§3. Численное решение нелинейных уравнений

Приближенное или численное нахождение изолированных действительных корней нелинейного уравнения $f(x)=0$ состоит из двух этапов:

1) ***отделение корней***, т.е. установление возможно тесных промежутков, в каждом из которых содержится один, и только один, корень уравнения;

2) ***уточнение корней***, т.е. доведение их до заданной степени точности.

Отметим два способа отделения действительных корней уравнения – *аналитический* и *графический*.

Для ***аналитического*** **отделения корней** используют следующие теоремы математического анализа:

1. Теорема Больцано–Коши. Если функция $f(x)$ непрерывна на отрезке $[a,b]$ и на его концах принимает значения разных знаков, т.е.

$$f(a)\cdot f(b)<0,$$

то внутри отрезка $[a,b]$ существует по крайней мере один корень уравнения $f(x)=0$.

2. Если функция $f(x)$ непрерывна на отрезке $[a,b]$, $f(a)\cdot f(b)<0$, производная $f'(x)$ существует и сохраняет постоянный знак в интервале (a,b), то внутри отрезка $[a,b]$ существует ровно один корень уравнения $f(x)=0$.

Таким образом, чтобы отделить корни аналитически, нужно:

- найти критические точки $x_1, x_2, ..., x_k, ..$ функции $f(x)$, т.е. точки из области определения функции, в которых производная $f'(x)$ равна нулю, бесконечности или в которых она не существует;

- вычислить значения функции $f(x)$ в этих точках и ее предельные значения на концах области определения;

- по знакам функции $f(x)$ и ее производной $f'(x)$ определить интервалы, в которых уравнение имеет ровно один корень;

- сузить полученные интервалы до нужной длины, так чтобы на его концах функция принимала значения разных знаков.

Критические точки также могут оказаться корнями четной кратности функции $f(x)$.

Решив задачу *отделения корней*, фактически, находят приближенные значения корней с погрешностями, не превосходящими длины отрезка, содержащего тот или иной корень.

Графическое **отделение корней** состоит в построении графика функции $y = f(x)$ и нахождении тех значений x, при которых график пересекает ось абсцисс. Они и являются корнями уравнения $f(x) = 0$.

Если график функции $y = f(x)$ построить трудно, то от исходного уравнения $f(x) = 0$ следует перейти к равносильному уравнению вида

$$\varphi_1(x) = \varphi_2(x)$$

таким образом, чтобы графики функций $y = \varphi_1(x)$ и $y = \varphi_2(x)$ были достаточно просты. Абсциссы точек пересечения этих графиков и будут корнями уравнения.

После выполнения этапа отделения корней переходят к следующему этапу приближенного решения уравнений – ***уточнению корней.***

Пусть искомый корень уравнения отделен, т.е. найден отрезок $[a, b]$, на котором имеется только один корень уравнения. Для вычисления этого корня с требуемой точностью ε строят последовательность приближений x_n, сходящуюся к нему. Начальное приближение x_0 выбирают из отрезка $[a, b]$. Вычисления продолжают до тех пор, пока не будет выполнено неравенство

$$|x_n - x_{n-1}| < \varepsilon.$$

При этом считают, что x_n – это и есть корень уравнения, найденный с заданной точностью ε.

При выборе метода построения последовательности приближений к корню большую роль играют такие свойства метода как простота, надежность, экономичность. Одной из его важнейших характеристик является *скорость сходимости*.

Последовательность (x_n), сходящаяся к числу x^*, имеет скорость сходимости порядка α, если

$$|x_n - x^*| = O\left(|x_n - x^*|^{\alpha}\right) \text{ при } n \to \infty.$$

При $\alpha = 1$ сходимость называется линейной, при $1 < \alpha < 2$ – сверхлинейной, при $\alpha = 2$ – квадратичной. С ростом α алгоритм, как правило, усложняется и условия сходимости становятся более жесткими.

Рассмотрим наиболее распространенные из итерационных методов уточнения корней уравнения $f(x)=0$.

Метод половинного деления (бисекции, дихотомии).

Это простейший и надежный алгоритм уточнения простого корня.

