ТЕХНИЧЕСКИ УНИВЕРСИТЕТ
СОФИЯ

ФАКУЛТЕТ "АВТОМАТИКА"
КАТЕДРА "СИСТЕМИ И УПРАВЛЕНИЕ"

ДИПЛОМНА РАБОТА

НА ТЕМА

Генетични алгоритми за оптимизация - приложение в задачата за синтез на регулатор

ДИПЛОМАНТ:
Андрей Пламенов Попов
ф.№ 01027005

РЪКОВОДИТЕЛ:
гл. ас. Иван Ценов

Ръководител катедра СУ:
проф. Петко Петков

СОФИЯ

ЮЛИ 2003г.

Увод

В продължение на милиони години живите организми са се развивали на нашата планета. Започнал от простите едноклетъчни организми, минал през различни морски животни и растения и накрая излязъл и на сушата животът е еволюирал. Живите организми са се опитвали да се приспособят към конкретните природните условия и само най-добре подготвените са успявали. Оцелелите видове са създавали потомство, по-добре приспособено от предишното. Настъпвали са катаклизми, при които доскоро не добре приспособени видове са се оказвали добре подготвени, а другите е трябвало или да се приспособят или да изчезнат.

Теорията за еволюцията на видовете е довела, до идеята за копиране на приспособимостта на видовете в природата и използването ѝ за решаване на различни инженерни задачи. Ако всеки индивид се разглежда като възможно решение, а природата като функция определяща кои решения са добри и кои не, лесно може да се направи аналогия с оптимизационните задачи.

По този начин се е появила идеята за еволюционни алгоритми за оптимизация. Оптимизацията, както и еволюцията, е процес на търсене на по-добро решение сред множество от възможни решения. В този смисъл еволюцията се явява оптимизационна задача, търсеща най-добре приспособеният индивид.

Връзката между природата и еволюционните алгоритми е интуитивна:

- Индивидите в природата - възможни решения на задачата
- Природа – функция, чиито оптимум търсим (целева функция)
- Естествен подбор, размножаване и мутация – основни еволюционни операции (аналог на действителните процеси).

Съдържание

Списък на фигурите

1. Въведение в генетичните алгоритми за оптимизация

1.1 Същност и история на генетичните алгоритми

Генетичните алгоритми[1] (ГА) са стохастичен метод за глобално търсене и оптимизация, който имитира еволюцията на живите индивиди, описана от Чарлз Дарвин в “За произхода на видовете и значението на естественият подбор”[2].

Идеята за еволюционни изчисления е била представена през 1960 от I. Rechenberg в неговия труд "Еволюционни стратегии"[3]. Неговите идеи след това са развити от други изследователи. Генетичните алгоритми са представени от John Holland в книгата "Адаптация на естествени и изкуствени системи"[4] публикувана през 1975.

В еволюционните алгоритми се използват трите основни принципа на естествената еволюция, описани от Дарвин: репродукция, естествен подбор и разнообразие на индивидите, поддържано чрез разликите на всяко поколение с предишното. През 60-те години на 20 век тези 3 характеристики на естествената еволюция вдъхновяват европейски и американски изследователи да създадат независими един от друг методи за стохастично търсене:

- Еволюционно програмиране
- Еволюционни стратегии
- Генетични алгоритми

И при трите метода се работи с набор от индивиди. 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.

По-подробно приликите и разликите между тези три еволюционни алгоритми ще бъдат разгледани в точка 1.4.

Многото учени и екипи работещи по проблемите на еволюционните алгоритми, както и голямото разнообразие от структури и настройки на

1 Genetic Algorithms (GA)

2 On the Origin of Species by Means of Natural Selection

3 Evolution strategies (“Evolutionsstrategie” в оригинал на немски)

4 Adaptation in Natural and Artificial Systems.

алгоритмите прави тяхното систематизиране трудна задача. Ето защо тук ще разгледам основните концепции в еволюционните алгоритми, като трябва да отбележа, че са възможни много и различни вариации.

Огромното разнообразие от проблеми, както инженерни така и от други области на познанието, налага използването на алгоритми от различен тип, характеристики и настройки. Тази нужда обяснява голямото разнообразие от еволюционни алгоритми и многото учени работещи върху тях.

1.2 Аналогия между живите организми и генетичните алгоритми

1.2.1 Хромозоми

Всички живи организми се състоят от клетки фиг. 1.1. Едни от органичните съединения изграждащи клетките са нуклеиновите киселини. Те биват два вида: ДНК[1] и РНК[2]. Хроматинът, съдържащ се в ядрото е изграден от ДНК, белтъци и РНК. При делене на клетката той 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. Условно може да се каже, че всеки ген кодира белег (примерно цвят на очите). Всеки ген има своя позиция в хромозомата. Тази позиция се нарича място.

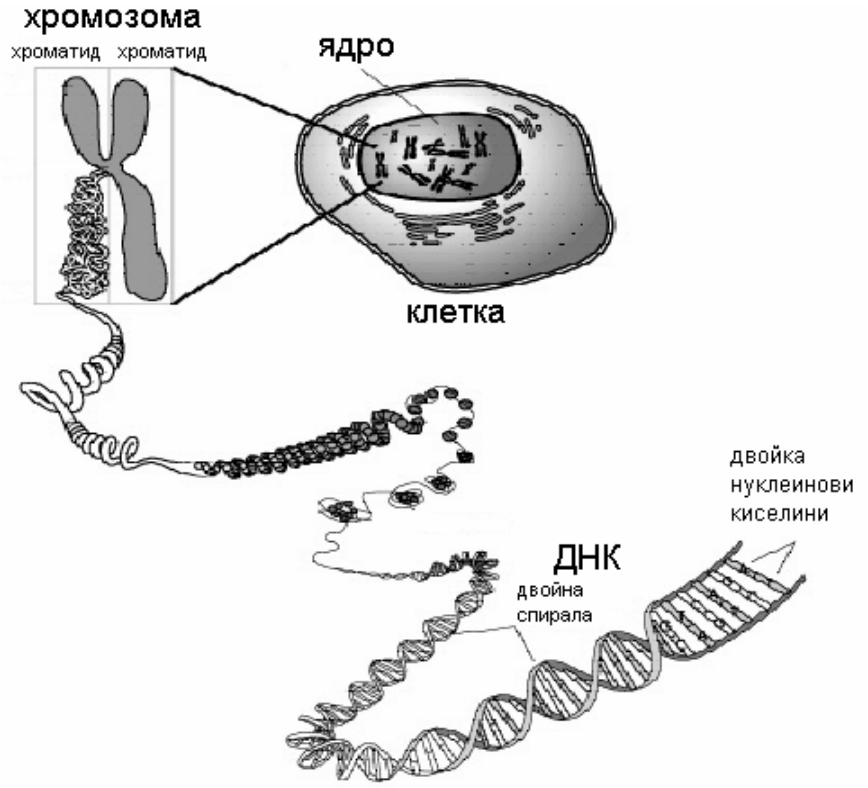
Възможните комбинации за белега (т.е. синьо, кафяво) се наричат алели, а сборът от всички гени в организъма – геном.

Съвкупността от всички наследствени фактори на организма се натича генотип. Това представлява съвкупността от всички локализирани в хромозомите гени на организма.

Съвкупността от проявените белези и свойства на организма се нарича фенотип. Той зависи от наследствените качества (генотипа) и влиянието на средата.

[1] ДНК – дезоксирибонуклеинова киселина

[2] РНК – рибонуклеинова киселина



фиг. 1.1 – Запис на генетичната информация в ДНК

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[1] променливи, както и всяка тяхна комбинация (табл. 1.1).

тип	пример
булев	[1 1 0 1 0 0 0 1 0]
целочислен	[5 73 12 -3 5 104]
плаваща запетая	[5.14 7.21 301.008]
стрингова	[BR AGS tFg K]
комбинация	[17 -24.12 AG 0]

табл. 1.1

[1] стринг – от англ. string – връв, низ – литерални променливи (букви и символи)

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

1.2.2 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[1]) 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 3.1 и 3.2.

1.2.3 Рекомбинация

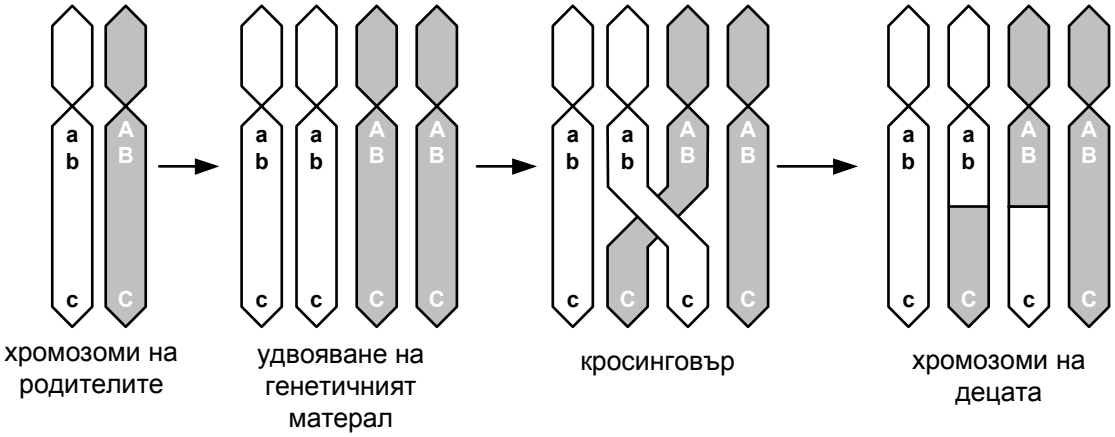
В природата деленето на клетките бива два вида:

- Амитоза (просто) – при него наследственият материал не се разпределя равномерно между дъщерните клетки;
- Митоза (сложно) – при него генетичния материал се удвоява преди деленето, при което двете дъщерни клетки получават еднаква наследствена информация.

При репродукцията, първо се появява рекомбинацията (кръстосване или кросинговър[2]) фиг. 1.2. При нея гените от родителите формират по някакъв начин изцяло нова хромозома.

[1] fitness function

[2] crossover, crossingover – пресичане, кръстосване



фиг. 1.2 – Кръстосване при митоза

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[1] и кръстосване със смесване[2] [7].

1.2.3.1 Стандартно кръстосване

При този тип рекомбинация родителите си разменят съответни гени. Кръстосването може да бъде едноточково или многоточково фиг. 1.3. За рекомбинацията се използва битова маска *Mask*. Уравнението описващо рекомбинацията е:

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

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

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. 1.3 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];

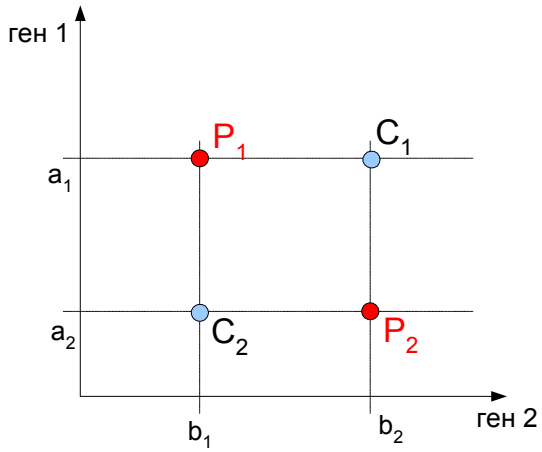
[1] Conventional Crossover

[2] Blend crossover

а) Едноточково кръстосване б) многоточково кръстосване

фиг. 1.3 – Кръстосване с битова маска

Геометричното представяне на този тип кръстосване на хромозома с два гена е показано на фиг. 1.4. Този тип кръстосване (с битова маска) може да се използва при всички по-горе изброени типове гени.



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. 1.4 – Графично представяне на кръстосване с битова маска

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

1.2.3.2 Кръстосване със смесване

Математическото описание на този тип кръстосване е:

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

$$C_2 = (1-\gamma).P_1 + \gamma.P_2$$

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

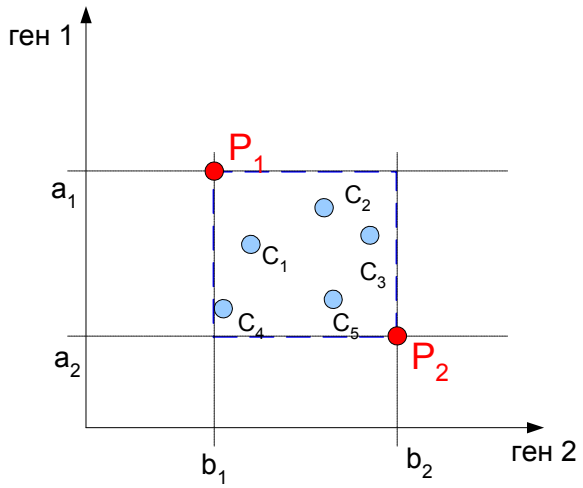
P_1, P_2 – хромозоми на родителите

C_1, C_2 – хромозоми на децата (резултантни индивиди)

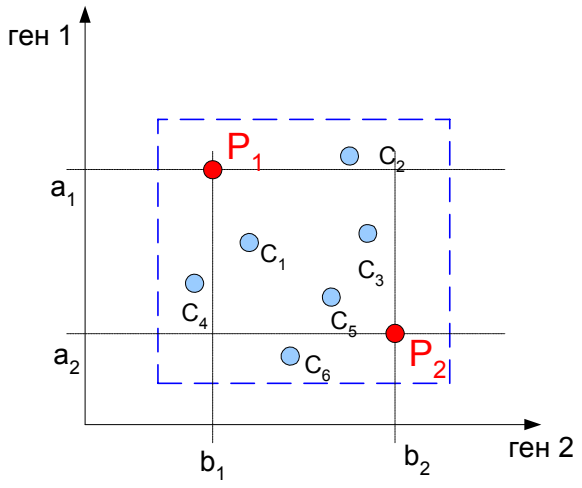
α - коефициент на изследване – задава се от потребителя

r – случайно число в границите (0, 1)

Графичното представяне е показано на фиг. 1.5 и фиг. 1.6.

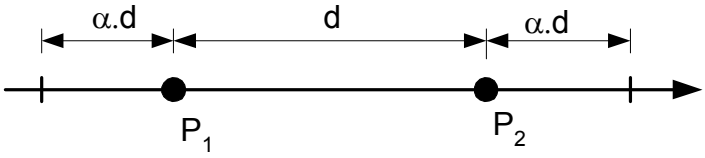


фиг. 1.5 – Графично представяне на кръстосване със смесване $\alpha = 0$



фиг. 1.6 – Графично представяне на кръстосване със смесване $\alpha > 0$

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. 1.7.



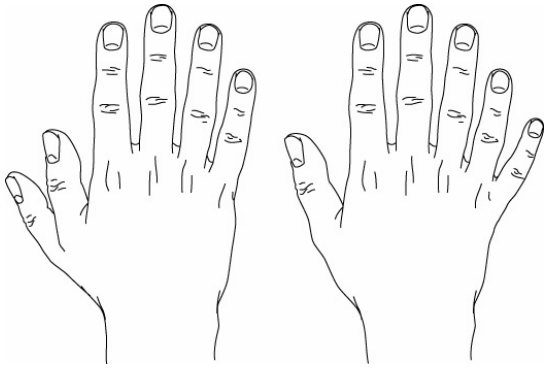
фиг. 1.7 – Промяна на зоната на търсене с промяна на α

В природата по подобен начин се предава информацията за пигментация на кожата, телосложение и т.н.

1.2.4 Мутация

Новосъздаденото чрез селекция и кръстосване потомство след това може да бъде подложено на мутация фиг. 1.8. 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. 1.9 а). Хромозомата, чиито ген ще бъде променен, и мястото на гена също се избират по случаен принцип фиг. 1.9 б).



фиг. 1.8 – Мутация в природата

а) мутация в хромозомата б) места на мутация в поколението

фиг. 1.9 – Мутация при генетичните алгоритми

1.3 Обща схема на еволюционните алгоритми

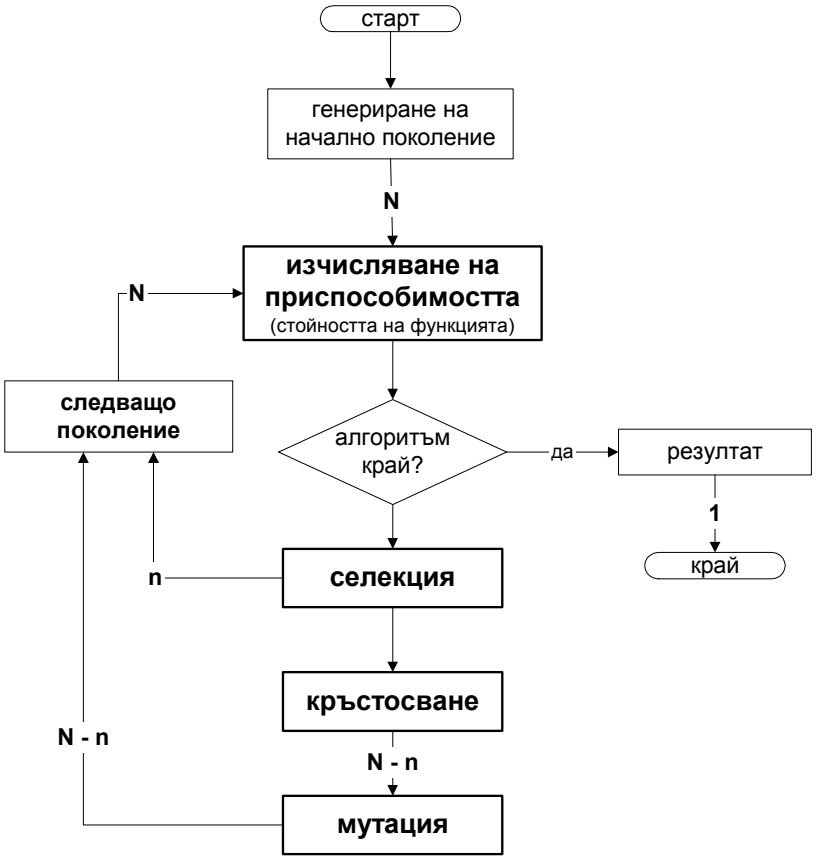
Еволюционните алгоритми поддържат популация от индивиди (хромозоми), които еволюират чрез използване на селекция и други операции, като кръстосване и мутация. Всеки индивид в популацията получава оценка за приспособимостта си (fitness function) към средата. В термините на оптимизацията това означава, 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. 1.10 е показана една от най-често използваните схеми на еволюционните 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).

Пр. Пример за начално поколение от 5 индивида, имащо 4 гени в хромозомата и целочислен тип на гените е:

Generation_0 = [5 6 7 1;
9 0 1 3;
4 1 8 4;
1 2 6 0;
1 3 7 2];



фиг. 1.10 – Обща схема на еволюционните алгоритми

2. Следващата стъпка е изчисляване на стойността на функцията, която минимизираме или максимизираме.

Пр. Ако прием, че стойността на функцията е равна на броя съвпадащи гени с целевият вектор (хромозома) [1 2 3 4], то за горния пример получаваме:

F = [0; 0; 1; 2; 1]

3. Проверка за край на алгоритъма – както при всички алгоритми за оптимизация и тук е възможно алгоритъмът да бъде прекратен по:

- 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;

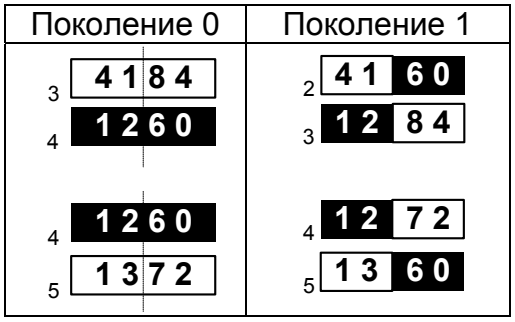
- Достигане на установена стойност – ако в продължение на предварително зададен брой итерации (поколения) не е настъпило подобрение на стойността на функцията алгоритъмът спира.

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).

Пр. За разглежданият пример това би означавало индивид 4 да бъде прехвърлен към следващото поколение, а индивиди 3, 4 и 5 да бъдат кръстосани

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.

Пр. Нека кръстосаме индивиди 3,4 и 5, така че да получим 4 индивида в потомството. Тъй като индивид 4 е с най-голяма стойност на функцията, правим следните кръстоски:



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

Пр. Както се вижда в разглежданият пример, нито един индивид не съдържа в гена си на място 3 стойност 3. Това означава, че колкото и кръстоски да бъдат направени между тези индивиди няма начин да се достигне до желаният вектор [1 2 3 4]. Ако бъде направена мутация, е възможно ген 3 на полученият по-горе индивид 3 да бъде променен от 8 на 3, с което да се максимизира функцията.

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

1.4 Еволюционни алгоритми – видове, прилики и разлики

В [1] се описани подробно трите основни типа еволюционни алгоритми. Тук ще бъдат представени основните им прилики и разлики.

1.4.1 Еволюционно програмиране

Еволюционното програмиране[1] (ЕП), развито от Fogel (1966), традиционно е използвало представяне свързано с представянето на проблема. (Пр. За оптимизационни проблеми, чиито променливи са с реални стойности, индивидите в популациите също са с реални стойности.) Този тип алгоритми често е използван за оптимизиране, въпреки, че първоначалната идея е била създаването на изкуствен интелект.

След създаването на началното поколение, всички *N* индивида се избират за родители и чрез мутация генерират *N* индивида в потомството (деца). За всички тези *2N* индивида се изчислява стойността на приспособимост (стойността на функцията). Избират се *N* индивида, които да създадат следващото поколение и т.н. Най-добрият индивид, винаги оцелява, гарантирайки запазване на достигнатият оптимум. Използваният тип мутация зависи от използваното представяне и често е адаптивен.

В повечето случаи кръстосване (рекомбинация) не се използва, тъй като използваните типове мутация са доста гъвкави и могат да доведат до същите решения, както и кръстосването.

Теоретичните основи на ЕП се базират на доказателството, получено чрез вериги на Марков, за глобалната сходимостта на алгоритмите (с вероятност 1). Използването на елитарен подход (най-добрият индивид винаги оцелява) гарантира, че когато алгоритъмът достигне до област близка до оптимума ще остане в нея, независимо от настъпващите мутации.

1.4.2 Еволюционни стратегии

Еволюционните стратегии[2] (ЕС) се развити от Rechenberg (1973), използвайки селекция, мутация и популация с размер 1. През 1981

[1] Evolutionary Programming

[2] Evolutionary Strategies

Schawefel представя рекомбинацията и използване на популации с по-голям размер. Поради характера на първоначално решаваните проблеми типичните еволюционните стратегии използват векторно представяне на променливи с реални стойности.

След създаване и оценяване на поколението, индивидите се избират с еднаква вероятност да бъдат родители. В стандартните ЕС, двойки родителски индивиди чрез кръстосване създават деца, които след това се подлагат на мутация. Броят на създадените деца е по-голям от броя *N* на родителите. Оцеляването е детерминистичен процес и може да бъде по един от следните два метода:

- N "най-добри" деца оцеляват, замествайки "най-лошите" родители. Нарича се (*N,* λ) оцеляване, където λ е броят на създадени деца.

- Вторият метод се отбелязва с (*N* + λ), което означава, че *N* на брой от най-добрите деца и родители да оцелеят. Тук се използва елитарен подход.

И тук адаптивната мутация играе важна роля, като основният стремеж е бил всяка променлива (респективно ген) да има коефициент на адаптивна мутация с нормално разпределение и нулево математическо очакване. За разлика от ЕП при ЕС кръстосването играе важна роля, особено в адаптивната мутация.

Повечето от теорията на Еволюционните стратегии е свързана с скоростта на сходимост на тези алгоритми, като целта е по-бързото схождане към оптимума. Има доказателство, че алгоритъма (*N* + λ) е глобално сходящ (с вероятност 1).

1.4.3 Генетични алгоритми

Генетичните алгоритми[1] (ГА), разработени от Holland (1975), били традиционно използвани с проблемно независимо представяне на променливите, каквото е булево представяне. Въпреки това много днешни приложения на ГА използват други представяния като вектори с реални променливи, наредени списъци, символни изрази и т.н.

След създаване на поколението родителите се избират според вероятностната функция базирана на относителното приспособеност на индивида. Чрез този тип селекция се взема предвид средната приспособеност на поколението. Индивидите имащи оценка над средната

[1] Genetic Algorithms

участват в създаването (средноаритметично) на повече от едно деца (дъщерни индивиди), а тези с оценка под средната участват в създаването на (средноаритметично) по-малко от един индивида. Това е подходящо нормализирано за създаването на *N* деца чрез кръстосване на *N* родители. След това тези индивиди се подлагат на мутация и формират новото поколение. Този тип селекция не е елитарен и може да се отбележи с (*N, N*).

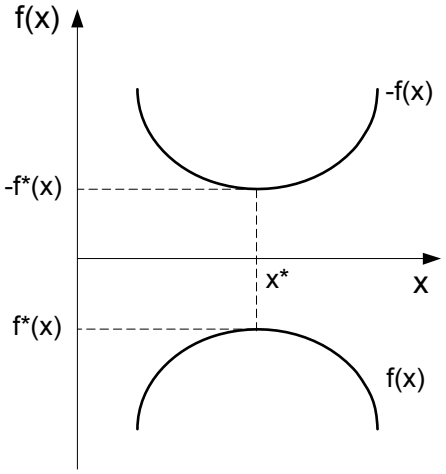
Значението на мутацията и кръстосването при ГА е противоположно на това при ЕП. Исторически погледнато, мутацията е ГА се смята за второстепенна операция, която просто променя алелите[1] с някаква вероятност, а кръстосването е основният оператор осъществяващ търсенето. Въпреки набляганeто на кръстосването, мутацията заема все по-голяма роля в Генетичните алгоритми, отчасти поради влиянието на ЕП и ЕС. Schaffer и Eshelman емпирично доказват, че мутацията е мощен оператор за търсене и заслужава съответното внимание, като същевременно се запазва употребата на рекомбинацията.

[1] Алели – възможните състояния на гена

2. 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).

Тук ще бъдат разгледани само задачи за минимизация (min(F(x))), тъй като задачите за максимизация могат да се представят като задачи за минимизация (фиг. 2.1).



max(F(x)) ⇔ -min(-F(x))

фиг. 2.1 – Преобразуване на задача за max в задача за min

Тук ще бъдат разгледани основните характеристики на генетичните алгоритми, от гледна точка на различните категории оптимизационни алгоритми в които могат да бъдат класифицирани.

2.1 Преки методи

Преки са тези методи за оптимизация, при които се изчислява само стойността на функцията, но не и нейните производни. Преките методи, каквито са генетичните алгоритми са подходящи за решаване на негладки и нелинейни задачи. Това е особено подходящо в задачите за управление, тъй като реалните обекти са нелинейни. За разлика от класическата теория за управление, където обектите се линеаризират, регулаторите се настройват по линеаризираните модели и след това се проверява поведението на системата с регулатора, при използването на генетични алгоритми е възможно да се използва направо нелинейният модел.

2.2 Стохастични методи

Стохастични са тези методи за търсене на решение, при които се използват случайни оператори и няма повторяемост на процеса на намиране на решение. Такъв е например методът Монте-Карло.

Генетичните алгоритми са стохастични методи, поради характера на използваните операции (селекция, рекомбинация и мутация). Въпреки стохастичният характер на търсене чрез вериги на Марков е доказано, че при използване на елитарен подход сходимостта е гарантирана с вероятност 1. Въпреки това достигането до глобалният екстремум може да не стане в рамките на разумно време. Ето защо при всички еволюционни алгоритми основният критерии за спиране е максимален брой от поколения.

Стохастичното търсене налага промени и в метода на сравнение между алгоритми и/или техните настройки. Използват се статистически методи, като за целта минимизацията се пуска няколко пъти и получените резултати се разглеждат статистически.

2.3 Едно- и многопараметрични задачи

В зависимост от броя на оптимизационните параметри задачите биват:

- Еднопараметрични

$$\min_{x \in \Re^1} f(x) \qquad (2\text{-}1)$$

- Многопараметрични

$$\min_{x \in \Re^n} F(x) \qquad (2\text{-}2)$$

Въпреки че стандартните генетичните алгоритми са създадени за многопараметрична оптимизация има модификации позволяващи еднопараметрична 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 това представяне е нелинейното преобразуване на двете пространства. При това е възможно близки в едното пространство стойности да са раздалечени в другото. Пример за това е разликата в представянето на 15

и 16 в десетична и двоична бройна система . В литературата това е известно като дупки на Хеминг[1].

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

табл. 2.1

2.4 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.

$$\min_{x\in\Re^H} F(x) = \left(f_1\left(x\right), f_2\left(x\right), \ldots f_k\left(x\right)\right)^T \qquad (2\text{-}3)$$

Един метод за решаване на многокритериални задачи е свеждането им до еднокритериални. Това може да стане например чрез метод на тегловните функции, но по този начин решението става функция на тегловните коефициенти.

Генетичните алгоритми се оказват подходящи за едновременно минимизиране на няколко критерии, тъй като при тях се работи с набор от решения (поколение), а не с единствено решение ([3]).

