

Лекция 3. Элементы управления и динамика

Для наших математических экспериментов очень важны возможности *Mathematica* по созданию рисунков, которые могут меняться. На прошлой лекции мы познакомились с командой `Manipulate` - одним из способов влиять на результат вычисления, который может состоять как из картинок, так и из чисел.

В первой части лекции мы разберем возможности функции `Manipulate`.

Затем обсудим создание динамических картинок = мультфильмов, которые меняются без вмешательства человека. Это полезно, например, для понимания поведения решений дифференциальных уравнений.

Затем мы разберем несколько примеров создания таких мультфильмов.

Функция `Manipulate[]`

Эта функция позволяет визуализировать зависимость выражения от параметров. Вот некоторые форматы этой команды.

`Manipulate[expr,{u,umin,umax}]` - параметр u меняется на отрезке от $umin$ до $umax$ (ползунок `Slider`);

`Manipulate[expr,{u,{xmin,ymin},{xmax,ymax}}]` - параметр u меняется в прямоугольнике, с нижним левым концом в $\{xmin,ymin\}$ и верхним правым концом в $\{xmax,ymax\}$ (двумерный ползунок `Slider2D`);

`Manipulate[expr,{u,umin,umax,du}]` параметр u меняется ползунком с шагом du ;

`Manipulate[expr,{{u,uinit},umin,umax}]` параметру u присваивается начальное значение $uinit$;

`Manipulate[expr,{{u,uinit},umin,umax,du}]` параметру u присваивается начальное значение $uinit$;

`Manipulate[expr,{u,{u1,u2,...}}]` параметр u пробегает дискретный набор значений $u1, u2, \dots$;
если список значений короткий, то u контролируется строкой кнопок (`setter bar`);
если список длинный - то выпадающим меню (`PopUp Menu`);

`Manipulate[expr,{u,color}]` параметр u задается с помощью цветового ползунка;

`Manipulate[expr,{u,{True,False}}]` параметр u задается с помощью флаговой кнопки (`CheckBox`);

`Manipulate[expr,{u,Locator}]` параметр u задается с помощью локатора (`Locator`);

`Manipulate[expr,{{u,uinit},Locator}]` начальное положение локатора равно $uinit$.

Можно также задавать не один параметр, а сразу несколько.

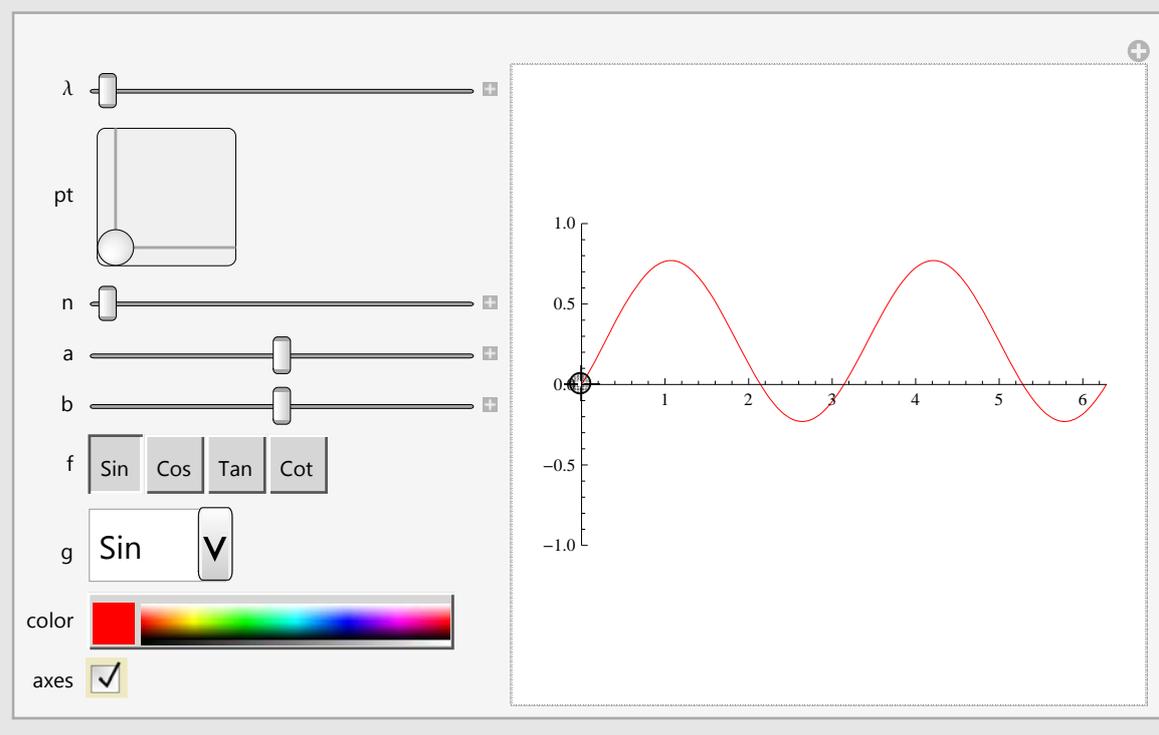
■ Забавный пример

Вот пример, который использует почти все возможности `Manipulate`

In[1]:=

```
Manipulate[Plot[λ f[a (x - loc[[1]) + b]^n g[x - loc[[1]) + loc[[2]], {x, 0, pt[[1]]},
  PlotRange → {{0, pt[[1]]}, {-pt[[2]], pt[[2]]}}, PlotStyle → {color}, Axes → axes],
  {λ, 1, 2},
  {pt, {2 π, 1}, {4 π, 2}},
  {n, 1, 5, 1},
  {{a, 1}, 0, 2},
  {{b, 1}, 0, 2, 0.2},
  {f, {Sin, Cos, Tan, Cot}},
  {g, {Sin, Cos, Tan, Cot, Sqrt, Log}},
  {color, Red},
  {axes, {True, False}},
  {{loc, {0, 0}}, Locator}
]
```

Out[1]=



- ▣ Редактируемая визуализация текущего значения параметра, задаваемого ползунком, изменение этого значения, автоматизация процесса изменения, последовательное изменение всех параметров (Autorun)

Если кликнуть на значок "+" справа от ползунка, то под ползунком возникнет окошко с текущим значением параметра, а также шесть кнопок справа от этого окошка. Окошко можно редактировать, изменяя тем самым значение параметра. Шесть кнопок позволяют задавать автоматическое изменение параметра (поэкспериментируйте с этими кнопками и разберитесь, за что они отвечают). Чтобы посмотреть зависимость от всех параметров, можно кликнуть на значок "+" в правом верхнем углу панели Manipulate и в появившемся меню выбрать Autorun. Появится управляющее окошко Autorun и параметры начнут последовательно менять свои значения. В управляющем окошке можно менять параметры демонстрации Autorun. Чтобы остановить процесс, нажмите в этом окошке на close.

- ▣ Управление скоростью движения ползунка

Если при смещении ползунка удерживать клавишу **Alt**, изменение параметра замедлится в 20 раз. Если к Alt добавить **Ctrl** или **Shift**, движение замедлится еще в 20 раз (проделайте это на примере, приведенном выше).

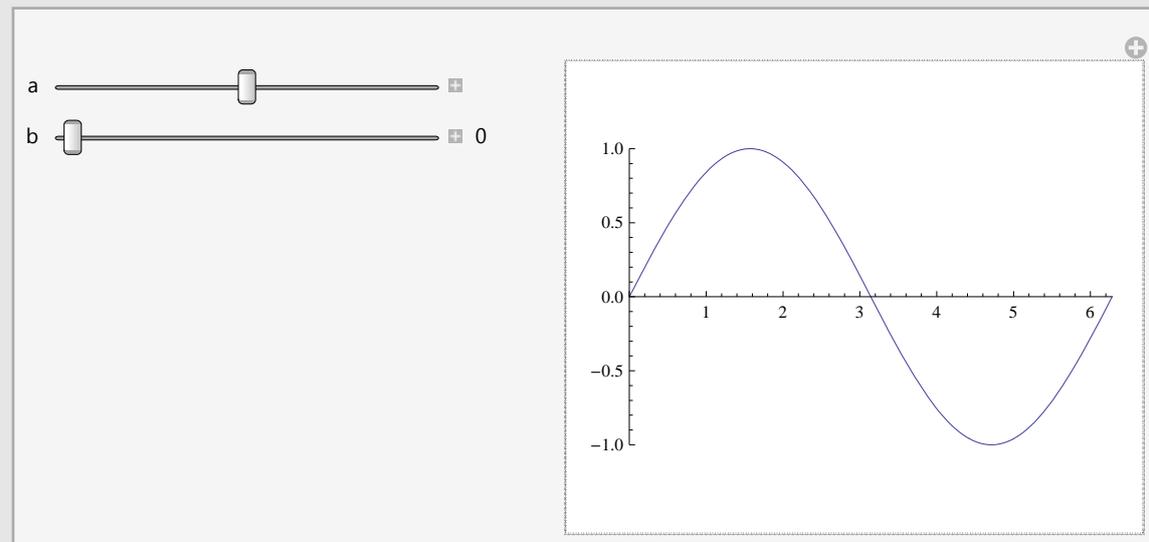
▣ **Нередактируемая визуализация значения параметра, задаваемого ползунком**

Чтобы вывести справа от ползунка текущее нередактируемое значение параметра, добавьте в список описания параметра опцию **Appearance** → **"Labeled"**

In[2]=

```
Manipulate[Plot[Sin[a x + b], {x, 0, 2 π}, PlotRange → {{0, 2 π}, {-1, 1}}],
  {{a, 1}, 0, 2},
  {b, 0, 2, Appearance → "Labeled"}
]
```

Out[2]=



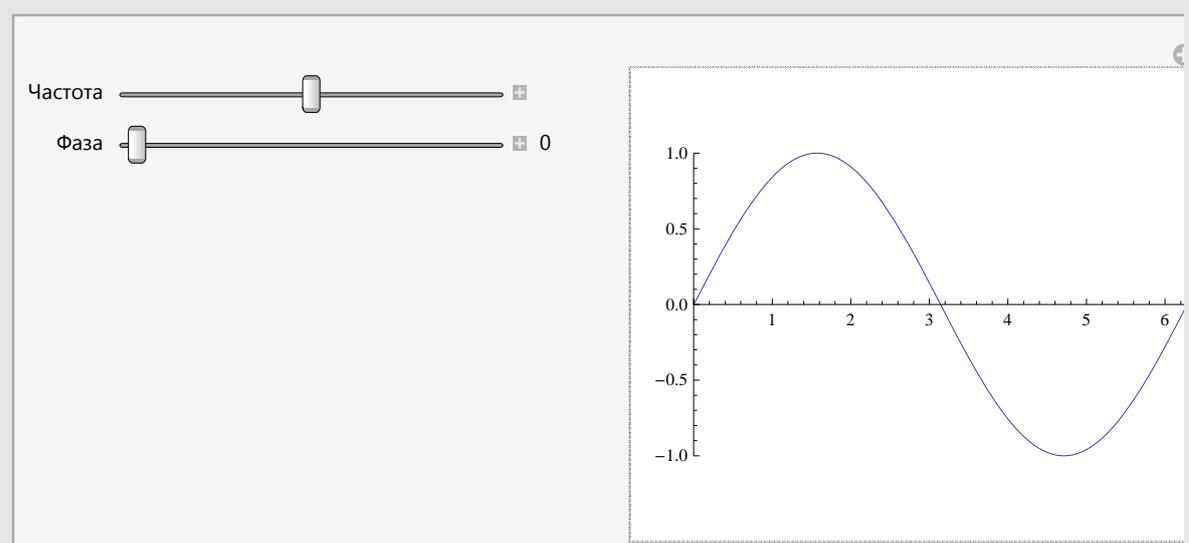
▣ **Замена имен параметров на метки**

Для этого после *обязательного* задания начального значения нужно указать название параметра.

