ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

Кафедра общей математики

Методы минимизации функции одной переменной

Выпускная квалификационная работа

студентки *очной* формы обучения
направления подготовки 01.03.02 «Прикладная математика и информатика»
4 курса группы 07001206
Сениной Анастасии Сергеевны

Научный руководитель:
доцент кафедры
общей математики,
кандидат физ.-мат. наук,
Флоринский В.В

БЕЛГОРОД 2016

Оглавление

Введение

Широкое внедрение вычислительной техники и систем управления во все сферы человеческой деятельности и усложнение задач, решаемых этими системами, требуют совершенствования существующих методов логического проектирования цифровых устройств и разработки новых. Логическое проектирование при этом понимается в широком смысле, включая не только статику систем, т.е. их структуру и функциональные связи, но и анализ динамики как на уровне структуры, а также на уровне переходных процессов, связанную с изменением временными характеристиками элементов и переменных. Поэтому исследования, направленные на развитие методов логического проектирования цифровых устройств, обеспечивающих упрощение процедуры проектирования, улучшение основных характеристик ЦУ, а также снижение времени и стоимости разработки никогда не потеряют своей актуальности.[1, 2]

Проблеме логического проектирования цифровых устройств посвящено большое число работ, большинство из них основано на представлении логических функций (ЛФ) в точках области их определении значениями логического 0 или 1.

В то же время со многими положительными сторонами такого представления оно имеет и ряд отрицательных, а именно при большом числе переменных, так как при возрастании числа переменных происходит обвальное увеличением количества точек области определения и, как следствие, затруднение в процедуре синтеза и анализа, связанное с решением объёмных совмещенных задач. Эти недостатки существенны не только при аналитических методах, но также при использований программных средств анализа и синтеза. [2, 12-15]

А так же многие исследователи пошли по пути нестандартных подходов к преобразованию и представлению логических функций, основанном на концепции многозначного алфавита, конечных логических

шкал, предикатов, арифметических и интерполяционных полиномов, спектрального представления и др., которые в некоторых случаях дали положительный эффект как при синтезе, а также анализе цифровых устройств. [3]

К предлагаемому нетрадиционному способу можно отнести представление функций одной переменной в форме обобщённых, когда значения функции в точках её области определения заданы не только значением логического 0 и 1, но и независимыми или зависимыми параметрами. В частности, к такой форме можно отнести неполное разложение Шеннона. Вместе с тем, коэффициенты, образуемые литералами переменных, по которым выполняется разложение, можно рассмотреть как координаты точек области определения, а остаточные функции — как параметра определяющие значения функции в этих точках. Такой подход ведёт к существенному уменьшению числа точек области определения, что служит облегчению процедуры логического проектирования цифровых устройств, учитывая, связанные с представлением и минимизацией функций, синтезом комбинационных схем и автоматов с памятью, решением задач динамики цифровых устройств, связанных с анализом состязаний в комбинационных схемах, а также нахождением булевых производных.

Цель работы — разработка методов и алгоритмов логического проектирования цифровых устройств, обеспечивающих упрощение процедуры и снижение времени проектирования, основанных на представлении и преобразовании функций одной переменной в обобщённой форме.

Поставленная цель работы предполагает решение следующих задач:

- провести анализ современного состояния методов логического проектирования цифровых устройств;
- разработать программное обеспечение для сравнения эвристических методов минимизации функций.

1 Методы минимизации функций одной переменной

1.1 Понятие оптимизации функций

Оптимизация – это выбор наилучшего решения. Математическая теория оптимизации включает в себя фундаментальные результаты и численные методы, позволяющие находить наилучший вариант из множества возможных альтернатив без их полного перебора и сравнения.

Для того чтобы использовать результаты и вычислительные процедуры теории оптимизации на практике, нужно сформулировать рассматриваемую задачу на математическом языке, т.е. построить математическую модель объекта оптимизации. Математическая модель – это более или менее полное математическое описание исследуемого процесса или явления.[3]

Во многих реальных ситуаций дать полное математическое представление оптимизируемой системы с учетом взаимодействий с внешним миром, всех взаимосвязей ее частей, всех целей ее функционирования бывает затруднительно или невозможно. Поэтому при создании математической модели необходимо, в большинстве случаев, выделять и учитывать в дальнейшем только наиболее важные, существенные стороны исследуемого объекта с тем, чтобы было возможным его математическое описание, а также последующее решение поставленной математической задачи. Вместе с тем неучтенные в математической модели факторы не должны существенно оказывать влиять на итоговый результат оптимизации. Таким образом, математическое моделирование является сложной и ответственной задачей, требующей от исследователя глубоких знаний в соответствующей области, практического опыта, интуиции и критического анализа получаемых результатов. [17]

Тем не менее, можно условно разбить процесс математического моделирования на следующие основные этапы, так как общего рецепта построения математических моделей оптимизации не существует.

1.1.1 Определение границ объекта оптимизации

Необходимость этого этапа диктуется невозможностью учета и исчерпывающего описания всех сторон большинства реальных систем. Определив главные параметры, ограничения, и переменные, необходимо приближенно представить систему как некоторую изолированную часть реального мира и упростить ее внутреннюю структуру.

Например, при оптимизации работы одного из цехов предприятия иногда можно пренебречь влиянием особенностей функционирования других цехов, сбыта всего предприятия и систем снабжения, его взаимодействием с другими организациями, конъюнктурой рынка и многими другими факторами. Тогда цех будет рассматриваться как изолированная система, а его связи с внешним миром либо не учитываются, либо считаются зафиксированными.

Может оказаться, что начальные границы объекта оптимизации выбраны не корректно. Это становится ясным при дальнейшем оценке системы и ее математической модели, при интерпретации результатов поиска оптимального решения, сопоставлении их с практикой и т.д.

Тогда в одних случаях границы системы следует расширить, а в других – сузить. Например, если выясняется, что влияние на работу исследуемого цеха других подразделений предприятия нельзя игнорировать при ее оптимизации, то необходимо включить в систему и эти подразделения. С другой стороны, может оказаться, что сам цех состоит из нескольких в большой степени независимо работающих участков, которые без значительного упрощения реальной ситуации можно рассматривать изолированно. Тогда для облегчения поиска оптимального решения разумно исследовать каждый участок как отдельную систему.[18]

В инженерной практике следует, насколько возможно, стремиться упрощать системы, подлежащие оптимизации, разбивать сложные системы на более простые подсистемы, если есть уверенность, что это повлияет на окончательный результат в допустимых пределах.

1.1.2 Выбор управляемых переменных

На этом этапе математического моделирования следует найти разницу между теми величинами, значения которых можно варьировать и отдавать предпочтение с целью достижения оптимального результата и величинами, которые константы или определяются внешними факторами. Нахождение тех значений управляемых переменных, которым уместна оптимальная ситуация, и представляет собой задачу оптимизации.

Одни и те же величины, в зависимости от выбранных границ оптимизируемой системы и уровня детализации её описания, могут оказаться либо управляемыми переменными, либо нет. Например, в упомянутой ситуации с оптимизацией работы цеха объем поставок какого–либо сырья из другого цеха в одних случаях следует закреплять, то есть не зависящих от нашего выбора, а в иных случаях – регулируемым, т.е. управляемой переменной. [11]

1.1.3 Определение ограничений на управляемые переменные

В реальных условиях на выбор значений управляемых переменных, как правило, наложены ограничения, связанные с ограниченностью имеющихся ресурсов, мощностей и других потенциалов. При построении математической модели эти ограничения необходимо внести в виде равенств и неравенств или указывают множества, которым должны принадлежать значения управляемых переменных. Сумма всех ограничений на управляемые переменные устанавливает допустимое множество задачи оптимизации.

Например, если годовой объем выпускаемой цехом продукции данного вида является управляемой переменной, то ее значения, во–первых, не могут быть отрицательными и, во–вторых, ограничены сверху максимальной производительностью оснащения (оборудования) цеха.

1.1.4 Выбор числового критерия оптимизации

Неизбежной частью математической модели объекта оптимизации является числовой критерий, минимальному или максимальному значению которого (в зависимости от конкретной задачи) отвечает лучшая вариация поведения изучаемого объекта. Величина этого критерия целиком определяется выбранными значениями управляемых переменных, т.е. он является функцией этих переменных и называется целевой функцией.

В инженерной практике употребляется широкий спектр критериев оптимизации. Это могут быть критерии экономического характера, например, прибыль, себестоимость, капитальные затраты и т.д., физические или технические параметры системы – длительность технологического процесса, используемая энергия, максимальная механическая нагрузка, завоеванная скорость движения и другие. [34]

Следует отметить, что во многих случаях выбор критерия оптимизации не является явным и однозначным. Часто бывает трудно поставить в соответствие всей совокупности целей функционирования системы какой–либо один критерий. Это объясняется такими причинами, как неясность формулировок некоторых целей, мешающая описанию их с помощью количественных характеристик, сложность целевой функции, описывающей большую совокупность смешанных целей, наличие двойственных целей, важность каждой из которых зависит от точки зрения и т.д. Например, невозможно найти решение, обеспечивающее одновременно максимальную надежность и быстродействие, но минимальные затраты и энергопотребление.

Выход из этого положения определяется в каждом конкретном случае. В частности, из многих критериев, характеризующих разнообразные цели оптимизации, выбирают один, причем, считая его важнейшим, а остальные – второстепенным. Далее второстепенные критерии либо не учитываются, либо учитываются частично с помощью дополнительных ограничений на

управляемые переменные. Эти ограничения обеспечивают изменение второстепенных критериев в заданных диапазонах приемлемых значений.

Другой путь состоит в формулировке комплексного критерия, т.е. целевой функции, включающей с разумно выбранными весовыми коэффициентами целевые функции, соответствующие различным целям. [28]

1.1.5 Формулировка математической задачи оптимизации

Связывая результаты предшествующих этапов построения математической модели, ее вносят в виде математической задачи оптимизации, в которую входят построенная целевая функция и найденные ограничения на управляемые переменные. В общем виде математическую задачу оптимизации можно выразить следующим образом; минимизировать (максимизировать) целевую функцию с учетом ограничений на управляемые переменные.

Под минимизацией (максимизацией) функции n переменных $f\ x = x_1, \dots, x_n$ на заданном множестве U n–мерного векторного пространства E_n понимается определение хотя бы одной из точек минимума (максимума) этой функции на множестве U, а также, если это необходимо, и минимального (максимального) на множестве U значения $f\ x$. При записи математических задач оптимизации в общем виде обычно используется следующая символика:

$$f(x) \to \min(max),$$
$$x \in U$$

где $f\ (x)$ – целевая функция, а U – допустимое множество, заданное ограничениями на управляемые переменные.

Для определения экстремумов дифференцируемой функции одной переменной $f(x)$ нужно решить уравнение $f'(x) = 0$. Но только в некоторых случаях, решение этого уравнения получается найти в явном виде. Обычно, его приходится решать каким-либо численным методом, причем задача

численного решения уравнения $f'(x)=0$ примерно так же сложна, как и исходная задача поиска экстремума функции $f(x)$. Более того, в практике оптимизации необходимо рассматривать функции $f(x)$, которые не являются дифференцируемыми. Поэтому рождается задача построения численных алгоритмов поиска экстремума функции одной переменной. Следует заметить, что универсальных методов, годных для оптимизации произвольных функций одной переменной не существует, в связи с чем приходится конструировать алгоритмы, ориентированные на разнообразные, встречающиеся в прикладных задачах, классы функций. [29]

Необходимость отдельного рассмотрения численных методов поиска экстремума функций одной переменной, истолковывает вытекающими условиями.

Во-первых, такие методы используются как вспомогательные процедуры во многих алгоритмах поиска экстремума функций многих переменных. Решение задачи (1) можно находить с помощью схемы повторной оптимизации, основанной на том, что

$$\min_{(X^1,X^2)\in D} f(X^1,X^2)=\min_{X^1\in D_1}(\min_{X^2\in D_2} f(X^1,X^2))=\min_{X^2\in D_2}(\min_{X^1\in D_1} f(X^1,X^2))$$

для произвольных множеств D_1, D_2. Здесь X^1, X^2 - вектора соответствующей размерности, $D=D_1\times D_2$ - декартово произведение множеств D_1, D_2. Если x_1, x_2 - скаляры, то с помощью алгоритма одномерной оптимизации можно вычислить значения функции

$$\varphi(x_1)=\min_{x_2\in D_2} f(x_1,x_2),$$

а затем минимизировать эту функцию и тем самым находить решение задачи

$$\min_{(x_1,x_2)\in D=D_1\times D_2} f(x_1,x_2)=\min_{x_1\in D_1}\varphi(x_1).$$

Абсолютно аналогично можно поступать, когда число переменных функции $f(x)$ больше двух.

Во-вторых, во многих алгоритмах поиска экстремума функций многих переменных. Действительно, пусть требуется решить задачу

$$f(X) \to \min,\ X = (x_1,..., x_n) \in D \subseteq R^n, \quad (1)$$

где $R^n - n-$мерное евклидово пространство, и для ее решения используется метод, задаваемый соотношением вида:

$$X^{k+1} = X^k + \beta_k H^k, k = 0,1,2,..., \quad (2)$$

где $X^k = (x_1^k,..., x_n^k) \in R^n, H^k = (h_1^k,..., h_n^k) \in R^n, \beta_k -$числовой параметр.

При этом конкретный алгоритм определяется заданием точки X^0, правилами выбора векторов H^k и чисел β_k на основе, полученной в результате вычислений информации, а также условием остановки. Если H^k является направлением убывания функции $f(x)$ в точке X^k, т.е. $f(X^k + \beta H^k) < f(X^k)$ при всех достаточно малых $\beta > 0$, то коэффициенты β_k в методе (2) выбираются из условия

$$f(X^k + \beta_k H^k) = \min_{\beta > 0} f(X^k + \beta H^k), X^k + \beta_k H^k \in D\ , \quad (3)$$

либо из условия

$$f(X^k + \beta_k H^k) < f(X^k),\ k = 0,1,2,....$$

Для нахождения β_k, удовлетворяющего неравенству (3), можно также использовать какой-либо метод оптимизации.

В-третьих, одной из особенностей задач оптимального проектирования является то, что в систему ограничений, описывающих требования, которые накладываются на характеристики проектируемого устройства, могут входить характеристики $g_i(X, v),\ j = 1,..., m$, зависящие от вектора варьируемых параметров $X \in D_X = \{X = (x_1,..., x_n) \in R^n : a_i \le x_i \le b_i, i = 1,..., n\}$ и параметра $v \in [v^-, v^+]$. Таким параметром может быть частота, время, температура и т.д. Например, при проектировании фильтров технические требования к частотным характеристикам $\varphi_j(X, \omega)$ связаны с выполнением условий

$$\varphi_j^- \le \varphi_j(X, \omega) \le \varphi_j^+, j = 1,...,m, \quad (4)$$

для всех значений частоты $\omega \in [\omega^-, \omega^+]$. Здесь компонентами вектора варьируемых параметров X являются значения сопротивлений, емкостей, индуктивностей. Учет ограничений (4) приводит к анализу неравенств

$$\max_{\omega\in[\omega^-,\omega^+]} \varphi_j(X,\omega) \le \varphi_j^+,$$
$$\min_{\omega\in[\omega^-,\omega^+]} \varphi_j(X,\omega) \ge \varphi_j^-, \qquad (5)$$

т.е. к решению одномерных задач оптимизации. Выполнение соотношений (5) означает справедливость неравенств (4) при заданном $X \in D_X$ для наихудших значений ω , с точки зрения выполнения неравенств, а, следовательно, и для всех $\omega \in [\omega^-, \omega^+]$.

И, наконец, классы функций одной переменной служат удобной моделью для теоретического исследования эффективности методов оптимизации. Здесь следует сказать, что некоторые из описываемых ниже алгоритмов являются в том или ином смысле оптимальными. В связи с этим остановимся подробнее на подходе, связанном с построением оптимальных алгоритмов.

