УДК 004.415.2, 004.588

ОПТИМИЗАТОР БАЗ ЗНАНИЙ НА ОСНОВЕ МЯГКИХ ВЫЧИСЛЕНИЙ

Литвинцева Людмила Васильевна[1] , Тятюшкина Ольга Юрьевна[2], Григорьев Павел Николаевич[3] , Ульянов Сергей Викторович[4]

[1] Кандидат физико-математических наук, доцент;
ГОУ ВПО «Международный Университет природы, общества и человека «Дубна»,
Институт системного анализа и управления;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: ulyanovsv@mail.ru.

[2] Старший преподаватель;
ГОУ ВПО «Международный Университет природы, общества и человека «Дубна»,
Институт системного анализа и управления;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: tyatyushkina@mail.ru.

[3]Аспирант;
ГОУ ВПО «Международный Университет природы, общества и человека «Дубна»,
Институт системного анализа и управления;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: pavloon@mail.ru.

[4] Доктор физико-математических наук, профессор;
ГОУ ВПО «Международный Университет природы, общества и человека «Дубна»,
Институт системного анализа и управления;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: ulyanovsv@mail.ru.

Описан разработанный программный инструментарий проектирования баз знаний для использования в ситуациях обучения и в непредвиденных ситуациях управления. Использование инструментария приводит к повышению уровня робастности интеллектуальных систем управления.

Ключевые слова: оптимизатор баз знаний, интеллектуальное управление, робастность.

KNOWLEDGE BASE OPTIMIZER BASED ON SOFT COMPUTING

Litvintseva Ludmila[1], Tyatyushkina Olga[2], Grigoryev Pavel[3] , Ulyanov Sergey[4]

[1] Candidate of Science in Physics and Mathematics, associate professor;
Dubna International University of Nature, Society, and Man,
Institute of system analysis and management;
141980, Dubna, Moscow reg., Universitetskaya str., 19;
e-mail: ulyanovsv@mail.ru.

[2] Senior teacher;
Dubna International University of Nature, Society, and Man,
Institute of system analysis and management;
141980, Dubna, Moscow reg., Universitetskaya str., 19;
e-mail: tyatyushkina@mail.ru.

[3] *Postgraduate student;*
Dubna International University of Nature, Society, and Man,
Institute of system analysis and management;
141980, Dubna, Moscow reg., Universitetskaya str., 19;
e-mail: pavloon@mail.ru.

[4] *Doctor of Science Physics and Mathematics, professor;*
Dubna International University of Nature, Society, and Man,
Institute of system analysis and management;
141980, Dubna, Moscow reg., Universitetskaya str., 19;
e-mail: ulyanovsv@mail.ru.

A developed toolkit of knowledge base design for application in learning situations and in unpredicted control situations is described. The applications of developed toolkit are shown that the robust level of intelligent control systems is increased.

Keywords: knowledge base optimizer, intelligent control, robustness.

Введение

Инженерные методы теории управления и технологии проектирования систем автоматического управления (САУ) сформированы в прошлом столетии. Были заложены, в частности, основы стохастического и адаптивного управления сложными динамическими системами (в общем случае с переменной структурой) в условиях информационной неопределённости и определены предельные возможности процессов проектирования точных моделей. Следующим шагом в этом направлении явилась разработка принципов моделирования и проектирования нечётких САУ в условиях неопределенности, учитывающая индивидуальные особенности поведения выборочных траекторий движения объектов управления (ОУ) [1]. За основу данной методологии проектирования принята теория лингвистической аппроксимации и нечеткого логического вывода (Л. Заде и др.) для создания робастных баз знаний (БЗ) интеллектуальных нечетких регуляторов (НР) [2, 3]. В рамках указанной методологии проектирования законов управления на основе физических методов (информационно–термодинамические и квантово–релятивистские методы описания ОУ и процессов управления [4]) в середине 70-х годов прошлого столетия были заложены основы технологии проектирования интеллектуальных САУ (ИСУ) [5 – 8][1].

ИСУ первого поколения представляли собой экспертную систему (ЭС) с различными по глубине представлениями знаний (данных). Такую систему можно спроектировать, например, введением в структуру САУ блока нечеткого логического вывода. Основную роль в этих ЭС играло качество используемой БЗ, которое зависит от опыта и субъективных знаний человека-эксперта. Однако в случае управления глобально неустойчивыми и существенно нелинейными ОУ, находящимися под воздействием сложных стохастических шумов, даже опытному человеку-эксперту трудно подобрать оптимальную (с точки зрения качества управления) БЗ НР. Хорошо известен факт, что при случайных возмущениях для нелинейной САУ с тремя входными и двумя выходными сигналами эксперту уже трудно сформировать описание функций принадлежности и количество продукционных правил в БЗ, обеспечивающие требуемый уровень робастности управления [2]. В результате ведущая фирма «Омрон» (Япония) приостановила серийный выпуск НР для объектов управления со сложной структурой. Эта проблема – узкое место всех первых (и их последующих модификаций) ИСУ [2, 3]. Поэтому использование ЭС в качестве инструментария извлечения знаний и формирования БЗ (как основы технологии проектирования ИСУ) не привело к ожидаемому существенному успеху (хотя существует много примеров эффективного промышленного внедрения [5, 9]) в силу сложности ОУ и субъективности информации, вносимой экспертом.

[1] Первые практические результаты, полученные на основе разработанной методологии проектирования НР в рамках программы «Интеркосмос», были доложены более 30 лет назад коллективом авторов совместно с акад. Б.Н. Петровым и акад. В.С. Пугачевым на 7-ом Конгрессе IFAC, Хельсинки, 1978 [6]. Доклад на конгресс был представлен Prof. L. Zadeh. Данная работа является развитием полученных школой Б.Н. Петрова результатов в данном направлении за период 1977 – 2010гг. [7].

С точки зрения технологии проектирования основной проблемой внедрения ИСУ первого поколения являлась их слабая адаптивность к изменениям параметров ОУ (вызванных, например, старением структуры ОУ или резким изменением внешней среды), а также низкая робастность полученных законов управления. Для решения подобных проблем были разработаны ИСУ второго поколения с глубинным представлением знаний, используя технологии так называемых мягких вычислений, объединяющие в единую цепочку генетические алгоритмы (ГА), нечеткие нейронные сети (ННС) и НР.

В дальнейшем второе поколение ИСУ стало использовать новый вид обратной связи, называемой глобальной интеллектуальной обратной связью (ГИОС) [1, 3, 8, 10]. Это дает возможность извлекать объективные знания непосредственно из самого динамического процесса поведения ОУ и исполнительного устройства САУ. Контур ГИОС включает ГА для получения информации об оптимальном сигнале управления (исходя из динамического и термодинамического поведения ОУ и ПИД-регулятора) и ННС, аппроксимирующей данный оптимальный сигнал управления с помощью заданной структуры нейронной сети. Такой поход рассмотрен в [1].

Анализ результатов моделирования САУ на базе первого этапа разработанной технологии проектирования БЗ показал основной недостаток – неоптимальный выбор структуры ННС. Как правило, в системах проектирования БЗ ИСУ такого типа построение соответствующей структуры ННС возложено на опытного человека-эксперта. Задача эксперта при определении структуры ННС сводится к выбору модели нечеткого вывода и главным образом к лингвистическому описанию заданного обучающего сигнала (ОС). Однако решение этой задачи вручную в сложных ситуациях управления представляет большую проблему даже для опытных экспертов.

