SOFTWARE MODULE FOR PORTFOLIO SELECTION

Georgieva Penka, Burgas Free University, pgeorg@bfu.bg
Dudinov Krastyu, Together Ltd., Sofia, krastio.dudinov@together.bg
Chanev Nikolay, Burgas Free University, bafitu@gmail.com
Andonov Andon, Telerig, Sofia, andon.andonov@gmail.com

***Abstract*:** *In this article a software module for selectin investment portfolios is described. The module is the third element of the autonomic softwere system FSSAM, designed to support the process of investment decisions making for assets traded on stock exchanges. The class PortfolioElement is described in detail.*

Keywords: *fuzzy logic, portfolio selection, UML diagram*

СОФТУЕРЕН МОДУЛ ЗА КОНСТРУИРАНЕ НА ПОРТФЕЙЛИ

Пенка Георгиева, Бургаски свободен университет, pgeorg@bfu.bg
Кръстю Дудинов, Тогедър ООД, София, krastio.dudinov@together.bg
Николай Чанев, Бургаски свободен университет, bafitu@gmail.com
Андон Георгиев Андонов, Телерик АД, София, andon.andonov@gmail.com

Abstract: *В тази статия е описан софтуерен модул за конструиране на инвестиционни портфейли. Модулът е трети елемент на автономната софтуерна система FSSAM, създадена за подпомагане на процеса за взимане на инвеститорски решения за активи, търгувани на фондови борси. Подробно е описан класът PortfolioElement.*

Keywords: *размита логика, конструиране на портфейл, UML диаграма*

Приложение на размитата логика за решаване на оптимизационни задачи

Развитието на компютърните технологии прави възможно решаването на все по-сложни проблеми. Въпреки това сложността на някои от тях не позволява решаването им с разумно количество от ресурси. Някои практически проблеми за оптимизация се характеризират с нуждата от известна гъвкавост при прилагане на ограничителните условия. Тази гъвкавост може да бъде използвана за постигане на компромис между подобряване на целевата функция и удовлетворяване на ограниченията.

Целта на метод, използващ размита логика и размита оптимизация, е едновременното удовлетворяване на целите и на ограниченията при непълна, неточна, неясна или ненадеждна информация.

Размита логика се използва в решаване на задачи, свързани с автоматично управление, класификация на данни, разпознаване на образи, анализ и управление на финансови и икономически системи [1], прогнозиране на времеви редове, клъстеризиране, взимане на решения, роботика и други.

Настоящата статия е фокусирана върху софтуерно приложение за взимането на инвестиционно решение за конструиране на инвестиционен портфейл, моделирано като размита система. [2]

Формулировка на задачата за конструиране на портфейл

Инвестиционният портфейл е специфичен вид инвестиция, която се състои от разнообразни активи с различни характеристики – възвращаемост, риск, ликвидност и инвестиционен хоризонт. Важно е да се подчертае, че инвестиционният портфейл не е просто съвкупност от ценни книжа - значение имат не характеристиките на индивидуалните ценни книжа, а характеристиките на портфейла като цяло.
Изграждането и управлението на инвестиционен портфейл е метод за управление на инвестиционния риск, като за разлика от диверсификацията целта не е механично увеличаване на броя на инвестиционните носители, а чрез разглеждане на портфейла като съвкупност от различни активи да се достигне желаната възвращаемост. [3]

Нека в момента t_0 е налична сумата S_0. Целта е, след даден период от време Δt, тази сума да нарасне до сума $S^* = S_0(1+R^*)$, където R^* е желаната възвращаемост.

Нека $A_1, A_2, \ldots, A_m$ са еднородни финансови активи.

