Конспект лекций к лабораторной работе по теме «Метод дихотомии»

Метод дихотомии (или метод половинного деления – иногда под ними подразумевают различные технологии, но обычно считают разными названиями одной и той же) используется для поиска корня уравнения f(x)=0, которое или не имеет аналитического решения (трансцендентное уравнение), или же вычисления при его поиске трудоемки, а итоговая их погрешность соизмерима с гораздо менее трудоемкой дихотомией. Перечислим условия применимости метода:

- известен отрезок [L; R], где есть ровно один корень;
- в границах этого отрезка у функции нет разрывов;
- справа и слева от корня функция меняет знак см. рис.1:

Из рисунка видно, что можно принять L=0, R=1.

Этот пример трансцендентного уравнения позволяет легко описать все последующие шаги:

1) вычисляем серидину отрезка m=(L+R)/2;
2) Проверяем знак произведения:
   a. если f(L)*f(m)>0, то здесь корня нет и можно задать L=m
   b. иначе корень есть, а занчит зададим R=m

В любом случае отрезок поиска корня уменьшился вдвое.

3) Повторяем пункты 1-2 пока R-L>заданной погрешности.

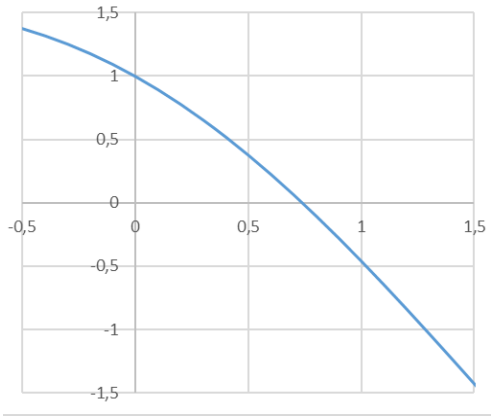


Рис.1. График f(x)=cos(x)-x

Ниже представлен текст программы на языке PascalABC.net решения данной задачи:

```
1   ##
2   function f(x:real):=cos(x)-x;
3   //L,R,m,e-координаты левой, правой границ и середины отрезка, e-погрешность
4   var (L,R,m,e):=(0.0,1.0,0.0,0.0001);//числа указать вещественными!!!
5   var c:=0; //счетчик итераций
6   while R-L> e do begin
7     m:=(L+R)/2; c+=1;
8     if f(L)*f(m)>0 then L:=m else R:=m
9   end;
10  writeln('x=',L, '; c=',c);
```

Для данного случая выведется: `x=0.73907470703125; c=14`

Это трансцендентное уравнение имеет только численное решение. Количество итераций в методе известно заранее: $c = \text{Log}_2((R - L)/e) = \text{Log}_2(10^4) \cong 14$, но часто нужно использовать не абсолютную, а относительную погрешность и строка 5 программы примет вид: `while (R-L)/(abs(R)+abs(L))> e do`

В этом случае формула для количества итераций не работает (в данном примере получим c=13).

Рассмотрим случай, когда для уравнения существует аналитическое решение, где накопленная при вычислениях погрешность превышает погрешность дихотомии, например: $f(x)=x^4-2\cdot x^3-3\cdot x^2+4\cdot x-0{,}9=0$

Для такого уравнения возможны от 0 до 4 корней и нужно найти отрезок [L; R] для каждого из них. Построив для этого график функции в Excel (см. Рис 2) можно увидеть, что число корней ≥2, а для определения их наличия и значений [L; R] для отрезка [0; 1] нужно рассмотреть его подробнее, изменив масштаб построения, причем как по X, так и по Y. Может оказаться также, что корни находятся ближе друг к другу, чем шаг по X, выбранный при вычислении таблицы для графика. Тогда для «подозрительного» участка строим отдельный график меньшим шагом - может оказаться, что здесь или нет корней (график ниже оси), или 1 корень (касается оси), или 2.

В приведенном примере очевидные отрезки: {[-2;-1], [2;3]}. Построив более детальный график (см. Рис.3.) увидим, что есть еще два отрезка: {[0;0.5], [0.5;1]}, которые можно использовать для дихотомии. Заметим, что *если бы график касалсяется X, то дихотомия была бы неприменима.*

Примерное значение корней: {-1.6; 0.3; 0.7; 2.6} (Excel).