Пусть на отрезке $[a,b]$ содержится только один корень уравнения $f(x)=0$. В качестве первого приближения x_1 к корню берут середину отрезка $[a,b]$. Если $f(x_1)=0$, то процесс окончен. В противном случае в качестве нового отрезка $[a_1,b_1]$ выбирают ту половину $[a,x_1]$ или $[x_1,b]$ отрезка $[a,b]$, на концах которой функция принимает значения разных знаков. Очередное приближение к корню является серединой такого отрезка. И так далее.

Расчетные формулы метода имеют вид:

$$x_n=(a_{n-1}+b_{n-1})/2,$$

$$a_n=a_{n-1},\ \ b_n=x_n,\ \text{если}\ \ f(a_{n-1})\cdot f(x_n)<0,$$

$$a_n=x_n,\ \ b_n=b_{n-1},\ \text{если}\ \ f(a_{n-1})\cdot f(x_n)>0.$$

где $[a_0,b_0]=[a,b],\ n=1,2,\ldots$

Погрешность между точным значением корня ξ и приближенным x_n не превосходит длины отрезка $[a_n,b_n]$:

$$|\xi-x_n|\le b_n-a_n=\frac{b-a}{2^n}.$$

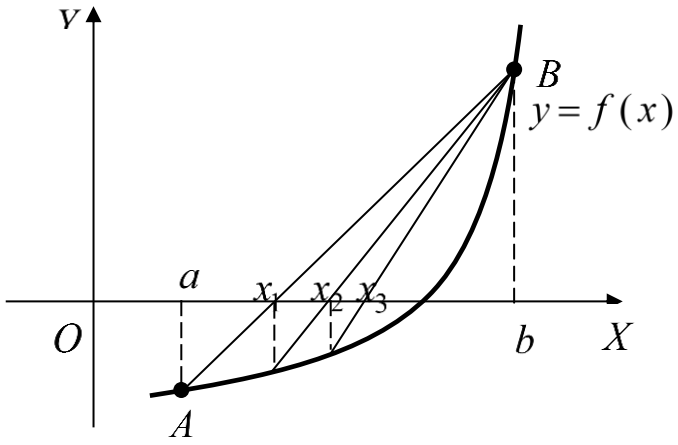
Метод сходится для любых непрерывных функций, в том числе недифференцируемых, устойчив к ошибкам округления. Скорость сходимости линейная. Для достижения заданной точности ε требуется $\log_2\frac{b-a}{\varepsilon}$ итераций.

Метод неприменим для нахождения корней четной кратности.

Метод хорд (пропорциональных частей (или отрезков)).

Метод хорд является более быстрым, чем метод половинного деления, способом нахождения корня уравнения $f(x)=0$.

Пусть функция $f(x)$ непрерывна на $[a,b]$ и на этом отрезке существует ровно один корень уравнения. Метод хорд сходится, если вторая производная $f''(x)$ сохраняет знак на отрезке $[a,b]$.



За последовательное приближение x_{n+1} к корню принимают абсциссу точки пересечения с осью OX хорды, соединяющей точки $(x_n, f(x_n))$ и $(a, f(a))$ (либо $(b, f(b))$. Возможны два случая:

1) если $f(a)\cdot f''(x)>0$ на отрезке $[a,b]$, то

$$\begin{cases} x_0 = b, \\ x_{n+1} = a - \dfrac{f(a)}{f(x_n)-f(a)}(x_n - a) \end{cases}$$

или, что равносильно, $x_{n+1} = x_n - \dfrac{f(x_n)}{f(x_n)-f(a)}(x_n - a)$;

2) если $f(b)\cdot f''(x)>0$ на отрезке $[a,b]$, то

$$\begin{cases} x_0 = a, \\ x_{n+1} = x_n - \dfrac{f(x_n)}{f(b)-f(x_n)}(b - x_n). \end{cases}$$

Метод сходится линейно, но близость двух очередных приближений не всегда означает, что корень найден с требуемой точностью. Более надежным практическим критерием окончания итераций в методе хорд является выполнение неравенства

