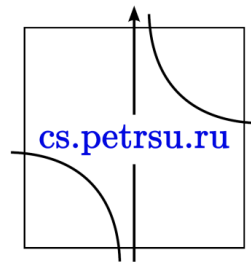


# Визуальное представление математических объектов

## Python

Глава №4

- Matplotlib
- Plotly
- Manim

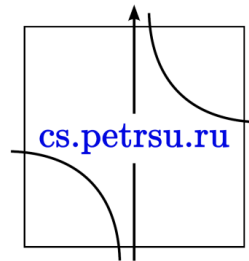


# Matplotlib

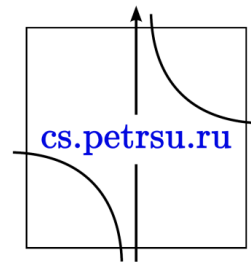
- Matplotlib — библиотека на языке программирования Python для визуализации данных двумерной графикой (3D графика также поддерживается).

Пакет поддерживает многие виды графиков и диаграмм:

- Графики (line plot)
- Диаграммы разброса (scatter plot)
- Столчатые диаграммы (bar chart) и гистограммы (histogram)
- Круговые диаграммы (pie chart)
- Ствол-лист диаграммы (stem plot)
- Контурные графики (contour plot)
- Поля градиентов (quiver)
- Спектральные диаграммы (spectrogram)



- Несложные трёхмерные графики можно строить с помощью `mplot3d`.
- Можно создавать простые анимации.
- Установка:  
`pip install matplotlib`
- Импорт  
`import matplotlib.pyplot as plt`

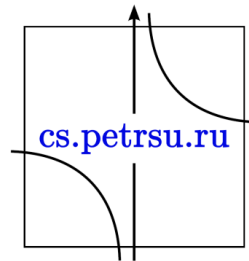


- `matplotlib` может строить графики из обычных списков или массивов:

```
import numpy as np
```

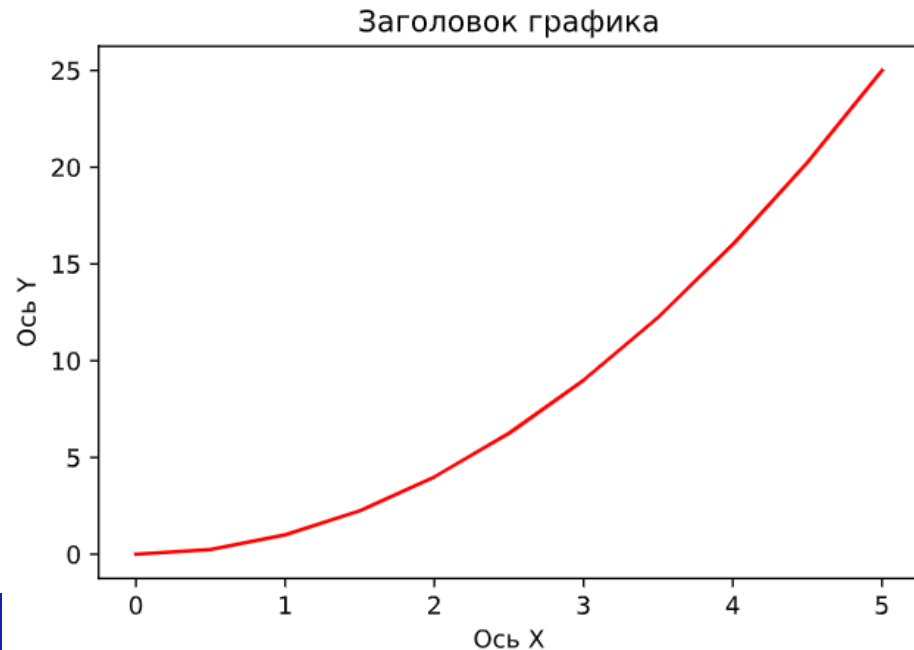
```
x = np.linspace(0, 5, 11)
```

```
y = x**2
```



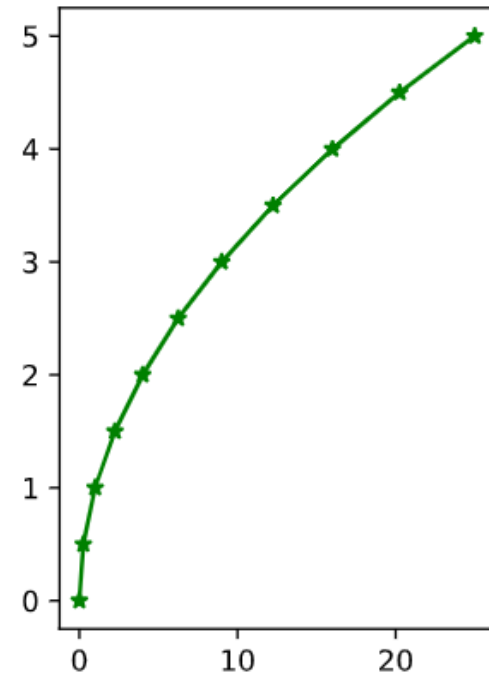
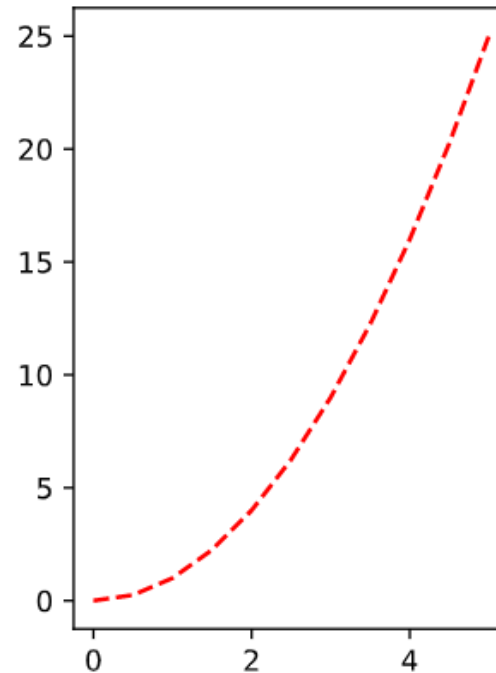
# Базовые команды

```
plt.plot(x, y, 'r') # r - значит red  
plt.xlabel('Ось X')  
plt.ylabel('Ось Y')  
plt.title('Заголовок графика')  
plt.show();
```



# Несколько графиков на одном рисунке

- `plt.subplot(1, 2, 1)`
- `plt.plot(x, y, 'r--')`
- `plt.subplot(1, 2, 2)`
- `plt.plot(y, x, 'g*-')`



# Объектно-ориентированный стиль

```
# Создадим пустое полотно
```

```
figure = plt.figure()
```

```
# Оси задаются через список из 4-х чисел:
```

```
# координаты левого нижнего угла, ширина и высота.
```

```
axes1 = figure.add_axes([0.1, 0.1, 0.8, 0.8])
```

```
axes2 = figure.add_axes([0.2, 0.5, 0.4, 0.3])
```

```
axes1.plot(x, y, 'b')
```

```
axes1.set_xlabel('Подпись оси X большого графика')
```

```
axes1.set_ylabel('Подпись оси Y большого графика')
```

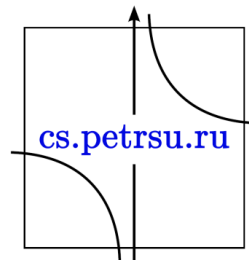
```
axes1.set_title('Заголовок большого графика')
```

```
axes2.plot(y, x, 'r')
```

```
axes2.set_xlabel('Подпись оси X\nмалого графика')
```

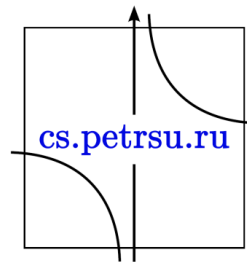
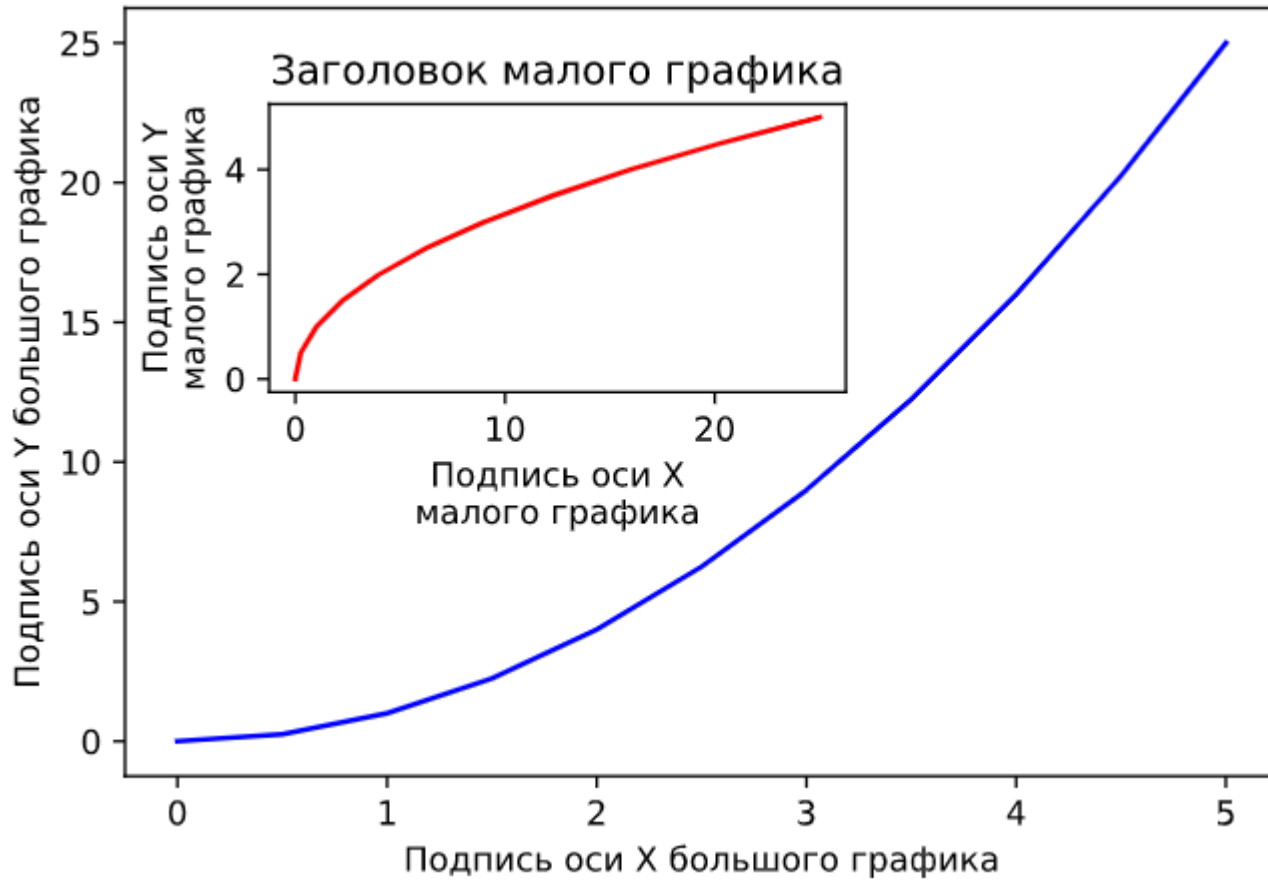
```
axes2.set_ylabel('Подпись оси Y\nмалого графика')
```

```
axes2.set_title('Заголовок малого графика');
```





### Заголовок большого графика



# Создание нескольких графиков `subplots()`

# Возвращает кортеж с фигурой и осями

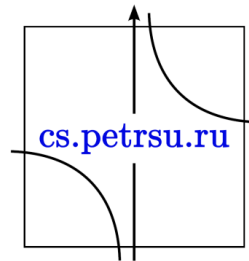
# Пример создания двух пустых графиков

```
fig, axes = plt.subplots(nrows=1, ncols=2)
```

# При построении несколько графиков оси

# часто накладываются друг на друга. Для

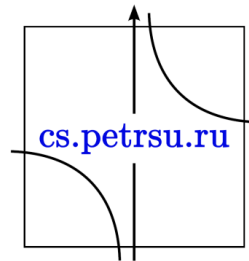
# устранения этой проблемы используйте  
`plt.tight_layout()`



# Сохранение графиков

- Можно сохранять графики в форматах PNG, JPG, EPS, SVG, PGF и PDF.
- Для сохранения графика надо использовать метод `savefig` класса `Figure`.

```
fig.savefig("filename.png", dpi=200)
```



# Легенды, подписи осей, заголовки

- Заголовки графиков (с поддержкой TeX-разметки)

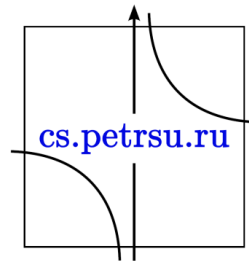
```
ax.set_title("Заголовок");
```

```
ax.set_title('Plot:  $y=x^2$ ');
```

- Подписи осей

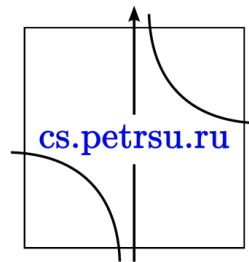
```
ax.set_xlabel("x")
```

```
ax.set_ylabel("y");
```



- Легенды - при построении графиков можно указывать аргумент `label="label text"`, указанные в нем подписи будут отображаться как подписи легенд при использовании метода `legend`:

```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.plot(x, x**2, label="x**2")
ax.plot(x, x**3, label="x**3")
ax.legend();
```



- Управление расположением легенды:

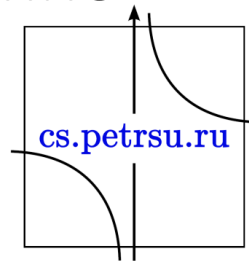
`ax.legend(loc=1) # верх право`

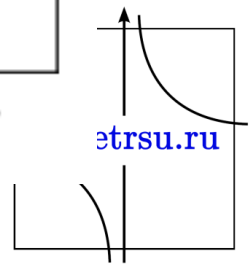
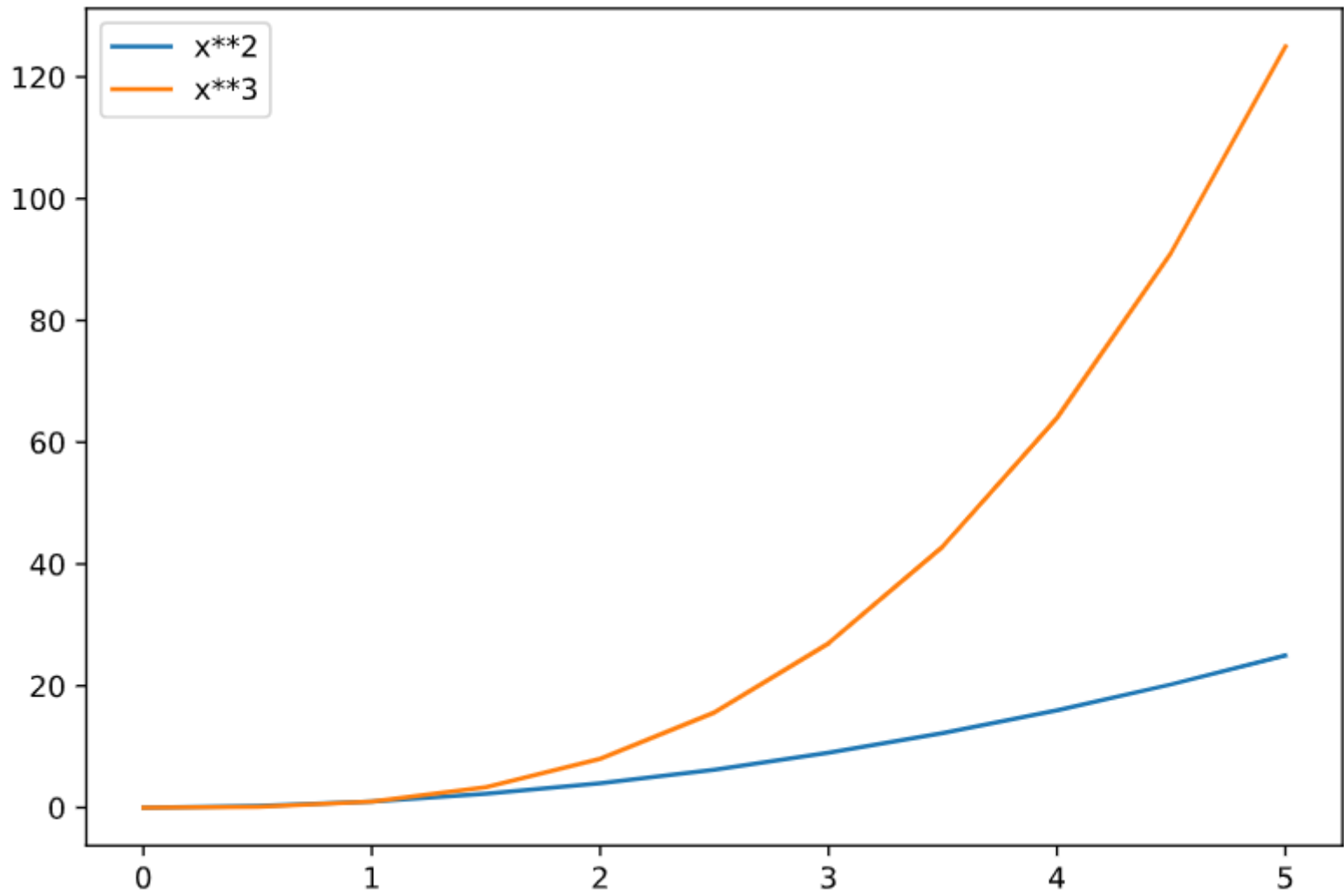
`ax.legend(loc=2) # верх лево`

`ax.legend(loc=3) # низ лево`

`ax.legend(loc=4) # низ право`

`ax.legend(loc=0) # оптимальное местоположение`





# Установка цвета, толщины и типа линий

- Стиль MatLab

```
ax.plot(x, x**2, 'b.-') # Синяя линия, точка пунктир
```

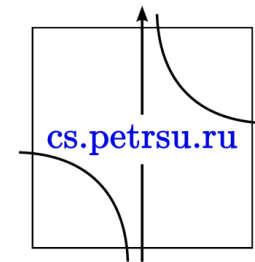
```
ax.plot(x, x**3, 'g--'); # Зелёна пунктирная линия
```

- Указание цветов в параметре color= и alpha

```
ax.plot(x, x+1, color="blue", alpha=0.5) # Полупрозрачная линия
```

```
ax.plot(x, x+2, color="#8B008B") # RGB
```

```
ax.plot(x, x+3, color="#FF8C00") # RGB
```

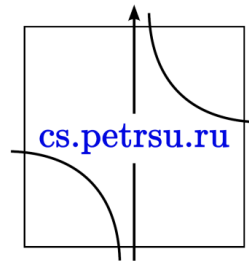




- **Стили линии и маркеров**

- толщина линии - linewidth или lw

- стиль линии - linestyle или ls



```
ax.plot(x, x+1, color="red", linewidth=0.25)
ax.plot(x, x+2, color="red", linewidth=0.50)
ax.plot(x, x+3, color="red", linewidth=1.00)
ax.plot(x, x+4, color="red", linewidth=2.00)
```

