ТЕХНИЧЕСКИ УНИВЕРСИТЕТ
СОФИЯ

ГЕНЕТИЧНИ АЛГОРИТМИ ЗА ОПТИМИЗАЦИЯ

Програми за MATLAB ®

Ръководство на потребителя

Андрей Попов
София
2003

Генетични Алгоритми за Оптимизация

Ръководство на потребителя

Разработено като част от дипломна работа:

“Генетични алгоритми за оптимизация – приложение в задачата за синтез на регулатор”

Андрей Попов

ТУ-София

2003г

Съдържание

1. ИНСТАЛИРАНЕ 2

2. СЪЩНОСТ НА ГЕНЕТИЧНИТЕ АЛГОРИТМИ 3

2.1 Основни елементи на ГА 3

2.1.1 Хромозоми 3

2.1.2 Селекция 3

2.1.3 Рекомбинация 4

2.1.3.1 Стандартно кръстосване 4

2.1.3.2 Кръстосване със смесване 5

2.1.4 Мутация 6

2.2 Обща схема на еволюционните алгоритми 6

2.3 Области на приложение на генетичните алгоритми за оптимизация 8

2.3.1 Едно- и многокритериални задачи 8

2.3.1.1 Паралелни алгоритми 9

2.4 Селекция 9

2.4.1 Еднокритериална оптимизация 9

2.4.1.1 Пропорционална на целевата функция селекция 9

2.4.1.2 Рангова селекция 10

2.4.1.3 Гаусова селекция 10

2.4.2 Многокритериална оптимизация 10

2.4.2.1 Метод на оптималните решения 12

2.4.2.2 Метод на недоминираното сортиране 12

3. ПРОГРАМИ ЗА ОПТИМИЗАЦИЯ 13

3.1 Настройки 13

3.2 Основни Функции 15

3.3 Допълнителни функции 16

3.3.1 Редуциране на броя на Парето решенията 16

3.3.2 Номериране на Парето оптималните решения 16

4. ПРИМЕРИ 17

4.1 Пример 1- Тригонометрична функция 17

4.2 Настройка на непрекъснат ПИД регулатор чрез оптимизационен критерии 18

ЛИТЕРАТУРНИ ИЗТОЧНИЦИ 21

1. Инсталиране

MATLAB ® е избран като среда за разработване на програмите поради:

- Голям брой toolbox-ове, позволяващи симулиране и връзка с други задачи;
- Ориентирана към работа с матрици среда;
- Близкият до C и PASCAL синтаксис на езика;
- Широкото използване за решаване на различни задачи от областта на управлението

Програмните модули за генетична оптимизация са общо 31 на брой (вариант A). Те са обединени в 4 основни функции, 3 спомагателни и един файл с настройки (MAT файл) (вариант B). В зависимост от нуждите си, потребителя може да инсталира вариант A или B на програмите.

Процесът на инсталиране се на програмите за MATLAB 6 и по-горни версии състой от следните стъпки:

1. Създава се папка **genetic** в папката **toolbox** на MATLAB;

   C:\MATLAB\toolbox\genetic

2. В новосъздадената папка се дезархивира избраният вариант (geneticA.zip или geneticB.zip);
3. Стартира се MATLAB, ако вече не е стартиран;
4. От падащите менюта се избира *File -> SetPath -> SetPath*. Посочва се новосъздадената директория и избора се потвърждава с *OK*, Save. Затваря се прозореца чрез Close;

За други версии на MATLAB, моля вижте в помощните файлове на MATLAB (*Help* от падащите менюта).

2. Същност на генетичните алгоритми

Генетичните алгоритми (ГА) (Genetic Algorithms (GA)) са стохастичен метод за глобално търсене и оптимизация, който имитира еволюцията на живите индивиди, описана от Чарлз Дарвин. ГА спадат към групата на еволюционните алгоритми.

В еволюционните алгоритми се използват трите основни принципа на естествената еволюция: репродукция, естествен подбор и разнообразие на индивидите, поддържано чрез разликите на всяко поколение с предишното.

При ГА се работи с набор от индивиди, представляващи възможни решения на задачата. Принципа на подбора се прилага, като се използва критерии, даващ оценка за близостта на индивида с желаното решение. В следващото поколение продължават най-добре приспособените индивиди.

Огромното разнообразие от проблеми, както инженерни така и от други области на познанието, налага използването на алгоритми от различен тип, характеристики и настройки.

2.1 Основни елементи на ГА

2.1.1 Хромозоми

При делене на човешка клетка хроматинът (намиращ се в ядрото и изграден от ДНК (дезоксирибонуклеинова киселина), белтъци и РНК (рибонуклеинова киселина) се уплътнява и образува спирални нишки – хромозоми. В хромозомите са разположени гени, които носят наследствените белези на клетката. Всеки ген кодира конкретен протеин и представлява самостоятелен фактор на генетичната информация, който обуславя изявата на определени белези.

При генетичните алгоритми хромозомите представляват набор от гени, които кодират неизвестните променливи. Всяка хромозома представлява допустимо решение на поставената задача. Индивид и вектор на променливите ще бъдат използвани като понятия еквивалентни на хромозома.

Гените от своя страна могат да бъдат булеви, целочислени, с плаваща запетая и стрингови променливи (низове от букви и символи), както и всяка тяхна комбинация.

Множеството от различни хромозоми (индивиди) съставят текущото поколение. Чрез еволюционни операции, като селекция, рекомбинация и мутация, се достига до следващото поколение (потомство).

2.1.2 Селекция

В природата селекцията на индивиди се извършва чрез естественият подбор. Колкото по-приспособен е даден индивид към заобикалящата среда, толкова по-голям е шансът му да оцелее и да създаде потомство, като по този начин предаде генетичната си информация на следващото поколение.

При еволюционните алгоритми селекцията на най-добрите индивиди става въз основа на функционал или функционали (функция на приспособимост (fitness function)) даващи оценка на конкретният индивид. Например такъв функционал може да е квадратичен критерий на грешката между изходите на желана от нас система и реалната, близост на полюсите на затворена система до желаните и т.н. Ако

задачата е за минимизация, то индивидите с по-малка стойност на функционала ще имат по-голям шанс да бъдат избрани за рекомбинация и съответно за продължаване на поколението.

Методи за еднокритериална и многокритериална селекция са разгледани съответно в точки 2.4.1 и 2.4.2.

2.1.3 Рекомбинация

При репродукцията, първо се появява рекомбинацията (кръстосване или кросинговър). При нея гените от родителите формират по някакъв начин изцяло нова хромозома.

Типичната рекомбинация при генетичните алгоритми е операция, изискваща два родителя (възможни са и схеми с повече родители). Два от най-често използваните алгоритми са стандартно кръстосване (Conventional Crossover) и кръстосване със смесване (Blend crossover) [8].

2.1.3.1 Стандартно кръстосване

При този тип рекомбинация родителите си разменят съответни гени. Кръстосването може да бъде едноточково или многоточково фиг. 2.1. За рекомбинацията се използва битова маска *Mask*. Уравнението описващо рекомбинацията е:

$$C_1 = Mask_1 \,\&\, P_1 + Mask_2 \,\&\, P_2 \qquad (2\text{-}1)$$

$$C_2 = Mask_2 \,\&\, P_1 + Mask_1 \,\&\, P_2$$

P_1, P_2 – хромозоми на родителите

C_1, C_2 – хромозоми на децата (резултантни индивиди)

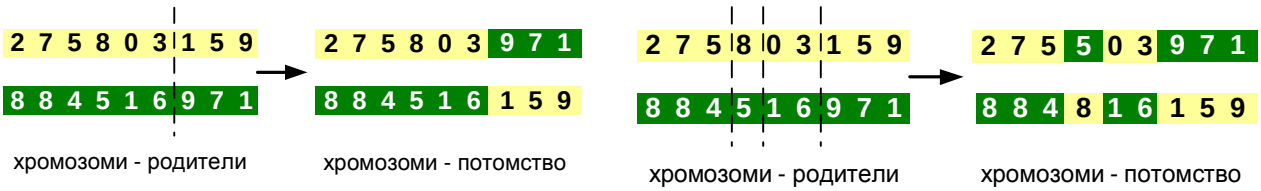
$Mask_1$ и $Mask_2$ - битови маски ($Mask_2$ = NOT($Mask_1$))

& означава побитова операция “*И*”.

