PARAMETER OPTIMIZATION WITH GENETIC FUZZY SYSTEMS

Georgieva Penka V., Burgas Free University, pgeorg@bfu.bg

Abstract: Genetic fuzzy systems are hybrid systems in artificial intelligence, which are created to combine the advantages of fuzzy systems and genetic algorithms. The degree of flexibility, that genetic algorithms implement, makes them suitable for optimization of fuzzy systems and development of systems for decision-making related to the diagnosis, monitoring and control. This article briefly traced the change of the concept of mathematical modeling, as well as basic concepts of artificial intelligence. Publishing reference and citation activity in these areas is shown, based on data from Web of Science. The main types of fuzzy systems are presented, as well as the concept of genetic fuzzy system. Successful interaction of genetic algorithms with existing simulations and models is realized on a previously created fuzzy rule-based system FSSAM. Part of the source code is displayed and some results from the application of the fuzzy genetic system for parameters optimization are presented.

Keywords: FSSAM, genetic fuzzy system, mathematical and computer modeling

ОПТИМИЗИРАНЕ НА ПАРАМЕТРИ С ГЕНЕТИЧНИ РАЗМИТИ СИСТЕМИ

Пенка В. Георгиева, Бургаски свободен университет, pgeorg@bfu.bg

Абстракт: Генетичните размити системи са хибридни системи от изкуствения интелект, създавани за комбиниране на предимствата на размитите системи и генетичните алгоритми. Степента на гъвкавост, която генетичните алгоритми притежават, ги прави подходящи при оптимизирането на размити системи и разработване на системи за взимане на решения, свързани с диагностика, наблюдение и управление. В тази статия накратко е проследено изменението на концепцията за математически моделиране, както и на основните идеи на изкуствения интелект. Предложени са справки за публикационната активност и цитируемостта в тези направления, като са използвани данните от базата Web of Science. Предсатвени са основните видове размити системи, както и концепцията за генетична размита система. Успешното взаимодействие на генетичните алгоритми със съществуващи симулации и модели е реализирано върху предварително създадената размита система, базирана на правила, и това е системата FSSAM. Показан е част от изходния сорс код и са представени някои резултати от прилагането на размитата генетична система при оптимизиране на параметри.

Ключови думи: FSSAM, генетична размита система, математическо и компютърно моделиране

1. Въведение- модели, математическо моделиране, изкуствен интелект, софт компютинг

Историческото развитие на науката е пряко свързано с моделирането и в частност с математическото моделиране. В развитието на човешката цивилизация могат ясно да се разграничат 3 етапа на развитие на математическото моделиране.

Първият етап е до възникването на понятията граница, безкрайно малка и безкрайно голяма величина, непрекъсната величина. Тогава най-важните резултати са свързани с моделирането на траекториите на движение на планетите, със създаването на геометрични конструкции и с изобретяването на различни техники за извършване на изчисления.

Например, в древен Вавилон са създадени таблици, които на съвременен език представляват табулирани стойности на синуси от малки ъгли (с точност до шестнадесетия десетичен знак) и използвани за описание на движението на видимите небесни тела. В древен Египет са създадени таблици за представяне на обикновени дроби като сума от дроби с числител 1 с цел улесняване на пресмятанията при разделяне цял обект на части. В този етап са и постиженията в древна Гърция, в древен Китай, както и в Европа до Ренесанса.

През втория етап, от създаването на концепцията за граница до изобретяването на изчислителните машини, се моделира предимно с чисто теоретични модели, доказват се теореми за съществуване на решение или за неговата единственост, понякога без да се извежда метод за намирането на това решение.

В третия, съвременен етап на развитие, се създават сложни теоретико-приложни модели във всички области на науката, но се реализират и чисто приложни модели. Основните понятия в математическото моделиране са:

- обект;
- теоретичен математически модел;
- компютърен математически модел.

Разликата между математически модел и модел въобще е, че в математическия модел се използва като основно средство математическия апарат - математическа логика, математически теореми и доказателства. Математическото моделиране е удобно за експерименталните науки. Например, физичните закони са математически модели на физичните явления, за които могат да бъдат проведени експерименти за потвърждаване или отхвърляне точността на модела.

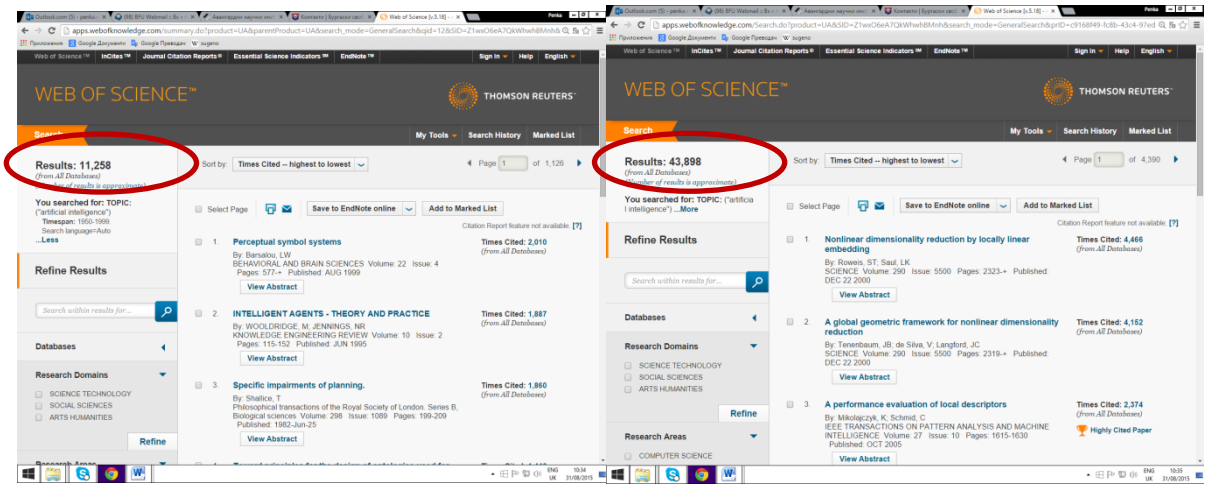
Всеки модел, и в частност математическият модел, е единствено и само приближение на реално съществуващия обект. И следователно, всеки модел е неточен в някаква степен.

Различните науки изучават обекти с различни особености и с различна сложност и по тази причина методите им за описание, анализиране и предвиждане се различават по своята точност и сигурност. Например, моделите за пресмятане на точната дата и час за настъпване на дадено астрономическо явление (изгревите и залезите на Слънцето,

фазите на Луната, затъмненията, настъпването на сезоните и пр.) са изключително точни. Моделите за предвиждане на климатичните промени (валежи, температура на въздуха, сила на вятъра и пр.) са донякъде точни и то в близък интервал от време, докато моделите, описващи социални явления (политически промени, икономическо развитие и др.) са почти безполезни [1].

През последните 60 години се наблюдава изключително бързо развитие в една област на науката, пряко свързана с математическото моделиране, и това е изкуствения интелект. Добър източник за информация за развитието на научния интерес в дадена научна област е базата Web of Science [2]. Според това приложение до 1999 година има приблизително 11 260 публикации в областта на изкуствения интелект, докато в последните 15 години броят на публикациите надвишава 43 800 (фиг. 1). Средният годишен брой на публикации се е увеличил над 10 пъти – от 225 броя годишно до 2920 броя годишно.

За дефиниране и обяснение на изкуствения интелект съществуват многобройни опити и подходи – от философски до строго математически. В научните среди преобладаващото разбиране за интелект (естествен или изкуствен) е, че интелект е способността да се възприемат измененията в заобикалящата среда и да се реагира на тези изменения. Терминът изкуствен интелект е използван за първи път от Джон МакКарти през 1955 като „науката и техниката за създаване на интелигентни машини“. Основните подобласти на изкуствения интелект са насочени към процеси и явления, свързани с разсъжденията, знанията, планирането, обучението, възприятията, общуването, способността за движение.



Фиг. 1. Брой публикации в областта Artificial Intelligence в периодите до 2000 и след 2000 година според Web of Science (данните са от 31.08.2015)

Изкуственият интелект има за предметна област разработването на различни техники за представяне и откриване на знания; интелигентно търсене; оперирането с неточността и/или несигурността на данни и знания; машинно обучение и други.

В настоящето изкуствения интелект е симбиоза на традиционния изкуствения интелект и различни методологии за числено пресмятане, като обекти на изкуствения интелект стават основно онези задачи, за които липсват директни математически или логически алгоритми и могат да бъдат решавани само интуитивно.

Към традиционните базисни за изкуствения интелект науки (философия, математика, алгоритми, логика, психология, информатика, лингвистика) постепенно се добавят нови: компютърни науки, неврология, когнитивна наука, онтологии, изследване на операциите, икономика, вероятности, оптимиране.

Предмет на изследване на съвременния изкуствения интелект са:

- обучаващи се системи с адаптивен обучителен процес, с базирано на аналогии обучение, с индуктивно обучение;
- представяне на знания и изводи, основани на знания с цел достигане на предварително дефинирано състояние от едно или няколко зададени начални състояния, с използване на правила, йерархии, предикатна логика, семантични мрежи и други;
- придобиване на знания, създаване на динамични структури за съществуващите знания, извличане на знания от обкръжаващата среда, прецизиране на знанието;
- планиране, чрез определяне на методология за успешно автоматично достигане на зададената цел при известни начални условия;
- интелигентно търсене;
- логическо програмиране.

Областите на приложение на средствата на изкуствения интелект са изключително разнообразни: експертни системи, игри, доказателство на теореми, обработка на естествени езици, разпознаване на образи, роботика, навигация, системи за управление, планиране, дейта майнинг, логистика и много други.

За алтернативно название на научната област *Изкуствен интелект* се използват още *Изчислителна интелигентност*, *Синтетична интелигентност*, *Интелигентни системи*, *Изчислителна рационалност*. Използването на тези алтернативни названия е опит да се разграничи традиционният изкуствен интелект, опериращ предимно със символи (или още *GOFAI=Good-Old-Fashioned-Artificial-Intelligence*), от съвременното му съдържание.

