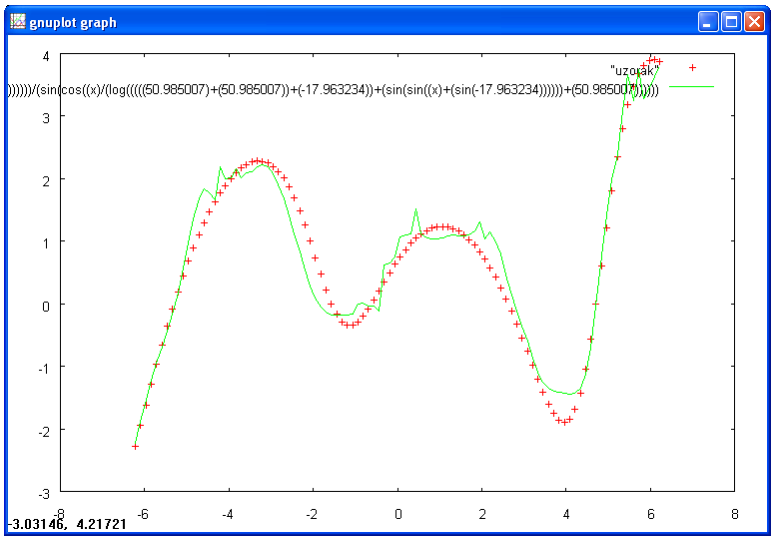
/cos(sin(.21623123366607951978402013537561*x/sin(sin(x-17.963233999999999923602445051074))))))

/sin(cos(x/(-32.577917486317431894349283538759+log(18998405050977420.+140737488355328.

*sin(sin(x+.774751469719501040245290823804 68))))))

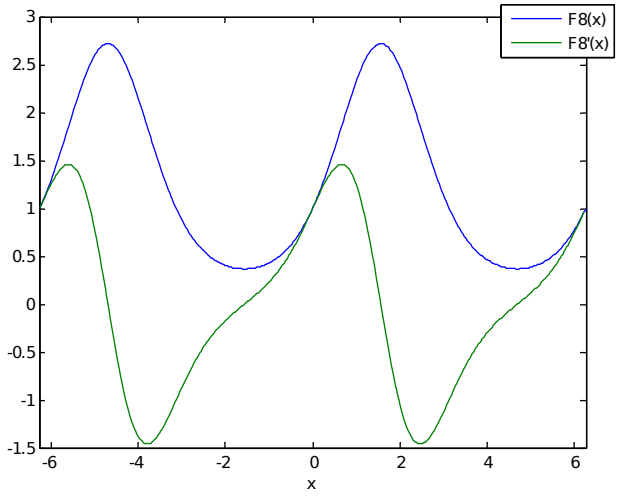
Iznos funkcije kazne: 8.87498

Prosječna relativna pogreška: 3.48%



Slika 4.14 Pronađena funkcija f7(x)

Funkcija F8



Slika 4.15 Ispitna funkcija F8(x) i njena derivacija F8'(x)

Pronađena funkcija:

f8(x) = (cos((((x)/(exp((cos((x)/(exp((cos(x))/((x)*(x))))))/(((x)/(exp((cos(x))/((x)*(-25.798841)))))

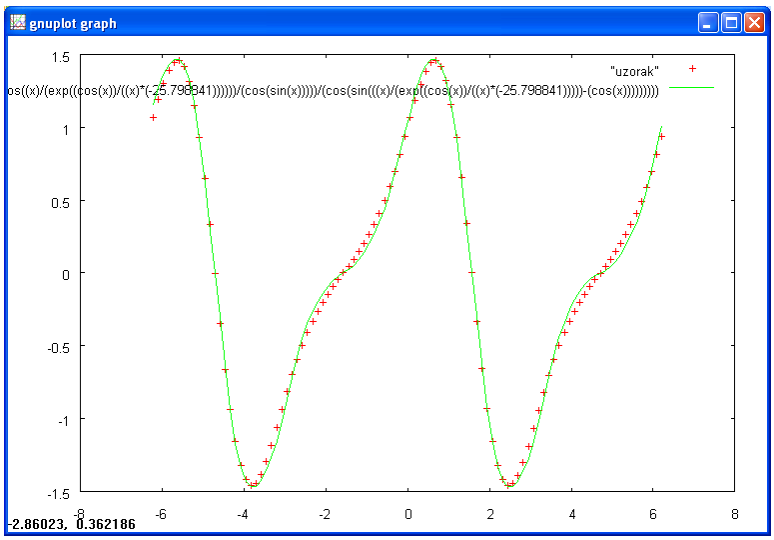
(-25.798841)))))/(exp(((cos((x)/(exp((cos(x))/(-25.798841)))))/(cos(sin(x))))/(((x)/(exp((cos(x))/((x)((x)*(-25.798841))))))

