

Интегрированные математические пакеты

Юдинцев В. В.

Кафедра теоретической механики. Самарский университет

Решение линейных уравнений

```
In[*]:= ClearAll["Global`*"]
```

Уравнения в символьном виде

```
In[*]:= eq = {x + 2 y + z == 1, 2 x - y - z == 2, z + y == 2};  
Solve[eq, {x, y, z}]
```

```
Out[*]:= {{x -> 2, y -> -3, z -> 5}}
```

Уравнения в матричной форме (задана матрица коэффициентов)

Решение уравнения вида

$$\mathbf{M} \mathbf{x} = \mathbf{B}$$

```
In[*]:= ca = CoefficientArrays[eq, {x, y, z}] // Normal
```

```
Out[*]:= {{-1, -2, -2}, {{1, 2, 1}, {2, -1, -1}, {0, 1, 1}}}
```

```
In[*]:= M = ca[[2]]; B = ca[[1]];  
LinearSolve[M, B]
```

```
Out[*]:= {-2, 3, -5}
```

Solve

Функция Solve ищет аналитическое решение уравнения

Все решения тригонометрического уравнения

```
In[*]:= Solve[Cos[x] + Sin[x]^2 == 0, x]
```

```
Out[*]= {{x -> ConditionalExpression[-π - ArcTan[√(2(-1 + √5)) / (1 - √5)] + 2π c1, c1 ∈ Z]},
{x -> ConditionalExpression[π + ArcTan[√(2(-1 + √5)) / (1 - √5)] + 2π c1, c1 ∈ Z]},
{x -> ConditionalExpression[-i ArcTanh[√(2 / (1 + √5))] + 2π c1, c1 ∈ Z]},
{x -> ConditionalExpression[i ArcTanh[√(2 / (1 + √5))] + 2π c1, c1 ∈ Z]}}
```

Только вещественные решения

```
In[*]:= sol = Solve[Cos[x] + Sin[x]^2 == 0, x, Reals]
```

```
Out[*]= {{x -> ConditionalExpression[-2 ArcTan[√(2 + √5)] + 2π c1, c1 ∈ Z]},
{x -> ConditionalExpression[2 ArcTan[√(2 + √5)] + 2π c1, c1 ∈ Z]}}
```

ConditionalExpression означает, что решение представлено для c_1 принадлежащих целым числам $\dots -2, -1, 0, 1, 2, \dots$

Решение для $c_1 = 0$

```
In[*]:= sol /. c1 -> 0
```

```
Out[*]= {{x -> -2 ArcTan[√(2 + √5)]}, {x -> 2 ArcTan[√(2 + √5)]}}
```

NSolve

Функция **NSolve** ищет аналитическое решение уравнения и представляет результат в приближенном виде

```
In[*]:= NSolve[Cos[x] + Sin[x]^2 == 0, x]
```

```
Out[*]:= {{x -> ConditionalExpression[1. (-2.237036 + 6.283185 c1), c1 ∈ ℤ]},
{x -> ConditionalExpression[1. (2.237036 + 6.283185 c1), c1 ∈ ℤ]},
{x -> ConditionalExpression[1. ((0. - 1.061275 i) + 6.283185 c1), c1 ∈ ℤ]},
{x -> ConditionalExpression[1. ((0. + 1.061275 i) + 6.283185 c1), c1 ∈ ℤ]}}
```

Только вещественные решения

```
In[*]:= sol = NSolve[Cos[x] + Sin[x]^2 == 0, x, Reals]
```

```
Out[*]:= {{x -> ConditionalExpression[1. (-2.237036 + 6.283185 c1), c1 ∈ ℤ]},
{x -> ConditionalExpression[1. (2.237036 + 6.283185 c1), c1 ∈ ℤ]}}
```

Решение для $c_1=0$

```
In[*]:= sol /. c1 -> 0
```

```
Out[*]:= {{x -> -2.237036}, {x -> 2.237036}}
```

Дополнительные условия

```
In[*]:= NSolve[{Cos[x] + Sin[x]^2 == 0, x > -π, x < π}, x]
```

```
Out[*]:= {{x -> -2.237036}, {x -> 2.237036}}
```

NSolve

Функция NSolve пытается найти все решения уравнения.

```
In[*]:= NSolve[Cos[x] + Sin[x]^2 == 0, x]
```

```
Out[*]:= {{x -> ConditionalExpression[1. (-2.237036 + 6.283185 c1), c1 ∈ Z]},
{x -> ConditionalExpression[1. (2.237036 + 6.283185 c1), c1 ∈ Z]},
{x -> ConditionalExpression[1. ((0. - 1.061275 i) + 6.283185 c1), c1 ∈ Z]},
{x -> ConditionalExpression[1. ((0. + 1.061275 i) + 6.283185 c1), c1 ∈ Z]}}
```

```
In[*]:= Solve[Cos[x] + Sin[x]^2 == 0, x]
```

```
Out[*]:= {{x -> ConditionalExpression[-π - ArcTan[ $\frac{\sqrt{2}(-1 + \sqrt{5})}{1 - \sqrt{5}}$ ] + 2 π c1, c1 ∈ Z]},
{x -> ConditionalExpression[π + ArcTan[ $\frac{\sqrt{2}(-1 + \sqrt{5})}{1 - \sqrt{5}}$ ] + 2 π c1, c1 ∈ Z]},
{x -> ConditionalExpression[-i ArcTanh[ $\sqrt{\frac{2}{1 + \sqrt{5}}}$ ] + 2 π c1, c1 ∈ Z]},
{x -> ConditionalExpression[i ArcTanh[ $\sqrt{\frac{2}{1 + \sqrt{5}}}$ ] + 2 π c1, c1 ∈ Z]}}
```

NSolve

Дополнительные условия

```
In[ ]:= NSolve[{Cos[x] + Sin[x]^2 == 0, x > -π, x < π}, x]
```

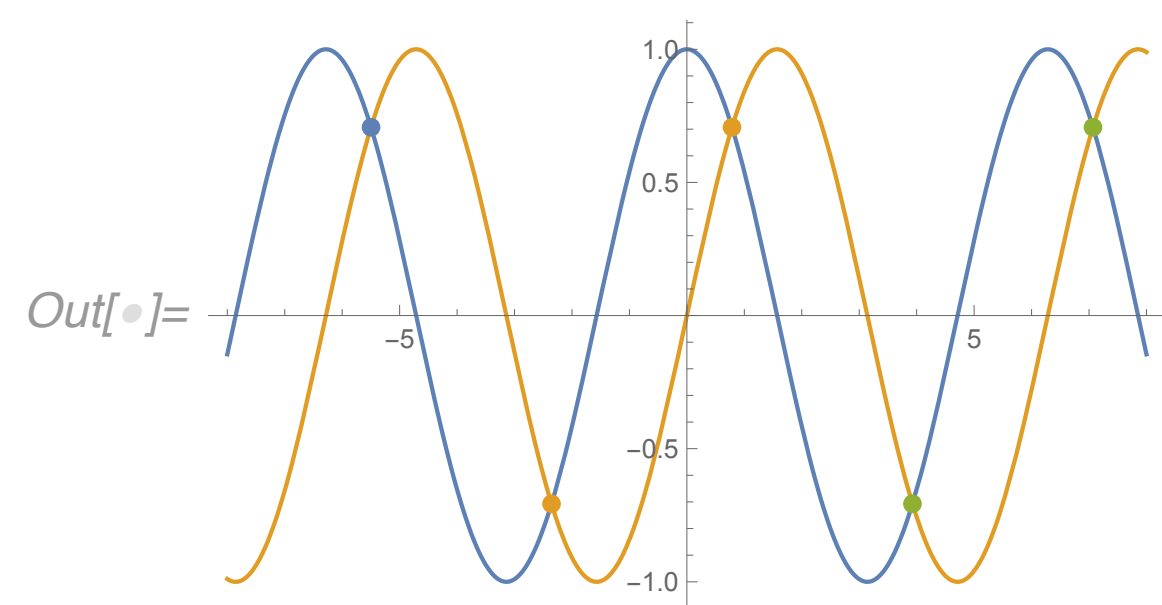
```
Out[ ]:= {{x → -2.237036}, {x → 2.237036}}
```

Пример

```
In[ ]:= sx = NSolve[Cos[x] == Sin[x], x] /. c1 → # & /@ Range[-1, 1]
```

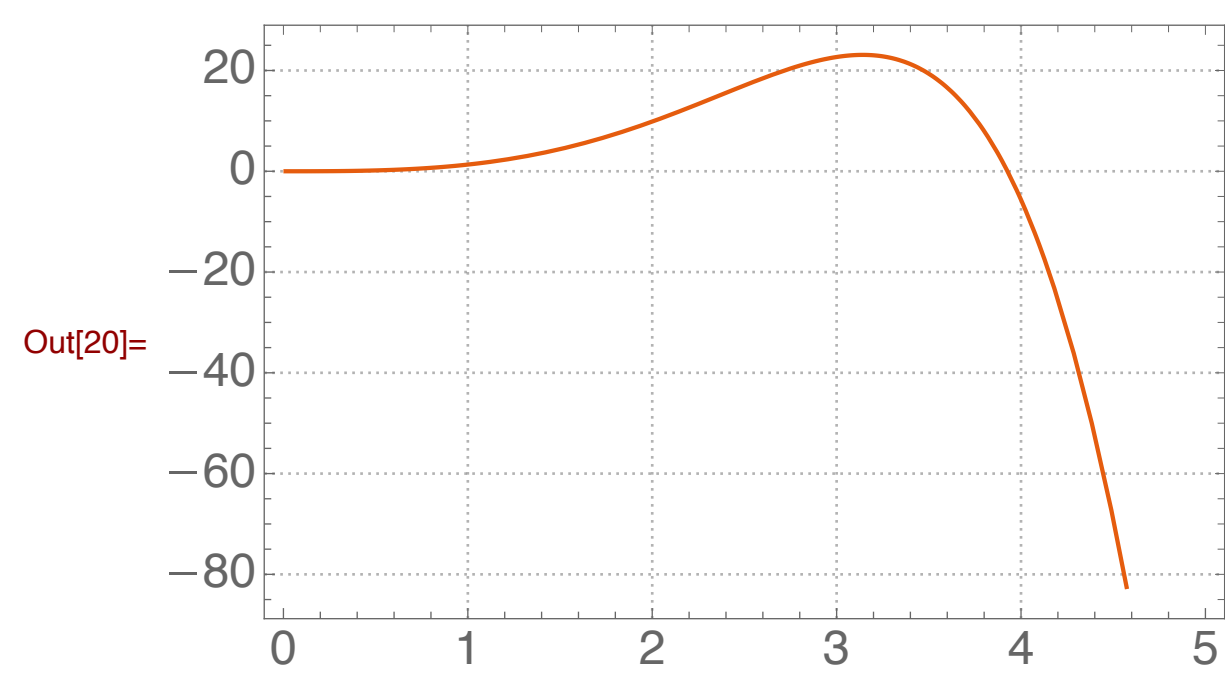
```
Out[ ]:= {{{x → -8.63938}, {x → -5.497787}}, {{x → -2.356194}, {x → 0.7853982}}, {{x → 3.926991}, {x → 7.068583}}}
```

```
In[ ]:= Show[
  Plot[{Cos[x], Sin[x]}, {x, -8, 8}],
  ListPlot[{x, Cos[x]} /. sx, PlotStyle → PointSize[Large]]
]
```



Решение нелинейных уравнений

```
In[19]:= f = 2 Cosh[s] Sin[s] - 2 Cos[s] Sinh[s];
Plot[f, {s, 0, 5}, PlotTheme -> {"Scientific", "FrameGrid"}, ImageSize -> 400, BaseStyle -> 18]
```



Не все уравнения функция Solve (NSolve) может решить

```
In[*]:= NSolve[f, s]
```

