SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

**SEMINAR**

# Genetski algoritam primijenjen na funkcijsku aproksimaciju

*Luka Mesarić*
Voditelj: *doc. dr. sc. Marko Čupić*

Zagreb, lipanj 2019.

# SADRŽAJ

# 1. Uvod

U ovom radu analizirat će se primjena genetskog algoritma na pretragu beskonačnog skupa matematičkih funkcija unaprijed poznatog oblika. Traži se funkcija jedne varijable u svrhu najbolje aproksimacije zadanog skupa podataka. Implementiran je algoritam za pronalazak najprikladnije funkcije.

U svrhu razumijevanja genetskog algoritma, rad započinje definiranjem i objašnjavanjem osnovnih pojmova, nakon čega se razmatra pristup pri rješavanju konkretnog problema funkcijske aproksimacije. U drugom dijelu rada opisane su mogućnosti izrađenog programa te se diskutira kvaliteta dobivenih rezultata.

Utvrđuje se da je algoritam odlično primjenjiv na dotičan problem.

# 2. Genetski algoritam

## 2.1. Heuristički algoritmi

*Heuristički algoritmi* (*heuristike*) tehnike su kojima se pronalaze približna rješenja problema čije bi egzaktno rješavanje iziskivalo previše vremena ili koje uopće nije moguće egzaktno riješiti [7]. Žrtvujući preciznost ili optimalnost konačnog rezultata, heuristike probleme rješavaju znatno brže od drugih uobičajenih metoda, često u polinomijalnoj složenosti [11].

*Metaheuristike* skupovi su algoritamskih koncepata koje koristimo za definiranje heurističkih metoda primjenjivih na širok skup problema [7]. Prema tome, metaheuristike su zapravo heuristike opće namjene kojima se usmjeravaju problemski specifične heuristike prema području u prostoru rješenja u kojem se nalaze dobra rješenja. One uzimaju uzorke iz skupa rješenja koji je prevelik za cjelovitu obradu. Ne garantiraju da će konačno rješenje problema ujedno biti i globalno optimalno [15].

## 2.2. Evolucijski algoritmi

*Evolucijski algoritmi* (EA) metaheuristike su inspirirane Darwinovom teorijom evolucije [13]. Pripadaju populacijskim algoritmima. Evolucijski algoritmi rade s populacijom rješenja nad kojima primjenjuju evolucijske operatore (selekcija, križanje, mutacija, rekombinacija) čime populacija svakom generacijom postaje sve bolja [7].

Najbitnije pretpostavke koje se koriste, a koje proizlaze iz Darwinove teorije, su sljedeće:

- plodnost vrsta: broj potomaka uvijek je veći nego što je nužno potrebno,
- broj jedinki u populaciji približno je konstantan,
- u populaciji nema identičnih jedinki,
- najveći dio varijacija prenosi se nasljeđivanjem.

## 2.3. Genetski algoritam

*Genetski algoritam* (GA) jedna je od najpopularnijih metaheuristika iz skupine evolucijskih algoritama [13].

Algoritam radi nad populacijom potencijalnih rješenja (engl. *candidate solutions*), koja se još nazivaju i *jedinkama* (engl. *individuals*). Svaka jedinka u populaciji ima skup svojstava koja ju određuju, a koja se nazivaju *kromosomima* [14]. Kromosom se sastoji od *gena* (engl. *genes*), svaki od kojih kodira jedan atribut jedinke. Vrijednosti gena nazivaju se *alelima* (engl. *alleles*) [4]. Geni, a posljedično i kromosomi, ono su što se mijenja evolucijom i što određuje *dobrotu* (engl. *fitness*) jedinke.

*Dobrota* označava u kojoj mjeri neka jedinka zadovoljava svojstva traženog rješenja, a predstavljena je realnim brojem bez mjerne jedinice. Što je dobrota jedinke veća, to je ona bliža ispravnom rješenju. Jedinke s malom dobrotom daleko su od traženog rješenja i imaju manju vjerojatnost biti odabrane za sljedeću iteraciju u evoluciji.

Za provođenje GA potrebno je odrediti dva modela:

- funkciju dobrote te
- reprezentaciju genoma (kromosoma).

*Funkcija dobrote* (engl. *fitness function*) koristi se za izračun dobrote pojedine jedinke, tj. kromosoma. Njeno modeliranje često predstavlja veliki izazov u primjeni GA pri rješavanju stvarnih problema [9]. Naime, tijekom brojnih iteracija GA funkcija dobrote evaluira se za svaku jedinku, što lako može rezultirati stotinama milijuna izračuna. Ako se evaluacija ne izvršava brzo, cijeli algoritam može biti usporen do mjere u kojoj više nije upotrebljiv. Iz tog, kao i iz drugih sličnih razloga, ponekad je nužno pribjeći aproksimativnim izračunima. Primjeri su modeli koji poznaju dobrotu nekog konačnog skupa uzoraka, a ostale vrijednosti računaju interpolacijom, regresijom ili pak umjetnim neuronskim mrežama [10].