2.5 Задачи с ограничения

Много често в практиката, при решаване на оптимизационни задачи, се оказва че се налага поставяне на ограничения върху различни величини. В задачите за управление подобно ограничение се оказва стойността на управляващият сигнал (управляващият орган няма неограничена енергия).

Ограниченията бива от тип равенство и неравенство:

- Ограничение от тип равенство

$c_i(x) = 0$ (2-4)

[1] Hamming cliffs

- Ограничение от тип неравенство

$$c_i(x) \geq 0 \qquad (2\text{-}5)$$

В зависимост от това, дали $c_i(x)$ е линейна или нелинейна функция, то и ограниченията биват линейни и нелинейни.

Най-често при използването на генетични алгоритми за оптимизация ограниченията се задачите чрез метода на наказателните функции (при излизане от ограничение се задава голяма стойност на функцията).

Един тип ограничение, което почти всички разработени генетични алгоритми използват е ограничението върху стойностите на параметрите за оптимизация. Това се налага поради използването на генератор на случайни числа (**rand** в MATLAB) генериращ стойност в интервала (0,1). За да се мащабира и премести тази стойност да съответните величини се налага познаване не долната и горна граница на съответната променлива.

2.6 Реално времеви задачи

Генетичните алгоритми са трудно приложими за реално времеви задачи, тъй като на всяка итерация се налага изчисляване на целевите функции на всеки набор параметри в поколението. Въпреки това в [9] се съобщава за постигнати резултати при замяна на метод Монте-Карло с ГА с адаптивни параметри.

2.7 Адаптивни алгоритми

В зависимост от използваните променливи, често ГА могат да се окажат неефикасни. При използване на булево представяне на 100 променливи с точност до 6 знак хромозомата става с над 2000 гена, а при използване на целочислено представяне на променливите гените могат да заемат безкраен брой състояния.

Един начин за решаване на тези проблеми е използването на адаптивни генетични алгоритми ([7], [9], [10]). Повечето адаптивни алгоритми използват адаптация на параметрите на ГА (размер на поколението, мутация, рекомбинация ([9])) или стесняване на търсената област ([7]). В [10] е предложена структура на ГА, при която се работни не с хромозоми, а с клетки. Всяка клетка съдържа както набор от хромозоми, така и инструкциите, които извършват декодирането им, и параметрите на ГА. В процеса на рекомбинация се променят не само хромозомите, но и параметрите на ГА.

2.8 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). Когато изчисляването на целевите функции е бавен процес, тази master-slave архитектура е оказва много подходяща.

- Разделяне на под-популации – цялото поколение се разделя на под-поколения (популации), всяко едно от които се изчислява на отделен процесор. Тези методи може да са по-робастни от стандартните, тъй като отделните популации са независими и могат да изследват различни области.

- Разпределяне на генетичните операции – този метод се явява като разширение на горният. Генетичните операции се прилагат само между съседни популации, като по този начин се запазва разнообразието на индивидите.

3. Алгоритми за оптимизация

Разработени са алгоритми, както за еднокритериална, така и за многокритериална оптимизация тъй като в инженерната практика често се налага едновременното удовлетворяване на няколко противоречащи си критерии.

Както вече беше обяснено в 1.2.2 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.

Тук са разгледани три метода за селекция описани в литературата и са предложени два нови метода (Гаусова селекция и Парето оптимална селекция), изискващи по-малък брой изчисления. Направено е сравнение с алтернативни методи за решаване на многокритериални задачи.

3.1 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.

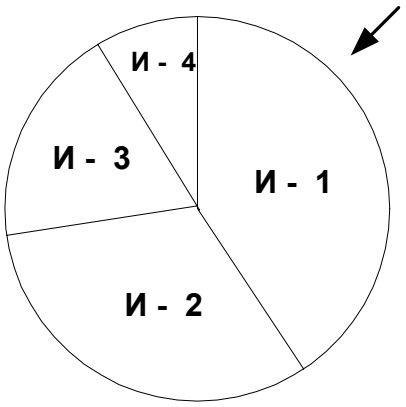
В създадените алгоритми са използвани три основни метода на селекция:

3.1.1 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.

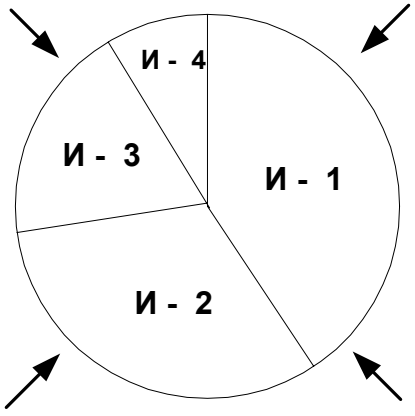
За самият избор на родители се използва метод на ролетното колело. При него окръжността се разделя на *N* на брой сектори пропорционални на вероятността за селекция на индивидите фиг. 3.1. Всеки път когато се завърти ролетното колело[1] се избира един индивид.



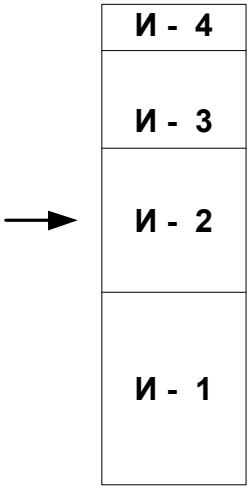
индивид	целева функция	вероятност за селекция
И – 1	4	0.4
И – 2	3	0.3
И – 3	2	0.2
И - 4	1	0.1

фиг. 3.1 – Метод на ролетното колело

За да се запази разнообразието в поколението се в поколението може да се използва модификация, при която при всяко въртене на ролетното колело се избират по няколко родителя (фиг. 3.2).



фиг. 3.2 – Модифициран метод на ролетното колело



фиг. 3.3 – Програмна реализация на ролетното колело

[1] Roulette Wheel Selection

При програмната реализация на метода на ролетното колело се използва линейно представяне на колелото фиг. 3.3. 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[1].

3.1.2 Рангова селекция[2]

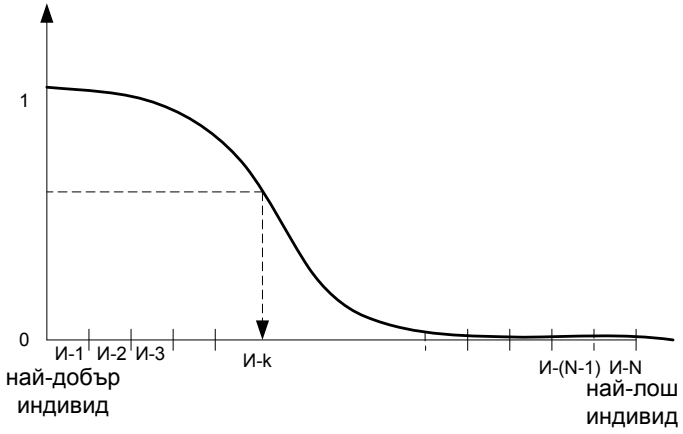
Индивидите се сортират според стойността на целевата функция и след това им се присвоява ранг. Рангът на най-добрият е 1, на следващият 2 и т.н. Вероятността за селекцията на всеки индивид се определя чрез нелинейната функция:

$$P = \beta.(1-\beta)^{(rank-1)} \qquad (3\text{-}1)$$

Където β е зададен от потребителя параметър. За селекцията se използва методът на ролетното колело.

3.1.3 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. 3.4). Тук настройваемият параметър на селекцията е средноквадратичното отклонение.



фиг. 3.4 – Гаусова селекция

[1] Повече на брой случайни числа могат да попаднат в границите му

[2] Ranking Selection

3.2 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 se влошават 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.

$$\min_{x\in\Re^n} F(x) = \left(f_1(x), f_2(x), \ldots f_k(x)\right)^T \qquad (3\text{-}2)$$

Пример за многокритериална задача в задачата за синтез на управление са:

- Да се синтезира регулатор за система с ОВ, така че грешката между входа и изход на системата да е минимална, и управлението да не надхвърля определена стойност (зависеща от възможностите на регулиращият орган);

- Да се синтезира обратна връзка по състояние, така че затворената система да има зададени полюси, като стойностите в матрицата на обратната връзка не могат да надхвърлят определена стойност.

За решаването на тези задачи има няколко основни методи:

3.2.1 Привеждане към еднокритериална задача

3.2.1.1 Обединяване на целевите функции

При тези методи целевите функции се обединяват[1] до един обобщен критерии (целева функция). След това задачата се решава като задача за еднопараметрична оптимизация.

Най-известният метод от тази група е метода на тегловните коефициенти. При него резултантната целева функция се формира като:

$$\min \sum_{j=1}^{k} \lambda_j f_j(x) \qquad (3\text{-}3)$$

където λ_i е теглото на i-тата тегловна функция.

[1] Aggregation methods

Обикновено този метод се използва при синтез на регулатор при минимизиране на целевата функция:

$$J = \frac{1}{2}\int_0^{\infty}\left(x^T Q x + u^T R.u\right)dt \quad Q \geq 0 \quad R \geq 0 \qquad (3\text{-}4)$$

Недостатъкът на този тип методи е, че се изисква предварителна информация за задачата (системата), за да се определят теглата (стойностите на Q и R) и полученото решение зависи от λ. Тъй като обикновено тегловните коефициенти не се знаят точно, а потребителят задава приблизителни стойности е възможно с този метод да се изпуснат решения, при които с несъществено влошаване на една от целевите функции да се подобрят съществено останалите.

3.2.1.2 Метод с ограничения

При този метод *k-1* целеви функции се разглеждат като ограничения, а се оптимизира по една от тях. След намиране на оптимума се сменя целевата функция. Недостатък на този метод е, че отново се получава единствен резултат, който зависи от реда по който са оптимизирани целевите функции.

3.2.2 Методи на Парето

При методите на Парето[1] се получават множество от недоминирани решения, от които потребителя избира това, което най-добре отговаря на нуждите му.

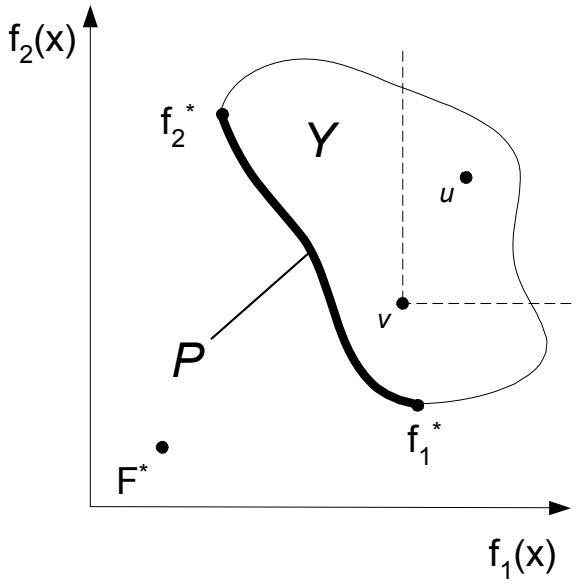
Дефиниция 1 (Доминиране): Векторът *v* доминира *u*, ако:

$$\forall i \in \{1,2...k\} : f_i(v) \leq f_i(u) \text{ и } \exists j \in \{1,2...k\} : f_j(v) < f_j(u) \quad (3\text{-}5)$$

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

Множеството от Парето оптимални (недоминирани) решения в $\Re^k$ формира повърхнина на Парето *P* (фиг. 3.5).

[1] Pareto



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. 3.5 – Повърхнина на Парето

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

3.2.2.1 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. 3.7). 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.

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:

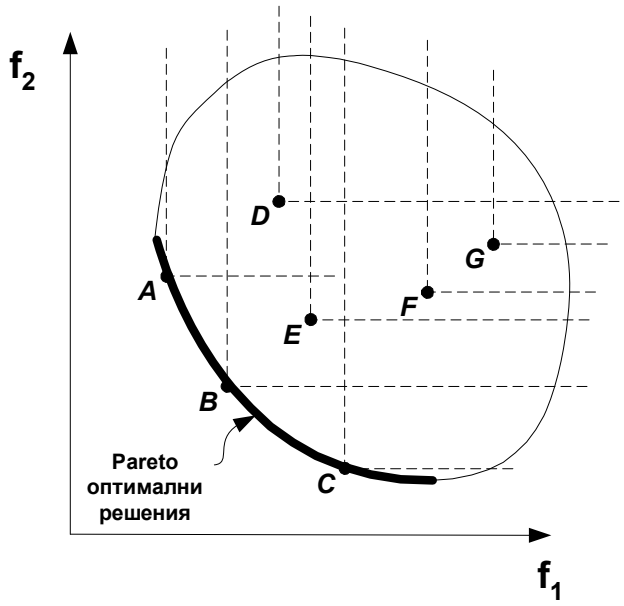
$$D(x) = \frac{\min\|x - x_i\| + \min\|x - x_k\|}{2} \qquad (3\text{-}6)$$

където $x \neq x_i \neq x_k \neq x$ са решения от повърхнината на Парето.

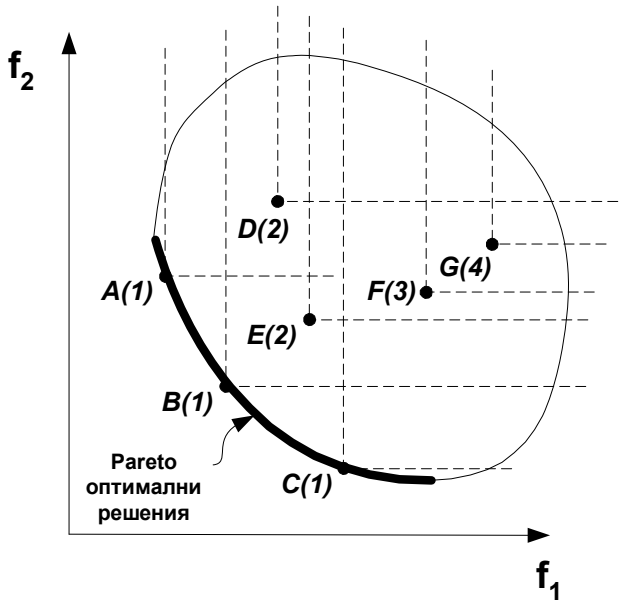
Точката с най-малка стойност на D(x) (най-малко разстояние до съседните точки) се премахва. Този процес продължава до редуциране на точките до желаният брой.

3.2.2.2 Метод на оптималните решения

От всички индивиди (хромозоми) в поколението се отделят недоминираните (На фиг. 3.6 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 на този метод е по-малкият брой изчисления.



фиг. 3.6 – Метод на оптималните решения



фиг. 3.7 – Метод на недоминираното сортиране

4. Програми за оптимизация

Разработени са общо четири алгоритъма за оптимизация (табл. 4.1). Алгоритмите са разработени на модулен принцип, така че да могат да се добавят или сменят генетични оператори, методи за визуализация и др.

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

табл. 4.1

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

Програмите са създадени върху MATLAB 6 (R 12). Всеки от четирите алгоритми има собствена програма за стартиране (4.2). Тези четири програми използват различни функции (общи функции) описани в 4.3. Потребителят може да използва тези функции за обработка на получените резултати, съставяне на собствени типове алгоритми и т.н.

Съществува трета група функции, наречена допълнителни функции. Това са функции, които не се използват директно от нито един от четирите алгоритъма, но са необходими за предварителна (настройки за оптимизация) и финална обработка (номериране на решенията).

Алгоритмите с кръстосване тип смесване могат да работят с разширено представяне на гените, описано подробно в 2.3.

Кодът на програмите е даден в Приложения А – В (табл. 4.2).

Приложение А	Основни програмни модули на четирите алгоритъма
Приложение Б	Общи функции използвани при оптимизацията
Приложение В	Допълнителни функции и модули

табл. 4.2

4.1 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. Възможните параметри за настройка 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 в поколението, които трябва да претърпят мутация на някой от гените си. При използване на разширено представяне на гените този коефициент се мащабира с процентното увеличаване на гените.
`BestRate`	реален $\in(0,1)$	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 $\in(0,1)$	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[1] 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)
`pRecomb`	реален	Параметър при рекомбинация – зависи от типа на рекомбинация (Пр. При кръстосване със смесване това е α)

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

`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. 4.3

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[1]:

```
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.

Кодът на програмата се намира в Приложение В.

4.2 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]`

[1] Оператор . (точка) в C, C++, …

	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. 4.4

Двете програми използващи стандартно кръстосване (***GAminSC, GAMOMinSC***) имат допълнителен входен аргумент, който може да зададе фиксирана битова маска за кръстосване (виж 1.2.3.1):

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

Кодът на тези програми е даден в Приложение А.

4.3 Общи функции

Кодът на тук разгледаните функции е даден в Приложение Б.

4.3.1 Комбиниране на настройките

Функцията ***CombineOpt*** комбинира зададените от потребителя настройки (`User_options`) със стандартните такива (`Default_options`)(тези записани в масива от настройки под номер 0 - ***GAopt(0)***). Ако в зададените от потребителя настройки липсва някое поле, то се взема от стандартните настройки.

```
result_Options = CombineOpt (User_options, Default_options)
```

4.3.2 Преобразуване на гените

С тези функции се формира разширена матрица на ограниченията. При стандартно кръстосване се добавя само разликата между долна и горна граница, така че да се спести това изчисляване в самите генетични оператори. При използване на разширено представяне на гените за всеки под-ген се записва долна и горна граница, степента на 10, по която трябва да бъде мащабиран и тип на променливата. Функциите използвани съответно при стандартно кръстосване и кръстосван със смесване са: ***TransfGenesSC*** и ***TransfGenesBC***.

При използване на разширено представяне на гените се налага обратното преобразуване на гените, за да може да бъде изчислена целевата функция, да бъдат показани междинни резултати и изведен краен резултат (резултати). За целта се използва функцията ***GenesToStandart***, на която се подават хромозомите за преобразуване и разширената матрица на границите.

4.3.3 Създаване на хромозоми

Всичките програми използват случайно генерирано начално поколение. В зависимост от вида на селекцията – стандартна или с кръстосване се използват съответно ***GARandBC*** и ***GARandSC***. При ***GARandBC*** се налага провеждане на проверка, за да се провери дали оригиналният (не разширен) ген е в поставените ограничения. Ако това не е изпълнено той се модифицира и се прави отново проверка.

Двете функции получават като входни данни матрицата на ограниченията и желаният брой от индивиди, които трябва да бъдат създадени (`Num_Ind`) и връщат създадените хромозоми.

```
FPopul =  GARandBC ( Bounds_Ext, Bounds_Stn, Num_Ind )
```

и съответно

```
FPopul =  GARandSC ( Bounds, Num_Ind )
```

4.3.4 Функции за сортиране

4.3.4.1 Еднокритериална сортиране

В еднокритериалните задачи преди сортиране на хромозомите се формира матрица, чиято първа колона са стойностите на целевата функция, а в останалите са записани гените на съответната хромозома. Сортирането става по стойностите на целевата функция (първата колона) чрез командата `sortrows`.

4.3.4.2 Многокритериално сортиране

В многокритериалните задачи се използва функцията ***sortPareto***. На функцията се подават хромозомите в текущото поколение (`Popul_Set`), стойностите на целевите им функции (`Popul_Fit`), както и текущите хромозоми и целевите им функции от множеството на Парето (`iPareto_Set, iParetoFit`). В функцията се прави проверка за всяка хромозома от текущото поколение, дали е доминирана и дали е доминираща. Ако съответната хромозома е недоминирана тя се добавя към множеството на Парето. Ако целевите функции на хромозомата са по-

добри от тези на индивидите в повърхнината на Парето (хромозомата е доминираща) тя става единствен член на множестовото на Парето (тя доминира всички досегашни решения).

```
[Pareto_Set, Pareto_Fit, NPareto_Set, NPareto_Fit]  = ...
     sortPareto(Popul_Set, Popul_Fit, iPareto_Set, iParetoFit)
```

Функцията връща както новото множество на Парето (`Pareto_Set, Pareto_Fit`), така и доминираните индивиди (`NPareto_Set, NPareto_Fit`).

4.3.5 Функции за селекция

За селекция на индивидите, които да участват в рекомбинация и мутация се използват алгоритмите описани в глава 1. Възможните методи за селекция зависят от използваният алгоритъм за оптимизация. Потребителят може да зададе желанието си чрез параметърът за настройка `Select`.

4.3.5.1 Еднокритериална селекция

Типовете селекция и използваните функции са дадени в табл. 4.5.

Тип селекция	Функция
Пропорционална на целевата функция селекция (3.1.1)	***GASelectFP***
Рангова селекция (3.1.2)	***GASelectRS***
Гаусова селекция (3.1.3)	***GASelectN***

табл. 4.5

На ***GASelectFP*** като входни параметри се подават стойностите на целевата функция на вече подредените хромозоми (`Fitness`) и броят на индивиди в, които трябва да се получат чрез рекомбинация[1] (Num) . Функцията връща матрица *m* x *2.* Всеки ред представлява двойка с поредните номера на родители, които да рекомбинират (`Parents`).

```
Parents = GASelectFP ( Fitness, Num )
```

При използване на другите два типа селекция не е необходимо да се подават стойностите на целевата функция, тъй като при тях единственото изискване е хромозомите да бъдат предварително подредени. Извикването им е съответно:

```
Parents = GASelectN ( populSize, Num, StDev )
```

[1] възможно е част от поколението да бъде запълнено с новосъздадени хромозими

```
Parents = GASelectRS ( populSize, Num, b )
```

където:

`populSize` – брой на хромозомите, които могат да станат родители (размер на поколението);

`StDev` – средноквадратично отклонение;

b – коефициент β при рангова селекция.

Функции ***GASelectFP*** и ***GASelectRS*** използват функцията ***Roulette***, в която се осъществява самото въртене на ролетното колело. На функцията се подава случайно число в границите от 0 до сумата от всички целеви функции (`randN`), стойностите на целевите функции (`Fitness`) и общият брой на родителите (`Num_Par`). Като резултат функцията връща номер на избраният родител.

```
ParentNum = Roulette ( randN, Fitness, Num_Par )
```

4.3.5.2 Многокритериална селекция

Използваните типове селекция (табл. 4.6) са подробно описани в 3.2.2.

Тип селекция	Функция
Метод на оптималните решения (3.2.2.2)	***GAParetoOpt***
Метод на недоминираното сортиране (3.2.2.1)	***GANDominSort***

табл. 4.6

И двете функции имат едни и същи входни и изходни променливи:

```
[SelParents, Pareto_Set_total] = GANDominSort (Pareto_Set,
            Pareto_Fit, Non_Pareto_Set, Non_Pareto_Fit, populSize)
```

където:

`SelParents`	матрица *m* x *2* с двойки хромозоми избрани за рекомбинация
`Pareto_Set_total`	матрица със всички хромозоми в текущото поколение, спрямо която са номерирани родителите в `SelParents`
`Pareto_Set`	текущи Парето оптимални решения
`Pareto_Fit`	целеви функции на `Pareto_Set`
`Non_Pareto_Set`	хромозоми, които са доминирани
`Non_Pareto_Fit`	целеви функции на `Non_Pareto_Set`
`populSize`	брой на индивидите в поколение (определя броя

родителски двойки в `SelParents`)

Функцията ***ReducePareto*** намалява броят на Парето оптималните решения, до дефинираният чрез параметър `BestRate` брой (`Max_Number`).

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

където: `New_P_Set` и `New_P_Fit` са съответно редуцираното множество на Парето и целевите функции на тези индивиди.

4.3.6 Рекомбинация

4.3.6.1 Стандартно кръстосване

Използваната функция е ***GACrossSC***[1].

```
FPopul = GACrossSC ( SelPar, IPopul, Num, Mask )
```

Входните и изходни параметри са дадени в табл. 4.7.

параметър	предназначение
`FPopul`	Хромозоми получени в резултат от кръстосването
`SelPar`	Матрица с избраните родителски хромозоми. Получава се с някоя от функциите за селекция.
`IPopul`	Хромозоми на родителите
`Num`	Брой хромозоми, които трябва да се получат
`Mask`	Маска на кръстосването зададена от потребителя. Ако не е зададена се използва случайна маска

табл. 4.7

4.3.6.2 Кръстосване със смесване

Функцията ***GARecombBC*** работи по описаният в 1.2.3.2 алгоритъм.

```
FPopul =  GARecombBC ( SelPar, IPopul, Num, VarType, Alfa )
```

Входните и изходните променливи съвпадат с тези на ***GACrossSC*** с изключение на:

`VarType` – тип на променливите (при променливи с плаваща запетая резултата не се закръглява).

[1] Алгоритъмът е обяснен в 1.2.3.1

Alfa - коефициент на изследване (α). Задава се чрез настройката pSelect. Препоръчва се използване на стойности под 1, тъй като в програмата не се гарантира допустимост на решението за α>1. Въпреки, че в повечето случай програмата ще работи нормално, възможно е зацикляне на алгоритъма, дължащо се на програмата за мутация (тя работи само с допустими решения и продължава да променя избраният ген, до достигане на допустимо решение).

4.3.7 *Мутация*

В зависимост от използваният тип кръстосване, стандартно или със смесване, се използва ***GAMutatSC*** и ***GAMutatBC***. Като входни аргументи функциите получават текущото поколение (IPopul), матрицата (матриците) с границите и броят на мутациите, които трябва да бъдат направени (Num_Mut). Функциите връщат мутиралото поколение.

```
FPopul = GAMutatBC ( IPopul, Bounds_Ext, Bounds_Stn, Num_Mut )
```

и

```
FPopul = GAMutatSC ( IPopul, Bounds, Num_Mut )
```

4.3.8 *Визуализации*

Тези функции се използват за извеждане на различни междинни и крайни резултати.

4.3.8.1 Определяне на визуализацията

Тези функции (***VisualSO*** и ***VisualMO***) са отделени основните модули с цел по-голяма нагледност. Като входни аргументи получават всички необходими променливи за определяне дали и какъв тип визуализация трябва да бъде направен.

4.3.8.2 Текстова визуализация

DispResultsSO и ***DispResultsMO*** служат за извеждане на информацията за номер на итерацията, стойност на целевата функция (функции) и съответните стойности на гените в текстов вид, съответно при едно- и многопараметрична оптимизация. При многокритериална оптимизация, когато настройката Visual е ‘all’ се извежда информация за всички индивиди в повърхнината на Парето. При Visual = ‘some’ се извежда информация само за първият подред индивид от повърхнината.

```
DispResultsSO( iteration, FitnessValue, GenesSet )
```

```
DispResultsMO( iteration, Pareto_Fit, Pareto_Set, Flag )
```

Параметърът Flag оказва какъв е типа на Visual.

4.3.8.3 Графична визуализация

Този тип визуализация работи при стойност на `Visual` различна от ‘none’.

В еднокритериалният случай се визуализира промяната на целевата функция на най-доброто решение във функция на итерациите

```
PlotResultsSO( iteration, FigHandle, RecordFitness )
```

Където:

`iteration` – номер на итерацията

`FigHandle` – указател към фигурата, в която се извършва визуализацията

`RecordFitness` – масив с стойността на най-доброто решение на всяка итерация.

При многокритериална оптимизация се използва ***PlotResultsMO***, която при 2 и 3 целеви функции поставя точки на местата на Парето оптималните решения и по този начин оформя областта на Парето.

```
PlotResultsMO ( iteration, FigHandle, Pareto_Fit)
```

Създадена е още една функция за визуализация, която потребителя може да използва след завършване на процеса на многокритериална оптимизация. На функцията ParetoNumber като входни данни се подават целевите функции на част или всички елементи от множеството на Парето. Функцията прави същото както ***PlotResultsMO***, но при двукритериална задача до всеки маркер поставя номера му в така подадената входна матрица.

