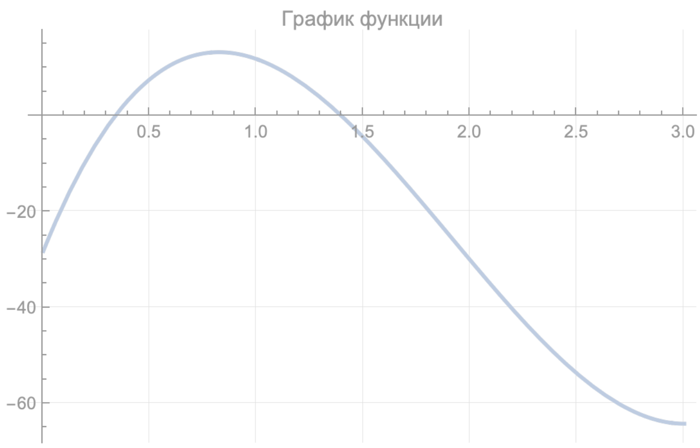
```
(*############################################
 ####################################################*)
(*1*)
f[x_] := 15 x^3 - 86 x^2 + 111 x - 28

Plot[f[x], {x, 0, 3}, GridLines → Automatic,
 AxesOrigin → {0, 0}, PlotLabel → "График функции"]

(*Метод хорд*)
SecantMethod[f_, x0_, x1_, eps_] :=
 Module[{x2, xold = x1, xnew = x0, count = 0}, While[Abs[f[xnew]] > eps,
    x2 = xnew - (f[xnew] * (xnew - xold)) / (f[xnew] - f[xold]);
    xold = xnew;
    xnew = x2;
    count++;];
  {xnew, count}]

root = SecantMethod[f, 1, 2, 10^-3]

(*Выведем найденное приближение корня и количество итераций*)
{N[root⟦1⟧], root⟦2⟧}
```

график функции | линии коорд··· | автоматический | точка пересечения осей | пометка графика | программный модуль | цикл··· | абсолютное значение | численное приближение

Out[7]=



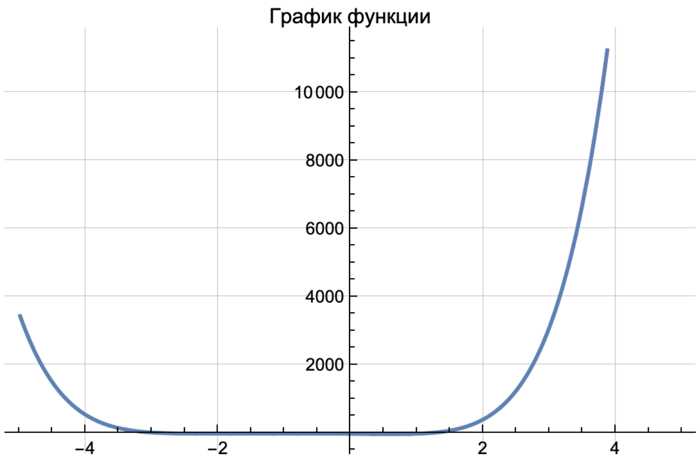
Out[10]=

```
{1.4, 5}

(*############################################
 ####################################################*)
(*2*)
```

In[29]:=

```
f[x_] := x^6 + 6 x^5 + 12 x^4 + 6 x^3 - 9 x^2 - 12 x - 4

Plot[f[x], {x, -5, 5}, GridLines → Automatic, PlotLabel → "График функции"]
график функции             линии коорд···  автоматиче···  пометка графика

(*Метод Solve-аналитическое решение*)
        решить уравнения
analyticalRoots = Solve[f[x] == 0, x]
                  решить уравнения

(*Метод NSolve-численное решение*)
        численное решение уравнений
numericalRoots = NSolve[f[x] == 0, x]
                 численное решение уравнений

(*Метод Roots-нахождение корней*)
        корни многочлена
rootsEquationForm = Roots[f[x] == 0, x]
                    корни многочлена

(*Метод FindRoot-нахождение корня при начальном приближении*)
        найти корень
findRootExample = FindRoot[f[x] == 0, {x, 0}]
                  найти корень

(*Шаг 3:Разложение многочлена на множители*)
factorization = Factor[f[x]]
                факторизовать
{analyticalRoots, numericalRoots,
 rootsEquationForm, findRootExample, factorization}
```