Другой важной проблемой остается определение требуемого соотношения между точностью описания (аппроксимации) ОС и необходимым уровнем робастности всей структуры ННС. Обе указанные проблемы решаются на втором этапе технологии построения БЗ ИСУ с помощью программных средств инструментария, названного оптимизатор баз знаний (ОБЗ) [3, 7, 10 – 13].

В данной статье рассмотрена программная архитектура, принятая за основу ОБЗ. Основное внимание сконцентрировано также на описании конкретных результатов моделирования ИСУ сложными существенно-нелинейными объектами управления со случайно изменяющейся структурой.

1. Технология проектирования робастных БЗ в ИСУ

ОБЗ является новым эффективным программным инструментарием построения БЗ робастных ИСУ на основе мягких вычислений с использованием новых критериев оптимизации (в виде новых типов функций пригодности ГА). В качестве таковых выступают термодинамические и информационно-энтропийные критерии качества управления [2]. Для построения БЗ с использованием ОБЗ необходимо наличие ОС, который может быть получен либо на этапе стохастического моделирования поведения ОУ, проведенного с использованием его математической модели, либо экспериментально, путем непосредственного измерения динамических параметров физической модели ОУ.

Создание БЗ происходит в несколько этапов:

Этап 1. *Выбор модели нечеткого вывода*. Пользователь конкретизирует тип нечеткой модели вывода (Сугено, Мамдани и т.д.), операцию нечеткого «**И»** (произведение или минимум), число входных и выходных переменных.

Этап 2. *Создание лингвистических переменных.* С помощью генетического алгоритма ($ГА_1$) определяется оптимальное число функций принадлежности для каждой входной лингвистической переменной, а также выбирается оптимальная форма представления их функций принадлежности (треугольная, Гауссовская и т.д.). В качестве критерия оптимальности той или иной конфигурации лингвистических переменных используется максимум совместной информационной энтропии и минимум информации о сигналах в отдельности, которые имеют вид:

$$H^{(j,l)}_{X_i|X_k} = H\left(x_j \left| x_i = \mu^{j}_{X_i}, x_k = \mu^{l}_{X_k} \right. \right) = -\frac{1}{N}\sum_{t=1}^{N}\left[\mu^{j}_{X_i}\left(x_i(t)\right) * \mu^{l}_{X_k}\left(x_k(t)\right)\right] \log\left[\mu^{j}_{X_i}\left(x_i(t)\right) * \mu^{l}_{X_k}\left(x_k(t)\right)\right] \quad (1.1)$$

и

$$H^{j}_{X_i} = -p^{j}_{X_i} \log\left(p^{j}_{X_i}\right) = -p\left(x_i \middle| x_i = \mu^{j}_{X_i}\right) \log\left[p\left(x_i \middle| x_i = \mu^{j}_{X_i}\right)\right] == -\frac{1}{N}\sum_{t=1}^{N} \mu^{j}_{X_i}\left(x_i(t)\right) \log\left[\mu^{j}_{X_i}\left(x_i(t)\right)\right] \quad (1.2)$$

соответственно.

Здесь * – выбранная операция нечёткого «**И**»; X_i, $i=1,...,m$ – набор лингвистических переменных, соответствующих компонентам ОС; $\mu^{j_i}_{X_i}$, $i = 1,...,m, j_i = 1,...,l_{X_i}$ терм-множество, соответствующее i-й компоненте ОС; $\mathbf{x}(t) = \left(x_1(t),\ldots,x_m(t)\right)$ – компоненты ОС; N – число компонентов ОС.

Этап 3. *Создание базы правил.* На данном этапе используется специальный алгоритм отбора наиболее «робастных правил» в соответствии со следующими двумя критериями:

(1) «суммарный» критерий: выбрать только те правила, которые удовлетворяют следующим условиям:

$$R^{l}_{total\_fs} \geq TL\,, \quad (1.3)$$

где *TL* (*threshold level*) – заданный (вручную или выбранный автоматически) уровень активации правила, и

$$R^{l}_{total\_fs} = \sum_{k=1}^{N} R^{l}_{fs}(t_k)\,, \text{ и } R^{l}_{fs}(t_k) = \prod\left[\mu^{l}_{j1}(x_1(t_k)), \mu^{l}_{j2}(x_1(t_k)),..., \mu^{l}_{jn}(x_n(t_k))\right], \quad (1.4)$$

где t_k – моменты времени, $k = 1,...,N$; $\mu^{l}_{jk}(x_k), k = 1,...,n$ – функции принадлежности входных переменных, l – индекс правила в БЗ; символ «Π» обозначает операцию нечеткой конъюнкции (в частности, может интерпретироваться как произведение или минимум);

(2) «максимальный» критерий: выбрать только те правила, которые удовлетворяют условию

$$\max_{t} R^{l}_{fs}(t) \geq TL\,. \quad (1.5)$$

Этап 4. *Оптимизация базы правил.* С помощью генетического алгоритма ($ГА_2$) оптимизируются правые части правил БЗ, определенной на этапе (шаге) 3. Критерием качества на этом этапе выступает минимум ошибки аппроксимации ОС (обучающего сигнала):

$$E = \sum_{p=1}^{N} E^{p}\,, \qquad E^{p} = 1/2\left(F\left(x^{p}_{1}, x^{p}_{2},\ldots, x^{p}_{n}\right) - d^{p}\right)^{2}, \quad (1.6)$$

где F – выход ИСУ, d – целевое состояния ОУ согласно ОС.

На данном этапе находится решение, близкое к глобальному оптимуму (минимум ошибки аппроксимации ОС). С помощью следующего этапа это решение может быть локально улучшено.

Этап 5. *Настройка базы правил.* С помощью генетического алгоритма ($ГА_3$) оптимизируются левые и правые части правил БЗ, т.е. подбираются оптимальные параметры функций принадлежности входных/выходных переменных (с точки зрения заданной функции пригодности ГА). В данном процессе оптимизации используются различные функции пригодности, выбранные пользователем. В качестве таких функций может выступать как минимум ошибки аппроксимации ($E = \sum_{p} E^{p}$), так и максимум совместной информационной энтропии ($H^{j}_{X_i}$). Кроме того, может происходить настройка БЗ с помощью других оптимизационных алгоритмов, таких как алгоритм обратного распространения ошибки.

2. Программная реализация ОБЗ

2.1. Модель ИСУ

Первая версия ОБЗ (оптимизатора базы знаний) [11, 13] использовала фиксированную структуру оптимизируемой нечеткой системы управления, состоящую из одного блока лингвистических переменных и одной базы нечетких правил, из которых одновременно могла использоваться только одна. Практическое использование оптимизатора показало, что такая схема является слишком жесткой и ограничивает возможности проектирования ИСУ. В качестве промежуточного решения для повышения гибкости оптимизатора в него была добавлена возможность поддержки нескольких баз данных. При этом только одна база могла быть выбрана в качестве активной (использовалась для оптимизации и работы), и все базы имели один и тот же тип. На этом возможности расширения имевшейся жесткой схемы оптимизатора были исчерпаны.

