Практическая работа № 2.
Анализ алгоритмов и программ.

Задание.

Найти локальный минимум (максимум) функции вида ***f(x) =*** $x^3 - x + e^{-x}$ на заданном интервале ***[a,b]*** с заданной точностью ***ε > 0*** одним из способов:

1) **Методом «деления отрезка» пополам;**
2) **Методом «золотого сечения»;**
3) **Методом «Фибоначчи».**

Провестианализ разработанного алгоритма и программы и сравнить с аналогичным решением с помощью алгоритма «пассивного поиска».

Вход: вид функции, границы отрезка, требуемая точность

Выход (на экране и в файле): таблица пошаговых приближений, количество итераций (шагов поиска), значение точки минимума, значение функции в этой точке.

Справочный материал к заданию.

1. **Время работы программы**

   Директива препроцессора: **`include <ctime>`**

   Начальное время работы программы:**`srand(time(0))`**

   Конечное время работы программы(в секундах): **`clock()/1000.0`**

2. **«Пассивный поиск»**

Графический вид для заданной в задаче функции:

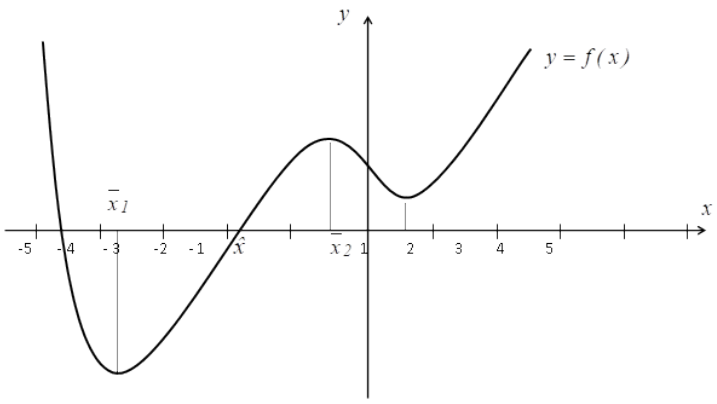


Рис. 1 К методу пассивного поиска

Пример Программы, которая осуществляет поиск локального минимума функции на интервале $[-5, +2]$ с шагом *h = 1*для функции ***f(x) =*** $x^3 - x + e^{-x}$

```
#include <iostream>
#include <Windows.h>
#include <math.h>
#include <stdio.h>
#include <ctime>

using namespace std;

int main()
{   SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
```

```
srand(time(0)); // начальное время
floatmin=0,x,y,k,h=1;// описание используемых в программе переменных
intstep=0;
for(x=-5.0;x<=-2.0;x=x+h)// основной цикл вычисления значений функции на
   { step++;                // заданном промежутке
y=x*x*x-x+exp(-x);
printf("  %d.  x  =  %fy  =  %f\n",step,x,y);  //  вывод  на  экран
if(y<min)  //условиепоиска минимального значения функции
{ min=y;
k=x;
};
printf("минимальное значение функции на  [-5,-2]  \n");
printf("y = %f, точка минимума x = %f\n",min,k);
printf("количество шагов = %d\n",step);
cout<< "Время работы программы = " <<clock()/1000.0<<endl;

cout<< endl<<endl;
}
```

Ответы:

1) Если **h=1**, то

```
 1. x = -5.000000  y = 28.413158
 2. x = -4.000000  y = -5.401850
 3. x = -3.000000  y = -3.914463
 4. x = -2.000000  y = 1.389056
минимальное значение функции на  [-5,-2]
y = -5.401850, точка минимума x = -4.000000
количество шагов = 4
Время работы программы = 0.013 сек.
```

2) Если **h=0.1**, то

```
 28. x = -2.300003  y = 0.107170
 29. x = -2.200003  y = 0.577002
 30. x = -2.100003  y = 1.005159
 31. x = -2.000003  y = 1.389046
минимальное значение функции на  [-5,-2]
y = -6.505695, точка минимума x = -3.700001
количество шагов = 31
Время работы программы = 0.053 сек.
```

