САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ ПЕТРА ВЕЛИКОГО
ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ
Кафедра «Системный анализ и управление»

Практическая работа №2

Дисциплина "Объектно-ориентированный подход в системном анализе и управлении"

Тема: «Синтез оптимального управления с использованием метода Бойчука»

Работу выполнил
студент гр. 43502/2 Гончаров А.Н.

Работу принял
к. т. н., доцент кафедры САиУ Болотин И.В.

Санкт-Петербург
2015

Оглавление

1 Постановка задачи

Пусть система автоматического регулирования, которая описывается нелинейным дифференциальным уравнением первого порядка:

$$u(t) = a_0 \frac{dx(t)}{dt} + e^{-\lambda t} x(t), \tag{1.1}$$

где $u(t)$ - управляющее воздействие, $x(t)$ - регулируемая величина, a_0 и λ - параметры описываемой системы. Обозначим $\psi = \psi(t)$ - задающее воздействие, $\psi_0 = \psi_0(t)$, граничное условие $x(t_0) = x_0$.

Задача: найти оптимальное управление, которое обеспечивает минимум интегральному критерию: $I = \int_0^{+\infty} \left(\varphi^2 + A\left(\varphi^{(1)}\right)^2\right) dt$, $\varphi^{(i)} = \psi^{(i)} + \varphi^{(i)}$ - ошибки координат системы, причем $\varphi^{(0)} = \varphi$, $\varphi^{(1)} = \frac{d\varphi}{dt}$. Рассматриваемый промежуток времени $[0, \infty]$.

2 Теоретическая часть

2.1 Синтез оптимального управления

Необходимое условие оптимального управления - выполнение уравнения Эйлера-Пуассона:

$$\frac{d}{dt}\frac{\partial F}{\partial \dot{\varphi}^{(i)}} - \frac{\partial F}{\partial \varphi^{(i)}} = 0, \qquad i \in [0,1] \Longrightarrow A\varphi^{(2)} - \varphi = 0 \tag{2.1}$$

Решение этого дифференциального уравнения с учётом граничных условий $\varphi(t_0) = \varphi_0, \varphi(+\infty) = 0$:

$$\varphi(t) = \varphi_0 e^{-\frac{t}{\tau}}, \tag{2.2}$$

где $\tau = \sqrt{A}$.

График переходного процесса $\varphi(t)$ является экспонентой с постоянной времени $\tau = \sqrt{A}$ $(A \neq 0)$, что практически невозможно.

Поскольку верхний предел интегрирования критерия оптимальности $T \to \infty$, а значение координат системы при этом равно нулю $\varphi^{(i)}(\infty) = 0$, то дифференциальное уравнение экстремали ($\sum_{i=0}^{n} C_i \varphi^{(i)} = 0, \quad i \in [0,1]$):

$$C_0\varphi + C_1\varphi^{(1)} = 0, \tag{2.3}$$

тогда: $(C_0 + C_1 r)(C_0 - C_1 r) = 1 - Ar^2$, отсюда $C_0 = 1, C_1 = \sqrt{A}$.

Уравнение (2.3):

$$1 * \varphi + \sqrt{A} * (\psi - x)^{(1)} = 0, \tag{2.4}$$

Отсюда можно получить значение высшей производной:

$$x^{(1)} = \psi^{(1)} + \frac{1}{\sqrt{A}}(\psi - x), \tag{2.5}$$

Тогда оптимальное управление с учётом (2.5):

$$u(t) = a_0\left(\psi^{(1)}(t) + \frac{1}{\sqrt{A}}(\psi(t) - x(t))\right) + e^{-\lambda t}x(t) \tag{2.6}$$

В данной оптимальной системе первого порядка требуется измерять задание и его первую производную, текущее положение координаты системы $x(t)$.

2.2 Структура построения проекта с использованием ООП

Развитие вычислительной техники привело к тому, что решение прикладных задач не обходится без использования ЭВМ. Рассмотрим пример построения структуры проекта с использованием списочных структур. Решение математической модели на ЭВМ имеет определенные этапы. Классифицируем эти этапы следующим образом. Синтез математической модели, метод решения, решение математической модели и форматирование результатов.