*Reprezentacija genoma* (engl. *genetic representation*) način je prikaza i pohrane kromosoma [8]. Najčešći načini zapisivanja su *prikaz nizom bitova* (engl. *binary encoding*), *prikaz poljem ili vektorom brojeva* (engl. *value encoding*), *prikaz permutacijama i matricama* (engl. *permutation encoding*) te *prikaz stablima* (engl. *tree encoding*) [4] [7]. Kao i u slučaju funkcije dobrote, ovo modeliranje može uzrokovati velike poteškoće u primjeni GA na komplicirane probleme. Razlog je zato što zapis kromosoma mora biti pogodan za smisleno korištenje evolucijskih operatora križanja i mutacije.

Kako bi se u svakoj generaciji populacija mogla poboljšavati, potrebni su evolucijski operatori selekcije, križanja i mutacije [7].

Operatorom *selekcije* (engl. *selection*) osigurava se mehanizam diferencijacije rješenja prema njihovoj kvaliteti. Cilj je kvalitetnijim rješenjima pružiti mogućnost generiranja novih rješenja, odnosno jedinke koje ne predstavljaju kvalitetna rješanja isključiti iz procesa reprodukcije. Primjeri selekcija su turnirske selekcije, selekcija linearnim rangiranjem i proporcionalna selekcija [7].

Operatorom *križanja* (engl. *crossover*) jedinke razmjenjuju gene. Iz dvije se jedinke *roditelja* stvara jedna nova jedinka *dijete*. Ideja odabira dva roditelja proizlazi iz najčešćeg načina razmnožavanja u prirodi. Međutim, toga se nije nužno pridržavati te je dopušteno koristiti i više od dvaju roditelja za stvanje jednog djeteta. Postoji iznimno velik broj postupaka kojima se može obavljati križanje. Ključnu ulogu ima reprezentacija genoma. Jednostavna su križanja s jednom ili više točaka prekida. U slučaju prikaza poljem brojeva, križanja se svode na razne aritmetičke operacije.

Operatorom *mutiranja* (engl. *mutation*) mijenja se vrijednost jednog ili više slučajno odabranih gena u kromosomu jedinke. Mutiranjem se proširuje genetska raznolikost populacije na način na koji to nije moguće napraviti samo križanjem. Optimalna vjerojatnost mutacije u velikoj mjeri ovisi o konkretnom problemu koji se rješava. Ipak, ako se vjerojatnost mutacije postavi na preveliku vrijednost, učinak će vrlo vjerojatno biti negativan. Zbog konstantnih mutacija često će biti onemogućena konvergencija k ispravnom rješenju jer će u svakoj generaciji populacija biti preplavljena slučajno generiranim jedinkama.

Pseudokod genetskog algoritma prikazan je u nastavku. Pseudokod je preuzet iz izvora [17].

GENETIC ALGORITHM PSEUDOCODE

```
1  t = 0;
2  initialize(P(t = 0));
3  evaluate(P(t = 0));
4  while isNotTerminated()
5      P_p(t) = P(t).selectParents();
6      P_c(t) = reproduction(P_p);
7      mutate(P_c(t));
8      evaluate(P_c(t));
9      P(t + 1) = buildNextGenerationFrom(P_c(t), P(t));
10     t = t + 1;
```

# 3. Teorijski pristup primjeni algoritma

Razmotrimo problem određivanja realne funkcije jedne realne varijable $f(x)$ koja najbolje opisuje ulazne podatke $P$. Podatci su predstavljeni skupom od $n \in \mathbb{N}$ točaka u Kartezijevom koordinatnom sustavu:

$$P = \{(x_i, y_i) \in \mathbb{R} \times \mathbb{R} \mid i = 1, \ldots, n\} \tag{3.1}$$

Za svaki ulazni skup točaka može se pronaći beskonačno mnogo funkcija koje ga opisuju. Stoga je potrebno postaviti neka ograničenja. Konkretno, zahtijeva se da oblik funkcije $f$ unaprijed bude poznat, dok će (neki) realni koeficijenti biti varijabilni. Ako u funkciji postoji $k \in \mathbb{N}$ koeficijenata, oni će biti opisani vektorom $\vec{Z} \in \mathbb{R}^k$.

Definirajmo realnu funkciju $g(x, \vec{Z})$ koja upravo predstavlja poopćenje funkcije $f(x)$ na način da definira mjesta za koeficijente, a fiksira poznati dio funkcije.