3) Если **h=0.01**, то

```
 297. x = -2.039979  y = 1.241024
 298. x = -2.029979  y = 1.278737
 299. x = -2.019979  y = 1.315994
 300. x = -2.009979  y = 1.352792
минимальное значение функции на  [-5,-2]
y = -6.509638, точка минимума x = -3.679978
количество шагов = 300
Время работы программы = 0.585 сек.
```

4) Если **h=0.001**, то

```
 2998. x = -2.003217  y = 1.377415
 2999. x = -2.002217  y = 1.381038
 3000. x = -2.001217  y = 1.384657
 3001. x = -2.000217  y = 1.388271
минимальное значение функции на  [-5,-2]
y = -6.509648, точка минимума x = -3.679096
количество шагов = 3001
Время работы программы = 4.927 сек.
```

5) Если **h=0.0001**, то

```
 30004. x = -2.000398  y = 1.387619
 30005. x = -2.000298  y = 1.387980
 30006. x = -2.000198  y = 1.388341
 30007. x = -2.000098  y = 1.388701
минимальное значение функции на  [-5,-2]
y = -6.509648, точка минимума x = -3.679074
количество шагов = 30007
Время работы программы = 54.507 сек.
```

6) Если **h=0.00001**, то

```
 299591. x = -2.000031  y = 1.388942
 299592. x = -2.000021  y = 1.388979
 299593. x = -2.000011  y = 1.389015
 299594. x = -2.000001  y = 1.389051
минимальное значение функции на  [-5,-2]
y = -6.509648, точка минимума x = -3.679099
количество шагов = 299594
Время работы программы = 509.268 сек.
```

≈ 8.5 минут!!!

Очевидно, что с ростом количества вычислений пропорционально (с некоторым коэффициентом) растет и время работы программы. Легко подсчитать, что сложность алгоритма равна **O(n).**
Также видно, что точные значения и функции, и точки, в которой эта функция достигает минимума можно определить только с некоторой погрешностью.
Поэтому для вычисления экстремумов функций используют методы, которые определяют значение с некоторой, заранее заданной точностью. Например, чтобы приблизительное решение отличалось от «истинного» с погрешностью ***ε=0.001.***Из примера «пассивного поиска» можно заметить, что искомый минимум лежит где-то в интервале **|-3.679099** $\pm$ ***ε*** **|.**

3. Метод «деления отрезка пополам»

Поставим задачу: найти отрезок длины не более $\varepsilon < a - b$, содержащий точку $\overline{x}$ локального минимума. Другими словами, найти точку $\tilde{x}$, с точностью до ε приближающую точку $\overline{x}$:

$$\left| \overline{x} - \tilde{x} \right| < \varepsilon .$$

Для удобства пояснения алгоритма метода обозначим исходный отрезок $[\, a,\, b\,] = [\, a_0, b_0\,]$ (рис. 2.), возьмем $0 < \delta \le \dfrac{b - a}{10}$, построим точки