(-25.798841)))))-(cos(x))))/(sin(cos(sin((sin((cos((x)/(exp((cos(x))/((x)(-25.798841))))))/(cos(sin(x)))))

/(cos(sin(((x)/(exp((cos(x))/((x)*(-25.798841)))))-(cos(x)))))))))

Iznos funkcije kazne: 0.22978

Prosječna relativna pogreška: 0.62%



Slika 4.16 Pronađena funkcija f8(x)

1.12 Parametri

Vrijednosti parametara programa znatno utječu na njegov rad, stoga je vrlo bitno optimalno ih odrediti. Empirijski su izmjerene vrijednosti uz koje je program dao najbolje rezultate, tablica 4.2. Te vrijednosti se primjenjuju u programu, ako u datoteci „parametri“ nije specificirano drukčije.

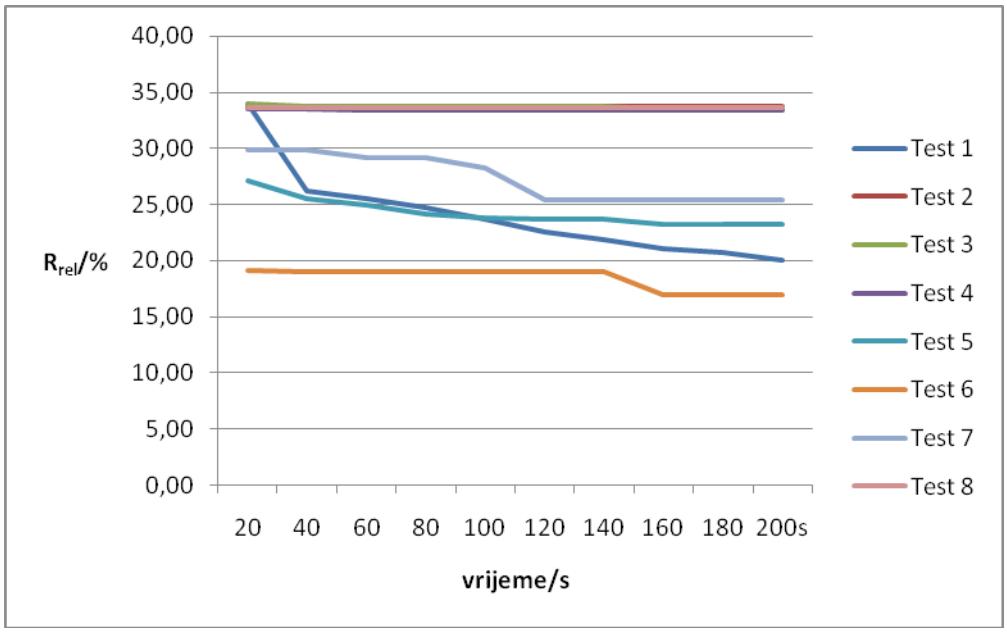
Tablica 4.2 Parametri programa

Ime parametra	Vrijednost	Opis
MAXD	12	Dubina stabla funkcije.
NJED	70	Broj jedinki
POPERATOR	0.7	Vjerojatnost da će čvor stabla u grow metodi biti operator
MAXKONST	100	Maksimalna vrijednost slučajne konstante
MINKONST	-100	Minimalna vrijednost slučajne konstante
