



Mathematica для математиков. Часть 2. Графические возможности системы

В системе *Mathematica* реализовано огромное количество графических функций. В данном разделе представлен обзор основных. Предполагается, что читатель выполняет приведенные команды в системе *Mathematica*, поэтому иногда результаты их выполнения (графики) не приводятся.

Примеры, приводимые в данном пособии, проверялись в версии пакета *Mathematica* 9.0. В более старых версиях часть примеров работать не будет. Особенности некоторых функций *Mathematica* 5 мы иногда описываем отдельно.

Оглавление

2.1 Двумерные графики.	2
2.1.1 Обыкновенные графики функций.	2
2.1.2 Параметрические графики.....	7
2.1.3 График в полярных координатах.....	8
2.1.4 Графики неявных функций.	9
2.1.5 Области между кривыми.	10
2.2 Трехмерные графики.....	16
2.2.1 График функций заданных явно.	16
2.2.2 График параметрических функций.	21
2.2.3 Графики в сферических и цилиндрических координатах.....	24
2.3 Другие графические возможности	27
2.3.1 Линии уровня функции.....	27
2.3.2 Трехмерные области и поверхности, заданные неявно.....	30
2.3.3 Графики списков значений.....	33
2.3.4 Анимация и манипуляторы	42
2.3.5 Комбинирование графиков	50
2.3.6 Графические примитивы.	52
2.4 Графические иллюстрации к решению прикладных задач.....	57
2.4.1 Несколько геометрических примеров.....	57
2.4.2 Моделирование механических движений	61
2.4.3 Движение жидкости в трубах	68
2.4.4 Двумерные волновые движения	71
2.4.5 Одномерные волновые колебания.....	77
2.4.6 Задачи сопротивления материалов и теории упругости	90
2.4.7 Задачи электростатики.....	99
Литература.	104

2.1 Двумерные графики.

2.1.1 Обыкновенные графики функций.

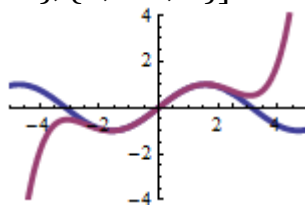
График функции, заданной явным уравнением $y=f(x)$, строится с помощью функции `Plot`.

`Plot[x2, {x, -1, 1}]`

Первый аргумент – выражение $f(x)$ (правая часть уравнения), график которого надо построить. Второй аргумент определяет интервал изменения переменной x . *Mathematica* автоматически выбирает масштаб по вертикальной оси и выбирает достаточное количество точек, чтобы кривая была гладкой. Все эти параметры можно изменять с помощью опций. Если вы рисуете график функции с большими значениями, такой как тригонометрическая функция `Tan[x]`, *Mathematica* пробует подобрать разумные вертикальные пределы.

Можно нарисовать два и более графика в одних и тех же осях, используя список функций, как первый аргумент команды `Plot`

`Plot[{Sin[x], x - x3/6 + x5/120}, {x, -5, 5}]`



Можно изменить цвет и/или штриховой узор кривой, используя опцию `PlotStyle`. Следующая кривая имеет правильный штриховой узор.

`Plot[Sin[x], {x, -7, 7}, PlotStyle -> Dashing[{0.05}]]`

Штрихи и пробелы имеют длину 0.05 ширины графика. Если у вас есть более чем одна кривая на одном графике, вы можете различить их по цвету и/или штриховому узору. В следующем примере первый элемент списка `PlotStyle` применяется к первому графику, второй – ко второму. Функция `Hue[...]` определяет цвет; ее аргумент должен изменяться в диапазоне от 0 до 1.

`Plot[{Sin[x], x - x3/6 + x5/120}, {x, -7, 7}, PlotStyle -> {Hue[2/3], Hue[1/3]}]`

Можно пометить график, используя опцию `PlotLabel`

`Plot[Sin[x], {x, 0, 2Pi}, PlotLabel -> "The Sine Function"]`

Любой текст в *Mathematica* должен быть заключен в двойные кавычки.

Можно пометить оси графика, используя опцию `AxesLabel`

`Plot[Sin[x], {x, 0, 2Pi}, AxesLabel -> {"x", "Sin[x]}]`

Можно поместить весь график в рамку

`Plot[Sin[x], {x, 0, 2Pi}, Frame -> True]`

Можно нарисовать координатную сетку

`Plot[Sin[x], {x, 0, 2Pi}, GridLines -> Automatic, Frame -> True]`

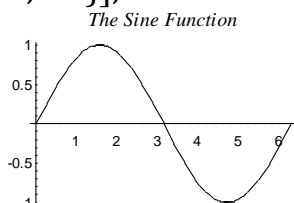
Можно изменить шрифты, используемые для текстовых элементов графиков.

В *Mathematica* 5 и более ранних версиях для этого использовалась опция `DefaultFont`. Шрифты задаются, как список: первый элемент – имя шрифта, второй – размер шрифта.

`Plot[Sin[x], {x, 0, 2Pi}, DefaultFont → {"Helvetica – Oblique", 18}]`

Для изменения шрифта индивидуального текстового элемента (метки, надписи), используют функцию `FontForm`. Она имеет два аргумента: первый – текст выражения, второй – спецификация шрифта.

`Plot[Sin[x], {x, 0, 2Pi}, PlotLabel → FontForm["The Sine Function", "Palatino – Italic", 18], DefaultFont → {"Helvetica", 14}]`



В *Mathematica* 6 использовались опции `TextStyle` и `StyleForm`. В *Mathematica* 9 для изменения начертания текстовых элементов используются опция `BaseStyle` и функция `Style`.

`Plot[Sin[x], {x, 0, 2Pi}, PlotLabel → Style["The Sine Function", {"Palatino – Italic", 18}], BaseStyle → {"Helvetica", 14}]` (* см. след. рисунок слева *)

Для изменения шрифта индивидуального текстового элемента (метки, надписи), используется функция `Style`, а для задания базового стиля других текстовых элементов используется опция `BaseStyle`.

`Plot[Sin[x]^2, {x, 0, 2π}, PlotLabel → "Квадрат синуса", BaseStyle → {FontWeight → "Bold", FontSize → 18, FontFamily → "Arial", FontSlant → "Italic"}]` (* след. рисунок в центре *)



Опция `LabelStyle` используется для изменения внешнего вида меток.

`Plot[Sin[x], {x, 0, 10}, LabelStyle → {Blue, 16}]` (* пред. рис. справа *)

Тот же график строится следующей командой

`Plot[Sin[x], {x, 0, 10}, LabelStyle → Directive[Blue, 16]]`

Любой текст, выводимый в пределах графической области, можно включить в тело функции `Style` первым аргументом и последующими аргументами изменить его внешний вид. Следующие несколько примеров демонстрируют возможности функции `Style` (часто функцию `Style` со всеми ее аргументами включают в тело функции `Text`).

`Style["Hello Kharkov", Blue, Italic, 24]`

Hello Kharkov

Table[Style["AaBbCc", Large, *p*], {*p*, {Bold, Italic, Underlined, Purple}}}]
{AaBbCc, AaBbCc, AaBbCc, AaBbCc}

Style["Mathematica", "Subtitle"]

Mathematica

Table[Style[Sqrt[Abs[x]], Background → *c*],
{*c*, {LightYellow, LightBlue, LightRed, LightGreen}}]

{ $\sqrt{\text{Abs}[x]}$, $\sqrt{\text{Abs}[x]}$, $\sqrt{\text{Abs}[x]}$, $\sqrt{\text{Abs}[x]}$ }

Style[1/Sqrt[2], FontColor → Red, Background → LightGray]

$\frac{1}{\sqrt{2}}$

Table[Style["AaBbCcDd", FontFamily → *p*], {*p*, {"Courier", "Times", "Helvetica"}}]
{AaBbCcDd, AaBbCcDd, AaBbCcDd}

Table[Text[Style["Kharkov", FontSize → *p*]], {*p*, {12, 24, 36}}]

{Kharkov, Kharkov, Kharkov}

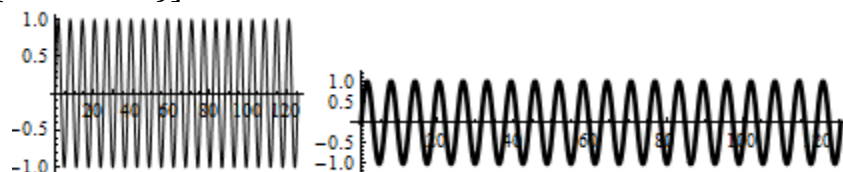
Table[Style[Sqrt[2], Magnification → *m*], {*m*, {1/2, 1, 2}}]

{ $\sqrt{2}$, $\sqrt{2}$, $\sqrt{2}$ }

Имеется и другие опции функции `Style`, с которыми вы можете познакомиться по справочной системе.

Mathematica обычно строит графики с отношением длина/ширина $\frac{1}{2}(1 + \sqrt{5})$, что приблизительно равно 1.61803 (известно, как Золотое Сечение). Изменить экранное соотношение высоты и ширины графической области можно с помощью опции `AspectRatio`. Например, следующий график слева выглядит густовато

Plot[Sin[x], {x, 0, 40Pi}]



Это будет выглядеть лучше, если график немного растянуть (т.е. уменьшить отношение высота/ширина)

Plot[Sin[x], {x, 0, 40Pi}, AspectRatio → 0.2] (* пред. рисунок справа *)

Опция `AspectRatio`→число определяет отношение длин осей графика (высота/длина). В следующем примере физический наклон линии отличается от того, как мы себе это представляем из вида функции

Plot[2x, {x, 0, 5}]

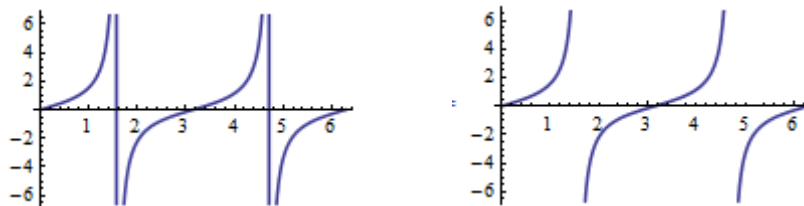
Так получается потому, что по умолчанию задается разный масштаб в горизонтальном и вертикальном направлении. Установка `AspectRatio`→`Automatic` обеспечивает для этого примера одинаковый масштаб по *x* и *y*

Plot[2x, {x, 0, 5}, AspectRatio → Automatic]

Для исключения из графика особых точек используется опция `Exclusions`, которой передается список точек, в которых график не следует строить.

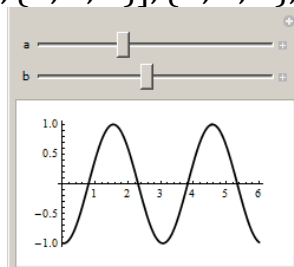
`Plot[Tan[x], {x, 0, 2Pi}]` (* следующий график слева *)

`Plot[Tan[x], {x, 0, 2Pi}, Exclusions → {x = Pi/2, 3Pi/2}];` (* след. гр. справа *)



Начиная с версии *Mathematica* 6 в системе появились специальные графические панели, содержащие элементы, с помощью которых можно динамически управлять выводом, в частности, строить графики. В следующем примере мы используем элемент «манипулятор»

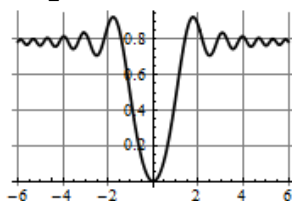
`Manipulate[Plot[Sin[ax + b], {x, 0, 6}], {a, 1, 4}, {b, 0, 10}]`



Перемещая указатели, в этом примере мы меняем значение параметров a и b , используемых при построении графика.

При построении графиков иногда нужно использовать функцию `Evaluate[expr]`. Она символично вычисляет выражение, стоящее в ее аргументе. В следующем примере команда `Evaluate` заставляет ядро *Mathematica* сначала вычислить интеграл символично, а потом построить график. Иначе для каждого конкретного x интеграл будет вычисляться заново.

`Plot[Evaluate $\left[\int_0^x \int_0^z \text{Cos}[zt] dt dz \right], \{x, -6, 6\}, \text{GridLines} \rightarrow \text{Automatic}]$`



Если попробовать построить график следующей командой, без использования функции `Evaluate`, то вероятно вам придется прервать вычисление с помощью комбинации клавиш `Alt - .` (точка).

`Plot[$\int_0^x \int_0^z \text{Cos}[zt] dt dz$, {x, -6, 6}]` (* не делайте так *)

Функция `Evaluate` выполняет символичные вычисления!

Функцию `Evaluate` приходится использовать тогда, когда выражение, график которого следует построить, создается с атрибутом `HoldFirst`. Многие объекты/функции системы *Mathematica* имеют атрибуты, которые

управляют поведением объектов/функций. Узнать атрибуты выражения или функции можно, например, так

```
Attributes[Cos]
```

```
{Listable, NumericFunction, Protected}
```

Приведенные здесь атрибуты означают, что функция `Cos` в качестве аргумента может принимать списки (`Listable`); функция `Cos` будет возвращать результат в виде десятичного числа, если аргумент является десятичным числом (`NumericFunction`); и имя `Cos` защищено (`Protected`), т.е. если пользователь попытается создать переменную с именем `Cos`, то получит сообщение об ошибке.

Атрибут `HoldFirst` функции означает, что ее первый аргумент находится в невычисляемом состоянии. Например, наложим этот атрибут на функцию `Sin`.

```
SetAttributes[Sin, HoldFirst]
```

```
Attributes[Sin]
```

```
{HoldFirst, Listable, NumericFunction, Protected}
```

Теперь попытка построить график выражения `Sin[x]` не увенчается успехом.

```
Plot[Sin[x], {x, 0, 2π}] (* График не строится *)
```

Но, используя функцию `Evaluate`, вы можете все же его построить

```
Plot[Sin[Evaluate[x]], {x, 0, 2π}] (* Все Ок *)
```

После того, как мы «поигрались» с атрибутами функции `Sin`, снимем с нее атрибут `HoldFirst`.

```
ClearAttributes[Sin, HoldFirst]
```

```
Attributes[Sin]
```

```
{Listable, NumericFunction, Protected}
```

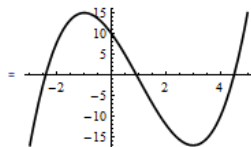
Еще один полезный случай использования функции `Evaluate` продемонстрирован в следующем примере.

```
i = 0;
```

```
f[x_] = x3 - 3x2 - 9x + 10;
```

```
g = Function[{x}, i ++; f[x]] (* создаем функцию g *)
```

```
Plot[g[x], {x, -3, 5}]
```



```
i
```

```
453
```

После построения графика переменная `i` равна 453. Это значит, что определение функция `g[x]` создавалось 453 раза, потому, что при вычислении значения функции в каждой точке графика также создавалось определение функции! Но

```
i = 0; Plot[Evaluate[g[x]], {x, -3, 5}]; i
```

```
1
```

Переменная $i=1$. Значит определение функция $g[x]$ создавалось только 1 раз. Затем, созданное определение, использовалось для вычисления значений в каждой точке графика. Второй подход значительно ускоряет вычисления. Такое поведение использовалось нами при построении графика функции $\int_0^x \int_0^z \text{Cos}[zt] dt dz$. Снова выполним код

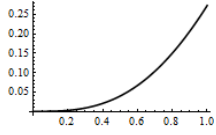
```
i = 0; Plot[g[x], {x, -3, 5}]; i
```

451

Без использования функции Evaluate определение функции $g[x]$ снова создавалось 451 раза.

Для определения времени вычисления в секундах какой – либо команды можно использовать функцию Timing.

```
Timing[Plot[Evaluate[ $\int_0^x \text{Sin}[t]^2 dt$ , {x, 0, 1}]]]
```

```
{0.046800,  }
```

Без использования функции Evaluate время работы команды значительно больше (в различных версиях Mathematica это время будет колебаться от сотых долей секунды до десятков секунд; кроме того время, конечно, зависит от быстродействия компьютера)

```
Timing[Plot[ $\int_0^x \text{Sin}[t]^2 dt$ , {x, 0, 1}];]
```

```
{12.636081, Null}
```

Еще с одним случаем, когда при построении графиков требуется использование функции Evaluate, мы столкнемся при решение дифференциальных уравнений.

2.1.2 Параметрические графики.

График кривых, задаваемых параметрически $X=X(t)$, $Y=Y(t)$, строится с помощью функции ParametricPlot.

```
ParametricPlot[{t^2, t}, {t, -2, 2}]
```

Первый аргумент - список двух выражений $\{X(t), Y(t)\}$, которые определяют координаты x и y как функцию параметра t . Второй аргумент определяет диапазон изменения параметра t . Здесь t изменяется от -2 до 2.

Если вы хотите нарисовать более чем одну параметрическую функцию, используйте список списков

```
ParametricPlot[{{t^3, t^2}, {t^2, t^3}}, {t, -2, 2}]
```

Большинство опций функции Plot могут использоваться также и в ParametricPlot. Например, опция AspectRatio->Automatic используется для одинакового масштабирования по осям. Опция PlotStyle

может быть использована для рисования кривых с разной штриховкой, толщиной и цветом

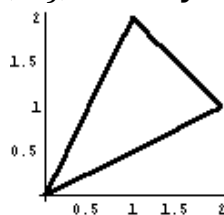
```
ParametricPlot[{ $t^3, t^2$ }, { $t^2, t^3$ }, { $t, -2, 2$ },  
PlotStyle → {}, Dashing[{0.02}], AspectRatio → Automatic]
```

Эта функция позволяет нарисовать практически любую кривую, если вы можете составить ее параметрическое уравнение. Вот несколько примеров.

```
 $x[t_]:= \text{Abs}[t] - 3\text{Abs}[t - 1]/2 + \text{Abs}[t - 3]/2;$ 
```

```
 $y[t_]:= \text{Abs}[t]/2 - 3\text{Abs}[t - 2]/2 + \text{Abs}[t - 3];$ 
```

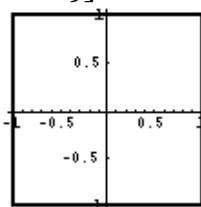
```
ParametricPlot{ $x[t], y[t]$ }, { $t, 0, 3$ }, PlotStyle → Thickness[0.01]
```



```
 $xs[t_]:= 1 - \text{Abs}[t]/2 + \text{Abs}[t - 2]/2 + \text{Abs}[t - 4]/2 - \text{Abs}[t - 6]/2;$ 
```

```
 $ys[t_]:= 1 - \text{Abs}[t - 2]/2 + \text{Abs}[t - 4]/2 + \text{Abs}[t - 6]/2 - \text{Abs}[t - 8]/2;$ 
```

```
ParametricPlot{ $xs[t], ys[t]$ }, { $t, 0, 8$ }
```

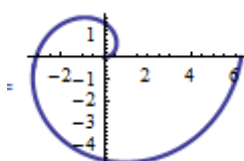


На сайте geometry@karazin.ua в нашем курсе «Аналитические методы геометрического моделирования» вы можете познакомиться с методикой составления параметрических и неявных уравнений составных кривых и поверхностей (уравнение кусков которых заданы). Эта методика научит вас составлять «аналитические» уравнения квадрата (см. выше), куба, пирамиды и практически любых непрерывных кривых и поверхностей.

2.1.3 График в полярных координатах

Для построения графиков в полярных координатах используется функция `PolarPlot` (в *Mathematica 5* она находится в пакете расширения, загрузку которого надо выполнить командой `Needs["Graphics`Graphics`"]`). Начиная с 6 – й версии эта функция находится в ядре. Функция очень похожа на функцию `Plot`

```
PolarPlot[ $t, \{t, 0, 2\text{Pi}\}$ , PlotStyle → Thickness[0.02]]
```



Первый аргумент – радиальная функция. Второй аргумент – интервал изменения угловой переменной t . `PolarPlot` автоматически выравнивает

вертикальный и горизонтальный масштабы. В следующих примерах окружность похожа сама на себя, а не на овал

PolarPlot[1, {t, 0, 2Pi}] (* центрированная окружность *)

или сдвинутая окружность

PolarPlot[2Cos[t], {t, 0, 2π}]

2.1.4 Графики неявных функций.

В *Mathematica* 5 для построения графиков кривых, определяемых неявными уравнениями, необходимо выполнить загрузку графического пакета `ImplicitPlot` командой `Needs["Graphics`ImplicitPlot`"]` или `<<Graphics`ImplicitPlot``. После этого используется одноименная функция `ImplicitPlot`. Например

ImplicitPlot[(x² + y²)² == x² - y², {x, -1, 1}, {y, -0.5, 0.5}]

Для построения графиков кривых, заданных неявным уравнением, в *Mathematica* 6 и более поздних версиях используется функция `ContourPlot`.

Эта функция очень похожа на `Plot`, только вместо явного выражения, содержащего одну переменную, вы задаете уравнение, содержащее две переменных

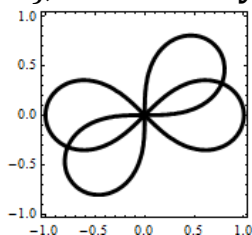
ContourPlot[x² + y² == 1, {x, -1.5, 1.5}, {y, -1.5, 1.5}]

Первый аргумент – уравнение. В *Mathematica* уравнения обозначаются символом двойного равенства (`==`). Второй и третий аргументы задают диапазоны изменения горизонтальной и вертикальной переменных.

**ContourPlot[(x² + y²)² == x² - y², {x, -1, 1}, {y, -0.5, 0.5},
AspectRatio → Automatic]**

Можно рисовать несколько кривых на одном графике, задавая список уравнений в качестве первого параметра.

**ContourPlot[{(x² + y²)² == x² - y², (x² + y²)² == 2xy},
{x, -1, 1}, {y, -1, 1}, ContourStyle → Thickness[0.02]]**

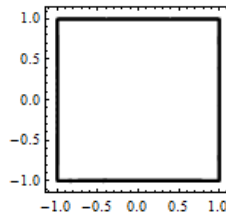


Обратите также внимание на то, что стиль кривых здесь задается опцией `ContourStyle`, а не `PlotStyle` как у других графических функций.

В следующем примере мы строим квадрат по его неявному уравнению.

W[x_, y_] = 2 - x² - y² - √((1 - x²)² + (1 - y²)²)

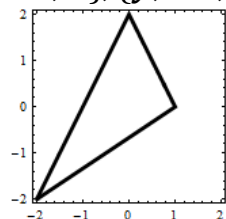
ContourPlot[W[x, y] == 0, {x, -1.1, 1.1}, {y, -1.1, 1.1}]



Функция `ContourPlot` вычисляет значения выражения на некоторой регулярной сетке в плоскости XOY и рисует отрезки линии, когда разность между двумя узлами меняет знак. Опция `PlotPoints` $\rightarrow n$ задает число точек разбиения интервалов изменения координат, определяя, тем самым, густоту сетки и, следовательно, гладкость кривой. Допустима форма опции `PlotPoints` $\rightarrow \{m, n\}$, определяющей различное количество точек разбиения по осям.

$$Wt[x, y] = 2 - x + y - Abs[x] - Abs[x - 2y - Abs[x]]$$

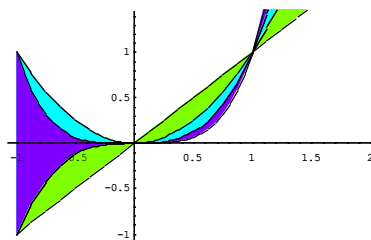
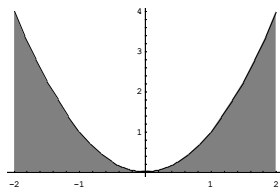
`ContourPlot[Wt[x, y] == 0, {x, -2, 2}, {y, -2, 2}, PlotPoints \rightarrow {50, 50}]`



2.1.5 Области между кривыми.

В *Mathematica 5* для построения графиков с заполненными областями между кривыми необходимо выполнить загрузку графического пакета `FilledPlot` командой `Needs["Graphics`FilledPlot`"]` и затем использовать одноименную функцию `FilledPlot`. Например

`FilledPlot[x2, {x, -2, 2}]` (* рисунок слева *)



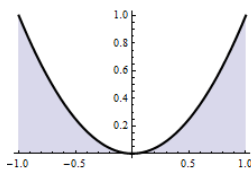
или

`FilledPlot[{x, x2, x3, x4}, {x, -1, 2}]` (* рисунок справа *)

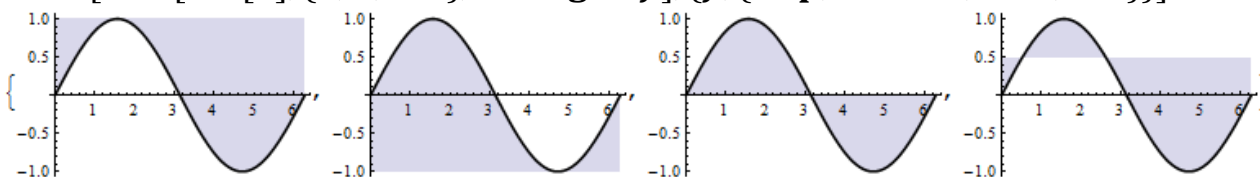
В *Mathematica 6* и более поздних версиях для тех же целей используется опция `Filling` графических функций `Plot`, `ListPlot` и других.

С использованием этой опции можно задать уровни, которые определяют заполняемую область: между кривой и осью (`Axis`), между кривой и нижней границей графика (`Bottom`), между кривой и горизонтальной прямой на заданном уровне (число) и т.д.

`Plot[x2, {x, -1, 1}, Filling \rightarrow Axis]`

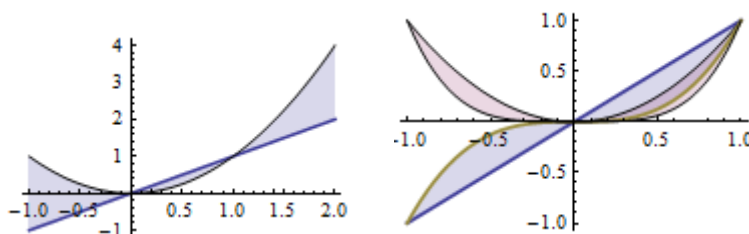


Table[Plot[Sin[x], {x, 0, 2Pi}, Filling → f], {f, {Top, Bottom, Axis, 0.5}}]



Эту опцию можно использовать для указания того, между какими кривыми графика следует заполнять область. Например, заполнение области между кривыми 1 и 2 выполняется командой

Plot[{x, x²}, {x, -1, 2}, Filling → {1 → {2}}] (* следующий рисунок слева *)

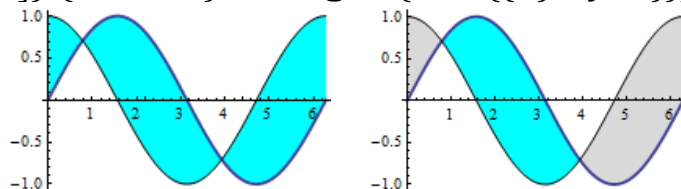


Вот пример заполнения областей между несколькими кривыми (предыдущий рисунок справа)

Plot[{x, x², x³, x⁴}, {x, -1, 1}, Filling → {1 → {3}, 2 → {4}}]

Цвет заполнения определяется, например, так (следующий рисунок слева)

Plot[{Sin[x], Cos[x]}, {x, 0, 2Pi}, Filling → {1 → {{2}, Cyan}}]



Можно определить два цвета заполнения. В следующем примере, если первая кривая находится ниже второй, то используется первый цвет, иначе – второй.

Plot[{Sin[x], Cos[x]}, {x, 0, 2Pi}, Filling → {1 → {{2}, {LightGray, Cyan}}}]

(предыдущий рисунок справа).

Опция `FillingStyle` используется для задания стиля заполнения областей. Предыдущие два графика можно построить с использованием этой опции так

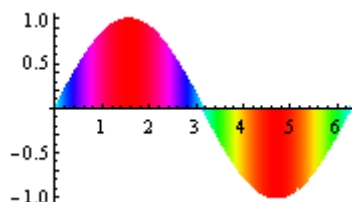
Plot[{Sin[x], Cos[x]}, {x, 0, 2Pi}, Filling → {1 → {2}}, FillingStyle → Cyan]

Plot[{Sin[x], Cos[x]}, {x, 0, 2Pi}, Filling → {1 → {2}},

FillingStyle → {LightGray, Cyan}]

Но опцию `FillingStyle` можно использовать и для определения более сложного стиля заполнения

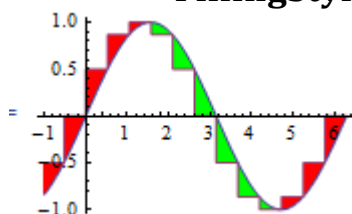
**Plot[Sin[x], {x, 0, 2Pi}, ColorFunction → Function[{x, y}, Hue[y]],
Filling → Axis, FillingStyle → Automatic]**



Вот пример, в котором заполнение используется для обозначения области отличия функции от ее кусочно – постоянной интерполяции.

```

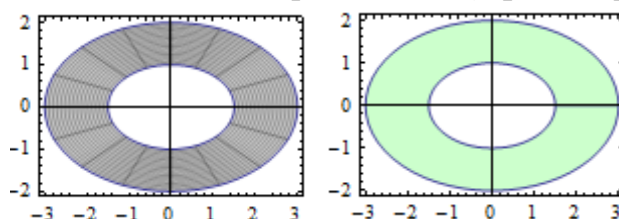
si = Interpolation[Table[{x, Sin[x]}, {x, -π/3, 2π, π/6}],
InterpolationOrder -> 0]
Plot[{Sin[x], si[x]}, {x, -π/3, 2π}, Filling -> {1 -> {2}},
FillingStyle -> {Red, Green}]
```



Функция ParametricPlot не имеет опции Filling, но она имеет возможность рисовать области, заданные параметрическими уравнениями $x = x(u, v)$, $y = y(u, v)$

```

ParametricPlot[{{3vCos[u], 2vSin[u]}}, {u, 0, 2Pi}, {v, 0.5, 1}] (* рис. слева *)
ParametricPlot[{{3vCos[u], 2vSin[u]}}, {u, 0, 2Pi}, {v, 0.5, 1}, Mesh -> None,
PlotStyle -> Green] (* рис. справа *)
```

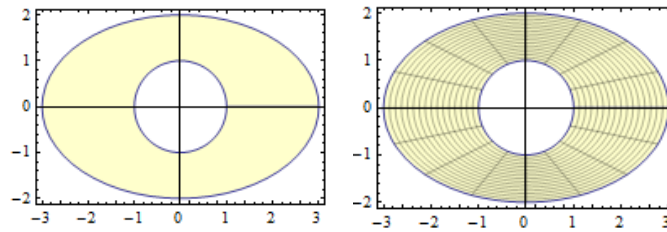


Для заполнения области между кривыми $\mathbf{r}_1(u) = (x_1(u), y_1(u))$ ($u_0 \leq u \leq u_1$) и $\mathbf{r}_2(u) = (x_2(u), y_2(u))$ ($u_0 \leq u \leq u_1$), заданными параметрически, нужно записать параметрическое уравнение области между этими кривыми. Оно может иметь вид $\mathbf{r}(u, v) = v\mathbf{r}_1(u) + (1-v)\mathbf{r}_2(u)$, $0 \leq v \leq 1$, ($u_0 \leq u \leq u_1$) или в покоординатной форме $x(u, v) = vx_1(u) + (1-v)x_2(u)$, $y(u, v) = vy_1(u) + (1-v)y_2(u)$.

Заполним, например, область между единичной окружностью $\mathbf{r}_1(u) = (\cos(u), \sin(u))$ ($0 \leq u \leq 2\pi$) и эллипсом $\mathbf{r}_2(u) = (3\cos(u), 2\sin(u))$ ($0 \leq u \leq 2\pi$). Имеем

```

r1[u_] := {Cos[u], Sin[u]}
r2[u_] := {3Cos[u], 2Sin[u]}
ParametricPlot[{vr1[u] + (1 - v)r2[u]}, {u, 0, 2π}, {v, 0, 1},
Mesh -> None, PlotStyle -> Yellow]
```

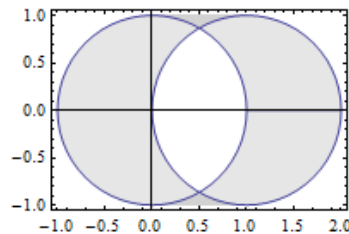


На правом графике мы построили ту же область, но оставили видимой координатную сетку (не использовали опцию `Mesh`). Из него видно, что заполняются точки области по прямым, соединяющим соответствующие (имеющие одинаковое значение параметра u) точки граничных кривых. Часто это не то, что вы имели в виду. Например

$$\mathbf{r1}[u_]:= \{\mathbf{Cos}[u], \mathbf{Sin}[u]\}$$

$$\mathbf{r2}[u_]:= \{1 + \mathbf{Cos}[u], \mathbf{Sin}[u]\}$$

`ParametricPlot` $\{v \mathbf{r2}[u] + (1 - v)\mathbf{r1}[u]\}, \{u, 0, 2\pi\}, \{v, 0, 1\}, \mathbf{Mesh} \rightarrow \mathbf{None},$
`PlotStyle` $\rightarrow \mathbf{Gray}$



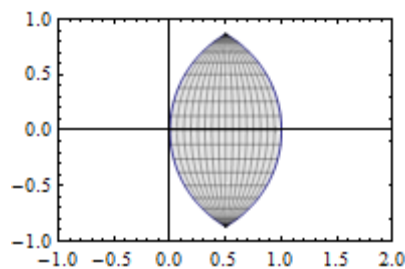
Чтобы заполнить луночку между окружностями надо записать параметрические уравнения дуг окружностей между точками их пересечения так, чтобы диапазоны изменения параметра u были одинаковыми и соответствовали желаемым направлениям движения вдоль дуг.

$$\mathbf{r1}[u_]:= \{\mathbf{Cos}[u], \mathbf{Sin}[u]\}$$

$$\mathbf{r2}[u_]:= \{1 - \mathbf{Cos}[u], \mathbf{Sin}[u]\}$$

`ParametricPlot` $\{v \mathbf{r2}[u] + (1 - v)\mathbf{r1}[u]\}, \{u, -\frac{\pi}{3}, \frac{\pi}{3}\}, \{v, 0, 1\},$

`PlotStyle` $\rightarrow \mathbf{Gray}, \mathbf{PlotRange} \rightarrow \{\{-1, 2\}, \{-1, 1\}\}$

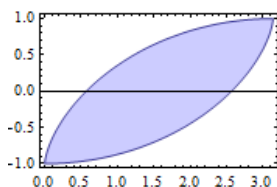


Вот еще пример заполнения области между кривыми, заданными параметрически

$$\mathbf{r1}[u_]:= \{u + \mathbf{Sin}[u], -\mathbf{Cos}[u]\}$$

$$\mathbf{r2}[u_]:= \{u - \mathbf{Sin}[u], -\mathbf{Cos}[u]\}$$

`ParametricPlot` $\{v \mathbf{r1}[u] + (1 - v) \mathbf{r2}[u]\}, \{u, 0, \pi\}, \{v, 0, 1\},$
`PlotStyle` $\rightarrow \mathbf{Blue}, \mathbf{Mesh} \rightarrow \mathbf{None}$

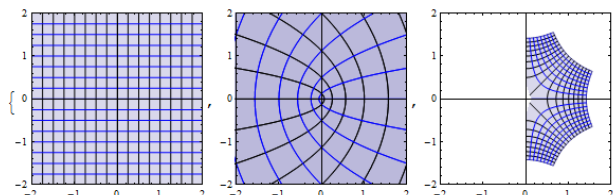


Пусть дана аналитическая функция $w = f(z)$ комплексного переменного $z = x + iy$ в плоскости Z и $w = u + iv$ – комплексная переменная в плоскости W . Одним из способов ее графического представления является построение сеток кривых в плоскостях Z и W , соответствующих друг другу при таком отображении. Умение функции `ParametricPlot` рисовать области, заданные параметрическими уравнениями $x = x(u, v)$, $y = y(u, v)$, позволяет решать эту задачу.

В следующем примере мы строим декартову сеть в плоскости комплексного переменного Z и ее образы в плоскостях W , полученные при отображении $w = z^2$ и $w = \sqrt{z}$.

Table[

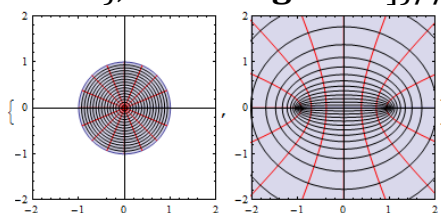
```
ParametricPlot[{Re[(u + i v)^n], Im[(u + i v)^n]}, {u, -2, 2}, {v, -2, 2},  
MeshStyle -> {Black, Blue}, PlotRange -> 2],  
{n, {1, 2, 1/2}}
```



Слева показана декартова сеть комплексной плоскости Z , в середине – ее образ в плоскости W при отображении $w = z^2$, справа – образ при отображении $w = \sqrt{z}$.

Рассмотрим функцию Жуковского $w = f(z) = \frac{1}{2} \left(z + \frac{1}{z} \right)$. Известно, что она реализует однолистное отображение, например, в области единичного круга $|z| < 1$. Построим картину отображения полярной системы координат плоскости Z в плоскость W .

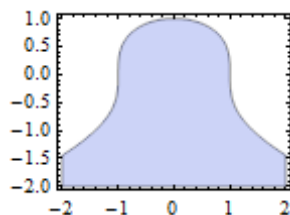
```
f[u_, r_] = 1/2 (r Cos[u] + i r Sin[u] + 1 / (r Cos[u] + i r Sin[u]));  
{ParametricPlot[{Re[r Cos[u] + i r Sin[u]], Im[r Cos[u] + i r Sin[u]]},  
{u, 0, 2 pi}, {r, 0, 1}, MeshStyle -> {Red, Black}, PlotRange -> 2],  
ParametricPlot[{Re[f[u, r]], Im[f[u, r]]}, {u, 0, 2 pi}, {r, 0, 1},  
MeshStyle -> {Red, Black}, PlotRange -> 2]}/Quiet
```



Здесь на левом рисунке показана сеть линий полярной системы координат в плоскости Z , на правом – образ кривых этой сети в плоскости W . Видно, что окружности переходят в эллипсы (черные кривые), прямые, проходящие через начало координат, переходят в гиперболы (красные кривые).

Для построения графиков заполненных областей предназначена функцию `RegionPlot`. Первым ее аргументом является *выражение*, зависящее, например, от переменных x и y , и которое может принимать значения `True` или `False`. Вторым и третьим аргументами являются диапазоны изменения переменных x и y , задаваемые в виде списков. Функция заполняет ту часть плоскости (x,y) , в которой *выражение* принимает значение `True`.

RegionPlot[$x^2 + y^3 < 1$, {x, -2, 2}, {y, -2, 1}, AspectRatio → Automatic]



Выражения можно объединять с использованием логических операций `And(&&)`, `Or(||)`, `Xor`, `Implies`, `Nand`, `Nor` смысл которых ясен из их названия.

