

## (V51) Genetické algoritmy

V roce 1618 Johannes Kepler objevil matematický vztah mezi vzdáleností planet od slunce a jejich oběžnou dobou<sup>12</sup>. Tento objev učinil pomocí jistého množství inteligentních úvah a experimentů. Protože nalezení takového harmonického řešení zdaleka nebývá jednoduché a souvisí i s genialitou, dovolíme si položit otázku: Je možné učinit počítač také tak inteligentním, aby s použitím empirických dat řešil stejné problémy jako např. astronom a matematik J. Kepler?

Zdá se, že odpověď může být kladná. Při počítačovém vyhledávání takových nejlepších řešení (optim, harmonií) výrazně pomáhají i tzv. genetické algoritmy. Jde o významné algoritmy poslední doby, ne-li nejvýznamější. Přitom jejich vývoj vůbec není ukončený.

Vydejme se teď na krátký, ale výstižný výlet do světa genetických algoritmů<sup>3</sup>. Významně pomáhají při vyhledávání optim i harmonií, které v reálném světě často objevujeme velmi obtížně. Napíšeme si program jednoho takového algoritmu do prostředí MATLAB.

Genetické algoritmy vycházejí z principů *genetické vědy*, a proto si ponechávají i stejné názvosloví. Budeme se tak dále setkávat s pojmy typu *populace* (population), *křížení* (crossover), *mutace* (mutation), *množení* (reproduction), *gen* (gen), *chromozóm* (chromosome), *vhodnost*, *schopnost*, *oprávnění* (fitness<sup>45</sup>) apod. Genetické algoritmy nevyžadují žádné hluboké matematické znalosti, spíše se uplatní inteligence a tvořivost řešitele schopného těžit z dostupné informace o problému a jeho symbióza (=vzájemně prospěšné soužití) s počítačem.

Dále budeme vycházet z citované literatury, převezmeme genetický algoritmus z literatury [ ] a opravíme v něm závažnou chybu a uvedeme si řadu příkladů, na kterých si genetický algoritmus vyzkoušíme. V první části budeme řešit jednoduchý symbolický problém, ve kterém od genetického algoritmu budeme přirozeně požadovat symbolické řešení ve významu uvedeného v úvodních větách. Ve druhé části použijeme genetického algoritmu ke hledání optim (=extrémů) nebo harmonií (=vázaných extrémů) různých funkcí.

### 1 Hlavní části genetického algoritmu (Ilustrační úroveň)

Dobrým počátkem bývá příklad. Zkusme jednoduchý, algebraický. Mějme např. rovnici

$$ae x - bd x = ec - bf,$$

kde  $x$  je reálná neznámá a  $a, b, c, d, e$  a  $f$  jsou reálné konstanty. Snadno vypočteme, že

$$x = \frac{ec - bf}{ae - bd}.$$

<sup>2</sup>Tzv. třetí Keplerův zákon

$$T^2 = const * r^3,$$

neboli 'Dvojmoci oběžných dob planet jsou v téměř poměru jako dvojmoci velkých poloos jejich drah (blíže viz např. [1]).

<sup>3</sup>Genetic Algorithms nebo prostě GA.

<sup>5</sup>Well suitability (suitableness), qualification, competence.

Kvůli zjednodušení problému předpokládejme, že  $ae - bd = 1$ .

Pro nás je to jednoduchý příklad. Když např. použijeme následující příkazovou řádku

```
>>x=e*c-b*f
```

dostaneme velmi snadno řešení. Ovšem, to jsme problém vyřešili sami a počítači svěřili pouze konečný vzorec.

Zkusíme naučit náš počítač tak, aby sám našel symbolický zápis takového řešení<sup>67</sup>. Jestliže se nám zdaří ‘takový kousek’, mohlo by to být významné i vzhledem k daleko obtížnějším problémům, se kterými se často setkáváme (viz také náš fyzikální úvod).

Genetický algoritmus začíná pracovat s tzv. *počáteční populací* (initial population). Jsou to možné typy na řešení vybrané náhodně. V této souvislosti hovoříme o tzv. *potenciálních řešeních* (potential solutions). Taková řešení nazýváme *geny*. Než vytvoříme počáteční populaci neboli náhodně vybranou množinu potenciálních řešení, měli bychom mít určitou představu o tvaru skutečného řešení. V našem případě skutečné řešení jistě musí být nějaká lineární kombinace konstant  $a, b, c, d, e, f$ <sup>8</sup>.

Takovými geny v počáteční populaci mohou pro náš případ být např. lineární kombinace  $ec - (d + ae)$ ,  $bf + a + (b - d)$ ,  $d + ae + a + (b - d)$  atd.<sup>9</sup> Základem genetického algoritmu je myšlenka ‘množit’ z genů počáteční populace pouze geny lepší tj. v jistém smyslu bližší ke skutečnému řešení. Takový postup řešení může být velmi efektivní.

Abychom mohli množit pouze ty lepší geny, musíme si rigorózně říci, co to vlastně znamená pojem ‘lepší’. To činíme tak, že každému genu přiřazujeme tzv. *míru vhodnosti* nebo také *sílu*, což je nějaké nezáporné reálné číslo. Nejvhodnější nebo nejsilnější gen v dané populaci je ten, který je nejbližší skutečnému řešení, neboli ten, který má největší sílu. V anglosaské literatuře se v této souvislosti často vyskytuje pojem *fitness value*.

