# Построение графиков и полей в matplotlib

Matplotlib (https://matplotlib.org/)— библиотека на языке программирования Python для визуализации данных двумерной и трёхмерной графики.

Пакет поддерживает многие виды графиков и диаграмм:

- Графики (англ. line plot)
- Диаграммы рассеяния (англ. scatter plot)
- Столбчатые диаграммы (англ. bar chart) и гистограммы (англ. histogram)
- Круговые диаграммы (англ. pie chart)
- Диаграммы стебель-листья (англ. stem plot)
- Контурные графики (англ. contour plot)
- Поля градиентов (англ. quiver)
- Спектральные диаграммы (англ. spectrogram)

Пользователь может указать оси координат, сетку, добавить надписи и пояснения, использовать логарифмическую шкалу или полярные координаты, визуализировать поля данных.

Типичные поддерживаемые форматы:

- Encapsulated PostScript (EPS)
- Enhanced Metafile (EMF)
- JPEG
- PDF
- PNG
- Postscript
- RGBA («сырой» формат)
- SVG
- SVGZ
- TIFF

Для импорта библиотеки Matplotlib в рабочий проект необходимо в головном модуле программы ввести команду:

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

где plt – это сокращённое название библиотеки, которое в дальнейшем будет использоваться в тексте программы.

Также для работы с Matplotlib могут понадобиться модули scipy и numpy:

```
import scipy
import numpy as np
```

Перед построением графиков нужно ознакомиться с их структурой (рисунок 1). Для отображения графика необходимо в коде программы прописать все структуры, которые необходимо отрисовать.

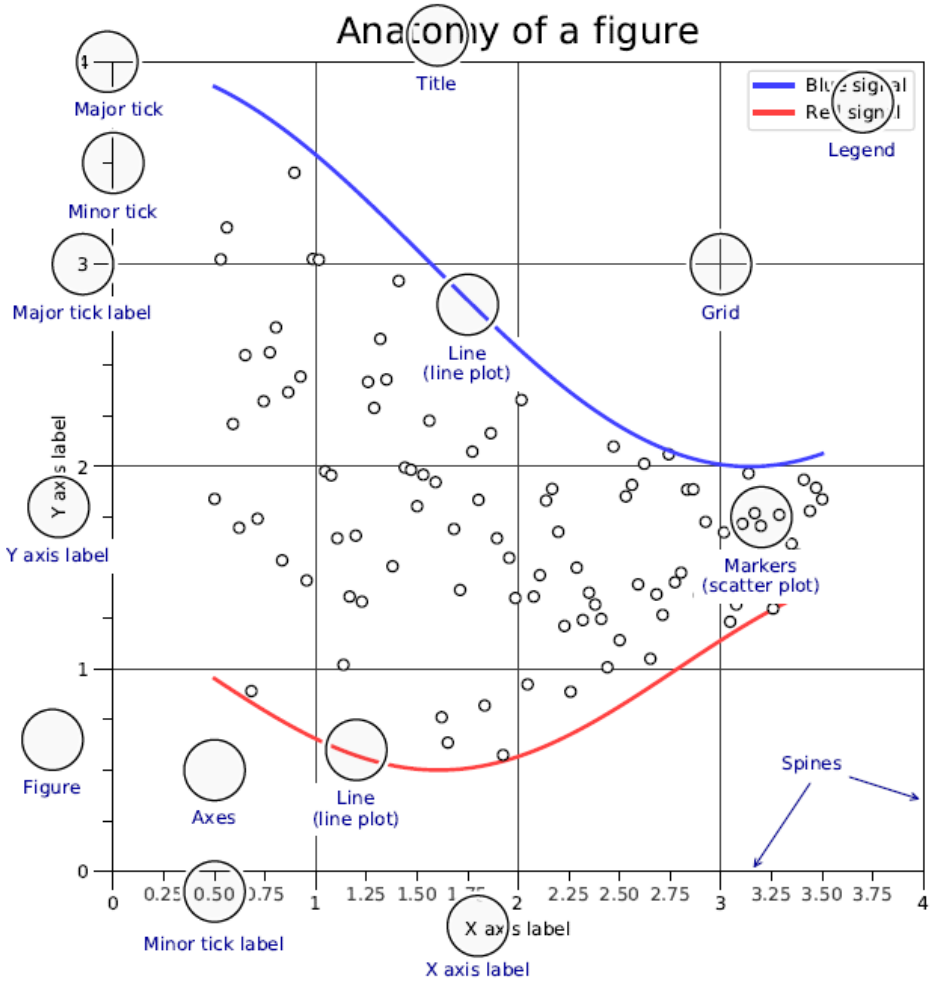


Рисунок 1 Строение графика

Элементы графика

**Figure**: самый важный элемент графика – он сам! **Figure** создаётся при вызове метода figure при этом сразу можно задать размер figure, цвет фона (facecolor) и заголовок или подзаголовок (title или suptitle). Важно отметить, что при сохранении рисунка, цвет фона будет заменён на белый, т.к. метод savefig в качестве аргумента для цвета фона по умолчанию имеет белый цвет.

