эксплуатационных характеристик возводимых объектов. Внедрение классификаторов окажет влияние на строительную отрасль ввиду необходимости и целесообразности применения данной системы в крупных государственных проектах, где приходится обрабатывать большой объем информации, предоставленной проектными и строительными организациями в ходе реализации. Создание общего всероссийского классификатора всех возможных строительных элементов нам кажется не совсем целесообразной идеей. Возможно, необходимо создать верхние уровни классификации, а далее уже – дело рук отдельных компаний, занимающихся информационным моделированием.

1. Дмитриев, А. Современный опыт инновационного развития строительства на основе технологий информационного моделирования в России и за рубежом/ Дмитриев А., Тамбовцева Т., Папикян Л., Цыганкова А. // Недвижимость: экономика, управление – 2019 - №1 – С.104-108.
2. Постнов, К. В. Применение современных информационных технологий в проектных организациях и их влияние на повышение качества проектных решений/Постнов К. В. // Известия КГАСУ - 2014. - №4(30) – С. 375-383.
3. Талапов, В. В. Технология BIM: стандарты и классификаторы. [Электронный ресурс]/Талапов В. В. //isicad.ru. Ваше окно в мир САПР – 2015– №126.
4. Давыдов, Н. С. Анализ требований по выпуску сметной документации по BIM-модели и сравнение ведущих программных комплексов / Давыдов, Н. С., Придвижкин С.В., Белькевич А. В. , Кожевников К.И. // Вестник гражданских инженеров – 2018 - № 5 (70) - С. 201-208.
5. Каллаур, Г. Ю. Анализ затрат жизненного цикла в управлении проектами, реализуемыми с применением BIM-технологий/ Каллаур Г. Ю., Л. М. Папикян// Современные проблемы управления проектами в инвестиционно-строительной сфере и природопользовании — 2018 – С. 313-320.
6. Талапов, В. В. Анализ опыта внедрения информационного моделирования в Великобритании / Талапов В. В. // Баландинские чтения - 2019 - С. 89-93

Русакова З.Н.

Программный инструментальный C++ исследования функций

*МГТУ им. Н.Э. Баумана
(Россия, Москва)*

doi: 10.18411/lj-02-2020-25

idsp: ljjournal-02-2020-25

Аннотация

Статья посвящена вопросам проектирования программного инструментария C++ для решения задач исследования функций. Разработаны иерархии полиморфных классов, реализующих задачи определения интервалов нулей и экстремумов, вычисления корней и точек экстремумов на интервалах, которые можно рассматривать как первое приближение к проектированию паттерна исследования функций.

Ключевые слова: Классы, полиморфизм, интерфейс, указатели, корни функции, экстремумы, методы поиска корня

Abstract

The article is devoted to the issues of C ++ tools software designing for solving problems of functions research. Hierarchies of polymorphic classes, null and extreme research interval determination tasks, calculation of roots and extreme research points at intervals, which can be considered as a first approximation to the design of the function investigation pattern, have been developed.

Keywords: classes, polymorphism, interface, pointers, function roots, extremums, root search methods

В процессе исследования нелинейных уравнений выделяются два этапа: отделения корней нелинейных уравнений и экстремумов, уточнения корней и

экстремумов итерационными методами. Для решения задачи используется аналитический подход. В основе этого подхода лежит задача табуляции: вычисление таблицы значений функции в узлах равномерной сетки на заданном интервале в заданном числе узлов n . В случае равномерной сетки шаг между узлами постоянный, равный частному от деления величины интервала на число узлов.

Используя решение задачи табуляции, определяются интервалы монотонности, на которых функция проходит через нуль и меняет знак. Для выявления интервалов нулей, на которых функция изменяет знак, определяются начальная и конечная точки интервала x_i, x_{i+1} , для которых произведение значений функции в этих узлах меньше нуля.

Для выявления интервалов экстремумов определяются точки интервалов x_{i-1}, x_{i+1} из условия, что значение функции в i -ой точке по модулю больше значений в соседних точках: $(i-1)$ точкой и $(i+1)$ точкой. Уменьшая величину шага, можно уточнить интервалы.

На каждом из этих интервалов для поиска корня используются численные методы уточнения корней. Используются основные методы поиска корня нелинейной функции: метод хорд, метод Ньютона, и метод золотого сечения для поиска экстремума. Метод хорд и метод половинного деления целесообразно использовать для выбора начального приближения метода Ньютона для попадания в область сходимости.