PKONSTX	0.7	Vjerojatnost da će se iz skupa konstanti odabrati varijabla x
PENAL	10000	Maksimalni doprinos kazni u jednoj točki funkcije
PMUTACIJE	0.05	Vjerojatnost da će rezultat križanja mutirati
NRUNS	12	Broj pokretanja evolucijskog procesa
MAXTIME	0.66	Maksimalno vrijeme trajanja jednog pokretanja evolucijskog procesa u minutama
MAXTIME1	0.1	Vrijeme nakon kojeg se zaustavlja evolucija ako unutar njega nije poboljšano rješenje.
POBOLJSANJE	0.99	Maksimalni omjer nove i stare kazne pri kojem smatramo da je došlo do poboljšanja
NTOC	100	Veličina uzorka - broj točaka u kojima računamo derivaciju originalne funkcije

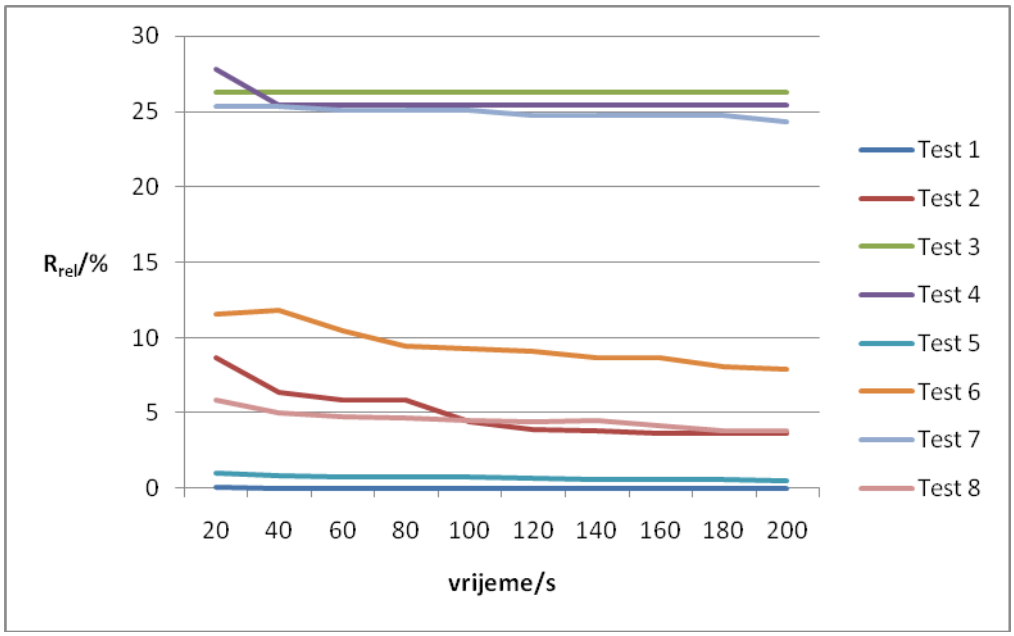
1.13 Mjerenje napretka evolucije kroz vrijeme

Broj pokretanja NRUNS postavljen je na 1. Za svaku funkciju, program je pokrenut osam puta, te je svakih 20 sekundi zabilježena prosječna relativna pogreška najboljeg nađenog rješenja.