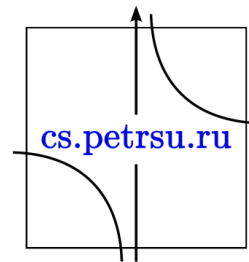
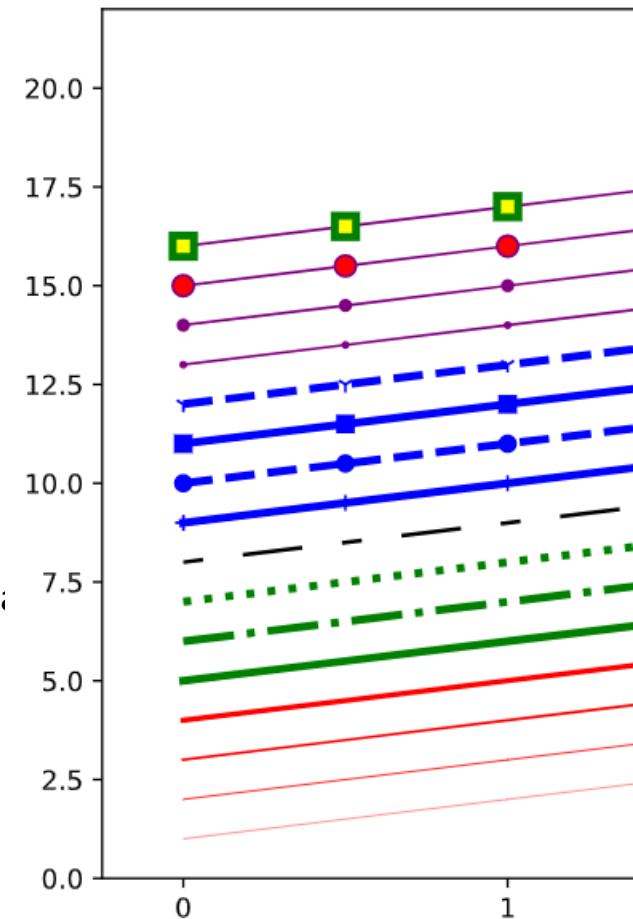
```
# Возможные варианты '-', '--', '-.', ':', 'steps'
ax.plot(x, x+5, color="green", lw=3, linestyle='-')
ax.plot(x, x+6, color="green", lw=3, ls='-.')
ax.plot(x, x+7, color="green", lw=3, ls=':')
```

```
# Кастомная пунктирная линия
line, = ax.plot(x, x+8, color="black", lw=1.50)
line.set_dashes([5, 10, 15, 10]) # формат: длина линия, длина пропуск:
```

```
# Возможные маркеры: marker = '+', 'o', '*', 's', ',', '!', '1', '2', '3', '4', ...
ax.plot(x, x+ 9, color="blue", lw=3, ls='-', marker='+')
ax.plot(x, x+10, color="blue", lw=3, ls='--', marker='o')
ax.plot(x, x+11, color="blue", lw=3, ls='-', marker='s')
ax.plot(x, x+12, color="blue", lw=3, ls='--', marker='1')
```

```
# Размер и цвет маркеров
```

```
ax.plot(x, x+13, color="purple", lw=1, ls='-', marker='o', markersize=2)
ax.plot(x, x+14, color="purple", lw=1, ls='-', marker='o', markersize=4)
ax.plot(x, x+15, color="purple", lw=1, ls='-', marker='o', markersize=8, markerfacecolor="red")
ax.plot(x, x+16, color="purple", lw=1, ls='-', marker='s', markersize=8,
        markerfacecolor="yellow", markeredgewidth=3, markeredgewidth="green");
```

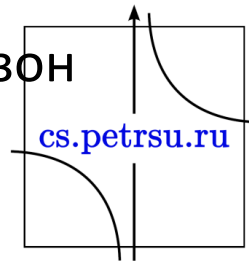


Диапазон значений графика - методы `set_ylim`, `set_xlim` и `axis(['off' | 'equal' | 'scaled' | 'tight' | 'image' | 'auto' | 'normal' | 'square'])`:

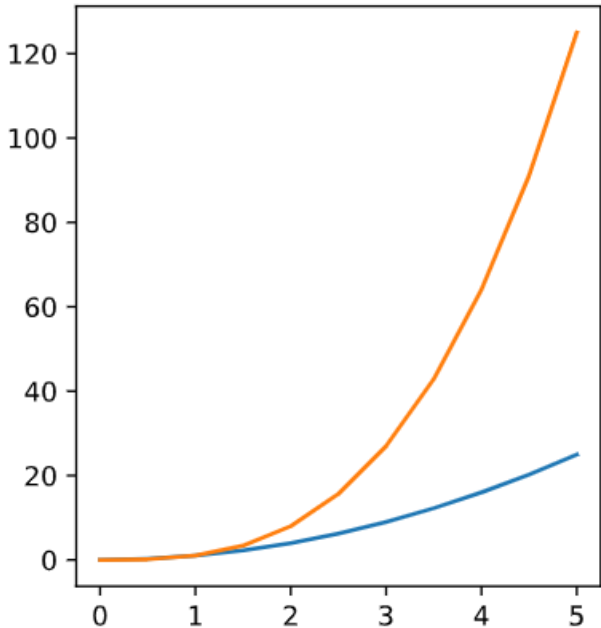
```
fig, axes = plt.subplots(1, 3, figsize=(12, 4))  
axes[0].plot(x, x**2, x, x**3)  
axes[0].set_title("Стандартный диапазон осей")
```

```
axes[1].plot(x, x**2, x, x**3)  
axes[1].axis('tight')  
axes[1].set_title("«Тесные» оси")
```

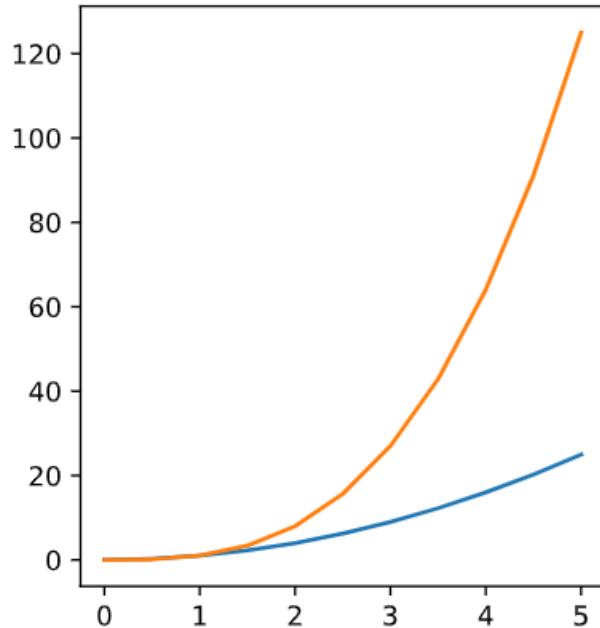
```
axes[2].plot(x, x**2, x, x**3)  
axes[2].set_ylim([0, 60])  
axes[2].set_xlim([2, 5])  
axes[2].set_title("Заданный точными значениями \ndиапазон осей");
```



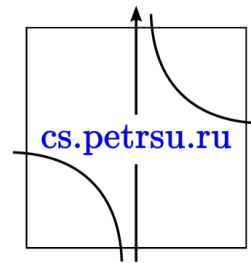
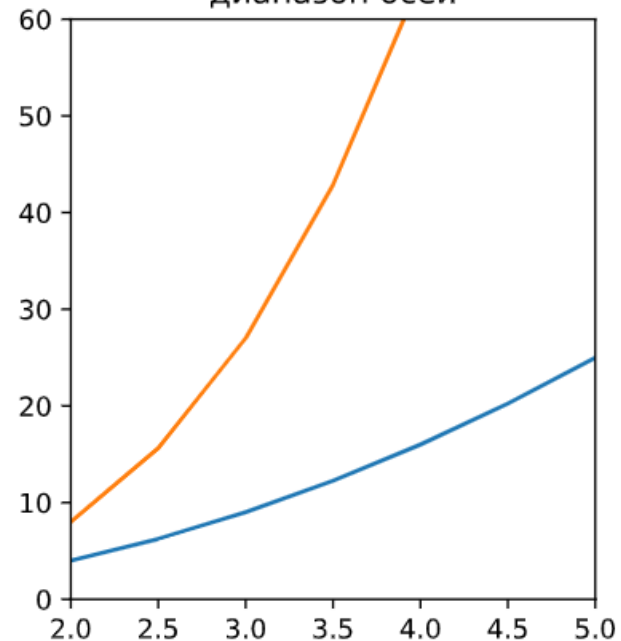
Стандартный диапазон осей



«Тесные» оси

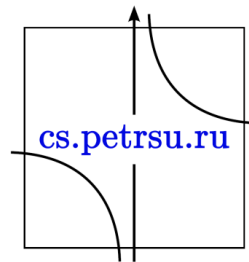
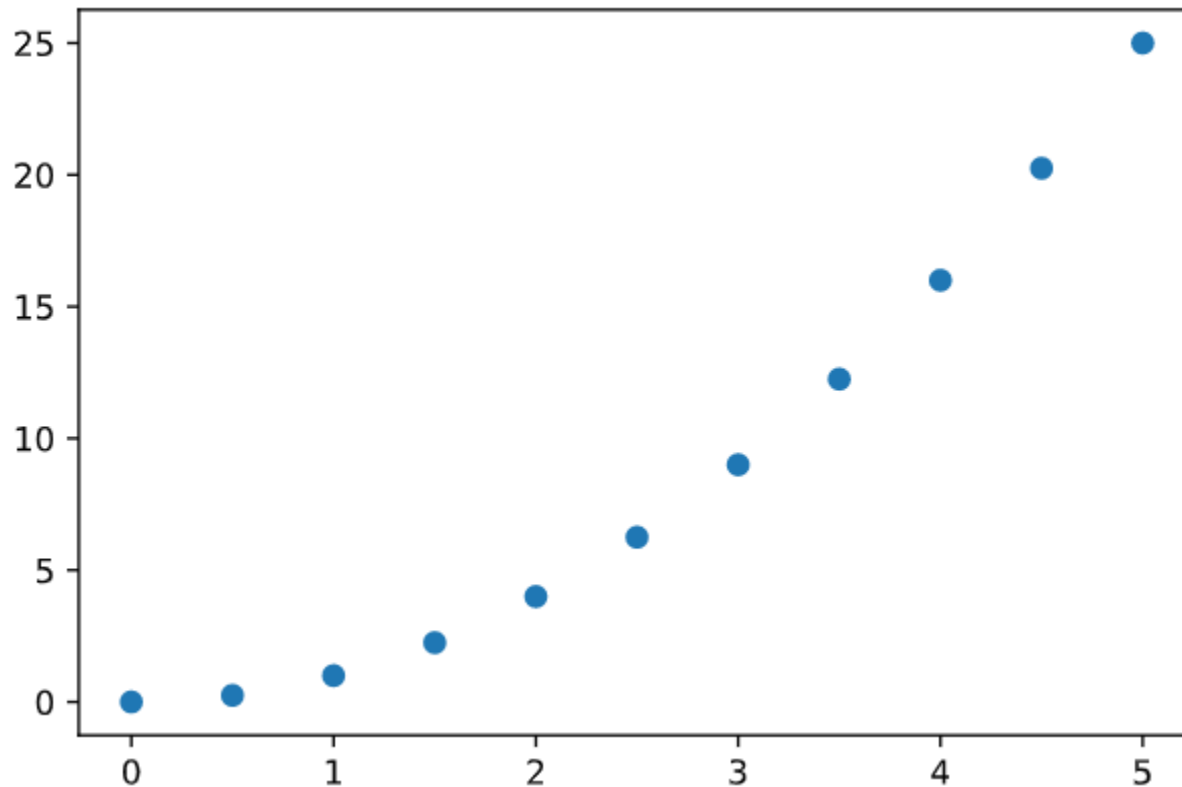


Заданный точными значениями диапазон осей



# Другие типы графиков

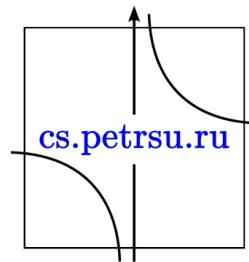
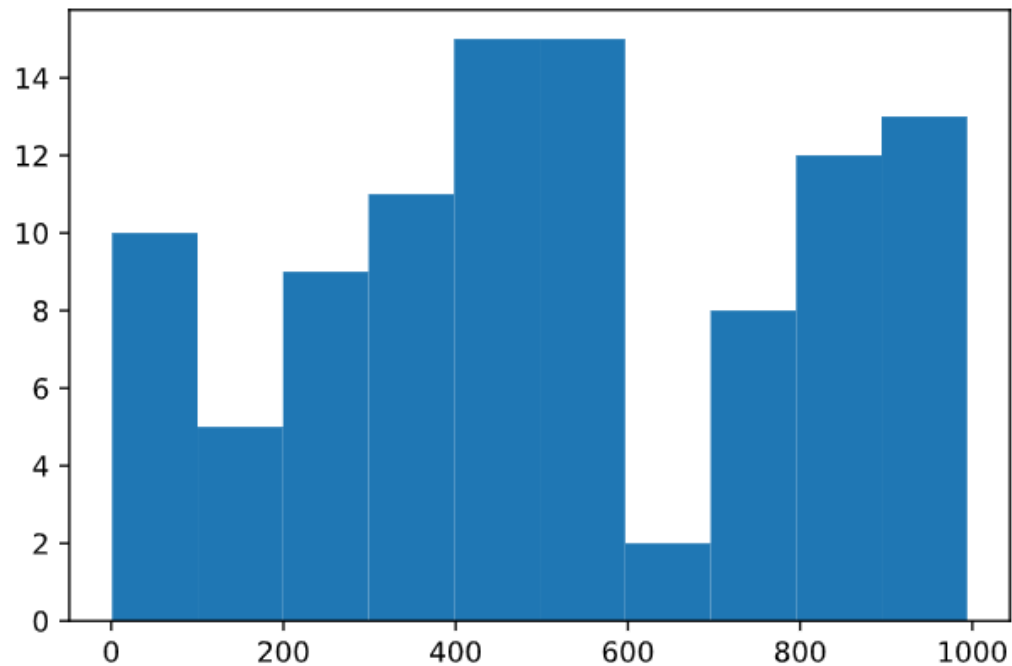
`plt.scatter(x,y)`



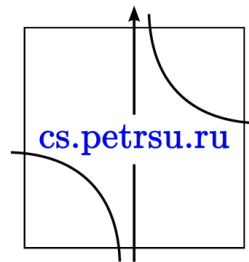
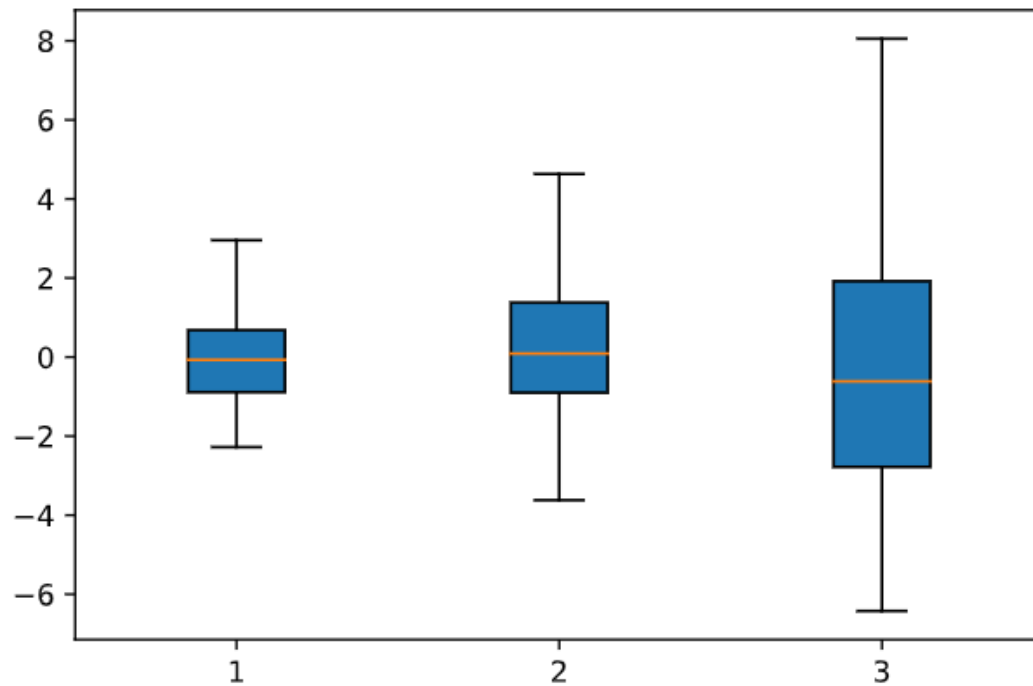
from random import

```
sample data = sample(range(1, 1000), 100)
```

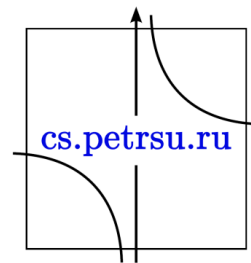
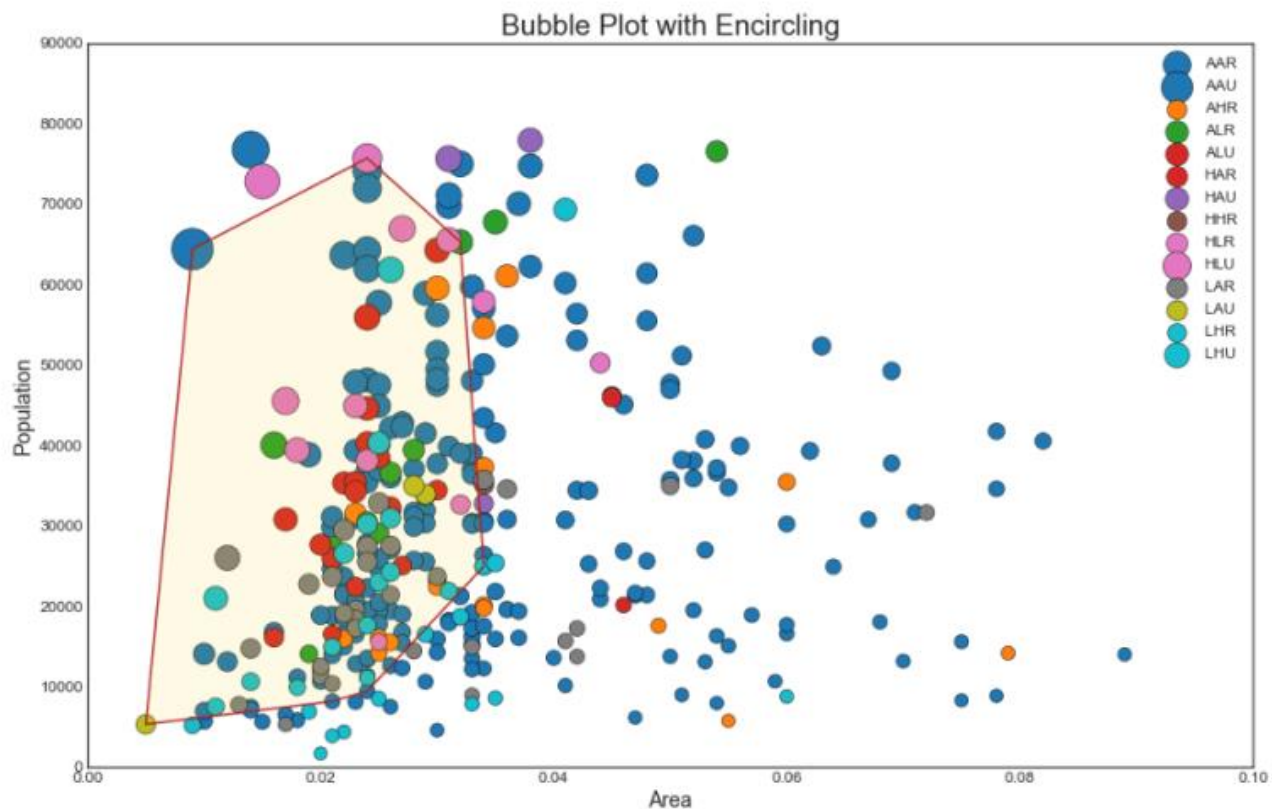
```
plt.hist(data);
```



```
data = [np.random.normal(0, std, 100) for std in range(1, 4)]  
# rectangular box plot  
plt.boxplot(data,vert=True,patch_artist=True);
```