In[3]=

```
Manipulate[Plot[Sin[a x + b], {x, 0, 2 π}, PlotRange → {{0, 2 π}, {-1, 1}}],
  {{a, 1, "Частота"}, 0, 2},
  {b, 0, "Фаза"}, 0, 2, Appearance → "Labeled"}
]
```

Out[3]=



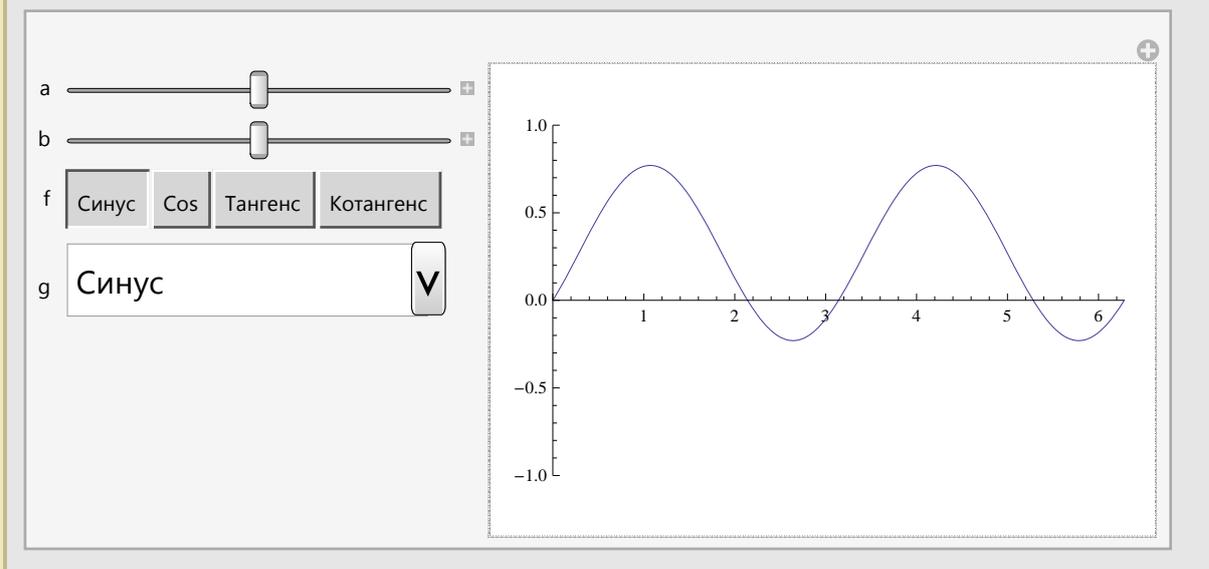
▣ Визуализация дискретных значений параметра метками

Для этого после задания дискретного значения нужно поставить стрелочку -> и указать метку (типа string)

In[4]=

```
Manipulate[Plot[f[a x + b] g[x], {x, 0, 2 π}, PlotRange → {{0, 2 π}, {-1, 1}},
  {a, 1}, 0, 2},
  {b, 1}, 0, 2, 0.2},
  {{f, Sin}, {Sin → "Синус", Cos, Tan → "Тангенс", Cot → "Котангенс"}},
  {{g, Sin}, {Sin → "Синус", Cos → "Косинус", Tan,
    Cot → "Котангенс", Sqrt → "Квадратный корень", Log → "Логарифм"}}
]
```

Out[4]=



▣ Задание вручную (переопределение заданных по умолчанию) элементов управления

Для этого в описании параметра нужно указать опцию `ControlType->ТипУправления`, где `ТипУправления` может принимать одно из следующих значений:

`Animator`, `Checkbox`, `CheckboxBar`, `ColorSetter`, `ColorSlider`, `InputField`, `Manipulator`, `PopupMenu`, `RadioButton`, `RadioButtonBar`, `Setter`, `SetterBar`, `Slider`, `Slider2D`, `TogglerBar`, `Trigger`, `VerticalSlider`, `None`.

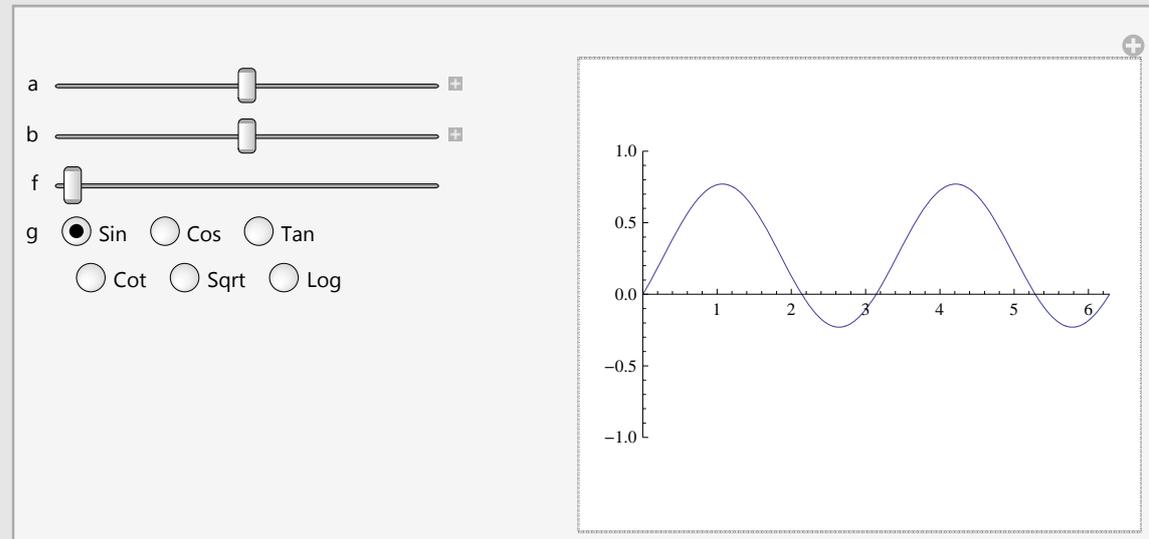
In[5]=

```

Manipulate[Plot[f[a x + b] g[x], {x, 0, 2 π}, PlotRange → {{0, 2 π}, {-1, 1}},
  {{a, 1}, 0, 2},
  {{b, 1}, 0, 2, 0.2},
  {{f, Sin}, {Sin, Cos, Tan, Cot}, ControlType → Slider},
  {{g, Sin}, {Sin, Cos, Tan, Cot, Sqrt, Log}, ControlType → RadioButton}
]

```

Out[5]=



Такое переопределение не всегда удается, обратите в следующем примере на Slider2D

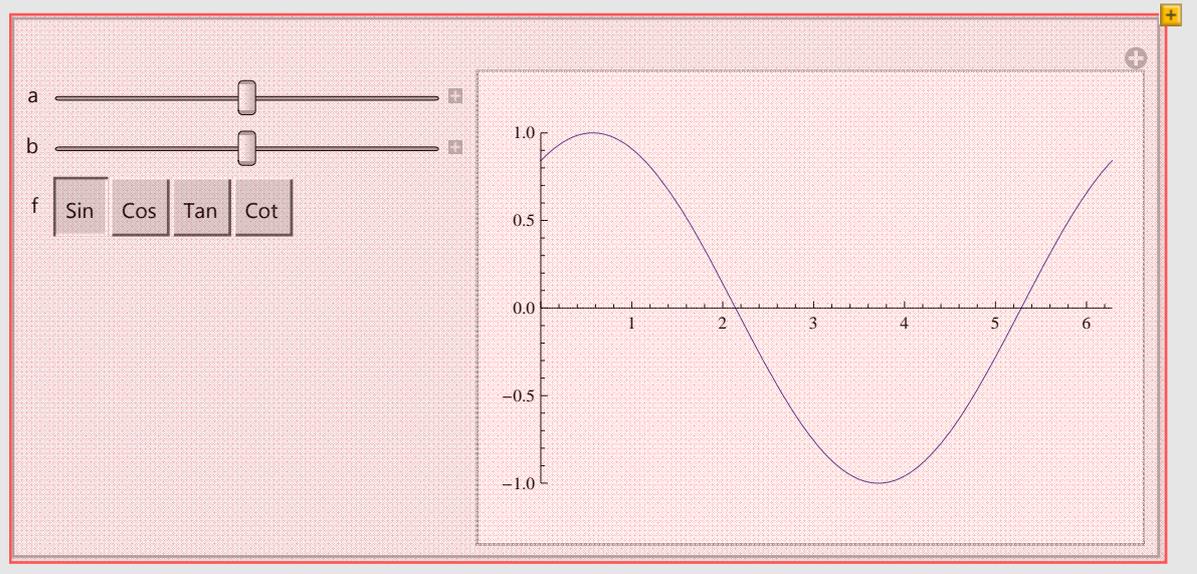
In[6]=

```

Manipulate[Plot[f[a x + b], {x, 0, 2 π}, PlotRange → {{0, 2 π}, {-1, 1}},
  {{a, 1}, 0, 2},
  {{b, 1}, 0, 2, 0.2},
  {{f, Sin}, {Sin, Cos, Tan, Cot}, ControlType → Slider2D}
]

```

Out[6]=

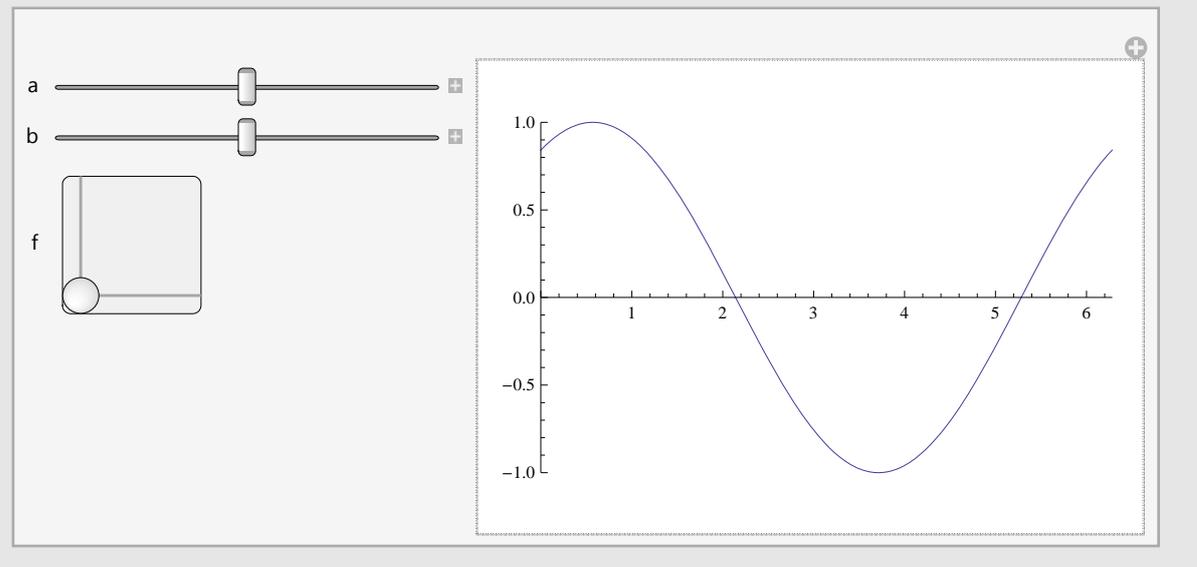


□ Использование функций в качестве элементов контроля

In[7]:=

```
Manipulate[Plot[f[a x + b], {x, 0, 2 π}, PlotRange → {{0, 2 π}, {-1, 1}}],
  {{a, 1}, 0, 2},
  {{b, 1}, 0, 2, 0.2},
  {{f, Sin}, (Which[
    pt[[1]] < 0.5 && pt[[2]] < 0.5, f = Sin,
    pt[[1]] < 0.5 && pt[[2]] ≥ 0.5, f = Cos,
    pt[[1]] ≥ 0.5 && pt[[2]] < 0.5, f = Tan,
    pt[[1]] ≥ 0.5 && pt[[2]] ≥ 0.5, f = Cot
  ] ; Slider2D[Dynamic[pt]]) &
]
```

Out[7]=



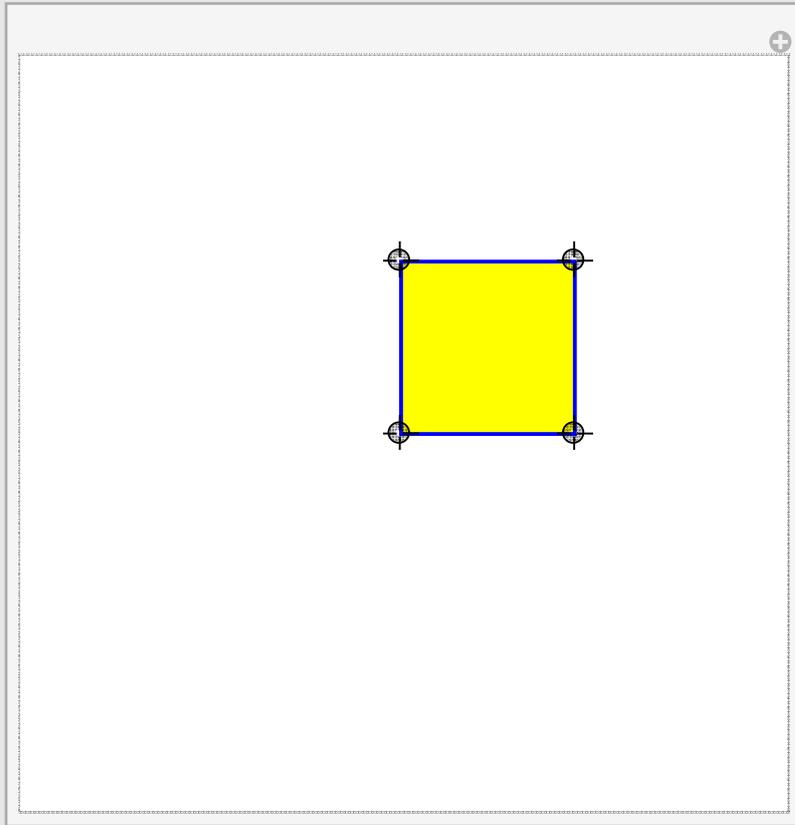
□ Интерактивное создание локаторов

Для этого в описании локатора нужно задать опцию **LocatorAutoCreate** → **True**. Теперь при удерживании клавиши Alt и клике на свободную часть поля мы будем создавать новые локаторы. При удерживании Alt и клике на существующий локатор мы удалим этот локатор.

In[8]:=

```
Manipulate[  
  Graphics[{EdgeForm[{Blue, Thick}], FaceForm[Yellow], Polygon[pt]}, PlotRange → 2],  
  {{pt, {{0, 0}, {1, 0}, {1, 1}, {0, 1}}}, Locator, LocatorAutoCreate → True]
```

Out[8]=



Пример программы рисования многоугольника на плоскости Лобачевского (Alt+Click на свободное поле добавляет вершины). Смещая локаторы, можно наблюдать за соответствующим изменением многоугольника.