За показаният на фиг. 2.1 б) пример:

$Mask_1$ = [1 1 1 0 1 1 0 0 0]; $Mask_2$ = NOT($Mask_1$) = [0 0 0 1 0 0 1 1 1];

P_1 = [2 7 5 8 0 3 1 5 9]; P_2 = [8 8 4 5 1 6 9 7 1];

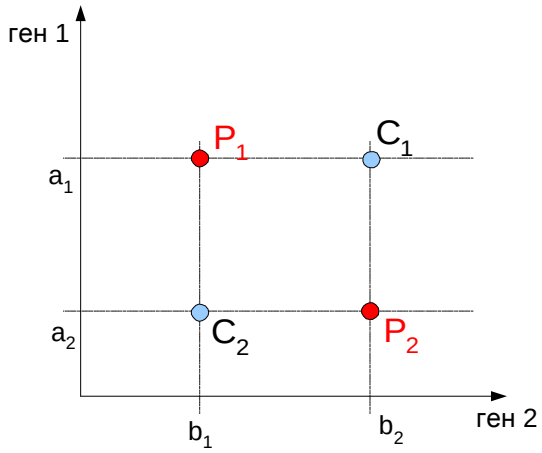


а) Едноточково кръстосване б) многоточково кръстосване

фиг. 2.1 – Кръстосване с битова маска

Геометричното представяне на този тип кръстосване на хромозома с два гена е показано на фиг. 2.2. Този тип кръстосване (с битова маска) може да се използва при всички по-горе изброени типове гени.

В природата такъв тип предаване на информация между поколенията е например цвят на очите, пол и т.н.



Гени в поколение n: (parent genes)
$P_1 = [a_1, b_1]$; $P_2 = [a_2, b_2]$

Mask = [1 0];

Гени в поколение n+1: (child genes)
$C_1 = [a_1, b_2]$; $C_2 = [a_2, b_1]$

фиг. 2.2 – Графично представяне на кръстосване с битова маска

2.1.3.2 Кръстосване със смесване

Математическото описание на този тип кръстосване е:

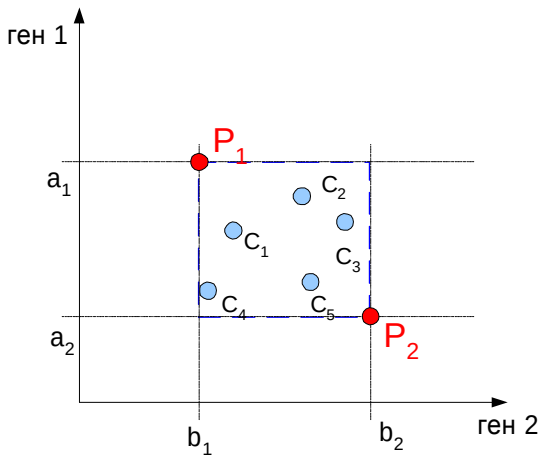
$$C_1 = g.P_1 + (1-g).P_2 \qquad (2\text{-}2)$$

$$C_2 = (1-g).P_1 + g.P_2$$

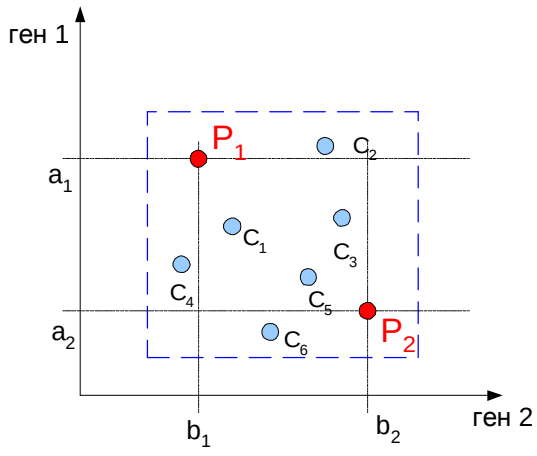
$$g = (1+2.\alpha).r - \alpha \qquad (2\text{-}3)$$

P_1, P_2 – хромозоми на родителите
C_1, C_2 – хромозоми на децата (резултантни индивиди)
α - коефициент на изследване – задава се от потребителя ($\alpha \geq 0$)
r – случайно число в границите (0, 1)

Графичното представяне е показано на фиг. 2.3 и фиг. 2.4.



фиг. 2.3 – Графично представяне на кръстосване със смесване a = 0

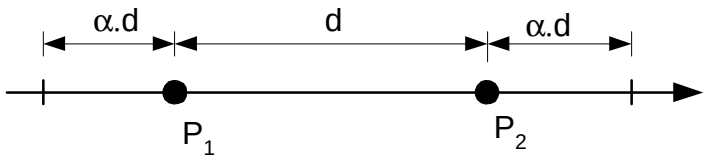


фиг. 2.4 – Графично представяне на кръстосване със смесване a > 0

С помощта на α се променя областта в която може да попадне стойността на резултантният ген. При α=0 се гарантира, че стойността на резултантният ген ще бъде между тази на родителите, а при по-големи стойности може да се изследват съседни области фиг. 2.5.

В природата по подобен начин се предава информацията за пигментация на

кожата, телосложение и т.н.

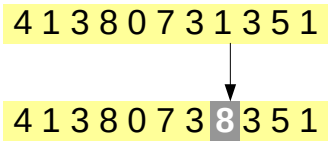


фиг. 2.5 – Промяна на зоната на търсене с промяна на a

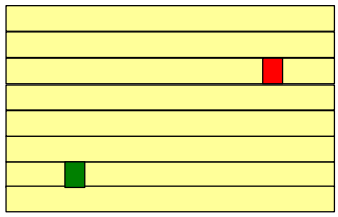
2.1.4 Мутация

Новосъздаденото чрез селекция и кръстосване потомство след това може да бъде подложено на мутация. Мутация означава, че елементи от ДНК се променят. Тези промени са породени главно от грешки при копирането на гените от родителите.

В термините на генетичните алгоритми мутация означава произволна промяна на стойността на ген в поколението фиг. 2.6 а). Хромозомата, чиито ген ще бъде променен, и мястото на гена също се избират по случаен принцип фиг. 2.6 б).