Пусть решается задача

$$f(x) \to \min, x \in [a,b], \qquad (6)$$

где функция $f(x)$ принадлежит некоторому классу функций F, а алгоритм ее решения α может быть выбран из некоторого класса алгоритмов A. Качество решения задачи (6) для некоторой функции $f(x) \in F$ при помощи алгоритма $\alpha \in A$ будем оценивать критерием эффективности

$$Q = Q(f, \alpha).$$

Заметим, что фиксация класса F не означает, что алгоритм, которому отдаем предпочтение, определен для минимизации (максимизации) многих функций. Даже если требуется минимизировать (максимизировать) единственную функцию, на этапе начального изучения необходимо определять имеются ли свойства функции, имеющие большое значение со стороны выбора оптимизационного алгоритма. Определив свойства функции, которые будут приняты во внимание при выборе алгоритма, мы тем самым фиксируем класс F. Задание же класса A алгоритмов, которому будет отдаваться предпочтение алгоритм α , зависит от того, возможно ли вычислить производную функции $f(x)$ или только ее значения, от порядка

поступления информации, приобретаемой в итоге вычислений, возможностей по ее обработке, хранению и т.д. Величина $Q(f,\alpha)$ определяет меру результативности решения задачи минимизации функции f алгоритмом α. В качестве критерия $Q(f,\alpha)$ может выступать, например, число шагов алгоритма или точность (погрешность) решения задачи при заданном объеме вычислений (числе шагов алгоритма), требуемое для получения решения задачи с заданной точностью.

Для оценки эффективности алгоритма $\alpha \in A$ на всем классе функций F необходимо внести некоторое дополнительное предположение в схему построения оптимальных алгоритмов. Обозначим через $Q(\alpha)$ гарантированное значение критерия эффективности при поиске минимума функции $f(x)$ из класса F с помощью алгоритма α, т.е.

$$Q(\alpha) = \sup_{f \in F} Q(f,\alpha).$$

Тогда оптимальным алгоритмом $\alpha^* \in A$ будет алгоритм, для которого верно

$$Q(\alpha^*) = \min_{\alpha \in A} Q(\alpha). \qquad (7)$$

Алгоритм α^ε называется $\varepsilon-$оптимальным на классе функций F, если

$$Q(\alpha^\varepsilon) \le \inf_{\alpha \in A} Q(\alpha) + \varepsilon, \ \ \varepsilon > 0. \qquad (8)$$

Изложенный подход является минимаксным и дает гарантированный результат. Это значит, что для любой функции $f \in F$ качество решения задачи (6) с помощью алгоритма α^* будет не хуже, чем $Q(\alpha^*)$. Например, если в роли критерия эффективности выступает погрешность решения задачи, то оптимальный алгоритм минимизирует максимальную на классе функций F погрешность и обеспечивает тем самым достижение наилучшего гарантированного на классе F результата. [44]

Допустимы и другие подходы к построению оптимальных алгоритмов.

Необходимо отметить, введённое понятия оптимальности, будем считать для определенности, что анализируемые алгоритмы образованы

лишь на вычислении значений функции. Тогда оптимальные минимаксные алгоритмы поиска экстремума, о которых говорилось ранее, гарантируют оптимальное значение критерия эффективности, если на каждом шаге вычислений функция получает наихудшее, возможное с точки зрения принятого критерия значение. Однако перед вычислением функции на очередном шаге алгоритма предыдущая информация о функции может оказаться не наихудшей. Поэтому необходимо отдавать предпочтение точкам вычисления функции на следующих шагах алгоритма так, чтобы улучшить оптимальное, гарантированное до начала вычислений, значение критерия эффективности. [48]

1.2 Постановка задачи на проектирование

Целью дипломной работы является разработка программного обеспечения для сравнения алгоритмов минимизации функций одной переменной.

Для достижения цели в указанной постановке необходимо решить ряд частных задач:

- провести анализ современного состояния методов логического проектирования цифровых устройств;
- использовать методы минимизации обобщённых функций одной переменной (ОЛФ) с независимыми и зависимыми параметрами;
- рассмотреть алгоритм и программные средства преобразования традиционных функций одной переменной в форму ОЛФ с зависимыми параметрами, основанные на сжатии их области определения;
- произвести выбор средств разработки программного обеспечения;
- разработать структуру программного обеспечения;
- разработать программное обеспечение;
- сравнить результаты использования алгоритмов минимизации функций одной переменной.

2 Выбор средств разработки и проектирование программного средства

2.1 Выбор языка и среды программирования

Программное обеспечение (ПО) представляет собой совокупность компьютерных программ, описаний и инструкций по их применению на ЭВМ. ПО делится на два класса:

- Общее ПО (операционные системы, операционные оболочки, компиляторы, интерпретаторы, программные среды для разработки прикладных программ, СУБД, сетевые программы и так далее);
- Специальное ПО (Совокупность прикладных программ, разработанных для конкретных задач в рамках функциональных подсистем).

Программное обеспечение (ПО) — совокупность программ для реализации целей и задач автоматизированной системы. [2]

Для качественной работы и использования программы необходима операционная система. Операционные системы реализовывают управление работой персональных компьютеров, их ресурсами, запускают на выполнение разнообразные прикладные программы, выполняют различные вспомогательные действия по запросу пользователя. ОС деляться на однопользовательские, многопользовательские, и сетевые. К факторам, которые влияют на выбор конкретной ОС, относятся:

- необходимое число поддерживаемых программных продуктов,
- требования к аппаратным средствам,
- требование поддержки сетевой технологии,
- наличие справочной службы для пользователя,
- быстродействие,
- наличие дружественного интерфейса и простота использования.

Для создания программ под Windows имеется огромное количество интегрированных сред разработки, ним можно отнести: VisualBasic, Visual C+ +, Delphi, C+ + Builder.

С появлением средств быстрой разработки приложений (RAD – rapidapplicationdevelopment) возникла вероятность программировать с помощью готовых компонентов и шаблонов. Сравним Delphi и C++Builder, выберем среду программирования для автоматизированной системы.

Система визуального программирования Delphi, разработана компанией BorlandInternational на базе языка ObjectPascal. Объектно-ориентированный подход к созданию компонент был серьезным шагом вперед. Дополнительный выигрыш обеспечивался за счет нормальной компиляции, обеспечивающей получение более производительных программ. Первые две версии Delphi довольно быстро вызвали интерес не только у вузовских аудиторий, где в основном изучали Pascal, но и среди профессионалов. Это подтолкнуло одно из подразделений фирмы BorlandInternational на замену визуальной технологии в среду C++. Так, в то же время с появлением Delphi 2.0 на рынке возникла первая версия Borland C++ Builder (BCB). Основание первой версии BCB составила библиотека визуальных компонент VCL (VisualComponentLibrary), перенесенная без изменений из Delphi 2. Интерфейсы сред Delphi и BCB похожи друг на друга как близнецы, да и большая часть BCB была разработана на языке ObjectPascalв среде Delphi. Благодаря своему происхождению система BCB оказалась двуязычной. Кроме своего основного языка программирования она позволяет практически без каких-либо доработок использовать формы, объекты и модули, разработанные в среде Delphi. Чтобы еще больше расширить сферу влияния среды BCB, ее авторы в последующих версиях обеспечили возможность использования библиотеки классов MFC (MicrosoftFoundationClasses), разработанной фирмой Microsoft.

Delphi 7 2010 г. – это прекрасный инструмент, но в ту же минуту и сложная программная среда, которая включает в себя множество элементов.

Состоит из нового интерфейса Galileo, а также interbase server и desktop, remote debugger server, Model Maker, Install Shield

Особенно привлекательными в Delphi являются такие начальные возможности, как объектно-ориентированный подход к программированию, созданный на формах, ее высокопроизводительный (32-разрядный оптимизирующий компилятор), отличная поддержка баз данных, близкая интеграция с программированием под Windows и технология компонентов. Но самой важной частью является язык ObjectPascal, на фундаменте которого строится все остальное.

Delphi 7 2010 г имеет открытую архитектуру, которая всецело поддерживает технологии MicrosoftOLEAutomation, ActiveX, ODBC. Компилятор разрешает иметь доступ ко всем ресурсам операционных систем, реализующих интерфейс Win32 (Windows XP и Windows 7).

Программы Delphi применяют объектно-ориентированную структуру под названием VCL – VisualComponentLibrary (Библиотека Визуальных Компонентов). Именно VCLподнимает быструю разработку приложений на новый уровень. Можно расширить свои возможности за счет создания своих собственных компонентов. К тому же независимые поставщики уже создали множество компонентов такого рода.

Delphi 7 2010г имеет много других улучшений IDE, расширенную поддержку баз данных (по специальным наборам данных ADOи InterBase), усовершенствованную версию MIDASc поддержкой Интернета, инструмент управления версиями TeamSours, возможности перевода, концепцию фреймов и большое количество новых компонентов. [22]

В основе Delphi лежат технологии визуального проектирования и программирование процедур обработки событий, применение которых позволяет существенно сократить время разработки и облегчить процесс создания приложений.

C++Builder 6 2010 – очередная версия системы объектно-ориентированного программирования для операционных систем Windows

2000, WindowsXP, WindowsVista и Windows 7. Интегрированная среда системы (IntegratedDevelopmentEnvironment, IDE) обеспечивает ускорение визуального проектирования, а также продуктивность многократно используемых компонентов в сочетании с усовершенствованными инструментами и разномасштабными средствами доступа к базам данных.

Стандарты пользовательских интерфейсов меняются и развиваются так же быстро, как и операционные системы. Открытость среды IDEпозволяет настраивать ее с учетом наиболее модных тенденций в области графических интерфейсов. Визуальный интерфейс сочетает в себе простоту использования для новичка и богатство возможностей для профессионала. [18]

Система C++Builder 6 2010 г может быть использована везде. Необходимо дополнение имеющихся уже приложений (как прикладные, так и системные) расширенным стандартом языка C++, повысить быстродействие и надежность программ, оформить пользовательский интерфейс на высоком профессиональном уровне, что позволит быстро создавать наиболее часто употребляемые сенсорный ввод данных, графические интерфейсы и приложения для КПК, сенсорных панелей и автономных общедоступных систем. А так же модернизировать существующие приложения с минимальным добавлением кода или без него.

В контексте C++BuilderRAD подразумевает не только реальное ускорение типичного цикла «редактирование – компиляция – компоновка – прогон – отладка», но и придает создаваемым проектам изящество компонентной модели. [20]

Уникальная среда разработки C++Builder 6 2010 IDEInsight дает возможность обращаться ко всем потенциалам, параметрам и компонентам интегрированной среды разработки, не теряя время на их поиск в меню и диалоговых окнах; обозреватель классов, обеспечивающий управление классами в проекте и быстрый переход между ними; объединяет Дизайнер форм, Инспектор объектов, Палитру компонентов, Менеджер проектов и

полностью интегрированные Редактор кода и Отладчик – основные инструменты RAD.

В Builder C++ 6 2010 г. добавлены поддерживаемые отладчиком средства визуализации данных, упрощающие отладку, что позволяет настраивать отображение типов данных в отладчике. Поддерживаемые отладчиком средства управления потоками, обеспечивающие заморозку, разморозку и изоляцию потоков, а также установку контрольных точек для выбранных потоков, что упрощает разрешение проблем; Builder C++ 6 2010 содержит новые параметры отладчика: Scroll new events into view («Прокрутка новых событий в представлении») и Ignore non-user breakpoints («Игнорирование не пользовательских контрольных точек»), как на уровне исходных инструкций, так и на уровне ассемблерных команд - в расчете удовлетворить высокие требования программистов-профессионалов.

Оптимизирующий 32-разрядный компилятор построен по оригинальной и проверенной адаптивной технологии, обеспечивающей исключительно надежную и быструю оптимизацию, как объема выходного исполняемого кода, так и требуемой памяти.

Визуальная разработка методом «перетаскивания» (drag-and-drop) многократно упрощает и ускоряет трудоемкий процесс программирования приложений СУБД. В основе объектно-ориентированного взаимодействия клиент – сервер лежит понятие наборов данных (dataset) – таблиц, запросов, хранимых процедур – основных сущностей БД, которыми оперируют компоненты доступа. Широкий выбор компонентов визуализации и редактирования позволяет легко изменять вид представления наборов данных. C++Builderиспользует проводник баз данных и масштабируемый словарь данных для того, чтобы автоматически настроить средства отображения и редактирования применительно к специфике вашей информации.

Наряду с дизайнерами и художниками-оформителями к прикладным разработкам в ключевых областях сети все чаще привлекаются

профессионалы-программисты. Качественное приложение способно динамически выбирать информацию с сервера и предоставлять ее в формате, удобном для потребителей разного уровня, так, чтобы они смогли составить свое заключение и принять адекватное решение в кратчайший срок. C++Builder 6 2010 полностью удовлетворяет этим требованиям, обеспечивая:

- высокую надежность, степень интеграции и качество управления;
- быстрое визуальное проектирование эффективных приложений для переработки больших объемов информации;
- поддержку механизмов принятия решения и доступа к удаленным базам данных.

C++Builderпредоставляет свою мощь и широкие возможности языка C++ всему семейству систем объектно-ориентированного программирования. Система C++Builderможет быть использована везде, где требуется дополнить существующие приложения расширенным промышленным стандартом языка C++, повысить быстродействие и придать пользовательскому интерфейсу профессиональный облик.

Все компоненты, формы и модули данных, которые накопили программисты, работающие в Delphi, могут быть многократно использованы в приложениях C++Builderбез каких бы то ни было изменений. C++Builder идеально подойдет тем разработчикам, которые предпочитают выразительную мощность языка C++, однако хотят сохранить продуктивность Delphi. Уникальная взаимосвязь этих систем программирования позволяет при создании приложения без труда переходить из одной среды разработки в другую.

Какую систему выбрать? Delphi использует язык Объектный Паскаль, который преподается во многих специализированных школах и учебных институтах. Система C++Builder, как следует из названия, построена на языке C++, который наиболее распространен в крупных фирмах, занимающихся разработкой математического обеспечения профессионального уровня. C++Builder является более мощной системой,

которая идеально подходит разработчикам, которые предпочитают выразительную мощь языка C++, однако хотят сохранить продуктивность Delphi. Сопровождение программных продуктов, написанных в системе Delphi проще, чем написанных в C++Builder.

Проведем выбора среды программирования методом экспертного оценивания. Выделим критерии оценки среды программирования. Важность каждого из представленных критериев была оценена экспертами по 100 бальной шкале. Исходя из полученных данных, находится средний балл и коэффициент относительной важности критерия. Результаты экспертизы представлены на рисунке 2.1 и в таблице 2.1.

Функция	Эксперт 1	Эксперт 2	Эксперт 3	Средний балл по 100 бальной шкале	Коэффициент относительной важности
Стоимость	75	90	85	83	13,7
Простота сопровождения	80	75	86	80	13,2
Временные затраты на разработку	90	85	95	90	14,8
Быстродействие	89	95	90	91	15
Удобный дизайн	85	81	90	85	14
Мощность пакета	75	92	84	84	13,8
Возможности языка	100	89	94	94	15,5
Сумма				**606**	**100,0%**

Рис.2.1 - Результаты экспертизы сред разработки, первый этап

Функция	Коэффициент относительной важности	Среда программирования	
		Delphi	C++ Builder
Стоимость	13,7	+	+
Простота сопровождения	13,2	+	-
Временные затраты на разработку	14,8	+	-
Быстродействие	15	+	+
Удобный дизайн	14	+	+
Мощность пакета	13,8	+	+
Возможности языка	15,5	+	+
Сумма	**100,0%**	**100**	**72**

Таблица 2.1 - Результаты экспертизы сред разработки, второй этап

Учитывая все вышесказанное и результаты анализа экспертным оцениванием можно сделать выбор среды программной разработки в пользу Delphi, который обеспечивает чрезвычайно высокую производительность и удобство использования. [36,37]

Для разработки будем использовать среду RAD Studio Professional.

RAD Studio Professional включает высокопроизводительные интегрированные среды разработки собственных приложений Windows и .NET. В интегрированную среду разработки Delphi и C++Builder с поддержкой Unicode для разработки собственных приложений входят сотни готовых компонентов и функций, к которым можно отнести рефакторинг, дополнение кода, выделение синтаксиса, интерактивные шаблоны, полнофункциональная отладка и тестирование модулей. Интегрированная среда разработки Delphi Prism поддерживает разработку приложений для платформы .NET, включая поддержку новейших технологий .NET. — Подключение к локальным базам данных InterBase®, Blackfish™ SQL и MySQL в Delphi и C++Builder. — Подключение к базам данных в Visual Studio с помощью ADO.NET, включая подключение к локальным базам данных InterBase и Blackfish в Delphi Prism. — Развертывание Blackfish SQL в системах с одним пользователем и размером базы данных 512 МБ. — Библиотека VCL для Интернета с ограничением числа подключений (не более пяти).