**Axes**: Это второй по важности элемент, которому соответствует фактическая область, где отображаются данные. Его также называют subplot. На одной **Figure** может отображаться несколько осей, каждые из которых могут быть обрамлены четырьмя краями (левый, верхний, правый и нижний), которые называются **spines**. Каждый **spines** может декорироваться основными и вспомогательными галочками (**ticks**), метками галочек (tick labels) и подписью (label). По умолчанию matplotlib отображает только левую и нижнюю оси.

**Axis**: декорированный spines называется осью (axis). Горизонтальная ось – xaxis, вертикальная – yaxis.

**Spines**: Spines — это линии, соединяющие деления осей и отмечающие границы области данных. Они могут быть размещены в произвольных положениях и могут быть видимым или невидимым. Каждый график может содержать линии, маркеры, стобцы и т.п. Каждый элемент имеет множество настроек: цвет, прозрачность, толщина линий, тип линий и т.п.

Например, чтобы изменить толщину оси X на жирную нужно ввести следующие команды:

```
fig, ax = plt.subplots(figsize=(5,2))
for label in ax.get_xaxis().get_ticklabels():
```

```
label.set_fontweight("bold")
plt.show()
```

Одним из важных свойств любого примитива является свойство **zorder**, которое указывает на виртуальную глубину объектов (рисунок 2). **Zorder** используется для сортировки примитивов от низшего к высшему перед их отображением. Он позволяет контролировать «слой» объекта. Большинство объектов имеют значение **zorder** по умолчанию. Например, spines, ticks и label обычно располагаются позади отображаемых данных.