а) мутация в хромозомата



б) места на мутация в поколението

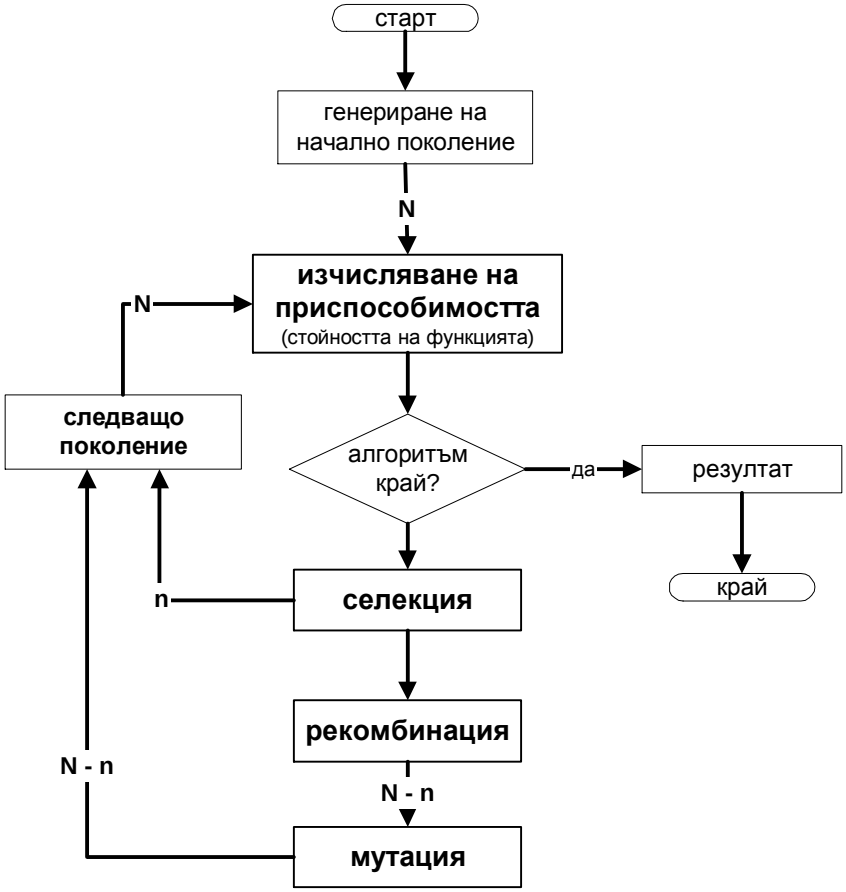
фиг. 2.6 – Мутация при генетичните алгоритми

2.2 Обща схема на еволюционните алгоритми

Еволюционните алгоритми поддържат популация от индивиди (хромозоми), които еволюират чрез използване на селекция и други операции, като кръстосване и мутация. Всеки индивид в популацията получава оценка за приспособимостта си (fitness function) към средата. В термините на оптимизацията това означава, че за всеки набор от променливи се изчислява стойността на функцията, която се минимизира или максимизира. Селекцията служи за избор на най-добрите комбинации, които чрез кръстосване и мутация трябва да доведат до по-добри решения в следващото поколение.

На фиг. 2.7 е показана една от най-често използваните схеми на генетичните алгоритми.

1. Създаване на начално поколение - при повечето алгоритми първото поколение се генерира по случаен принцип – гените на отделните хромозоми се избират случайно измежду азбуката на допустимите гени. Заради по-лесната изчислителна процедура се приема, че всички поколения се състоят от еднакъв брой индивиди (*N* на брой).
2. Следващата стъпка е изчисляване на стойността на функцията, която минимизираме или максимизираме.
3. Проверка за край на алгоритъма – както при всички алгоритми за оптимизация и тук е възможно алгоритъмът да бъде прекратен по:



фиг. 2.7 – Обща схема на еволюционните алгоритми

- Стойност на функцията – стойността на функцията на най-добрия индивид става достатъчно близка до зададена стойност. Обикновено не се препоръчва използването само на този критерии, защото поради стохастичният характер на търсенето не може да се гарантира достигане до желания екстремум в разумно време;
- Максимален брой итерации – това е най-често използвания критерии за спиране. Той гарантира, че независимо дали алгоритъмът е достигнал до екстремум или не ще спре след определено време;
- Достигане на установена стойност – ако в продължение на предварително зададен брой итерации (поколения) не е настъпило подобрение на стойността на функцията алгоритъмът спира.

4. Селекция – измежду всички индивиди в текущото поколение се избират тези, които да продължат развитието си в кръстосването и мутацията. На този етап може да се използва елитарен подход. Това означава част от най-добрите индивиди (*n* на брой) да се прехвърлят без промяна в следващото поколение. По този начин се гарантира, че достигнатата стойност на функцията не може да се влоши (веднъж достигнат екстремумът няма да бъде изпуснат).
5. Кръстосване - избраните чрез селекция индивиди се кръстосват. По този начин се получават нови индивиди, като стремежът е тези индивиди да наследят възможно най-добрата комбинация от характеристики на родителите си.

6. Мутация – чрез случайна промяна на някои от гените се гарантира, че дори нито един от индивидите в текущото поколението не съдържа необходимият ген, пак е възможно да се достигне до екстремум.

7. Ново поколение – избраните от селекцията индивиди се обединяват с получените чрез селекция и мутация и образуват следващото поколение.

2.3 Области на приложение на генетичните алгоритми за оптимизация

Генетичните алгоритми са пряк, стохастичен метод за оптимизация. Тъй като използват поколения от допустими решения (индивиди) те спадат към групата на паралелните алгоритми. Поради стохастичният характер на търсене се налага поставяне на ограничения поне по условие на параметрите на оптимизация (неизвестните променливи).

2.3.1 Едно- и многопараметрични задачи

В зависимост от броя на оптимизационните параметри задачите биват:

- Еднопараметрични $\min_{x\in\Re^1} f(x)$ *(2-4)*
- Многопараметрични $\min_{x\in\Re^n} F(x)$ (2-5)

Въпреки че стандартните генетичните алгоритми са създадени за многопараметрична оптимизация има модификации позволяващи еднопараметрична оптимизация. Такава възможност съществува и в разработените програми за оптимизация. Всеки ген, представящ променлива с реална стойност се разделя на под-гени, кодиращи различните степени на числото 10. Всеки от тези под-гени се променя независимо от другите. Например генът кодиращ числото 437,5391 се разделя на 7 под-гена: 4, 3, 7, 5, 3, 9, 1. Друг начин за представяне е двоичният, където генът се разделя на под-гени, които са степени на двойката. Недостатък на тези представяния е нелинейното преобразуване на двете пространства. При това е възможно близки в едното пространство стойности да са раздалечени в другото (дупки на Хеминг (Hamming cliffs)). Пример за това е разликата в представянето на 15 и 16 в десетична и двоична бройна система.

число	десетична система	двоична система
15	15	[0 1 1 1 1]
16	16	[1 0 0 0 0]

табл. 2.1

2.3.2 Едно- и многокритериални задачи

Инженерните задачи, често изискват удовлетворяване на няколко противоречиви изисквания. Ето защо е подходящо инженерните задачи да се разглеждат като многокритериални задачи. Пример за такава задача в теория на управлението е синтезът на регулатор, при което искаме да имаме както минимална разлика между заданието и изхода на системата, така и минимална стойност на управлението.

$$\min_{x \in \Re^n} F(x) = \left(f_1(x), f_2(x), \ldots f_k(x) \right)^T \qquad (2\text{-}6)$$

2.3.3 Паралелни алгоритми

Тъй като генетичните алгоритми наподобяват еволюцията в природата, където търсенето на решение е паралелен процес, те могат лесно да бъдат пуснати на паралелни машини. Съществуват 3 основни начина за паралелното им изпълнение:

- Обща популация, но паралелно изчисляване на целевите функции – целевата функция на всеки индивид се изчислява на отделен процесор (slave), а генетичните оператори се изпълняват на отделен (master).
- Разделяне на под-популации – цялото поколение се разделя на под-поколения (популации), всяко едно от които се изчислява на отделен процесор.
- Разпределяне на генетичните операции (разширение на горният) - генетичните операции се прилагат само между съседни популации, като по този начин се запазва разнообразието на индивидите.