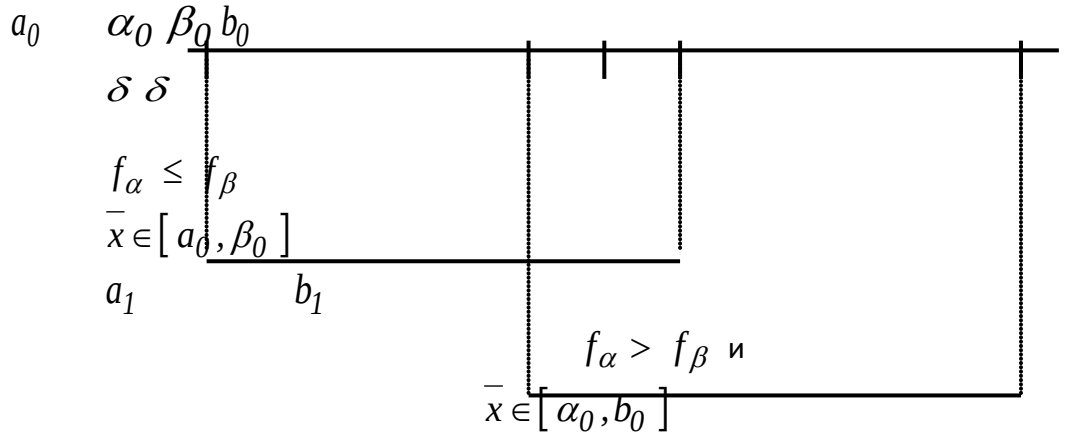


Рис.2. Графическая интерпретация метода

$\alpha_0 = \dfrac{1}{2}\,(\, a_0 + b_0\,) - \delta,\ \beta_0 = \alpha_0 + 2\delta$ и вычислим $f_\alpha = f(\alpha_0),\ f_\beta = f(\beta_0).$

Если $f_\alpha \le f_\beta$, то заключаем, что $\bar{x} \in [\, a_0, \beta_0\,]$. Если же $f_\alpha > f_\beta$, то $\bar{x} \in [\, \alpha_0, b_0\,]$.

В первом случае в качестве приближения x_1 точки $\bar{x}$ возьмем $x_1 = \alpha_0$ и примем обозначение $a_1 = a_0, b_1 = \beta_0$. Во втором случае – $a_1 = \alpha_0, x_1 = \beta_0, b_1 = b_0$. Таким образом, в $[\, a_0, b_0\,]$ найден отрезок $[\, a_1, b_1\,]$, где содержится $\bar{x}$ и указано приближение $x_1 \approx \bar{x}$. Если $b_1 - a_1 < \varepsilon$, то поставленная задача решена. В противном случае отрезок $[\, a_1, b_1\,]$ следует подвергнуть аналогичному дроблению (схема 2) и продолжить процесс дробления до тех пор, пока на некотором шаге n не выполнится неравенство $b_n - a_n < \varepsilon$. При этом на отрезке $[a_n, b_n]$ будет получено приближение $x_n \approx \bar{x}$ – точка, в которой функция $f(x)$ достигает приближенное значение локального минимума: $f(\bar{x}) \approx f(x_n)$.

Приведенный алгоритм является классическим. Заметим, что в нем на каждом шаге необходимо произвести два вычисления значения функции: $f_\alpha = f(\alpha_0),\ f_\beta = f(\beta_0).$