- Пузырьковая диаграмма – метод encircle()





```
import matplotlib.pyplot as plt
```

```
types = 'Group 1', 'Group 2', 'Group 3'
```

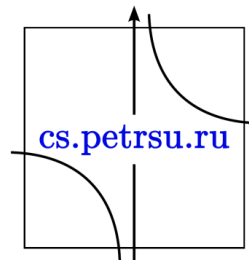
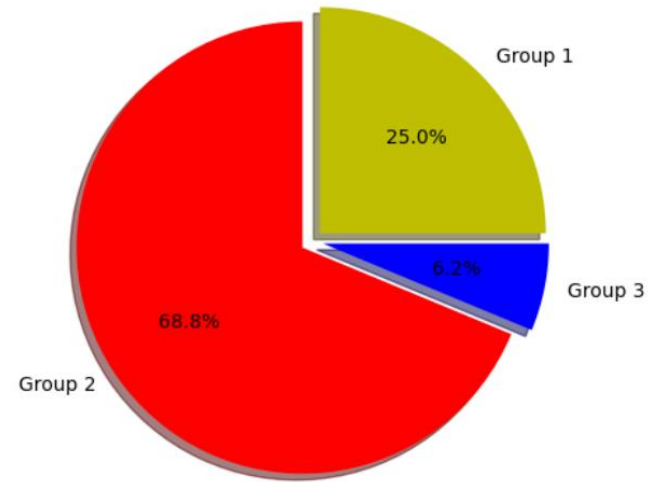
```
types_data = [8,22,2]
```

```
colors = ['y','r','b']
```

```
plt.pie(types_data, labels = types, colors=colors ,shadow =  
True, explode = (0.05, 0.05, 0.05), autopct = '%1.1f%%')
```

```
plt.axis('equal')
```

```
plt.show()
```



- Круговая диаграмма

```
# Import
```

```
df_raw = pd.read_csv("https://github.com/selva86/datasets/raw/master/  
mpg_ggplot2.csv")
```

```
# Prepare Data
```

```
df = df_raw.groupby('class').size()
```

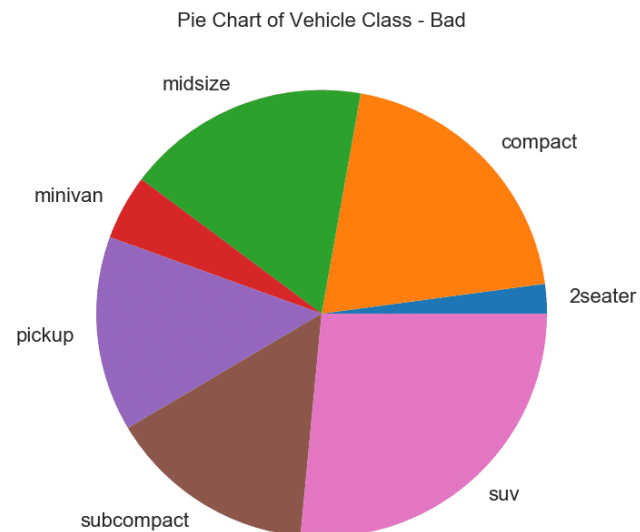
```
# Make the plot with pandas
```

```
df.plot(kind='pie', subplots=True, figsize=(8, 8), dpi= 80)
```

```
plt.title("Pie Chart of Vehicle Class - Bad")
```

```
plt.ylabel("")
```

```
plt.show()
```



- Полярные координаты:

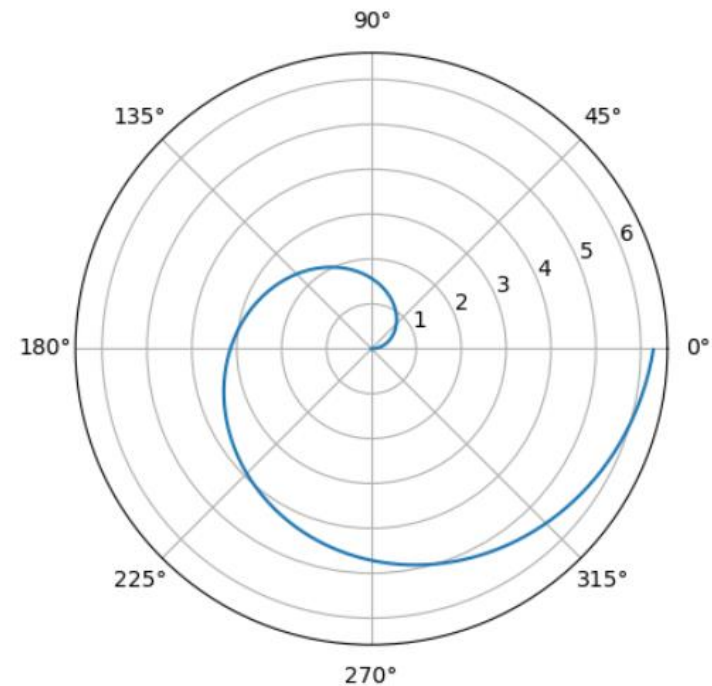
```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
t = np.arange(0,2*np.pi, 0.02)
```

```
plt.polar(t, t)
```

```
plt.show()
```



- Поверхности:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

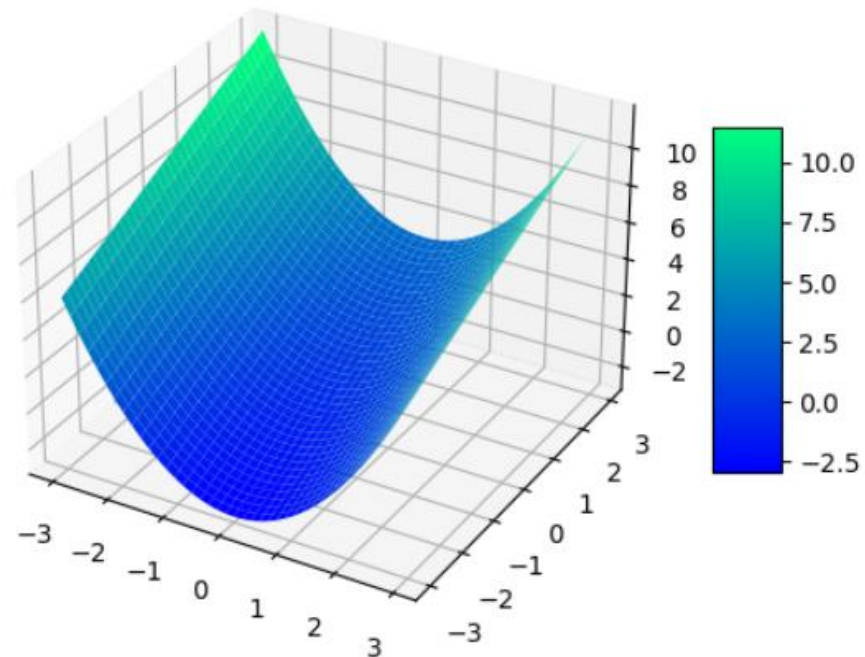
```
def fun(x, y):
    return x**2 + y
```

```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
x = y = np.arange(-3.0, 3.0, 0.05)
X, Y = np.meshgrid(x, y)
zs = np.array(fun(np.ravel(X), np.ravel(Y)))
Z = zs.reshape(X.shape)

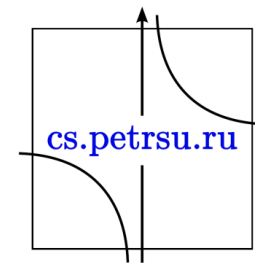
surf = ax.plot_surface(X, Y, Z, cmap='winter')

fig.colorbar(surf, shrink=0.5, aspect=5)

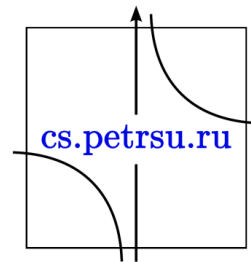
plt.show()
```



- Подробнее в документации:  
<https://matplotlib.org/2.0.2/users/index.html>
- Примеры от разработчиков:  
<https://matplotlib.org/2.0.2/examples/index.html>
- Описание диаграмм на хабре:  
<https://habr.com/ru/post/468295/>

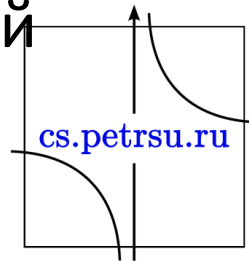


- Настроить на своей машине.
- Использовать сервер карра / рабочие станции в комп.классах (guix).
- Online сервисы:
  - <https://colab.research.google.com/>
  - [https://www.tutorialspoint.com/execute matplotlib online.php](https://www.tutorialspoint.com/execute_matplotlib_online.php)
  - <https://www.codabrainy.com/en/python-compiler/>
  - Другие ....



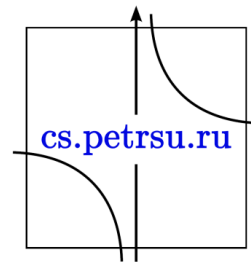
# Plotly

- бесплатная библиотека
  - но можно использовать в коммерческих целях
- работает offline
- позволяет строить интерактивные визуализации
  - можно изучать данные «на лету» ( не надо перестраивать график в matplotlib, изменять масштаб, включать/выключать какие-то данные)
  - можно построить полноценный интерактивный отчёт



- Установка:  
pip install plotly  
+ зависимости Pandas и Numpy
- Вначале работы необходимо импортировать:  
import plotly  
import plotly.graph\_objs as go  
import plotly.express as px  
from plotly.subplots import make\_subplots

```
import numpy as np  
import pandas as pd
```





# Линейный график

- Рассмотрим построение простого графика по точкам:

$$f(x)=x^2$$

- Вариант 1:
  - Создать график с помощью функции `scatter` из подмодуля `plotly.express` (передавая 2 списка точек: координаты X и Y)
  - Показать его с помощью метода `show()`

п.с.: график интерактивный, если навести на него курсор, то можно его приближать и удалять, выделять участки, по наведению курсора на точку получать подробную информацию, возвращать картинку в исходное положение, сохранять как файл ([cs.petrSU.ru](http://cs.petrSU.ru) все это реализуется на js в браузере).



- Код для создания графика:

```
x = np.arange(0, 5, 0.1)
```

```
def f(x):
```

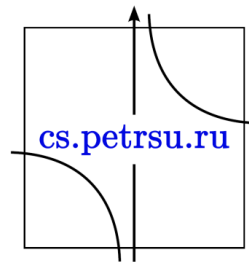
```
    return x**2
```

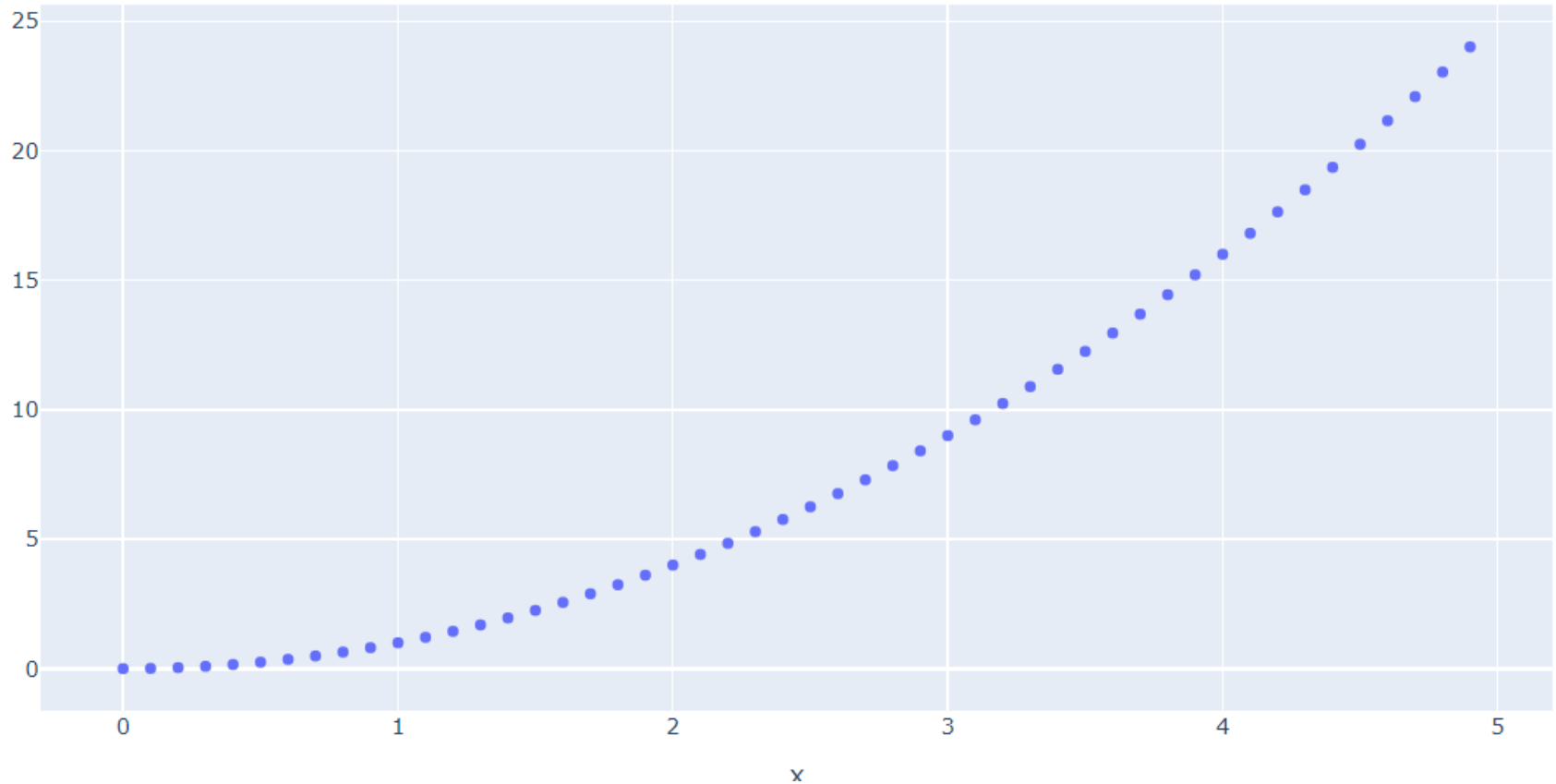
```
px.scatter(x=x, y=f(x)).show()
```

- Но лучше так:

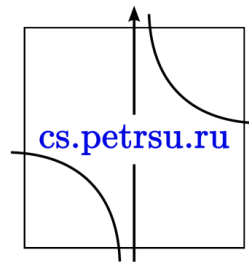
```
figure = px.scatter(x=x, y=f(x))
```

```
figure.show()
```





- п.с.: В отличие от Matplotlib отдельные объекты осей не создаются.

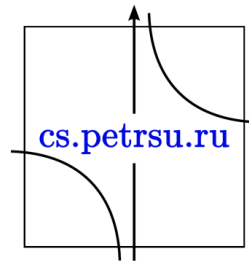


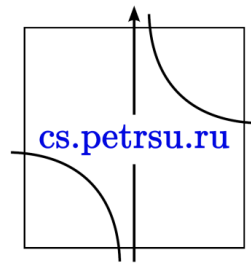
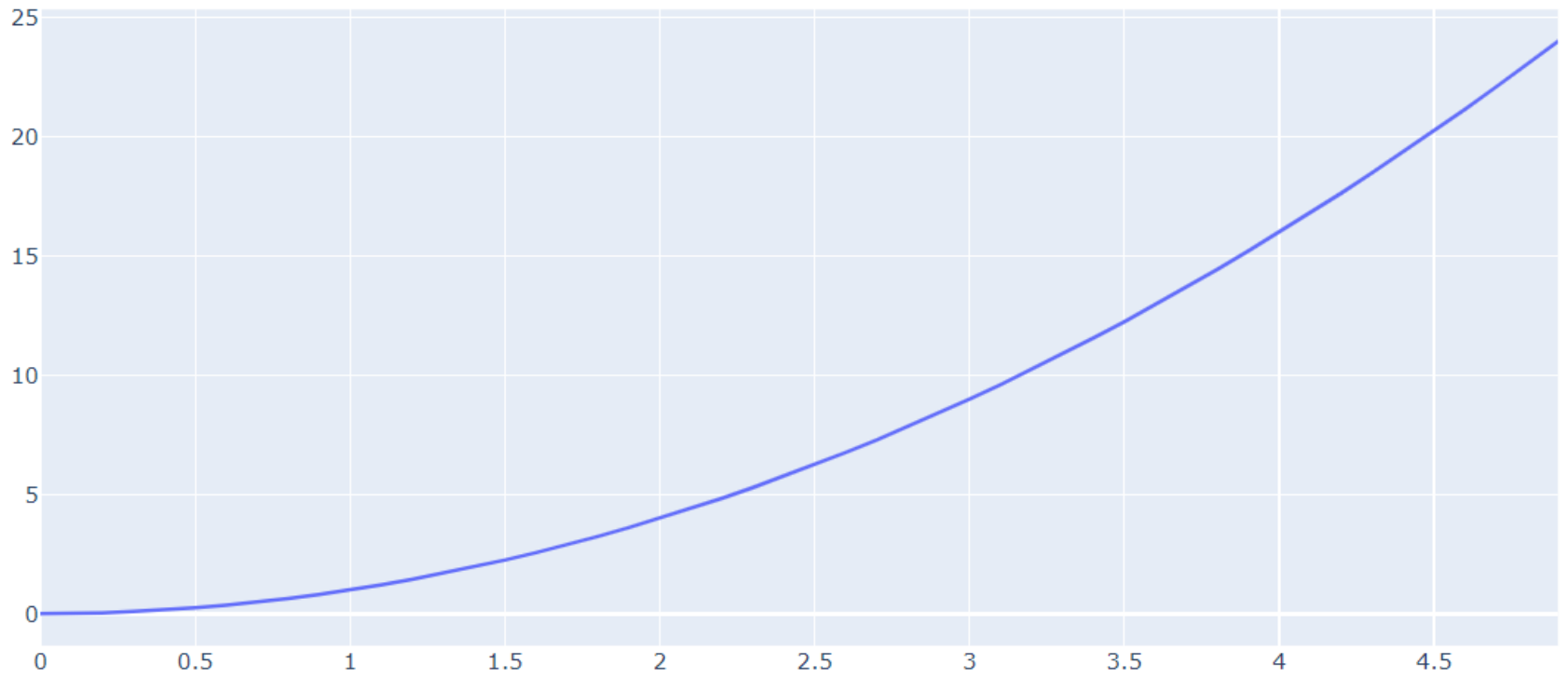
- У варианта `express` мало гибкости, поэтому рассмотрим второй вариант:
  - создадим фигуру и нанесём на неё объекты
  - выведем фигуру для показа с помощью метода `show()`

```
figure = go.Figure()
```

```
figure.add_trace(go.Scatter(x=x, y=f(x)))
```