2.4 Селекция

Както вече беше обяснено селекцията е процес, при който се избират индивидите, които да бъдат подложени на генетичните оператори и да създадат следващото поколение. Селекцията има две основни функции:

1. Да избере най-перспективните индивиди, които да участват в създаване на следващото поколение или да бъдат директно копирани (елитарна стратегия);
2. Да осигури възможност на индивиди с сравнително лоша стойност на целевата функция (функции) да участват в създаването на следващото поколение. По този начин се осигурява запазване на глобалното търсене и не се позволява на един индивид да доминира популацията, като по този начин доведе до локален екстремум.

2.4.1 Еднокритериална оптимизация

При еднокритериалната оптимизация има само една функция, чиито оптимум се търси. За всяка хромозома (индивид) в поколението се изчислява стойността на тази функция и въз основа на селекция се избира кои индивиди да създадат следващото поколение.

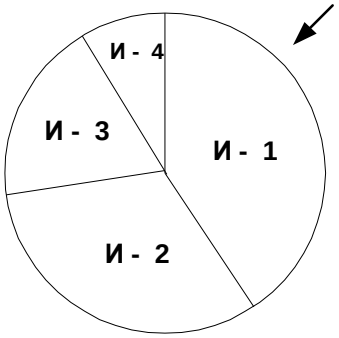
В създадените алгоритми са използвани три основни метода на селекция:

2.4.1.1 Пропорционална на целевата функция селекция

Вероятността за селекция (P) на всеки индивид се определя като отношение на целевата му функция към сумата от целевите функции на всички индивиди. Трябва да се има в предвид, че в този вид селекцията работи за задачи за максимизация. За да работи за задачи за минимизация се налага да се преизчислят стойностите на целевата функция на всеки индивид, така че стойността на най-добре приспособения да бъда максимална.

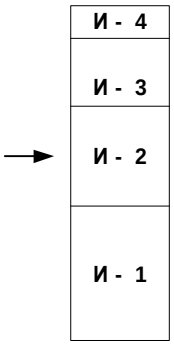
За самият избор на родители се използва метод на ролетното колело. При него окръжността се разделя на *N* на брой сектори пропорционални на вероятността за селекция на индивидите фиг. 2.8. Всеки път когато се завърти ролетното колело (Roulette Wheel Selection) се избира един индивид.

При програмната реализация на метода на ролетното колело се използва линейно представяне на колелото фиг. 2.9. Използва се генератор на случайни числа в интервала (0,1). Това число се умножава по сумата от всички целеви функции. Започва събиране на целевите функции от долу на горе (И-1, И-2, ...), докато тя не надмине случайното число. Взима се номера на последната прибавена целева функция. По този начин по-големите целеви функции получават пропорционална на стойността си вероятност за избор.



фиг. 2.8 – Метод на ролетното колело

индивид	целева функция	P
И – 1	4	0.4
И – 2	3	0.3
И – 3	2	0.2
И - 4	1	0.1



фиг. 2.9 – Програмна реализация на ролетното колело

2.4.1.2 Рангова селекция

Индивидите се сортират според стойността на целевата функция и след това им се присвоява ранг. Рангът на най-добрият е 1, на следващият 2 и т.н. Вероятността за селекцията на всеки индивид се определя чрез нелинейната функция:

$$P = b.(1- b)^{(rank - 1)} \qquad (2\text{-}7)$$

Където b е зададен от потребителя параметър. За селекцията се използва методът на ролетното колело.

2.4.1.3 Гаусова селекция

Поколението се сортира според целевата функция. След това се генерира случайно число с нормално (Гаусово) разпределение, което е мащабирано до размера на поколението (фиг. 2.10). Тук настройваемият параметър на селекцията е средноквадратичното отклонение.

2.4.2 Многокритериална оптимизация

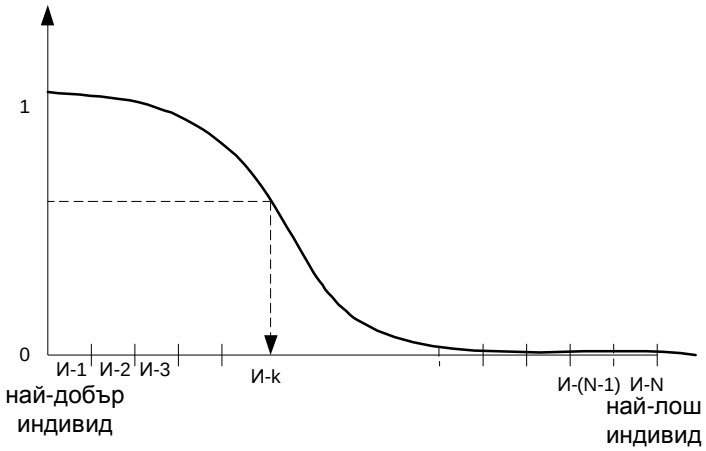
В реалните инженерни задачи обикновено има няколко критерия, които трябва едновременно да бъдат удовлетворени. Обикновено тези критерии нямат оптимум в една и съща точка, което от своя страна означава, че подобрявайки един критерии ние влошаваме друг(и). Това поражда въпроса как да бъдат използвани тези функции за да се намери оптимално решение и как да бъде претърсвана областта на параметрите.

$$\min_{x \in \Re^n} F(x) = \left(f_1(x), f_2(x), \ldots f_k(x) \right)^T \qquad (2\text{-}8)$$

Приложените в алгоритмите методи за селекция спадат към групата на Парето методите. При методите на Парето (Pareto) се получават множество от недоминирани решения, от които потребителя избира това, което най-добре отговаря на нуждите му.

Дефиниция 1 (Доминиране): Векторът *v* доминира *u*, ако:

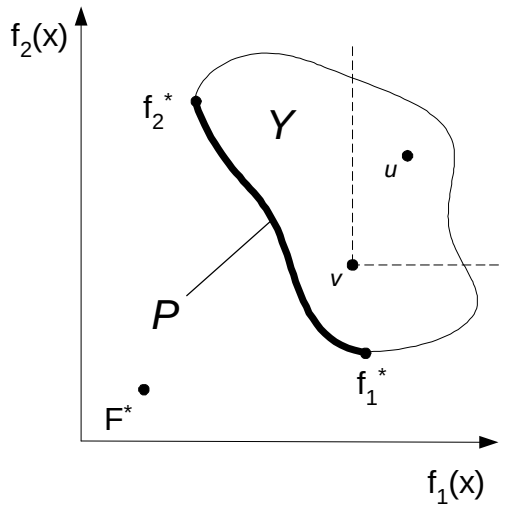
$$\forall i \in \{1, 2 \ldots k\} : f_i(v) \le f_i(u) \text{ и } \exists j \in \{1, 2 \ldots k\} : f_j(v) < f_j(u) \qquad (2\text{-}9)$$



фиг. 2.10 – Гаусова селекция

Дефиниция 2 (Парето оптималност): Векторът x∈S е Парето оптимално решение, ако и само ако не съществува y∈S, за което $v = f(y) = \left(f_1(y) \ldots f_k(y) \right)$ доминира $u = f(x) = \left(f_1(x) \ldots f_k(x) \right)$.

Множеството от Парето оптимални (недоминирани) решения в $\Re^k$ формира повърхнина на Парето *P* (фиг. 2.11).



Y – пространство на решенията (допустима област)

P – повърхнина (множество) на Парето

$f_1(x)$, $f_2(x)$ … - целеви функции

f_1^*, f_2^* - оптимално решение на съответната целева функция

F^* - утопично най-добро решение

фиг. 2.11 – Повърхнина на Парето

Приложени са два метода за многопараметрична оптимизация – недоминирано сортиране и Парето оптимално сортиране.