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

```
Out[*]:= NSolve[2 Cosh[s] Sin[s] - 2 Cos[s] Sinh[s], s]
```

Численное решение в окрестности начального приближения

```
In[*]:= FindRoot[f, {s, 2.0}]
```

```
Out[*]:= {s -> 2.209943 × 10-8}
```

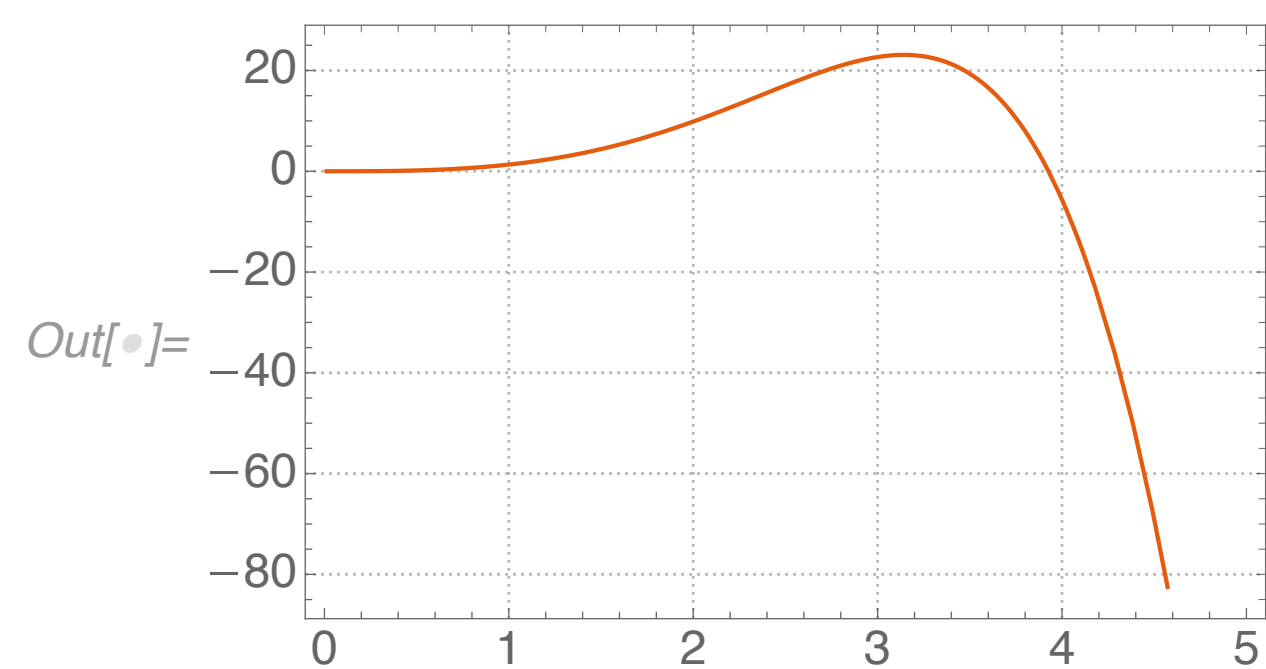
```
In[*]:= FindRoot[f, {s, 3.2}]
```

```
Out[*]:= {s -> 3.926602}
```

Решение нелинейных уравнений

```
In[*]:= f = 2 Cosh[s] Sin[s] - 2 Cos[s] Sinh[s];
```

```
Plot[f, {s, 0, 5}, PlotTheme -> {"Scientific", "FrameGrid"}, ImageSize -> 400, BaseStyle -> 18]
```



Начальное приближение и диапазон

```
In[*]:= FindRoot[f, {s, 2.5, 2, 3}]
```

... **FindRoot**: The point {2.} is at the edge of the search region {2., 3.} in coordinate 1 and the computed search direction points outside the region.

```
Out[*]:= {s -> 2.}
```

В диапазоне от 2 до 3 корней нет

```
In[*]:= FindRoot[f, {s, 3.5, 3, 5}]
```

```
Out[*]:= {s -> 3.926602}
```

В диапазоне от 3 до 5 есть корень при $s = 3.92\dots$

FindRoot

Решение не всегда ближайшее к начальному приближению

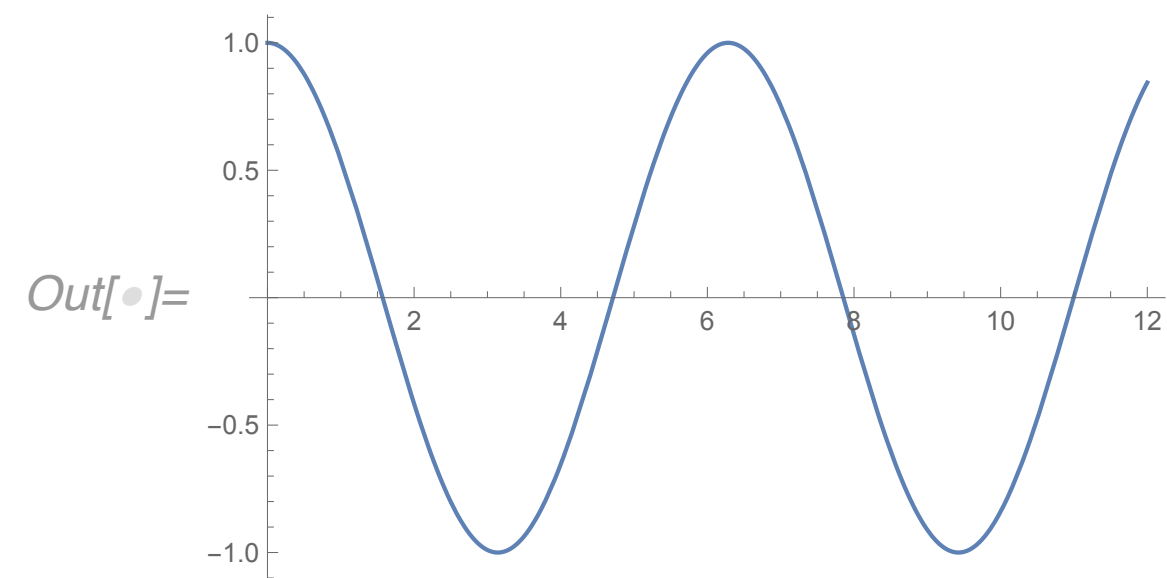
```
In[*]:= FindRoot[Cos[x] == 0, {x, 0.0001}]
```

```
Out[*]:= {x -> 10.99557}
```

```
In[*]:= FindRoot[Cos[x] == 0, {x, 1.0}]
```

```
Out[*]:= {x -> 1.570796}
```

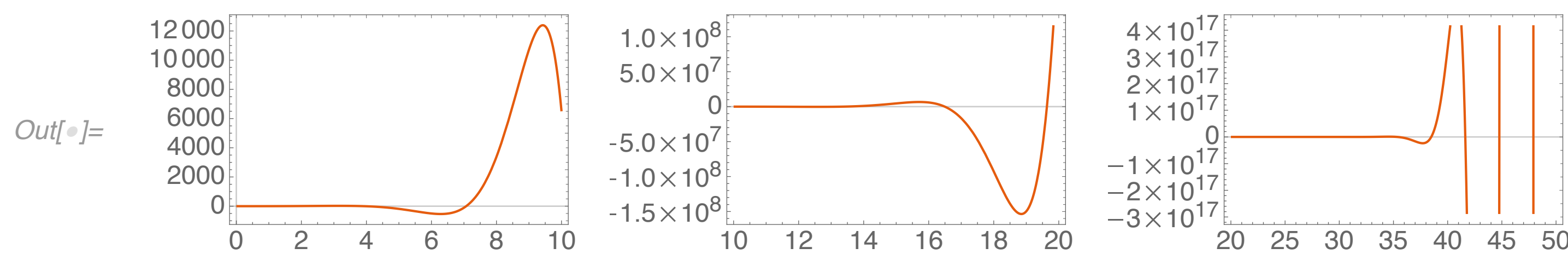
```
In[*]:= Plot[Cos[x], {x, 0, 12}]
```



Пример: поиск множества корней

Функция с бесконечным количеством корней

```
In[*]:= f = 2 Cosh[s] Sin[s] - 2 Cos[s] Sinh[s];
GraphicsRow[Plot[f, #, PlotTheme -> "Scientific", BaseStyle -> 18] & /@ {{s, 0, 10}, {s, 10, 20}, {s, 20, 50}},
ImageSize -> 1000]
```



Если список решений рассмотреть как множество, то можно применить функцию `Union`, которая попытается исключить одинаковые элементы, однако поскольку корни вещественные числа, все дубли исключить не получится.

```
In[*]:= Union[{1, 2, 3, 3, 5, 5, 9}]
```

```
Out[*]:= {1, 2, 3, 5, 9}
```

```
In[*]:= s /. FindRoot[f, {s, #}] & /@ Range[0, 20, 0.2] // Union
```

```
Out[*]:= {-1.795555 × 10-8, -5.390809 × 10-9, 0., 1.549848 × 10-9, 4.367251 × 10-9, 5.701977 × 10-9, 9.664084 × 10-9, 1.230899 × 10-8,
1.387491 × 10-8, 1.422146 × 10-8, 1.791917 × 10-8, 2.153272 × 10-8, 2.16778 × 10-8, 2.168496 × 10-8, 2.196831 × 10-8,
2.209943 × 10-8, 2.395051 × 10-8, 3.926602, 3.926602, 3.926602, 7.068583, 7.068583, 7.068583, 7.068583, 10.21018, 10.21018,
10.21018, 10.21018, 13.35177, 13.35177, 13.35177, 16.49336, 19.63495}
```

Если допустить, что корни отличаются друг от друга не менее чем на 0,001, можно в начале округлить массив решений до 1 тысячной, а затем к округленным значениям применить функцию `Union`

```
In[*]:= Round[s, 0.001] /. FindRoot[f, {s, #}] & /@ Range[0, 20, 0.2] // Union
```

```
Out[*]:= {0., 3.927, 7.069, 10.21, 13.352, 16.493, 19.635}
```

```
In[*]:= roots = Round[s, 0.001] /. FindRoot[f, {s, #}] & /@ Range[0, 20, 0.2] // Union
```

```
Out[*]:= {0., 3.927, 7.069, 10.21, 13.352, 16.493, 19.635}
```

Таким образом в диапазоне от 0 до 20 у рассматриваемой функции 7 корней.