Для построения второго поколения ОБЗ ИСУ была выбрана модульная схема, которая подразумевает построение пользователем модели ИСУ из набора функциональных блоков, таких как модули ввода и вывода данных, лингвистических переменных, нечеткого вывода и других. При этом пользователь может произвольным образом соединять эти блоки, формируя модели сложных ИСУ. В основе модульной модели ИСУ лежат классы Model, DataArchive и Block.

Класс Model инкапсулирует все объекты, необходимые для построения модели ИСУ и обеспечивает взаимодействие между ними. Он отвечает за ведение списка имеющихся в модели ИСУ модулей, отслеживает корректность связей между модулями, предоставляет интерфейсы для передачи данных модели для вычисления, а также получения результатов. Одной из основных функций этого класса является обеспечение правильного порядка вызовов модулей при проведении вычисления. Корректный порядок вычислений подразумевает выполнение следующих условий:

- все модули, на вход которых поступили новые данные, будут вызваны;
- модули, на вход которых не поступило новых данных, не вызываются;
- модуль не должен вызываться до тех пор, пока все новые данные на его входах не будут доступны[2].

Замечание. Для реализации такого порядка, все модули разбиваются на классы, в зависимости от порядка вычислений. К классу с наивысшим приоритетом относятся модули, не имеющие входных портов. Выполнение функций этих модулей производится в первую очередь. Для остальных модулей приоритет полагается равным минимуму приоритетов модулей, выходные порты которых связаны с входными портами рассматриваемого модуля, уменьшенному на единицу. Значения приоритета обновляются при изменении связей между модулями.

Класс Block является базовым классом для реализаций различных модулей ИСУ. Он предоставляет общий интерфейс, необходимый для проведения расчетов, а также обеспечивает базовые функции, необходимые большинству модулей. Каждый модуль может иметь один или несколько входных и выходных портов, через которые производится обмен данными с другими модулями. Объекты этого класса также могут являться целевыми объектами оптимизации для генетических алгоритмов.

Класс DataArchive обеспечивает хранение и извлечение данных, передаваемых внутри модели. В этом классе хранятся данные, порождаемые выходными портами всех модулей в системе, а также входные и выходные данные модели. Во время вычислений данные с выходного порта каждого модуля заносятся в архив. Если имеются другие модули, входные порты которых подключены к рассматриваемому выходному порту, то для проведения вычислений этот модуль извлекает соответствующие данные из архива. Данные каждой итерации вычисления в архиве могут сохраняться для последующего изучения после завершения вычислений, а также могут использоваться в следующих итерациях, если система определит, что какие-то из данных не требуют пересчета.

[2] Система не поддерживает циклические ссылки между модулями.

2.2. Реализация генетической оптимизации

В оптимизаторе реализовано два генетических алгоритма – обычный генетический алгоритм, обеспечивающий минимизацию вещественной целевой функции, а также алгоритм типа NSGA [14 - 16], который находит множество парето-оптимальных решений для задачи минимизации нескольких целевых функций. Эти алгоритмы могут использоваться с двумя типами хромосом, в которых кодирование решений происходит с помощью бинарной «ДНК» или набором вещественных чисел.

Для поддержки генетической оптимизации были разработаны несколько иерархий классов: OptimizationTarget, Chromosome, ChromosomeFactory и OptimizationEngine. Класс OptimizationTarget объявляет интерфейс для объектов, которые могут быть оптимизированы с помощью генетических алгоритмов. Основные функции, которые должны предоставить такие объекты – это функции кодирования и декодирования состояния объекта в хромосоме, а также вычисление функции полезности для текущего состояния объекта.

Также в этом интерфейсе предусмотрены вспомогательные функции, извещающие объект о начале и конце процесса оптимизации, текущем прогрессе и функцию Optimizeable, позволяющую проверить готовность объекта к оптимизации. Класс OptimizationTarget обеспечивает реализацию этих функций по умолчанию, которая предотвращает повторное использование объекта несколькими алгоритмами оптимизации одновременно. К дочерним объектам этого класса, наряду с некоторыми другими объектами, относится базовый класс модулей ИСУ – Block, что позволяет проводить оптимизацию любых из имеющихся в составе ИСУ модулей.

Модули, не поддерживающие оптимизацию, должны переопределить функцию Optimizeable, чтобы запретить применение оптимизации к ним.

Класс Chromosome является базовым для двух классов хромосом – BinaryChromosome и RealChromosome, обеспечивающим соответственно двоичное кодирование и кодирование с помощью действительных чисел.

Этот класс выполняет две функции. С одной стороны он определяет общий интерфейс для работы с хромосомами, который обеспечивает генетические операции и операции кодирования/декодирования отдельных параметров оптимизируемого объекта в хромосоме.

С другой стороны, он также обеспечивает дочерние классы информацией о структуре оптимизируемого объекта. Для этой цели в нем реализован интерфейс кодирования информации об объекте, который обеспечивает фиксацию параметров кодируемых объектов (какого типа параметры запрашивались, какие минимальные и максимальные значения параметров, какое количество бит и другая информация). В дальнейшем эта информация используется для построения содержащих информацию хромосом, которым передается общее число бит для кодирования в случае двоичных хромосом или число параметров и диапазоны их изменения для хромосом с действительными числами.

Эта информация также может использоваться для просмотра текущих популяций генетических алгоритмов во время оптимизации, а также экспорта соответствующих данных во внешние файлы для дальнейшего анализа, не перегружая дочерние объекты класса OptimizationTarget реализацией соответствующих функций.

Поскольку реализованные в хромосомах генетические операции (а также ряд других операций между хромосомами) имеют смысл только при выполнении над хромосомами одного типа и одной структуры (с одинаковыми размером «ДНК» или числом параметров), необходимо гарантировать совместное использование хромосом одной структуры. Это реализовано с помощью объектов, относящихся к иерархии ChromosomeFactory.

Объекты этой иерархии обеспечивают создание совместимых друг с другом хромосом. Класс ChromsomeFactory объявляет интерфейс, который реализован в двух дочерних классах – BinaryChromosomeFactory, создающем двоичные хромосомы, и RealChromosomeFactory, создающем хромосомы с действительными параметрами. Перед использованием фабрики необходимо инициализировать ее, передав информацию об оптимизируемом объекте (которая будет использована для выявления его структуры) и необходимых свойствах хромосом (размерность вектор-функции полезности, число дополнительных параметров) а также параметры генетических операций (такие как вероятности мута-

ций и скрещиваний). Фабрика хранит необходимую информацию, а также выделяет необходимые для выполнения генетических операций дополнительные ресурсы. Такой подход позволяет одновременно избежать как выделения этих ресурсов при создании каждой хромосомы, так и динамического выделения их во время операции, что существенно снизило бы производительность. После инициализации фабрика может быть использована для создания хромосом определенного типа, со структурой совместимой для кодирования целевого объекта.

Конструкторы объектов класса Chromosome сделаны защищенными, что исключает создание этих объектов в обход использования фабрик (которые, в свою очередь, объявлены как «друзья» соответствующих классов). При создании объектов типа Chromosome им передается адрес создавшей их фабрики, который в дальнейшем используется хромосомами для получения доступа к информации о параметрах генетических операций.

