# Подпрограммы – параметры других подпрограмм (манипуляторы функций MATLAB)

*лекция №6*

# В каких задачах используются процедуры-параметры?

- Использование параметра-подпрограммы необходимо, когда некоторый алгоритм, описанный как подпрограмма, применим к множеству алгоритмов, каждый из которых также задается подпрограммой.
- **Классические примеры** таких ситуаций дают численные методы. В подпрограммах численных методов (вычисления определенного интеграла, нахождения экстремумов и нулей функций, вывода графиков, линий уровня, таблиц функций) обрабатываемые функции задаются как параметры.
- Средства для использования параметров-подпрограмм имеются во всех алгоритмических языках, предназначенных для решения вычислительных задач (СИ, Фортран, MatLab, …).

# Описание функции в MATLAB

*формальные параметры, хранятся в рабочей области (памяти) функции*

function [*СписокВыхода*]=*ИмяФункции*(*СписокВхода*)

% *комментарии*

*исполняемые операторы*

*Здесь могут быть имена функций – формальных параметров*

# MATLAB: инструмент для работы с функциями-параметрами – манипулятор функции

1. Манипулятор функции – это ссылка на функцию (можно считать адресом входа в функцию). Обозначается символом @.
2. **В простейшем случае** это возможность **переобозначения** функции, например:

```
>> h=@sin
h =
  @sin
>> y=sin(pi/6)
y =
   0.5000
>> y=h(pi/6)
y =
  0.5000
```

3. Функция feval позволяет вычислить значение функции по ее манипуляру и аргументу: feval(*манипулятор*, *аргумент*). Например:

```
>> y=feval(h,pi/6) % эквивалентно y=h(pi/6)
y =
   0.5000
```

```
>> y=feval(@sin,pi/6)
y =
   0.5000
```

# Манипулятор функции может использоваться как формальный входной параметр другой функции

**Пример:** функция plot_fhandle строит график для заданной функции одной переменной и диапазона аргумента:

```
function x = plot_fhandle(fh, data)
plot(data, feval(fh, data))
```

*Это манипулятор функции –формальный параметр*

%можно так: plot(data, fh (data))

Вызов функции plot_fhandle:

```
>> plot_fhandle(@sin, -pi:0.01:pi)
>> plot_fhandle(@log, 0.1:0.01:3)
```

*Что делаю эти операторы?*

*Фактические параметры - манипуляторы функций*

Фактическая функция должна иметь такой же заголовок, как формальная функция с точностью до обозначений.

# Пример: приближенное решение уравнения на отрезке

Известно, что уравнение

$$F(x)=0 \qquad (*)$$

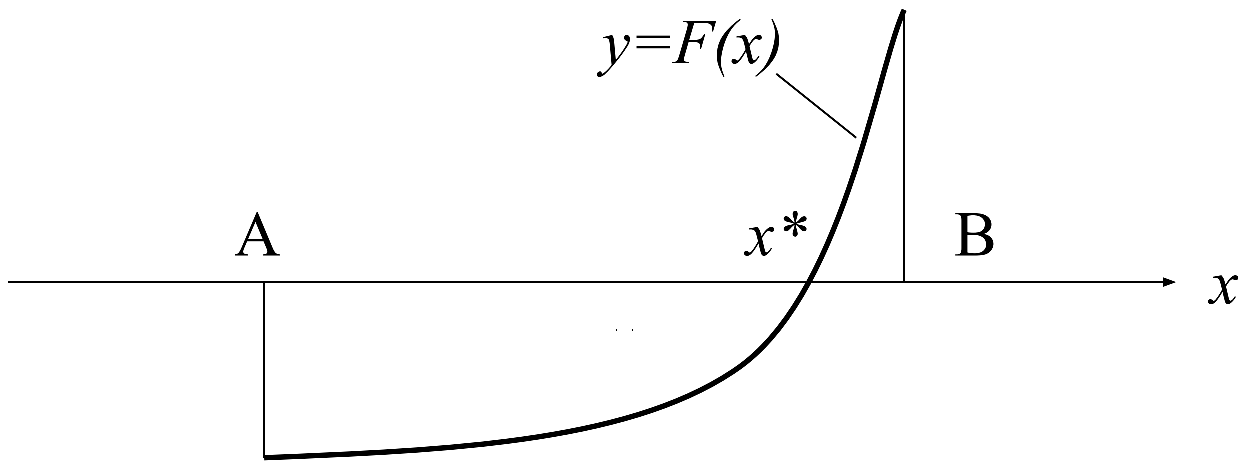
на отрезке [A,B] имеет ровно один корень.