Kvantificiranje koliko dobro neka funkcija $g(x, \vec{Z_0})$ opisuje ulazne podatke $P$ provodit će se koristeći formulu za *sumu kvadrata odstupanja* (engl. *residual sum of squares*) [16]:

$$RSS(\vec{Z}) = \sum_{i=1}^{n} \left(y_i - g(x_i, \vec{Z})\right)^2 \tag{3.2}$$

U formuli (3.2) oznake $x_i$ i $y_i$ predstavljaju vrijednosti $x$ i $y$-koordinata $i$-te točke iz ulaznih podataka $P$. Zbog toga što je $\vec{Z_0} \in \mathbb{R}^k$ neki konkretan vektor, $g(x, \vec{Z_0})$ je funkcija jedne varijable. Slično vrijedi i za izraz $g(x_i, \vec{Z})$ iz formule (3.2), jer je $\vec{Z}$ zapravo fiksiran kroz argument funkcije $RSS$.

Kako bi pri rješavanju problema bilo moguće primijeniti genetski algoritam, potrebno je odrediti reprezentaciju genoma, funkciju dobrote i genetske operatore.

S obzirom na to da se traži niz realnih koeficijenata $\vec{Z}$, prikaz poljem brojeva (engl. *value encoding*) nameće se kao najprikladnija reprezentacija genoma. U tom slučaju geni predstavljaju pojedine koeficijente, dok su aleli vrijednosti tih koeficijenata. Radi jednostavnost, jedinku ćemo poistovjetiti s vektorom koeficijenata te ćemo i nju označavati sa $\vec{Z}$. Primijetimo da same jedinke nisu direktno povezane s traženom

matematičkom funkcijom, već pohranjuju *neke realne brojeve*. Značenje tih brojeva, tj. alela, ponajprije interpretira funkcija dobrote.

Zbog toga što se genetskim algoritmom traže (lokalni) maksimumi, jedinkama za koje smatramo da su bliže ispravnom rješenju potrebno je pridijeliti veću dobrotu. S druge strane, funkcija $RSS$ definirana formulom (3.2) davat će manje vrijednosti za jedinke $\vec{Z}$ koje bolje opisuju podatke $P$, i te će vrijednosti uvijek biti nenegativne. Prema tome, dva moguća jednostavna modela funkcije dobrote su negativna vrijednost i recipročna vrijednost funkcije $RSS$:

$$fitness_1(\vec{Z}) = -RSS(\vec{Z}) \tag{3.3}$$

$$fitness_2(\vec{Z}) = \frac{1}{RSS(\vec{Z})} \tag{3.4}$$

Funkcija $fitness_1$ za svaki će $\vec{Z}$ vraćati negativne vrijednosti, dok će funkcija $fitness_2$ vraćati samo pozitivne vrijednosti. U oba slučaja kvalitetnijim jedinkama bit će pridružena veća vrijednost dobrote.

Realizacija operatora selekcije ne mora biti komplicirana. U ovom slučaju dovoljna je standardna 3-turnirska selekcija.

Zbog toga što se za reprezentaciju genoma koristi prikaz poljem realnih brojeva, za implementaciju operatora križanja najprikladnije je odabrati aritmetičko križanje, $BLX - \alpha$ križanje, prošireno komponentno križanje, ili neko njima slično.

Kod operatora mutiranja nema smisla odabrati verzije koje mijenjaju poredak gena. Naime, već i susjedni geni mogu predstavljati koeficijente na sasvim različitim mjestima i sa sasvim različitim utjecajima na vrijednosti funkcije. Na primjer, zamjenom koeficijenata uz kubni i linearni član polinoma vrlo se vjerojatno neće dobiti kvalitetna jedinka. U ovom je slučaju mutiranje prikladno implementirati kao malu promjenu vrijednosti nekog gena u kromosomu.

# 4. Programska implementacija

Svi programski kodovi javno su dostupni na *GitHub* repozitoriju:
`https://github.com/LMesaric/Seminar-FER-2019`

Kako bi se program mogao izvoditi, na računalu je potrebno imati instalirane sve biblioteke navedene u nastavku.

Programska implementacija pridržava se svih ideja i principa razmatranih u prethodnom poglavlju.

Za implementaciju je korišten programski jezik *Python*. Od javno dostupnih i gotovih biblioteka, korištene su sljedeće:

- *DEAP Framework (Distributed Evolutionary Algorithms in Python)* [2] — za obavljanje genetskih operacija križanja i mutacije te za selekciju,
- *Matplotlib* [3] — za crtanja grafa funkcije koja je dobivena kao rješenje,
- *NumPy* [6] — za ubrzanje provođenja računskih operacija,
- *SymPy* [5] — za parsiranje korisničkog unosa (simbolički račun).

Prilikom pokretanja program od korisnika traži da kroz konzolu unese podatke:

1. relativan ili apsolutan put do datoteke s podatcima za aproksimaciju,
2. tekstualnu reprezentaciju traženog oblika funkcije,
3. maksimalan broj iteracija (generacija) prije zaustavljanja provođenja algoritma.