In[9]:=

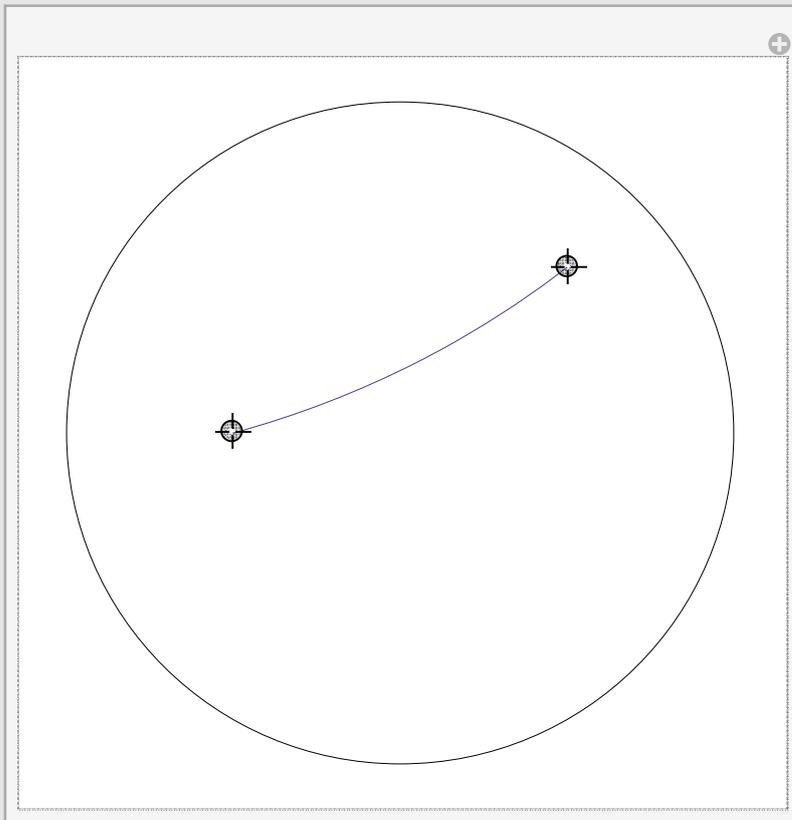
```

g[z_, φ_, z0_] := e^{i φ} \frac{z - z0}{1 - z \text{Conjugate}[z0]};
ginv[z_, φ_, z0_] := e^{-i φ} \frac{z + e^{i φ} z0}{1 + z e^{-i φ} \text{Conjugate}[z0]};

line[z1_, z2_, t_] := ginv[t g[z2, 0, z1], 0, z1];
Manipulate[res = p; Show[{
  Graphics[Circle[]],
  Table[
    ParametricPlot[With[{z = line[Complex[Sequence@@p[[i]], Complex[Sequence@@p[
      If[i == Length[p], 1, i + 1]]], t}], {Re[z], Im[z]}, {t, 0, 1}],
    {i, If[Length[p] == 2, 1, Length[p]]}
  ]], {{p, {{-1/2, 0}, {1/2, 1/2}}}, Locator, LocatorAutoCreate → True}
]

```

Out[12]=



Элементы управления (Controls) вне Manipulate

Рассмотренные нами элементы управления в команде Manipulate, можно использовать и сами по себе.

С их помощью мы можем организовать управление так, как нам удобно, если функция Manipulate предоставляет недостаточно возможностей.

Приведем некоторые примеры.

▣ Ползунок

In[13]:= `Slider[]`

Out[13]= 

Ползунок можно вставлять в формулы

In[14]:= `1 + Slider[]`

Out[14]= `1 +` 

Ползунок можно вставлять в формулы в виде ползунка, если скопировать его из вывода

In[15]:= `1 +` 

Out[15]= `1 +` 

То же самое можно сделать, пометив ползунок и выполнив команду `Ctrl+Shift+Enter`, которая вычислит и заменит на результат ровно то выражение на входе, которое было помечено. Проверьте это на примере приведенного ниже выражения

In[16]:= `1 + Slider[]`

Out[16]= `1 +` 

Однако, как видно, ползунок не участвует в вычислениях. Если хочется использовать ползунок для задания конкретных значений, можно обрмить его `DynamicSetting`, а затем скопировать результат в поле ввода. Вычисление результата будет учитывать конкретное значение, заданное ползунком.

In[17]:= `1 + DynamicSetting[`  `]`

Out[17]= `1 +` 

In[18]:= `1 +` 

Out[18]= `2.`

In[19]:= `1 +` 

Out[19]= `1.825`

Ползунок может иметь разные параметры, например

`Slider[x]` устанавливает ползунок в положение x (предполагается, что x лежит между 0 и 1)

`Slider[x, {xmin, xmax}]` устанавливает ползунок в положение x (предполагается, что x лежит между $xmin$ и $xmax$)

`Slider[x, {xmin, xmax, dx}]` устанавливает ползунок в положение x (предполагается, что x лежит между x_{\min} и x_{\max}), причем ползунок движается дискретно с шагом dx

In[20]:=

```
Slider[0.5, {0, 2, 0.5}]
```

Out[20]=

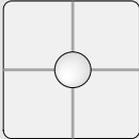


▣ Аналогично работает двумерный ползунок `Slider2D[]`

In[21]:=

```
DynamicSetting[Slider2D[]]
```

Out[21]=



▣ Еще несколько примеров

```
In[22]:= Manipulator[]
Checkbox[]
RadioButton[]
RadioButtonBar[2, {1, 2, 4, 5}]
ColorSlider[]
Graphics[Locator[{0, 0}], PlotRange -> 1]
```

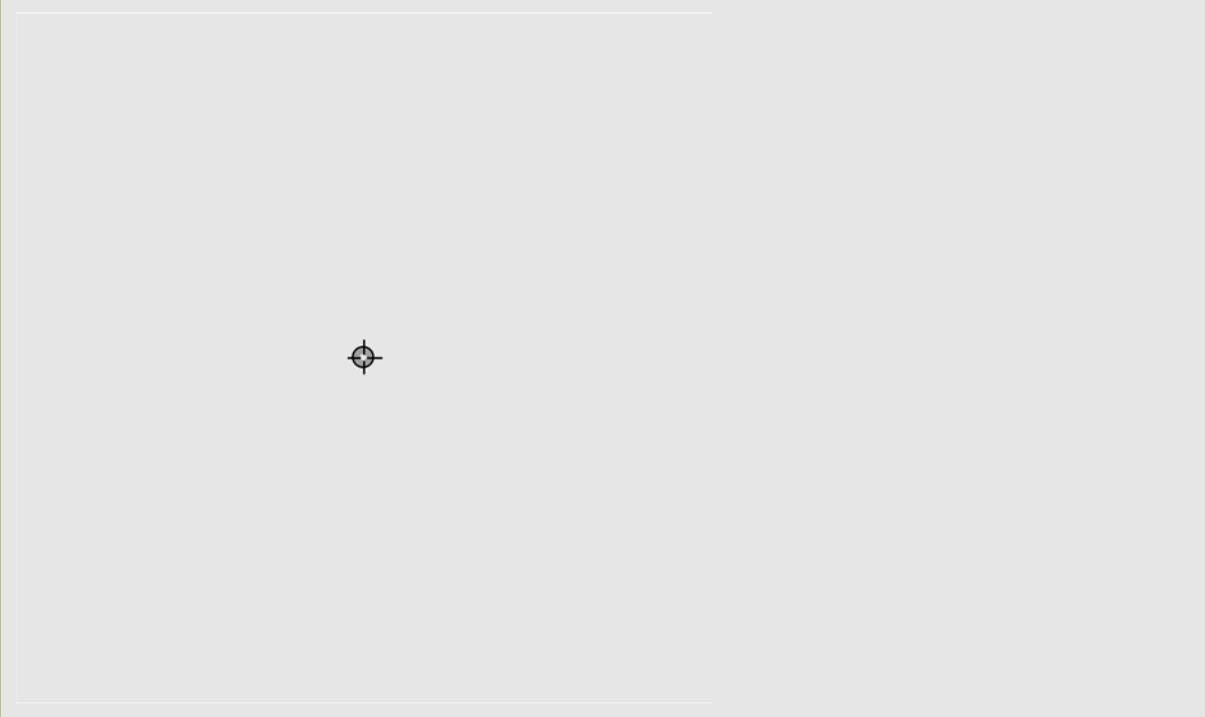
Out[22]= 

Out[23]= 

Out[24]= 

Out[25]= 

Out[26]= 

Out[27]= 

Как правило, чтобы "оживить" элементы управления, их используют вместе с оператором `Dynamic[]`, про который речь пойдет ниже.

Оператор `Dynamic[]`

▣ Пример

Обычные вводы/выводы в Mathematica статичные. Вывод результата уже вычисленного выражения не меняется при дальнейшем изменении переменных, входящих в выражение. Последнее полезно для наблюдения за историей вычислений.

```
In[28]:= x = 3;
x^2
Out[29]= 9
```

```
In[30]:= x = 5
x^2
Out[30]= 5
```

```
Out[31]= 25
```

(заметим, что первый вывод по-прежнему дает значение 9, хотя "x" уже поменялся)

Если хочется, чтобы вывод результата вычисления выражения менялся при изменении входящих переменных, можно использовать Dynamic

```
In[32]:= Dynamic[x^2]
Out[32]= x^2
```

выполним следующие команды и обратим внимание на то, как будет меняться вывод предыдущей команды

```
In[33]:= x = 7;
```

```
In[34]:= x = ∫ y dy;
```

```
In[35]:= x = Plot[x^2, {x, -1, 1}];
```

■ Динамика и элементы управления

Ползунок не влияет на значение аргумента x

```
In[36]:= x = 0.5;
```

```
In[37]:= Slider[x]
Out[37]= 
```

```
In[38]:= x
Out[38]= 0.5
```

Если ползунок использовать вместе с Dynamic, то переменная x будет динамически обновлять свое значение при перемещении ползунка

In[39]:= `{Slider[Dynamic[x]], Dynamic[x]}`

Out[39]= 

In[40]:= `x`

Out[40]= 0.5

In[41]:= `Dynamic[x]`

Out[41]= x

In[42]:= `{Slider[Dynamic[x]], x}`

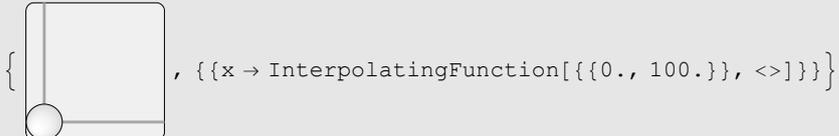
Out[42]= 

In[43]:= `{Slider[x], Dynamic[x]}`

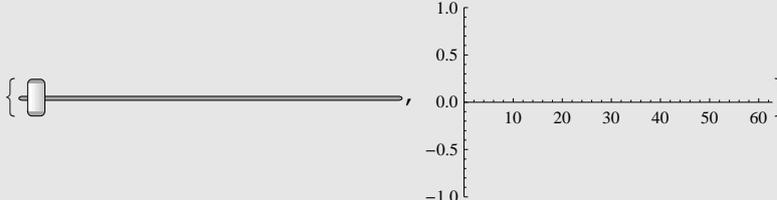
Out[43]= 

Приведем пример использования динамики в двумерном ползунке.

In[44]:= `{Slider2D[Dynamic[s]], Dynamic[s]}`

Out[44]= 

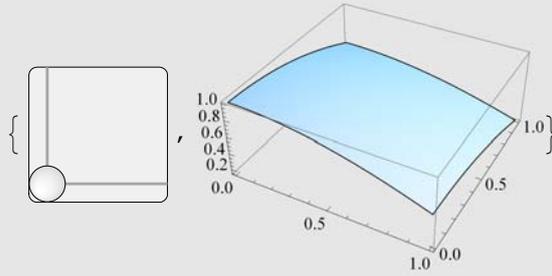
In[45]:= `{Slider[Dynamic[k]], Dynamic[Plot[Sin[k x], {x, 0, 20 π}, PlotRange → {{0, 20 π}, {-1, 1}}]]}`

Out[45]= 

In[46]:=

```
{Slider2D[Dynamic[r]],  
Dynamic[Plot3D[Cos[Norm[{x, y} - r]], {x, 0, 1}, {y, 0, 1}, Mesh -> None]]}
```

Out[46]=



Приведем примеры использования динамики в других элементах управления

In[196]=

```

Clear[x]
{Manipulator[Dynamic[x1]], Dynamic[x1]}
{Checkbox[Dynamic[x2]], Dynamic[x2]}
Map[RadioButton[Dynamic[x3], #] &, {1, 2, 3}]
{ColorSlider[Dynamic[col]], Dynamic[Graphics[{col, Disk[]}]}]}
p = {0.5, 0.5};
Graphics[Locator[Dynamic[p]], PlotRange -> 1]
Dynamic[p]

```

Out[197]=

Out[198]=

Out[199]=

Out[200]=

Out[202]=

Out[203]=

```
{0.494444, 0.2}
```

```
In[55]:= Clear[k, x, x1, x2, p]
```

■ Локализация переменных

В приводимых ниже двух примерах ползунков оба они изменяются одинаково при изменении одного из них (так как "x" - глобальная переменная). Кстати, выше была переменная x, объявленная динамической, так вот она тоже меняется, если двигать эти ползунки.

```
In[56]:= {Slider[Dynamic[x]], Dynamic[x]}
         {Slider[Dynamic[x]], Dynamic[x]}
```

```
Out[56]= {, x}
```

```
Out[57]= {, x}
```

Чтобы избавиться от такой зависимости, можно локализовать переменную "x" (по аналогии с тем, как это делалось в Module[])

```
In[58]:= DynamicModule[{x}, {Slider[Dynamic[x]], Dynamic[x]}]
         DynamicModule[{x}, {Slider[Dynamic[x]], Dynamic[x]}]
```

```
Out[58]= {, 0.}
```

```
Out[59]= {, 0.}
```

Локализация с помощью DynamicModule имеет свои преимущества, в частности, при сохранении проекта и выхода из него текущие значения локализованных переменных сохраняются (этого не происходит со значениями глобальных переменных и локальных переменных из Module).