Out[30]=



Out[31]=

```
{{x → -2}, {x → -2}, {x → -1}, {x → -1}, {x → -1}, {x → 1}}
```

Out[32]=

```
{{x → -2.}, {x → -2.}, {x → -1.}, {x → -1.}, {x → -1.}, {x → 1.}}
```

Out[33]=

```
x == -2 || x == -2 || x == -1 || x == -1 || x == -1 || x == 1
```

Out[34]=

```
{x → -0.999993}
```

Out[35]=

$$(-1+x)\ (1+x)^{3}\ (2+x)^{2}$$

Out[36]=

```
{{{x → -2}, {x → -2}, {x → -1}, {x → -1}, {x → -1}, {x → 1}},
 {{x → -2.}, {x → -2.}, {x → -1.}, {x → -1.}, {x → -1.}, {x → 1.}},
 x == -2 || x == -2 || x == -1 || x == -1 || x == -1 || x == 1,
 {x → -0.999993}, (-1 + x) (1 + x)^3 (2 + x)^2}
```

FindRoot: The line search decreased the step size to within tolerance specified by AccuracyGoal and PrecisionGoal but was unable to find a sufficient decrease in the merit function. You may need more than MachinePrecision digits of working precision to meet these tolerances.

```
(*##########################################
 #################################################*)
(*3*)
```

In[66]:=
```
f[x_] := 3^x - 4 x - 2
fPrime[x_] := D[f[x], x]
              дифференцировать

(*Метод Ньютона*)
newtonIteration[x0_, tol_, maxIter_] := Module[{x = x0, xNext, i = 0},
                                        программный модуль
  While[i < maxIter && Abs[f[x]] > tol, xNext = x - f[x] / fPrime[x];
  цикл-пока                 абсолютное значение
   If[Abs[xNext - x] < tol, Break[]];
   у··· абсолютное значение   прервать цикл
   x = xNext;
   i++];
  {N[x], i}] (*Применяем N[] для численного результата*)
   численное приближение  численное приближение

(*Метод секущих*)
secantIteration[x0_, x1_, tol_, maxIter_] :=
 Module[{xPrev = x0, x = x1, xNext, i = 0}, While[
 программный модуль                          цикл-пока
   i < maxIter && Abs[f[x]] > tol, xNext = x - f[x] (x - xPrev) / (f[x] - f[xPrev]);
                  абсолютное значение
   If[Abs[xNext - x] < tol, Break[]];
   у··· абсолютное значение   прервать цикл
   xPrev = x;
   x = xNext;
   i++];
  {N[x], i}] (*Применяем N[] для численного результата*)
   численное приближение  численное приближение

(*Вычислим корень методом Ньютона с начальным приближением x0=0.5*)
rootNewton = newtonIteration[0.5, 10^-3, 100]

(*Вычислим корень методом секущих с начальными приближениями x0=0 и x1=1*)
rootSecant = secantIteration[0, 1, 10^-3, 100]

{rootNewton, rootSecant}
```

D: 0.5` is not a valid variable.