Novou populaci tedy musíme vyvinout tak, aby byla silnější (součet sil jednotlivých genů byl vyšší) než populace předchozí. Přitom pravděpodobnost množení silnějších genů je větší než množení slabých genů. Tímto způsobem si genetický algoritmus zajišťuje evoluci genů směrem ke skutečnému řešení.

Jak zavést sílu každého genu? Přímo se nabízí jeden úplně přirozený způsob. Vezmeme-li v úvahu rovnici, kterou řešíme, pak si můžeme definovat tzv. *chybovou funkci* vztahem

$$e(x) = (aex - bdx) - (ec - bf),$$

čili jako rozdíl levé a pravé strany rovnice. Řešení původní rovnice nalezneme, jestliže  $e(x) = 0$ . Jestliže chceme, aby  $e(x) = 0$ , můžeme místo toho minimalizovat čtverec této funkce, t.j.

$$E(x) = [e(x)]^2.$$

A právě veličina  $E$  může představovat sílu každého genu. Velkou sílu budou mít geny s malou hodnotou  $E$  a naopak. Chceme-li normalizovanou a invertovanou sílu genu (je to často výhodné z výpočetních důvodů a vzhledem k souhlasu síly a hodnoty a také je to v souhlasu s obecnou definicí genetického algoritmu, jak uvidíme dále), potom volíme funkci

$$E_{norm}(x) = \frac{1}{1 + E(x)},$$

<sup>7</sup>We want to make the computer ‘learn’ the symbolic solution to this problem

<sup>8</sup>Takové lineární kombinace lze v počítači poměrně snadno reprezentovat. Např. pomocí tzv. polské notace, binárních stromů apod.

<sup>9</sup>Jistě můžeme systematicky vytvářet a prověřovat spoustu lineárních kombinací s tím, že jednou určitě nalezneme tu správnou. Tento vyčerpávající postup však vůbec nemusí být výhodný. Možností může totiž být exponenciálně mnoho, a to by mohlo být velmi a velmi zdlouhavé až neřešitelné. V našem případě bychom zřejmě mohli prověřovat i  $6^6 = 46656$  možností. A to jde o velmi jednoduchý příklad.

čili nejlepší geny mají hodnotu blízkou jedné a nejhorší geny hodnotu blízkou nule.

Ještě poznamenejme, že zatím uvažujeme pouze jednu proměnnou  $x$ . V případě, kdy se setkáme s jednou chybovou funkcí více proměnných  $e(x_1, x_2, \dots, x_m)$ , zavádíme chybovou funkci ve tvaru

$$E(\mathbf{x}) = [e(\mathbf{x})]^2,$$

kde  $\mathbf{x} = (x_1, x_2, \dots, x_m)$ . Jestliže náš problém vede k více chybovým funkcím (např. řešíme-li soustavu dvou rovnic o dvou neznámých)  $e_1(\mathbf{x}), e_2(\mathbf{x}), \dots, e_m(\mathbf{x})$ , které správné řešení musí nulovat současně, potom definujeme chybovou funkci jako

$$E(\mathbf{x}) = \sum_{i=1}^m [e_i(\mathbf{x})]^2$$

nebo v normalizovaném tvaru

$$E_{norm}(\mathbf{x}) = \frac{1}{1 + \sum_{i=1}^m [e_i(\mathbf{x})]^2}.$$

Ohodnocení konstant v původní rovnici např. stylem  $a = 8, b = 5, c = 2, d = 3, e = 2, f = 1$ , vede k ohodnocení jednotlivých genů počáteční populace  $2 * 2 - (3 + 8 * 2) = 4 - 19 = -15$ ,  $5 * 1 + 8 + (5 - 3) = 15$ ,  $8 + 8 * 2 + 8 + (5 - 3) = 34$ . Protože každý gen  $g$  z počáteční populace je potenciální řešení, přiřadíme každému takovému genu sílu

$$(aeg - bdx - ec + bf)^2.$$

Nejlepší gen (=správné řešení) bude mít tuto sílu rovnou nule, pro špatné geny bude tato síla vysoká. Při tomto ohodnocení dostáváme síly pro jednotlivé geny počáteční populace v pořadí 196, 256, 1225. Tomu odpovídají normalizované síly 0.005, 0.0039, 0.0008. Nejlepším řešením podle míry vhodnosti bude tedy první potenciální řešení s největší normalizovanou silou. Tohle řešení bude hrát dále hlavní úlohu v pokračování genetického algoritmu, tj. při vytváření další populace. Jeho pravděpodobnost množení bude totiž nejvyšší. Jestliže sečteme jednotlivé síly genů této populace, budeme požadovat, aby následující populace byla lepší, což znamená, aby měla menší celkovou sílu.

Krátce rekapitulujme. Sestrojili jsme počáteční populaci a ohodnotili každý gen této populace podle toho, jak je blízko skutečnému řešení. Ačkoliv jsme do první populace náhodně vybrali pouze tři potenciální řešení, neznamená to, že jich nemůže být více. S růstem jejich počtu roste možnost dosažení skutečného řešení za málo generací, klesá však rychlost výpočtu.