Структуру решения задачи определим как связанный список, содержащий этапы решения задачи как узлы списка.

Дадим определение линейного связанного списка.

Линейный список — это множество, состоящее из $n \geq 0$ узлов $\{X[1], X[2], \ldots, X[n]\}$, свойства которого ограничиваются линейным (одномерным) относительным положением узлов, такими, что если $n > 0$, то $X[k]$ предшествует $X\ [k-1]$ и за ним следует $X\ [k+1]$; $X[n]$ является последним узлом.

Структура связанного списка приведена на рис. 2.1.

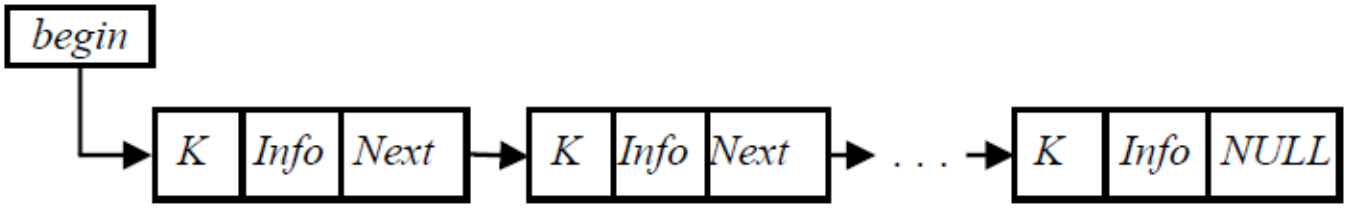

Рис. 2.1 - структура линейного связанного списка

Здесь begin — указатель на начало списка; K — поле ключа, дополнительное поле элемента списка, содержащее уникальное значение (например, номер элемента в списке); Info — поле данных; Next – указатель на следующий элемент списка; NULL - нулевой указатель (если отсутствует следующий элемент в списке).

При необходимости можно определить множество операций со списком. Например:

1. Получить доступ и k-му узлу списка, чтобы проанализировать и/или изменить содержимое его полей.
2. Включить новый узел непосредственно перед k-м узлом.
3. Исключить k-й узел.

4. Объединить два (или более) линейных списка в один список.
5. Разбить линейный список на два (или более).
6. Сделать копию линейного списка.
7. Определить сортировку узлов списка в возрастающем порядке по некоторым полям в узлах.
8. Найти в списке узел с заданным значением в некотором поле.

На рис. 2.2 приведена структура связного списка.

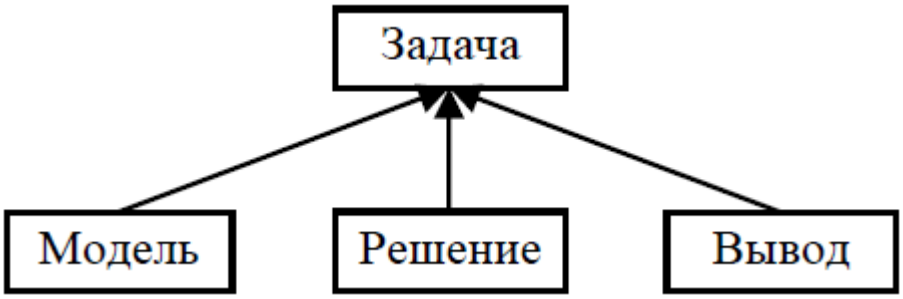

Рис. 2.2 - направленный ациклический граф наследования классов

В качестве базового класса определим класс «Задача», как абстрактный класс. Определим производные классы: «Модель», как класс, определяющий математическую модель задачи; класс «Решение», как класс, определяющий метод решения модели; класс «Вывод», как класс, определяющий вывод результатов полученного решения. С помощью класса «Список» будет формироваться связный линейный список элементами которого будут объекты разных классов, производные от абстрактного класса «Задача». Параметр конструктора класса «Список» должен содержать указатель на абстрактный базовый класс «Задача». В качестве фактических параметров будут использоваться ссылки (указатели) на конкретные объекты классов (производные от абстрактного класса). Т. е. в односвязный список включаются этапы решаемой задачи (Модель-Решение-Вывод).