В настоящия момент е прието да се разграничават две основни направление в изкуствения интелект:

(1) конвенционален изкуствен интелект, основан на формализъм и статистически анализ;
(2) изчислителен изкуствен интелект, основан на изчислителни техники.

Второто направление е известно като *софт компютинг*.

Ключова теза на софт компютинга е фактът, че точността и сигурността при реализирането на моделите на изкуствения интелект имат висока цена и при извършване на изчисления, разсъждения и вземане на решения е допустима толерантност към неточност и несигурност. Следвайки тази теза, в софт компютинга се моделират процеси, протичащи в човешкия мозък, както и процеси, познати от живата природа. [3]

Идеята за софт компютинг е предложена от Заде през 1991, като стъпка към създаване на нов вид изкуствен интелект.

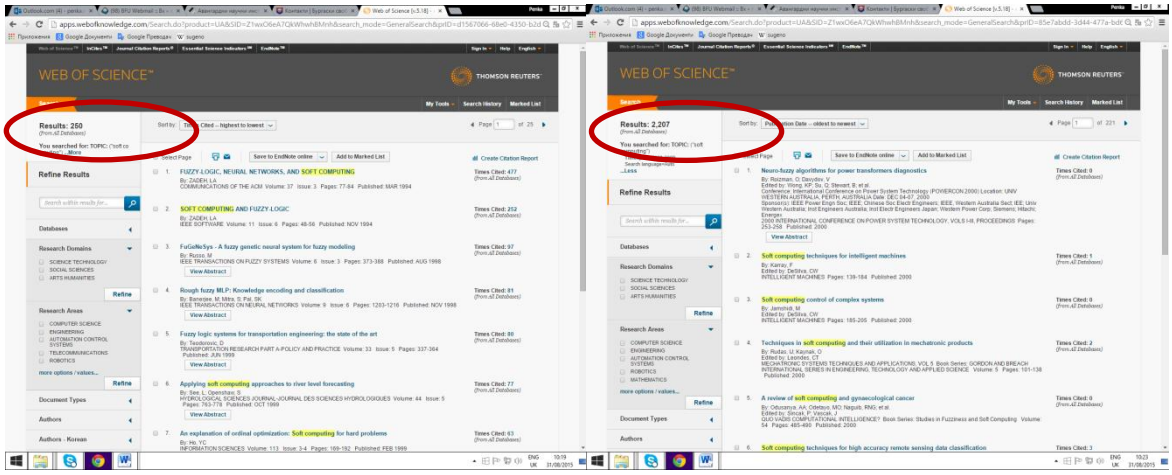
На 13.03.1991 на конференцията на *Industrial Liason Program (ILP)* в Бъркли е създадена организацията *Berkeley Initiative on Soft Computing (BISC)*, която очертава основните принципи и цели на софт компютинга.

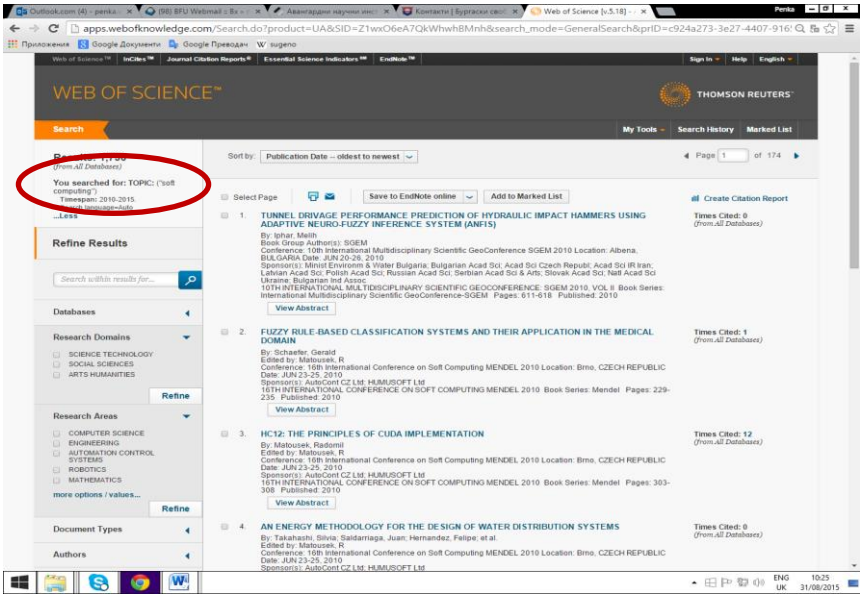
> „Като се използва допустимата степен на неточност, несигурност, частична истинност и приближение, да се постигне ясно, стабилно и на ниска цена решение на фундаментални проблеми, свързани със съвременното технологично развитие и така да се компенсира липсата на изискваната интелигентност в сегашните информационни технологии, така че да позволи човешка функционалност, т.е. човешкият мозък, да бъде използван за модел.“

Софт компютингът се състои от следните изчислителни парадигми:

- размита логика с основен апарат теорията на размитите множества, като елементите на имат степени на принадлежност в интервала [0 ; 1];
- изкуствени невронни мрежи, като невроните, получават и обработват сигнали и в зависимост от активиращата функция задействат (или не задействат) промяна в състоянието;
- приблизителни изводи, които се прилагат при решаване на проблеми, свързани с бази от данни и бази от знания при условия на неточност;
- машинно обучение;
- недиференциални методи за оптимизация, включващи еволюционни пресмятания (в частност генетични алгоритми); симулирано закаляване и др.

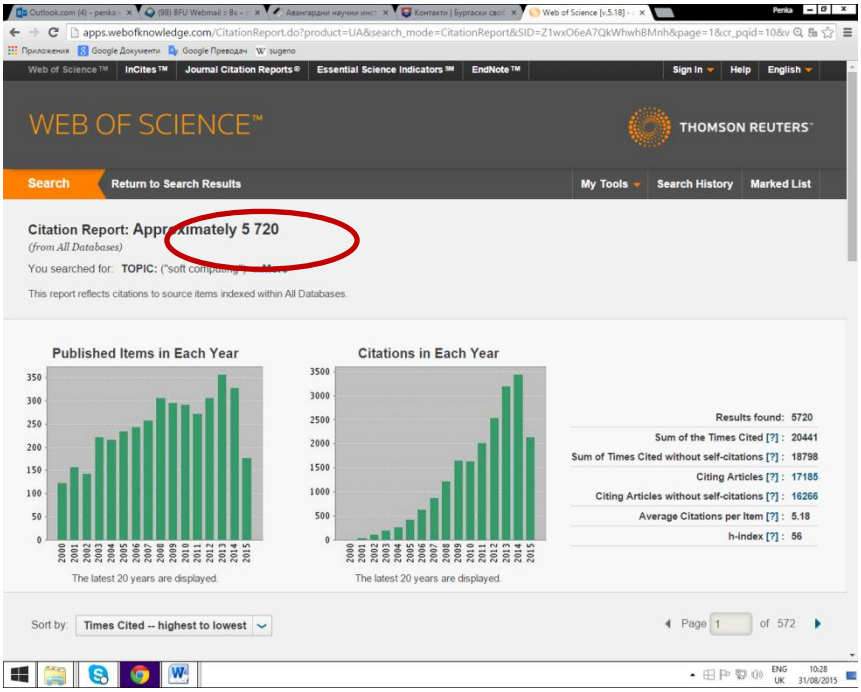
В *Web of Science* може да се проследи публикационната активност и цитиранията в областта на софт компютинга (фиг. 2). Първите публикации са от 1985 година (Т. Митчел; Бр. Смит), като до 1999 общият брой е приблизително 250, в следващия десетгодишен период (до 2009) са около 2200, а в последните пет години – над 1700. [2]





Фиг. 2. Брой публикации в областта Soft Computing в периодите до 1999, 2000-2009 и след 2010 година според Web of Science (данните са от 31.08.2015)

Интерес представлява и графиката за цитируемостта (фиг. 3) за последните 15 години. През 2015 година има спад както в броя на публикациите, така и в броя на цитиранията и това не е изненадващо поради факта, че проектът *BISC* е приключен на 07.11.2014. [2]



Фиг. 3. Брой публикации и брой цитирания в областта Soft Computing в периода 2000-2015 според Web of Science (данните са от 31.08.2015)

По-пълен обзор на литературата, свързана с основните направления в софт компютинга и техните приложения, както и с приносите на българските учени в областта може да бъде намерен в [4].

2. Размити системи

За използването на размита логика в процеса на моделиране на системи има многобройни примери [5], [6, [7], [8], [9] и други. Те имат няколко съществени предимства при подпомагане на процеса за вземане на решения:

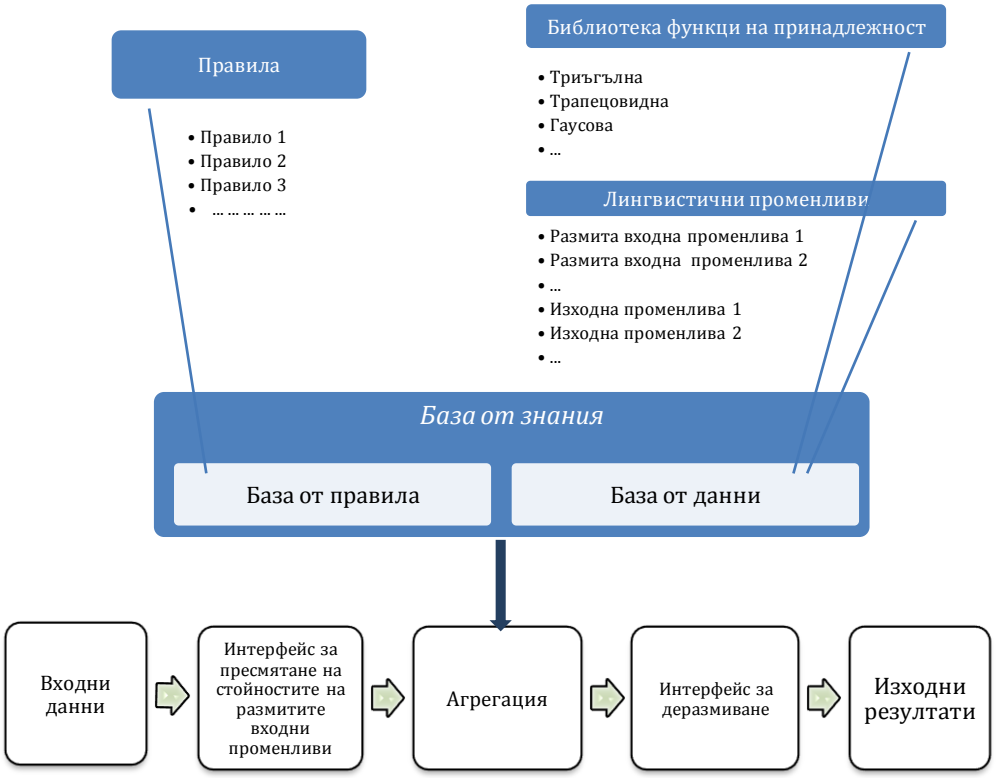
- размитата логика е лесна за имплементация, тъй като използваните математически концепции имат относително ниска изчислителна цена;
- една размита система може да бъде доразвивана без да се налага ново проектиране, защото добавянето на нови правила и функции не променят основно структурата ѝ;
- размитата логика е толерантна към неточности в информацията, т.е. възможно е използването на непрецизна, непълна и неясна информация;
- с помощта на размита логика могат да се моделират нелинейни функции с произволна сложност;
- размита система може да бъде построена въз основа на опита на експерти;
- размитата логика може да бъде съчетана със стандартни техники за управление и въпреки, че не е задължително размитите системи да заменят стандартните техники за управление, но в много случаи те опростяват тяхното изпълнение;
- размитата логика е базирана на естествения език.

Видове размити системи

Размита система за изводи (*fuzzy inference system=FIS*) е изчислителна структура, създадена с теорията на размитите множества, размитата логика и правила от вида *ако-то*. Тъй като размитите системи за изводи имат разнообразна структура и предназначение, за тях се използват различни наименования като: размита експертна система, размит модел, размита асоциативна памет, контролер с размита логика, размита система и др.

Общата структура на една размита система съдържа три концептуални компонента:

- база от правила, включваща всички размити правила за вземане на решения;
- база от данни, където са дефинирани всички функции на принадлежност; всички терми, използвани в размитите правила и лингвистичните променливи на размитата система;
- машина за изводи, която изпълнява процедурата за вземане на решения чрез използване на правилата и дадените факти, за да се получи коректен изход или заключение.



Фиг. 4. Размита система, базирана на правила

Базата от правила и базата от данни образуват база от знания.
Функционирането на размитата система, базирана на правила става, като входните данни се размиват за получаване на степен на принадлежност към всеки от термите на входните размити променливи; машината за изводи прилага правилата за агрегация,

като се използва базата от знания и се получават степените за принадлежност към термите на изходните променливи; след деразмиване се получава изходния резултат (фиг. 4.)

Размити системи за изводи от тип Мамдани

Размитата система, предложена през 1975 от Ейбрахам Мамдани за управление на парна машина и парен котел чрез лингвистични правила, съобразени с човешкия опит на операторите [10], е сред първите системи за управление, използващи теорията на размитите множества. Система от вида *Мамдани* се характеризира с това, че термите на изходните променливи са размити променливи, които се деразмиват по различни методи.

При модел от тип *Мамдани* на всеки оператор в машината за изводи съответства норма или конорма, като:

- за оператора *AND* се използва T -норма;
- за оператора *OR* се използва T -конорма;
- за оператора за импликация се използва T -норма при дадени тегла на правилата;
- за оператора за агрегация се използва T -конорма;
- за оператора за деразмиване се използват различни подходи за трансформиране на изходната размита променлива до единствена точна стойност.

За T -нормата и T -конормата основно се използват два варианта:

- *min* за T -нормата и *max* за T -конормата с максиминна композиция;
- алгебрично произведение за T -нормата и *max* за T -конормата с максимално произведение за композицията.

По-подробно описание на споменатите оператори и действия може да бъде намерено в [4].

С избор на други оператори и композиции се получават различни варианти на модела, но общото на размитите модели от тип *Мамдани* и отличаващото ги от другите модели е, че изходните резултати са размити множества и следователно задължително е необходимо деразмиване. Има различни подходи за деразмиване на агрегираните изходни размити променливи:

- метод на центъра на тежестта

$$x_{CoG} = \frac{\int_X \mu_A(x).x.dx}{\int_X \mu_A(x).dx} \quad ;$$

- метод на медианата

$$x_{CoA} : \int_{\alpha}^{x_{CoA}} \mu_A(x).dx = \int_{x_{CoA}}^{\beta} \mu_A(x).dx \quad ;$$

където $\alpha = min\{x \mid x \in X\}$ и $\beta = max\{x \mid x \in X\}$;

- метод на средното на максимумите

$$x_{AoM} = \frac{\int_{X'} x.dx}{\int_{X'} dx} \quad ;$$

където $X' = \{x \mid \mu_A(x) = \mu^* : \mu^* = \max_X(\mu_A(x))\}$;

- метод на минимума на максимумите

$$x_{Inf\,oM} = min\left\{x \;\middle|\; \mu_A(x) = \mu^* : \mu^* = \max_X\left(\mu_A(x)\right)\right\};$$

- метод на максимума на максимумите

$$x_{Sup\,oM} = max\left\{x \;\middle|\; \mu_A(x) = \mu^* : \mu^* = \max_X\left(\mu_A(x)\right)\right\}.$$

Размити системи за изводи от тип Сугено

Този модел е предложен от Такаги, Сугено и Канг като опит за систематичен подход за генериране на размити правила от предварително зададени множества от входни и изходни данни [11], [12].
Размитите правила в модел от тип *Сугено* имат вида:

ако x е A и y е B, то z = f(x;y),

където *A* и *B* са размити лингвистични променливи, а *f(x;y)* е реална функция.

Когато *z = f(x;y)* е константа, моделът се нарича модел на Сугено от нулев ред и изходът е сингелтон; ако е полином от първа степен – модел на Сугено от първи ред и изходът е равнина в тримерното пространство.
Моделите от тип *Сугено* са подходящи за управление на множество от линейни контролери, както и за управление на динамични нелинейни системи. Размитите системи от тип *Сугено* са изключително подходящи в случаите, в които върху пространството от входни данни се прилагат малки непрекъснати изменения, и за моделиране на нелинейни системи чрез множество линейни компоненти.

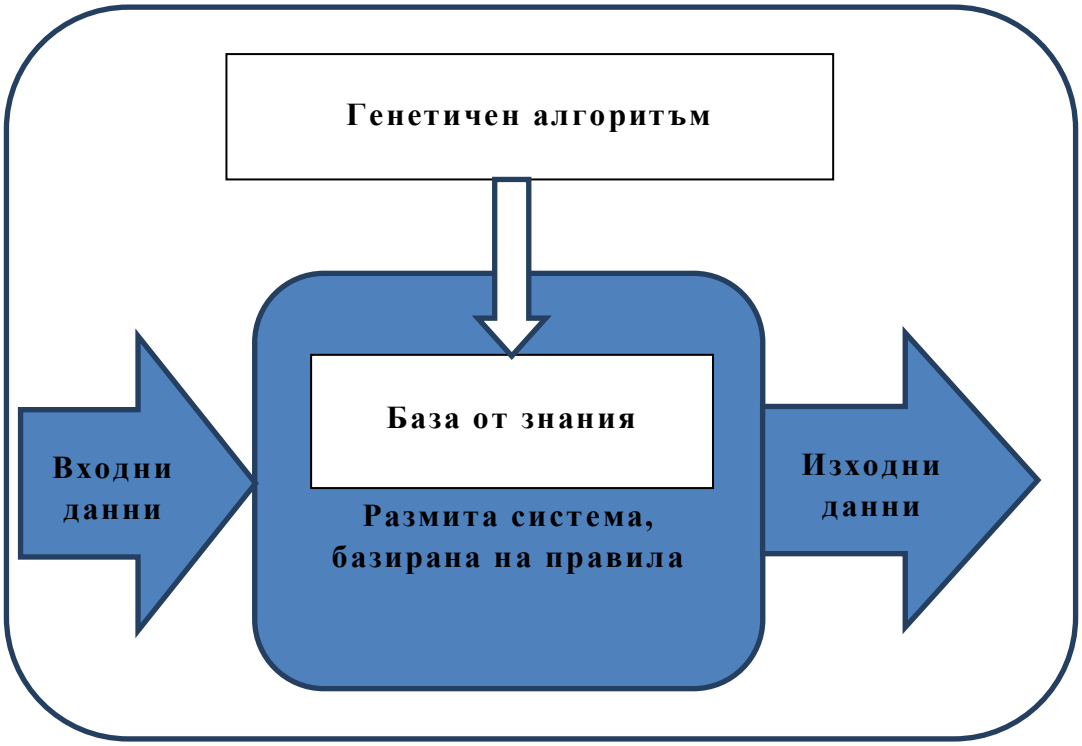
Размити системи за изводи от тип Цукамото

В размит модел от тип Цукамото изходната променлива се състои от размити множества с монотонни функции на принадлежност [12]. След изпълнението на всяко размито правило се получава точна стойност, индуцирана от силата на изпълнение на това правило. Окончателно на изхода се получава точна стойност, която е претеглено средно на резултата от изпълнението на всяко правило. При този модел дефъзификация също е ненужна.