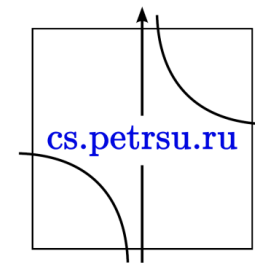
```
figure.show()
```





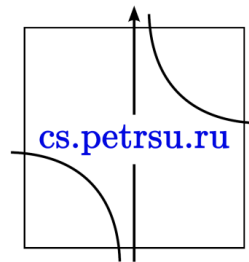
- Основные отличия в результате
  - получилась гладкая кривая
  - можно строить несколько кривых на один график

```
figure = go.Figure()  
figure.add_trace(go.Scatter(x=x, y=f(x)))  
figure.add_trace(go.Scatter(x=x, y=x))  
figure.show()
```



# Настройка отображения

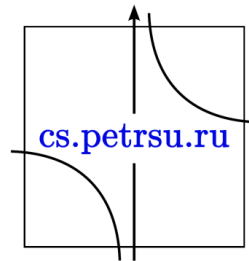
- Plotly поддерживает LATEX в подписях (как и matplotlib через использование \$\$ с обеих сторон).
    - минус в том что подсказки в сыром виде
    - можно использовать теги HTML
- $f(x)=x^{2}$



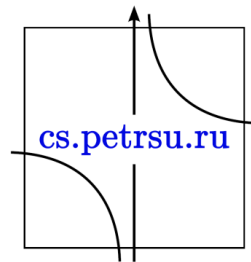
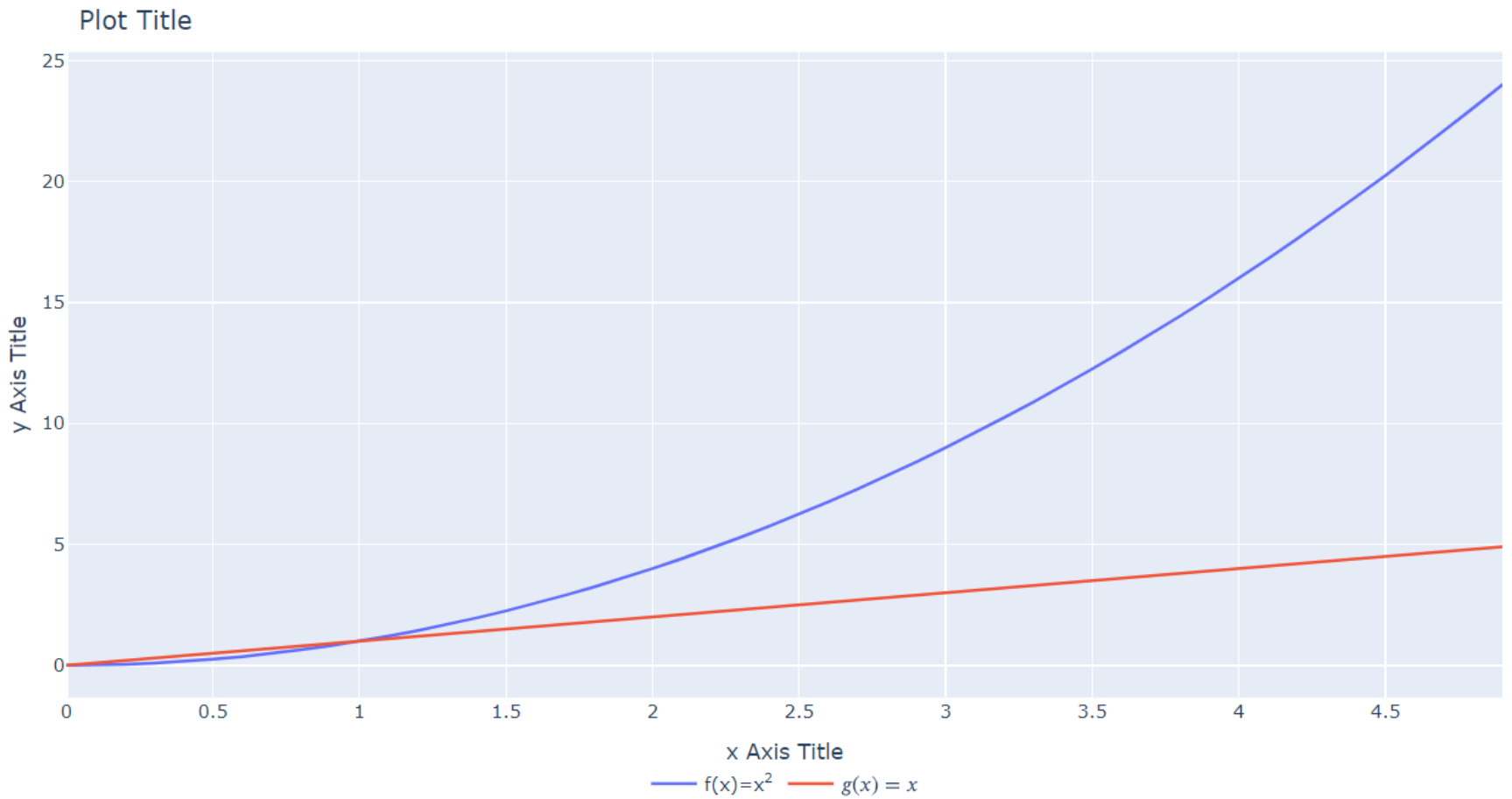
# update\_layout

- Изменение расположение легенды, отступов, подписей к осям и графику через `update_layout`:
  - параметры `legend`, `legend_orientation`
  - параметр `margin`
  - параметры `title`, `xaxis_title`, `yaxis_title`

```
figure = go.Figure()
figure.add_trace(go.Scatter(x=x, y=f(x), name='f(x)=x2'))
figure.add_trace(go.Scatter(x=x, y=x, name='$g(x)=x$'))
figure.update_layout(legend_orientation="h",
                    legend=dict(x=.5, xanchor="center"),
                    title="Plot Title",
                    xaxis_title="x Axis Title",
                    yaxis_title="y Axis Title",
                    margin=dict(l=0, r=0, t=30, b=0))
figure.show()
```







# add\_trace

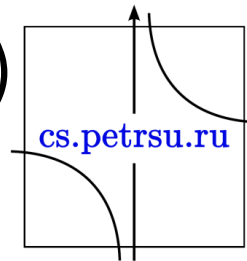
- Настройка варианта отображения через:

- ЛИНИЯ И ТОЧКИ:

```
figure.add_trace(go.Scatter(x=x, y=f(x),  
                             mode='lines+markers',  
                             name='f(x)=x2'))
```

- ТОЛЬКО ТОЧКИ:

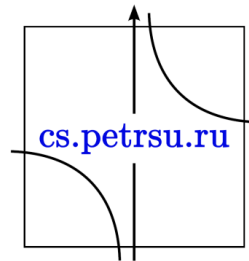
```
figure.add_trace(go.Scatter(x=x, y=x,  
                             mode='markers', name='$g(x)=x$'))
```

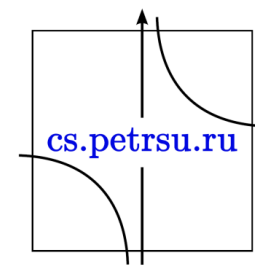
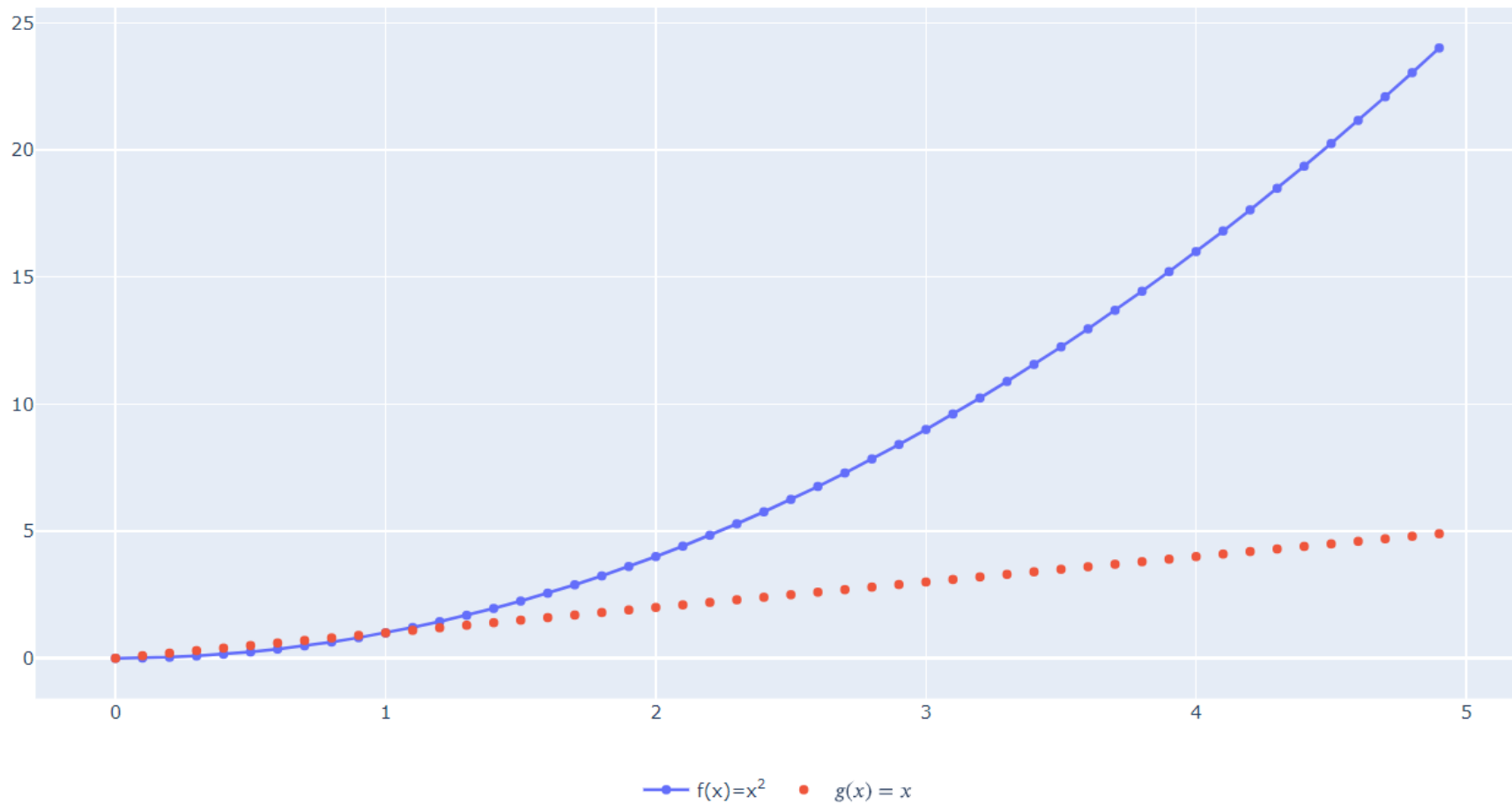


# add\_trace

- Можно настроить маркеры для конкретной кривой:

```
figure.add_trace(go.Scatter(x=x, y=x,  
                             mode='markers', name='g(x)=x',  
                             marker=dict(color='LightSkyBlue', size=20,  
                             line=dict(color='MediumPurple', width=3))))
```





# update\_traces

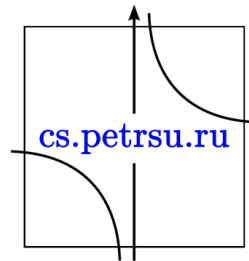
Изменение параметров отображения через `update_traces`:

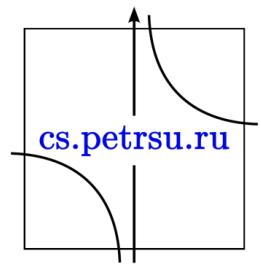
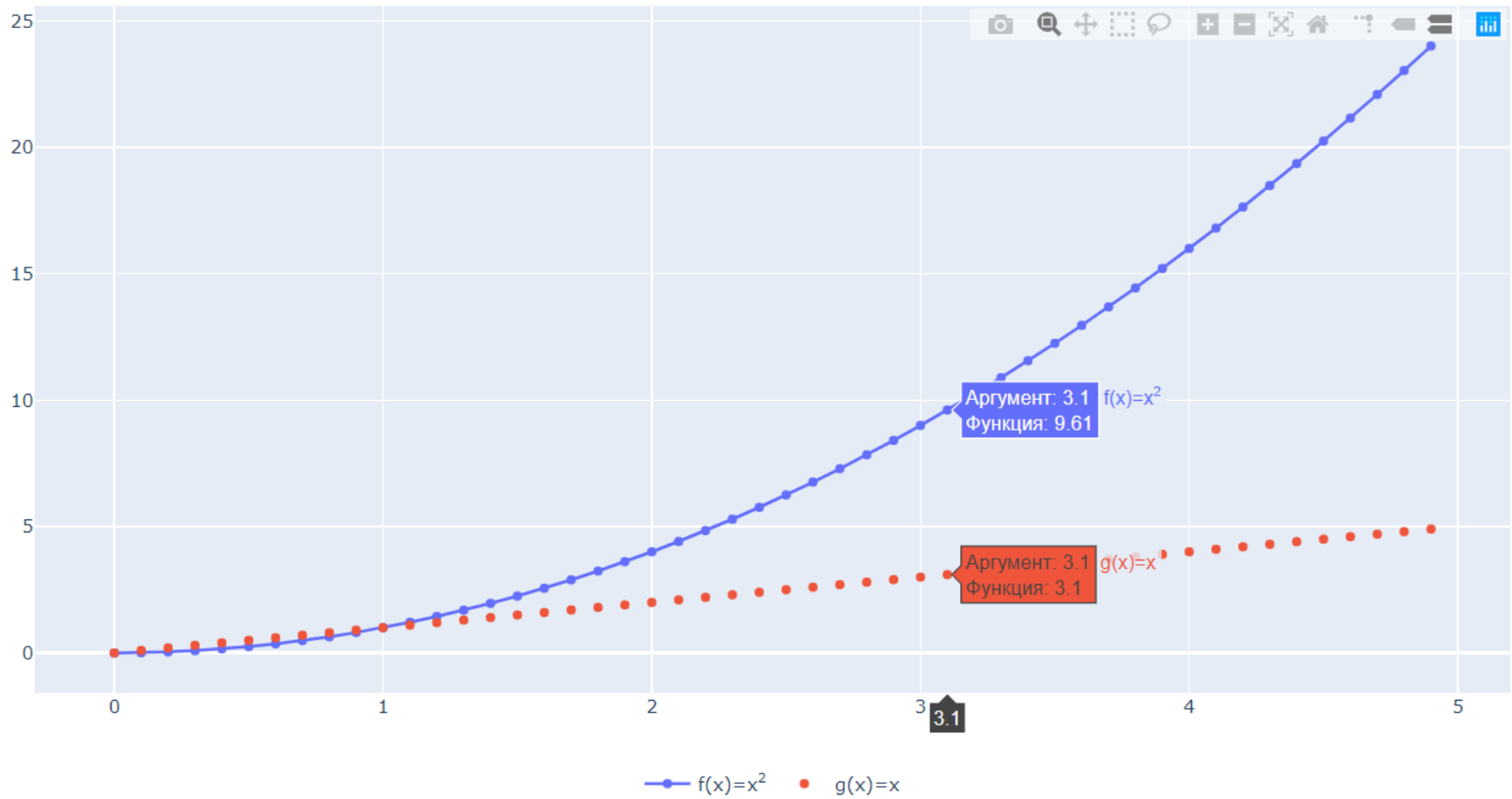
- Вывод информации и форматированный шаблон вывода (параметры `hoverinfo+hovertemplate`):

```
figure.update_traces(hoverinfo="all",  
                    hovertemplate="Аргумент: %{x}<br>Функция: %{y}")
```

- 
- Для синхронного вывода нескольких кривых в одной точке можно настроить режим показа подсказок (параметр `hovermode`):

```
figure.update_layout(legend_orientation="h",  
                    legend=dict(x=.5, xanchor="center"),  
                    hovermode="x",  
                    margin=dict(l=0, r=0, t=0, b=0))
```



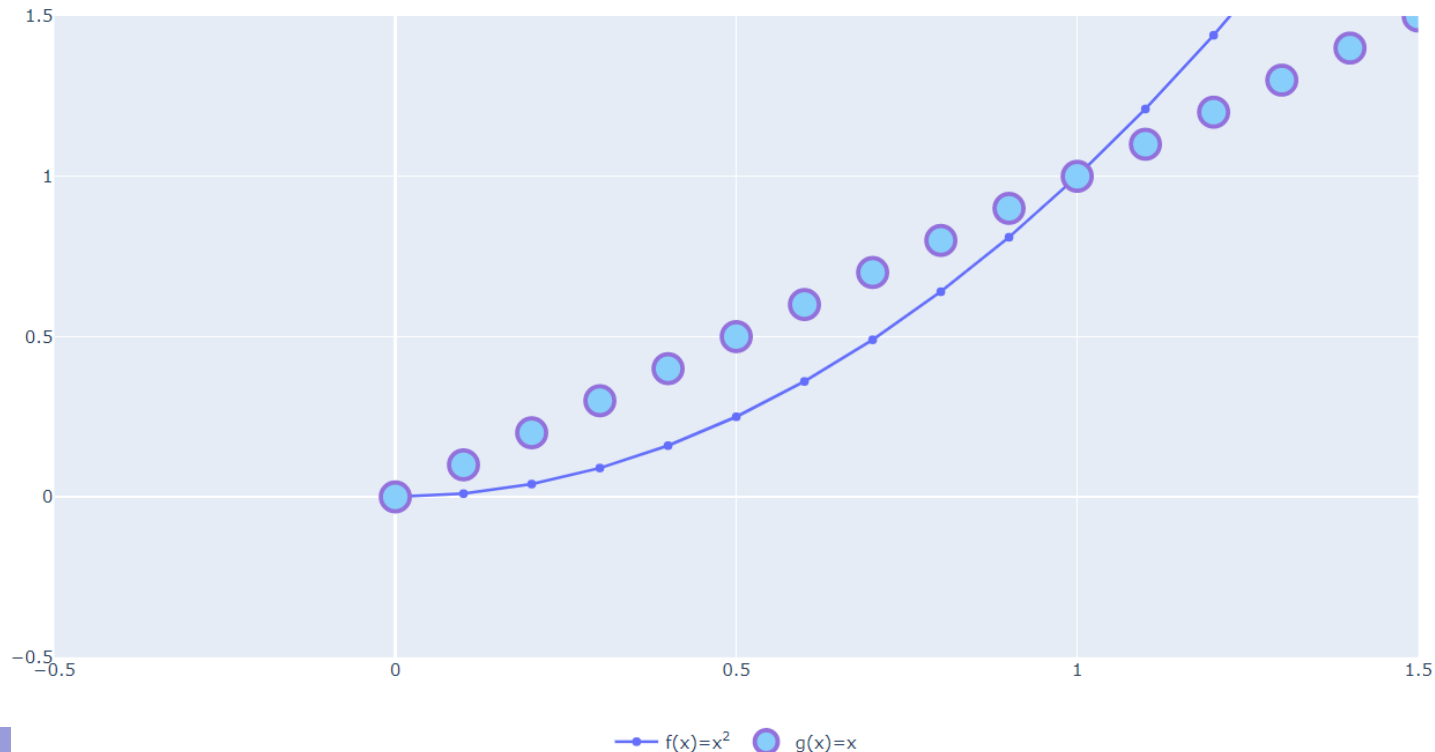


# update\_yaxes, update\_xaxes

- Изменение интервалов отображения для осей:

```
figure.update_yaxes(range=[-0.5, 1.5])
```

```
figure.update_xaxes(range=[-0.5, 1.5])
```