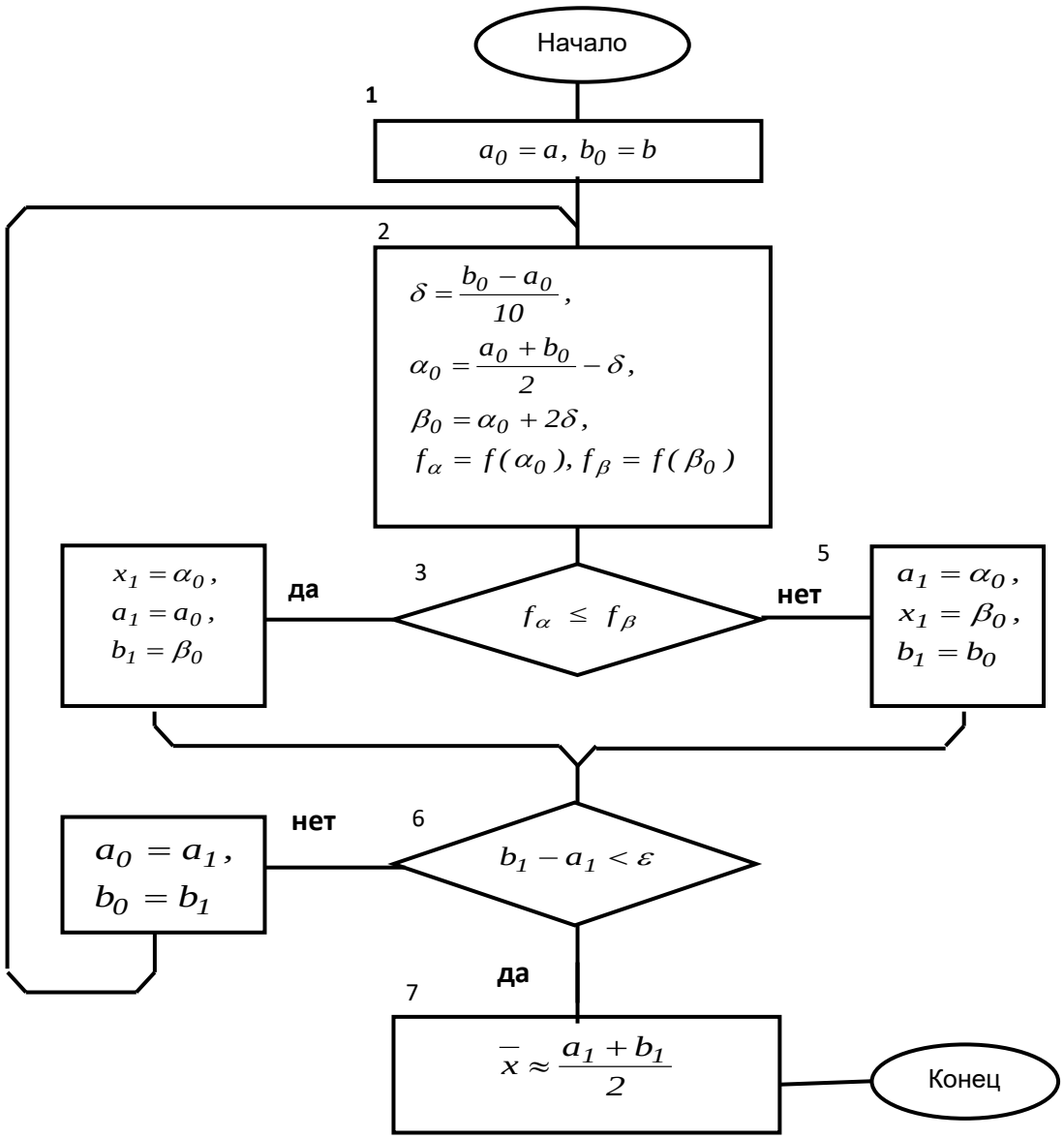


Схема 2.Метод “деления отрезка пополам”

Пример.

Определим с точностью $\varepsilon = 10^{-3}$ точку $\bar{x}$ локального минимума функции $f(x)=x^3 - x + e^{-x}$. Поиск будем производить на отрезке локализации $[-4, -3]$.

Установим $a_0 = a = -4,\ b_0 = b = -3$.

Первая итерация:

1) вычислим $\delta = \frac{b_0 - a_0}{10} = 0.1,$ $\alpha_0 = \frac{a_0 + b_0}{2} - \delta = -3.6,$

$\beta_0 = \alpha_0 + 2\delta = -3.4,$ $f_\alpha = f(\alpha_0) = -6.457766,$ $f_\beta = f(\beta_0) = -5.939899;$

2) сравним f_α и f_β. Так как $f_\alpha \le f_\beta$, то установим в качестве нового отрезка локализации отрезок $[a_1 = a_0 = -4,\ b_1 = \beta_0 = -3.4]$;

3) и так как условия $\Delta_1 = b_1 - a_1 < \varepsilon$ (и $b_1 - a_1 < 2\varepsilon$) не выполняются , то произведем очередную итерацию.

Итерационный процесс будем продолжать до тех пор, пока для установленного вновь отрезка локализации данные условия не будут выполнены. Результаты итераций сведем в таблицу 1.