3. Генетични размити системи

Генетичните размити системи са хибридни системи в софт компютинга, в които се комбинират предимствата на еволюционните пресмятания и размитите системи [13], [14], [15]. Използват се различни техники от еволюционните пресмятания: генетични алгоритми, генетично програмиране и еволюционни стратегии. На фигура 5 е представена обща схема на генетична размита система.
При проектирането на такива системи основна цел е да се автоматизира процесът на генериране на базата знания, което в същността си е оптимизация или търсене на оптимално решение. В случая това се изразява в намиране на подходяща за ситуацията база знания, и след след параметризация на базата знания да се пресметнат онези стойности на параметрите, които са оптимални според оптимизационните критерии.
Първоначално трябва да се реши коя част от базата знания ще се оптимизира с генетичен алгоритъм. Важни аспекти при това решение играят подбора на пространството за търсене на оптимално решение и ефективността на това търсене.

Колкото пространството на търсене е по-голямо и се извършва по-прецизно, фино търсене, толкова по-малка е ефективността.

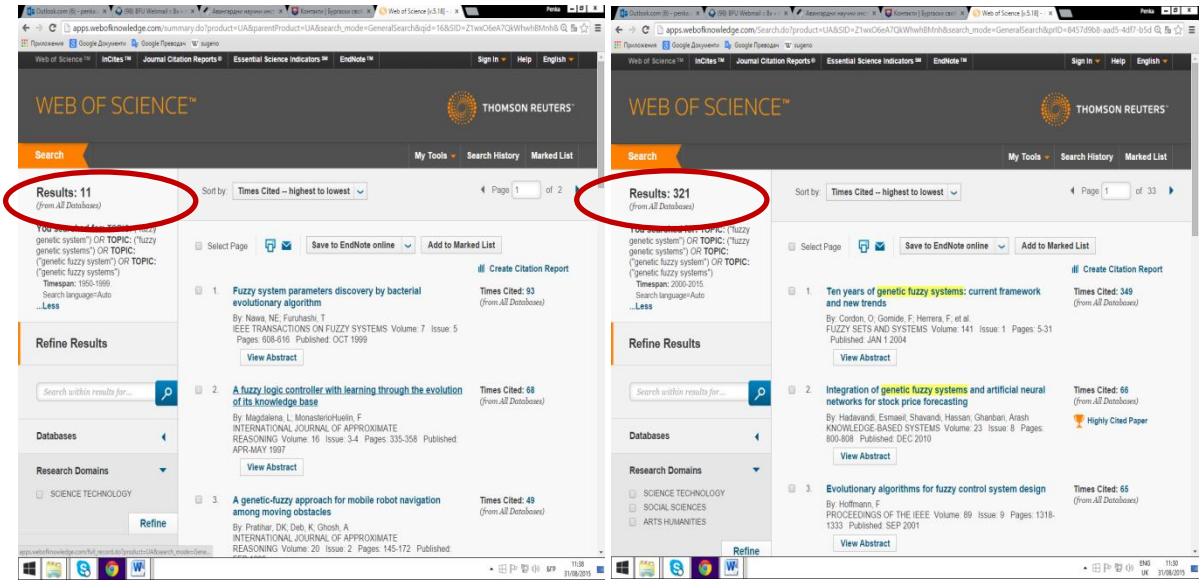


Фиг. 5. Генетична размита система

Според *Web of Science* публикационната активност в областта Генетични размити системи е нарастнала повече от 5 пъти за последните 15 години (фиг. 6). Първите публикации (според тази база)[1] са от 1997 година [8], като до 1999 общият брой е 11 (приблизително 4 публикации годишно), в следващите 15 петнадесет години – 321 (21,4 публикации годишно).

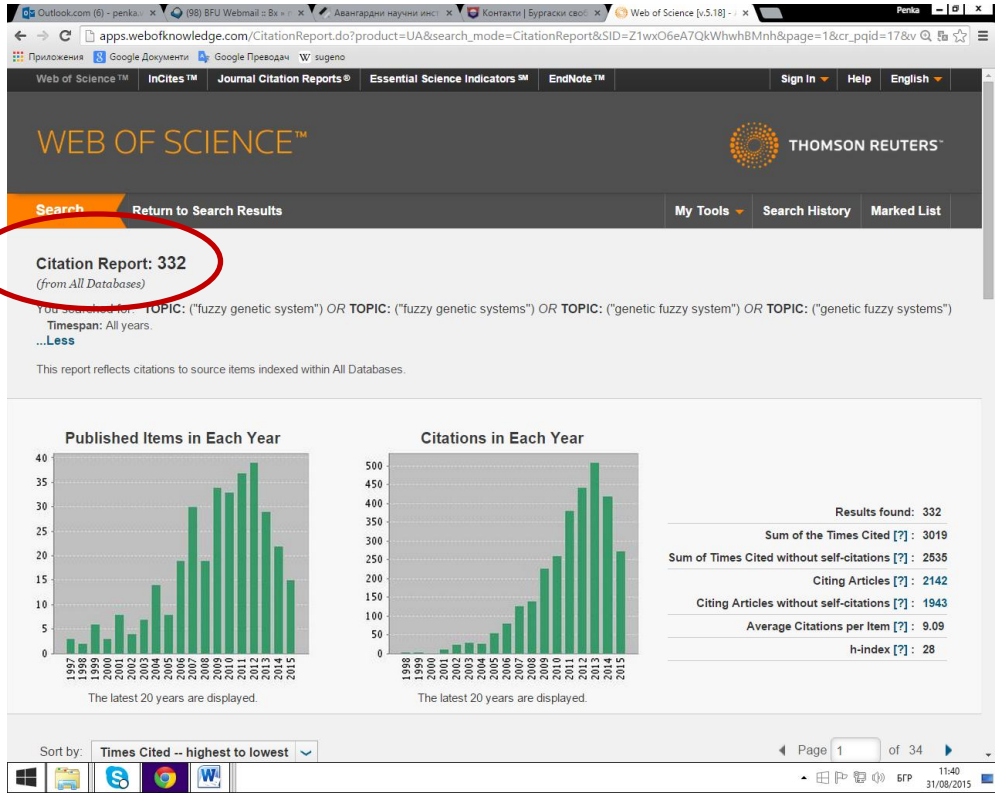
Интерес представлява и графиката за цитируемостта в областта Генетични размити системи за целия период на отчитане на данните (фиг. 7). Докато до 2013 цитируемостта рязко се увеличава с времето, през 2014 се забелязва спад. Ще бъде интересно да се наблюдава този процес през 2015 и следващите няколко години.

[1] Могат да бъдат открити и по-стари публикации, например [8], [13], [14], [15], [16], [17], но те на са в базата *Web of Science.*



Фиг. 6. Брой публикации в областта Генетични размити системи в периодите до 1999 и 2000-2015 година според Web of Science (данните са от 31.08.2015)

Изводът, който може да бъде направен е, че това е област от софт компютинга, в която няма изключително голям брой публикации. Причините за това могат да са различни, но за тях може само да се спекулира, а това е извън обхвата на тази работа.



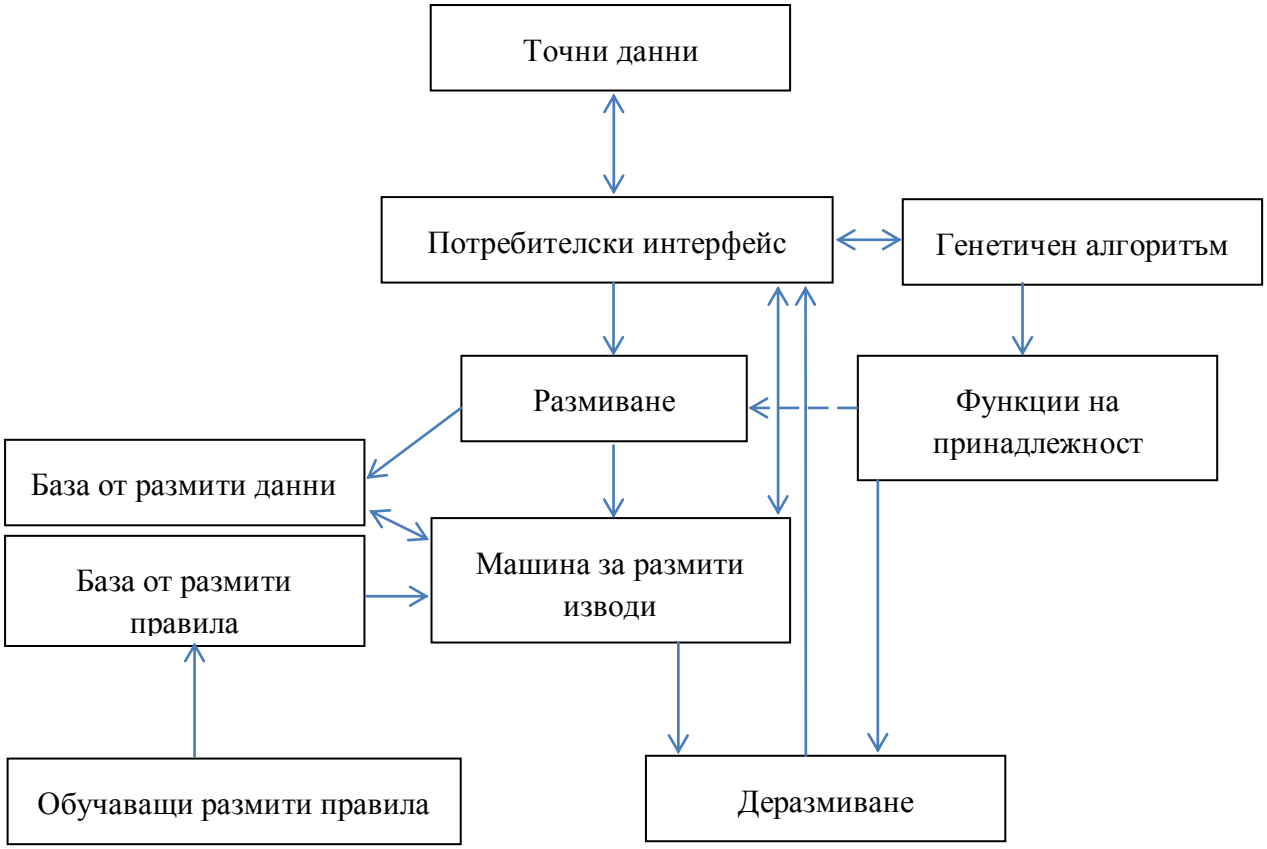
Фиг. 7. Брой публикации и брой цитирания в Генетични размити системи до 2015 според Web of Science (данните са от 31.08.2015)

4. Оптимизиране на параметри с генетични размити системи

В тази работа е представено една проектирана, реализирана и тествана генетична размита система, която оптимизира параметрите на предварително създадена размита система от тип Мамдани. Целта при създаване на това приложение е, да се използва генетичният алгоритъм за оптимизиране на параметрите на функциите на принадлежност на генетичната система FSSAM [5].
На фигура 8 е представен концептуален модел на генетична размита система, в която генетичен алгоритъм оптимизира базата знания на размита система.
Целевата функция на оптимизационната задача е:

$$(Q_1 - Q_2)^2 + (Q_1 - Q_3)^2 + (Q_2 - Q_3)^2 \ ,$$

където $Q_k, k = 1, 2, 3$ са точни стойности на изходната променлива на размитата система, получени в три различни момента. Q_k са всъщност деразмитите стойности на размитата изходна променлива $Q - measure$ на размитата система FSSAM, която отчита три величини за финансовия актив: възвръщаемост, риск и отношението им. За дефиниране на целевата функция, след въвеждане на входните стойности, се определят времевите интервалите, в които да се пусне FSSAM.