Dále pokračujeme podobně jako 'přírodní' genetický proces těmito kroky, které podle potřeby neustále opakujeme:

1. Množení (reprodukce) závisující na síle genů (Reproduction based on fitness).
2. Křížení (Crossover).
3. Mutace (Mutation).

*Množení* závisující na síle genů probíhá následovně. Vypočteme celkovou sílu populace jako  $0.005 + 0.0039 + 0.0008 = 0.0097$ . Jestliže stanovíme procentní podíly jednotlivých genů na celkové síle, dostaneme ..... . Jelikož první potenciální řešení je z tohoto pohledu nejlepší, mělo by mít největší pravděpodobnost množení (podobně jako nejsilnější jedinec). K pravděpodobnostem můžeme přejít následujícím způsobem. Vezmeme síly jednotlivých řešení vzhledem k celkové síle. Potom pro jednotlivé pravděpodobnosti bude platit ..... Součet všech pravděpodobností je přirozeně jedna.

Další populaci vytváříme takto. Představme si virtuální kostku se třemi poli (nebo ruletu), která padají s uvedenými pravděpodobnostmi. Příklad další (druhé) populace může tedy být např.:

$$\begin{aligned} ec - (d + ae) \\ ec - (d + ae) \\ bf + a + (b - d) \end{aligned}$$

**Křížení** je fáze, která teď následuje. Může mít různou odobu. Jeho podstata je v tom, že si podle síly genu vytypujeme dva geny, které si vzájemně vymění určitou část svého vyjádření. Nechť je to např. první a třetí gen a provedme vzájemnou záměnu části  $(d + ae)$  prvního genu a části  $bf$  třetího genu. Dostaneme tak populaci

$$\begin{aligned} ec - bf \\ ec - (a + ae) \\ (d + ae) + a + (b - d). \end{aligned}$$

Vidíme, že v tomto případě první potenciální řešení této populace je již správným řešením.

Někdy v populaci, zpravidla však s velmi malou pravděpodobností, uplatňujeme tzv. *mutace*, tj. změny, vyznačující se tím, že vznikají náhle zdánlivě bez vnější příčiny. Příkladem mutace může být, že např. a pravděpodobností 0.01 změním znaménko některého genu, tj. např. + na - apod.

To je celý genetický algoritmus z úrovně preferující didaktické účely. V našem příkladu vidíme, že jsme již ve druhé populaci 'náhodou' našli správné řešení. Takhle snadné to však nebývá.

Celkově můžeme v pseudokódu psát genetický algoritmus v jeho obecné podobě následujícím způsobem:

```
k = 0;
P0 = [ náhodně vygenerovaná počáteční populace genů];
Ohodnotíme každý gen z populace P0 jeho fitness hodnotou
fork = 0 : kmax
iP = [populace nových genů vzniklá křížením a případně mutací náhodně vybraných genů z Pk-1 s
nejlepšími fitness hodnotami];
Ohodnotíme každý gen z iP fitness hodnotou;
dP = [náhodně vybrané geny z Pk-1 s nejhorší fitness hodnotou];    Pk = (Pk-1 - dP) sjednoceno iP;
end;
```

V algoritmu  $P_k$  označuje populaci genů v čase  $k$ ,  $k_{max}$  max. množství populací,  $iP$  je částečná populace nejlepších genů,  $dP$  je částečná populace nejhorších genů. Populace  $P_k$  vzniká z populace  $P_{k-1}$  tak, že nové kvalitnější geny vytěsní část původních nekvalitních genů.

Počty částečných populací  $iP$  a  $dP$  jsou ohraničeny podmínkami  $|iP| \leq |P_k|$  a  $|iP| = |dP|$ , kde operátor  $|\cdot|$  označuje mohutnost populace.

## 2 Genetický algoritmus formálně

Prezentujeme standardní formulaci *genetického algoritmu* tak, jak byla zavedena v literatuře [1]. Ve své nejjednodušší podobě *genetický algoritmus* řeší problém

$$\text{Maximalizovat } f(\mathbf{s}) \text{ pi omezen } \mathbf{s} \in \Omega = \{0, 1\}^n.$$

V tomto případě,  $f : \Omega \rightarrow \mathbb{R}$  je tzv. *funkce vhodnosti* (*fitness function* a  $n$ -dimensionální binární vektory z  $\Omega$  se nazývají *řetězce* (*strings*). My budeme předpokládat, že funkce vhodnosti je reálná nezáporná funkce.

Hlavní odlišnost od jiných metod řešících tento extrémní problém (zejména tzv. programovací metody) je v tom, že *genetický algoritmus* v každé fázi výpočtu udržuje celou sbírku potenciálních řešení z  $\Omega$ , zatímco ostatní metody pouze jediný bod. Taková sbírka potenciálních řešení se nazývá *populací* (*population*) řetězců.