Задачата е да се конструира съвкупност от тези активи със съответни дялове $x_1, x_2, \ldots, x_m$, така че в момента $t^* = t_0 + \Delta t$ да са изпълнени следните условия:

$$\sum_{j=1}^{m} \nu_j . P_{j0} \le S_0$$

$$\nu_j \ge 0,$$

ν_j - цяло число,

$$\sum_{j=1}^{m} x_j . r_j^* \ge R^* \text{ като } x_j = \frac{\nu_j . P_{j0}}{\sum_{i=1}^{m} \nu_i . P_{i0}};$$

където x_j е дялът на актива A_j в портфейла p, $j = 1, 2, \ldots, m$;

r_j^* е възвращаемостта на актива A_j в момента t^*;

P_{j0} е цената на актива A_j в момента t_0;

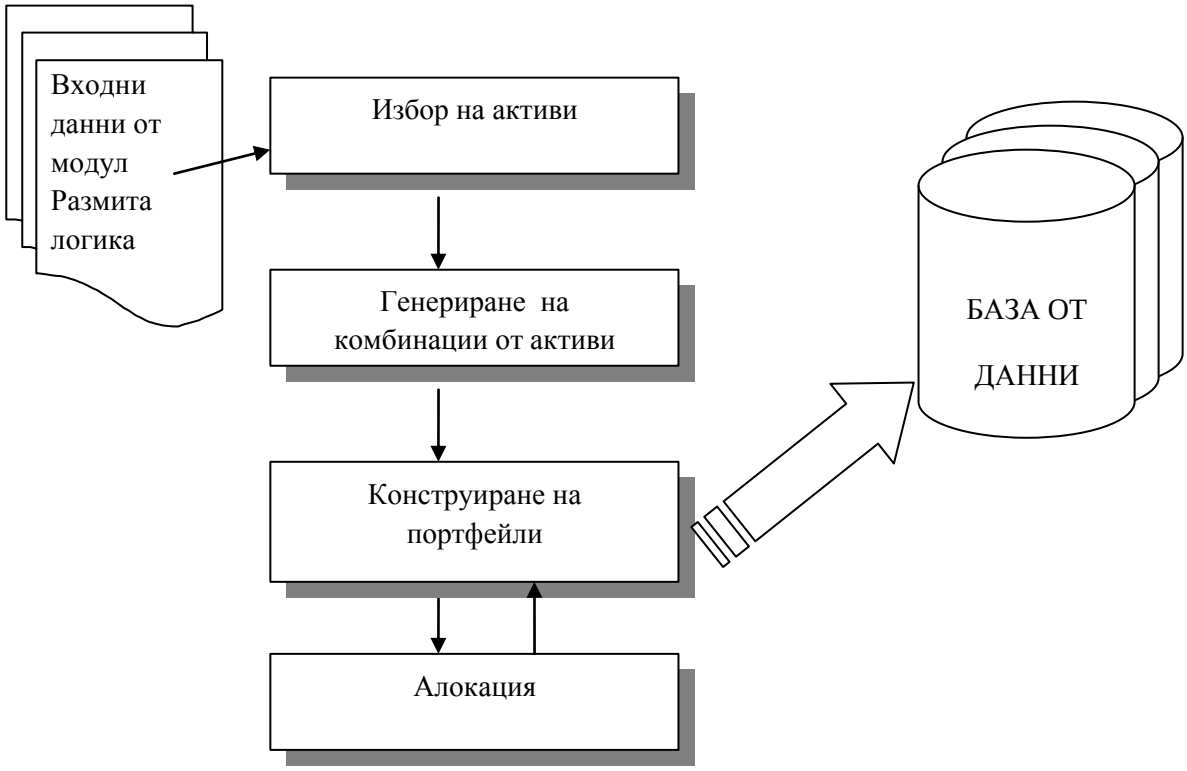
ν_j е броят на включените в p носители от вида A_j за $j = 1, 2, \ldots, m$.

Конструиран е портфейл $p = \{x_1; x_2; \ldots; x_m\}$.