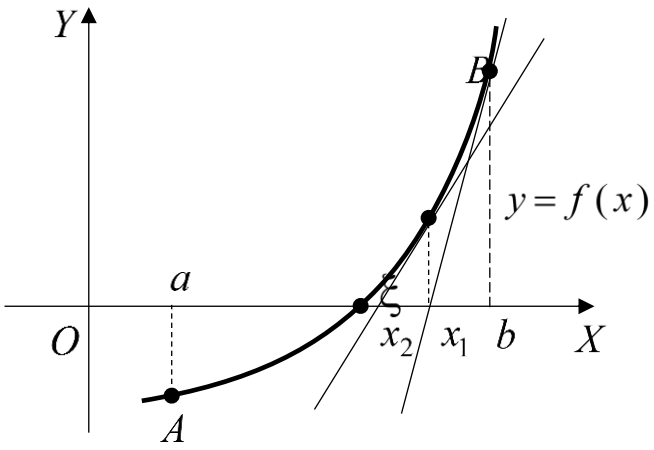
$$\frac{(x_{n+1}-x_n)^2}{|x_{n+1}+x_{n-1}-2x_n|} < \varepsilon .$$

Метод Ньютона (касательных).

Пусть функция $f(x)$ непрерывна на отрезке $[a,b]$, ее производные $f'(x)$ и $f''(x)$ сохраняют знак на отрезке $[a,b]$.

Если x_n – n-е приближение к корню уравнения $f(x)=0$, то в качестве следующего приближения x_{n+1} берут абсциссу точки пересечения с осью OX касательной к графику функции $f(x)$ в точке x_n. Таким образом,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$



Метод Ньютона сходится при любом начальном приближении x_0, но скорость сходимости существенно зависит от выбора точки x_0. В качестве начального приближения x_0 следует брать тот конец отрезка $[a, b]$, для которого выполняется условие

$$f(x_0) \cdot f''(x_0) > 0.$$

В этом случае метод имеет квадратичную скорость сходимости для простого корня.

Модифицированный метод Ньютона.

Чтобы избежать многократного вычисления производной функции $f(x)$ в методе Ньютона, используют следующую расчетную формулу:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_0)},$$

где $f'(x_0) \neq 0$.

Метод имеет линейную скорость сходимости.

Метод секущих.

Этот метод также является модификацией метода Ньютона, в котором производная $f'(x_n)$ заменена ее разностным приближением:

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

Таким образом, расчетная формула метода имеет вид:

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \cdot f(x_n).$$

Метод является двушаговым, каждое новое приближение x_{n+1} к корню строится с использованием двух предыдущих приближений x_{n-1} и x_n, поэтому для начала итерационного процесса следует выбрать два приближения x_0 и x_1 из отрезка $[a, b]$. Метод имеет линейную скорость сходимости.

Метод простой итерации (последовательных приближений, итераций).

Пусть дано уравнение $f(x) = 0$, где $f(x)$ – функция, непрерывная на отрезке $[a, b]$.

В первую очередь от этого уравнения следует перейти к равносильному уравнению вида

$$x = \varphi(x),$$

что может быть выполнено бесконечным числом способов (один из них указан ниже).

Расчетная формула метода имеет вид:

$$x_{n+1} = \varphi(x_n),$$

где x_0 (начальное приближение) – любое число из отрезка $[a, b]$.

Достаточным условием сходимости итерационного процесса (при любом начальном приближении) является выполнение неравенства

$$|\varphi'(x)| < 1 \text{ для любого } x \in [a, b].$$

Например, в качестве $\varphi(x)$ можно взять функцию вида

$$\varphi(x) = x - \lambda f(x),$$

причем λ должно удовлетворять условию

$$|\lambda| < \frac{2}{M},$$

где $M = \max\limits_{[a,b]} |f'(x)|$ и знак числа λ совпадает со знаком производной $f'(x)$ на отрезке $[a, b]$.

Такой вариант перехода к уравнению, пригодному для итераций (т.е. $x = \varphi(x)$), также называют **методом релаксации** (ослабления).

Метод имеет линейную скорость сходимости.