2.2 Описание основных алгоритмов

2.2.1 Алгоритм оптимизации центральной силы

Алгоритм CFO представляет собой детерминированную модель зондов, движущихся в области поиска под действием силы тяжести. Исходное распределение зондов в области поиска является детерминированным и равным.

Считаем, что каждый из зондов s_i имеет переменную массу, равную значению функции в точке текущего положения зонда. Координаты зондов изменяем в процессе итераций по формуле

$$x'_{i,j} = x_{i,j} + \frac{1}{2} a_{i,j}$$

где величину $a_{i,j}$ интерпретируем как текущее ускорение зонда s_i по j-координате, равное

$$a_{i,j} = \gamma \sum_k \chi\left(sign(\varphi_k - \varphi_i)(\varphi_k - \varphi_i)^b \frac{x_{k,j} - x_{i,j}}{\|X_k - X_i\|^c} \right)$$

$$k, i = 1,2,\ldots,|S|, k \neq i, j \in [:|X|], \varphi_l = \varphi(X_l)$$

Здесь $\chi(\bullet)$ - функция Хэвисайда, используемая в целях исключения возможности получения отрицательной массы; b, c, γ - заданные константы, из которых γ отождествляется с гравитационной постоянной.

Если некоторый зонд «вылетает» из области решений, то его следует возвратить в середину между его предыдущим и текущем (недопустимом) положении.

2.2.2 Гармонический поиск

В 2001 году Geem разработал и предложил свой алгоритм гармонического поиска (Harmony Search, или HS). Некоторые авторы заявляют, что навеян данный алгоритм был игрой джаз-музыкантов, другие говорят, что в основе лежит просто процесс создания приятной мелодии.

Важно лишь то, что опытный музыкант может достаточно быстро как сыграться с другими музыкантами, так и сымпровизировать что-нибудь прекрасное. Это и легло в основу алгоритма.

Классический HS оперирует понятием гармонической памяти (по аналогии с родительским пулом генетических алгоритмов). Здесь хранятся вектора, представляющие приближенные решения задачи оптимизации. Изначально они генерируются случайным образом в области, которая исследуется на наличие максимума или минимума (в зависимости от того, что вам нужно). Размер этой памяти, естественно, ограничен. Далее начинается магия импровизация (итеративная часть алгоритма).

Сама импровизация заключается в следующем:

1. генерируется пробный вектор (либо абсолютно случайно, либо с использованием векторов из памяти),
2. пробный вектор модифицируется (происходит небольшой сдвиг в компонентах, если пробный вектор был создан с помощью памяти),
3. модифицированный пробный вектор заменяет худший из имеющихся в памяти.

Итак, имеем некоторую функцию $f\ x : \mathbb{R}^n \to \mathbb{R}$, которую необходимо минимизировать, и область поиска $\mathbb{S} = s_n^0 : s_1^1 \times \ldots \times s_n^0 : s_n^1 \subset \mathbb{R}^n$ (для простоты предполагаем ее многомерным прямоугольным параллелепипедом), в которой точка минимума, а точнее ее приближение, и будет искаться. Следующим шагом генерируем в нашей области поиска начальные гармоники $h_i \in \mathbb{S}\ _{i=1}^{HMS}$ (случайным образом). Величина HMS задается пользователем, и, как несложно догадаться, указывает на количество гармоник, которые могут храниться в памяти. Кроме этого, пользователь еще должен задать HMCR— вероятность выбора из гармоник в памяти, PAR — вероятность модификации и bw — величину той самой модификации. Теперь, когда все приготовления сделаны, с чистой совестью можем приступать минимизировать функцию. Итеративная часть продолжается до выполнения критерия остановки алгоритма. Им может быть либо

ограничение по количеству итераций, либо количество прогонов без обновления памяти гармоник, либо близость добавляемой гармоники в смысле той или иной метрики. Все зависит от вашего желания. В ходе итеративной части происходит следующее:

1. Создаем нулевую новую гармонику $h_{new} = 0,0,\dots,0 \in \mathbb{R}^n$.
2. Далее для каждой компоненты гармоники выполняем следующее: генерируем случайное число $\mathcal{E}$, равномерно распределенное от нуля до единицы. Если $\mathcal{E}$ меньше, чем HMCR, то в текущую компоненту записываем соответствующую компоненту из наугад выбранной гармоники из памяти. В противном случае компонента генерируется случайно, с учетом принадлежности соответствующей компоненте области поиска $\mathbb{S}$. Если компонента была сгенерирована с использованием памяти, то, возможно, она подлежит модификации. Для этого опять генерируется случайное число $\mathcal{E}$, равномерно распределенное от нуля до единицы. Если $\mathcal{E}$ меньше, чем PAR, то компонента меняется на величину $\xi \cdot bw$, где ξ — случайная величина, равномерно распределенная от -1 до 1.
3. Если $f(h_{new}) < \max_{i=1\dots HMS} f(h_i)$, то h_{new} заменяет $\arg\max_{i=1\dots HMS} f(h_i)$(происходит обновление памяти). [48-51]

Плюсы:

- простота (как реализации, так и понимания). Действительно, алгоритм не нагроможден операциями (по сути лишь генерация случайных чисел и модификация вектора);
- достаточно малое количество настраиваемых параметров и наличие рекомендаций по выбору параметров;
- является методом нулевого порядка (отпадает необходимость численного дифференцирования, а, следовательно, все сопутствующие этому процессу сложности);
- метод легко и удобно встраивается в другие методы (например, в качестве дополнительного контура генетического алгоритма).

Минусы:

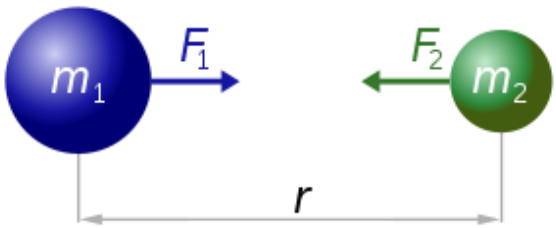
- низкая скорость сходимости;
- на мой взгляд метод плохо применим для решения задач больших размерностей (как и метод симуляции отжига) из-за простоты составляющих метод операций (в этом случае те же генетические алгоритмы работают на порядок быстрее).

2.2.3 Гравитационный поиск

Гравитационный поиск (GS) является очень молодым алгоритмом. Появился он в 2009 году и являлся логическим развитием метода центральной силы. Основу GS составляют законы гравитации и взаимодействия масс.

GS оперирует двумя законами:

- тяготения: каждая частица притягивает другие и сила притяжения между двумя частицами прямо пропорциональна произведению их масс и обратно пропорциональна расстоянию между ними.



$$F_1 = F_2 = G\frac{m_1 \times m_2}{r^2}$$

- движения: текущая скорость любой частицы равна сумме части скорости в предыдущий момент времени и изменению скорости, которое равно силе, с которой воздействует система на частицу, деленной на инерциальную массу частицы.

Имея в арсенале эти два закона, метод работает по следующему плану:

1. генерация системы случайным образом;
2. определение приспособленности каждой частицы;
3. обновление значений гравитационной постоянной, лучшей и худшей частиц, а так же масс;
4. подсчет результирующей силы в различных направлениях;
5. подсчет ускорений и скоростей;
6. обновление позиций частиц;
7. повторений шагов 2 — 6 до выполнения критерия окончания (либо превышение максимального количества итераций, либо слишком малой изменение позиций, либо любой другой осмысленный критерий).

Итак, у нас есть некоторая функция, которую необходимо минимизировать: $f\ x : \mathbb{R}^n \to \mathbb{R}$. Кроме этого, есть область $\mathbb{X}$, в которой генерируются начальные позиции частиц. В соответствии с планом работы GS, начинается все с генерации системы частиц $\mathbb{S} = \ p_i = \ p_i^1 \cdot p_i^2, \dots, p_i^n \ \in \mathbb{X}_{i=1}^{N}$, где N — максимальное количество частиц в системе. Сила, действующая в момент времени t на i-ю частицу со стороны j-й, рассчитывается по формуле $F_{ij}\ t\ = G(t)\frac{M_{pi}(t)\times M_{aj}(t)}{\|p_i \cdot p_j\|+\mathcal{E}}\ p_j\ t\ - p_i(t)$, где M_{aj} — активная гравитационная масса j-й частицы, M_{pj} — пассивная гравитационная масса i-й частицы, $G(t)$ — гравитационная постоянная в соответствующий момент времени, $\mathcal{E}$ — малая константа, $\| \cdots \|$ — евклидово расстояние между частицами.

Чтобы алгоритм был не детерминированным, а стохастическим, в формулу расчета результирующей силы $F_i\ t$ добавляются случайные величины ξ_j (равномерно распределенные от нуля до единицы). Тогда результирующая сила равна $F_i\ t\ = \ \sum_{j=1, j\neq i}^{N} \xi_j \cdot F_{ij}(t)$.

Посчитаем ускорения и скорости:

$$v_i\ t+1\ = \ \xi_1, \dots, \xi_n\ ^T * v_i\ t\ + \underbrace{\frac{F_i(t)}{M_{ii}(t)}}_{a_i(t)},$$

где $*$ — операция покомпонентного умножения векторов, ξ_i — случайная величина, равномерно распределенная от нуля до единицы, $M_{ii}(t)$— инертная масса i-й частицы.

Необходимо пересчитать положение частиц:

$$p_i\ t+1\ = p_i\ t\ + v_i\ t+1\ .$$

К текущему моменту осталось два вопроса: как изменяется гравитационная постоянная и как рассчитывать массы частиц. Значение гравитационной постоянной должно определяться монотонно убывающей функцией, зависящей от начального значений постоянной G_0 и момента времени t, т.е. $G\ t\ = G\ G_0, t : \mathbb{R}^+ \rightarrow \mathbb{R}^+$. [49]

Теперь можно приступить к заключительной части повествования: к пересчету масс. В простейшем случае все три массы (пассивная, активная и инерциальная) приравниваются: $M_{ai} = M_{pi} = M_{ii} = M_i$. Тогда значение масс можно пересчитать по формуле: $M_i\ t\ = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)}$, где $m_i\ t\ = \frac{f\ p_i - \max_{j\in\ 1\ldots N}\ f(p_j)}{\min_{j\in\ 1\ldots N}\ f\ p_j\ - \max_{j\in\ 1\ldots N}\ f(p_j)}$.

Плюсы

- как и в случае с гармоническим поиском простота реализации,
- на практике метод точнее, чем генетические алгоритмы с вещественным кодированием и классический PSO,
- большая скорость сходимости, чем у генетических алгоритмов с вещественным кодированием и классического PSO.

Минусы

- не самая большая скорость за счет необходимости пересчета многих параметров,
- большая часть достоинств теряется при оптимизации мультимодальных функций (особенно больших размерностей), так как метод начинает быстро сходиться к некоторому локальному оптимуму, из которого

сложно выбраться, так как не предусмотрены процедуры, похожие на мутации в генетических алгоритмах.

2.3 Разработка структуры программы

Приложение было разработано с использованием принципов объектно-ориентированного программирования, что позволило сэкономить время и избежать ряда проблем, связанных с повторным написанием однообразного программного кода и отладкой уже готового приложения.

При проектировании приложения был разработан класс «TCustomSearchEngine», который реализует в себе базовые особенности поисковика экстремумов. В этом классе были реализованы следующие функции:

- Счетчик количества итераций;
- Методы накопления и сохранения истории вычислений;
- Методы управления историей вычислений;
- Методы определения входных параметров алгоритмов оптимизации (целевая функция, начальная точка вычислений, ограничение количества итераций);
- Методы, предоставляющие возможность реализации разнообразных алгоритмов оптимизации.

Данный класс содержит метод «Search», который и запускает процесс поиска экстремума. Но тело алгоритма (для наследников данного класса) должно быть реализовано внутри метода «InternalSearch». Внутри метода «Search» выполняется ряд подготовительных процедур (очистка истории вычислений, проверка необходимых условий), после которых выполняется непосредственный запуск алгоритма (метод «InternalSearch»).

Как было сказано ранее, алгоритм оптимизации должен принимать на вход определенную функцию. Для реализации функции был предусмотрен и

реализован класс «TCustomFunction», который позволяет задавать функцию от произвольного количества переменных. Т.е. если нам необходимо использовать в проект произвольную функцию, то необходимо добавить к проекту новый класс, наследующий своё поведение от класса «TCustomFunction» и переопределить его функцию «InternalCompute». Вычисление значения функции выполняется через вызов метода «Compute», который принимает в качестве входной/выходной переменной объект класса «TFunctionVariable».

Класс «TFunctionVariable» является основой, на базе которой выполняются все вычисления, накопление истории вычислений и последующая визуализация результатов вычислений, т.к. внутри данного класса хранятся не только значения аргументов, но и значение функции, а также множество других сведений относительно вычислений.

Класс «TFunctionVariable» включает поля, которые разделяются на входные и выходные. К входным полям относятся:

- Набор аргументов $x_1, x_2, ..., x_n$ (используется при вычислении значения функции);
- Ограничение область значений по каждому из аргументов (используется алгоритмами поиска);

К выходным полям относятся:

- Значение функции в данной точке (y , вычисляется при вызове метода «Compute», наследниками класса «TCustomFunction»);
- Переменная, указывающая на то, что значение функции было вычислено (вычисляется при вызове метода «Compute», наследниками класса «TCustomFunction»);
- Идентификатор точки (опциональный параметр, может быть использован программистом в собственных целях);

Также, класс «TFunctionVariable» включает в себя ряд вспомогательных методов:

- Методы копирования переменной («GetClone», «Copy»);
- Метод получения текстового описания точки («ToString»);
- Метод очистки всех полей («Clear»);
- Метод расчета Евклидова расстояния между аргументами («GetEuclideanDistanceBetweenX»).

Вышеописанные класс («TFunctionVariable», «TCustomFunction», «TCustomSearchEngine») являются базисом для последующей реализации конкретных методов оптимизации. Хочу отметить, что данные классы описывают универсальный подход при реализации алгоритмов оптимизации и за счет этого трио возможна реализация произвольных алгоритмов.

В проекте реализовано три метода оптимизации:

- Алгоритм оптимизации центральной силы («CFO»);
- Гармонический поиск («HS»);
- Гравитационный поиск («GS») .

При изучении материалов по реализации данных алгоритмов была выявлена их общая черта, которая позволяет их выделить в отдельный класс – алгоритмы, основанные на использовании набора частиц (аналогично: зондов – для CFO и GS, гармоник – для HS). Т.е. при вычислениях используется набор точек, который определяет процесс расчета экстремума. Для данных алгоритмов было принято создать отдельный класс «TParticleBasedSearchEngine», наследующий поведение «TCustomSearchEngine».

Класс «TParticleBasedSearchEngine» включает в себя следующие функции:

- Функцию инициализации списка частиц;
- Функции управления списком частиц;
- Функции навигации по списку частиц.

Непосредственная реализация методов выполнена в следующих классах:

- «TCentralForceOptimizationEngine» - Алгоритм оптимизации центральной силы («CFO»);
- «THarmonySearchEngine» - Гармонический поиск («HS»);
- «TGravitationalSearchEngine» - Гравитационный поиск («GS») .