```
ParetoNumber( Pareto_Fit )
```

Кодът на програмата е даден в Приложение В.

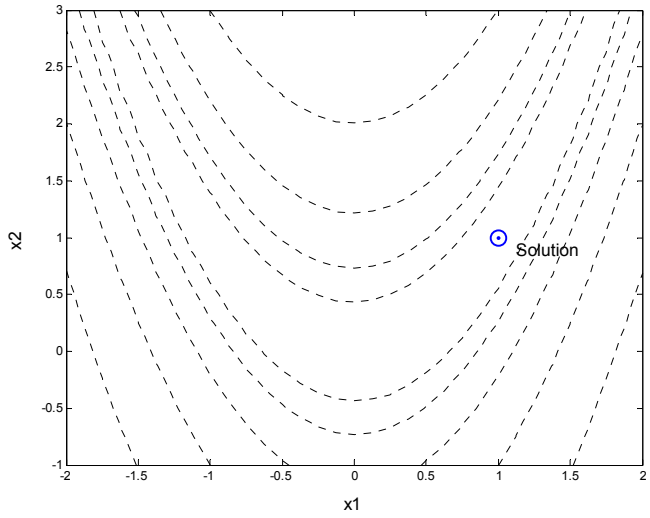
4.4 Илюстративни примери

4.4.1 Пример 1 - Функция на Розенброг

Търси се минимума на функцията на Розенброг (банановидна функция):

$$\min f(x) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2 \qquad (4\text{-}1)$$

Тази функция има единствен минимум в x_1=1; x_2=1. Линиите на еднакво ниво на област близка до решението са показани на фиг. 4.1.



фиг. 4.1 – Функция на Розенброг – линии на еднакво ниво

Използва се ***GAminSC***. Съставя се програмата `eval_Rosen`, която при подадени x_1 и x_2 ще изчислява стойността на функцията:

```
function f = eval_Rosen(x)
% Evaluation of Rosenbrock's function
x1=x(1);
x2=x(2);

f = 100*(x2-x1^2)^2 + (1-x1)^2 ;
```

Зареждаме опциите:

```
>> opt = GAopt(-3)
opt =
      Comment: ':D Iter.100;  Popul.20;  TolX=1e-2; No Plot'
      MaxIter: 100
    PopulSize: 20
    MutatRate: 0.30000000000000
     BestRate: 0.10000000000000
      NewRate: 0.10000000000000
         TolX: 0.01000000000000
      pSelect: 1
      pRecomb: 0
       Select: 1
      RecIter: 1
       Visual: 'some'
     Graphics: 'off'
```

За да има графика в края на оптимизацията:

```
>> opt.Graphics = 'final'
opt =
      Comment: ':D Iter.100;  Popul.20;  TolX=1e-2; No Plot'
      MaxIter: 100
    PopulSize: 20
    MutatRate: 0.30000000000000
```

```
     BestRate: 0.10000000000000
      NewRate: 0.10000000000000
         TolX: 0.01000000000000
      pSelect: 1
      pRecomb: 0
       Select: 1
      RecIter: 1
       Visual: 'some'
     Graphics: 'final'
```

Задава се матрицата с границите и типът на променливите

```
>> Bounds = [-10 10 0; -10 10 0]
Bounds =
   -10    10     0
   -10    10     0
```

Вече може да се пусне самата оптимизация. Тъй като задачата има само две променливи (два гена) може да се зададе и маската на кръстосване:

```
>> [RGenes, RFit, RecGenes, RecFit] = GAminSC ( 'eval_rosen',
    Bounds, opt, [0 1] );

 GENETIC ALGORITHM for MATLAB, ver 1.0
 Minimization of function "eval_rosen"
 ->Iter:100 Popul:20<-
 Started at 13:32:22  29.6.2003

    Iter:      Fit:      Gen:
       1     291.614   -2.59725   8.41507
       4     3.38361   0.38735   -0.0234034
       7    0.961485   0.497096   0.331281
      19    0.276411   0.497096   0.231775
      20    0.276411   0.497096   0.231775
      40    0.276411   0.497096   0.231775
      60    0.276411   0.497096   0.231775
      80    0.276411   0.497096   0.231775
     100    0.276411   0.497096   0.231775

 Fitness Value: 0.276411

 Result Genes:
   0.49709552858896   0.23177473712632
```

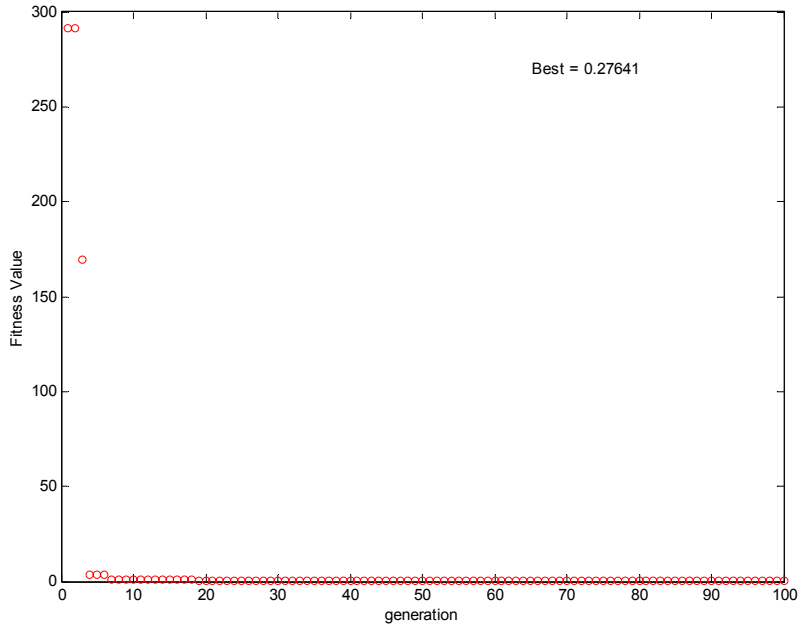
Графиката на функцията е показана на фиг. 4.2.

Нека се използва кръстосване със смесване:

```
>> [RGenes, RFit, RecGenes, RecFit] = GAminBC ( 'eval_rosen',
    Bounds, opt);

 GENETIC ALGORITHM for MATLAB, ver 1.0
 Minimization of function "eval_rosen"
 ->Iter:100 Popul:20<-
 Started at 13:33:16  29.6.2003

    Iter:      Fit:      Gen:
```

```
     1     57.3563   -0.87301   0.0283344
     3     13.3531    2.13656    4.91219
    16     2.27336    0.41196    0.308548
    24    0.430395    0.360315   0.144387
    25    0.430395    0.360315   0.144387
    42   0.0100134    1.09836    1.20823
    50   0.0100134    1.09836    1.20823
    75   0.0100134    1.09836    1.20823
   100   0.0100134    1.09836    1.20823

 Fitness Value: 0.010013

 Result Genes:
   1.09835956042529   1.20823447581827
```

Графиката на функцията е показана на фиг. 4.3.



фиг. 4.2 – Резултат при Розенброг и GAminSC

За използването на разширено гени и точност 10^{-2} се налага промяна на матрицата на границите:

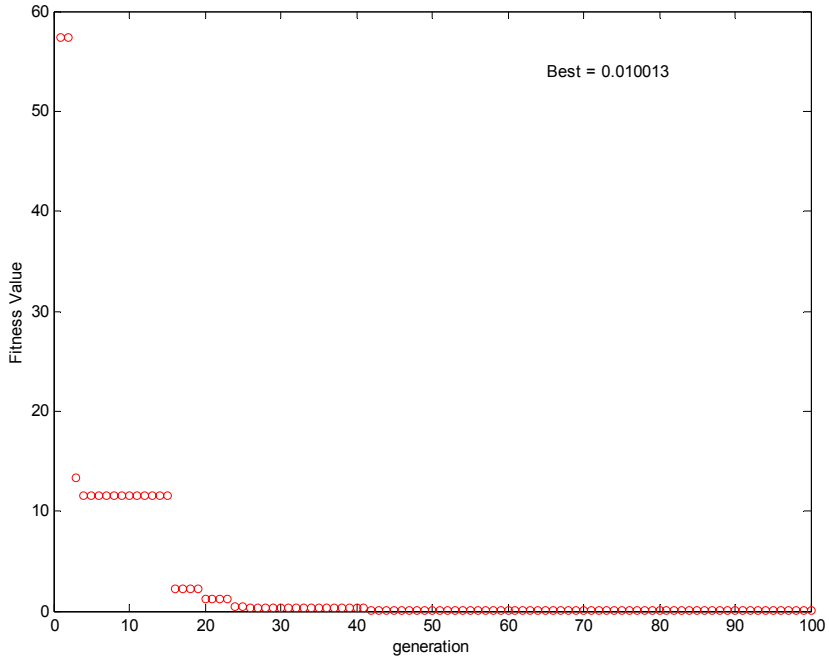
```
>> Bounds = [-10 10 1e-3; -10 10 1e-3]
Bounds =
  -10.0000   10.0000    0.0010
  -10.0000   10.0000    0.0010
```

Стартира се оптимизацията:

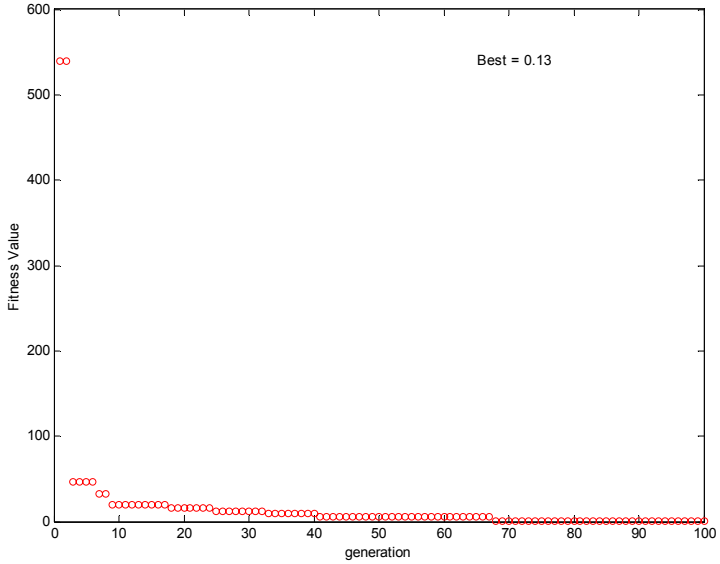
```
>> [RGenes, RFit, RecGenes, RecFit] = GAminBC ( 'eval_rosen',
    Bounds, opt);

GENETIC ALGORITHM for MATLAB, ver 1.0
 Minimization of function "eval_rosen"
 ->Iter:100 Popul:20<-
```

```
 Started at 13:46:29  29.6.2003

    Iter:       Fit:       Gen:
       1      539.963     1.75     0.74
       3       46.936    -2.64     7.55
       9      19.5331     2.45     6.42
      25      11.7152     2.47     6.41
      41      5.37813     2.55     6.33
      50      5.37813     2.55     6.33
      68         0.13      1.3     1.67
      75         0.13      1.3     1.67
     100         0.13      1.3     1.67

Fitness Value: 0.130000

Result Genes:
  1.3000     1.6700
```

Графиката е показана на фиг. 4.4.



фиг. 4.3 – Резултат при Розенброг и GAminBC (с плаваща запетая)



фиг. 4.4 - Резултат при Розенброг и GAminBC (точност 10^{-2})

4.4.2 *Пример 2 – Тригонометрична функция*

Да се намери минимум на функцията:

$$f(x) = 5 + x/10 + 5\sin(8x) + 5\cos(3x) \qquad (4\text{-}2)$$

в границите $x \in (0, 20)$. Графиката й е показана на фиг. 4.5

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).

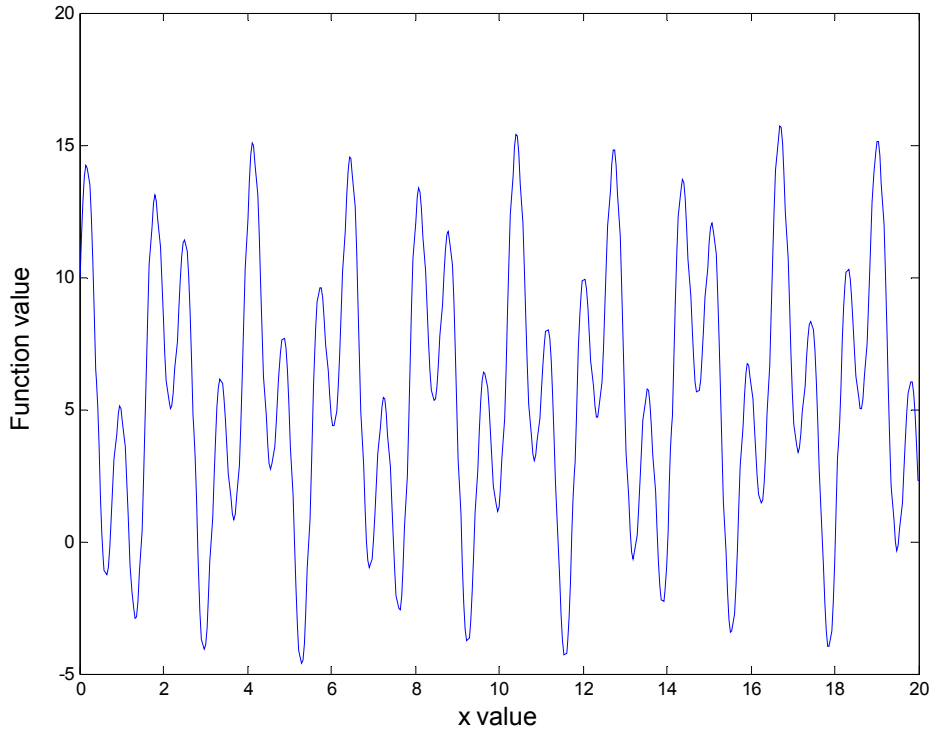
Създава се функцията eval_sincos в която се изчислява функцията:

```
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';
```

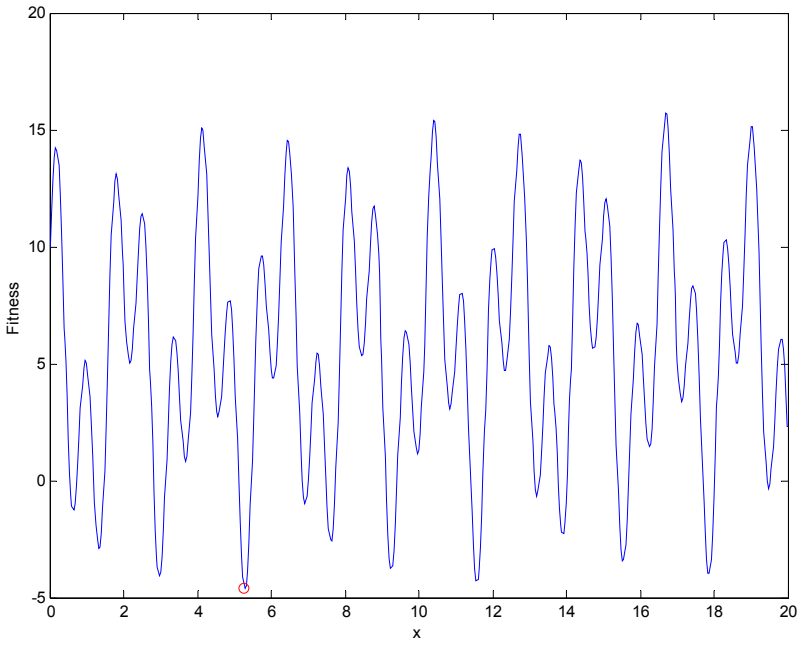


фиг. 4.5 – Графика на тригонометричната функция

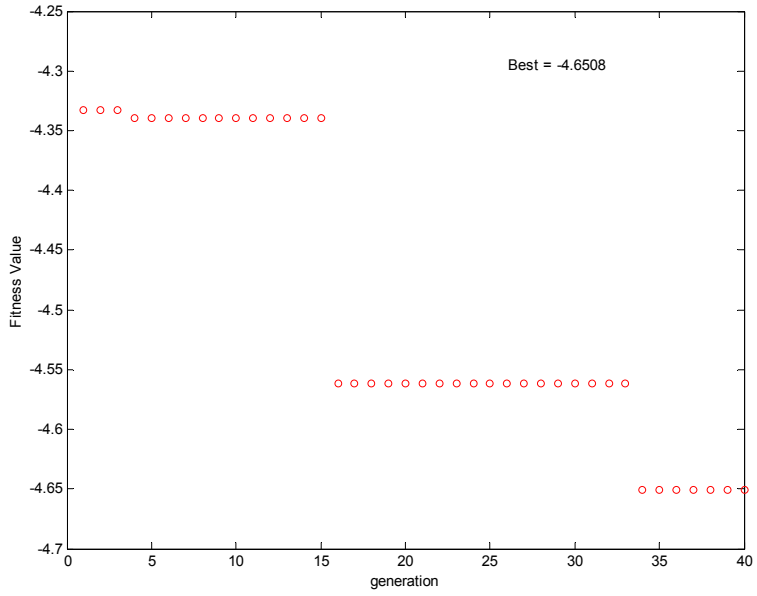
Вече може да се пусне оптимизацията. Графичният резултат е показан на фиг. 4.7.

```
>> [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
```

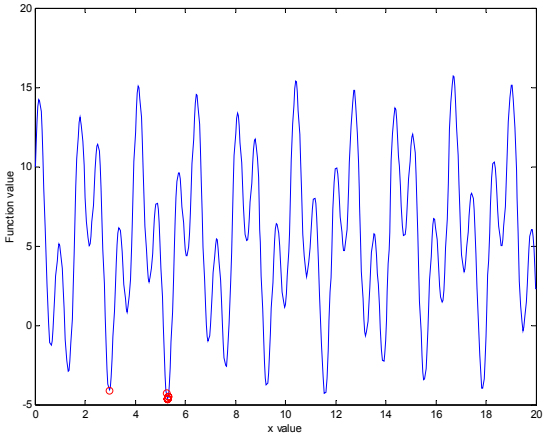
Визуализация на резултата в пространството на променливата е даден на фиг. 4.7. Резултатите от 10 последователни оптимизации, при намаляване на броят поколения до 20, са показани на фиг. 4.8.



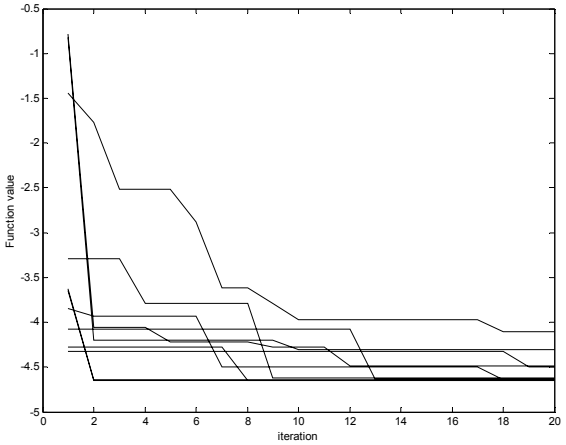
фиг. 4.6 – Решение на тригонометричната задача



фиг. 4.7 – Резултат на тригонометричната функция



а) получени резултати



б) графика на стойността на функционала

фиг. 4.8 – Тригонометрична функция – резултати при 10 оптимизации

4.4.3 Пример 3 – Двукритериален Розенброг

Функцията на Розенброг се разделя на две функции, които трябва едновременно да се минимизират:

$$\begin{cases} \min f_1(x) = \left(x_2 - x_1^2\right)^2 \\ \min f_2(x) = \left(1 - x_1\right)^2 \end{cases} \qquad (4\text{-}3)$$

Функцията изчисляваща стойностите на двата критерия е:

```
function f=eval_ros2(x)
% Evaluation of the Rosenberg function
%            2 variables
x1=x(1);
x2=x(2);

f(1) = (x2-x1^2)^2 ;
f(2) = (1-x1)^2 ;
```

Зареждат се настройките и дефинира максималният брой елементи в множеството на Парето.

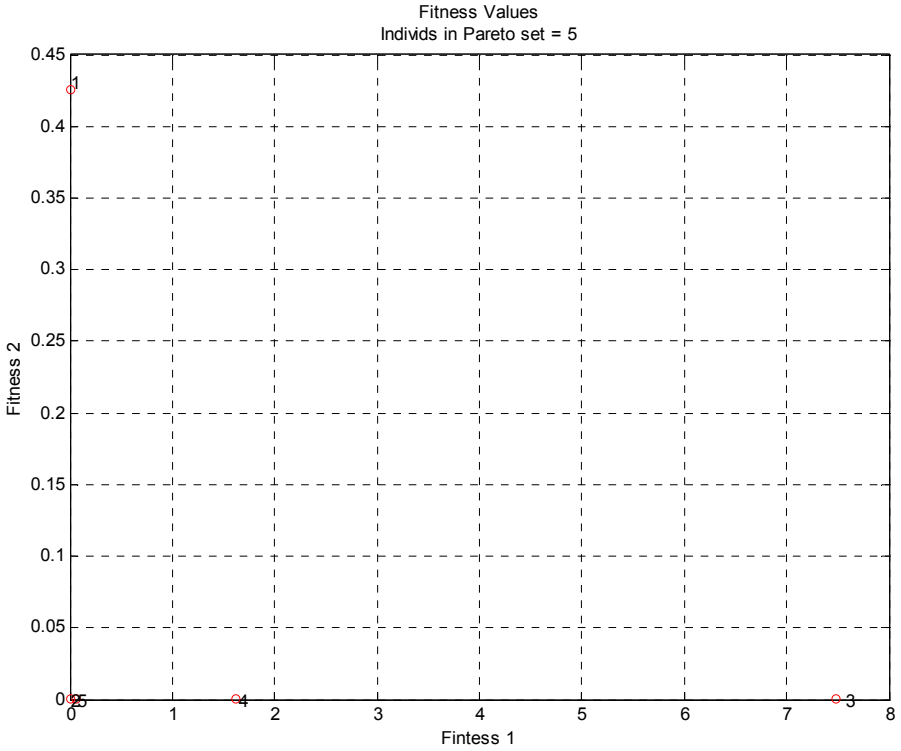
```
>> opt = GAopt(-2);
>> opt.BestRate = 0.5;
```

Задават се долна и горна граница на променливите, както и типът им. Стартира се минимизацията:

```
>> Bounds = [-10 10 0; -10 10 0];
>> [rGenes, rFit, Rec_Genes, Rec_Fit] = GAMOminBC('tst_ros', Bounds,
    opt);

 GENETIC ALGORITHM for MATLAB, ver 1.0
 MultiObjective Minimization of function "tst_ros"
 ->Iter:40 Popul:20<-
 Started at 13:26:21  1.7.2003
```

```
Iter:    2 ##>Fit:       8.12     0.1983   ##>Genes:   1.44532
    4.93844
Iter:    4 ##>Fit:   7.825e-005   0.000406   ##>Genes:   1.02015
    1.03186
Iter:   10 ##>Fit:   7.825e-005   0.000406   ##>Genes:   1.02015
    1.03186
Iter:   20 ##>Fit:   7.825e-005   0.000406   ##>Genes:   1.02015
    1.03186
Iter:   30 ##>Fit:   7.825e-005   0.000406   ##>Genes:   1.02015
    1.03186
Iter:   40 ##>Fit:   7.224e-009     0.4258   ##>Genes:   0.347452
    0.120808

 Fitness Values:
    0.0000    0.4258
    0.0020    0.0002
    7.4765    0.0000
    1.6180    0.0000
    0.0657    0.0001

 Genes Values:
    0.3475    0.1208
    0.9876    0.9311
    0.9997   -1.7350
    0.9949   -0.2822
    0.9910    0.7258
```

Чрез ParetoNumber се визуализират резултатите в множеството на Парето фиг. 4.9.



фиг. 4.9 – Множество на Парето при задача за Розенброг

5. Приложение на разработените програми в задачата за синтез на регулатор

Разгледани се три приложения на генетичните алгоритми в задачите за управление. Разгледани са пример за синтез на матрица на ОВ по зададени полюси на затворената система, настройка на ПИД регулатор по оптимизационен критерии и робастен синтез на матрица на ОВ.

5.1 Синтез на система с желано разположение на полюсите

Дадена е неустойчива система, описваща се с уравненията:

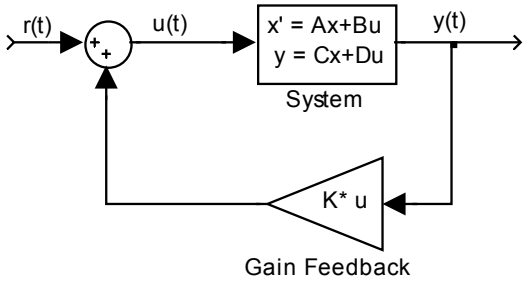
$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases} \qquad (5\text{-}1)$$

$$A = \begin{bmatrix} -0.5 & 0 & 0 \\ 0 & -2 & 10 \\ 0 & 1 & -2 \end{bmatrix}; B = \begin{bmatrix} 1 & 0 \\ -2 & 2 \\ 0 & 1 \end{bmatrix}; C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}; D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad (5\text{-}2)$$

Целта на задачата е да се синтезира матрична обратна връзка по изхода (фиг. 5.1), осигуряваща желано разположение на полюсите: [-5; -3; -1]. Като допълнително условие е поставено изискването стойностите на елементите на матрицата на обратната връзка трябва да са в границите (-4, 4). Уравненията описващи затворената система са:

$$u(t) = r(t) + Ky(t) = r(t) + KCx(t)$$

$$\begin{cases} \dot{x}(t) = Ax(t) + B\left(r(t) + KCx(t)\right) = \left(A + BKC\right)x(t) + Br(t) \\ y(t) = Cx(t) \end{cases} \qquad (5\text{-}3)$$



фиг. 5.1 – Схема на затворената система

Използва се критерии отразяващ близостта на полюсите на затворената система със зададените, чрез нормата на разликата между зададените и действителните полюси. Програмата, в която се изчислява стойността на целевата функция е:

```
% Evaluating the current regulator for the system
function [FValue] = eval_KS_1(xvec)

global A B C D

K = [xvec(1) xvec(2);xvec(3) xvec(4)];

X = A+B*K*C;
X = eig(X);
R = sort(X);

GOAL = [-5; -3; -1];

f1 = norm(R(1) - GOAL(1));
f2 = norm(R(2) - GOAL(2));
f3 = norm(R(3) - GOAL(3));

FValue = f1 + f2 + f3;
```

Зареждат се настройките и се задават матриците A, B, C и D. Използват се разширени гени, със зададена точност 10^{-4}, поколение от 50 хромозоми и 20 поколения.