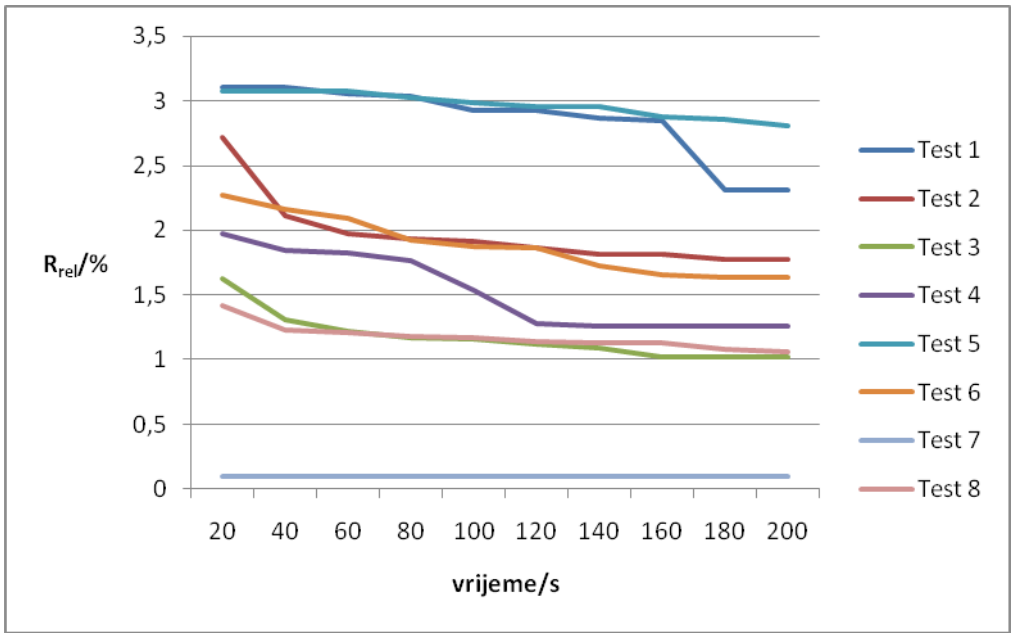
Rezultati mjerenja za svaku ispitnu funkciju prikazani su grafovima na kojima x os predstavlja vrijeme izvršavanja programa, a y os prosječnu relativnu pogrešku u postotcima.



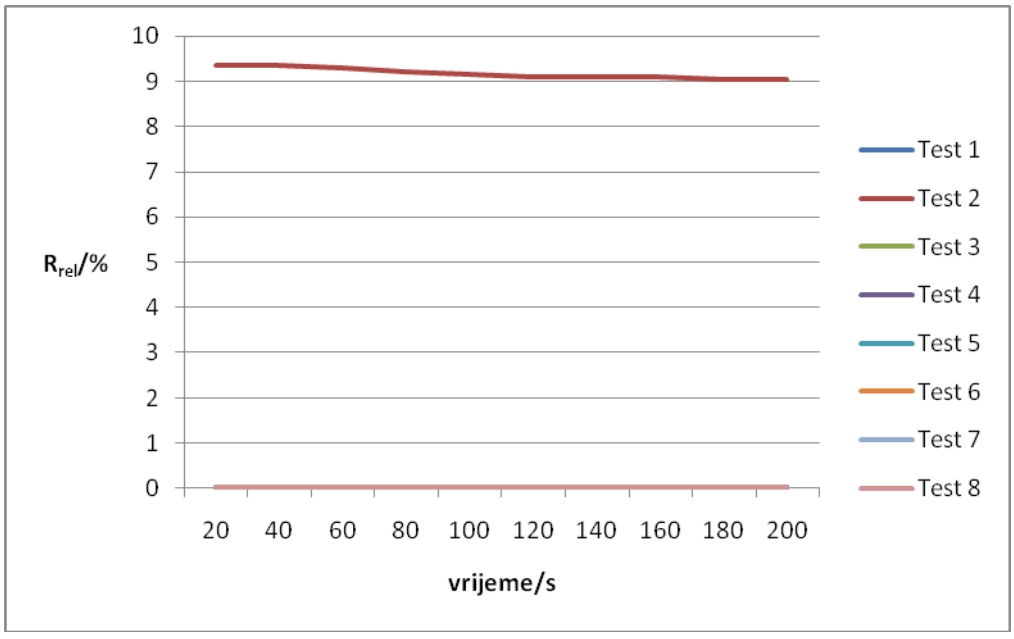
Slika 4.17 Napredak evolucije kroz vrijeme za funkciju F1