Требуется найти приближенное значение корня с точностью ε:

$$|x^* - x_{пр}| < \varepsilon,$$

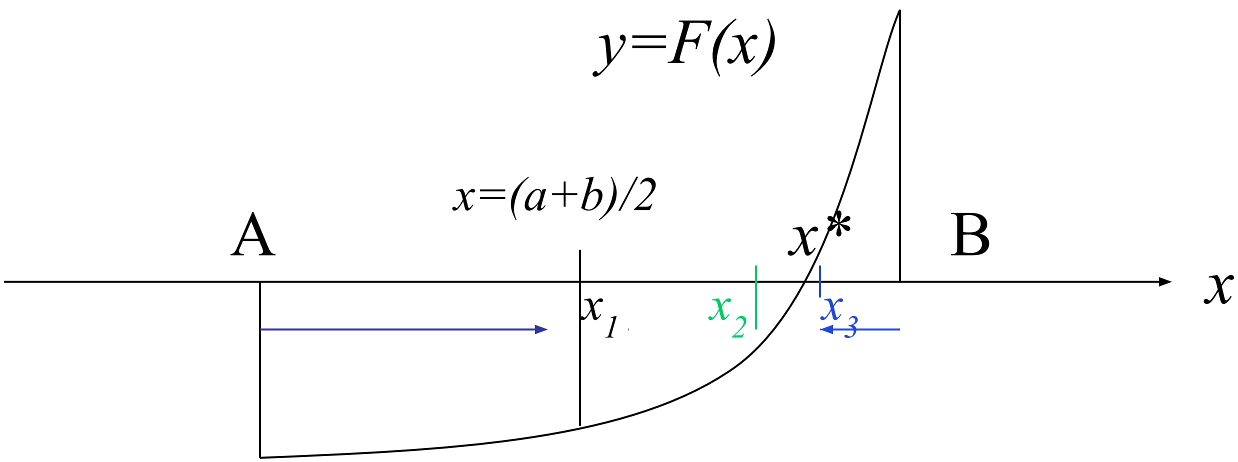
где $x^*$ - точное значение корня,
$x_{пр}$ – приближенное значение корня.

# Приближенное решение уравнения на отрезке




Если уравнение (*) имеет на отрезке [A,B] ровно один корень, то $\boldsymbol{F(A)*F(B)\leq 0}$.

# Метод деления отрезка пополам (дихотомии)




Если $F(x)*F(\mathrm{B})\le 0$, то $x^*\in[x,\mathrm{B}]$ $\Rightarrow$ корень надо искать на правой половине отрезка: **A=*x***;

иначе $x^*\in[A,x]$ $\Rightarrow$ корень надо искать на левой половине отрезка: **B=*x*.**