```
>> opt=GAopt(-4)
opt =
      Comment: ':D Iter.100;  Popul.50;  Float; No Visualization'
      MaxIter: 100
    PopulSize: 50
    MutatRate: 0.3000
     BestRate: 0.1000
      NewRate: 0.1000
         TolX: 0
      pSelect: 1
      pRecomb: 0
       Select: 1
      RecIter: 1
       Visual: 'none'
     Graphics: 'off'

>> opt.Visual = 'some';
>> opt.MaxIter = 200;

>> Bounds=ones(4,1)*[ -4 4 1e-4 ];

>> A = [ -0.5  0  0;  0  -2  10;  0  1  -2 ];
>> B = [ 1  0;  -2  2;  0  1 ];
>> C = [ 1  0  0;  0  0  1 ];
>> D = [0 0; 0 0];
>> global A B C D
```

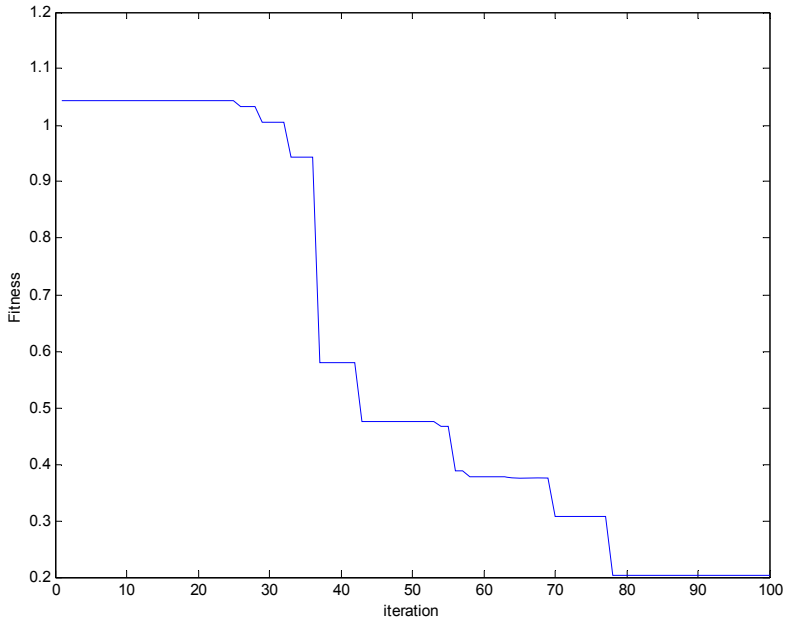
Вече може да се стартира минимизацията:

```
>> [RGenes, RFit, RecGenes, RecFit] = GAminBC('eval_KS_1', Bounds,
    opt);

 GENETIC ALGORITHM for MATLAB, ver 1.0
 Minimization of function "eval_KS_1"
 ->Iter:200 Popul:50<-
 Started at 20:45:21  29.6.2003

    Iter:      Fit:      Gen:
       1     2.42538   -0.8415   1.8684   2.0894   -1.4586
       8      1.1962   -0.4756   2.0881   -0.0416   -2.8282
      33    0.421564   -1.5624   1.3506   -0.5264   -3.2447
      50    0.330474   -1.5224   1.3506   -0.5268   -3.2447
      95    0.140141   -1.4524   1.3516   -0.4314   -3.0347
     100    0.140141   -1.4524   1.3516   -0.4314   -3.0347
     150    0.100714   -1.4644   1.3316   -0.4324   -3.1317
     200      0.0793   -1.4645   1.3316   -0.3303   -3.1148

 Fitness Value: 0.079300

 Result Genes:
   -1.4645    1.3316   -0.3303   -3.1148
```

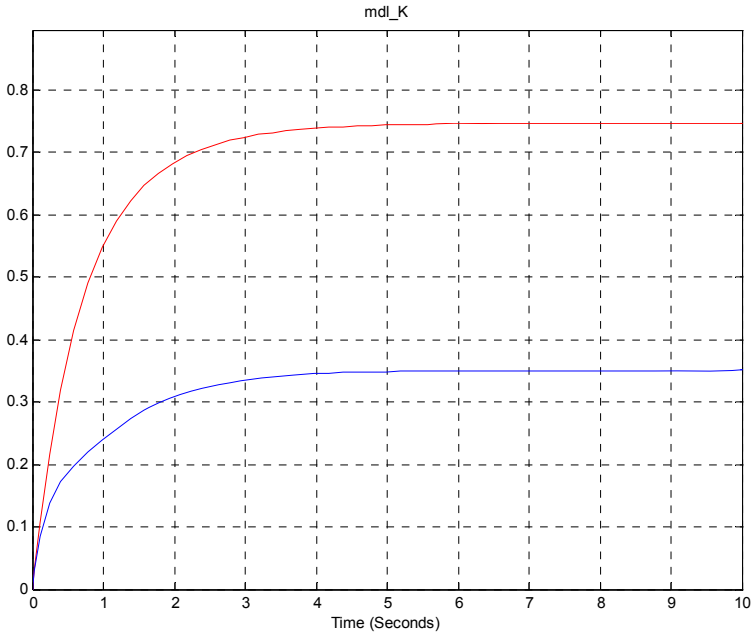
Проверява се разположението на полюсите на затворената система:

```
>> K = [RGenes(1:2); RGenes(3:4)];
>> G = sort(eig(A+B*K*C))'
G =
   -5.0085   -3.0315   -1.0393

>> sim('mdl_K');
```

Графиките на целевата функция в течение на итерациите и на изходите на системата са показани на фиг. 5.2 и фиг. 5.3.



фиг. 5.2 – Синтез на ОВ - графика на целевата функция



фиг. 5.3 – Синтез на ОВ – преходен процес

5.2 Настройка на непрекъснат ПИД регулатор

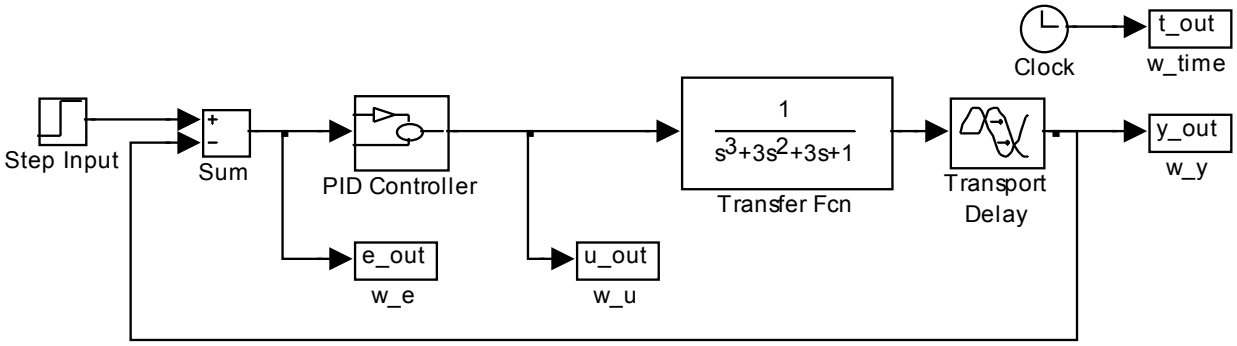
Да се настроят параметрите на ПИД регулатор (k_p, T_i и T_d) за системата, описана с предавателната функция:

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

Затворената система е показана на фиг. 5.4.

Разглеждат се три критерия за настройка:

- минимална стойност на квадрата на грешката между изхода и заданието;
- минимална стойност на квадрата на грешката и на управлението;
- минимална стойност на квадрата на грешката, минимум на управлението и минимално време за регулиране (t_p);



фиг. 5.4 – Схема на системата с ОВ

Първият критерии отчита само стойността на грешката, но не и реални физични ограничения, като максимална стойност на управлението. С вторият критерии се отчита и големината на приложеният управляващ сигнал. В третият критерии се добавя и изискване за времето на регулиране, с което се отчитат и технологични изисквания, но това става за сметка на по-голям брой изчисления. Тук ще бъдат показани предимствата на генетичните алгоритми при оптимизация на задачи с няколко, противоречащи си критерия.

5.2.1 Критерии min(e)

ПИД регулатора се настройва така, по минимум на функционала:

$$F = \int e^2(t)dt = e\_out'^* e\_out \qquad (5\text{-}5)$$

```
function F = eval_reg02(X)

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

sim('reg02',100);

F = e_out'*e_out;

return
```

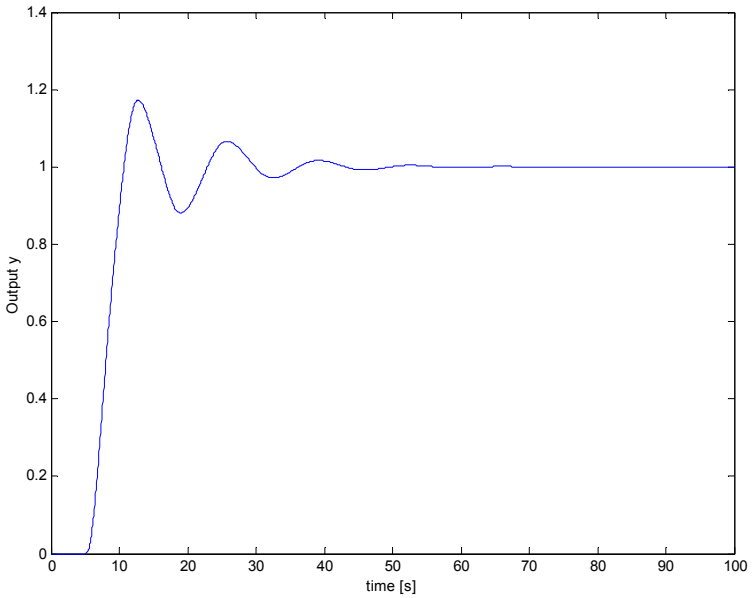
Стартира се минимизацията:

```
opt = GAopt(-6);
opt.MaxIter = 50;
opt.Graphics = 'no';
opt.Select = 3;
opt.pSelect = 0.8;

global Kp Ti Td

Bounds = [];
for i=1:3
    Bounds = [Bounds; 0 10 1e-4];
end

[RGenes, RFit, RecGenes, RecFit] = GAminBC ( 'eval_reg02', Bounds,
    opt);

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

sim('reg02',100);
figure;
plot(t_out,y_out);xlabel('time [s]');ylabel('Output y');
figure
plot(RecFit);xlabel('iteration');ylabel('Fitness value');
```

Получените резултати са дадени по-долу (фиг. 5.5 и фиг. 5.6).

```
GENETIC ALGORITHM for MATLAB, ver 1.0
 Minimization of function "eval_reg02"
 ->Iter:50 Popul:20<-
 Started at 14:28:36  1.7.2003

    Iter:      Fit:      Gen:
       1     226.666   0.2689   4.6772   3.6443
      13     155.354   0.5572   4.2552   1.5614
```

```
      26     153.233   0.5971   5.1552   1.9133
      39     150.539   0.6995   5.6252   1.8536

Fitness Value: 150.462862

Result Genes:
   0.6975    5.4273    1.9666
```



фиг. 5.5 – Преходен процес с оптимизационно настроен ПИД

Резултатите от 10 стартирания на минимизацията с различни начални поколения са показани на фиг. 5.7.

5.2.2 Критерии min(e,u)

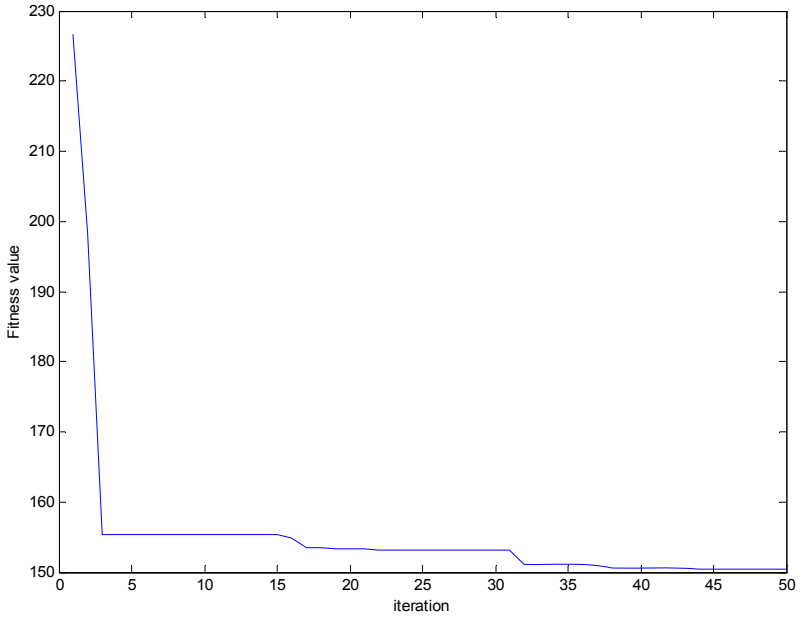
Въвежда се допълнителен критерии, целящ минимизиране на управляващият сигнал, т.е. отчитане на реални физически и икономически ограничения:

$$\begin{cases} f_1(x) = \int e^2(t)dt = e\_out' * e\_out \\ f_2(x) = \int u^2(t)dt = u\_out' * u\_out \end{cases} \qquad (5\text{-}6)$$

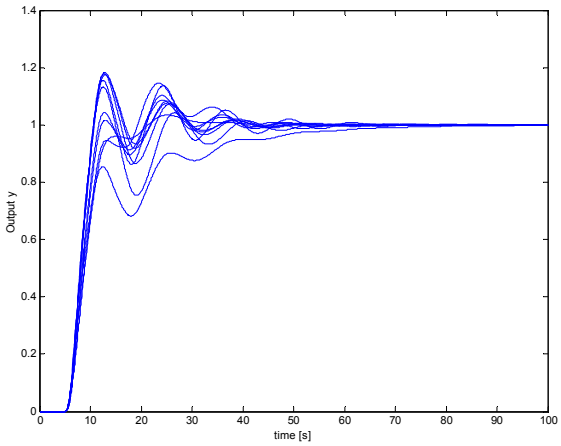
Функцията изчисляваща $f_1(x)$ и $f_2(x)$ е:

```
function F = eval_reg03(X)

global Kp Ti Td

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

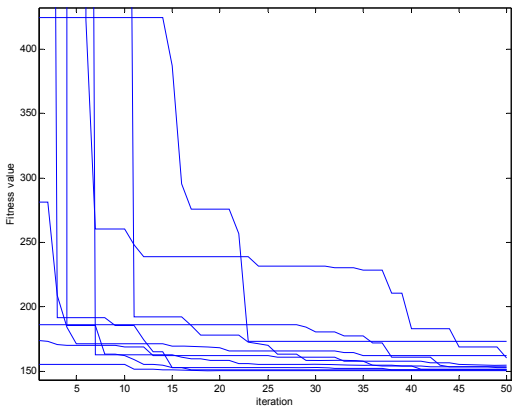
```
sim('reg02',100);

F = [];

F(1) = e_out'*e_out;
F(2) = u_out'*u_out;

return
```



фиг. 5.6 – Графика на целевата функция



а) Преходни процеси



б) Графика на функционала

фиг. 5.7 – Резултати от 10 минимизации

Стартирането на минимизацията се извършва чрез следните команди:

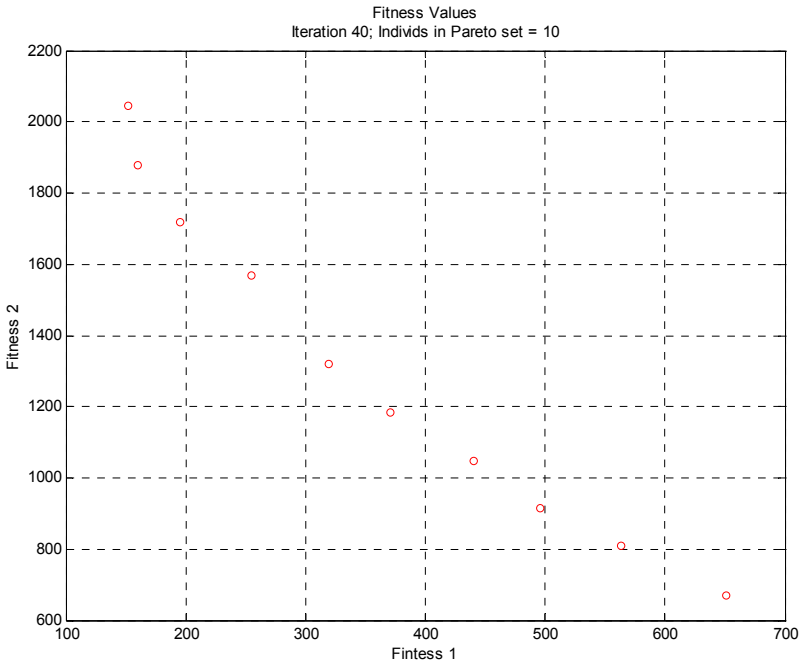
```
opt = GAopt(-5);
opt.MaxIter = 40;
opt.Select = 1;
opt.BestRate = 0.2; %10 Pareto optimal
```

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

[RGenes, RFit, RecGenes, RecFit] = GAMOminBC ( 'eval_reg03', Bounds,
    opt);
```

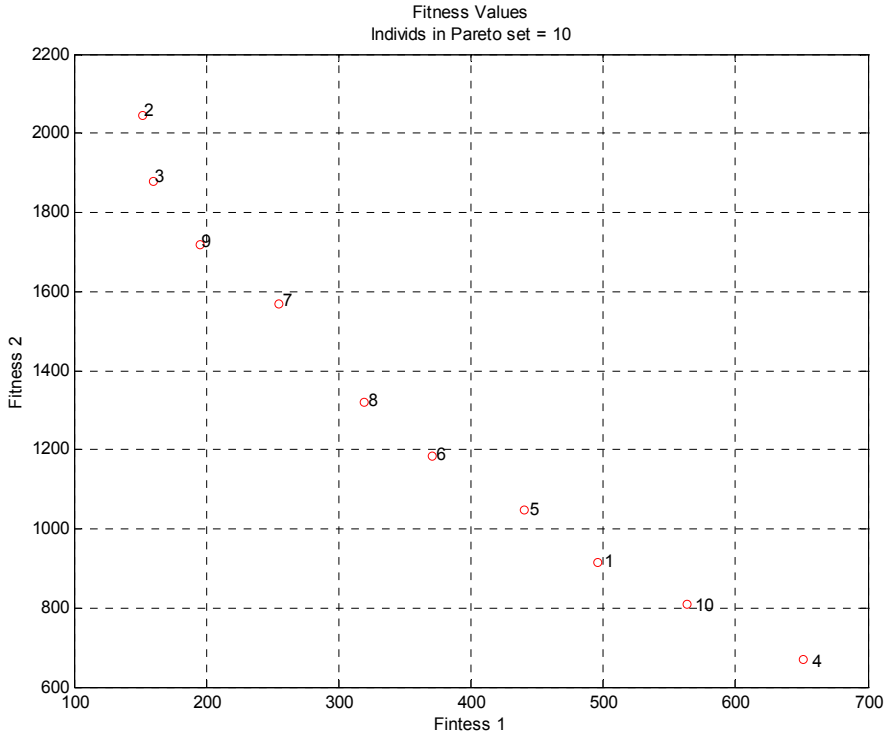
Получените резултати са:

```
GENETIC ALGORITHM for MATLAB, ver 1.0
 MultiObjective Minimization of function "eval_reg03"
 ->Iter:40 Popul:50<-
 Started at 15:12:58  1.7.2003
Iter:    2 ##>Fit:      223.5       1622   ##>Genes:   0.3563
    6.8466   1.6596
Iter:   10 ##>Fit:      223.5       1622   ##>Genes:   0.3563
    6.8466   1.6596
Iter:   20 ##>Fit:      171.9       1869   ##>Genes:   0.4632
    5.4654   2.6333
Iter:   30 ##>Fit:      431.8       1110   ##>Genes:   0.1145
    4.4457   3.5243
Iter:   40 ##>Fit:      496.2      915.8   ##>Genes:   0.1551
    7.5555   3.5424

 Fitness Values:
  1.0e+003 *
    0.4962    0.9158
    0.1516    2.0435
    0.1595    1.8769
    0.6514    0.6689
    0.4401    1.0460
    0.3703    1.1850
    0.2552    1.5675
    0.3196    1.3214
    0.1950    1.7171
    0.5631    0.8086

 Genes Values:
    0.1551    7.5555    3.5424
    0.6455    4.6454    2.3215
    0.6556    7.5155    1.5414
    0.1035    6.8654    2.6325
    0.1545    6.4444    2.5434
    0.2145    7.5654    2.6325
    0.2654    5.6655    3.6355
    0.2553    7.5955    3.5424
    0.4665    7.5445    2.5434
    0.1185    6.5654    2.6325
```

Графично повърхнината на Парето е показана на фиг. 5.8, а след номериране чрез `ParetoNumber` на фиг. 5.9. Номерата до точките показват поредният номер на хромозомата в получените резултати (`RGenes`).

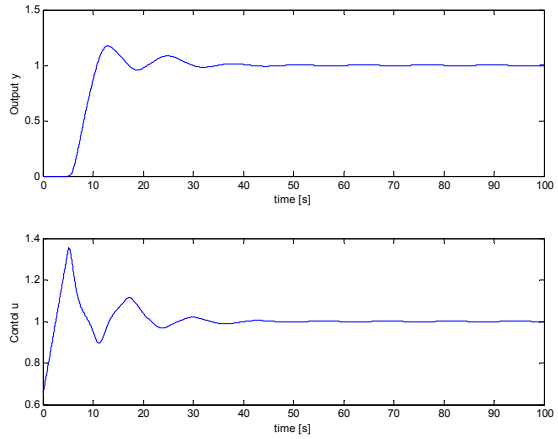


фиг. 5.8 – Повърхнина на Парето

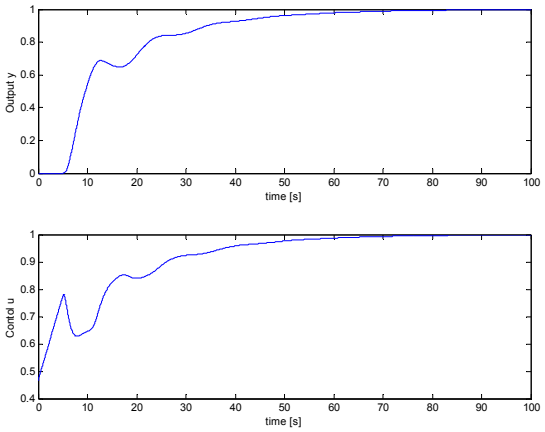


фиг. 5.9 – Повърхнина на Парето с номерирани точки
(Fitness 1 -> min e; Fitness 2 –> min u)

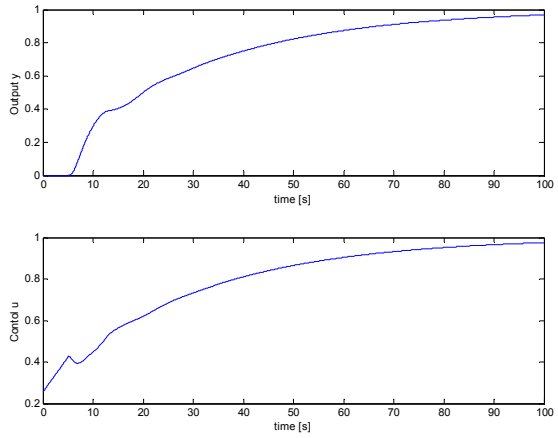
Преходната характеристика и управлението при 5 от получените решения (хромозоми номер 2, 9, 8, 1 и 4) са показани на



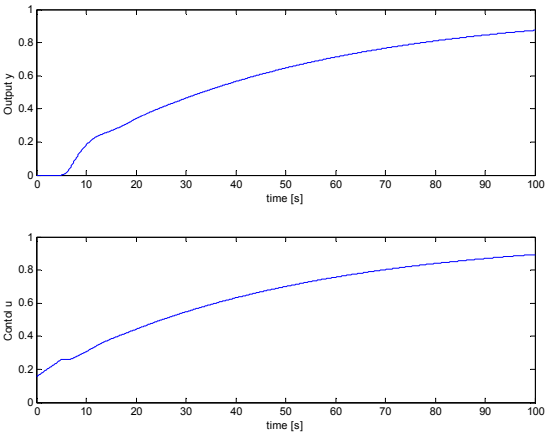
а) хромозома 2



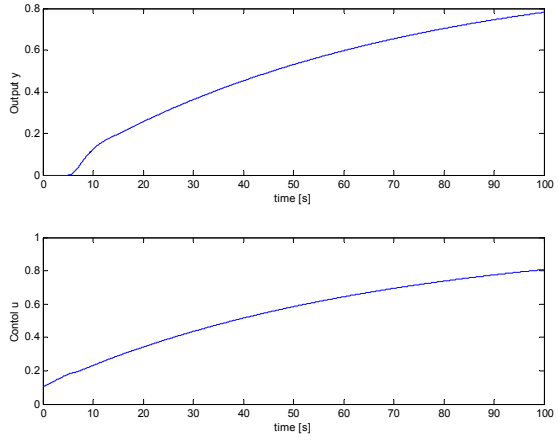
б) хромозома 9



в) хромозома 8



г) хромозома 1



д) хромозома 4

фиг. 5.10 – Преходни характеристики (различни хромозоми)

От получените резултати ясно се вижда, че така подбраният критерии за минимизиране на управлението е неподходящ при използване на толкова малко множество на Парето оптимални решения. Максималният брой от 10 решения не позволява да се получат решения в близост до оптималните за минимум на грешката (стойност на функционала под 160).

Опитвайки се да минимизира управлението алгоритъмът допуска недопустими за целите на управлението стойности на грешката.

Съставя се нов критерии за управлението, отчитащ само стойностите на управляващият сигнал, по-големи от 1.

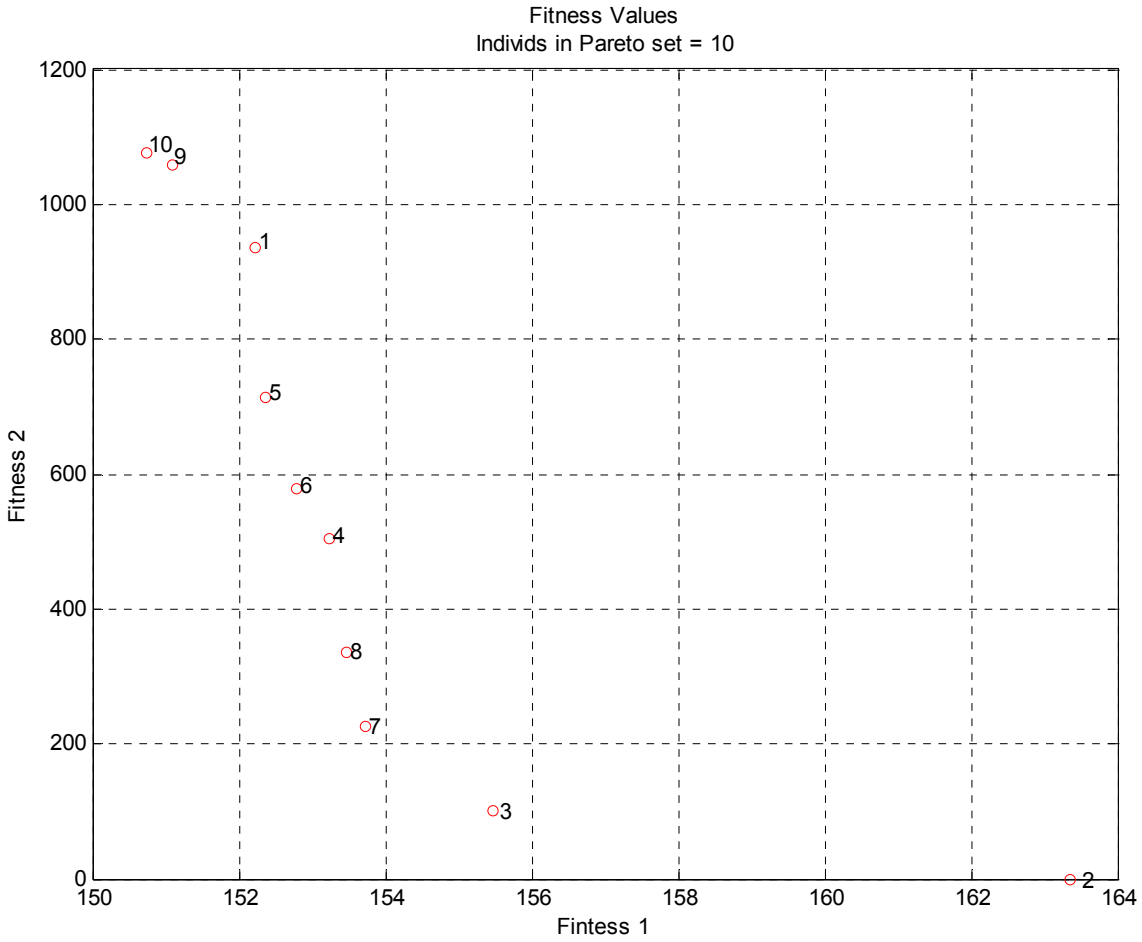
```
function F = eval_reg04(X)

global Kp Ti Td

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

sim('reg02',100);

F = [];

F(1) = e_out'*e_out;

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

F(2) = U_OUT'*U_OUT;

return
```

Получените резултати са:

```
GENETIC ALGORITHM for MATLAB, ver 1.0
 MultiObjective Minimization of function "eval_reg04"
 ->Iter:40 Popul:50<-
 Started at 15:47:43  1.7.2003
Iter:    2 ##>Fit:      622.8          0   ##>Genes:   0.0842
    5.1203   1.8316
Iter:   10 ##>Fit:      155.5       1092   ##>Genes:   0.5552
    4.3566   3.1445
Iter:   20 ##>Fit:      162.5      13.32   ##>Genes:   0.5654
    6.3455   2.1464
Iter:   30 ##>Fit:      160.7      41.44   ##>Genes:   0.5865
    6.4475   2.1578
Iter:   40 ##>Fit:      152.7      690.9   ##>Genes:   0.6865
    6.3485   1.2344

 Fitness Values:
  1.0e+003 *
    0.1522    0.9348
    0.1633         0
    0.1555    0.1014
    0.1532    0.5046
    0.1524    0.7142
    0.1528    0.5779
    0.1537    0.2254
    0.1535    0.3361
    0.1511    1.0587
    0.1507    1.0762