При така формулираната задача, основна роля за определяне на дяловете x_j на съответните активи играят величините r_j^*. Тъй като това са възвращаемости на активите за бъдещ момент, то те са неизвестни величини. Каквито и методи да се използват при оценяване или предвиждане на тези бъдещи възвращаемости, съществува разлика между оценката им и реалните възвращаемости. Оценката на тази разлика е всъщност оценка на приетия при инвестирането риск.
Важно е да се отбележи, че оценките на възвращаемостта (и на риска) се променят динамично във времето, поради което се налагат промени и в инвеститорските решения. Един естествен подход е да се преформулира портфейлната задача през дадени интервали от време и дяловете на активите да се променят в зависимост както от поставената инвестиционна цел, така и от вече променените инвестиционни условия. Този процес общо се нарича управление на портфейл.

Модул за конструиране на инвестиционен портфейл

Модулът за конструиране на инвестиционни портфейли е самостоятелен модул на автономната софтуерна система FSSAM, подробно описана в [2]. Системата е съставена от три модула: модул *събиране и съхранение на данни*, модул *размита логика* и модул *конструиране на инвестиционни портфейли.*
На фигура 1 са представени съставните елементи на модула за конструиране на инвестиционни портфейли.



Фиг. 1. Схема на модул конструиране на инвестиционни портфейли

Входните данни се извличат от база от данни. Те представляват характеристиките на финансовите активи, след като е приключила работата на модула размита логика. *Избор на активи* селектира определен брой активи спрямо даден критерий, определен от инвеститора. От избраните активи се генерират всевъзможните комбинации от активи. След това се конструират портфейли от генерираните комбинации от активи. Тъй като е възможно избраният за инвестицията капитал да не бъде изразходен изцяло, се прилага процедура по алокация. Тази процедура модифицира портфейла с цел изчерпване на по-голямо количество от дадения капитал. Конструираните портфейли се съхраняват в база данни и инвеститорът може да вземе своето решение.

Математически модел за конструиране на инвестиционен портфейл със средствата на размита логика

Нека $A = \{a_1,\ a_2,\ a_3, \dots,\ a_n\}$ е множество от финансови активи. За всеки актив е изчислена Q-мярка на базата на размити правила и размита изходна променлива, дефинирани в модула *размита логика*.
Нека k е максималният брой финансови активи, определени да участват в инвестиционен портфейл.

Целта е конструиране на инвестиционен портфейл (подмножество на *A*), който притежава максимална Q-мярка.
Елементите на множеството от финансови активи се сортират спрямо тяхната Q-мярка. От сортираните активи се избират първите *k* на брой, след което се генерират комбинации без повторение. Първоначално се генерират комбинациите от *1*-ви клас от *k* елемента, след което се генерират комбинациите от *2*-ри клас от *k* елемента и така до *k* елемента *k*-ти клас. Така се изчерпват всички подмножества от избраните активи. Броят на тези подмножества е $2^k - 1$, тъй като не се включва празното множество, т.е. не съществува инвестиционен портфейл без активи. Всяка от генерираните комбинации преставлява инвестиционен портфейл.
За всеки актив, участващ в инвестиционен портфейл, се пресмята относителния дял на актива в портфейла. Този относителен дял се означава с x_j и се пресмята по следната формула:

$$x_j = \frac{Q_j}{\sum_{j=1}^{J} Q_j},$$

където *J* е броят на финансовите активи в инвестиционния портфейл, а Q_j е Q-мярката на актива.
Броят акции n_j, които трябва да се закупят от всеки актив се изчислява чрез следната формула:

$$n_j = \left\lfloor \frac{C}{x_j P_j} \right\rfloor,$$

където x_j е относителния дял на актива в инвестиционния портфейл, P_j е цената на една акция от актива, а *C* е капиталът, с който се извършва инвестицията.
Изчерпаният капитал *Cu* се изчислява по следната формула:

$$C_u = \sum_{j=1}^{J} n_j P_j ,$$

където P_j е цената на една акция, а n_j броят закупени акции от даден актив.
За така конструираните портфейли се прави проверка, в каква степен оползотворяват инвестиционния капитал. Ако след инвестицията останалият капитал е по-голям от даден процент от началния капитал, то трябва да се приложи процедура по допълнителна алокация. Целта на тази процедура е закупуването на допълнително количество акции, започвайки от актива с най-голяма Q-мярка, докато цената на една акция стане по-голяма от останалия капитал, при което се опитва да купи акции от следващия по ред актив в списъка. Този процес се повтаря, докато останалият капитал е достатъчно малък и не могат да се закупят допълнителни акции. На всяка стъпка броят на допълнително закупените акции n_{j_a} за даден актив се изчислява по следната формула:

$$n_{j_a} = \left\lfloor \frac{x_j(C - C_u)}{P_j} \right\rfloor,$$

където x_j е дяла на актива в инвестиционния портфейл, преди прилагане на процедурата по алокация, C е началния капитал, C_u е изразходения до момента капитал, а P_j е цената на една акция от актива.
За всеки от конструираните *M* на брой инвестиционни портфейли се изчисляват характеристиките риск и възвращаемост. Възвращаемостта R_p се изчислява по следната формула:

$$P_p = \sum_{j=1}^{J} x_j r_j ,$$

където x_j е относителният дял на актива j в портфейла, а r_j е възвращаемостта на актива.
Инвестиционният риск на портфейла се изчислява както следва:

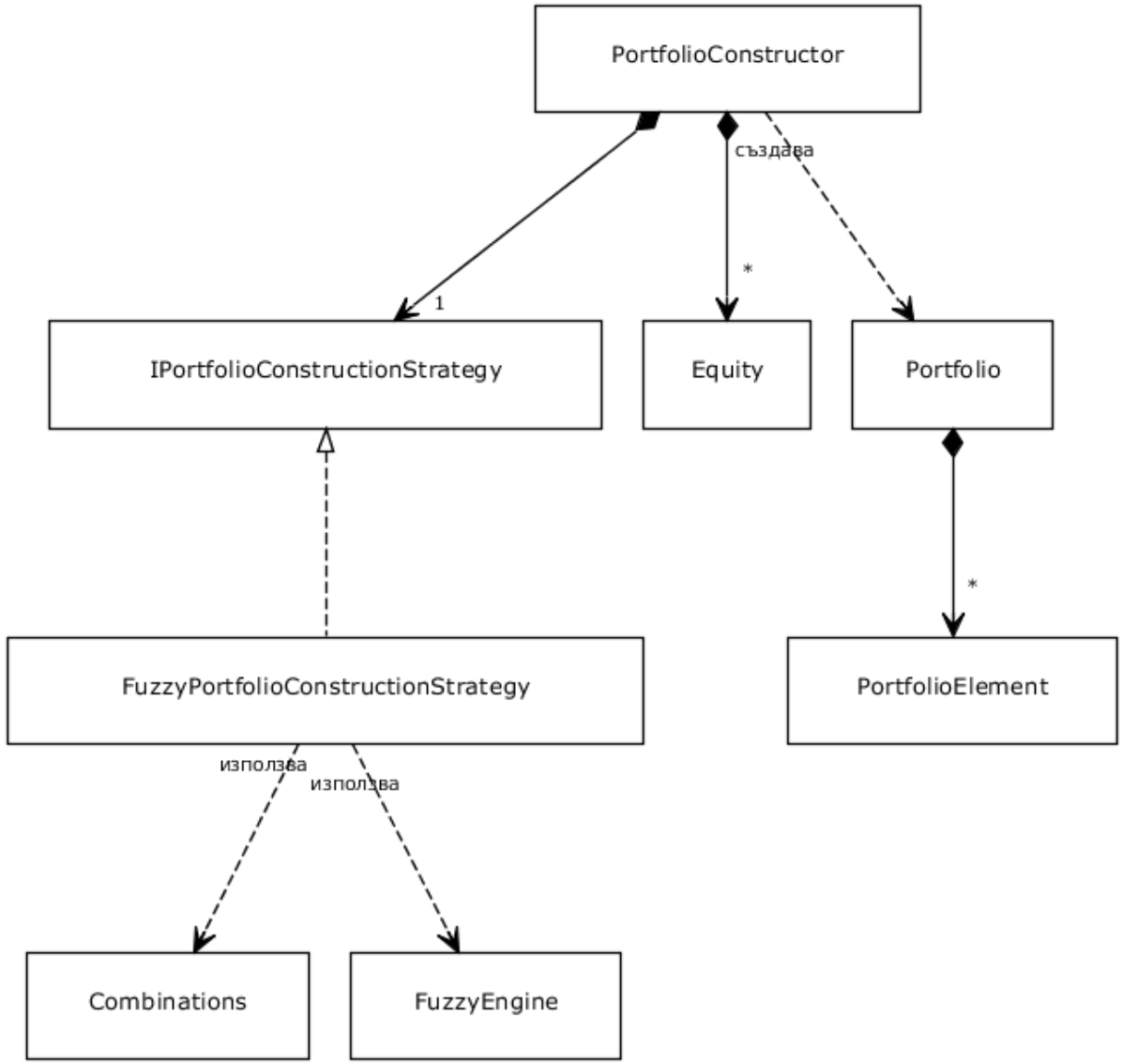
$$\sigma_p = \sum_{j=1}^{J} x_j \sigma_j ,$$