Линейное программирование

Определить максимальный план производства при ограничениях:

Вид сырья	Продукт 1	Продукт 2	Продукт 3	Продукт 4	Запасы сырья
Сырьё 1	4 кг	2 кг	1 кг	8 кг	≤ 1200 кг
Сырьё 2	2 кг	10 кг	6 кг	0 кг	≤ 600 кг
Сырьё 3	3 кг	0 кг	6 кг	1 кг	≤ 1500 кг
Прибыль	15 р	6 р	12 р	24 р	максимум

$$S1*x1 + S2*x2 + S3*x3 + S4*x4 \rightarrow \max$$

$$r11*x1+r12*x2+r13*x3+r14*x4 \leq 1200$$

$$r21*x1+r22*x2+r23*x3+r24*x4 \leq 600$$

$$r31*x1+r32*x2+r33*x3+r34*x4 \leq 1500$$

$$x1>0, x2>0, x3>0, x4>0$$

Линейное программирование

Определить максимальный план производства при ограничениях:

Вид сырья	Продукт 1	Продукт 2	Продукт 3	Продукт 4	Запасы сырья
Сырьё 1	4 кг	2 кг	1 кг	8 кг	≤ 1200 кг
Сырьё 2	2 кг	10 кг	6 кг	0 кг	≤ 600 кг
Сырьё 3	3 кг	0 кг	6 кг	1 кг	≤ 1500 кг
Прибыль	15 р	6 р	12 р	24 р	максимум

```
In[*]:= Maximize[{15 x1 + 6 x2 + 12 x3 + 24 x4, 4 x1 + 2 x2 + 1 x3 + 8 x4 ≤ 1200 && 2 x1 + 10 x2 + 6 x3 + 0 x4 ≤ 600 && 3 x1 + 0 x2 + 6 x3 + 1 x4 ≤ 1500 && x1 > 0 && x2 > 0 && x3 > 0 && x4 > 0}, {x1, x2, x3, x4}]
```

... **Maximize**: Warning: there is no maximum in the region in which the objective function is defined and the constraints are satisfied; a result on the boundary will be returned.

```
Out[*]:= {4500, {x1 → 0, x2 → 0, x3 → 100, x4 →  $\frac{275}{2}$ }}
```

```
In[*]:= NMaximize[{15 x1 + 6 x2 + 12 x3 + 24 x4, 4 x1 + 2 x2 + 1 x3 + 8 x4 ≤ 1200 && 2 x1 + 10 x2 + 6 x3 + 0 x4 ≤ 600 && 3 x1 + 0 x2 + 6 x3 + 1 x4 ≤ 1500 && x1 > 0 && x2 > 0 && x3 > 0 && x4 > 0}, {x1, x2, x3, x4}]
```

```
Out[*]:= {4500., {x1 → 0., x2 → 0., x3 → 100., x4 → 137.5}}
```

Функция *LinearProgramming*

Функция ищет минимум целевой функции при $\mathbf{Mx} > \mathbf{b}$

```
In[*]:= LinearProgramming[-{15, 6, 12, 24}, -{{4, 2, 1, 8}, {2, 10, 6, 0}, {3, 0, 6, 1}}, -{1200, 600, 1500}] // N
```

```
Out[*]:= {0., 0., 100., 137.5}
```

```
In[*]:=
```

Максимум функции

FindMaximum

Функция **FindMaximum** ищет локальный максимум функции

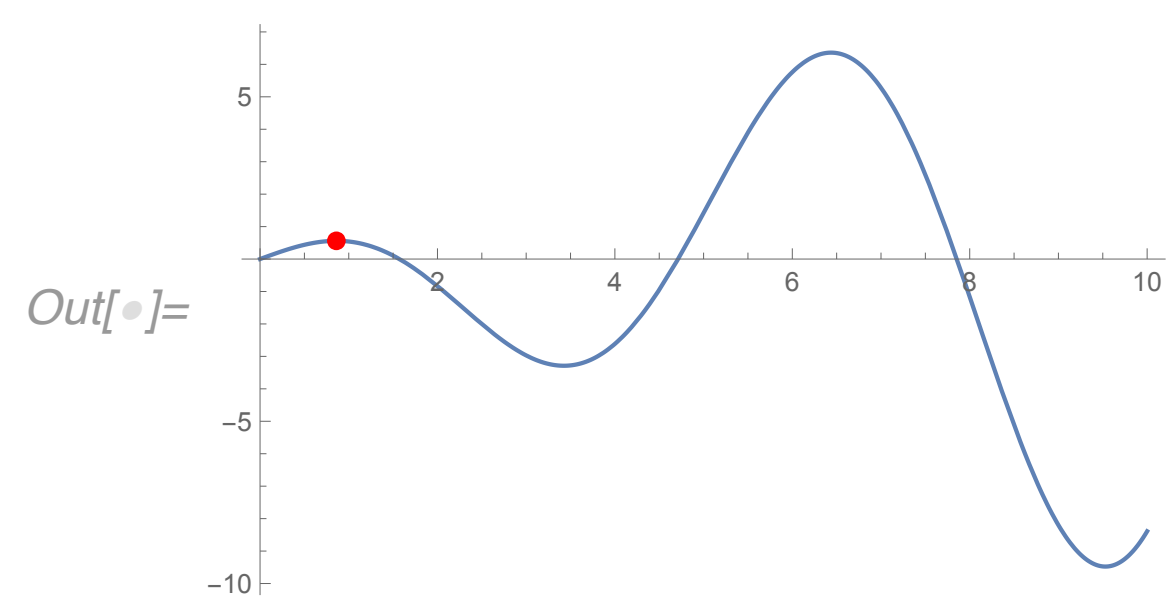
```
In[*]:= f = x Cos[x];
```

Автоматический выбор начального приближения

```
In[*]:= max1 = FindMaximum[f, x]
```

```
Out[*]:= {0.5610963, {x -> 0.8603336}}
```

```
In[*]:= Plot[f, {x, 0, 10}, Epilog -> {PointSize[Large], Red, Point[{x, f} /. max1[[2]]}]
```

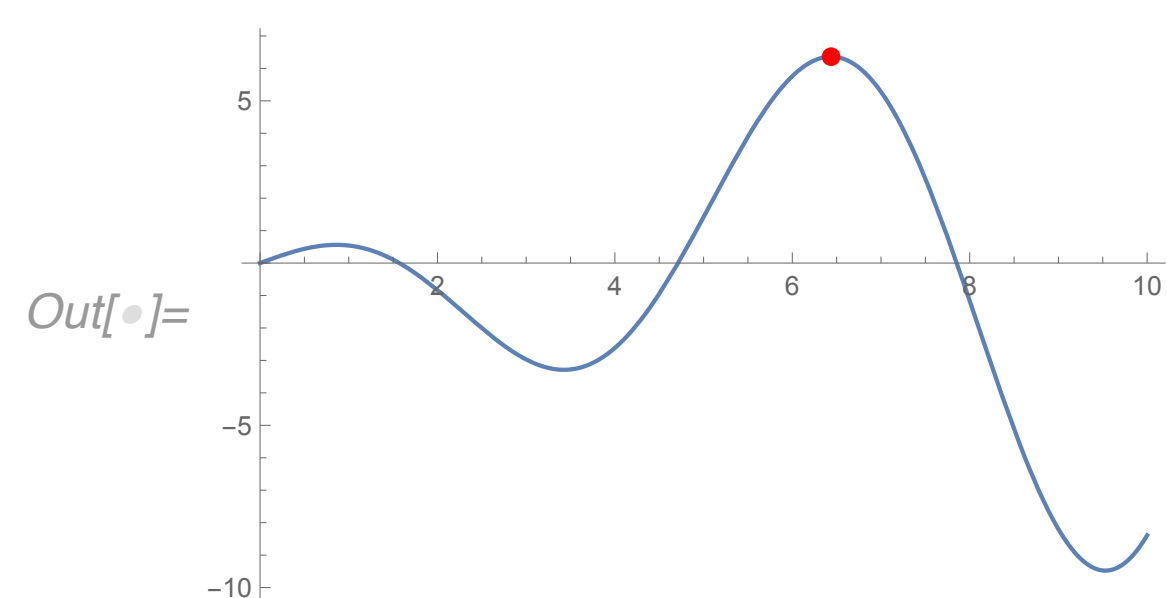


Поиск в другом месте

```
In[*]:= max2 = FindMaximum[f, {x, 6}]
```

```
Out[*]:= {6.361004, {x -> 6.437298}}
```

```
In[*]:= Plot[f, {x, 0, 10}, Epilog -> {PointSize[Large], Red, Point[{x, f} /. max2[[2]]}]
```



Дополнительно могут быть заданы ограничения, в этом случае максимум может быть достигнут на границе области

```
In[*]:= max2 = FindMaximum[{f, 0 < x < 6}, {x, 5}]
```

```
Out[*]:= {5.761022, {x -> 6.}}
```

Максимум функции

Maximize

Функция **Maximize** ищет глобальный максимум

Если функция неограниченная, то максимум не будет найден

```
In[*]:= max1 = Maximize[{f}, x]
```

... **Maximize**: The maximum is not attained at any point satisfying the given constraints.

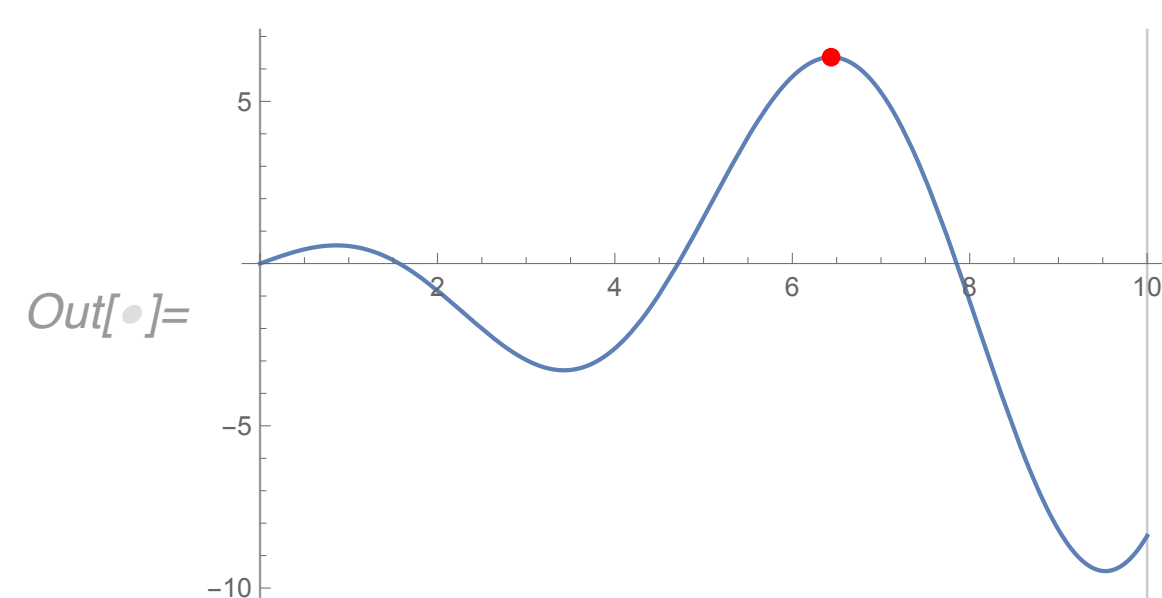
```
Out[*]:= {∞, {x → -∞}}
```

Необходимы дополнительные ограничения

```
In[*]:= max1 = Maximize[{f, 0 < x < 10}, x] // N
```

```
Out[*]:= {6.361004, {x → 6.437298}}
```

```
In[*]:= Plot[f, {x, 0, 10}, Epilog → {PointSize[Large], Red, Point[{x, f] /. max1[[2]]}], GridLines → {{0, 10}, None}]
```



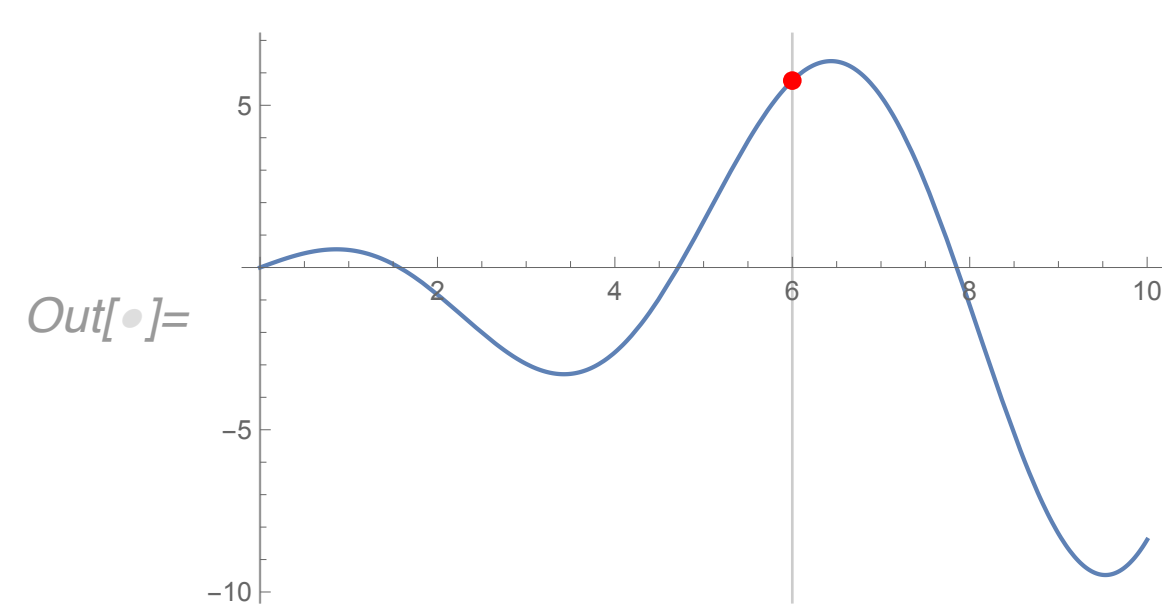
Результат может лежать на границе

```
In[*]:= max1 = Maximize[{f, 0 < x < 6}, x] // N
```

