

# \* Итерационные алгоритмы и программы

Лекция 7

В итерационных циклах производится проверка некоторого условия, и в зависимости от результата этой проверки происходит либо выход из цикла, либо повторение выполнения тела цикла. Если проверка условия производится перед выполнением блока операторов, то такой итерационный цикл называется циклом с **предусловием** (цикл "пока"), а если проверка производится после выполнения тела цикла, то это цикл с **постусловием** (цикл "до").

Особенность этих циклов заключается в том, что тело цикла с постусловием всегда выполняется хотя бы один раз, а тело цикла с предусловием может ни разу не выполниться. В зависимости от решаемой задачи необходимо использовать тот или иной вид итерационных циклов.

# \* Цикл с постусловием

Оператор повтора `repeat` состоит из

- *заголовка (repeat),*
- *тела*
- *условия окончания (until).*

**repeat**

**Инструкции**

**until Условие\_выхода\_из\_цикла;**

Условие выхода из цикла — это выражение логического типа.

# \* Описание работы

Цикл работает так: *вначале* выполняется тело цикла — инструкции, которые находятся между **repeat** и **until**, *затем* проверяется значение **Условия выхода из цикла**. В том случае, если оно равно **false** (ложь), т. е. не выполняется, — инструкции цикла повторяются еще раз. Так продолжается до тех пор, пока условие не станет **true** (истина).

Для успешного завершения цикла **repeat** в его теле обязательно должны находиться инструкции, выполнение которых *влияет* на условие завершения цикла, иначе цикл будет выполняться бесконечно — программа зациклится. Т. е., *переменная, которая участвует в условии выхода из цикла, обязательно должна изменяться* в теле цикла;

Нижняя граница операторов тела цикла четко обозначена словом **until**, поэтому нет необходимости заключать эти операторы в операторные скобки **begin** и **end**.

Цикл **repeat** – это цикл с *постусловием*, т. е. инструкции тела цикла будут выполнены *хотя бы один раз*. Поэтому цикл **repeat** удобно использовать в тех случаях, когда тело цикла гарантированно *должно выполниться хотя бы один раз*.

Оператор **repeat** часто используют для *проверки правильности ввода исходных данных*, т. к. это именно тот самый случай, когда тело цикла гарантированно выполняется один раз, а при наличии ошибки в исходных данных – более одного раза.

```
var c: char;  
begin  
  repeat  
    writeln(' Введите Y или N');  
    readln(c);  
  until (c='Y') or (c='N');  
end.
```

Цикл `repeat` удобно использовать для *обработки ошибки ввода* при несоответствии типа введенных данных типу переменной.

В Pascal предусмотрена возможность **отключения стандартного контроля ошибок ввода** при помощи директивы компилятора `{I-}`, а контроль правильности ввода выполняет стандартная функция `ioResult`. Эта функция имеет нулевое значение, если последняя операция ввода/вывода закончилась успешно, и ненулевое значение, равное коду ошибки, если ввод/вывод закончился неудачей.

После вызова функция `ioResult` сбрасывает свое значение, поэтому обычно ее значение сразу присваивается какой-либо переменной. Рекомендуется после анализа результата, возвращаемого `ioResult`, снова установить стандартный контроль ошибок ввода/вывода с помощью директивы `{I+}`.

```
var
```

```
x: integer; ok: boolean;
```

```
begin
```

```
  repeat
```

```
    writeln('Введите целое x');
```

```
    {$i-} {отмена стандартного контроля оп-й ввода/вывода }
```

```
    readln(x);
```

```
    {$i+} {включение стандартного контроля оп-й вв/вывода}
```

```
  until ioresult=0;
```

```
end.
```

# \* Цикл с предусловием

**while** Условие\_выполнения\_цикла **do**

**begin**

Инструкции

**end;**

Условие выполнения цикла — это выражение логического типа. Тело цикла — оператор, чаще всего составной. Если тело цикла представляет собой одиночный оператор, то ключевые слова **begin** и **end** лучше не писать, хотя их наличие не будет ошибкой.

# \* Описание работы

Цикл `while` работает так: *вначале* выполняется проверка значения **Условие выполнения цикла** – если значение условия равно `true` (истина), то *выполняются инструкции цикла*, находящиеся между `begin` и `end` и снова вычисляется выражение **Условие выполнения цикла**. Так продолжается до тех пор, пока значение **Условие выполнения цикла** не станет равно `false` (ложь).