Данные алгоритмы могут быть применимы для функций множества переменных, хотя при тестировании, для простоты и наглядности визуализации были использованы функции лишь одной переменной. [46]

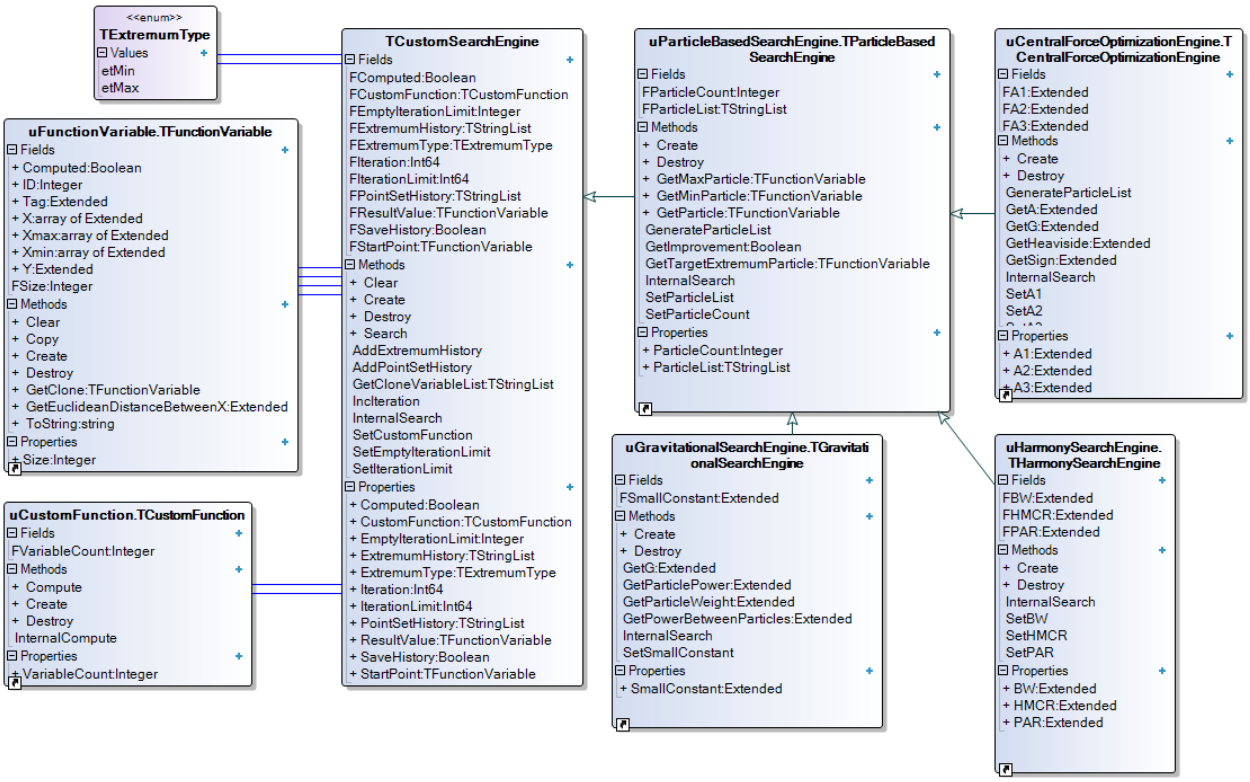


Рис. 2.1 - Диаграмма классов

Описания классов приведены в таблицах ниже.

2.3.1 Класс «TFunctionVariable»

Field Summary	
public Boolean	Computed Флаг успешных вычислений
internal Integer	FSize Размерность массива переменных
public Integer	ID Идентификатор

public Extended	Tag Вспомогательная переменная (для нужд программиста)
public array of Extended	X Массив переменных функции
public array of Extended	Xmax Ограничение значений переменных сверху
public array of Extended	Xmin Ограничение значений переменных снизу
public Extended	Y Вычисленный результат

Property Summary	
public Integer	Size Размерность массива переменных

Constructor Summary	
Create(Size: Integer) Конструктор класса	

Method Summary	
public Sub	Clear() Метод очистки
public Sub	Copy(Source: TFunctionVariable) Копироватьданныеиз Source
public Sub	Destroy() Деструктор класса
public function TFunctionVariable	GetClone() Метод клонирования экземпляра класса
public function Extended	GetEuclideanDistanceBetweenX(P: TFunctionVariable) Функция расчета Евклидова расстояния
public function string	ToString() Перевод в текст

2.3.2 Класс «TCustomFunction»

Field Summary	
protected internal **Integer**	**FVariableCount** Количество переменных

Property Summary	
public **Integer**	**VariableCount** Количество переменных

Constructor Summary

Create() Конструктор класса	

Method Summary	
public **Sub**	**Compute**(V: uFunctionVariable.TFunctionVariable) Тело пользовательской функции
public **Sub**	**Destroy**() Деструктор класса
protected internal **Sub**	**InternalCompute**(V: uFunctionVariable.TFunctionVariable) Телопользовательскойфункции

2.3.3 Класс «TCustomFunction»

Field Summary	
protected internal Boolean	FComputed Флаг успешных вычислений
internal uCustomFunction.TCustomFunction	FCustomFunction Целевая функция
internal Integer	FEmptyIterationLimit Допустимое количество пустых итераций
internal TStringList	FExtremumHistory История вычислений (движение к экстремуму)
internal TExtremumType	FExtremumType Тип поиска
internal Int64	FIteration Номер итерации
internal Int64	FIterationLimit Предел для итераций (на случай расхождения метода оптимизации)
internal TStringList	FPointSetHistory История вычислений (списки)
internal uFunctionVariable.TFunctionVariable	FResultValue Результат вычислений (экстремум)
internal Boolean	FSaveHistory Флаг, который указывает, записывать ли историю вычислений
internal uFunctionVariable.TFunctionVariable	FStartPoint Стартовая точка и область ограничений точки

Property Summary	
public Boolean	Computed Флаг успешных вычислений
public uCustomFunction.TCustomFunction	CustomFunction Целевая функция
public Integer	EmptyIterationLimit

	Допустимое количество пустых итераций
public TStringList	ExtremumHistory История вычислений (движение к экстремуму)
public TExtremumType	ExtremumType Тип поиска
public Int64	Iteration Номер итерации
public Int64	IterationLimit Предел для итераций (на случай расхождения метода оптимизации)
public TStringList	PointSetHistory История вычислений (списки)
public uFunctionVariable.TFunctionVariable	ResultValue Результат вычислений (экстремум)
public Boolean	SaveHistory Флаг, который указывает, записывать ли историю вычислений
public uFunctionVariable.TFunctionVariable	StartPoint Стартовая точка и область ограничений точки

Constructor Summary	
Create() Конструктор класса	

Method Summary	
protected internal Sub	AddExtremumHistory(P: uFunctionVariable.TFunctionVariable) Метод записи истории движения к экстремуму
protected internal Sub	AddPointSetHistory(L: TStringList) Метод записи истории вычислений (списки)
public Sub	Clear() Функция очистки переменных, содержащих результаты вычислений
public Sub	Destroy() Деструктор класса
protected internal function TStringList	GetCloneVariableList(Source: TStringList) Копированиелистаточек
protected internal Sub	IncIteration() Инкремент для итератора
protected internal Sub	InternalSearch() Функция поиска экстремума
public Sub	Search() Функция поиска экстремума
internal Sub	SetCustomFunction(Value: uCustomFunction.TCustomFunction) Целеваяфункция
internal Sub	SetEmptyIterationLimit(Value: Integer) Указать допустимое количество пустых итераций
internal Sub	SetIterationLimit(Value: Int64)

	Предел для итераций (на случай расхождения метода оптимизации)

2.3.4 Класс «TParticleBasedSearchEngine»

Field Summary	
internal Integer	FParticleCount Количество частиц системы
internal TStringList	FParticleList Набор частиц

Property Summary	
public Integer	ParticleCount Количество частиц системы
public TStringList	ParticleList Набор частиц

Constructor Summary	
Create() Конструктор класса	

Method Summary	
public Sub	Destroy() Деструктор класса
protected internal Sub	GenerateParticleList() Генерация начальных значений списка частиц
protected internal function Boolean	GetImprovement(NewExtremum: Extended; OldExtremum: Extended) Метод сравнения результатов (в зависимости от того, максимум или минимум мы ищем)
public function uFunctionVariable.TFunctionVariable	GetMaxParticle() Найти максимальную частицу
public function uFunctionVariable.TFunctionVariable	GetMinParticle() Найти минимальную частицу
public function uFunctionVariable.TFunctionVariable	GetParticle(Index: Integer) Выбрать точку по индексу
protected internal function uFunctionVariable.TFunctionVariable	GetTargetExtremumParticle() Метод определения целевой точки (в зависимости от того, максимум или минимум мы ищем)
protected internal Sub	InternalSearch() Функция поиска экстремума
internal Sub	SetParticleCount(Value: Integer) Задать количество частиц системы
protected internal Sub	SetParticleList(L: TStringList)

	Установить список частиц

2.3.5 Класс «TCentralForceOptimizationEngine»

Field Summary	
protected internal Extended	FA1 Коэффициент A1
protected internal Extended	FA2 Коэффициент A2
protected internal Extended	FA3 Коэффициент A3

Property Summary	
public Extended	A1 Задать коэффициент A1
public Extended	A2 Задать коэффициент A2
public Extended	A3 Задать коэффициент A3

Constructor Summary	
Create() Конструктор класса	

Method Summary	
public Sub	Destroy() Деструктор класса
protected internal Sub	GenerateParticleList() Генерация списка частиц
protected internal function Extended	GetA(Index: Integer; Dimension: Integer) Текущееускоренийзонда
protected internal function Extended	GetG() Функция вычисления гравитационной постоянной
protected internal function Extended	GetHeaviside(X: Extended) ФункцияХевисайда
protected internal function Extended	GetSign(X: Extended) Кусочно-постоянная функция Sign
protected internal Sub	InternalSearch() Функция поиска экстремума
protected internal Sub	SetA1(Value: Extended) Задать коэффициент A1
protected	SetA2(Value: Extended)

internal Sub	Задать коэффициент A2
protected internal Sub	SetA3(Value: Extended) Задать коэффициент A3

2.3.6 Класс «THarmonySearchEngine»

Field Summary	
protected internal **Extended**	**FBW** Величина модификации
protected internal **Extended**	**FHMCR** Вероятность выбора из гармоник в памяти
protected internal **Extended**	**FPAR** Вероятность модификации

Property Summary	
public **Extended**	**BW** Величина модификации
public **Extended**	**HMCR** Вероятность выбора из гармоник в памяти
public **Extended**	**PAR** Вероятность модификации

Constructor Summary	
Create() Конструктор класса	

Method Summary	
public **Sub**	**Destroy**() Деструктор класса
protected internal **Sub**	**InternalSearch**() Функция поиска экстремума
protected internal **Sub**	**SetBW**(Value: Extended) Указать величина модификации
protected internal **Sub**	**SetHMCR**(Value: Extended) Указать вероятность выбора из гармоник в памяти
protected internal **Sub**	**SetPAR**(Value: Extended) Указать вероятность модификации

2.3.7 Класс «TGravitationalSearchEngine»

Field Summary	
protected internal **Extended**	**FSmallConstant** Малая константа

Property Summary	
public **Extended**	**SmallConstant** Установка малой константы

Constructor Summary	
Create() Конструктор класса	

Method Summary	
public **Sub**	**Destroy**() Деструктор класса
protected internal **function Extended**	**GetG**(X: Extended) Функция вычисления гравитационной постоянной
protected internal **function Extended**	**GetParticlePower**(Index: Integer; Dimension: Integer) Вычислитьсилуточки
protected internal **function Extended**	**GetParticleWeight**(Index: Integer) Вычислитьмассуточки
protected internal **function Extended**	**GetPowerBetweenParticles**(Index1: Integer; Index2: Integer; Dimension: Integer) Рассчетсилымеждуточками
protected internal **Sub**	**InternalSearch**() Функция поиска экстремума
protected internal **Sub**	**SetSmallConstant**(Value: Extended) Установка малой константы

3 Анализ полученного решения

3.1 Описание работы программы

Для запуска программы достаточно нажать на файл pSearch.exe, после чего появится главное окно программы. В главном окне можно выбрать минимизируемую функцию, метод минимизации и тип представления результата.

При этом используются три метода минимизации – гравитационный поиск, гармонический поиск, оптимизация центральной силы.

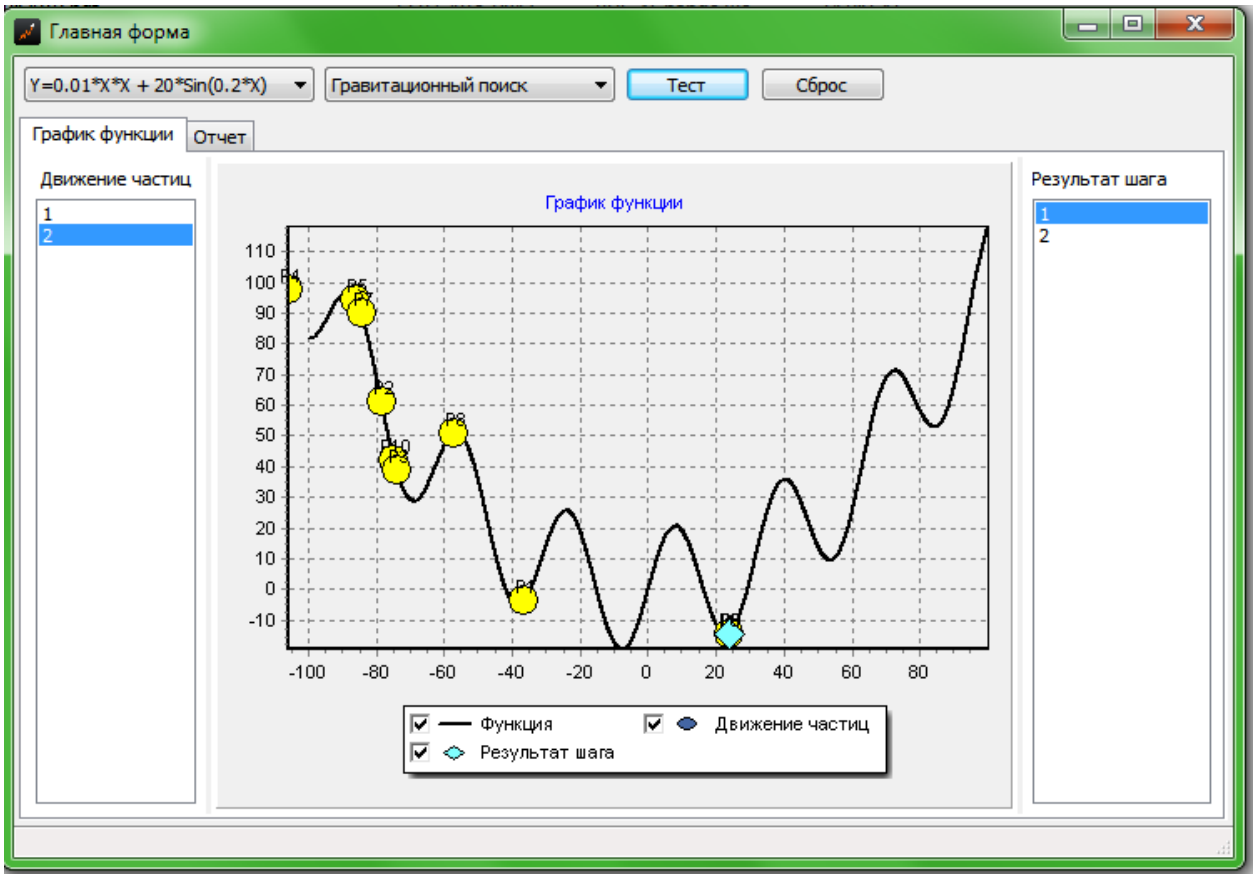


Рис. 3.1 - Главная форма программы

Рассмотрим порядок использования программы по использованию функций и методов оптимизации.

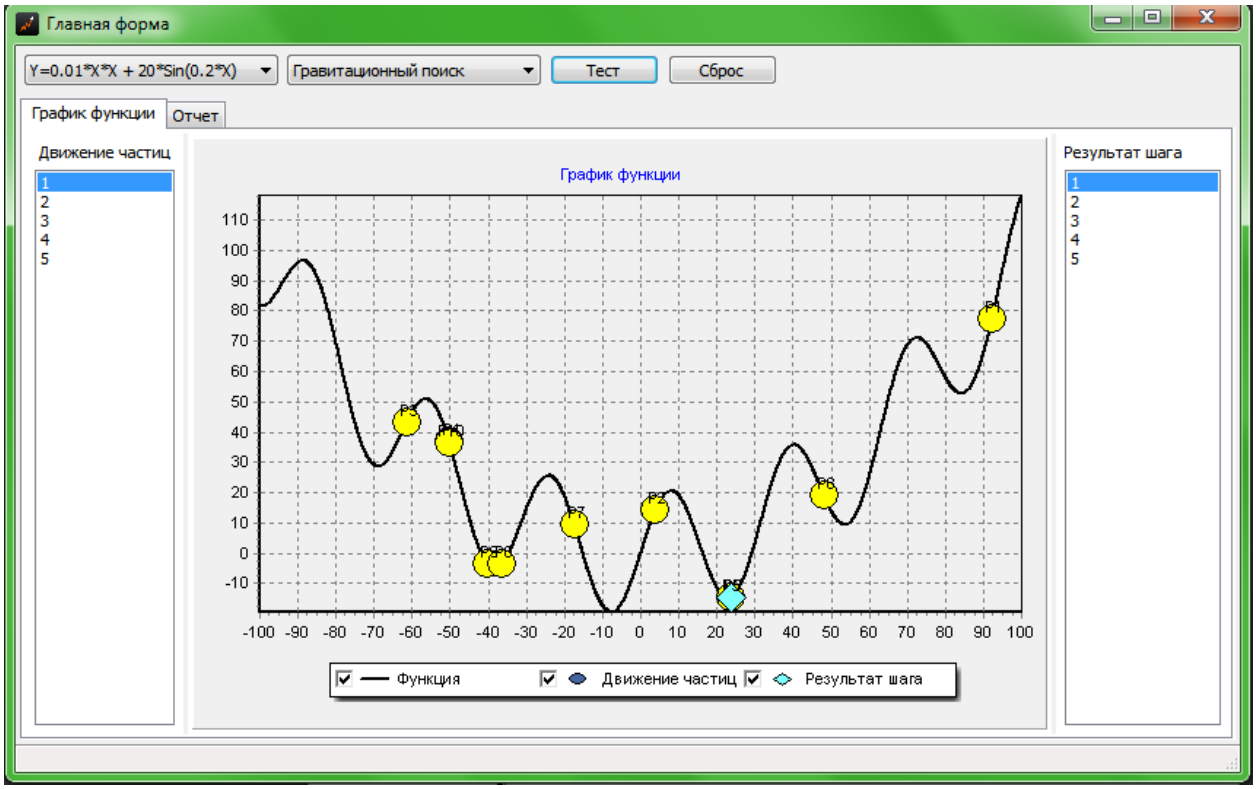


Рис. 3.2 - Функция № 1, гравитационный поиск, результаты тестирования.