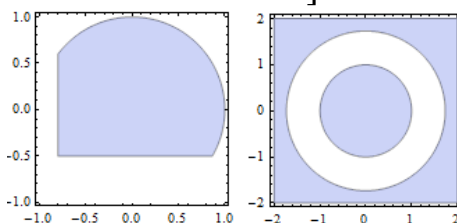
RegionPlot[$x^2 + y^2 < 1 \ \&\& \ y > -\frac{1}{2} \ \&\& \ x > -0.8$,

{x, -1, 1}, {y, -1, 1}, AspectRatio → Automatic] (* след. рис. слева *)

RegionPlot[$x^2 + y^2 < 1 \ || \ x^2 + y^2 > 3$, {x, -2, 2}, {y, -2, 2},

AspectRatio → Automatic]

(* след. рис. справа *)



В следующем примере мы поясняем смысл логических операций между областями. Фактически здесь мы строим диаграммы Эйлера для операций над множествами.

$$a = x^2 + y^2 < 1;$$

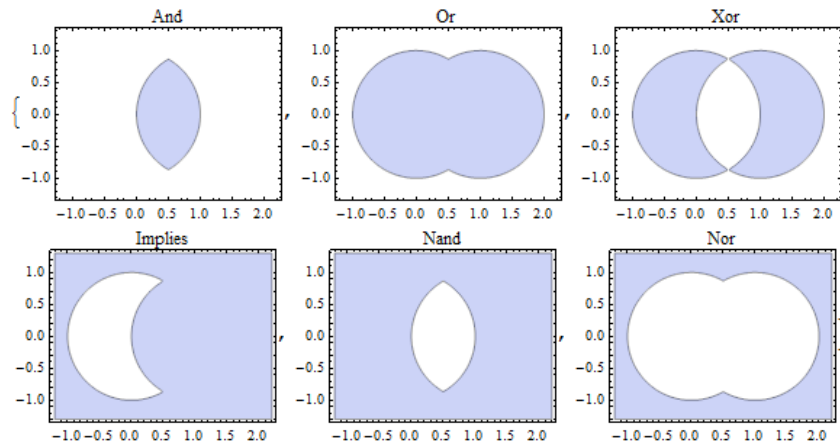
$$b = (x - 1)^2 + y^2 < 1;$$

Table[

RegionPlot[f[a, b], {x, -1.2, 2.2}, {y, -1.3, 1.3},

PlotLabel → f, AspectRatio → Automatic],

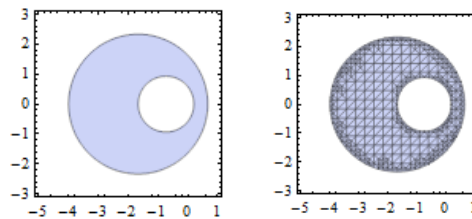
{f, {And, Or, Xor, Implies, Nand, Nor}}}]



Можно также использовать комплексные выражения для построения областей

RegionPlot $[1 < Abs[\frac{(x + iy) - 3}{2(x + iy) + 1}] < 2, \{x, -5, 1\}, \{y, -3, 3\},$

AspectRatio \rightarrow **Automatic**]



Правый рисунок построен предыдущей командой с добавлением опции `Mesh` \rightarrow `All`. Обратите внимание, что в местах быстрого изменения функции, система автоматически сгущает сетку.

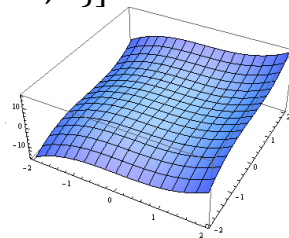
2.2 Трехмерные графики.

Система *Mathematica* снабжена большим количеством трехмерных графических функций. Основные из них описаны в данном разделе.

2.2.1 График функций заданных явно.

Для построения поверхности функции $z = f(x, y)$ предназначена функция `Plot3D`.

Plot3D $[x^3 + y^3, \{x, -2, 2\}, \{y, -2, 2\}]$



Первый аргумент – выражение $f(x, y)$, график которого должен быть построен. Второй и третий аргументы определяют границы изменения переменных x и y , задаваемые в виде списков. Обычно функция вычисляется на сетке 15 на 15 и каждый кусок закрашивается согласно светоотражающей модели.

Mathematica размещает график в ящик, ориентация которого определяется положением точки, из которой он рассматривается. По умолчанию ориентация имеет следующие особенности:

- Шкала x почти горизонтальная, находится впереди слева, x увеличивается направо
- Шкала y проходит спереди назад вправо, y увеличивается назад.
- Шкала z - вертикальная слева, значения z возрастают вверх.

С помощью опций функции `Plot3D` ориентацию можно изменить. Для улучшения графика можно использовать опцию `Mesh→30` (в *Mathematica 5* `PlotPoints→30`), которая увеличивает количество точек до 30 (или любого другого числа).

`Plot3D[Sin[x Sin[x y]], {x, 0, 4}, {y, 0, 3}, Mesh → 30]`

Размер сетки вырос с 15 на 15 по умолчанию до 30 на 30. Теперь посмотрим на график той же функции с другой точки зрения.

`Plot3D[Sin[x Sin[x y]], {x, 0, 4}, {y, 0, 3}, Mesh → 30, ViewPoint → {-2.4, 1., 2.}]`

Здесь мы использовали опцию `ViewPoint`. Она определяет точку, из которой вы наблюдаете «куб», охватывающий поверхность.

По умолчанию *Mathematica* раскрашивает графики согласно простой модели освещенности. Цвет каждого участка зависит в основном от ориентации каждого сегмента поверхности относительно источников света.

Такой способ раскраски иногда приводит к бедным цветным графикам. Если мы добавим опцию `Lighting→None`, то *Mathematica* отключит модельное освещение (в *Mathematica 5* `Lighting→False` и система будет закрашивать график, увеличивая яркость в соответствии со значением z).

`Plot3D[x3 + y3, {x, -2, 2}, {y, -2, 2}, Lighting → None]`

Управлять цветовой гаммой функции `Plot3D` можно с помощью опции `ColorFunction`. Она задается в виде функции и должна иметь определенное количество аргументов, которое зависит от графической функции, и должна возвращать строго определенный объект: `RGBColor`, `Hue` или `GrayLevel`. Для функции `Plot3D` функция `ColorFunction` будет принимать 3 аргумента – координаты x , y , используемые в выражении, и z – значение выражения. На поведение `ColorFunction` влияет опция `ColorFunctionScaling`, которая может принимать значения `True` или `False`. Если `ColorFunctionScaling→True` (значение по умолчанию), то все аргументы функции `ColorFunction` автоматически приводятся к диапазону $[0, 1]$. Если `ColorFunctionScaling→False`, то масштабирование аргументов не выполняется. Поскольку аргументы функций `RGBColor` и `Hue` должны изменяться в диапазоне от 0 до 1, то это в этом случае масштабирование аргументов становится вашей задачей.

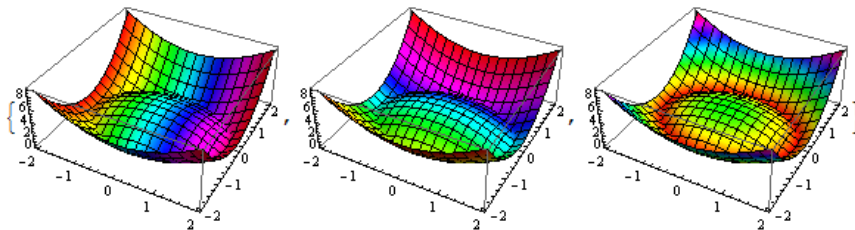
**`{Plot3D[Abs[3 - (x2 + 2y2)], {x, -2, 2}, {y, -2, 2},
ColorFunction → Function[{x, y, z}, Hue[x]]],`**

`Plot3D[Abs[3 - (x2 + 2y2)], {x, -2, 2}, {y, -2, 2},`

```

ColorFunction → Function[{x, y, z}, Hue[y]],
Plot3D[Abs[3 - (x2 + 2y2)], {x, -2, 2}, {y, -2, 2},
ColorFunction → Function[{x, y, z}, Hue[z]]]

```

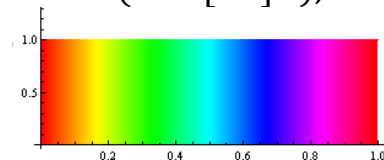


Посмотрите, как на графиках меняются цвета – в соответствии с изменением координат x , y или z . Здесь Hue – это графическая директива. В простейшей форме Hue[h] определяет, что используемый объект должен быть раскрашен тоном, задаваемым аргументом h ($0 \leq h \leq 1$). На следующем рисунке вы можете увидеть значения цветового тона, который отсчитывается по горизонтальной оси, и сам цвет.

```

Plot[1, {x, 0, 1}, ColorFunction → (Hue[#1]&), Filling → Axis]

```



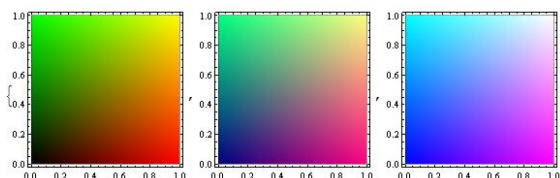
В приведенном здесь примере следует обратить внимание на то, что функция, передаваемая опции ColorFunction в функции Plot, принимает только два аргумента – координату x и значение выражения y .

Функция RGBColor[r, g, b] принимает три числовых аргумента в диапазоне от 0 до 1, каждый из которых определяет относительное содержание красной (r), зеленой (g) и синей (b) составляющих цвета. Она создает объект, который должна возвращать любая функция, представляющая значение опции ColorFunction. Вот пример цветовых палитр, создаваемых функцией RGBColor[x, y, b] при значении синей составляющей $b = \{0, 1/2, 1\}$.

```

Table[RegionPlot[True, {x, 0, 1}, {y, 0, 1},
ColorFunction → Function[{x, y}, RGBColor[x, y, b]],
{b, {0, 1/2, 1}}]

```



Обратите внимание на то, что функция, передаваемая опции ColorFunction в функции RegionPlot, принимает два аргумента – координаты x и y точек внутри области.

Напомним, что в Mathematica функции можно создавать по-разному. Например, две следующие функции эквивалентны

```

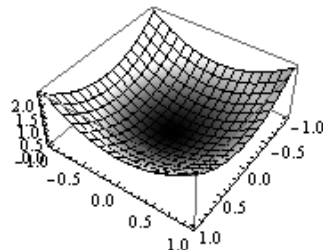
RGBColor[#1, #2, #3]&
Function[{x, y, z}, RGBColor[x, y, z]]

```

#1 в первой записи (первый аргумент функции) соответствует x во второй записи. Аналогично, #2 и #3 в первой записи (второй и третий аргументы функции) соответствуют y и z во второй записи. Знак & (амперсанд) в первой записи является постфиксной формой функции `Function`. В примерах вы можете встретить обе эти формы записи функций.

Директива `GrayLevel[h]` определяет, что объект должен быть раскрашен серым цветом с интенсивностью h ($0 \leq h \leq 1$). При этом 0 соответствует черному цвету, 1 – белому, промежуточные значения соответствуют серому цвету различной яркости.

`Plot3D[x2 + y2, {x, -1, 1}, {y, -1, 1}, ColorFunction -> (GrayLevel[2#3]&)]`



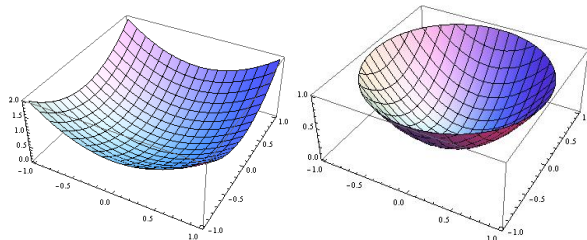
Подробнее о цветовых гаммах, используемых в системе, вы можете узнать, получив справку по этим функциям.

У многих графических функций есть еще одна важная опция `RegionFunction`. Она позволяет рисовать график поверхности $z = f(x, y)$ не во всем прямоугольнике, определяемом передаваемыми диапазонами $\{x, x_{\min}, x_{\max}\}$, $\{y, y_{\min}, y_{\max}\}$, а в некоторой его подобласти. Подобласть определяется булевой функцией (для `Plot3D` – это функция трех переменных x, y, z). График рисуется только там, где эта булева функция принимает значение `True`. Сравните, например, два следующих графика, построенные функцией `Plot3D` без использования опции `RegionFunction` и с ее использованием.

`Plot3D[x2 + y2, {x, -1, 1}, {y, -1, 1}]`

`Plot3D[x2 + y2, {x, -1, 1}, {y, -1, 1},`

`RegionFunction -> Function[{x, y, z}, x2 + y2 < 1]]`



Интересной является опция `MeshFunctions`. Она позволяет рисовать любую криволинейную сеть на поверхности. Для функции `Plot3D` эта опция может принимать список двух функций (не выражений!), например так

`MeshFunctions -> {Функция1(x, y, z), Функция2(x, y, z)}`

Линии постоянного значения (линии уровня) `Функции1` определяют одно семейство линий сети, линии уровня `Функции2` определяют второе семейство.

По умолчанию положение линий сети определяется равномерным распределением значений этих функций.

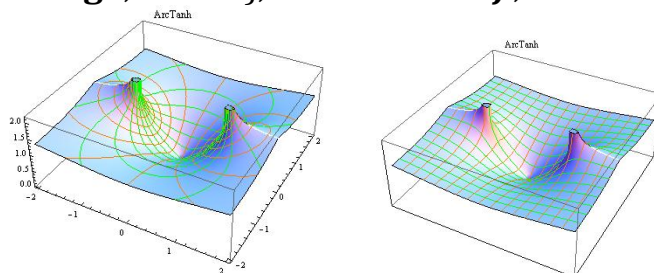
Если опция `MeshFunctions` не установлена, то используется значение по умолчанию `MeshFunctions` → `{#1&, #2&}`.

Вот пример одной и той же поверхности $z = |\text{Arctanh}(x + iy)|$ с криволинейной и декартовой сетями на поверхности. В первом случае линии сети образуются линиями уровня вещественной и мнимой частей z .

$f = \text{ArcTanh}$;

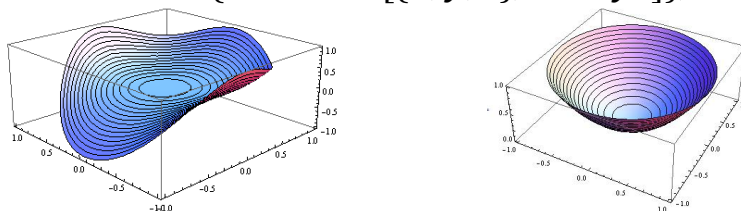
`Plot3D[Abs[f[x + Iy]], {x, -2, 2}, {y, -2, 2}, MeshFunctions → {Function[{x, y, z}, Re[f[x + Iy]]], Function[{x, y, z}, Im[f[x + Iy]]]}, MeshStyle → {Orange, Green}, PlotLabel → f]`

`Plot3D[Abs[f[x + Iy]], {x, -2, 2}, {y, -2, 2}, MeshFunctions → {#1&, #2&}, MeshStyle → {Orange, Green}, PlotLabel → f, Ticks → None]`



Конечно можно строить и одну сеть на поверхности (следующий рисунок слева)

`Plot3D[x3 + y4, {x, -1, 1}, {y, -1, 1}, RegionFunction → Function[{x, y, z}, x2 + y2 < 1], MeshFunctions → {Function[{x, y, z}, x2 + y2], Mesh → 15]`



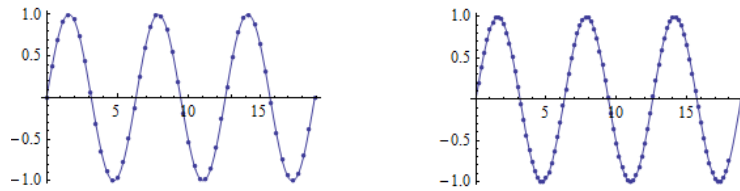
При задании функций для опций можно использовать z координату, чтобы не повторять выражение для функции

`Plot3D[x2 + y2, {x, -1, 1}, {y, -1, 1}, RegionFunction → Function[{x, y, z}, z < 1], MeshFunctions → {Function[{x, y, z}, z], Mesh → 15]`

(предыдущий рисунок справа).

Сетью для кривой является множество точек, расположенных на этой кривой. Например, для кривых, заданных явным уравнением, «сеть» можно построить так

`Plot[Sin[x], {x, 0, 6Pi}, Mesh → Full]` (* след. рисунок слева *)

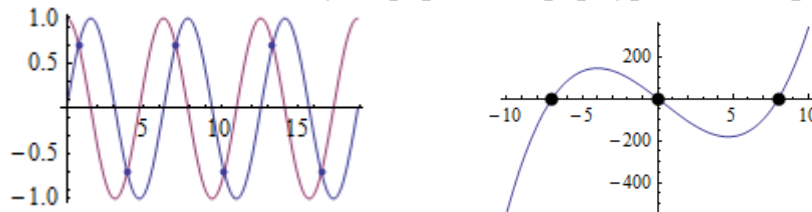


или можно задать список x – координат точек одномерной «сети»

Plot[Sin[x], {x, 0, 6Pi}, Mesh → {Table[0.2i, {i, 90}]}] (* пред. рис. справа *)

Одномерную «сеть» можно использовать для обозначения точек пересечения кривых. Для этого достаточно задать Mesh – функцию, являющуюся разностью тех функций, пересечение которых нас интересует, и единственное нулевое значение для опции Mesh. В этом случае точками сети будут точки, для которых Mesh – функция обращается в ноль, т.е. точки пересечения кривых.

Plot[{Sin[x], Cos[x]}, {x, 0, 6Pi}, Mesh → {{0}}, MeshFunctions → {Sin[#] – Cos[#]&}] (* след. рис. слева *)



Для обозначения точек пересечения кривой с осью X можно выполнить следующий код

$f[x_] = x^3 - x^2 - 56x;$

Plot[f[x], {x, -10, 10}, MeshFunctions → {(#2)&}, Mesh → {{0}},

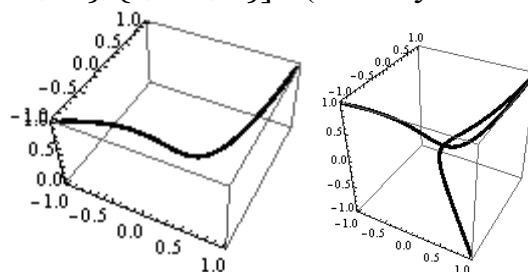
MeshStyle → PointSize[Large]] (* пред. рис. справа *)

Здесь #2 представляет значение функции.

2.2.2 График параметрических функций.

Существует две формы функции ParametricPlot3D для построения трехмерных кривых и трехмерных поверхностей. Для построения кривой необходимо выполнить команду

ParametricPlot3D[{t, t³, t²}, {t, -1, 1}] (* следующий рисунок слева *)

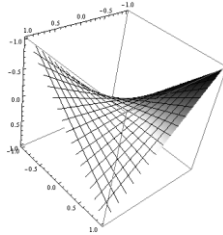


Первый аргумент – это список трех выражений, которые определяют x , y и z координаты, как функции параметра t . В нашем случае $x = t$, $y = t^3$, $z = t^2$.

Второй аргумент/список определяет область изменения параметра (здесь от -1 до 1). Если вы хотите нарисовать более чем одну кривую на одном графике, вы должны задать список списков выражений, например

ParametricPlot3D[[{ t, t^3, t^2 }, { t^2, t, t^3 }}, { $t, -1, 1$ }] (* пред. рисунок справа *)
 Уравнение поверхности задается списком трех выражений с использованием двух параметров.

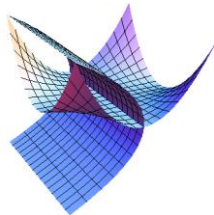
ParametricPlot3D[{ uv, u, v }, { $u, -1, 1$ }, { $v, -1, 1$ }]



Первый аргумент – это список трех выражений, которые определяют x, y и z координаты, как функции двух параметров. В этом примере $x = uv, y = u, z = v$. Второй и третий аргументы – списки, которые определяют диапазоны изменения параметров u и v .

Большинство опций команды `Plot3D` можно использовать и с `ParametricPlot3D`. Например, следующие опции можно использовать для удаления границы ящика и осей (это полезно, когда поверхности пересекаются)

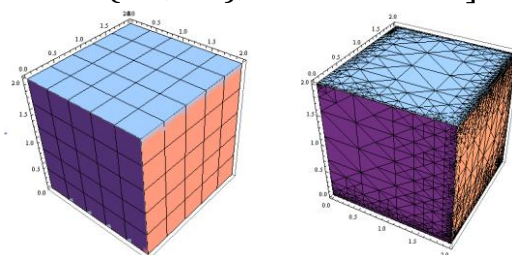
ParametricPlot3D[[{ $u, -v^2, v$ }, { u, v, u^2v^2 }}, { $u, -1, 1$ }, { $v, -1, 1$ },
Boxed \rightarrow **False**, **Axes** \rightarrow **None**]



Для параметрических уравнений поверхностей с ребрами важно правильно использовать опцию `PlotPoints`. Она определяет начальное количество точек разбиения для диапазонов изменения параметров u и v . Вот пример параметрических уравнений куба и его поверхности, построенной по таким уравнениям (следующий рисунок слева)

$x[u, v] := \text{Abs}[u] - \text{Abs}[u - 1] - \text{Abs}[u - 2] + \text{Abs}[u - 3];$
 $y[u, v] := 1 + \text{Abs}[v] - \text{Abs}[v - 1];$
 $z[u, v] := 1 - \text{Abs}[\text{Abs}[u - 1] - \text{Abs}[u - 2] - \text{Abs}[u - 3] + \text{Abs}[u - 4] +$
 $\text{Abs}[v - 1] - \text{Abs}[v - 2] + \text{Abs}[v] - \text{Abs}[v + 1]]/2 +$
 $\text{Abs}[2 + \text{Abs}[u - 1] - \text{Abs}[u - 2] - \text{Abs}[u - 3] +$
 $\text{Abs}[u - 4] + \text{Abs}[v - 1] - \text{Abs}[v - 2] + \text{Abs}[v] -$
 $\text{Abs}[v + 1]]/2;$

ParametricPlot3D[[$x[u, v], y[u, v], z[u, v]$], { $u, 0, 4$ }, { $v, -1, 2$ },
PlotPoints \rightarrow {**21, 16**}, **Mesh** \rightarrow **Full**]



Чтобы ребра на графике не сглаживались нужно, чтобы через них проходили координатные линии. В нашем примере правильными значениями для опции `PlotPoints` будет пара чисел, определяемая по правилу $\{\Delta u \cdot n + 1, \Delta v \cdot m + 1\}$, где Δu - длина диапазона изменения параметра u (в нашем примере 4), Δv - длина диапазона изменения параметра v (в нашем примере 3), m и n любые натуральные числа. В примере мы взяли $n = 5, m = 5$. Поэтому значение обсуждаемой опции получилось следующим `PlotPoints`→{21,16}. Попробуйте в последней команде предыдущего кода задать другие значения опции, например, `PlotPoints`→{22,17}. Используемая здесь другая опция `Mesh`→`Full` определяет, что диапазоны изменения параметров разбиваются координатными линиями равномерно.

Другой способ правильно рисовать ребра состоит в сгущении координатной сетки в областях быстрого изменения координатных функций, т.е. в окрестности ребер. Эта процедура в функции `ParametricPlot3D` выполняется автоматически и максимальное количество делений ячеек координатной сетки определяется опцией `MaxRecursion`.

`ParametricPlot3D`{`x`[u, v], `y`[u, v], `z`[u, v]}, { $u, 0, 4$ }, { $v, -1, 2$ },
`Mesh` → `All`, `MaxRecursion` → 4}

Приведенный здесь код строит поверхность куба, показанную на предыдущем рисунке справа. Как видим, сетка, по которой строилась поверхность, сгущается в окрестности ребер. Чтобы сетку отобразить использовалась опция `Mesh`→`All`.

Если вы забыли, какие опции имеются у функции, то можно выполнить команду `Options`. В качестве аргумента ей надо передать имя функции, опции которой вы хотите узнать.

`Options`[`ParametricPlot3D`]

В окне документа будет отображен список всех опций функции и их значения.

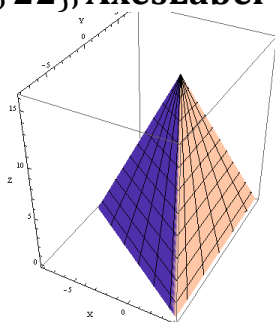
Вот еще пример параметрических уравнений пирамиды.

$$x[u_, v_] := (2Abs[v] - 3Abs[v - 1] + Abs[v - 3]) * (-4 + 3Abs[u] - 3Abs[u - 1] - 3Abs[u - 2] + 3Abs[u - 3])/2;$$

$$y[u_, v_] := -(2Abs[v] - 3Abs[v - 1] + Abs[v - 3]) * (3Abs[u - 1] - 3Abs[u - 2] + Abs[u - 3] - Abs[u]);$$

$$z[u_, v_] := 4(2 + Abs[v - 1] - Abs[v - 3]);$$

`ParametricPlot3D`{`x`[u, v], `y`[u, v], `z`[u, v]}, { $u, 0, 3$ }, { $v, 0, 3$ },
`PlotPoints` → {22, 22}, `AxesLabel` → {"X", "Y", "Z"}}



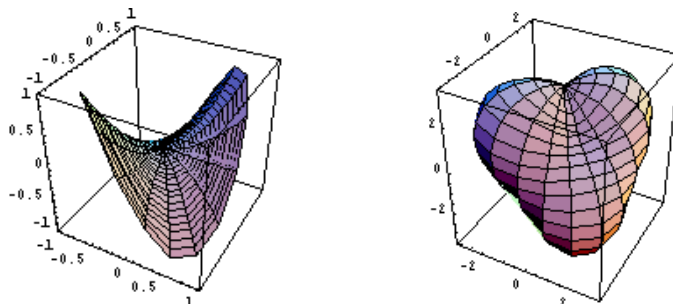
С методикой построения параметрических уравнений поверхностей с ребрами вы можете познакомиться по пособиям к нашему курсу «Аналитические методы геометрического моделирования», которые вы можете найти на сайте geometry@karazin.ua в разделе «Документы» автора.

2.2.3 Графики в сферических и цилиндрических координатах.

В *Mathematica 5* для построения графиков в сферических и цилиндрических координатах необходимо выполнить загрузку пакета `ParametricPlot3D` командой `Needs["Graphics`ParametricPlot3D`"]` и использовать функции `CylindricalPlot3D` и `SphericalPlot3D`. Например

`CylindricalPlot3D[r^2 Sin[2 t],{r,0,1},{t,0,2π}]` (* рисунок слева *)

Первый аргумент задает z как функцию радиуса и полярного угла. Вторым и третьим аргументами определяют области изменения радиального параметра r и углового параметра t .



`SphericalPlot3D[3+Sin[2 t]Sin[2 φ],{t,0, π},{φ,0, 2π}]` (* рисунок справа *)

Первый аргумент задает радиус как функцию углов t и φ . Вторым и третьим аргументами определяют области изменения угла t , образуемого радиус – вектором точки с осью Z , и угла φ в плоскости XY .

В *Mathematica version* ≥ 6 для построения поверхности, уравнение которой задано в цилиндрических координатах (r, φ, z) , используется функция `RevolutionPlot3D`. В этом случае формат ее вызова следующий

`RevolutionPlot3D[fz(r, φ), {r, rmin, rmax}, {φ, φmin, φmax}]`

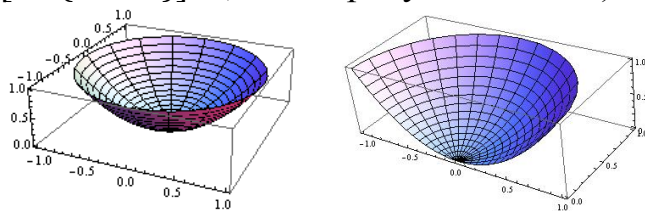
Вот пример построения предыдущей поверхности по ее уравнению в цилиндрической системе координат

`RevolutionPlot3D[r^2 Sin[2 φ],{r, 0, 1},{φ, 0, 2π}]` (* пред. рисунок слева *)

Первый аргумент представляет выражение вертикальной координаты $z = r^2 \sin(2\varphi)$ как функцию полярного радиуса r и угла φ . Вторым и третьим аргументами являются списками, определяющими интервалы изменения параметров r и φ . Полярный угол измеряется в радианах и отсчитывается против часовой стрелки от положительного направления оси X , если смотреть сверху.

Но функция `RevolutionPlot3D` имеет и другие возможности. В формате `RevolutionPlot3D[fz(x), {x, xmin, xmax}]` функция строит поверхность вращения кривой $z = f_z(x)$, заданной в плоскости XZ , вокруг оси Z .

RevolutionPlot3D[t², {t, 0, 1}] (* след. рисунок слева *)

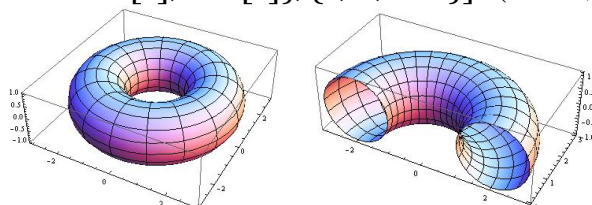


Можно построить часть поверхности вращения. Для этого нужно использовать диапазон изменения еще одной переменной, например θ , представляющую начальный и конечный углы поворота кривой относительно плоскости XZ.

RevolutionPlot3D[t², {t, 0, 1}, { θ , 0, π }] (* пред. рисунок справа *)

В формате **RevolutionPlot3D[{f_x, f_z}, {t, t_{min}, t_{max}}]** функция строит поверхность вращения кривой $x = f_x(t)$, $z = f_z(t)$, заданной параметрически в плоскости XZ, вокруг оси Z.

RevolutionPlot3D[{2 + Cos[t], Sin[t]}, {t, 0, 2Pi}] (* след. рисунок слева *)

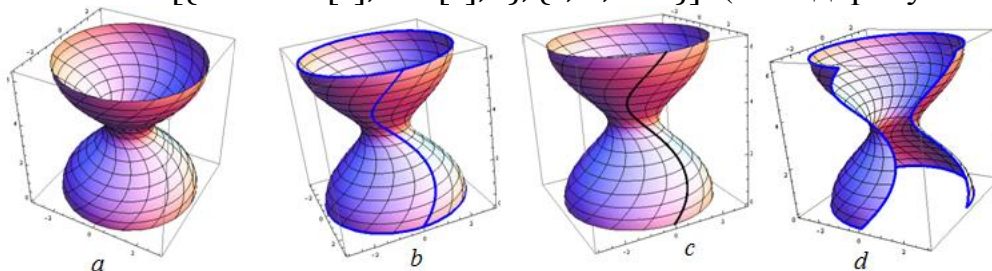


Аналогично предыдущему, можно построить часть поверхности вращения, задав диапазон угла поворота кривой относительно плоскости XZ.

RevolutionPlot3D[{2 + Cos[t], Sin[t]}, {t, 0, 2Pi}, { θ , 0, π }] (* пред. рис. справа *)

В формате **RevolutionPlot3D[{f_x, f_y, f_z}, {t, t_{min}, t_{max}}]** строится поверхность вращения пространственной кривой $x = f_x(t)$, $y = f_y(t)$, $z = f_z(t)$, заданной параметрически, вокруг оси Z.

RevolutionPlot3D[{2 + Cos[t], Sin[t], t}, {t, 0, 2Pi}] (* след. рисунок a *)



Чтобы выделить на поверхности ее края можно использовать опцию **BoundaryStyle**.

**RevolutionPlot3D[{2 + Cos[t], Sin[t], t}, {t, 0, 2Pi},
BoundaryStyle → Directive[Blue, Thick]]** (* пред. рисунок b *)

Краями поверхности считаются кривые, получаемые при граничных значениях независимых переменных. В случае поверхности вращения, кроме кривых, получаемых при вращении нижней и верхней точек (значения $t = 0$ и $t = 2\pi$ - верхняя и нижняя окружности), краями считаются начальное и конечное положение кривой вращения.

Опция `BoundaryStyle` имеется также у других графических функций, которые строят поверхности. Вместо значения `Thick` можно использовать графическую директиву `AbsoluteThickness[h]`, где `h` положительное число, указывающее толщину линии.

В последнем примере первым аргументом функции `RevolutionPlot3D` выступает параметрическое уравнение пространственной кривой. Ее можно изобразить отдельно функцией `ParametricPlot3D`. Чтобы объединить два графических объекта на одном графике (например, кривую в пространстве и поверхность) используется функция `Show`, которой через запятую передаются графические объекты. Следующий код строит кривую вращения и поверхность, и объединяет их на одном графике (предыдущий рисунок *c*).

```
Show[
  ParametricPlot3D[{2 + Cos[t], Sin[t], t}, {t, 0, 2Pi}],
  RevolutionPlot3D[{2 + Cos[t], Sin[t], t}, {t, 0, 2Pi}]]
```

Также как и выше, можно построить часть поверхности вращения. При этом мы показываем края поверхности (предыдущий рисунок *d*).

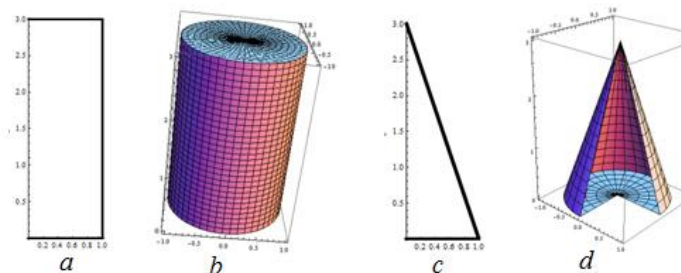
```
RevolutionPlot3D[{2 + Cos[t], Sin[t], t}, {t, 0, 2Pi}, {θ, 0, 3π/2},
  BoundaryStyle → Directive[Blue, AbsoluteThickness[4]]]
```

Заметим, что формат вызова `RevolutionPlot3D[fz(x), {x, xmin, xmax}` эквивалентен формату `RevolutionPlot3D[{t, 0, fz}, {t, tmin, tmax}`. А формат вызова `RevolutionPlot3D[{fx, fz}, {t, tmin, tmax}` эквивалентен формату `RevolutionPlot3D[{fx, 0, fz}, {t, tmin, tmax}`.

Если в плоскости `XZ` кривая будет ломаной, то получаемая поверхность вращения будет иметь ребра.

На следующем рисунке (*a*) показана «скоба», параметрическое уравнение $x(t), y(t)$ которой приведено в следующем коде. Вращением этой ломаной получается поверхность цилиндра с доньшками (рисунок *b*).

```
x[t_] = (5 - Abs[t - 1] - Abs[t - 4])/2;
y[t_] = (3 + Abs[t - 1] - Abs[t - 4])/2;
ParametricPlot[{x[t], y[t]}, {t, 0, 5}]
RevolutionPlot3D[{x[t], y[t]}, {t, 0, 5}, PlotPoints → 51, Mesh → Full]
```



На предыдущем рисунке (с) показана ломаная, параметрическое уравнение $x(t), y(t)$ которой приведено в следующем коде. Вращением этой ломаной получается поверхность конуса с доньшком (рисунок d).

```

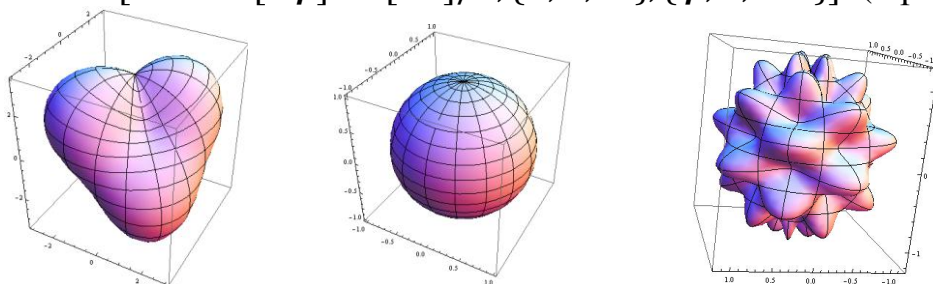
x[t_] = (t + 2)/3 - 2/3 Abs[t - 1];
y[t_] = (t - 1 + Abs[t - 1])/2;
ParametricPlot[{x[t], y[t]}, {t, 0, 4}]
RevolutionPlot3D[{x[t], y[t]}, {t, 0, 4}, {θ, 0, 3π/2},
  PlotPoints → 21, Mesh → Full]

```

Функция SphericalPlot3D также как и RevolutionPlot3D теперь находится в ядре пакета.

SphericalPlot3D[3 + Sin[2t]Sin[2φ], {t, 0, π}, {φ, 0, 2π}] (* след. рис. слева *)
 Первый аргумент представляет выражение длины радиус – вектора точки как функцию сферических углов t и φ . Второй и третий аргументы являются списками, которые задают интервал изменения угла t , образуемого радиус – вектором точки с осью Z, и интервал изменения угла φ , который образует проекция радиус – вектора на плоскость XY с фиксированным направлением в этой плоскости.

SphericalPlot3D[1, {t, 0, π}, {φ, 0, 2π}] (* следующий рисунок в центре *)
SphericalPlot3D[1 + Sin[4φ]Sin[8θ]/4, {θ, 0, Pi}, {φ, 0, 2Pi}] (* рис.справа *)



2.3 Другие графические возможности

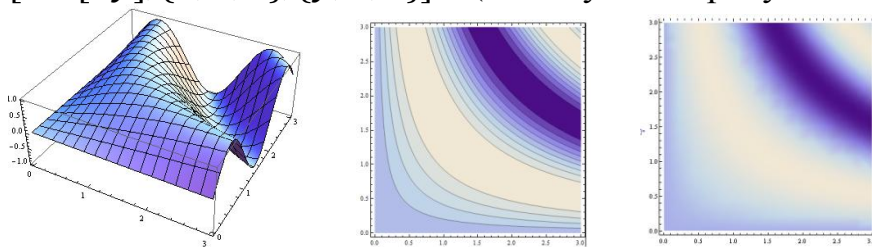
2.3.1 Линии уровня функции

Функции Plot3D, ContourPlot и DensityPlot можно использовать почти взаимозаменяемо. Приведем примеры графиков при использовании всех трех функций:

```

Plot3D[Sin[xy], {x, 0, 3}, {y, 0, 3}] (* следующий рисунок слева *)
ContourPlot[Sin[xy], {x, 0, 3}, {y, 0, 3}] (* следующий рисунок в центре *)
DensityPlot[Sin[xy], {x, 0, 3}, {y, 0, 3}] (* следующий рисунок справа *)

```

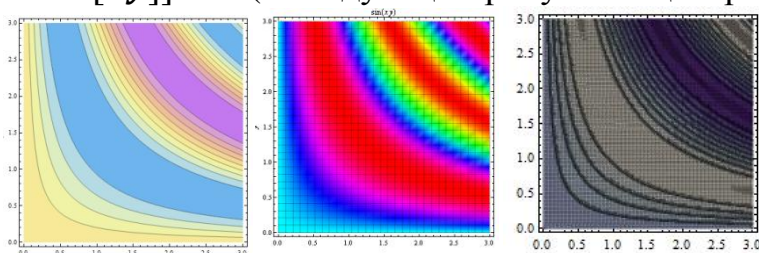


В каждом случае первый аргумент задается как выражение для z координаты от аргументов x, y . Второй аргумент определяет область изменения x . Третий аргумент определяет область изменения y .

Функции `ContourPlot` и `DensityPlot` могут использовать большинство опций, которые есть у `Plot3D`. Например, можно увеличить количество точек, установить цвет и т.д.

**ContourPlot[Sin[xy], {x, 0, 3}, {y, 0, 3}, PlotPoints → 20,
ColorFunction → "Pastel"]** (* следующий рисунок слева *)

**DensityPlot[Sin[xy], {x, 0, 3}, {y, 0, 3}, PlotPoints → 30,
Mesh → Full, ColorFunction → Hue, FrameLabel → {x, y},
PlotLabel → Sin[xy]]** (* следующий рисунок в центре *)

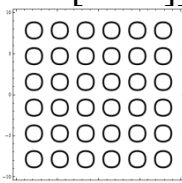


Одной из полезных опций для `DensityPlot` является `Mesh→False`. Это важно в том случае, когда строится высококачественная картинка с большим числом точек. Если оставить сетку, то можно получить очень черную картинку (предыдущий рисунок справа).

ContourPlot[Sin[xy], {x, 0, 3}, {y, 0, 3}, PlotPoints → 100, Mesh → All]

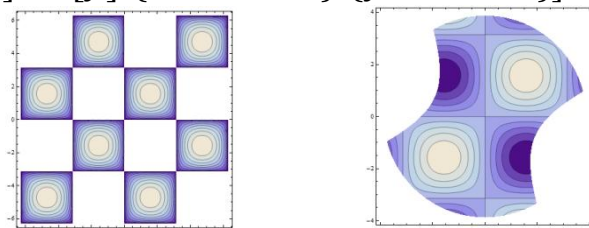
Напомним, что функцию `ContourPlot` мы ранее использовали для построения плоских кривых, уравнения которых заданы неявно.