3 Выполнение работы

Для проверки работы программы зададим параметры системы:

$$A = 7, a_0 = -3, \lambda = 3, x_0 = 2, \psi(t) = 1.$$

Код функции main:

```
int main()
{
	//Задание параметров модели
	double A = 7; //весовая константа
	double a0 = -3; //параметр модели a0
	double l = 3; //параметр модели лямбда
	double x0 = 2; //начальное значение x
	double psi = 1; //функция единичного возмущения
	//Задание шага и предела интегрирования
	double h = 0.1;
	double end = 50;

	Model model(A, a0, l, x0); //создание модели
	Task_list task_list(model); //создание односвязного списка на основе модели

	Solution solution(model, h, end, psi);//решение (включая интегрирование)
	task_list.push_front(solution); //добавление решения в список

	Output output(solution, "x.dat", "psi.dat", "u.dat", "t.dat"); //вывод значений в файлы
	task_list.push_front(output); //добавление вывода в список
	return 0;
}
```

Результаты вычислений выводятся в файлы, с помощью Matlab строятся графики (рис. 3.1-3):

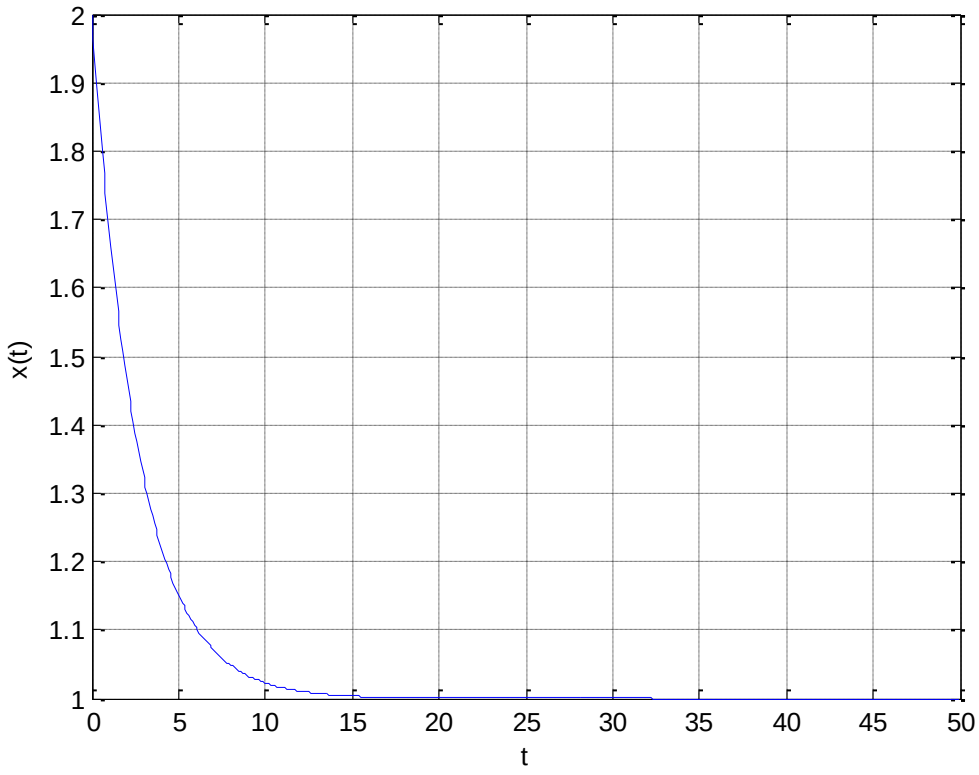