*Genetický algoritmus* při svém vyhledávání extrému startuje z nějaké počáteční populace řetězců  $S(0) = \{\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_M\} \subset \Omega$ , každý složený z  $n$  bitů. Obvykle je počáteční populace vytvořena náhodně. Z této počáteční populace se podle jistých pravidel odvozují další populace  $S(1), S(2), \dots, S(t), \dots$ . Těmito pravidly jsou tři genetické operátory: *výběr* (*selection*), *křížení* (*crossover*) a *mutace* (*mutation*).

Standardní genetický algoritmus používá k výběru metodu ruletového kola. Každý řetězec je na tomto kole zastoupen úměrně podle své síly (své hodnoty funkce vhodnosti). Řetězec přežije do příští generace, je-li vybrán na ruletovém kole. Ruleta se přitom hraje  $M$  krát.

**Příklad:** Předpokládejme, že  $M = 5$  a předpokládejme následující populaci řetězců:  $S(0) = \{(10110), (11000), (11110), (01001), (00110)\}$ . Pro každý řetězec  $\mathbf{s}_i$  v této populaci můžeme vypočítat odpovídající hodnotu funkce vhodnosti  $f(\mathbf{s}_i)$ . Odpovídající velikost pokrytí ruletového kola (také *relativní vhodnost*) je dána vydělením této hodnoty součtem všech hodnot funkce vhodnosti:

$$\frac{f(\mathbf{s}_i)}{\sum_{j=1}^M f(\mathbf{s}_j)}$$

Na tyto hodnoty se můžeme dívat jako na pravděpodobnosti. Zatočíme-li s ruletovým kolem, je řetězec vybrán právě s touto pravděpodobností. Následující tabulka ukazuje seznam řetězců v počáteční populaci a odpovídající hodnoty funkce vhodnosti:

	Řetězec	Vhodnost	Relativní vhodnost
	$\mathbf{s}_i$	$f(\mathbf{s}_i)$	
$\mathbf{s}_1$	10110	2.23	0.14
$\mathbf{s}_2$	11000	7.27	0.47
$\mathbf{s}_3$	11110	1.05	0.07
$\mathbf{s}_4$	01001	3.35	0.21
$\mathbf{s}_5$	00110	1.69	0.11

Abychm vytvořili další populaci, zatočíme 5 krát ruletovým kolem. Řetězce takto vybrané jsou pouze kandidáty pro další populaci. Musí projít ještě křížením a mutací.

Křížení se aplikuje na pár řetězců ovšem s určitou pravděpodobností označovanou  $P_c$ . Pár řetězců pro křížení je přitom vybrán náhodně. Dále je náhodně vybráno přirozené číslo z  $\{1, 2, \dots, n-1\}$  nazvané *místo křížení* (*crossing site*). Následuje vzájemná výměna (swapping) bitů vpravo od místa křížení a to s pravděpodobností  $P_c$ . Proces křížení se opakuje, dokud není generace řetězců (při  $M$  sudém) prázdná nebo může zůstat jeden řetězec (při  $M$  lichém). Následující tabulka ukazuje názorně průběh operace křížení např. pro dva 6-bitové řetězce, přičemž ve sloupci (a) jsou dva řetězce vybrané pro křížení, ve sloupci (b) je náhodně zvoleno místo křížení ( $k = 4$ ) a ve sloupci (c) jsou oba řetězce po křížení:

```
110101  1101|01  110100
100100  1001|00  100101
```

Konečně, po křížení aplikujeme na tyto kandidáty pro příští generaci, operátor mutace. Mutace je komplement bitu v řetězci prováděný s jistou rovnoměrnou pravděpodobností  $P_m$ . Tj. s touto pravděpodobností je hodnota bitu v řetězci změněna buď z 1 na 0 nebo z 0 na 1. Jako příklad, předpokládejme  $P_m = 0.1$  a řetězec  $\mathbf{s} = 11100$  podstupuje mutaci. Nejsnadnější cestou jak určit, který bit to bude, je generovat pro každý bit rovnoměrně rozložené náhodné číslo  $r \in (0, 1)$ . Jestliže  $r \neq P_m$ , daný bit komplementujeme, v opačném případě neděláme žádnou akci. Pro výše uvedený řetězec  $\mathbf{s}$  předpokládejme náhodná čísla  $(0.91, 0.43, 0.03, 0.67, 0.29)$ . Výsledek mutace ukazuje následující tabulka:

Před mutací 11100

Po mutaci 11000

Provedením operace mutace se kandidáti (=řetězce) pro další generaci  $S(t+1)$  stávají jejími členy a celý proces se opakuje v pořadí: vyčíslíme vhodnost pro každý řetězec této generace, užitím ruletového kola provedeme výběr kandidátů pro další populaci na které po té aplikujeme operaci křížení a mutace.

Poznamenejme, že analýza konvergence této metody se provádí užitím tzv. *schémat (schemata)* a je možné ji nalézt např. v [1].