■ Второй аргумент в Dynamic[]

Когда Dynamic[] используется вместе с ползунком в виде Slider[Dynamic[expr]], при движении ползунка все время выполняется присвоение $expr = new$, где new - значение, вычисленное по положению ползунка. Если присвоение успешно, ползунок смещается, иначе - остается на месте.

В приведенном ниже примере можно двигать лишь левый ползунок. Попытка движения правого приводит к ошибке, так как не возможно присвоить новое значение выражению $1 - x$.

```
In[60]:= DynamicModule[{x = 0}, {Slider[Dynamic[x]], Slider[Dynamic[1 - x]]}]
```

```
Out[60]= {, 
```

Однако, если использовать второй аргумент в Dynamic, то с помощью него можно задать динамику первого аргумента

```
In[61]:= DynamicModule[{x = 0}, {Slider[Dynamic[x]], Slider[Dynamic[1 - x, (x = 1 - #) &]]}]
```

```
Out[61]= {, 
```

Этим способом можно реализовать обратную функцию.

In[62]:=

```
DynamicModule[{x = 0},
  {Slider[Dynamic[x]], Slider[Dynamic[1 - 2 x, (x = (1 - #) / 2) &], {-1, 1}]}]
```

Out[62]=



Таким образом, формат команды такой : `Dynamic[expr, f]`, где `f[expr, val]` вычисляется при каждом изменении значения `val`

В приводимом примере используем тот факт, что `DateList[]`, выдающая временные данные в формате {год, месяц, день, час, минута, секунда}, так же, как и функция `RandomReal[]`, самостоятельно не обновляется. Однако при обновлении второго аргумента динамики, первый также пересчитывается.

In[63]:=

```
Slider[Dynamic[x]]
Dynamic[DateList[], Dynamic[x]]
```

Out[63]=



Out[64]=

```
{2016, 3, 22, 0, 8, 41.3762122}
```

Можно заставить самостоятельно обновляться `Dynamic[]`, задав опцию `UpdateInterval`
`UpdateInterval -> t` (обновляться не реже чем каждые `t` секунд)
`UpdateInterval -> 0` (обновляться как можно чаще)
`UpdateInterval -> Infinity` (не обновляться)

In[65]:=

```
Dynamic[DateList[], UpdateInterval -> 1]
```

Out[65]=

```
{2016, 3, 22, 0, 13, 52.2062799}
```

■ Где нужно располагать `Dynamic[]`?

Оператор `Dynamic[]` ведет себя достаточно хитро, и далеко не каждое его расположение приводит к ожидаемому результату. Опишем несколько важных принципов, которые позволяют избежать многочисленных ошибок.

1) *Mathematica* состоит из оболочки (front end), в которой мы сейчас работаем, и ядра (kernel), выполняющего вычисления выражений (перед тем, как выполнится первая команда, скажем, `2+2`, набранная в оболочке и активированная командой оболочки `shift+Enter`, *Mathematica* загружает в память свое ядро и команда обрабатывается именно ядром; в дальнейшем ядро остается в памяти и производит вычисление выражений, активизированных той же командой `shift+Enter`). Используя команду меню `Evaluation/Quit Kernel/Local` можно выгрузить ядро, завершив тем самым сеанс. Если затем выполнить команду `shift+Enter`, позиционировав предварительно курсор в какой-нибудь клетке, ядро вновь загрузится и команды из клетки будут обработаны ядром.

2) Главная особенность оператора `Dynamic[]` состоит в том, что он выполняется оболочкой, а не ядром. В частности, не возможно вычислить функцию от переменной, обрамленной командой `Dynamic[]` (хотя значение `x` подставляется в `Sin[]`, тем не менее команда `N[]` не может вычислить значение `Sin[10]`)

In[66]:=

```
x = 10;
N[Sin[Dynamic[x]]]
```

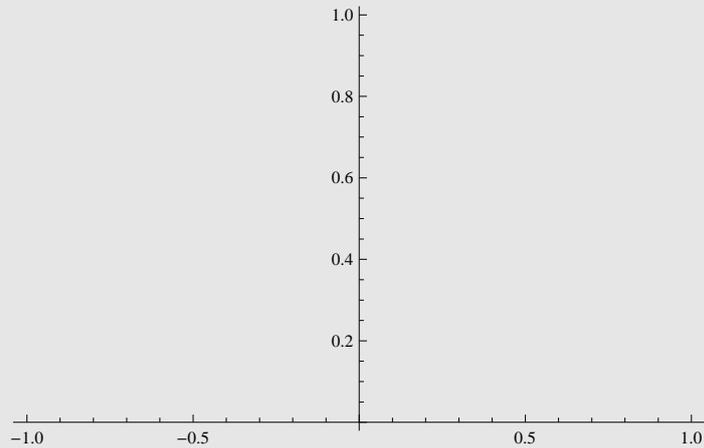
Out[67]=

```
Sin[x]
```

In[68]:=

```
x = .
Plot[Dynamic[x2], {x, -1, 1}]
```

Out[69]=



3) `Dynamic[expr]` имеет атрибут `HoldFirst`, поэтому аргумент `expr` не меняется при вычислении

In[70]:=

```
Attributes[Dynamic]
Dynamic[1 + 1] // InputForm
```

Out[70]=

```
{HoldFirst, Protected, ReadProtected}
```

Out[71]/InputForm=

```
Dynamic[1 + 1]
```

In[72]:=

```
Dynamic[Evaluate[1 + 1]] // InputForm
```

Out[72]/InputForm=

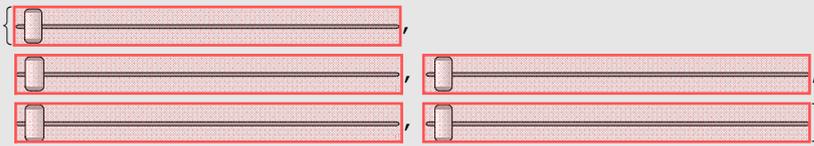
```
Dynamic[2]
```

Важное следствие: ошибки при создании списка динамических данных (временная переменная `i` внутри `data[[i]]` не меняет своего значения, потому что `Dynamic` имеет атрибут `HoldFirst`)

```
In[73]:= data = {.1, .5, .3, .9, .2};
Dynamic[data]
Table[Slider[Dynamic[data[[i]]], {i, 5}]
```

```
Out[74]:= {0.1, 0.5, 0.3, 0.9, 0.2}
```

```
Out[75]=
```

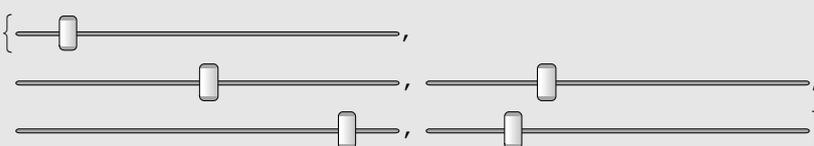


Чтобы конструкция заработала, можно использовать правило замены, чтобы итератор оказался снаружи Dynamic

```
In[76]:= data = {.1, .5, .3, .9, .2};
Dynamic[data]
Table[Slider[Dynamic[data[[i]]] /. i -> ii, {ii, 5}]
```

```
Out[77]:= {0.1, 0.5, 0.3, 0.9, 0.2}
```

```
Out[78]=
```



Еще один вариант - использовать команду `With[{x=x0,y=y0,...},expr]`, состоящую в том, что при каждом появлении символов `x`, `y` ... они заменяются на `x0`, `y0`, ...

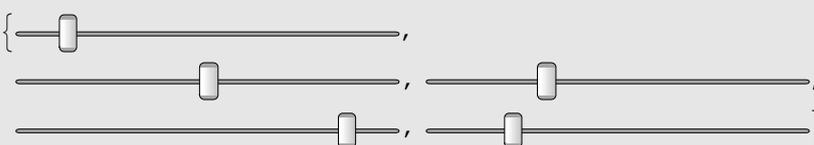
```
In[79]:= With[{y = p - 1}, 1 + y - y^2]
```

```
Out[79]:= - (-1 + p)^2 + p
```

```
In[80]:= data = {.1, .5, .3, .9, .2};
Dynamic[data]
Table[With[{i = ii}, Slider[Dynamic[data[[i]]]], {ii, 5}]
```

```
Out[81]:= {0.1, 0.5, 0.3, 0.9, 0.2}
```

```
Out[82]=
```



4) "Dynamic" не выводится при стандартном выводе, хотя на самом деле присутствует, что можно увидеть, заменив формат вывод на `InputForm`.

Кроме того, при выводе в стандартном режиме аргумент `expr` из `Dynamic[expr]` визуализируется в вычисленном виде, несмотря на то, что реально `Dynamic[expr]` представлено невычисленным выражением `expr`.

```
In[83]:= Dynamic[1 + 1]
Dynamic[1 + 1] // InputForm
```

```
Out[83]= 2
```

```
Out[84]/InputForm=
```

```
Dynamic[1 + 1]
```

```
In[85]:= x = .
```

```
In[86]:= 2 + Slider[x]
```

```
Out[86]= 2 + Slider[x]
```

```
In[87]:= Slider[x]
```

```
Out[87]= Slider[x]
```

5) При использовании Dynamic[] с элементами управления нужно придерживаться следующего правила:

первый аргумент элемента управления почти всегда обрамлен Dynamic, например, Slider[Dynamic[x]], Checkbox[Dynamic[x]], RadioButton[Dynamic[x]] ...

```
In[88]:= {Dynamic[Slider[x]], Dynamic[x]}
```

```
Out[88]= {Slider[x], x}
```

6) Расположение Dynamic в правильном месте - высокое искусство. Часто это - самый внешний оператор, хотя, как мы уже видели на примере элементов контроля, это не всегда так. Кроме того, иногда имеется свобода в расположении Dynamic, а также Dynamic могут быть вложенными в друг друга.

```
In[89]:= Dynamic[Table[x, {i, 10}]]
Table[Dynamic[x], {i, 10}]
```

```
Out[89]= {x, x, x, x, x, x, x, x, x, x}
```

```
Out[90]= {x, x, x, x, x, x, x, x, x, x}
```

В первом случае Dynamic применяется ко всей таблице, и обновление, происходящее при каждом изменении x, сопровождается пересчетом всей таблицы Table.

Во втором случае Table работает один раз, создавая 10 динамических переменных Dynamic[x], каждая из которых обновляется независимо при изменении x.

В первом случае при изменении x посылается только один сигнал ядру, требующий пересчета всей таблицы.

Во втором случае посылается много сигналов, требующих пересчет каждой динамической переменной.

Который из способов более эффективен зависит от конкретного случая.

7) Dynamic может быть расположен в правой части опций, однако далеко не во всех случаях. Например, Dynamic не может регулировать PlotPoints, потому что Plot должен знать PlotPoints до

того, как будет порождено то, что потом изобразится на экране (напомним, что Dynamic выполняется оболочкой, а не ядром). Приведем примеры возможного использования Dynamic в опциях.

(Style работает с оболочкой, настраивая вид вывода, поэтому Style может работать с Dynamic)

```
In[91]:= {Slider[Dynamic[h], {6, 100}], Dynamic[Style["Это - текст", FontSize -> h]]}
{Slider[Dynamic[h], {6, 100}], Style["Это - текст", FontSize -> Dynamic[h]]}
```

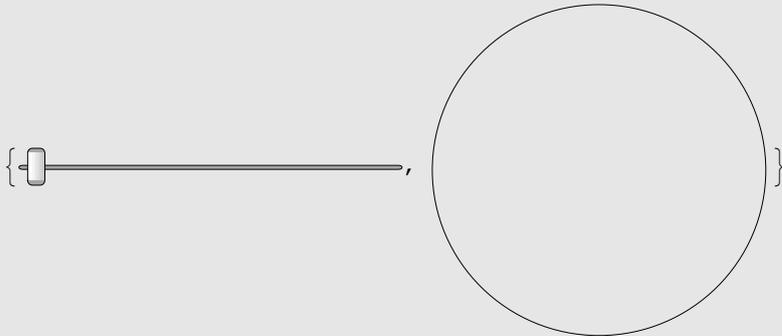
Out[91]= 

Out[92]= 

Dynamic может быть также использована с командой SetOptions.

```
In[93]:= SetOptions[Graphics, Background -> Dynamic[Hue[x]]]
{Slider[Dynamic[x]], Graphics[Circle[]]}
```

Out[93]= {AlignmentPoint -> Center, AspectRatio -> Automatic, Axes -> False, AxesLabel -> None, AxesOrigin -> Automatic, AxesStyle -> {}, Background -> Hue[x], BaselinePosition -> Automatic, BaseStyle -> {}, ColorOutput -> Automatic, ContentSelectable -> Automatic, CoordinatesToolOptions -> Automatic, DisplayFunction -> \$DisplayFunction, Epilog -> {}, FormatType -> TraditionalForm, Frame -> False, FrameLabel -> None, FrameStyle -> {}, FrameTicks -> Automatic, FrameTicksStyle -> {}, GridLines -> None, GridLinesStyle -> {}, ImageMargins -> 0., ImagePadding -> All, ImageSize -> Automatic, ImageSizeRaw -> Automatic, LabelStyle -> {}, Method -> Automatic, PlotLabel -> None, PlotRange -> All, PlotRangeClipping -> False, PlotRangePadding -> Automatic, PlotRegion -> Automatic, PreserveImageOptions -> Automatic, Prolog -> {}, RotateLabel -> True, Ticks -> Automatic, TicksStyle -> {}}

Out[94]= 

```
In[95]:= Clear[h, x]
```

Примеры использования Dynamic, DynamicModule, Manipulate для изучения решений дифференциальных уравнений

- Гармонический и круговой маятник

Как мы все слышали, уравнение гармонических колебаний обладает замечательным свойством : период колебаний (или частота) не зависит от начальных условий (не считая покоя в положении равновесия, конечно), в частности, не зависит от амплитуды колебаний. Совсем другая картина наблюдается для кругового маятника: у него период колебаний зависит от амплитуды. Лишь при малых углах отклонения уравнение колебаний кругового маятника приближенно можно заменить на уравнение гармонических колебаний.