Пример. Найти НОД двух натуральных чисел, используя алгоритм Евклида.

```
var
    a, b:integer;
begin
    writeln('Введите два натуральных числа');
    readln(a, b);
    while a<>b do
        if a>b then a:=a-b
            else b:=b-a;
    writeln('НОД=', a);
    readln;
end.
```

Пример. Пользователь вводит целое положительное число. Определите, какое минимальное количество последовательно расположенных натуральных чисел необходимо сложить, чтобы их сумма превзошла введенное пользователем число.

```
var n,i,s:integer;
begin
  writeln('Vvedite chislo');
  readln(n);
  i:=0;
  s:=0;
  repeat
    i:=i+1;
    s:=s+i;
  until s>n;
  writeln('kolichestvo =',i);
  readln;
end.
```

```
var n,i,s:integer;
begin
  writeln('Vvedite chislo');
  readln(n);
  i:=0;
  s:=0;
  while s<=n do
  begin
    i:=i+1;
    s:=s+i;
  end;
  writeln('kolichestvo =',i);
  readln;
end.
```

# \* Численные методы решения алгебраических и трансцендентных уравнений

Дано нелинейное уравнение:  $f(x)=0$ .

Если функцию можно привести к виду

$$f(x)=a_0x^m+a_1x^{m-1}+a_2x^{m-2}+\dots+a_{m-1}x+a_0,$$

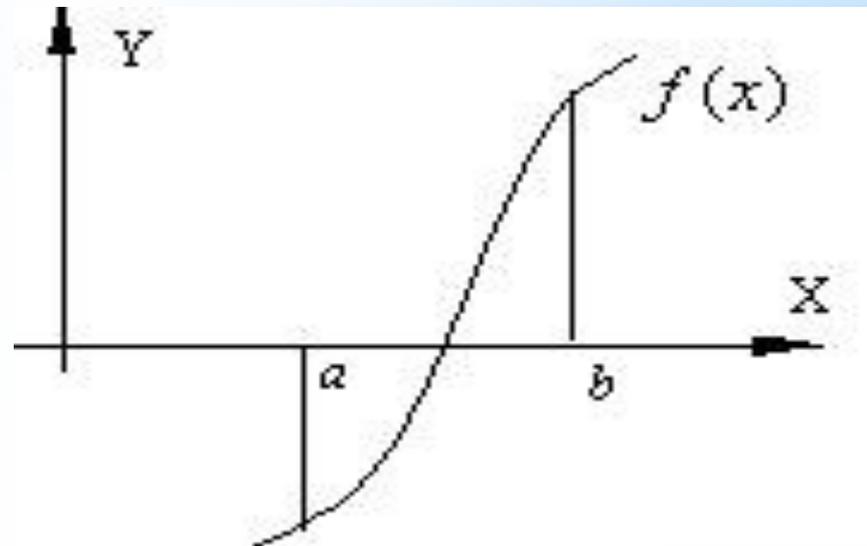
где  $a_i$  - коэффициенты многочлена, , то уравнение  $f(x)=0$  называется алгебраическим.

Если функция  $f(x)$  включает в себя тригонометрические или экспоненциальные функции от некоторого аргумента  $x$  и т. д., то уравнение называется трансцендентным уравнением.

*Как известно, не всякое уравнение может быть решено точно. В первую очередь это относится к большинству трансцендентных уравнений.*

Однако точное решение уравнения не всегда является необходимым. Задачу отыскания корней уравнения можно считать практически решенной, если мы сумеем найти корни уравнения с заданной степенью точности. Для этого используются приближенные (численные) методы решения.

Большинство употребляющихся приближенных методов решения уравнений являются, по существу, способами уточнения корней. Для их применения необходимо знание интервала изоляции  $[a, b]$ , в котором лежит уточняемый корень уравнения.



Процесс определения интервала изоляции  $[a, b]$ , содержащего только один из корней уравнения, называется отделением этого корня.

Процедура отделения корней основана на известном свойстве непрерывных функций: если функция непрерывна на замкнутом интервале  $[a, b]$  и на его концах имеет различные знаки, т.е.  $f(a)f(b) < 0$ , то между точками  $a$  и  $b$  имеется хотя бы один корень уравнения. Если при этом знак функции  $f'(x)$  на отрезке  $[a, b]$  не меняется, то корень является единственным на этом отрезке.