Если $|\varphi'(x)| \le q < 1$ для любого $x \in [a, b]$, то точность вычислений может быть оценена с помощью одного из соотношений:

$$|\xi - x_n| \le \frac{q^n}{1-q} \cdot |x_1 - x_0| \qquad \text{или} \qquad |\xi - x_n| \le |x_n - x_{n-1}|,$$

где ξ – точное значение корня.

Основные операторы и функции пакета Mathematica для решения нелинейных уравнений и организации циклов.

***Plot*[** $f[x], \{x, x_{\min}, x_{\max}\}$**]** – строит график функции f переменной x на промежутке от $x_{\min}$ до $x_{\max}$.

***Plot*[** $\{f_1[x], f_2[x], \ldots\}, \{x, x_{\min}, x_{\max}\}$**]** – строит на одном рисунке графики функций f_1, $f_2, \ldots$ переменной *x* на промежутке от $x_{\min}$ до $x_{\max}$.

***ContourPlot*[** $g[x, y] == 0, \{x, x_{\min}, x_{\max}\}, \{y, y_{\min}, y_{\max}\}$**]** – строит кривую, заданную неявно уравнением $g(x, y) = 0$, в прямоугольнике $x_{\min} \le x \le x_{\max}$, $y_{\min} \le y \le y_{\max}$.

***Solve*[** $lhs == rhs, x$**]** – находит решение уравнения $lhs = rhs$ относительно переменной *x*.

***Solve*[** $\{eqn1, eqn2, \ldots\}, \{x_1, x_2, \ldots\}$**]** – находит решение системы уравнений $eqn1, eqn2, \ldots$ относительно переменных $x_1, x_2, \ldots$

***NSolve*[** eqn, x**]** – находит численное решение уравнения *eqn* относительно переменной *x*.

***NSolve*[{** $\{eqn1, eqn2, \ldots\}, \{x_1, x_2, \ldots\}$**]** – находит численное решение заданной системы уравнений.

***NSolve*[** $\{eqn1, eqn2, \ldots\}, \{x_1, x_2, \ldots\}$, Reals**]** – находит численное решение заданной системы уравнений на множестве действительных чисел.

FindRoot [$lhs == rhs$, $\{x, x_0\}$] – находит численное решение уравнения $lhs = rhs$, если для переменной *x* выбрано начальное приближение x_0.

FindRoot [$\{eqn1, eqn2\}, \{x, x_0\}, \{y, y_0\}$] – находит численное решение системы уравнений $eqn1, eqn2$ с двумя переменными *x* и *y*, если выбрано начальное приближение (x_0, y_0).

***Roots*[** *eqn ,x*] – находит корни полиномиального уравнения *eqn* относительно переменной *x*.

***Factor*[** $P[x]$**]** – раскладывает многочлен $P(x)$ на множители с целыми коэффициентами, если это возможно.

Expand*[expr*] – раскрывает скобки в выражении *expr*.

D[*f, x*] – находит производную (также и частную) функции *f* по указанной переменной *x*.

D[*f,* {*x, n*}] – находит (частную) производную *n*-го порядка функции *f* по переменной *x*.

Do*[expr,* $\{i, i_{\min}, i_{\max}\}$**]** – вычисляет *expr* с переменной *i*, последовательно принимающей значения от $i_{\min}$ до $i_{\max}$. Выражение *expr* может состоять из нескольких команд (функций), отделенных друг от друга точкой с запятой «;».

Do*[expr,* $\{i, i_{\min}, i_{\max}, h\}$**]** – вычисляет *expr* с переменной *i*, последовательно принимающей значения от $i_{\min}$ до $i_{\max}$ с шагом *h*.

For*[start, test, incr, body*] – сначала вычисляет выражение *start*, а затем раз за разом вычисляет выражения *body* (основное тело цикла) и *incr* (изменяющееся значение, от которого зависит *test*) до тех пор, пока условие *test* не перестанет давать логическое значение **True**. Каждая из частей функции ***For*** (т.е. *start, test, incr, body*) может состоять из нескольких команд (функций), отделенных друг от друга точкой с запятой «;».

While*[test, expr*] – выполняет *expr* до тех пор, пока условие *test* не перестанет давать логическое значение True.

