$$A_{qi} = \int_0^\infty D^i\{x^\circ(t)\}\, e^{-\rho_q t} dt = A_q \rho_q^{i-1}, \quad i = 0,1,\ldots,n,$$

$$B_{qi} = \int_0^\infty D^i\{F[x^\circ(t)]\}\, e^{-\rho_q t} dt = B_q \rho_q^{i-1}, \quad i = 0,1,\ldots,u,$$

$$C_{qi} = \int_0^\infty D^i\{f(t)\}\, e^{-\rho_q t} = C_q \rho_q^{i-1} dt, \quad i = 0,1,\ldots,v, \qquad (2)$$

здесь $x^0(t)$ – желаемое программное движение [1, 2], $F[x^0(t)]$ – нелинейная характеристика, $f(t)$ – внешнее входное воздействие; $e^{-\rho_q t}$ – система из m непрерывно дифференцируемых линейно-независимых координатных функций, представляющих собой полную систему функций.

Как следует из соотношений (1) и (2) для распространения обобщенного метода Галеркина на многосвязные системы с аппроксимацией нелинейных характеристик аналитическими или иррациональными функциями необходимо определить интегралы B_{qi}.

Данные соотношения были определены [7 – 9] для программного движения вида

$$x^0(t) = \left(x_y + H^* e^{-\alpha t}\right) 1(t) \quad ,$$

где x_y – значение желаемого процесса $x^0(t)$ при $t = \infty$; $H^* = x_0 - x_y$; x_0 – начальное значения желаемого процесса в момент времени $t = +0$, а показатель затухания процесса α, определяется исходя из соотношения

$$\alpha = \frac{3 \div 4}{T_{\text{п.п.}}} \quad ,$$

здесь $T_{\text{п.п.}}$ – время переходного процесса.

Таким образом, полученные рекуррентные соотношения [7 – 9], дают возможность полностью алгебраизировать решение задачи синтеза параметров законов управления нелинейных непрерывных многосвязных САУ, динамика которых описывается дифференциальными уравнениями произвольно высокого порядка и содержащих нелинейные элементы, характеристики которых целесообразно аппроксимировать аналитическими и иррациональными функциями.

В ходе решения поставленной задачи обобщенный метод Галеркина был распространен на решение задачи синтеза непрерывных многосвязных систем автоматического управления при аппроксимации нелинейных характеристик аналитическими и иррациональными функциями. Показаны преимущества данных видов аппроксимации по сравнению с кусочно-линейной, при использовании обобщенного метода Галеркина в качестве математического аппарата для решения задачи синтеза параметров регуляторов САУ.

Литература

1. Шишлаков В. Ф. Синтез нелинейных САУ с различными видами модуляции: Монография. СПб.: СПбГУАП, 1999. 268с

2. Никитин А. В., Шишлаков В. Ф. Параметрический синтез нелинейных систем автоматического управления: Монография / Под. Ред. В. Ф. Шишлакова. СПб. СПбГУАП, 2003. 358с

3. Шишлаков В. Ф., Шишлаков Д. В. Параметрический синтез многосвязных систем автоматического управления обобщенным методом Галеркина // Информационно-управляющие системы. 2006. № 3. С. 51-62.

4. Шишлаков В. Ф., Шишлаков Д. В. Параметрический синтез многосвязных систем автоматического управления во временной области // изв. Вузов сер. Проблемы энергетики. 2006. № 12. С.49-54.

5. Цветков С. А., Шишлаков В. Ф., Шишлаков Д. В. Синтез многосвязных систем автоматического управления во временной области // изв. Вузов сер. Приборостроение. 2007. №12. С.13-17.

6. Шишлаков В.Ф., Цветков С.А., Шишлаков Д.В. Синтез параметров непрерывных и импульсных многосвязных систем автоматического управления: Монография. Под ред. В.Ф. Шишлакова, ГУАП, СПб, 2008, 180с.

7. Турубанов М.А., Шишлаков В.Ф., Шишлаков А.В. Импульсная система управления комбинированной солнечно- и ветроэнергетической установкой со сверхпроводниковым оборудованием // Информационно-управляющие системы. 2014. 33. С.8 – 18.