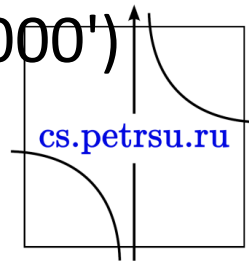


# update\_yaxes, update\_xaxes

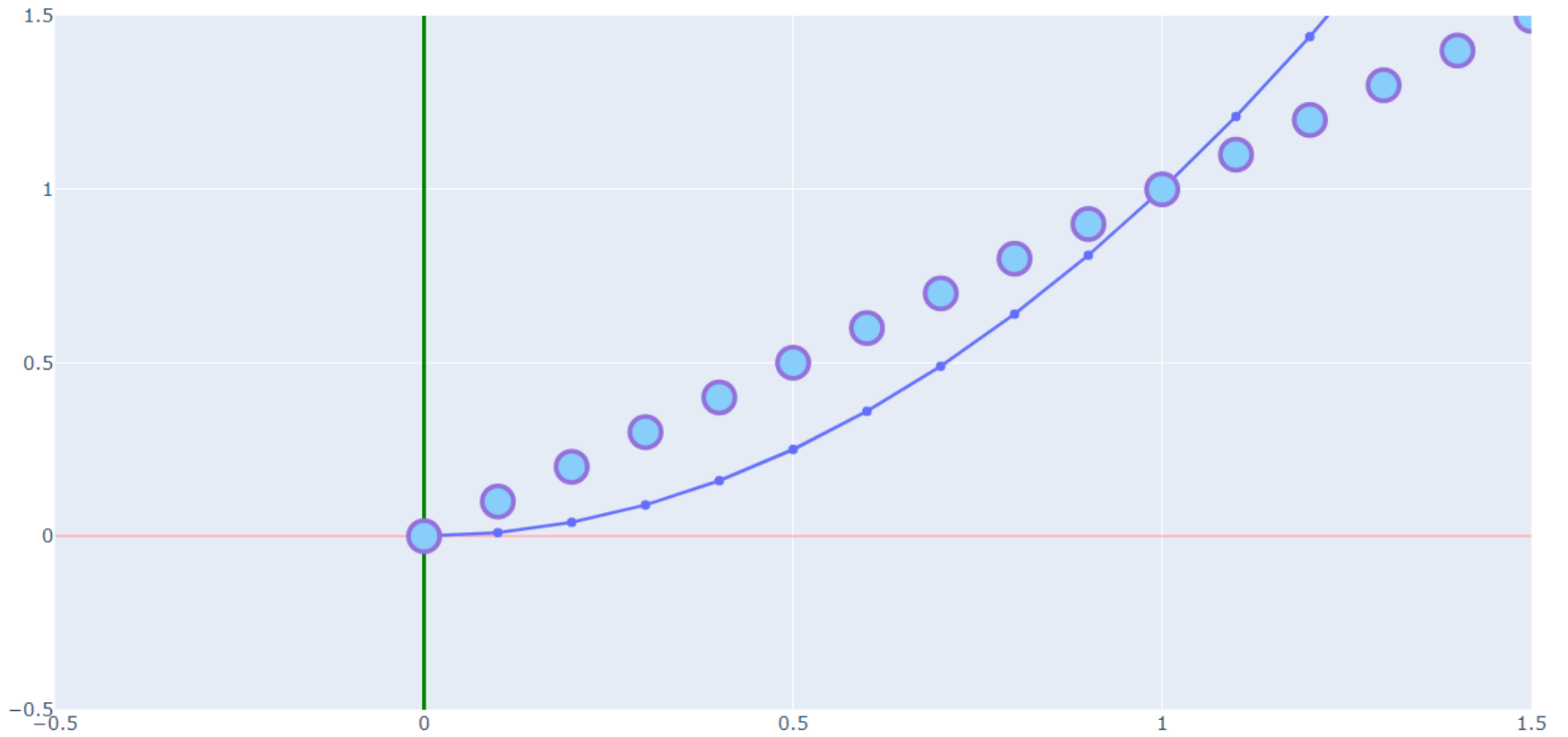
- Нанесем осевые линии:
  - zeroline — выводить или нет осевую линию
  - zerolinewidth — задаёт толщину осевой (в пикселях) линии
  - zerolinecolor — задаёт цвет осевой линии

```
figure.update_yaxes(range=[-0.5, 1.5], zeroline=True,  
                    zerolinewidth=2, zerolinecolor='LightPink')
```

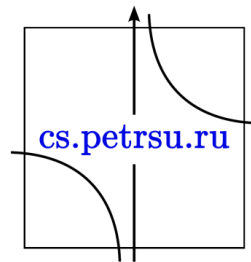
```
figure.update_xaxes(range=[-0.5, 1.5], zeroline=True,  
                    zerolinewidth=2, zerolinecolor='#008000')
```







—•—  $f(x)=x^2$     ●  $g(x)=x$



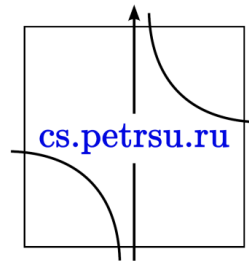
# Скрытие графиков: visible

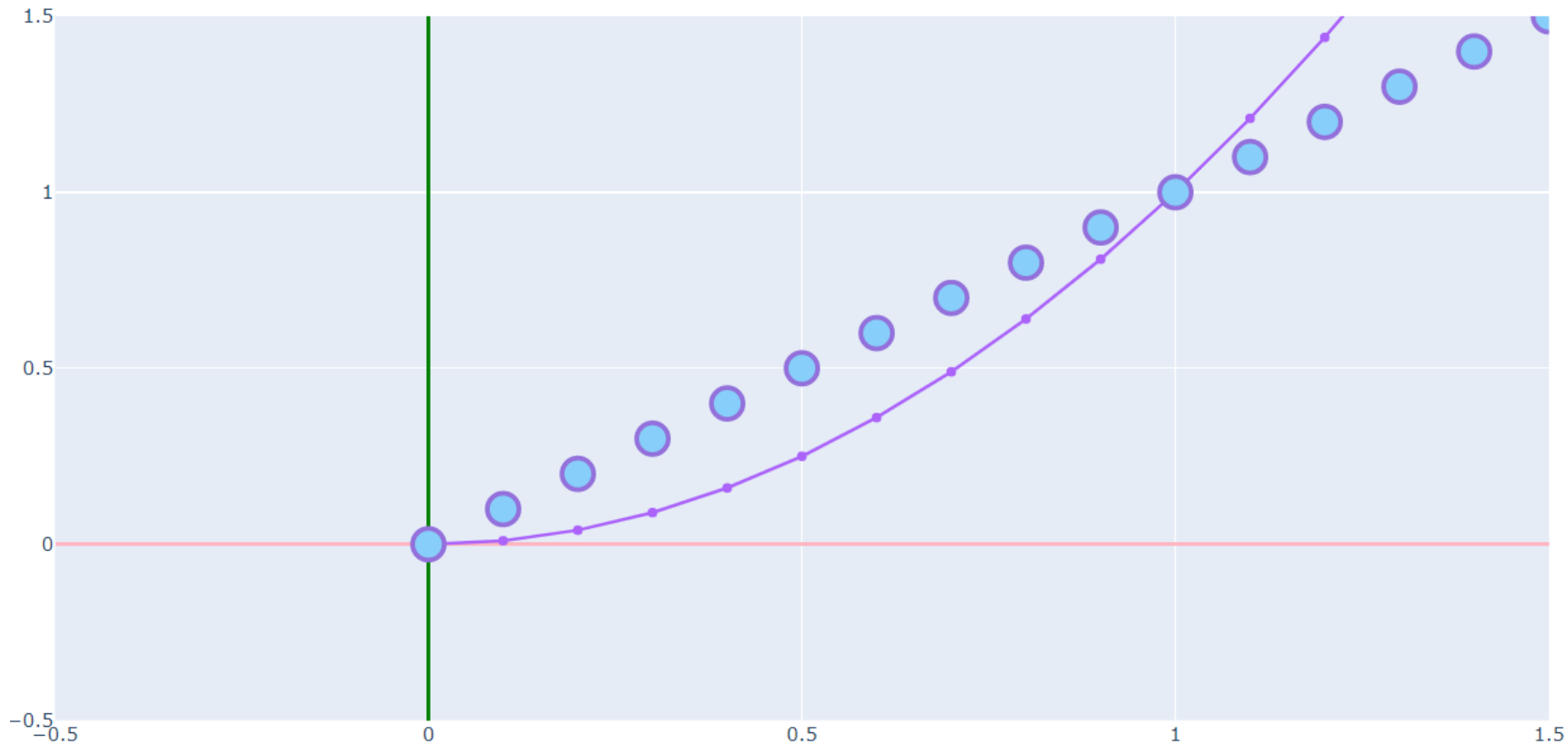
```
def h(x):  
    return np.sin(x)
```

```
def k(x):  
    return np.cos(x)
```

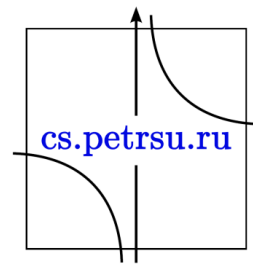
```
def m(x):  
    return np.tan(x)
```

```
figure = go.Figure()  
figure.update_yaxes(range=[-0.5, 1.5], zeroline=True, zerolinewidth=2, zerolinecolor='LightPink')  
figure.update_xaxes(range=[-0.5, 1.5], zeroline=True, zerolinewidth=2, zerolinecolor='#008000')  
figure.add_trace(go.Scatter(visible='legendonly', x=x, y=h(x), name='h(x)=sin(x)'))  
figure.add_trace(go.Scatter(visible='legendonly', x=x, y=k(x), name='k(x)=cos(x)'))  
figure.add_trace(go.Scatter(visible='legendonly', x=x, y=m(x), name='m(x)=tg(x)'))  
figure.add_trace(go.Scatter(x=x, y=f(x), mode='lines+markers', name='f(x)=x2'))  
figure.add_trace(go.Scatter(x=x, y=x, mode='markers', name='g(x)=x',  
    marker=dict(color='LightSkyBlue', size=20, line=dict(color='MediumPurple', width=3))))  
figure.update_layout(legend_orientation="h",  
    legend=dict(x=.5, xanchor="center"),  
    hovermode="x",  
    margin=dict(l=0, r=0, t=0, b=0))  
figure.update_traces(hoverinfo="all", hovertemplate="Аргумент: %{x}<br>Функция: %{y}")  
figure.show()
```





—  $h(x)=\sin(x)$   
 —  $k(x)=\cos(x)$   
 —  $m(x)=\text{tg}(x)$   
 —  $f(x)=x^2$   
 ●  $g(x)=x$



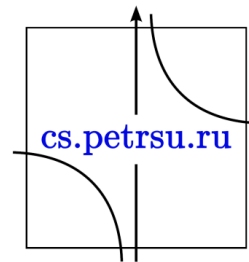
# make\_subplots

- Модуль для создания фигур с несколькими графиками
  - Необходимо указать количество: row — строк, col — столбцов.
- При построении графика передать «координаты» графика в этой «матрице» (сперва строка, потом столбец)

```
figure = make_subplots(rows=1, cols=2,  
                        specs=[[{'type':'domain'}, {'type':'domain'}]])
```
- Можно задать заголовки графиков через аргумент subplot\_titles  
кортеж/список с названиями.

```
figure = make_subplots(rows=1, cols=2,  
                        subplot_titles=("Plot 1", "Plot 2"))
```
- Можно изменять размеры графиков через соотношение 2:1  
(column\_widths — задаёт отношения ширины графиков,  
row\_heights — высот графиков)

```
figure = make_subplots(rows=1, cols=2, column_widths=[2, 1])
```



```

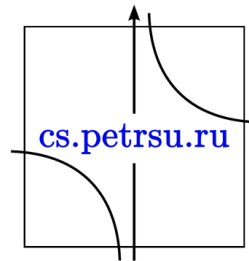
figure = make_subplots(rows=1, cols=2)

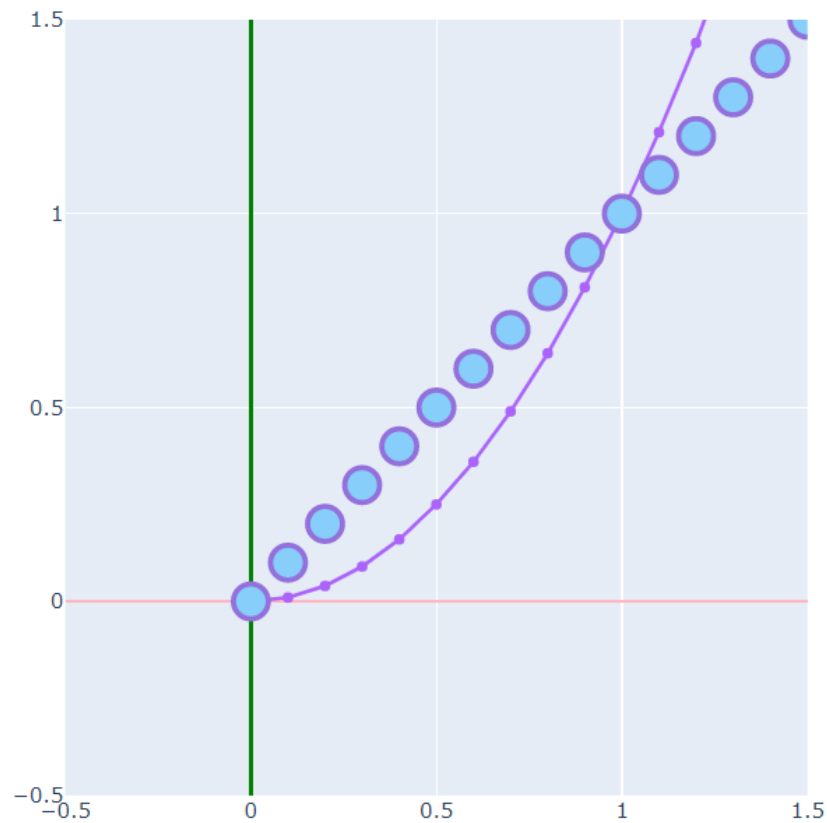
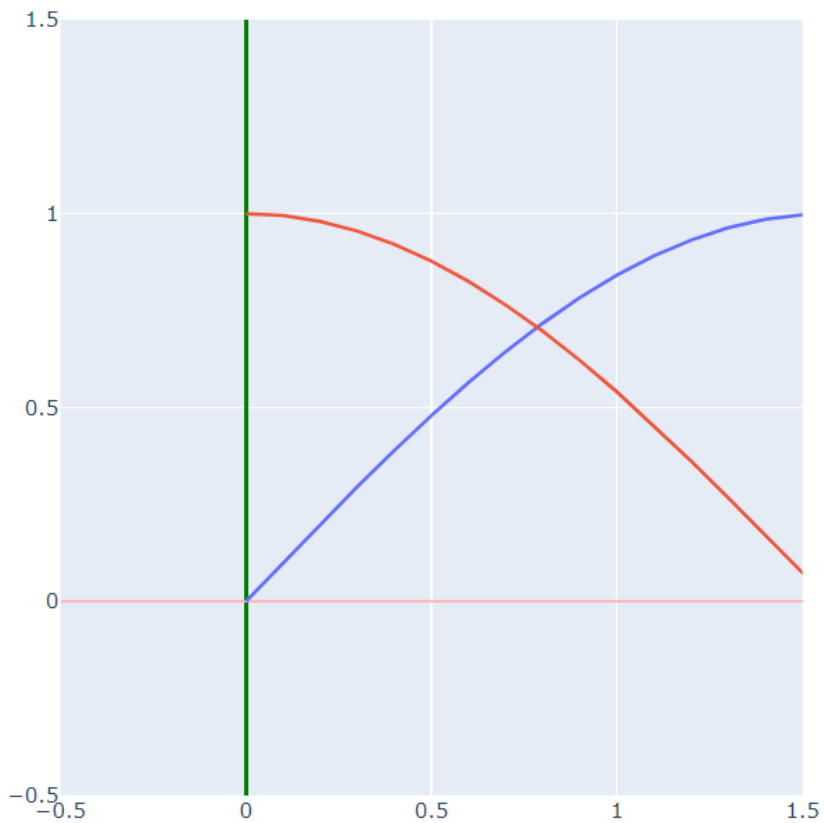
figure.update_yaxes(range=[-0.5, 1.5], zeroline=True, zerolinewidth=2, zerolinecolor='LightPink')
figure.update_xaxes(range=[-0.5, 1.5], zeroline=True, zerolinewidth=2, zerolinecolor='#008000')

figure.add_trace(go.Scatter(x=x, y=h(x), name='h(x)=sin(x)'), 1, 1)
figure.add_trace(go.Scatter(x=x, y=k(x), name='k(x)=cos(x)'), 1, 1)
figure.add_trace(go.Scatter(visible='legendonly', x=x, y=m(x), name='m(x)=tg(x)'), 1, 1)

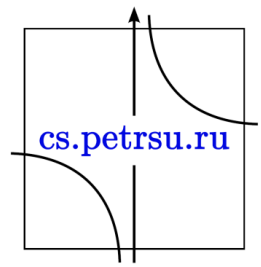
figure.add_trace(go.Scatter(x=x, y=f(x), mode='lines+markers', name='f(x)=x<sup>2</sup>'), 1, 2)
figure.add_trace(go.Scatter(x=x, y=x, mode='markers', name='g(x)=x',
    marker=dict(color='LightSkyBlue', size=20, line=dict(color='MediumPurple', width=3))), 1, 2)
figure.update_layout(legend_orientation="h",
    legend=dict(x=.5, xanchor="center"),
    hovermode="x",
    margin=dict(l=0, r=0, t=0, b=0))
figure.update_traces(hoverinfo="all", hovertemplate="Аргумент: %{x}<br>Функция: %{y}")
figure.show()