If*[condition, tr, fls*] – возвращает *tr*, если условие *condition* является верным (True), и возвращает *fls*, если *condition* ложно (False).

If*[condition, tr, fls, u*] – возвращает *tr*, если условие *condition* верно (True); *fls*, если *condition* ложно (False); *u*, если в результате вычисления *condition* не было получено ни True, ни False.

Замечание. Каждый оператор и функция пакета **Mathematica** имеет дополнительные опции, список которых можно получить, например, с помощью команды ***Options*[***Имяфункции*].

Например, ***Options*[***NSolve*] показывает, что функция ***NSolve*** имеет только одну дополнительную опцию ***WorkingPrecision***, которая задает число верных цифр результата (по умолчанию 16).

Примеры решения нелинейных уравнений средствами пакета Mathematica

Пример 3.1. Отделить графически корни уравнения

$$36x^3 + 72x^2 - 241x - 182 = 0.$$

Δ Введем функцию $f(x) = 36x^3 + 72x^2 - 241x - 182$.

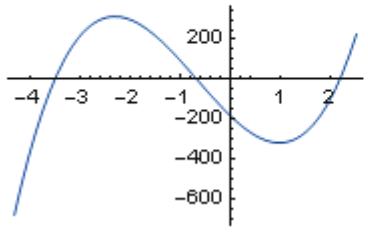
```
In[1]:= f[x_] := 36 x^3 + 72 x^2 - 241 x - 182
```

Построим ее график сначала на произвольном достаточно широком интервале, который затем уменьшим так, чтобы было понятно, между какими соседними целыми числами находятся нули функции. Возможно, потребуется построение нескольких графиков функции для отделения различных корней уравнения.

Поскольку в пакете **Mathematica** оси координат могут пересекаться не в начале координат, то чтобы не допустить ошибки при отделении корней уравнения, следует использовать опцию **AxesOrigin → {0,0}** команды **Plot**.

```
In[2]:= Plot[f[x], {x, -4.3, 2.5}, AxesOrigin → {0, 0}, ImageSize → Small]
        график функции   точка пересечения осей   размер изо···   малый
```

Out[2]=



По графику видно, что уравнение имеет три действительных корня. Они расположены в интервалах $(-4, -3)$, $(-1, 0)$, $(2, 3)$. ▲

Пример 3.2. Найти один из корней уравнения

$$36x^3 + 72x^2 - 241x - 182 = 0$$

с точностью $\varepsilon = 10^{-3}$: а) методом половинного деления; б) методом Ньютона. Указать потребовавшееся число итераций.

Δ Корни данного уравнения были отделены в примере 3.1. Доведем до заданной точности корень, расположенный на отрезке $[2, 3]$.

Введем функцию, концы отрезка и точность.

```
In[1]:= f[x_] := 36 x^3 + 72 x^2 - 241 x - 182;

In[2]:= a = 2; b = 3; e = 0.001;
```

а) В соответствии с методом половинного деления, организуем цикл с помощью оператора **Do**, в котором будем находить последовательные приближения к решению. Найдем середину отрезка $[a, b]$ – точку c. Проверим

выполнение условия $f(a)\cdot f(c)<0$, чтобы выбрать ту половину отрезка $[a,b]$, внутри которой содержится корень. Эту половину снова обозначим как $[a,b]$. Оценим точность вычислений $|b-a|<\varepsilon$. Если это неравенство верно, или если $f(c)=0$, то процесс окончен.

```
In[3]:= Do[c = (a + b) / 2.;
           оператор цикла
        fc = f[c];
        If[f[a] * fc < 0, b = c, If[fc ≠ 0, a = c]];
        условный оператор        условный оператор
          If[Abs[b - a] < e || fc == 0,
          у··· абсолютное значение
          Print["Решение x=", c // N, " получено на ", n, " шаге."];
          печатать                  численное приближение
          Break[]],
          прервать цикл
       {n, 1, 100}]

Решение x=2.16699 получено на 10 шаге.
```

б) Проверим выполнение условий сходимости метода Ньютона. Функция $f(x)$ принимает на концах отрезка $[2,3]$ значения разных знаков; ее первая и вторая производные ($f'(x)=108x^2+144x-241$ и $f''(x)=216x+144$) положительны на этом отрезке. В качестве начального приближения возьмем правый конец отрезка $b=3$, так как $f(b)\cdot f''(x)>0$.