Мы рассмотрим несколько способов изучить насколько сильно отличается уравнение колебаний кругового маятника от гармонических колебаний.

□ Гармонические колебания

In[96]=

```
s1 = NDSolve[{x'[t] == -4 x[t], x'[0] == 0, x[0] == 2}, x, {t, 0, 100}]
s2 = NDSolve[{x'[t] == -4 x[t], x'[0] == 0, x[0] == 3}, x, {t, 0, 100}]
```

Out[96]=

```
{x -> InterpolatingFunction[{{0., 100.}}, <>]}
```

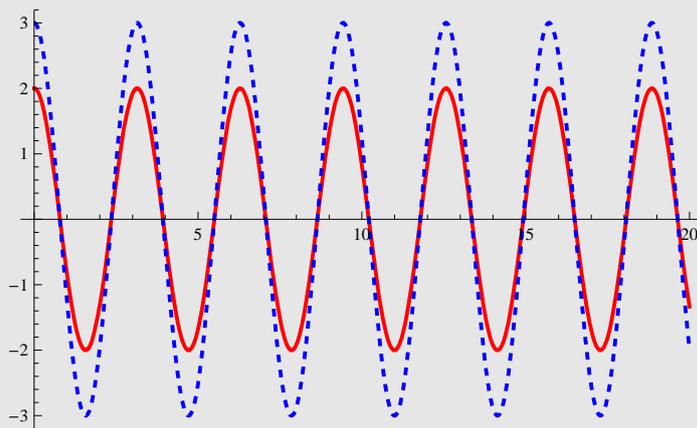
Out[97]=

```
{x -> InterpolatingFunction[{{0., 100.}}, <>]}
```

In[98]=

```
Plot[{Evaluate[x[t] /. s1], Evaluate[x[t] /. s2]}, {t, 0, 20},
PlotRange -> All, PlotStyle -> {{Red, Thick}, {Dashed, Blue, Thick}}]
```

Out[98]=



Теперь попробуем сделать так, чтобы можно было динамически менять начальное отклонение

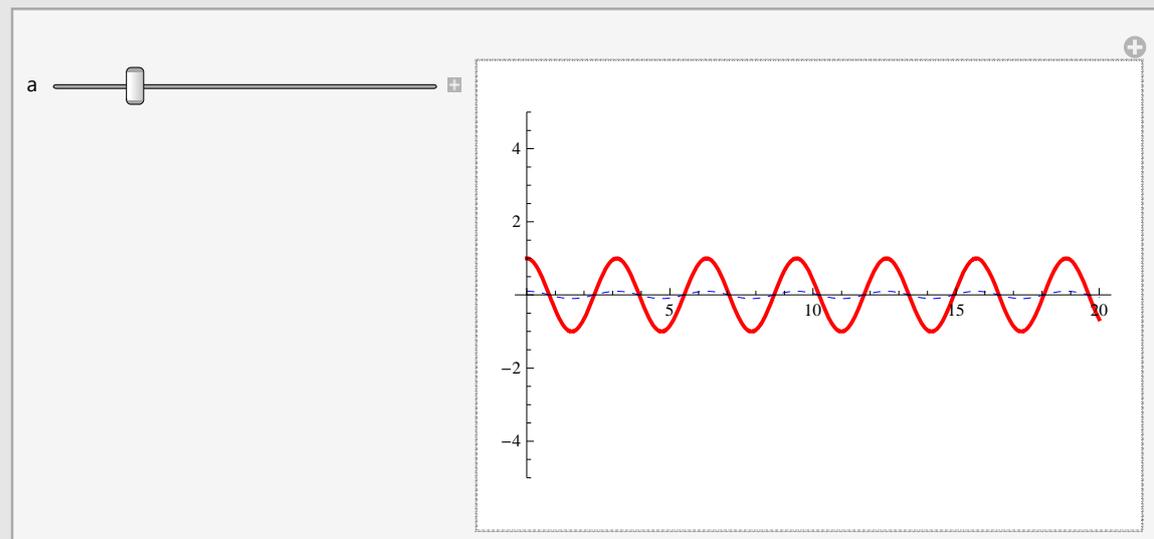
In[99]:=

```

ClearAll;
s0 = NDSolve[{x'[t] == -4 x[t], x'[0] == 0, x[0] == 1}, x, {t, 0, 100}];
DynamicModule[{a, s},
  Manipulate[
    s = NDSolve[{x'[t] == -4 x[t], x'[0] == 0, x[0] == a}, x, {t, 0, 100}];
    Plot[{Evaluate[x[t] /. s], Evaluate[x[t] /. s0]}, {t, 0, 20}, PlotRange -> 5,
      PlotStyle -> {{Red, Thick}, {Dashed, Blue, Thin}}, {{a, 1}, 0.1, 5}
    ]
  ]
]

```

Out[101]=



А теперь сюрприз для тех, кто не любит смотреть на графики. Моделируем реальный эксперимент!

Только обратите внимание, что эксперимент будет длиться только до тех пор, пока tt не дойдет до 100 - значения аргумента, для которого еще вычислены решения $s0[t]$ и $s[t]$.

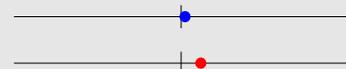
In[102]:=

```

ClearAll;
s0 = NDSolve[{x'[t] == -4 x[t], x'[0] == 0, x[0] == 1}, x, {t, 0, 100}];
s = NDSolve[{x'[t] == -4 x[t], x'[0] == 0, x[0] == 2}, x, {t, 0, 100}];
tt = 0;
Dynamic[tt += 0.01;
  Column[{
    Graphics[{Line[{{-3, 0}, {3, 0}}], Line[{{0, -0.2}, {0, 0.2}}],
      Blue, Disk[Evaluate[x[tt] /. s0] ~Join~ {0}, 0.1]}],
    Graphics[{Line[{{-3, 0}, {3, 0}}], Line[{{0, -0.2}, {0, 0.2}}],
      Red, Disk[Evaluate[x[tt] /. s] ~Join~ {0}, 0.1]}]
  ],
  UpdateInterval -> 1
]

```

Out[106]=



Ну и наконец, добавим в наш эксперимент возможность менять начальное отклонение.

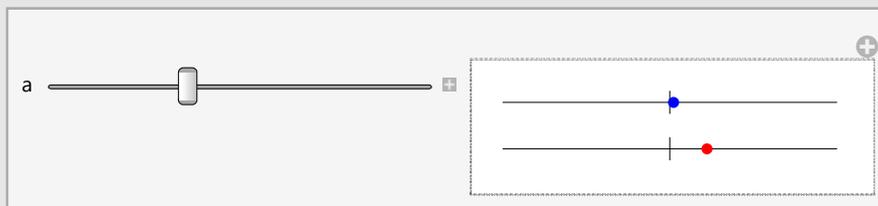
In[107]:=

```

ClearAll;
s0 = NDSolve[{x'[t] == -4 x[t], x'[0] == 0, x[0] == 1}, x, {t, 0, 100}];
DynamicModule[{a, s},
  Manipulate[
    s = NDSolve[{x'[t] == -4 x[t], x'[0] == 0, x[0] == a}, x, {t, 0, 100}];
    tt = 0;
    Dynamic[tt += 0.01;
      Column[{
        Graphics[{Line[{{-3, 0}, {3, 0}}], Line[{{0, -0.2}, {0, 0.2}}],
          Blue, Disk[Evaluate[x[tt] /. s0] ~Join~ {0}, 0.1]}],
        Graphics[{Line[{{-3, 0}, {3, 0}}], Line[{{0, -0.2}, {0, 0.2}}],
          Red, Disk[Evaluate[x[tt] /. s] ~Join~ {0}, 0.1]}]
      ]
    , UpdateInterval -> 1
  ]
  , {{a, 1}, 0.1, 2.7}
]
]

```

Out[109]=



▣ Круговой маятник

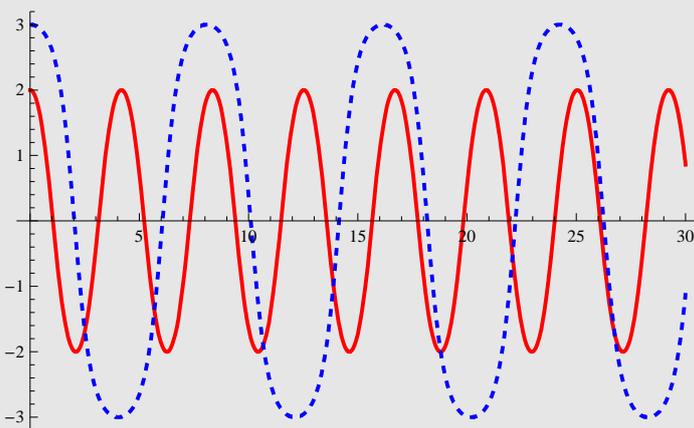
In[110]:=

```

s1 = NDSolve[{x'[t] == -4 Sin[x[t]], x'[0] == 0, x[0] == 2}, x, {t, 0, 100}];
s2 = NDSolve[{x'[t] == -4 Sin[x[t]], x'[0] == 0, x[0] == 3}, x, {t, 0, 100}];
Plot[{Evaluate[x[t] /. s1], Evaluate[x[t] /. s2]}, {t, 0, 30},
  PlotRange -> All, PlotStyle -> {{Red, Thick}, {Dashed, Blue, Thick}}]

```

Out[112]=



Теперь попробуем сделать так, чтобы можно было динамически менять начальное отклонение. Синий пунктирный - маятник с малым отклонением 0.1, у красного сплошного отклонение меняется слайдером.

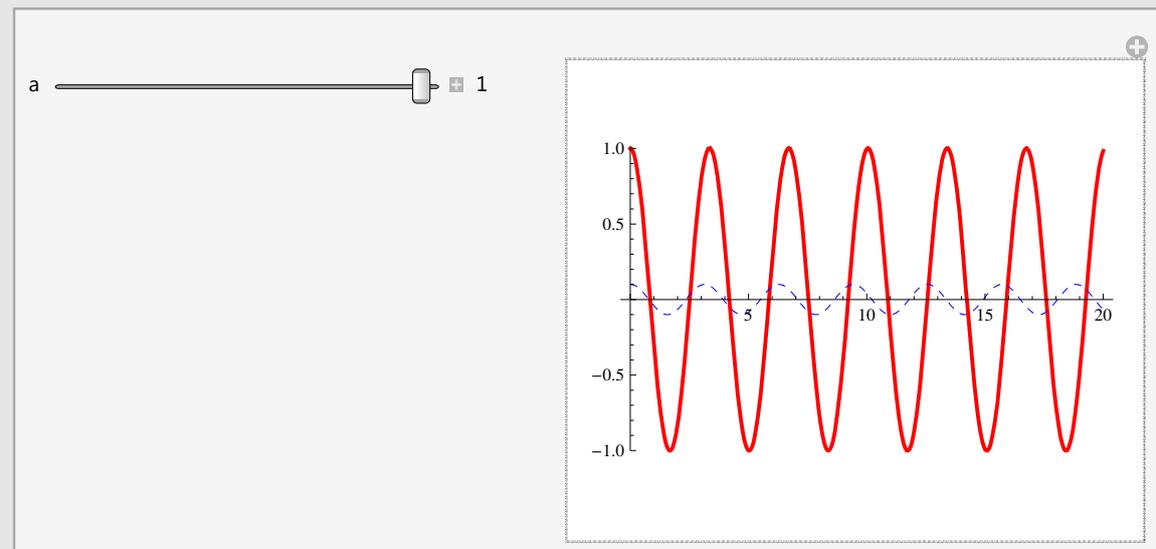
In[113]:=

```

ClearAll;
s0 = NDSolve[{x'[t] == -4 Sin[x[t]], x'[0] == 0, x[0] == 0.1}, x, {t, 0, 100}];
DynamicModule[{a, s},
  Manipulate[
    s = NDSolve[{x'[t] == -4 Sin[x[t]], x'[0] == 0, x[0] == a}, x, {t, 0, 100}];
    Plot[{Evaluate[x[t] /. s], Evaluate[x[t] /. s0]}, {t, 0, 20},
      PlotRange -> 1, PlotStyle -> {{Red, Thick}, {Dashed, Blue, Thin}},
      , {{a, 1}, 0.01, 1, Appearance -> "Labeled"}
    ]
  ]

```

Out[115]=



А теперь сюрприз для тех, кто не любит смотреть на графики. Моделируем реальный эксперимент!

Только обратите внимание, что эксперимент будет длиться только до тех пор, пока tt не дойдет до 100 - значения аргумента, для которого еще вычислены решения $s_0[t]$ и $s[t]$. Бороться с этим не будем, хотя это не является хорошим стилем программирования.