Генетические алгоритмы реализованы в объектах, принадлежащих иерархии OptimizationEngine. Базовый класс объявляет интерфейс, который включает следующие операции:

- инициализацию объекта с передачей ему ссылки на целевой объект типа OptimizationTarget;
- запрос у пользователя параметров алгоритма, с использованием диалогового окна;
- выбор фабрики для создания хромосом;
- получение или установка текущего состояния (набора хромосом, образующего текущее поколение);
- старт алгоритма оптимизации.

Процесс оптимизации протекает следующим образом.

Сначала выбирается объект для оптимизации, и создается объект, реализующий оптимизационный алгоритм. Параметры алгоритма запрашиваются у пользователя, используя диалоговое окно и соответствующий класс для обработки ввода пользователя. Затем вызывается процедура оптимизации. В первую очередь она вызывает инициализацию фабрики хромосом. Фабрика создает объект класса Chromosome и с его помощью определяет структуру оптимизируемого объекта, которая сохраняется для последующего использования. Оптимизационный алгоритм использует фабрику для создания первого поколения хромосом со случайным содержимым. Хромосомы этого поколения по очереди декодируются оптимизируемым объектом и для них вычисляются соответствующие значения функции полезности. После этого оптимизационный алгоритм создает новое поколение, используя определенные генетические операции и сортировки хромосом на базе вычисленных значений функций полезности. Процесс продолжается до завершения оптимизации, согласно выбранным пользователем условиям.

Подобная схема позволяет эффективно реализовать систему, которая может использовать для оптимизации различные виды генетических алгоритмов, а также другие алгоритмы, взаимодействие которых с оптимизируемым объектом можно организовать аналогичным образом. В частности, в ОБЗ были реализованы алгоритмы градиентного спуска, а также алгоритм Downhill Simplex Method [16].

2.3. Подсистема вычисления функции полезности

Одной из основных проблем практического применения генетической оптимизации для создания систем ИСУ является необходимость проведения большого числа вычислений функции полезности [17]. Проведение генетической оптимизации с проверкой работоспособности ИСУ на реальном объекте управления, таким образом, занимает значительное время и требует больших ресурсов. Кроме того, во многих случаях на точность и повторяемость вычисления функции пригодности существенное влияние могут оказывать погрешности измерений и различия в начальных состояния ОУ. Время и стоимость оптимизации может быть сокращено за счет применения оптимизации с использованием обучающего сигнала и математических моделей объекта управления.

Таким образом, практически целесообразным представляется подход к созданию баз знаний ИСУ, при котором начальные этапы оптимизации БЗ проводятся с использованием ОС или математических моделей, а на финальном этапе решения, показавшие себя достаточно хорошо на моделях, проверяются на реальном объекте управления.

Подводя итог вышесказанному, можно сделать вывод, что ОБЗ должен обеспечивать следующие возможности:

- Различные режимы работы, включая:
    - Оптимизация по заданному ОС, в случае чего вычисление функции полезности производится непосредственно оптимизатором путем сравнения с компонентами ОС.
    - Оптимизация с использованием проверки работы ИСУ на модели, реализованной во внешней системе, или на ОУ. При этом функция полезности может вычисляться или во внешней системе/устройстве, или самим ОБЗ путем сравнения одного из внешних сигналов с желаемым значением.
- Проверку соответствия начального состояния ОУ некоторым заданным начальным условиям перед началом вычисления функции полезности.
- Возможность одновременной работы нескольких оптимизационных алгоритмов, что может значительно ускорить оптимизацию не связанных друг с другом частей ИСУ.
- Поиск и выявление повторных запросов на вычисление функции полезности с дальнейшей возможностью использования значения, полученного при первой проверки либо усреднения значения функции по результатам нескольких испытаний в зависимости от предпочтений пользователя.
- Возможность приостанавливать оптимизацию и продолжать ее в дальнейшем, возможно используя различные настройки алгоритмов или даже разные алгоритмы.
- Режим многократного тестирования решений с целью повышения точности измерения функции пригодности.

Поскольку эту функциональность необходимо реализовать во всех оптимизационных алгоритмах, представляется разумным вынести ее за пределы реализации собственно алгоритмов оптимизации и выполнить ее в виде отдельной подсистемы, управляющей процессом оптимизации. Основные функции этой подсистемы в ОБЗ выполняет класс OptimizationManager. Этот класс реализует интерфейс OptimizationTarget, что позволяет передавать его оптимизационным алгоритмам в качестве оптимизируемого объекта. Этот класс обрабатывает запросы алгоритма оптимизации к оптимизационному объекту и перенаправляет их собственно оптимизируемым объектам, выполняя необходимые действия по синхронизации вызовов, проверке повторений хромосом, реализует различные стратегии вычисления функции полезности.

Каждый оптимизационный алгоритм работает в отдельном потоке, что обеспечивает независимость работы всех алгоритмов. Все вычисления функции пригодности производятся в отдельном потоке. Optimization Manager содержит список активных в данный момент алгоритмов оптимизации, в котором фиксирует их запросы на вычисление функции полезности. При поступлении нового запроса проверяется, все ли активные алгоритмы сделали запросы, и запросивший вычисление поток переводится в режим ожидания. После поступления запросов от всех алгоритмов активизируется поток, отвечающий за вычисление функции полезности. Он производит вычисления согласно выбранному режиму (например, производит прогон работы модели ИСУ по обучающему сигналу или вызывает внешнюю систему моделирования). Завершив вычисление функции полезности, поток активизирует потоки оптимизации, а сам переходит в режим ожидания следующего запроса.

Список активных алгоритмов также содержит хэш-таблицы хромосом, для которых данный алгоритм уже проводил вычисление функций пригодности, которые используются для исключения повторных вычислений или усреднения значения функции полезности.

Для обеспечения возможности приостановки вычислений с возможностью дальнейших вычислений предусмотрен механизм сохранения состояния генетических алгоритмов. OptimizationManager обрабатывает вызовы функции информирования объекта OptimizationTarget о прогрессе оптимизации и может приостанавливать работу генетического алгоритма после каждого поколения. При этом пользователь может активировать операцию экспорта текущего поколения. Экспорт производится, используя специальный интерфейс классов OptimizationEngine, который позволяет скопировать текущее поколение в отдельный класс OptimizationState, являющийся хранилищем набора хромосом. Сохраненное таким образом состояние может затем сохранено в оперативной памяти, в файле на диске и использовано как начальное состояние при запуске генетических алгоритмов.

2.4. Интерфейс пользователя

Модульный подход к построению модели ИСУ диктует построение интерфейса пользователя, обеспечивающего вид на систему как на набор блоков и связей между ними. Пользователь может добавлять или удалять блоки, модифицировать связи между ними. Доступ к настройкам каждого блока осуществляется с помощью отдельного диалогового окна, которое можно вызвать щелчком мыши на блоке.

Такой интерфейс обеспечивает максимальную гибкость при создании сложных ИСУ. Однако в случае, если ОБЗ применяется для оптимизации системы управления простой структуры, применение подобного интерфейса будет создавать неудобство для пользователя. В случае, если моделируемая СУ содержит один или несколько однотипных блоков, соединенных фиксированным образом, целесообразно обеспечить пользователю прямой доступ к настройкам блоков.