Рис. 3.1 – график регулируемой величины $x(t)$

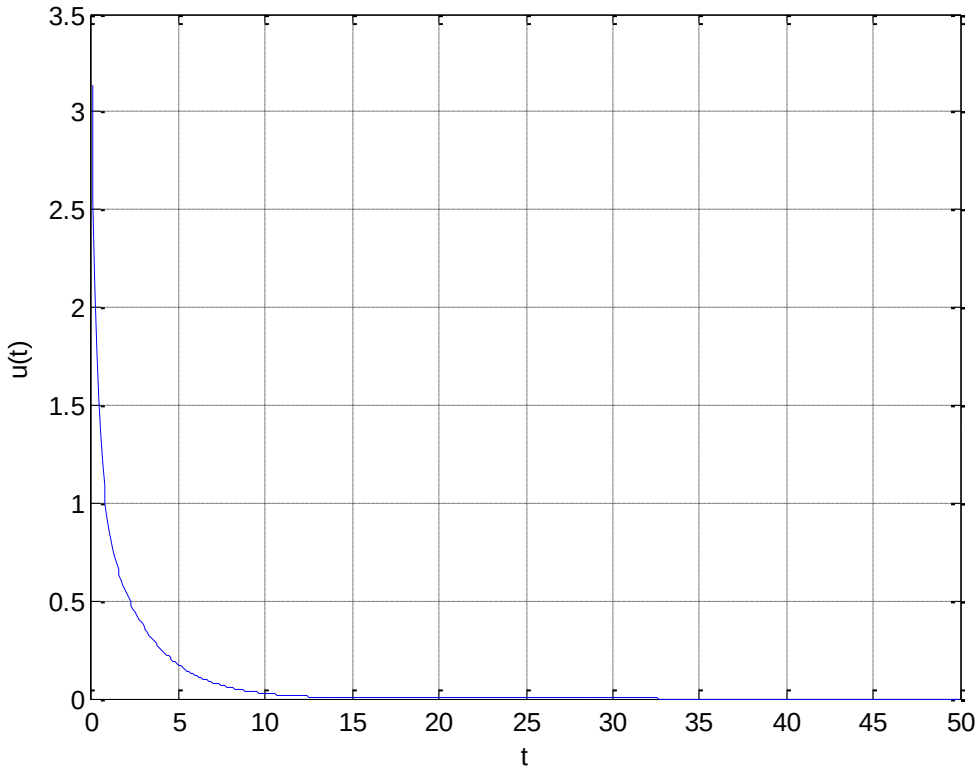

Рис. 3.2 – график управления $u(t)$

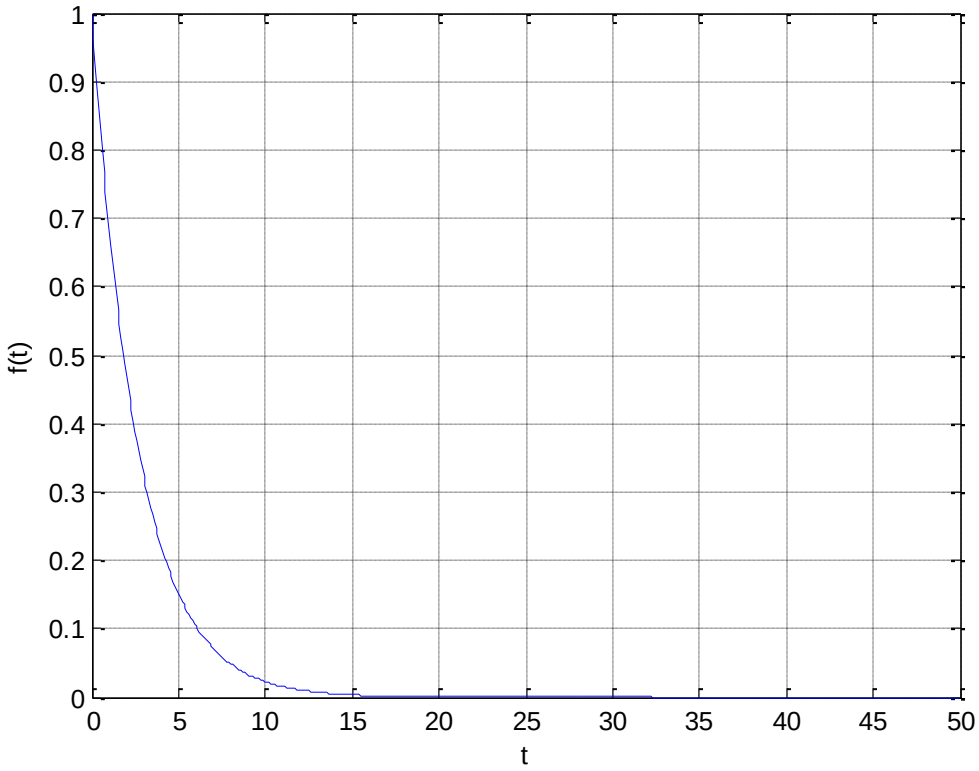

Рис. 3.3 – график ошибки $\varphi(t)$

Из последнего графика можно заметить, что ошибка становится нулевой, что говорит о корректности решения.

4 Программная реализация

4.1 Описание программы

Ниже на рисунке 4.1 приведена структура классов проекта. Класс *Task* является абстрактным, у него есть конструктор, виртуальный деструктор и чистый виртуальный (абстрактный) метод *work()*. От него наследуются 3 класса: *Model*, *Solution* и *Output*.

Класс *Model* в своём конструкторе принимает параметры модели и его экземпляр хранит параметры модели в своих приватных полях. Его метод *work()* ничего не делает. Для класса объявлен дружественный класс *Solution*.

Класс *Solution* в своём конструкторе принимает ссылку на объект класса *Model*, а также параметры интегрирования и значение функции возмущения ψ. Его приватный метод *F()* принимает параметр x и возвращает функцию высшей производной $x^{(1)} = \psi^{(1)} + \frac{1}{\sqrt{A}}(\psi - x)$. Его метод *work()* выполняет численное интегрирование функции *F()* методом Рунге-Кутты 4-го порядка, полученные значения $t, x(t), u(t)$ он хранит в своих приватных полях в стандартных контейнерах *vector<double>*. Для класса объявлен дружественный класс *Output*.

Класс *Output* в своём конструкторе принимает ссылку на объект класса *Solution*, а также имена файлов, в которые будут записаны результаты. Его метод *work()* Записывает в созданные файлы значения из контейнеров класса *Solution*.

Класс *Task_list* хранит ссылки на объекты абстрактного класса *Task* в виде связного списка. Его конструктор принимает как параметр указатель на абстрактный базовый класс *Task* и добавляет его в список в своём приватном поле. Его метод *push_forward()* также принимает указатель и также ставит его на первое место в списке. Таким образом в связном списке хранятся этапы задачи (Модель-Решение-Вывод).

Визуализация графиков происходит с помощью программы Matlab.