```





—  $h(x)=\sin(x)$   
 —  $k(x)=\cos(x)$   
 —  $m(x)=\text{tg}(x)$   
 —●—  $f(x)=x^2$   
 ●  $g(x)=x$

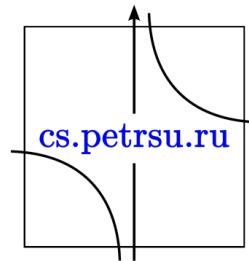


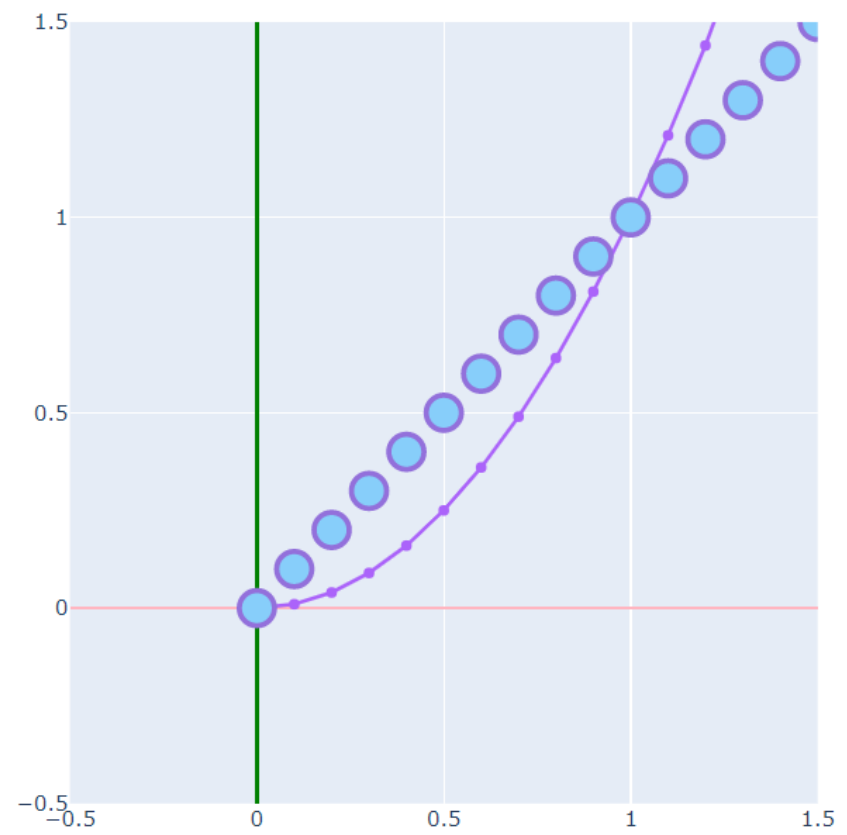
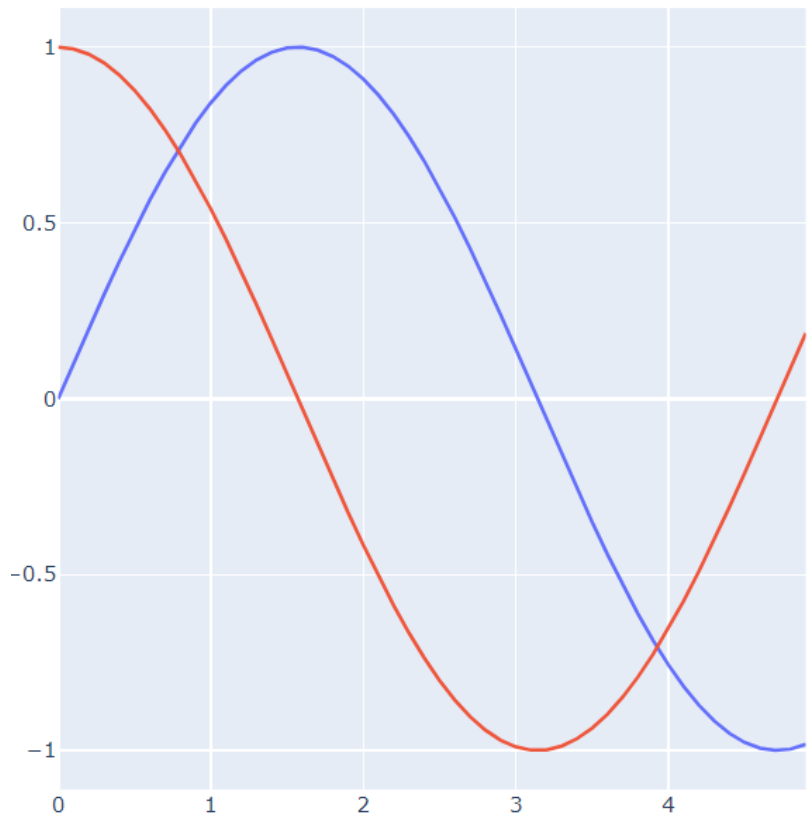
# update\_xaxes/update\_yaxes

- Можно изменить ось только на конкретном графике:

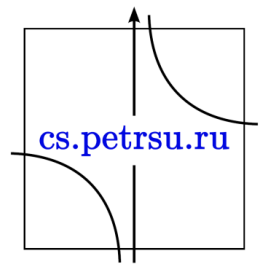
```
figure.update_yaxes(range=[-0.5, 1.5],  
                    zeroline=True, zerolinewidth=2,  
                    zerolinecolor='LightPink', col=2)
```

```
figure.update_xaxes(range=[-0.5, 1.5],  
                    zeroline=True, zerolinewidth=2,  
                    zerolinecolor='#008000', col=2)
```





—  $h(x)=\sin(x)$   
 —  $k(x)=\cos(x)$   
 —  $m(x)=\text{tg}(x)$   
 —●—  $f(x)=x^2$   
 ●  $g(x)=x$





# update\_xaxes/update\_yaxes

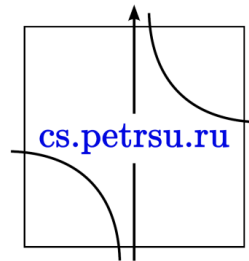
- Переопределим подписи осей для разных подграфиков через title/row/col

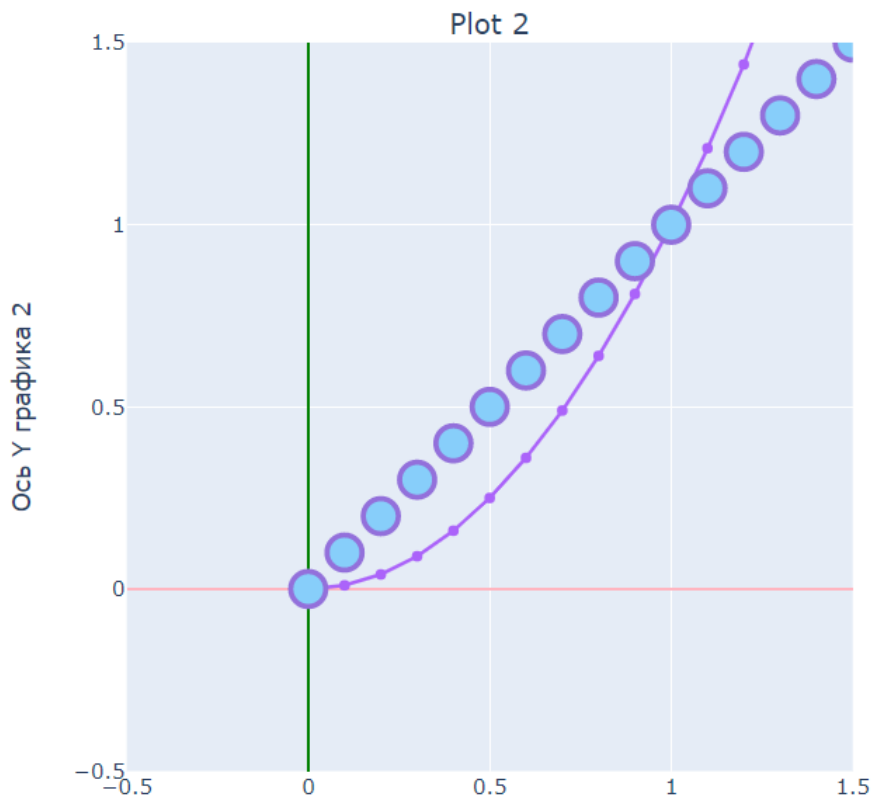
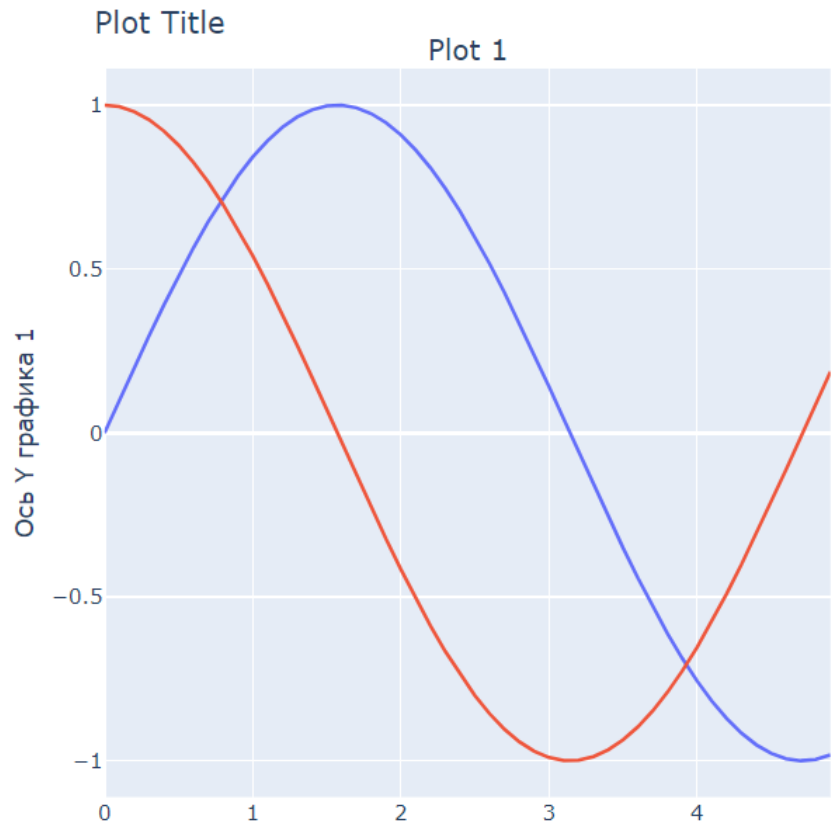
```
figure.update_xaxes(title='Ось X графика 1', col=1, row=1)
```

```
figure.update_xaxes(title='Ось X графика 2', col=2, row=1)
```

```
figure.update_yaxes(title='Ось Y графика 1', col=1, row=1)
```

```
figure.update_yaxes(title='Ось Y графика 2', col=2, row=1)
```





Ось X графика 1

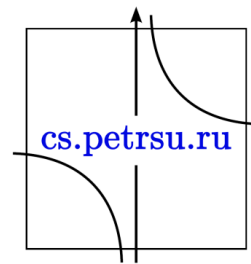
—  $h(x)=\sin(x)$  —  $k(x)=\cos(x)$

—  $m(x)=\text{tg}(x)$

—  $f(x)=x^2$

Ось X графика 2

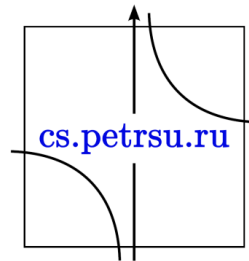
●  $g(x)=x$



# Примеры subplots

- Больше примеров с подграфиками в документации:

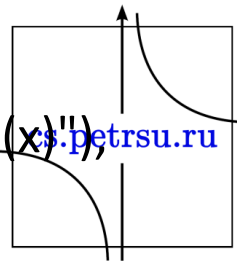
<https://plotly.com/python/subplots/>



# Тепловая карта

- Модифицируем первый график, но увеличим количество отображаемой информации, используя цветовую кодировку.
  - используя тепловую карту — чем выше значение некой величины, тем «теплее» цвет.
- Для этого у объекта `go.Scatter` используем атрибут `marker`, передаём следующие атрибуты в словарь:
  - `color` — список значений по которым будут выбираться цвета. Элементов списка должно быть столько же, сколько и точек.
  - `colorbar` — словарь, описывающий индикационную полосу цветов справа от графика. Принимает на вход словарь. Используем `title` — заголовок полосы.

```
figure.add_trace(go.Scatter(x=x, y=f(x), mode='lines+markers',  
                            name='f(x)=x2',  
                            marker=dict(color=h(x),  
                                        colorbar=dict(title="h(x)=sin(x)",  
                                                    colorscale='Inferno'))))
```



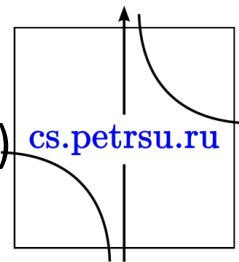
```

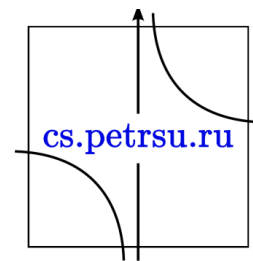
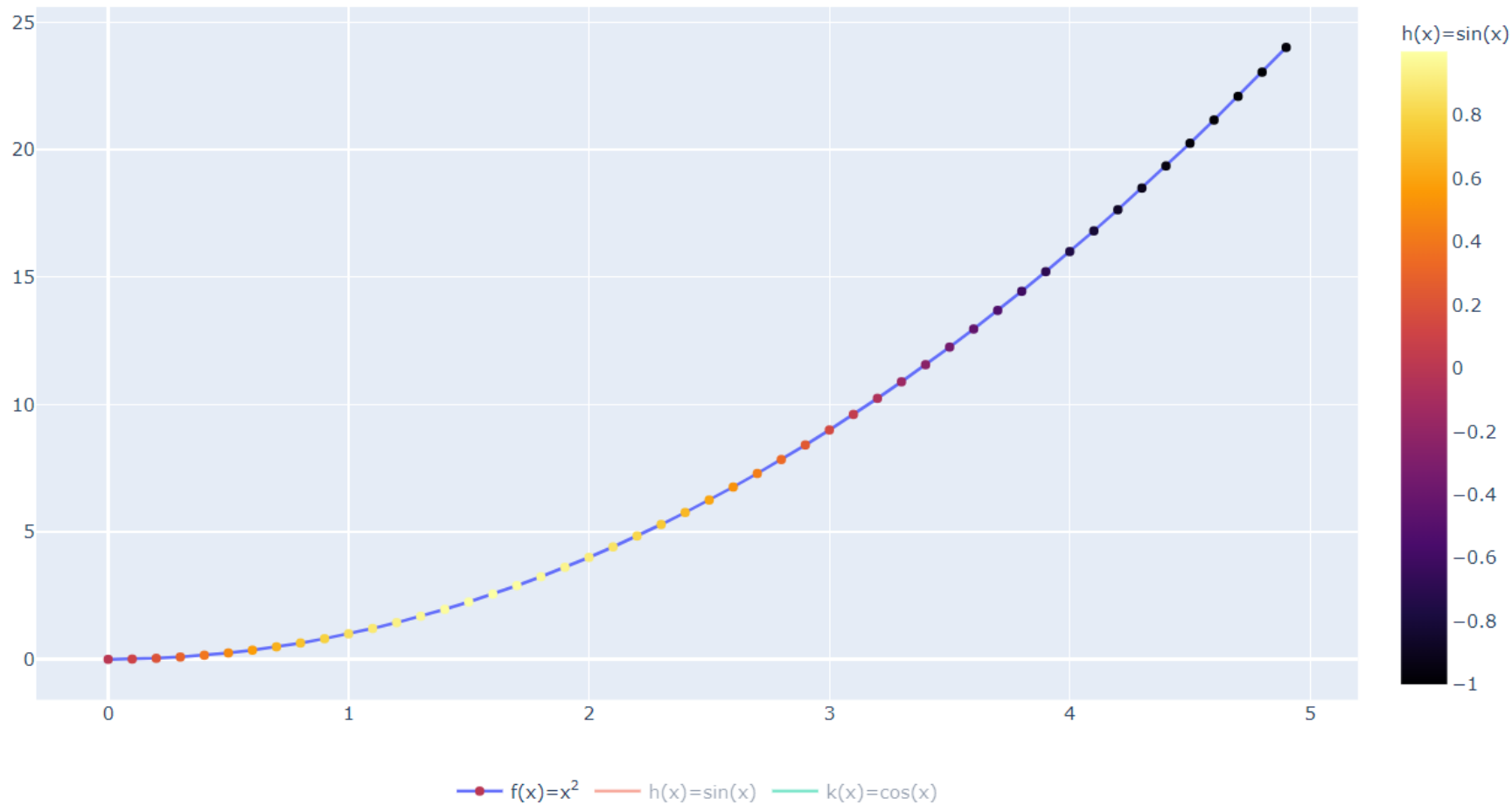
fig = go.Figure()
fig.add_trace(go.Scatter(x=x, y=f(x), mode='lines+markers',
                        name='f(x)=x2',
                        marker=dict(color=h(x),
                                    colorbar=dict(title="h(x)=sin(x)"),
                                    colorscale='Inferno'))))

fig.add_trace(go.Scatter(visible='legendonly', x=x, y=h(x),
                        name='h(x)=sin(x)'))
fig.add_trace(go.Scatter(visible='legendonly', x=x, y=k(x),
                        name='k(x)=cos(x)'))

fig.update_layout(legend_orientation="h",
                  legend=dict(x=.5, xanchor="center"),
                  margin=dict(l=0, r=0, t=0, b=0))
fig.update_traces(hoverinfo="all",
                  hovertemplate="Аргумент: %{x}<br>Функция: %{y}")
fig.show()

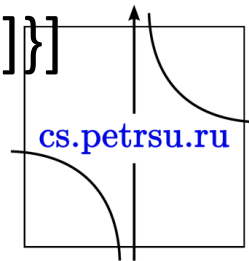
```



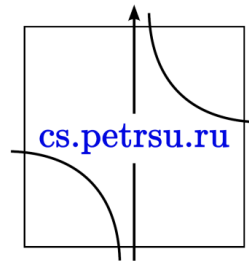


# Анимация

- создается из списка кадров - фреймов
- фрейм - готовый график
- создается через `go.Frame()` с аргументом `data`
- Кнопки (добавляются через `update_layout`):  
"updatemenus": [{"type": "buttons",  
                  "buttons": [{"label": "Your Label",  
                                "method": "animate",  
                                "args": [See Below]}]}]



- Рассмотрим пример создания последовательности:  
frames=[]  
for i in range(1, len(x)):  
frames.append(go.Frame(data=[go.Scatter(x=x[:i],  
y=f(x[:i]))]))
- Далее фреймы необходимо передать в фигуру (атрибут frames):  
figure.frames = frames





# Анимация

- Второй вариант - задать начальное состояние, слой (с кнопками) и все фреймы передать в объект `go.Figure`:

```
frames=[]
```

```
for i in range(1, len(x)):
```

```
    frames.append(go.Frame(data=[go.Scatter(x=x[:i+1], y=f(x[:i+1]))]))
```

```
figure = go.Figure(data=go.Scatter(x=[x[0]], y=[f(x[0])], mode='lines+markers',  
                                   name='f(x)=x2'),
```

```
                    frames=frames,
```

```
                    layout=dict(legend_orientation="h",
```

```
                                legend=dict(x=.5, xanchor="center"),
```

```
                                updatemenus=[dict(type="buttons",
```

```
                                                buttons=[dict(label="Play",
```

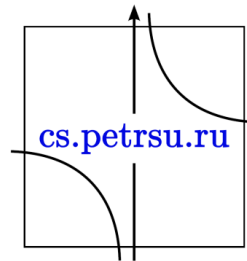
```
                                                            method="animate", args=[None])]),
```

```
                                margin=dict(l=0, r=0, t=0, b=0)))
```

```
figure.update_traces(hoverinfo="all",
```

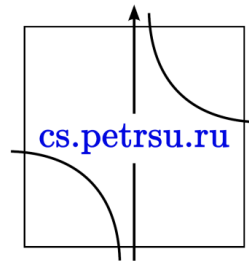
```
                    hovertemplate="Аргумент: %{x}<br>Функция: %{y}")
```

```
figure.show()
```



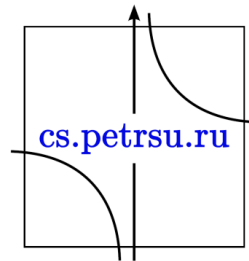
# Слайдер

- похож на анимацию
- это элемент навигации – полоска по которой скользит ползунок, который управляет состоянием графиков на фигуре
  - все графики сразу есть, скрываются/показываются в зависимости от положения ползунка.
- Примеры тут:  
<https://plotly.com/python/animations/>



# Показ plotly графиков на сайте

- Plotly — это не только библиотека для Python, но ещё и JS, это значит, что любые графики, которые вы строите в jupyter notebook, вы можете показывать и на сайте (если он на Python, либо если вы заранее выгрузите всё необходимое).

