РАЗДЕЛ XV. ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

Русакова З.Н.

Использование полиморфизма С++ для проектирования шаблона решения нелинейных уравнений

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

doi: 10.18411/lj-03-2020-43
idsp: ljournal-03-2020-43

Аннотация

Статья посвящена вопросам проектирования программного инструментария С++ для решения нелинейных уравнений с заданной точностью. Разработана полиморфная иерархия классов, реализующих задачи определения нулей и экстремумов для множества функций с одинаковой сигнатурой, определяемых в базе функций. Абстрактный класс содержит объявления виртуальных функций, отражающих интерфейс для использования в производных классах. Конкретный метод реализуется виртуальными методами полиморфных классов

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

Abstract

The article is devoted to the design of C ++ software tools for solving non-linear equations with a given accuracy. A polymorphic hierarchy of classes has been developed that implements the tasks of determining zeros and extrema for a set of functions with the same signature defined in the function base. An abstract class contains declarations of virtual functions that reflect an interface for use in derived classes. A specific method is implemented by virtual methods of polymorphic classes

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. 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 разработана полиморфная иерархия классов, предназначенная для решения нелинейных уравнений с заданной точностью. В этой иерархии классов используется полиморфные классы и интерфейсный абстрактный класс, который содержит чистую виртуальную функции 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 классам. Класс поиска использует результаты методов 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 функций 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.

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; }
);
```

Базовый класс поиска корней и экстремумов с заданной точностью Baza_poisk_virt строится как абстрактный интерфейсный класс. В нем объявляется чистая виртуальная функция:

virtual double poisk(double eps)=0,которая замещается в производных классах, реализующих конкретный метод поиска: метод хорд, Ньютона, золотое сечение. Параметр метода – заданная точность.

```
class Baza_poisk_virt{
    public:
    virtual double poisk( double eps )=0;
     virtual   ~Baza_poisk_virt(){}
};
```

Указатель на функцию объявляется как поле в производных классах. Производные классы открыто наследуются от базового интерфейсного класса

```
class Met_xord_virt: public Baza_poisk_virt {
double  a, b;    int n;
fun f;    ..указатель на функцию
public:
double poisk( double eps );
Met_xord_virt(){}
Met_xord_virt(double av, double bv , fun ff ) {a=av; b=bv; f=ff;}
 };
```

Аналогично описываются другие полиморфные классы. В классе Met_newton_virt чистая виртуальная функция замещается методом класса, в котором осуществляется поиск по методу Ньютона:

```
class Met_newton_virt:  public Baza_poisk_virt {
//поля конструкторы
double poisk( double eps );
};
```

В основе метода хорд лежит правило отсечения бесперспективной зоны поиска путем изменения границ интервала. На каждом шаге поиска изменяется правая или левая граница интервала по следующему правилу. Вычисляется точка пересечения хорды с осью x и произведение значения исследуемой функции в этой точке на

значения функции в одном из концов заданного интервала. Если функция меняет знак на интервале (произведение меньше 0), то интервал включает корень и одна из границ становится равной точке пересечения хорды обратно.

```
double Met_xord_virt:: poisk( double eps ) {
 double   d , xt, xp, fa, fb ,fx ;
 fa=f(a) ;      fb=f(b) ;  d=1;  n=1;
xp=a - fa*( (b-a)/(fb-fa)) ; //  точка пересечения
fx=f(xp); //функция в точке приближения
while  ( fabs ( fx ) > eps  &&   fabs(d) > eps  && n < 1000  ) {
      if (  fa* fx  < 0 )
                { b=xp;      fb=fx;   }
            else    {  a=xp;      fa=fx;       }
//Вычисляется точка пересечения хорды с осью x для новых границ
        xt=a - fa* ( (b-a) /(fb-fa) ) ;
         fx=f(xt);   // новое значение функции
        n++;  d= xt - xp;    .. погрешность
         xp=xt; // текущее в предыдущее }
   }
   return xp;
 }
```

В методе класса double Met_newton_virt:: poisk(double eps) выполняется уточнение корня по методу Ньютона. Итерационный процесс начинается с задания начального приближения значением одного из концов интервала. Следующие приближения вычисляются по рекуррентным соотношениям метода. В коде используется аппроксимация производной функции значением конечных разностей. Критерий завершения итерационного процесса: модуль разности между последовательными приближениями меньше заданной точности и значение модуля функции в текущей точке меньше заданной точности