Чтобы обеспечить возможность создания интерфейсов различного типа, ОБЗ был выполнен в виде библиотеки, содержащей основные функции оптимизатора, а также те части интерфейса, которые могут быть использованы в различных проектах. Эта библиотека затем может быть использована для создания ОБЗ с различными интерфейсами. Нами были разработаны два интерфейса: первый вариант, предусматривающий возможность манипулирования блоков и связей между ними (рис. 1), и второй, предназначенный для создания ПИД-регуляторов (рис. 2).

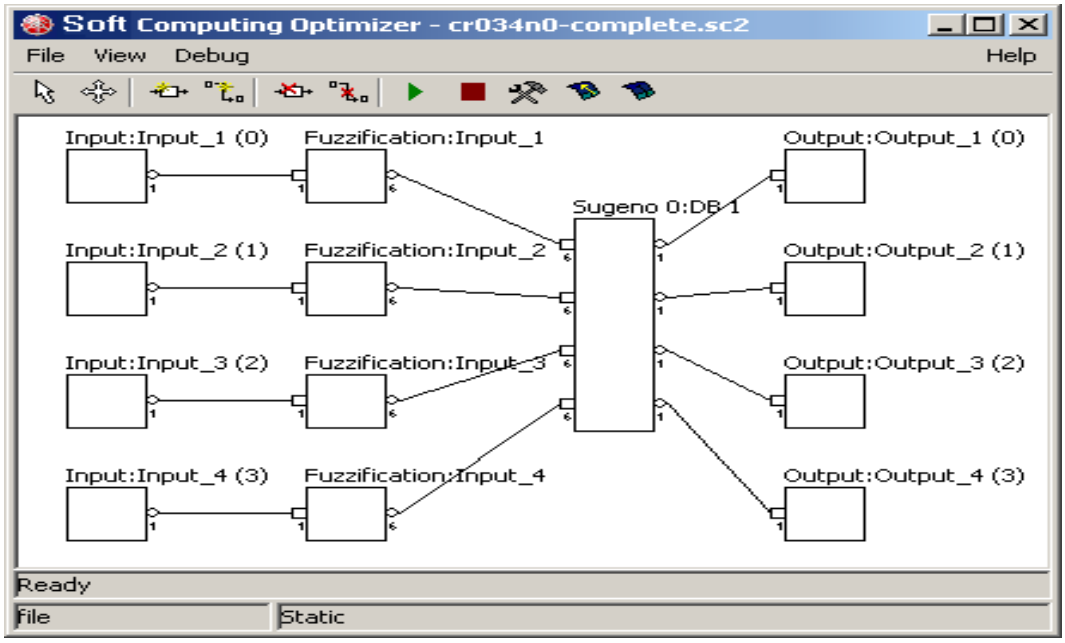

Рис. 1. Универсальный интерфейс ОБЗ

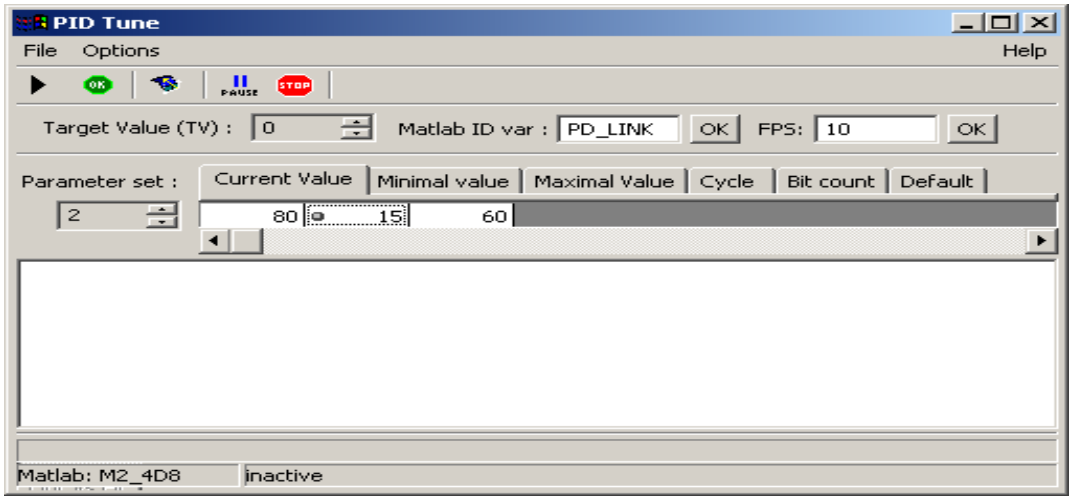

Рис. 2. Интерфейс оптимизатора ПИД-регуляторов

Во втором случае пользовательский интерфейс полностью скрывает от пользователя использование блочной структуры модели. Этот интерфейс самостоятельно обеспечивает создание модели с необходимым числом ПИД-регуляторов, конфигурирование входов и выходов и установление необходимых связей между блоками. Пользователю предоставляется непосредственный доступ к управлению коэффициентами контроллеров и пространством поиска для оптимизации.

3. Пример использования оптимизатора баз знаний

Рассмотрим результаты моделирования структур ИСУ с применением ОБЗ.

3.1. Система «перевернутый маятник – каретка перемещения»

Движение динамической системы «перевернутый маятник – каретка перемещения», физическая модель которой показана на рисунке 3, описывается следующими уравнениями:

$$\ddot{\theta} = \frac{g\sin\theta + \cos\theta\left(\frac{(u+\xi(t)) + \{a_1\dot{z} + a_3 z\} - ml\dot{\theta}^2\sin\theta}{m_c + m}\right) - k\dot{\theta}}{l\left(\frac{4}{3} - \frac{m\cos^2\theta}{m_c + m}\right)}, \qquad (3.1)$$

$$\ddot{z} = \frac{u + \xi(t) + \{-a_1\dot{z} - a_2 z\} + ml(\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta)}{m_c + m},$$

где g – ускорение свободного падения ($9.8\,m/\sec^2$), m_c – масса каретки, m – масса маятника, l – половина длины маятника, $\xi(t)$ – стохастическое воздействие и u – управляющая сила, действующая на каретку.

Уравнения для скорости производства энтропии в ОУ и ПИД-регуляторе имеют следующий вид:

$$\frac{d}{dt}S_\theta = \frac{k\dot{\theta}^2 + \frac{ml\dot{\theta}^3\sin 2\theta}{m_c + m}}{l\left(\frac{4}{3} - \frac{m\cos^2\theta}{m_c + m}\right)}; \quad \frac{d}{dt}S_z = a_1\dot{z}^2; \frac{d}{dt}S_u = k_d\dot{e}^2 \qquad (3.2)$$

соответственно.

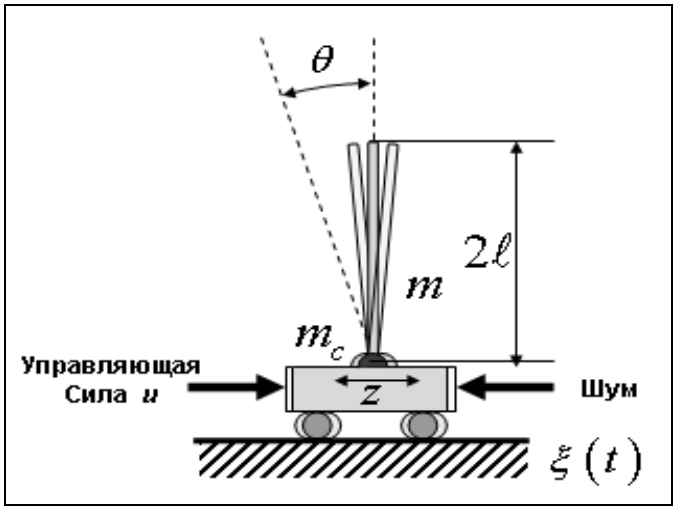

Рис. 3. Физическая модель ОУ

Рассмотрим следующую задачу управления движением перевернутого маятника (ОУ).