Фиг. 8. Модел на генетична размита система

Целта е да се минимизира разликата между получените от размитата система Q_k, с цел да се постигне стабилност.

Изпълнението на генетичния алгоритъм следва шест етапа.

1. Декларират се стойности на параметрите, които определят действията на операторите на генетичния алгоритъм: *брой поколения, размер на популацията, брой променливи, брой битове за променлива, вероятност за мутация* P_m, *вероятност за рекомбинация* P_c, *елитност, горни и долни граници.* Горните и долни граници задават интервалите, в които могат да се изменят параметрите, т.е. областта на на търсене на решения.

2. Създава се първоначална популация от кандидат-решения на оптимизационната задача. Тя е случайно генерирана с предварително зададния *размер на популацията*. Необходимо е хромозомите да се кодират от двоични в реални стойности. Преобразуването на векторите се осъществява според зададените *размер на популацията, брой променливи, брой битове и долна и горна граници.* След като вече е образувана популацията, се изчислява пригодността й към дадената целева функция.

3. От получените стойности на целевата функция се избират най-пригодните индивиди и върху тях се изпълнява операторът селекция. В случая селекцията се осъществява по рулетковия метод. Вероятността за селекция на всеки индивид се определя като отношение на стойността на целевата му функция към сумата от стойностите на целевата функция на всички индивиди. За да работи за задачи за минимизация се преизчисляват стойностите на целевата функция на всеки индивид, така че стойността на най-пригодния да бъде максимална. При този метод окръжността се разделя на N на брой сектори, пропорционални на вероятността за селекция на индивидите. Всеки път когато се завърти рулетката, се избира един индивид. При програмната реализация на рулетковия метод се използва линейно представяне на колелото. Използва се генератор на случайни числа в интервала [0,1]. Това число се умножава по сумата от всички стойности на целевата функция, като се натрупва сума до достигане на случайното число. Взима се номерът на индивида с последната прибавена стойност на целева функция. По този начин индивиди с по-големи стойности на целевата функция получават пропорционално по-големи вероятности да бъдат избрани.

4. След като са селектирани родители, започва репродукцията за получаване на ново поколение. Изпълнява се рекомбинация. Избрана е рекомбинация в една точка, при която родителите разменят генната си информация в една точка с вероятност за рекомбинация P_c, зададена в началото на алгоритъма.

5. При операторът мутация за вход се използва рекомбинираното поколение, което мутира с вероятност за мутация P_m. За генетичния алгоритъм е избрана мутация с промяна на бита. Създава се и матрица, която показва в кой ген е извършена мутацията.

6. За новосъздаденото поколение отново се пресмятат стойностите на целевата функция и за най-пригодните се извършва проверка за елитност. След генерирането на всяко поколение, по-пригодните индивиди и техния генетичен материал може да бъде загубен, заради селекция (при него не се избират пригодни индивиди), а рекомбинация и мутация могат да доведат до влошаване на пригодните индивиди. За да се запазят техните добри белези, генетичният им материал трябва да се запази в алгоритъма. Този процес се нарича елитност.

Последователно се повтарят 3, 4, 5 и 6.

5. Реализация на модела

Предложеният модел е реализиран в програмната среда MatLab.

Първоначално се създава основен файл *start.m*, който служи за стартиране и извличане на резултата от работата на генетичния алгоритъм *GeneticAlgorithm5.m*.

Start.m

```
clc
clear
MMM=GeneticAlgorithm5
```

GeneticAlgorithm5.m

Във файла *GeneticAlgorithm5.m* са декларирани стойностите на параметрите на генетичния алгоритъм. Броят на поколенията на алгоритъма се задават с *nGen*, размерът на всяка популация е *nPop*, броят променливи е *nVar*, декларират се брой битове за всяка променлива, вероятността за мутация е *Pm*, а за рекомбинация *Pc*. Допълнителен параметър е *Elitism (0 – false; 1- true*), в който се задава дали да се изпълни елитизъм. Дефинират се горната и долна граница на задачата и в случая това са:

UBound = [0.125 0.375 0.625 0.875 1.125]

LBound = [-0.125 0.125 0.375 0.625 0.875]

Създава се първоначална нулева матрица с размери *nPop x 4*. След това има обръщение към *InitPop*, *CalculateReals*, *RSelection*, *Crossover*, *Mutation* и *CheckElite*. За всеки оператор е създаден отделен *.m* файл.

```
function [Data, String_Population, Fitness] = GeneticAlgorithm()
nGen = 10;
nPop = 6;
nVar = 5;
nBits = 12;
Pm = 0.1;
Pc = 0.75;
Elitism = 1;
UBound = [0.125 0.375 0.625 0.875 1.125];
LBound = [-0.125 0.125 0.375 0.625 0.875];
Data = zeros (nPop,4);
PerformanceType = 1;
String_Population = InitPop(nPop);
[proba,String_Population]=CalculateReals(String_Population, nPop, nVar,
nBits, LBound, UBound)
```

```
 [Fitness] = CalculateFitness(String_Population, nPop, nVar,
PerformanceType);

for Run = 1:nGen

   Run

   [Selected_Population,Selected_Fitness] = RSelection(String_Population
,Fitness, nPop);

   [Crossed_Population] = CrossOver(Selected_Population,Pc);

   [Mutated_Population, M,Mcount] = Mutation(Crossed_Population,Pm)

   [MFitness] = CalculateFitness(Mutated_Population, nPop, nVar,
PerformanceType);

      if Elitism

      [String_Population, Fitness, MPmax, OldMax] = CheckElite
(Mutated_Population, MFitness, String_Population, Fitness);

   else

     String_Population=Mutated_Population;

     Fitness = MFitness;

   end

   disp ([Run max(Fitness) mean(Fitness) min(Fitness)]);

   Data(Run, : ) = [Run max(Fitness) mean(Fitness) min(Fitness)] ;

end

end
```

InitPop.m

Във файла *InitPop.m* се създава първоначална популация. Функцията *iPopulation* използва блока *switch* за избор на една от три ситуации. В *case1* се създават хромозоми с размер *Chromesize*.

```
function iPopulation = InitPop(nPop, InitPop)

switch nargin

   case 1

      ChromeSize = 363;

      iPopulation = rand (nPop, ChromeSize)> 0.5;

   case 2

      iPopulation = InitPop;
```

```
    otherwise
        ChromeSize = 363;
        iPopulation = rand (nPop, ChromeSize) > 0.5;
    end
end
```

CalculateReals.m

След като вече е създадена популация от произволни хромозоми, те се трансформират от двоични в реални стойности, за да се изпълни оценката след това. Изпълнява се *CalculateReals*. Първоначално се създава матрица, която извлича първите *(nBits - 1): -1:0* елементи. За да се изпълни преобразуването на всички елементи, се използва цикъл, който се изпълнява 1:nPop пъти. В него се изчисляват целочислените стойности от популацията и след това се конвертират чрез линейна трансформация:

```
Integers(counter, count)= Pop(counter, index:index + nBits - 1)*Convert'
        m = (UB(count) - LB(count))/MaxCon;
        c = UB(count) - m*MaxCon;
        Reals(counter,count) = m*Integers(counter,count) + c;
        index = index + nBits;
```

След като вече са преобразувани хромозомите, оценката на пригодност може да се изпълни. Създава се цикъл, който се изпълнява 1:nPop пъти, и в него се извиква файлът FfSY.m

FfSY.m

В този файл се дефинира фитнес функцията (целева функция за оптимизажионната задача):

```
function [y,PAR1, PAR2, PAR3, PAR4, PAR5]=FfSY(x,nVars,A)
    PAR1=x(1)
    PAR2=x(2);
    PAR3=x(3);
    PAR4=x(4);
    PAR5=x(5);
    Q = TheSystem(PAR1, PAR2, PAR3, PAR4, PAR5);
    y=(Q(1)-Q(2))^2+(Q(1)-Q(3))^2+(Q(2)-Q(3))^2;
end
```

Стойностите на PAR1, PAR2, PAR3, PAR4, PAR5 са изходни на FSSAM, която е създадена в друг файл - *TheSystem.m*.