 Genes Values:
    0.6555    5.4776    1.2354
    0.5555    6.4476    1.2243
    0.6355    6.4636    1.5364
```

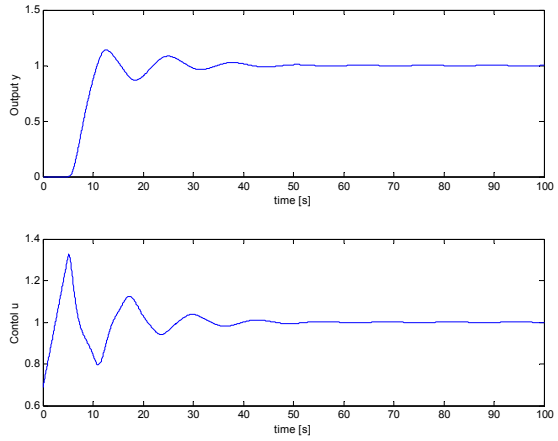
```
0.6245    5.6485    1.3353
0.6865    6.3485    1.4344
0.6855    6.4466    1.3364
0.6566    6.3466    1.3353
0.6249    5.7465    1.3353
0.6655    5.4475    2.2344
0.6864    5.4655    2.2443
```

Формата на Парето повърхнината е дадена на фиг. 5.11, а преходните процеси за точки номера 10, 1, 4, 3 и 2 на

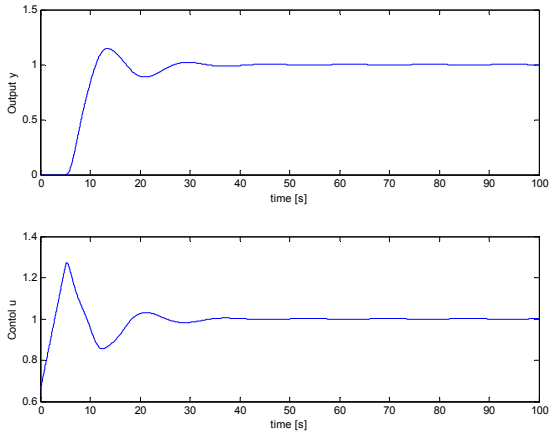


фиг. 5.11 – Повърхнина на Парето (модифицирана двукритериална задача)

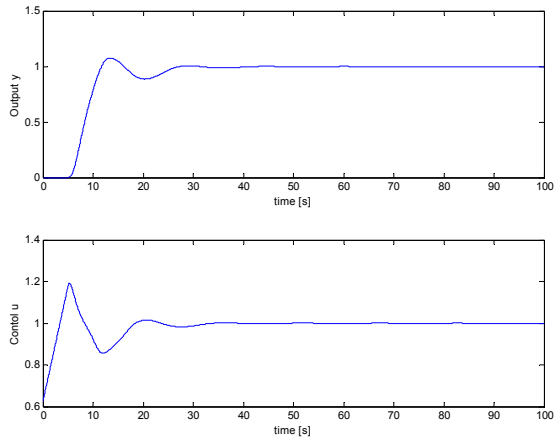
От получените графики ясно се вижда, как с намаляване на необходимият управляващ сигнал преходният процес се променя, и съответно грешката между задание и изход расте. Тези резултати илюстрират предимството на Парето методите – решението се взима апостериорно, а не априорно (избираме какви настройки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



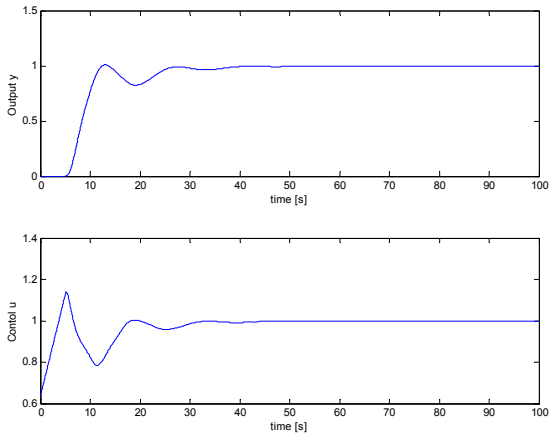
а) хромозома 10



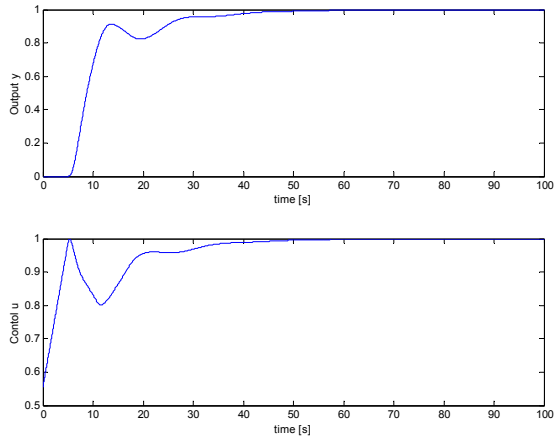
б) хромозома 1



в) хромозома 4



г) хромозома 3



д) хромозома 2

фиг. 5.12 – Преходни (модифицирана двукритериална задача)

Както се вижда при k_p, T_i и T_d взети от хромозома номер 2, управляващият сигнал не надминава 1 и стойността на функционала е 0.

5.2.3 Критерии min(e,t_p,u)

Добавя се трети критерии, отразяващ времето на преходният процес (времето за влизане в 5% зона около установената стойност):

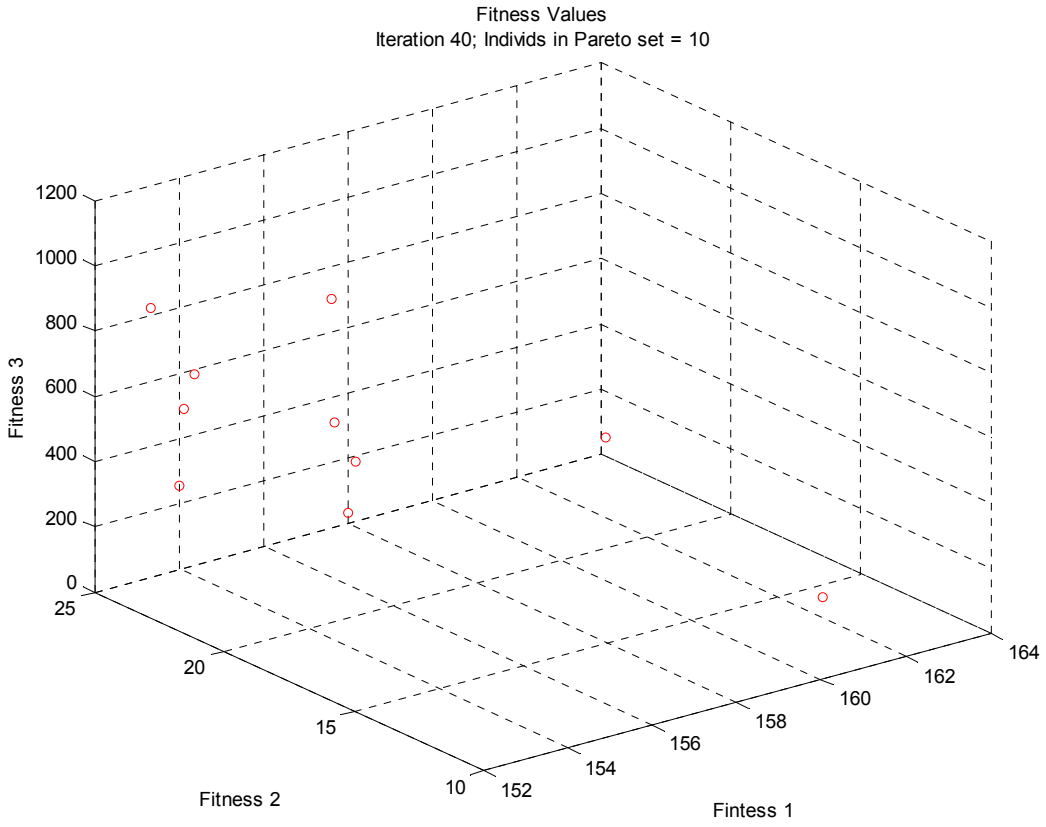
```
function F = eval_reg06(X)

global Kp Ti Td

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

sim('reg02',100);

F = [];

F(1) = e_out'*e_out;

ylen = length(y_out);
for i = ylen:-1:1
    if (y_out(i) > 1.05 | y_out(i) < 0.95)
     break;
    end
end

F(2) = t_out(i);

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

F(3) = U_OUT'*U_OUT;

return
```

Парето повърхнината в 3 измерено пространство е дадена на фиг. 5.13, а на фиг. 5.14 са дадени двумерни сечения.

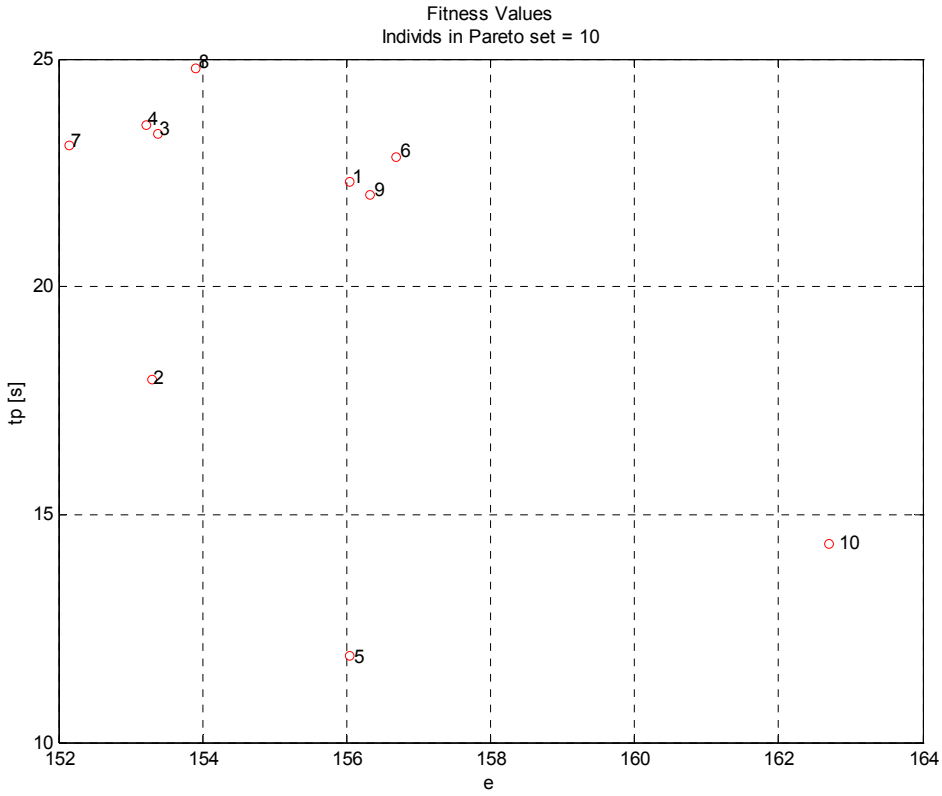
Преходните характеристики са показани за хромозоми с номера 7, 2, 5, 6 и 10 са показани на . Разпечатаните от оптимизацията резултати са:

```
GENETIC ALGORITHM for MATLAB, ver 1.0
 MultiObjective Minimization of function "eval_reg06"
 ->Iter:40 Popul:50<-
 Started at 16:39:46  1.7.2003
Iter:    2 ##>Fit:      170.8       26.5       1081   ##>Genes:
    0.3731   2.8526   1.2567
Iter:   10 ##>Fit:      172.2       17.8          0   ##>Genes:
    0.3766   3.9983   1.1739
Iter:   20 ##>Fit:      162.7       14.7        251   ##>Genes:
    0.4655   4.5763   1.5767
Iter:   30 ##>Fit:        156       22.3        478   ##>Genes:
    0.5525   5.0756   1.5517
Iter:   40 ##>Fit:        156       22.3        478   ##>Genes:
    0.5525   5.0756   1.5517

 Fitness Values:
  1.0e+003 *
    0.1560    0.0223    0.4780
    0.1533    0.0180    1.1123
```

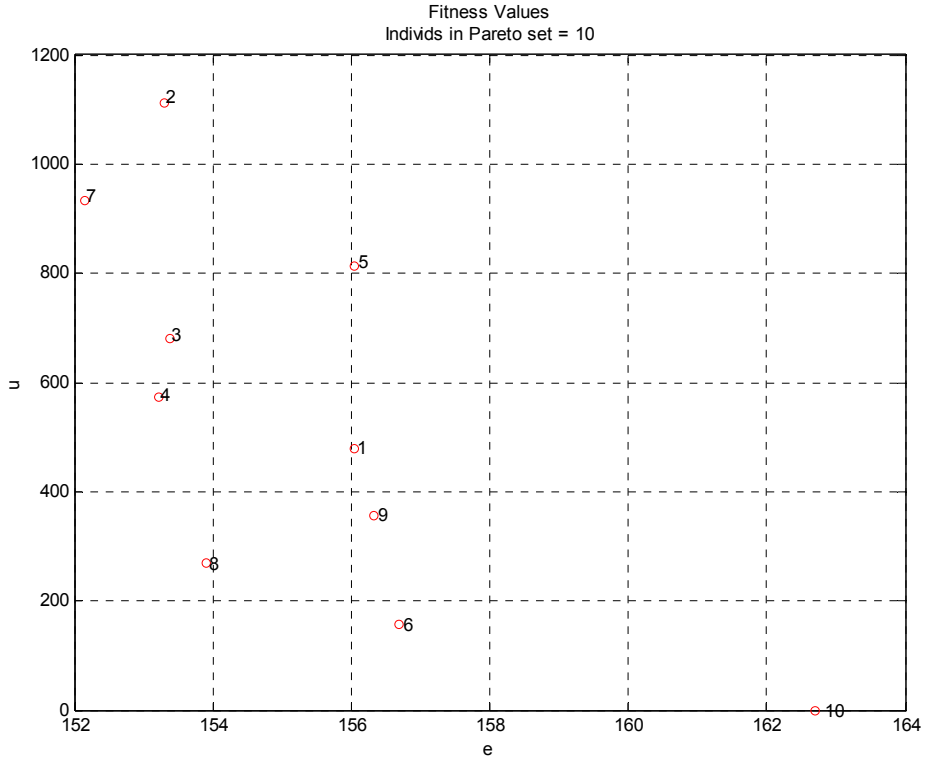
```
    0.1534    0.0234    0.6803
    0.1532    0.0236    0.5721
    0.1560    0.0119    0.8121
    0.1567    0.0229    0.1566
    0.1522    0.0231    0.9330
    0.1539    0.0248    0.2682
    0.1563    0.0220    0.3568
    0.1627    0.0144         0

Genes Values:
    0.5525    5.0756    1.5517
    0.6246    4.4887    1.6645
    0.6144    5.5665    1.5415
    0.6346    5.8886    1.5644
    0.5445    4.8663    1.4656
    0.5645    5.4672    1.7535
    0.6355    5.4663    1.5534
    0.6144    5.6665    1.2425
    0.5445    4.9663    1.4656
    0.4645    4.5660    1.4636
```



фиг. 5.13 – Повърхнина на Парето – 3 измерения

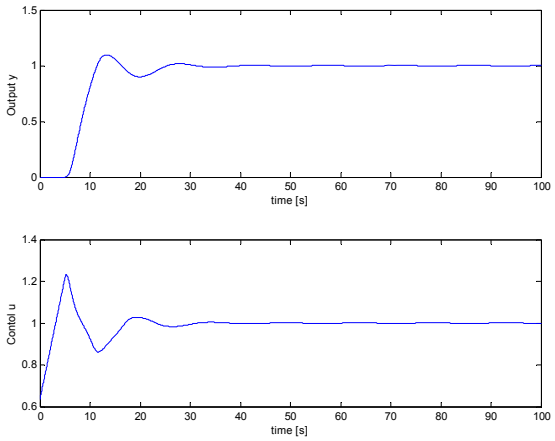


а) грешката във функция на времето на регулиране

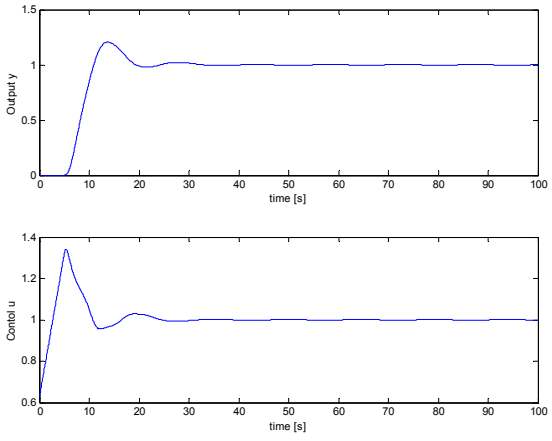


б) грешката във функция на управлението

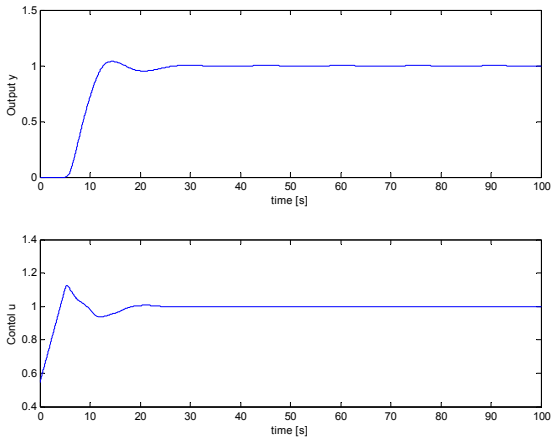
фиг. 5.14 – Сечения на повърхнината на Парето



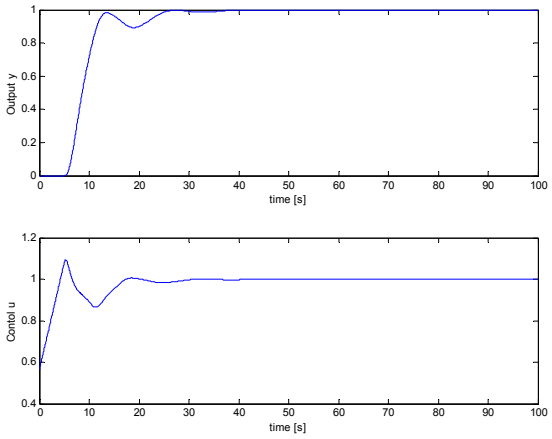
а) хромозома 7 (t_p=23,1s)



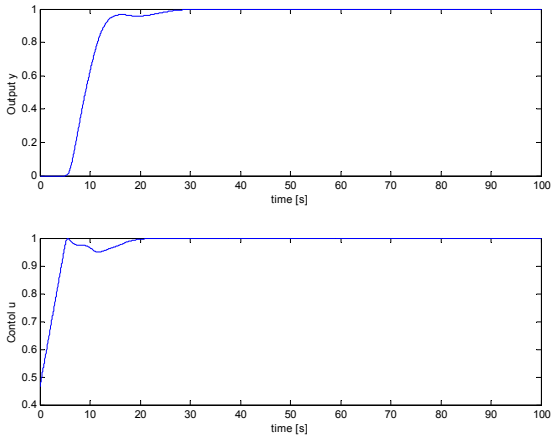
б) хромозома 2 (t_p=18s)



в) хромозома 5 (t_p=11,9s)



г) хромозома 6 (t_p=22,9s)



д) хромозома 10 (t_p=14,4s)

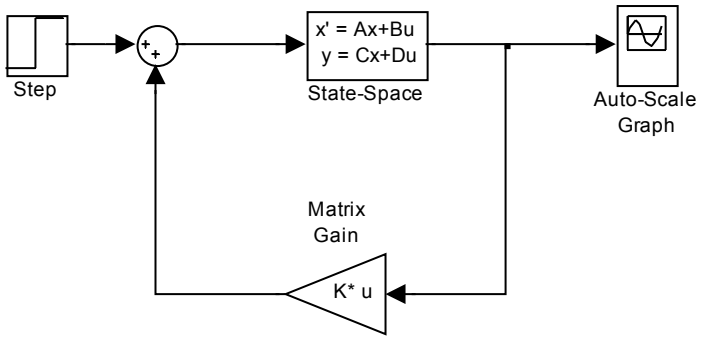
При настройките от хромозома 10 няма наложени наказания на управлението, т.е. управляващият сигнал на надхвърля 1.

фиг. 5.15 – Преходни характеристики при използване на 3 критерия

5.3 Робастен синтез на обратна връзка

Разглежда се отново задачата за синтез на ОВ по изход (фиг. 5.16) от пример 1 (точка 5.1). Тук матрицата се приема, че в процеса на работа на системата елементите на матрицата A, могат да се променят с ± 0.2 от текущите си стойности.

$$A = \begin{bmatrix} -0.5 & 0 & 0 \\ 0 & -2 & 10 \\ 0 & 1 & -2 \end{bmatrix}$$



фиг. 5.16 – SIMULINK схема на затворената система

Задачата се преобразува в min-max задача:

$$\min_{H} \max_{S} f(K, A) \qquad (5\text{-}7)$$

С H е отбелязано множеството на матриците на обратната връзка, с S – множеството от обекти.

Значението на горният израз може да се формулира и като: Да се намери тази ОВ множеството на ОВ, за която целевата функция с този обект от множеството на обектите, който дава най-лош показател за качество, има минимум. В термините на ГА това на практика означава, че за всяка матрица на ОВ от текущото поколение се търси този обект, при който полюсите на затворената система са най-отдалечени от зададените. След се избират тези матрици, които имат най-малка стойност на функционала.

Тази задача се решава чрез генетични алгоритми, тъй като min-max задачата е нелинейна и не могат да се използват стандартни алгоритми.

За решаването на тази задача за всяка хромозома от текущото поколение се стартира генетичен алгоритъм за минимизация, търсещ “най-лошият обект”.

Програмата използвана за определяне на стойността на функционала е:

```
% Evauate the Current System -> search for the worst system
function [FValue] = eval_S(X)

global K A B C

Ao =  [ X(1:3); X(4:6); X(7:9) ];

Ar = A + Ao;

Asys = Ar + B*K*C;
Psys = eig(Asys);
R = sort(Psys);

GOAL = [-5; -3; -1];

f1 = norm(R(1) - GOAL(1));
f2 = norm(R(2) - GOAL(2));
f3 = norm(R(3) - GOAL(3));

FValue = -(f1 + f2 + f3);     % Maximization Problem
```

Програмата определяща стойността на целевата функция за определена матрица на ОВ е:

```
% Evauate the Current Regulator -> search for the best regulator
function [FitValue] = eval_K(X)

global K A B C

K = [X(1:2); X(3:4)];    % X is vector row

opt=GAopt(-6);

opt.PopulSize = 20;
opt.MaxIter = 40;
opt.Visual   = 'none';
opt.Graphics = 'off';
Bounds_S = ones(9,1)*[ -0.2 0.2 1e-3 ];

[RGenes, RFit, RecGenes, RecFit] = GAminBC('eval_S', Bounds_S, opt);

FitValue = -RFit;
```

Основната програма стартираща синтеза е:

```
opt=GAopt(-6);
opt.PopulSize = 20;
opt.Visual = 'some';
opt.BestRate = 0.15;

Bounds_K = ones(4,1)*[ -4 4 1e-3 ];

global K A B C
```

```
RG = []; RF = [];recF = [];
for i = 1:5

    A =  [ -0.5  0  0;  0  -2  10;  0  1  -2 ];
    B =  [ 1  0;  -2  2;  0  1 ];
    C =  [ 1  0  0;  0  0  1 ];

    [RGenes, RFit, RecGenes, RecFit] = GAminBC('eval_K', Bounds_K,
    opt);

    RG = [RG; RGenes];
    RK = [RF RFit];
    recF = [recF RecFit];
end

save robast4

K = [RGenes(1:2); RGenes(3:4)];
G = sort(eig(A+B*K*C))'

try
    sim('mdl_K');
catch
    disp('Simulation unable');
end
```

Резултатите получени в следствие на синтеза са:

```
GENETIC ALGORITHM for MATLAB, ver 1.0
 Minimization of function "eval_K"
 ->Iter:100 Popul:20<-
 Started at 1:26:31  3.7.2003

    Iter:      Fit:      Gen:
       1     4.01419   -0.133   2.377    0.89   -0.846
      25     2.45255   -2.256   -0.333   -0.538   -2.647
      50     2.25716   -1.483   0.756   0.664   -1.535
      75     2.09908   -1.467   0.956   -0.339   -1.565
     100     1.95525   -1.467   0.956   -0.339   -1.565

 Fitness Value: 1.955251

 Result Genes:
   -1.4670    0.9560   -0.3390   -1.5650


G =

   -4.9077   -2.3885   -0.2358
```

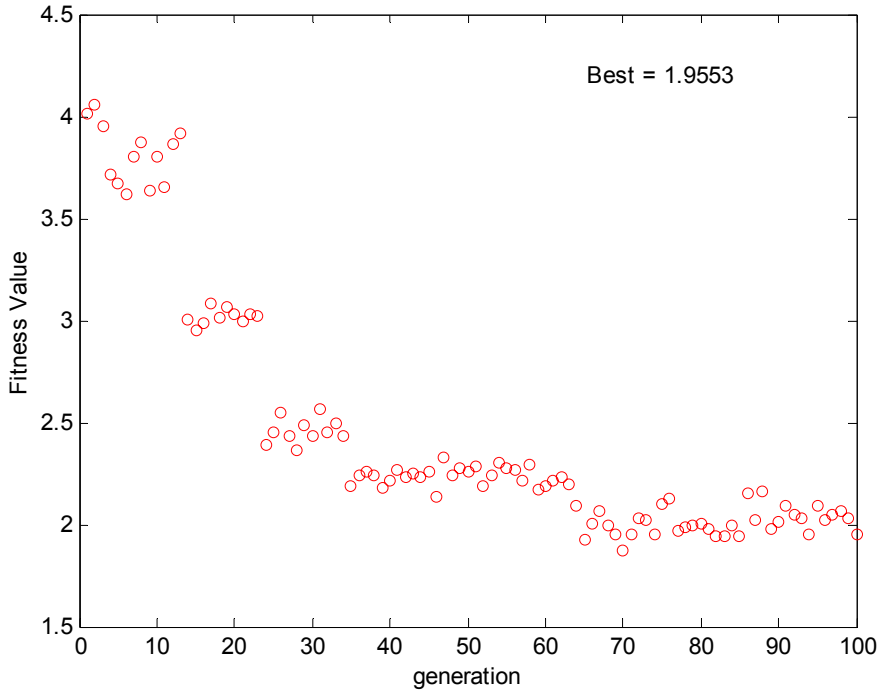
Стойностите на целевата функция по време на синтеза са показани на фиг. 5.17. Вижда се, че стойностите на целевата функция в някои от итерациите са по-големи от тези в предходните. Тъй като се използва на генетичен алгоритъм за изчисляване на целевата функция, не винаги се получава еднакъв резултат.

За така полученият регулатор се намира обекта, даващ най-голяма стойност на функционала чрез:

```
opt=GAopt(-2);

opt.PopulSize = 20;
opt.MaxIter = 40;

Bounds_S = ones(9,1)*[ -0.2 0.2 0 ];

[RGenes, RFit, RecGenes, RecFit] = GAminBC('eval_S', Bounds_S, opt);
```

Получената стойност за функционала е 1.940685, а за матрицата Ar (променена матрица A):

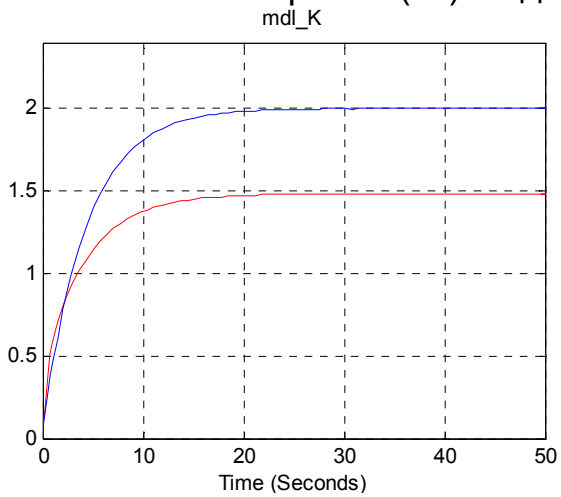
```
Ar =
   -0.4717    0.1749   -0.1750
    0.1394   -2.0633    9.9374
    0.0168    1.1975   -2.1866

>> A

A =
   -0.5000         0         0
         0   -2.0000   10.0000
         0    1.0000   -2.0000
```



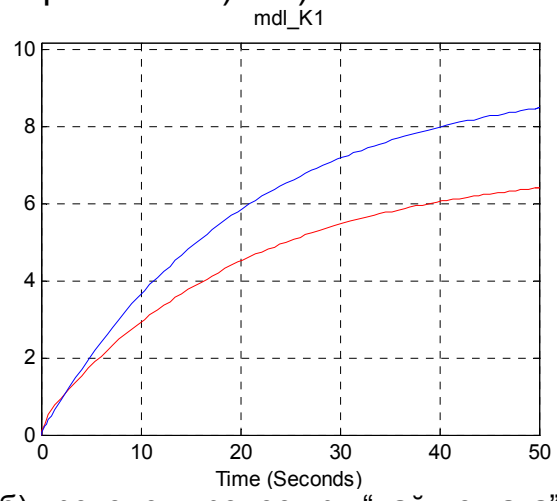
фиг. 5.17 – Целева функция на робастният синтез

Полюсите на затворената система са: [-5.3472 -2.3569 -0.0496].

Преходните характеристики на системата при номиналната матрица A и при най-лошият вариант (Ar) са дадени на фиг. 5.18 а) и б).



а) преходен процес при A номинално



б) преходен процес при “най-лошата” матрица A

фиг. 5.18 – Преходни процеси след робастен синтез на ОВ

За определяне на матрицата на ОВ са извършени общо 1 600 000 изчислявания на целевата функция. За сравнение може да се посочи, че общият брой на възможните комбинации между параметрите на матрицата K и параметрите на матрицата A, при избраната точност е повече от $1.63.10^{19}$.

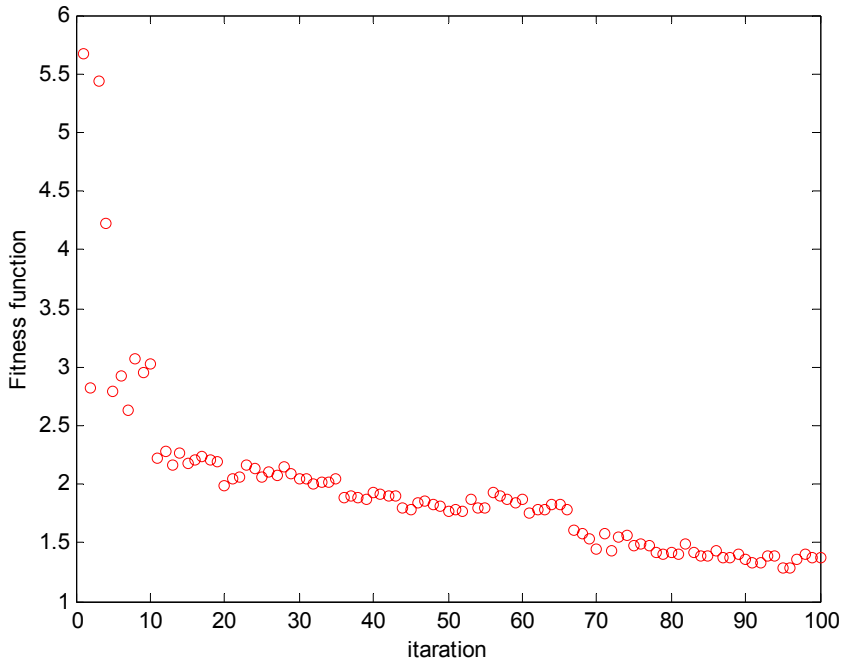
Задачата е пусната и при брой на поколенията 100 и размер на поколението 30 индивида. Това означава общо 9 000 000 изчислявания на целевата функция.