... **Maximize**: Warning: there is no maximum in the region in which the objective function is defined and the constraints are satisfied; a result on the boundary will be returned.

```
Out[*]:= {5.761022, {x → 6.}}
```

```
In[*]:= Plot[f, {x, 0, 10}, Epilog → {PointSize[Large], Red, Point[{x, f] /. max1[[2]]}], GridLines → {{0, 6}, None}]
```



Интерполяция

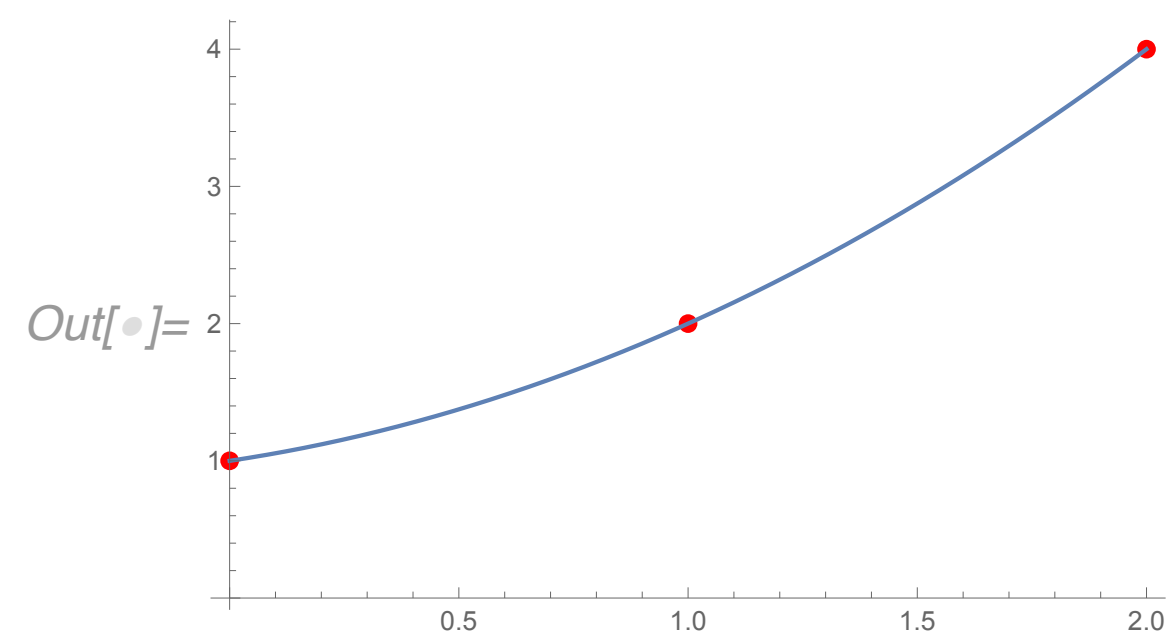
Линейная интерполяция

```
In[*]:= f = .;  
data = {{0, 1}, {1, 2}, {2, 4}};  
f[x_] = Interpolation[data, x, InterpolationOrder -> 2]  
f[0.5]
```

```
Out[*]= InterpolatingFunction[ Domain: {{0, 2}}  
Output: scalar][x]
```

```
Out[*]= 1.375
```

```
In[*]:= Show[  
ListPlot[data, PlotStyle -> {Red, PointSize[Large]}],  
Plot[f[x], {x, 0, Max[Transpose[data][[1]]]}]  
]
```



Интерполяция

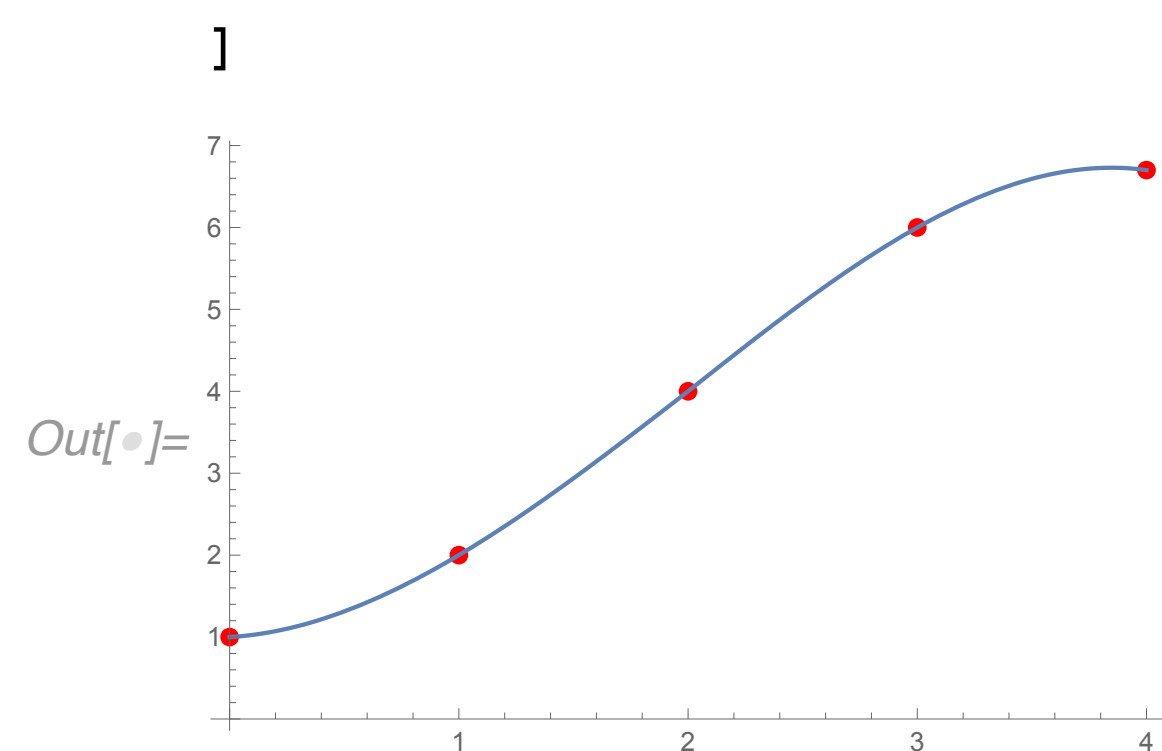
По умолчанию используется сплайн-интерполяция -- кусочная интерполяция полиномами 3 степени (если количество точек данных достаточно для этого)

```
In[ ]:= f = .;
data = {{0, 1}, {1, 2}, {2, 4}, {3, 6}, {4, 6.7}};
f[x_] = Interpolation[data, x]
f[0.5]
```

```
Out[ ]:= InterpolatingFunction[ Domain: {0., 4.}
Output: scalar] [x]
```

```
Out[ ]:= 1.3125
```

```
In[ ]:= Show[
  ListPlot[data, PlotStyle -> {Red, PointSize[Large]}],
  Plot[f[x], {x, 0, Max[Transpose[data][[1]]]},
  PlotLegends -> Placed[{"dsd", "1"}, {Right, Top}]
]
```



Интерполяция данных из таблицы-файла

Импорт таблицы из текстового файла. Первый две строки файла содержат описание столбцов таблицы.

```
In[*]:= NotebookDirectory[]
```

```
Out[*]= /Users/vadim/Documents/Classes/Интегрированные_мат_пакеты/mathematica/
```

```
In[*]:= data = Import[NotebookDirectory[] <> "/atm.txt", "Table"];
data[[1 ;; 10]] // TableForm
```

```
Out[*]//TableForm=
```

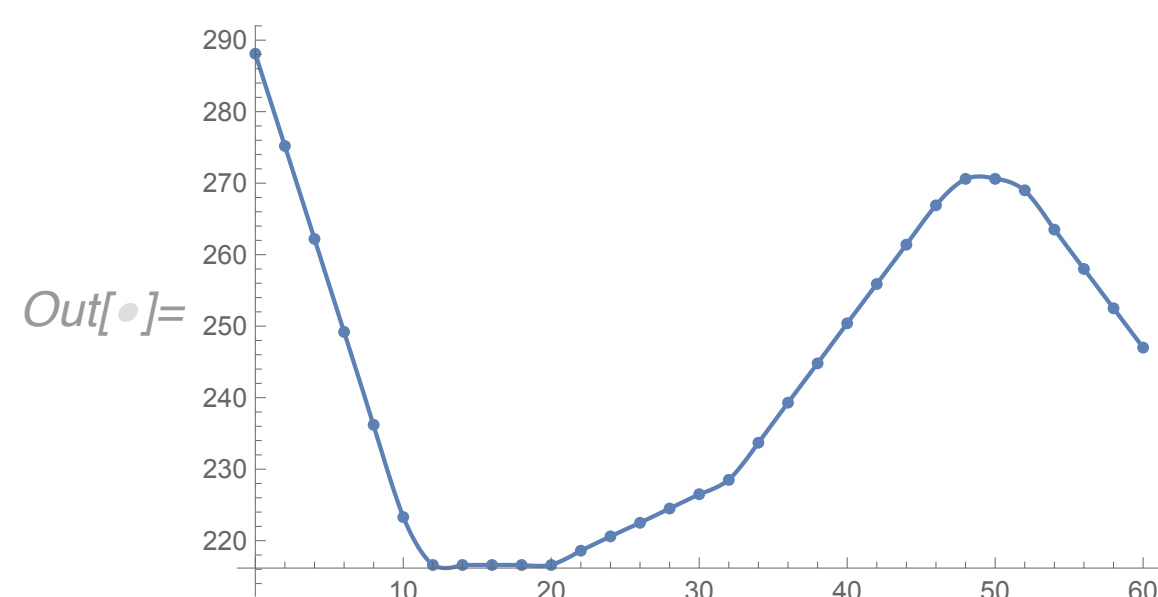
alt	sigma	delta	theta	temp	press	dens	a	visc	k.visc
km	K	N/sq.m	kg/cu.m	m/s	kg/m-s	sq.m/s			
-2	1.2067	1.2611	1.0451	301.2	127800.	1.478	347.9	18.51	0.0000125
0	1.	1.	1.	288.1	101300.	1.225	340.3	17.89	0.0000146
2	0.82168	0.78462	0.9549	275.2	79500.	1.007	332.5	17.26	0.0000171
4	0.66885	0.60854	0.9098	262.2	61660.	0.8193	324.6	16.61	0.0000203
6	0.53887	0.466	0.8648	249.2	47220.	0.6601	316.5	15.95	0.0000242
8	0.42921	0.35185	0.8198	236.2	35650.	0.5258	308.1	15.27	0.000029
10	0.33756	0.26153	0.7748	223.3	26500.	0.4135	299.5	14.58	0.0000353
12	0.25464	0.19146	0.7519	216.6	19400.	0.3119	295.1	14.22	0.0000456

```
In[*]:= data[[3 ;;, {1, 5}]]
temperature[h_] = Interpolation[%][h]
```

```
Out[*]= {{-2, 301.2}, {0, 288.1}, {2, 275.2}, {4, 262.2}, {6, 249.2}, {8, 236.2}, {10, 223.3}, {12, 216.6}, {14, 216.6}, {16, 216.6},
{18, 216.6}, {20, 216.6}, {22, 218.6}, {24, 220.6}, {26, 222.5}, {28, 224.5}, {30, 226.5}, {32, 228.5}, {34, 233.7},
{36, 239.3}, {38, 244.8}, {40, 250.4}, {42, 255.9}, {44, 261.4}, {46, 266.9}, {48, 270.6}, {50, 270.6}, {52, 269.},
{54, 263.5}, {56, 258.}, {58, 252.5}, {60, 247.}, {62, 241.5}, {64, 236.}, {66, 230.5}, {68, 225.1}, {70, 219.6},
{72, 214.3}, {74, 210.3}, {76, 206.4}, {78, 202.5}, {80, 198.6}, {82, 194.7}, {84, 190.8}, {86, 186.9}}
```

```
Out[*]= InterpolatingFunction[ Domain: {{-2., 86.}} Output: scalar][h]
```

```
In[*]:= Show[
Plot[temperature[h], {h, 0, 60}],
ListPlot[data[[3 ;;, {1, 5}]]]
]
```



```
In[*]:=
```