където x_j е относителният дял на актива j в портфейла, а σ_j е рискът на актива.
За определяне на Q-мярката на инвестиционния портфейл се използва размита система за взимане на решение. Входните стойности на системата са рискът и възвращаемостта на портфейла, както и тяхното отношение q_p, където:

$$q_p = \frac{R_p}{\sigma_p}.$$

Софтуерна реализация на създадения модел

Конструирането на инвестиционни портфейли се различава логически от останалите модули на софтуерната система. Поради тази причина класовете на модула са отделени в отделна динамична библиотека. UML диаграма на класовете е представена на фигура 2.
Моделът включва класове за описание на инвестиционен портфейл и актив, който представлява елемент от инвестиционния портфейл. Това са класовете *Portfolio* и *PortfolioElement*.
Също така е необходимо наличието на клас, който конструира инвестиционните портфейли, т.е. избира оптимално количество акции от даден актив, така че да бъде максимизирана възвращаемостта и минимизиран риска на портфейла като цяло. Този клас е *PortfolioConstructor*. За да изпълни своята задача е необходимо предоставянето на модел за конструиране на портфейли, както и множество от активи. Характеристиките на активите се съхраняват в база от данни и са получени в следствие на работата на втория модул на софтуерната система. Характеристиките на всеки актив се записват в полета на обекти от тип *Equity*, който се ползват от класа, конструиращ инвестиционните портфейли.
Съществуват множество различни подходи и модели за конструиране на инвестиционни портфейли, например, прилагайки модела на Марковиц или какъвто е случаят, използвайки средствата на размитата логика. За да се даде възможност да бъдат конструирани инвестиционни портфейли с различни подходи, е необходимо да се премахне директната връзка между класа, конструиращ портфейлите и конкретната стратегия за конструиране. Това може да се реализира, като се създаде интерфейс, описващ функциите, които всяка стратегия за конструиране на инвестиционен портфейл трябва да имплементира. Интерфейсът *IPortfolioConstructionStrategy* изпълнява именно тази задача.
Тъй като целта е конструирането на инвестиционен портфейл със средствата на размитата логика е необходимо да бъде създаден клас, който имплементира интерфейса и това е класът *FuzzyPortfolioConstructionStrategy*.



Фигура 2. UML диаграма на структурата на библиотека за конструиране на инвестиционен портфейл

Стойността на Q-мярката на конструираните портфейли се изчислява от *FuzzyEngine*, който е част от библиотеката за размита логика, реализирана във втория модул на софтуерният продукт.
Класът за генериране на комбинации *Combinations* е логически отделен в отделна библиотека за комбинаторика.