Таблица 1

№ шага k	a_k	b_k	α_k	β_k	f_α	f_β	Δ_k
0	-4.000000	-3.000000	-3.600000	-3.400000	-6.457766	-5.939899	1.000000
1	-4.000000	-3.400000	-3.760000	-3.640000	-6.448950	-6.496707	0.600000
2	-3.760000	-3.400000	-3.616000	-3.544000	-6.476333	-6.363350	0.360000
3	-3.760000	-3.544000	-3.673600	-3.630400	-6.509402	-6.489656	0.216000
4	-3.760000	-3.630400	-3.708160	-3.682240	-6.502006	-6.509551	0.129600
5	-3.708160	-3.630400	-3.677056	-3.661504	-6.509618	-6.507023	0.077760
6	-3.708160	-3.661504	-3.689497	-3.680166	-6.508658	-6.509634	0.046656
7	-3.689497	-3.661504	-3.678300	-3.672701	-6.509645	-6.509312	0.027994
8	-3.689497	-3.672701	-3.682779	-3.679420	-6.509517	-6.509646	0.016796
9	-3.682779	-3.672701	-3.678748	-3.676732	-6.509648	-6.509607	0.010078
10	-3.682779	-3.676732	-3.680360	-3.679151	-6.509630	-6.509648	0.006047
11	-3.680360	-3.676732	-3.678909	-3.678183	-6.509648	-6.509644	0.003628
12	-3.680360	-3.678183	-3.679489	-3.679054	-6.509645	-6.509648	0.002177
13	-3.679489	-3.678183	-3.678967	-3.678706	-6.509648	-6.509648	0.001306
14	-3.679489	-3.678706	-3.679176	-3.679019	-6.509648	-6.509648	0.000784

Из таблицы видно, что искомый результат на 14-ом шаге вычислений. При этом:

$$\bar{x} \approx \frac{b_{14} - a_{14}}{2} = -3.679098, \quad f(\bar{x}) \approx -6.509648 .$$

4. Метод «золотого сечения»

Отличие метода золотого сечения от метода деления отрезка пополам заключается в специальном разбиении отрезка локализации относительно его центра точками.

Термин «золотое сечение» ввел Леонардо да Винчи. Принципы, заложенные в основу этого сечения, широко использовались при композиционном построении многих произведений искусства античности и эпохи Возрождения: особенно в живописи и архитектуре.

Определение. *Золотым сечением* отрезка $[a,b]$ называется такое разбиение отрезка на две неравные части точками α и β, что отношение длины всего отрезка $\Delta = b - a$ к длине его большей части Δ^+ равно отношению длины большей части к длине Δ^- меньшей части: $\frac{\Delta}{\Delta^+} = \frac{\Delta^+}{\Delta^-}$. При этом (рис.2.5) точки α и β располагаются симметрично относительно центра отрезка $[a,b]$ и могут быть определены с помощью следующих соотношений:

$$\alpha = a + \frac{2\Delta}{3+\sqrt{5}}, \quad \beta = a + \frac{2\Delta}{1+\sqrt{5}}.$$

Замечательно также, что точки α и β осуществляют золотое сечение не только всего отрезка $[a,b]$, но и соответственно подотрезков $[a,\beta]$ и $[\alpha,b]$.

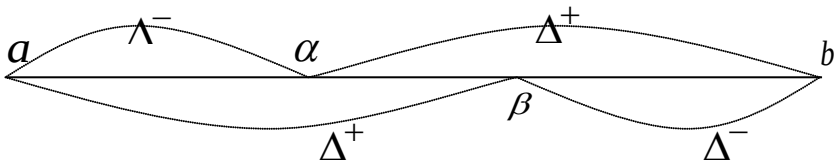


Рис.3. Выбор точек разбиения отрезка в методе

Итерационный процесс локализации минимума функции на заданном числовом отрезке в данном методе ведется аналогично локализации минимума функции в методе деления отрезка пополам. В отличие от него точки α_k и β_k на очередном шаге итерации находятся по формулам:

$$\alpha_k = a_k + \frac{2\Delta_k}{3+\sqrt{5}}, \quad \beta_k = a_k + \frac{2\Delta_k}{1+\sqrt{5}}.$$

Свойства золотого сечения позволяют также несколько упростить процедуру поиска точки локализации x_{k+1} на очередном шаге вычислений. Действительно, какой бы из отрезков $[a_k, \beta_k]$ или $[\alpha_k, b_k]$ не был бы выбран за очередной отрезок локализации, точка x_{k+1} ($x_{k+1} = \alpha_k$, если $x_{k+1} \in [a_k, \beta_k]$ и $x_{k+1} = \beta_k$, если $x_{k+1} \in [\alpha_k, b_k]$) совпадет с одной из точек α_{k+1} или β_{k+1} (рис 4). Поэтому на очередном шаге достаточно вычислить значение функции лишь в одной недостающей точке.