Вектор-функция

Импорт табличных данных из файла. Функция **Import**

```
In[*]:= data = Import[NotebookDirectory[] <> "/atm.txt", "Table"];
data[[1 ;; 5]] // TableForm
```

Out[*]//TableForm=

alt	sigma	delta	theta	temp	press	dens	a	visc	k.visc
km	K	N/sq.m	kg/cu.m	m/s	kg/m-s	sq.m/s			
-2	1.2067	1.2611	1.0451	301.2	127800.	1.478	347.9	18.51	0.0000125
0	1.	1.	1.	288.1	101300.	1.225	340.3	17.89	0.0000146
2	0.82168	0.78462	0.9549	275.2	79500.	1.007	332.5	17.26	0.0000171

Заголовок таблицы занимает первые две строки файла, игнорируем их **{3;;}**.

Выбираем столбцы 1, 5 и 8

Превращаем список троек в список значений пар: значение аргумента, значения вектор-функции

```
In[*]:= {#[[1]], {#[[2]], #[[3]]}} & /@ data[[3 ;;, {1, 5, 8}]]
```

```
tempSpeed[h_] = Interpolation[%][h]
```

```
Out[*]= {{-2, {301.2, 347.9}}, {0, {288.1, 340.3}}, {2, {275.2, 332.5}}, {4, {262.2, 324.6}}, {6, {249.2, 316.5}},
{8, {236.2, 308.1}}, {10, {223.3, 299.5}}, {12, {216.6, 295.1}}, {14, {216.6, 295.1}}, {16, {216.6, 295.1}},
{18, {216.6, 295.1}}, {20, {216.6, 295.1}}, {22, {218.6, 296.4}}, {24, {220.6, 297.7}}, {26, {222.5, 299.1}},
{28, {224.5, 300.4}}, {30, {226.5, 301.7}}, {32, {228.5, 303.}}, {34, {233.7, 306.5}}, {36, {239.3, 310.1}},
{38, {244.8, 313.7}}, {40, {250.4, 317.2}}, {42, {255.9, 320.7}}, {44, {261.4, 324.1}}, {46, {266.9, 327.5}},
{48, {270.6, 329.8}}, {50, {270.6, 329.8}}, {52, {269., 328.8}}, {54, {263.5, 325.4}}, {56, {258., 322.}},
{58, {252.5, 318.6}}, {60, {247., 315.1}}, {62, {241.5, 311.5}}, {64, {236., 308.}}, {66, {230.5, 304.4}},
{68, {225.1, 300.7}}, {70, {219.6, 297.1}}, {72, {214.3, 293.4}}, {74, {210.3, 290.7}}, {76, {206.4, 288.}},
{78, {202.5, 285.3}}, {80, {198.6, 282.5}}, {82, {194.7, 279.7}}, {84, {190.8, 276.9}}, {86, {186.9, 274.1}}}
```

```
Out[*]= InterpolatingFunction[ Domain: {{-2., 86.}}
Output dimensions: {2}] [h]
```

```
In[*]:= tempSpeed[35]
```

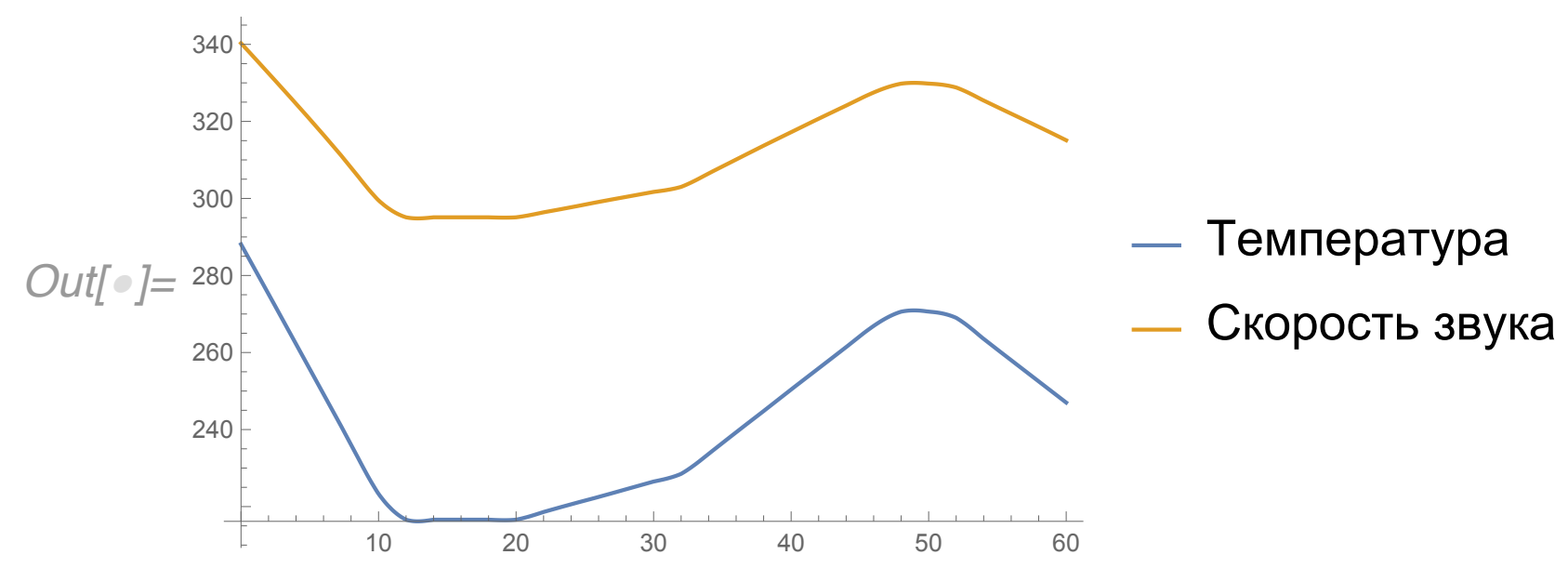
```
Out[*]= {236.4813, 308.2938}
```

Вектор-функция

```
In[*]:= tempSpeed[10]
```

```
Out[*]:= {223.3, 299.5}
```

```
In[*]:= Plot[{tempSpeed[h][[1]], tempSpeed[h][[2]]}, {h, 0, 60}, PlotLegends -> {"Температура", "Скорость звука"}]
```



```
In[*]:=
```

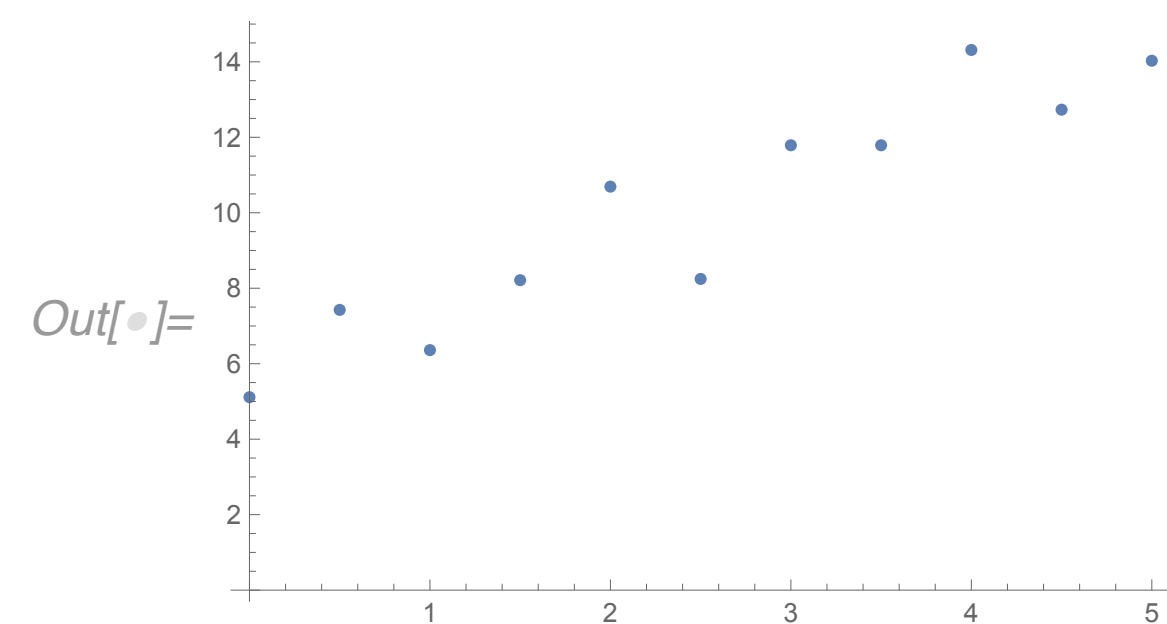
Линейная регрессия

Определение коэффициентов линейной по параметрам β_i функции $g = \sum_{i=1}^n \beta_i f_i$ методом наименьших квадратов,

Предположим, что есть некоторый набор пар значений x и y , полученный, например, в результате эксперимента.

```
In[ ]:= data = Table[{x, 2 x + 5 + RandomReal[{-2, 2}]}, {x, 0, 5, 0.5}]
ListPlot[data]
```

```
Out[ ]:= {{0., 5.11227}, {0.5, 7.426347}, {1., 6.360115}, {1.5, 8.21346}, {2., 10.69225}, {2.5, 8.246482}, {3., 11.78824},
{3.5, 11.78848}, {4., 14.31369}, {4.5, 12.73157}, {5., 14.02862}}
```

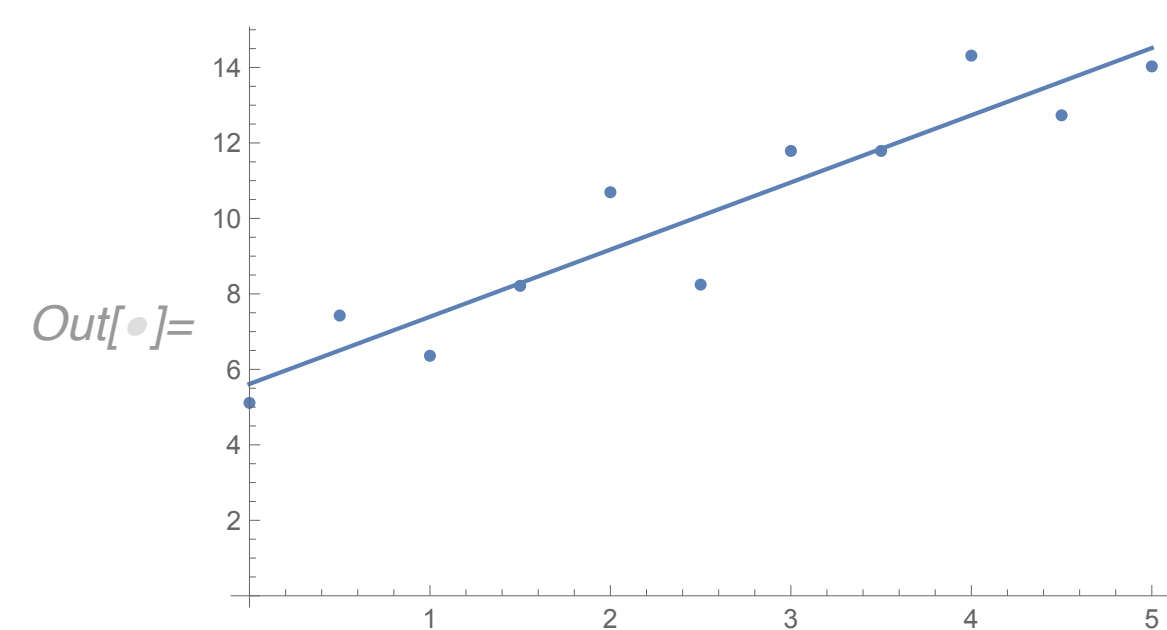


Необходимо оценить параметры этой зависимости, задавшись некоторой моделью.