**ContourPlot[Sin[x]²Sin[y]² == 0.3, {x, -10, 10}, {y, -10, 10},
ContourStyle → Thickness[0.01]]**



Области изменения переменных, где решений нет (или решения комплексные) функциями `DensityPlot` и `ContourPlot` исключаются из построения автоматически (следующий рисунок слева)

ContourPlot[$\sqrt{\text{Sin}[x]\text{Sin}[y]}$, {x, -2π, 2π}, {y, -2π, 2π}]



Но вы можете самостоятельно ограничить область отображения графиков с помощью опции `RegionFunction`. Она позволяет рисовать график не во всем прямоугольнике, определяемом передаваемыми диапазонами

$\{x, x_{\min}, x_{\max}\}$, $\{y, y_{\min}, y_{\max}\}$, а в некоторой его подобласти. Подобласть определяется булевой функцией трех переменных x, y, z . График рисуется только там, где эта булева функция принимает значение True.

ContourPlot[Sin[x]Sin[y], {x, -4, 4}, {y, -4, 4},

RegionFunction → **Function**[{x, y, z}, $x^2 - 2xy - y^2 < 6 \ \&\& \ x^2 + y^2 < 15$]]

(предыдущий рисунок справа). Опция RegionFunction имеется у многих графических функций.

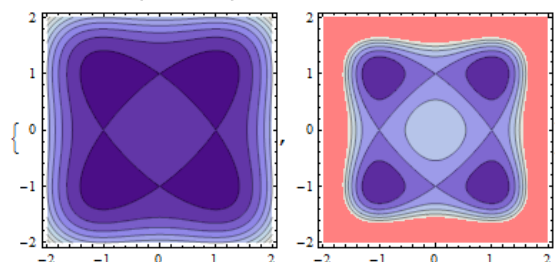
Когда мы строим контурный график выражения $f(x, y)$, то с помощью опции PlotRange можно уточнить диапазон изменения значений f , который следует обозначить на рисунке. При этом области плоскости XY, в которых значение f выходит за указанные в PlotRange пределы, обрезается. Опция ClippingStyle можно использовать для определения стиля/цвета заполнения обрезаемой области.

{ContourPlot[$x^4 - 2x^2 + y^4 - 2y^2 + 1$, {x, -2, 2}, {y, -2, 2},

ClippingStyle → None],

ContourPlot[$x^4 - 2x^2 + y^4 - 2y^2 + 1$, {x, -2, 2}, {y, -2, 2},

PlotRange → {-2, 2}, ClippingStyle → Pink]}

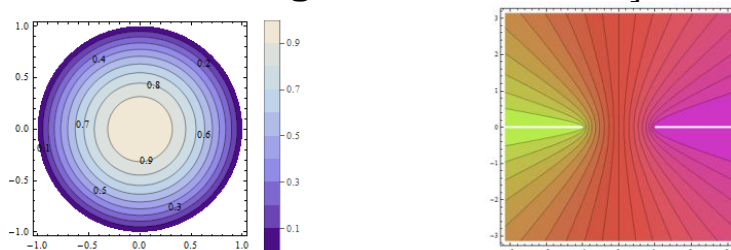


На линии уровня можно наносить метки с помощью опции ContourLabels→True, а опция PlotLegends→Automatic позволяет вывести рядом с графиком цветовую палитру.

ContourPlot[$1 - x^2 - y^2$, {x, -1, 1}, {y, -1, 1},

ContourLabels → **True**, **Contours** → **9**, **PlotRange** → **{0, 1}**,

ClippingStyle → **None**, **PlotLegends** → **Automatic**]



Использование комплексных функций также допустимо, но выражение, график которого строится, должно быть «вещественнозначным», а области «комплекснозначности» исключаются из построения.

ContourPlot[Re[ArcSin[x + Iy]], {x, -Pi, Pi}, {y, -Pi, Pi},

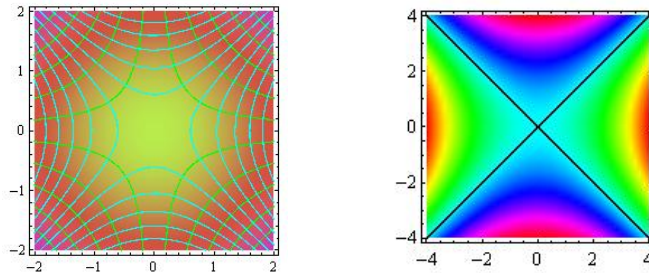
ColorFunction → "NeonColors", **Contours** → **20**]

(предыдущий график справа).

Также как и для других графических функций, в функциях `DensityPlot` и `ContourPlot` можно использовать опции `Mesh`, `MeshFunctions` и `MeshStyle` для построения криволинейной сети (следующий рисунок слева).

$f = ((\#1)^2) \&$

DensityPlot[Abs[f[x + Iy]], {x, -2, 2}, {y, -2, 2}, MeshFunctions → Function@@@{{{x, y, z}, Re[f[x + Iy]]}, {{x, y, z}, Im[f[x + Iy]]}}, ColorFunction → NeonColors, Ticks → None, Mesh → 10, MeshStyle → {Cyan, Green}]



Опцию `Mesh` можно использовать для обозначения точек пересечения поверхности с плоскостью $z=0$

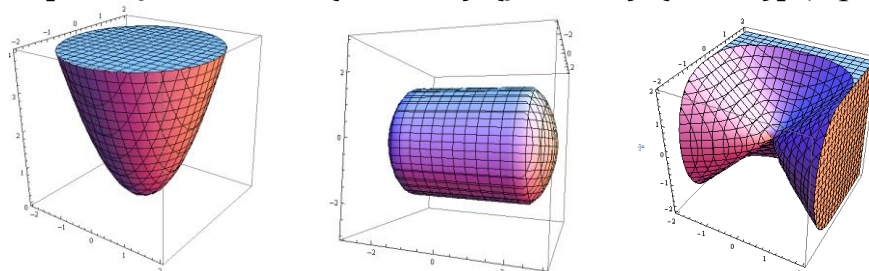
DensityPlot[y² - x², {x, -4, 4}, {y, -4, 4}, MeshFunctions → {#3&}, MeshStyle → Thickness[Medium], Mesh → {{0}}, ColorFunction → Hue, PlotPoints → 50]

(предыдущий рисунок справа). Здесь `Mesh` – функция возвращает значение $z = y^2 - x^2$, а линии сети строятся только для уровня $z=0$ (`Mesh`→{ {0} }).

2.3.2 Трехмерные области и поверхности, заданные неявно

Функция `RegionPlot3D` рисует поверхность трехмерной области, используя булеву функцию (предикат) трех переменных.

RegionPlot3D[x² + y² - z < 0, {x, -2, 2}, {y, -2, 2}, {z, 0, 4}] (* рис. слева *)



RegionPlot3D[x² + y² + z² ≤ 9 && y² + z² ≤ 4, {x, -3, 3}, {y, -3, 3}, {z, -3, 3}]

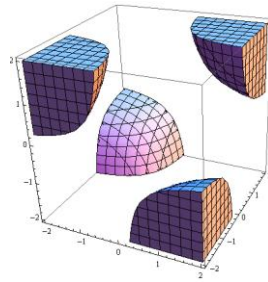
(предыдущий рисунок в центре).

RegionPlot3D[x² + y³ - z² > 0, {x, -2, 2}, {y, -2, 2}, {z, -2, 2}]

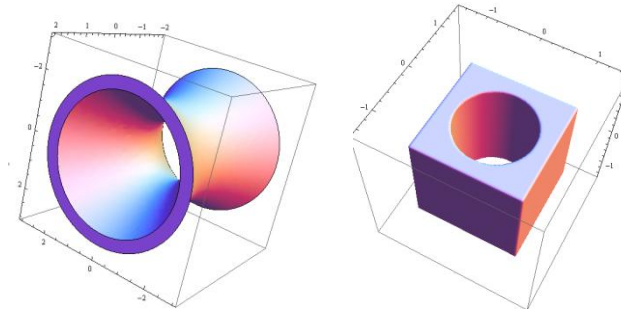
(предыдущий рисунок справа).

Область не обязана быть связной

RegionPlot3D[xyz ≥ 1, {x, -2, 2}, {y, -2, 2}, {z, -2, 2}]

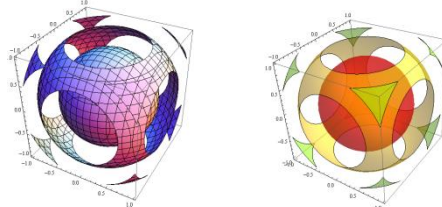


Предикат может быть составлен из любой логической комбинации неравенств
RegionPlot3D $[x^2 + y^2 - z^2 \leq 3 \ \&\& \ x^2 + y^2 - z^2 \geq 1,$
 $\{x, -3, 3\}, \{y, -3, 3\}, \{z, -2, 2\}, \text{Mesh} \rightarrow \text{False}, \text{PlotPoints} \rightarrow 30$
 (следующий рисунок слева)



RegionPlot3D $[-1 \leq x \leq 1 \ \&\& \ -1 \leq y \leq 1 \ \&\& \ -1 \leq z \leq 1 \ \&\& \ x^2 + y^2 \geq \frac{1}{2}, \{x, -1.5, 1.5\}, \{y, -1.5, 1.5\}, \{z, -1.5, 1.5\},$
 $\text{Mesh} \rightarrow \text{None}, \text{PlotPoints} \rightarrow 100]$ (* предыдущий рисунок справа *)

Функция **ContourPlot3D** предназначена для построения поверхностей постоянного значения (поверхностей уровня) функций трех переменных
ContourPlot3D $[x^2 + y^2 + z^2, \{x, -1, 1\}, \{y, -1, 1\}, \{z, -1, 1\}]$ (* рис. слева *)



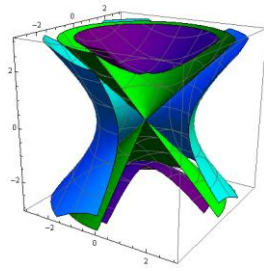
Поскольку поверхности часто закрывают друг друга, то их лучше рисовать полупрозрачными и не рисовать сетку.

ContourPlot3D $[x^2 + y^2 + z^2, \{x, -1, 1\}, \{y, -1, 1\}, \{z, -1, 1\},$
 $\text{ContourStyle} \rightarrow \text{Opacity}[0.5], \text{Mesh} \rightarrow \text{None},$
 $\text{ColorFunction} \rightarrow (\text{Hue}[\#4/4]\&)]$ (* предыдущий рисунок справа *)

Директива **Opacity** $[h]$ определяет степень h прозрачности объекта ($0 \leq h \leq 1$).

Можно также рисовать не всю поверхность, оставляя пространство для того, чтобы заглянуть внутрь поверхности.

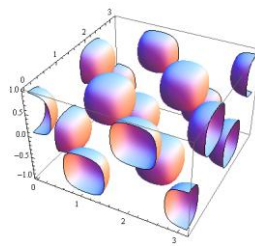
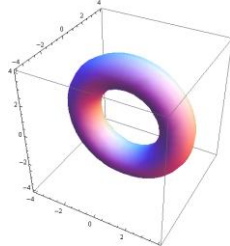
ContourPlot3D $[x^2 + y^2 - z^2, \{x, -3, 3\}, \{y, -3, 3\}, \{z, -3, 3\},$
 $\text{Contours} \rightarrow \{-3, 0, 3\}, \text{ContourStyle} \rightarrow \{\text{Purple}, \text{Green}, \text{Cyan}\},$
 $\text{Mesh} \rightarrow 5, \text{MeshStyle} \rightarrow \text{GrayLevel}[0.4],$
 $\text{RegionFunction} \rightarrow \text{Function}[\{x, y, z\}, x < 0 || y > 0]]$



Функция `ContourPlot3D` может использоваться для построения поверхностей, уравнения которых заданы неявно. Вот пример неявного уравнения поверхности тора (следующий рисунок слева)

$R = 3; r = 1;$

`ContourPlot3D[(x2 + y2 + z2 + R2 - r2)2 - 4R2(x2 + z2) == 0, {x, -4, 4}, {y, -4, 4}, {z, -4, 4}, Mesh → None]`



Неявное уравнение может задавать несвязную поверхность

`ContourPlot3D[Cos[3x]Cos[3y]Sin[3z] == 1/4, {x, 0, π}, {y, 0, π}, {z, -1, 1}, Mesh → None, BoxRatios → Automatic, PlotPoints → 10]`

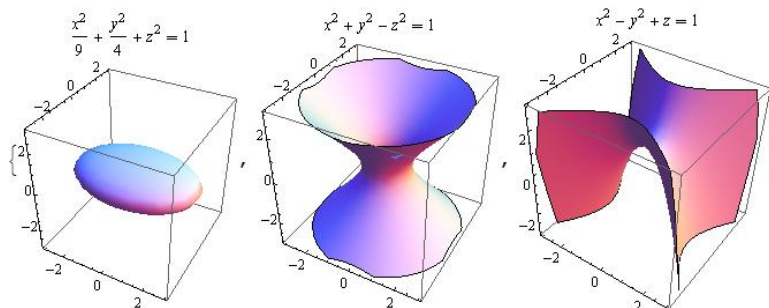
(предыдущий рисунок справа). Директива `BoxRatios` используется для задания пропорций между длинами сторон охватывающего параллелепипеда.

Вот пример построения поверхностей эллипсоида, однополостного гиперболоида и гиперболического параболоида по их неявным уравнениям.

`Table[`

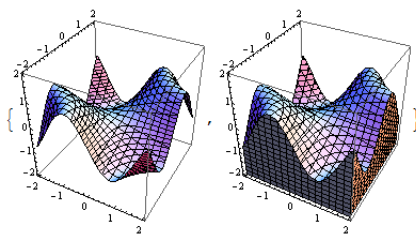
`ContourPlot3D[Evaluate[ff], {x, -3, 3}, {y, -3, 3}, {z, -3, 3}, PlotLabel → ff, Mesh → None],`

`{ff, {x2/9 + y2/4 + z2 == 1, x2 + y2 - z2 == 1, x2 - y2 + z == 1}}`



Функции `ContourPlot3D` и `RegionPlot3D` могут строить одну и ту же явно заданную поверхность с той разницей, что во втором случае рисуются также боковые грани получаемого трехмерного тела.

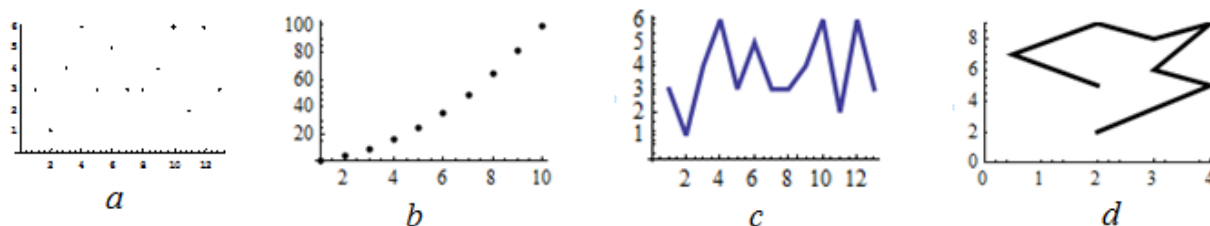
`{ContourPlot3D[Sin[xy] == z, {x, -2, 2}, {y, -2, 2}, {z, -2, 2}], RegionPlot3D[Sin[xy] ≥ z, {x, -2, 2}, {y, -2, 2}, {z, -2, 2}]}`



2.3.3 Графики списков значений.

Функция `ListPlot` графически изображает список значений в двух измерениях.

`ListPlot[{3, 1, 4, 6, 3, 5, 3, 3, 4, 6, 2, 6, 3}]` (* следующий рис. *a* *)



Номер элемента в списке представляет горизонтальную координату точки, а само значение – вертикальную. Точки, которые используются для графика, часто бывают очень маленькими. Их можно увеличить с помощью опции `PlotStyle`

`ListPlot[{3, 1, 4, 6, 3, 5, 3, 3, 4, 6, 2, 6, 3}, PlotStyle → PointSize[0.02]]`

Число 0.02 в этом примере для опции `PointSize` определяет размер точки, как доля ширины графика.

Часто используют функцию `Table` вместе с функцией `ListPlot`

`ListPlot[Table[k2, {k, 10}], PlotStyle → {PointSize[0.03], Black}, PlotRange → All]` (* предыдущий рис. *b* *)

Чтобы точки соединить в *Mathematica 5* использовалась опция `PlotJoined`.

`ListPlot[{3, 1, 4, 6, 3, 5, 3, 3, 4, 6, 2, 6, 3}, PlotJoined → True]`

Сейчас для соединения точек используется опция `Joined`.

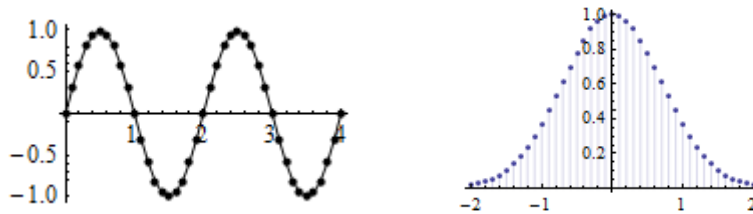
`ListPlot[{3, 1, 4, 6, 3, 5, 3, 3, 4, 6, 2, 6, 3}, Joined → True, PlotStyle → Thickness[0.02]]` (* предыдущий рис. *c* *)

Если вы хотите определить значения *x* и *y* каждой точки, вы можете задать список пар чисел для `ListPlot`

`ListPlot[{{2, 5}, {0.5, 7}, {2, 9}, {3, 8}, {4, 9}, {3, 6}, {4, 5}, {2, 2}}, Joined → True, PlotRange → {{0, 4}, {0, 9}}, PlotStyle → {Thickness[0.02], Black}]` (* предыдущий рис. *d* *)

Как обычно существует очень много опций и дополнительных возможностей для `ListPlot`. Если вы хотите нарисовать кривую и точки, то можно использовать опцию `Mesh→Full` (следующий рисунок слева).

`ListPlot[Table[{x, Sin[πx]}, {x, 0, 4, 0.1}], Mesh → Full, Joined → True, PlotRange → All, PlotStyle → {PointSize[0.02], Black}]`

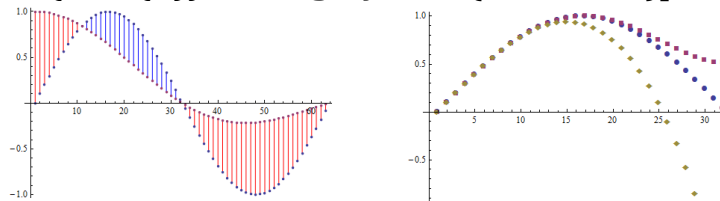


Как и у других графических функций, у ListPlot есть опция Filling.

ListPlot[Table[{k, Exp[-k²]}, {k, -2, 2, 0.1}], Filling -> Axis]

(предыдущий рисунок справа). Опцию Filling. можно использовать для выделения различий между множествами точек (следующий рисунок слева).

**ListPlot[{Table[Sin[x], {x, 0, 2Pi, 0.1}], Table[Sinc[x], {x, 0, 2Pi, 0.1}]],
Filling -> {1 -> {2}}, FillingStyle -> {Red, Blue}]**



Можно использовать разные маркеры для обозначения точек.

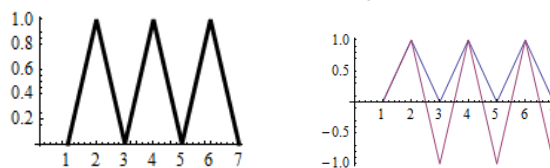
ListPlot[{Table[Sin[x], {x, 0, pi, 0.1}], Table[x - $\frac{x^3}{3!} + \frac{x^5}{5!}$, {x, 0, pi, 0.1}],

Table[x - $\frac{x^3}{3!}$, {x, 0, pi, 0.1}], PlotMarkers -> Automatic]

(предыдущий рисунок справа).

Таким образом, основное назначение функции ListPlot состоит в рисовании множества точек. Если вам нужно рисовать ломаную, проходящую через точки, то можно использовать функцию ListLinePlot. Если ее аргументом является одинарный список, то его элементы трактуются как вертикальные координаты точек, а горизонтальными являются номера точек в списке (следующий рисунок слева).

ListLinePlot[{0, 1, 0, 1, 0, 1, 0}, PlotStyle -> {Thickness[0.02], Black}]



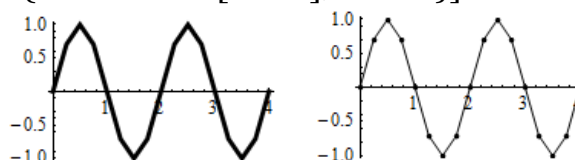
Можно строить ломаные для нескольких списков значений

ListLinePlot[{{0, 1, 0, 1, 0, 1, 0}, {0, 1, -1, 1, -1, 1, -1}}]

(предыдущий рисунок справа).

Если вы хотите определить значения x и y каждой точки, вы можете задать список пар чисел для ListLinePlot (следующий рисунок слева)

**ListLinePlot[Table[{x, Sin[pi x]}, {x, 0, 4, 0.25}],
PlotStyle -> {Thickness[0.02], Black}]**



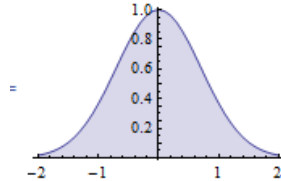
Используя опцию `Mesh`, вы можете нарисовать и ломанную и ее узлы

```
ListLinePlot[Table[{x, Sin[πx]}, {x, 0, 4, 0.25}],
             PlotStyle → {Black}, Mesh → All]
```

(предыдущий рисунок справа).

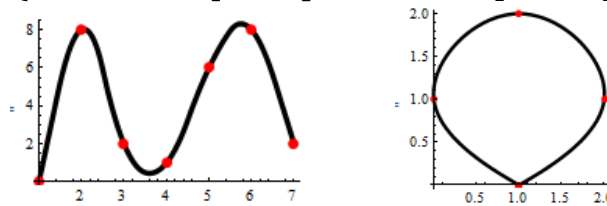
Как и других графических функций, у `ListLinePlot` есть опция `Filling`.

```
ListLinePlot[Table[{k, Exp[-k^2]}, {k, -2, 2, 0.1}], Filling → Axis]
```



По умолчанию функция `ListLinePlot` строит интерполяционную ломаную, проходящую через множество точек. Но порядок интерполяционной кривой вы можете изменить. Использование опции `InterpolationOrder → n` означает, что кривая интерполируется кусочными полиномами степени `n`, которые сглаживаются между собой. Например, в следующем примере используется квадратичная сплайн – интерполяция.

```
ListLinePlot[{0, 8, 2, 1, 6, 8, 2}, InterpolationOrder → 2,
             Mesh → Full, MeshStyle → {PointSize[0.04], Red},
             PlotStyle → {Thickness[0.02], Directive[Black]}]
```

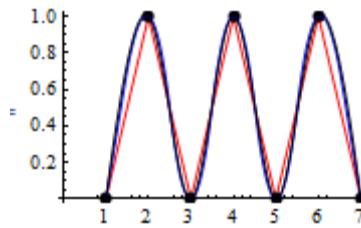


В следующем коде строится кубическая сплайн кривая

```
ListLinePlot[{{1, 0}, {2, 1}, {1, 2}, {0, 1}, {1, 0}}, InterpolationOrder → 3,
             Mesh → Full, MeshStyle → {PointSize[0.04], Red},
             PlotStyle → {Thickness[0.02], Directive[Black]},
             AspectRatio → Automatic] (* предыдущий рисунок справа *)
```

Вот как выглядят интерполяционные кривые при разном порядке интерполирования и одном и том же множестве точек

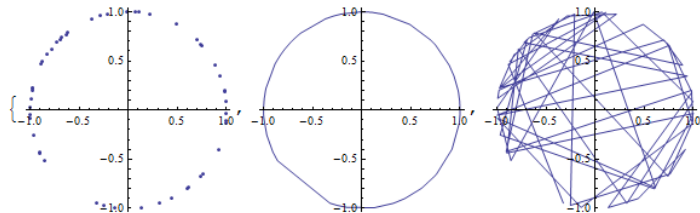
```
Show[
  ListLinePlot[{0, 1, 0, 1, 0, 1, 0}, Mesh → Full,
               MeshStyle → PointSize[0.04], PlotStyle → Red],
  ListLinePlot[{0, 1, 0, 1, 0, 1, 0},
               InterpolationOrder → 2, PlotStyle → Blue],
  ListLinePlot[{0, 1, 0, 1, 0, 1, 0},
               InterpolationOrder → 3, PlotStyle → Black]]
```



Напомним, что функция `Show` объединяет несколько графиков.

Функция `ListLinePlot` пытается нарисовать кривую, проходящую через множество точек в том порядке, в котором они заданы в списке, а функция `ListCurvePathPlot` пытается построить гладкую кривую, проходящую через множество точек без учета их последовательности в интерполируемом множестве. В следующем примере генерируются точки, лежащие на окружности, в случайном порядке.

```
data = Table[{Cos[t], Sin[t]}, {t, RandomReal[{0, 2Pi], 50}]];
{ListPlot[data, AspectRatio -> Automatic],
 ListCurvePathPlot[data],
 ListLinePlot[data, AspectRatio -> Automatic]}
```



Здесь левый рисунок отображает множество точек, правый – ломаную, которая получена последовательным соединением точек в том порядке, в котором они идут в множестве/списке `data`, а средний – сглаженную функцией `ListCurvePathPlot` кривую.

Элементы вещественной матрицы, задаваемой в *Mathematica* как список списков, можно интерпретировать как яркости пикселей изображения. Функцией `Image` элементы такой матрицы, находящиеся в диапазоне от 0 до 1, рисуются в виде небольших квадратов; значение 0 соответствует черному цвету, значение 1 – белому, промежуточные значения соответствуют оттенкам серого цвета (следующий рисунок слева).

```
Image[{{0, 0.1, 0.2, 0.3}, {0.4, 0.5, 0.6, 0.7}, {0.8, 0.9, 1, 0}}]
```



Если элементы такой матрицы сами являются списками трех значений из диапазона от 0 до 1, то такие тройки/списки интерпретируются как цвет пикселя/квадрата в формате RGB.

```
Image[{{{1, 1, 0}, {1, 0, 1}, {1, 0, 0}},
 {{0, 1, 0}, {0, 0, 1}, {0, 1, 0}},
 {{0.4, 0.4, 0.4}, {1, 0, 1}, {0, 1, 1}}}] (* предыдущий рисунок в центре *)
```

Вот пример изображения матрицы случайных чисел

```

n = 100; m = 80;
g = RandomReal[{0, 1}, {m, n}];
Image[g] (* предыдущий рисунок справа *)

```

Вот пример грубого представления круга как небольшой матрицы (следующий рисунок слева)

```

f[i, j] = If[i2 + j2 ≤ 4, 1, 0];
g = Table[f[i, j], {i, -3, 3}, {j, -3, 3}];
g//TableForm

```

```

Image[g]
0 0 0 0 0 0 0
0 0 0 1 0 0 0
0 0 1 1 1 0 0
0 1 1 1 1 1 0
0 0 1 1 1 0 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0

```



Вот пример представления круга в виде матрицы большего размера

```

R = 20; R1 = R + 5;
f[i, j] = If[i2 + j2 ≤ R2, 1, 0];
g = Table[f[i, j], {i, -R1, R1}, {j, -R1, R1}];
Image[g] (* предыдущий рисунок в центре *)

```

Вот пример генерирования случайного трехмерного массива и его отображения (предыдущий рисунок справа).

```

Image[RandomReal[{0, 1}, {12, 12, 3}]]

```

Кроме представления матрицы как изображения, функция Image умеет создавать растровое изображение из графического объекта. Например,

```

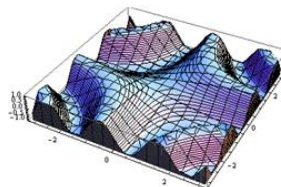
pp = RegionPlot3D[Sin[xy] > z, {x, -π, π}, {y, -π, π}, {z, -1, 1},
BoxRatios → {2π, 2π, 1}];

```

```

pi = Image[pp]

```



Объект **pi** невозможно вращать, как обычный графический 3D объект – это уже растровое изображение. Но его можно сохранить в графический файл, с помощью функции Export.

```

Export["Surf01.gif", pi];

```

На моем компьютере файл сохраняется в каталог C:\Users\User\Documents, однако, это зависит от настроек в вашей системе.

Функция Export поддерживает несколько графических форматов с которыми вы можете познакомиться по справочной системе. Например

```
Export["Surf01.bmp", pi];
```

Прочитать растровое изображение можно с помощью функции `Import`.

```
pi2 = Import["Surf01.gif"]
```

В результате будет показана картинка, приведенная на предыдущем рисунке.

GIF файлы способны хранить несколько растровых изображений, которые могут воспроизводиться как анимация. Создадим, например, список нескольких графических объектов (графиков)

```
p1 = Plot3D[Sin[xy], {x, -pi, pi}, {y, -pi, pi}, BoxRatios -> {2pi, 2pi, 1}];
```

```
p2 = RegionPlot3D[Sin[xy] > z, {x, -pi, pi}, {y, -pi, pi}, {z, -1, 1},  
BoxRatios -> {2pi, 2pi, 1}];
```

```
p3 = ContourPlot3D[Sin[xy] - z == 0, {x, -pi, pi}, {y, -pi, pi}, {z, -1, 1},  
BoxRatios -> {2pi, 2pi, 1}];
```

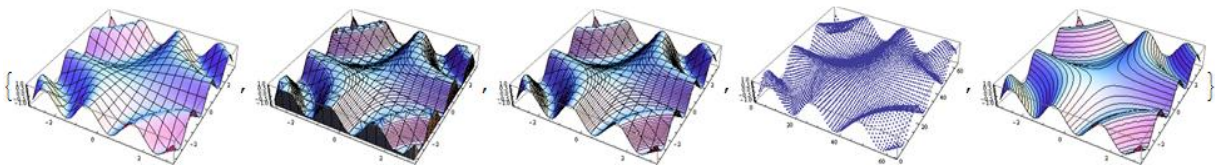
```
p4 = ListPointPlot3D[Table[Sin[i * j], {i, -pi, pi, 0.1}, {j, -pi, pi, 0.1}],  
BoxRatios -> {2pi, 2pi, 1}];
```

```
p5 = Plot3D[Sin[xy], {x, -pi, pi}, {y, -pi, pi}, BoxRatios -> {2pi, 2pi, 1},  
MeshFunctions -> {#1 * #2 &}, Mesh -> 50];
```

```
tp = {p1, p2, p3, p4, p5};
```

Преобразуем каждый из них в растровый образ и создадим из них список

```
rtp = Map[Image, tp]
```



Здесь функция `Map` применяет функцию `Image` (первый аргумент) к каждому элементу списка `tp` (второй аргумент).

Экспортируем список `rtp` в графический файл

```
Export["Surf02.gif", rtp];
```

Теперь файл `Surf02.gif` содержит список растровых изображений, которые будут воспроизводиться как анимация.

Функция `Export` поддерживает несколько форматов анимированных файлов, с которыми вы можете познакомиться по справочной системе. Например, анимацию можно сохранить в файл формата AVI.

```
tl = Table[Plot[Sin[nx], {x, 0, 10}], {n, 1, 10, 0.2}];
```

```
rtl = Map[Image, tl];
```

```
Export["Surf03.avi", rtl];
```

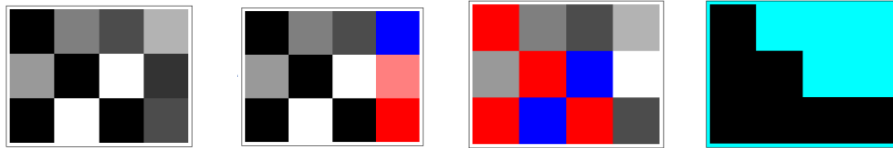
Для отображения массивов данных предназначены функции `ArrayPlot`, `MatrixPlot` и `ReliefPlot`. Вот несколько примеров.

```
ArrayPlot[{{1, 0.5, 0.7, 0.3}, {0.4, 1, 0, 0.8}, {1, 0, 1, 0.7}}]
```

```
ArrayPlot[{{1, 0.5, 0.7, Blue}, {0.4, 1, 0, Pink}, {1, 0, 1, Red}}]
```

```
ArrayPlot[{{1, 0.5, 0.7, 0.3}, {0.4, 1, 0, 0.3}, {1, 0, 1, 0.7}},  
ColorRules -> {1 -> Red, 0 -> Blue}]
```

```
ArrayPlot[{{1}, {1, 1}, {1, 1, 1, 1}}, Background -> Cyan]
```

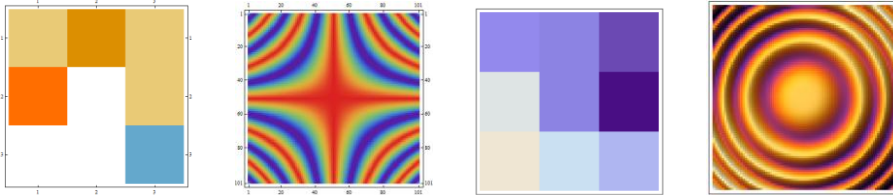


MatrixPlot[{{1, 2, 1}, {3, 0, 1}, {0, 0, -1}}]

MatrixPlot[Table[Cos[x y/150.], {x, -50, 50}, {y, -50, 50}],
ColorFunction → "Rainbow"]

ReliefPlot[{{4, 2, 1}, {3, 0, -2}, {0, 0, -1}}]

ReliefPlot[Table[i j/16 + Cos[i² + j²], {i, -4, 4, .1}, {j, -4, 4, .1}],
ColorFunction → "SunsetColors"]



У функции Image, ArrayPlot, MatrixPlot и ReliefPlot есть много вариантов использования и опций с которыми вы можете познакомиться по справочной системе. В основном они используются при обработки изображений. Это отдельная большая тема, которая выходит за рамки нашего пособия.

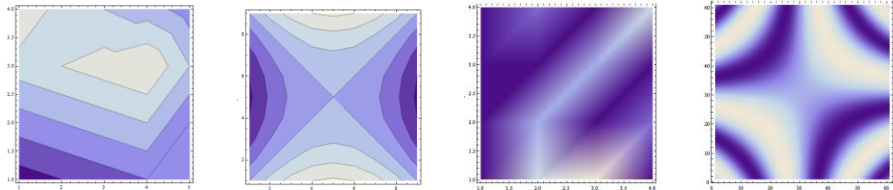
Функция ListContourPlot создает контурный график из двумерного массива, а функция ListDensityPlot создает график плотности.

ListContourPlot[{{1, 2, 3, 4, 6}, {5, 6, 7, 8, 4}, {9, 10, 11, 12, 8}, {12, 9, 8, 7, 5}}]

ListContourPlot[Table[i² - j², {i, -4, 4}, {j, -4, 4}]]

ListDensityPlot[{{1, 4, 5, 1}, {1, 3, 1, 2}, {1, 1, 3, 1}, {1, 2, 1, 4}}]

ListDensityPlot[Table[Sin[xy], {x, -3, 3, 0.1}, {y, -3, 3, 0.1}]]



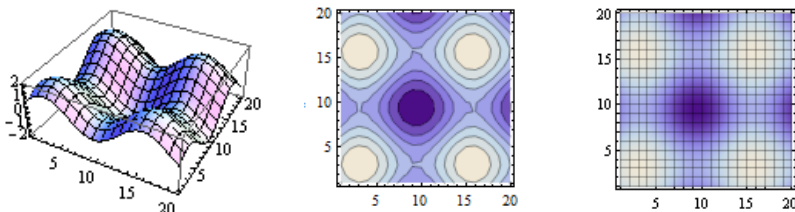
Вот пример представления одного и того же набора данных с использованием трех разных функций

data = Table[Sin[i/2] + Sin[j/2], {i, 1, 20}, {j, 1, 20}];

ListPlot3D[data]

ListContourPlot[data]

ListDensityPlot[data, Mesh → Full]

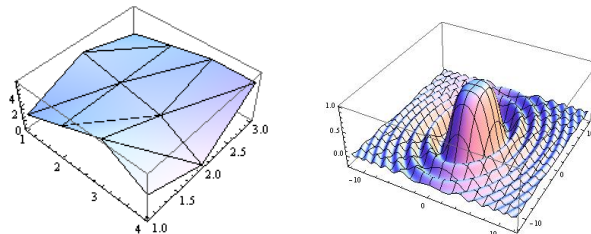


С другими вариантами использования этих функций вы можете познакомиться по справочной системе.

У многих двумерных функций, отображающих списки, есть свои трехмерные аналоги. Перечислим некоторые из них: ListPlot3D, ListPointPlot3D, ListSurfacePlot3D, ListContourPlot3D.

Функция ListPlot3D требует матрицу (список списков) значений z , заданных в узлах квадратной целочисленной сетки (i, j) на плоскости XY.

ListPlot3D[[{1, 3, 5, 2}, {4, 3, 2, -1}, {2, 3, 4, 4}], Mesh → All] (* рис.слева *)



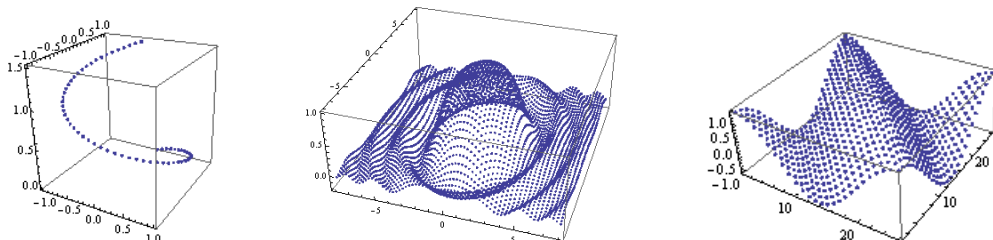
Первый аргумент должен представлять прямоугольный массив или он должен быть списком троек чисел $\{x_i, y_i, z_i\}$, где z_i представляет высоту поверхности в точке $\{x_i, y_i\}$. Вот пример второго формата вызова этой функции

```
data = Flatten[Table[{x, y, Sin [x^2 + y^2] / (x^2 + y^2)},
{x, -12, 12, 0.2}, {y, -12, 12, 0.2}], 1]; //Quiet
ListPlot3D[data, PlotRange → All]
```

(предыдущий рисунок справа). Здесь функция Flatten[expr, 1] потребовалась для реорганизации тройного списка в двойной, который требует функция ListPlot3D во втором формате вызова. Кроме того, мы использовали постфиксный вызов функции Quiet для того, чтобы отменить вывод сообщения о делении на ноль в начале координат.

Приведем примеры использования других трехмерных «списочных» функций. Функция ListPointPlot принимает список пространственных координат точек $\{x_i, y_i, z_i\}$ и рисует их. Точки могут находиться как на кривой так и на поверхности. Вот пример множества точек, расположенных на кривой (следующий рисунок слева)

```
data = Table[{Sin[u], Cos[u], u/4}, {u, 0, 6, 0.1}];
ListPointPlot3D[data, BoxRatios → 1, PlotStyle → PointSize[0.02]]
```



Вот два примера множества точек, расположение которых указывает на поверхность (предыдущий рисунок в центре)

```
data = Flatten[Table[{x, y, Sin [x^2 + y^2] / (x^2 + y^2)},
{x, -8, 8, 0.25}, {y, -8, 8, 0.25}], 1]; //Quiet
```


ListPointPlot3D[data]

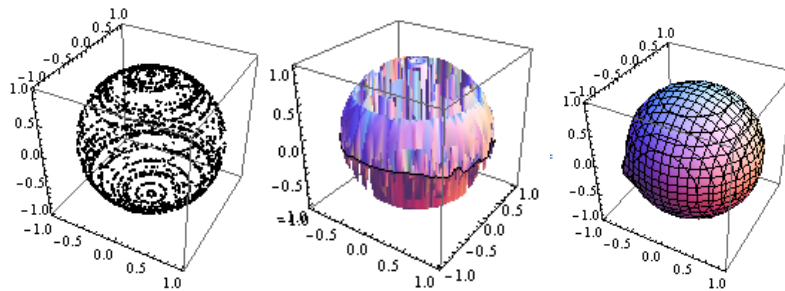
и предыдущий рисунок справа

```
ListPointPlot3D[Table[Cos[j + i], {i, -π, π, 0.25}, {j, -π, π, 0.25}]]
```

Функция ListSurfacePlot3D принимает список координат точек в пространстве и пытается нарисовать поверхность, проходящую через эти точки

В следующем примере мы создаем множество точек, расположенных на сфере в случайном порядке, и строим поверхность, проходящую через это множество с помощью функций ListPlot3D и ListSurfacePlot3D.