Для решения рассмотренных задач разработаны две иерархии классов. В базовом классе первой иерархии наследования осуществляется решение задачи табуляции, в производных классах определяются интервалы нулей и экстремумов. В другой иерархии классов, предназначенной для решения с заданной точностью, используется полиморфные классы и интерфейсный абстрактный класс, который содержит чистую виртуальную функцию поиска корня или экстремума с заданной точностью. Абстрактный интерфейс – это абстрактный класс, составленный из чистых виртуальных функций и не обладающих данными членам. Этот класс содержит объявления виртуальных функций, отражающих интерфейс для использования в производных классах. Конкретный метод реализуется виртуальными методами полиморфных классов при открытом наследовании от базового абстрактного интерфейсного класса. Две иерархии классов находятся в зависимости: класс поиска использует результаты методов первой иерархии классов, в которых осуществляется поиск интервалов.

Множество функций с одинаковой сигнатурой, для которых выполняется исследование, описывается в отдельном модуле, представляющим базу функций. Эта база может быть дополнена или изменена. В модуле базы на множества функций объявляется тип указателя на функции с одинаковой сигнатурой. В данной версии используются функции от одной переменной функции с сигнатурой:

```
typedef double (* fun ) (double );
```

Имена всех функций хранятся в массиве данного типа. В общем случае описывается типы указателя на функции с различной сигнатурой.

Рассмотрим описание разработанных классов. Все классы используют описание класса Point: в данной версии – точка на плоскости. Для задания закрытых полей используется конструктор, для чтения – геттеры

```
Class Point{
    double x, y;
public:
    Point(){x=0; y=0;} //конструкторы класса
    Point(double xv, double yv ){x=xv; y=yv; }
    Point(Point &p){x=p.x; y=p.y; }
    double get_x(){ return x; } //геттеры
    double get_y(){ return y; }
};
```

Решение задачи табуляции выполняется методом `tabul(double (* f) (double x) )` базового класса `Baza_tab`, аргументом которого является указатель на функцию. В методе вычисляется динамический массив точек табуляции и возвращается указатель на этот массив. Указатель на выделенную область памяти сохраняется в защищенном поле `Point *p`. Сформированный динамический массив наследуется в производных классах, в методах которых решаются конкретные задачи исследования, такие как поиск заданных интервалов.

В метод класса табуляции передается указатель на функции с сигнатурой из базы функций, соответствующим аргументом является имя конкретной функции. Защищенными полями класса объявляются задаваемые значения интервала и число узлов. Задание полей и выделение памяти под формируемый массив осуществляется в конструкторе класса. Выделенная память освобождается в деструкторе класса. Описание базового класса :

```
class Baza_tab{
protected:
    double a, b; int n; // интервал и число узлов
    Point *p; // указатель на массив
public:
    //метод табуляции
    Point* tabul( double (* f) (double x) ) {
        double h, x,y ; int i;
        for (i=0 , h=(b-a)/n, x=a ; i<n; x+=h, i++)
            // вызывается конструктор класса Point
            p[i]=Point(x,f(x));
        return p;
    }
    // конструкторы класса
    Baza_tab(){ a=0; b=0; n=0 ; p=0' }
    Baza_tab(double av, double bv ,int nv ) {
        a=av; b=bv ; n=nv; p=new Point[n];
    }
    void Destroy() { // освобождение динамической памяти
        if (p && n > 0) { delete [] p; n = 0; }
    }
    ~Baza_tab(){ Destroy(); } // деструктор класса
};
```

Производные классы `Interval_nul` и `Interval_extrum_max` для вычисления интервалов нулей и экстремумов используют открытое наследование от базового класса `Baza_tab`. Полями классов объявляются указатели `Point *` на динамические массивы точек. Элемент массива содержит значения границ интервалов: в поле `x` записывается начало интервала, в поле `y` – конец интервала, их количество определяется в полях `int kol_in`. Выделенная динамическая память освобождается в деструкторах класса

```
class Interval_nul: public Baza_tab{
protected:
    int kol_in; Point *p_in;
public:
    Point* interwal_nul(); // прототип метода класса
    int get_kol(){ return kol_in; }
    //конструкторы и деструкторы класса
    Interval_nul():Baza_tab(){ kol_in=0; p_in=0; }
    Interval_nul ( double av, double bv , int nv):Baza_tab(av,bv ,nv )
    { kol_in=0; p_in=0; }
    void Destroy() { if (p_in && kol_in > 0) {
        delete [] p_in; kol_in = 0; } }
    ~Interval_nul(){ Destroy(); Baza_tab::~Baza_tab(); }
};
```

```
};
Описание метода вычисления интервалов нулей:
Point* Interval_nul::interwal_nul(); {
    int i, j; Point *dt;    dt = new Point[n];
    for (i = 0, j = 0; i < n - 1; i++)
        if ( p[i].get_y() * p[i + 1].get_y() < 0)
            dt[j++] = Point( p[i].get_x(), p[i + 1].get_x() );
    p_in = new Point[j];    kol_in = j;
    for (i = 0; i < j; i++) p_in[i] = dt[i];
    delete []dt; return p_in;    }
Аналогично проектируется класс вычисления интервалов экстремумов:
class Interval_extrum_max :public Baza_tab{
    int kol_in;    Point *px;
    Point* extrum_max(); // прототип метода класса
    .. //конструкторы и деструкторы класса
    Interval_extrum_max () :Baza_tab(){}
Interval_extrum_max (double av, double bv ,int nv ):
Baza_tab(av,bv ,nv ){ kol_in=0; px=0;}
};
Описание метода класса вычисления интервалов экстремумов:
Point* Interval_extrum_max:: extrum_max(); {
    int i, j;    Point *dt;    dt = new Point[n];
    for(i=1, j=0; i<n-1; i++)
        if ( p[i].get_y() > p[i+1].get_y() && p[i].get_y() > p[i- 1].get_y() )
            dt[j++] = Point( p[i-1].get_x(), p[i + 1].get_x() );
    px = new Point[j];    kol_in = j;
    for (i = 0; i < j; i++) px[i] = dt[i];    delete [] dt;
    return px;
}
```