U datoteci s podatcima u svakoj nepraznoj liniji mora biti zapisan jedan par $x$ i $y$ koordinata, razvojenih jednim ili više razmaka ili tabulatora. Linije koje započinju znakovima "$//$" tretiraju se kao komentari i ignoriraju se. Ako su podatci u datoteci generirani nekom poznatom (originalnom) funkcijom bez varijabilnih koeficijenata, ona se također može zapisati u datoteku. Takva linija mora započeti znakom "$\#$". Zapisana funkcija ni na koji način ne utječe na izvođenje algoritma ni na konačan rezultat, već samo služi kao kontrolni mehanizam za provjeru kvalitete rješenja.

```
# 0.4*sin(1.3*x-2.1)*exp(0.3*x-0.2)
// sine and exponential, wide interval
-10.00000        -0.009313
-9.000000        -0.020770
-8.000000        +0.001970
-7.000000        +0.039268
-6.000000        +0.024768
-5.000000        -0.053665
-4.000000        -0.083886
-3.000000        +0.037204
-2.000000        +0.179718
-1.000000        +0.061997
+0.000000        -0.282694
+1.000000        -0.317120
+2.000000        +0.286088
+3.000000        +0.784435
+4.000000        +0.045211
+5.000000        -1.396684
+6.000000        -1.091025
+7.000000        +1.757017
```

**Primjer 4.1:** Primjer datoteke s podatcima koja sadrži komentare i originalnu funkciju.

Funkcija koju korisnik zadaje kroz konzolu sadržava nepoznate koeficijente. Preporuča se te koeficijente označiti velikim slovima, poput $A, B, C, \ldots$, iako su moguće i druge oznake. Slova $E$ i $I$ su izuzetak – ona će se interpretirati kao Eulerov broj, odnosno imaginarna jedinica. Primjeri dopuštenih unosa slijede u nastavku.

```
A*x^3+B*x^2+C*x+D                (4.1)
A+B*sin(C*x+D)                   (4.2)
A*sin(B*x+C)*exp(D*x+F)          (4.3)
A+B*sqrt(abs(C*x+D))             (4.4)
sqrt(abs(x))                     (4.5)
AH*sin(BU*x+CR)*exp(D*x+F)       (4.6)
```

Primjer 4.5 pokazuje da unos uopće ne mora imati slobodnih koeficijenata, dok primjer 4.6 pokazuje da se pojedini koeficijenti mogu označiti i nizom znakova. Primjer 4.6 ekvivalentan je primjeru 4.3.

Od mogućih implementacija operatora selekcije, 3-turnirska selekcija pokazala je najbolje rezultate.

Za operator križanja odabrana je mješavina aritmetičkog križanja i proširenog komponentnog križanja [7]. *DEAP* takvo križanje obavlja `cxBlend` funkcijom. Uočeno je da se najbolji rezultati dobivaju za koeficijent $\alpha = 0.20$, odnosno kada je $\lambda \in [-0.20, 1.20]$. Parametar $\lambda$ slučajno se odabire za svaki gen u kromosomu.

Za operator mutiranja korištena je Gaussova mutacija [12], odnosno `mutGaussian` funkcija. Ako je jedinka odabrana za mutiranje, neki se njezini koeficijenti uvećavaju ili umanjuju za slučajnu vrijednost dobivenu iz Gaussove distribucije. Najbolji rezultati dobiveni su za $\mu = 0$ i $\sigma = 0.15$, uz vjerojatnost mutacije postavljenu na $0.4$. Uz razne veće i manje vrijednosti parametra $\sigma$ također se mogu dobiti kvalitetni rezultati. Bitno je napomenuti da se uz veće vjerojatnosti mutiranja redovito dobivaju bolji rezultati. Razlog tome je što je selekcija implementirana na način da se kvalitetne jedinke kloniraju, a nekvalitetne se eliminiraju. Klonovi ne pridonose raznolikosti populacije pa ih je potrebno mutirati, što je jednostavno ostvarivo povećanjem vjerojatnosti mutiranja. Dodatno, učestalije mutacije pomažu da populacija prerano ne zapne u krivom lokalnom maksimumu. Uz veliku vjerojatnost mutacije populacija rijetko konvergira.

Za funkciju dobrote odabrana je implementacija bazirana na izrazu (3.3), dakle negativna vrijednost sume kvadrata odstupanja. Razlog takve odluke jest svojstvo *DEAP Frameworka* koje omogućava dodavanje težinskog faktora izračunatoj dobroti. *DEAP* uvijek traži jedinke s najvećom dobrotom pa se postavljanjem težine na vrijednost $-1.0$ vrlo jednostavno dobiva efekt traženja minimalne vrijednosti dobrote. Dodatno, tada se dobrota računa samo kao $RSS$, bez ikakvih izmjena, čime ujedno predstavlja i grešku aproksimacije. Radi potpunosti, testirana je i implementacija bazirana na izrazu (3.4) te su postignuti vrlo slični rezultati.