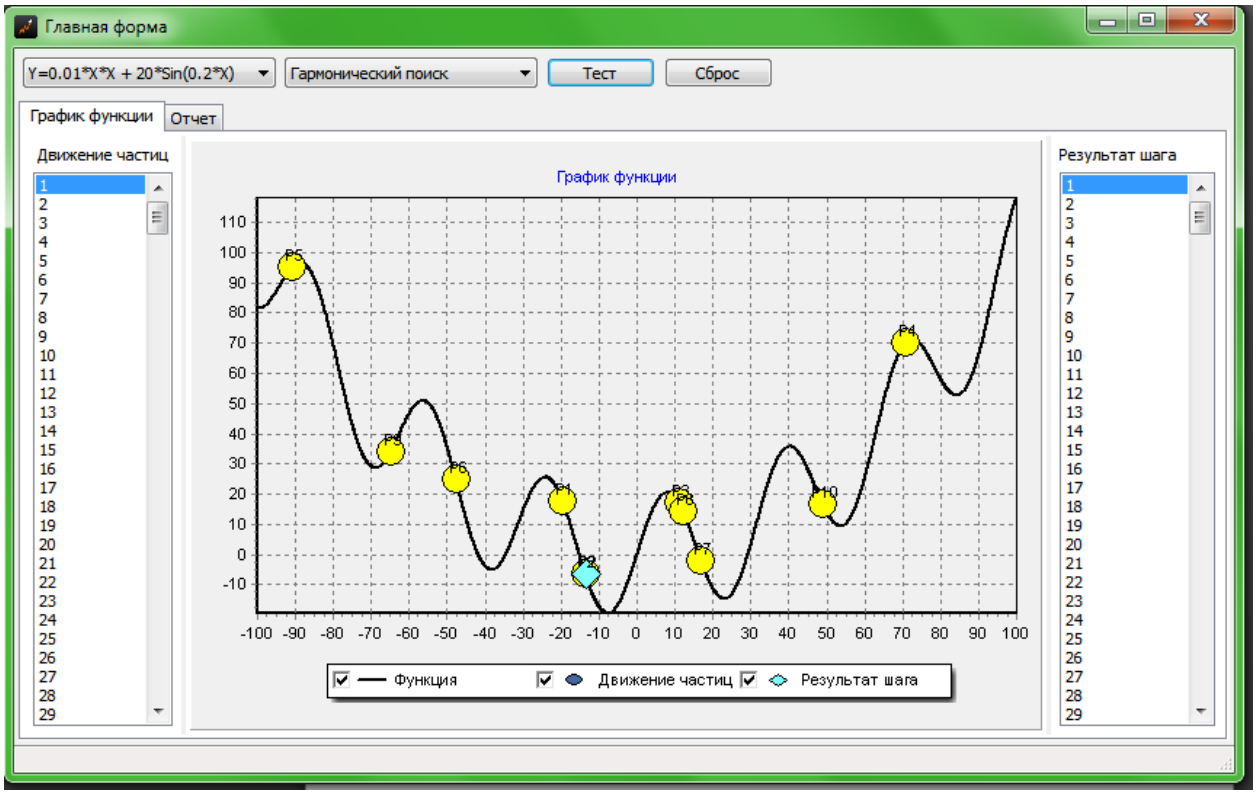


Рис. 3.3 - Функция № 1, гармонический поиск, результаты тестирования.

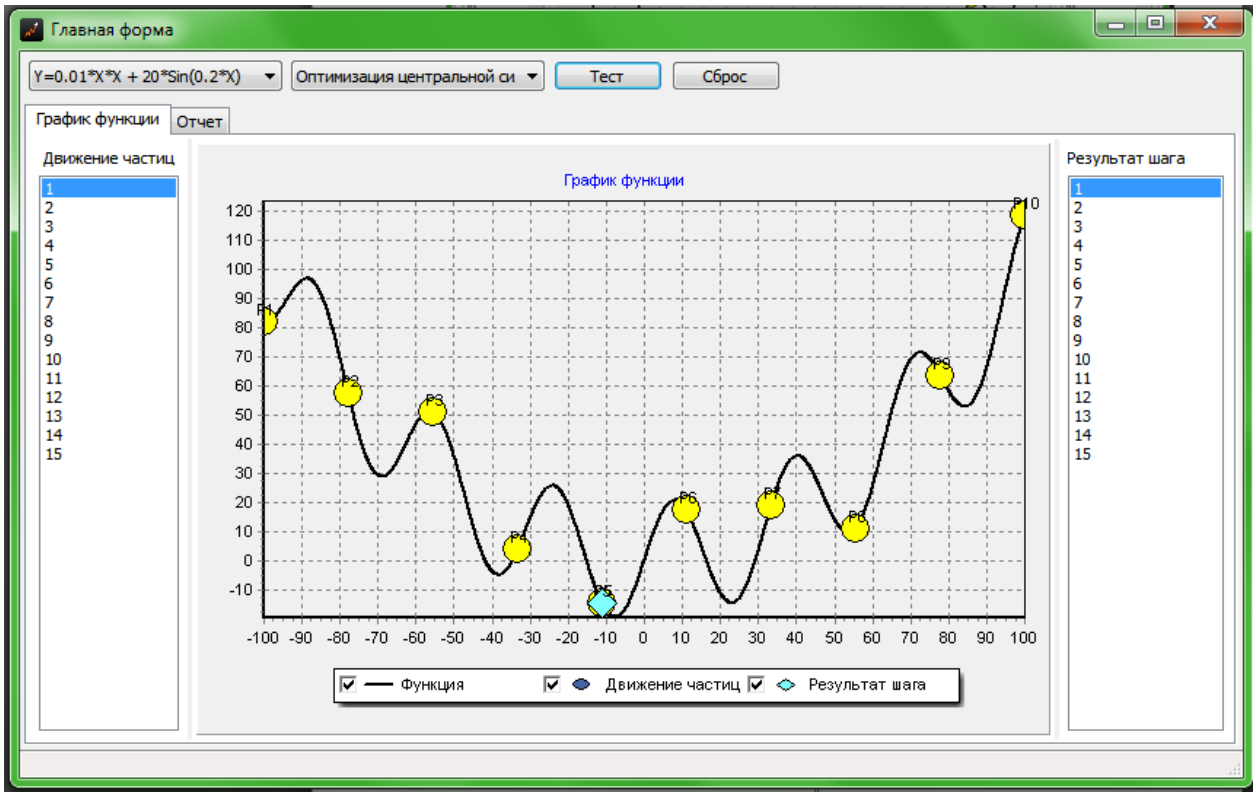


Рис. 3.4 - Функция № 1, оптимизация центральной силы, результаты тестирования.

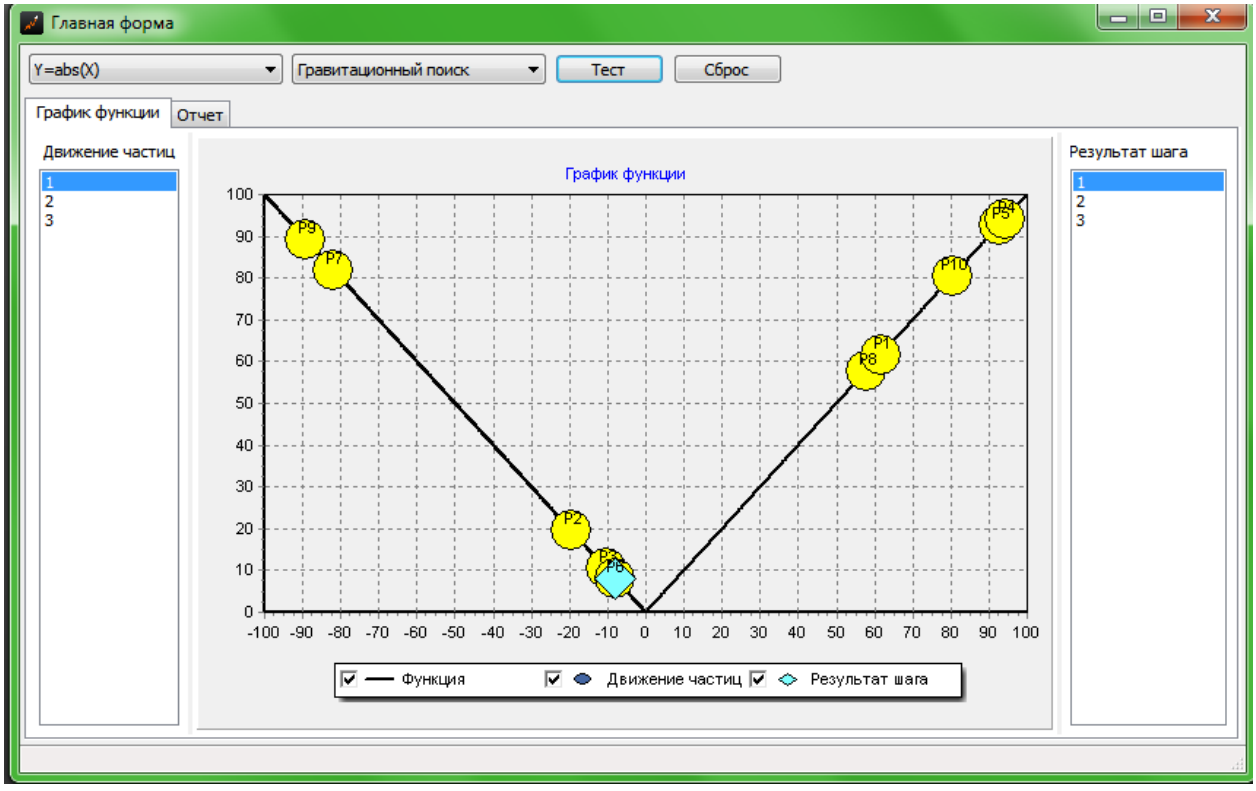


Рис. 3.5 - Функция № 2, гравитационный поиск, результаты тестирования.

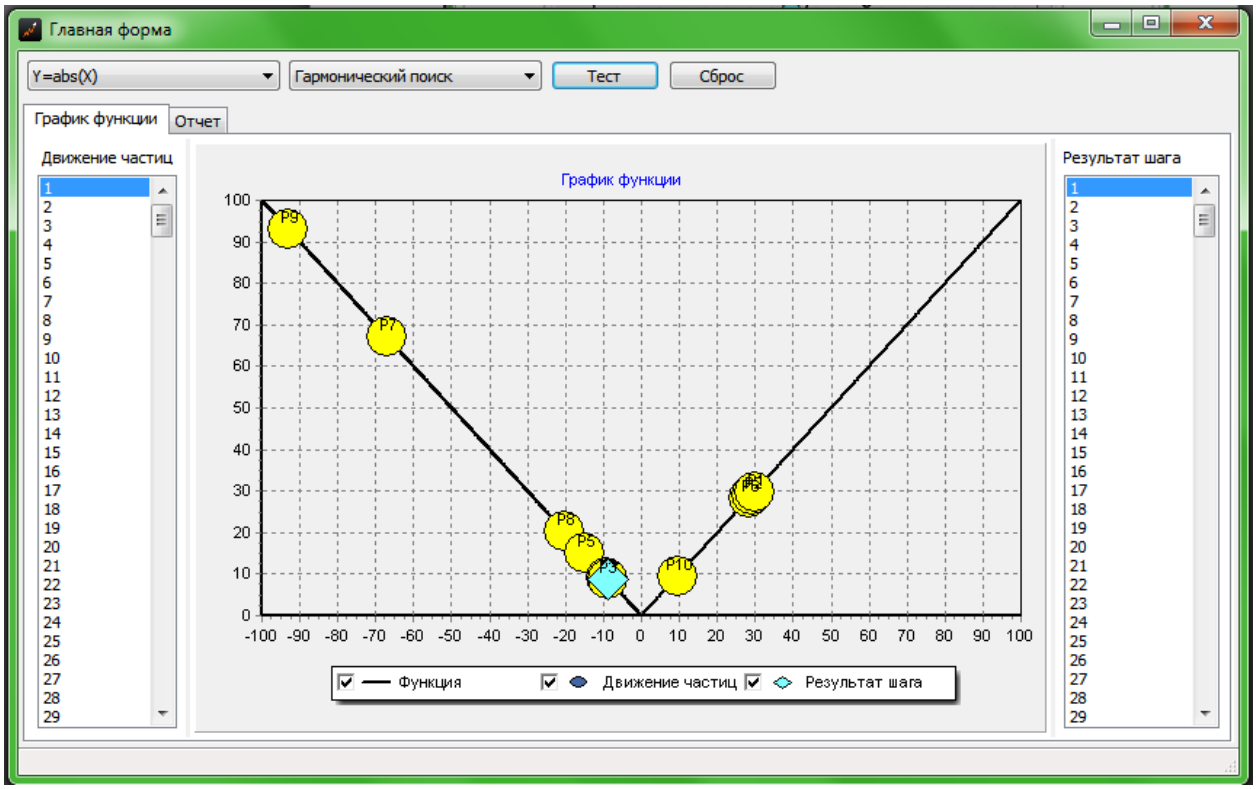


Рис. 3.6 - Функция № 2, гармонический поиск, результаты тестирования.