В модуле клиенте объявляются следующие переменные: указатели на класс Point *tv, *te;, указатель на указатель на класс Interval_nul **fz;. По указателю fz выделяется память под динамический массив указателей fz= new Interval_nul * [m], где m – число рассматриваемых функций. Во внешнем цикле определяются интервалы нулей для каждой функции базы, имена функций записаны в массиве mas_f. Фрагмент программы клиента:

```
Point *tv, *te; Interval_nul **fz;
double a,b; int i,j,n,m;
cin>>m>>a>>b>>n;
fz= new Interval_nul * [m];
for (i=0; i< m; i++){
    fz [i]=new Interval_nul (a,b,n);
    fz[i]->tabul_v(mas_f[i]);
    tv=fz [i]->interwal_nul();
    for (j=0; j< fz [i]->get_kol() ; j++){
        cout<< tv[j].get_x()<< " " << tv[j].get_y()<< endl;
    };
};
```

Аналогично для поиска интервалов экстремумов объявляется указатель на указатель на класс Interval_extrum_max **fze, по которому выделяется память под динамический массив указателей:

```
fzt= new Interval_extrum_max * [m],
где m – число рассматриваемых функций.
```

На каждом шаге цикла по числу исследуемых функций вызывается конструктор класса, метод предка fzt[i]->tabul_v(mas_f[i]) и метод класса поиска интервалов локальных экстремумов tv=fzt [i]-> extrum_max();

```

for (i=0; i< m; i++){
    fze [i]=new Interval_extrum_max (a,b,10);
    fze[i]->tabul_v(mas_f[i]);
    tv=fze [i]-> extrum_max();
}

```

Разработанные иерархии классов можно рассматривать как первое приближение к проектированию паттерна исследования функций, в методах которого решаются задачи поиска интервалов, включающих корни или локальные экстремумы.

1. Страуструп Б. Язык программирования C++. СПб.: Бином, 1999. с. 990
2. Русакова З.Н. Динамические структуры данных и вычислительные алгоритмы Visual C++. Санкт-Петербург: 2014, 272 с.
3. Русакова З.Н. Система моделирования и интеллектуализации задач принятия решений. Инженерный журнал: Наука и инновации #2.14/2013, с.9.
4. Грешилов А.А. Математические методы принятия решений. МГТУ им. Н.Э. Баумана, Москва 2016. 320 с.

Тазабеков Д.Ю., Бужинская Н.В.

Применение среды Unity для разработки обучающих компьютерных игр

*Нижнетагильский государственный социально-педагогический институт, филиал
ФГАОУ ВО «Российский государственный профессионально-педагогический
университет»
(Россия, Нижний Тагил)*

doi: 10.18411/lj-02-2020-26

idsp: ljournal-02-2020-26

Аннотация

В статье рассматриваются особенности разработки обучающей компьютерной игры в среде Unity. Данная игра предназначена для обучения школьников информатике в занимательной форме. Графическая оболочка игры создавалась в Adobe Photoshop CS6, игровая механика – на игровом движке Unity. Для написания скриптов использовался язык программирования C#.

Ключевые слова: Unity, обучающая компьютерная игра, уровень, объект, скрипт, сцена, C#, разработка, тестирование.

Abstract

The article discusses the features of developing a training computer game in the Unity environment. This game is designed to teach students computer science in an entertaining way. The graphical shell of the game was created in Adobe Photoshop CS6, the game mechanics were written on the Unity game engine. The C#programming language was used to write scripts.

Keywords: Unity, educational computer game, level, object, script, scene, C#, development, testing.

Разработка компьютерных игр может стоить миллионы рублей, причем в течении последних десятилетий затраты в данной индустрии непрерывно возрастают. Однако, благодаря развитию рынка инди-игр, многие разработчики компьютерных игр получили возможность работать над различными игровыми проектами самостоятельно, без обязательств перед другими компаниями. Инди-игры (независимые компьютерные игры) – это компьютерные игры, созданные отдельными разработчиками или