Процесс определения корней алгебраических и трансцендентных уравнений состоит из 2 этапов:

- отделение корней, - т.е. определение интервала  $[a, b]$ , внутри которого лежит корень уравнения;
- уточнение корней, - т.е. сужение интервала  $[a, b]$  до величины равной заданной степени точности .

# \* Метод половинного деления

Дано нелинейное уравнение:  $f(x)=0$ . Найти корень уравнения, принадлежащий интервалу  $[a,b]$ , с заданной точностью  $\varepsilon$ .

Для уточнения корня методом половинного деления последовательно осуществляем следующие операции:

1. Делим интервал пополам:  $t = \frac{a+b}{2}$
2. В качестве нового интервала изоляции принимаем ту половину интервала, на концах которого функция имеет разные знаки.

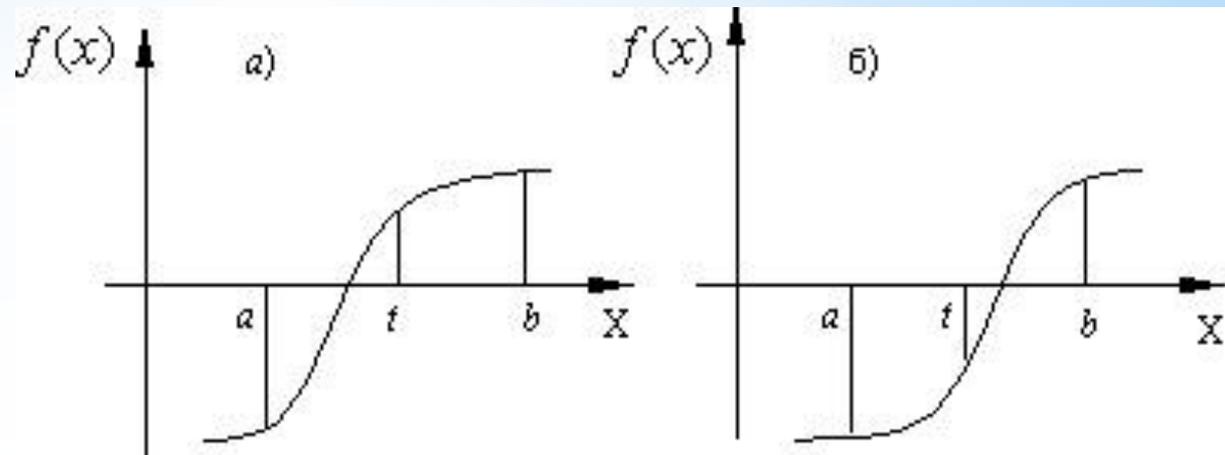
Для этого:

Вычисляем значение функции  $f(x)$  в точках  $a$  и  $t$ .

б) Проверяем: если  $f(a)f(t) < 0$ , то корень находится в левой половине интервала  $[a, b]$ . Тогда отбрасываем правую половину интервала и делаем  $b=t$ .

с) Если  $f(a)f(t) < 0$  не выполняется, то корень находится в правой половине интервала  $[a, b]$ . Тогда отбрасываем левую половину и делаем  $a=t$ . В обоих случаях мы получим новый интервал  $[a, b]$  в 2 раза меньший предыдущего.

3. Процесс, начиная с пункта 1, циклически повторяем до тех пор, пока длина интервала  $[a, b]$  не станет равной либо меньшей заданной точности, т.е.  $|b - a| \leq \varepsilon$ .



```
Const e=0.001;
```

```
Var x, t, a, b:real;
```

```
function f1(x:real):real;
```

```
begin
```

```
    f1:=sin(x);
```

```
end;
```

```
begin
```

```
    writeln('Введите a, b');
```

```
    readln(a, b);
```

```
    repeat
```

```
        t:=(a+b)/2;
```

```
        if f1(a)*f1(t)<=0 then b:=t else a:=t;
```

```
    until abs(b-a)<=e;
```

```
    x:=(b+a)/2;
```

```
writeln('Корень равен', x:7:4); end.
```

# \* Метод простых итераций

Пусть с точностью  $\varepsilon$  необходимо найти корень уравнения  $f(x)=0$ , принадлежащий интервалу изоляции  $[a,b]$ . Функция  $f(x)$  и ее первая производная непрерывны на этом отрезке.