**Далее деление пополам нового отрезка.**

# Метод деления отрезка пополам (дихотомии)

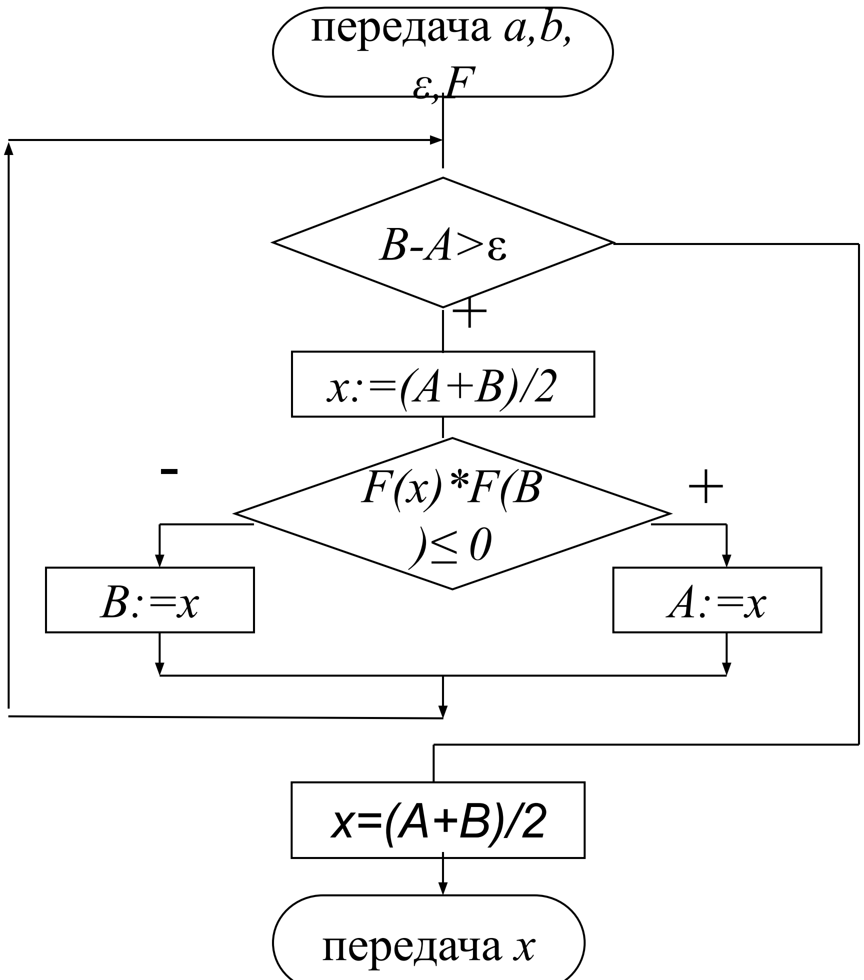
i-ая итерация (цикл): вычисление $x_i$ - середины i-го отрезка и выбор его левой или правой половины.

$$\{x_i\} \rightarrow x^* \text{ при } i \rightarrow \infty.$$

Условие продолжения цикла: $B\text{-}A > \varepsilon$.

# Метод деления отрезка пополам (дихотомии) – блок-схема функции root

*Алгоритм для идеального случая: на [A,B] ровно один корень.*




*Можно определить число N итераций (циклов), необходимых для обеспечения погрешности* ε. *В конце N-го цикла длина отрезка, накрывающего корень, равна:*

$$l = \frac{B - A}{2^N}.$$

*Число итераций можно вычислить из соотношения:*

$$l \leq \varepsilon.$$

*Откуда:*

$$\log_2(B-A) - N \leq \log_2(\varepsilon),$$

*и, следовательно,*

$$N = \lceil \log_2(B-A) - \log_2 \varepsilon \rceil,$$

*где* ⌈ .⌉ - *ближайшее максимальное целое.*

# Функция вычисления корня уравнения методом деления отрезка пополам

```
function x=root(a,b,eps,F)
%вычисление корня уравнения методом деления отрезка пополам
while b-a>eps
    x=(a+b)/2;
    if F(x)*F(b)<=0
        a=x;
    else
        b=x;
    end
end
x=(a+b)/2;
```

**Задача 1.8.** Методом дихотомии найдите корень уравнения *F(x) = 0* на отрезке [*a, b*] с заданной погрешностью *E*. Используйте в качестве *F(x)* формулу из табл. 1 и значения границ отрезка *a* = 0,1 и *b* = 1.

1.8.1:

$$F(x) = \frac{\ln(x+1)}{0{,}001 + \sqrt[4]{x}\sin^2 x} - \frac{1}{\pi x \sqrt[3]{x}} - e^{x/7}$$

```
function y=f_1_8_1(x)
y=log(x+1)./(0.001+x.^(1/4).*sin(x).^2)-1./(pi.*x.*x.^(1/3))-exp(x./7);
```

Вызов функции root:

```
>> coren=root(0.1,1,0.0001,@f_1_8_1)

coren =

   0.5774
```

# Как протестировать функцию root?

Вывести не только корень уравнения, но и значение функции в корне. Это значение функции должно быть близко к нулю. Если оно сильно отличается от нуля, то root работает неправильно. Однако близость f_1_8_1 в точке корня к нулю не гарантирует правильность программы.

Как протестировать функцию root? – построить график функции, лучше не только в среде MATLAB, но и в другой вычислительной среде.