U najgorem slučaju izvođenje programa zaustavlja se u trenutku kada je dosegnut maksimalno dozvoljen broj iteracija. Otprilike $200$ generacija dovoljno je za kvalitetnu aproksimaciju većine funkcija. Ako se u manjem broju generacija pronađe rješenje čija je pogreška aproksimacije manja od $10^{-7}$, program će se automatski zaustaviti. Korisnik tijekom izvođenja programa može uočiti da nakon nekog trenutka više nema značajnog napretka u pronalasku boljeg rješenja. Ako ne želi čekati da se provede maksimalan broj iteracija, program je moguće prijevremeno zaustaviti pritiskom `Ctrl+C` na tipkovnici. Izvođenje neće biti nasilno prekinuto, već će uredno biti dovršene sve operacije nad trenutnom populacijom i nakon toga će biti prikazani rezultati kao i u slučaju kada program završi bez korisnikove intervencije.

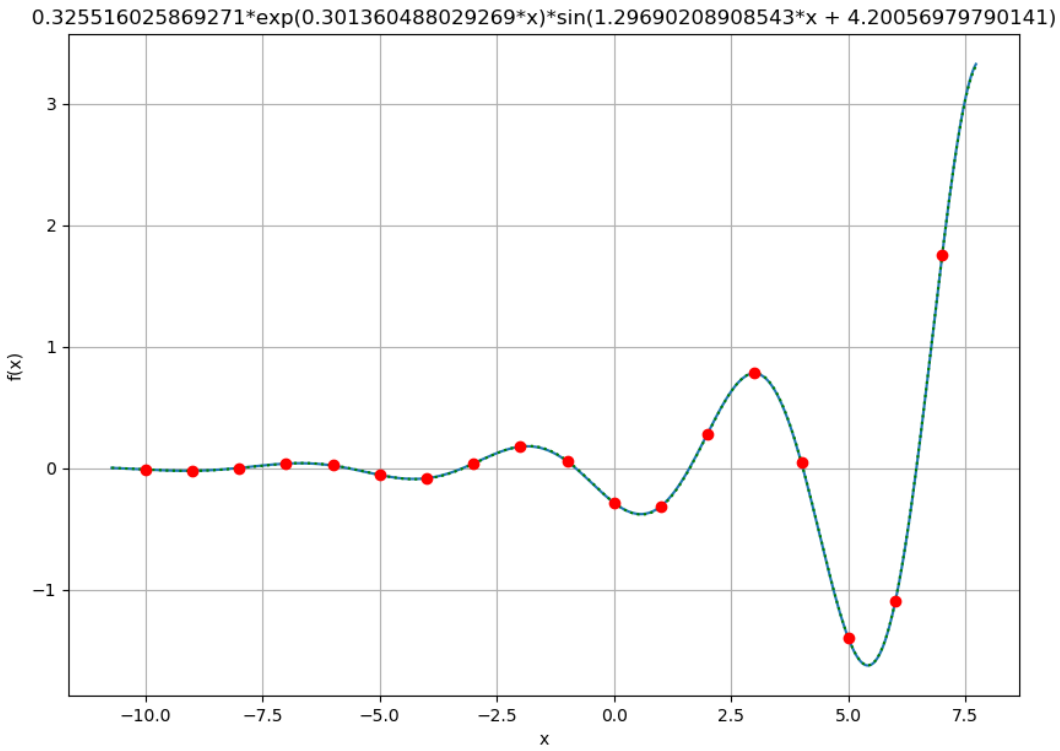
Tijekom izvođenja programa za svaku se generaciju u konzolu ispisuje pogreška aproksimacije najbolje jedinke. Po završetku izvođenja prikazuju se oznake i vrijednosti svakog koeficijenta najbolje jedinke posljednje populacije. Dodatno, ispisuje se i konačan oblik funkcije dobivene aproksimacijom, kao i njezina apsolutna pogreška. Uz ispis u konzolu prikazuje se i graf rješenja.

Crvene točke na grafu predstavljaju ulazne podatke. Punom linijom nacrtana je funkcija dobivena genetskim algoritmom. Njezina jednadžba napisana je odmah iznad grafa. Ako je zapisana u datoteci, originalna funkcija bit će nacrtana točkastom linijom zelene boje.

Graf je moguće pomicati i uvećavati. Inicijalno će koordinatni sustav biti upravo dovoljno velik kako bi mogli biti prikazani svi podatci i vrijednosti funkcije.