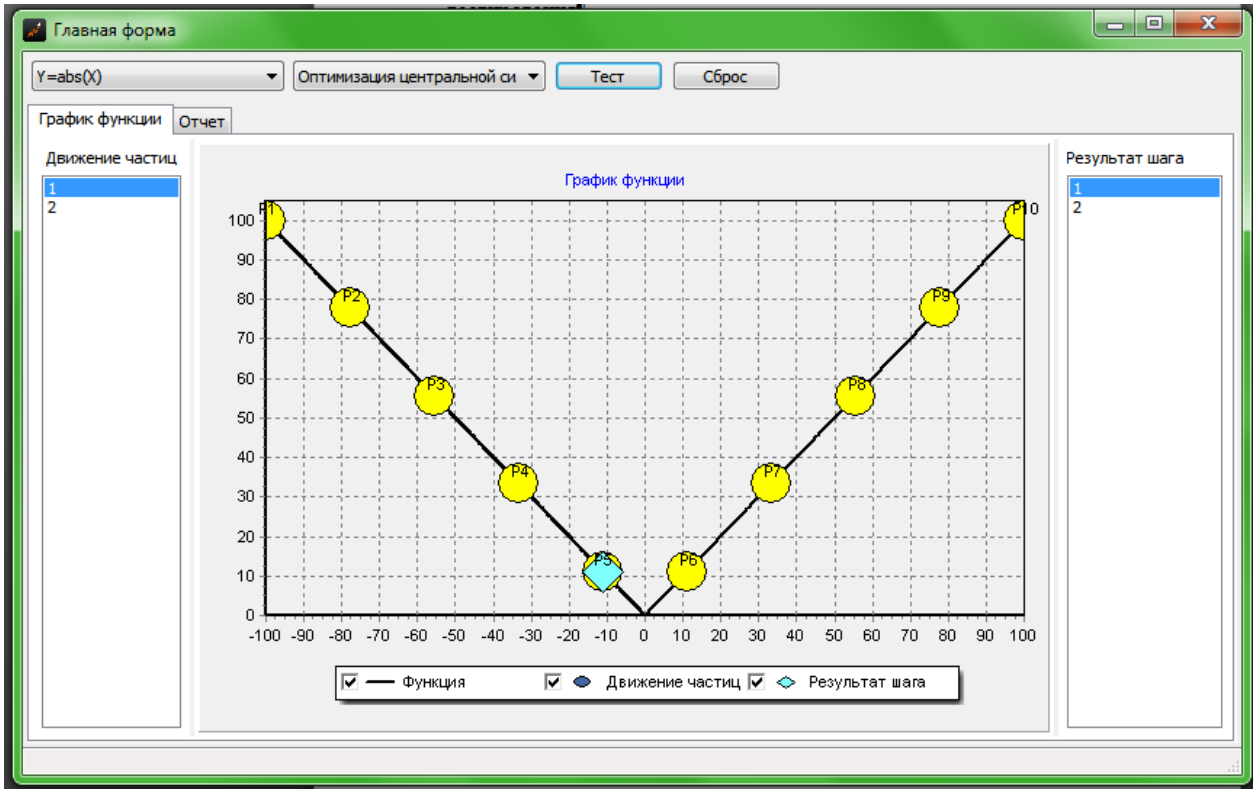


Рис. 3.7 - Функция № 2, оптимизация центральной силы, результаты тестирования

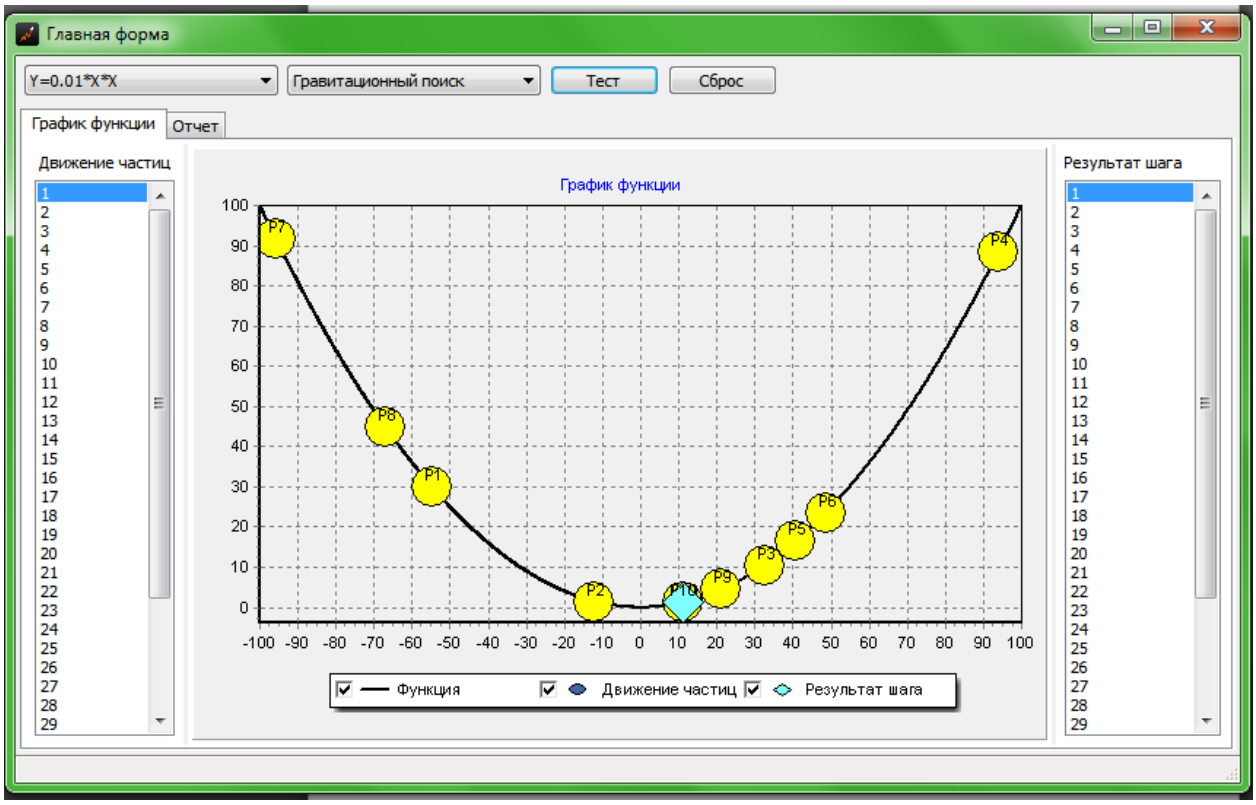


Рис. 3.8 - Функция № 3, гравитационный поиск, результаты тестирования

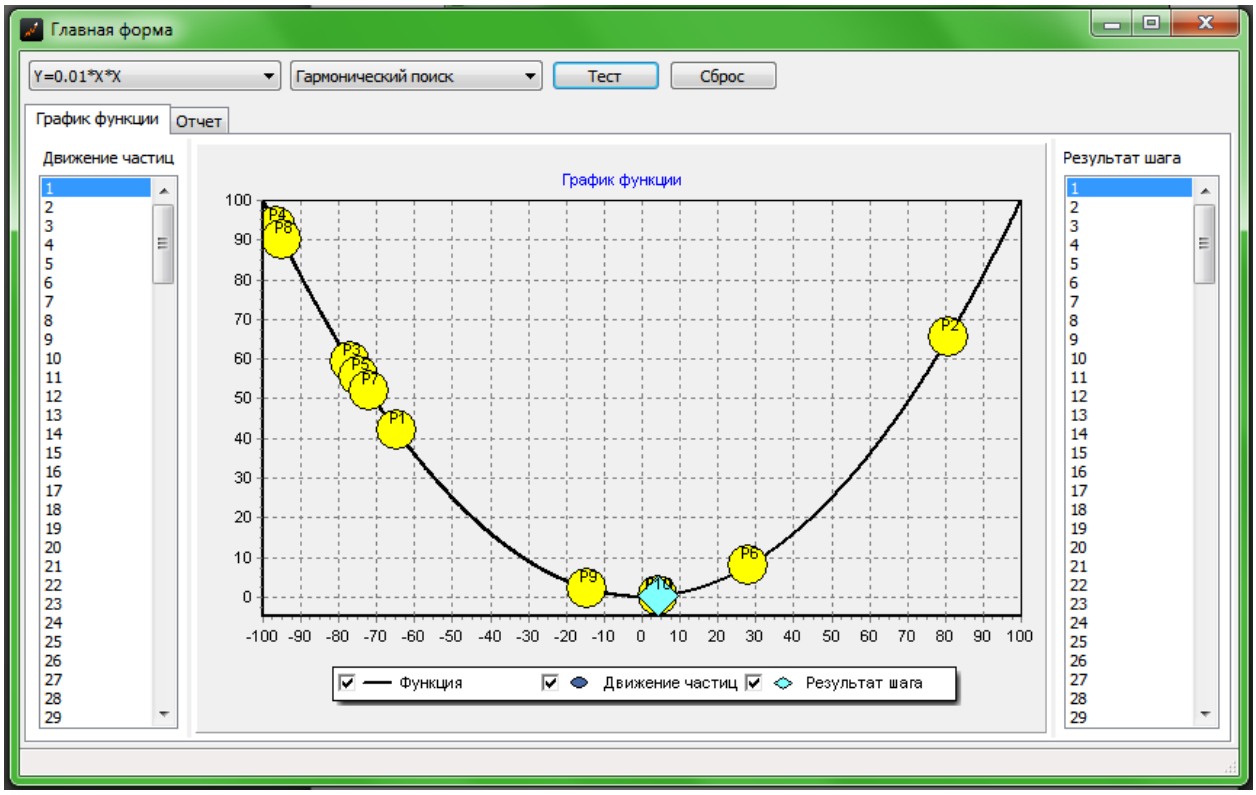


Рис. 3.9 - Функция № 3, гармонический поиск, результаты тестирования

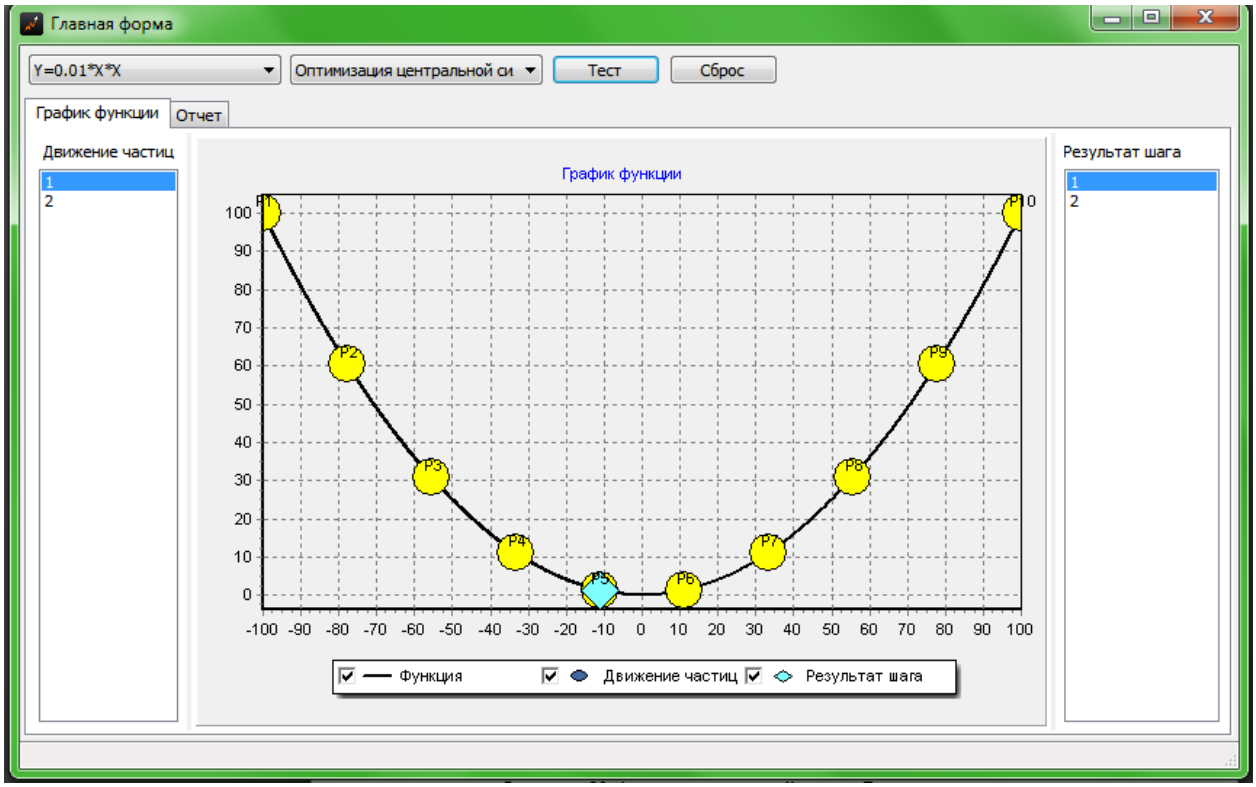


Рис. 3.10 - Функция № 3, оптимизация центральной силы, результаты тестирования

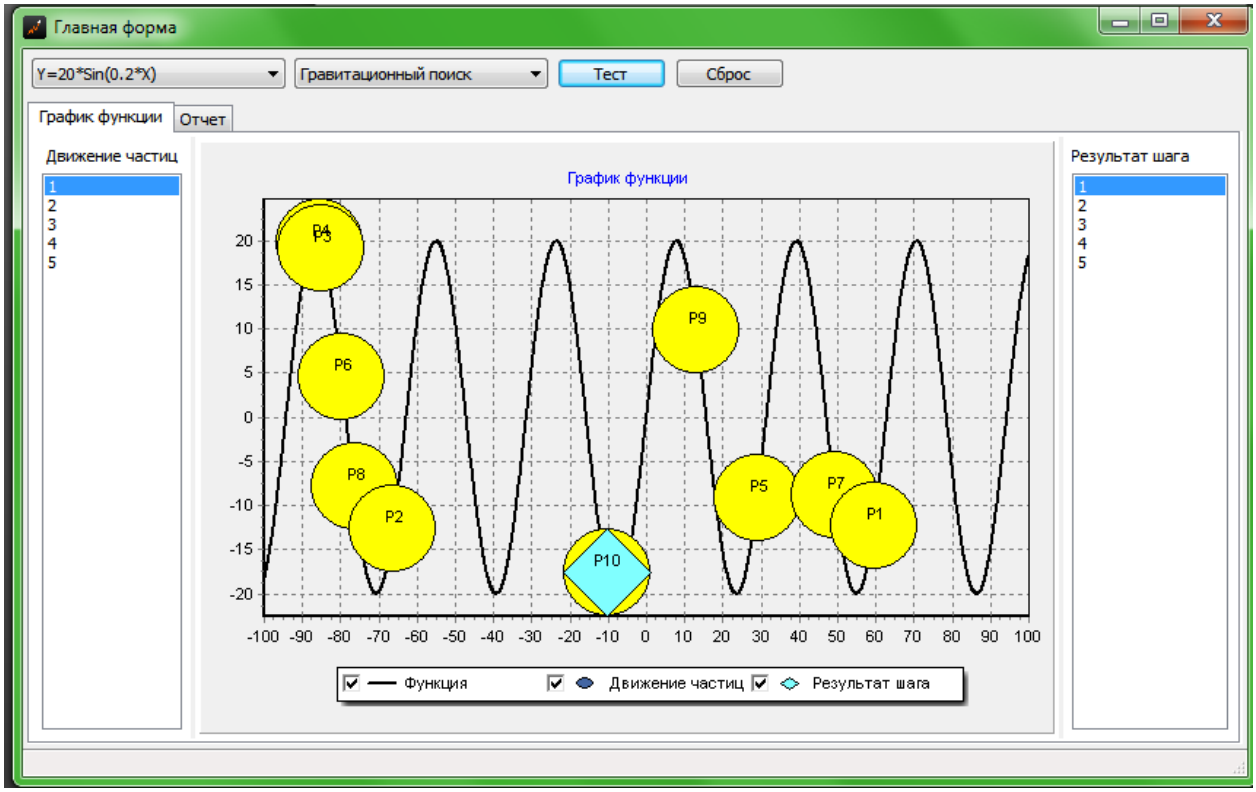


Рис. 3.11 - Функция № 4, гравитационный поиск, результаты тестирования

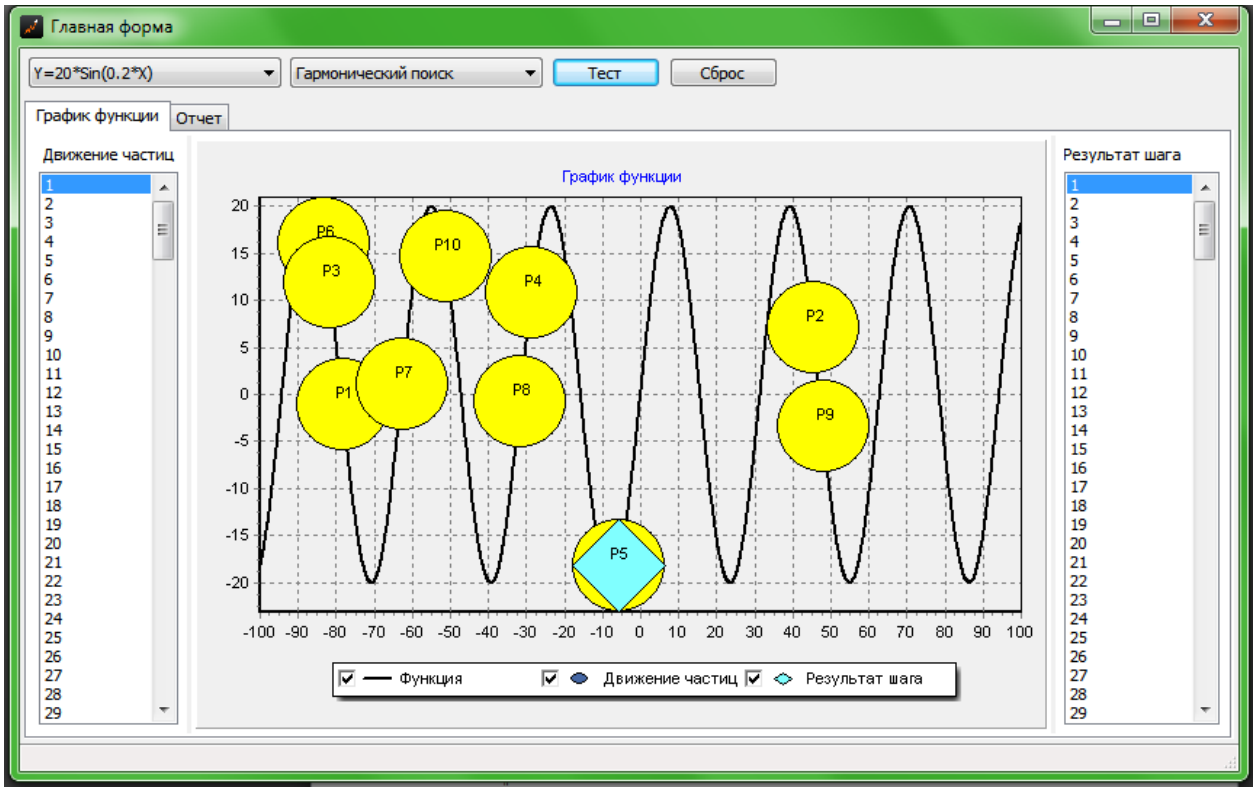


Рис. 3.12 - Функция № 4, гармонический поиск, результаты тестирования

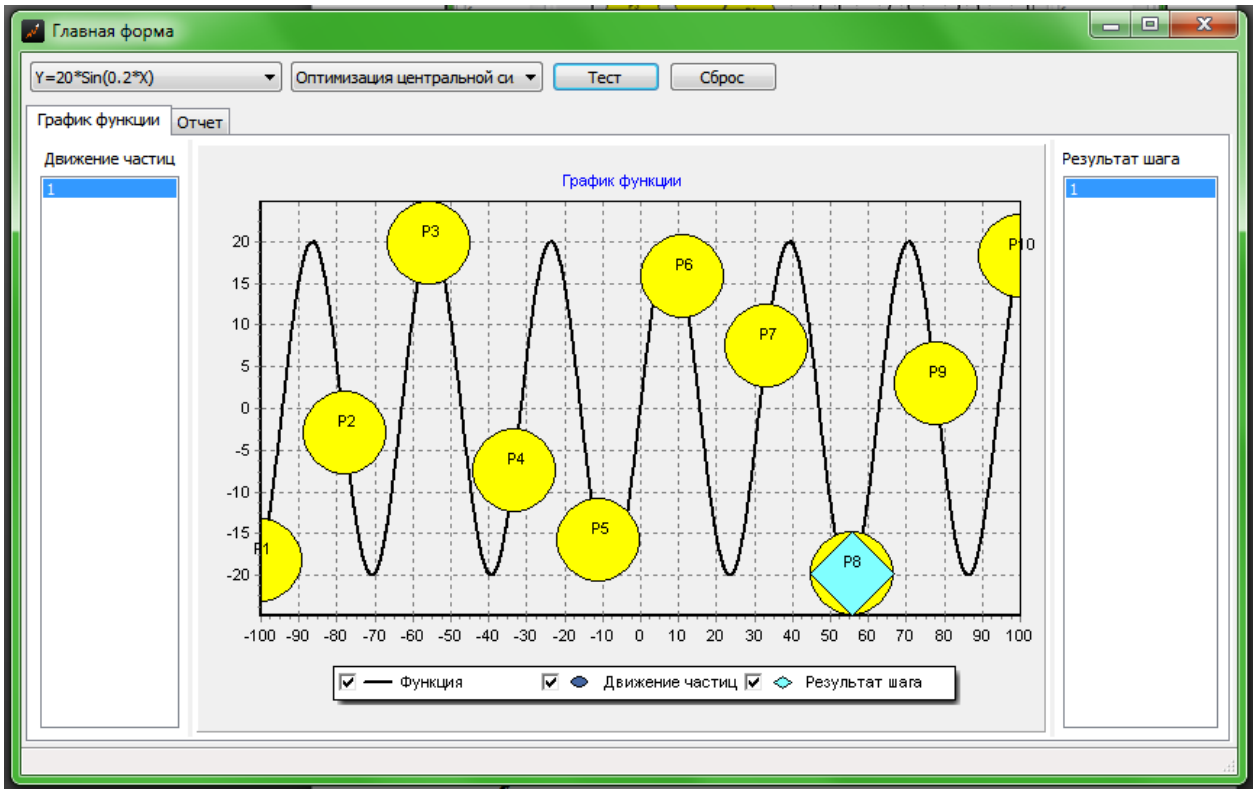


Рис. 3.13 - Функция № 4, оптимизация центральной силы, результаты тестирования

3.2 Сравнительный анализ рассмотренных методов

К плюсам гармонического поиска отнесем:

- простота (как реализации, так и понимания). Действительно, алгоритм не нагроможден операциями (по сути лишь генерация случайных чисел и модификация вектора);
- достаточно малое количество настраиваемых параметров и наличие рекомендаций по выбору параметров;
- является методом нулевого порядка (отпадает необходимость численного дифференцирования, а, следовательно, все сопутствующие этому процессу сложности);
- метод легко и удобно встраивается в другие методы (например, в математические алгоритмы, или в качестве дополнительного контура генетического алгоритма).

Минусы:

- низкая скорость сходимости;
- на мой взгляд метод плохо применим для решения задач больших размерностей (как и метод симуляции отжига) из-за простоты составляющих метод операций (в этом случае те же генетические алгоритмы работают на порядок быстрее).

К плюсам гравитационного поиска можно отнести:

- как и в случае с гармоническим поиском простота реализации,
- на практике метод точнее, чем генетические алгоритмы с вещественным кодированием и классический PSO,
- большая скорость сходимости, чем у генетических алгоритмов с вещественным кодированием и классического PSO.

Минусы:

- не самая большая скорость за счет необходимости пересчета многих параметров,

- большая часть достоинств теряется при оптимизации мультимодальных функций (особенно больших размерностей), так как метод начинает быстро сходиться к некоторому локальному оптимуму, из которого сложно выбраться, так как не предусмотрены процедуры, похожие на мутации в генетических алгоритмах.

Заключение

В дипломной работе выполнено теоретическое обобщение и получены новые решения научно-прикладных задач, которые состоят в развитии известных и разработке новых методов и алгоритмов анализа и синтеза цифровых устройств вычислительной техники и систем управления.

Предложен и исследован метод многоверсионной минимизации традиционных переключательных функций, основанный на сжатии области определения функции по комплементарным подмножествам с последующей минимизацией в каждой из полученных областей. В ходе исследования установлено, что при корректном выделении массива минтермов, которому соответствует простая импликанта минимально возможного ранга, номера точек, образующие такой массив в области определения, сжатой по одному из подмножеств переменных, совпадают с индексами минтермов, образующих массив в области определения, сжатой по комплементарному подмножеству переменных. Несовпадение результатов свидетельствует об ошибке, допущенной на этапе выделения правильных конфигураций, результатов их описания или процедуре минимизации.

Предложен и исследован метод анализа состязаний в комбинационных схемах, основанный на последовательном сжатии области определения функции по каждой из переменных. При этом установлено, что представление функций в форме ОЛФ в этом случае выступает как значительно большее, чем простое сокращение числа точек области определения, поскольку позволяет определить наличие или отсутствие состязаний по каждой из переменных непосредственно по характеру полученных значений функции в точках сжатой области определения.

При синтезе комбинационных схем (многоразрядных параллельных сравнивающих устройств), а также многофункциональных триггерных устройств представление функций в форме ОЛФ позволяет уменьшить число точек области определения, что упрощает процедуру проектирования за счёт

сокращения размера таблиц функционирования и, как следствие, сокращает время проектирования..

При проектировании конечных автоматов Мили, в частности, формирователей временных интервалов с перестраиваемыми параметрами выходных сигналов, основной проблемой является не собственно синтез автомата как такового, а нахождение оптимального по сложности представления функций выхода автомата. Представление функций в форме ОЛФ, обеспечило не только уменьшение числа точек области определения, что привело к упрощению процедуры проектирования, но и позволило оптимальным образом привязать режимы настройки формирователей к его состояниям, что обеспечило выбор оптимального варианта.

Проведенное компьютерное моделирование одного из вариантов программируемого интервального таймера с перестраиваемой длительностью тактов, подтвердило достоверность полученных результатов его синтеза.

При синтезе универсальных логических модулей с памятью, реализующего программируемый список функций от двух переменных, выполненного в виде триггера Эрла, представление функций в форме ОЛФ позволило выполнить оптимальное их размещение в точках области определения, которому соответствует минимальное число простых импликант, определяющих структуру УЛМ с памятью, исключив процедуру перебора.

Представление функций в форме ОЛФ дало возможность упростить инженерную методику нахождения частных, кратных и векторных булевых производных, используемых при моделировании неисправностей, декомпозиции булевых функций, задачи обнаружения статических и динамических ошибок в комбинационных схемах, и т.д., за счёт исключения аналитических представлений к связанных с ними преобразований.

Практическая ценность работы заключается в доведении полученных теоретических результатов до конкретных алгоритмов, методов, программ и

схем цифровых устройств, проиллюстрированных примерами. Основные практические результаты сводятся к следующему:

- представление традиционных функций одной переменной в форме ОЛФ позволяет упростить процедуру минимизации, а разработанные программные средства сжатия области определения дают возможность использования их в пакетах систем автоматизированного проектирования;

- предложенный метод многовариантной минимизации функций одной переменной позволяет обеспечить контроль достоверности проведенных преобразований и минимальность полученных результатов.

Список использованной литературы

1. Астелс, Дэвид; Миллер Гранвилл; Новак, Мирослав, Практическое руководство по экстремальному программированию, Пер. с англ. - М.: Издательский дом "Вильямс", 2008. - 320 с.: ил. - Парал. тит. англ

2. Баженова И. Ю. , Основы проектирования приложений баз данных, Издательства: Бином. Лаборатория знаний, Интернет-университет информационных технологий, 2008 г., , 328 стр.

3. Бибило П.Н. Синтез комбинационных ПЛМ-структур для СБИС. - Минск: Наука и техника, 1992. - 232с.

4. Биргер А.Г. Многозначное дедуктивное моделирование цифровых устройств // Автоматика и вычислительная техника. 1982. №4. С.77-82.

5. Бохманн Д., Постхоф Х. Двоичные динамические системы.- М.: Энергоатомиздат, 1986,- 400с.

6. Введение в системы баз данных – СПб: Издательский дом "Вильямс", 2008. - 848 с.;

7. Вигерс Карл, Разработка требований к программному обеспечению, Пер, с англ. - М.:Издательско-торговый дом "Русская Редакция", 2007. -576с.: ил

8. Вигерс Карл, Разработка требований к программному обеспечению, Пер, с англ. - М.:Издательско-торговый дом "Русская Редакция", 2008. -576с.: ил

9. Виленкин Н.Я. Популярная комбинаторика.- М.: Наука, 1975. - 208 с.

10. Гашков С. Б., Э. А. Применко, М. А. Черепнев Криптографические методы защиты информации, М, Издательство: Академия, 2010 г., 304 стр.

11. Гвоздева Т. В., Б. А. Баллод, Проектирование информационных систем, М, Издательство: Феникс, 2009 г., 512 стр.

12. Глушков В.М. Некоторые проблемы синтеза цифровых автоматов /г Вычислительная математика и математическая физика, 1961, т.1, №3. — С. 371-411.

13. Глушков В.М. Об одном алгоритме синтеза конечных автоматов // Украинский математический журнал, 1960, т. 12, №2. - С. 147-156.

14. Глушков В.М. Об одном методе анализа абстрактных автоматов // ДАН УССР, 1960, т. 12, №9. - С. 1151-1154.

15. Глушков В.М. Синтез цифровых автоматов.- М.:Физматгиз, 1962.-476 с.

16. Голицына О. Л., И. И. Попов, Н. В. Максимов, Т. Л. Партыка, Информационные технологии, М, Издательство Инфра-М, 2009 г., 608 стр.

17. Гольдберг Е.И. Метод оптимальной реализации на ПЛМ цифровых устройств, описываемых многозначными функциями.- Дис.канд.техн.наук: 05.13.12.-Минск, Институт технической кибернетики АН Беларуси, 1995.177 с.

18. ГОСТ 34.601-90. Информационная технология. Автоматизированные системы. Стадии создания

19. ГОСТ Р ИСО/МЭК 12207/99. Государственный стандарт РФ. Информационная технология. Процессы жизненного цикла информационных систем. Издание официальное. - М., 1999

20. Девятков В.В. Методы реализации конечных автоматов на сдвиговых, регистрах. - М.: Энергия, 1974. - 80 с.

21. Денисенко Е. Л. Иерархический синтез асинхронных автоматах на программируемых логических интегральных схемах (ПЛИС) с учетом, ограничений // УСиМ, 1997, №1/3, - С.72-77.

22. Денисенко Е.Л. Сеть параллельных автоматов // УСиМ, 1998, №3. -С. 3436.

23. Дж.Ф.Уэйкерли. Проектирование цифровых устройств. - М.: Постмаркет, 2002. т. 1 - 544 е., т. II-528 с.

24. Дэвид Флэнаган. JavaScript. Подробное руководство: Учебник – М.: Символ Плюс, 2008. 243 – 249 с.

25. Емельянова Н. З., Партыка Т. Л., И. И. Попов, Проектирование информационных систем, М, Издательство: Форум, 2009 г., 432 стр.

26. Закревский А.Д. Алгоритмы синтеза дискретных автоматов. - Москва: Наука, 1971.-512 с.

27. Закревский А.Д. Логический синтез каскадных схем. - Москва: Наука, 1981.-416 с.

28. Илюшечкин В. М. , Основы использования и проектирования баз данных, М, Издательство Юрайт, 2010 г., 224 стр.

29. Коробкова Е.Н. О применении метода сжатия области определений функций одной переменной к нахождению булевых производных // Открытые информационные и компьютерные интегрированные технологии- Харьков: НАКУ. -2003. - Вып. 18.- С. 177-186.

30. Коробкова Е.Н. Приложение свойств обобщённых функций одной переменной к синтезу многофункциональных триггерных устройств // XVIII научные чтения. БГТУ им. В.Г. Шухова. Часть 6. - Белгород. - 2007. - С. 40-42.

31. Котляров В. П., Т. В. Коликова, Основы тестирования программного обеспечения, Издательства: Интернет-университет информационных технологий, Бином. Лаборатория знаний, 2009 г., 288 стр.

32. Кузин А. В., С. В. Левонисова, Базы данных, М, Издательство: Академия, 2008 г., 320 стр.

33. Леффингуелл Д., Уидриг Д, Принципы работы с требованиями к программному обеспечению, М.: ИД "Вильямс", 2008

34. Макарова Н.В Информатика: Учебник, М.: Финансы и статистика, 2008. - 768 с

35. Меняев М.Ф, Информационные технологии управления: Книга 3: Системы управления организацией, М.: Омега-Л, 2007. - 464 с

36. Молчанов А. Ю., Системное программное обеспечение, М, Издательство: Питер, 2010 г., 400 стр.

37. Незнанов А. А., Программирование и алгоритмизация, М, Издательство: Академия, 2010 г., 304 стр.

38. Пирогов В. Ю., Информационные системы и базы данных. М, Организация и проектирование, Издательство: БХВ-Петербург, 2009 г.528 стр.

39. Реляционные базы данных: практические приемы оптимальных решений. – СПб.: БХВ-Петербург, 2009 – 400с.:ил;

40. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. ГОСТ 19.701-90 (ИСО 5807-85) / Государственный комитет СССР по управлению качеством продукции и стандартам, 01.01.1992.

41. Фаулер М, Скотт К, UML в кратком изложении. Применение стандартного языка объектного моделирования, Пер. с англ. - М.:Мир, 2009. - 191 с., ил.

42. Чипига А. Ф., Информационная безопасность автоматизированных систем, М, Издательство: Гелиос АРВ, 2010 г., 336 стр.

43. Чернышев Ю.О. Оптимизация вычислительных структур целочисленными методами теории потоков в сетях: дис. ... докт. техн. наук. / Ю.О. Чернышев. - Таганрог, 1979. - 429 с.

44. Чернов Н.И. Разработка основ теории логического синтеза компонентов СБИС в линейных пространствах: дис. ... докт. техн. наук. / Н.И. Чернов. - Таганрог, 2003.- 335 с.