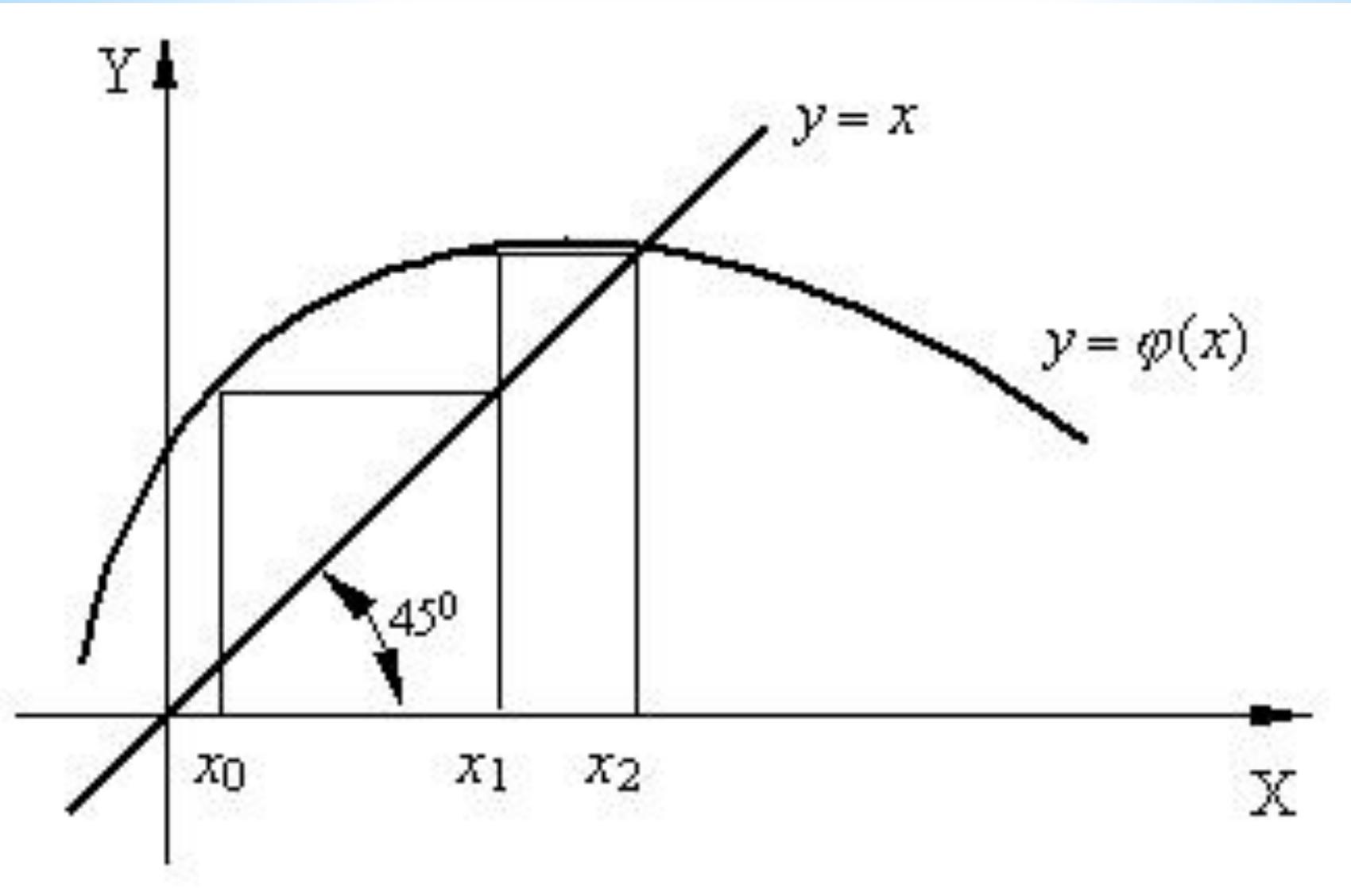
Для применения этого метода исходное уравнение  $f(x)=0$  должно быть приведено к виду  $x=\varphi(x)$ . В качестве начального приближения выбираем любую точку интервала  $[a,b]$ .

Далее итерационный процесс поиска корня строится по схеме:

$$x_1=\varphi(x_0), \quad x_2=\varphi(x_1), \quad \dots, \quad x_n=\varphi(x_{n-1}),$$

Процесс поиска прекращается, как только выполняется условие  $|x_n - x_{n-1}| \leq \varepsilon$  или число итераций превысит заданное число  $N$ .