TheSystem.m

При стартиране на системата входните данни за се въвеждат с използване на функцията xlsread, който извлича от файл реални данни за цени, свалени от БфБ – София. Създава се матрица с данните. Изчисляват се променливите: възвръщаемост, риск, q-ratio. Пресмята се тяхната степен на принадлежност към всеки от термите на съответните входни лингвистични променливи. При агрегацията на правилата на всяка наредена тройка от стойности на входните променливи се съпоставя степен на принадлежност към съответния терм на изходната променлива, например:

```
ru1teta5=min([m15 m21 m33])*1;

ru2teta5=min([m15 m22 m33])*1;

ru5teta4=min([m15 m21 m33])*.8;

ru6teta4=min([m15 m23 m33])*.8
```

Стойностите на параметрите на функциите на принадлежност за съответните терми са въведени като *PAR1*, *PAR2*, *PAR3*, *PAR4* и *PAR5*. Например:

```
G1=memfGAUS(PAR1,.1,x);

   teta1;

if G1<teta1

     mu1=G1;

else

   mu1=teta1;

end
```

След това се извлича най-голямата величина за mu, и се използва при процеса на деразмиване.

```
mu=max([mu1 mu2 mu3 mu4 mu5]);

     su4=su4+x*mu;

     su3=su3+mu;

end

Q(k)=su4/su3;
```

RSelection.m

Генетичният алгоритъм продължава със селекция. Тя е създадена във файла RSelection.m. Рулетковият метод може да се програмира като процес на натрупване с cumsum. След избиране на *rand()* вероятност, се намират индивидите, които вече са преминали през рулетковия метод. Натрупват се възможните и се сравняват с *rand*. Процедурата се изпълнява *nPop* пъти. Някои индивиди могат да бъдат избрани повече от веднъж, а други изобщо няма да бъдат избрани, ако „стрелката“ не спре в техния сектор.

```
for counter = 1:nPop
   Prob = rand();
   index = find(cumsum (Fitness_Prob)>= Prob, 1, 'first');
   Selected_Population(counter,:) = Population(index, :);
   Selected_Fitness(counter,:) = Fitness(index)
```

CrossOver.m

За всеки два индивида, избрани за рекомбинация, се определя произволна стойност на *CBit* между *1* и *ChromLen-1*, където *ChromLen* е дължината на хромозомат. Родителите променят гените си след тази точка, като получените нови хромозоми са следващото поколение. Ако двама родители не се кръстосат според вероятността за рекомбинация, те се включват към поколението.

```
for count = 1:2:nPop
    if rand() <= Pc
            CBit = randi([1 ChromLen-1]);
    Crossed_Population(count,:) = [Selected_Population(count,1:CBit)
Selected_Population(count + 1, CBit + 1:ChromLen)];
    Crossed_Population(count+1,:) = [Selected_Population(count+1,1:CBit)
Selected_Population(count,CBit +1:ChromLen)];
     else
   Crossed_Population(count,:) = Selected_Population(count,:);
  Crossed_Population(count+1,:) = Selected_Population(count+1,:);
     end
```

Mutation.m

След като е получено рекомбинирано поколение, се избира произволна точка за мутиране на един индивид. Избрана е мутация с промяна на бита на хромозомите *ChromLen*. Популацията се задава с *Population*, а *Mutated_Population* е мутиралата популация, *M* е матрица, отбелязваща къде е извършена мутация, *Mcount* е броя мутации, извършени за поколението.

```
for count = 1:nPop
   for counter = 1:ChromLen
      if rand()<=Pm
       Mutated_Population(count,:) = Population(count,:);
       if Mutated_Population(count, counter) == 1
          Mutated_Population(count, counter) = 0;
                else
          Mutated_Population(count, counter) = 1;
       end
       M(count, counter) = 1;
       Mcount = Mcount + 1;
        else
       Mutated_Population(count,:) = Population(count,:);
       M(count, counter) = 0;
     end
  end
```

След като поколението е мутирало, отново се оценява неговата пригодност в основния генетичен алгоритъм.

```
[MFitness]= CalculateFitness(Mutated_Population, nPop, nVar, PerformanceType);
```

В края на алгоритъма се извършва елитност, която се изпълнява с условен оператор, който в единия случай извиква *CheckElite.m*, а в другия *проверява* отново първоначалното поколение и пригодните за „изгубен материал“.

CheckElite.m

CheckElite.m проверява отново мутиралото поколение, пригодното мутирало поколение, популацията и първоначалните пригодни индивиди. Ако стойностите на целевата функция на най-добрите мутирали индивиди са по- малки от първоначалните, се създава нова популация, в която в първият ред включва популацията, след което я оценя, а във втория ред се включва вече преминалото през оценка мутиралото поколение.

```
if (MPmax < OldMax)
   NPopulation(1,:)= Old_Population(Oldindex,:);
   Fitness(1,:) = OldMax;
```

```
    NPopulation(2,:) = Mutated_Population(Mindex,:);
    Fitness(2,:) = MPmax;
else
```

Размитата система FSSAM е подробно описана в [4], [5] и [6], като тук за пълнота е показан изходният код, реализиран във файла *TheSystem.m*.

```
function Q = TheSystem(PAR1, PAR2, PAR3,PAR4,PAR5)
Price = xlsread('4ID.xls');
[TT,l]=size(Price);
 P=Price(:,1);
r0=0.055;sigma0=0;
I=10 ;% брой дни за пресмятаен на  Q(k)
M=3;  % брой k
for k=1:M
      T=TT-I*(k-1);
for i=2:T
       rt(i)=P(i)/P(i-1);
       rlog(i)=log(rt(i));
end
TR=0;
for i=2:T
   TR=TR+rlog(i);
end
rg=TR*(1/(T-1));
AR=exp(rg*(T-1));
ANR=(AR-1)*360/T;
r=ANR+1;
r=ANR-r0;
s=0;
for i=2:T
  s=s+(rlog(i)-rg)^2;
```

```
end
s=s/(T-2);

 risk=(s)^(1/2);
 sigma=risk-sigma0;
q1=rg/risk;
  q=r/sigma;
  %input variables
  x=r;
k1=20;a1=0;b1=0.5;c1=1.5;d1=2;f1=3;
sig1=0.05;sig11=0.08;
m11= memfSIGM(-k1, a1,x);
 m12=memfGAUS(sig1,b1,x);
 m13=memfGAUS(sig11,c1,x);
 m14=memfGAUS(sig1,d1,x);
 m15= memfSIGM(k1, f1,x);
  xx=sigma;
a2=0; k2=5;b2=0.1;c2=0.3;d2=0.5;f2=0.7;
sig2=0.05; sig21=0.07;
m21= memfSIGM(-k1, a2,xx);
 m22=memfGAUS(sig2,b2,xx) ;
 m23=memfGAUS(sig21,c2,xx);
 m24=memfGAUS(sig2,d2,xx);
 m25= memfSIGM(k1, f2,xx);
  xxx=q;
  a3=0; b3=20; c3=4; d3=40; f3=60; e3=900;
  sig3=0.3;sig31=20;
  m31= memfSIGM(-sig3, b3,xxx);
  m32= memfBELL(sig31, c3,d3,xxx);
```

```
m33= memfSIGM(sig3, f3,xxx);
%rules
ru1teta5=min([m15 m21 m33])*1;
ru2teta5=min([m15 m22 m33])*1;
ru3teta5=min([m14 m21 m33])*1;
ru4teta5=min([m14 m22 m33])*1;
ru5teta4=min([m15 m21 m33])*.8;
ru6teta4=min([m15 m23 m33])*.8;
ru7teta4=min([m14 m22 m33])*.8;
ru8teta4=min([m14 m23 m33])*.8;
ru9teta4=min([m15 m21 m32])*.8;
ru10teta4=min([m15 m22 m32])*.8;
ru11teta3=min([m13 m23 m32])*.7;
ru12teta3=min([m13 m22 m32])*.7;
ru13teta3=min([m12 m23 m32])*.7;
ru14teta3=min([m13 m23 m31])*.7;
ru15teta1=min([m11 m25 m31])*1;
ru16teta1=min([m11 m24 m31])*1;
ru17teta1=min([m12 m25 m31])*1;
ru18teta1=min([m12 m24 m31])*1;
ru19teta2=min([m11 m25 m31])*.8;
ru20teta2=min([m11 m23 m31])*.8;
ru21teta2=min([m12 m24 m31])*.8;
ru22teta2=min([m12 m23 m31])*.8;
ru23teta2=min([m11 m25 m32])*.8;
ru24teta2=min([m11 m24 m32])*.8;

teta1=max([ru15teta1 ru16teta1 ru17teta1 ru18teta1]);
teta2=max([ru19teta2 ru20teta2 ru21teta2 ru23teta2 ru24teta2]);
```

```
teta3=max([ru11teta3 ru12teta3 ru13teta3 ru14teta3]);
 teta4=max([ru5teta4 ru6teta4 ru7teta4 ru8teta4 ru9teta4 ru10teta4]);
teta5=max([ru1teta5 ru2teta5 ru3teta5 ru4teta5]);
a=-1;b=2;N=30; h=(b-a)/N;
su4=0;su3=0;
for i=0:N
   x=a+h*i;
   G1=memfGAUS(PAR1,.1,x);
   teta1;
  if G1<teta1
      mu1=G1;
   else
      mu1=teta1;
  end
    G2=memfGAUS(PAR2,.1,x);
   teta2;
  if G2<teta2
      mu2=G2;
   else
      mu2=teta2;
  end
    G3=memfGAUS(PAR3,.1,x);
   teta3;
  if G3<teta3
      mu3=G3;
   else
      mu3=teta3;
   end
   G4=memfGAUS(PAR4,.1,x);
```

```
    teta4;
   if G4<teta4
      mu4=G4;
    else
      mu4=teta4;
   end
    G5=memfGAUS(PAR5,.1,x);
    teta1;
   if G5<teta5
      mu5=G5;
    else
      mu5=teta5;
    end
   mu=max([mu1 mu2 mu3 mu4 mu5]);
   su4=su4+x*mu;
   su3=su3+mu;
  end

  Q(k)=su4/su3;
  PAR1;
  PAR2;
  PAR3;
  PAR4;
  PAR5;
end
```