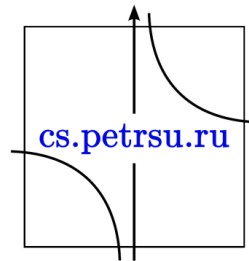


# Пример:

```
import json
#создаём свой график в фигуре figure и выгружаем его в формат json
graphJSON = json.dumps(figure, cls=plotly.utils.PlotlyJSONEncoder)

#выгрузим json в файл
with open('example.JSON', 'w') as file:
    file.write('var graphs = {}'.format(graphJSON))
```

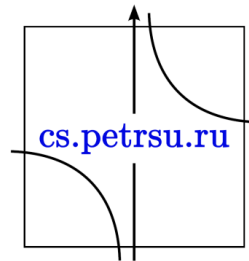
Далее создаем файл html...



```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Пример работы Plotly на сайте</title>
</head>
<body>
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/d3/3.5.6/d3.min.js"></script>

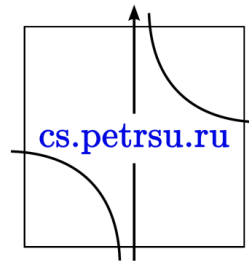
  <h1>Пример работы Plotly на сайте</h1>
  <div class="chart" id="plotly_graph"></div>

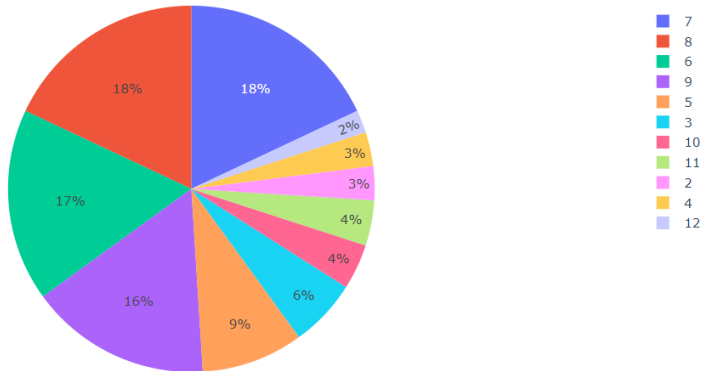
  <script src="example.JSON"></script>
  <script>Plotly.plot('plotly_graph',graphs,{});</script>
</body>
</html>
```



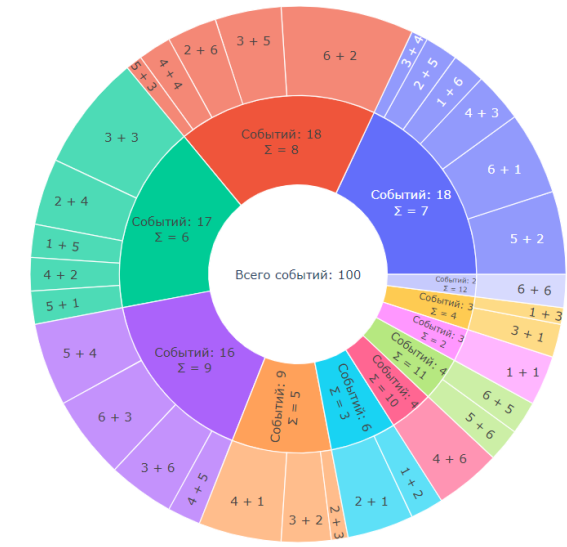
# Что можно ещё

- Круговые диаграммы
- Пузырьковые диаграммы
- Гистограммы
- Контурные графики
- Полярные карты
- Ящики с усами (Box Plots)
- Географические карты(Scattermapbox)
- Поверхности
- Визуализации
- и многое другое... <https://plotly.com/python/>

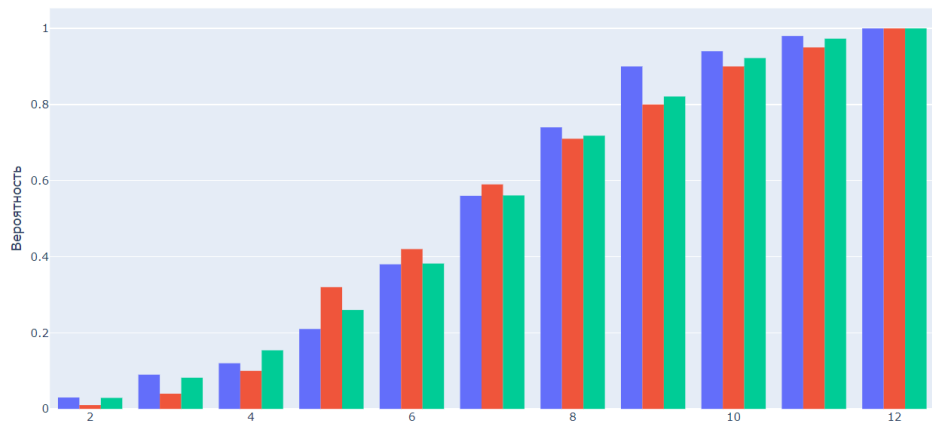
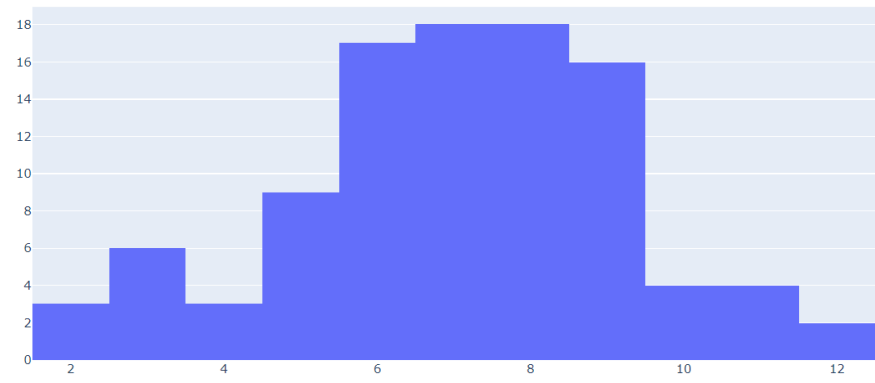




- 7
- 8
- 6
- 9
- 5
- 3
- 10
- 11
- 2
- 4
- 12

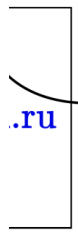
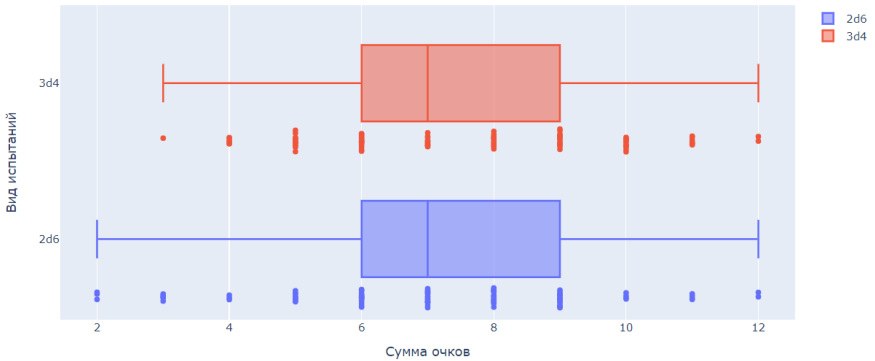


Пример накопительной гистограммы на основе бросков пары игральных костей



Сравнение испытаний по 100 бросков игральных костей

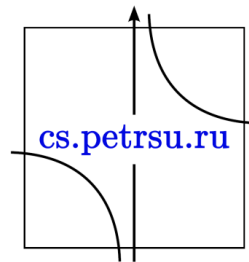
По России на поезде



# Manim

— это средство для создания точных анимации, с помощью которых поясняются разные математические операции

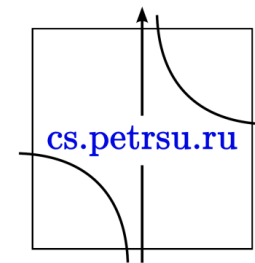
- визуализация для Data Science
- иллюстрировать свои идеи и рассуждения о математике
- улучшение понимания математических концепций (напр. алгоритмы машинного обучения)





# Установка

- <https://docs.manim.community/en/stable/installation.html>
- После установки зависимостей введите команду:  
`pip install manim`



- Код для создания анимации определяется внутри метода `construct` получаемого из `Scene` класса:

```
from manim import *
```

```
config.background_color = DARK_GRAY
```

```
class PointMovingOnShapes(Scene):
```

```
    def construct(self):
```

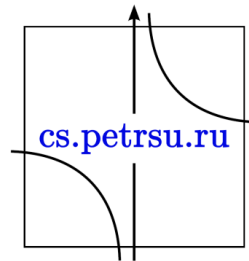
```
        square = Square(color=BLUE) # Create a square
```

```
        square.flip(RIGHT) # Flip the square to the right
```

```
        square.rotate(-3 * TAU / 8) # Rotate the square  $-3/8 * 2 * \pi$ 
```

```
        # Play the animation of a square growing from the center
```

```
        self.play(GrowFromCenter(square))
```



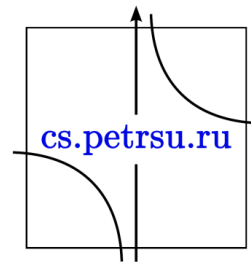
- Сохраните этот скрипт с именем `start.py`.  
Запустите команду ниже, чтобы сгенерировать видео из скрипта.

```
manim -p -ql start.py PointMovingOnShapes
```

```
# Будет сохранено видео PointMovingOnShapes.mp4
```

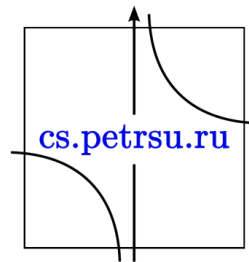
```
manim -p -ql -i start.py PointMovingOnShapes
```

```
# -i сгенерировать GIF-анимацию
```



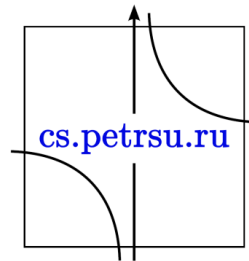
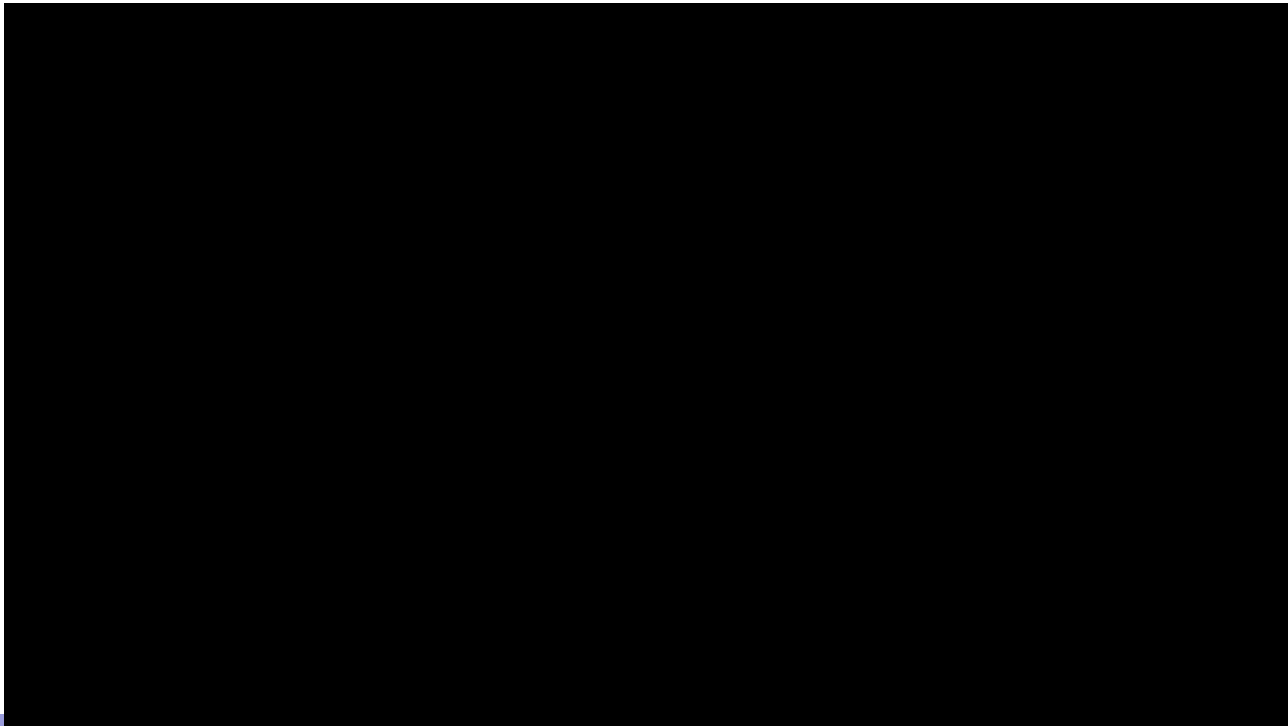
- Более подробно:

<https://docs.manim.community/en/stable/reference/manim.mobject.geometry.html>



# Уравнения с подвижной рамкой

- Также можно создавать анимации, выводящие математические уравнения с подвижной рамкой, например такие:



```
class MovingFrame(Scene):
    def construct(self):
        # Write equations
        equation = MathTex("2x^2-5x+2", "=", "(x-2)(2x-1)")

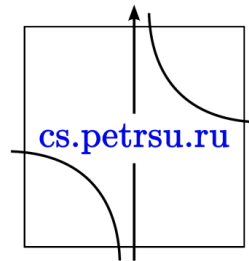
        # Create animation
        self.play(Write(equation))

        # Add moving frames
        framebox1 = SurroundingRectangle(equation[0], buff=.1)
        framebox2 = SurroundingRectangle(equation[2], buff=.1)

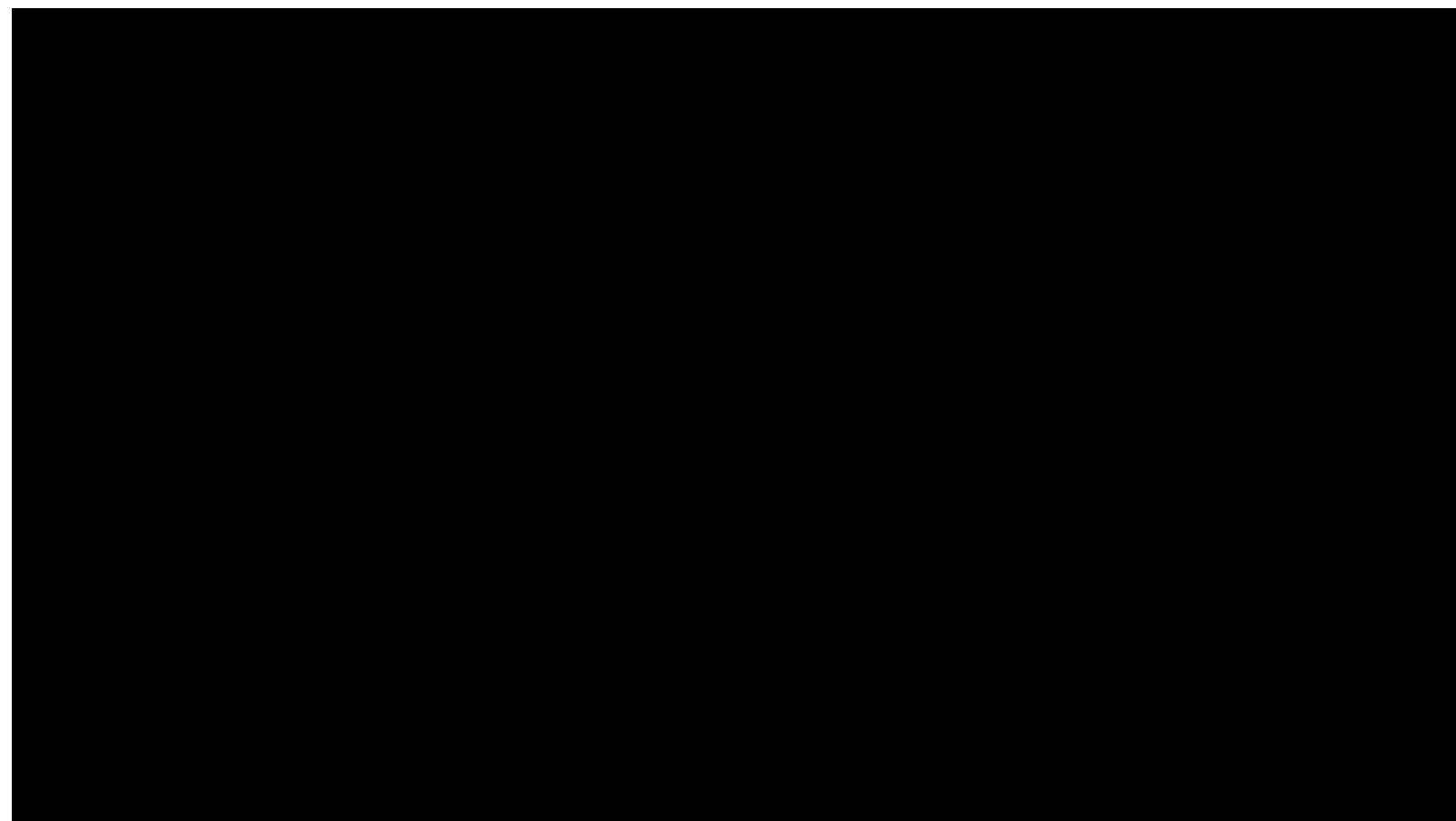
        # Create animations
        self.play(Create(framebox1)) # creating the frame

        self.wait()
        # replace frame 1 with frame 2
        self.play(ReplacementTransform(framebox1, framebox2))

        self.wait()
```