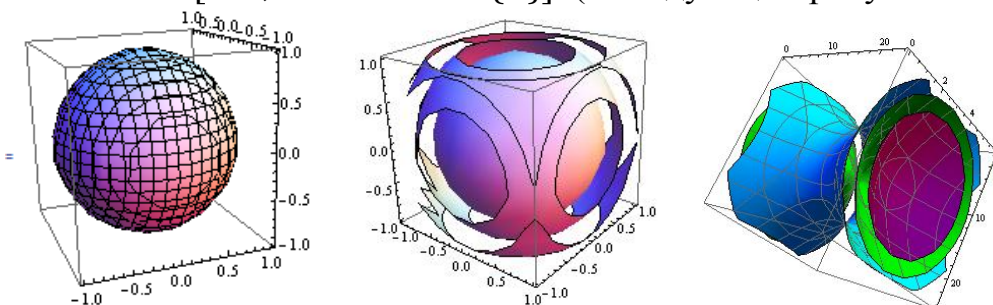
```
dat = Flatten[Table[{Cos[u]Cos[v], Cos[u]Sin[v], Sin[u]},  
  {u, RandomReal[{0, 2π}, 50]}, {v, RandomReal[{0, 2π}, 50]}], 1];  
ListPointPlot3D[dat, BoxRatios → 1]  
ListPlot3D[dat, BoxRatios → {1, 1, 1}, Mesh → Full]  
ListSurfacePlot3D[dat, BoxRatios → 1]
```



Здесь на левом рисунке показано сгенерированное множество точек, на среднем – поверхность, которая получается при попытке функции ListPlot3D представить это множество (она учитывает порядок точек в множестве/списке), на правом – поверхность, аппроксимирующая это множество (без учета порядка точек).

Функция ListContourPlot3D выполняет работу аналогичную функции ListSurfacePlot3D, но умеет строить несколько поверхностей. В формате ListContourPlot3D[{{x₁, y₁, z₁}, ...], Contours->{c} она принимает массив координат точек – список вида {{x₁, y₁, z₁}, ...}, создает список {{x₁, y₁, z₁, c}, ...} и путем интерполяции пытается представить эти значения в форме поверхности уровня некоторой функции $f(x, y, z) = c$. Если опция Contours->{c₁, c₂, ..., c_n} содержит список значений, то функция ListContourPlot3D пытается построить поверхности уровня для нескольких значений c_i. Например для предыдущего набора точек, случайно расположенных на поверхности сферы, получаем

```
ListContourPlot3D[dat, Contours → {0}] (* следующий рисунок слева *)
```



Использование опции `Contours` с несколькими значениями создает несколько поверхностей

```
ListContourPlot3D[dat, Contours → {0, 0.2, 0.4}, Mesh → None]
```

(предыдущий рисунок в центре). Предыдущий рисунок справа создан следующими командами

```
data = Table[x2 + y2 - z2, {x, -3, 3}, {y, -3, 3, 0.25}, {z, -3, 3, 0.25}];
```

```
ListContourPlot3D[data, Contours → {-3, 0, 3},  
  ContourStyle → {Purple, Green, Cyan},  
  Mesh → 5, MeshStyle → GrayLevel[0.4]]
```

2.3.4 Анимация и манипуляторы

Анимация – это последовательность изображений, которые быстро сменяют друг друга, в результате чего появляется движение. Любую последовательность графиков можно анимировать. Анимация в *Mathematica 5* и старших версиях немного отличается.

Опишем выполнение анимации в *Mathematica 5*. Для этого надо создать несколько подряд идущих графиков. Например, следующая команда будет генерировать последовательность из шести графиков, немного отличающихся друг от друга.

```
Do[Plot[Sin[n x], {x, 0, 2 Pi}], {n, 1, 2, 0.2}]
```

Команда `Do` – универсальная итерационная команда/функция. В этом примере функция `Plot` повторяется шесть раз путем изменяя значение `n` от 1 до 2 с шагом 0.2.

Чтобы увидеть анимацию необходимо щелкнуть мышью на большой квадратной скобке, которая объединяет все графики. Группа графических ячеек (cells), содержащая изображения, будет помечена (скобка справа превратится в толстую черную линию). После этого необходимо выполнить команду меню запуска анимации `Cell - Animate Selected Graphics` или нажать комбинацию клавиш `Ctrl-Y`. Можно просто выполнить двойной щелчок мышью по любой картинке последовательности. Перемещения появляются в одной из ячеек, содержащей графическое изображение. Вы можете изменить скорость и направление движения анимации, нажимая кнопки, которые появляются во время анимации в левой нижней части панели `StatusBar`.

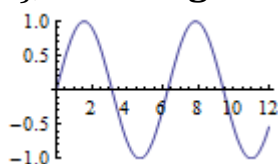


Кнопки изменения скорости
Изменения направления анимации
Включение прокрутки вперед-назад

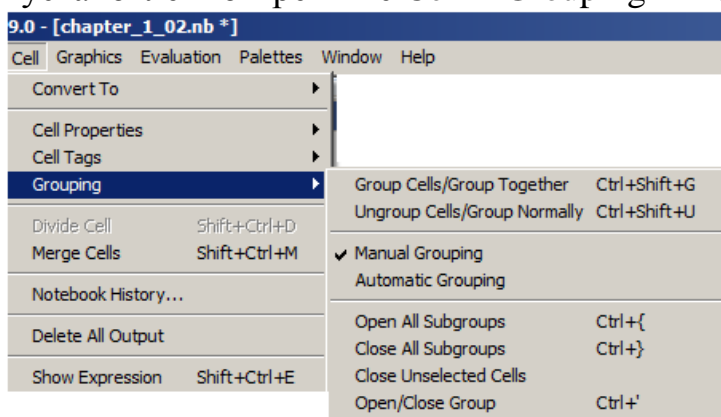
Вы можете создавать 3D анимацию таким же образом, используя функцию `Plot3D` вместо `Plot`.

```
Do[Plot3D[Sin[x y], {x, -n, n}, {y, -n, n}, PlotRange->{-1, 1},  
  PlotPoints->20 ], {n, 1, 3, 0.5}]
```

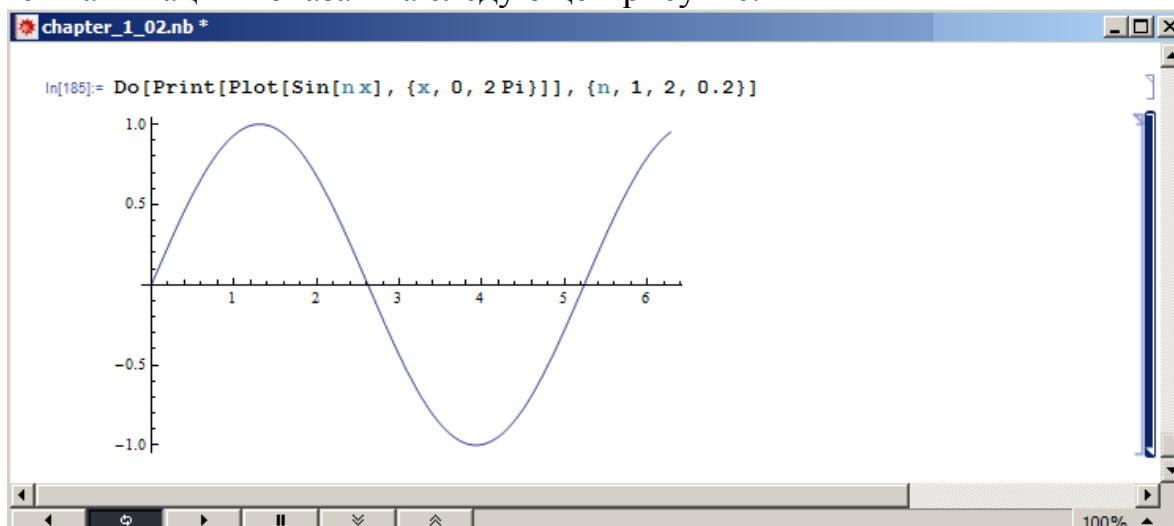
В новых версиях Mathematica (*version* ≥ 6) необходимо использовать функцию `Print`. Она должна обрамлять графическую функцию, строящую изображения.
Do[Print[Plot[Sin[ax], {x, 0, 12}, PlotRange → {-1, 1}], {a, 1, 3, 0.1}]



Лучший результат анимации получается, когда есть охватывающая все графики скобка. Если ее нет, то можно вручную перегруппировать секции. Для этого можно использовать меню `Cell – Grouping – Group Cells` и `Cell – Grouping – Ungroup Cells` при установленном режиме `Cell – Grouping – Manual Grouping`.



После создания скобки, охватывающей все секции с графиками, выполните по ней двойной щелчок; это свернет последовательность секций, оставив видимой только первую секцию. Теперь выделите скобку этой секции (или охватывающую все графики скобку) и из меню `Graphics - Rendering - Animate Selected Graphics` запускайте анимацию (или нажмите комбинацию клавиш `Ctrl-Y`). Внизу появляется панель управления анимацией. Вид окна документа в момент анимации показан на следующем рисунке.



Вы можете изменить скорость и направление движения, нажимая кнопки, которые появляются во время анимации в левой нижней части окна документа.



Кнопки изменения скорости
 Изменения направления анимации
 Включение прокрутки вперед-назад

Вы можете создавать 3D анимацию таким же образом, используя команду `Plot3D` вместо `Plot`.

```
Do[Print[
  Plot3D[Sin[xy], {x, -n, n}, {y, -n, n}, PlotRange -> {-1, 1},
  PlotPoints -> 20]],
{n, 1, 3, 0.5}]
```

Не забывайте перегруппировывать секции, иначе анимация будет захватывать ненужные секции!

Вы можете создавать анимацию с любой графической функцией *Mathematica*. Например,

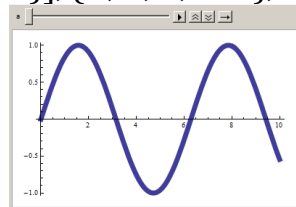
```
Do[Print[
  DensityPlot[Sin[xy], {x, -n, n}, {y, -n, n},
  PlotRange -> {-1, 1}, PlotPoints -> 20]],
{n, 1, 3, 0.5}]
```

В *Mathematica* 6 и последующих версиях для выполнения анимации предназначена функция `Animate`. Она имеет синтаксис

```
Animate[expr, {t, t1, t2}],
```

где кадр от кадра отличается значением параметра t , который изменяется в диапазоне от t_1 до t_2 . Функция создает специальную панель, в которой выполняются все построения

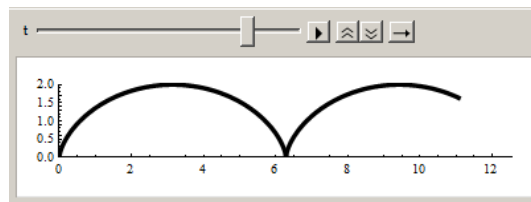
```
Animate[Plot[Sin[ax], {x, 0, 10}], {a, 1, 5, 0.1}, AnimationRunning -> False]
```



$$x[t_] = t - \text{Sin}[t];$$

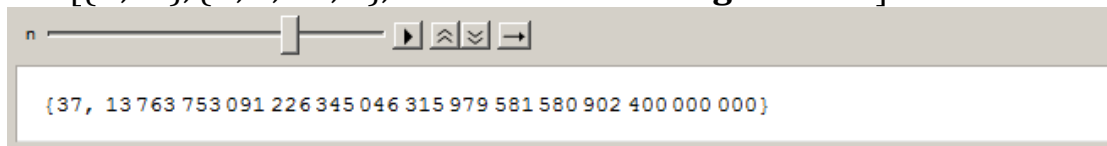
$$y[t_] = 1 - \text{Cos}[t];$$

```
Animate[
  ParametricPlot[{x[tau], y[tau]}, {tau, 0, t}, PlotRange -> {{0, 4pi}, {0, 2}},
  PlotStyle -> {Thickness[0.01], Black}],
{t, 0.1, 4pi, 0.1}]
```



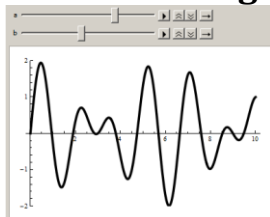
Опция `AnimationRunning->False` используется для того, чтобы сразу после создания панели, анимация не начинала работать. Выражение `expr` может быть не графическим объектом.

Animate[{n, n!}, {n, 1, 50, 1}, AnimationRunning → False]



Разрешается использовать несколько параметров анимации

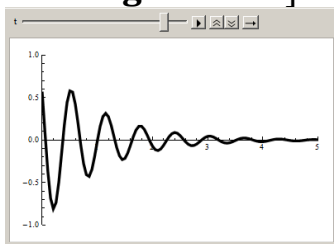
Animate[
Plot[Sin[ax] + Sin[bx], {x, 0, 10}, PlotRange → 2,
PlotStyle → {Thickness[0.01], Black}],
{a, 1, 5}, {b, 1, 5}, AnimationRunning → False]



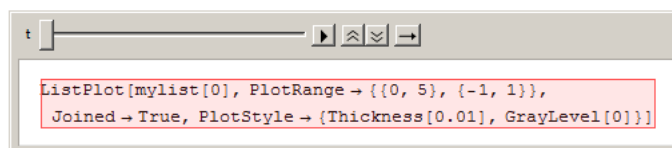
При построении анимации можно использовать любые графические функции.

mylist[t_] = Table[{x, Cos[10x - t]Exp[-x]}, {x, 0, 5, 0.05}];

Animate[
ListPlot[mylist[t], PlotRange → {{0, 5}, {-1, 1}}, Joined → True],
{t, 0, 6, 0.1}, AnimationRunning → False]



Важной опцией у функции **Animate** является опция **Initialization**. После выполнения предыдущего примера сохраните документ в файл, закройте окно документа и Математику, а потом откройте систему и файл. Вполне вероятно, что после открытия файла панель анимации будет пуста или подкрашена в розовый цвет, являющийся признаком ошибки. Например, она может иметь вид



Если вы выполните код секции, то все станет нормально, но до этого такая панель анимации будет вас раздражать. Дело в том, что панель анимации работает в оболочке системы (Front End), а не в ее ядре. Когда система загружает файл, ядро автоматически не вычисляет все выражения и функции, созданные и находящиеся в файле. Поэтому функция **Animate** после повторной загрузки файла не распознает объект **mylist**[t] (код выражения, создающего эту функцию еще не находится в рабочем пространстве системы) и оболочка системы не может правильно заполнить панель анимации данными. Если вы выполните код секции, то в памяти системы появится объект/функция

`mylist[t]` и графическое поле панели анимации будет обновлено. Опция `Initialization` предназначена для автоматического выполнения кода, указанного в этой опции, при первом появлении панели анимации в окне документа. Она имеет синтаксис `Initialization:→(код)`, где код – это любая последовательность команд системы. Таким образом, для предыдущего примера правильным будет следующий код

```
Animate[ListPlot[mylis[t],
                PlotRange → {{0, 5}, {-1, 1}}, Joined → True,
                PlotStyle → {Thickness[0.01], Black}},
{t, 0, 6, 0.1}, AnimationRunning → False,
Initialization :→ (mylis[t_]
                    = Table[[x, Cos[10x - t]Exp[-x]], {x, 0, 5, 0.05]])]
```

Здесь определение функции `mylist[t]` перенесено в тело опции `Initialization`, которое выполняется при появлении окна панели анимации в видимой области документа. Заметим, что опция `Initialization` имеется у многих функций, которые выполняются в оболочке системы.

Также корректно будет работать следующий код

```
DynamicModule[[mylst],
mylst[t_] = Table[[x, Cos[10x - t]Exp[-x]], {x, 0, 5, 0.05]];
Animate[ListPlot[mylst[t],
            PlotRange → {{0, 5}, {-1, 1}}, Joined → True,
            PlotStyle → {Thickness[0.01], Black}},
{t, 0, 6, 0.1}, AnimationRunning → False]]
```

Функция `DynamicModule` используется для того, чтобы ограничить область действия динамических переменных. О них мы будем говорить отдельно в других частях нашего пособия. Функция имеет следующий синтаксис

```
DynamicModule[[x, y, ...], выражение]
```

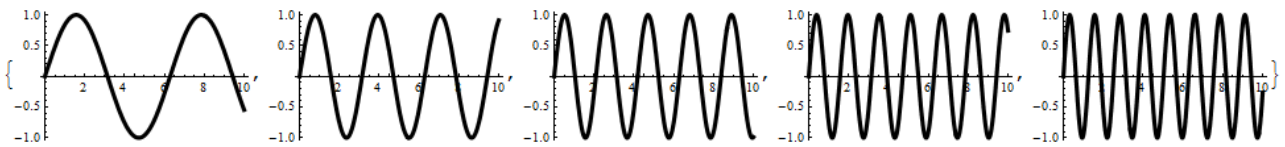
или

```
DynamicModule[[x=x0, y=y0, ...], выражение]
```

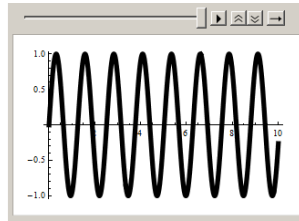
и ограничивает динамическую область для переменных `x, y, ...`. Вторым способом задает начальные значения динамических переменных. В нашем примере функция `DynamicModule` создает блок кода, в котором имя `mylst` является динамической защищенной в пределах этого блока переменной. Значения/выражения локальных переменных динамического блока сохраняются в файле и существуют в следующих сессиях работы с этим файлом.

Есть еще один способ создания анимации с помощью функции `ListAnimate`. Эта функция принимает список графических объектов и воспроизводит их последовательно в панели анимации. Это похоже на анимацию с использованием способа `Do[Print[...]`, только последовательность графических объектов воспроизводится в панели анимации.

```
tp = Table[Plot[Sin[nx], {x, 0, 10}], {n, 5}]
```



ListAnimate[tp]



Обычное использование функции ListAnimate будет с точкой с запятой после вызова функции Table. Это необходимо, чтобы предотвратить отображение большого количества графиков. Кроме того, как правило, будет использоваться опция AnimationRunning → False. Таким образом, приведенные команды чаще будут использоваться следующим образом

tp = Table[Plot[Sin[nx], {x, 0, 10}], {n, 5};

ListAnimate[tp, AnimationRunning → False]

В примере список **tp** состоял из графиков (графических объектов) и создавался с помощью функции Table. Но список из графических объектов можно создавать любым способом, например, перечислением объектов в списке. В следующем коде строится несколько различных графиков одной и той же функции $z = \sin(xy)$, которые затем последовательно воспроизводятся функцией ListAnimate.

p1 = Plot3D[Sin[xy], {x, -π, π}, {y, -π, π}, BoxRatios → {2π, 2π, 1}];

**p2 = RegionPlot3D[Sin[xy] > z, {x, -π, π}, {y, -π, π}, {z, -1, 1},
BoxRatios → {2π, 2π, 1}];**

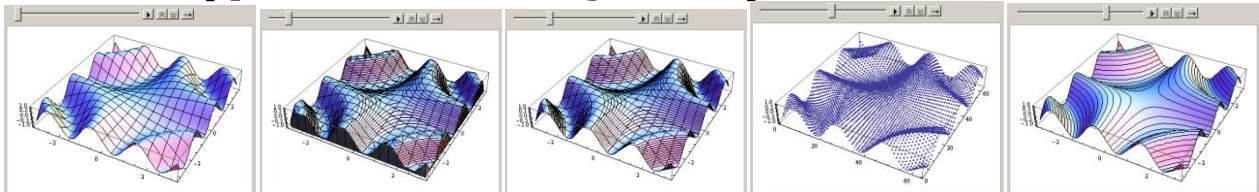
**p3 = ContourPlot3D[Sin[xy] - z == 0, {x, -π, π}, {y, -π, π}, {z, -1, 1},
BoxRatios → {2π, 2π, 1}];**

**p4 = ListPointPlot3D[Table[Sin[i * j], {i, -π, π, 0.1}, {j, -π, π, 0.1}],
BoxRatios → {2π, 2π, 1}];**

**p5 = Plot3D[Sin[xy], {x, -π, π}, {y, -π, π}, BoxRatios → {2π, 2π, 1},
MeshFunctions → {#1 * #2 &}, Mesh → 50];**

tp = {p1, p2, p3, p4, p5};

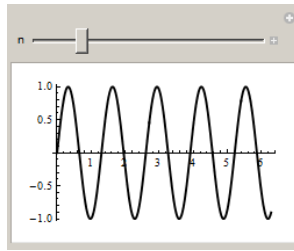
ListAnimate[tp, AnimationRunning → False]



Кроме панели анимации в Mathematica version ≥ 6 есть другие панели, динамически управляющие своим содержимым, например, графиком

Manipulate[

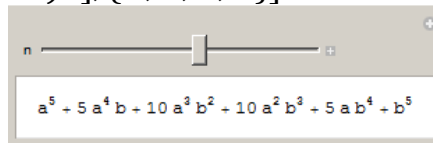
**Plot[Sin[nx], {x, 0, 2Pi}, PlotStyle → {Thickness[0.01], Black}],
{n, 1, 20}]**



Передвигая бегунок панели Manipulate, вы будете менять значение параметра n, к которому он «привязан».

Содержимое панели Manipulate не обязано быть графическим объектом

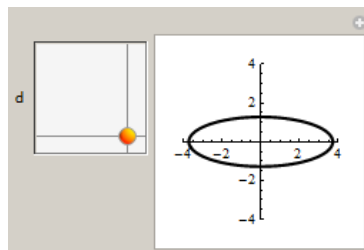
Manipulate[Expand[(a + b)ⁿ], {n, 1, 8, 1}]



Можно создать 2D манипулятор. Перемещение его указателя в пределах прямоугольного окна управляет значением (парой чисел), связанным с манипулятором.

Manipulate[
ParametricPlot[{d[[1]]Cos[t], d[[2]]Sin[t]}, {t, 0, 2Pi},
PlotRange → {{-4, 4}, {-4, 4}},
PlotStyle → {Thickness[0.01], Black}],
{d, {1, 1}, {4, 4}}, **ControlPlacement** → Left]

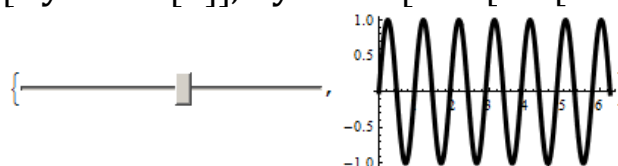
Здесь переменная d представляет список из двух чисел. Эта пара чисел представляет координаты указателя манипулятора и определяется его положением. Список вида {d, {1, 1}, {4, 4}} задает минимальные значения координат указателя по обеим направлениям {1, 1} и максимальные значения {4,4}.



Функция Manipulate очень богатая на различные представления. Только описание ее возможностей могло бы занять целую главу.

Кроме Manipulate существуют другие элементы, которые могут динамически изменять параметры. Например

DynamicModule[{a = .5},
Slider[Dynamic[a], Dynamic[Plot[Sin[12a t], {t, 0, 2π}]]]]



Здесь функция `Dynamic` создает динамическую переменную `a`, привязанную к бегунку (`Slider`). Его перемещение интерактивно меняет значение переменной `a`. Вторая функция `Dynamic[Plot[...]]` следит за изменениями динамических переменных (в нашем случае за значением переменной `a`), и обновляет свое содержимое в соответствии с новым значением `a`. Заключение кода внутрь функции `DynamicModule` разрывает связь динамической переменной `a`, например, с переменной `a` предыдущего примера, но оставляет связь с переменной `a` функции `Plot`, которая расположена в блоке/теле функции `DynamicModule`.

Функция `LocatorPane` предоставляет область с «локатором», который можно перемещать с помощью мыши. В формате

```
LocatorPane[Dynamic[pt], тело]
```

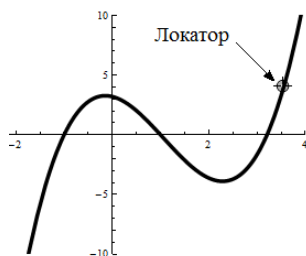
она позволяет динамически менять положение указателя и использовать его координаты в теле (последовательности команд).

В следующем примере мы строим график кубического полинома с двумя корнями в точках ± 1 , который проходит через точку (a, b) . Уравнение такого полинома имеет вид

$$f(x) = (x^2 - 1) \left(x - \frac{a^3 - a - b}{a^2 - 1} \right)$$

Точка (a, b) привязана к «локатору». Его перемещение меняет координаты точки (a, b) и, тем самым, меняет график кубического полинома.

```
DynamicModule[{a = 0, b = 0, f},
  f[x_, a_, b_] = (x^2 - 1) (x - (a^3 - a - b)/(a^2 - 1));
  LocatorPane[Dynamic[{a, b}],
    Dynamic[Plot[f[x, a, b], {x, -4, 4},
      PlotRange -> {{-4, 4}, {-10, 10}},
      PlotStyle -> {Black, Thickness[0.01]}]]]]]
```

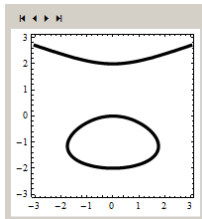


Захватите мышью «локатор» и перемещайте его. Кривая будет автоматически перерисовываться в соответствии с новым положением точки (a, b) . Назначение функций `Dynamic` и `DynamicModule` такое же, как и в предыдущем примере.

Функция `SlideView` отображает панель, в которой можно дискретно менять значения какой – либо переменной. И если она (переменная) входит в выражение по которому строится график внутри этой панели, то каждое ее изменение будет приводить к изменению графика.

SlideView[Table[

```
ContourPlot[x2 - y3 == p y, {x, -3, 3}, {y, -3, 3},  
{p, -4, 4}]]
```



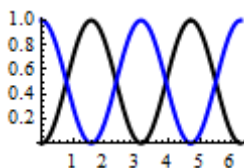
Щелкая по управляющим кнопкам «слайдера», вы будете менять значение параметра p в диапазоне от -4 до 4 с шагом 1 , а следовательно и выражение, график которого отображается в панели.

Имеется еще большое количество управляющих элементов, которые позволяют динамически менять графику, расположенную внутри их области ответственности. Например, еще имеются функции `FlipView`, `MenuView`, `PopupMenu`. С подробным описанием этих и других подобных функций вы можете познакомиться по справочной системе.

2.3.5 Комбинирование графиков

Все графические функции возвращают/создают графические объекты. Уже созданные, такие объекты можно отображать в общем координатном пространстве. Так функция `Show` позволяет отобразить несколько графиков одновременно или показать тот же график, но с другими опциями. Фактически она накладывает один графический объект на другой.

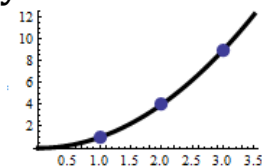
```
ps = Plot[Sin[x]2, {x, 0, 2Pi}, DisplayFunction -> Identity];  
pc = Plot[Cos[x]2, {x, 0, 2Pi}, DisplayFunction -> Identity];  
Show[ps, pc, DisplayFunction -> $DisplayFunction]
```



Установка опции `DisplayFunction -> Identity` отменяет вывод графика на экран, хотя он создается в памяти. Затем функция `Show` отображает созданные графические объекты (здесь `ps` и `pc`) в окне документа. Обратное, опция `DisplayFunction -> $DisplayFunction` при вызове функции `Show` включает отображение графических объектов. Этот способ работает во всех версиях *Mathematica*, начиная с версии 4. В *Mathematica version* ≥ 6 можно не использовать опцию `DisplayFunction -> Identity`. Достаточно завершить вызов графической функции точкой с запятой. Поэтому создание и рисование двух последних графических объектов можно выполнить короче

```
ps = Plot[Sin[x]2, {x, 0, 2Pi};  
pc = Plot[Cos[x]2, {x, 0, 2Pi};  
Show[ps, pc]
```

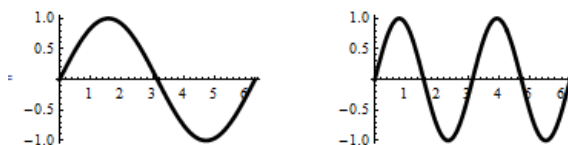
Графики могут быть созданы различными графическими функциями
Show[Plot[x², {x, 0, 3.5}, PlotStyle → {Thickness[0.01], Black}],
ListPlot[{1, 4, 9}, PlotStyle → PointSize[0.04]]]



При использовании функции Show необходимо побеспокоиться о выравнивании масштабов графиков, налагаемых друг на друга.

В *Mathematica 5* функция GraphicsArray создает графический объект, составленный из нескольких других, расположенных рядом (в одну строку).

p1 = Plot[Sin[x], {x, 0, 2Pi}, DisplayFunction → Identity];
p2 = Plot[Sin[2x], {x, 0, 2Pi}, DisplayFunction → Identity];
Show[GraphicsArray[{p1, p2}]]



Изменить расстояние между графиками в окне документа можно с помощью опции GraphicsSpacing.

Show[GraphicsArray[{p1, p2}, GraphicsSpacing → 0.5]]

В *Mathematica version ≥ 6* появились новые функции. Для отображения нескольких графиков в ряд используется функция GraphicsRow. Ей в качестве аргумента передается список имен графических объектов. Например, чтобы отобразить два графических объекта ps и pc из предыдущего примера в одной строке можно выполнить команду

Show[GraphicsRow[{ps, pc}]]

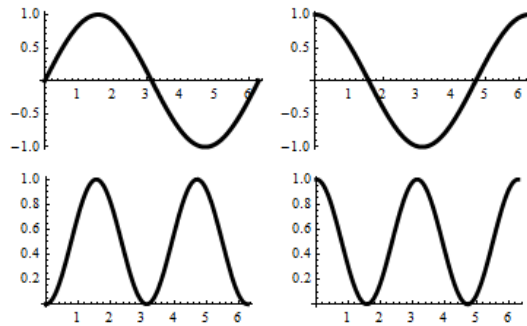
При этом использовать функцию Show необязательно

GraphicsRow[{ps, pc}]

Имеется аналогичная функция GraphicsColumn, располагающая графики один под одним.

Функция GraphicsGrid отображает графики таблично – в несколько строк и столбцов. Ее аргументом должен быть список списков графических объектов.

ps1 = Plot[Sin[x], {x, 0, 2Pi}];
pc1 = Plot[Cos[x], {x, 0, 2Pi}];
ps2 = Plot[Sin[x]², {x, 0, 2Pi}];
pc2 = Plot[Cos[x]², {x, 0, 2Pi}];
GraphicsGrid[{{ps1, pc1}, {ps2, pc2}}]



Есть еще функция `Grid`, которая в виде таблицы может отображать не только графические объекты. Применительно к графикам `ps1`, `pc1`, `ps2`, `pc2` из предыдущего примера, она может быть использована следующим образом

`Grid[{{ps1, pc1}, {ps2, pc2}}, Frame → All]`

В результате будут построены те же графики, что и выше, но оформленные в виде таблицы (с рамкой и разделительными линиями).

Функция `Grid` обладает большими возможностями по оформлению объектов различного типа в виде таблицы. Например

`Grid[{{(a + b)2, (a + b)3}, {Expand[(a + b)2], Expand[(a + b)3]}, Frame → All]`

$(a + b)^2$	$(a + b)^3$
$a^2 + 2ab + b^2$	$a^3 + 3a^2b + 3ab^2 + b^3$

Подробнее с ее возможностями вы можете познакомиться по справочной системе.

2.3.6 Графические примитивы.

Примитивами называют стандартные геометрические фигуры. Они создаются с помощью функций `Graphics[primitives, options]` и `Graphics3D[primitives, options]`. Одним из аргументов этих функций являются имена примитивов, а точнее функции создания примитивов. Они создают математические объекты, которые функциями `Graphics` и `Graphics3D` преобразуются в графические объекты. Эти функции добавляют в структуру объекта примитива стили (цвет, толщину линии и т.д.). Полученные графические объекты могут быть отображены в окне документа с помощью функции `Show` или другим способом. Основными двумерными примитивами являются:

`Circle[{x, y}, r]`

окружность с центром в точке $\{x, y\}$ и радиусом r ;

`Disk[{x, y}, r]`

круг (область) с центром в точке $\{x, y\}$, радиусом r ;

`Line[{{x1, y1}, ...}]`

ломаная с вершинами в точках $\{\{x_1, y_1\}, \dots\}$;

`Point[{x, y}]`

точка с координатами $\{x, y\}$;

`Polygon[{{x1, y1}, ...}]`

замкнутый многоугольник (область) с вершинами в точках $\{\{x_1, y_1\}, \dots\}$;

`Rectangle[{x1, y1}, {x2, y2}]` прямоугольник (область), определяемый двумя диагонально расположенными вершинами;

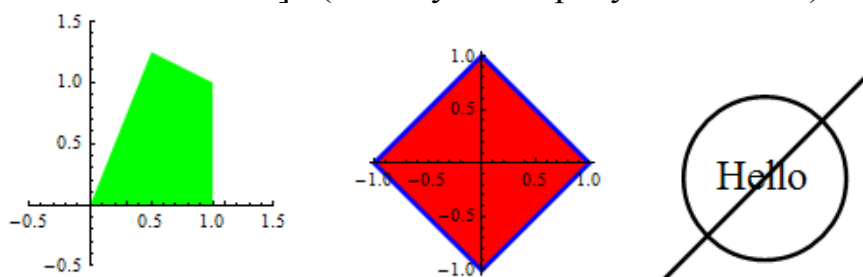
`Text["string", {x, y}]` текстовая строка, расположенная в точке с координатами $\{x, y\}$;

Приведем примеры использования этих примитивов.

Show[Graphics[Disk[{0, 0}, 1]]]

Чтобы добавить цвет функции `Graphics` надо передать первым аргументом директиву задания цвета `RGBColor`. Директивы и примитив, к которому они относятся, следует объединять в список. При этом директива должна стоять перед примитивом.

**Show[Graphics[{RGBColor[0, 1, 0],
Polygon[{{0, 0}, {1, 0}, {1, 1}, {0.5, 1.25}}]},
PlotRange → {{-0.5, 1.5}, {-0.5, 1.5}}, Axes → True,
AspectRatio → Automatic] (* следующий рисунок слева *)**



В *Mathematica version* ≥ 6 при отображении графического примитива можно не использовать функцию `Show` – достаточно функции `Graphics` (или `Graphics3D`). Например две следующие строки кода эквивалентны

Show[Graphics[p]]

Graphics[p]

Вот пример построения красного квадрата с синим контуром без использования функции `Show`.

vertices = {{0, -1}, {1, 0}, {0, 1}, {-1, 0}, {0, -1}};

**p = Graphics[{EdgeForm[{Blue, Thickness[0.02]}],
RGBColor[1, 0, 0], Polygon[vertices]}]**

Графическая директива `EdgeForm[{Blue, Thickness[0.02]}]` определяет, что контур последующего примитива будет иметь заданный цвет и толщину.

Если на рисунок надо наложить какие либо дополнительные графические элементы, например, показать оси координат, то все же следует использовать функцию `Show`.

Show[p, AspectRatio → Automatic, Axes → True] (* пред. рис. в центре *)

Вот пример использования текстового примитива совместно с другими примитивами.

g1 = Graphics[{Thickness[.02], Line[{{-1, -1}, {1, 1}}]}];

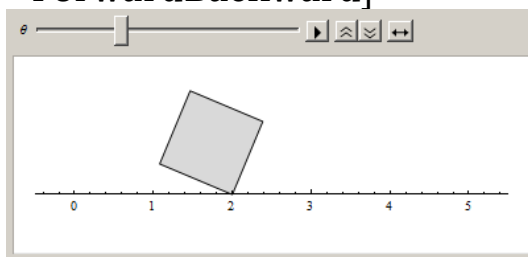
g2 = Graphics[{Thickness[.02], Circle[{0, 0}, 0.8]}];

```
g3 = Graphics[Text[Style["Hello", Large]], {0, 0}];
Show[g1, g2, g3] (* предыдущий рисунок справа *)
```

Реже используемыми двумерными примитивами и связанными с ними функциями являются Arrow, BezierCurve, BSplineCurve, FilledCurve, GraphicsComplex, GraphicsGroup, JoinedCurve, Locator, Raster.

Графические примитивы можно использовать для создания анимации. В следующем примере мы создаем анимацию катящегося по прямой квадрата

```
Animate[With[{q = Quotient[θ, π/2], m = Mod[θ, π/2]},
  Graphics[{EdgeForm[Black], LightGray,
    Rotate[Rectangle[{q, 0}], -m, {q + 1, 0}],
    Axes → {True, False},
    PlotRange → {{-0.5, 5.5}, {-0.5, 1.5}}}],
  {θ, 0, 2Pi}, AnimationRunning → False,
  AnimationDirection → ForwardBackward]
```



Поясним некоторые моменты приведенного кода. Функция `With[{q=expr1, m=expr2}, expr(q, m)]` в выражении `expr(q, m)` выполняет замену идентификаторов `q` и `m` значениями `expr1` и `expr2`. Функции `Quotient[θ, π/2]` вычисляет целую часть от деления первого аргумента θ на второй аргумент $\pi/2$, а `Mod[θ, π/2]` вычисляет остаток деления θ на $\pi/2$. Графическая директива `EdgeForm[Black]` определяет, что контур последующего примитива (закрашенного в светло серый цвет квадрата) должен иметь заданный цвет. Функция `Rectangle[{q, 0}]` создает примитив – квадрат с левой нижней вершиной в точке $\{q, 0\}$ и стороной 1 (по умолчанию). Функция `Rotate[Rectangle[{q, 0}], -m, {q+1, 0}]` поворачивает графический примитив `Rectangle[{q, 0}]` на угол $-m$ вокруг точки с координатами $\{q+1, 0\}$, т.е. вокруг правой нижней вершины квадрата. Положительное направление угла отсчитывается против часовой стрелки, поэтому угол поворота равен $-m$. Опция `AnimationDirection` определяет направление анимации.