6. Резултати

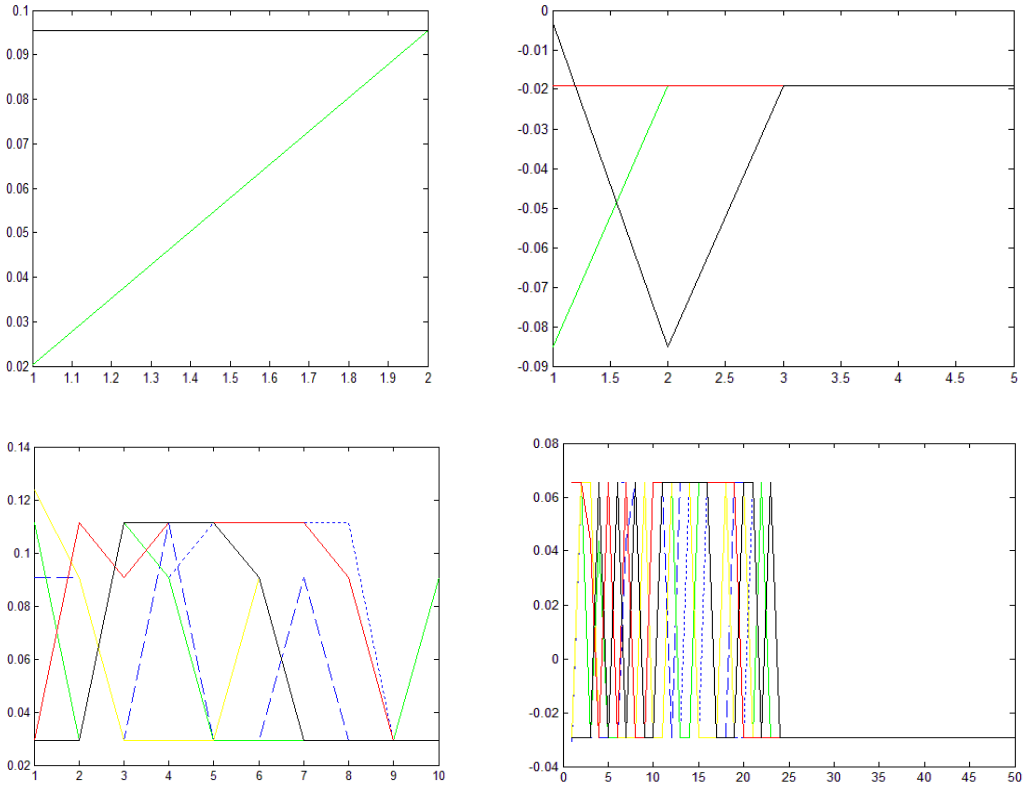
Описаният модел е успешно реализиран в средата MatLab и са проведени експерименти за изследване на генетичния алгоритъм при различни стойности на параметрите му. При тестовете са използвани реални данни от БфБ-София АД. Тук са показани резултати от провеждане на тестове с цените на един актив. Получените резултати за

оптималните стойности на параметрите при използване на данни за други активи са идентични.

PAR1, *PAR2*, *PAR3*, *PAR4*, *PAR5* са стойности на α на Гаусовата функция на принадлежност на съответните терми Y_1, Y_2, Y_3, Y_4 *и* Y_5, а z е стойността на целевата функцията на оптимизационната задача. Преди оптимизацията размитата система FSSAM намира стойност за $Q = 0{,}7746$ при фиксирани стойности за *PAR1 = 0; PAR2 = 0,25; PAR3 = 0,5; PAR4 = 0,75; PAR5 = 1,0*.

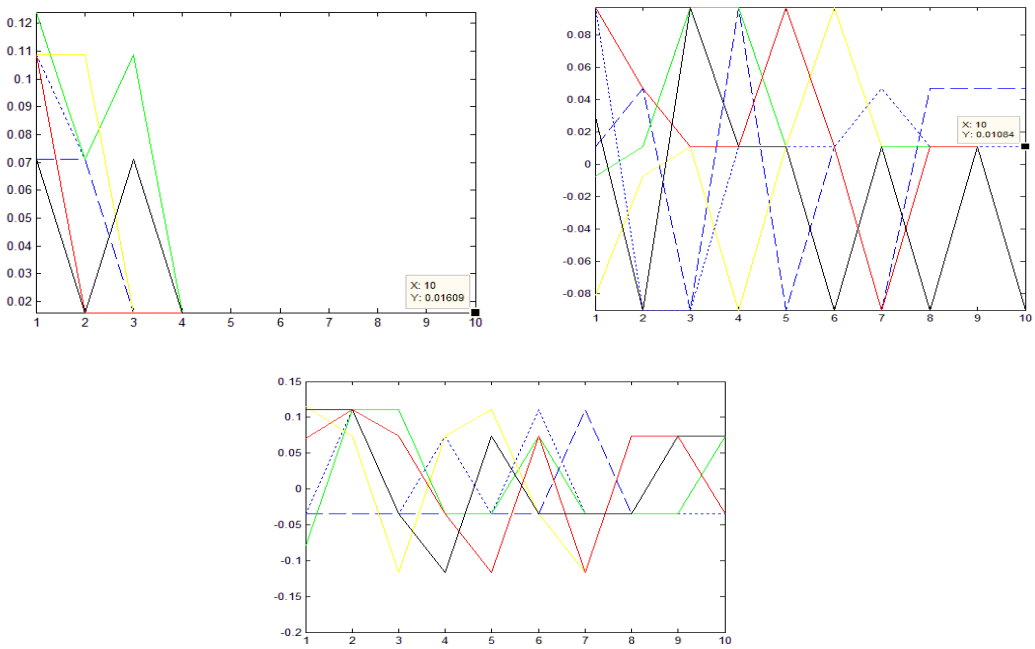
На следващите фигури е показано поведението на *PAR1* при промяна на параметрите на генетичния алгоритъм – брой поколения *nGen*, размер на популацията *nPop*, вероятност за мутация P_m и вероятност за рекомбинация P_c .

На фигура 9 е демонстрирана сходимостта на генетичния алгоритъм (до достигане на оптимални стойности на *PAR1*) съответно при *nGen* = 2, 5, 10 и 50. Интересно е, че най-бързо се дотига сходимост при *nGen* = 5 – още на третото поколение.



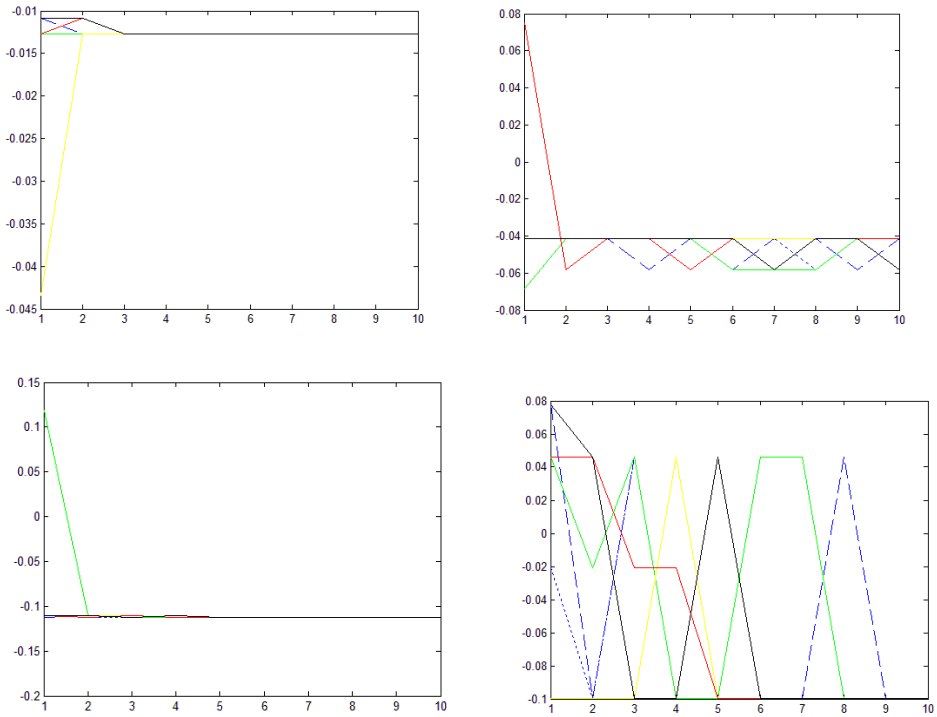
Фиг. 9. Сходимост на PAR1 при стойност на параметрите nPop = 6; Pm = 0,1; Pc = 0,75 и съответно nGen = 2; nGen = 5; nGen = 10; nGen = 50

На фигура 10 е показана сходимостта на генетичния алгоритъм (до достигане на оптимални стойности на *PAR1*) съответно при *nPop* = 10, 20 и 30. При стойност на *nPop* 10 и 20 получените стойностите на z са равни, но сходимостта се достига по-бързо при *nPop* = 10.



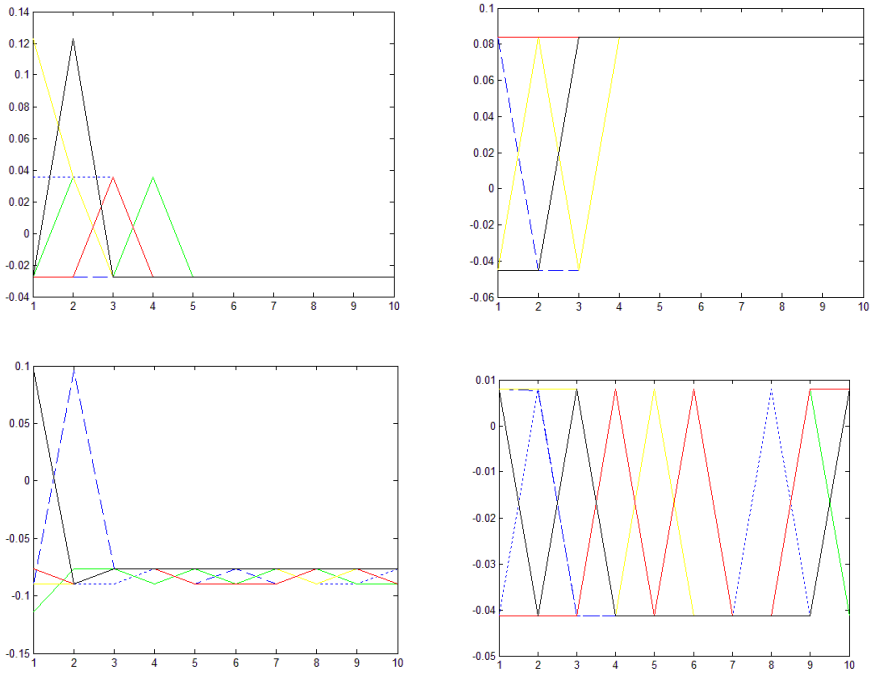
Фиг. 10. Сходимост на PAR1 при стойност на параметрите nGen = 10 (по абсцисната ос); Pm = 0,1; Pc = 0,75 и съответно nPop = 10; nPop = 20; nPop = 30

Фигура 11 показва изменението на *PAR1* съответно при P_c = 0,25; 0,50; 0,75 и 0,90. Както се вижда, най бързо се достига сходимост (още на третото поколение) при най-малката стойност на P_c, което води до извода, че по-малка стойност на вероятността за рекомбинация е за предпочитане.