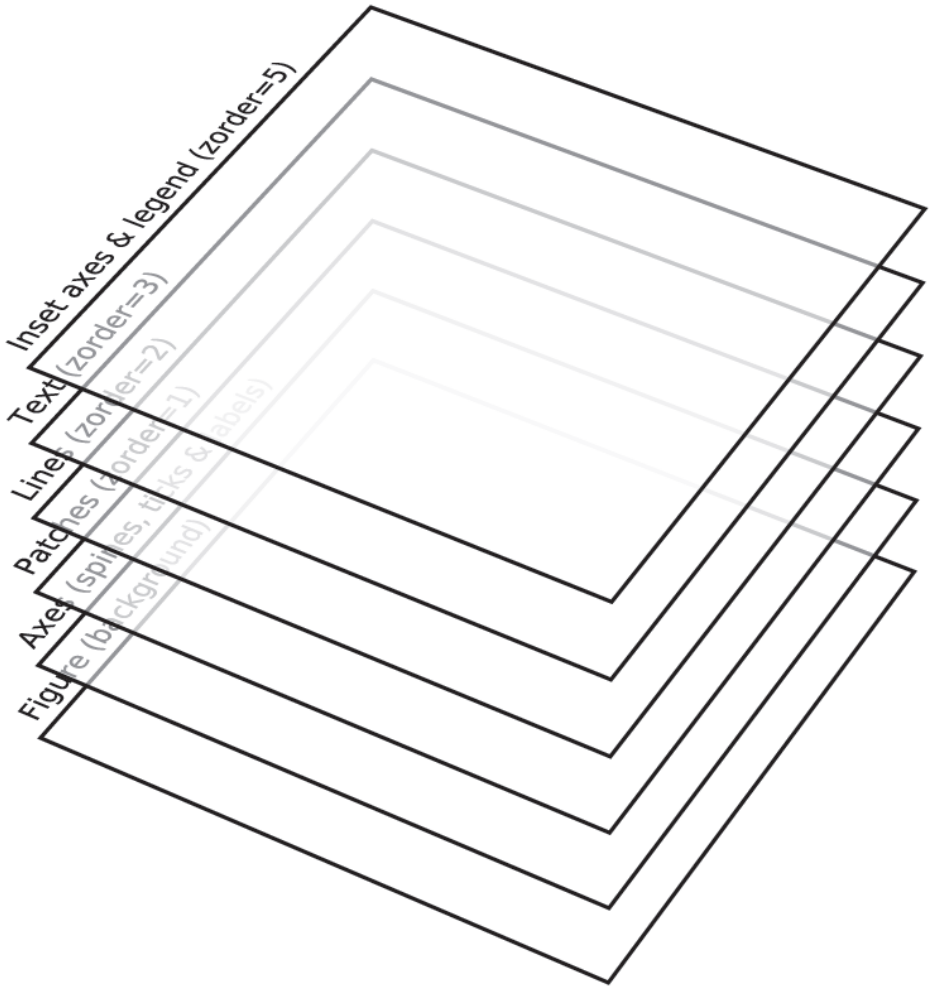


Рисунок 2 Расположение элементов графика по слоям

## Создание простого рисунка с параметрами по умолчанию

Создадим скрипт для рисования простого рисунка — графика зависимости $y = \sin(x)$. Для отображения набора пар значений $x$ и $y$ используется функция **plot()** из модуля **matplotlib.pyplot**. Синтаксис вызова функции:

**matplotlib.pyplot.plot(x, y, args)**

где $x$ — массив значений, отображаемых по оси абсцисс (массив координат), $y$ — массив значений, откладываемых по оси ординат (массив значений функции), **args** — перечисленные через запятую необязательные дополнительные аргументы для настройки графика. Если в функцию **plot()** передаётся только один аргумент-массив, то в качестве массива координат для рисования графика будут использованы индексы элементов этого массива.

Для изображения функциональной зависимости $y = f(x)$ необходимо создать массивы координат в соответствующих точках. Далее приведён текст программы для рисования графика зависимости $y = \sin(x)$. Массив координат создаётся с помощью функции **linspace()**, а массив значений функции — с помощью универсальной функции **sin()** из библиотеки **NumPy**. В конце программы вызывается функция **show()** из модуля **pyplot**, предназначенная для отображения рисунка на экране компьютера.

```
# подключение модуля pyplot под псевдонимом plt
import matplotlib.pyplot as plt
# подключение библиотеки numpy под псевдонимом np
import numpy as np
# массив координат - 30 точек, равнораспределенных в диапазоне от 0 до 10
x = np.linspace(0.0, 10.0, 30)
# массив значений функции в заданных координатах
y = np.sin(x)
# рисование графика функции с помощью функции plot
plt.plot(x, y)
# отображение рисунка на экране
plt.show()
```

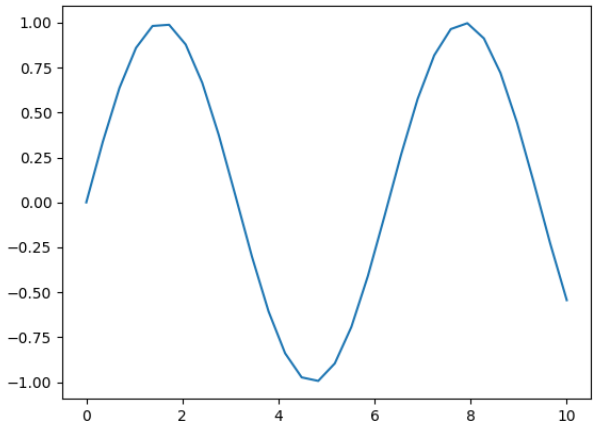


**Рисунок 3 Пример отображения графика с параметрами по умолчанию**

Далее можно настроить построенный график. Рассмотрим основные свойства:

**set_xlim(**a, b**)** — установка диапазона отображаемых по оси абсцисс значений от a до b;

s**et_ylim(**a, b**)** — установка диапазона отображаемых по оси ординат значений от a до b;

**set_xlabel(**str**)** — установка строки str в качестве подписи оси абсцисс;

**set_ylabel(**str**)** — установка строки str в качестве подписи оси ординат;

**set_title(**str**)** — установка строки str в качестве заголовка панели;

**set_xscale(**str**)** — установка масштаба по оси абсцисс, если str=”lin”, то будет использован линейный масштаб, если str=”log”, то логарифмический. По умолчанию все оси отображаются с линейным масштабом;

**set_yscale(**str**)** — установка масштаба по оси ординат;

**legend()** — отображение легенды на рисунке.

В **Matplotlib** реализовано большое количество функций для рисования графиков, например:

**plot(**x, y, args**)** — рисование графика зависимости $y$ от $x$ с помощью линий или маркеров. Через запятую после $y$ можно перечислить необязательные дополнительные аргументы **args** для настройки свойств линий и маркеров (цвет, тип, толщина линии, тип маркера и т. п.);