Заметим, что функция `Rotate` может поворачивать не только графические объекты. Например в следующем коде мы поворачиваем текст.

```
Rotate["Hello", π/4]
```

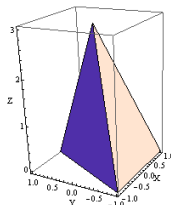
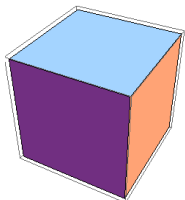
Hello

Функция Graphics3D преобразует трехмерные примитивы в трехмерные графические объекты, добавляя в структуру примитива стили (цвет, толщину и т.д.). Основными трехмерными примитивами являются:

Cuboid[{x1, y1, z1}, ...]	параллелепипед (3D область), определяемый двумя диагонально расположенными вершинами;
Line[{{x1, y1, z1}, ...}]	ломаная в пространстве с вершинами в точках {{x1, y1, z1}, ...};
Point[{x, y, z}]	точка с координатами {x, y, z};
Polygon[{{x1, y1, z1}, ...}]	многогранная двумерная область в пространстве с вершинами в точках {{x1, y1, z1}, ...};
Text["string", {x, y, z}]	текстовая строка, расположенная в точке с координатами {x, y, z};

```
g1 = Graphics3D[Cuboid[{1, 1, 1}]];
```

```
Show[g1] (* следующий рисунок слева *)
```



Можно рисовать сложные фигуры, комбинируя уже имеющиеся примитивы. В следующем примере мы создаем пирамиду с вершинами в точках $(1, -1, 0)$, $(0, 1, 0)$, $(-1, -1, 0)$, $(0, 0, 3)$.

```
piramid = {Polygon[{{1, -1, 0}, {0, 1, 0}, {-1, -1, 0}}],  
            Polygon[{{1, -1, 0}, {0, 0, 3}, {0, 1, 0}}],  
            Polygon[{{0, 1, 0}, {0, 0, 3}, {-1, -1, 0}}],  
            Polygon[{{-1, -1, 0}, {0, 0, 3}, {1, -1, 0}}]};
```

```
Show[Graphics3D[piramid], Axes → Automatic, AxesLabel → {"X", "Y", "Z"}]
```

(предыдущий рисунок справа). Тот же графический объект может быть получен с использованием одной функции Polygon

```
piramid = Polygon[{{{1, -1, 0}, {0, 1, 0}, {-1, -1, 0}},  
                  {{1, -1, 0}, {0, 0, 3}, {0, 1, 0}},  
                  {{0, 1, 0}, {0, 0, 3}, {-1, -1, 0}},  
                  {{-1, -1, 0}, {0, 0, 3}, {1, -1, 0}}]}];
```

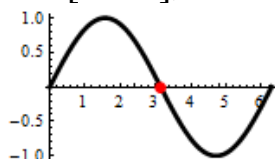
```
Show[Graphics3D[piramid], Axes → Automatic, AxesLabel → {"X", "Y", "Z"}]
```

В случае, когда параметрические уравнения поверхностей неизвестны, примитивы являются самым надежным способом создания графического объекта, представляющего поверхность. Отметим однако, что параметрические уравнения поверхности куба или пирамиды можно построить и мы приводили их ранее.

Графические примитивы можно отображать не только с помощью функции Show. Опции Epilog и Prolog позволяют отображать примитивы

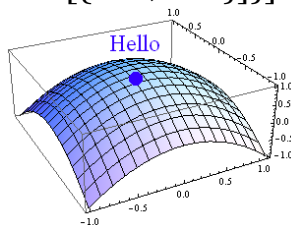
на графиках функций. Опция `Prolog` позволяет отобразить примитив перед рисованием графика, а `Epilog` – после.

```
Plot[Sin[x], {x, 0, 2Pi},
Epilog → {PointSize[0.05], Hue[1], Point[{Pi, Sin[Pi]}]},
PlotStyle → {Thickness[0.02], Black}]
```



На трехмерный график можно нанести двумерный примитив. При этом его положение следует задавать в относительных координатах (0,1).

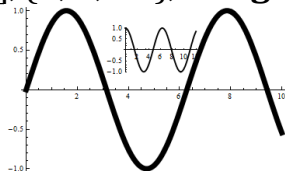
```
Plot3D[1 - x2 - y2, {x, -1, 1}, {y, -1, 1},
Epilog → {PointSize[0.05], Hue[0.7], Text[Style["Hello", Large],
{0.5, 0.75}], Point[{0.5, 0.6}]}
```



Здесь опция `Epilog` отображает два примитива: текстовый и точку. Список {0.5,0.75} определяет положение левой нижней точки текстового примитива, а список {0.5,0.6} определяет положение точки.

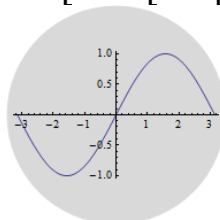
Полезной является функция `Inset`, создающая графический объект, который может быть вставлен в другой графический объект. Ее аргументами являются графический объект, например, график функции, и необязательные опции, управляющие размером вставляемого объекта и местом его расположения в родительском графическом объекте. Функция `Inset` может использоваться везде, где можно использовать графический примитив. Например, в следующем примере мы вставляем один график в другой

```
Plot[Sin[x], {x, 0, 10},
Epilog → Inset[Plot[Cos[x], {x, 0, 12}, ImageSize → 120], {5, 0.5}]]
```



Следующий код рисует графический примитив `Disk[]` с вставленным в него графиком.

```
Graphics[{LightGray, Disk[], Inset[Plot[Sin[x], {x, -π, π}]]}]
```



В следующем примере мы создаем графический объект – диск, в который с помощью функции `Inset` вставлен график функции синус. Этот диск с графиком катится по прямой

```
Animate[Graphics[
  Rotate[{LightGray, Disk[{ $\theta$ , 1}, 1],
  Inset[Plot[Sin[ $x$ ], { $x$ ,  $-\pi$ ,  $\pi$ }], { $\theta$ , 1}, {0, 0}, {2, Automatic}],  $-\theta$ , { $\theta$ , 1}],
  PlotRange  $\rightarrow$  {{-1, 8}, {0, 2}},
{ $\theta$ , 0, 2Pi}, AnimationRunning  $\rightarrow$  False,
AnimationDirection  $\rightarrow$  ForwardBackward]
```



Здесь следует пояснить дополнительные аргументы функции `Inset[Plot[Sin[x], {x, - π , π }], { θ , 1}, {0, 0}, {2, Automatic}]`.

Второй ее аргумент `{ θ , 1}` определяет точку родительского графического объекта, в которую вставляется точка `{0, 0}` дочернего объекта (графика). Последний аргумент `{2, Automatic}` определяет размер дочерней вставки. Задание размера в виде `{w, Automatic}` определяет, что вставляемый объект должен иметь ширину `w` в координатной системы родительского графического объекта.

2.4 Графические иллюстрации к решению прикладных задач

В этом разделе собраны примеры, демонстрирующие возможности графических функций системы *Mathematica* для представления решений прикладных задач. При этом мы не даем подробных постановок задач и, как правило, приводим только решение, графическое представление которого нас интересует. Если «физическая» постановка задачи вам неясна, то вы можете пропустить пример.

2.4.1 Несколько геометрических примеров

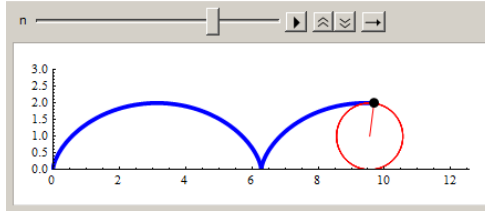
Пример 1.1. Циклоида. Траектория фиксированной точки окружности, которая катится по прямой, называется циклоидой. Если радиус окружности равен 1 и скорость движения ее центра $v=1$, то параметрическое уравнение циклоиды будет иметь вид $x(t)=t-\sin t$, $y(t)=1-\cos t$. Следующий код строит анимацию качения окружности, движение фиксированной точки на окружности, и заметаемую точкой траекторию (циклоиду).

```
Animate[
  p1 = ParametricPlot[{ $t - \text{Sin}[t]$ ,  $1 - \text{Cos}[t]$ }, { $t$ , 0,  $n$ }
  PlotRange  $\rightarrow$  {{0, 4 $\pi$ }, {0, 3}}, PlotStyle  $\rightarrow$  {Blue, Thickness[0.01]};
  p2 = Graphics[{Red, Line[[{ $n - \text{Sin}[n]$ ,  $1 - \text{Cos}[n]$ }, { $n$ , 1}]]];
```

```

p3 = ParametricPlot[{n - Sin[n * t], 1 - Cos[n * t]}, {t, 0, 2π},
PlotStyle -> {Red}];
p4 = Graphics[Disk[{n - Sin[n], 1 - Cos[n]}, 0.15]];
Show[p1, p2, p3, p4],
{n, 1, 4π}, AnimationRunning -> False]

```



Пример 1.2. Лента Мебиуса – простейшая неориентируемая поверхность. Она получается движением и вращением отрезка прямой вдоль замкнутой пространственной кривой. Пусть эта кривая будет окружностью радиуса R , а n обозначает количество полуоборотов отрезка при обходе кривой. Тогда уравнение поверхности можно записать в параметрическом виде

$$x(u, v) = \left(R + v \cos\left(\frac{nu}{2}\right) \right) \cos u$$

$$y(u, v) = \left(R + v \cos\left(\frac{nu}{2}\right) \right) \sin u$$

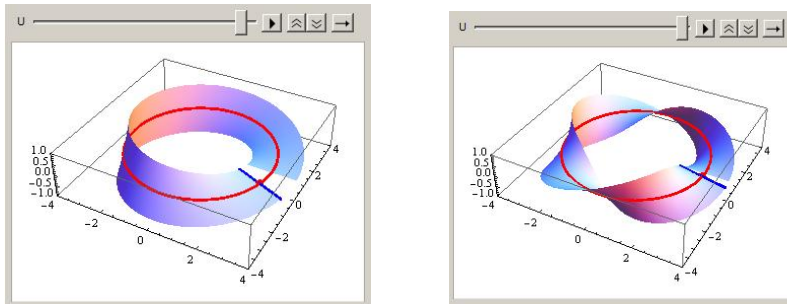
$$z(u, v) = v \sin\left(\frac{nu}{2}\right)$$

Вдоль окружности движется центр отрезка. Следующий код строит анимацию его движения и заматаемую им поверхность (лента Мебиуса).

```

DynamicModule[{R = 3, H = 2, n = 1, x, y, z, po, pt, pc, ps},
x[u_, v_] = (R + vCos[ $\frac{nu}{2}$ ])Cos[u];
y[u_, v_] = (R + vCos[ $\frac{nu}{2}$ ])Sin[u];
z[u_, v_] = vSin[ $\frac{nu}{2}$ ];
po = ParametricPlot3D[{x[u, 0], y[u, 0], z[u, 0]}, {u, 0, 2π},
PlotStyle -> {Red, Thickness[0.01]},
PlotRange -> {{-4, 4}, {-4, 4}, {-1, 1}}];
Animate[
pt = Graphics3D[
{Red, PointSize[0.03], Point[{x[U, 0], y[U, 0], z[U, 0]}]}];
pc = ParametricPlot3D[{x[U, v], y[U, v], z[U, v]}, {v, -H/2, H/2},
PlotStyle -> {Blue, Thickness[0.01]},
PlotRange -> {{-4, 4}, {-4, 4}, {-1, 1}}];
ps = ParametricPlot3D[{x[u, v], y[u, v], z[u, v]}, {u, 0, U}, {v, -H/2, H/2},
AspectRatio -> Automatic, Mesh -> None, PlotPoints -> {50, 15},
PlotRange -> {{-4, 4}, {-4, 4}, {-1, 1}}];
Show[pc, ps, po, pt],
{U, 0.01, 2π}, AnimationRunning -> False]]

```



Здесь n обозначает ширину ленты (длину отрезка). При нечетном n получается неориентированная поверхность. На предыдущем рисунке слева показана лента Мебиуса, построенная при $n=1$, а справа – при $n=3$.

Заметим, что картинка ленты Мебиуса имеется в наборе примеров геометрических поверхностей, которую можно построить командой `ExampleData[{"Geometry3D", "MoebiusStrip"}]` □

Пример 1.3. Бутылка Клейна из лоскутов поверхностей.

Составим поверхность бутылки Клейна из 4-х лоскутов/поверхностей, параметрические уравнения которых достаточно просты.

Строим первый лоскут (следующий рисунок *a*)

$a = 2.5; b = 1.5;$

$x1[u, v] = (a + b \cos[u]) \cos[v];$

$y1[u, v] = (a + b \cos[u]) \sin[v];$

$z1[u, v] = -a \sin[u];$

$p1 = \text{ParametricPlot3D}[\{x1[u, v], y1[u, v], z1[u, v]\}, \{u, 0, \pi\}, \{v, 0, 2\pi\}, \text{PlotStyle} \rightarrow \text{Opacity}[0.5], \text{Mesh} \rightarrow \text{None}]$

Строим второй лоскут (следующий рисунок *b*)

$x2[u, v] = (a + b \cos[u]) \cos[v];$

$y2[u, v] = (a + b \cos[u]) \sin[v];$

$z2[u, v] = 3u;$

$p2 = \text{ParametricPlot3D}[\{x2[u, v], y2[u, v], z2[u, v]\}, \{u, 0, \pi\}, \{v, 0, 2\pi\}, \text{PlotStyle} \rightarrow \text{Opacity}[0.5], \text{Mesh} \rightarrow \text{None}]$

Строим третий лоскут (следующий рисунок *c*)

$x3[u, v] = 2 - 2 \cos[u] + \sin[v];$

$y3[u, v] = \cos[v];$

$z3[u, v] = 3u;$

$p3 = \text{ParametricPlot3D}[\{x3[u, v], y3[u, v], z3[u, v]\}, \{u, 0, \pi\}, \{v, 0, 2\pi\}, \text{PlotStyle} \rightarrow \text{Opacity}[0.5], \text{Mesh} \rightarrow \text{None}]$

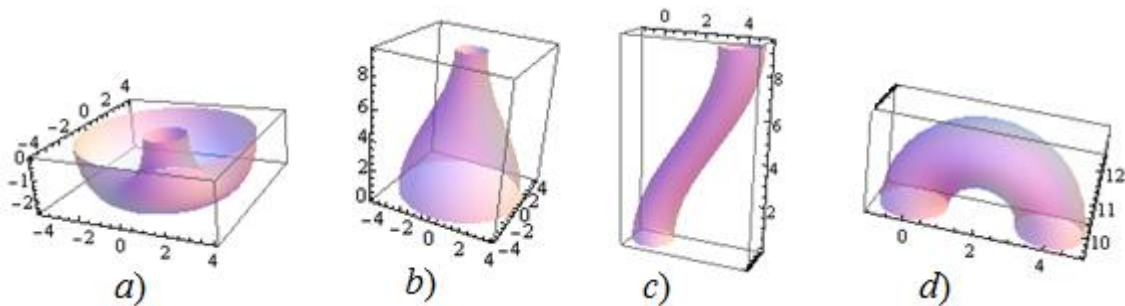
Строим четвертый лоскут (следующий рисунок *d*)

$x4[u, v] = 2 + (2 + \cos[v]) \cos[u];$

$y4[u, v] = \sin[v];$

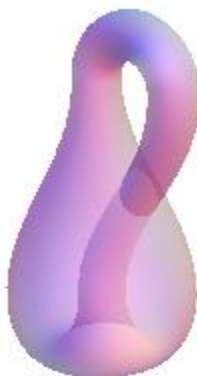
$z4[u, v] = 3\pi + (2 + \cos[v]) \sin[u];$

$p4 = \text{ParametricPlot3D}[\{x4[u, v], y4[u, v], z4[u, v]\}, \{u, 0, \pi\}, \{v, 0, 2\pi\}, \text{PlotStyle} \rightarrow \text{Opacity}[0.5], \text{Mesh} \rightarrow \text{None}]$



Теперь рисуем все сегменты/куски поверхности вместе.

Show[p1, p2, p3, p4, PlotRange → All, Boxed → False, Axes → None]



Пример 1.4. Бутылка Клейна. Известны примеры параметрических уравнений поверхностей, которые представляют бутылку Клейна. Здесь мы используем одно из таких уравнений.

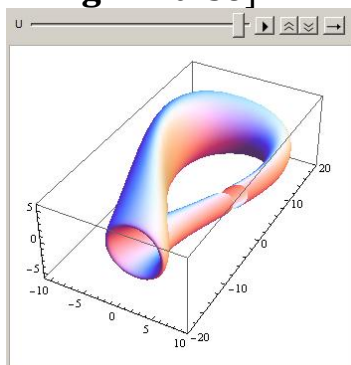
Animate[ParametricPlot3D[

$$\{6\cos[u](1 + \sin[u]) + 4\left(1 - \frac{\cos[u]}{2}\right)\cos\left[\frac{u}{2} + \frac{\pi}{4}\right]\cos[v],$$

$$16\sin[u] + 4\left(1 - \frac{\cos[u]}{2}\right)\sin\left[\frac{u}{2} + \frac{\pi}{4}\right]\cos[v], 4\left(1 - \frac{\cos[u]}{2}\right)\sin[v]\}$$

$$\{u, 0, U\}, \{v, 0, 2\pi\}, \text{PlotRange} \rightarrow \{\{-11, 10\}, \{-20, 20\}, \{-6, 6\}\},$$

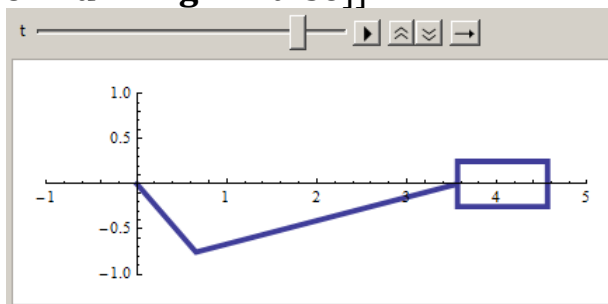
$$\text{PlotPoints} \rightarrow 30, \text{Mesh} \rightarrow \text{None}, \text{ImageSize} \rightarrow 280],$$

$$\{U, 0.01, 2\pi\}, \text{AnimationRunning} \rightarrow \text{False}]$$


2.4.2 Моделирование механических движений

Пример 2.1. Кривошипно – шатунный механизм, является устройством, которое вращательное движение преобразует в поступательное или наоборот. Смоделируем схематически движение этого механизма, изобразив его в виде ломаной.

```
Clear[xl, yl, pt]; w = 1;
xll = {0, Cos[wt],  $\sqrt{9 - \text{Sin}[wt]^2}$ , 0, 1, 0, -1, 0};
yll = {0, Sin[wt], -Sin[wt], -1/4, 0, 1/2, 0, -1/4};
xl = Accumulate[xll];
yl = Accumulate[yll];
DynamicModule[{pt},
  pt[t_] = Transpose[{xl, yl}];
  Animate[ListLinePlot[pt[t],
    AspectRatio  $\rightarrow$  Automatic, PlotRange  $\rightarrow$  {{-1, 5}, {-1, 1}},
    PlotStyle  $\rightarrow$  Thickness[0.01]],
    {t, 0, 2 $\pi$ }, AnimationRunning  $\rightarrow$  False]
```



Здесь списки **xll** и **yll** содержат длины проекций отрезков ломаной, представляющей механизм, на оси X и Y соответственно. Функция `Accumulate[List]` возвращает список накопительных сумм элементов списка `List`. Например,

```
Accumulate[{a, b, c, d}]
{a, a + b, a + b + c, a + b + c + d}
```

В результате списки **xl** и **yl** содержат *x* и *y* координаты узлов ломаной. Из них создается список **pt[t]** пар координат $\{\dots, \{x_i, y_i\}, \dots\}$ узлов, которые используются функцией `ListLinePlot` для рисования ломаной.

Пример 2.2. Поперечные колебания невесомой струны с грузами.

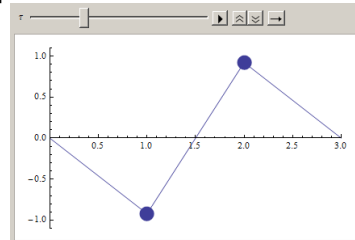
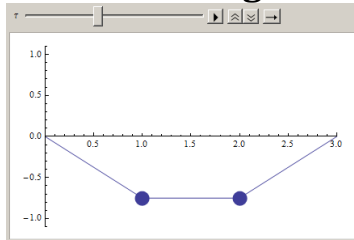
Рассмотрим колебания механической системы, которая состоит из двух частиц, каждая массой *M*, закрепленных на невесомой упругой струне. Массы разбивают струну на три участка одинаковой длины. Через y_1 обозначим вертикальное смещение 1-й массы и через y_2 – смещение 2-й массы. Известно, что колебание системы может быть представлено в виде [1]

$$\begin{aligned} y_1(t) &= A_1 \cos(\omega_1 t + \varphi_1) + A_2 \cos(\omega_2 t + \varphi_2), \\ y_2(t) &= A_1 \cos(\omega_1 t + \varphi_1) - A_2 \cos(\omega_2 t + \varphi_2), \end{aligned}$$

где значения постоянных определяется из начальных условий и механических характеристик системы.

Рассмотрим колебание системы без начальной скорости (начальные скорости частиц равны нулю). Длина струны равна 3. Колебание, при котором массы двигаются синхронно, называется первой модой. Следующий код моделирует первую моду колебаний (следующий рисунок слева).

```
DynamicModule[{w1 = 1, y1, y2, ptpt, pt},
  y1[t_] = Cos[w1 t];
  y2[t_] = Cos[w1 t];
  Animate[
    ptpt = {{1, y1[t]}, {2, y2[t]}};
    pt = {{0, 0}, {1, y1[t]}, {2, y2[t]}, {3, 0}};
    p1 = ListLinePlot[pt/. t -> τ, PlotRange -> {{0, 3}, {-1.1, 1.1}}];
    p2 = ListPlot[ptpt/. t -> τ, PlotStyle -> PointSize[0.05]];
    Show[p1, p2],
    {τ, 0, 6}, AnimationRunning -> False]]
```



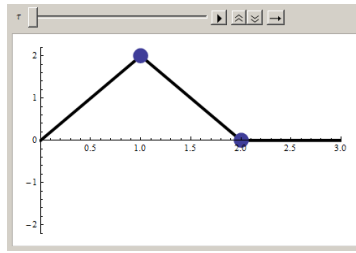
Вторая мода колебаний представляет движение частиц в противофазе. Чтобы построить такое движение в предыдущем коде мы должны изменить знак у функции y_2 и, возможно, изменить угловую частоту. Первые три строки предыдущего кода примут новый вид

```
DynamicModule[{w2 = √3, y1, y2, ptpt, pt},
  y1[t_] = Cos[w2 t];
  y2[t_] = -Cos[w2 t];
  ...
]
```

Остальные строки кода остаются без изменений. В результате получается движение, представленное в панели анимации предыдущего рисунка справа.

Смешанное колебание состоит из суперпозиции обеих мод, вклад каждой из которых определяется начальным положением масс системы. Построим колебание, начальное положение масс которого показано на следующем рисунке. Для этого изменим первые три строки кода следующим образом

```
DynamicModule[{w1 = 1, w2 = √3, y1, y2, ptpt, pt},
  y1[t_] = Cos[w1 t] + Cos[w2 t];
  y2[t_] = Cos[w1 t] - Cos[w2 t];
  ...
]
```



Пример 2.3. Математический бильярд в прямоугольнике.

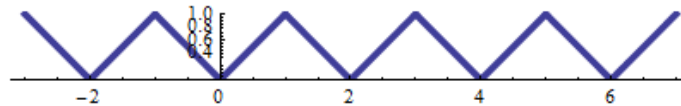
Имеется горизонтальный бильярдный стол произвольной формы без луз (плоская область). По этому столу без трения движется точечный шар, упруго отражаясь от бортов. Требуется определить траекторию этого шара/точки. Такая механическая система – точечный шар в области Q, ограниченной бортом Г (границей области Q) – называется математическим бильярдом. Траектория точки в математическом бильярде определяется ее начальным положением и начальным вектором скорости.

Введем вспомогательную функция $stc(x, w)$, которая будет обозначать пилообразную непрерывную (saw tooth continuous) кусочно-линейную функцию, определяемую, например, уравнением

$$stc(x, w) = \left| x - w \left[\frac{x}{w} + \frac{1}{2} \right] \right| \quad \text{или} \quad stc(x, w) = \frac{w}{2\pi} \arccos \cos \frac{2\pi x}{w},$$

где $[z]$ (квадратные скобки) является функцией взятия наибольшего целого, не превосходящего z .

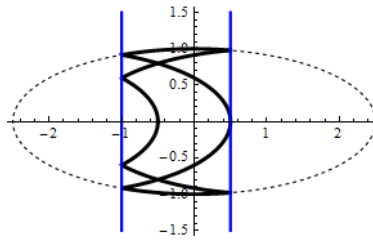
```
Stc[x_, w_] = Abs[x - w Floor[x/w + 1/2]];
Plot[Stc[x, 2], {x, -3, 7}, , AspectRatio -> Automatic]
```



Приведем без доказательства некоторые факты, связанные с преобразованием отражения кривых.

Пусть кривая C задана параметрическими уравнениями $x = x(t)$, $y = y(t)$. Уравнение кривой C_t , полученной многократным внутренним отражением от пары вертикальных прямых $x = x_l, x = x_r$ ($x_l < x_r$), имеет вид $x_t = x_l + stc(x(t) - x_l, 2(x_r - x_l))$, $y_t = y(t)$. Вот пример кривой, полученной отражением эллипса от двух вертикальных прямых.

```
xl = -1; xr = 1/2;
xt[t_] = 2.5 Cos[t];
yt[t_] = Sin[t];
xt[t_] = xl + Stc[x[t] - xl, 2(xr - xl)];
yt[t_] = y[t];
pt = ParametricPlot[{xt[t], yt[t]}, {t, 0, 2π}];
pe = ParametricPlot[{x[t], y[t]}, {t, 0, 2π}];
pl = Graphics[{Blue, Thick,
  {Line[{{xl, -1.5}, {xl, 1.5}}], Line[{{xr, -1.5}, {xr, 1.5}}]}}];
Show[pe, pt, pl, PlotRange -> All]
```



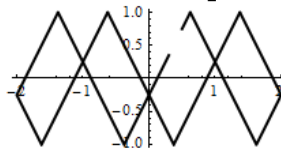
Аналогично, уравнение кривой C_t , полученной многократным внутренним отражением C от пары горизонтальных прямых $y = y_d, y = y_u$ ($y_d < y_u$) имеет вид $x_t = x(t), y_t = y_d + stc(y(t) - y_d, 2(y_u - y_d))$.

Параметрическое уравнение кривой C_t , полученной многократным внутренним отражением C от границ прямоугольника $x_l \leq x \leq x_r, y_d \leq y \leq y_u$, имеет вид

$$\begin{aligned} x_t(t) &= x_l + stc(x(t) - x_l, 2(x_r - x_l)) \\ y_t(t) &= y_d + stc(y(t) - y_d, 2(y_u - y_d)) \end{aligned} \quad (1)$$

Вот пример уравнений ломаной, полученной многократным внутренним отражением прямой/луча от границ прямоугольника $-2 \leq x \leq 2, -1 \leq y \leq 1$. Луч стартует в точке $(0.5, 0.75)$ и имеет направляющий вектор $(1, 2)$.

```
xl = -2; xr = 2; yd = -1; yu = 1;
x[t_] = 0.5 + t;
y[t_] = 0.75 + 2t;
xt[t_] = xl + Stc[x[t] - xl, 2(xr - xl)];
yt[t_] = yd + Stc[y[t] - yd, 2(yu - yd)];
ParametricPlot[{xt[t], yt[t]}, {t, 0, 7.8},
PlotStyle -> {Black, Thickness[0.01]}]
```



Траектория точки в прямоугольном бильярде является ломаной, получаемой многократным отражением луча от «бортов» прямоугольника. Пусть из точки (x_0, y_0) вылетает бесконечно малый шар в направлении вектора $(\cos \alpha, \sin \alpha)$. Трение отсутствует. Если бы бортов области не было, то он двигался бы вдоль прямой с уравнением $x = x_0 + ct \cos \alpha, y = y_0 + ct \sin \alpha$, где t – время, c – скорость, α – угол наклона луча. Уравнение бильярдной траектории будет иметь вид (1), где в качестве функций $x(t), y(t)$ следует использовать уравнение этой прямой. Следующий код моделирует движение точки в прямоугольном бильярде.

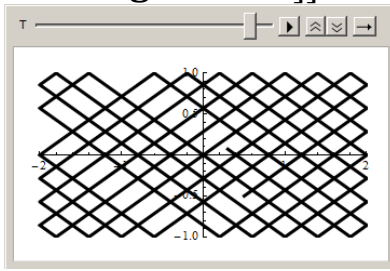
```
DynamicModule[{xl = -2, xr = 2, yd = -1, yu = 1,
  x0 = 0.5, y0 = -0.5, alpha = ArcTan[5/7], x, y, xt, yt},
  x[t_] = x0 + t Cos[alpha];
  y[t_] = y0 + t Sin[alpha];
  xt[t_] = xl + Stc[x[t] - xl, 2(xr - xl)];
  yt[t_] = yd + Stc[y[t] - yd, 2(yu - yd)];
```



```

Animate[ParametricPlot[{xt[t], yt[t]], {t, 0, T},
PlotStyle → {Black, Thickness[0.01]},
PlotRange → {{xl, xr}, {yd, yu}}, PlotPoints → 200],
{T, 0.01, 68}, AnimationRunning → False]]

```



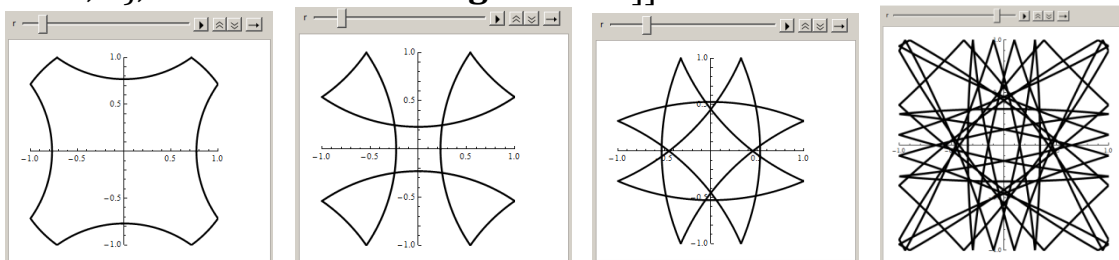
Пример 2.4. Отражение кругового волнового фронта.

Формулы отражения (1) кривой $x = x(t)$, $y = y(t)$ от «бортов» прямоугольника можно использовать для кривой любой формы (не только прямой). Например, уравнение кругового волнового фронта распространяющегося из некоторой точки и отражающегося от «берегов» прямоугольника, тоже будут преобразовываться в соответствии с формулами (1). В следующем примере мы строим анимацию распространения и отражения кругового волнового фронта в квадратной области. Начальная точка расположена в центре.

```

DynamicModule[{xl = -1, xr = 1, yd = -1, yu = 1, x, y, xt, yt},
x[r_] = rCos[a];
y[r_] = rSin[a];
xt[r_] = xl + Stc[x[r] - xl, 2(xr - xl)];
yt[r_] = yd + Stc[y[r] - yd, 2(yu - yd)];
Animate[ParametricPlot[{xt[r], yt[r]], {a, 0, 2π},
PlotStyle → {Black, Thickness[0.01]},
PlotRange → {{xl, xr}, {yd, yu}}, PlotPoints → 200],
{r, 0.01, 6}, AnimationRunning → False]]

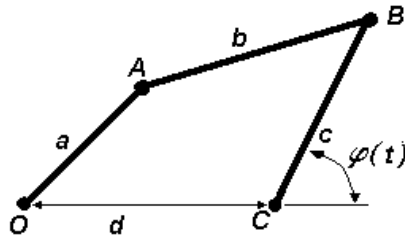
```



Несколько кадров волнового фронта в последовательные моменты времени представлены на предыдущем рисунке.

Пример 2.5. Моделирование кривых с помощью шарнирных механизмов

Смоделируем движение шарнирного механизма, состоящего из трех звеньев. Ведущее звено OA вращается вокруг точки O с постоянной угловой скоростью, звенья OA , AB и BC шарнирно соединены в точках A и B . Звено BC также может совершать свободное вращение вокруг точки C . Обозначим длины звеньев через a , b , c так, как показано на рисунке и расстояние по горизонтали между точками O и C обозначим через d .



Чтобы точка А могла пройти горизонтальное положение слева от точки О необходимо, чтобы выполнялось соотношение $a+d \leq b+c$.

Очевидно, что точка В будет двигаться по дуге окружности вокруг точки С, однако закон этого движения нам неизвестен. Обозначим угол, который образует отрезок СВ с осью Х через $\varphi(t)$. Тогда

$$\begin{cases} x_A = a \cos t \\ y_A = a \sin t \end{cases} \quad \begin{cases} x_B = d + c \cos \varphi(t) \\ y_B = c \sin \varphi(t) \end{cases}$$

Длина отрезка АВ равна $(x_A - x_B)^2 + (y_A - y_B)^2 = b^2$. Подставим сюда выражения координат точек А и В из предыдущих соотношений и получим уравнение для определения неизвестной функции $\varphi(t)$.

$$(d + c \cos \varphi(t) - a \cos t)^2 + (c \sin \varphi(t) - a \sin t)^2 = b^2.$$

После преобразований получаем $\alpha \cos \varphi(t) - \beta \sin \varphi(t) = \gamma$, где

$$\alpha = d - a \cos t, \quad \beta = a \sin t, \quad \gamma = \frac{b^2 - a^2 - c^2 - d^2 + 2ad \cos t}{2c}$$

Разделим обе части последнего уравнения на $\sqrt{\alpha^2 + \beta^2}$ и обозначим

$$\sin \delta = \frac{\alpha}{\sqrt{\alpha^2 + \beta^2}}, \quad \cos \delta = \frac{\beta}{\sqrt{\alpha^2 + \beta^2}}$$

Тогда уравнение относительно $\varphi(t)$ может быть записано в виде $\sin(\delta - \varphi(t)) = \frac{\gamma}{\sqrt{\alpha^2 + \beta^2}}$ или $\varphi(t) = \delta(t) - \arcsin \frac{\gamma}{\sqrt{\alpha^2 + \beta^2}}$. Учитывая, что имеет

место равенство $\alpha^2 + \beta^2 = d^2 - 2ad \cos t + a^2$, после преобразований получаем

$$\varphi(t) = \delta(t) - \zeta(t),$$

где

$$\delta(t) = \arccos \frac{a \sin t}{\sqrt{d^2 - 2ad \cos t + a^2}} \quad \text{и} \quad \zeta(t) = \arcsin \frac{b^2 - a^2 - c^2 - d^2 + 2ad \cos t}{2c \sqrt{d^2 - 2ad \cos t + a^2}}.$$

Следующий код моделирует движение трехзвенного механизма

DynamicModule[{**a** = 2, **b** = 3, **c** = 3, **d** = 3.1, **delta**, **zeta**, **phi**, **ln**, **pw**, **pc**, **pts**},

$$\delta[t_] = \text{ArcCos} \left[\frac{a \text{Sin}[t]}{\sqrt{a^2 + d^2 - 2ad \text{Cos}[t]}} \right];$$

$$\zeta[t_] = \text{ArcSin} \left[\frac{b^2 + 2ad \text{Cos}[t] - a^2 - c^2 - d^2}{2c \sqrt{a^2 + d^2 - 2ad \text{Cos}[t]}} \right];$$

$$\varphi[t_] = \delta[t] - \zeta[t];$$

Manipulate[

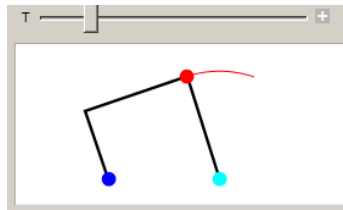
```
In = Line[{{0, 0}, {aCos[t], aSin[t]}, {d + cCos[φ[t]], cSin[φ[t]]}, {d, 0}}];
```

```
pts = Graphics[{{Blue, Disk[{0, 0}, 0.2]},
               {Red, Disk[{d + cCos[φ[T]], cSin[φ[T]]}, 0.2]},
               {Cyan, Disk[{d, 0}, 0.2]}}];
```

```
pw = Graphics[{Thickness[0.01], Black, In /. t -> T}];
```

```
pc = ParametricPlot[{d + cCos[φ[t]], cSin[φ[t]]},
                   {t, π/4 + 0.001, T}, PlotStyle -> Red];
```

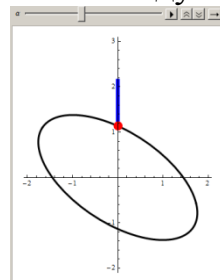
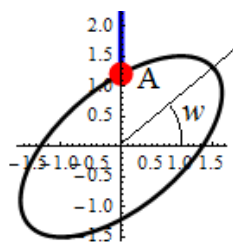
```
Show[pw, pc, pts, PlotRange -> {{-a, c + d}, {-c - 0.2, c + 0.2}},
     {T, π/4, 2π + π/4}]]
```



Здесь мы использовали панель манипулятора, а не панель анимации, как в предыдущих примерах.

Пример 2.6. Кулачковый механизм. Эксцентрик в форме эллипса совершает вращательное движение и взаимодействует с кулачком в форме прямолинейного стержня. Смоделируем движение такого механизма, используя графические функции системы.

Пусть полуоси эллипса равны a и b , а длина кулачка/стержня L . В следующем коде функции $x_e(t, w)$, $y_e(t, w)$ при фиксированном w представляют параметрическое уравнение повернутого на угол w эллипса. Функция $y_A(w)$ представляет y -координату толкателя (точка А на следующем рисунке слева).



```
DynamicModule[{a = 2, b = 1, L = 1, xe, ye, yA, ys},
```

```
xe[t_, w_] = aCos[t]Cos[w] - bSin[t]Sin[w];
```

```
ye[t_, w_] = aCos[t]Sin[w] + bSin[t]Cos[w];
```

$$yA[w_] = \frac{a b}{\sqrt{a^2 \cos^2[w] + b^2 \sin^2[w]}};$$

```
ys[t_, w_] = yA[w] + t;
```

```
Animate[
```

```
p1 = ParametricPlot[{xe[t, α], ye[t, α]}, {t, 0, 2π},
```

```
PlotStyle -> {Black, Thickness[0.01]};
```

```

p2 = ParametricPlot[{0, ys[t,  $\alpha$ ]}, {t, 0, L},
    PlotStyle  $\rightarrow$  {Blue, Thickness[0.02]};
pt = Graphics[{Red, Disk[{0, yA[ $\alpha$ ]}, 0.1]};
Show[p1, p2, pt, PlotRange  $\rightarrow$  {{-a, a}, {-a, a + 1}},
    { $\alpha$ , 0, 2 $\pi$ }, AnimationRunning  $\rightarrow$  False]]

```

Панель анимации со схематическим изображением кулачкового механизма показана на предыдущем рисунке справа.

Параметрическое уравнение повернутого эллипса $x_e(t, w)$, $y_e(t, w)$ получается умножением вектора $(a \cos t, b \sin t)$ на матрицу поворота на угол w . Для определения y координаты точки А мы определяем значение параметра t в точке А на повернутом эллипсе. Для этого приравняем нулю функцию $x_e(t, w)$ и выражаем t через w (точнее выражаем $\tan t$). Получаем $\tan t = \frac{a \cos w}{b \sin w}$.

Подстановка этого значения в выражение $y_e(t, w)$ после упрощений дает функцию $y_A(w)$.

2.4.3 Движение жидкости в трубах

Смоделируем поле скоростей жидкости в цилиндрической трубе с эллиптическим сечением. Скорость жидкости w направлена вдоль оси трубы (ось X), одинакова во всех поперечных сечениях, и в точке сечения с координатами (y, z) определяется формулой

$$w(y, z) = \frac{Ca^2b^2}{a^2 + b^2} \left(1 - \frac{y^2}{a^2} - \frac{z^2}{b^2} \right),$$

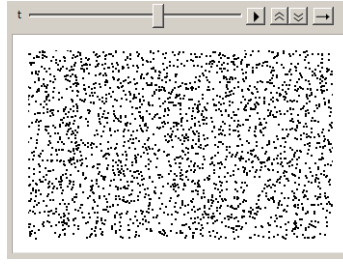
где a и b являются длинами полуосей эллипса/сечения, C – некоторая постоянная, зависящая от разности давлений на концах трубы и вязкости жидкости.

Пример 3.1. Смоделируем движение жидкости как движение частиц в продольном сечении $z=0$ трубы. Ось X направлена вдоль оси трубы вправо, ось Y – вертикально вверх. Положим $a=1$, $b=1$, $C=2$. Сгенерируем большое количество случайно расположенных в продольном сечении точек и, в зависимости от их вертикальной координаты y , зададим им скорость движения $w(y) = 1 - y^2$.

```

DynamicModule[{n = 5000, h = 6, w, xl, yl, pt},
    w[y_] = 1 - y^2;
    xl = RandomReal[{-h, h}, n];
    yl = RandomReal[{-1, 1}, n];
    Animate[
        pt = Transpose[{xl + w[yl]t, yl}];
        ListPlot[pt, PlotStyle  $\rightarrow$  {Black, PointSize[0.005]},
            PlotRange  $\rightarrow$  {{0, h}, {-1, 1}}, Axes  $\rightarrow$  None],
        {t, 0, h, 0.01}, AnimationRunning  $\rightarrow$  False]]

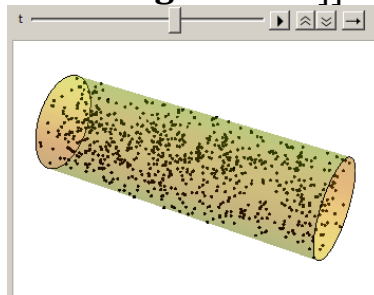
```



Отметим, что приведенное здесь статическое изображение, не передает процесс движения в отличие от динамического движения, воспроизводимого в панели анимации.