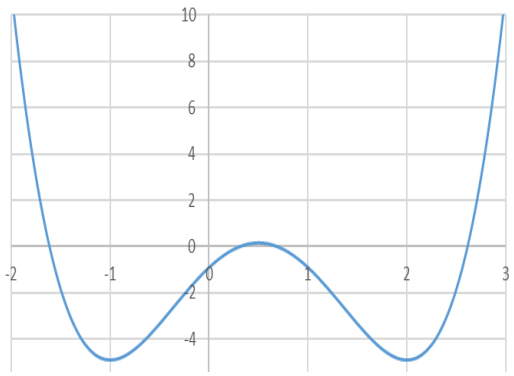


Рис. 2. График полинома 4-й степени

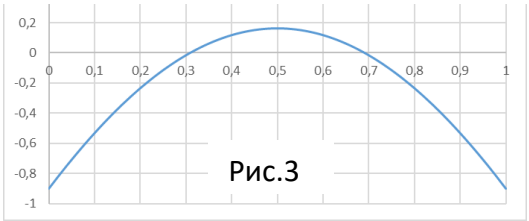


Рис.3

Конспект лекций к лабораторной работе по теме «Метод простых итераций»

В отличие от дихотомии, где известен отрезок поиска корня, будем полагать, что ни число, ни границы корней не известны. Строго говоря, прежний метод также является итерационным, а потому выбранное название достаточно условно, а потому примем, что здесь важно лишь начальное значение аргумента (*стартовая точка*) и выбранный *алгоритм приближения к корню*. Если корень один, то всегда можно построить нужный итерационный процесс, но если их несколько, то обычно удается найти лишь некоторые из них. Рассмотрим метод подробнее.

Пусть есть уравнение f(x)=0 и мы пытаемся привести его к виду $x_{i+1}=\varphi(x_i)$, где скорость изменения функции справа не быстрее линейной. Можно предположить, что в этом случае x_{i+1} находится ближе к корню уравнения, чем x_i. Как пример рассмотрим функцию f(x)=cos(x)-x (график – см.Рис.1.).

Нас интересует f(x)=0, тогда можно записать итерационное уравнение: $x_{i+1}=\cos(x_i)$. Ниже представлен фрагмент компьютерной программы для метода простых итераций:

```
1  ##
2  function f(x:real):=cos(x);
3  var (xnew,xold,e,c):=(0.0,0.0,0.0001,0);
4  repeat
5    xold:=xnew; c+=1; xnew:=f(xold);
6  until (abs(xnew-xold)/(abs(xnew)+abs(xold))<e) or (c>1000);
7  if c=1001 then println('Процесс не сходится, корень не найден')
8  else writeln(' x=',xnew,'; c=',c);
```

Окно вывода

```
x=0.739130176529671; c=23
```

Корень найден, но по сравнению с методом дихотомии понадобилось больше шагов алгоритма. В то же время здесь не нужно знать границы поиска корня, и если взять в качестве *стартовой точки* x=1000, то нужная точность будет достигнута даже за меньшее число шагов: c=22.

Рассмотрим уравнение $f(x)=x^4-2\cdot x^3-3\cdot x^2+4\cdot x-0{,}9=0$. По аналогии с прежним решением запишем итерационную формулу, хотя сделать это уже сложнее. Покажем это.

Добавив к обеим частям уравнения переменную x, получим: $x_{i+1}=x_i^4-2\cdot x_i^3-3\cdot x_i^2+5\cdot x_i-0{,}9$. Итерации сходятся при скорости изменения функции не выше линейной, а здесь полином четвертой степени! Можно найти корень с ожидаемым отезком сходимости [-1; 1], расширяемый при неявной схеме: xold:=(xnew+k*xold)/(1+k). Обычно k порядка степени полинома, тогда, заменив формулу для xold в 5 строке алгоритма: xold:=(xnew+4*xold)/5, найдем корень **0.69** со *стартовой точкой* Є[-11; 11].

Выразим x^4 через оставшуюся часть уравнения: $x_{i+1}=(2\cdot x_i^3+3\cdot x_i^2-4\cdot x_i+0{,}9)^{1/4}$. Здесь скорость роста правой части соответствует степени ¾, – медленнее линейной. При составлении программы следует учитывать, что в процессе решения возможно отрицательное подкоренное выражение, а потому его нужно брать как абсолютное значение. Соответствующая функция имеет вид:

```
2  function f(x:real):=(abs(2*x**3+3*x*x-4*x+0.9))**0.25;
```

В этом случае найден один верный корень: **2.61** (см. Рис.2) при широком диапазоне значений *стартовой точки*. Другие положительные корни не найдены, а отрицательные в принципе не могли быть найдены, поскольку для корня четной степени в итерационном процессе это не предусмотрено.

Домножим уравнение на x и применим предыдущую схему! Тогда: $x_{i+1}=(2\cdot x_i^4+3\cdot x_i^3-4\cdot x_i^2+0{,}9\cdot x_i)^{1/5}$. В этом случае можно найти и отрицательные корни, но, к сожалению, ни один язык программирования не позволяет выполнять извлечение корня из отрицательного числа даже в случае нечетной степени этого корня! Извлечем корень из абсолютного значения, добавив нужный знак:

```
1  function f(x:real):real;
2  begin
3    var t:=2*x**4+3*x**3-4*x*x+0.9*x;
4    result:=sign(t)*abs(t)**0.2;
5  end;
```

Этот подход позволяет найти и отрицательные корни в области сходимости итерационного процесса (в нашем случае *найдены корни*: **-1.61** и **2.61** при стартовых точках 0.1 и 1.0). В целом можно утверждать:

- методом простых итераций находится минимум 1 корень при верном выборе итерационной схемы;
- у полиномов четных степеней удобно перейти к нечетной, домножив обе его части на аргумент, а затем использовать алгоритм, указанный выше, включая извлечение корня.