Для того, чтобы последовательность  $x_1, x_2, \dots, x_n$  приближалась к искомому корню, необходимо, чтобы выполнялось условие сходимости:  $|\varphi'(x)| < 1$ .



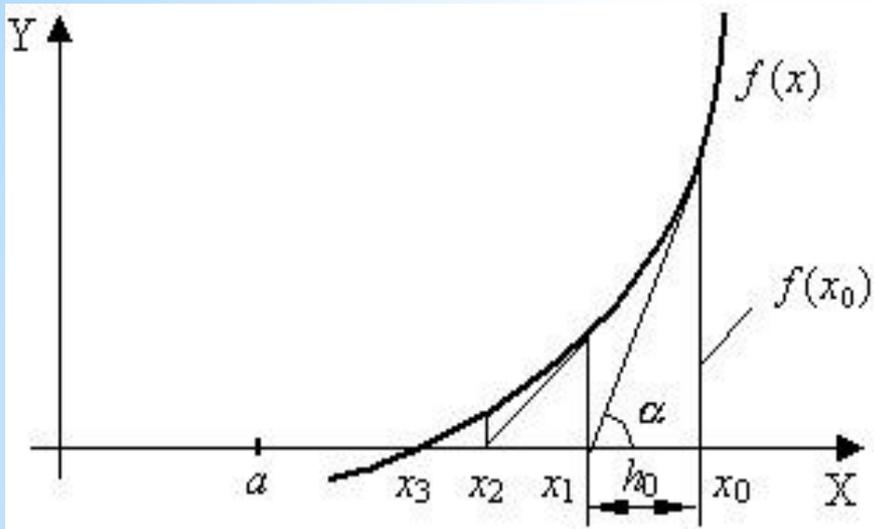
# \* Метод Ньютона (метод касательных)

Метод Ньютона относится к градиентным методам, в которых для нахождения корня используется значение производной.

Дано нелинейное уравнение:  $f(x)=0$ . Найти корень на интервале  $[a,b]$  с точностью  $\varepsilon$ .

Метод Ньютона основан на замене исходной функции  $f(x)$ , на каждом шаге поиска касательной, проведенной к этой функции. Пересечение касательной с осью  $X$  дает приближение корня.

Выберем начальную точку  $x_0=b$  (конец интервала изоляции). Находим значение функции в этой точке и проводим к ней касательную, пересечение которой с осью  $X$  дает нам первое приближение корня  $x_1$ .



$x_1 = x_0 - h_0$ , где  $h_0 = \frac{f(x_0)}{\operatorname{tg}(\alpha)} = \frac{f(x_0)}{f'(x_0)}$ . Поэтому  $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$ . В результате итерационный процесс схождения к корню реализуется рекуррентной формулой  $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ . Процесс поиска продолжаем до тех пор, пока не выполнится условие  $|x_n - x_{n-1}| \leq \varepsilon$ . Упростим условие  $\left| \frac{f(x_n)}{f'(x_n)} \right| \leq \varepsilon$ .