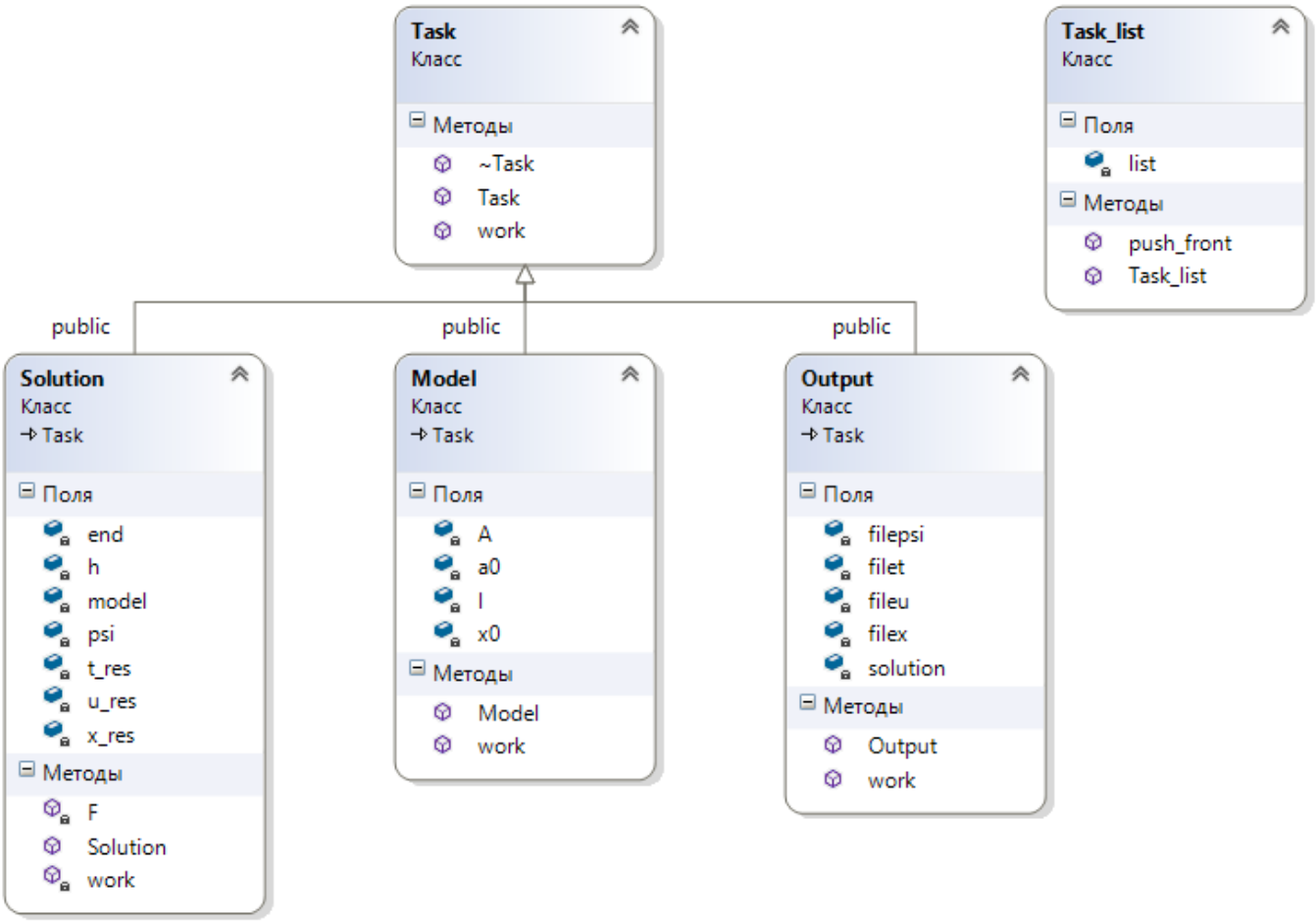

Рис. 4.1 – структура класса Model

4.2 Текст программы

Файл Classes.h

```
#include <string>
#include <vector>
#include <forward_list>
using namespace std;

class Task {//абстрактный базовый класс
public:
	Task(){}
	virtual ~Task(){}
	virtual void work() = 0;
};
class Solution;
class Output;

class Model : public Task {
public:
	Model(double A, double a0, double l, double x0) : A(A), a0(a0), l(l), x0(x0) {}
	void work(){};
	friend Solution;
private:
	double A, a0, l, x0; //параметры модели

};
```

```
class Solution : public Task {
public:
	Solution(Model &model, double h, double end, double psi) : model(&model), h(h),
end(end), psi(psi) {
		this->work();
	}
friend Output;

private:
	Model *model;
	double psi;
	double h;
	double end;
	//векторы результатов расчётов
	vector<double> u_res;
	vector<double> x_res;
	vector<double> t_res;

	double F(double x) { //подынтегральная функция
		return (psi - x) / sqrt(model->A);
	}
		void work() { //метод решения
			double x0 = model->x0;
			double a0 = model->a0;
			double l = model->l;
			double t = 0;
			double x_prev = x0;
			double x = x0;
			double u = a0 * F(x) + x;
			x_res.push_back(x);
			t_res.push_back(t);
			u_res.push_back(u);
			double k1 = 0;
			double k2 = 0;
			double k3 = 0;
			//интегрирование с помощью метода Рунге-Кутты 4-го порядка
			for (double i = 0; i < end; i += h){
				t = i;
				k1 = h * F(x);
				t += 0.5 * h;
				x = x_prev + 0.5*k1;
				k2 = h * F(x);
				x = x_prev + 0.5 * k2;
				k3 = h * F(x);
				t += 0.5 * h;
				x = x_prev + k3;
				x = x_prev + (k1 + 2 * k2 + 2 * k3 + h * F(x)) / 6.0;
				u = a0 * F(x) + exp(-l * t) * x;
				x_prev = x;
				u_res.push_back(u);
				x_res.push_back(x);
				t_res.push_back(t);
			}
		}
};

class Output : public Task {
private:
	Solution * solution;
	string filex, filepsi, fileu, filet;
public:
	Output(Solution &solution, string filex, string filepsi, string fileu, string filet) :
solution(&solution), filex(filex), filepsi(filepsi), fileu(fileu), filet(filet) {
		this->work();
	};
```

```
	void work() { //функция вывода данных в файлы
		FILE * File1;
		FILE * File2;
		FILE * File3;
		FILE * File4;
		//открытие файлов