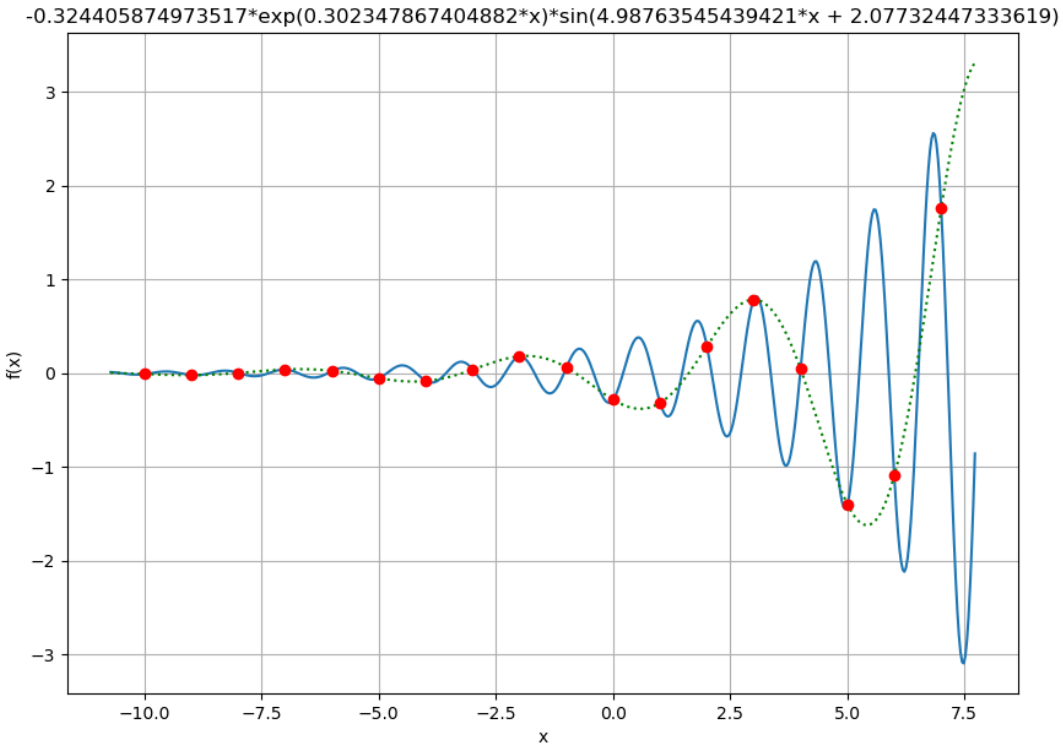
U nastavku slijede primjeri rezultata izvođenja programa za ulazne podatke prikazane u primjeru 4.1. Tražena funkcija ima oblik `A*sin(B*x+C)*exp(D*x+F)`. Maksimalan broj generacija postavljen je na 200, dok je veličina populacije postavljena na 2000 jedinki.

Na slici 4.1 prikazana je najbolja jedinka 200. iteracije, koja se u potpunosti preklapa s originalnom funkcijom. Njezina pogreška aproksimacije iznosi $1.8 \cdot 10^{-4}$.




**Slika 4.1:** Najbolja jedinka nakon 200 iteracija, preklopljena s originalnom funkcijom.

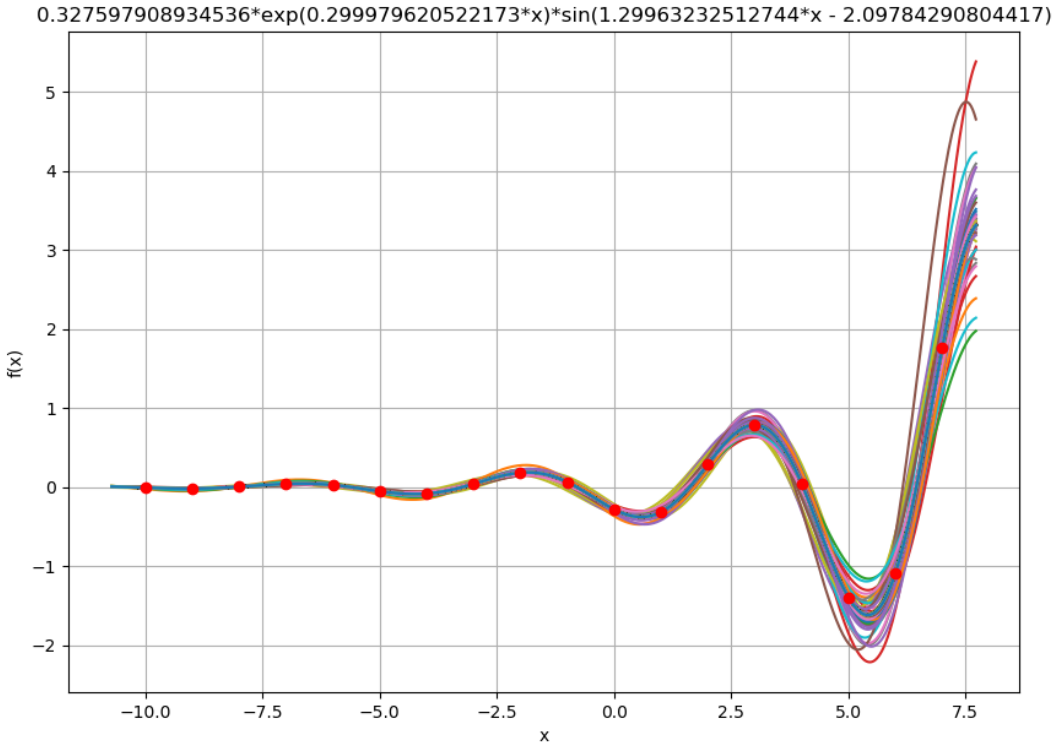
Rijetko je kada originalna funkcija jedina funkcija kojom se podatci mogu aproksimirati. Kada se koriste trigonometrijske funkcije, poput sinusne, nerijetko se događa da dobiveno rješenje ima višestruko veću frekvenciju od frekvencije originalne funkcije. Primjer takvog rezultata prikazan je na slici 4.2. Iako se dobivena funkcija ne može koristiti za interpoliranje podataka, njezina pogreška zapravo je vrlo mala: $4.1 \cdot 10^{-4}$.