При реализацията на софтуерният модул е използван езикът C#. Той е съвременен, обектно-ориентиран и типово обезопасен език за програмиране, който е наследник на C и C++. В него е комбинирана леснотата на използване на Java с мощността на C++. В C# са заимствани много от силните страни на Delphi – свойства, индексатори, компонентна ориентираност. В C# са въведени и нови концепции – разделяне на типовете на два вида – стойностни (value types) и референтни (reference types), автоматично управление на паметта, делегати и събития, атрибути, XML документация и други. C# е специално проектиран за .NET Framework и е съобразен с неговите особености. Той е сравнително нов, съвременен език, който е заимствал силните страни

на масово използваните езици за програмиране от високо ниво, като C, C++, Java, Delphi, PHP и др. [4]
За реализацията на модела са създадени следните класове и интерфейси в пространството от имена *StockExchange.PortfolioConstruction*:

- *PortfolioElement;*
- *Portfolio*;
- *IPortfolioConstructionStrategy*;
- *FuzzyPortfolioConstructionStrategy*;
- *PortfolioConstructor*;
- *PortfolioUtilities.*

Освен това е дефиниран клас *Combinations* в пространството от имена *StockExchange.Algorithms.Combinatorics*.
За запис на конструираните портфейли се добавят методи в класа *StockExchangeDAL* от пространството от имена *StockExchange.Data.*
Внедряването на функционалността на библиотеката за конструиране на инвестиционни портфейли се извършва в метода Main на класа Program от пространството на имена *StockExchange.DataManager*.

Тук подробно ще бъде описан само класът *PortfolioElement.*

Реализацията на описания в предходната глава модел започва със създаване на клас, който представлява абстракцията на актив (елемент от финансов портфейл) за нуждите на модела. Класът в обектно-ориентираното програмиране (ООП) дефинира абстрактните характеристики на даден обект. Той е план или шаблон, чрез който се описва природата на даден обект. Класовете са градивните елементи на ООП и са неразделно свързани с обектите. Нещо повече, всеки обект е представител на точно един клас [5]. Представена е дефиницията на класа, като реализацията на методите е заменена със символа „...“.

```
public class PortfolioElement
{
    private string equityCode;
    private double price;
    private int count;
    private double ret;
    private double risk;
    private double ratio;
    private double qMeasure;

    public string EquityCode { get { ... } set { ... } }
    public double Price { get { ... } set { ... } }
    public int Count { get { ... } set { ... } }
    public double Return { get { ... } set { ... } }
    public double Risk { get { ... } set { ... } }
    public double Ratio { get { ... } set { ... } }
    public double QMeasure { get { ... } set { ... } }

    public PortfolioElement(string equityCode, double price, int count,
       double ret, double risk, double ratio, double qMeasure) { ... }

    public PortfolioElement()
       : this("", 0, 0, 0, 0, 0, 0) { }

    public  PortfolioElement (PortfolioElement elem)
```

```
        :this(elem.EquityCode, elem.Price, elem.Count, elem.Return,
            elem.Risk, elem.Ratio, elem.QMeasure) { }

    public override string ToString() { ... }
}
```

Полета са променливи, декларирани в тялото на класа. В тях се пазят данни, отразяващи състоянието на обекта и са нужни за работата на методите на класа. Класът притежава набор от полета, дефинирани с ниво на достъп private. Това означава, че те са достъпни само от елементите на класа, в който са дефинирани [5]. Целта на тези полета е съхранението на информация, специфична за даден актив, участващ в инвестиционен портфейл.
Полето *equityCode* е дефинирано от тип string, тъй като предназначението му е да запазва кода на актива, дефиниран от Българската фондова борса. Примери за такива кодове са: „3JR“, отговарящ на „Софарма АД“ и „57B“, отговарящ на „Булгартабак – Холдинг” АД.
Полето *price* съхранява цената на една акция от дадения актив. Цената на всеки актив се оповестява публично от Българската фондова борса с точност два знака след десетичната запетая, т.е. стойността е реално число. Поради този факт е избран тип double за съхранение на този тип променливи.
Полето *ret* съхранява стойността на възвращаемостта на даден актив. За тази цел се използва променлива от тип *double*.
Полето *risk* дава възможност на класа да опише инвестиционния риск на даден актив. Рискът е реално число, дефинирано от тип *double*.
Ratio също е поле, дефинирано от *double*. То съхранява отношението, с което даденият актив участва в инвестиционния портфейл.
Полето *qMeasure* е от особена важност в реализирания модел за конструиране на инвестиционен портфейл. То съхранява стойността, която е основен критерий за качеството на даден актив - Q-мярката на актива. Полето е дефинирано от тип *double*.