Ještě zmiňme následující problém. Často totiž nepotřebujeme nalézt bod ve kterém má daná funkce maximum, nýbrž bod, ve kterém má daná funkce minimum. To řeší následující věta:

**Věta:** Nechť  $f(\mathbf{x})$  je nezáporná funkce vhodnosti. Jestliže má funkce  $f$  v bodě  $\mathbf{x}^*$  lokální maximum, má funkce

$$\frac{1}{1+f(\mathbf{x})}$$

v bodě  $\mathbf{x}^*$  lokální minimum. Jestliže má funkce  $f$  v bodě  $\mathbf{x}^*$  lokální minimum, má funkce

$$\frac{1}{1+f(\mathbf{x})}$$

v bodě  $\mathbf{x}^*$  lokální maximum.

Zkusme to ověřit. Jestliže  $f'(x) = 0$  potom

$$\left(\frac{1}{1+f(x)}\right)' = \frac{-f'(x)}{(1+f(x))^2} = 0.$$

Jestliže  $f''(x) < 0$  potom

$$\left(\frac{1}{1+f(x)}\right)'' = \frac{-f''(x)(1+f(x))^2 + 2(f'(x))^2(1+f(x))}{(1+f(x))^4} > 0$$

atd.

### 3 Genetický algoritmus v MATLAB

Hledejme extrém  $\mathbf{x}^*$  dané nezáporné reálné funkce  $f(\mathbf{x})$  při známém omezení  $x_{min} \neq *x \neq x_{max}$ . V souladu s běžnou 'genetickou' terminologií budeme každý řetězec nazývat *chromozómem (chromosome)* a každý bit v řetězci *genem*. Formálně tedy pro každý chromozóm  $\mathbf{s}$  platí, že  $\mathbf{s} \in \{0, 1\}^n$  a  $|\mathbf{s}| = n$  a  $\{0, 1\}^n$  je množinou všech řetězců délky  $n$  a jejichž prvky jsou pouze nuly a jedničky. Počet genů v řetězcích je nutné na počátku volit. Je dobré si uvědomit, že malý počet genů snižuje přesnost a naopak velký počet genů neúměrně protahuje dobu řešení.

Populací tedy nazýváme nějakou množinu chromozómů stejné délky. Velikost populace je také nutné na počátku volit. Je dobré si uvědomit, že se zvětšováním mohutnosti populace roste možnost dosažení řešení při malém počtu populací, klesá však výpočtová rychlost.

Během výkladu budeme řešit následující příklad. Budeme hledat maximum funkce  $f(x) = (x^2 - 3x + 2)^{2/3}$  v uzavřeném intervalu  $[1, 2]$ .

#### POČÁTEČNÍ POPULACE

Následující MATLAB funkce `genbin` vytváří počáteční populaci. Má dva vstupní parametry: `bitl` – počet genů chromozómů a `numchrom` – počet chromozómů v populaci. Výstupem je Booleovská matice

(matice obsahující pouze nuly a jedničky) typu  $numchrom \times bitl$ . Každý řádek této matice je jedním chromozómem v počáteční populaci.

```
function chromosome=genbin(bitl,numchrom)
```

```
maxchros=2^bitl;
if (numchrom>=maxchros)
    numchrom=maxchros;
end;
chromosome=round(rand(numchrom,bitl));
```

Abychom např. vygenerovali počáteční populaci pěti chromozómů, každý se šesti geny, použijeme tuto funkci jako

```
>>population=genbin(7,5)
population =
    1 0 0 0 0 1 0
    1 1 1 0 0 0 0
    1 0 1 0 1 0 1
    1 1 1 1 0 0 0
    1 0 0 0 0 0 1
```

Protože extrém  $*x$  očekáváme v intervalu  $x_{min} \neq *x \neq x_{max}$ , je nutné všechny chromozómy, reprezentující čísla v pevné řádové čárce, do tohoto intervalu přetransformovat.

Toho dosáhneme použitím MATLAB funkce `binvreal`, která převádí binární hodnotu chromozómu do žádaného reálného intervalu. Tato funkce bude mít tyto vstupní parametry: `chrom` – chromozóm, jehož binární hodnotu chceme transformovat a spodní  $a = x_{min}$  a horní  $b = x_{max}$  mez reálného intervalu, do něhož chceme binární hodnotu chromozómu transformovat.

```
function rval=binvreal(chrom,a,b)
```

```
[pop,bitlength]=size(chrom);
maxchrom=2^bitlength-1;
realel=chrom.*((2*ones(1,bitlength)).^fliplr([0:bitlength-1]));
tot=sum(realel);
rval=a+tot*(b-a)/maxchrom;
```

Jestliže nyní chceme konvertovat celou počáteční populaci, použijeme např. při intervalu  $x_{min} = 2 \neq *x \neq 4 = x_{max}$

```
>> for i=1:5, rval(i)=binvreal(population(i,:),2,4); end;
>> rval =
    .....

```

## FUNKCE VHODNOSTI

O reálných hodnotách z počáteční populace nemůžeme zatím říci, která je lepší nebo horší vzhledem ke skutečnému řešení. Vyhodnotíme tedy v této fázi sílu jednotlivých chromozómů z počáteční populace. To uděláme jednoduše. Jako MATLAB funkci naprogramujeme známou funkci vhodnosti. Vstupním parametrem  $\mathbf{x}$  této funkce bude vektor reálných hodnot (tj. kandidátů na extrém) dané populace a výstupem vektor hodnot  $\mathbf{fv}$  funkce vhodnosti  $f(\mathbf{x})$  (síly chromozómů populace). Zopakujme, že síla každého chromozómu je nezáporné reálné číslo (fitness value).