- Или записывать шаги решения уравнения:



```

class MathematicalEquation(Scene):
    def construct(self):

        # Write equations
        equation1 = MathTex("2x^2-5x+2")
        eq_sign_1 = MathTex("=")
        equation2 = MathTex("2x^2-4x-x+2")
        eq_sign_2 = MathTex("=")
        equation3 = MathTex("(x-2)(2x-1)")

        # Put each equation or sign in the appropriate positions
        equation1.next_to(eq_sign_1, LEFT)
        equation2.next_to(eq_sign_1, RIGHT)

        eq_sign_2.shift(DOWN)
        equation3.shift(DOWN)

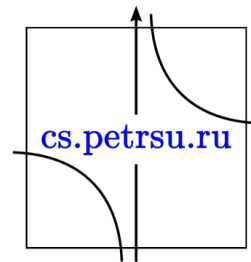
        # Align bottom equations with the top equations
        eq_sign_2.align_to(eq_sign_1, LEFT)
        equation3.align_to(equation2, LEFT)

        # Group equations and sign
        eq_group = VGroup(equation1, eq_sign_1, equation2, eq_sign_2, equation3)

        # Create animation
        self.play(Write(eq_group))

        self.wait()

```

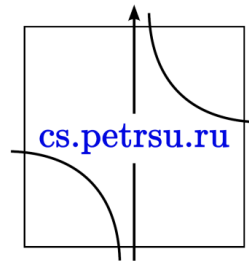




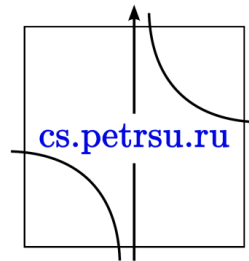
# Перемещение и панорамирование

Также можно с помощью объекта  
MovingCameraScene:

- направлять "камеру" на отдельные части уравнения
- увеличивать масштабы отображения таких мест

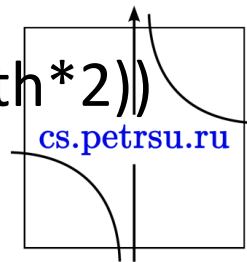


$$2x^2 - 5x + 2 = (x - 2)(2x - 1)$$



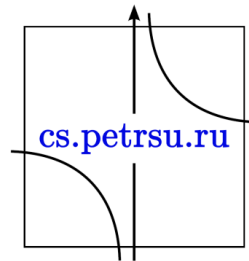
```
class MovingAndZoomingCamera(MovingCameraScene):
    def construct(self):
        # Write equations
        equation = MathTex("2x^2-5x+2", "=", "(x-2)(2x-1)")

        self.add(equation)
        self.play(self.camera.frame.animate.move_to(
            equation[0]).set(width=equation[0].width*2))
        self.wait(0.3)
        self.play(self.camera.frame.animate.move_to(
            equation[2]).set(width=equation[2].width*2))
```



# Построение графиков

- Пакет `manim` также можно использовать для создания аннотированных графиков:

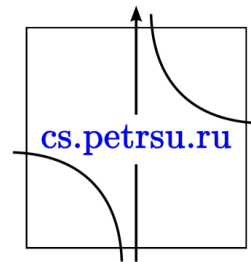


```

class Graph(GraphScene):
    def __init__(self, **kwargs):
        GraphScene.__init__(
            self,
            x_min=-3.5,
            x_max=3.5,
            y_min=-5,
            y_max=5,
            graph_origin=ORIGIN,
            axes_color=BLUE,
            x_labeled_nums=range(-4, 4, 2), # x tickers
            y_labeled_nums=range(-5, 5, 2), # y tickers
            **kwargs
        )

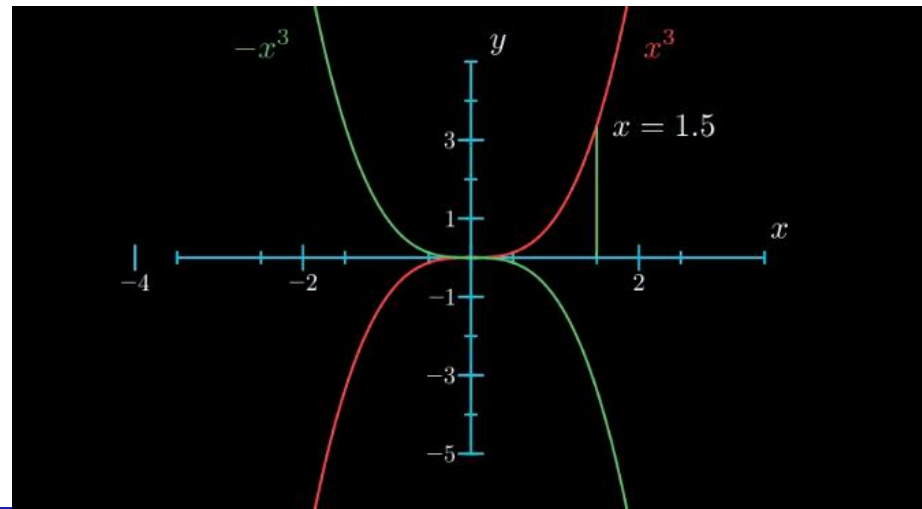
    def construct(self):
        self.setup_axes(animate=False)
        # Draw graphs
        func_graph_cube = self.get_graph(lambda x: x**3, RED)
        func_graph_ncube = self.get_graph(lambda x: -x**3, GREEN)
        # Create labels
        graph_lab = self.get_graph_label(func_graph_cube, label="x^3")
        graph_lab2 = self.get_graph_label(func_graph_ncube, label="-x^3", x_val=-3)
        # Create a vertical line
        vert_line = self.get_vertical_line_to_graph(1.5, func_graph_cube, color=YELLOW)
        label_coord = self.input_to_graph_point(1.5, func_graph_cube)
        text = MathTex(r"x=1.5")
        text.next_to(label_coord)
        self.add(func_graph_cube, func_graph_ncube, graph_lab, graph_lab2, vert_line, text)
        self.wait()

```



- Если нужно получить изображение последнего кадра сцены, добавьте к команде опцию `-s`:

```
manim -p -qh -s more.py Graph
```

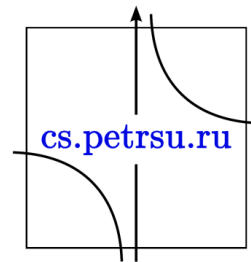


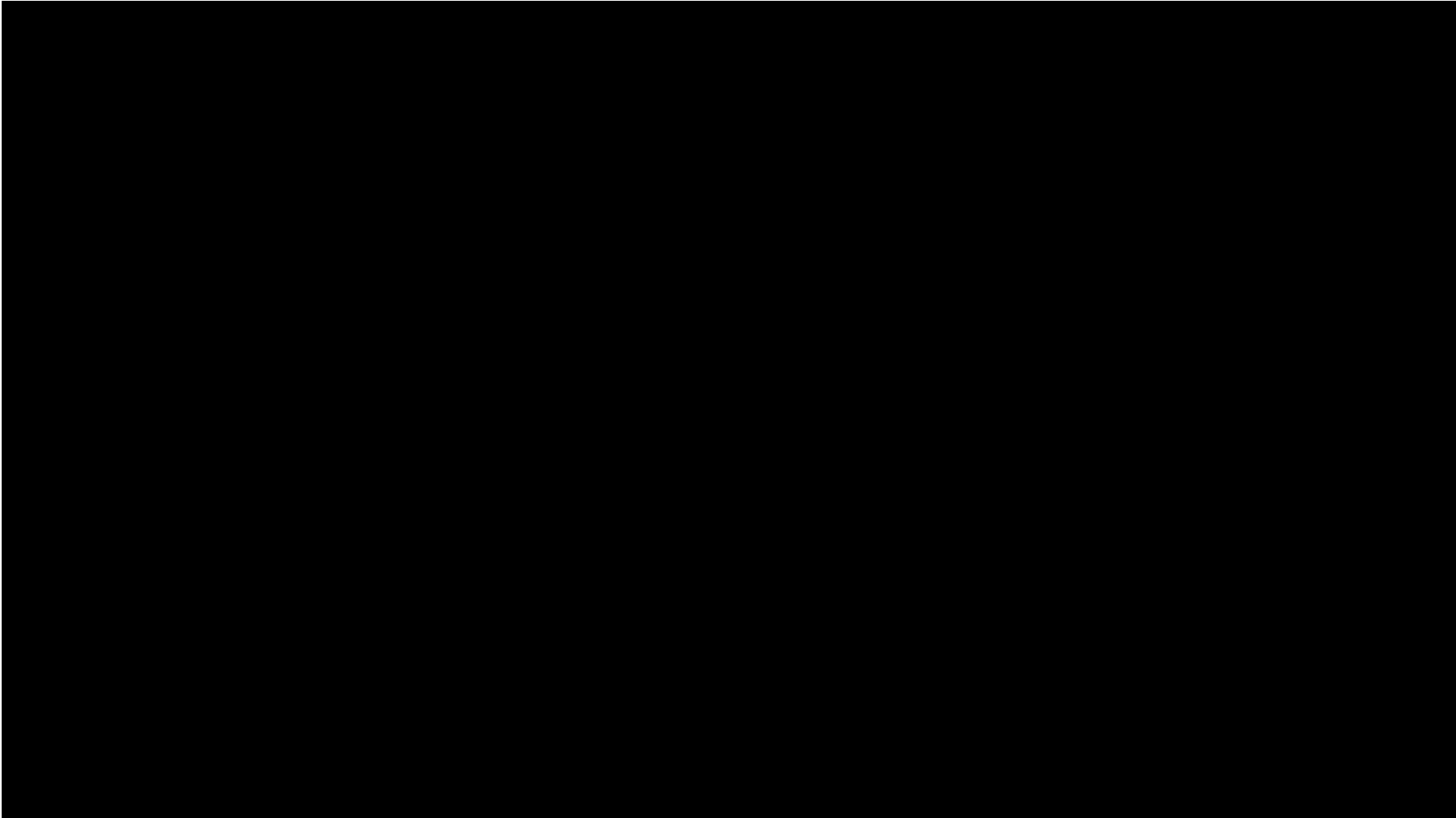
- Также можно анимировать процесс добавления осей: для этого нужно установить параметр `animate=True`.

```
def construct(self):  
    self.setup_axes(animate=True)
```

Создаем анимацию:

```
manim -p -qh -i more.py Graph
```

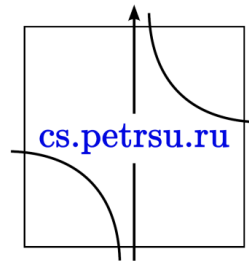
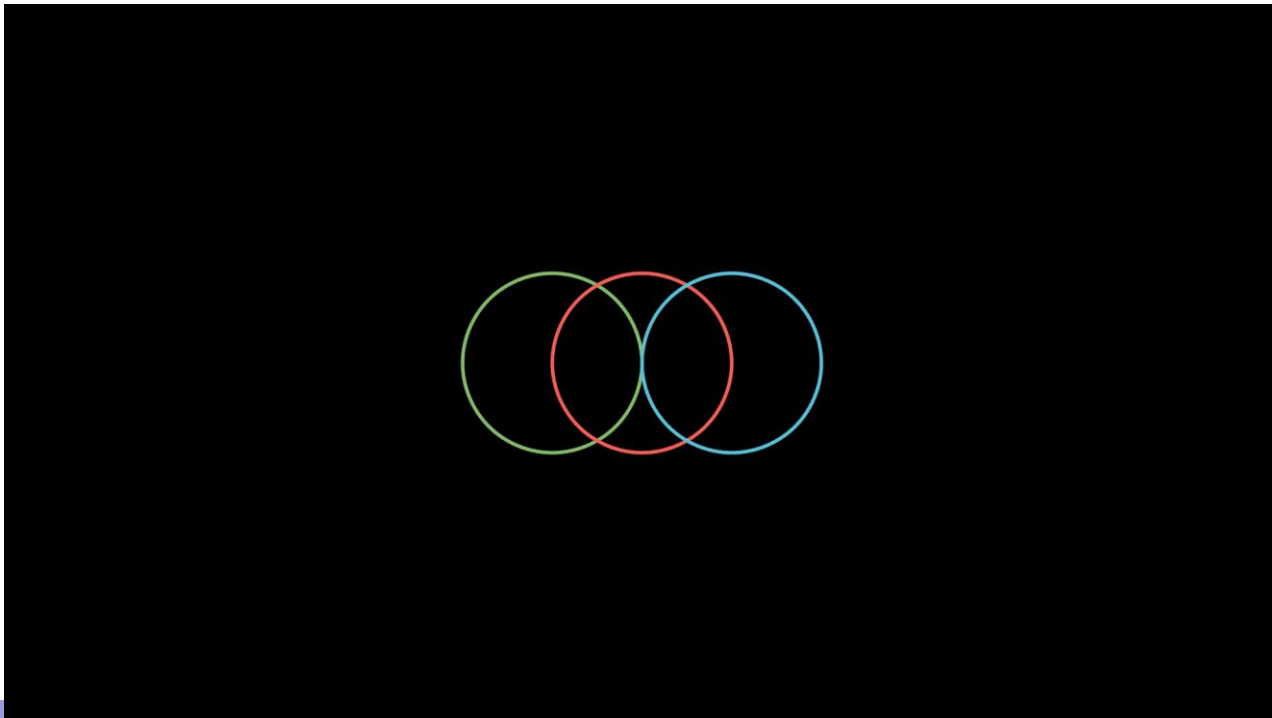






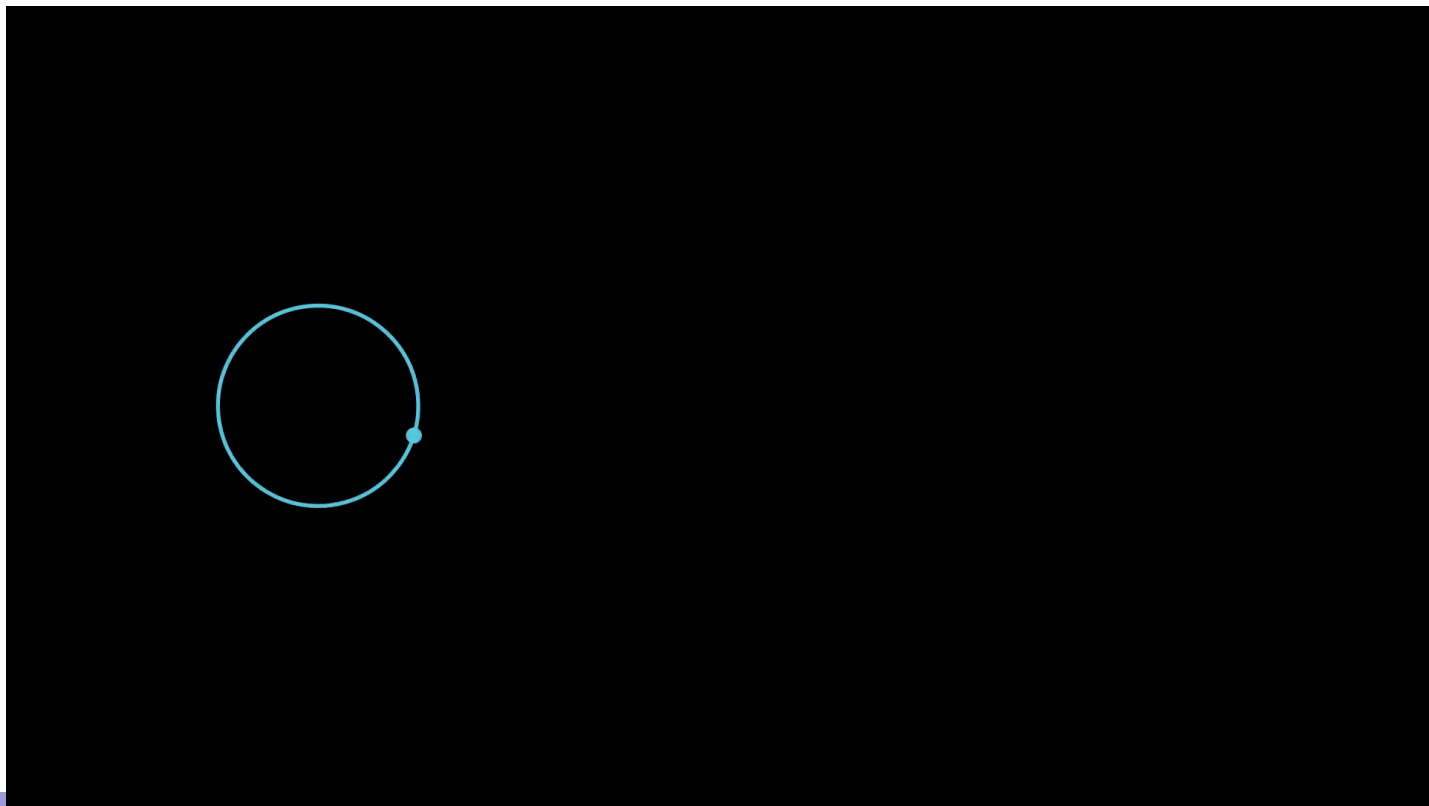
# Совместное перемещение объектов

- Для группировки различных объектов Manim и их совместного перемещения можно воспользоваться `VGroup`:



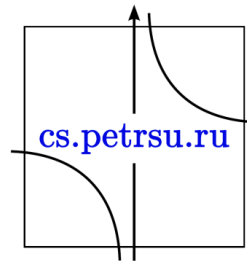
# Отслеживание перемещения

- Для отображения следа движущегося объекта можно воспользоваться TracedPath:



# Итоги

- Пакет `manim` работает с тремя видами объектов:
  - `Objects`: объекты, которые могут выводиться на экран, например `Circle` (Окружность), `Square` (Квадрат), `Matrix` (Матрица), `Angle` (Угол) и пр.
  - Сцены: "холсты" для создания анимаций, например `Scene`, `MovingCameraScene` и пр.
  - Анимации: анимации, применяемые к объектам `Objects`, например `Write` (Записать), `Create` (Создать), `GrowFromCenter` (Увеличить от центра), `Transform` (Преобразовать) и пр.
- Ознакомиться с руководством по использованию `manim` можно тут <https://docs.manim.community/en/stable/tutorials.html>
- [https://notebooks.gesis.org/binder/jupyter/user/manimcommunity-jupyter-examples-5qle6fbb/lab/workspaces/automatic/tree/basic\\_example\\_scenes.ipynb](https://notebooks.gesis.org/binder/jupyter/user/manimcommunity-jupyter-examples-5qle6fbb/lab/workspaces/automatic/tree/basic_example_scenes.ipynb)
- При помощи `manim` красота математики приобретает вещественное выражение.



# Другие...

- py-gnuplot

<https://pypi.org/project/py-gnuplot/>

- ...