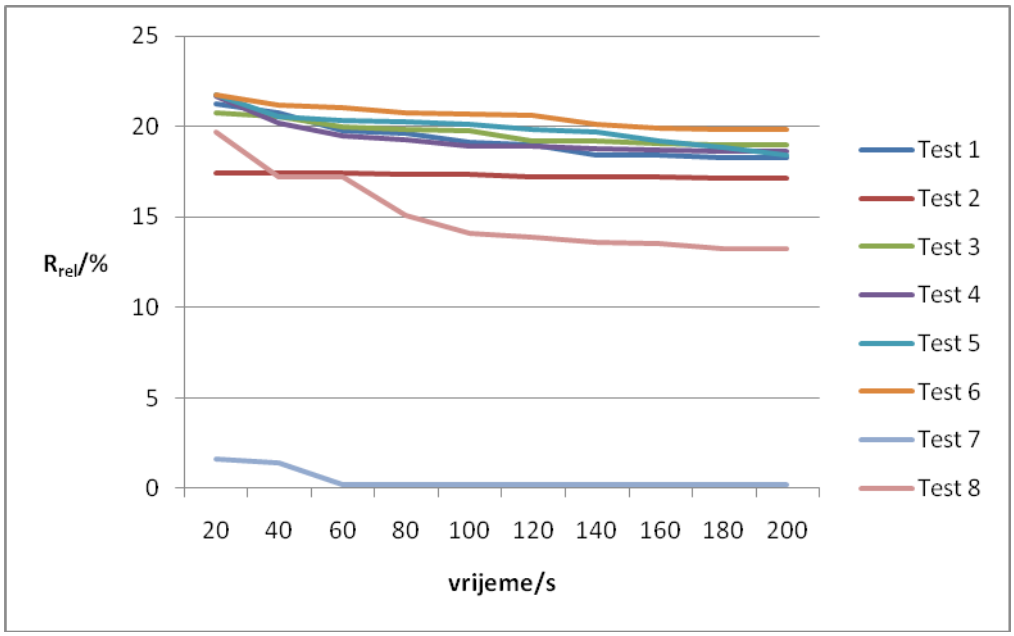
Slika 4.18 Napredak evolucije kroz vrijeme za funkciju F2



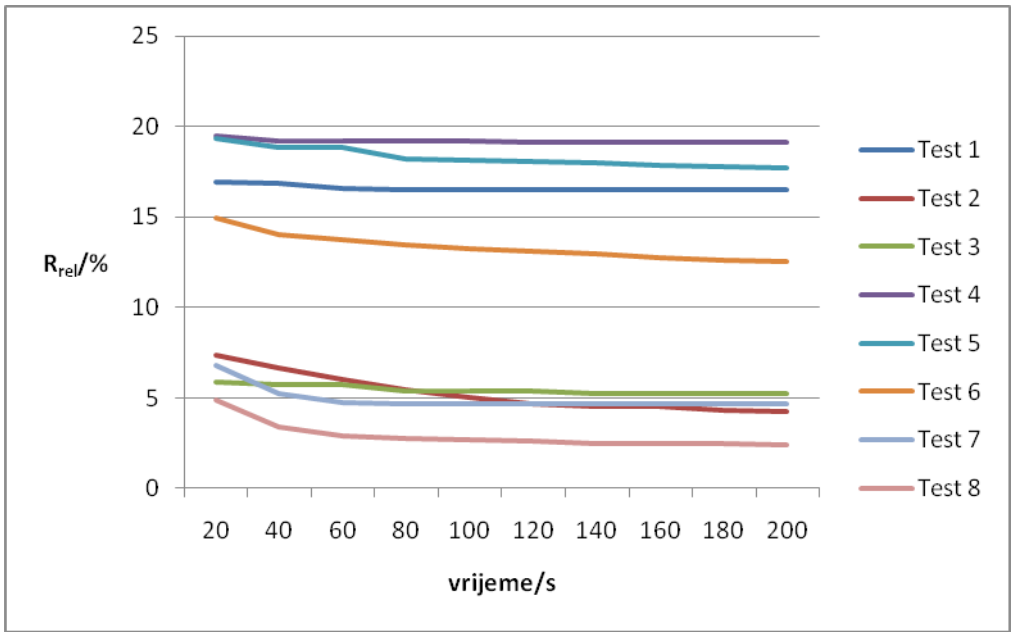
Slika 4.19 Napredak evolucije kroz vrijeme za funkciju F3