Na počátku avizovaný příklad maxima funkce  $f(x) = (x^2 - 3x + 2)^{2/3}$ , která bude mít v MATLAB tvar

```
function fv=f802(x)
```

```
fv=(x.^2-3.*x+2).^2/3$;
```

Jestliže nyní chceme vyhodnotit reálné hodnoty z počáteční populace, provedeme to v MATLAB

```
>>fit= f802(rval)
fit=
    ....
```

## VÝBĚR – RULETOVÉ KOLO

Souhrnná (totální) sílu populace vypočteme jednoduše jako

```
>>sum(fit)
ans=
    ....
```

Chromozómy se stávají kandidáty pro výběr do další populace v závislosti na jejich síle neboli na velikosti jejich vhodnosti. Silné chromozómy mají větší pravděpodobnost přežití tj. toho, že budou vybrány do další populace. Ještě jinak, nejsilnější chromozómy se s velkou pravděpodobností reprodukují (množí). Takto reprodukováné chromozómy se připravují na operace křížení a mutace a po té se stávají členy další generace.

Jak již jsme uvedli, toto množení provádíme stylem ruletového kola. Každý chromozóm má na ruletovém kole takovou plochu, která je v souladu s jeho silou. Pravděpodobnost, že v ruletě vyhraje daný chromozóm je přímo úměrná tomu, jak tento chromozóm fituje<sup>1011</sup> skutečné řešení. Procentní vyjádření ploch na ruletovém kole můžeme vyjádřit následující MATLAB funkcí

```
>>percent=fit/sum(fit)*100
percent =
    ....
```

přičemž  $\text{sum}(\text{percent})=100$ .

To nám ukazuje šance, jakou v naší zvláštní ruletě mají jednotlivé chromozómy pro výběr do dalších populací. Tuto zvláštní ruletu můžeme v MATLAB implementovat také takto:

```
function newchrom=selectga(criteria,chrom,a,b)
```

```
[pop bitlength]=size(chrom);
fit=[ ];
% calculate fitness
[fit,fitot]=fitness(criteria,chrom,a,b);
for chromnum=1:pop
    sval(chromnum)=sum(fit(1,1:chromnum));
end;
% select according to fitness
parname=[ ];
for m=1:pop
    rval=floor(fitot*rand);
    if (rval<sval(1))
        parname=[parname 1];
```

---

<sup>11</sup>Je připravený, schopný

```

else
  for n=1:pop-1
    sl=sval(n);
    su=sval(n)+fit(n+1);
    if (rval>=sl) & (rval<=su)
      parname=[parname n+1];
    end;
  end;
end;
end;
newchrom(1:pop,:)=chrom(parname,:);

```

Vstupními parametry jsou: `criteria` – jméno funkce vhodnosti, podle našeho předchozího příkladu např. 'f802', `chrom` je daná populace chromozómů, pro níž odpovídající plochu na ruletovém kole počítáme. Parametry `a`, `b` již známe. Jsou jimi povolené meze, kde maximum hledáme, tj.  $x_{min}, x_{max}$ . Výstupem `newchrom` je booleovská matice vzniklá 'přirozeným' výběrem silných kandidátů na extrém v dané populaci (také kandidáti pro další populaci).

Nyní již můžeme vykonat fázi výběru nebo množení následující příkazovou řádkou v MATLAB

```

>>matepool=selectga('f802',population,2,4)
matepool =
    1 1 1 1 0 1
    1 1 1 1 0 1
    0 1 1 1 0 0
    0 1 1 1 0 1
    0 1 1 1 0 0

```

## KŘÍŽENÍ

Nyní můžeme vybrané či namnožené kandidáty začít křížit (pářit – to mate). Jak již jsme si řekli, křížení chromozómů, např.  $s_1$  a  $s_2$  je proces, při kterém vzniknou dva nové chromozómy  $s_1^*$ ,  $s_2^*$  tak, že si původní chromozómy vymění svoje pravé podřetězce délky  $i$ , kde  $1 \neq i \neq n$ . Tato délka se vybírá náhodně. Samotné křížení rovněž probíhá náhodně. Poznamenejme, že existují i jiné způsoby křížení [viz např. ]. Samotný proces křížení (výměny genů) jsme si již ilustrovali.

Funkce v MATLAB realizující křížení může mít následující podobu. Vstupním parametrem `chrom` je Booleovská matice reprezentující celou populaci a pravděpodobnost `matenum` =  $P_c$  jejich křížení. Výstupem `chrom1` je Booleovská matice nové 'křížené' populace.

```

function chrom1=matesome(chrom,matenum)

mateind=[ ];
chrom1=chrom;
[pop bitlength]=size(chrom);
ind=1:pop;
u=floor(pop*matenum);
if (floor(u/2)~=u/2)
  u=u-1;
end;

% Select percentage to mate randomly