```
K =

   -1.9680    0.6620
   -0.5650   -2.6360
```

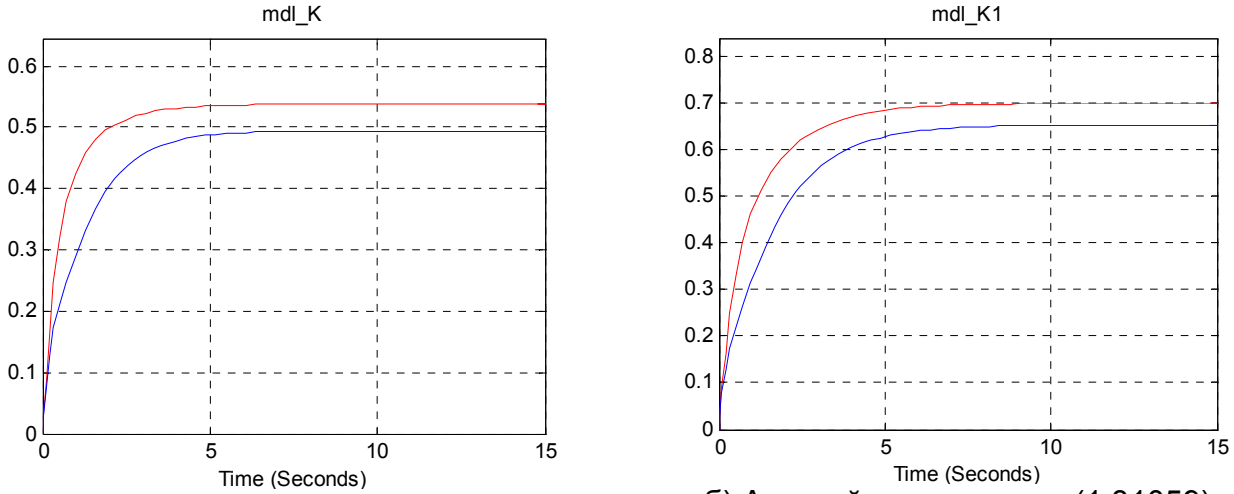
Полюсите на затворената система при номинални стойности на матрицата A са [-5.6407 -2.7098 -0.6568]. Графиката на целевата функция и на преходните процеси при A номинално (а) и Ar (б) са дадени съответно на фиг. 5.19.

```
Ar =

   -0.3110    0.1214   -0.1404
   -0.0137   -2.0202   10.0923
    0.1354    1.1328   -2.0721
```

Решаването на тази задача при предварително компилиране на програмите до DLL на компютър с процесор Pentium III 500MHz, 128 MB RAM отнема приблизително 2:30 часа.



фиг. 5.19 – Графика на целевата функция при 9 млн. изчислявания



а) A – номинална матрица

б) Ar – най-лош вариант (1.31059)

фиг. 5.20 – Преходни характеристики при 9 млн. изчислявания

6. Анализ на резултатите и изводи

В тази работа е направен преглед на генетичните алгоритми и е обяснена същността и работата на основните генетични операции. Разгледани са методи за решаване на едно- и многокритериални задачи, като са предложени и два нови метода за селекция изискващи по-малък изчислителен ресурс. Направена е класификация на генетичните алгоритми от гледан точка на областта им на приложение

Съставени са четири основни програмни модула за MATLAB, позволяващи работа с едно- и многокритериални задачи и стандартно кръстосване и кръстосване със смесване. Основните програми използват общ набор от функции, създадени на модулен принцип с цел по-лесна замяна и разширяване с нови функции.

В работата са разгледани три примера за приложение на генетичните алгоритми в задачите за управление:

1. Синтез на система с желано разположение на полюсите – синтезирана е матрична обратна връзка по изхода за неустойчива система;

2. Настройка на ПИД регулатор – извършена е настройка на ПИД регулатор по няколко различни критерия, като е показано предимството на използване на Парето оптимални решения;

3. Робастен синтез на ОВ – генетичните алгоритми са използвани за синтез на ОВ по изход (задача 1) неустойчива система при неопределеност в коефициентите на матрицата A. Критерият по който се извършва оценката е близост на полюсите на затворената система до желано разположение.

От направените експерименти може да се обобщи, че генетичните алгоритми:

- Приложими са за решаване на широк кръг от инженерни задачи;

- Подходящи са за многокритериални задачи, при които няма предварителна количествена информация за важността на отделните критерии. В този случай потребителят може да избере резултата отговарящ на нуждите му след завършване на оптимизацията;

- Подходящи са за решаване на нелинейни задачи (например min-max);

- Благодарение на използването на поколения от решения и стохастичният начин на търсене ГА са подходящи за многомодални задачи (имащи повече от един екстремума);

- Поради използването на случайни величини полученият резултат може да се различава при различни стартирания на алгоритъма, т.е. резултата се получава с определена вероятност (ясно личи на графиката на целевата функция при робастният регулатор);

- Генетичните алгоритми изискват голям брой изчислявания на целевата функция на всяка итерация, което от своя страна води до голямо изчислително време;

Последният проблем може от части да се реши в средата на MATLAB, като се използва вграденият в средата компилатор. Чрез него може да се избегне изпълняването на програмите в режим на интерпретатор и да се премине към DLL. При настройката на ПИД регулатор изчисляването на целевите функции е ускорено чрез използване на настройката за бързо изпълнение (Accelerator) в средата на SIMULINK. При използване на тази настройка преди първата симулация файла на модела (MDL файл) се компилира също до DLL.

Като препоръки при използването на генетични алгоритми, може да се посочат:

- Използване на решаване на нелинейни задачи, при които други методи са трудно или изобщо неприложими;

- Решаване на многоцелеви задачи;

- Решаване на задачи за глобален екстремум;

- Комбиниране с локални методи за търсене – след като ГА намери решение в близост до оптимума се стартира локален метод за търсене или се стартира отново генетичен алгоритъм със стеснен диапазон на търсене;

- Приложение върху процесорни архитектури позволяващи паралелна обработка.

7. Ръководство на потребителя

Ръководството на потребителя е изготвено като отделна книжка формат A5 и файл във формат PDF. Ръководството е предназначено за придобиване на основна информация за генетичните алгоритми и тяхното приложение. Описанието на създадените програми и функции е достатъчно за решаване на оптимизационни задачи не изискващи задълбочени познания за генетичните алгоритми.

Обемът на ръководството е 24 страници. В него са описани:

- основните принципи на генетичните алгоритми;
- използваните видове рекомбинация;
- използвани видове селекция;
- използваните видове представяне на променливите;
- приложение на генетичните алгоритми;
- четирите основни модули на разработените програми, както и допълнителните функции;
- разгледани са два примера илюстриращи употребата на генетичните алгоритми.

Използвана литература

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

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

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

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

[5] - 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

[6] - 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

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

[8] - 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

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

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

[11] – Practical Optimization – Gill Ph., Murray W., Wright M., Systems Optimization Laboratory, Dept. of Operation Research, Standford University, Academic Press, 1981

Приложение

Приложение А - Основни програми

GAminBC

еднокритериална минимизация с кръстосване със смесване

```
% GAminBC
%
% Genetic Algorithm Optimization
%       Blend CrossOver
%
% Minimization of single fitness function. The function is evaluated in user
    defined
% file. The fitness function can be function of many (N) variables. Each variable
% could be represented as real number (floating point), integer or subgenes from
% digits (0-9) (536.17 is represented by [5 3 6 1 7]).
%
% [ResultGenes, ResultFit, RecordGenes, RecordFit] = GAminBC ( Function, Bounds,
    Options )
%
%   Output:
%       ResultGenes - the result of the minimization
%       ResultFit  - result fitness value
%       RecordGenes - the best element (gen) taken with step, defined in Options
%       RecordFit  - Fitness function values
%
%   Input:
%       Function    - Name of the file which evaluates the fitness function
%                     Function should return value between ( -Inf; +Inf )
%                     Small value means better gen
%                     Input of function is column vector
%       Bounds      - matrix with the upper and lower boundary of the variables
%                     N x 3 matrix  ->  [min max Tol]
%                        min - minimal value of the variable
%                        max - maximal value of the variable
%                        Tol - tolerance for the variable OR variable type
%                              (if not given default value is used)
%                                     Tol = 1e-4 - tolerance 0.0001
%                                     Tol = 0    - float variable
%                                     Tol = -1   - integer variable
%       Options     - Options structure - use GAopt to create, load and save the
%                     structure to file. If Options is not set default options
    are used
%                            Select:
%                                  1 - Fitness Proportional Selection
%                                  2 - Gausinan Selection
%                                  3 - Ranking Selection
%
%
%   Andrey Popov                                  andrey.popov@mail.bg
%   www.automatics.hit.bg                         Last update: 20.06.20003
function [Result_Genes, Result_Fitness, RecordGenes, RecordFitness] =
    GAminBC(Function, bounds_orig, options)

alg_ver = '1.0';

if (nargin < 2)
    error('   Too few input arguments. See help "GAminDR" for details');
end

%================ Check the input arguments ================%
if bounds_orig(:,1) < bounds_orig(:,2)
else
    error('   Lower bound is greater than the Upper bound\n Bounds = [min max]');
end

%=================== Define the options ====================%
```

```
def_opt = GAopt(0);
if nargin < 3
    % no options set -> default options will be used
    options = def_opt;
else
    % update only fields which are not set
    options = CombineOpt(options,def_opt);
end

Comment     = options.Comment;
Num_cicles  = options.MaxIter;
populSize   = options.PopulSize;
mutatGenes  = ceil( populSize*options.MutatRate ); % will be update later
bestGenes   = ceil( populSize*options.BestRate );
newGenes    = ceil( populSize*options.NewRate );
TolX        = options.TolX;
rec_Iter    = options.RecIter;
par_Select  = options.pSelect;
par_Recomb  = options.pRecomb;
if (options.Select == 2)           type_Select = 2;  % Gausian law selection
elseif (options.Select == 3)  type_Select = 3;  % Ranking Selection
else type_Select = 1;                                                % Fitness
     Proportional Selection
end

if ( rec_Iter == 0 )  rec_Iter = Inf;  % no record
end
visual_iter = ceil(Num_cicles/4);

val = lower( options.Visual );
if ( strcmp(val,'none') )      visualize = -1; % no visualization at all
elseif ( strcmp(val,'no') )    visualize =  0; % no temporary results
elseif ( strcmp(val,'all') )   visualize =  2; % all temporary results
else                           visualize =  1; % some temporary results
end
val = lower( options.Graphics );
if ( strcmp(val,'on') )             graph = 1;   % graphics on
elseif ( strcmp(val,'final') )  graph = 0;   % only final graphics
else                                graph = -1;  % graphics off
end

clear def_opt val;

%============== Transform Genes to extended form ==============%
Bounds = TransfGenesBC(bounds_orig, TolX);
[ng, mg] = size(Bounds);
ng_ = ng + 1;
[ngo,mgo] = size(bounds_orig);

% Updating Mutation Rate with the Gen's extention rate
mutatGenes = mutatGenes*floor(ng/ngo);

RecordFitness           = NaN*ones(Num_cicles,1);   % record - every 'rec_Iter'
     iterations a new "best" fitness value is added
RecordGenes             = NaN*ones(Num_cicles,ngo); % record - every 'rec_Iter'
     iterations a new "best" gen is added

clear ngo mgo;

%=================== Display Start Message =====================%
if ( visualize >= 0 )
    disp(' ');
    clk = clock;
    disp([' GENETIC ALGORITHM for MATLAB, ver ',alg_ver]);
    disp([' Minimization of function "',Function,'"']);
    V = sprintf(' ->Iter:%d Popul:%d<-',Num_cicles, populSize); disp(V);
    clk = sprintf(' Started at %d:%d:%d
     %d.%d.%d',clk(4),clk(5),floor(clk(6)),clk(3),clk(2),clk(1));
    disp(clk);
end
if (graph > 0)
    FigHandle = figure;
```

```
else
    FigHandle = 0;
end

%%%%%%%%%%%%%%%% Generate initial population %%%%%%%%%%%%%%%%%%
Popul_Genes = GARandBC(Bounds, bounds_orig, populSize);
GBEST = Popul_Genes(1,:);                       % best result;

iteration = 0;

Result_Genes            = [];                   % Result of the optimization
bestFitnessValue     = Inf;                     % Best evaluation
bestFitnessValue_old = Inf;

if visualize == 1
    fprintf('\n    Iter:      Fit:      Gen:\n');
end

%%%%%%%%%%%%%%%%%%%%%%%%%% Start the cicle %%%%%%%%%%%%%%%%%%%%%%%%%%
while iteration < Num_cicles
    Popul_Fit_Genes = [];
    iteration = iteration + 1;

    %++++++++++++ Transform genes to standart type
    GenesStandart = GenesToStandart( Popul_Genes, Bounds );

    %++++++++++++ Evaluating the genes
    for i = 1:populSize
        ff = [ feval(Function, GenesStandart(i,:)),  Popul_Genes(i,:) ];
        Popul_Fit_Genes = [ Popul_Fit_Genes; ff ];
    end

    %++++++++++++ sorting the genes according to Fitnes Function
    Popul_Fit_Genes = sortrows ( Popul_Fit_Genes, 1 );

    GBEST = Popul_Fit_Genes ( 1:bestGenes, 2:ng_ );
    bestFitnessValue = Popul_Fit_Genes(1,1);

    BESTGenesStandart = GenesToStandart( Popul_Fit_Genes(1,2:ng_), Bounds );

    %++++++++++++ Make a record
     RecordFitness(iteration) = Popul_Fit_Genes(1,1);
     RecordGenes(iteration,:)  = BESTGenesStandart;

    %++++++++++++ Printing some results
     [bestFitnessValue_old] = VisualSO( visualize, graph, iteration, visual_iter,
     ...
                                                    bestFitnessValue,
     bestFitnessValue_old, ...
                                                    BESTGenesStandart,
     FigHandle, RecordFitness);

     %XXXXXXXXXXX Selection
     if ( type_Select == 2)   % Normal Gausian Selection
      SelParents = GASelectN ( populSize, populSize - bestGenes - newGenes,
     par_Select );
     elseif ( type_Select == 3)  % Ranking Selection
      SelParents = GASelectRS ( populSize, populSize - bestGenes - newGenes,
     par_Select );
     else                                  % Fitness Proportional Selection +
     Roulette Wheel
      SelParents = GASelectFP ( Popul_Fit_Genes(:,1), populSize - bestGenes -
     newGenes );
     end
     %XXXXXXXXXXX Recombination
    Cross_Genes = GARecombBC ( SelParents, Popul_Fit_Genes(:,2:ng_) , populSize -
     bestGenes - newGenes, Bounds(:,6), par_Recomb );
     %XXXXXXXXXXX Mutation
     GMut = GAMutatBC ( Cross_Genes, Bounds, bounds_orig, mutatGenes );
     %XXXXXXXXXXX New
     GNew = GARandBC ( Bounds, bounds_orig, newGenes );
```

```
     Popul_Genes = [GBEST; GNew; GMut];

end % while
%%%%%%%%%%%%%%%%%%%%%%%%%% End the cicle %%%%%%%%%%%%%%%%%%%%%%%%%%%%
Result_Fitness = Popul_Fit_Genes(1,1);
Result_Genes    = [ GenesToStandart( Popul_Fit_Genes(1,2:ng_), Bounds ) ];

if ( visualize >= 0 )
    fprintf ('\n Fitness Value: %f\n ', Result_Fitness);
    fprintf ('\n Result Genes:\n');disp(Result_Genes);
    fprintf ('\n');
     if (graph >= 0)
      if (graph == 0)         FigHandle = figure;
      end
      PlotResultsSO( iteration, FigHandle, RecordFitness );
     end
end

RecordFitness  = RecordFitness(1:rec_Iter:Num_cicles);
RecordGenes    = RecordGenes(1:rec_Iter:Num_cicles,:);

return
```

GAminSC

еднокритериална минимизация със стандартно кръстосване

```
% GAminSC
%
% Genetic Algorithm Optimization
%       Conventional CrossOver
%
% [ResultGenes, ResultFit, RecordGenes, RecordFit] = GAminSC ( Function, Bounds,
    Options, Mask )
%
%   Output:
%       ResultGenes - the result of the minimization
%       ResultFit  - result fitness value
%       RecordGenes - the best element (gen) taken with step, defined in Options
%       RecordFit  - Fitness function values
%
%   Input:
%       Function    - Name of the file which evaluates the fitness function
%                     Function should return value between ( -Inf; +Inf )
%                     Small value means better gen
%                     Input of function is column vector
%       Bounds      - matrix with the upper and lower boundary of the variables
%                     N x 3 matrix  ->  [min max VT]
%                        min - minimal value of the variable
%                        max - maximal value of the variable
%                        VT  - variable type
%                                    VT = 1  - integer variable
%                                    VT = 0  - float varaible
%       Options     - Options structure - use GAopt to create, load and save the
%                     structure to file. If Options is not set default options
    are used
%                              Select:
%                                    1 - Fitness Proportional Selection
%                                    2 - Gausinan Selection
%                                    3 - Ranking Selection
%       Mask        - boolean mask used for the crossover
%                     [NULL] - random mask is used
%
%
%   Andrey Popov                              andrey.popov@mail.bg
%   www.automatics.hit.bg                     Last update: 20.06.20003
function [Result_Genes, Result_Fitness, RecordGenes, RecordFitness] =
    GAminSC(Function, bounds_orig, options, mask)
```

```
alg_ver = '1.0';

if (nargin < 2)
    error('   Too few input arguments. See help "GAminDR" for details');
end

%================= Check the input arguments ==================%
if nargin < 4 | isempty(mask) == 1
    mask = [];
end
if bounds_orig(:,1) < bounds_orig(:,2)
else
    error('   Lower bound is greater than the Upper bound\n Bounds = [min max]');
end

%==================== Define the options ======================%
def_opt = GAopt(0);                % default options
if nargin < 3
    % no options set -> default options will be used
    options = def_opt;
else
    % update only fields which are not set
    options = CombineOpt(options,def_opt);
end

Comment     = options.Comment;
Num_cicles  = options.MaxIter;
populSize   = options.PopulSize;
mutatGenes   = ceil( populSize*options.MutatRate );
bestGenes    = ceil( populSize*options.BestRate );
newGenes     = ceil( populSize*options.NewRate );
TolX        = options.TolX;
rec_Iter    = options.RecIter;
par_Select  = options.pSelect;
par_Recomb  = options.pRecomb;
if (options.Select == 2)           type_Select = 2;  % Gausian law selection
elseif (options.Select == 3)  type_Select = 3;  % Ranking Selection
else type_Select = 1;                                          % Fitness
     Proportional Selection
end

if ( rec_Iter == 0 )  rec_Iter = Inf;  % no record
end
visual_iter = ceil(Num_cicles/4);

val = lower( options.Visual );
if ( strcmp(val,'none') )      visualize = -1; % no visualization at all
elseif ( strcmp(val,'no') )    visualize =  0; % no temporary results
elseif ( strcmp(val,'all') )   visualize =  2; % all temporary results
else                           visualize =  1; % some temporary results
end
val = lower( options.Graphics );
if ( strcmp(val,'on') )            graph = 1;   % graphics on
elseif ( strcmp(val,'final') )  graph = 0;   % only final graphics
else                                    graph = -1;  % graphics off
end

clear def_opt val;

%============== Transform Bounds to extended form ==============%
Bounds = TransfGenesSC(bounds_orig, TolX);
[ng, mg] = size(Bounds);
ng_ = ng+1;
[ngo,mgo] = size(bounds_orig);

%---------- Update Mask
Mask = [];
if ~isempty(mask)
    start_pos = 0;
    original  = 1;
    for i = 1:ng
        if Bounds(i,2) == 1
```

```
            Mask = [Mask mask(original)*ones(1,(i-start_pos))];
            start_pos = i;
            original = original + 1;
        end
    end
end

Result_Genes                    = [];                % Result of the optimization
bestFitnessValue     = Inf;                    % Best evaluation
bestFitnessValue_old = Inf;
RecordFitness = NaN*ones(Num_cicles,1);  % record - every 'rec_Iter' iterations a
     new "best" fitness value is added
RecordGenes    = NaN*ones(Num_cicles,ngo);     % record - every 'rec_Iter'
     iterations a new "best" gen is added

clear ngo mgo;

%=================== Display Start Message ======================%
if ( visualize >= 0 )
    disp(' ');
    clk = clock;
    disp([' GENETIC ALGORITHM for MATLAB, ver ',alg_ver]);
    disp([' Minimization of function "',Function,'"']);
    V = sprintf(' ->Iter:%d Popul:%d<-',Num_cicles, populSize); disp(V);
    clk = sprintf(' Started at %d:%d:%d
     %d.%d.%d',clk(4),clk(5),floor(clk(6)),clk(3),clk(2),clk(1));
    disp(clk);
end
if (graph > 0)
    FigHandle = figure;
else
    FigHandle = 0;
end

%%%%%%%%%%%%%%%% Generate initial population %%%%%%%%%%%%%%%%%%%
Popul_Genes = GARandSC(Bounds, populSize);
GBEST = Popul_Genes(1,:);                    % best result;

iteration = 0;

if visualize == 1
    fprintf('\n    Iter:      Fit:      Gen:\n');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%% Start the cicle %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while iteration < Num_cicles
    Popul_Fit_Genes = [];
    iteration = iteration + 1;

    %++++++++++++ Evaluating the genes
    for i = 1:populSize
        ff = [ feval(Function, Popul_Genes(i,:)),  Popul_Genes(i,:) ];
        Popul_Fit_Genes = [ Popul_Fit_Genes; ff ];
    end

    %++++++++++++ sorting the genes according to Fitnes Function
    Popul_Fit_Genes = sortrows ( Popul_Fit_Genes, 1);

    GBEST = Popul_Fit_Genes ( 1:bestGenes, 2:ng_ );
    bestFitnessValue = Popul_Fit_Genes(1,1);

    BESTGenes = Popul_Fit_Genes(1,2:ng_);

    %++++++++++++ Make a record
     RecordFitness(iteration) = Popul_Fit_Genes(1,1);
     RecordGenes(iteration,:)  = BESTGenes;

    %++++++++++++ Printing some results
     [bestFitnessValue_old] = VisualSO( visualize, graph, iteration, visual_iter,
     ...
                                                          bestFitnessValue,
     bestFitnessValue_old, ...
```

```
                                                                BESTGenes, FigHandle,
     RecordFitness);

     %XXXXXXXXXX Selection
     if ( type_Select == 2)   % Normal Gausian Selection
      SelParents = GASelectN ( populSize, populSize - bestGenes - newGenes,
     par_Select );
     elseif ( type_Select == 3)  % Ranking Selection
      SelParents = GASelectRS ( populSize, populSize - bestGenes - newGenes,
     par_Select );
     else                                % Fitness Proportional Selection +
     Roulette Wheel
      SelParents = GASelectFP ( Popul_Fit_Genes(:,1), populSize - bestGenes -
     newGenes );
     end

     %XXXXXXXXXX Recombination
     Cross_Genes = GACrossSC ( SelParents, Popul_Fit_Genes(:,2:ng_), populSize -
     bestGenes - newGenes, Mask );
     %XXXXXXXXXX Mutation
     GMut = GAMutatSC ( Cross_Genes, Bounds, mutatGenes );
     %XXXXXXXXXX New
     GNew = GARandSC( Bounds, newGenes );

     Popul_Genes = [GBEST; GNew; GMut];

end % while
%%%%%%%%%%%%%%%%%%%%%%%%%%%% End the cicle %%%%%%%%%%%%%%%%%%%%%%%%%%%%
Result_Fitness = Popul_Fit_Genes(1,1);
Result_Genes   = Popul_Fit_Genes(1,2:ng_);

if ( visualize >= 0 )
    fprintf ('\n Fitness Value: %f\n ', Result_Fitness);
    fprintf ('\n Result Genes:\n');disp(Result_Genes);
    fprintf ('\n');
     if (graph >= 0)
      if (graph == 0)         FigHandle = figure;
      end
      PlotResultsSO( iteration, FigHandle, RecordFitness );
     end
end

RecordFitness  = RecordFitness(1:rec_Iter:Num_cicles);
RecordGenes     = RecordGenes(1:rec_Iter:Num_cicles,:);

return
```

GAMOminBC

многокритериална минимизация с кръстосване със смесване

```
% GAMOminBC
%
% Genetic Algorithm MultiObjective Optimization
%       Blend CrossOver
%
% [ResultGenes, ResultFit, RecordGenes, RecordFit] = GAminBC ( Function, Bounds,
     Options )
%
%   Output:
%       ResultGenes - the result of the minimization
%       ResultFit   - result fitness value
%       RecordGenes - the best individual taken with step, defined in Options
%       RecordFit   - Fitness function values
%
%   Input:
%       Function    - Name of the file which evaluates the fitness function
%                     Function should return value between ( -Inf; +Inf )
%                     Small value means better gen
```

```
%                       Input of function is column vector
%       Bounds      - matrix with the upper and lower boundary of the variables
%                     N x 3 matrix  ->  [min max Tol]
%                        min - minimal value of the variable
%                        max - maximal value of the variable
%                        Tol - tolerance for the variable OR variable type
%                              (if not given default value is used)
%                                     Tol = 1e-4 - tolerance 0.0001
%                                     Tol = 0    - float variable
%                                     Tol = -1   - integer variable
%       Options     - Options structure - use GAopt to create, load and save the
%                     structure to file. If Options is not set default options
     are used
%                               Select:
%                                     Select - type of selection
%                                           1 - Non-Dominated Sorting Selection
%                                           2 - Pareto Optimal Selection
%
%
%   Andrey Popov                                  andrey.popov@mail.bg
%   www.automatics.hit.bg                         Last update: 28.06.2003
function [Result_Genes, Result_Fitness, RecordGenes, RecordFitness] = GAMOminBC
     (Function, bounds_orig, options)

alg_ver = '1.0';

if (nargin < 2)
    error('   Too few input arguments. See help "GAMOminDR" for details');
end

%================= Check the input arguments ==================%
if bounds_orig(:,1) < bounds_orig(:,2)
else
    error('   Lower bound is greater than the Upper bound\n Bounds = [min max
     VarType]');
end

%==================== Define the options ======================%
def_opt = GAopt(0);                % default options
if nargin < 3
    % no options set -> default options will be used
    options = def_opt;
else
    % update only fields which are not set
    options = CombineOpt(options,def_opt);
end

Comment     = options.Comment;
Num_cicles  = options.MaxIter;
populSize  = options.PopulSize;
mutatGenes  = ceil( populSize*options.MutatRate );
% Maximal number of non-dominated chromosomes
maxPareto   = max ( ceil( populSize*options.BestRate ), 3);
newGenes    = ceil( populSize*options.NewRate );
TolX        = options.TolX;
rec_Iter    = options.RecIter;
par_Select  = options.pSelect;
par_Recomb  = options.pRecomb;
if (options.Select == 2)     type_Select = 2;  % Pareto Optimal Selection
else                               type_Select = 1;  % Non-Dominated
     SortingSelection
end
if ( rec_Iter == 0 )  rec_Iter = Inf;  % no record
end
visual_iter = ceil(Num_cicles/4);

val = lower( options.Visual );
if ( strcmp(val,'none') )      visualize = -1; % no visualization at all
elseif ( strcmp(val,'no') )    visualize =  0; % no temporary results
elseif ( strcmp(val,'all') )   visualize =  2; % all temporary results
else                           visualize =  1; % some temporary results
end
```

```
val = lower( options.Graphics );
if ( strcmp(val,'on') )            graph = 1;   % graphics on
elseif ( strcmp(val,'final') )  graph = 0;   % graphics only when done
else                                       graph = -1;  % graphics off
end

clear def_opt val;

%============== Transform Bounds to extended form ==============%
Bounds = TransfGenesBC(bounds_orig, TolX);
[ng, mg] = size(Bounds);
ng_ = ng+1;
[ngo,mgo] = size(bounds_orig);

% Updating Mutation Rate with the Gen's extention rate
mutatGenes = mutatGenes*floor(ng/ngo);

clear ngo mgo;

%=================== Display Start Message ====================%
if ( visualize >= 0 )
    disp(' ');
    clk = clock;
    disp([' GENETIC ALGORITHM for MATLAB, ver ',alg_ver]);
    disp([' MultiObjective Minimization of function "',Function,'"']);
    V = sprintf(' ->Iter:%d Popul:%d<-',Num_cicles, populSize); disp(V);
    clk = sprintf(' Started at %d:%d:%d
     %d.%d.%d',clk(4),clk(5),floor(clk(6)),clk(3),clk(2),clk(1));
    disp(clk);
end
if (graph > 0)
    FigHandle = figure;
else
    FigHandle = 0;
end

%%%%%%%%%%%%%%%%% Generate initial population %%%%%%%%%%%%%%%%%%%
Popul_Genes = GARandBC(Bounds, bounds_orig, populSize);

iteration = 0;

RecordFitness           = [];%NaN*ones(Num_cicles,1);     % record - every
     'rec_Iter' iterations a new "best" fitness value is added
RecordGenes             = [];%NaN*ones(Num_cicles,ng);    % record - every
     'rec_Iter' iterations a new "best" gen is added