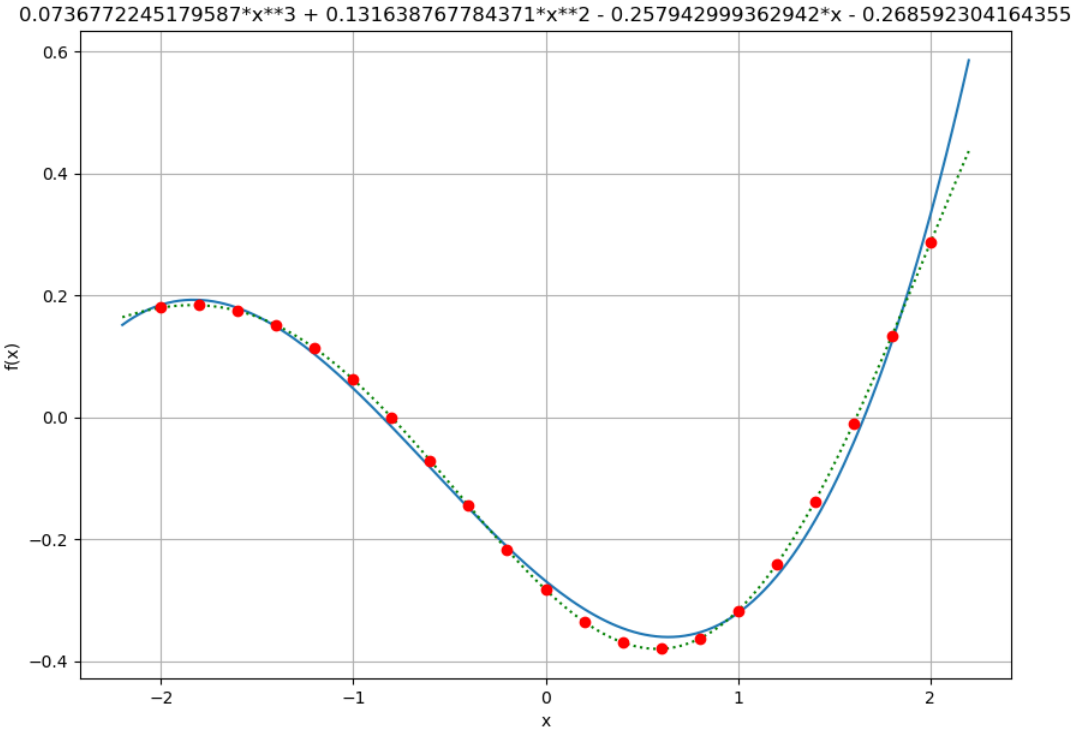

**Slika 4.2:** Najbolja jedinka nakon 200 iteracija, uz jasno vidljivu originalnu funkciju.

Na grafu prikazanom slikom 4.3 nacrtano je svih 2000 jedinki nakon 200 iteracija. Jasno je uočljiva konvergencija populacije prema optimalnom rješenju.

Posljednji primjer malo se razlikuje od ostalih. Originalna funkcija u potpunosti je jednaka, ali se aproksimiraju podatci na manjem intervalu oko ishodišta. Dodatno, traži se aproksimacija polinomom trećeg stupnja: `A*x^3+B*x^2+C*x+D`. Rezultat takve aproksimacije prikazan je na slici 4.4. Već se u približno 30. generaciji dobiva rješenje čija se pogreška aproksimacije, zaokružena na dvije značajne znamenke, više ne mijenja i iznosi 0.0071. Program je pokrenut dvadesetak puta i svako izvršavanje rezultiralo je takvim ponašanjem. Zanimljivo je primijetiti da je dobivena funkcija upravo jednaka najboljem mogućem rješenju za polinom trećeg stupnja, što je potvrđeno kubnom regresijom.