```

```

while (length(mateind)~=u)
    i=round(rand*pop);
    if (i==0)
        i=1;
    end;
    if (ind(i)~=-1)
        mateind=[mateind i];
        ind(i)=-1;
    end;
end;

% Perform single point crossover
for i=1:2:u-1
    splitpos=floor(rand*bitlength);
    if (splitpos==0)
        splitpos=1;
    end;
    i1=mateind(i);
    i2=mateind(i+1);
    tempgene=chrom(i1,splitpos+1:bitlength);
    chrom(i1,splitpos+1:bitlength)=chrom(i2,splitpos+1:bitlength);
    chrom(i2,splitpos+1:bitlength)=tempgene;
end;

```

Použijme tuto funkci pro páření chromozómů v naší populaci `matepool`, která již prošla výběrem.

```

newgen=matesome(matepool,.6)
newgen =
    1 1 1 1 0 1
    1 1 1 1 0 0
    0 1 1 1 0 1
    0 1 1 1 0 1
    0 1 1 1 0 0

```

## MUTACE

Mutace je poslední fází genetického algoritmu. Co mutace v našem pojetí obnáší? Jak jsme již uvedli, mutace mění jednotlivé geny v jednotlivých chromozómech a to tím způsobem, že jestliže je někde 0, nahradíme ji 1 a opačně. Tento proces zpravidla probíhá s velmi malou pravděpodobností, to znamená, že změna jednotlivých genů je spíše vyjímečná, pravděpodobnost  $P_m$  bývá např. 0.001 apod.

Proces mutace můžeme realizovat následující MATLAB funkcí

```

function chrom=mutate(chrom,mu)

[pop bitlength]=size(chrom);
for m=1:pop
    for n=1:bitlength
        if (rand<=mu)
            if (chrom(m,n)==1)
                chrom(m,n)=0;
            else

```

```

        chrom(m,n)=1;
    end;
end;
end;
end;

```

Vstupním parametrem `chrom` je Booleovská matice reprezentující celou populaci a pravděpodobnost  $\mu = P_m$  mutace genu v chromozómu. Výstupem `chrom` je Booleovská matice nové ‘zmutované’ populace.

Tento proces vykonáme např. příkazovou řádkou

```

>>mutate(newgen,0.005)
ans =
    1 1 1 1 0 1
    1 1 1 1 0 0
    0 1 1 1 0 1
    0 1 1 1 0 1
    0 1 1 1 0 0

```

## GENETICKÝ ALGORITMUS

Předchozími kroky jsme vytvořili novou generaci chromozómů (těch nejsilnějších). Uvedený proces mnohokrát opakujeme vytvářením stále nových a nových silnějších generací vzhledem ke hledanému extrému (tj. chromozómy nové generace by měly být v průměru blíže tomuto extrému než chromozómy generace předchozí).

Následující MATLAB funkce obsahuje všechny tyto kroky a kompletně tak realizuje celý genetický algoritmus:

```

function [xval,maxf]=optga(fun,range,bits,pop,gens,mu,matenum)

newpop=[ ];
a=range(1);
b=range(2);
newpop=genbin(bits,pop);
for i=1:gens
    selpop=selectga(fun,newpop,a,b);
    newgen=matesome(selpop,matenum);
    newgen1=mutate(newgen,mu);
    newpop=newgen1;
end;
[fit,fitot]=fitness(fun,newpop,a,b);
[maxf,mostfit]=max(fit);
xval=binvreal(newpop(mostfit,:),a,b);

```

Vstupními parametry této funkce jsou: `fun` je jménem funkce vhodnosti, `range` je přípustným definičním oborem hodnot, mezi kterými hledáme tu s extrémní hodnotou funkce vhodnosti, v našem případě [1,2], `bits` je počtem bitů chromozómu, `pop` značí mohutnost jedné populace, `gens` počet generovaných populací, `mu` pravděpodobností mutace a `matenum` je pravděpodobností křížení. Výstupními parametry jsou této funkce jsou: `xval` je hodnota z přípustného definičního oboru, ve kterém *genetický algoritmus* vidí maximum funkce vhodnosti a `maxf` je maximum funkce vhodnosti v tomto bodě.

Nyní aplikujme *genetický algoritmus* na řešení našeho problému. Známe rozsah  $x_{min} =$ ,  $x_{max} =$ , použijeme osmi genové chromozómy, každá populace nechť má 6 chromozómů. Pravděpodobnost křížení (mating proportion)  $P_c = 0.6$  a pravděpodobnost mutace  $P_m = 0.005$ . Při hledání generujeme 20 nových populací.

Náš problém zřejmě vyřešíme příkazovou řádkou v MATLAB

```
[xast,maxf]=optga('f802',[1 2],6,5,50,0.005,0.9)
xast =
    ...
maxf =
    ...
```

**Příklad 1:** Objem  $V$  jednoho gramu vody (v  $\text{cm}^3$ ) je v závislosti na teplotě  $\theta$  vody vyjádřen empirickou formulí

$$V(\theta) = 1 - 57.577 \cdot 10^{-6}\theta + 75.601 \cdot 10^{-6}\theta^2 - 35.07 \cdot 10^{-9}\theta^3.$$

Při jaké teplotě bude tento objem minimální. Uvažujeme přitom teplotní rozsah  $\theta \in [3, 4]^\circ\text{C}$ . Teplotu budeme hledat genetickým algoritmem.