8. Чубраева Л.И., Шишлаков А.В. Синтез электромеханических систем автоматического управления при аналитической аппроксимации характеристик нелинейных элементов // информационно-управляющие системы. 2014. №2. С.2 – 7.

9. Шишлаков В.Ф., Анисимова Е.В., Шишлаков А.В., Шишлаков Д.В. Синтез параметров закона управления для нелинейных САУ при различных видах аппроксимации характеристик // изв. Вузов сер. Приборостроение. 2015. Т.58, №9. С.701-706.

Л.В. Яблокова

ИСПОЛЬЗОВАНИЕ ОБЪЕКТНО-ОРИЕНТИРОВАННОЙ ПАРАДИГМЫ ПРИ МАТЕМАТИЧЕСКИХ РАСЧЕТАХ В СРЕДЕ MATLAB

(Самарский национальный исследовательский университет
имени академика С.П. Королёва)

Информатика, как технологическая наука, должна по определению включать прагматику, а это, в свою очередь, должно предполагать постоянное сравнение ее составляющих. Как известно, количественной мерой любого результата является оценка, основанная на соответствующих критериях и нормах, которые действуют в рамках размерности, определяемой природой их ценности. А прагматическая природа информатики предполагает, что остается и используется только самое эффективное. Все остальное либо выбраковывается, либо переосмысливается и, в конечном счете, обновляется. Чего же мы ждем от наших программ предназначенных для научных исследований, чего хотим от конкретной реализации в частности? Мы хотим, чтобы программа работала правильно и надежно, но кроме этого, нам нужно, чтобы она была легко адаптируема к но-

вым условиям, которые часто меняются в ходе решения той или иной научной проблемы. Иными словами мы должны учитывать требования к исходному коду программ, чтобы он был удобным в сопровождении. Сложность может быть присуща как самой научной проблеме, так и всей предметной области, в рамках которой должно проводиться исследование. Но она также может быть и следствием неудачно выбранного способа или стиля изложения идей заложенных в программе. Неправильно выбранный инструментарий или стратегия конструирования приводят к проблемам, которые, к сожалению, могут быть не сразу выявлены на этапе проектирования, а проявят себя уже после начала использования. В итоге, труд программиста превращается в то, что можно назвать плохо спроектированным и не элегантным, и как следствие, имеющим очень ограниченную область применения. Такие программы, как правило, имеют очень короткий жизненный цикл и их проще заменить новыми, чем пытаться повторно использовать, адаптировать или совершенствовать. Одним из критериев качества программы является то, насколько внимательно разработчик отнесся к процессу формализации типичных взаимодействий между участвующими в ней компонентами. Если на этапе проектирования таким механизмам уделить достаточное внимание, то архитектура программы получится более компактной, простой и понятной, а код более структурированным и адаптируемым к вновь специфицируемым требованиям предметной области.

Механизм внедрения зависимостей (Dependency Injection) – это набор принципов и паттернов проектирования программного обеспечения, которые дают возможность разрабатывать слабосвязанный код.

```
classdef(Abstract = true) SolutionImpl < handle
    methods(Abstract = true)
        solve(this, solver);
    end
end
```

Рис. 1. Интерфейс внедряемой зависимости


```
classdef MaxwellSolution < SolutionImpl & SolutionParams
    properties(Constant = true)
        Mu_0 = 1.2566370614e-12;
        Epsilon_0 = 8.8541878174e-18;
    end
    properties(Dependent = true, SetAccess = private)
        Lz;
        Nt;
        Nz;
        Ht;
        Hz;
        C1;
        C4;
        C5;
    end
    methods
        function this = MaxwellSolution(q, qt, qT, lambda)
            %Initialization of protected class members
        end
        function solve(this, solver)
            solver.solveMaxwell(this);
        end
        %Property accessor methods
        %       .
        %       .
        %       .
    end
end
```

Рис. 2. Реализация внедряемой зависимости для уравнений Максвелла

```
classdef D_AlembertSolution < SolutionImpl & SolutionParams
    properties(Dependent = true, SetAccess = private)
        Lz;
        Nt;
        Nz;
        Ht;
        Hz;
        C1;
        C5;
    end
    methods
        function this = D_AlembertSolution(q, qt, qT, lambda)
            %Initialization of protected class members
        end
        function solve(this, solver)
            solver.solveD_Alembert(this);
        end
        %Property accessor methods
        %       .
        %       .
        %       .
    end
end
```