2.4.2.1 Метод на недоминираното сортиране

На всички недоминирани индивиди в популацията се присвоява еднакъв ранг (1). След това тези индивиди се отделят и сред останалите се търси нова група от недоминирани. На тях се присвоява ранг 2 и т.н. до изчерпване на индивидите в популацията (фиг. 2.13). След това ранговете се променят, така че на текущите Парето оптимални решения да стане най-висок и се използва метод на ролетното колело за определяне на родителите.

2.4.2.2 Метод на оптималните решения

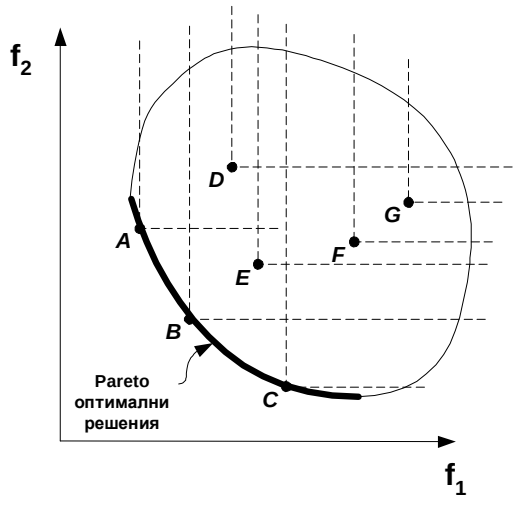
От всички индивиди (хромозоми) в поколението се отделят недоминираните (На фиг. 2.12 това са A, B и C). Те образуват текущата повърхнина на Парето. Тези хромозоми рекомбинират помежду си и създават следващото поколение. Когато се получи само един индивид, доминиращ всички останали, търси се втората граница на Парето и тя се включва в за рекомбинацията. При този метод е възможно попадане в локален минимум, тъй като няколко доминиращи индивида определят следващото поколение. Предимството спрямо недоминираното сортиране е по-бързата селекция.

И при двата метода е предвидена възможност за ограничаване на броя индивиди в повърхнината на Парето. Ако броят на Парето оптималните решения надхвърли предварително дефинирана стойност, изчислява се следната функция на близост между елементите:

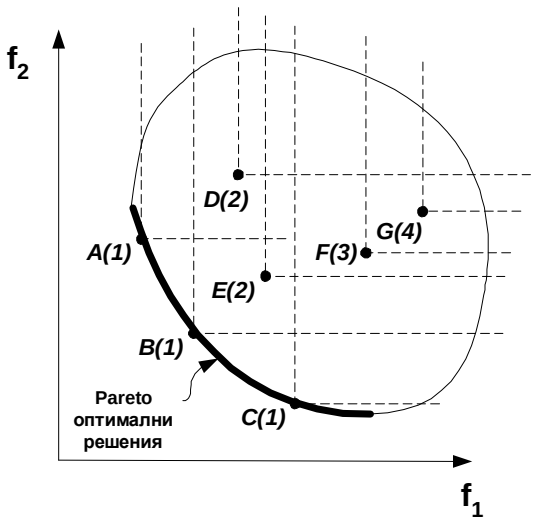
$$D(x) = \frac{\min\|x - x_i\| + \min\|x - x_k\|}{2} \qquad (2\text{-}10)$$

където $x^1 x_i^1 x_k^1 x$ са решения от повърхнината на Парето.

Точката с най-малка стойност на D(x) (най-малко разстояние до съседните точки) се премахва. Този процес продължава до редуциране на точките до желаният брой.



фиг. 2.12 – Метод на оптималните решения



фиг. 2.13 – Метод на недоминираното сортиране

3. Програми за оптимизация

Разработени са общо четири алгоритъма за оптимизация (табл. 3.1). Алгоритмите са разработени на модулен принцип, така че да могат да се добавят или сменят генетични оператори, методи за визуализация и т.н. BC[1] имат възможност за използване на разширено представяне на гените.

	еднокритериална	многокритериална
стандартно кръстосване	**GAminSC**	**GAMOminSC**
със смесване кръстосване	**GAminBC**	**GAMOminBC**

табл. 3.1

3.1 Настройки

Всички алгоритми използват една и съща програма за дефиниране на параметрите на оптимизация: ***GAopt***. Чрез ***GAopt*** могат да се зареждат и запазват дефинирани настройки, както и да са променят отделни настройки. Настройките са:

име	тип	описание
`MaxIter`	целочислен	Брой поколения (итерации), за които да се извърши оптимизацията.
`PopulSize`	целочислен	Брой хромозоми (индивиди) в поколение
`MutatRate`	реален ≥ 0	Процент от хромозомите, които трябва да претърпят мутация на някой от гените си. При използване на разширено представяне на гените (виж 2.3.2) коефициента се мащабира с процентното увеличение на броят на гените.
`BestRate`	реален ∈(0,1)	При еднопараметрична оптимизация това е процент от поколението, което се копира в следващото поколение без изменения (елитарен коефициент), а при многопарамерична оптимизация задава максимален брой индивиди в повърхнината на Парето (като процент от `PopulSize`) (минимална стойност 3 индивида).
`NewRate`	реален ∈(0,1)	Процент от поколението, което се запълва с ново-генерирани хромозоми
`TolX`	реален > 0; 0 -1	Когато това число е > 0 и се работи с някой от BC методите TolX представлява точност по отношение на променливата. (Пр. TolX = 1e-4 означава точност 0,0001). Когато TolX = 0 се работи с числа с плаваща запетая, а при TolX = -1 с целочислени числа.
`pSelect`	реален	Параметър при селекция – зависи от типа на използваната селекция. (Пр. при рангова селекция това е параметъра β)

[1] Blend Crossover – кръстосване със смесване (*GAminBC* и *GAMOminBC*)

`pRecomb`	реален	Параметър при рекомбинация – зависи от типа на рекомбинация (Пр. При кръстосване със смесване това е α)
`Select`	целочислен	Тип на използваната селекция – зависи от използваната функция за минимизация
`RecIter`	целочислен	Интервал от итерации през който да бъде правен запис на текущата най-добра хромозома и стойностите на оптимизираните функции.
`Visual`	'none' 'no' 'some' 'all'	`Visual` е променлива указваща вида на показваният междинен резултат. '*none*' - на екрана не се печати нищо. '*no*' - показва се само съобщение за старт и за финал на програмата. На всеки 20 поколения се печати точка, показваща че програмата все още работи. '*some*' – печати се резултат само при намаляване на целевата функция два пъти спрямо предното печатане и на всеки 25% от максималният брой итерации. '*all*' – на всяка итерация се показва текущият най-добър ген и стойността на функцията (функциите).
`Graphics`	'off' 'on' 'final'	Включване и изключване на графична визуализация на резултатите. При еднокритериална оптимизация се показва графика на стойността на критерия във функция на итерациите, а при многокритериална с 2 или 3 променливи се показва множеството на Парето. '*off*' – няма графична визуализация; '*on*' – графична визуализация само когато и Visual печати информация; '*final*' – графиката се чертае след приключване на оптимизацията
`Comment`	текст	Описание на съответната група от настройки

табл. 3.2

Функцията използва MAT файл, в който са записани 10 предварително дефинирани настройки и могат да се добавят 10 дефинирани от потребителя. Предварително дефинираните са с номера: -9, ... 0, а потребителските с 1 ... 10.

Извикването на запазена настройка става чрез: `options = GAopt (x)`, където `x` е номера на желаната настройка, а options е структура, която се подава на алгоритъма за оптимизация.