Предположим, что эта зависимость линейна: $y = a x + b$. Оценим a и b .

```
In[ ]:= Fit[data, {1, x}, x]
Show[ListPlot[data], Plot[%, {x, 0, 5}]]
```

```
Out[ ]:= 5.613348 + 1.78017 x
```



Линейная регрессия

Другой набор данных

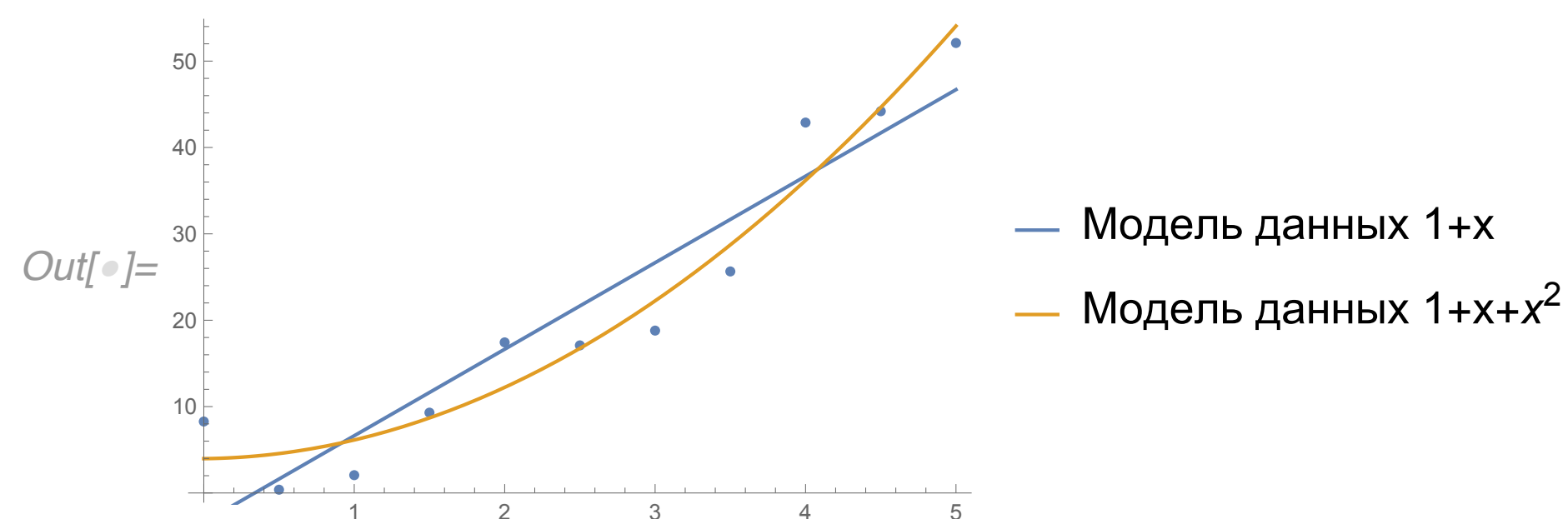
```
In[*]:= data = Table[{x, 2 x^2 + 5 + 3 RandomReal[{-2, 2}]}, {x, 0, 5, 0.5}]
```

```
Out[*]:= {{0., 8.271574}, {0.5, 0.3785568}, {1., 2.061507}, {1.5, 9.297876}, {2., 17.43403}, {2.5, 17.08787}, {3., 18.79875},
{3.5, 25.64751}, {4., 42.88278}, {4.5, 44.19712}, {5., 52.09712}}
```

Сравним две модели $y = a + b x$ и $y = a + b x + c x^2$

```
In[*]:= ListPlot[data];
{
  Fit[data, {1, x}, x],
  Fit[data, {1, x, x^2}, x]
}
Show[%%, Plot[%, {x, 0, 5}, PlotLegends -> {"Модель данных 1+x", "Модель данных 1+x+x^2"}]]
```

```
Out[*]:= {-3.391838 + 10.01691 x, 3.975195 + 0.1941947 x + 1.964542 x^2}
```



```
In[*]:= {Fit[data, {1, x}, x], Fit[data, {1, x, x^2}, x]}
```

```
Out[*]:= {-3.391838 + 10.01691 x, 3.975195 + 0.1941947 x + 1.964542 x^2}
```

Квадратичная зависимость от x лучше отражает поведение “экспериментальных” данных.

Линейная регрессия

Использование функции **LinearModelFit** для оценки качества регрессионного анализа

```
In[ ]:= data = Table[{x, 2 x^2 + 5 + RandomReal[{-2, 2}]}, {x, 0, 5, 0.2}];
```

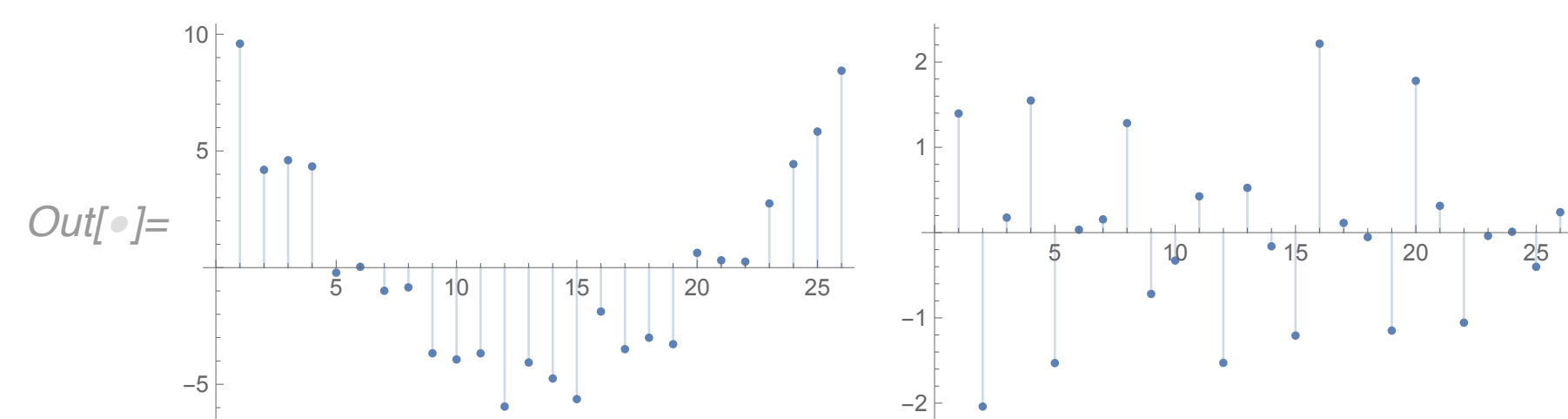
```
In[ ]:= lm1 = LinearModelFit[data, {1, x}, x]
      lm2 = LinearModelFit[data, {1, x, x^2}, x]
```

```
Out[ ]:= FittedModel[-2.791856 + 9.761013 x]
```

```
Out[ ]:= FittedModel[5.405162 - 0.4852606 x + 2.049255 x^2]
```

Ошибки -- отклонение точек от аппроксимирующей функции

```
In[ ]:= GraphicsRow[ListPlot[#, {"FitResiduals"}, Filling -> Axis] & /@ {lm1, lm2}]
```



```
In[ ]:= lm1["FitResiduals"]
```

```
Out[ ]:= {9.593488, 4.189974, 4.602119, 4.335284, -0.2179638, 0.03344504, -0.9924388, -0.8476186, -3.669966, -3.934746,
-3.673446, -5.953897, -4.065913, -4.751152, -5.633915, -1.883939, -3.493783, -3.003571, -3.280616, 0.6314058,
0.3133172, 0.2553309, 2.748732, 4.435621, 5.828143, 8.436106}
```


Линейная регрессия

Для оценки качества модели может использоваться коэффициент детерминации.

Коэффициент детерминации для модели с константой принимает значения от 0 до 1.

Чем ближе значение коэффициента к 1, тем сильнее зависимость.

Коэффициент детерминации показывает долю вариации зависимой переменной, “объясняемой” используемой моделью.

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}, \quad SS_{\text{res}} = \sum_{i=1}^n (y_i - \hat{y}_i)^2, \quad SS_{\text{tot}} = \sum_{i=1}^n (y_i - \bar{y})^2, \quad \bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$$

SS_{res} – сумма квадратов остатков регрессии

SS_{tot} – общая сумма квадратов

```
In[*]:= {lm1["RSquared"], lm2["RSquared"]}
```

```
Out[*]:= {0.932087, 0.9966779}
```

Вторая модель лучше отражает “экспериментальные” данные

```
In[*]:= lm1["ParameterTable"]
```

```
Out[*]:=
```

	Estimate	Standard Error	t-Statistic	P-Value
1	-3.248521	1.621129	-2.003863	0.05649724
x	10.09173	0.5560428	18.1492	1.602532 × 10 ⁻¹⁵

```
In[*]:= lm2["ParameterTable"]
```

```
Out[*]:=
```

	Estimate	Standard Error	t-Statistic	P-Value
1	4.689531	0.5244599	8.94164	6.03827 × 10 ⁻⁹
x	0.1691669	0.4857524	0.3482576	0.7308142
x ²	1.984513	0.09384526	21.14665	1.429492 × 10 ⁻¹⁶

Параметр t-Statistics и p-value характеризуют достоверность гипотезы (нулевой гипотезы) о том, что значение параметра равно нулю, что означает отсутствие влияния независимой переменной на зависимую.

Чем меньше значение p-Value тем лучше. Обычно, если значение p-value ниже некоторого порогового уровня, например 0.05 или 0.01, то нуль-гипотеза считается **ложной**, что означает значимость влияния фактора (независимой переменной), поскольку найденный коэффициент при этом факторе не равен нулю.

В рассматриваемом примере значимо влияние параметра при x^2 и свободного коэффициента (1). Гипотезу о равенстве нулю множителя при x следует принять, поскольку $p\text{-value} > 0.05$.