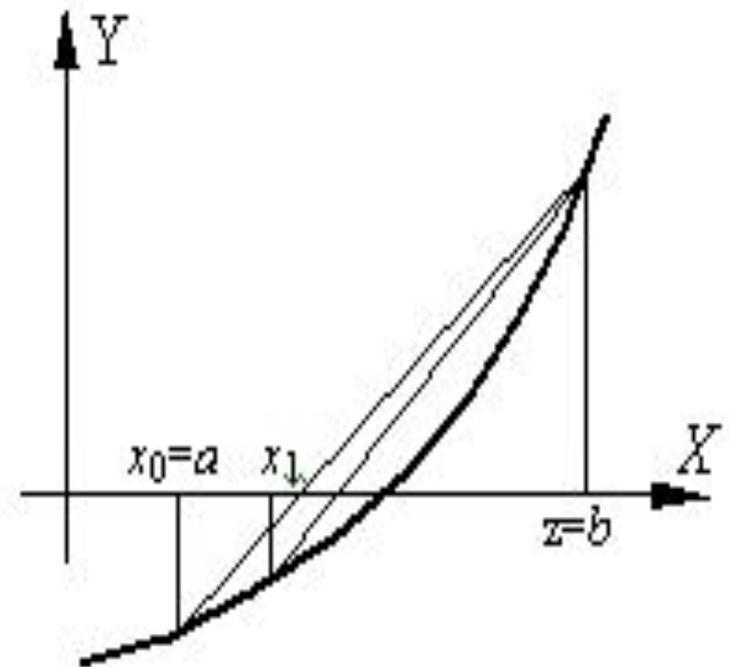
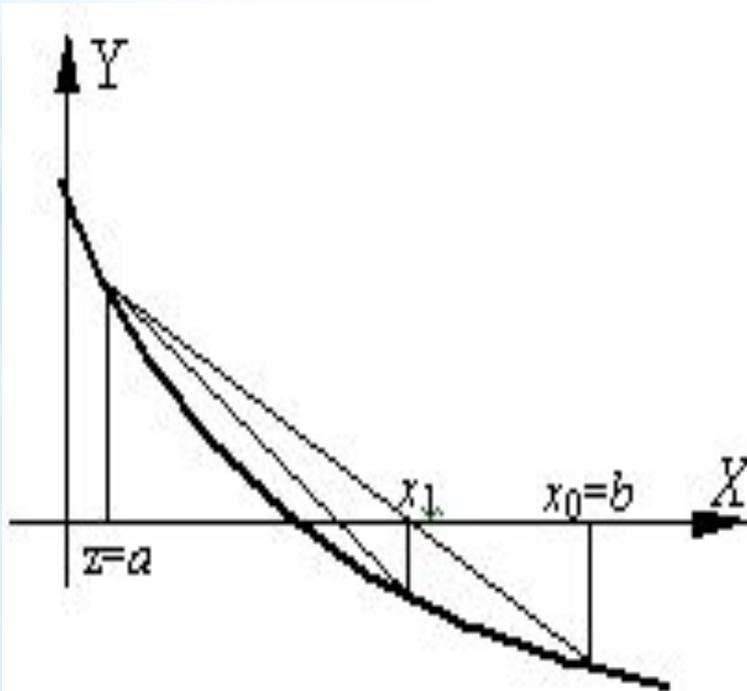
Метод обеспечивает быструю сходимост, если выполняется условие:  $f(x_0) \cdot f''(x_0) > 0$ , т.е. первую касательную рекомендуется проводить в той точке интервала  $[a, b]$ , где знаки функции  $f(x_0)$  и ее кривизны  $f''(x_0)$  совпадают.

# \* Метод хорд

Метод основан на замене функции  $f(x)$  на каждом шаге поиска хордой, пересечение которой с осью  $X$  дает приближение корня.

При этом в процессе поиска семейство хорд может строиться:

- при фиксированном левом конце хорд, т.е.  $z=a$ , тогда начальная точка  $x_0=b$ ;
- при фиксированном правом конце хорд, т.е.  $z=b$ , тогда начальная точка  $x_0=a$ ;



В результате итерационный процесс схождения к корню реализуется рекуррентной формулой:

$$\text{для случая а) } x_{n+1} = x_n - \frac{f(x_n)}{f(x_n) - f(a)} (x_n - a)$$

$$\text{для случая б) } x_{n+1} = x_n - \frac{f(x_n)}{f(x_n) - f(b)} (x_n - b)$$

Процесс поиска продолжается до тех пор, пока не выполнится условие  $|x_n - x_{n-1}| \leq \varepsilon$ .

Метод обеспечивает быструю сходимость, если  $f(z)f''(z) > 0$ , т.е. хорды фиксируются в том конце интервала  $[a, b]$ , где знаки функции  $f(z)$  и ее кривизны  $f''(z)$  совпадают.

# \* Домашнее задание

1. Составить опорный конспект лекции по теме «Итерационные алгоритмы и программы» на основе презентации.

2. Программирование на языке Pascal. Рапаков Г. Г., Ржеуцкая С. Ю. СПб.: БХВ-Петербург, 2004, стр. 123-132.

3. Составить программы:

- Пользователь вводит число. Сколько последовательно расположенных натуральных чисел необходимо перемножить, чтобы их произведение стало больше введенного пользователем числа.
- Найдите сумму членов ряда  $S = 1 + 1/2 + 1/4 + 1/8 + 1/n$ . Остановите вычисления, когда очередное слагаемое станет меньше  $\epsilon$  (вводится пользователем).