Переменные a и b использовались и изменялись в цикле, реализующем метод половинного деления, поэтому снова введем концы отрезка $[a,b]=[2,3]$.

```
In[4]:= a = 2; b = 3;
```

Далее организуем цикл с помощью оператора **Do**, в котором будем находить последовательные приближения к решению в соответствии с методом Ньютона.

```
In[5]:= x1 = b;
        Do[x2 = x1;
        оператор цикла
         x1 = (x1 - f[x1] / f'[x1]) // N;
                                    численное приближение
         If[Abs[x2 - x1] < e,
         y··· абсолютное значение
           Print["Решение x=", x2 // N, " получено на ", n, " шаге."];
           печатать                    численное приближение
           Break[]],
           прервать цикл
         {n, 1, 100}]

        Решение x=2.16691 получено на 4 шаге.
```

Итак, чтобы получить корень уравнения с точностью $\varepsilon = 10^{-3}$ потребовалось 10 шагов для метода половинного деления и 4 шага для метода Ньютона. ▲

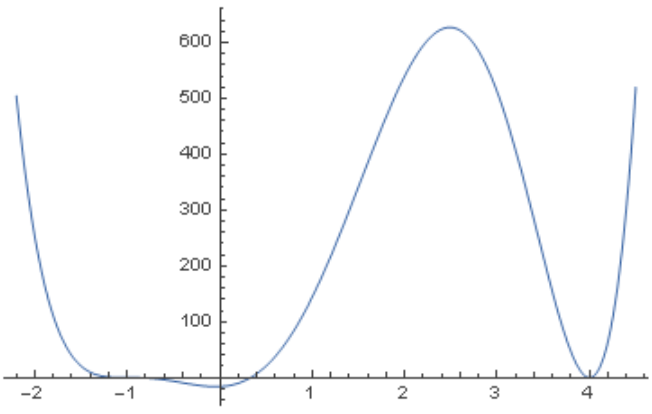
Пример 3.3. Отделить графически и найти с помощью встроенных функций пакета **Mathematica** корни уравнения

$$3x^6 - 16x^5 - 10x^4 + 80x^3 + 95x^2 + 8x - 16 = 0.$$

Δ Для отделения корней зададим левую часть уравнения как функцию $f(x)$ и построим ее график.

```
In[1]:= f[x_] := 3 x^6 - 16 x^5 - 10 x^4 + 80 x^3 + 95 x^2 + 8 x - 16

In[2]:= Plot[f[x], {x, -2.2, 4.5}]
        график функции
```

Out[2]=



Корни данного уравнения находятся на промежутках $(-1{,}5; -0{,}5)$, $(0, 1)$, $(3{,}5; 4{,}5)$. Рассмотрим несколько способов решения данного уравнения средствами пакета **Mathematica** – с помощью функций **Solve**, **NSolve**, **Roots**, а

также посредством разложения многочлена $f(x)$ на множители функцией **Factor**.

```
In[3]:= Solve[f[x] == 0]
        решить уравнения
Out[3]= {{x → -1}, {x → -1}, {x → -1}, {x → 1/3}, {x → 4}, {x → 4}}

In[4]:= NSolve[f[x] == 0]
        численное решение уравнений
Out[4]= {{x → -1.}, {x → -1.}, {x → -1.}, {x → 0.333333}, {x → 4.}, {x → 4.}}

In[5]:= Roots[f[x] == 0, x]
        корни многочлена
Out[5]= x == 1/3 || x == -1 || x == -1 || x == -1 || x == 4 || x == 4

In[6]:= Factor[f[x]]
        факторизовать
Out[6]= (-4 + x)^2 (1 + x)^3 (-1 + 3 x)
```

Как видно, данное уравнение имеет три корня, причем корни -1 и 4 являются трехкратным и двукратным корнями соответственно.