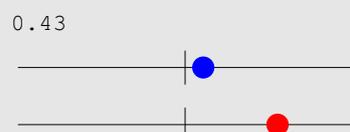
In[116]:=

```

ClearAll;
s0 = NDSolve[{x'[t] == -4 Sin[x[t]], x'[0] == 0, x[0] == 0.1}, x, {t, 0, 100}];
s = NDSolve[{x'[t] == -4 Sin[x[t]], x'[0] == 0, x[0] == 0.5}, x, {t, 0, 100}];
tt = 0;
Dynamic[tt += 0.01;
  Column[{tt,
    Graphics[{Line[{{-0.6, 0}, {0.6, 0}}], Line[{{0, -0.06}, {0, 0.06}}],
      Blue, Disk[Evaluate[x[tt] /. s0] ~Join~ {0}, 0.04]}],
    Graphics[{Line[{{-0.6, 0}, {0.6, 0}}], Line[{{0, -0.06}, {0, 0.06}}],
      Red, Disk[Evaluate[x[tt] /. s] ~Join~ {0}, 0.04]}]
  ],
  UpdateInterval -> 1
]

```

Out[120]=



Добавим в наш эксперимент возможность менять начальное отклонение.

Обратите внимание на нормировочные коэффициенты у координат маятников - это сделано для

наглядности,
иначе при малом начальном отклонении движение будет еле заметным.

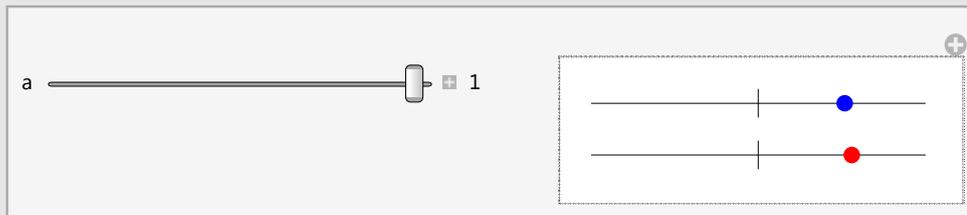
In[121]=

```

ClearAll;
s0 = NDSolve[{x''[t] == -4 Sin[x[t]], x'[0] == 0, x[0] == 0.1}, x, {t, 0, 100}];
DynamicModule[{a, s},
  Manipulate[
    s = NDSolve[{x''[t] == -4 Sin[x[t]], x'[0] == 0, x[0] == a}, x, {t, 0, 100}];
    tt = 0;
    Dynamic[tt += 0.01;
      Column[{
        Graphics[{Line[{{-1.2, 0}, {1.2, 0}}], Line[{{0, -0.1}, {0, 0.1}}],
          Blue, Disk[10 Evaluate[x[tt] /. s0] ~Join~ {0}, 0.06]}],
        (*Растянули немного эталонный маятник, чтобы его было лучше видно*)
        Graphics[{Line[{{-1.2, 0}, {1.2, 0}}], Line[{{0, -0.1}, {0, 0.1}}],
          Red, Disk[1/a (Evaluate[x[tt] /. s] ~Join~ {0}), 0.06]}]
      ]},
    UpdateInterval -> 1
  ],
  {{a, 1}, 0.01, 1, 0.01, Appearance -> "Labeled"}
]
]

```

Out[123]=



Наконец, сравним гармонический маятник с круговым

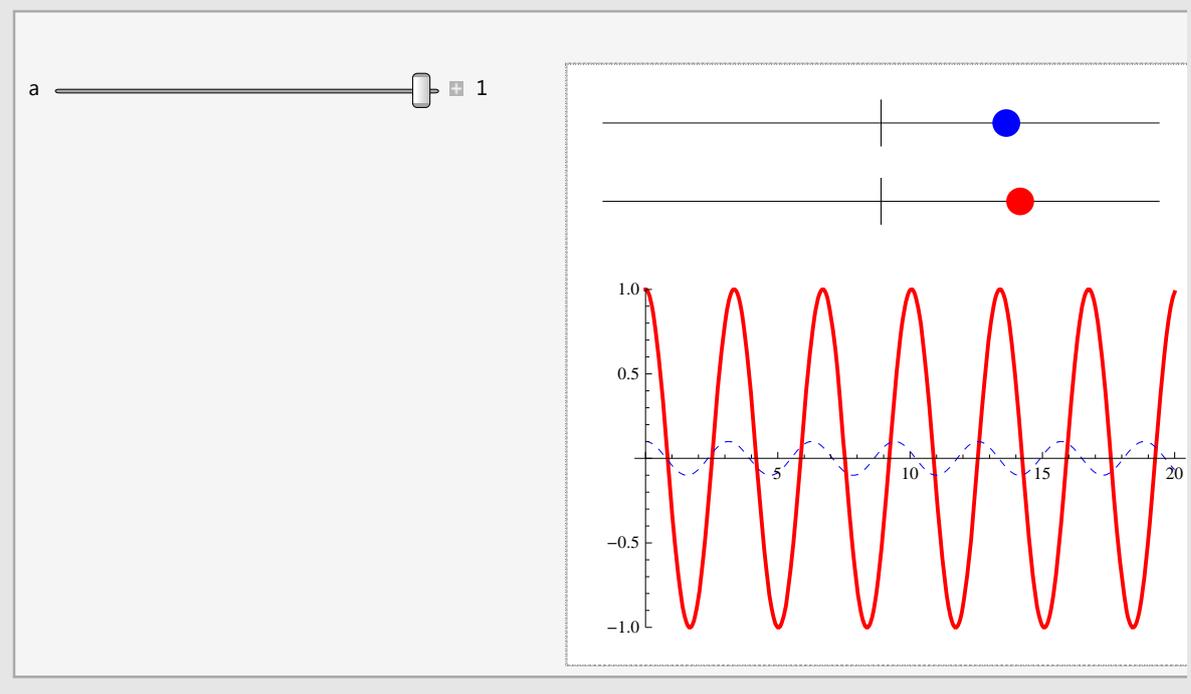
In[124]=

```

ClearAll;
s0 = NDSolve[{x'[t] == -4 x[t], x'[0] == 0, x[0] == 0.1}, x, {t, 0, 100}];
DynamicModule[{a, s},
  Manipulate[
    s = NDSolve[{x'[t] == -4 Sin[x[t]], x'[0] == 0, x[0] == a}, x, {t, 0, 100}];
    tt = 0;
    Row[{
      Dynamic[tt += 0.01;
        Column[{
          Graphics[{Line[{{-1.2, 0}, {1.2, 0}}], Line[{{0, -0.1}, {0, 0.1}}], Blue,
            Disk[10 Evaluate[x[tt] /. s0]~Join~{0}, 0.06]}, ImageSize -> 300],
          (*Растянули немного эталонный маятник, чтобы его было лучше видно*)
          Graphics[{Line[{{-1.2, 0}, {1.2, 0}}], Line[{{0, -0.1}, {0, 0.1}}], Red,
            Disk[1/a (Evaluate[x[tt] /. s]~Join~{0}), 0.06]}, ImageSize -> 300]
        }],
      UpdateInterval -> 1
    ],
    Plot[{Evaluate[x[t] /. s], Evaluate[x[t] /. s0]}, {t, 0, 20}, PlotRange -> 1,
      PlotStyle -> {{Red, Thick}, {Dashed, Blue, Thin}}, ImageSize -> 300]
  ],
  {a, 1}, 0.01, 1, 0.01, Appearance -> "Labeled"
]
]

```

Out[126]=



■ Система Лотки--Вольтерра

Теперь научимся рисовать решения систем дифференциальных уравнений на примере системы Лотки-Вольтерра, описывающей динамику популяции хищники - жертвы.

S = Sheep

W=Wolf

In[127]=

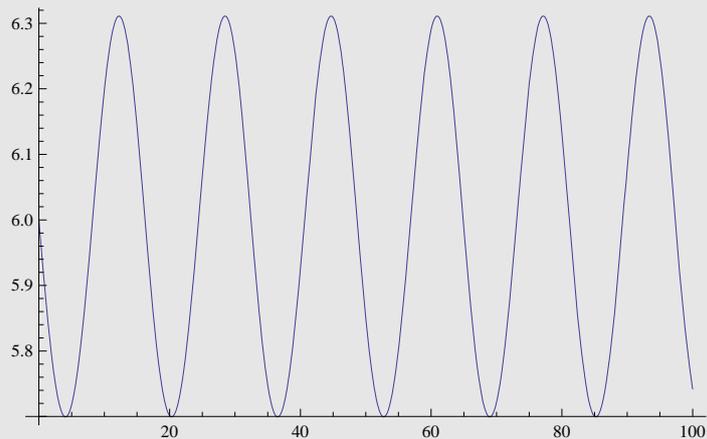
```

ClearAll;
AA = 0.5;
BB = 0.1;
CC = 0.3;
DD = 0.05;

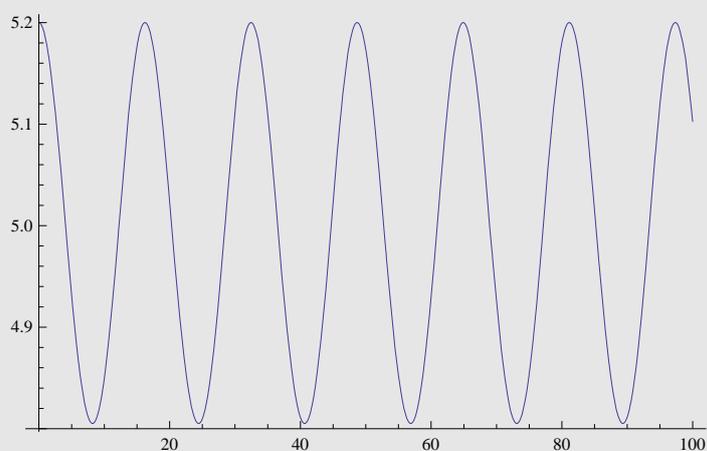
W0 = 1.04 AA / BB;
S0 = CC / DD;
Window = 1;
T = 100;
(* положение равновесия
   W0=AA/BB; S0=CC/DD; *)
res = NDSolve[{
  S'[t] == (AA - BB W[t]) S[t],
  W'[t] == (-CC + DD S[t]) W[t], (*Вот это уравнения Лотки-Вольтерра*)
  S[0] == S0, W[0] == W0
},
  {S, W}, {t, 0, T}];
Plot[Evaluate[S[t] /. res], {t, 0, T}]
Plot[Evaluate[W[t] /. res], {t, 0, T}]
ParametricPlot[Evaluate[{S[t], W[t]} /. res], {t, 0, T/10}, PlotStyle -> {Red, Thick},
  PlotRange -> {{S0 - Window, S0 + Window}, {W0 - Window, W0 + Window}}
]

```

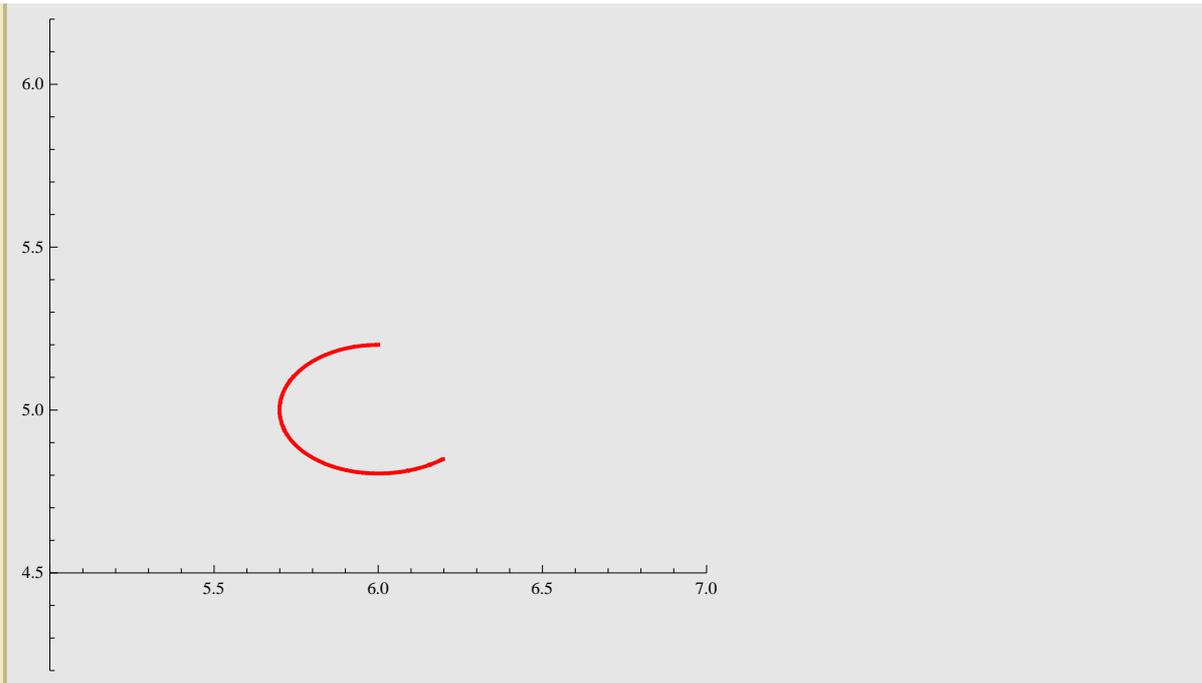
Out[137]=



Out[138]=



Out[139]=



Теперь покрасим траекторию так, чтобы цвет соответствовал времени.
От синего к красному - из прошлого в будущее.