45. Курейчик В.М. Адаптация на основе самообучения. / В.М. Курейчик, Б.К. Лебедев, О.Б. Лебедев, Ю.О. Чернышев. - Ростов н/Д: РГАС- ХМ ГОУ, 2004. - 146 с.

46. Поспелов Д.А. Логические методы анализа и синтеза схем; изд. 3-е, перераб. и доп. / Д.А. Поспелов. - М.: Энергия, 1974. - 368 с.

47. Лебедев Б.К. Адаптация в САПР: монография. / Б.К. Лебедев. Таганрог: Изд-во ТРТУ, 1999. - 160 с.

48. Гольдберг Е.И. Метод оптимальной реализации на ПЛМ цифровых устройств, описываемых многозначными функциями.- Дис.канд.техн.наук: 05.13.12.-Минск, Институт технической кибернетики АН Беларуси, 1995.177 с.

49. Коробкова Е.Н. О применении метода сжатия области определений функций одной переменной к нахождению производных // Открытые информационные и компьютерные интегрированные технологии- Харьков: НАКУ. -2003. - Вып. 18.- С. 177-186.

50. Поттосин Ю.В. Методы минимизации числа состояний дискретного автомата (обзор) // АиТ, 1971, №8. - С.78-93.

51. Поттосин Ю.В., Шестаков Е.А. Приближенные алгоритмы параллельной декомпозиции автоматов // АВТ, 1981, №2. - С. 31-38.

52. Проектирование и диагностика компьютерных систем и сетей: Учебн. пособие /М.Ф.Бондаренко, Г.Ф.Кривуля, В.Г. Рябцев, С.А.Фрадков, В.И.Хаханов.- К: НМЦ ВО, 2000.- 306с.

Приложение. Листинг программных модулей

```
unit uCustomSearchEngine;

interface

uses Classes, SysUtils, uCustomFunction, uFunctionVariable;

type
  TExtremumType = (etMin, etMax);
  /// <summary>
///   Класс, определяющий базовые особенности поисковиков (Max, Min)
/// </summary>
  TCustomSearchEngine = class
    private
    /// <summary>
    ///   Номеритерации
    /// </summary>
    FIteration : Int64;
/// <summary>
    ///   История вычислений (движение к экстремуму)
/// </summary>
    FExtremumHistory : TStringList;
    /// <summary>
    ///   Историявычислений (списки)
/// </summary>
    FPointSetHistory : TStringList;
    /// <summary>
    ///   Флаг, который указывает, записывать ли историю вычислений
```

```
    /// </summary>
        FSaveHistory : Boolean;
        /// <summary>
        ///   Целеваяфункция
        /// </summary>
        FCustomFunction : TCustomFunction;
        /// <summary>
        ///   Стартоваяточкаиобластьограниченийточки
        /// </summary>
        FStartPoint : TFunctionVariable;
        /// <summary>
        ///   Результатвычислений (экстремум)
        /// </summary>
        FResultValue : TFunctionVariable;
        /// <summary>
        ///   Типпоиска
        /// </summary>
        FExtremumType : TExtremumType;
        /// <summary>
        ///   Допустимоеколичествопустыхитераций
    /// </summary>
        FEmptyIterationLimit : Integer;
        /// <summary>
        ///      Предел для итераций (на случай расхождения метода оптимизации)
    /// </summary>
        FIterationLimit : Int64;
        /// <summary>
        ///   Целеваяфункция
        /// </summary>
```

```
        procedure SetCustomFunction (Value : TCustomFunction);
    /// <summary>
        ///   Указать допустимое количество пустых итераций
    /// </summary>
        procedure SetEmptyIterationLimit (Value : Integer);
    /// <summary>
        ///      Предел для итераций (на случай расхождения метода оптимизации)
    /// </summary>
        procedure SetIterationLimit (Value : Int64);


        protected
        /// <summary>
        ///   Флагуспешныхвычислений
        /// </summary>
    FComputed : Boolean;
        /// <summary>
        ///   Метод записи истории движения к экстремуму
    /// </summary>
        procedure AddExtremumHistory (P : TFunctionVariable);
    /// <summary>
        ///   Метод записи истории вычислений (списки)
    /// </summary>
        procedure AddPointSetHistory (L : TStringList);
    /// <summary>
        ///   Функция поиска экстремума
        /// </summary>
    procedure InternalSearch;  virtual;
        /// <summary>
        ///   Инкрементдляитератора
```

```
        /// </summary>
        procedure IncIteration;
        /// <summary>
        ///   Копированиелистаточек
        /// </summary>
        function GetCloneVariableList (Source : TStringList) : TStringList;

        public
        /// <summary>
        ///   Конструкторкласса
        /// </summary>
        constructor Create;
        /// <summary>
        ///   Деструкторкласса
        /// </summary>
        destructor Destroy; override;
    /// <summary>
        ///      Функция очистки переменных, содержащих результаты вычислений
    /// </summary>
        procedure Clear;
        /// <summary>
        ///   Функцияпоискаэкстремума
    /// </summary>
        procedure Search;
        /// <summary>
        ///   История вычислений (движение к экстремуму)
    /// </summary>
        property ExtremumHistory : TStringList read FExtremumHistory;
    /// <summary>
```

```
        ///   История вычислений (списки)
        /// </summary>
    property PointSetHistory : TStringList read FPointSetHistory;
    /// <summary>
        ///   Результат вычислений (экстремум)
        /// </summary>
    property ResultValue : TFunctionVariable read FResultValue;
    /// <summary>
        ///   Флаг успешных вычислений
        /// </summary>
    property Computed : Boolean read FComputed;
    /// <summary>
        ///   Флаг, который указывает, записывать ли историю вычислений
    /// </summary>
        property SaveHistory : Boolean read FSaveHistory write FSaveHistory;
        /// <summary>
        ///   Номеритерации
        /// </summary>
        property Iteration : Int64 read FIteration;
        /// <summary>
        ///   Целеваяфункция
        /// </summary>
        property CustomFunction : TCustomFunction read FCustomFunction
write SetCustomFunction;
    /// <summary>
        ///   Стартовая точка и область ограничений точки
    /// </summary>
        property StartPoint : TFunctionVariable read FStartPoint write
FStartPoint;
        /// <summary>
```

```
        ///   Типпоиска
        /// </summary>
        property ExtremumType : TExtremumType read FExtremumType write FExtremumType;
    /// <summary>
        ///   Допустимое количество пустых итераций
        /// </summary>
    property EmptyIterationLimit : Integer read FEmptyIterationLimit write SetEmptyIterationLimit;
    /// <summary>
        ///     Предел для итераций (на случай расхождения метода оптимизации)
    /// </summary>
        property IterationLimit : Int64 read FIterationLimit write SetIterationLimit;
      end;


    implementation

    procedure TCustomSearchEngine.SetIterationLimit (Value : Int64);
    begin
      //
      if Value < 1 then
        raise Exception.Create('Предел количества итераций должен находиться в области целых чисел [1, +)');
    FIterationLimit := Value;
    end;
```

```
procedure TCustomSearchEngine.SetCustomFunction (Value : TCustomFunction);
begin
  //
  if Assigned (FResultValue) then FreeAndNil (FResultValue);
  FCustomFunction := Value;
  if FCustomFunction <> nil then
  begin
    //
    FResultValue := TFunctionVariable.Create(FCustomFunction.VariableCount);
  end;
end;

procedure TCustomSearchEngine.SetEmptyIterationLimit (Value : Integer);
begin
  //
  if Value < 1 then
    raise Exception.Create('Количество пустых итераций должно находиться в области целых чисел [1, +)');
  FEmptyIterationLimit := Value;
end;

function TCustomSearchEngine.GetCloneVariableList (Source : TStringList) : TStringList;
var
  I : Integer;
  P : TFunctionVariable;
begin
  //
```

```
    Result := TStringList.Create;
    try
      //
      Result.OwnsObjects := True;
      for I := 0 to Source.Count - 1 do
      begin
        //
        P := Source.Objects[I] as TFunctionVariable;
        Result.AddObject(IntToStr (I + 1), P.GetClone);
      end;
    except
      Result.Free;
      raise;
    end;
  end;

  procedure TCustomSearchEngine.IncIteration;
  begin
    //
    FIteration := FIteration + 1;
    if FIteration > FIterationLimit then
  raise Exception.Create('Превышено максимально-допустимое
количество итераций');
  end;

  procedure TCustomSearchEngine.InternalSearch;
  begin
    //

  end;
```

```
procedure TCustomSearchEngine.AddExtremumHistory (P : TFunctionVariable);
begin
  //
  if FSaveHistory then
  begin
    //
    FExtremumHistory.AddObject(IntToStr (FExtremumHistory.Count + 1), P);
  end
  else
  begin
    //
    if Assigned (P) then P.Free;
  end;
end;

procedure TCustomSearchEngine.AddPointSetHistory (L : TStringList);
begin
  //
  if FSaveHistory then
  begin
    //
    FPointSetHistory.AddObject(IntToStr (FPointSetHistory.Count + 1), L);
  end
  else
  begin
    //
```

```
      if Assigned (L) then
      begin
        L.OwnsObjects := True;
        L.Free;
      end;
    end;
end;

procedure TCustomSearchEngine.Clear;
begin
  //
  FIteration := 0;
  FComputed := False;
  FResultValue.Clear;
  FExtremumHistory.Clear;
  FPointSetHistory.Clear;
end;

constructor TCustomSearchEngine.Create;
begin
  //
  inherited Create;
  FResultValue := nil;
  IterationLimit := 5000;
  EmptyIterationLimit := 10;
  ExtremumType := etMin;
  SaveHistory := True;
  FExtremumHistory := TStringList.Create;
  FExtremumHistory.OwnsObjects := True;
  FPointSetHistory := TStringList.Create;
```

```
   FPointSetHistory.OwnsObjects := True;
end;

destructor TCustomSearchEngine.Destroy;
begin
   //
   Clear;
FExtremumHistory.Free;
   FPointSetHistory.Free;
   if Assigned (FResultValue) then FResultValue.Free;
   inherited Destroy;
end;

procedure TCustomSearchEngine.Search;
begin
   //
   if Assigned (FCustomFunction) = False then
      raise Exception.Create('Незаданацелеваяфункция');
   if Assigned (FStartPoint) = False then
raise Exception.Create('Не задана начальная точка вычислений');
Clear;
   InternalSearch;
end;

end.
unit uGravitationalSearchEngine;

interface
```

```
uses Classes, SysUtils, uCustomFunction, uFunctionVariable,
uParticleBasedSearchEngine,
  Math, uCommonFunctions;

type
  /// <summary>
  ///   Класс, реализующийгравитационныйпоиск (Gravitational Search)
  /// </summary>
  TGravitationalSearchEngine = class (TParticleBasedSearchEngine)
    protected
    /// <summary>
    ///   Малаяконстанта
    /// </summary>
    FSmallConstant : Extended;
/// <summary>
    ///   Функция вычисления гравитационной постоянной
    /// </summary>
  function GetG (X : Extended) : Extended;
/// <summary>
    ///   Вычислить массу точки
    /// </summary>
  function GetParticleWeight (Index : Integer) : Extended;
/// <summary>
    ///   Вычислить силу точки
    /// </summary>
  function GetParticlePower (Index : Integer; Dimension : Integer) : Extended;
/// <summary>
    ///   Рассчет силы между точками
    /// </summary>
```

```
        function GetPowerBetweenParticles (Index1 : Integer; Index2 : Integer; Dimension : Integer) : Extended;
        /// <summary>
            ///   Функция поиска экстремума
            /// </summary>
        procedure InternalSearch; override;
            /// <summary>
            ///   Установкамалойконстанты
            /// </summary>
            procedure SetSmallConstant (Value : Extended);



            public
            /// <summary>
            ///   Конструкторкласса
            /// </summary>
            constructor Create;
            /// <summary>
            ///   Деструкторкласса
            /// </summary>
            destructor Destroy; override;
        /// <summary>
            ///   Установка малой константы
            /// </summary>
        property SmallConstant : Extended read FSmallConstant write SetSmallConstant;



          end;
```

```
implementation

procedure TGravitationalSearchEngine.SetSmallConstant (Value : Extended);
begin
  //
  FSmallConstant := Value;
end;

function TGravitationalSearchEngine.GetPowerBetweenParticles (Index1 : Integer; Index2 : Integer; Dimension : Integer) : Extended;
var
  //
  I : Integer;
  P1, P2 : TFunctionVariable;
begin
  //
  Result := 0;
  P1 := GetParticle(Index1);
  P2 := GetParticle(Index2);
  Result := (P2.X[Dimension] - P1.X[Dimension]) * GetG(Iteration) * (GetParticleWeight (Index1)*GetParticleWeight (Index2)) / ZeroCorrect (P1.GetEuclideanDistanceBetweenX(P2) + FSmallConstant, 1);
end;

function TGravitationalSearchEngine.GetParticlePower (Index : Integer; Dimension : Integer) : Extended;
var
  //
```

```
    I : Integer;
    P : TFunctionVariable;
  begin
    //
    Result := 0;
    P := GetParticle(Index);
    for I := 0 to ParticleList.Count - 1 do
    begin
      if I <> Index then
      begin
        Result := Result + Random * GetPowerBetweenParticles (Index, I,
Dimension);
      end;
    end;
  end;


  function TGravitationalSearchEngine.GetParticleWeight (Index : Integer) :
Extended;
  var
    //
    I : Integer;
    Wcurrent, Wmax, Wmin, S : Extended;
  begin
    //
    S := 0;
    Wmax := GetMaxParticle.Y;
    Wmin := GetMinParticle.Y;

    for I := 0 to ParticleList.Count - 1 do
```

```
    begin
      //
      Wcurrent := GetParticle(I).Y;
      S := S + (Wcurrent - Wmin) / ZeroCorrect (Wmax - Wmin, 1);
    end;
    Wcurrent := GetParticle(Index).Y;
    Result := ((Wcurrent - Wmin) / ZeroCorrect (Wmax - Wmin, 1)) /
ZeroCorrect (S, 1);
  end;



  function TGravitationalSearchEngine.GetG (X : Extended) : Extended;
  begin
    //
    Result :={ 6.67428 * Power (10, -11)} 100 / Exp (X);
  end;



  constructor TGravitationalSearchEngine.Create;
  begin
    //
    Randomize;
    inherited Create;
    SmallConstant := 1;
  end;

  destructor TGravitationalSearchEngine.Destroy;
```

```
begin
  //
  inherited Destroy;
end;

procedure TGravitationalSearchEngine.InternalSearch;
var
  //
  NewParticleList : TStringList;
  I, J, Counter : Integer;
  P : TFunctionVariable;
  Weight, A, Power : Extended;
  NewExtremum, OldExtremum  : Extended;
begin
  //
  Counter := 0;
  GenerateParticleList;
  AddPointSetHistory(GetCloneVariableList (ParticleList));
  AddExtremumHistory (GetTargetExtremumParticle.GetClone);
  try
    //
    while True do
    begin
      //
      NewParticleList := TStringList.Create;
      try
        NewParticleList.OwnsObjects := True;
        OldExtremum := GetTargetExtremumParticle.Y;
        for I := 0 to ParticleList.Count - 1 do
        begin
```

```
        //
        P := GetParticle(I).GetClone;
        Weight := ZeroCorrect (GetParticleWeight(I), 1);
        for J := 0 to P.Size - 1 do
        begin
          //
          Power := GetParticlePower(I, J);
          A := Power / Weight;
          P.Tag := Random * P.Tag + A;
          P.X[J] := P.X[J] + P.Tag;
        end;
        CustomFunction.Compute(P);
        NewParticleList.AddObject('', P);
      end;
      ParticleList.Free;
      SetParticleList(NewParticleList);
      NewExtremum := GetTargetExtremumParticle.Y;
    except
      NewParticleList.Free;
      raise;
    end;

    IncIteration;
    if GetImprovement (NewExtremum, OldExtremum) then
    begin
      //
      Counter := 0;
      AddExtremumHistory (GetTargetExtremumParticle.GetClone);
      ResultValue.Copy (GetTargetExtremumParticle);
      AddPointSetHistory(GetCloneVariableList (ParticleList));
```

```
            end
            else
            begin
               Inc (Counter);

               if Counter >= EmptyIterationLimit then
                  Exit;
            end;
         end;
         FComputed := True;
      finally
      end;
   end;

   end.
```