Пример 3.2. Смоделируем пространственное движение жидкости как движение частиц в круглой трубе. Ось X направлена вдоль оси трубы вправо. Положим $a = 1$, $b = 1$, $C = 2$. Сгенерируем большое количество случайно расположенных в трубе точек и, в зависимости от их поперечных координат y и z , зададим им скорость движения $w(y, z) = 1 - y^2 - z^2$.

```
DynamicModule[{n = 2000, h = 6, w, xl, yl, zl, pt},
  w[y_, z_] = 1 - y^2 - z^2;
  xl = RandomReal[{-h, h}, n];
  yl = RandomReal[{-1, 1}, n];
  zl = RandomReal[{-1, 1}, n];
  Animate[
    pt = Transpose[{xl + w[yl, zl] t, yl, zl}];
    pl = ListPointPlot3D[pt, PlotStyle -> {Black, PointSize[0.005]},
      PlotRange -> {{0, h}, {-1, 1}, {-1, 1}},
      RegionFunction -> Function[{x, y, z}, y^2 + z^2 < 1];
    pr = RegionPlot3D[y^2 + z^2 < 1, {x, 0, h}, {y, -1, 1}, {z, -1, 1},
      PlotStyle -> Directive[Yellow, Opacity[0.3]], Mesh -> None];
    Show[pl, pr, BoxRatios -> Automatic, Boxed -> False, Axes -> None],
    {t, 0, h, 0.001}, AnimationRunning -> False]]
```



Особенностью этого примера является создание полупрозрачной трехмерной области **pr**, представляющей участок трубы, и использование у функции `ListPointPlot3D` опции `RegionFunction`, отсекающей изображения точек, выходящие за пределы кругового сечения трубы.

□

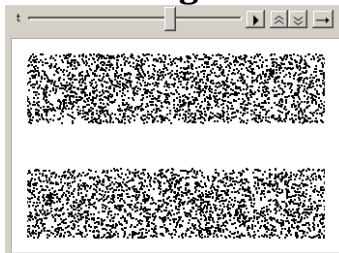
Поле скоростей жидкости в области между двумя коаксиальными круглыми цилиндрами радиусов r_1 и r_2 ($r_2 > r_1$) определяется по формуле

$$w(r) = C \left(r_1^2 - r^2 + \frac{(r_2^2 - r_1^2) \ln(r/r_1)}{\ln(r_2/r_1)} \right),$$

где r – расстояние точки сечения до оси и C – некоторая постоянная, зависящая от вязкости жидкости и разности давлений на входе и выходе трубы.

Пример 3.3. Смоделируем движение жидкости в продольном сечении $z=0$ такой трубы. Ось X направим вдоль оси трубы вправо и положим $r_1 = 0.5$, $r_2 = 2$, $C = 1$. Сгенерируем большое количество случайно расположенных в продольном сечении точек и, в зависимости от их расстояния r до оси, зададим им скорость движения $w(r)$.

```
DynamicModule[{n = 5000, h = 6, r1 = 0.5, r2 = 2, w, xl, yl1, yl2, pt},
  w[r_] = r1^2 - r^2 + (r2^2 - r1^2)Log[r/r1]/Log[r2/r1];
  xl = RandomReal[{-h, h}, n];
  yl1 = RandomReal[{r1, r2}, n];
  yl2 = RandomReal[{-r2, -r1}, n];
  Animate[
    pt = Join[Transpose[{xl + w[Abs[yl1]]t, yl1}],
      Transpose[{xl + w[Abs[yl2]]t, yl2}]];
    ListPlot[pt, PlotStyle -> {Black, PointSize[0.005]},
      PlotRange -> {{0, h}, {-2, 2}}, Axes -> None],
    {t, 0, h - 1, 0.01}, AnimationRunning -> False]]
```



В этом примере мы создаем два случайных числовых множества $yl1$ и $yl2$, которые затем используются при создании единого списка pt пар координат точек.

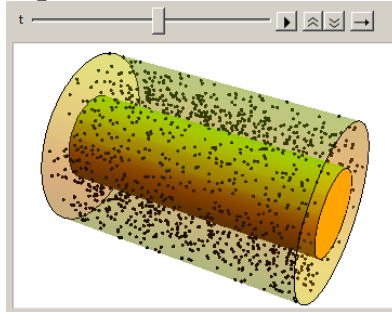
Пример 3.4. Смоделируем пространственное движение жидкости в области между двумя коаксиальными круглыми цилиндрами. Ось X направим вдоль оси цилиндров и положим $r_1 = 1$, $r_2 = 2$, $C = 1$. Сгенерируем большое количество случайно расположенных в трубе точек и, в зависимости от их расстояния r до оси, зададим им скорость движения $w(r)$.

```
DynamicModule[{n = 3000, h = 6, r1 = 1, r2 = 2, w, rl, phi, xl, yl, zl, pt},
  w[r_] = r1^2 - r^2 + (r2^2 - r1^2)Log[r/r1]/Log[r2/r1];
  xl = RandomReal[{-h, h}, n];
  rl = RandomReal[{r1, r2}, n];
  phi = RandomReal[{0, 2π}, n];
  yl = rl Cos[phi];
  zl = rl Sin[phi];
```

```

Animate[
  pt = Transpose[{xl + w[rl] t, yl, zl}];
  pl = ListPointPlot3D[pt, PlotStyle -> {Black, PointSize[0.005]},
    PlotRange -> {{0, h}, {-r2, r2}, {-r2, r2}},
    RegionFunction -> Function[{x, y, z},
      y2 + z2 > r12 && y2 + z2 < r22]];
  pr1 = RegionPlot3D[y2 + z2 < r22, {x, 0, h}, {y, -r2, r2}, {z, -r2, r2},
    PlotStyle -> Directive[Yellow, Opacity[0.3]], Mesh -> None];
  pr2 = RegionPlot3D[y2 + z2 < r12, {x, 0, h}, {y, -r2, r2}, {z, -r2, r2},
    PlotStyle -> Directive[Yellow, Opacity[1]], Mesh -> None];
  Show[pl, pr1, pr2, BoxRatios -> Automatic, Boxed -> False,
    Axes -> None],
  {t, 0, h, 0.1}, AnimationRunning -> False,
  DisplayAllSteps -> True, AnimationRate -> 2]]

```



В этом примере мы создаем полупрозрачный внешний цилиндр и непрозрачный внутренний. Координаты случайных точек создаются путем генерирования множества rl случайных радиусов в диапазоне от $r1$ до $r2$ и множества ϕl случайных полярных углов, которые затем используются для создания списка pt координат $\{\dots, \{x_i, y_i, z_i\}, \dots\}$ точек.

2.4.4 Двумерные волновые движения

Пример 4.1. Поверхностные волны Релея. Английский физик Релей показал, что в упругом полупространстве могут существовать специальные волны, которые он назвал поверхностными. У таких волн амплитуда колебаний быстро уменьшается с удалением от поверхности. Одно из таких плоских волновых движений может быть описано с помощью следующих функций [9]

$$U(x, y, t) = C(-e^{-\alpha y} + c e^{-\beta y}) \sin(\xi x + pt) \quad (1)$$

$$V(x, y, t) = C a(-e^{-\alpha y} - d e^{-\beta y}) \cos(\xi x + pt)$$

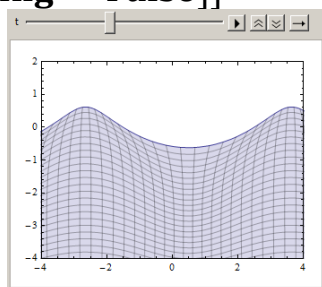
где U и V представляют продольное и поперечное смещение точек среды (x, y) относительно положения равновесия. Здесь $\alpha = a\xi$, $\beta = b\xi$ и a, b, c, d некоторые постоянные, зависящие от свойств среды, C – произвольная постоянная. Ось Y направлена в глубину среды.

Направим ось Y вертикально вверх (у системы *Mathematica* нет простого способа менять направления осей). В силу этого в примере в формулах Релея (1) мы изменим знак перед переменной y и перед функцией V .

Полупространство в состоянии покоя будем моделировать в виде области, заданной в параметрическом виде $x(u, v) = u$, $y(u, v) = v$, где $-\infty < u < \infty, v \leq 0$. Тогда смещенные точки области в момент времени t будут иметь координаты $X = x(u, v) + U(x(u, v), y(u, v), t)$, $Y = y(u, v) + V(x(u, v), y(u, v), t)$, где $v \leq 0$.

Следующий код моделирует движение среды – поверхностную волну Релея.

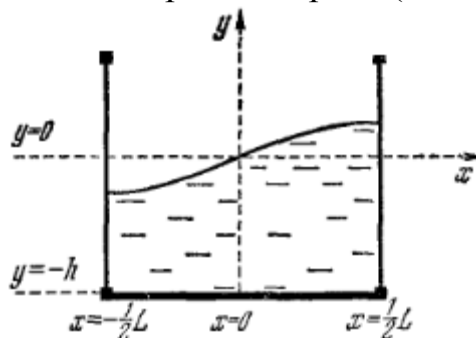
```
DynamicModule[{a = 0.8475, b = 0.3933,
               c = 0.5753, d = 1.732, p = 1, Cc = 1, U, V, X, Y},
  U[x_, y_, t_] = Cc(-Exp[-ay] + cExp[-by])Sin[x + pt];
  V[x_, y_, t_] = Cca(Exp[-ay] - dExp[-by])Cos[x + pt];
  X[u_, v_, t_] = u + U[u, -v, t];
  Y[u_, v_, t_] = v - V[u, -v, t];
  Animate[ParametricPlot[
    {{X[u, v, t], Y[u, v, t]}}, {u, -5, 5}, {v, -5, 2}, Mesh -> {31, 26},
    RegionFunction -> Function[{x, y, u, v}, v < 0],
    PlotRange -> {{-4, 4}, {-4, 2}}, Axes -> None],
    {t, 0, 2π}, AnimationRunning -> False]]
```



Использование параметрического уравнения области в этом примере позволяет показать не только форму поверхности, но и образ декартовой координатной сетки внутри среды.

Пример 4.2. Установившаяся синусоидальная волна в прямоугольном аквариуме постоянной глубины [1].

Рассмотрим прямоугольный аквариум шириной L с жидкостью налитой до уровня h . Направим ось Y вертикально вверх с началом на свободной поверхности жидкости, а ось X направим вправо (см. следующий рисунок).



Одно из возможных установившихся волновых движений жидкости может быть представлено в виде

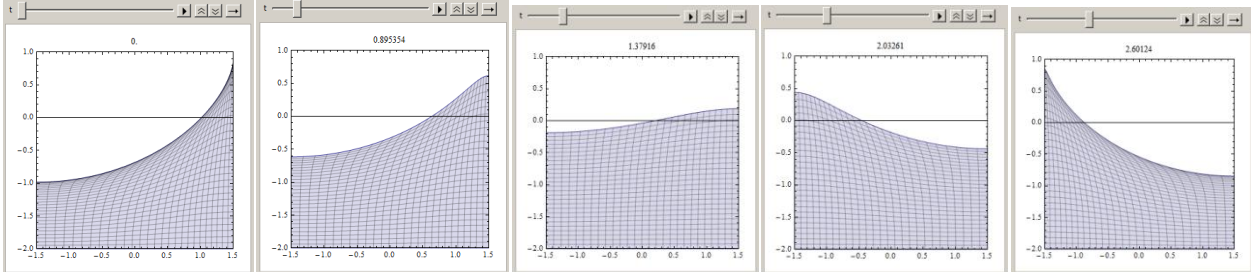
$$\begin{aligned} U &= a \cos(\omega t) \cos(k x) (e^{k y} + e^{-2kh} e^{-k y}) \\ V &= a \cos(\omega t) \sin(k x) (e^{k y} - e^{-2kh} e^{-k y}) \end{aligned} \quad (2)$$

где $k = \pi/L$ и a – произвольная постоянная. Функции $U(x, y, t)$ и $V(x, y, t)$ дают мгновенные значения смещений частицы жидкости с равновесными координатами x, y в направлениях осей X и Y .

Жидкость в состоянии покоя будем моделировать в виде плоской области, заданной в параметрическом виде $x(u, v) = u, y(u, v) = v$, где $-L/2 \leq u \leq L/2, -h \leq v \leq 0$. Точки области в момент времени t будут иметь координаты $X = x(u, v) + U(x(u, v), y(u, v), t), Y = y(u, v) + V(x(u, v), y(u, v), t)$, где надо положить $x = u, y = v$ и $-L/2 \leq u \leq L/2, -h \leq v \leq 0$.

```
DynamicModule[{w = 1, L = 3, a = 1, h = 2, k, X, Y}, k = π/L;
  X[u_, v_, t_] = u + aCos[wt]Cos[ku](Exp[kv] + Exp[-2kh]Exp[-kv]);
  Y[u_, v_, t_] = v + aCos[wt]Sin[ku](Exp[kv] - Exp[-2kh]Exp[-kv]);
  Animate[
    ParametricPlot[{{X[u, v, t], Y[u, v, t]}}, {u, -L/2, L/2}, {v, -h, 0},
      Mesh → 26, PlotRange → {{-L/2, L/2}, {-h, 1.}},
      PlotLabel → t, Axes → {True, False}],
    {t, 0, 2π}, AnimationRunning → False]]
```

Область, занимаемая жидкостью, в начальный и некоторые последующие моменты времени показана на следующем рисунке.



Наблюдая анимацию, можно заметить, что у стенок и дна движение происходит вдоль границ. Из уравнений (2) видно, что движение любой фиксированной точки (x, y) состоит из гармонического колебания по некоторому прямолинейному отрезку в плоскости XY . Этот факт можно продемонстрировать графически, если среду смоделировать в виде множества точек и нарисовать траектории некоторых из них.

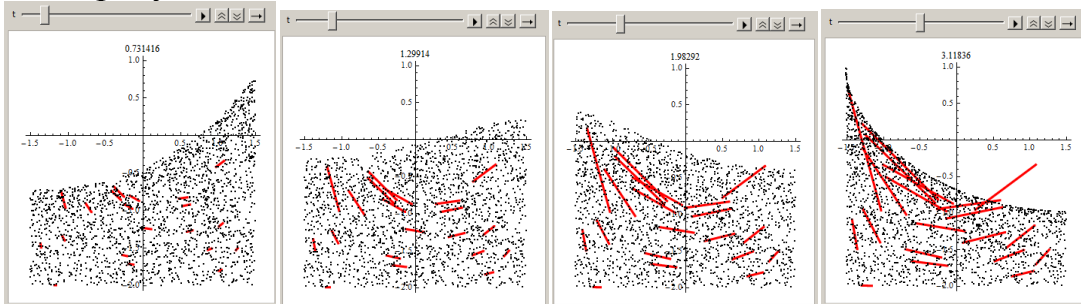
```
DynamicModule[{n = 2000, L = 3, h = 2, a = 1, w = 1, np = 20,
  k, ul, vl, X, Y, nl, up, vp, Xp, Yp, Xl, Yl, pt, pp, pl},
  ul = RandomReal[{-L/2, L/2}, n];
  vl = RandomReal[{-h, 0}, n];
  k = π/L;
  X[u_, v_, t_] = u + aCos[wt]Cos[ku](Exp[kv] + Exp[-2kh]Exp[-kv]);
  Y[u_, v_, t_] = v + aCos[wt]Sin[ku](Exp[kv] - Exp[-2kh]Exp[-kv]);
  SeedRandom[1234]; (* инициализация генератора случайных чисел *)
  nl = RandomInteger[{1, n}, np];
```

```

up = ul[[nl]]; vp = vl[[nl]];
Xp[τ_] = X[up, vp, τ]; Yp[τ_] = Y[up, vp, τ];
Animate[
  Xl = X[ul, vl, t];
  Yl = Y[ul, vl, t];
  pt = Transpose[{Xl, Yl}];
  pp = ParametricPlot[Transpose[{Xp[τ], Yp[τ]}], {τ, 0, t},
    PlotStyle → {Red, Thickness[0.01]};
  pl = ListPlot[pt, PlotStyle → {Black, PointSize[0.01]},
    PlotRange → {{-L/2, L/2}, {-h, 1.}}];
  Show[pp, pl, PlotRange → {{-L/2, L/2}, {-h, 1.}},
    AxesOrigin → Automatic],
  {t, 0.01, 2π}, AnimationRunning → False]]

```

Положение точек жидкости в последовательные моменты времени показано на следующем рисунке.



В этом коде из n случайных точек выбираются np . Для них строятся траектории движения, которые, как видно из рисунков, являются отрезками.

Пример 4.3. Установившееся волновое движение жидкости в резервуаре конечной ширины и бесконечной глубины [1].

Рассмотрим сосуд шириной L бесконечной глубины. Ось X направлена вправо, ось Y вверх. Начало координат расположено в середине свободной поверхности жидкости. Одно из возможных установившихся волновых движений жидкости может быть получено из формул (2) при $h = \infty$.

$$\begin{aligned}
 X &= x + a \cos(\omega t) \cos(kx) e^{ky} \\
 Y &= y + a \cos(\omega t) \sin(kx) e^{ky}
 \end{aligned}
 \quad (-L/2 < u < L/2, v \leq 0), \quad (3)$$

где (x, y) координаты точек среды в состоянии покоя, X и Y координаты тех же точек в момент времени t , и a, ω, k – некоторые постоянные.

Для создания анимации аналогично предыдущему примеру, жидкость в состоянии покоя будем моделировать в виде плоской области, заданной в параметрическом виде $x(u, v) = u, y(u, v) = v$, где $-L/2 < u < L/2, v \leq 0$.

Положение точек области в движении будут определяться формулами (3).

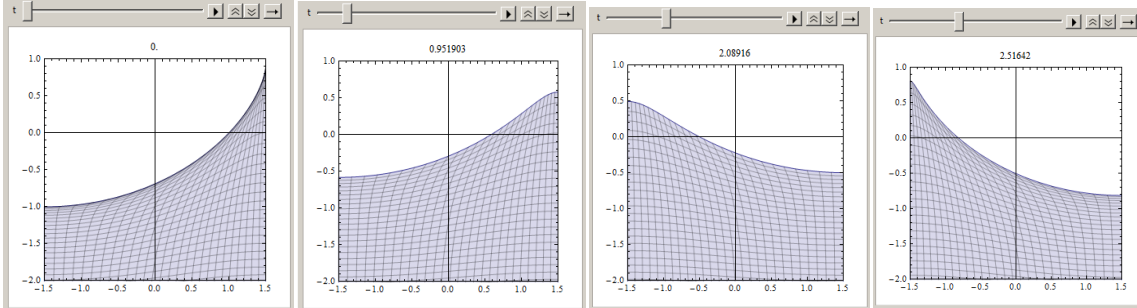
```

DynamicModule[{w = 1, L = 3, k = π/L, a = 1, X, Y},
  X[u_, v_, t_] = u + a Cos[wt] Cos[ku] Exp[kv];
  Y[u_, v_, t_] = v + a Cos[wt] Sin[ku] Exp[kv];
  Animate[ParametricPlot[

```

```
{X[u, v, t], Y[u, v, t]}, {u, -L/2, L/2}, {v, -2.5, 0}, Mesh → 26,
PlotRange → {{-L/2, L/2}, {-2, 1}}, PlotLabel → t],
{t, 0, 2π}, AnimationRunning → False]]
```

Область, занимаемая жидкостью, в начальный и некоторые последующие моменты времени показана на следующем рисунке.



Здесь, как и в предыдущем примере, частицы жидкости двигаются по прямолинейным отрезкам.

Пример 4.4. Стоячие волны в неглубокой воде [1]. Под ними понимают волны, которые возникают в сосуде, равновесная глубина h которого мала по сравнению с длиной волн $\lambda = 2\pi/k$. В этом случае мы можем аппроксимировать зависимости U и V от y , оставив в разложении в ряд Тейлора только первые члены. В результате уравнения (2) примут вид

$$U = 2a \cos(\omega t) \cos(kx)$$

$$V = 2a \cos(\omega t) \sin(kx) (k(y+h))$$

Для такой волны горизонтальное смещение частиц не зависит от y , а вертикальное смещение меняется линейно с глубиной, достигая нуля на дне и максимума на поверхности.

```
DynamicModule[{w = 1, L = 3, k = π/L, a = 0.2, h = 1, X, Y},
```

```
X[u_, v_, t_] = u + 2aCos[wt]Cos[ku];
```

```
Y[u_, v_, t_] = v + 2aCos[wt]Sin[ku]k(v + h);
```

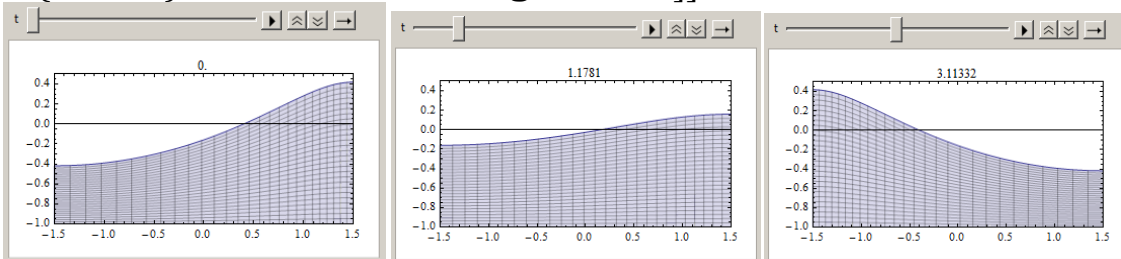
```
Animate[
```

```
ParametricPlot[{{X[u, v, t], Y[u, v, t]}, {u, -L/2, L/2}, {v, -h, 0},
```

```
Mesh → 26, PlotRange → {{-L/2, L/2}, {-h, 0.5}},
```

```
PlotLabel → t, Axes → {True, False}},
```

```
{t, 0, 2π}, AnimationRunning → False]]
```



Заметим, что в нашем примере $\lambda = 2\pi/k = 2\pi/(\pi/L) = 2L = 6$, $h=1$ и соотношение $h \ll \lambda$ для волн на неглубокой воде выполняется (приблизительно).

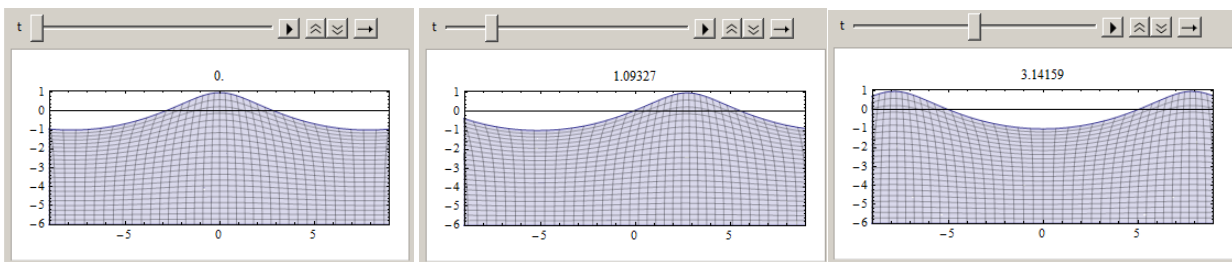
Пример 4.5. Бегущие волны в водоеме конечной глубины, неограниченном в горизонтальном направлении, можно смоделировать уравнениями [1]

$$\begin{aligned} U &= a \sin(\omega t - kx) (e^{ky} + e^{-2kh} e^{-ky}) \\ V &= a \cos(\omega t - kx) (e^{ky} - e^{-2kh} e^{-ky}) \end{aligned} \quad (4)$$

где U и V мгновенные значения смещений частицы жидкости с равновесными координатами x, y в направлениях осей X и Y . Следующий код моделирует движение жидкости в соответствии с уравнениями (4).

```
DynamicModule[{w = 1, L = 6, a = 1, h = 6, k = 0.4, X, Y},
  X[u_, v_, t_] = u + a Sin[wt - ku] (Exp[kv] + Exp[-2kh] Exp[-kv]);
  Y[u_, v_, t_] = v + a Cos[wt - ku] (Exp[kv] - Exp[-2kh] Exp[-kv]);
  Animate[
    ParametricPlot[{{X[u, v, t], Y[u, v, t]}}, {u, -2L, 2L}, {v, -h, 0},
      Mesh -> {49, 21}, PlotRange -> {{-1.5L, 1.5L}, {-h, 1.1}},
      PlotLabel -> t, Axes -> {True, False}],
    {t, 0, 2π}, AnimationRunning -> False]]
```

На следующем рисунке представлена двумерная область, имитирующая бегущую волну в последовательные моменты времени.



Из формул (4) следует, что частицы жидкости двигаются по эллипсам

$$\frac{(X - x)^2}{(e^{ky} + e^{-2kh} e^{-ky})^2} + \frac{(Y - y)^2}{(e^{ky} - e^{-2kh} e^{-ky})^2} = 1, \quad \text{которые с увеличением глубины}$$

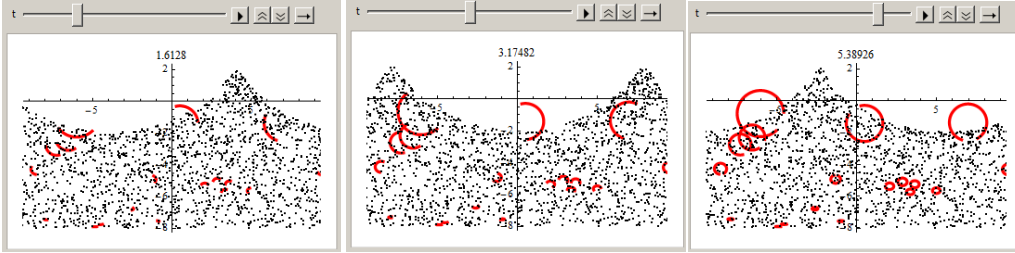
постепенно «сплюсчиваются», а на дне вырождаются в горизонтальные отрезки. Этот факт можно продемонстрировать графически, если среду смоделировать в виде множества точек и нарисовать траектории некоторых из них.

```
DynamicModule[{n = 2000, L = 6, h = 8, a = 2, w = 1, np = 20, k = 0.4,
  ul, vl, X, Y, nl, up, vp, Xp, Yp, Xl, Yl, pt, pp, pl},
  ul = RandomReal[{-1.8L, 1.8L}, n];
  vl = RandomReal[{-h, 0}, n];
  X[u_, v_, t_] = u + a Sin[wt - ku] (Exp[kv] + Exp[-2kh] Exp[-kv]);
  Y[u_, v_, t_] = v + a Cos[wt - ku] (Exp[kv] - Exp[-2kh] Exp[-kv]);
  SeedRandom[Floor[SessionTime[ ]]];
  nl = RandomInteger[{1, n}, np];
  up = ul[[nl]]; vp = vl[[nl]];
  Xp[τ_] = X[up, vp, τ];
  Yp[τ_] = Y[up, vp, τ];
  Animate[
```

```

Xl = X[ul, vl, t];
Yl = Y[ul, vl, t];
pt = Transpose[{Xl, Yl}];
pp = ParametricPlot[Transpose[{Xp[τ], Yp[τ]}], {τ, 0, t},
  PlotStyle → {Red, Thickness[0.01]};
pl = ListPlot[pt, PlotStyle → {Black, PointSize[0.005]},
  PlotRange → {{-1.5L, 1.5L}, {-h, 0}}];
Show[pp, pl, PlotRange → {{-1.5L, 1.5L}, {-h, 2.1}},
  AxesOrigin → Automatic, PlotLabel → t],
{t, 0.01, 2π}, AnimationRunning → False]]

```



Если рассмотреть водоем бесконечной глубины, то траектории точек будут окружностями, радиус которых зависит от глубины y . Смоделируйте самостоятельно такое движение.

2.4.5 Одномерные волновые колебания

Поперечные колебания струны, продольные колебания упругого стержня и многие другие явления описываются одномерным волновым уравнением

$$\frac{\partial^2 u(x, t)}{\partial t^2} = a^2 \frac{\partial^2 u(x, t)}{\partial x^2} \quad (1)$$

Для однозначного определения решения необходимо задать начальные условия

$$u(x, 0) = \varphi(x), \quad u'_t(x, 0) = \psi(x) \quad (2)$$

и граничные условия, если у колеблющегося объекта имеются границы. Обычно функция $u(x, t)$ представляет смещение из положения равновесия точки x одномерной среды в момент времени t .

Известно общее решение Даламбера задачи (1), (2) для неограниченной прямой [2]

$$u(x, t) = \frac{\varphi(x + at) + \varphi(x - at)}{2} + \frac{1}{2a} \int_{x-at}^{x+at} \psi(\alpha) d\alpha \quad (-\infty < x < \infty, t \geq 0). \quad (3)$$

Формула (3) при любых функциях $\varphi(x)$ и $\psi(x)$ дает решение уравнения (1). В данном параграфе во всех задачах начальная скорость колебаний равна нулю $\psi(x) = 0$ и поэтому решение будет иметь вид

$$u(x, t) = \frac{\Phi(x + at) + \Phi(x - at)}{2}, \quad (4)$$

а функция $\Phi(x)$ будет подбираться так, чтобы удовлетворить начальному условию $u(x,0) = \varphi(x)$ и дополнительным условиям на границе. Во всех задачах также будем считать, что параметр $a=1$.

Пример 5.1. Колебание конечной струны с синусоидальным начальным профилем и нулевой начальной скоростью.

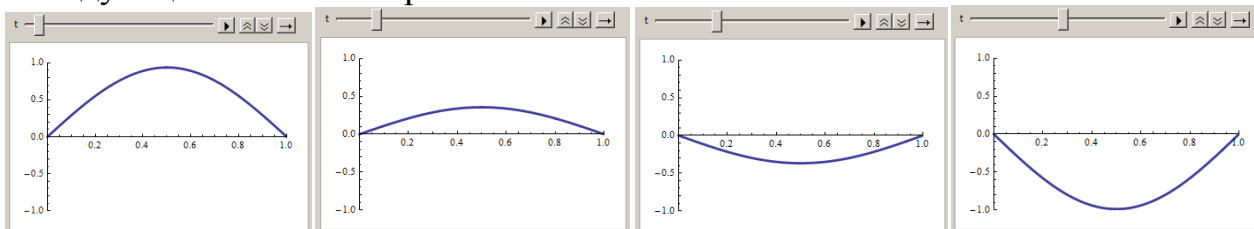
Пусть струна имеет конечную длину $L=1$ и ось X направлена вправо вдоль струны. Если начальные условия имеют вид $u(x,0) = \sin \pi x$, $u'_t(x,0) = 0$ и струна закреплена на концах, т.е. $u(0,t) = u(1,t) = 0$, то решение уравнения (1), удовлетворяющее указанным условиям, может быть получено из (4) при $\Phi(x) = \sin \pi x$

$$u(x,t) = \frac{\sin(\pi(x+t)) + \sin(\pi(x-t))}{2}$$

В следующем коде мы моделируем движение струны в соответствии с этим решением.

```
DynamicModule[{u, f},
  f[x_] = Sin[πx];
  u[x_, t_] = (f[x + t] + f[x - t]) / 2;
  Animate[Plot[Evaluate[u[x, t]], {x, 0, 1},
    PlotStyle → Thickness[0.01], PlotPoints → 1000,
    PlotRange → {{0, 1}, {-1, 1}},
    {t, 0, 2}, AnimationRunning → False, DisplayAllSteps → True]]
```

На следующем рисунке показана панель анимации в начальный и несколько последующих моментов времени.



Пример 5.2. Колебание полубесконечной струны, закрепленной на конце.

Рассмотрим задачу о распространении начального возмущения $u(x,0) = \varphi(x)$ по полубесконечной струне с закрепленным концом $x=0$. Начальная скорость равна нулю $\psi(t) = 0$ и граничное условие имеет вид $u(0,t) = 0$. Можно показать, что, если начальную функцию $\varphi(x)$ продолжить с левых $x \leq 0$ нечетно на всю вещественную ось и подставить в (4), то получим решение поставленной задачи [2]. Это значит, что полученная таким образом функция $u(x,t)$, рассматриваемая только для $x \geq 0$, будет удовлетворять уравнению, граничному и обоим начальным условиям.

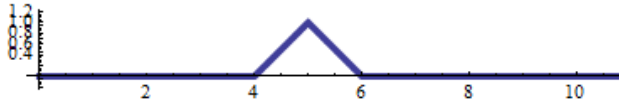
Нечетное продолжение $\Phi(x)$ любой функции $\varphi(x)$, заданной только для положительных x , на всю вещественную ось можно получить по формуле $\Phi(x) = \text{sign}(x) \varphi(|x|)$. Тогда из (4) получаем

$$u(x,t) = \frac{\varphi(x+at) + \text{sign}(x-at)\varphi(|x-at|)}{2} \quad (5)$$

Пусть начальное отклонение струны равно нулю везде, кроме интервала (4,6), в котором функция $\varphi(x)$ изображается равнобедренным треугольником единичной высоты. На следующем рисунке показана форма начального профиля струны.

$$\varphi[x_] = \frac{1}{2} \text{Abs}[x - 4] - \text{Abs}[x - 5] + \frac{1}{2} \text{Abs}[x - 6];$$

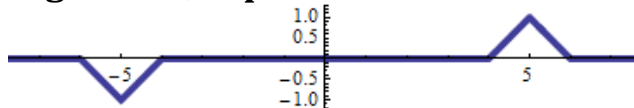
Plot[$\varphi[x]$, { x , 0, 12}, **PlotStyle** → **Thickness**[0.01],
PlotPoints → 1000, **PlotRange** → **All**, **AspectRatio** → **Automatic**]



Ниже показан график функции $\Phi(x)$, являющейся нечетным продолжением функции $\varphi(x)$.

$$f[x_] = \text{Sign}[x]\varphi[\text{Abs}[x]];$$

Plot[$f[x]$, { x , -8, 8}, **PlotStyle** → **Thickness**[0.01], **PlotPoints** → 1000,
PlotRange → **All**, **AspectRatio** → **Automatic**]



Следующий код моделирует движение струны, используя формулу (5).

DynamicModule{ u ,

$$u[x_, t_] = \frac{f[x+t] + f[x-t]}{2};$$

Animate[

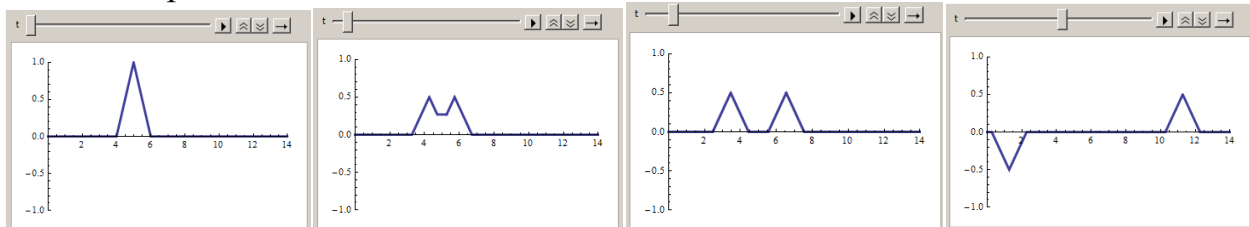
Plot[**Evaluate**[$u[x, t]$], { x , 0, 14},

PlotStyle → **Thickness**[0.01], **PlotPoints** → 1000,

PlotRange → {{0, 14}, {-1, 1}},

{ t , 0, 12}, **AnimationRunning** → **False**, **DisplayAllSteps** → **True**]

Ниже показана панель анимации в начальный и несколько последующих моментов времени.



Заметим, что если начальная функция $\varphi(x)$ не дифференцируема в некоторых точках, то полученное решение тоже не дифференцируемо в некоторых точках. Такое решение называют обобщенным.

Пример 5.3. *Колесание полубесконечной струны со свободным левым концом.*

Рассмотрим задачу о распространении начального возмущения $u(x,0) = \varphi(x)$ по полубесконечной струне со свободным левым концом $x=0$. Начальная скорость равна нулю $\psi(t) = 0$ и граничное условие имеет вид $u'_x(0,t) = 0$. Можно показать, что, если начальную функцию $\varphi(x)$ продолжить с левых $x \geq 0$ четно на всю вещественную ось и подставить в (4), то получим решение поставленной задачи [2]. Функцию $u(x,t)$ следует рассматриваемая только для $x \geq 0$ и она будет удовлетворять уравнению (1), граничному и обоим начальным условиям.

Четное продолжение $\Phi(x)$ на вещественную ось любой функции $\varphi(x)$, заданной только для $x \geq 0$, можно получить по формуле $\Phi(x) = \varphi(|x|)$. Тогда из (4) получим

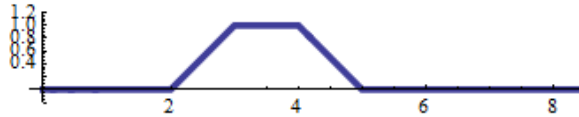
$$u(x,t) = \frac{\varphi(x+at) + \varphi(|x-at|)}{2} \quad (6)$$

Пусть начальное отклонение струны равно нулю везде, кроме интервала (2,5) в котором функция $\varphi(x)$ изображается равнобедренной трапецией высотой 1. На следующем рисунке показан график начального профиля струны.

```

 $\varphi[x_] = \frac{1}{2} \text{Abs}[x - 2] - \frac{1}{2} \text{Abs}[x - 3] - \frac{1}{2} \text{Abs}[x - 4] + \frac{1}{2} \text{Abs}[x - 5];$ 
Plot[ $\varphi[x]$ , {x, 0, 10}, PlotStyle → Thickness[0.01], PlotPoints → 1000,
PlotRange → All, AspectRatio → Automatic]

```



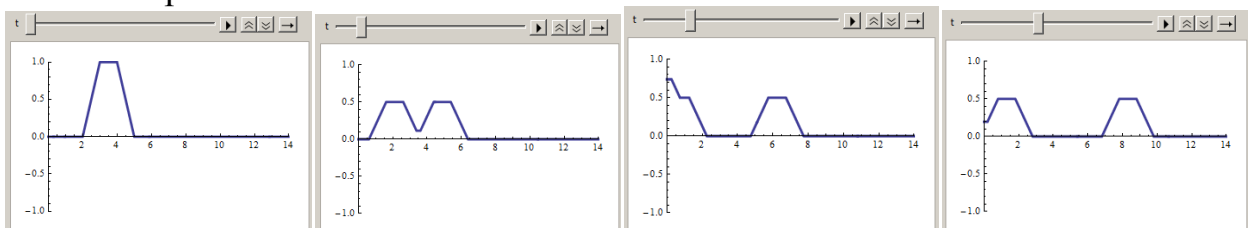
Следующий код моделирует движение струны, используя формулу (6).

```

DynamicModule{u,  $\varphi$ , f},
 $\varphi[x_] = \frac{1}{2} \text{Abs}[x - 2] - \frac{1}{2} \text{Abs}[x - 3] - \frac{1}{2} \text{Abs}[x - 4] + \frac{1}{2} \text{Abs}[x - 5];$ 
f[x_] =  $\varphi[\text{Abs}[x]]$ ;
 $u[x_, t_] = \frac{f[x+t] + f[x-t]}{2}$ ;
Animate[
Plot[Evaluate[ $u[x, t]$ ], {x, 0, 14}, PlotStyle → Thickness[0.01],
PlotPoints → 1000, PlotRange → {{0, 14}, {-1, 1}},
{t, 0, 12}, AnimationRunning → False, DisplayAllSteps → True]

```

Ниже показана панель анимации в начальный и несколько последующих моментов времени.