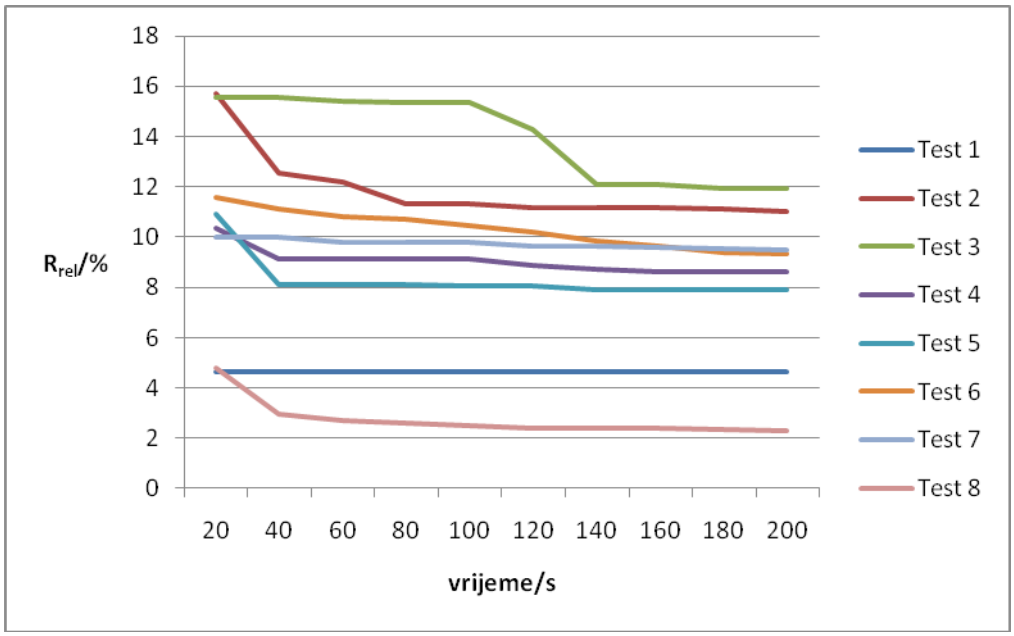
Slika 4.20 Napredak evolucije kroz vrijeme za funkciju F4



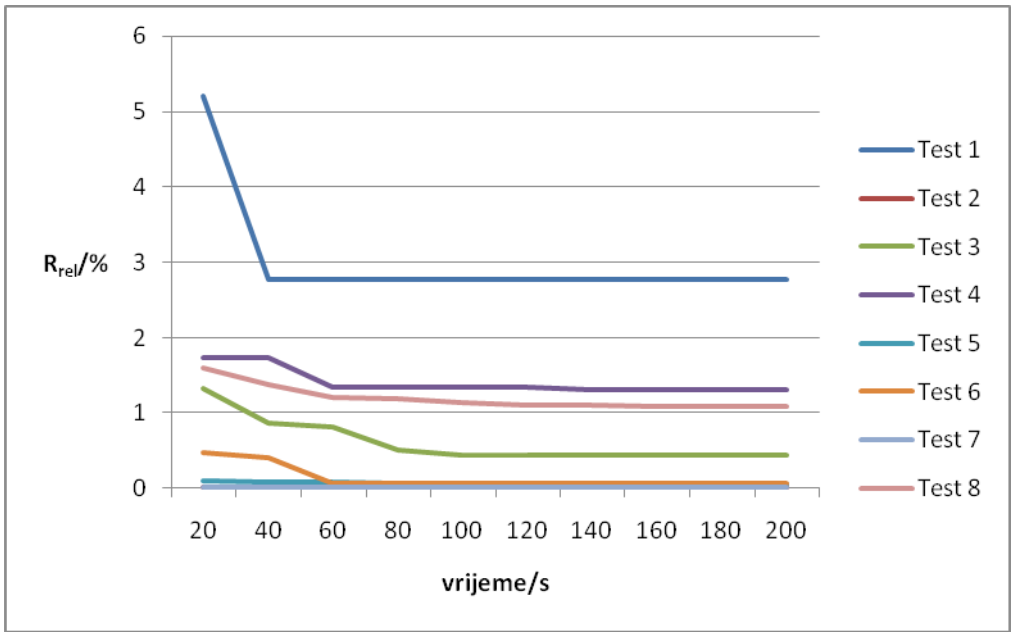
Slika 4.21 Napredak evolucije kroz vrijeme za funkciju F5