```
double Met_newton_virt::poisk( double eps )   {
  double xp, xt, d, h,  fp,pfa ,pfb;
   int i;  h= 0.1e-11;     xp=a;   d = 1;            n=0;
    while ( fabs(d )> eps &&  n < 100 )  {
                  fp=( f(xp+h)-f(xp) )/ h ;
                  if (  fabs( fp ) > 0.1e-11  )
                    xt = xp - f(xp)/ fp;
else      break;
                 n++;   d = (xt - xp);
        xp = xt;
} //текущее переписывается в предыдущее
  return xt;
 }
```

Производный класс открыто наследуемый от базового интерфейсного класса class Gold_max разработан для поиска приближенного решения экстремумов унимодальных функций. Метод золотого сечения основан на делении отрезка по правилу золотого сечения и сужения интервала поиска благодаря отсечению бесперспективной зоны. На каждом шаге поиска изменяется правая или левая граница интервала по следующему правилу. Вычисляются координаты двух точек внутри заданного интервала по методу золотого сечения: к = (sqrt(5) – 1)/2; x1 = a + (1 - k)*(b - a); x2 = a + k*(b - a);

В этих точках вычисляются значения исследуемой функции f(x1), f(x2). Из условия выполнения условия f(x1) < f(x2) изменяется левая или правая граница интервала поиска по правилу: если f(x1) < f(x2), то на следующем шаге поиска

интервал от a до x1 исключается из зоны поиска и левая граница интервала становится равной x1. Если условие не выполняется, то изменяется правая граница: b = x2. Решение с заданной точностью достигается при выполнении условия: abs(a - b) < eps — величина интервала поиска меньше заданной точности и включает экстремум. Метод класса Gold_max

```
double poisk( double  (* f) (double x)){
 double k, d, x1, x2, eps=0.1e-10; int i=0;
  k = (sqrt(5.0) - 1)/2;    x1 = a + (1 - k)*(b - a);   x2 = a + k*(b - a);
 do {
   if (f(x1) < f(x2))  {
        a = x1;   //  текущая граница отрезка
        d = b - a;
     // вычисление новых пробных точек по методу золотого сечения
         x1 = x2;    x2 = a + k*d;          }
    else  {
          b = x2; // текущая граница отрезка
            d = b - a;  x2 = x1;  x1 = a + (1 - k)*d;       }
  } while (fabs(d) > eps );
return (x1+x2)/2;  }
```

В модуле клиенте объявляются переменные: указатель на интерфейсный класс Baza_poisk_virt *zpf. Для каждой функции из базы функций имена записываются в массив mas_f. В предположении, что интервалы известны и записаны в массивах a, b, во внешнем цикле для каждой функции вызываются методы вычисления корня с заданной точностью для каждого интервала. Значения выбираются из массивов a, b. Фрагмент программы клиента:

```
Baza_poisk_virt  *zpf;
 int i,j,n,m; double s,  eps=0.1e-15;
for (i=0;  i< m; i++)
   for (j=0;  j< n ; j++){
    zpf= new Met_xord_virt( a[j], b[j], mas_f[i] );
   s=zpf->poisk(eps);
   zpf= new Met_newton_virt (a[j], b[j], mas_f[i]   );
   s=zpf->poisk(eps);
)
```

Во вложенном цикле решение поисковых задач осуществляется полиморфными объектами указателей на базовый класс. Разработанную полиморфную иерархию классов можно рассматривать как первое приближение к проектированию паттерна исследования функций.

Разработанную иерархию полиморфных классов, реализующих задачи определения нулей и экстремумов для множества функций с одинаковой сигнатурой, определяемых в базе функций, можно рассматривать как базовый класс для полиморфной реализации классов решения систем нелинейных уравнений.

***

1. Страуструп Б. Язык программирования C++. СПб.: Бином, 1999. с. 990
2. Русакова З.Н. Динамические структуры данных и вычислительные алгоритмы Visual C++. Санкт-Петербургбург.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.