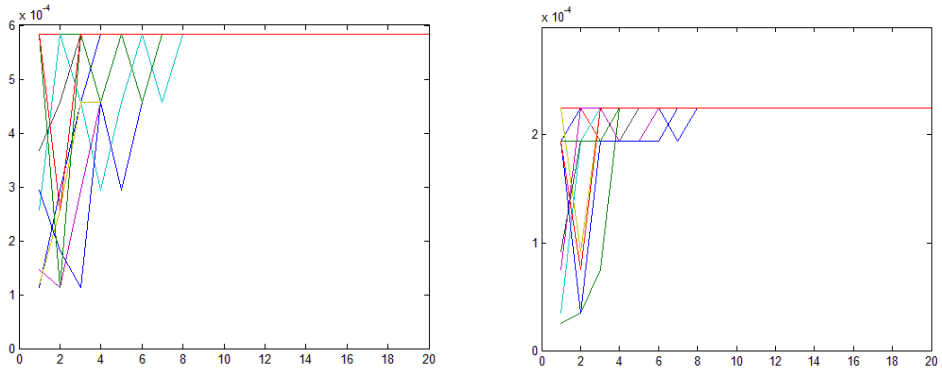
Фиг. 11. Сходимост на PAR1 при стойност на параметрите nGen = 10 (по абсцисната ос); nPop = 6; Pm = 0,1; и съответно Pc = 0,25; Pc = 0,50; Pc = 0,75; Pc = 0,90

Фигура 12 показва изменението на *PAR1* съответно при P_m *= 0,05; 0,20; 0,50 и 0,80*. При по-малки стойности на P_m се достига по-бърза сходимост.



Фиг. 12. Сходимост на PAR1 при стойност на параметрите nGen = 10 (по абсцисната ос); nPop =6; Pc = 0,75 и съответно Pm = 0,05; Pm = 0,20; Pm = 0,50; Pm = 0,80

Сходимостта на целевата функция е демонстрирана в два случая на фигура 13. Важно е да се забележи, че стойностите по ординатната ос са от порядък 10^{-4}, което показва че при всякакви стойности на параметрите намереното решение е оптимално или много близко до оптималното.



а) б)

Фиг. 13. Сходимост на целевата функция при стойности на параметрите

а) nGen = 10; nPop = 6; Pc = 0,75; Pm = 0,2

б) nGen = 20; nPop = 20; Pc = 0,75; Pm = 0,2

След направените изследвания може да се направи извод, че най-бърза сходимост за *PAR1* и *z* се постига при стойности на *nGen* = 10, *nPop* = 6, P_c = 0,25 и P_m = 0,1. Генетичният алгоритъм намира достатъчно добри резултати и при други стойности на брой поколения, размер на популацията, вероятност за мутация и вероятност за рекомбинация, но за целите на тази генетична размита система не е необходимо да се дават повече от 20 на брой поколения, 10 като стойност на размер на популацията, 0,1 за стойност на вероятност за мутация и 0,25 за стойност на вероятност на рекомбинацията.

В таблица 1 са показани оптимизираните от генетичния алгоритъм стойности на *PAR1, PAR2, PAR3, PAR4, PAR5* и *z*, и съответните стойности на параметрите на генетичния алгоритъм - брой поколения *nGen*, размер на популацията *nPop*, вероятност за мутация P_m и вероятност за рекомбинация P_c .

nGen	*nPop*	P_m	P_c	*PAR1*	*PAR2*	*PAR3*	*PAR4*	*PAR5*	*z*
10	20	0,2	0,75	0,0549	0,3435	0,4027	0,8634	1	0,0006
20	20	0,05	0,5	-0,0137	0,1944	0,3785	0,8727	1,0824	0,0006
12	8	0,02	0,2	-0,1225	0,1480	0,4074	0,7670	1,0653	0,0005
10	8	0,5	0,05	0,0764	0,1679	0,3866	0,8616	1	0,0006

Табл. 1. Оптимизирани стойности на PAR1, PAR2, PAR3, PAR4, PAR5 и z при съответните стойности на параметрите на генетичния алгоритъм

Още резултати могат да бъдат намерени в [20].

7. Заключение

Генетичните алгоритми могат да бъдат приложени при оптимизационни задачи в разнообразни области. Пространството от възможни оптимални решения е на практика почти неограничено, сложните задачи се решават сравнително бързо и надеждно, като едно от най-силните предимства е възможността за взаимодействие със съществуващи симулации и модели. Генетичните алгоритми имат и степен на гъвкавост, което ги прави подходящи при оптимизирането на размити системи и разработване на системи за взимане на решения, свързани с диагностика, наблюдение и управление.

Успешното взаимодействие на генетичните алгоритми със съществуващи симулации и модели е реализирано върху предварително създадената размита система, базирана на правила, и това е системата FSSAM.

Направените експерименти показват, че стойността на целевата функция след оптимизацията е с точност до 10^{-4}, което доказва, че създаденият генетичен алгоритъм оптимизира параметрите на размитата система. Предстоят още изследвания върху поведението на генетичния алгоритъм и неговите оператори – селекция, рекомбинация и мутация, които да оптимизират и други компоненти на размитата система, като истинското предизвикателство ще е използването на генетичен алгоритъм в процеса на оптимизиране на базата от правила.

Важно е да се отбележи, че оприимизираните стойности на параметрите на функциите на принадлежност на размитата система се различават, но незначително, от получените след много експерименти стойности, зализени в FSSAM.

References

[1] BAILEY, D. H., J. M. BORWEIN, M. LÓPEZ DE PRADO, Q. ZHU. Pseudo-Mathematics and Financial Charlatanism: The Effects of Backtest Overfitting on Out-of-Sample Performance. Notices of the AMS, 61(5), pp. 458- 471, 2014

[2] Web of Science. Изтеглено на 04.09.2015 г. от Web of Science: http://apps.webofknowledge.com/

[3] KONAR A. Artificial Intelligence and Soft Computing Behavioral and Cognitive Modeling of the Human brain. CRC Press, 2000

[4] ГЕОРГИЕВА П. Изследване на модели на софт компютинг за управление в реално време. София: АИ " проф. Марин Дринов", 2013

[5] GEORGIEVA P. V., I. P. POPCHEV. Fuzzy Q-measure Model for Managing Financial Investments. Comptes Rendus de l'Academie Bulgare des Sciences, 66(5), 2013

[6] GEORGIEVA P. V., I. P. POPCHEV AND S. N. STOYANOV. A Multi-step procedure for asset allocation in case of limited resources. Cybernetics and information technologies, 15(3), 2015

[7] KENDEL A. Fuzzy Expert Systems. Florida, USA: CRC Press, 1991.

[8] LEE M.A., and H. Tagaki. Dynamic control of genetic algorithm using fuzzy logic techniques. Fifth Int. Conf. on Genetic Algorithms (pp. 76-83). Morgan Kaufmann, CA, 1993

[9] MAGDALENA, L; F. MONASTERIO HUELIN. A fuzzy logic controller with learning through the evolution of its knowledge base. International Journal of Approximate Reasoning.Volume: 16 Issue: 3-4, Apr-May 1997.

[10] MAMDANI, E. A. An experiment in linguistic synthesis with a fuzzy logic controller. International Journal of Man-Machine Studies. Vol. 7(No. 1), 1975

[11] SUGENO, M. Industrial applications of fuzzy control. Elsevier Science Pub. Co., 1985

[12] JANG, R.; C. SUN; E. MIZUTANI. Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence. Prentice-Hall, 1997

[13] HERRERA H., O. CORDON AND M. LOZANO. A classified review on the combination fuzzy logic-genetic algorithms bibliography. Granada: Universidad de Granada, 1995

[14] MECHGOUG R., TITAOUINE. Forecasting US Dollar/Euro Rate with Genetic Fuzzy Predictor. Engineering and Technology International Journal of Social Education, 2014

[15] ZAFARI, A. Developing a fuzzy inference system by using genetic algorithm and expert knowledge. The Netherlands: Enschede, 2014

[16] KARR C.L. Design of an Adaptive Fuzzy Logic Controller using Genetic Algorithms. Int. Conf. on Genetic Algorithms (ICGA'91), (pp. 450-456). San Diego, CA, 1991

[17] SURMANN H., A. KANSTEIN AND K. GOSER. Self-Organizing and Genetic Algorithms for an Automatic Design of Fuzzy Control and Decision Systems. EUFIT'93, (pp. 1097-1104). Aachen, Germany, 1993.

[18] KINZEL J., F. KLAWOON, AND R. KRUSE. Modifications of Genetic Algorithms for Designing and Optimizing Fuzzy controllers. First IEEE Conf. on Evolutionary Computing (ICEC'94), (pp. 28-33). Orlando, FL, 1994

[19] HERRERA F. AND M. LOZANO. Adaptive GENETIC Algorithms Based on Fuzzy Techniques. IPMU'96, (pp. 775-780). Granada, 1996

[20] GEORGIEVA P. V., A. DIMITROVA. Настройване на параметрите на система FSSAM с генетичен алгоритъм. Компютърни науки и комуникации, Том 4 (1), 2015