		File1 = fopen(filex.c_str(), "w");
		File2 = fopen(filepsi.c_str(), "w");
		File3 = fopen(fileu.c_str(), "w");
		File4 = fopen(filet.c_str(), "w");
		//запись
		for (int i = 0; i < solution->end / solution->h; i++){
			fprintf(File1, "%+.5f\n", solution->x_res[i]);
			fprintf(File2, "%+.5f\n", solution->psi);
			fprintf(File3, "%+.5f\n", solution->u_res[i]);
			fprintf(File4, "%+.5f\n", solution->t_res[i]);
		}
		//закрытие файлов
		fclose(File1);
		fclose(File2);
		fclose(File3);
		fclose(File4);
	}
};

class Task_list {
public:
	Task_list(Task &task) {
		list.push_front(&task);
	}
	void push_front(Task &task){
		list.push_front(&task);
	}
private:
	forward_list <Task *> list;
};
```

Файл Main.cpp

```
#include "Classes.h"
int main()
{
	//Задание параметров модели
	double A = 7; //весовая константа
	double a0 = -3; //параметр модели a0
	double l = 3; //параметр модели лямбда
	double x0 = 2; //начальное значение x
	double psi = 1; //функция единичного возмущения
	//Задание шага и предела интегрирования
	double h = 0.1;
	double end = 50;

	Model model(A, a0, l, x0); //создание модели
	Task_list task_list(model); //создание односвязного списка на основе модели

	Solution solution(model, h, end, psi);//решение (включая интегрирование)
	task_list.push_front(solution); //добавление решения в список

	Output output(solution, "x.dat", "psi.dat", "u.dat", "t.dat"); //вывод рассчитаных
значений в файлы
	task_list.push_front(output); //добавление вывода в список

	return 0;
}
```

Вывод

В процессе выполнения задания было получено оптимальное управление для системы автоматического регулирования. Была написана программа на C++ с использованием ООП и получены графики для текущего положения системы $x(t)$, управления $u(t)$ и ошибки $\varphi(t)$. Анализ графиков показал, что ошибка стремится к 0, а значит система стабилизируется и расчёты произведены верно.

Список литературы

1. Болотин И. В. Системный анализ, оптимизация и принятие решений. Практикум: учебное пособие / Под ред. В. Н. Козлова. - СПб.: Издательство Политехнического университета, 2010. — 111 с.
2. Бойчук Л.М. Метод структурного синтеза нелинейных систем автоматического управления. – М.: Энергия, 1971. – 112 с.