Нелинейная регрессия

```
In[ ]:= ClearAll["Global`*"]
```

Функция `FindFit` определяет коэффициенты нелинейной функции “наилучшим” образом, приближающей табличные данные

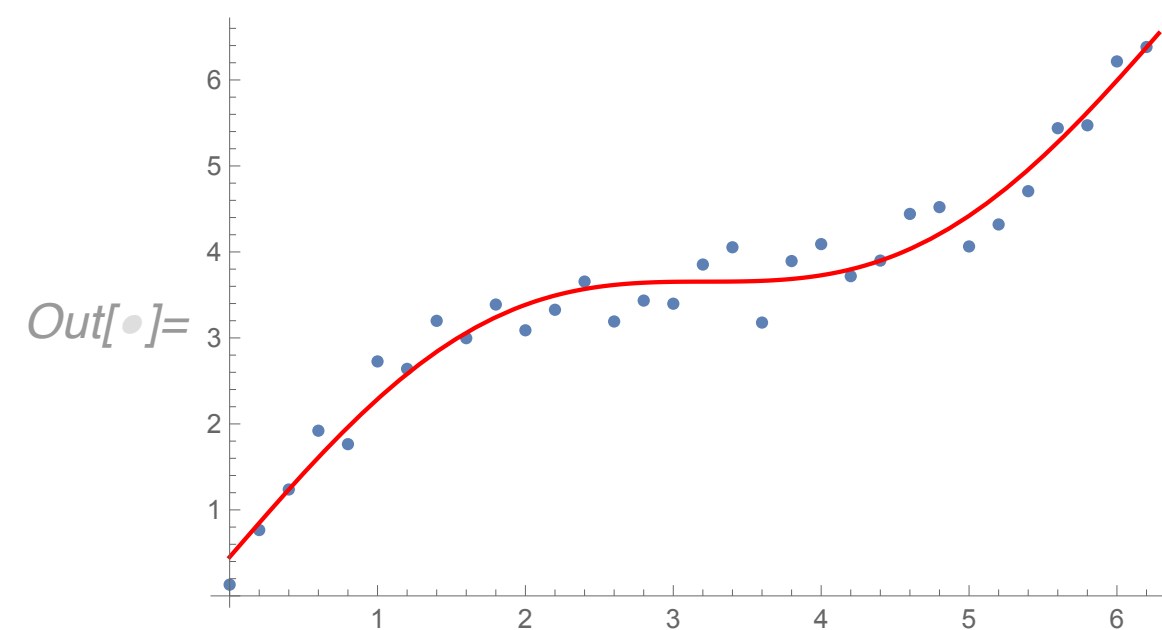
```
In[ ]:= data = Table[{x, x + Sin[1 x] + Random[]}, {x, 0, 2 π, 0.2}];
```

```
model = a + b x + c Sin[d x];
```

```
FindFit[data, a + b x + c Sin[d x], {a, b, c, d}, x]
```

```
Show[ListPlot[data], Plot[model /. %, {x, 0, 2 π}, PlotStyle → Red]]
```

```
Out[ ]:= {a → 0.4518937, b → 0.9942621, c → 1.017256, d → 0.9753453}
```



Нелинейная регрессия

Функция FindFit определяет коэффициенты нелинейной функции “наилучшим” образом, приближающей табличные данные

```
In[*]:= model = a + b x + c Sin[d x];
nlm = NonlinearModelFit[data, a + b x + c Sin[d x], {a, b, c, d}, x]
nlm["ParameterTable"]
```

```
Out[*]:= FittedModel [0.4518937 + 0.9942621 x + 1.017256 Sin[0.9753453 x]]
```

```
Out[*]:=
```

	Estimate	Standard Error	t-Statistic	P-Value
a	0.4518937	0.158964	2.842742	0.008256768
b	0.9942621	0.04816668	20.64212	1.768031×10^{-18}
c	1.017256	0.11831	8.598229	2.414936×10^{-9}
d	0.9753453	0.02185721	44.62351	1.461638×10^{-27}

Доверительные интервалы для параметров (по умолчанию, уровень доверия при вычислении доверительного интервал равен 95%)

```
In[*]:= nlm["ParameterConfidenceIntervals"] // TableForm
```

```
Out[*]//TableForm=
```

0.1262707	0.7775168
0.8955972	1.092927
0.7749094	1.259603
0.9305729	1.020118

“Большая уверенность” в результате => больший интервал

```
In[*]:= nlm["ParameterConfidenceIntervals", ConfidenceLevel -> 0.99] // TableForm
```

```
Out[*]//TableForm=
```

0.01263441	0.891153
0.861165	1.127359
0.6903349	1.344178
0.9149482	1.035743

Интегрирование дифференциальных уравнений

Остановка процесса интегрирования при заданном условии

Задача баллистики при действии на тело сопротивления воздуха. Сопротивление воздуха пропорционально скоростному напору

$$m \frac{d\mathbf{v}}{dt} = m \mathbf{g} - \frac{\mathbf{v}}{|\mathbf{v}|} C_x \frac{\rho v^2}{2} S_m$$

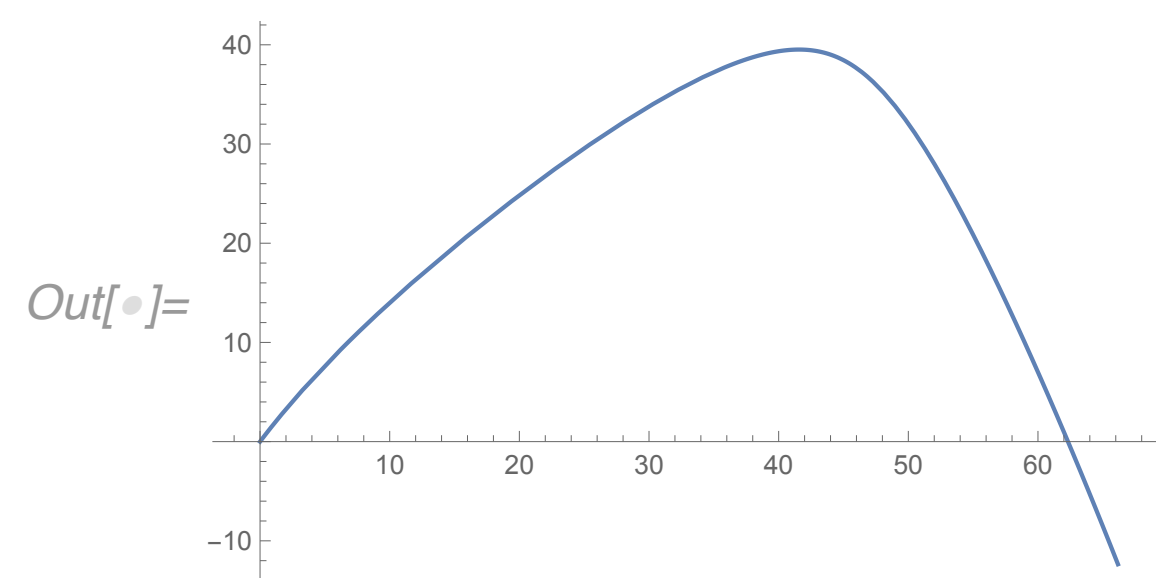
где m -- масса тела, \mathbf{g} -- вектор ускорения свободного падения, \mathbf{v} -- вектор скорости тела, C_x -- коэффициент лобового сопротивления, ρ -- плотность воздуха, S_m -- характерная площадь.

Ox - горизонтальная ось, Oy - вертикальная ось, направленная вверх

```
In[ ]:= v = {x'[t], y'[t]};
p = {g -> 9.807, Cx -> 1.0, rho -> 1.2, m -> 1.0, V0 -> 100, alpha0 -> 60°, Sm -> 0.1};
eq = {MapThread[Equal, {m D[v, t], {0, -1} m g - (v / Sqrt[v.v]) Cx (rho v^2 / 2) Sm}],
      {x'[0] == V0 Cos[alpha0], y'[0] == V0 Sin[alpha0], x[0] == 0, y[0] == 0} /. p;
solution = NDSolve[eq, {x[t], y[t]}, {t, 0, 7}]

Out[ ]:= {{x[t] -> InterpolatingFunction[Domain: {{0., 7.}} Output: scalar][t], y[t] -> InterpolatingFunction[Domain: {{0., 7.}} Output: scalar][t]}}
```

```
In[ ]:= ParametricPlot[{x[t], y[t]} /. solution, {t, 0, 7}, AspectRatio -> 0.6]
```




Процесс интегрирования продолжается 5 секунд, высота становится отрицательной

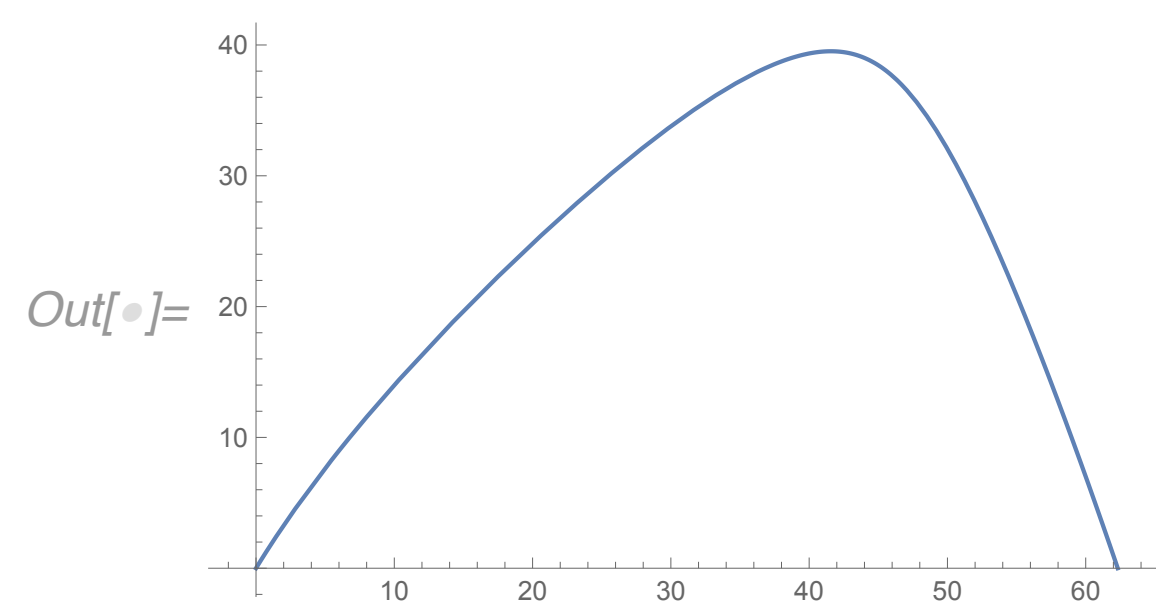
Интегрирование дифференциальных уравнений

WhenEvent

```
In[ ]:= solution = NDSolve[{eq, WhenEvent[y[t] < 0.0, "StopIntegration"]} // Flatten, {x[t], y[t]}, {t, 0, 7}]
te = solution[[1, 1, 2, 0, 1, 1, 2]]
ParametricPlot[{x[t], y[t]} /. solution, {t, 0, te}, AspectRatio -> 0.6]
```