Out[70]=

$$\left\{0.5 + \frac{2.26795}{\partial_{0.5}(-2.26795)}, 1\right\}$$

Out[71]=

{-0.325094, 4}

Out[72]=

$$\left\{\left\{0.5 + \frac{2.26795}{\partial_{0.5}(-2.26795)}, 1\right\}, \{-0.325094, 4\}\right\}$$

```
(*######################################
 #############################################*)
(*4*)
```

••• D: 0.5` is not a valid variable.

```
In[73]:= phi[x_] := Log[4 x + 2] / Log[3]
                    натуральный ⋯  натуральный логарифм

(*Метод простых итераций*)
simpleIteration[x0_, tol_, maxIter_] := Module[{x = x0, xNext, i = 0},
                                        программный модуль
  While[i < maxIter && Abs[phi[x] - x] > tol, xNext = phi[x];
  цикл-пока                абсолютное значение
   If[Abs[xNext - x] < tol, Break[]];
   у⋯ абсолютное значение         прервать цикл
   x = xNext;
   i++];
  {N[x], i}]
   численное приближение

(*Применяем метод простых итераций с начальным приближением x0=0.5*)
rootSimpleIter = simpleIteration[0.5, 10^-3, 100]

Out[75]=
{2.1477, 8}

(*######################################
 #############################################*)
(*5*)
```

In[76]:=
```
eqn = 3^x == 4 x + 2;

(*Решаем уравнение с помощью Solve*)
                               решить уравнения
solveRoot = Solve[eqn, x]
            решить уравнения

(*Решаем уравнение с помощью NSolve для численного решения*)
                               численное решение уравнений
nsolveRoot = NSolve[eqn, x]
             численное решение уравнений

(*Решаем уравнение с помощью FindRoot с начальным приближением*)
                               найти корень
findRoot = FindRoot[eqn, {x, 0.5}]
           найти корень
```

Solve: Inverse functions are being used by Solve, so some solutions may not be found; use Reduce for complete solution information.

Out[77]=

$$\left\{\left\{x \to \frac{-\text{Log}[3] - 2\,\text{ProductLog}\left[-\frac{\text{Log}[3]}{4\sqrt{3}}\right]}{2\,\text{Log}[3]}\right\}, \left\{x \to \frac{-\text{Log}[3] - 2\,\text{ProductLog}\left[-1, -\frac{\text{Log}[3]}{4\sqrt{3}}\right]}{2\,\text{Log}[3]}\right\}\right\}$$

Out[78]=

```
{{x → -0.325081}, {x → 3.1091 - 6.6992 i}, {x → 3.1091 + 6.6992 i},
 {x → 3.61294 + 12.5806 i}, {x → 3.9372 + 18.3717 i}, {x → 4.17643 + 24.1324 i},
 {x → 4.36599 + 29.8789 i}, {x → 4.52296 + 35.6175 i},
 {x → 4.65689 + 41.3513 i}, {x → 4.77367 + 47.0819 i},
 {x → 4.8772 + 52.8103 i}, {x → 4.97018 + 58.537 i}, {x → 6.14241 - 213.012 i},
 {x → 2.14839}, {x → 5.05455 + 64.2625 i}, {x → 5.13178 + 69.9871 i}}
```

Out[79]=

```
{x → -0.325081}

(*#############################################
 ###################################################*)
(*6*)
```

In[85]:=

```
eq1 = CubeRoot[x^2] + CubeRoot[y^2] == CubeRoot[25];
      кубический корень   кубический корень   кубический корень
eq2 = x^3 + y^3 - 9*x*y == 0;

(*Решаем систему уравнений*)
solutions = Solve[{eq1, eq2}, {x, y}];
            решить уравнения

(*Выводим найденные решения*)
solutions

(*Графическое отображение кривых*)
ContourPlot[{CubeRoot[x^2] + CubeRoot[y^2] == CubeRoot[25],
контурный гра…  кубический корень  кубический корень  кубический корень
  x^3 + y^3 - 9*x*y == 0}, {x, -5, 5}, {y, -5, 5},
 PlotLegends → {"f(x,y) = 0", "g(x,y) = 0"}, ContourStyle → {Red, Blue}]
 легенды графика                             контурный стиль   кра…  синий
```

Out[88]=

```
{{x → 0.875…, y → 18 -2.85…}, {x → -2.85…, y → 18 0.875…},
 {x → 2.81…, y → 18 0.904…}, {x → 0.904…, y → 18 2.81…}}
```

Out[89]=