Задача управления: При наличии Рэлеевского стохастического шума (с несимметричной функцией распределения плотности вероятности), действующего на каретку, и при наличии времени задержки сигнала в системе измерения положения маятника (равного 0.001 сек.), перевести маятник из начального положения в целевое вертикальное ($\theta = 0$) и удерживать движение ОУ в заданном вертикальном положении. Зададим следующие значения параметров:

$$m_c = 1; m = 0.1; l = 0.5; k = 0.4; a_1 = 0.1; a_2 = 5$$

и начального положения $[\theta_0; \dot{\theta}_0; z_0; \dot{z}_0] = [10; 0.1; 0; 0]$. Введем также ограничение на силу управления: $-5.0 < u < 5.0[N]$.

При наличии действующих на каретку стохастических шумов и при наличии времени задержки сигнала в системе измерения положения маятника, традиционный ПИД-регулятор (с постоянными коэффициентами) также как и в предыдущем примере не всегда справляется с вышеуказанной задачей управления (как, например, в ситуации, показанной на рис. 4). При значениях коэффициентов усиления ПИД-регулятора $K = [50\ 50\ 50]$ маятник глобально неустойчив.

Примечание. Кроме того, определить вручную оптимальные коэффициенты усиления ПИД-регулятора для поддержания маятника в вертикальном положении, является сложной комбинаторной задачей и не всегда под силу даже опытному проектировщику САУ. Попытки решить данную задачу средствами интеллектуальных НР со скользящими режимами [18] приводили к усложнённым законам управления и не включали важные вопросы оценки робастности ИСУ.

Рассмотрим ИСУ, содержащего нечеткий ПИД-регулятор, управляющий движением маятника, с использованием разработанного инструментария ОБЗ. Прототипами физической модели рассматриваемой динамической системы (3.1) являются мотоцикл с учётом биомеханических характеристик водителя или её обобщение на трёхмерный случай система «одноколёсный робот – велосипед».

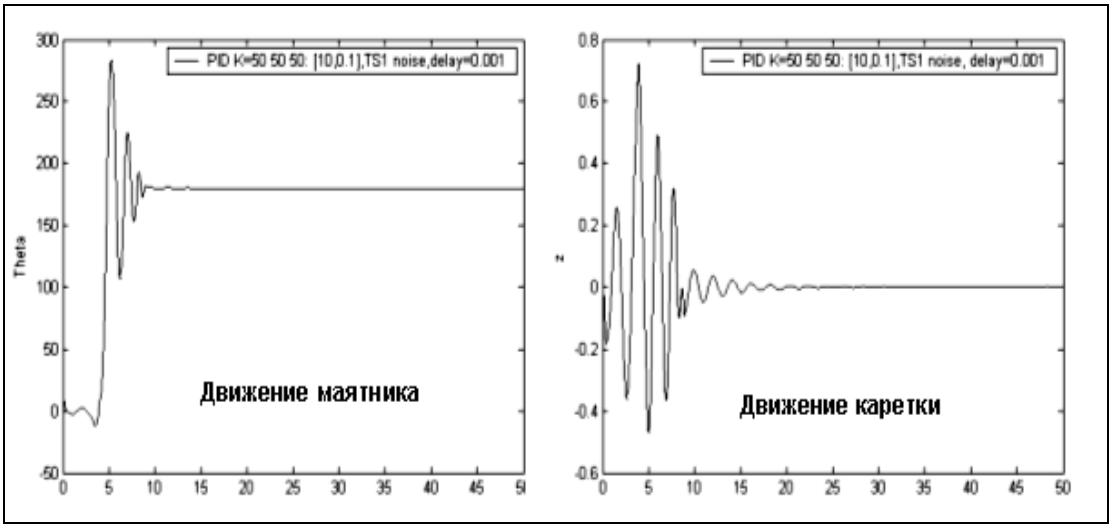

Рис. 4. Результаты моделирования управления ОУ на основе ПИД-регулятора

На первом шаге разработки ИСУ строится БЗ для заданной ситуации управления (т.н. обучающая ситуация), затем на следующем шаге проверяется робастность полученной БЗ в новых (непредвиденных) ситуациях управления.

На рисунках 5, 6, 7, и 8 показаны результаты моделирования движения системы в двух случаях управления: (1) с помощью НР, БЗ которого была построена с использованием разработанного ОБЗ; и (2) с помощью классического ПИД-регулятора с коэффициентами усиления $K = [80\ 15\ 60]$, полученными усреднением коэффициентов усиления НР.

Примечание. На рисунках 5, 6, 7, и 8 использованы следующие обозначения: X – pendulum angle – угол отклонения маятника; Z – cart position – положение каретки; control error – ошибка управления; control force – управляющая сила; S_P – производство энтропии в ОУ; S_C – производство энтропии в регуляторе; K_P, K_D, K_L – коэффициенты усиления ПИД-регулятора, соответственно.

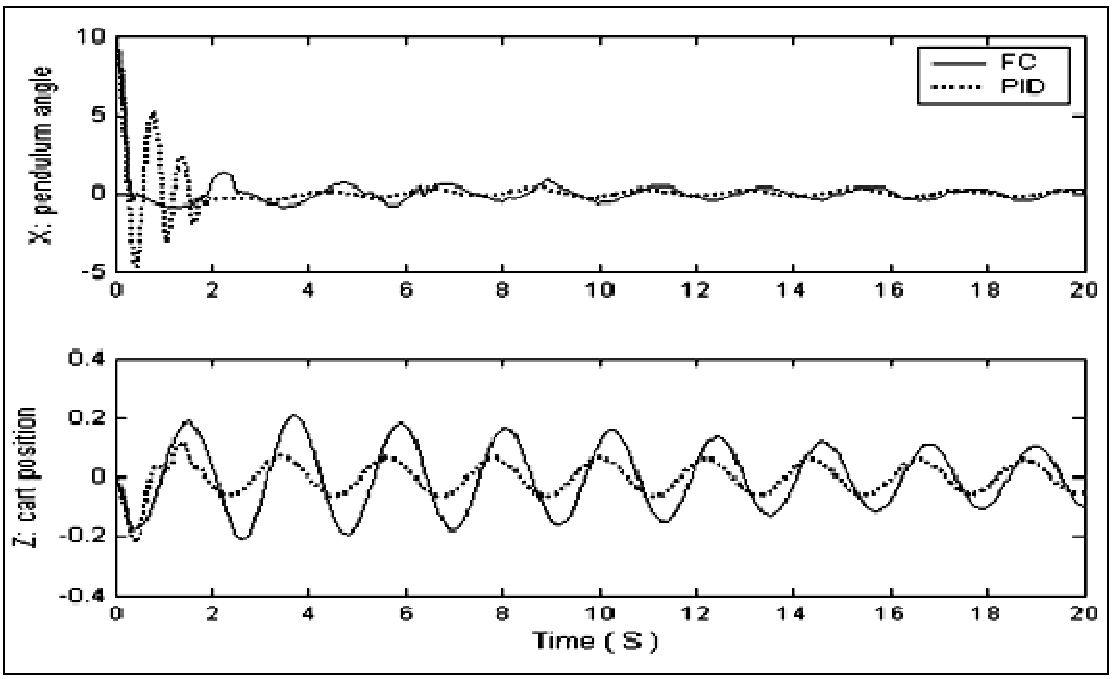

Рис. 5. Сравнение результатов моделирования управления ОУ с помощью НР (FC) и традиционного ПИД (PID)-регулятора