Пример 5.4. *Колебание конечной струны с закрепленными концами.*

Рассмотрим задачу о колебании конечной струны длины L с закрепленными концами, начальным смещением $u(x,0)=\varphi(x)$ и нулевой начальной скоростью $\psi(t)=0$. Граничные условия имеют вид $u(0,t)=0$, $u(L,t)=0$. Если начальную функцию $\varphi(x)$, обращающуюся в ноль на концах отрезка $[0, L]$, продолжить на всю вещественную ось нечетно и периодически (с отрезка $[0, L]$ на отрезок $[-L, 0]$ нечетно, а затем с отрезка $[-L, L]$ периодически на всю ось) и, созданную функцию $\Phi(x)$ подставить в (4), то получим решение рассматриваемой задачи [3]. Функция $u(x,t)$, рассматриваемая только для $0 \leq x \leq L$, будет удовлетворять обоим начальным и обоим граничным условиям.

Нечетное периодическое продолжение $\Phi(x)$ любой функции $\varphi(x)$, заданной на отрезке $0 \leq x \leq L$, на всю вещественную ось можно получить по формуле [3]

$$\Phi(x) = (-1)^{\left[\frac{x}{L}\right]} \varphi(stc(x, 2L)), \quad (6)$$

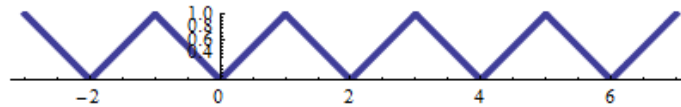
где $[x]$ (квадратные скобки) представляет функцию взятия наибольшего целого, не превосходящего x , а $stc(x, w)$ является периодической пилообразной непрерывной кусочно-линейной функцией, определяемой формулой

$$stc(x, w) = \left| x - w \left[\frac{x}{w} + \frac{1}{2} \right] \right| \quad \text{или} \quad stc(x, w) = \frac{w}{2\pi} \arccos \left(\cos \frac{2\pi x}{w} \right),$$

где w является периодом этой функции.

Stc[x_, w_] = Abs[x - w Floor[x/w + 1/2]];

Plot[Stc[x, 2], {x, -3, 7}, , AspectRatio -> Automatic]



Из (4) имеем

$$u(x,t) = \frac{(-1)^{\left[\frac{x+at}{L}\right]} \varphi(stc(x+at, 2L)) + (-1)^{\left[\frac{x-at}{L}\right]} \varphi(stc(x-at, 2L))}{2} \quad (7)$$

Пусть $L=1$ и начальная функция $\varphi(x)$ имеет треугольную форму, как показано на следующем рисунке слева. Следующий код моделирует движение точек струны, смещение $u(x,t)$ которых задается формулой (7).

DynamicModule[{L = 1, \varphi, F, u},

\varphi[x_] = 1/2 - Abs[x - 1/2];

F[x_] = (-1)^Floor[x/L] \varphi[Stc[x, 2L]];

u[x_, t_] = \frac{F[x+t] + F[x-t]}{2};

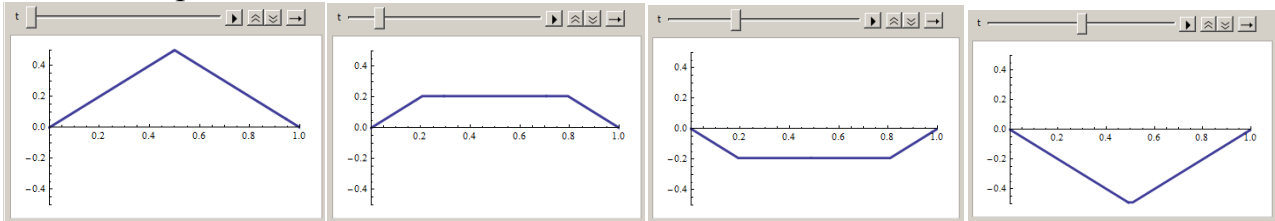
Animate[

Plot[Evaluate[u[x, t]], {x, 0, L}, PlotStyle -> Thickness[0.01],

PlotPoints -> 1000, PlotRange -> {{0, L}, {-0.5, 0.5}},

{t, 0, 2}, AnimationRunning -> False, DisplayAllSteps -> True]]

Ниже показана панель анимации в начальный и несколько последующих моментов времени.



Формула (7) дает решение задачи о колебании конечной струны длины L , закрепленной на концах, имеющей нулевую начальную скорость и начальное смещение $u(x,0)=\varphi(x)$. Приведем несколько примеров колебаний при различных функциях $\varphi(x)$. Во всех примерах левые панели анимации показаны для момента времени $t=0$, т.е. на них показаны начальные профили струн.

DynamicModule[{ $L = 4, f, F, u$ },

$$f[x_] = \frac{1}{2} \text{Abs}[x - 1] - \text{Abs}[x - 2] + \frac{1}{2} \text{Abs}[x - 3];$$

$$F[x_] = (-1)^{\text{Floor}[x/L]} f[\text{Stc}[x, 2L]];$$

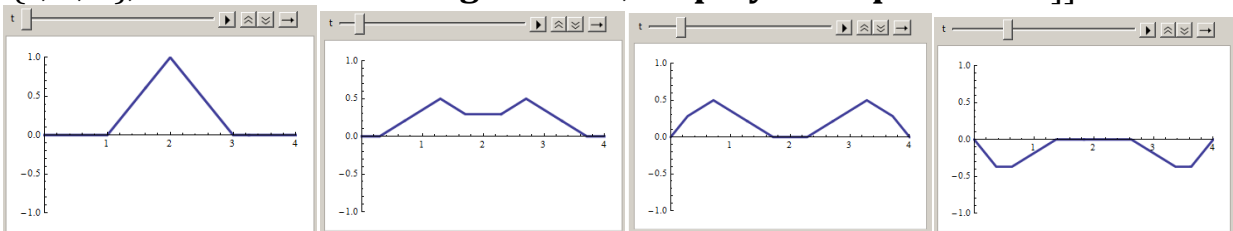
$$u[x_, t_] = \frac{F[x + t] + F[x - t]}{2};$$

Animate[

Plot[**Evaluate**[$u[x, t]$], { $x, 0, L$ }, **PlotStyle** → **Thickness**[0.01],

PlotPoints → 1000, **PlotRange** → {{0, L }, {-1, 1}},

{ $t, 0, 8$ }, **AnimationRunning** → **False**, **DisplayAllSteps** → **True**]



DynamicModule[{ $L = 1, f, F, u$ },

$$f[x_] = 2\sqrt{x}(1 - x);$$

$$F[x_] = (-1)^{\text{Floor}[x/L]} f[\text{Stc}[x, 2L]];$$

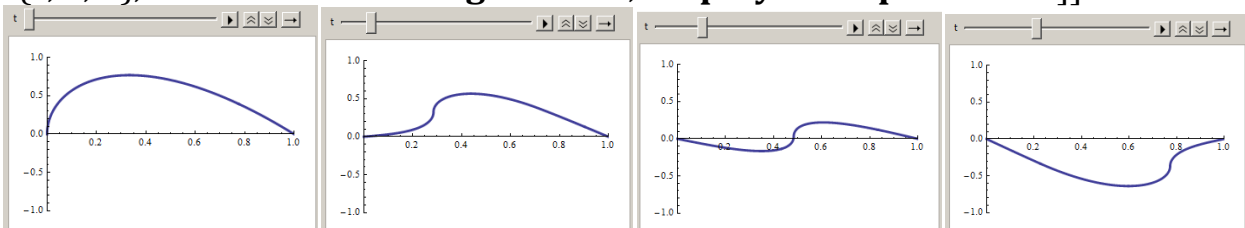
$$u[x_, t_] = \frac{F[x + t] + F[x - t]}{2};$$

Animate[

Plot[**Evaluate**[$u[x, t]$], { $x, 0, L$ }, **PlotStyle** → **Thickness**[0.01],

PlotPoints → 1000, **PlotRange** → {{0, L }, {-1, 1}},

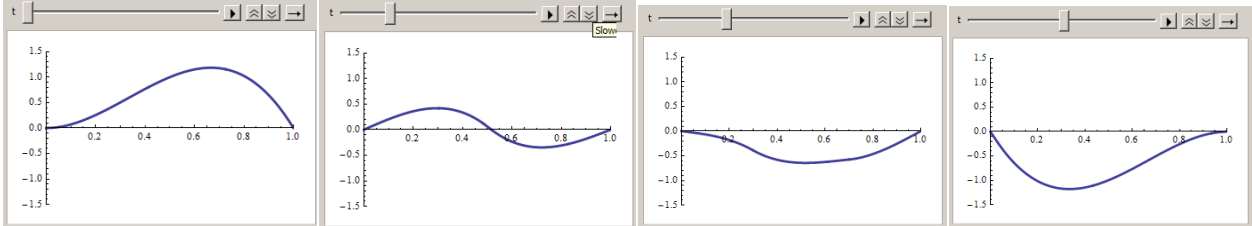
{ $t, 0, 2$ }, **AnimationRunning** → **False**, **DisplayAllSteps** → **True**]



```

DynamicModule[{L = 1, f, F, u},
  f[x_] = 8x^2(1 - x);
  F[x_] = (-1)^Floor[x/L]f[Stc[x, 2L]];
  u[x_, t_] =  $\frac{F[x + t] + F[x - t]}{2}$ ;
  Animate[
    Plot[Evaluate[u[x, t]], {x, 0, L}, PlotStyle → Thickness[0.01],
      PlotPoints → 1000, PlotRange → {{0, L}, {-1.5, 1.5}},
    {t, 0, 2}, AnimationRunning → False, DisplayAllSteps → True]]

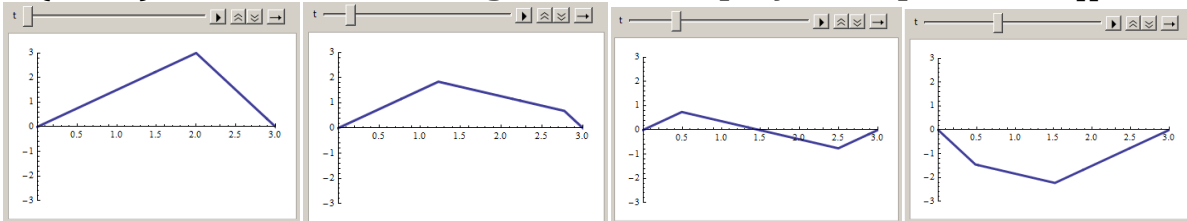
```



```

DynamicModule[{L = 3, f, F, u},
  f[x_] =  $-\frac{3}{4}x + \frac{9}{2} - \frac{9}{4}\text{Abs}[x - 2]$ ;
  F[x_] = (-1)^Floor[x/L]f[Stc[x, 2L]];
  u[x_, t_] =  $\frac{F[x + t] + F[x - t]}{2}$ ;
  Animate[
    Plot[Evaluate[u[x, t]], {x, 0, L}, PlotStyle → Thickness[0.01],
      PlotPoints → 1000, PlotRange → {{0, L}, {-3, 3}},
    {t, 0, 6}, AnimationRunning → False, DisplayAllSteps → True]]

```



Заметим, что во всех приведенных выше примерах используется определение функции $stc(x, w)$, приведенное нами ранее.

Пример 5.5. Решим задачу о колебание конечной струны с закрепленным левым концом и свободным правым. Для этого надо решить уравнение колебаний (1) с начальными условиями (2) и граничными условиями

$$u(0, t) = 0, u'_x(L, t) = 0 \quad (8)$$

Известно, если начальные данные в задаче о распространении колебаний на неограниченной прямой являются нечетными функциями относительно некоторой точки x_0 , то соответствующее решение в этой точке равно нулю.

Если начальные данные в задаче о распространении колебаний на неограниченной прямой являются четными функциями относительно некоторой точки x_0 , то производная по x соответствующего решения в этой точке равна нулю.

Рассмотрим функцию $\varphi(x)$ заданную на отрезке $[0, L]$ такую, что $\varphi(0) = 0$. Продолжим ее четно относительно точки $x=L$ на отрезок $[L, 2L]$, т.е. построим функцию $\varphi_1(x)$, $x \in [0, 2L]$ такую, что для $x \in [0, L]$, $\varphi_1(x) = \varphi(x)$, а для $x \in [L, 2L]$ $\varphi_1(x) = \varphi(2L - x)$. Функцию $\varphi_1(x)$ с отрезка $[0, 2L]$ продолжим нечетно на отрезок $[-2L, 0]$, т.е. построим функцию $\varphi_2(x)$, $x \in [-2L, 2L]$ такую, что для $x \in [0, 2L]$, $\varphi_2(x) = \varphi_1(x)$, а для $x \in [-2L, 0]$, $\varphi_2(x) = -\varphi_1(-x)$. Полученную функцию $\varphi_2(x)$ с отрезка $[-2L, 2L]$ продолжим периодически на всю ось с периодом $4L$. В результате получим периодическую функцию $\Phi(x)$ которая на отрезке $x \in [0, L]$ совпадает с $\varphi(x)$, является четной относительно точки $x=L$ и нечетной относительно точки $x=0$. Назовем описанное продолжение функции $\varphi(x)$ четно-нечетным периодическим продолжением. В силу сказанного выше, подстановка такой функции $\Phi(x)$ в формулу (4) даст решение поставленной задачи (1), (2), (8).

Известно [4], что для любой функции $\varphi(x)$ заданной на отрезке $[0, L]$ ($L > 0$), такой что $\varphi(0) = 0$, формула четно-нечетного периодического продолжения имеет следующий вид

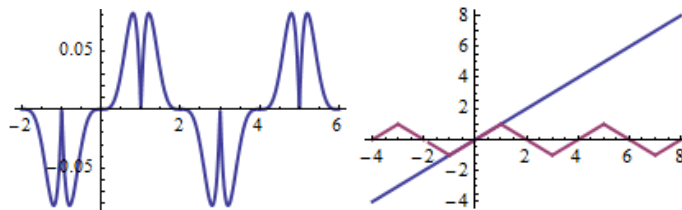
$$\Phi(x) = (-1)^{\left\lfloor \frac{x}{2L} \right\rfloor} \varphi(\text{Stc}(x, 2L)) \quad (9)$$

Вот пример четно-нечетного продолжения функции $f(x) = x^4(1-x)$ с отрезка $[0, 1]$ на всю вещественную ось (следующий рисунок слева)

$L = 1; f[x_] = x^4(L - x);$

$F[x_] = (-1)^{\text{Floor}[x/(2L)]} f[\text{Stc}[x, 2L]];$

$\text{Plot}[F[x], \{x, -2, 6\}, \text{PlotRange} \rightarrow \text{All}, \text{PlotStyle} \rightarrow \text{Thickness}[0.01]]$



Четно – нечетное продолжение функции $f(x) = x$ с отрезка $[0, 1]$ на всю вещественную ось показано на предыдущем рисунке справа, где кроме продолженной функции, для сравнения мы изобразили и исходную.

Таким образом, решением задачи (1), (2), (8) при $\psi(x) = 0$ будет функция [4]

$$u(x, t) = \frac{(-1)^{\left\lfloor \frac{x+at}{2L} \right\rfloor} \varphi(\text{Stc}(x + at, 2L)) + (-1)^{\left\lfloor \frac{x-at}{2L} \right\rfloor} \varphi(\text{Stc}(x - at, 2L))}{2} \quad (10)$$

Пусть $L=4$ и начальная функция $\varphi(x) = x$. На следующем рисунке слева показана начальная форма струны. Код, приведенный ниже, моделирует движение точек струны, смещение которых $u(x, t)$ задается формулой (10).

DynamicModule[{**L = 4, f, F, u**},

f[x_] = x;

F[x_] = (-1)^Floor[x/(2L)] f[Stc[x, 2L]];

$$u[x_, t_] = \frac{F[x + t] + F[x - t]}{2};$$

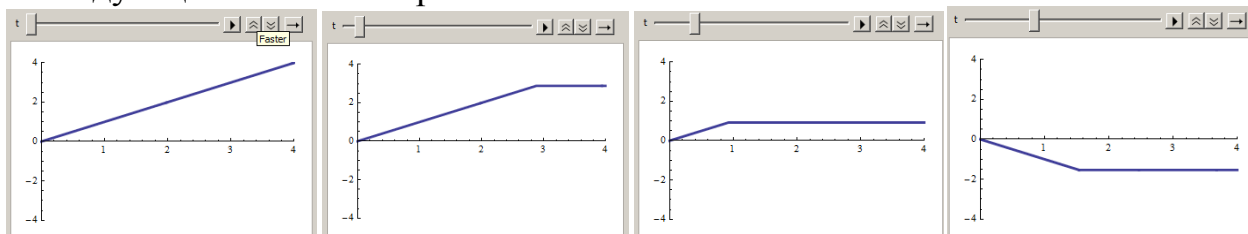
Animate[

Plot[**Evaluate**[$u[x, t]$], { $x, 0, L$ }, **PlotStyle** → **Thickness**[0.01],

PlotPoints → 1000, **PlotRange** → {{0, L}, {-L, L}},

{ $t, 0, 16$ }, **AnimationRunning** → **False**, **DisplayAllSteps** → **True**]

На следующих рисунках показана панель анимации в начальный и несколько последующих моментов времени.



Заметим, что в коде используется определение функции $stc(x, w)$, приведенное нами ранее.

Пример 5.6. Продольные колебания стержня постоянного поперечного сечения описываются уравнением (1) с начальными условиями (2) и граничными условиями, зависящими от способа закрепления концов [5].

Пусть круглый стержень длиной L , закрепленный на одном конце и свободный на другом, подвержен линейному растяжению. Начало координат расположим в центре левого (фиксированного) сечения, и ось X направим вправо вдоль оси стержня. Начальное смещение будет иметь вид $u(x, 0) = \varphi(x) = kx$, где k некоторая постоянная. В начальный момент времени растянутый конец освобождается и стержень начинает сжиматься, а затем удлиняться – начинаются продольные колебания. Такие колебания будут описываться уравнением (1), где $u(x, t)$ является продольным смещением в момент времени t сечения стержня, имеющего в нулевой момент времени координату x . Начальные условия будут иметь вид

$$u(x, 0) = kx, \quad u'(x, 0) = 0, \quad (11)$$

а граничные условия будут иметь вид (8). Решение этой задачи нами построено в предыдущем примере. Здесь мы смоделируем движение круглого стержня, сечения которого смещаются в соответствии с решением $U(x, t)$ задачи (1), (8), (11). Для функции решения мы используем обозначение U , чтобы не путать его с параметром u в уравнениях поверхности.

Пусть параметрические уравнения поверхности стержня в состоянии покоя (нулевой момент времени) имеют вид $x = x(u, v)$, $y = y(u, v)$, $z = z(u, v)$. Все точки одного и того же поперечного сечения стержня смещаются в момент t на одну и ту же величину $U(x, t)$. Поэтому в момент t они будут иметь продольную координату $X = x(u, v) + U(x(u, v), t)$. В плоскости сечения смещения точек нет, поэтому y и z координаты точек сечения не меняются. В

результате в момент времени t точки стержня, имевшие при $t=0$ координаты $x = x(u, v)$, $y = y(u, v)$, $z = z(u, v)$, будут иметь координаты

$$X = x(u, v) + U(x(u, v), t), \quad Y = y(u, v), \quad Z = z(u, v). \quad (12)$$

В следующем коде мы моделируем продольные колебания круглого стержня радиуса r , точки поверхности которого в момент t имеют координаты (12).

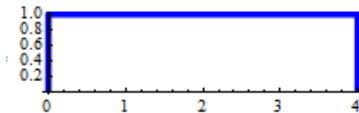
Круглый стержень в п.2.2.3 мы строили, как поверхность вращения ломаной, имеющей форму «скобы». Параметрическое уравнение «скобы» имеет вид

$$L = 4; \quad r = 1;$$

$$xc[t_] = \frac{L}{2} + \frac{1}{2} \text{Abs}[t - r] - \frac{1}{2} \text{Abs}[t - r - L];$$

$$zc[t_] = \left(r + \frac{L}{2}\right) - \frac{1}{2} \text{Abs}[t - r] - \frac{1}{2} \text{Abs}[t - r - L];$$

$$\text{ParametricPlot}[\{xc[t], zc[t]\}, \{t, 0, L + 2r\}, \\ \text{PlotStyle} \rightarrow \{\text{Thickness}[0.02], \text{Blue}\}]$$



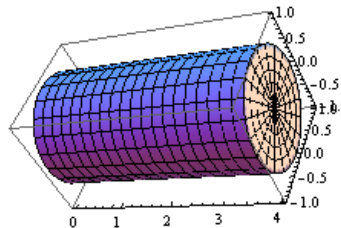
Параметрическое уравнение поверхности вращения кривой $x_c(u)$, $y_c(u)$ вокруг оси X имеет вид $x(u, v) = x_c(u)$, $y(u, v) = z_c(u) \cos v$, $z(u, v) = z_c(u) \sin v$.

$$x[u_, v_] = xc[u];$$

$$y[u_, v_] = \text{Cos}[v]zc[u];$$

$$z[u_, v_] = \text{Sin}[v]zc[u];$$

$$\text{ParametricPlot3D}[\{x[u, v], y[u, v], z[u, v]\}, \{u, 0, L + 2r\}, \{v, 0, 2\pi\}, \\ \text{Mesh} \rightarrow \text{Full}, \text{PlotPoints} \rightarrow \{31, 21\}, \text{BoxRatios} \rightarrow \{L, 2r, 2r\}]$$



Записываем определение координатных функций (12).

$$\text{Stc}[x_, w_] = \text{Abs}[x - w \text{Floor}[x/w + 1/2]]; \\ f[x_] = x/L;$$

$$F[x_] = (-1)^{\text{Floor}[x/(2L)]} f[\text{Stc}[x, 2L]]; \\ U[x_, t_] = \frac{F[x + t] + F[x - t]}{2};$$

$$X[u_, v_, t_] = x[u, v] + U[x[u, v], t];$$

$$Y[u_, v_, t_] = y[u, v];$$

$$Z[u_, v_, t_] = z[u, v];$$

$$X[u_, v_, t_] = x[u, v] + U[x[u, v], t];$$

$$Y[u_, v_, t_] = y[u, v];$$

$$Z[u_, v_, t_] = z[u, v];$$

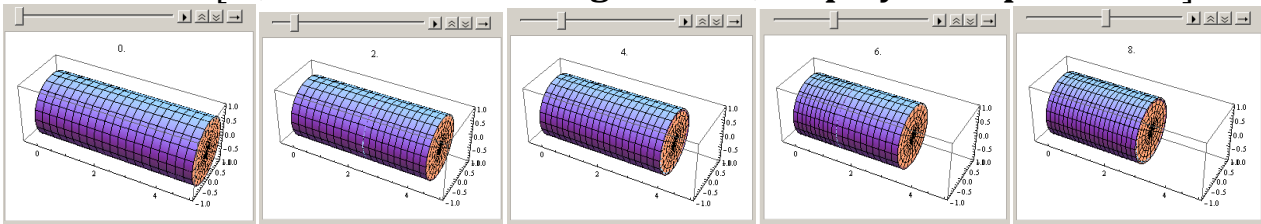
Создаем список `tc` изображений стержня в различные моменты времени и анимируем его с помощью функции `ListAnimate`.

$$tc = \text{Table}[\text{ParametricPlot3D}[\{X[u, v, t], Y[u, v, t], Z[u, v, t]\}, \{u, 0, L + 2r\}, \{v, 0, 2\pi\},$$

```

Mesh → Full, PlotPoints → {31, 21},
PlotRange → {{-0.5, L + 1.1}, {-r, r}, {-r, r}}, PlotLabel → t],
{t, 0, 4L - 0.25, 0.25}];
ListAnimate[tc, AnimationRunning → False, DisplayAllSteps → True]

```



Обратите внимание на то, что стержень не просто сжимается/разжимается, но в отдельных частях сжимается/разжимается координатная сетка на его поверхности.

Причина по которой мы использовали функцию `ListAnimate`, а не `Animate`, состоит в том, что построение анимации большого количества 3D графиков занимает у системы много времени. В примере процесс разделен на два шага. Первый состоит в создании списка изображений. Второй шаг состоит воспроизведении этого вписка, что происходит более гладко.

Отметим, что приведенный код не собран нами в динамический модуль для того, чтобы продемонстрировать результаты работы отдельных блоков кода. Однако, вы можете сделать это самостоятельно.

Пример 5.7. Продольные колебания пружины. Уравнение продольных колебаний для стержня и пружины записываются одинаково [2].

Пусть спиральная пружина длиной L , закрепленная на одном конце и свободная на другом, подвержена линейному растяжению. Начало координат расположим в на левом краю пружины, и ось X направлена вправо вдоль ее оси. Если к свободному концу пружины приложить растягивающее усилие, то деформация будет линейной. Допустим, что усилие таково, что правый конец сместится на единицу. Тогда начальное смещение будет иметь вид $u(x,0) = x/L$. В начальный момент времени растянутый конец освобождается и пружина начинает сжиматься и разжиматься – начинаются продольные колебания. Такие колебания описываются уравнением (1), где $u(x,t)$ является продольным смещением в момент времени t точек пружины, имевших в нулевой момент времени координату x .

Решение уравнения (1) с начальными условиями (11) и граничными условиями (8) нами уже построено ранее. Здесь мы смоделируем движение спиральной пружины, точки которой смещаются в соответствии с решением $U(x,t)$ задачи (1), (8), (11).

Пусть параметрические уравнения спиральной кривой, представляющей пружину, имеют вид $x = x(u)$, $y = y(u)$, $z = z(u)$. Точка пружины в момент времени t смещается в продольном направлении на величину $U(x,t)$. Поэтому в момент t она будет иметь новую координату $X = x(u) + U(x(u),t)$. Координаты y и z точки не меняются. В результате в момент времени t точка пружины,

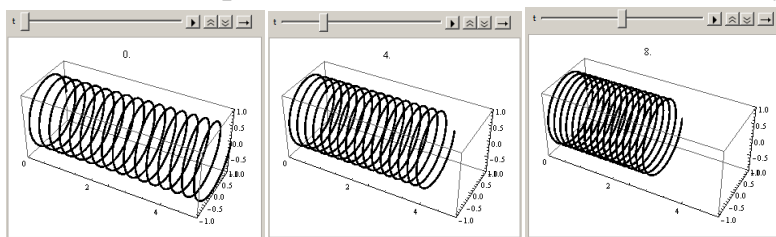
имевшая при $t=0$ координаты $x = x(u)$, $y = y(u)$, $z = z(u)$, будет иметь координаты

$$X = x(u) + U(x(u), t), Y = y(u), Z = z(u). \quad (13)$$

В следующем коде мы моделируем продольные колебания пружины радиуса r , точки которой в момент t имеют координаты (13). Пружину рисуем в виде пространственной круговой спиральной кривой длиной $L=4$ с $n=4$ витками на единице длины.

```
DynamicModule[{L = 4, n = 4, x, y, z, f, F, U, X, Y, Z},
  x[w_] = w;
  y[w_] = Cos[2 * Pi * w * n];
  z[w_] = Sin[2 * Pi * w * n];
  f[x_] =  $\frac{z}{L}$ ;
  F[x_] = (-1)^Floor[x/(2L)]f[Stc[x, 2L]];
  U[x_, t_] =  $\frac{F[x - t] + F[x + t]}{2}$ ;
  X[w_, t_] = x[w] + U[x[w], t];
  Y[w_, t_] = y[w];
  Z[w_, t_] = z[w];
  Animate[
    ParametricPlot3D[Evaluate[{X[w, t], Y[w, t], Z[w, t]}, {w, 0, L},
      PlotStyle → Thickness[0.01], AspectRatio → Automatic,
      PlotPoints → 1000, PlotRange → {{0, L + 1}, {-1, 1}, {-1, 1}},
      PlotLabel → t],
    {t, 0, 4L}, AnimationRunning → False, DisplayAllSteps → True]]
```

Форма пружины в моменты времени $t = 0, 4, 8$ показана на следующем рисунке.



Пример 5.8. Крутильные колебания кругового стержня.

Рассмотрим круглый вертикальный стержень длиной L и радиуса r , зафиксированный в нижнем сечении и закрученный парой сил в верхнем. Круговые сечения стержня остаются круговыми в процессе закручивания, а диаметр сечений и расстояния между ними не меняется при условии, что угол закручивания мал. Начало координат расположим в центре основания стержня и ось Z направим вдоль его оси вертикально вверх. В начальный момент верхнее сечение освобождается и в стержне возникают крутильные колебания. Известно [5], что для поперечного сечения с координатой z угол поворота $\theta(z, t)$ удовлетворяет волновому уравнению (1).

Начальный угол поворота $\theta(z,0)$ будет линейно меняться вдоль стержня $\theta(z,0) = k z$, где постоянная k зависит от свойств материала стержня и пары закручивающих сил. Пусть в параметрических уравнениях поверхности цилиндра $x(u, v), y(u, v), z(u, v)$ параметр v представляет угол. Тогда в уравнениях поверхности закрученного цилиндра этот параметр должен быть увеличен на величину угла θ . Угол поворота сечений является функцией координаты z и момента времени t , т.е. $\theta = U(z, t)$. В результате уравнения поверхности закрученного стержня будут иметь вид

$$X = x(u, v + U(z(u, v), t)), Y = y(u, v + U(z(u, v), t)), Z = z(u, v + U(z(u, v), t)), \quad (14)$$

а функцию $U(z, t)$ следует искать как решения волнового уравнения (1) с начальными условиями $U(z, 0) = k z, U'_t(z, 0) = 0$ и граничными условиями $U(0, t) = 0$ (нижнее сечение закреплено, поворота нет), $U'_x(L, t) = 0$ (верхнее сечение свободно) [5]. Решение этой задачи мы уже построили в примере 5.5. Используем его для графического представления крутильных колебаний.

Вначале составляем параметрические уравнения поверхности цилиндрического стержня, как поверхности вращения ломаной в форме «скобы» вокруг оси Z .

$$L = 6; r = 2;$$

$$xc[t_] = (r + \frac{L}{2}) - \frac{1}{2} \text{Abs}[t - r] - \frac{1}{2} \text{Abs}[t - r - L];$$

$$zc[t_] = \frac{L}{2} + \frac{1}{2} \text{Abs}[t - r] - \frac{1}{2} \text{Abs}[t - r - L];$$

$$\text{ParametricPlot}[\{xc[t], zc[t]\}, \{t, 0, L + 2r\},$$

$$\text{PlotStyle} \rightarrow \{\text{Thickness}[0.02], \text{Blue}\}] \quad (* \text{ рисунок скобы слева } *)$$

$$x[u_, v_] = \text{Cos}[v]xc[u];$$

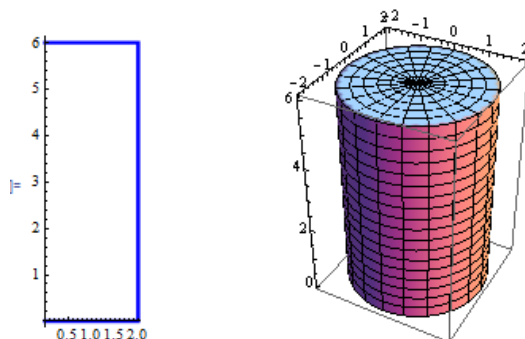
$$y[u_, v_] = \text{Sin}[v]xc[u];$$

$$z[u_, v_] = zc[u];$$

$$\text{ParametricPlot3D}[\text{Evaluate}[\{x[u, v], y[u, v], z[u, v]\}],$$

$$\{u, 0, L + 2r\}, \{v, 0, 2\pi\}, \text{Mesh} \rightarrow \text{Full}, \text{PlotPoints} \rightarrow \{31, 21\},$$

$$\text{BoxRatios} \rightarrow \{2r, 2r, L\}] \quad (* \text{ рисунок цилиндра справа } *)$$



Затем строим решение уравнения крутильных колебаний (1) с начальными условиями $U(z, 0) = z/L, U'_t(z, 0) = 0$ и граничными условиями $U(0, t) = 0, U'_x(L, t) = 0$.

$$\text{Stc}[x_, w_] = \text{Abs}[x - w \text{Floor}[x/w + 1/2]];$$

$$f[x_] = x/L;$$

$$F[x_] = (-1)^{\text{Floor}[x/(2L)]} f[\text{Stc}[x, 2L]];$$

$$U[x_, t_] = \frac{F[x + t] + F[x - t]}{2};$$

Составляем параметрические уравнения поверхности закрученного стержня в соответствии с формулами (14)

$$X[u_, v_, t_] = x[u, v + U[z[u, v], t]];$$

$$Y[u_, v_, t_] = y[u, v + U[z[u, v], t]];$$

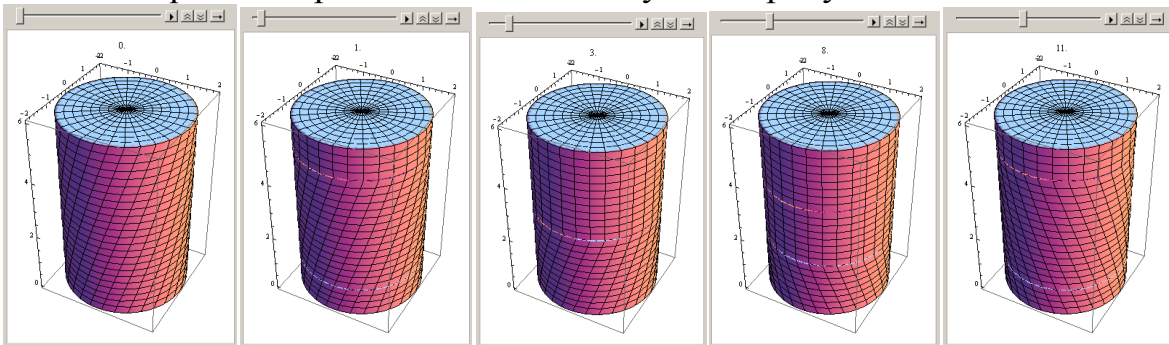
$$Z[u_, v_, t_] = z[u, v + U[z[u, v], t]];$$

Создаем список `tc` изображений стержня в различные моменты времени и анимируем его с помощью функции `ListAnimate`.

```
tc = Table[ParametricPlot3D[
    Evaluate[{X[u, v, t], Y[u, v, t], Z[u, v, t]}, {u, 0, L + 2r}, {v, 0, 2π},
    PlotPoints → {41, 31}, Mesh → Full,
    PlotRange → {{-r, r}, {-r, r}, {-0.1, L + 0.1}}, PlotLabel → t],
    {t, 0, 4L - 0.25, 0.25}];
```

```
ListAnimate[tc, AnimationRunning → False, DisplayAllSteps → True]
```

Изображения крутящегося цилиндра в начальный и некоторые последующие моменты времени представлены на следующем рисунке.



Обратите внимание, что геометрическое тело – цилиндр остается цилиндром, однако меняется координатная сеть на его поверхности. Она показывает, как закручены те или иные сечения цилиндра.

2.4.6 Задачи сопротивления материалов и теории упругости

Пример 6.1. Изгиб балки равномерно распределенной нагрузкой.

Пусть начало координат находится на левом конце балки и ось X направим вдоль оси балки вправо, ось Y направим вертикально вверх. Под действием приложенных к балке вертикальных сил она прогибается.

Уравнение изгибающего момента балки имеет вид

$$\frac{d^2 M}{dx^2} = -q, \quad (1)$$

где q внешняя сила, рассчитанная на единицу длины, а $M(x)$ изгибающий момент в балке. Для определения прогиба балки нужно решать уравнение

$$EJ \frac{d^2 u(x)}{dx^2} = -M(x), \quad (2)$$

где J – момент инерции поперечного сечения балки, E – модуль упругости материала балки, $u(x)$ – отклонение нейтральной оси балки от положения равновесия. Из (1) и (2) получаем дифференциальное уравнение прогиба

$$\frac{d^2}{dx^2} \left(EJ \frac{d^2 u}{dx^2} \right) = q(x) \quad (3)$$

Для свободно опертой на краях балки граничные условия имеют вид

$$u(0) = 0, u(L) = 0, u''(0) = 0, u''(L) = 0 \quad (4)$$

При ином закреплении граничные условия будут другими.

Рассмотрим балку длины L и высотой/толщиной h , свободно опертую на концах. Известно [6], что прогиб такой балки определяется по формуле

$$u(x) = \frac{q}{24 EJ} (L^3 x - 2 L x^3 + x^4), \quad (5)$$

и

$$M(x) = \frac{q x}{2} (L - x) \quad (6)$$

Легко проверить, что функция $u(x)$ удовлетворяет дифференциальному уравнению (3) и граничным условиям (4), а функция $M(x)$ получается из (2) подстановкой в нее функции $u(x)$.

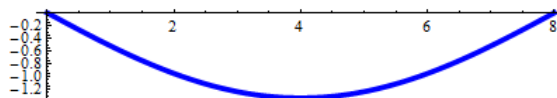
В соответствии с (5) нагруженная балка может быть представлена как изогнутый отрезок.

$x = .; L = 8; h = 2; Em = 10; J = 4; q = -1; (* Em - модуль упругости *)$

$U[x_] = q/(24 * Em * J) * (L^3 * x - 2 * L * x^3 + x^4);$

$Plot[U[x], {x, 0, L}, PlotStyle -> {Thickness[0.01], Blue},$

$AspectRatio -> Automatic]$

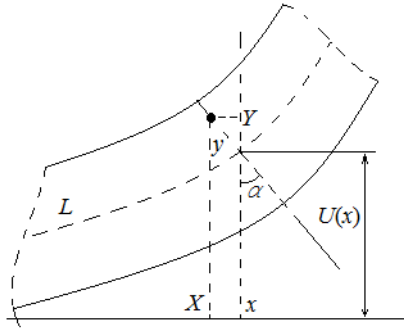


Здесь для модуля упругости мы использовали идентификатор Em , поскольку символ E в системе защищен (он используется для обозначения основания натуральных логарифмов). Нагрузке q мы присвоили отрицательное значение, поскольку она направлена вниз, а ось Y мы направили вверх.

После выполнения одномерного расчета смоделируем поле напряжений внутри балки. Для этого балку рассмотрим как двумерную область, параметрические уравнения которой имеют вид

$$x(u, v) = u, \quad y(u, v) = v \quad (0 \leq u \leq L, -h/2 \leq v \leq h/2).$$

Используя уравнения плоской области $x(u, v)$, $y(u, v)$ недеформированной балки и уравнение $U(x)$ ее нейтральной оси, можно написать параметрическое уравнение области деформированной балки. В соответствии с гипотезой плоских сечений, все поперечные сечения, которые были перпендикулярны нейтральной оси балки до деформирования, остаются перпендикулярными нейтральной оси балки после ее деформирования.



Точки, имевшие до деформирования координаты (x, y) , переходят в точки с координатами (X, Y) , где $X = x - y \sin \alpha$, $Y = U(x) + y \cos \alpha$ (см. рисунок). Но $\operatorname{tg} \alpha = U'(x)$ и, поэтому, $X = x - \frac{U'_x(x)y}{\sqrt{1+(U'_x(x))^2}}$ и $Y = U(x) + \frac{y}{\sqrt{1+(U'_x(x))^2}}$. Тогда функции $X(u, v), Y(u, v)$, представляющие параметрические уравнения области изогнутой балки, будут иметь вид

$$\begin{aligned} X(u, v) &= x(u, v) - \frac{U'_x(x(u, v))y(u, v)}{\sqrt{1+(U'_x(x(u, v)))^2}}, \\ Y(u, v) &= U(x(u, v)) + \frac{y(u, v)}{\sqrt{1+(U'_x(x(u, v)))^2}} \end{aligned} \quad (7)$$

Строим область деформированной балки.