Отметим еще один способ отыскания корня уравнения при заданном начальном приближении – с помощью функции **FindRoot**.

```
In[7]:= FindRoot[f[x] == 0, {x, -2}]
        найти корень
Out[7]= {x → -1.}

In[8]:= FindRoot[f[x] == 0, {x, 0}]
        найти корень
Out[8]= {x → 0.333333}

In[9]:= FindRoot[f[x] == 0, {x, 3}]
        найти корень
Out[9]= {x → 4.}
```

▲

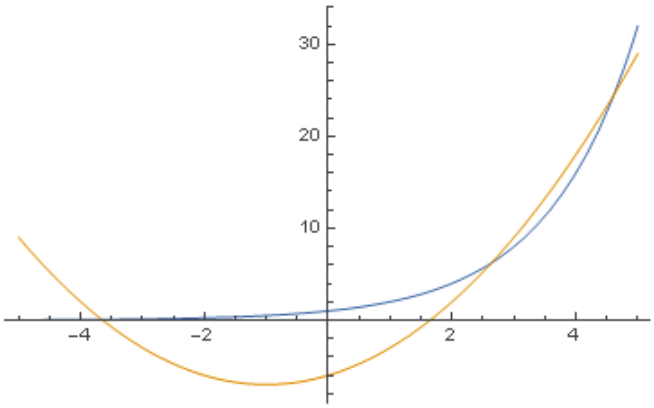
Пример 3.4. Отделить графически и найти с помощью встроенных функций пакета **Mathematica** корни уравнения

$$2^x = x^2 + 2x - 6.$$

Δ Введем две функции $f(x)=2^x$ и $g(x)=x^2+2x-6$ и построим их графики в одной системе координат. Корнями данного уравнения являются абсциссы точек пересечения этих графиков.

```
In[1]:= f[x_] := 2^x; g[x_] := x^2 + 2 x - 6;

In[2]:= Plot[{f[x], g[x]}, {x, -5, 5}]
        график функции
```

Out[2]=



Уравнение имеет три корня. Они расположены в интервалах $(-4,-3)$, $(2,3)$ и $(4,5)$.

При попытке численно решить данное уравнение с помощью встроенной функции **NSolve** программа выдает сообщение о том, что это уравнение не может быть решено методами, доступными **NSolve**.

```
In[3]:= NSolve[f[x] == g[x], x]
        численное решение уравнений

        NSolve: This system cannot be solved with the methods available to NSolve.

Out[3]= NSolve[2^x == -6 + 2 x + x^2, x]
```

Получим корни уравнения, используя функцию **FindRoot**, задавая начальное приближение к каждому из корней.

```
In[4]:= FindRoot[f[x] == g[x], {x, -4}]
        найти корень

Out[4]= {x → -3.66065}
```

```
In[5]:= FindRoot[f[x] == g[x], {x, 2}]
        найти корень

Out[5]= {x → 2.6345}
```

```
In[6]:= FindRoot[f[x] == g[x], {x, 4}]
        найти корень

Out[6]= {x → 4.61903}
```

▲

Пример 3.5. Решить средствами пакета **Mathematica** систему уравнений

$$\begin{cases} x^2 + y^2 - 2x = 5, \\ x^3 + \sin y = 1. \end{cases}$$

Δ Решение системы получим с помощью функции **FindRoot**, для которой нужно указать начальное приближение к решению.

Введем две функции $f(x,y) = x^2 + y^2 - 2x - 5$ и $g(x,y) = x^3 + \sin y - 1$.

```
In[1]:= f[x_, y_] = x^2 + y^2 - 2 x - 5
        g[x_, y_] = x^3 + Sin[y] - 1
                    синус
```

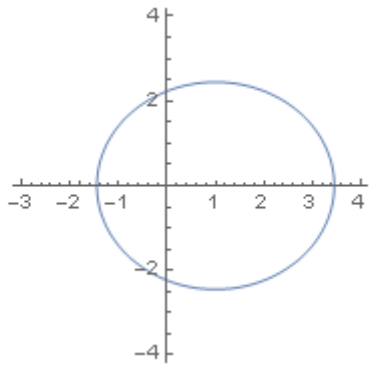
```
Out[1]= -5 - 2 x + x^2 + y^2

Out[2]= -1 + x^3 + Sin[y]
```

Решением данной системы являются точки пересечения кривых $f(x,y) = 0$ и $g(x,y) = 0$. Построим их с помощью функции **ContourPlot**, а затем объединим в одном графическом окне функцией **Show**.