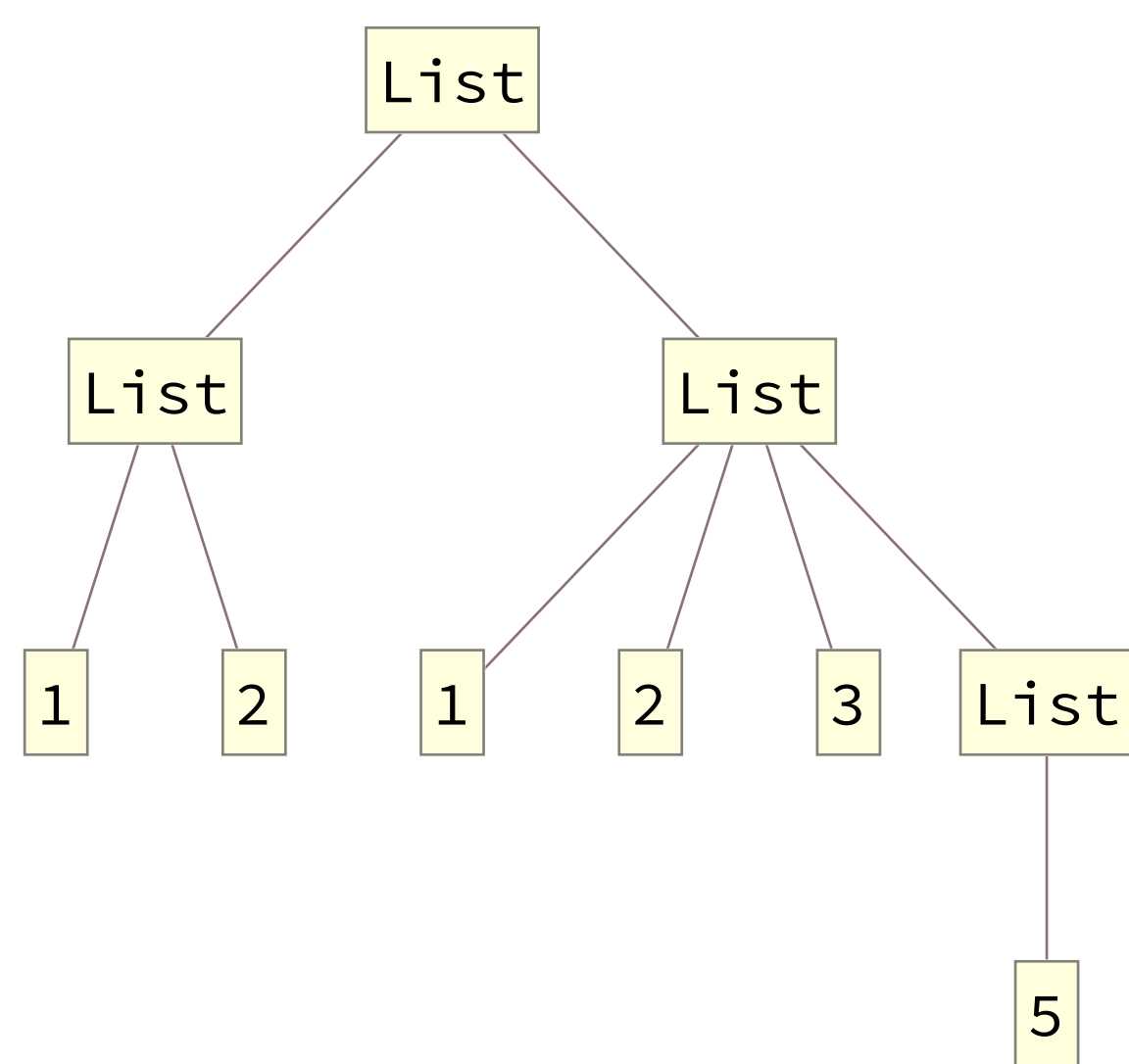
```
Out[ ]:= {{x[t] -> InterpolatingFunction[ Domain: {{0., 6.06}} Output: scalar][t], y[t] -> InterpolatingFunction[ Domain: {{0., 6.06}} Output: scalar][t]}}
```

```
Out[ ]:= 6.055457
```



```
In[ ]:= TreeForm[{{1, 2}, {1, 2, 3, {5}}}]
```

```
Out[ ]:= TreeForm=
```



Интегралы

Неопределенный интеграл

```
In[*]:= Integrate[Cos[x] x^2, x]
```

```
Out[*]:= 2 x Cos[x] + (-2 + x^2) Sin[x]
```

Определенный интеграл

```
In[*]:= Integrate[Cos[x] x^2, {x, 0, 1}]
```

```
Out[*]:= 2 Cos[1] - Sin[1]
```

Численное интегрирование

```
In[*]:= NIntegrate[Cos[x] x^2, {x, 0, 1}]
```

```
Out[*]:= 0.2391336
```

```
In[*]:= NIntegrate[Cos[0.1 Exp[x]] Sin[0.1 π Exp[x]], {x, 0, 10}]
```

```
Out[*]:= 1.258824
```

Интеграл. Работа силы.

Проверим правильность решения задачи баллистики с учетом сопротивления воздуха. Проинтегрируем уравнения до момента падения на землю. Изменение потенциальной энергии равно нулю (движение началось и закончилось на нулевой высоте), поэтому изменение кинетической энергии тела будет равно работе силы сопротивления воздуха -- единственной непотенциальной силой, действующей на тело.

```
In[*]:= solution = NDSolve[{eq, WhenEvent[y[t] < 0.0, "StopIntegration"]} // Flatten, {x[t], y[t], x'[t], y'[t]}, {t, 0, 7}];
te = solution[[1, 1, 2, 0, 1, 1, 2]]
```

Out[*]= 6.055457

Сила сопротивления воздуха (вектор)

$$\text{In[*]} := \mathbf{F} = -\frac{\mathbf{v}}{\sqrt{\mathbf{v} \cdot \mathbf{v}}} Cx \frac{v^2}{2} \rho S m$$

$$\text{Out[*]} = \left\{ -\frac{Cx S m \rho x'[t]^3}{2 \sqrt{x'[t]^2 + y'[t]^2}}, -\frac{Cx S m \rho y'[t]^3}{2 \sqrt{x'[t]^2 + y'[t]^2}} \right\}$$

Элементарная работа силы сопротивления

```
In[*]:= dA = F.v /. solution /. p
```

$$\text{Out[*]} = \left\{ -\frac{0.06 \text{ InterpolatingFunction} \left[\left\{ \left\{ 0., 6.06 \right\} \right\} \right] [t]^4}{\sqrt{\text{InterpolatingFunction} \left[\left\{ \left\{ 0., 6.06 \right\} \right\} \right] [t]^2 + \text{InterpolatingFunction} \left[\left\{ \left\{ 0., 6.06 \right\} \right\} \right] [t]^2}}, -\frac{0.06 \text{ InterpolatingFunction} \left[\left\{ \left\{ 0., 6.06 \right\} \right\} \right] [t]^4}{\sqrt{\text{InterpolatingFunction} \left[\left\{ \left\{ 0., 6.06 \right\} \right\} \right] [t]^2 + \text{InterpolatingFunction} \left[\left\{ \left\{ 0., 6.06 \right\} \right\} \right] [t]^2}} \right\}$$

Интеграл. Работа силы.

Суммарная работа силы сопротивления:

```
In[*]:= totalA = NIntegrate[dA, {t, 0, te}][[1]]
```

```
Out[*]:= -4905.529
```

Начальная кинетическая энергия + работа сил сопротивления

```
In[*]:= (m V0^2 / 2 /. p) + totalA
```

```
Out[*]:= 94.47054
```

Конечная кинетическая энергия

```
In[*]:= (m v.v / 2 /. solution /. p /. t -> te)[[1]]
```

```
Out[*]:= 94.47046
```

Результаты совпадают до 3 знака после десятичной точки.

Если увеличить точность численного интегрирования **NDSolve**, указав опцию **PrecisionGoal -> 20**, то результаты совпадут до 5 знака после десятичной точки.

Краевая задача





Краевая задача (граничная задача) — задача о нахождении решения заданного дифференциального уравнения (системы дифференциальных уравнений), удовлетворяющего краевым (граничным) условиям в концах интервала или на границе области.

Для описанной ранее системы дифференциальных уравнений движения груза найдем решение для нулевой начальной и конечной высоты и конечного расстояния от точки броска 10 метров, при этом время движения должно быть равно 3 секундам.

```
In[*]:= eq = {MapThread[Equal, {m D[v, t], {0, -1} m g -  $\frac{v}{\sqrt{v.v}}$  Cx  $\frac{v^2}{2}$  ρ Sm}], x[0] == 0, y[0] == 0, y[3] == 0, x[3] == 10} /. p
```

```
Out[*]:= {{1. x''[t] == - $\frac{0.06 x'[t]^3}{\sqrt{x'[t]^2 + y'[t]^2}}$ , 1. y''[t] == -9.807 -  $\frac{0.06 y'[t]^3}{\sqrt{x'[t]^2 + y'[t]^2}}$ }, x[0] == 0, y[0] == 0, y[3] == 0, x[3] == 10}
```

```
In[*]:= sol = NDSolve[eq, {x[t], y[t], x'[t], y'[t]}, {t, 0, 3}]
```

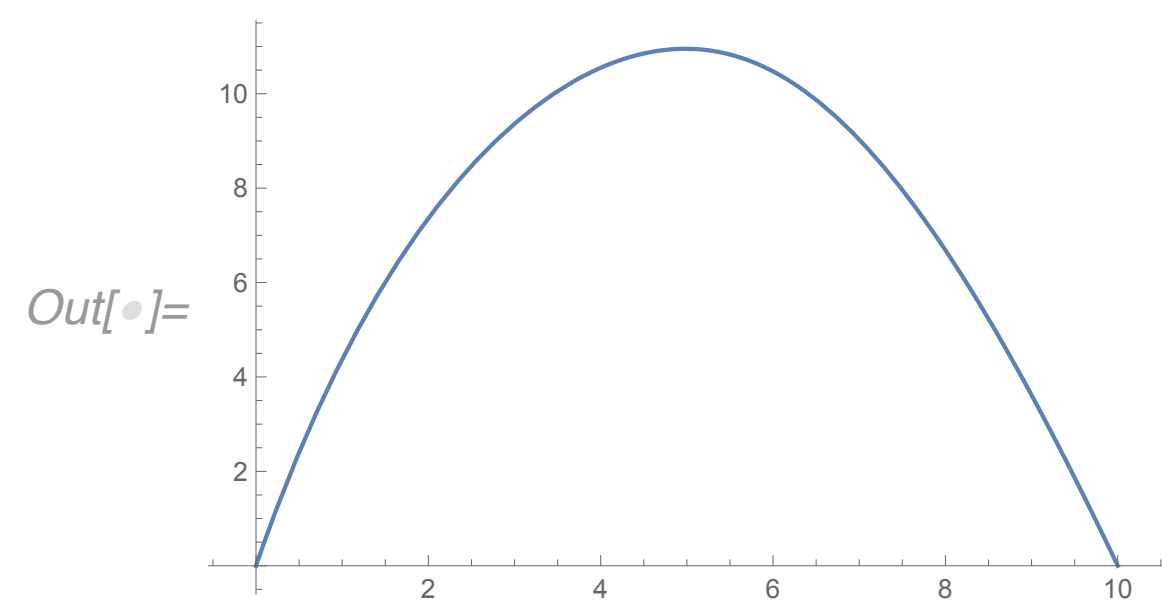
```
Out[*]:= {{x[t] → InterpolatingFunction[ Domain: {{0., 3.}} Output: scalar][t], y[t] → InterpolatingFunction[ Domain: {{0., 3.}} Output: scalar][t], x'[t] → InterpolatingFunction[ Domain: {{0., 3.}} Output: scalar][t], y'[t] → InterpolatingFunction[ Domain: {{0., 3.}} Output: scalar][t]}}
```

```
In[*]:= ParametricPlot[{x[t], y[t]} /. sol, {t, 0, 3}, AspectRatio → 0.6]
```



Краевая задача

```
In[ ]:= ParametricPlot[{x[t], y[t]} /. sol, {t, 0, 3}, AspectRatio -> 0.6]
```



Начальный угол бросания

```
In[ ]:= ArcTan[y'[t] / x'[t]] * 180 / Pi /. sol /. t -> 0
```

Out[]:= {79.34529}

Начальная скорость

```
In[ ]:= Norm[v] /. sol /. t -> 0
```

Out[]:= {21.07218}

```
In[ ]:= Plot[Sqrt[v.v] /. sol, {t, 0, 3}, PlotRange -> All, AxesLabel -> {"t, c", "V, м/с"}]
```