....

**Příklad 2:** Navrhneme pomocí genetického algoritmu rozměry pro plechovky na pivo. Do plechovky se má vejít právě 400 ml ( $=400\text{cm}^3$ ) piva. Máme navrhnout její průměr  $D$  a výšku  $H$  tak, aby její povrch byl minimální. Tím ušetříme potíštěný plech a naše náklady na výrobu plechovky budou minimální. V souladu se světovou normou budeme požadovat, aby  $3.5 \leq D \leq 8$  cm.

Povrch plechovky je zřejmě

$$P = \pi DH + \frac{\pi}{2}D^2$$

a objem je

$$V = \frac{\pi}{4}D^2H.$$

Naší úlohou tedy je nalézt dvojici  $D^*, H^*$  pro kterou je  $P$  nejmenší. Vyjádříme-li  $H$  z rovnice pro povrch, potom

$$H = \frac{1600}{\pi D^2}.$$

Dosadíme-li tuto výšku do rovnice pro povrch, dostaneme

$$P = \frac{1600}{D} + \frac{\pi}{2}D^2$$

Pro který průměr má povrch  $P$  nejmenší hodnotu?

....

**Příklad 3:** Pomocí genetického algoritmu vyřešíme následující úlohu. Nechť  $c_1, c_2, \dots, c_m$  jsou daná reálná čísla. Najdeme minimum následující funkce

$$f(x) = (x - c_1)^2 + (x - c_2)^2 + \dots + (x - c_m)^2,$$

tj takové číslo  $x$ , pro které funkce  $f(x)$  nabývá své nejmenší hodnoty na intervalu  $\min(c_i) \leq x \leq \max(c_i)$ . Ověříme, že je to hodnota

$$x^* = \frac{1}{m} \sum_{i=1}^m c_i.$$

Jde o základ metody nejmenších čtverců.

**Úlohy:** Pomocí genetického algoritmu si najděte minima následujících funkcí  $f(x)$  v daném intervalu.

$$f(x) = e^x + \frac{1}{x} \quad [0.5, 1]$$

$$f(x) = e^x - \ln(x) \quad [0.3, 1]$$

$$f(x) = \tan(x) + \frac{1}{x} \quad [0.5, 1]$$

$$f(x) = e^x - \sin(x) \quad [0, 1]$$

$$f(x) = \frac{x^2}{2} - \sin(x) \quad [0.5, 1]$$

$$f(x) = x + \frac{1}{\ln(x)} \quad [1.5, 2.5]$$

$$f(x) = x + \ln^2(x) \quad [0.3, 1]$$

$$f(x) = x - \ln \ln(x) \quad [0.5, 1.5]$$

$$f(x) = e^{-x} + x^2 \quad [0, 1]$$

$$f(x) = e^{1/x} + \ln(x) \quad [1, 3]$$

$$f(x) = \frac{1}{x} + \ln^2(x) \quad [1, 3]$$

$$f(x) = e^{x-1} + \frac{1}{x} \quad [0.5, 1.5]$$

## 4 Závěrem

**Kdy použijeme genetického algoritmu ?** Zpravidla ve složitějších případech hledání lokálních extrémů, kdy klasické (např. gradientní) metody selhávají. Dále jsou užitečné při hledání globálních extrémů, kdy klasické metody lokalizují pouze lokální extrém.

**Existují nějaké hlubší teoretické výsledky související s genetickými algoritmy ?** Ano existují, již jsme se zmínili. Je to např. fundamentální teorém o šíření hodnoty funkce vhodnosti pomocí tzv. schémat. Viz např. literatura [1].

**Jak lze využít genetické algoritmy v případě většího počtu proměnných ?** Jde o to, že k dosažení harmonie potřebujeme najít extrém funkce více proměnných. V tomto případě můžeme také použít `optga` funkci. Avšak je potřebné modifikovat funkci vhodnosti tak, že např. první polovina chromozómu odpovídá proměnné  $x$  a druhá polovina chromozómu proměnné  $y$  v případě dvou proměnných nebo na více částí v případě více proměnných. Např. v případě dvou proměnných reprezentuje osmi genový chromozóm 10010111 proměnné  $x = 9$  a  $y = 7$ .

## Použitá a doporučená literatura:

### Použitá literatura:

- [1 ] G. Lindfield, J. Penny (1995): *Numerical Methods using MATLAB*. Ellis Horwood.
- [2 ] P. Wayner (1991): *Genetic Algorithms*. BYTE, January, Vol. 16, No. 1, pp. 361-368
- [3 ] M.H. Hassoun (1995): *Fundamentals of Artificial Neural Networks*. MIT Press.
- [4 ] V. Kvasnička, J. Pospíchal (1995): *Evoluční algoritmy I: Genetické algoritmy*. Computer World č. 31.

### Doporučená literatura:

- [1 ] D.E. Goldberg (1988): *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.
- [2 ] K. Kristinsson, G.A. Dumont (1992): *System Identification and Control Using Genetic Algorithms*. IEEE Transactions on Systems, Man, and Cybernetics, Vol. 22, No. 5.
- [3 ] Sborník Mendel ... .