Свойствата са специален вид елементи, които разширяват функционалността на полетата като дават възможност за допълнителна обработка на данните при извличането и записването им в полетата от класа [5]. По функционалност много наподобяват мутаторните функции и функциите за достъп в C++.
Класът *PortfolioElement* дефинира свойства за всяко едно от полетата, дефинирани по-горе. Те позволяват както промяна на стойностите на полетата на класа, така и достъп до тези полета. За разлика от полетата, свойствата са дефинирани с идентификатор за достъп public. Използвайки модификатора public, се указва на компилатора, че това свойство е достъпно от всеки друг клас.

В обектно-ориентираното програмиране, когато се създава обект от даден клас, е необходимо да се извика елемент от този клас, наречен конструктор. Конструкторът на даден клас е метод, който няма тип на връщана стойност и носи името на класа за който е дефиниран. Задачата на конструктора е да инициализира заделената за обекта памет, в която ще се съхраняват неговите полетата (тези, които не са *static*). [5]
Класът *PortfolioElement* дефинира три конструктора, всеки от които притежава различна сигнатура (брой и подредба на параметрите). Това се прави с цел удобство при работа с класа в различни ситуации. Класът *PortfolioElement* се възползва от възможността на C# да преизползва конструкторите, като един конструктор може да извиква друг конструктор, деклариран в същия клас.

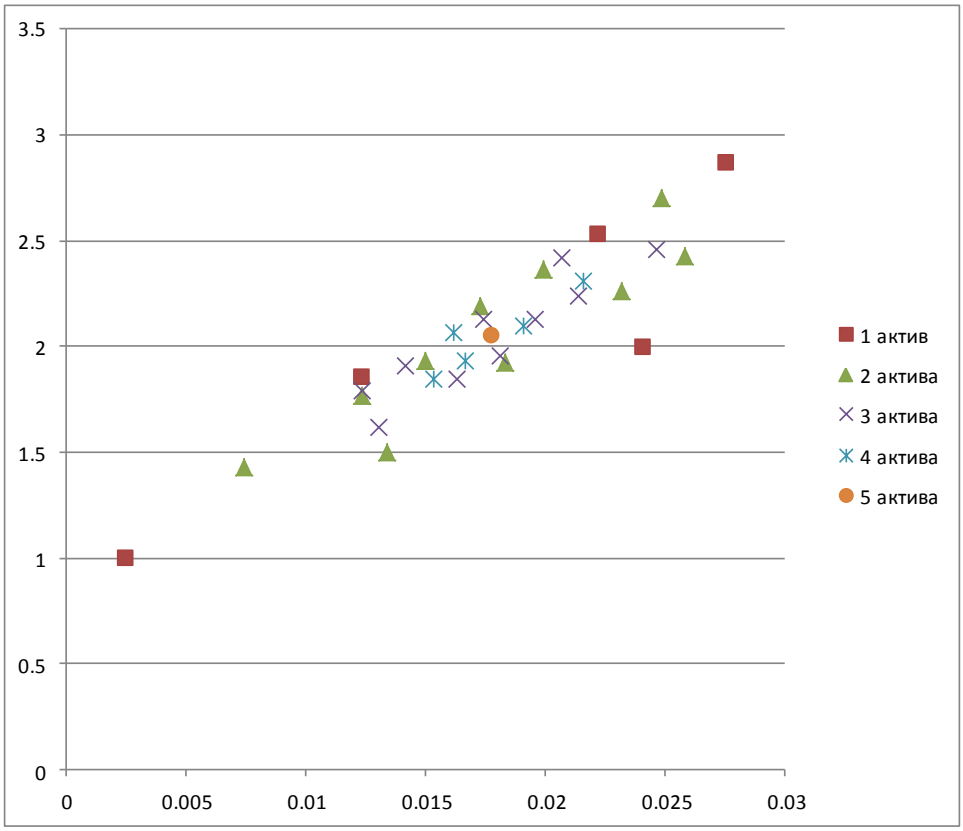
Първият конструктор създава обекти от класа, използвайки подадените му параметри. Той приема седем параметъра - по един за всяко поле на класа.
Вторият конструктор създава обект от класа, като всяко от неговите полета е инициализирано със стойност по подразбиране. В случаят за кода на актива (полето от тип string) това е празният низ "", а за останалите полета от тип *double* това е стойността 0.
Третият конструктор на класа създава нов обект от класа на базата на друг, вече създаден обект, подаден като параметър. Стойностите на полетата на новият обект са идентични с тези на параметъра. Ефектът от използването на този конструктор е подобен на клониране на даден обект.