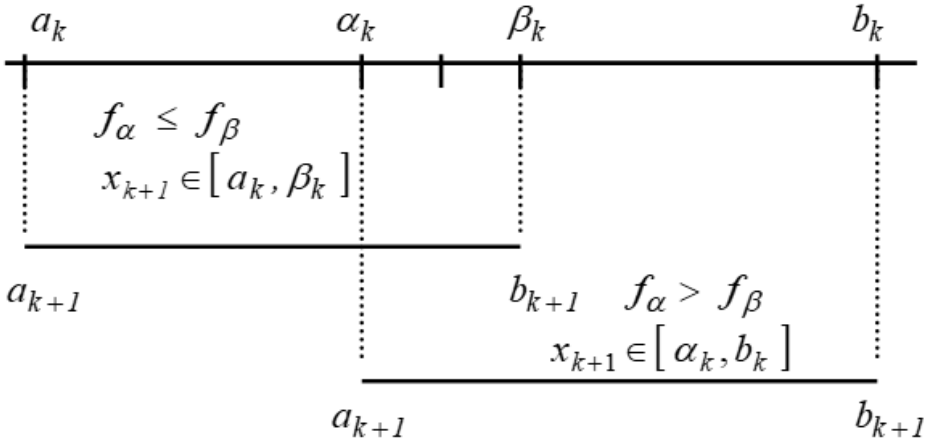


Рис.4. Графическая интерпретация метода золотого сечения

5. Метод Фибоначчи

Данный метод обеспечивает максимальное гарантированное сокращение отрезка локализации при заданном числе *N* вычислений функции и основан на использовании чисел Фибоначчи F_n таких, что: $F_{n+1} = F_{n-1} + F_n$ для всех $n \ge 2$ при начальных значениях $F_0 = 1, F_1 = 1.$

Метод Фибоначчи состоит из $N-1$ шагов.

Вначале определяется такое число Фибоначчи F_n, для которого справедливо условие $\frac{\Delta_0}{F_{n+1}} \le \varepsilon$, где $\Delta_0 = \beta_0 - \alpha_0$.

Очередной $(k+1)$-й шаг выполняется аналогично $(k+1)$-й итерации метода деления отрезка пополам, но точки α_k и β_k находятся по формулам

$$\alpha_k = a_k + \frac{F_{n-k-1}}{F_{n-k+1}}\Delta_k, \qquad \beta_k = a_k + \frac{F_{n-k}}{F_{n-k+1}}\Delta_k,$$

где $\Delta_k = b_k - a_k$ – длина отрезка локализации $[a_k, b_k]$ (рис.2.7).

Аналогично методу деления отрезка пополам определяется новый отрезок локализации:

если $f(\alpha_k) \le f(\beta_k)$, то $\bar{x} \in [a_k, \beta_k]$;

если $f(\alpha_k) > f(\beta_k)$, то $\bar{x} \in [\alpha_k, b_k]$.

В первом случае за очередное приближение к точке минимума $\bar{x}$ принимают $x_{k+1} = \alpha_k$, во втором случае – $x_{k+1} = \beta_k$.