Промяната на настройките може да става по два начина:

```
GAopt (options, Parameter1, Value1, Parameter2, Value2, …)
```

options - структура на настройките, която променяме;
Parameter - име на настройката;
Value - нова стойност на параметъра.

или чрез използване на обръщението към елемент на структура (Оператор **.** (точка) в C, C++ и т.н.):

```
options.Parameter = Value;
```

След промяна на желаните настройки новият набор настройки може да бъде записан:

`GAopt(options, x)`, където x е мястото където да се запишат настройките.

Чрез извикване на `GAopt` без входни и изходни аргументи се показват коментарите на запаметените настройки.

3.2 Основни Функции

Четирите основни (***GAminBC, GAminSC, GAMOminBC, GAMOMinSC***) програми имат 3 еднакви входни аргументи.

```
[Result_Genes, Result_Fitness, RecordGenes, RecordFitness] = ...
                    GAminBC ( Function, Bounds, Options )
```

аргумент	предназначение
`Result_Genes`	Хромозома с полученото оптимално решение (при многокритериална оптимизация може да се получат няколко Парето оптимални решения)
`Result_Fitness`	Стойност на целевата функция за Result_Genes
`RecordGenes`	Матрица със записани стойности на най-доброто през определен брой поколения, дефинирани чрез опцията RecIter
`RecordFitness`	Стойност на целевата функция за `RecordGenes`
`Function`	Име на функцията, която при подадени променливи изчислява стойността на целевата функция (или целевите функции)
`Bounds`	Матрица с долните и горни граници за всяка променлива `Bounds = [min max Tol]` min – долна граница max – горна граница Tol – тип на използваната променлива. Възможните типова зависят от типа на кръстосване. Ако не е указан се използва зададеният от опциите тип. (виж. TolX в опциите)
`Options`	Структура с използваните опции. Създава се чрез ***GAopt***. Ако не е указана се използват запазените в ***GAopt(0)*** опции

табл. 3.3

Двете програми използващи стандартно кръстосване (***GAminSC, GAMOMinSC***) имат допълнителен входен аргумент, който може да зададе фиксирана битова маска за кръстосване (виж 2.1.3.1):

```
[Result_Genes, Result_Fitness, RecordGenes, RecordFitness] = ...
                    GAMOminSC ( Function, Bounds, Options, Mask )
```

3.3 Допълнителни функции

3.3.1 Редуциране на броя на Парето решенията

Функцията ***ReducePareto*** намалява броят на Парето оптималните решения, зададен чрез `Max_Number` брой.

```
[New_P_Set, New_P_Fit] = ReducePareto(Pareto_Set, Pareto_Fit,
                         Max_Number)
```

параметър	предназначение
`Pareto_Set`	Хромозоми на Парето множеството
`Pareto_Fit`	Стойности на целевите функции за хромозомите `Pareto_Set`
`Max_Number`	брой на оптималните решения след редукцията
`New_P_Set`	Хромозоми в редуцираното Парето множестово
`New_P_Fit`	Целеви функции на New_P_Set

3.3.2 Номериране на Парето оптималните решения

Функцията ***ParetoNumber*** поставя точки на местата на Парето оптималните решения и по този начин оформя областта. При двукритериална задача функцията поставя до всеки маркер номера му в подадената входна матрица на целевите функции (`Pareto_Fit`).

```
ParetoNumber( Pareto_Fit )
```

При 3 и повече критериална задача това номериране може да стане, ако потребителя подаде само колоните на две от целевите функции като входен аргумент.

4. Примери

4.1 Тригонометрична функция

Да се намери минимум на функцията:

$$f(x) = 5 + x/10 + 5\sin(8x) + 5\cos(3x) \qquad (4\text{-}1)$$

в границите $x \in (0, 20)$. Графиката ѝ е показана на фиг. 4.1

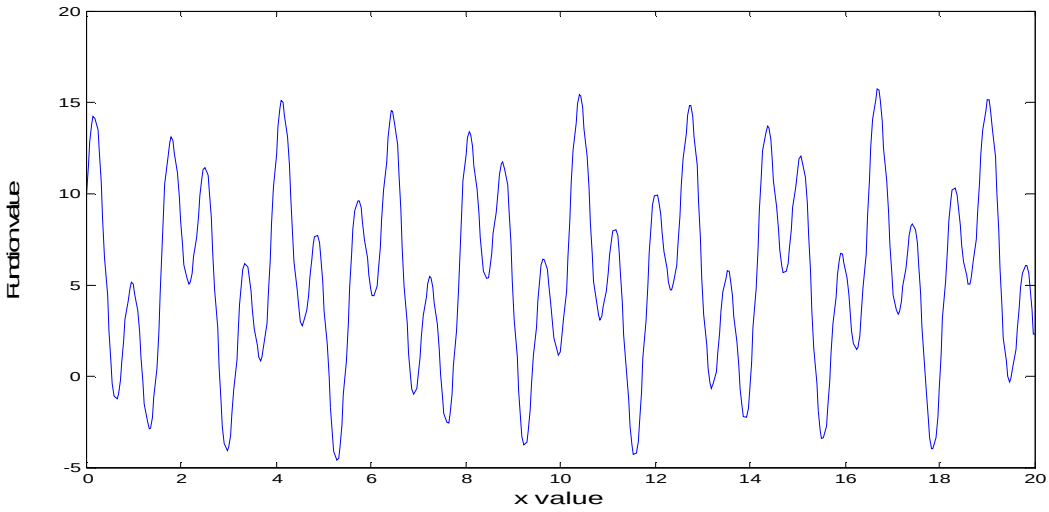
Тук има смисъл да се използва само ***GAminBC***, тъй като с със стандартен кросовър при само една променлива алгоритъма (***GAminSC***) се изражда в еволюционна стратегия (използва са само мутация за достигане до оптимума).

Създава се функцията `eval_sincos` в която се изчислява функцията:

```
function f = eval_sincos(x)

f = 5 + x/20 + 5*sin(8*x)+5*cos(3*x);
```

Зарежда се структурата с настройките:

```
>> opt = GAopt(-2)
opt =
      Comment: ':D Iter.40;   Popul.20;  TolX=1e-4, No Plot'
      MaxIter: 40
    PopulSize: 20
    MutatRate: 0.3000
     BestRate: 0.1000
      NewRate: 0.1000
         TolX: 1.0000e-004
      pSelect: 1
      pRecomb: 0
       Select: 1
      RecIter: 1
       Visual: 'some'
     Graphics: 'off'
>> opt.Graphics = 'final';
```



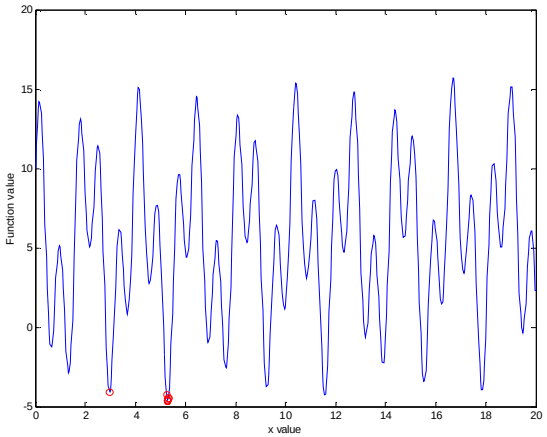
фиг. 4.1 – Графика на тригонометричната функция

Вече може да се пусне оптимизацията.