Pareto_Set              = [];
Pareto_Fit              = [];
Non_Pareto_Set          = [];
Non_Pareto_Fit          = [];
top_Pareto              = Inf;
top_Pareto_old          = Inf;
TD                      = 0; % flag: 1 is there have been a total
                               %   dominating individual in current population

%%%%%%%%%%%%%%%%%%%%%%%%%%% Start the cicle %%%%%%%%%%%%%%%%%%%%%%%%%%%
while iteration < Num_cicles
    iteration = iteration + 1;

    %++++++++++++ Transform genes to standart type
    GenesStandart = GenesToStandart( Popul_Genes, Bounds );

     Fitness_Value             = [];
    %++++++++++++ Evaluating the genes
    for i = 1:populSize
        ff = feval(Function, GenesStandart(i,:));
        Fitness_Value = [ Fitness_Value; ff ];
    end

    %++++++++++++ finding the individuals in Pareto set
     [Pareto_Set, Pareto_Fit, Non_Pareto_Set, Non_Pareto_Fit]  = ...
                         sortPareto( Popul_Genes, Fitness_Value, Pareto_Set,
```

```
     Pareto_Fit );

    %++++++++++++ Make a record
     RecordFitness = [ RecordFitness; Pareto_Fit(1,:) ];
     RecordGenes   = [ RecordGenes; GenesToStandart( Popul_Genes, Bounds ) ];

    %++++++++++++ Printing some results
     Pareto_Set_Stn = GenesToStandart( Pareto_Set, Bounds );
     if (iteration == 1)
      old_Pareto_best          = Inf*ones(size(Pareto_Fit(1,:)));
     else
      old_Pareto_best = VisualMO ( visualize, graph, iteration, visual_iter, ...
                                   Pareto_Set_Stn, Pareto_Fit, old_Pareto_best,
     FigHandle);
     end

     %XXXXXXXXXXX Selection
     if ( type_Select == 2)
      [SelParents, Pareto_Set_total] = GAParetoOpt (Pareto_Set, Pareto_Fit,
     Non_Pareto_Set, Non_Pareto_Fit, populSize);
     else
      [SelParents, Pareto_Set_total] = GANDominSort (Pareto_Set, Pareto_Fit,
     Non_Pareto_Set, Non_Pareto_Fit, populSize);
     end

     %XXXXXXXXXXX Recombination
    Cross_Genes = GARecombBC ( SelParents, Pareto_Set_total , populSize -
     newGenes, Bounds(:,6), par_Recomb );

     %XXXXXXXXXXX Mutation
     GMut = GAMutatBC ( Cross_Genes, Bounds, bounds_orig, mutatGenes );

     %XXXXXXXXXXX New
     GNew = GARandBC ( Bounds, bounds_orig, newGenes );

     %XXXXXXXXXXX Reducing number of Pareto optimal solutions
     [Pareto_Set, Pareto_Fit] = ReducePareto(Pareto_Set, Pareto_Fit, maxPareto);

     Popul_Genes = [GNew; GMut];

end % while
%%%%%%%%%%%%%%%%%%%%%%%%%%%% End the cicle %%%%%%%%%%%%%%%%%%%%%%%%%%%%
Result_Fitness = Pareto_Fit;
Result_Genes   = GenesToStandart (Pareto_Set, Bounds);
RecordFitness  = RecordFitness (1:rec_Iter:Num_cicles, :);
RecordGenes    = RecordGenes (1:rec_Iter:Num_cicles, :);

fprintf('\n');
if ( visualize >= 0 )
    fprintf (' Fitness Values:\n');
     disp(Result_Fitness);
    fprintf ('\n Genes Values:\n');
    disp(Result_Genes);
    fprintf ('\n');
     if (graph >= 0)
      if (graph == 0)         FigHandle = figure;
      end
      PlotResultsMO( iteration, FigHandle, Pareto_Fit );
     end
end
return
```

GAMOminSC

многокритериална минимизация със стандартно кръстосване

```
% GAMOminSC
%
% Genetic Algorithm MultiObjective Optimization
```

```
%       Conventional CrossOver
%
% [ResultGenes, ResultFit, RecordGenes, RecordFit] = GAminSC ( Function, Bounds,
    Options, Mask )
%
%   Output:
%       ResultGenes - the result of the minimization
%       ResultFit   - result fitness value
%       RecordGenes - the best individual taken with step, defined in Options
%       RecordFit   - Fitness function values
%
%   Input:
%       Function    - Name of the file which evaluates the fitness function
%                     Function should return value between ( -Inf; +Inf )
%                     Small value means better gen
%                     Input of function is column vector
%       Bounds      - matrix with the upper and lower boundary of the variables
%                     N x 3 matrix  ->  [min max VT]
%                        min - minimal value of the variable
%                        max - maximal value of the variable
%                        VT  - variable type
%                                    VT = -1 - integer variable
%                                    VT = 0  - float varaible
%       Options     - Options structure - use GAopt to create, load and save the
%                     structure to file. If Options is not set default options
    are used
%                                  Select - type of selection
%                                        1 - Non-Dominated Sorting Selection
%                                        2 - Pareto Optimal Selection
%       Mask        - boolean mask used for the crossover
%                     [NULL] - random mask is used
%
%
%   Andrey Popov                                andrey.popov@mail.bg
%   www.automatics.hit.bg                       Last update: 20.06.2003
function [Result_Genes, Result_Fitness, RecordGenes, RecordFitness] = GAMOminSC
    (Function, bounds_orig, options, mask)

alg_ver = '1.0';

if (nargin < 2)
    error('   Too few input arguments. See help "GAMOminDR" for details');
end

%================= Check the input arguments ==================%
if nargin < 4 | isempty(mask) == 1
    mask = [];
end
if bounds_orig(:,1) < bounds_orig(:,2)
else
    error('   Lower bound is greater than the Upper bound\n Bounds = [min max
    VarType]');
end

%==================== Define the options ======================%
def_opt = GAopt(0);                % default options
if nargin < 3
    % no options set -> default options will be used
    options = def_opt;
else
    % update only fields which are not set
    options = CombineOpt(options,def_opt);
end

Comment     = options.Comment;
Num_cicles  = options.MaxIter;
populSize  = options.PopulSize;
mutatGenes  = ceil( populSize*options.MutatRate );
% Maximal number of non-dominated chromosomes
maxPareto   = max ( ceil( populSize*options.BestRate ), 3);
newGenes    = ceil( populSize*options.NewRate );
TolX        = options.TolX;
```

```
rec_Iter    = options.RecIter;
par_Select  = options.pSelect;
par_Recomb  = options.pRecomb;
type_Select = 1; % options.Select;
if ( rec_Iter == 0 )  rec_Iter = Inf;  % no record
end
visual_iter = ceil(Num_cicles/4);

val = lower( options.Visual );
if ( strcmp(val,'none') )      visualize = -1; % no visualization at all
elseif ( strcmp(val,'no') )    visualize =  0; % no temporary results
elseif ( strcmp(val,'all') )   visualize =  2; % all temporary results
else                           visualize =  1; % some temporary results
end
val = lower( options.Graphics );
if ( strcmp(val,'on') )            graph = 1;   % graphics on
elseif ( strcmp(val,'final') )  graph = 0;   % graphics only when done
else                                    graph = -1;  % graphics off
end

clear def_opt val;

%============== Transform Bounds to extended form ==============%
Bounds = TransfGenesSC(bounds_orig, TolX);
[ng, mg] = size(Bounds);
ng_ = ng+1;

%---------- Update Mask
Mask = [];
if ~isempty(mask)
    start_pos = 0;
    original  = 1;
    for i = 1:ng
        if Bounds(i,2) == 1
            Mask = [Mask mask(original)*ones(1,(i-start_pos))];
            start_pos = i;
            original = original + 1;
        end
    end
end

%=================== Display Start Message ======================%
if ( visualize >= 0 )
    disp(' ');
    clk = clock;
    disp([' GENETIC ALGORITHM for MATLAB, ver ',alg_ver]);
    disp([' MultiObjective Minimization of function "',Function,'"']);
    V = sprintf(' ->Iter:%d Popul:%d<-',Num_cicles, populSize); disp(V);
    clk = sprintf(' Started at %d:%d:%d
     %d.%d.%d',clk(4),clk(5),floor(clk(6)),clk(3),clk(2),clk(1));
    disp(clk);
end
if (graph > 0)
    FigHandle = figure;
else
    FigHandle = 0;
end

%%%%%%%%%%%%%%%%% Generate initial population %%%%%%%%%%%%%%%%%%%
Popul_Genes = GARandSC(Bounds, populSize);

iteration = 0;

RecordFitness           = [];%NaN*ones(Num_cicles,1);     % record - every
     'rec_Iter' iterations a new "best" fitness value is added
RecordGenes             = [];%NaN*ones(Num_cicles,ng);    % record - every
     'rec_Iter' iterations a new "best" gen is added

Pareto_Set              = [];
Pareto_Fit              = [];
Non_Pareto_Set          = [];
Non_Pareto_Fit          = [];
```

```
top_Pareto               = Inf;
top_Pareto_old           = Inf;
TD                       = 0; % flag: 1 is there have been a total
                              %   dominating individual in current population

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Start the cicle %%%%%%%%%%%%%%%%%%%%%%%%%%%%
while iteration < Num_cicles
    iteration = iteration + 1;

     Fitness_Value            = [];
    %++++++++++++ Evaluating the genes
    for i = 1:populSize
        ff = feval(Function, Popul_Genes(i,:));
        Fitness_Value = [ Fitness_Value; ff ];
    end

    %++++++++++++ finding the individuals in Pareto set
     [Pareto_Set, Pareto_Fit, Non_Pareto_Set, Non_Pareto_Fit]  = ...
                         sortPareto( Popul_Genes, Fitness_Value, Pareto_Set,
     Pareto_Fit );

    %++++++++++++ Make a record
     RecordFitness = [ RecordFitness; Pareto_Fit(1,:) ];
     RecordGenes   = [ RecordGenes; Pareto_Set(1,:) ];

    %++++++++++++ Printing some results
     if (iteration == 1)
      old_Pareto_best          = Inf*ones(size(Pareto_Fit(1,:)));
     else
      old_Pareto_best = VisualMO ( visualize, graph, iteration, visual_iter,
     Pareto_Set, Pareto_Fit, old_Pareto_best, FigHandle);
     end

     %XXXXXXXXXXX Selection
     if ( type_Select == 2)
      [SelParents, Pareto_Set_total] = GAParetoOpt (Pareto_Set, Pareto_Fit,
     Non_Pareto_Set, Non_Pareto_Fit, populSize);
     else
      [SelParents, Pareto_Set_total] = GANDominSort (Pareto_Set, Pareto_Fit,
     Non_Pareto_Set, Non_Pareto_Fit, populSize);
     end

     %XXXXXXXXXXX CrossOver
     Cross_Genes = GACrossSC ( SelParents, Pareto_Set_total, populSize-newGenes,
     Mask );

     %XXXXXXXXXXX Mutation
    GMut = GAMutatSC ( Cross_Genes, Bounds, mutatGenes );

     %XXXXXXXXXXX New
     GNew = GARandSC( Bounds, newGenes );

     %XXXXXXXXXXX Reducing number of Pareto optimal solutions
     [Pareto_Set, Pareto_Fit] = ReducePareto(Pareto_Set, Pareto_Fit, maxPareto);

     Popul_Genes = [GNew; GMut];

end % while
%%%%%%%%%%%%%%%%%%%%%%%%%%%% End the cicle %%%%%%%%%%%%%%%%%%%%%%%%%%%%
Result_Fitness = Pareto_Fit;
Result_Genes   = Pareto_Set;
RecordFitness  = RecordFitness(1:rec_Iter:Num_cicles, :);
RecordGenes    = RecordGenes(1:rec_Iter:Num_cicles, :);

fprintf('\n');
if ( visualize >= 0 )
    fprintf (' Fitness Values:\n');
     disp(Result_Fitness);
    fprintf ('\n Genes Values:\n');
    disp(Result_Genes);
    fprintf ('\n');
     if (graph >= 0)
```

```
      if (graph == 0)         FigHandle = figure;
      end
      PlotResultsMO( iteration, FigHandle, Pareto_Fit);
     end
end
return
```

Приложение Б – Програми и модули

CombineOpt

комбиниране на настройките

```
function result = CombineOpt(options,def_opt)
% CombineOpt
%
% Combines User Options with Default Options
%
%    result_Options = CombineOpt (User_options, Default_options)
%
%   Andrey Popov                                andrey.popov@mail.bg
%   www.automatics.hit.bg                       Last update: 28.06.2003

result = def_opt;
% Empty options structure
result.Comment = 'Combination of User defined and Default options';
REAL_STRUCT = struct(  ...
        'Comment',[], ...
        'MaxIter', [], ...
        'PopulSize', [], ...
        'MutatRate', [], ...
        'BestRate', [], ...
        'NewRate', [], ...
        'TolX', [], ...
      'pSelect', [], ...
      'pRecomb', [], ...
      'Select', [], ...
        'RecIter', [], ...
        'Visual', [], ...
        'Graphics',[]);

Names = fieldnames(REAL_STRUCT);
[m,n] = size(Names);
names = lower(Names);

opt_names = lower(fieldnames(options));
try
    for i = 1:m
        k = strmatch(opt_names(i), names);
        if ( k )
            % Identical fields names
            Real_Field_Name = Names{k};
            Value = getfield(options, Real_Field_Name);
            if exist('Value')
                % the parameter is set in options
                if (strcmp(opt_names(i),'visual') |
     strcmp(opt_names(i),'graphics') | strcmp(opt_names(i),'select'))
                    value = lower(Value);
                    result = setfield(result, Real_Field_Name, value);
                else
                    result = setfield(result, Real_Field_Name, Value);
                end
            end
        end
    end
```

```
catch
    V = sprintf('Options structure is not correct.\nSee: help GAopt for detail
     about option settings');
    disp(V);
end
return;
```

TransfGenesSC

преобразуване на гените

```
function Bounds = TransfGenesSC(bounds_orig, TolX)
%    TransformGenesSC
%
%    Transforms Bounds matrix and if necessary the genes
%
%    Bounds = TransformGenesBC(bounds_original, TolX)
%
%   Andrey Popov                                andrey.popov@mail.bg
%   www.automatics.hit.bg                       Last update: 28.06.2003
[n, m] = size(bounds_orig);
Bounds = [];

dB = bounds_orig(:,2) - bounds_orig(:,1);

%-> float variables
if ( m < 3 )       % if tolerance not set
    bounds_orig(:,3) = TolX;
end
Bounds = [];

for ( i = 1:n )
     if ( bounds_orig(i,3) == -1 )                                  % integer
     representation
      Bounds = [Bounds; [dB(i) bounds_orig(i,1:2) 1]];
     else
      Bounds = [Bounds; [dB(i) bounds_orig(i,1:2) 0]];
     end
end

return;
```

TransfGenesBC

преобразуване на гените в разширен тип

```
function Bounds = TransfGenesBC(bounds_orig, TolX)
%    TransformGenesBC
%
%    Transforms Bounds matrix and if necessary the genes
%
%    Bounds = TransformGenesBC(bounds_original, TolX)
%
%   Andrey Popov                                andrey.popov@mail.bg
%   www.automatics.hit.bg                       Last update: 28.06.2003
[n, m] = size(bounds_orig);
Bounds = [];

dB = bounds_orig(:,2) - bounds_orig(:,1);

%-> float variables
if ( m < 3 )       % if tolerance not set
    bounds_orig(:,3) = TolX;
end
for ( i = 1:n )
```

```
    if ( bounds_orig(i,3) > 0 )                                    % extended
    varaible
     tolDegree = 1/bounds_orig(i,3);                               % tolerance for
    variable No. i
     num_genes = ceil(log10( dB(i)*tolDegree + 1 ));% number of genes in
    extended form,

     % which will code the variable
        for j=1:num_genes
            NG = 10^(num_genes-j);
            if ( j==1 )
                b = floor( dB(i) / 10^floor( log10(dB(i)) ));
                B = [b NG];
            else
                B = [9 NG];
            end
            Bounds = [Bounds; [B tolDegree bounds_orig(i,1:2) 1]];
           % [Upper_Bound  End_Flad  Multiplier  Lower_Bound  Upper_Bound
    Var_Type]
        end

    % real number representation
    elseif ( bounds_orig(i,3) == 0 )                               % float
    representation
     Bounds = [Bounds; [dB(i), 1, 1, bounds_orig(i,1:2) 0] ];
    else                                                                %
    integer representation
     Bounds = [Bounds; [dB(i), 1, 1, bounds_orig(i,1:2) 1] ];
     % [Distance  End_Flad  Multiplier  Lower_Bound  Upper_Bound  Var_Type]
    end

end

return;
```

GenesToStandart

преобразуване на гените в нормален вид

```
function StnGenes = GenesToStandart( ExtGenes, Bounds )
% GenesToStandart
%
% Transform Extended Genes to Standard Genes
%
%    StnGenes = GensToStandart( ExtGenes, Bounds_Ext )
%
%   result:
%       StnGenes  - individuals with original (user defined) genes values
%
%   parameters:
%       Bounds_Ext - extendet matrix of the boundaries
%                    [Dist  End_Flag  Multiplier  L_Bound  U_Bound  Var_Type]
%                        Dist = U_Bound - L_Bound     or    Dist = U_Bound
%                        End_Flag = 1 if this is the end of original gene
%                        Multiplier - gain for the subgene
%                        L_Bound  -  lower bound
%                        U_Bound  -  upper bound
%                        Var_Type:  0 - float;  -1 - integer;
%
%
%   Andrey Popov                        andrey.popov@mail.bg
%   www.automatics.hit.bg               Last update: 22.06.2003

StnGenes = []; % genes with standart values (float/integer - not digits)
start_pos = 1;

[ng, mg] = size(Bounds);

% float variables
```

```
for gn = 1:ng
    if Bounds(gn, 2) == 1 % the last digit from the integer is found
        Current_Var = ExtGenes(:,start_pos:gn) * Bounds(start_pos:gn,2) /
     Bounds(gn,3) + Bounds(gn,4);
        StnGenes = [StnGenes Current_Var];
        start_pos = gn + 1;
    end
end

return;
```

GARandSC

генериране на случайни хромозоми

```
% GARandSC
%
% Generates Random strings when Standard CrossOver is Used
%
% FPopul =  GARandSC ( Bounds, Num_Ind )
%
%   result:
%       FPopul    - population of random individuals
%
%   parameters:
%       Bounds    - extendet matrix of the boundaries
%                       [ Distance Lower_Bound Upper_Bound Var_Type]
%                       Distance = Upper_Bound - Lower_Bound
%                   VarType: 1 - integer;  0 - float
%       Num_Ind - number of randomly created individuals
%
%   Andrey Popov                        andrey.popov@mail.bg
%   www.automatics.hit.bg               Last update: 22.06.2003
function [result] = GARandSC(Bounds, num)

if nargin < 2
    error('Wrong number of input parameters');
end

[nb, mb] = size(Bounds);

result = zeros(num, nb);

result(:,1:nb) = ( ones(num,1)*Bounds(:,1)' ).*rand(num,nb) + (
     ones(num,1)*Bounds(:,2)' );

for ( i = 1:nb )
     if ( Bounds(i,4) == 1 )                    % the gen is integer
       result(:,i) = round( result(:,i) );
     end
end

return
```

GARandBC

генериране на случайни хромозоми

```
% GARandBC
%
% Generates Random strings when Blend CrossOver is used
%
% FPopul =  GARandBC ( Bounds_Ext, Bounds_Stn, Num_Ind )
%
%   result:
%       FPopul          - population of random individuals
```

```
%
%   parameters:
%       Bounds_Ext - extendet matrix of the boundaries
%                    [Dist  End_Flag  Multiplier  L_Bound  U_Bound  Var_Type]
%                        Dist = U_Bound - L_Bound     or    Dist = U_Bound
%                        End_Flag = 1 if this is the end of original gene
%                        Multiplier - gain for the subgene
%                        L_Bound  -  lower bound
%                        U_Bound  -  upper bound
%                        Var_Type:  0 - float;  -1 - integer;
%       Bounds_Stn - standart matrix of the boundaries
%                    [Low High]
%       Num_Ind - number of randomly created individuals
%
%   Andrey Popov                        andrey.popov@mail.bg
%   www.automatics.hit.bg               Last update: 22.06.2003
function [result] = GARandBC(BoundsE, BoundsS, num)

if nargin < 3
    error('Wrong number of input parameters');
end

result=[];
[nb, mb] = size(BoundsE);

BoundsS = BoundsS';

while (num > 0)
     RandStn = BoundsS(2,:) + 1;                        % RANDom STaNdart chromosome
     (individual)
     while ( any(RandStn > BoundsS(2,:) ) )
         % the gen is out of boundaries
         RandExt = rand(1,nb).*BoundsE(:,1)'.*1;           % RANDom EXTended
     chromosome

      for ( i = 1:nb )
            if ( BoundsE(i,6) ~= 0 )                       % the gen is not float
                  RandExt(i) = round( RandExt(i) );
            end
      end

      RandStn = GenesToStandart( RandExt, BoundsE );         % Standart Genes
     end

    result = [result; RandExt];
    num = num - 1;
end

return
```

---

sortPareto

намиране на Парето оптималните решения

```
function [Pareto_Set, Pareto_Fit, NPareto_Set, NPareto_Fit]  = ...
                      sortPareto( Popul_Set, Popul_Fit, Pareto_Set, Pareto_Fit )
% sortPareto
%
% Sorting Genes acording to Pareto's criteria
%
% [Pareto_Set, Pareto_Fit, NPareto_Set, NPareto_Fit]  = ...
%                      sortPareto( Popul_Set, Popul_Fit, iPareto_Set, iParetoFit
    )
%
%   result:
%       Pareto_Set   - result individuals in Pareto set
%       Pareto_Fit       - Fitness functions of current Pareto Set
%       NPareto_Set  - individuals which are non in Pareto set
%     NPareto_Fit  - Fintess functions of non Pareto individuals
```

```
%
%   arguments:
%       Popul_Set    - initial Population
%       Popul_Fit    - Fitness Function of initial Population
%       iPatero_Set  - initial members of Pareto_Set
%       iPatero_Fit  - initial fitness functions of Pareto_Set members
%
%   Andrey Popov                                andrey.popov@mail.bg
%   www.automatics.hit.bg                       Last update: 22.06.2003

[num_individ, num_genes] = size(Popul_Set);
[num_indiv, num_fitness] = size(Popul_Fit);
[num_pareto, num_fit]    = size(Pareto_Fit);
% num_pareto  - number of strings in Pareto Set
% num_individ - number of strings in the set
% num_fitness - number of evaluated fitness functions

NPareto_Set = []; % Non-Pareto set (dominated solutions)
NPareto_Fit = [];

start_num = 1;
if num_pareto == 0 % if no previous Pareto Set is available
    Pareto_Fit = Popul_Fit(1,:);
    Pareto_Set = Popul_Set(1,:);
    num_pareto = 1;
    start_num  = 2;
elseif  num_fitness ~= num_fit
    error('Number of fitness functions in Popul_Fit and Pareto_Set is
     different');
end

for i = start_num : num_individ

    count_better = zeros(1,num_fitness);
    to_be_removed = [];
    flag_worst = 0;
    Equal_min  = 0;      % there are few minimumums with equal fitness values but
     different Popul_Set

    for h = 1 : num_pareto
        count_btcp = 0; % better than current pareto
        count_wtcp = 0; % worst than current pareto

        for k = 1 : num_fitness
            if Popul_Fit(i,k) < Pareto_Fit(h,k)
                count_better(k) = count_better(k) + 1;
                count_btcp = count_btcp + 1;
            elseif Popul_Fit(i,k) > Pareto_Fit(h,k)
                count_wtcp = count_wtcp + 1;
            else
                count_better(k) = count_better(k) + 1;
                count_btcp = count_btcp + 1;
                count_wtcp = count_wtcp + 1;
            end
        end

        if count_btcp == num_fitness            % the currently checked element
     in Pareto Set is totaly dominated
            to_be_removed = [to_be_removed h];
        elseif count_wtcp == num_fitness
            flag_worst = 1;
        end

        if Popul_Fit(i,:) == Pareto_Fit(h,:)     % The Function sets are
     identical
            if Popul_Set(i,:) == Pareto_Set(h,:)
            else                                 % The Genes are identical
                Equal_min = 1;
                to_be_removed = to_be_removed(1:(length(to_be_removed)-1));
            end
        end
```

```
    end

    if sum(count_better) >= num_fitness*num_pareto & Equal_min == 0
        % Current Popul_Fit is better than all from Pareto_Fit
        NPareto_Set = [Pareto_Set; NPareto_Set];
        NPareto_Fit = [Pareto_Fit; NPareto_Fit];
        Pareto_Set = Popul_Set(i,:);
      Pareto_Fit = Popul_Fit(i,:);
        num_pareto = 1;

    elseif ( max(count_better) >= 1 & flag_worst == 0) | Equal_min
        % Current_Popul_Fit is better than some and worst than others from
     Pareto_Fit
        num_to_rem = length(to_be_removed);

        if num_to_rem ~= 0      % removing the dominated Genes from Pareto_Set
            for v = num_to_rem:-1:1
              NPareto_Set = [NPareto_Set; Pareto_Set( to_be_removed(v),: )];
              NPareto_Fit = [NPareto_Fit; Pareto_Fit( to_be_removed(v),: )];
                Pareto_Set( to_be_removed(v),: ) = [];
                Pareto_Fit( to_be_removed(v),: ) = [];
                num_pareto = num_pareto - 1;
            end
        end

        Pareto_Fit = [Pareto_Fit; Popul_Fit(i,:)];
        Pareto_Set = [Pareto_Set; Popul_Set(i,:)];
        num_pareto = num_pareto + 1;
    else
        NPareto_Set = [NPareto_Set; Popul_Set(i,:)];
        NPareto_Fit = [NPareto_Fit; Popul_Fit(i,:)];
    end

end

return
```

GASelectFP

Пропорционална на целевата функция селекция

```
% GASelectFP
%
% Fitness Proportional Select + Roulette Wheel
%
% Parents  =  GASelectFP ( Fitness, Num )
%
%   result:
%       Parents - choosen parent chromosomes
%                   matrix with couples [Parent1 Parent2]
%
%   arguments:
%     Fitness - Fitness Values of all Parents
%       Num     - number of children chromosomes to be produced
%
%
%   Andrey Popov                      andrey.popov@mail.bg
%   www.automatics.hit.bg             Last update: 19.06.2003
function [result] = GASelectFP ( Fitness, Num, param )

if (nargin < 2)
    error('Wrong number of input parameters');
end

% we need just half of the pairs --> 1 pair = 2 children
Num = ceil(Num/2);

[ num, N ] = size(Fitness);
```

```
% finding the sum of positiv and negative Fitness functions
LB=0; UB=0;    % Lower and Upper Boundary
for i = 1:num
     if ( Fitness(i) > 0 )
      UB = UB + Fitness(i);
     else
      LB = LB + Fitness(i);
     end
end

% recalculating the Fitness values
fit_new = (UB - LB)*ones(num,1) - Fitness;
Scale = sum(fit_new);

result=[];
M = Num;

while M > 0
     % random number with uniform distribution
     rh = rand*Scale;
     H = Roulette( rh, fit_new, num );
     K = H;
     while (K == H)
      rk = rand*Scale;
      K = Roulette( rk, fit_new, num );
     end

     M = M - 1;
     result = [result; H, K];
end %while

return
```

GASelectRS

Рангова селекция

```
% GASelectRS
%
% Ranking Selection + Roulette Wheel
%
% Parents =  GASelectRS ( populSize, Num, b )
%
%   result:
%       Parents   - choosen parent chromosomes
%                     matrix with couples [Parent1 Parent2]
%
%   arguments:
%     populSize - Number of possible Parents
%       Num       - number of couples (respectively number of children)
%     b             - parameter betta
%
%    Selection probability is evaluated by:
%     P = b*(1-b)^(rank-1)
%           where rank = 1 for the best individual, 2 for the second, etc.
%
%
%   Andrey Popov                        andrey.popov@mail.bg
%   www.automatics.hit.bg               Last update: 27.06.2003
function [result] = GASelectRS ( populSize, Num, b )

if (nargin < 3)
    error('Wrong number of input parameters');
end
if (b==0 | b==1)
    error ('Parameter b could not be 0 or 1. Please change options.pSelect');
end

% we need just half of the pairs --> 1 pair = 2 children
```

```
Num = ceil(Num/2);

Fit = b*(1-b).^(0:populSize - 1);

Scale = sum(Fit);

result=[];
M = Num;

while M > 0
     % random number with uniform distribution
     rh = rand*Scale;
     H = Roulette( rh, Fit, populSize );
     K = H;
     while (K == H)
      rk = rand*Scale;
      K = Roulette( rk, Fit, populSize );
     end