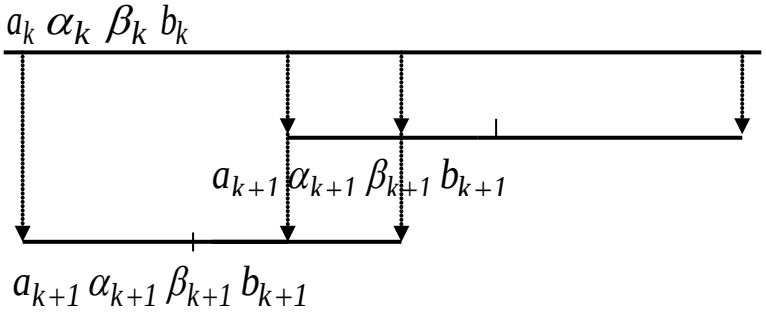
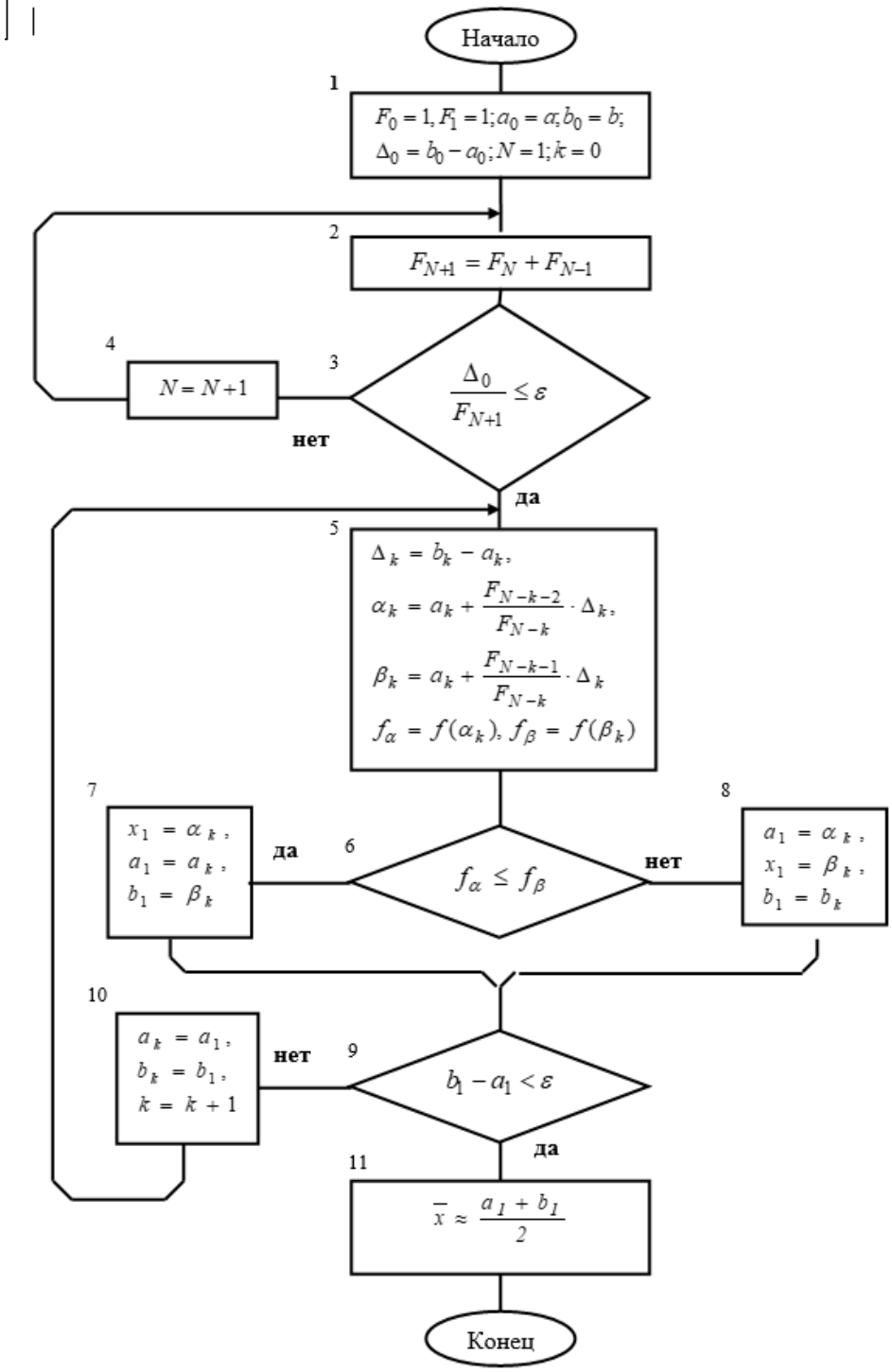


Рис. 5. Графическая интерпретация метода Фибоначчи

Примерная схема алгоритма:



Все 4 метода реализовать для поиска экстремальных точек в интервалах переменной x—[-5,-3], [0,3] —точки минимума и [-3,0] — точка максимума.