Slika 4.22 Napredak evolucije kroz vrijeme za funkciju F6



Slika 4.23 Napredak evolucije kroz vrijeme za funkciju F7



Slika 4.24 Napredak evolucije kroz vrijeme za funkciju F8

Primjećuje se da se većina napretka dogodi u prvih 20 sekundi izvođenja, što sugerira da nema potrebe dugo izvršavati jedno izvođenje. Također se može primjetiti da postoji velika razlika između pojedinih pokretanja iz čega se zaključuje da je bolje više puta pokrenuti evolucijski proces.

U skladu s tim se odabire vrijednost parametara: NRUNS 8, te MAXTIME 0.66 (40 sekundi). Da bi se brzo prekinula ona pokretanja koja već na početku daju loše rezultate, dodaje se i parametar MAXTIME1 -. vrijeme nakon kojeg se izvođenje prekida ako se unutar njega ne popravi rješenje, te se postavlja na 10 sekundi.

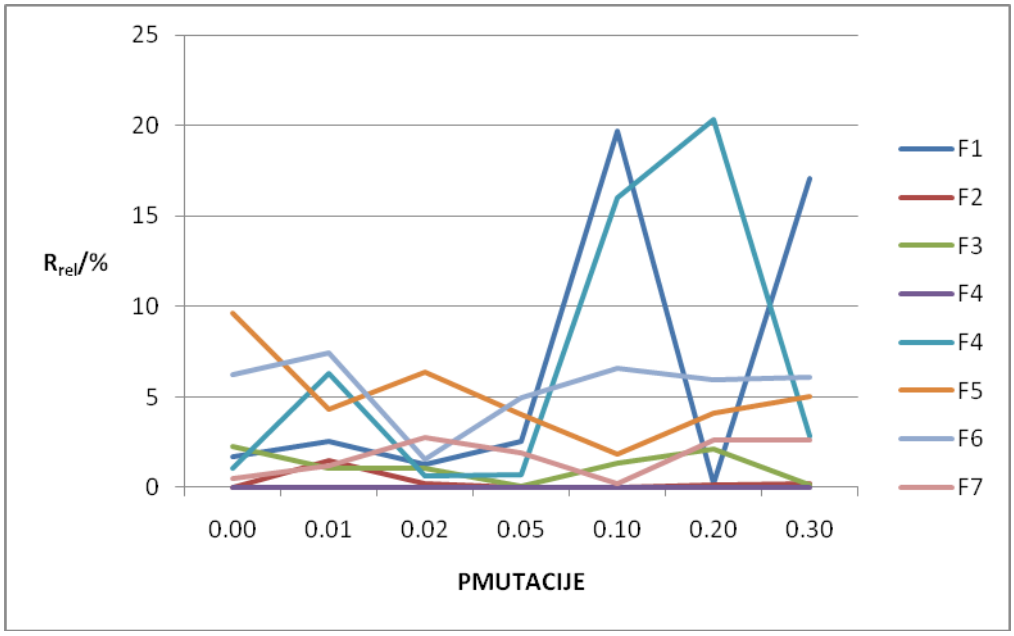
1.14 Utjecaj učestalosti mutacije na proces evolucije

Program će se pokretati s različitom učestalosti mutacije (parametar PMUTACIJE) za svaku od funkcija. U tablici 4.3 se u prvom redu nalaze vrijednosti vjerojatnosti mutacije za koje pokrećemo program. Program se pokreće dva puta i u tablicu se upisuje bolja vrijednost.

Svaki slijedeći redak odgovara izvršavanju programa za jednu od ispitnih funkcija. U poljima se nalazi prosječno relativno odstupanje nađene funkcije od traženog uzorka u postotcima.