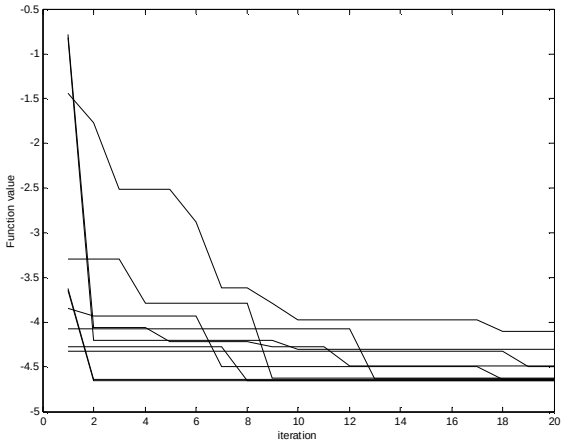
```
>> [RGenes, RFit, RecGenes, RecFit] = GAminBC ( 'eval_sincos', [0 20 1e-
    4], opt);
```

```
GENETIC ALGORITHM for MATLAB, ver 1.0
Minimization of function "eval_sincos"
->Iter:40 Popul:20<-
Started at 14:42:32  29.6.2003

   Iter:      Fit:      Gen:
      1    -4.33242   11.5715
     10    -4.33964   5.3347
     20    -4.56185   5.3154
     30    -4.56185   5.3154
     40    -4.65082   5.2943

Fitness Value: -4.650824

Result Genes:
   5.2943
```

Резултатите от 10 последователни оптимизации, при намаляване на броят поколения до 20, са показани на фиг. 4.2.



а) получени резултати



б) графика на стойността на функционала

фиг. 4.2 – Тригонометрична функция – резултати при 10 оптимизации

4.2 Настройка на непрекъснат ПИД регулатор чрез оптимизационен критерии

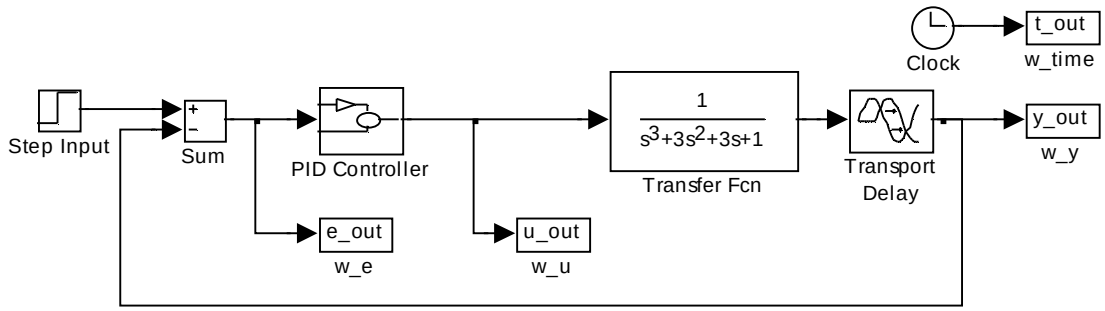
Да се настроят параметрите на ПИД регулатор, управляващ системата, описана с предавателната функция:

$$W(p) = \frac{1}{p^3 + 3p^2 + 3p + 1} e^{-5p} \qquad (4\text{-}2)$$

Затворената система е показана на фиг. 4.3.

Настройката става по оптимизационен критерии, минимизиращ грешката между изхода и заданието, и стойността на управляващият сигнал.

$$\begin{vmatrix} f_1(x) = \int e^2(t)dt \\ f_2(x) = \int U^2(t)dt \end{vmatrix} \qquad U = \begin{cases} u & u > 1 \\ 0 & u \le 1 \end{cases} \qquad (4\text{-}3)$$



фиг. 4.3 – Схема на системата

Функцията изчисляваща $f_1(x)$ и $f_2(x)$ е:

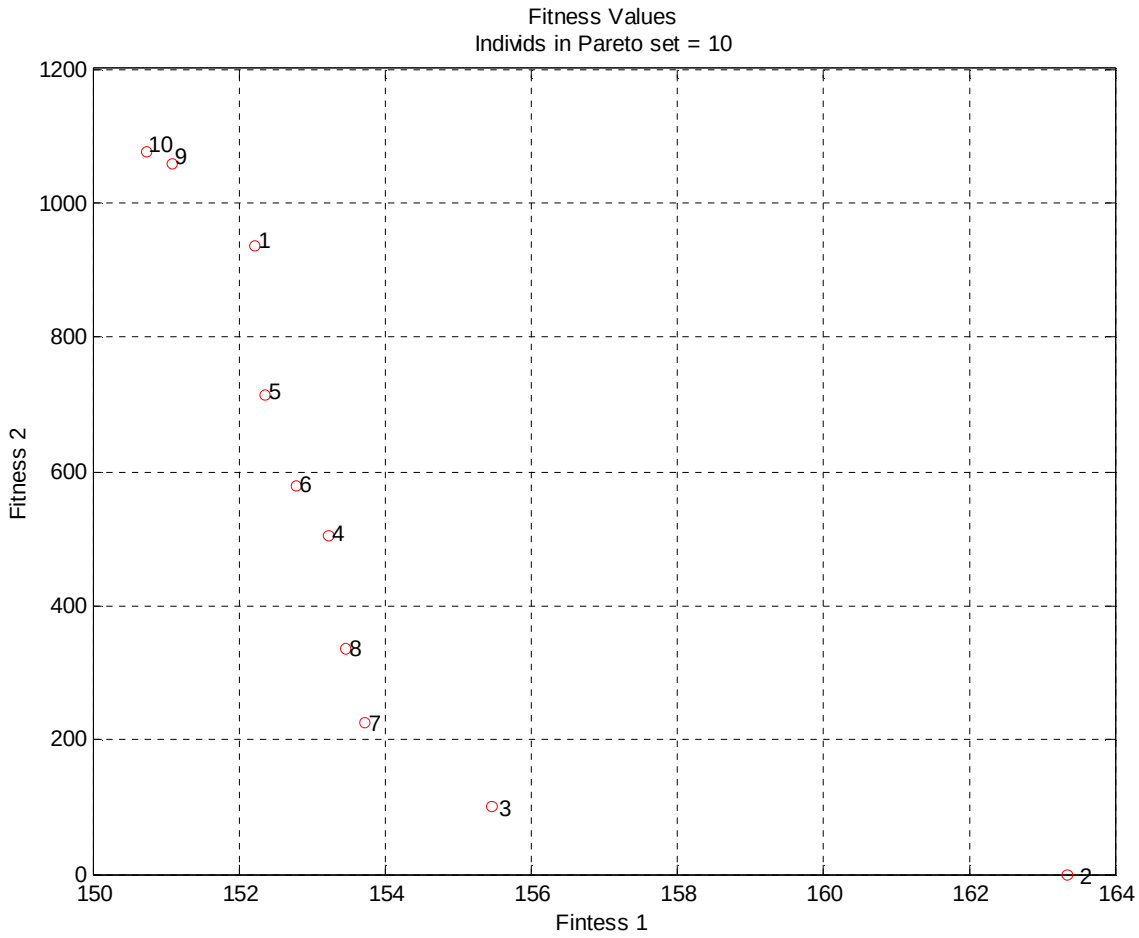
```
function F = eval_reg04(X)

global Kp Ti Td

Kp = X(1);Ti = X(2);Td = X(3);

sim('reg02',100);

F = [];

ti = 0:.05:100;
E_out = INTERP1(t_out, e_out, ti);
U_out = INTERP1(t_out, u_out, ti);

U = U_out > 1; % 1 only when u_out > 1
U_OUT = U.*U_out;

F(1) = E_out*E_out';
F(2) = U_OUT*U_OUT';

return
```

Функцията $f_2(x)$ е пропорционална на стойностите на управляващият сигнал по-големи от 1.