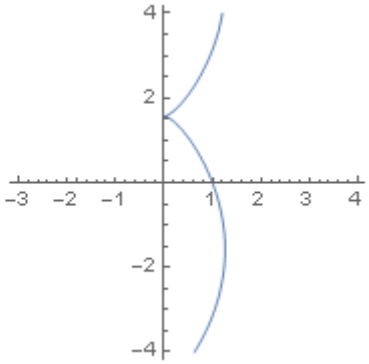
```
In[3]:= gr1 = ContourPlot[f[x, y] == 0, {x, -3, 4}, {y, -4, 4}, Axes → True,
            контурный график                                    оси    истина
        Frame → False, ImageSize → Small]
        рамка   ложь   размер изо···   малый
```

```
In[4]:= gr2 = ContourPlot[g[x, y] == 0, {x, -3, 4}, {y, -4, 4}, Axes → True,
            контурный график                                    оси    истина
        Frame → False, ImageSize → Small]
        рамка   ложь   размер изо···   малый
```

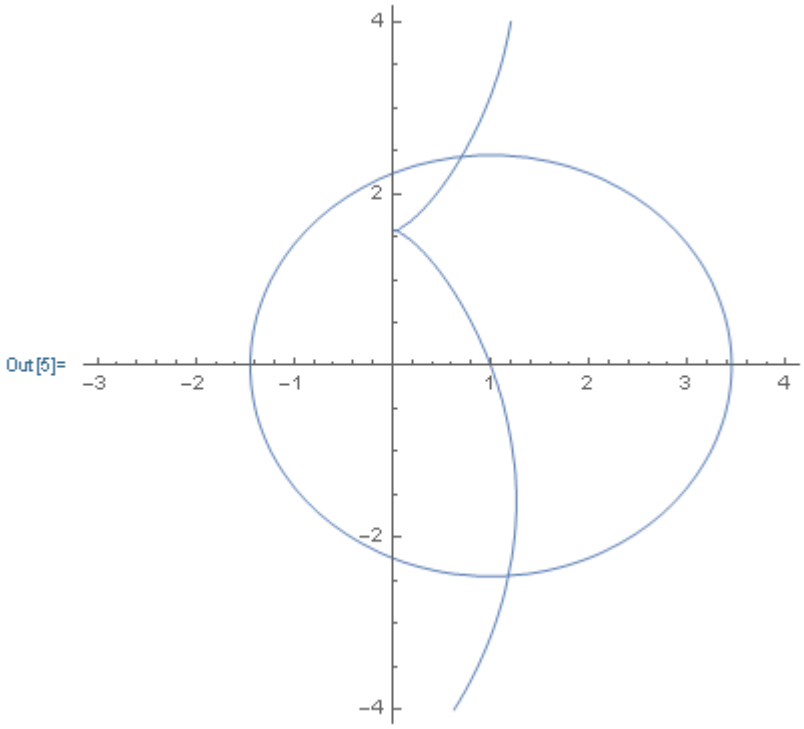
Out[3]=



Out[4]=




```
In[5]:= Show[gr1, gr2, ImageSize → Medium]
        показать   размер изо···   средний
```



По графику видно, что кривые пересекаются в двух точках. Одна из них находится вблизи точки $(1, 2)$, вторая – возле точки $(1, -2)$. Эти точки и будем использовать как начальные приближения.

```
In[6]:= FindRoot[{f[x, y] == 0, g[x, y] == 0}, {x, 1}, {y, 2}]
        найти корень
Out[6]= {x → 0.703435, y → 2.43147}

In[7]:= FindRoot[{f[x, y] == 0, g[x, y] == 0}, {x, 1}, {y, -2}]
        найти корень
Out[7]= {x → 1.18005, y → -2.44286}
```

▲