In[140]=

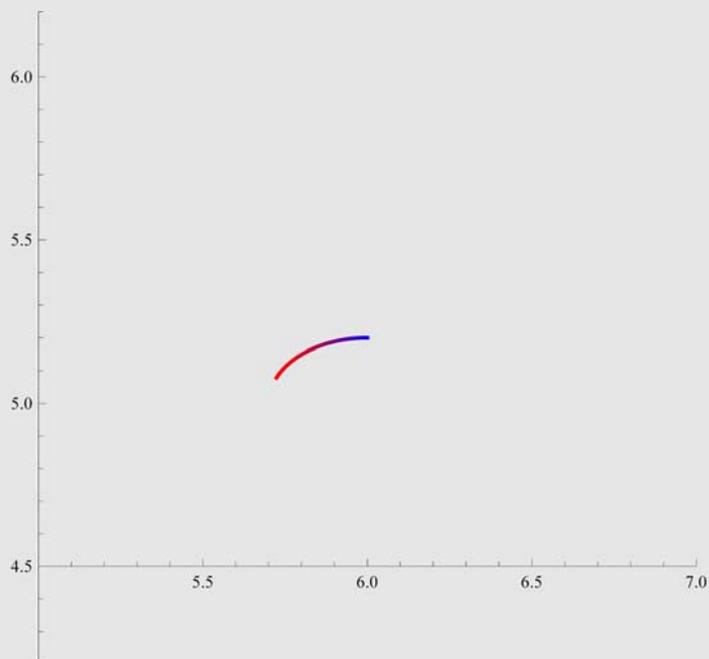
```

ClearAll;
AA = 0.5;
BB = 0.1;
CC = 0.3;
DD = 0.05;

W0 = 1.04 AA / BB;
S0 = CC / DD;
Window = 1;
T = 100;
TPlot = 3;
(* положение равновесия
   W0=AA/BB; S0=CC/DD; *)
res = NDSolve[{
  S'[t] == (AA - BB W[t]) S[t],
  W'[t] == (-CC + DD S[t]) W[t], (*Вот это уравнения Лотки-Вольтерра*)
  S[0] == S0, W[0] == W0
},
  {S, W}, {t, 0, T}];
(*
Plot[Evaluate[S[t]/.res], {t, 0, T}]
Plot[Evaluate[W[t]/.res], {t, 0, T}] *)
ParametricPlot[Evaluate[{S[t], W[t]} /. res], {t, 0, TPlot}, PlotStyle -> {Thick},
  PlotRange -> {{S0 - Window, S0 + Window}, {W0 - Window, W0 + Window}},
  ColorFunction -> Function[{x, y, u}, Blend[{Red, Blue}, (1 - u)^2]]
]

```

Out[151]=



In[152]=

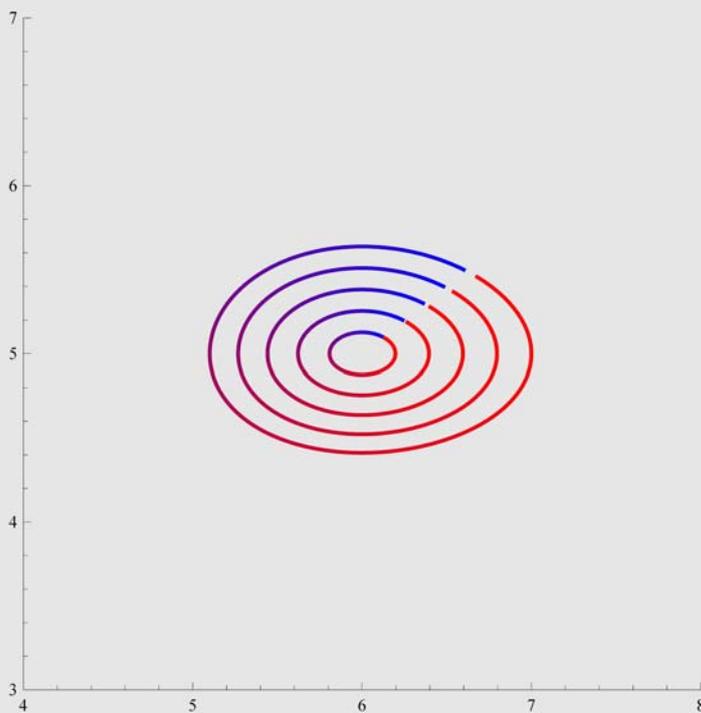
```

ClearAll;
AA = 0.5;
BB = 0.1;
CC = 0.3;
DD = 0.05;

W0 = AA / BB;
S0 = CC / DD;
DW = 0.02 W0;
DS = 0.02 S0;
InitialValues = Table[{S0 + DS i, W0 + DW i}, {i, 1, 5}];
Window = 2;
T = 100;
TPlot = 16;
(* положение равновесия
   W0=AA/BB; S0=CC/DD; *)
res = Table[NDSolve[{
  S'[t] == (AA - BB W[t]) S[t],
  W'[t] == (-CC + DD S[t]) W[t], (*Вот это уравнения Лотки-Вольтерра*)
  S[0] == InitialValues[[i]][[1]], W[0] == InitialValues[[i]][[2]]
},
  {S, W}, {t, 0, T}], {i, 1, 5}];
(*
Plot[Evaluate[S[t]/.res], {t, 0, T}]
Plot[Evaluate[W[t]/.res], {t, 0, T}] *)
ParametricPlot[Evaluate[{S[t], W[t]}/.res], {t, 0, TPlot}, PlotStyle -> {Thick},
  PlotRange -> {{S0 - Window, S0 + Window}, {W0 - Window, W0 + Window}},
  ColorFunction -> Function[{x, y, u}, Blend[{Red, Blue}, (1 - u)^2]]
]

```

Out[166]=



Добавим динамики, чтобы можно было менять начальное возмущение (отклонение) от положения равновесия и максимальное время, до которого мы рисуем траектории.

In[167]:=

```

ClearAll;
AA = 0.5;
BB = 0.1;
CC = 0.3;
DD = 0.05;

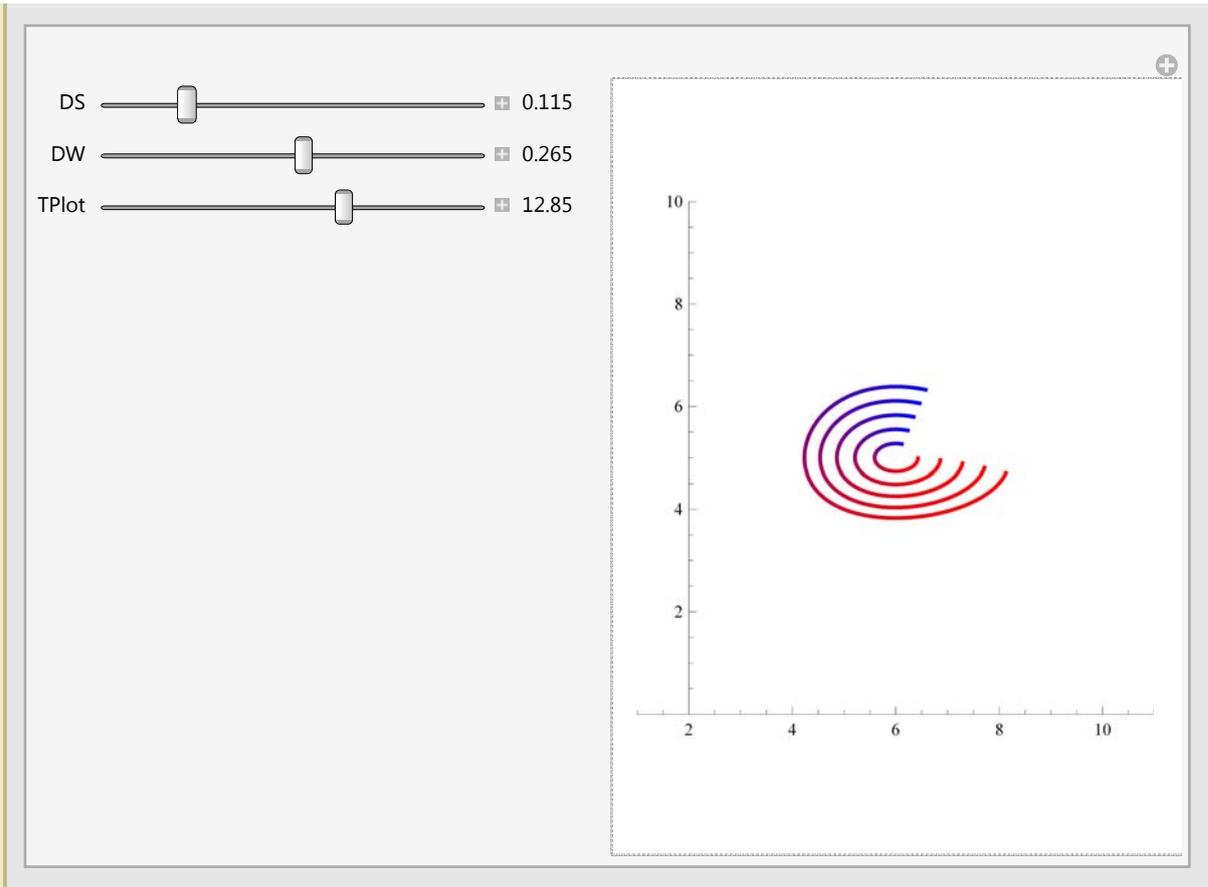
Window = 2;
T = 100;
TPlot = 16;

W0 = AA / BB;
S0 = CC / DD;
Manipulate[
  (*DW=0.02 W0;
  DS=0.02 S0;*)
  InitialValues = Table[{S0 + DS i, W0 + DW i}, {i, 1, 5}];
  (* положение равновесия
  W0=AA/BB; S0=CC/DD; *)
  res = Table[NDSolve[{
    S'[t] == (AA - BB W[t]) S[t],
    W'[t] == (-CC + DD S[t]) W[t], (*Вот это уравнения Лотки-Вольтерра*)

    S[0] == InitialValues[[i]][[1]], W[0] == InitialValues[[i]][[2]]
  },
  {S, W}, {t, 0, T}], {i, 1, 5}];
  (*
  Plot[Evaluate[S[t]/.res], {t, 0, T}]
  Plot[Evaluate[W[t]/.res], {t, 0, T}] *)
  ParametricPlot[Evaluate[{S[t], W[t]}/.res], {t, 0, TPlot}, PlotStyle -> {Thick},
  PlotRange -> {{S0 - Window, S0 + Window}, {W0 - Window, W0 + Window}},
  ColorFunction -> Function[{x, y, u}, Blend[{Red, Blue}, (1 - u)^2]]
  ]
  , {{DS, 0}, 0, S0 / 10, Appearance -> "Labeled"},
  {{DW, 0}, 0, W0 / 10, Appearance -> "Labeled"},
  {{TPlot, 1}, 0, 20, Appearance -> "Labeled"}
]

```

Out[177]=



И последнее уточнение. Пусть внутри каждого из видов есть конкуренция. Тогда вид уравнения немного меняется.

Коэффициенты описывающие конкуренцию добавим в управляемые параметры.

In[178]:=

```

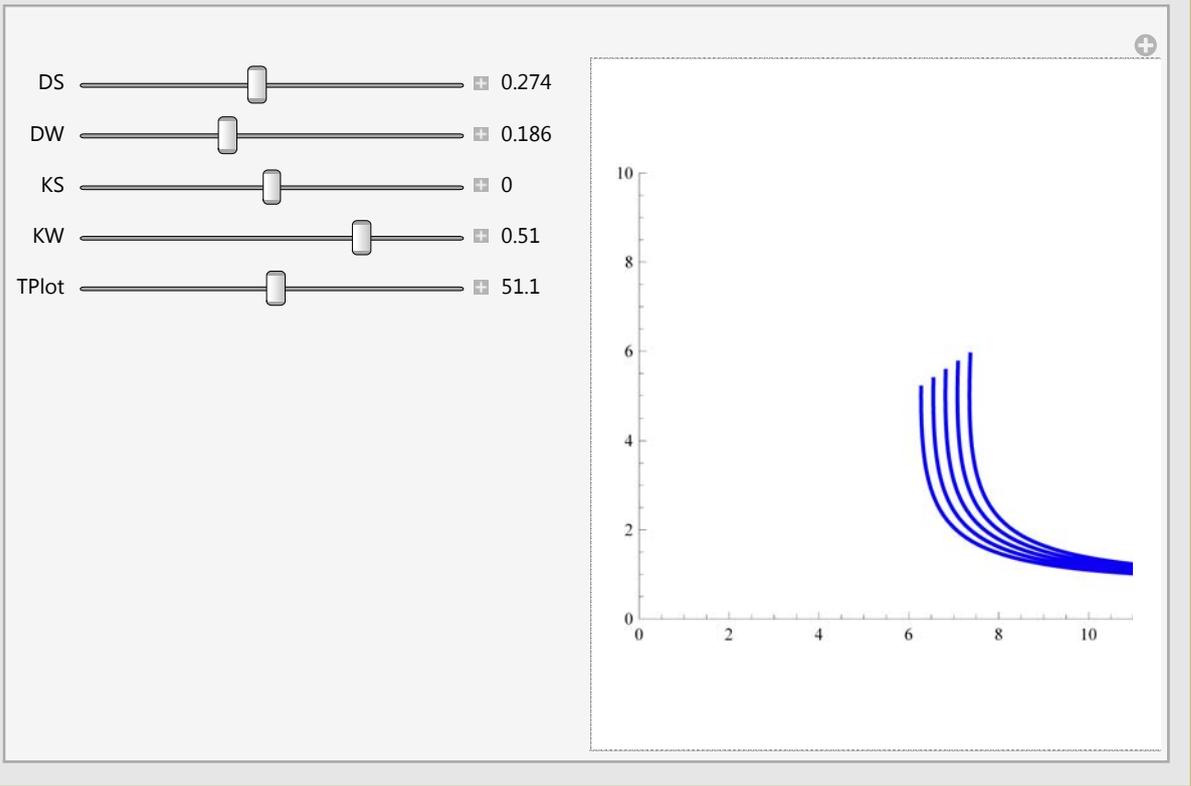
ClearAll;
AA = 0.5;
BB = 0.1;
CC = 0.3;
DD = 0.05;

Window = 5;
T = 100;
TT = 100;

W0 = AA / BB;
S0 = CC / DD;
Manipulate[
  InitialValues = Table[{S0 + DS i, W0 + DW i}, {i, 1, 5}];
  (*это список начальных отклонений от положения равновесия*)
  (* положение равновесия
    W0=AA/BB; S0=CC/DD; *)
  res = Table[NDSolve[{
    S'[t] == (AA - BB W[t] - KS S[t]) S[t], W'[t] == (-CC + DD S[t] - KW W[t]) W[t],
    (*Вот это уравнения Лотки-Вольтерра с внутривидовой конкуренцией*)
    S[0] == InitialValues[[i]][[1]], W[0] == InitialValues[[i]][[2]]
  },
    {S, W}, {t, 0, T}], {i, 1, 5}];
  (*
  Plot[Evaluate[S[t]/.res], {t, 0, T}]
  Plot[Evaluate[W[t]/.res], {t, 0, T}] *)
  ParametricPlot[Evaluate[{S[t], W[t]}/.res], {t, 0, TPlot},
    PlotStyle -> {Thick}, PlotRange -> {{0, S0 + Window}, {0, W0 + Window}},
    ColorFunction -> Function[{x, y, u}, Blend[{Red, Blue}, (1 - u)^2]]
  ]
  , {{DS, 0}, 0, S0 / 10, Appearance -> "Labeled"},
  {{DW, 0}, 0, W0 / 10, Appearance -> "Labeled"},
  {{KS, 0}, -1, 1, 0.01, Appearance -> "Labeled"},
  {{KW, 0}, -1, 1, 0.01, Appearance -> "Labeled"},
  {{TPlot, 1}, 0.1, TT, Appearance -> "Labeled"}
]