Рис. 3. Реализация внедряемой зависимости для уравнения д’Аламбера

```
classdef(Abstract = true) SolutionParams < handle
    properties(Constant = true)
        Lv = 2.99792458e14;
    end
    properties(SetAccess = protected)
        Q;
        Qt;
        QT;
        Lambda;
    end
end
```

Рис. 4. Абстрактный класс параметров решения

```
classdef Solution < handle
    properties(Access = private)
        impl_;
    end
    methods
        function this = Solution(impl)
            this.impl_ = impl;
        end
        function accept(this, solver)
            this.impl_.accept(solver);
        end
    end
end
```

Рис. 5. Внедрение зависимости через конструктор

Двойная диспетчеризация (Double Dispatch) – это способ организации кода на базе паттерна Посетитель (Visitor), позволяющий расширить интерфейс базового класса путем создания отдельной иерархии для виртуализации операций.

```
classdef(Abstract = true) Solver < handle
    methods(Abstract = true)
        solveMaxwell(this, solution);
        solveD_Alembert(this, solution);
    end
end
```

Рис. 6. Интерфейс решателя

Абстракция – это важнейшая часть инструментария при проектировании приложений для научных исследований. Справиться с постоянным ростом сложности исследовательских компьютерных систем без нее невозможно. Разные парадигмы подразумевают собственные принципы абстракции, общие для множества методик разработки. Каждая методика – это свой способ группировки абстрактных представлений в соответствии с их общими свойствами, т.е. отдельная техника абстрагирования. А хорошая техника абстрагирования, учитывая сложность программного обеспечения и реального мира, помогает упорядочивать знания и синтезировать более качественные решения на основе проведенного анализа.


```
classdef MaxwellSolver < Solver
    methods
        function solveMaxwell(~, solution)
            amp = exp(0.0 * 1i);
            src = fix(solution.Nz / 2);
            ex = zeros(1, solution.Nz);
            hy = zeros(1, solution.Nz - 1);
            %Solving Maxwell equations
            %        .
            %        .
            %        .
        end
        function solveD_Alembert(~, solution)
            error('Invlid operation for solution %s with solver %s',...
            class(solution), 'MaxwellSolver');
        end
    end
end
```

Рис. 7. Реализация решателя уравнений Максвелла

```
classdef D_AlembertSolver < Solver
    methods
        function solveMaxwell(~, solution)
            error('Invlid operation for solution %s with solver %s',...
            class(solution), 'D_AlembertSolver');
        end
        function solveD_Alembert(~, solution)
            a = exp(0.0 * 1i);
            ex = zeros(1, solution.Nz);
            extt = zeros(1, solution.Nz);
            exttt = zeros(1, solution.Nz);
            rad_locat = solution.Nz - solution.Q;
            %Solving D_Alembert equation
            %        .
            %        .
            %        .
        end
    end
end
```

Рис. 8. Реализация решателя уравнения д’Аламбера

Литература

1. Фаулер, М. Рефакторинг. Улучшение существующего кода / Мартин Фаулер – Санкт-Петербург: Символ-Плюс, 2003. - 432 с.

2. Макконнелл, С. Совершенный код / Стив Макконнелл – М.: Питер, 2007. - 893 с.

3. Гамма, Э. Приемы объектно-ориентированного проектирования / Э. Гамма, Р. Хелл, Р. Джонсон, Дж. Влиссидес – М.: Питер, 2006. – 368 с.

4. Головашкин Д. Л. Совместное разностное решение уравнений Даламбера и Максвелла. Одномерный случай / Д.Л. Головашкин, Л.В. Яблокова // Компьютерная оптика. - 2012. - Т. 36, №4. - с. 527-534.

5. Golovashkin D.L., Long vectors algorithms for solving grid equations of explicit difference schemes / D.G Vorotnikova, D.L. Golovashkin // Computer Optics. - 2015. - Vol. 39(1). - P. 87-93.

6. http://www.mathworks.com/help/matlab/index.html (дата обращения: 12.03.16).