Покажем, как данную технологию можно применять для решения систем нелинейных уравнений:

Пусть дана система двух нелинейных уравнений: $\begin{cases} -3 \cdot x^7 + 5 \cdot y^5 = -13 \\ 2 \cdot x^9 - 7 \cdot y^{11} = -5 \end{cases}$

Выразив из первого уравнения x, а из второго – y, получим: $\begin{cases} x_{new} = \sqrt[7]{(13 + 5 \cdot y_{old}^5)/3} \\ y_{new} = \sqrt[11]{2 \cdot x_{old}^9 + 5)/7} \end{cases}$

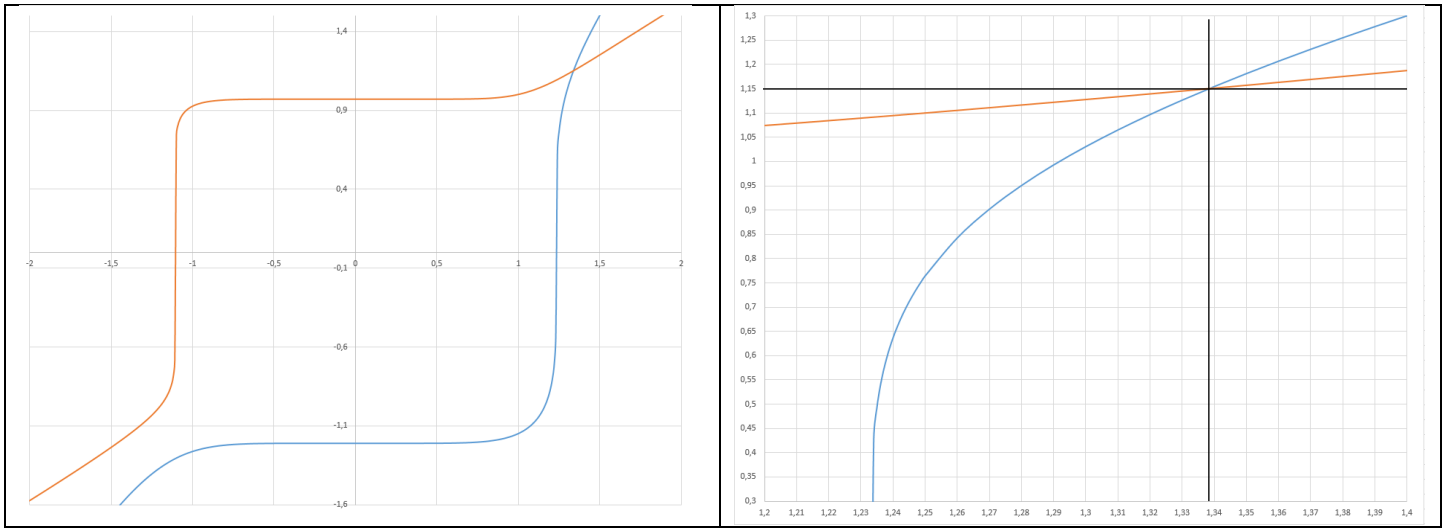
Теперь построим итерационный процесс по простой или неявной схеме, где для оценки используем относительную погрешность отдельно по каждой переменной, как указано в программе ниже:

```
##
function fx(x,y:real):real;
begin result:=(5*y**5+13)/3;
  result:=sign(result)*abs(result)**(1.0/7);
end;
function fy(x,y:real):real;
begin result:=(2*x**9+5)/7;
  result:=sign(result)*abs(result)**(1.0/11);
end;
 var (xo,xn,yo,yn,e,c):=(0.0,0.0,0.0,0.0,0.000001,0);
 repeat
   xo:=xn;yo:=yn; c+=1; xn:=fx(xo,yo);yn:=fy(xo,yo);
 until ((abs(xn-xo)/(abs(xn)+abs(xo))<e)and
        (abs(yn-yo)/(abs(yn)+abs(yo))<e)) or (c>1000);
 println(xn,yn,c);
```

Окно вывода

```
1.33815320939555 1.14983590820001 16
```

Как видно из результата, итерации сходятся очень быстро. Можно показать, что решение единственное:



Здесь, как и раньше, обязательным условием сходимости алгоритма является непрерывность функции.