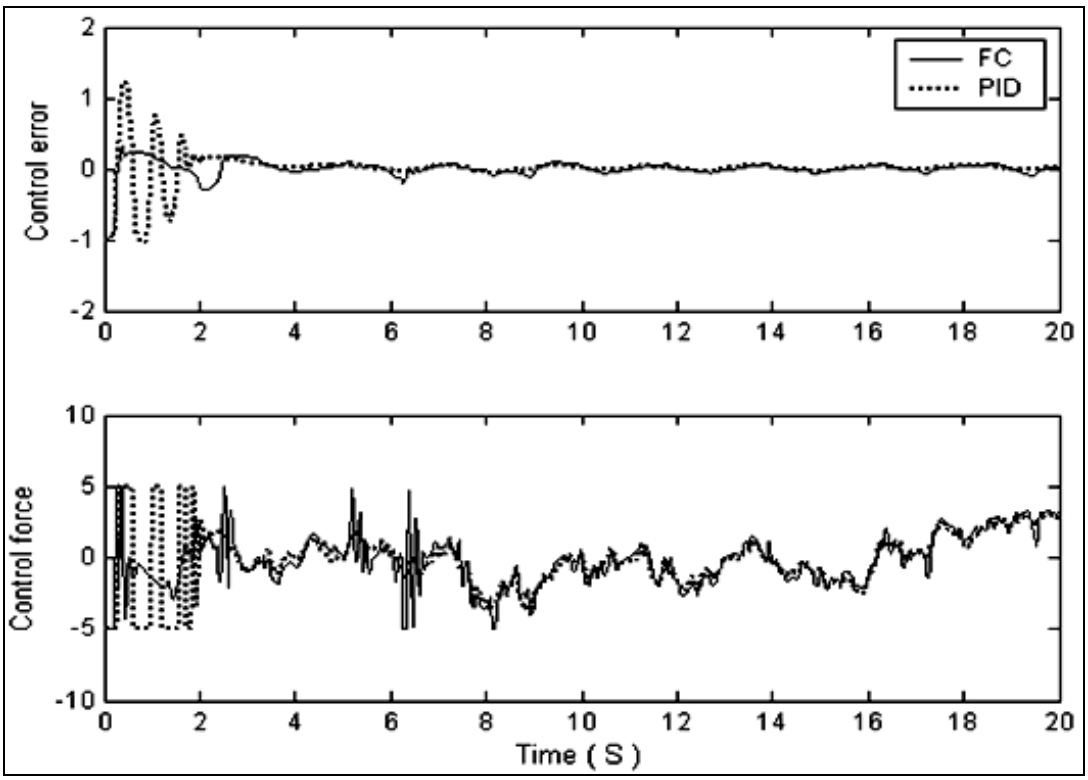

Рис. 6. Сравнение результатов моделирования управления ОУ с помощью НР (FC) и традиционного ПИД (PID)-регулятора

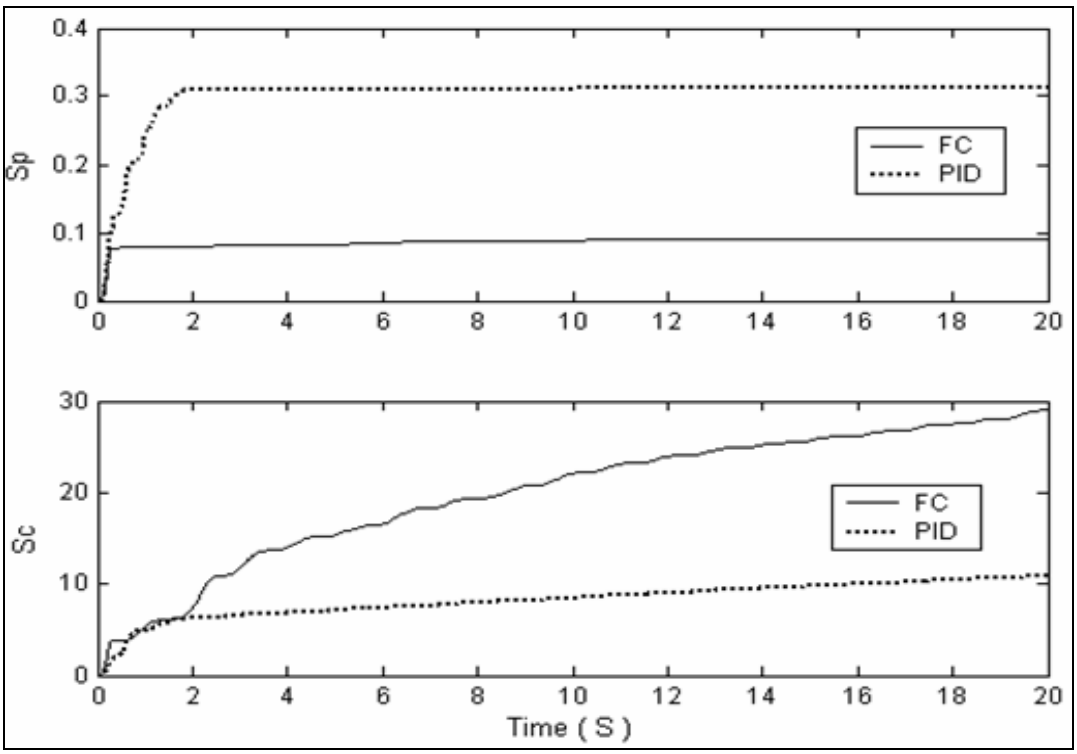

Рис. 7. Сравнение результатов моделирования производства энтропии в ОУ, НР (FC) и в традиционном ПИД (PID)-регулятора

Результаты моделирования показывают следующее. С точки зрения оптимизации САУ по критериям качества управления, таким как: минимум ошибки управления; минимум производства энтропии в объекте управления (т.е. минимум тепловых потерь, потерь полезной работы и энергии), а также с учетом ограничений на управляющую силу – ИСУ, разработанная на основе ОБЗ, является эффективнее традиционных ПИД-регуляторов.