**Slika 4.3:** Populacija od 2000 jedinki nakon 200 iteracija.




**Slika 4.4:** Aproksimacija polinomom trećeg stupnja.

# 5. Zaključak

Genetski algoritam moguće je primijeniti pri rješavanju raznolikih problema, pa tako i tematiziranog problema funkcijske aproksimacije. Izrađenim programom uspješno je dokazana upotrebljivost algoritma.

# 6. Literatura


[1] Kenneth A. De Jong. *Evolutionary Computation: A Unified Approach*. MIT Press, Cambridge, MA, USA, 2016. ISBN 0262529602, 9780262529600.

[2] F. De Rainville, F. Fortin, M. Gardner, M. Parizeau, i C. Gagné. DEAP: A Python Framework for Evolutionary Algorithms. U *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012, Companion Material Proceedings*, stranice 85–92, 2012. doi: 10.1145/2330784.2330799. URL https://doi.org/10.1145/2330784.2330799.

[3] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[4] T. Kuthan i J. Lánský. Genetic algorithms in syllable-based text compression. U *DATESO*, 2007. URL http://ceur-ws.org/Vol-235/paper3.pdf.

[5] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, Š. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, i A. Scopatz. Sympy: symbolic computing in Python. *PeerJ Computer Science*, 3: e103, siječanj 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL https://doi.org/10.7717/peerj-cs.103.

[6] T. Oliphant. NumPy: A guide to NumPy. USA: Trelgol Publishing, 2006–. URL http://www.numpy.org/.

[7] M. Čupić. *Prirodom inspirirani optimizacijski algoritmi. Metaheuristike*. prosinac 2013. URL http://java.zemris.fer.hr/nastava/pioa/knjiga-0.1.2013-12-30.pdf.

[8] Wikipedia contributors. Genetic representation — Wikipedia, the free encyclopedia, svibanj 2017. URL https://en.wikipedia.org/wiki/Genetic_representation.

[9] Wikipedia contributors. Fitness function — Wikipedia, the free encyclopedia, svibanj 2018. URL https://en.wikipedia.org/wiki/Fitness_function.

[10] Wikipedia contributors. Fitness approximation — Wikipedia, the free encyclopedia, prosinac 2018. URL https://en.wikipedia.org/wiki/Fitness_approximation.

[11] Wikipedia contributors. Heuristic (computer science) — Wikipedia, the free encyclopedia, studeni 2018. URL https://en.wikipedia.org/wiki/Heuristic_(computer_science).

[12] Wikipedia contributors. Mutation (genetic algorithm) — Wikipedia, the free encyclopedia, listopad 2018. URL https://en.wikipedia.org/wiki/Mutation_(genetic_algorithm).

[13] Wikipedia contributors. Evolutionary algorithm — Wikipedia, the free encyclopedia, ožujak 2019. URL https://en.wikipedia.org/wiki/Evolutionary_algorithm.

[14] Wikipedia contributors. Genetic algorithm — Wikipedia, the free encyclopedia, travanj 2019. URL https://en.wikipedia.org/wiki/Genetic_algorithm.

[15] Wikipedia contributors. Metaheuristic — Wikipedia, the free encyclopedia, svibanj 2019. URL https://en.wikipedia.org/wiki/Metaheuristic.

[16] Wikipedia contributors. Residual sum of squares — Wikipedia, the free encyclopedia, veljača 2019. URL https://en.wikipedia.org/wiki/Residual_sum_of_squares.

[17] A. Zell. JCell Documentation: Evolutionary Algorithms, 2004. URL http://www.ra.cs.uni-tuebingen.de/software/JCell/tutorial/ch03s05.html.

# 7. Sažetak

U ovom radu analizirana je primjena genetskog algoritma na pretragu beskonačnog skupa matematičkih funkcija unaprijed poznatog oblika. Svrha je pronalazak funkcije jedne varijable koja najbolje aproksimira zadani skup ulaznih podataka. Razmatranje je popraćeno konkretnom implementacijom algoritma za pronalazak najprikladnije funkcije te njezinim grafičkim prikazom.

Nakon pregleda osnovnih pojmova nužnih za razumijevanje genetskog algoritma slijedi teorijsko razmatranje načina njegove primjene na konkretan problem funkcijske aproksimacije. U drugom dijelu rada opisane su mogućnosti izrađenog programa te se diskutira kvaliteta dobivenih rezultata.

Utvrđuje se da je algoritam odlično primjenjiv na dotičan problem.

Svi programski kodovi javno su dostupni na *GitHub* repozitoriju:
`https://github.com/LMesaric/Seminar-FER-2019`