Стартирането на минимизацията се извършва чрез следните команди:

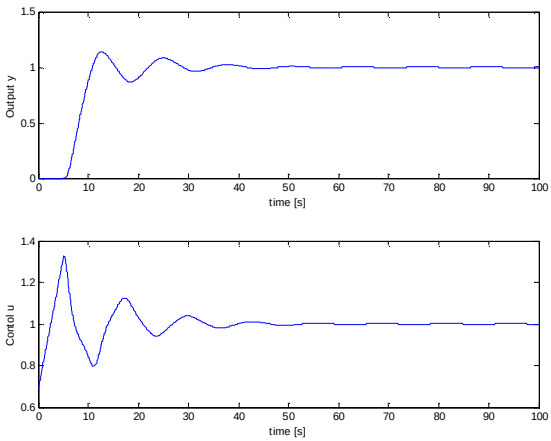
```
opt = GAopt(-5);
opt.MaxIter = 40;
opt.Select = 1;
opt.BestRate = 0.2; %10 Pareto optimal

global Kp Ti Td
x = [0 10 1e-4];
Bounds = [];
for i=1:3
    Bounds = [Bounds; x];
end

[RGenes, RFit, RecGenes, RecFit] = GAMOminBC ( 'eval_reg04', Bounds,
    opt);
```

Формата на Парето повърхнината е дадена на фиг. 4.4, а преходните процеси за точки номера 10, 1, 3 и 2 на фиг. 4.5.

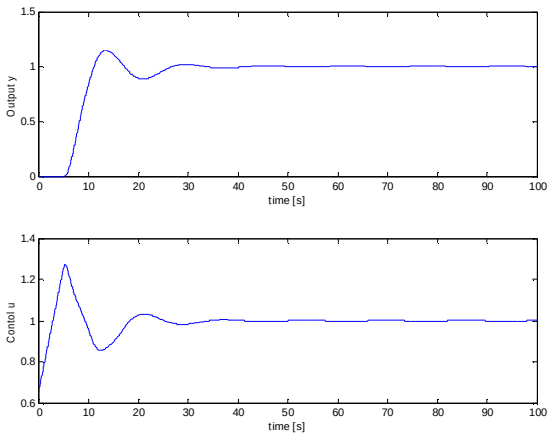


фиг. 4.4 – Повърхнина на Парето

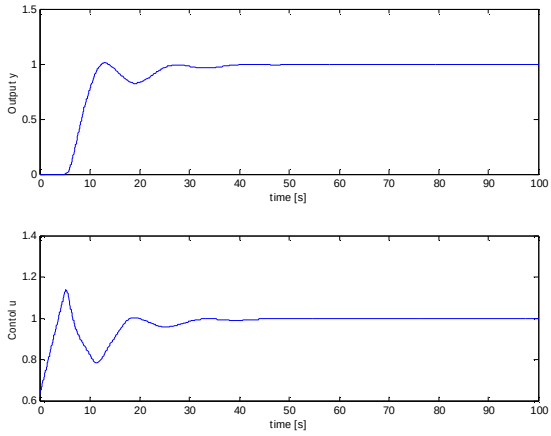
От получените графики ясно се вижда, как с намаляване на необходимият управляващ сигнал преходният процес се променя, и съответно грешката между задание и изход расте. Тези резултати илюстрират предимството на Парето методите – решението се взима апостериорно, а не априорно (избираме какви настройки искаме, като виждаме възможните резултати, а не задаваме предварително тегловни коефициенти, с което да получим единствено решение).



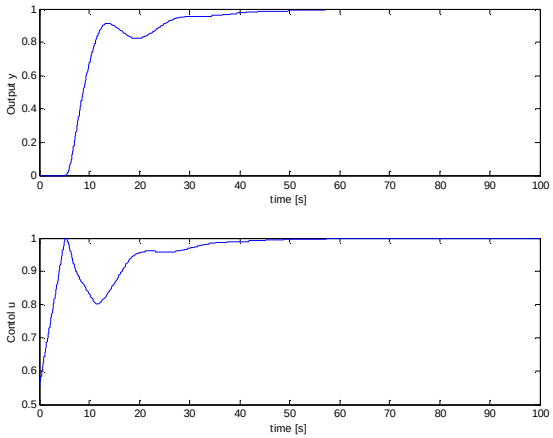
а) хромозома 10



б) хромозома 1



г) хромозома 3



д) хромозома 2

фиг. 4.5 – Преходни процеси

Литературни източници

[1] – Генетични алгоритми за оптимизация – приложение в задачата за синтез на регулатор – Попов. А., дипломна работа, катедра Системи и Управление, факултет Автоматика, Технически Университет – София, 2003

[2] - An Empirical Study of Evolutionary Techniques for Multiobjective Optimization in Engineering Design – Coello C., Department of Computer Science, Tulane University

[3] - The Role of Mutation and Recombination in Evolutionary Algorithms – Spears William, dissertation for Doctor of Philosophy at George Mason University

[4] - A survey of Multiobjective Optimization in Engineering Design - Johan Andersson, Department of Mechanical Engineering, Linköping University, Sweden

[5] - Comparison of Two Multiobjective Optimization Techniques with and within Gentec Algorithms – Azar S, Reynolds B., Narayanan S, Department of Mechanical Engineering, University of Maryland

[6] - PDE: A Pareto–Frontier Differential Evolution Approach for Multi-objective Optimization Problems, Hussein A. Abbass, Sarker R., Newton C., School of Computer Science, University of New South Wales, University College, Canberra, Australia

[7] - An Analysis of Multiobjective Optimization within Genetic Algorithms - Bentley P., Wakefield J., Division of Computing and Control Systems Engineering, The University of Huddersfield The University of Huddersfield

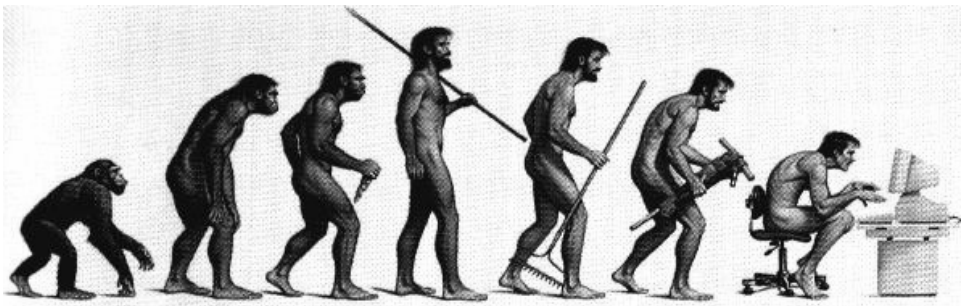
[8] - Wing Design Using Evolutionary Algorithms - Akira Oyama, Department of Aeronautics and Space Engineering of Tohoku University

[9] - Non-linear Goal Programming Using Multiobjective Genetic Algorithms - Kalyanmoy Deb., Kanpur Genetic Algorithms laboratory (KanGAL), Department of Mechanical Engineering, Indian Institute of Technology, Kanpur, India

[10] – Genetic Algorithms Applied to Real Time Multiobjective Optimization Problems - Z. Bingul, A. Sekmen, S. Palaniappan, S. Sabatto, Tennessee State University, Nashville, USA

[11] - A genetic algorithm with adaptable parameters - D. Quagliarella, A. Vicini, C.I.R.A., Centro Italiano Ricerche Aerospaziali, Via Maiorise, Capua, Italy

[12] – Practical Optimization, Gill Ph., Murry W., Wright M., Academic Press, 1981



Генетични Алгоритми за Оптимизация

програми за MATLAB

Андрей Попов

andrey.popov@mail.bg

www.automatics.hit.bg

ТУ – София

2003