Как протестировать функцию root? – применить ее к уравнению, корень которого известен.

$$x^2 - 0,5x = 0$$

```
function y=fsimple(x)
y=x.*x-0.5*x;
```

```
>> y=root(0.1,1,0.0001,@fsimple)

y =

    0.5000
```

# Ситуация, в которой рекомендуется использовать глобальные переменные

Пусть надо решить уравнение, заданное с точностью до параметра ***p*,** например, ***p*** задается вводом:

$$x - p\ cosx=0.$$

В программе надо обратиться к ***root***, подставив вместо формального параметра *F* фактический

$$g(x,p)= x - p\ cosx.$$

Но *g* имеет два аргумента, а *F* один. Выход из этой ситуации состоит в том, чтобы параметр ***p*** считать **глобальным** (если функцию root изменять нельзя).

# Описание функции, использующей глобальную переменную

```
function y=fglobal(x)
global p
y=x-p.*cos(x);
```

## Пример вызова функции fglobal:

```
global p
i=1;
for p=0.3:0.1:0.6
    z(i)=root(0.1,1,0.0001,@fglobal); i=i+1;
end
>> z

z =

   0.2877    0.3725    0.4502    0.5205
```

# Класс Function Functions

Функции этого класса работают с нелинейными функциями скалярного аргумента как с функциями-параметрами.

Класс предназначен для решения следующих задач:

- нахождение нулей функций (решение уравнений);
- оптимизация;
- вычисление определенных интегралов;
- обыкновенные дифференциальные уравнения.

# Некоторые функции класса Function Functions

- fminsearch(*манипулятор_функции*, *начальное_приближение*) вычисляет точку локального минимума функции;
- fmaxsearch(*манипулятор_функции*, *начальное_приближение*) вычисляет точку локального максимума функции;
- fzero(*манипулятор_функции*, *начальное_приближение*) вычисляет точку локальный нуль функции;
- quad(манипулятор_функции, нижняя_граница, верхняя_граница) вычисляет определенный интеграл по методу Симпсона.

# Примеры работы функций класса Function Functions

Исследуем функцию:
function y = humps(x)
y = 1./((x-.3).^2 + .01) + 1./((x-.9).^2 + .04) - 6;

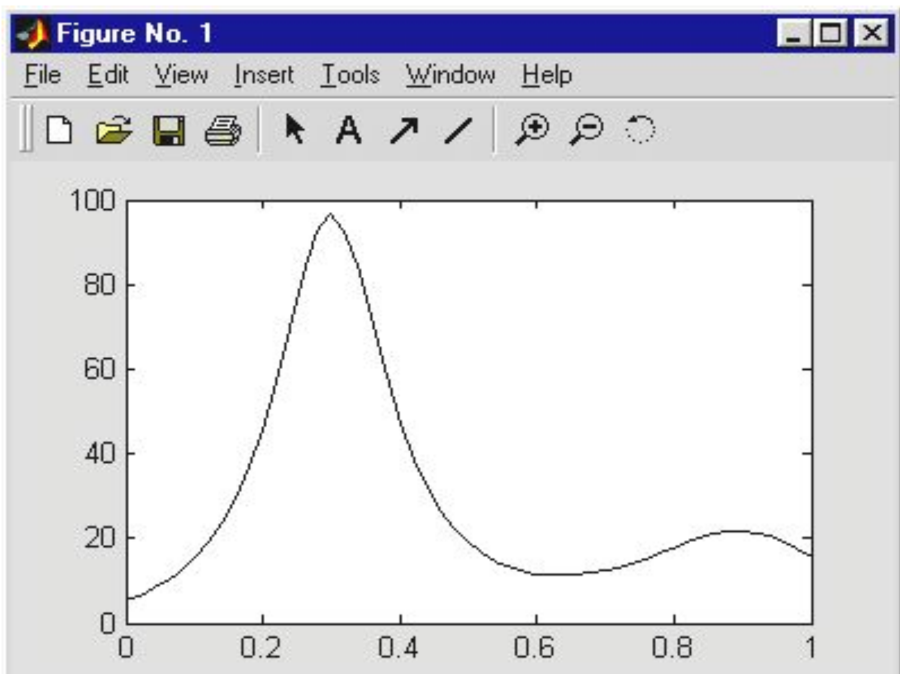



Рис. 4.1. График humps(x)