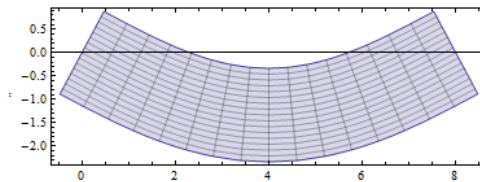
$$x[\mathbf{u}_-, \mathbf{v}_-] = \mathbf{u}; \quad y[\mathbf{u}_-, \mathbf{v}_-] = \mathbf{v};$$

$$\mathbf{U1} = U';$$

$$X[\mathbf{u}_-, \mathbf{v}_-] = x[\mathbf{u}, \mathbf{v}] - \frac{\mathbf{U1}[x[\mathbf{u}, \mathbf{v}]] y[\mathbf{u}, \mathbf{v}]}{\sqrt{1 + \mathbf{U1}[x[\mathbf{u}, \mathbf{v}]]^2}};$$

$$Y[\mathbf{u}_-, \mathbf{v}_-] = U[x[\mathbf{u}, \mathbf{v}]] + \frac{y[\mathbf{u}, \mathbf{v}]}{\sqrt{1 + \mathbf{U1}[x[\mathbf{u}, \mathbf{v}]]^2}};$$

$$\text{ParametricPlot}[\{\{X[\mathbf{u}, \mathbf{v}], Y[\mathbf{u}, \mathbf{v}]\}, \{\mathbf{u}, 0, L\}, \{\mathbf{v}, -h/2, h/2\}\}$$



Следующим шагом будет показать в цвете продольные напряжения σ_x в балке. Если поперечное сечение балки прямоугольное, то σ_x вычисляются по формуле

$$\sigma_x(x, y) = \frac{M(x) \cdot y}{J}, \quad \text{где } M(x) \text{ – момент инерции в сечении } x \text{ балки, } y \text{ –}$$

вертикальное смещение точки балки относительно нейтральной оси. Чтобы использовать эту функцию в качестве функции цвета, ее нужно привести к диапазону $[0, 1]$. Для этого нужно знать минимальные и максимальные значения напряжения σ_x . В нашей задаче они, очевидно, равны напряжениям в

срединном сечении балки в верхней и нижней точках

$$\sigma_{x \max/\min} = \pm \frac{M(L/2) \cdot (h/2)}{J}, \text{ где } h - \text{высота балки.}$$

Создаем функцию продольных напряжений S_x и вычисляем ее максимальное и минимальное значения S_{\max} и S_{\min} . Они используются для построения шкалы цветов напряжения справа от рисунка. Затем рисуем область изогнутой балки, раскрашенную в соответствии со значениями функции продольных напряжений S_x .

$$M = .; M[x_] = \frac{q x(L - x)}{2};$$

$$Sx[u_, v_] = M[x[u, v]]y[u, v]/J;$$

$$Smax = Sx[L/2, h/2]$$

$$Smin = Sx[L/2, -h/2]$$

2

-2

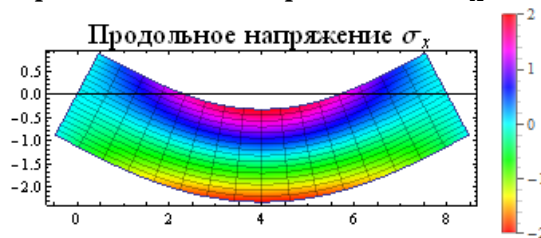
ParametricPlot[[{X[u, v], Y[u, v]}, {u, 0, L}, {v, -h/2, h/2},

ColorFunction → **Function** [{x, y, u, v}, Hue[Rescale[Sx[u, v], {-2, 2}]]],

ColorFunctionScaling → **False**,

PlotLegends → **BarLegend**[{Hue, {-2, 2}}],

PlotLabel → **Style**["Продольное напряжение" σ_x , 18]]



Здесь функция **Rescale** используется для приведения диапазона [-2,2] значений функции S_x к диапазону [0,1], который требуется для функции **Hue**. Опция **ColorFunctionScaling**→**False** нужна для того, чтобы система не пыталась приводить аргументы функции **ColorFunction** к диапазону [0,1], что она делает по умолчанию.

Пример 6.2. Действие сосредоточенной силы на упругую полуплоскость.

Рассмотрим плоскую упругую среду, ограниченную сверху горизонтальной прямой. В некоторой точке к границе приложена вертикальная сосредоточенная сила P . Ось X направим вправо вдоль границы, а ось Y из точки приложения силы вертикально вверх. Среда расположена в полуплоскости $y < 0$. Вертикальные напряжения Y_y в полуплоскости вычисляются по формулам [8]

$$Y_y = \frac{2P}{\pi} \frac{y^3}{(x^2 + y^2)^2} \quad (8)$$

Напряжение Y_u представляет отнесенную к единице площади силу, направленную вдоль оси Y , приложенную к горизонтальной площадке (перпендикулярной направлению y).

Напряжение Y_u стремится к минус бесконечности в точке приложения силы. Чтобы его отобразить в цвете на полуплоскости диапазон изменения функции Y_u необходимо ограничить. Для этого создадим обрезанную снизу функцию напряжений $TensY_u(x, y)$, используя функцию `Piecewise`. Значение обрезания подбираем из «эстетических» соображений.

PP = 1;

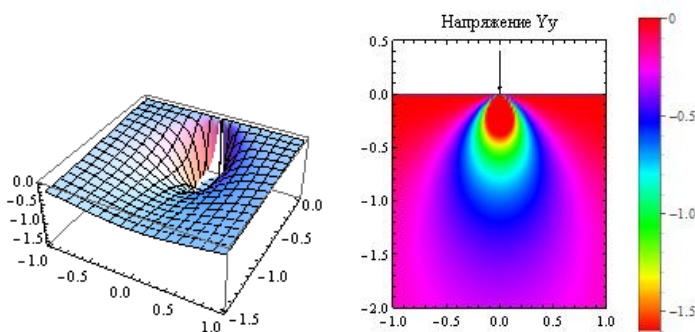
$$Y_u[x_, y_] = \frac{2PP}{\pi} \frac{y^3}{(x^2 + y^2)^2};$$

TensY_u[x_, y_] = Piecewise[{{Y_u[x, y], Y_u[x, y] > -1.6}}, -1.6];

Plot3D[TensY_u[x, y], {x, -1, 1}, {y, 0, -1.5}] (* следующий рисунок слева *)

Теперь строим полуплоскость с окраской, соответствующей значениям функции $TensY_u$, предварительно нормируя ее к диапазону $[0, 1]$ (следующий рисунок справа)

```
p = ParametricPlot[{{u, v}}, {u, -1., 1.}, {v, -2, 0.5},
  PlotPoints → 100, Mesh → None,
  RegionFunction → Function[{x, y, u, v}, y < 0],
  PlotRange → {{-1., 1.}, {-2, 0.5}},
  ColorFunction → Function[{x, y, u, v},
    Hue[Rescale[TensY_u[u, v], {-1.6, 0}]]],
  ColorFunctionScaling → False,
  PlotLegends → BarLegend[{{Hue, {-1.6, 0}}],
  Axes → None, PlotLabel → "Напряжение Y_u"];
```



На последнем рисунке ярко красная зона (под точкой приложения нагрузки) соответствует большим отрицательным (сжимающим) напряжениям, а холодный красный (у поверхности вдали от точки приложения нагрузки) соответствует близким к нулю напряжениям. Это показывает шкала цветов, построенная справа от графика.

Известно [7], что любой элемент среды, расположенный на расстоянии r от точки приложения силы, подвергается простому сжатию в радиальном направлении. При этом компоненты напряжения определяются формулами

$$\sigma_r = \frac{2P \cos \theta}{\pi r}, \quad \sigma_\theta = \tau_{r\theta} = 0, \quad \text{где } \theta \text{ – угол, образуемый радиус – вектором точки с}$$

вертикалью. Выберем окружность произвольного диаметра d с центром на вертикальной оси и касательную к поверхности полуплоскости. Для любой точки этой окружности имеем $d \cos \theta = r$. Отсюда, согласно выражению для

напряжения σ_r , получаем $\sigma_r = \frac{2P}{\pi d}$, т.е. напряжения σ_r во всех точках

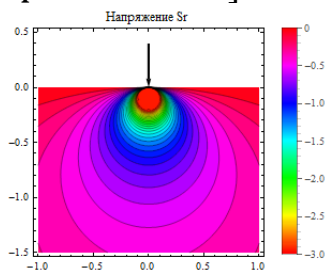
окружности остаются одинаковыми, за исключением точки приложения нагрузки. Проиллюстрируем этот факт графически. Для этого изобразим

напряжения $\sigma_r(x, y) = \frac{2P \cos \theta}{\pi r} = \frac{2P y}{\pi(x^2 + y^2)}$ в области полуплоскости в виде

контурного графика

$$Sr[x_, y_] = \frac{2 P y}{\pi(x^2 + y^2)};$$

```
p = ContourPlot[TensSr[x, y], {x, -1, 1}, {y, -1.5, 0.5},
  RegionFunction -> Function[{x, y, f}, y < 0],
  Contours -> Table[pp, {pp, -3, 0, 0.1}],
  PlotRange -> {-3, 0}, ClippingStyle -> Automatic,
  ColorFunction -> Function[{f}, Hue[Rescale[f, {-3, 0}]]],
  ColorFunctionScaling -> False, Axes -> None,
  PlotLegends -> Automatic];
ar = Graphics[{Thickness[0.01], Arrow[{{0, 0.4}, {0, 0}}]}];
Show[p, ar, PlotLabel -> "Напряжение Sr"]
```



Кака видно, линии уровня функции $S_r(x, y)$ (напряжения σ_r) являются окружностями. Ярко красный круг под точкой приложением нагрузки представляет область обрезания, в которой $\sigma_r \leq -3$.

Обратите внимание на следующие особенности этого кода. Опция `PlotRange->{-3, 0}` задает диапазон изменения значений функции $S_r(x, y)$ и, тем самым, определяет область отсекаемых значений. Чтобы отсекаемые зоны изображались как часть поверхности, а не как «дырки», используется опция `ClippingStyle->Automatic`. Диапазон `{-3, 0}`, задаваемый опцией `PlotRange`, также используется опцией `PlotLegends->Automatic`, которая строит контрольную цветовую полосу справа от графика.

Пример 6.3. Действие нескольких сосредоточенной силы на упругую полуплоскость.

В предыдущем примере мы изобразили сжимающие напряжения Y_y в бесконечной полуплоскости, возникающие под действием одной сосредоточенной силы. Поскольку уравнения теории упругости линейные, то напряжения, возникающие от действия нескольких сосредоточенных сил, будут суммой напряжений (8), возникающих от действия каждой из сил.

Пусть точки приложения трех сосредоточенных сил имеют координату $x = -0.5, 1.0, 0.5$. Изобразим суммарное Y_y напряжение в полуплоскости в виде контурного графика

$$PP = 1; Yy[x_, y_] = \frac{2PP}{\pi} \frac{y^3}{(x^2 + y^2)^2};$$

$minYy = -2.5;$ (* уровень обрезания напряжения *)

$SYy[x_, y_] = Yy[x, y] + Yy[x - 0.5, y] + Yy[x + 0.5, y];$

$p = \text{ContourPlot}[SYy[x, y], \{x, -1.5, 1.5\}, \{y, -2, 0.5\},$

$\text{RegionFunction} \rightarrow \text{Function}[\{x, y, f\}, y < 0],$

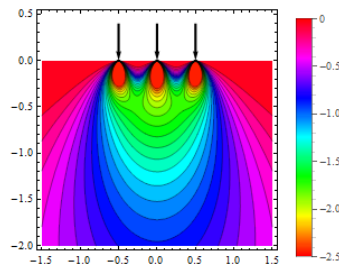
$\text{Contours} \rightarrow \text{Table}[pp, \{pp, minYy, 0, 0.1\}],$

$\text{ColorFunction} \rightarrow \text{Function}[\{f\}, \text{Hue}[\text{Rescale}[f, \{minYy, 0\}]]],$

$\text{ColorFunctionScaling} \rightarrow \text{False}, \text{Axes} \rightarrow \text{None},$

$\text{PlotRange} \rightarrow \{minYy, 0\}, \text{ClippingStyle} \rightarrow \text{Automatic},$

$\text{PlotLegends} \rightarrow \text{Automatic}];$



Опция $\text{PlotRange} \rightarrow \{minYy, 0\}$ задает диапазон изменения значений функции $SY_y(x, y)$. Чтобы отсекаемые зоны изображались как часть поверхности используется опция $\text{ClippingStyle} \rightarrow \text{Automatic}$. Опция $\text{PlotLegends} \rightarrow \text{Automatic}$ строит справа от графика шкалу цветов с диапазоном $\{minYy, 0\}$, который был указан в опции PlotRange .

Пример 6.4. Напряжения в бесконечной плоской пластинке при наличии круглого отверстия.

Решим задачу о визуализации напряжений в бесконечной пластинке с отверстием, подверженной однородному растяжению величиной P (усилие, приходящееся на единицу длины). Введем полярную систему координат (r, θ) , начало которой положим в центре отверстия радиуса a , а полярную ось направим вдоль усилия P . Известно [7], что напряжения в такой пластинке определяются по формулам

$$\begin{aligned}\sigma_r &= \frac{P}{2} \left(1 - \frac{a^2}{r^2} \right) + \frac{P}{2} \left(1 + \frac{3a^4}{r^4} - \frac{4a^2}{r^2} \right) \cos 2\theta \\ \sigma_\theta &= \frac{P}{2} \left(1 + \frac{a^2}{r^2} \right) - \frac{P}{2} \left(1 + \frac{3a^4}{r^4} \right) \cos 2\theta \\ \tau_{r\theta} &= -\frac{P}{2} \left(1 - \frac{3a^4}{r^4} + \frac{2a^2}{r^2} \right) \sin 2\theta\end{aligned}\quad (9)$$

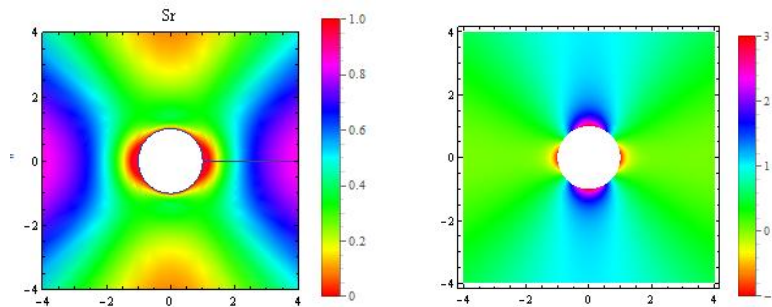
Изобразим напряжения σ_r в виде функциональной окраски области с круговым отверстием. Для этого создадим функцию $\sigma_r = S_r(r, \theta)$

$P = 1; a = 1;$

$$\text{Sr}[\mathbf{r}_-, \mathbf{v}_-] = \frac{P}{2} \left(1 - \frac{a^2}{r^2} \right) + \frac{P}{2} \left(1 + 3 \frac{a^4}{r^4} - 4 \frac{a^2}{r^2} \right) \text{Cos}[2v];$$

Из выражения для напряжения σ_r видно, что его минимальное и максимальное значения равны 0 и P. Эти величины мы используем при нормировке ColorFunction и задания диапазона значений для шкалы цветов.

```
ParametricPlot[{{uCos[v], uSin[v]}}, {u, 0, 6}, {v, 0, 2π},
  Mesh → None, PlotPoints → 50,
  RegionFunction → Function[{x, y, u, v}, x2 + y2 > 1],
  PlotRange → {{-4, 4}, {-4, 4}},
  ColorFunction → Function[{x, y, u, v}, Hue[Rescale[Sr[u, v], {0, 1}]]],
  ColorFunctionScaling → False,
  PlotLegends → BarLegend[{Hue, {0, 1}}],
  Axes → None, PlotLabel → "Sr" (* следующий рисунок слева *)
```



Напомним, что растяжение выполняется в горизонтальном направлении вдали от отверстия.

Напряжение σ_θ графически представим с помощью функции DensityPlot, предварительно записав его в декартовой системе координат (а не полярной, как в формулах (9)). Легко видеть, что $\cos 2\theta = \frac{x^2 - y^2}{x^2 + y^2}$. Поэтому

$$\text{StC}[\mathbf{x}_-, \mathbf{y}_-] = \frac{P}{2} \left(1 + \frac{a^2}{x^2 + y^2} \right) - \frac{P}{2} \left(1 + 3 \frac{a^4}{(x^2 + y^2)^2} \right) \frac{x^2 - y^2}{x^2 + y^2};$$

Минимальное и максимальное значение σ_θ равны $-P$ и $3P$. График плотности функции $\sigma_\theta = StC(x, y)$, изображенный на предыдущем рисунке справа, строится следующим образом

```
DensityPlot[StC[x, y], {x, -4, 4}, {y, -4, 4},
  RegionFunction → Function[{x, y, f}, x2 + y2 > 1],
  PlotPoints → 30, PlotRange → {-1, 3},
  ColorFunction → Hue, PlotLegends → Automatic]
```

Нашей следующей задачей является представление продольных напряжения σ_x в области пластинки. Напряжение σ_x , заданное в декартовой системе координат, выражается через напряжения $\sigma_r, \sigma_\theta, \tau_{r\theta}$ в полярной системе по формуле $\sigma_x = \sigma_r \cos^2 \theta + \sigma_\theta \sin^2 \theta - \tau_{r\theta} \sin 2\theta$, где

$$\sin(2\theta) = \frac{2xy}{x^2 + y^2}, \quad \cos^2(\theta) = \frac{x^2}{x^2 + y^2}, \quad \sin^2(\theta) = \frac{y^2}{x^2 + y^2}, \quad r^2 = x^2 + y^2.$$

Следующий код выполняет все необходимые преобразования и использует функцию $\sigma_\theta = StC(x, y)$, определенную нами выше.

P = 1; a = 1;

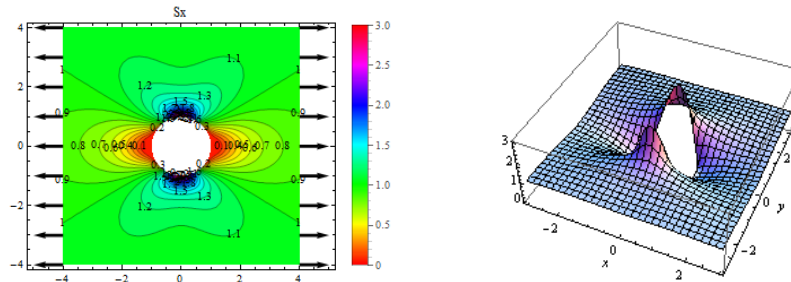
$$SrC[x_, y_] = \frac{P}{2} \left(1 - \frac{a^2}{x^2 + y^2} \right) + \frac{P}{2} \left(1 + 3 \frac{a^4}{(x^2 + y^2)^2} - 4 \frac{a^2}{x^2 + y^2} \right) \frac{x^2 - y^2}{x^2 + y^2};$$

$$TrtC[x_, y_] = -\frac{P}{2} \left(1 - 3 \frac{a^4}{(x^2 + y^2)^2} + 2 \frac{a^2}{x^2 + y^2} \right) \frac{2xy}{x^2 + y^2};$$

$$SxC[x_, y_] = SrC[x, y] \frac{x^2}{x^2 + y^2} + StC[x, y] \frac{y^2}{x^2 + y^2} - TrtC[x, y] \frac{2xy}{x^2 + y^2};$$

Для построения контурного графика и шкалы цветов надо определить минимальное и максимальное значение напряжения σ_x . Они равны 0 и $3P$ соответственно. Тогда

```
pc = ContourPlot[SxC[x, y], {x, -4, 4}, {y, -4, 4},
  RegionFunction → Function[{x, y, f}, x2 + y2 > 1],
  Contours → Table[pp, {pp, 0, 3, 0.1}], ColorFunction → Hue,
  ClippingStyle → Automatic, Axes → None, PlotLabel → Sx,
  PlotLegends → BarLegend[{Hue, {0, 3}}],
  PlotRange → {0, 3}, ContourLabels → All]
arR = Graphics[
  Table[{Thickness[0.01], Arrow[{{4, i}, {5, i}}]}, {i, -4, 4, 1}]];
arL = Graphics[
  Table[{Thickness[0.01], Arrow[{{-4, i}, {-5, i}}]}, {i, -4, 4, 1}]];
Show[pc, arR, arL, PlotRange → All, AspectRatio → Automatic]
```



Естественно, что любую функцию двух переменных можно изобразить как поверхность. Продольные напряжения, показанные на предыдущем рисунке справа, строятся как график функции двух переменных следующим кодом

```
Plot3D[SxC[x, y], {x, -3, 3}, {y, -3, 3},
  RegionFunction -> Function[{x, y, f}, x^2 + y^2 > a^2],
  PlotPoints -> 30, Mesh -> Full, PlotRange -> All]
```

Из этих графиков видно, что максимальное значения продольное напряжение σ_x достигается на концах диаметра круга, перпендикулярного к направлению растяжения, и оно в 3 раза больше постоянного напряжения P , приложенного на концах пластинки.

2.4.7 Задачи электростатики

Пример 7.1. Электростатический потенциал точечного заряда в трехмерном пространстве вычисляется по формуле [2]

$$\varphi(\mathbf{r}, \mathbf{p}_0) = \varphi(x, y, z, x_0, y_0, z_0) = \frac{q}{\sqrt{(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2}}, \quad (1)$$

где $\mathbf{p}_0 = (x_0, y_0, z_0)$ – точка расположения заряда. Потенциал поля n зарядов вследствие суперпозиции силовых полей будет выражаться формулой

$$\varphi = \sum_{i=1}^n \varphi(\mathbf{r}, \mathbf{p}_i). \quad (2)$$

Изобразим потенциал поля двух единичных зарядов противоположного знака, расположенных в точках $(-1, 0, 0)$ и $(1, 0, 0)$, с помощью контурного графика. Для этого создадим функцию, реализующую формулу (2)

```
 $\varphi[\mathbf{q}_-, \mathbf{p}_-, \mathbf{r}_-] := \text{Sum} \left[ \frac{\mathbf{q}[[i]]}{\text{Norm}[\mathbf{r} - \mathbf{p}[[i]]]}, \{i, \text{Length}[\mathbf{q}]\} \right];$ 
```

Здесь \mathbf{q} представляет список значений зарядов, \mathbf{p} является списком координат точек расположения этих зарядов, \mathbf{r} списком имен пространственных переменных. Функция Norm вычисляет норму вектора/списка

Norm[{x, y, z}]

$$\sqrt{\text{Abs}[x]^2 + \text{Abs}[y]^2 + \text{Abs}[z]^2}$$

Или

Norm[{x₁, y₁, z₁} - {x₂, y₂, z₂}]

$$\sqrt{\text{Abs}[x_1 - x_2]^2 + \text{Abs}[y_1 - y_2]^2 + \text{Abs}[z_1 - z_2]^2}$$

Вот, что возвращает функция $\varphi[\mathbf{q}, \mathbf{p}, \mathbf{r}]$ для двух единичных зарядов противоположного знака, расположенных в точках $(-1, 0, 0)$ и $(1, 0, 0)$.

$\varphi[\{-1, 1\}, \{\{-1, 0, 0\}, \{1, 0, 0\}\}, \{x, y, z\}] // \text{TraditionalForm}$

$$\frac{1}{\sqrt{|x-1|^2 + |y|^2 + |z|^2}} - \frac{1}{\sqrt{|x+1|^2 + |y|^2 + |z|^2}}$$

Команда построения контурного графика функции φ в трехмерном пространстве может иметь вид (следующий график слева)

ContourPlot3D[

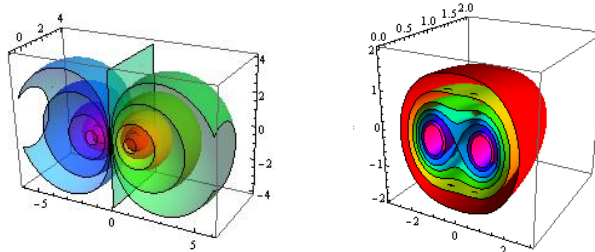
Evaluate [$\varphi[\{1, -1\}, \{\{-1, 0, 0\}, \{1, 0, 0\}\}, \{x, y, z\}]$],

$\{x, -6, 6\}, \{y, -1, 4\}, \{z, -4, 4\}$,

Contours $\rightarrow \{-1, -0.5, -0.25, -0.1, -0.05, 0, 0.05, 0.1, 0.25, 0.5, 1\}$,

ContourStyle $\rightarrow \text{Table}[\{\text{Opacity}[0.5], \text{Hue}[\frac{i}{11}]\}, \{i, 0, 10\}]$,

Mesh $\rightarrow \text{None}$, **BoxRatios** $\rightarrow \{12, 5, 8\}$ (* следующий рисунок слева *)



Для двух положительных единичных зарядов контурный график, показанный на предыдущем рисунке справа, построен следующим кодом

ContourPlot3D[

Evaluate [$\varphi[\{1, 1\}, \{\{-1, 0, 0\}, \{1, 0, 0\}\}, \{x, y, z\}]$],

$\{x, -3, 3\}, \{y, 0, 2\}, \{z, -2, 2\}$,

Contours $\rightarrow \{1, 1.25, 1.5, 1.75, 2, 2.5, 3\}$,

ContourStyle $\rightarrow \text{Table}[\text{Hue}[i/7], \{i, 0, 6\}]$, **Mesh** $\rightarrow \text{None}$]

Используя функции $\varphi[\mathbf{q}, \mathbf{p}, \mathbf{r}]$ и **ContourPlot3D**, можно построить потенциал электростатического поля любого конечного множества точечных зарядов.

Пример 7.2. Рассмотрим распределение зарядов в пространстве, обладающее цилиндрической симметрией, т.е. рассмотрим электростатическое поле заряженной прямой. Электростатический потенциал прямой в трехмерном пространстве, совпадающей с осью Z , задается формулой

$$\varphi = 2q \ln \frac{1}{\rho}, \quad (3)$$

где $\rho = \sqrt{x^2 + y^2}$ и q – линейная плотность заряда вдоль линии. Решение (3) называется логарифмическим потенциалом и обладает круговой симметрией вокруг полюса в точке $\rho = 0$, в которой он обращается в бесконечность. Аналогично, потенциал однородной прямой, перпендикулярной плоскости $z=0$ и проходящей через точку (x_0, y_0) , дает плоское поле и выражается формулой.

$$\varphi(\mathbf{r}, \mathbf{p}_0) = \varphi(x, y, x_0, y_0) = 2q \ln \frac{1}{\sqrt{(x-x_0)^2 + (y-y_0)^2}} \quad (4)$$

Потенциал поля n зарядов вследствие суперпозиции силовых полей будет выражаться формулой (2).

Изобразим потенциал поля двух противоположно заряженных прямых, расположенных в точках $(-1, 0)$ и $(1, 0)$, с помощью контурного графика на плоскости XY . Для этого создадим функцию, реализующую формулу (4)

$$\varphi[\mathbf{q}, \mathbf{p}, \mathbf{r}] := \text{Sum}\left[2q[[i]] \text{Log}\left[\frac{1}{\text{Norm}[\mathbf{r} - \mathbf{p}[[i]]]}\right], \{i, \text{Length}[\mathbf{q}]\}\right];$$

Здесь \mathbf{q} представляет список значений линейных плотностей зарядов, \mathbf{p} является списком координат $\{x_i, y_i\}$ точек пересечения прямых с плоскостью $z=0$, \mathbf{r} списком имен переменных (координат в плоскости), например, $\{x, y\}$.

Вот, что дает функция $\varphi[\mathbf{q}, \mathbf{p}, \mathbf{r}]$ для двух логарифмических потенциалов с плотностями q_1, q_2 , расположенных в точках (x_1, y_1) и (x_2, y_2) .

$$\varphi[\{q_1, q_2\}, \{\{x_1, y_1\}, \{x_2, y_2\}\}, \{x, y\}] // \text{TraditionalForm}$$

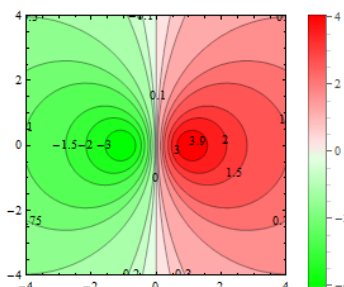
$$2q_1 \log\left(\frac{1}{\sqrt{|x-x_1|^2 + |y-y_1|^2}}\right) + 2q_2 \log\left(\frac{1}{\sqrt{|x-x_2|^2 + |y-y_2|^2}}\right)$$

Создадим список цветов для контурных кривых. Для положительных значений потенциала будем использовать красные оттенки, для отрицательных – зеленые.

$$\mathbf{c} = \text{Join}\left[\text{Table}[\text{Lighter}[\text{Green}, i/9], \{i, 0, 8\}], \text{Table}[\text{Lighter}[\text{Red}, i/9], \{i, 8, 0, -1\}]\right];$$

Функция $\text{Lighter}[\text{Color}, f]$ создает оттенок цвета Color , определяемого числом f ($0 \leq f \leq 1$). Теперь строим контурный график.

$$\text{ContourPlot}\left[\text{Evaluate}[\varphi[\{-1, 1\}, \{\{-1, 0\}, \{1, 0\}\}, \{x, y\}]], \{x, -4, 4\}, \{y, -4, 4\}, \text{PlotRangePadding} \rightarrow \text{None}, \text{Contours} \rightarrow \{-3, -2, -1.5, -1, -0.75, -0.5, -0.2, -0.1, 0, 0.1, 0.2, 0.5, 0.75, 1, 1.5, 2, 3\}, \text{ClippingStyle} \rightarrow \text{Automatic}, \text{ContourShading} \rightarrow \mathbf{c}, \text{PlotLegends} \rightarrow \text{Automatic}, \text{ContourLabels} \rightarrow \text{All}\right]$$



Ниже мы хотим смоделировать в манипуляторе поле нескольких точечных «зарядов». Но вначале мы выполним построения по шагам.

Предварительно скорректируем функцию потенциала φ так, чтобы избавиться от модулей.

$\varphi[q_-, p_-, r_-] := \text{Simplify}[\text{Sum}[2q[[i]]\text{Log}[\frac{1}{\text{Norm}[r - p[[i]]]}], \{i, \text{Length}[q]\}], \text{Join}[\{\text{Element}[r, \text{Reals}]\}, \text{Map}[\text{Element}[\#, \text{Reals}] \&, p]]];$

Тогда, например,

$\varphi[\{q_1, q_2\}, \{\{x_1, y_1\}, \{x_2, y_2\}\}, \{x, y\}] // \text{TraditionalForm}$
 $q_1(-\log((x - x_1)^2 + (y - y_1)^2)) - q_2 \log((x - x_2)^2 + (y - y_2)^2)$
или

$\varphi[\{q_1\}, \{\{x_1, y_1\}\}, \{x, y\}] // \text{TraditionalForm}$
 $q_1(-\log((x - x_1)^2 + (y - y_1)^2))$

Для вычисления градиента можно использовать функцию дифференцирования D в виде

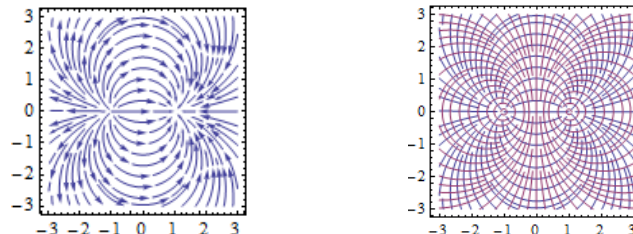
$\text{Clear}[f, x, y]; D[f[x, y], \{\{x, y\}\}]$
 $\{f^{(1,0)}[x, y], f^{(0,1)}[x, y]\}$

Вторым аргументом такого использования D должен быть список списков имен координат. Тогда, например

$D[\varphi[\{1\}, \{\{0, 0\}\}, \{x, y\}], \{\{x, y\}\}]$
 $\left\{-\frac{2x}{x^2 + y^2}, -\frac{2y}{x^2 + y^2}\right\}$

В *Mathematica* есть функция `StreamPlot`, которая умеет строить линии тока векторного поля (одного или нескольких). Например силовые линии поля двух зарядов -1 и $+1$, расположенных в точках $(-1, 0)$ и $(+1, 0)$, можно построить следующим образом (следующий рисунок слева)

$\text{StreamPlot}[\text{Evaluate}[D[\varphi[\{-1, 1\}, \{\{-1, 0\}, \{1, 0\}\}, \{x, y\}], \{\{x, y\}\}]],$
 $\{x, -3, 3\}, \{y, -3, 3\}, \text{StreamScale} \rightarrow 0.2]$



Кроме силовых линий мы будем строить линии уровня потенциала. Их можно строить по – разному, например, как контурный график. Мы поступим по – другому. Создадим векторное поле, ортогональное полю градиента, и построим его линии тока. Очевидно, они будут линиями уровня потенциала. Этот способ мы выбираем из – за того, что линии тока обеих векторных полей можно построить одной функцией `StreamLine`. Например для предыдущего поля, создаваемого двумя зарядами, следующий код строит силовые линии поля и линии уровня потенциала (предыдущий рисунок справа)

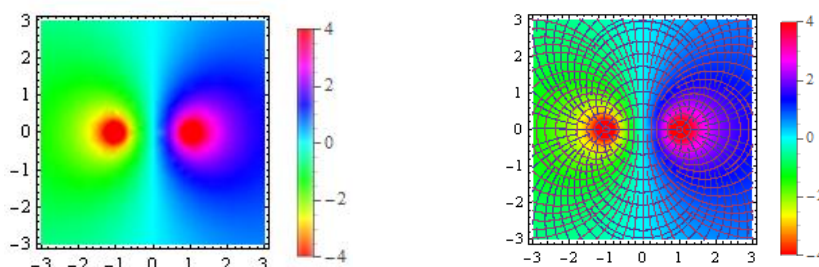
$gd = \text{Simplify}[D[\varphi[\{-1, 1\}, \{\{-1, 0\}, \{1, 0\}\}, \{x, y\}], \{\{x, y\}\}]]$
 $gdu = \text{Reverse}[gd * \{1, -1\}];$
 $ps = \text{StreamPlot}[\text{Evaluate}[\{gd, gdu\}], \{x, -3, 3\}, \{y, -3, 3\},$
 $\text{StreamScale} \rightarrow \text{None}]$

$$\left\{ \frac{4 - 4x^2 + 4y^2}{(1 - 2x + x^2 + y^2)(1 + 2x + x^2 + y^2)}, -\frac{8xy}{(1 - 2x + x^2 + y^2)(1 + 2x + x^2 + y^2)} \right\}$$

Здесь опция `StreamScale->None` отменяет рисование стрелок на линиях поля. Векторное поле градиента приведено сразу после кода, а векторное поле **gdu** ортогонально к полю градиента и получается перестановкой его координатных функций, перед одной из которых изменен знак.

Теперь строим график плотности потенциала (следующий рисунок слева)

```
pd=DensityPlot[
  Evaluate[φ[{-1, 1}, {{-1, 0}, {1, 0}}, {x, y}], {x, -3, 3}, {y, -3, 3},
  ColorFunction -> Hue, PlotLegends -> Automatic,
  PlotRange -> {-4, 4}, ClippingStyle -> Automatic]
```



Оба графика совмещаем с помощью функции `Show`. При этом первым должен рисоваться график плотности **pd**, иначе он закроет график линий тока **ps**.

```
Show[pd, ps] (* предыдущий рисунок справа *)
```

Наконец создадим манипулятор, в котором построим поле трех точечных «зарядов» в виде графика плотности с силовыми линиями и линиями уровня потенциала. Вот результирующий код.

```
DynamicModule[{φ, z, gd, gdu, m = 3, ps, pd},
```

```
  φ[q_, p_, r_] := Simplify[Sum[2q[[i]]Log[ $\frac{1}{\text{Norm}[r - p[[i]]]}$ ], {i, Length[q]}],
```

```
    Join[{Element[r, Reals]}, Map[Element[#, Reals]&, p]]];
```

```
  z[x_, y_, q_, p_] := φ[q, p, {x, y}];
```

```
  Manipulate[
```

```
    gd = Simplify[D[z[x, y, {q1, q2, q3}, p], {{x, y}}];
```

```
    gdu = Reverse[gd * {1, -1}];
```

```
    ps = StreamPlot[Evaluate[{gd, gdu}], {x, -m, m}, {y, -m, m},
      StreamScale -> None];
```

```
    pd = DensityPlot[Evaluate[z[x, y, {q1, q2, q3}, p],
      {x, -m, m}, {y, -m, m}, ColorFunction -> Hue,
      PlotLegends -> Automatic, PlotRange -> {-6, 6},
      ClippingStyle -> Automatic];
```

```
    Show[pd, ps, PlotRange -> {{-m, m}, {-m, m}},
```

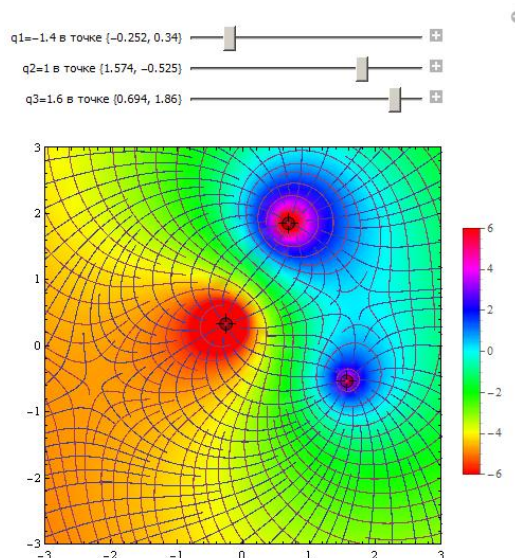
```
      PlotRangePadding -> None],
```

```
    {{q1, -1, Dynamic["q1 = " <> ToString[q1] <> " в точке" <>
      ToString[p[[1]]]}], -2, 2, 0.1},
```

```

{{q2, 1, Dynamic["q2 = " <> ToString[q2] <> " в точке" <>
ToString[p[[2]]]], -2, 2, 0.1},
{{q3, 1, Dynamic["q3 = " <> ToString[q3] <> " в точке" <>
ToString[p[[3]]]], -2, 2, 0.1},
{{p, {{-1, 0}, {1, 0}, {0, 1}}, {-2, -2}, {2, 2}, Locator},
Paneled → False, ContinuousAction → False]]

```



Основные пояснения для этого кода были выполнены выше. Код внутри функции `Manipulate` после функции `Show` выводит текстовую информацию слева от ползунков – значение заряда и его координаты, и создает три «локатора», координаты которых определяют положение зарядов. Опция **ContinuousAction** → **False** позволяет перерисовывать область внутри манипулятора только, когда вы отпускаете ползунок. Иначе перерисовка выполняется непрерывно при движении ползунков.

Вы можете менять значение всех трех «зарядов», перемещая ползунки. Перемещая «локаторы», вы меняете расположение «зарядов». Одно из возможных состояний манипулятора показано на предыдущем рисунке.

Литература.

1. Крауфорд Ф. Волны. М., 1974 г., 528 стр.
2. Тихонов А.Н., Самарский А.А. Уравнения математической физики. – М.: Наука, 1977. – 736с.
3. Доля П.Г. Периодическое продолжение функций и решение уравнения колебаний струны в системах символьной математики.// Вестник Харьк. нац. ун-та., - 2006.- № 733. Сер. ”Математическое моделирование.
4. Dolya P.G. Solution to the homogeneous boundary value problems of free vibrations of a finite string // Journal of Mathematical Physics, Analysis, Geometry, - 2008, vol.4, No 2 , pp. 237 – 251.

5. Тимошенко С.П. Колебания в инженерном деле. – М.: Гос. изд. физ.-мат. литературы, 1959.
6. Тимошенко С.П. Сопротивление материалов. Т.1 – М.: Наука, 1965.
7. Тимошенко С.П., Гудьер Дж. Теория упругости. – М.: Наука, 1979.
8. Филоненко Бородич М.М. Теория упругости. – М.: Гос. изд. физ.-мат. лит., 1959.
9. Снеддон И.Н., Берри Д.С. Классическая теория упругости. – М.: Гос. изд. физ.-мат. лит., 1961.