Tablica 4.3 Utjecaj vjerojatnosti pojave mutacije na evolucijski proces

PMUTACIJE	0.00	0.01	0.02	0.05	0.10	0.20	0.30
F1	1.70	2.51	1.24	2.50	19.69	0.18	17.05
F2.	0.02	1.47	0.21	0.03	0.00	0.17	0.23
F3	2.27	1.04	1.06	0.07	1.36	2.15	0.16
F4	0.02	0.02	0.02	0.02	0.02	0.02	0.02
F5	1.10	6.32	0.65	0.70	16.00	20.35	2.83
F6	9.64	4.32	6.40	4.03	1.87	4.16	5.07
F7	6.23	7.47	1.56	5.00	6.61	5.96	6.13
F8	0.51	1.22	2.74	1.91	0.23	2.63	2.63



Slika 4.25 Graf za tablicu 4.3

Povećavanjem vjerojatnosti mutacije usporava se konvergencija populacije, ali se zato konstantno dodaje novi genetski materijal, čime se pokušava izbjeći konvergencija prema lokalnom minimumu. Također mutacija daje šansu da se neko dobro rješenje „pogodi“ (pogotovo kad se u traženoj funkciji nalaze konstante, kao u primjeru 1). Iz rezultata mjerenja vidimo da su najpouzdaniji rezultati pri malim vrijednostima vjerojatnosti mutacije (0.02, 0.05). Pri većim vrijednostima mutacija počinje znatno usporavati konvergenciju, te su rezultati izvođenja vrlo slučajni - proces počinje nalikovati slijepom slučajnom pretraživanju.

Zaključak

Metoda genetskog programiranja danas postiže izuzetne rezultate na raznim područjima. Velika prednost metode je što se može primijeniti na probleme o kojima malo znamo i za koje ne postoje egzaktna rješenja.

Na primjeru rješenja problema deriviranja funkcije možemo vidjeti da genetskim programiranjem možemo doći do vrlo dobre aproksimacije funkcije, bez da smo primjenili ikakva znanja o postupku deriviranja – funkcija derivacije koju dobijemo kao rezultat jednostavno je posljedica definicije funkcije dobrote.

Mjerenja nam govore da u ovom rješenju populacija vrlo brzo konvergira, ali ne uvijek prema točnom rješenju, već često prema nekom lokalnom minimumu. S obzirom na to, da bi se povećala vjerojatnost pronalaska dobrog rješenja, nužno je pokrenuti proces više puta.

Literatura

Genetic programming: On the Programming of Computers by Means of Natural Selection; John R. Koza; The MIT Press; 1992.

Seminar: Genetsko programiranje; Donđivić, Kokan, 2008.

A Genetic programming tutorial; John R. Koza, Riccardo Poli;

www.genetic-programming.com/jkpdf/burke2003tutorial.pdf; 2003.

Introduction to Genetic programming; Matthew Walker;

www.massey.ac.nz/~mgwalker/gp/intro/introduction_to_gp.pdf; 2001.

Genetic programming, Chapter 6; A. E. Eiben, J. E. Smith;

www.cs.vu.nl/~eblaauw/ec0708/slides/Lecture06-Chapter6-GeneticProgramming.ppt

Genetic programming – Wikipedia, the free encyclopedia;

http://en.wikipedia.org/wiki/Genetic_programming, 2007.

The Genetic programming Notebook; http://www.geneticprogramming.com/; 2003.

Genetic-programming.org-Home-Page; http://www.genetic-programming.org/; 2007.

Sažetak

Rad se bavi metodom genetskog programiranja i njenom primjenom na problem aproksimacije derivacije funkcije.

U okviru rada napravljen je program u programskom jeziku C++ koji implementira metodu.

Rad sadrži mjerenja efikasnosti programa i mjerenje utjecaja parametara na performanse alogritma.

Ključne riječi: genetsko programiranje, derivacija funkcije, parametri genetskog programiranja, C++

Abstract

This work describes method of genetic programming and it's application on function derivation.

The method is implemented in C++ programming language.

The work measures effect of changing program parameters on program performance.

Key words: genetic programming, function derivation, genetic programming parameters, C++