     M = M - 1;
     result = [result; H, K];
end %while

return
```

GASelectN

Гаусова селекция

```
% GASelectN
%
% Normal Law Distribution Selection (Gaussian Selection)
%
% Parents =  GASelectN ( populSize, Num, StDev )
%
%   result:
%       Parents   - choosen parent chromosomes
%                     matrix with couples [Parent1 Parent2]
%
%   arguments:
%     populSize - Number of possible Parents
%       Num       - number of couples (respectively number of children)
%     StDev     - Standart Deviation
%
%
%   Andrey Popov                      andrey.popov@mail.bg
%   www.automatics.hit.bg             Last update: 19.06.20003
function [result] = GASelectN ( populSize, Num, StDev )

if (nargin < 2)
    error('Wrong number of input parameters');
elseif (nargin < 3)
     StDev = 1;
end

M_C = populSize*StDev;

result=[];
num = ceil(Num/2);

while num > 0
      % random number with normal distribution
        rn = 2;
        while rn > 1
            rn = abs(randn)/4;
        end
        H = ceil ( rn * M_C);
        K = H;
        while ( K == H )
```

```
            rn = 2;
            while rn > 1
                rn = abs(randn)/4;
            end
            K = ceil ( rn * M_C);
        end

      result = [result; H, K];
      num=num-1;

end %while

return
```

Roulette

Ролетно колело

```
% Roulette Wheel
%
% Calculates which parent is taken
%
%    ParentNum = Roulette( randN, Fitness, Num_Par )
%
%   result:
%       ParentNum - choosen parent chromosome
%
%   arguments:
%     randN - random number between 0 and sum of all Fitness functions
%     Fitness - vector with fitness functions of the parents
%     Num_Par - Number of possible Parents
%
%
%   Andrey Popov                        andrey.popov@mail.bg
%   www.automatics.hit.bg               Last update: 27.06.2003
function ParentNum = Roulette( randN, fitness, Num_Par )

W = 0;
for i = 1:Num_Par
     if ( W >= randN )
       break;
     end
     W = W + fitness(i);
end
ParentNum = i;
return
```

GAParetoOpt

Парето оптимална селекция

```
function [SelParents, Pareto_Set_total] = GAParetoOpt (Pareto_Set, Pareto_Fit,
     Non_Pareto_Set, Non_Pareto_Fit, populSize)
%    Genetic Algorithm
%           Pareto Optimality Selection
%
%    [PARENTS, GENES_ALL] = GAParetoOpt (Pareto_Set, Pareto_Fit, ...
%                                    Non_Pareto_Set, Non_Pareto_Fit, populSize)
%
%   result:
%       PARENTS    - choosen parent chromosomes
%                    matrix with couples [Parent1 Parent2]
%     GENES_ALL  - List of all Genes
%
%   arguments:
%     Pareto_Set     - Pareto Optimal Set
```

```
%     Pareto_Fit     - Pareto Otpimal Fitness Values
%     Non_Pareto_Set - The other genes
%     Non_Pareto_Fit - Fitness value of Non_Pareto_Set
%       populSize          - Population Size
%
%   Andrey Popov                          www.automatics.hit.bg
%   andrey.popov@mail.bg                  Last update: 27.06.2003
[numPareto, x] = size(Pareto_Set);

if (numPareto <= 2)          % only one member in Pareto set
     [Pareto_Set2, Pareto_Fit2, Non_Pareto_Set, Non_Pareto_Fit]  = ...
                  sortPareto( Non_Pareto_Set, Non_Pareto_Fit, [], [] );
     Pareto_Fit_total = [Pareto_Fit; Pareto_Fit2];
     Pareto_Set_total = [Pareto_Set; Pareto_Set2];
else
     Pareto_Fit_total = Pareto_Fit;
     Pareto_Set_total = Pareto_Set;
end

%XXXXXXXXXXX Selection
SelParents  = GAMOSelect ( Pareto_Fit_total, populSize, numPareto );

return
```

GANDominSort

Недоминирано сортиране - селекция

```
function [SelParents, Pareto_Set_total] = GANDominSort (Pareto_Set, Pareto_Fit,
     Non_Pareto_Set, Non_Pareto_Fit, populSize)
%    Genetic Algorithm
%           Non-Dominated Sorting Selection
%
%    [PARENTS, GENES_ALL] = GAParetoOpt (Pareto_Set, Pareto_Fit, ...
%                                  Non_Pareto_Set, Non_Pareto_Fit, populSize)
%
%   result:
%       PARENTS    - choosen parent chromosomes
%                    matrix with couples [Parent1 Parent2]
%     GENES_ALL  - List of all Genes
%
%   arguments:
%     Pareto_Set     - Pareto Optimal Set
%     Pareto_Fit     - Pareto Otpimal Fitness Values
%     Non_Pareto_Set - The other genes
%     Non_Pareto_Fit - Fitness value of Non_Pareto_Set
%       populSize          - Population Size
%
%   Andrey Popov                          www.automatics.hit.bg
%   andrey.popov@mail.bg                  Last update: 27.06.2003
[numPareto, x] = size(Pareto_Fit);
Fitness = ones(numPareto, 1);% assigning rank to the Pareto optimal solutions
Pareto_Set_total = Pareto_Set;

i = 2;
[numNonPareto, x] = size(Non_Pareto_Set);

while ( numNonPareto>0 )
     [Pareto_Set_, Pareto_Fit_, Non_Pareto_Set, Non_Pareto_Fit]  = ...
                  sortPareto( Non_Pareto_Set, Non_Pareto_Fit, [], [] );

     [numPareto, x] = size(Pareto_Fit_);
     Fitness = [Fitness; ones(numPareto,1)*i];
     Pareto_Set_total = [Pareto_Set_total; Pareto_Set_];

     [numNonPareto, x] = size(Non_Pareto_Set);

     i = i + 1;
end
```

```
Fitness = i - Fitness;
% renumbering (Pareto optimal solutions receive heightest rank)

SelParents = [];

M = ceil(populSize/2);
Scale = sum(Fitness);
[Num_Par,x] = size(Fitness);

while M > 0
     % random number with uniform distribution
     rh = rand*Scale;
     H = Roulette( rh, Fitness, Num_Par );
     K = H;
     while (K == H)
       rk = rand*Scale;
       K = Roulette( rk, Fitness, Num_Par );
     end

     M = M - 1;
     SelParents = [SelParents; H, K];
end %while

return
```

ReducePareto

Намаляване броя на Парето оптималните решения

```
function [Pareto_Set, Pareto_Fit] = ReducePareto(Pareto_Set, Pareto_Fit,
     Max_Number);
% ReducePareto
%
% Reducing the number of Pareto Optimal Solutions
%
% [New_P_Set, New_P_Fit] = ReducePareto(Pareto_Set, Pareto_Fit, Max_Number)
%
%   Output:
%       New_P_Set  - reduced Pareto Set
%       New_P_Fit  - reduced Pareto Fitness
%
%   Input:
%     Pareto_Set - Pareto Set to be reduced
%       Pareto_Fit - Pareto Fitness
%     Max_Number - Number of solutions to remain
%
%
%   Andrey Popov                                  andrey.popov@mail.bg
%   www.automatics.hit.bg                         Last update: 28.06.2003
[num_Pareto, num_Fit] = size (Pareto_Fit);

while (num_Pareto > Max_Number)
     D = zeros(num_Pareto, 1);      % vector with the neighbor distance
     M = zeros(num_Pareto, num_Pareto);  % matrix with distance to the other
     solutions
     for i = 1:num_Pareto
       for k = i:num_Pareto
             if k == i
                   M(k,k) = Inf;
             else  % Euclidian distance
                   M(k,i) = norm( Pareto_Fit(i,:) - Pareto_Fit(k,:) );
                   M(i,k) = M(k,i);
             end
       end
     end
     for i = 1:num_Pareto
       [m1, ind1] = min(M(i,:));
       M(i,ind1) = Inf;
```

```
      [m2, ind2] = min(M(i,:));
      D(i) = ( m1 + m2 )/2;
     end
     [d, ind] = min(D);
     Pareto_Fit(ind, :) = [];
     Pareto_Set(ind, :) = [];
     num_Pareto = num_Pareto - 1;
end

return
```

GACrossSC

Стандартно кръстосване

```
% GACrossSC
%
% CrossOver Operation - Standard Crossover
%
% FPopul =  GACrossSC ( SelPar, IPopul, Num, Mask )
%
%   result:
%       FPopul  - Offspring (population after CrossOver)
%
%   parameters:
%     SelPar  - matrix with selected parents [Parent1 Parent2]
%       IPopul  - Population with parent individuals ( P x N matrix )
%                       P - number of parents
%                       N - number of elements in chromosome
%       Num     - number in 'children' in next generation
%       Mask    - 1xN boolean vector - mask showing which elements are changed in
    gen
%                 if MASK is empty or missing than random mask is used
%
%   Andrey Popov                                        www.automatics.hit.bg
%   andrey.popov@mail.bg                      Last update: 22.06.2003
function [result] = GACrossSC ( SelParents, InitPopul, Num, Mask )

     if (nargin < 3)
        error('Wrong number of input parameters in GACrossSC');
     end

     [ m,N ] = size(InitPopul);
     if (m==1)
      error('Only 1 gen in GENS matrix');
     end
     result=[];

     if ( nargin == 3 & ~isempty(Mask) ) | ( N == 2 )
      % Used Defined crossover Mask is used
      if (N == 2)
            Mask = [1 0];
      else
            [ u,n ] = size(Mask);
            if (n ~= N)
              error(' GENS and MASK matrix are not with the same dimension');
          end
      end
      MaskType = 0;
         Mask_ = Mask < 0.5;
     else
      MaskType = 1;
     end

     i = 1;
     while Num > 0
      H = SelParents(i,1);
      K = SelParents(i,2);
```

```
      if ( MaskType == 1 )    % random mask is used
             Mask = rand(1,N) < 0.5;
             Mask_ = Mask < 0.5;
      end

      Q = Mask.*InitPopul(H,:);
      W = Mask_.*InitPopul(K,:);
         result = [result; Q + W];
         Num = Num - 1;

         if ( Num==0 )
            break;
         end

         Q = Mask.*InitPopul(K,:);
         W = Mask_.*InitPopul(H,:);

         result = [result; Q + W];
         Num = Num - 1;
      i = i + 1;
     end %while

return
```

GARecombBC

Кръстосване със смесване

```
% GARecombBC
%
% Recombination - Blend CrossOver
%
% FPopul =  GARecombBC ( SelPar, IPopul, Num, VarType, Alfa )
%
%   result:
%       FPopul  - Offspring (population after CrossOver)
%
%   parameters:
%     SelPar  - matrix with selected parents [Parent1 Parent2]
%       IPopul  - Population with parent individuals ( P x N matrix )
%                       P - number of parents
%                       N - number of elements in chromosome
%       Num     - number of 'children' in next generation
%     VType - variable type
%                       0 - float;  1 - integer
%     Alfa  - exploration coeficient
%                       Child1 = y * Parent1 + (1-y) * Parent2
%                       Child2 = (1-y) * Parent1 + y * Parent2
%                       y = (1 + 2*Alfa)*rand - Alfa
%
% Andrey Popov                                      www.automatics.hit.bg
% andrey.popov@mail.bg                                    Last update: 22.06.2003
function [result] = GARecomb ( SelParents, InitPopul, Num, VType, Alfa )

if (nargin < 5)   Alfa = 0;
     if (nargin < 3)          error('Wrong number of input parameters');
     end
end

if (nargin < 4 | isempty(VType))
     [n,m] = size(InitPopul);
     VType = ones(m,1);
end
result=[];
M = Num;    % number of childrens (remaining)
i = 1;            % parent couple number

while M > 0
     H = SelParents(i,1);
```

```
     K = SelParents(i,2);

     % random number for the crossover
     RN = ( 1 + 2*Alfa)*rand - Alfa;

    cGen   = RN*InitPopul(H,:) + (1-RN)*InitPopul(K,:);   % current child
    result = [result; cGen];
     M      = M - 1;

     if ( M==0 )
         break;
    end

    cGen   = (1-RN)*InitPopul(H,:) + RN*InitPopul(K,:);
    result = [result; cGen];
    M      = M - 1;
     i      = i + 1;
end %while

for ( i = 1:length(VType) )
     if ( VType(i) == 1 )
      result(:,i) = round(result(:,i));
     end
end

return
```

GAMutatSC

Мутация при стандартно кръстосване

```
% GAMutatSC
%
% Generates random Mutations when Standard CrossOver is Used
%
% FPopul = GAMutatSC ( IPopul, Bounds, Num_Mut )
%
%   result:
%       FPopul     - population after the mutation
%
%   parameters:
%       IPopul   - initial population
%       Bounds     - extendet matrix of the boundaries
%                       [ Distance Lower_Bound Upper_Bound Var_Type]
%                           Distance = Upper_Bound - Lower_Bound
%                           VarType: 1 - integer;  0 - float
%       Num_Mut  - total number of mutations for the population
%
%   Andrey Popov                                    www.automatics.hit.bg
%   andrey.popov@mail.bg                    Last update: 20.06.2003
function result = GAMutatSC(InitPopul, Bounds, num)

if nargin < 3
    error('Wrong number of input parameters. See "help GAMutatSC"');
end

result   = InitPopul;
[nb, mb] = size(Bounds);
[mg, ng] = size(InitPopul);
% mg - number of chromosomes
% ng - number of genes in a chromosome

if (nb ~=  ng)
    error('Number of bounds NOT equal to number of elements in the string');
end

while ( num > 0 )
     n = ceil(rand*ng);   % number of the gen to mutate
    m = ceil(rand*mg);   % number of the chromosome to mutate
```

```
     if ( Bounds(n,4) == 1 )  % integer
      R = round( Bounds(n,1)*rand ) + Bounds(n,2);
     else                          % float
      R = Bounds(n,1)*rand + Bounds(n,2);
     end

    result(m,n) = R;

    num = num - 1;
end

return
```

GAMutatBC

Мутация при кръстосване със смесване

```
% GAMutatBC
%
% Generates random Mutations when Blend CrossOver is Used
%
% FPopul = GAMutatBC ( IPopul, Bounds_Ext, Bounds_Stn, Num_Mut )
%
%   result:
%       FPopul       - population after the mutation
%
%   parameters:
%       IPopul     - initial population
%       Bounds_Ext - extendet matrix of the boundaries
%                    [Dist  End_Flag  Multiplier  L_Bound  U_Bound  Var_Type]
%                        Dist = U_Bound - L_Bound     or    Dist = U_Bound
%                        End_Flag = 1 if this is the end of original gene
%                        Multiplier - gain for the subgene
%                        L_Bound  -  lower bound
%                        U_Bound  -  upper bound
%                        Var_Type:  0 - float;  -1 - integer;
%       Bounds_Stn - standart matrix of the boundaries
%                    [Low High]
%       Num_Mut  - total number of mutations for the population
%
%   Andrey Popov                                    www.automatics.hit.bg
%   andrey.popov@mail.bg                    Last update: 22.06.2003
function [result] = GAMutatBC(InitPopul, BoundsE, BoundsS, num)

if nargin < 3
    error('Wrong number of input parameters. See "help GAmutatBC"');
end

result   = InitPopul;
[nb, mb] = size(BoundsE);
[mg, ng] = size(InitPopul);
% mg - number of InitPopul
% ng - number of strings in a gen

if (nb ~=  ng)
    error('Number of bounds NOT equal to number of elements in the string');
end
BoundsS = BoundsS';

while ( num > 0 )
     n = ceil(rand*ng);       % number of the gen to mutate
    m = ceil(rand*mg);        % number of the chromosome to mutate
     GEN = InitPopul(m,:);    % taking the gen to mutate

     GenStn = BoundsS(2,:) + 1;

    while ( any(GenStn > BoundsS(2,:) ) )                % the upper boundary
     is exceeded
```

```
       % the gen is out of boundaries
      GEN(1,n) = rand*BoundsE(n,1);                          % mutation of the gen
      if ( BoundsE(n, 6) ~= 0 )                              % extended type
         GEN(1,n) = round ( GEN(1,n) );                % gen to integer
      end

         GenStn = GenesToStandart( GEN, BoundsE );    % Transform to standart
     InitPopul
     end

    result(m,:) = GEN;

    num = num - 1;
end

return
```

VisualSO

Визуалзиация при еднокритериална минимизация

```
function [bestFitnessValue_old] = VisualSO( visualize, graph, iteration, ...
            visual_iter, bestFitnessValue, bestFitnessValue_old, BESTGens, ...
            FigHandle, RecordFitness)

if ( visualize >= 0 )   % temporary result allowed?
     if ( visualize == 1 )    % SOME results
      if bestFitnessValue >= 0
            if bestFitnessValue < bestFitnessValue_old/2 | (~rem(iteration,
     visual_iter))
                  DispResultsSO( iteration, bestFitnessValue, BESTGens );
                  if ( graph > 0 )
                        PlotResultsSO( iteration, FigHandle, RecordFitness );
                  end
                  bestFitnessValue_old = bestFitnessValue;
            end
      else
            if bestFitnessValue < bestFitnessValue_old*2 | (~rem(iteration,
     visual_iter))
                  DispResultsSO( iteration, bestFitnessValue, BESTGens )
                  if ( graph > 0 )
                        PlotResultsSO( iteration, FigHandle, RecordFitness );
                  end
                  bestFitnessValue_old = bestFitnessValue;
            end
      end
     elseif ( visualize == 2 )      % ALL results
      DispResultsSO( iteration, bestFitnessValue, BESTGens );
      if ( graph > 0 )
            PlotResultsSO( iteration, FigHandle, RecordFitness );
      end
     else % NO results, only dots sign action is taking place
      if ( rem(iteration, 20) == 0 )       fprintf('.');
            if (rem(iteration, 1000) == 0)   fprintf('\n');
            end
      end
     end
end % if visualize
return
```

VisualMO

Визуалзиация при многокритериална минимизация

```
function old_Pareto_best = VisualMO( visualize, graph, iteration, visual_iter,
     ...
```

```
                        Pareto_Set, Pareto_Fit, old_Pareto_best, FigHandle);

if ( visualize >= 0 )   % temporary result allowed?
     if ( visualize == 1 )          % some results
      if ( old_Pareto_best > Pareto_Fit(1,:) )
            TD = 1;
            old_Pareto_best = Pareto_Fit(1,:);
      else  TD = 0;
      end
      if ( TD == 1 | (~rem(iteration, visual_iter) ) )
            DispResultsMO( iteration, Pareto_Fit, Pareto_Set, 0 );
            if ( graph > 0 )
                  PlotResultsMO( iteration, FigHandle, Pareto_Fit );
            end
      end
     elseif ( visualize == 2 )      % all results
      DispResultsMO( iteration, Pareto_Fit, Pareto_Set, 1 );
      if ( graph > 0 )
            PlotResultsMO( iteration, FigHandle, Pareto_Fit );
      end
     else % no results, only dots sign action is taking place
      if ( rem(iteration, 20) == 0 )       fprintf('.');
            if (rem(iteration, 5000) == 0)   fprintf('\n');
            end
      end
     end
end % if visualize

return
```

DispResultsSO

Разпечатване на междинни резултати при еднокритериална минимизация

```
function DispResultsSO( iteration, FitnessValue, GenesSet )
    V = sprintf(' %%7.0f   %%9.6g');
    for i = 1:length(GenesSet)
        V = [V sprintf('   %%5.6g')];
    end
    V = sprintf(V,iteration, FitnessValue, GenesSet);
    disp(V);
return
```

DispResultsMO

Разпечатване на междинни резултати при многокритериална минимизация

```
function DispResultsMO( iteration, Pareto_Fit, Pareto_Set, Flag )
    V = sprintf('Iter:%%5.0f ##>Fit:');
     [n,m] = size(Pareto_Fit);
     for i = 1:m
        V = [V sprintf('   %%8.4g')];
    end
     [n,m] = size(Pareto_Set);
     V = [V sprintf('   ##>Genes:')];
     for i = 1:m
        V = [V sprintf('   %%5.6g')];
    end

     if (Flag == 0)
        MSG = sprintf(V,iteration, Pareto_Fit(1,:), Pareto_Set(1,:));
     else
      MSG = [];
```

```
      for i=1:n
            if (i==n)
                  MSG = [MSG sprintf(V,iteration, Pareto_Fit(i,:),
     Pareto_Set(i,:))];
            else
                  MSG = [MSG sprintf([V 13],iteration, Pareto_Fit(i,:),
     Pareto_Set(i,:))];
            end
      end
     end
    disp(MSG);
return
```

PlotResultsSO

Графични визуализиране при еднокритериална минимизация

```
function PlotResultsSO( iteration, FigHandle, RecordFitness )
     figure(FigHandle);
         plot(RecordFitness,'ro', 'MarkerSize', 5); xlabel('generation');
     ylabel('Fitness Value');
      text(0.65,0.9,['Best = ',
     num2str(RecordFitness(iteration))],'Units','normalized');
    drawnow;
return
```

PlotResultsMO

Графични визуализиране при многокритериална минимизация

```
function PlotResultsMO ( iteration, FigHandle, Pareto_Fit)
     figure(FigHandle);
     [n,m] = size(Pareto_Fit);
     if (m==2)          % 2-D
      MSG = sprintf('Fitness Values\nIteration %d; Individs in Pareto set =
     %d',iteration,n);
      plot(Pareto_Fit(:,1),Pareto_Fit(:,2),'ro','MarkerSize', 5);grid;
      title(MSG);xlabel('Fintess 1');ylabel('Fitness 2');
     elseif (m==3)      % 3-D
      MSG = sprintf('Fitness Values\nIteration %d; Individs in Pareto set =
     %d',iteration,n);
      plot3(Pareto_Fit(:,1),Pareto_Fit(:,2),Pareto_Fit(:,3),'ro','MarkerSize',
     5);grid;
      title(MSG);xlabel('Fintess 1');ylabel('Fitness 2');zlabel('Fitness 3');
     end
     drawnow;
return
```

Приложение В – Допълнителни програми

GAopt

Програма за зареждане, запазване и промяна на настройките

```
function result = GAopt ( varargin )
% Genetic Algorithm - Options
%
% Sets options for Genetic Algorithm
% Options = GAopt(Options, 'Param1', Value1, 'Param2', Value2, ... )
```

```
%
% Options = GAopt(Dafault);
%
% Options = GAopt(Options, Save);
%
%      Parameters
%  ----------------
%         *   Some of the Parameters are percents from the Population Size
     (GensPopul)
%         **  The Default values are in breckets. Example: {50}
%
%      Comment     - comment about the options (up to 50 characters)
%      MaxIter     - maximal number of population (iterations)
%                    [ positive scalar ] {1000}
%      PopulSize   - number of individs (gens) in population
%                    [ positive scalar ] {50}
%      MutatRate   - mutation rate: number of mutations in the population =
     MutatRate*PopulSize
%                    [ positive scalar ] {0.3}
%      BestRate    - percentage of gens that copied to next genertion (survive)
%                    (optimal result is preserved)
%                    [ positive scalar (0 - 1) ] {0.1}
%      NewRate     - percentage of newly generated gens in population
%                    [ positive scalar ] {0.1}
%      TolX        - default precision for the variables - (used when in bounds
     the thirs column
%                    is not set). If TolX = 1 then integer variables are used.
%                    [ positive scalar ] {1e-4}
%
%       pSelect      - Select Parameter - usage depends on the used selection type
%       pRecomb      - Recombination Parameter - usage depends on the used
     recombination type
%                               See Manual or recombination function's help for
     details
%
%       Select      - type of selection used - depends on the minimization
     function used
%                        [ positive scalar ] { 1 }
%
%      RecIter     - record of Fitness and Gens is made through RecIter itaration
%                    [ positive integer ] {1}
%      Visual      - visualization of temporary results
%                    [ none | off | {some} | all ]
%      Graphics    - graphical representation of temporary results - only when
     Visual is
%                    set to 'some' or 'all'
%                    [ off | {on} | final ]
%
%       Options
%   ----------------
%      Load        - Loads default or user deffined settings saved in "GAopt.mat"
%                    If used with other properties, the properties not listed are
     taken from
%                    default properties. Load is integer in range:
%                    ( -9 : 0 ) -> Default properties
%                    (  1 : 9 ) -> UserDefined
%      Save        - Saves Options as a User Defined properties in "GAopt.mat"
%                    Save:   ( 1 : 9 )
%
%      Examples
%  -----------------
%   > GAopt             - shows the Comment of all stored options
%   > opt = GAopt(-5)   - loads default options -5
%   > opt = GAopt(7)    - loads user defined options 7
%   > GAopt(opt, 4)     - saves options in 'opt' on place 4 in GAopt.m
%   > opt = GAopt('BestRate',0.17, 'Visual','no') - updates opt: sets BestRate to
     0.17
%                                                    and stops visualization
%   > opt.MutatRate = 0.2 - sets mutation rate to 0.2
%
%   Andrey Popov                          www.automatics.hit.bg
%   andrey.popov@mail.bg                  Last update: 27.06.2003
```

```
try
     % load options from MAT file
     load GAopt.mat OPTIONS
catch
    error('Could not open "GAopt.mat"! File is missing or corrupt');
end

% Print out list with setting's comments
if (nargout == 0 & nargin==0)
    fprintf('\n No output arguments. For default values use: Options = GAopt\n
     For more information see: help GAopt\n\n   ==>List of recorded
     settings<==');
    try
        for i = 1:20
            fprintf('\n %2.0f -> %s',i-10,OPTIONS(i).Comment);
        end
        return
    catch
        error('Problem reading GAopt.mat. File is missing or corrupted');
    end
end

true = 1; false = 0;
LoadDefault = false;

if (nargin == 0)
    % no input argumenets => default options #0 are used
    Load = 0;
    LoadDefault = true;
elseif (nargin == 1)
    %only 1 input argument
        tmp = varargin{1};
        try
            tmp = round(tmp);
            if (abs(tmp) <= 9)
                Load = tmp;
                LoadDefault = true;
            else
                error('Default is integer between -9 and 9. See: Default paremter
     in help GAopt');
            end
        catch
            error('Wrong arguments. See: Default paremter in help GAopt');
        end
end

if (LoadDefault)
    result = OPTIONS(Load+10);
    if (Load)
%        V = sprintf('Loaded options %d\n->%s<-', Load, result.Comment);
     disp(V);
    end
    return;
end

if (nargin == 2)
    try
        options = varargin{1};
        Save = varargin{2};
        tmp = round(Save);
        if (tmp > 0 & tmp < 10)
            OPTIONS(Save+10) = options;
            try
                save GAopt.mat OPTIONS
            catch
                error('Could not write "GAopt.mat"! File is missing or write
     protected');
            end
            result = options;
%            V = sprintf('Saved options on %d\n->%s<-', Save,
     options.Comment);disp(V);
```

```
            return;
        else
            error('Save should be between 1 and 9. See: Save parameter in help
     GAopt');
        end
    catch
        error('Wrong arguments. See: Default paremter in help GAopt');
    end
end

if ( rem(nargin, 2)~=1 )
    % not correct pairs
    V = sprintf('Input arguments should be by couples:\noptions = GAopt(options,
     Param1, Value1, Param2, Value2...)');
    error(V);
end

try
    % update the fields
    old_opt = varargin{1};

    % Empty options structure
    options = struct(  ...
        'Comment',[], ...
        'MaxIter', [], ...
        'PopulSize', [], ...
        'VarType', [], ...
        'MutatRate', [], ...
        'BestRate', [], ...
        'NewRate', [], ...
        'TolX', [], ...
      'pSelect', [], ...
      'pRecomb', [], ...
      'Select', [], ...
        'RecIter', [], ...
        'Visual', [], ...
        'Graphics',[]);

    Names = fieldnames(options);
    [m,n] = size(Names);
    names = lower(Names);

    for i=2:2:nargin
        Field_Name = varargin{i};
        field_name = lower(Field_Name);
        k = strmatch(field_name, names,'exact');
        if (k)
            % there is such field in the structure
            Real_Field_Name = Names{k,:};
            Value = varargin{i+1};
            old_opt = setfield(old_opt, Real_Field_Name, Value);
       else
            V = sprintf('There is no structure field named %s\nCheck: help GAopt
     for details',Field_Name);
            error(V);
        end
    end
    result = old_opt;
catch
    V = sprintf('There are inpropriate Prameters or Values\nPlease check: help
     GAopt');
    error(V);
end

return;
```

ParetoNumber

Номериране на Парето оптималните решения

```
function ParetoNumber( Pareto_Fit )
[n,m] = size(Pareto_Fit);
if (m <= 3)
     figure;
     if (m==2)          % 2-D
      MSG = sprintf('Fitness Values\nIndivids in Pareto set = %d',n);
      plot(Pareto_Fit(:,1),Pareto_Fit(:,2),'ro','MarkerSize', 5);grid;
      title(MSG);xlabel('Fintess 1');ylabel('Fitness 2');
      A = axis;
      for i = 1:n
            P = zeros(2,1);
            for k = 1:m
                  P(k) = ( Pareto_Fit(i,k) - A(2*k-1) )/(A(2*k) - A(2*k-1))*1.01;
            end
            MSG = sprintf('%d',i);
            text(P(1),P(2), MSG, 'Units','normalized');
      end
     elseif (m==3)      % 3-D
      MSG = sprintf('Fitness Values\nIndivids in Pareto set = %d',n);
      plot3(Pareto_Fit(:,1),Pareto_Fit(:,2),Pareto_Fit(:,3),'ro','MarkerSize',
     5);grid;
      title(MSG);xlabel('Fintess 1');ylabel('Fitness 2');zlabel('Fitness 3');
     end
     drawnow;
end
return
```