Класът *PortfolioElement* съдържа метод *ToString(),* който представя полетата на класа в текстов формат, удобен за отпечатване на конзола.

Приложение

Модулът за конструиране на инвестиционен портфейл, като елемент на софтуерната система FSSAM, има редица приложения. Например, той може да бъде използван за:

- изследване на зависимостта между начина на определяне на относителния дял на активите в портфейл и Q-мярката на портфейла;



Фигура 3. Инвестиционни портфейли с различен брой участващи активи

- конструиране на нвестиционни портфейли с относителни дялове, зависещи от Q-мярката;
- инвестиционен портфейл с равни относителни дялове;

- изследване на зависимостта между големината на инвестиционния капитал и необходимостта от прилагане на процедура по допълнителна алокация;
- за конструиране на портфейли с различно количество инвестиционен капитал;
- изследване на възвращаемостта и риска на портфейли с различен брой активи.

На фигура 3 е показан един пример за приложение на модула. Конструирани са инвестиционни портфейли като са избрани най-добрите пет актива спрямо Q-мярката и са генерирани всички възможни комбинации от тях. Получени са 31 инвестиционни портфейла. Всеки един от тях е преставен като точка на фигурата. Инвестиционният риск е по абсцисната ос, а възвращаемостта е по ординатната ос.

Изводи

Разработеният математически модел и програмната реализация на библиотеката за конструиране на инвестиционни портфейли работят стабилно и ефективно. Моделът не определя най-добрия портфейл, а дава възможност на инвеститора да вземе своето решение на базата на информацията за финансовия пазар, предоставена от модела.
Създаденият модул констуира инвестиционни портфейли с данни за финансовите активи от Българската фондова борса и е елемент от софтурено приложение, което извършва управление на финансови данни и конструиране на инвестиционен портфейл.
В модула се прилага процедура по допълнителна алокация, която изчерпва инвестиционния капитал в оптимална степен.

Възможно бъдещо развитие на създадения модул от софтуерна система е създаването на удобен потребителски интерфейс.

References

[1] Altrock, Constantin von, Fuzzy Logic & NeuroFuzzy Applications in Business and Finance.: Prentice hall PT, 1995
[2] Георгиева, П., Изследване на модели на софт компютинг за управление в реално време, БАН, София, 2012
[3] Пътев, П.; Канарян Н., Управление на портфейла. Велико Търново: Абагар, 2008
[4] Наков, С., Програмиране на за .NET Framework I том, Фабер
[5] Наков, С., Въведение в програмирането със C#, 2011. ISBN 978-954-400-527-6