```

Out[188]=



- Другие примеры

- Для кривой γ визуализировать ее каустику (множество центров кривизны) и динамику волновых фронтов (эквидистант). Убедиться, что особенности волновых фронтов замечают каустику.

In[189]=

```

SetAttributes[wave, HoldFirst];
wave[{x_[t_, px_], y_[t_, py_]}, {px0_, pxmin_, pxmax_}, {py0_, pymin_, pymax_},
  {tmin_, tmax_}, {min_, max_}, OptionsPattern[ParametricPlot]] :=
DynamicModule[{ $\gamma$ , s, v, a, n,  $\sigma$ , k, caustic},
  Manipulate[
     $\gamma$ [s_] := {x[s, px], y[s, py]};
    n[s_] =  $\frac{\{-\gamma'[s][[2]], \gamma'[s][[1]]\}}{\text{Norm}[\gamma'[s]]}$ ;
    k[s_] =  $\frac{\text{Det}[\{\gamma'[s], \gamma''[s]\}]}{\text{Norm}[\gamma'[s]]^3}$ ;
    caustic[s_] :=  $\gamma[s] + \frac{1}{k[s]} n[s]$ ;
    Show[{
      ParametricPlot[ $\gamma$ [s], {s, tmin, tmax}],
      ParametricPlot[caustic[s], {s, tmin, tmax}, PlotStyle -> Directive[Yellow]],
      ParametricPlot[ $\gamma$ [s] + n[s]  $\sigma$ ,
        {s, tmin, tmax}, PlotStyle -> OptionValue[PlotStyle]]
    }, PlotRange -> OptionValue[PlotRange], AspectRatio -> OptionValue[AspectRatio]
  ], {{ $\sigma$ , 0}, min, max},
  {{px, px0}, pxmin, pxmax}, {{py, py0}, pymin, pymax}, Paneled -> False
]
]

```

In[191]:=

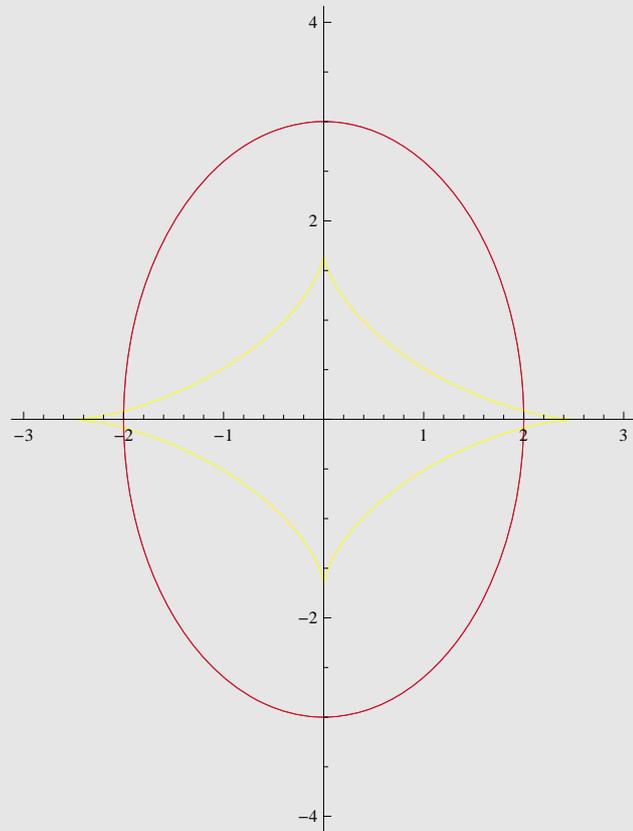
```
wave[{{#2 Cos[#1] &[t, px], #2 Sin[#1] &[t, py]}, {2, 1, 5}, {3, 1, 5}, {0, 2  $\pi$ }, {-1, 5},  
{AspectRatio  $\rightarrow$  Automatic, PlotRange  $\rightarrow$  {{-3, 3}, {-4, 4}}, PlotStyle  $\rightarrow$  Directive[Red]}}
```

σ

px

py

Out[191]=



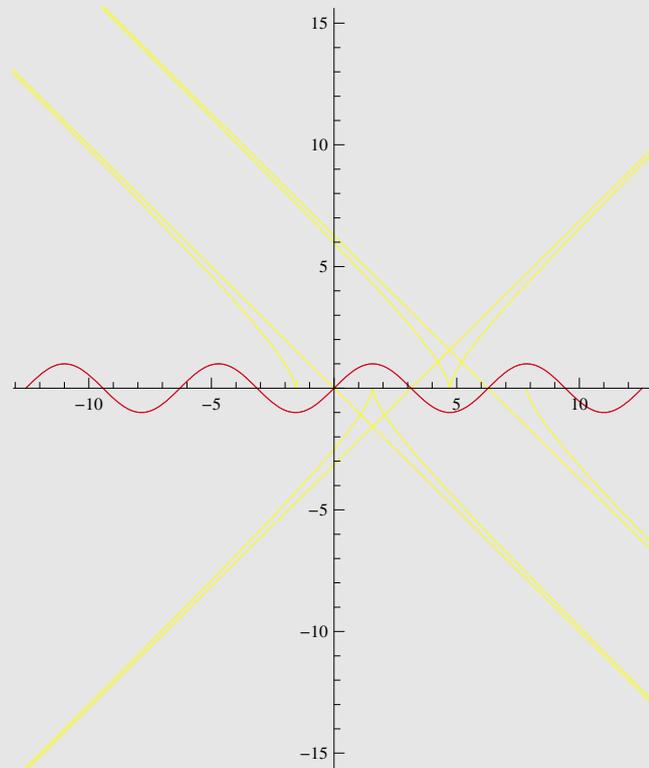
In[192]:=

```
wave[ $\{ \#1 \&[t, px], \text{Sin}[\#1] \&[t, py] \}$ ,  $\{2, 1, 5\}$ ,  $\{3, 1, 5\}$ ,  $\{-4 \pi, 4 \pi\}$ ,  
 $\{-20, 20\}$ ,  $\{\text{AspectRatio} \rightarrow \text{Automatic}, \text{PlotRange} \rightarrow \{\{-4 \pi, 4 \pi\}, \{-15, 15\}\}\}$ ,  
 $\text{PlotStyle} \rightarrow \text{Directive}[\text{Red}]$ ]
```

σ

px

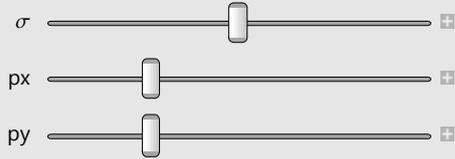
py



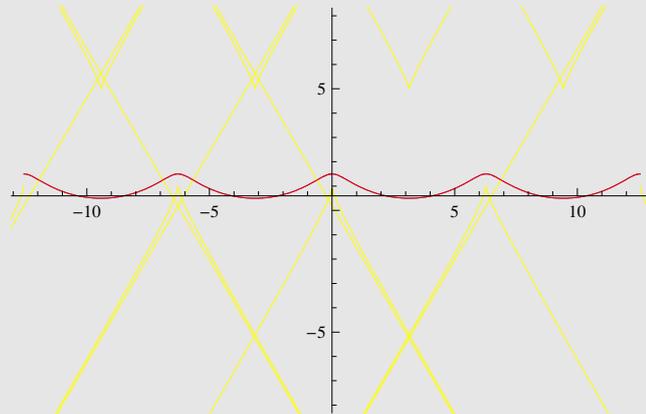
Out[192]=

In[193]=

```
wave[{{#1 + #2 Cos[#1 +  $\frac{\pi}{2}$ ] &[t, px], 1 + #2 Sin[#1 +  $\frac{\pi}{2}$ ] &[t, py]}, {0.5, 0, 2}, {0.5, 0, 2},
{-4  $\pi$ , 4  $\pi$ }, {-3  $\pi$ , 3  $\pi$ }, {AspectRatio -> Automatic, PlotRange -> {{-4  $\pi$ , 4  $\pi$ }, {-8, 8}},
PlotStyle -> Directive[Red]}
```



Out[193]=



▣ **Смоделировать часы со стрелками.**

In[194]=

```
Dynamic[DateList[], UpdateInterval -> 1]
```

Out[194]=

```
{2016, 3, 22, 0, 13, 10.0558691}
```

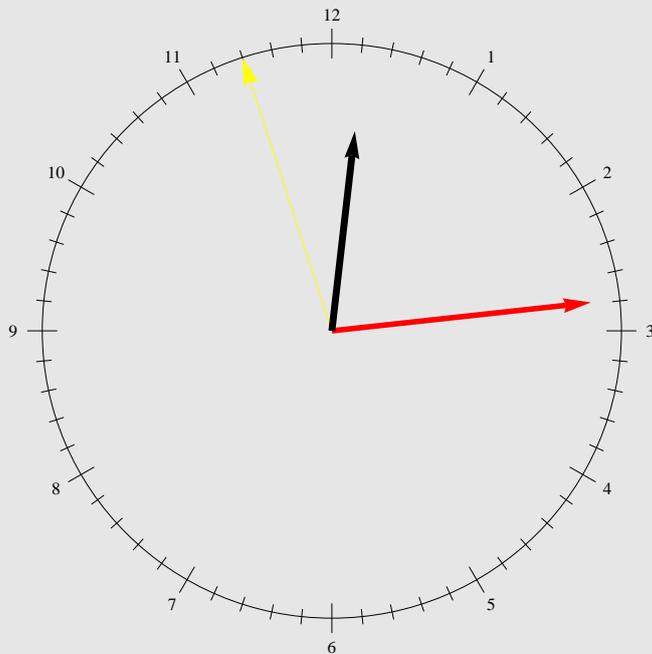
In[195]=

```

DynamicModule[{α, β, γ, δ = 0.05, δ1, φ},
  Dynamic[
    Graphics[
      {
        Circle[],
        Sequence@@Table[If[Mod[i, 5] == 0, δ1 = δ, δ1 = δ / 2];
          φ =  $\frac{2\pi}{60} i$ ; Line[{Cos[φ], Sin[φ]} # & /@ {1 - δ1, 1 + δ1}], {i, 0, 59}],
        Sequence@@Table[φ =  $\frac{2\pi}{4} - \frac{2\pi}{12} i$ ; Text[ToString[i],
          (1 + 2 δ) {Cos[φ], Sin[φ]}, Automatic], {i, 1, 12}],
        Yellow, α =  $\frac{2\pi}{4} - 2\pi \frac{\text{Round}[\text{DateList[]}][[6]]}{60}$ ; Arrow[{0, 0}, {Cos[α], Sin[α]}],
        Red, β =  $\frac{2\pi}{4} - 2\pi \frac{\text{DateList}[][[5]] + \text{DateList}[][[6]] / 60}{60}$ ;
        Thickness[0.008], Arrow[{0, 0}, (1 - 2 δ) {Cos[β], Sin[β]}],
        Black, γ =  $\frac{2\pi}{4} - 2\pi \frac{\text{Mod}[\text{DateList}[][[4]], 12] + \text{DateList}[][[5]] / 60}{12}$ ;
        Thickness[0.01], Arrow[{0, 0}, (1 - 6 δ) {Cos[γ], Sin[γ]}]
      },
      PlotRange → {{-1.2, 1.2}, {-1.2, 1.2}}, AspectRatio → Automatic],
    UpdateInterval → 1
  ]
]

```

Out[195]=



КОНЕЦ ТРЕТЬЕЙ ЛЕКЦИИ