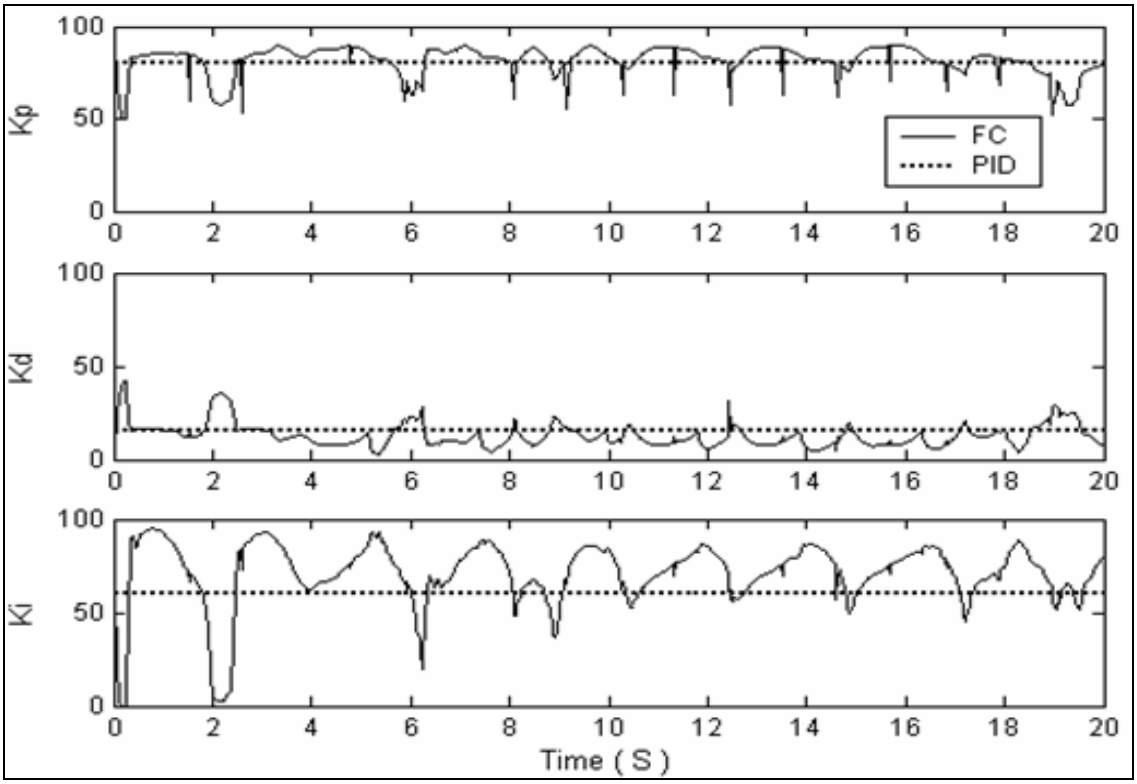

Рис. 8. Законы управления коэффициентами усиления нечёткого и усреднённого ПИД (PID)-регулятора

Выводы

Рассмотренная в статье архитектура ОБЗ позволила создать инструментарий, позволяющий проектировать ИСУ сложной конфигурации. На основе предложенного инструментария ОБЗ могут быть рассмотрены актуальные задачи формирования БЗ для проектирования робастных НР, например задача координационного управления коэффициентами усиления двух ПИД-регуляторов, представляющая самостоятельный интерес для теории и систем управления. Использование инструментария ОБЗ позволяет одновременно реализовать процесс проектирования робастных БЗ на основе алгоритмов обучения и адаптации.

Список литературы

1. Кураваки И., Литвинцева Л.В., Язенин А.В. и др. Построение робастных баз знаний нечётких регуляторов для интеллектуального управления существенно-нелинейными динамическими системами. I // Изв. РАН, ТиСУ, 2004. – № 4.
2. Петров Б.Н., Гольденблат И.И., Ульянов С.В. и др. Теория моделей в процессах управления: Информационные и термодинамические аспекты. – М.: Наука, 1978.
3. Litvintseva L.V., Takahashi K., Ulyanov S.S. et all. Intelligent robust control design based on new types of computations // Note del Polo Ricerca, Universita degli Studi di Milano Publ. – 2004. – Vol. 60.
4. Петров Б.Н., Уланов Г.М., Ульянов С.В. и др. Проблемы управления релятивистскими и квантовыми динамическими системами. – М.: Наука, 1982.
5. Алиев Р.А., Ульянов С.В. Нечёткие модели процессов и систем управления // Итоги науки и техники (ВИНИТИ). – Сер. Техн. кибернетика. – 1990. – Т. 29. – 1991. – Т. 32.
6. Petrov B.N., Pugachev V.S., Ulyanov S.V. et all. Informational foundations of qualitative theory of control systems // Proc. 7th IFAC. Helsinki, Finland. – 1978. – Vol. 3.
7. Ульянов С.В., Язенин А.В., Такахаши К. и др. Моделирование и проектирование интеллектуальных робастных систем управления с использованием квантовых и мягких вычислений // Тез. докл. конф. по теории управления, посвященной памяти академика Б.Н. Петрова. – М.: ИПУ РАН, 2003.
8. Ulyanov S.V. Self-organized control system // US patent. – 1997. – № 6,411,944 B1.
9. Захаров В.Н., Ульянов С.В. Промышленные нечёткие системы управления и нечёткие регуляторы. Ч.1. Научно-организационные и технико-экономические аспекты // Изв. АН СССР. – Техн. кибернетика, 1992. – № 5.
10. Ulyanov S.V., Yamafuji K., Kurawaki I. et all. Computational intelligence for robust control algorithms of complex dynamic systems with minimum entropy production. Pt 1 // J. Advanced Computational Intelligence and Intelligent Informatics. – 1999. – Vol. 3. – № 2.
11. Панфилов С.А., Литвинцева Л.В., Ульянов С.С. и др. Программная поддержка процессов формирования, извлечения и проектирования баз знаний робастных интеллектуальных систем управления // ППС. – 2004. – № 2.
12. Panfilov S.A., Litvintseva L.V., Ulyanov S.V. et all. Soft computing optimizer of intelligent control system structures // US patent. – 2005. – № 20050119986.
13. Panfilov S.A., Litvintseva L.V., Ulyanov S.V. et all. Soft computing optimizer for intelligent control systems design: the structure and applications // J. Systemics, Cybernetics and Informatics (USA). – 2003. – Vol. 1. – № 5.
14. Deb K., Pratap A., Agarwal S. et all. A fast and elitist multi-objective genetic algorithm: NSGA-II // IEEE Trans. on Evolutionary Computation. – 2002. – Vol. 6. – № 2. – Pp. 182-197.
15. Ohwi J., Ulyanov S.V., Yamafuji K. GA in continuous space and fuzzy classifier system for opening door with manipulator of mobile robot: New benchmark of evolutionary intelligent computing // J. of Robotics and Mechatronics. – 1996. – Vol. 8. – № 2. – Pp. 297-301.

16. Hiroshiya T., Nakayama S., Miki M. Comparison study of SPEA2+, SPEA2, and NSGA-II in diesel engine emissions and fuel economy problems // The IEEE Congress on Evolutionary Computation, – 2005. – Vol. 1. – Pp. 236-242.

17. Ulyanov S.V System and method for stochastic simulation of nonlinear dynamic systems with a high degree of freedom for soft computing applications // US patent. – 2004. – № 2004/0039555 A1.

18. Tao C.W., Taurb J.S., Wang C.M. Fuzzy hierarchical swing-up and sliding position controller for the inverted pendulum–cart system // Fuzzy Sets and Systems. – 2008. – Vol. 159. – № 2. – Pp. 2763-2784.