**errorbar(**x, y, xerr, yerr**)** — рисование графика зависимости $y$ от $x$ с указанием баров ошибок **xerr** для $x$ и **yerr** для $y$;

**hist(**x, bins**)** — построение гистограммы величины $x$, разбитой на число столбцов, равное целому числу bins;

**contour(**x, y, z, [levels], args**)** — рисование контуров величины Z, представляющей собой двумерный массив чисел. Аргументы $x$ и $y$ — массивы координат, в которых определены значения величины $z$. Аргумент [levels] представляет собой список значений величины $z$, которые необходимо изобразить контурами;

**contourf(**x, y, z, [levels], [args]**)** — то же, что **contour()**, только пространство между контурами заполняется цветовой заливкой.

## Размер графика и разрешение:

Размер рисунка задаётся опцией figsize:

```
fig = plt.figure(figsize=(6,6))
plt.savefig("output.png")
```

В данном примере размер рисунка будет 6 на 6 дюймов (количество точек на дюйм (dpi) по умолчанию равно 100). Изменить dpi можно при сохранении рисунка:

```
plt.savefig("name.png".format(dpi), dpi=dpi).
```

## Масштаб по осям

Очень часто приходится строить графики с различными масштабами по осям. Например, в курсе тепломассообмена очень часто используются логарифмические шкалы для отображения зависимостей числа Нуссельта от числа Рейнольдса.

Ниже представлены примеры изменения шкал на различных слоях графика. Отображаются зависимости: $y = 10^x$, $y = x$, $y = \log(x)$.

Тип оси задаётся через параметр set_xscale .

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter, MultipleLocator

X = np.linspace(0.001, 90, 5000)
figure = plt.figure(figsize=(6, 6))

# Style
# ------------------------------------------------------------------------
plt.rc("xtick", labelsize="small")
```

```
plt.rc("ytick", labelsize="small")
plt.rc("axes", labelsize="medium", titlesize="medium")


# X-linear Y-linear
# ------------------------------------------------------------------------------
ax1 = plt.subplot(2, 2, 1, xlim=(0.0, 10), ylim=(0.0, 10))
ax1.plot(X, 10 ** X, color="C0")
ax1.plot(X, X, color="C1")
ax1.plot(X, np.log10(X), color="C2")
ax1.set_ylabel("Linear")
ax1.xaxis.set_major_locator(MultipleLocator(2.0))
ax1.xaxis.set_minor_locator(MultipleLocator(0.4))
ax1.yaxis.set_major_locator(MultipleLocator(2.0))
ax1.yaxis.set_minor_locator(MultipleLocator(0.4))
ax1.grid(True, "minor", color="0.85", linewidth=0.50, zorder=-20)
ax1.grid(True, "major", color="0.65", linewidth=0.75, zorder=-10)
ax1.tick_params(which="both", labelbottom=False, bottom=False)

ax1.text(1.25, 8.50, "$f(x) = 10^x$", color="C0")
ax1.text(5.75, 5.00, "$f(x) = x$", color="C1")
ax1.text(5.50, 1.50, "$f(x) = log_{10}(x)$", color="C2")
ax1.set_title("X linear - Y linear")


# X-log Y-linear
# ------------------------------------------------------------------------------
ax2 = plt.subplot(2, 2, 2, xlim=(0.001, 100), ylim=(0.0, 10), sharey=ax1)
ax2.set_xscale("log")
ax2.tick_params(which="both", labelbottom=False, bottom=False)
ax2.tick_params(which="both", labelleft=False, left=False)
ax2.plot(X, 10 ** X, color="C0")
ax2.plot(X, X, color="C1")
ax2.plot(X, np.log10(X), color="C2")
ax2.grid(True, "minor", color="0.85", linewidth=0.50, zorder=-20)
ax2.grid(True, "major", color="0.65", linewidth=0.75, zorder=-10)
ax2.set_title("X logarithmic - Y linear")


# X-linear Y-log
# ------------------------------------------------------------------------------
--
ax3 = plt.subplot(2, 2, 3, xlim=(0.0, 10), ylim=(0.001, 100), sharex=ax1)
ax3.set_yscale("log")
ax3.plot(X, 10 ** X, color="C0")
ax3.plot(X, X, color="C1")
ax3.plot(X, np.log10(X), color="C2")
ax3.set_ylabel("Logarithmic")
ax3.set_xlabel("Linear")
ax3.grid(True, "minor", color="0.85", linewidth=0.50, zorder=-20)
ax3.grid(True, "major", color="0.65", linewidth=0.75, zorder=-10)
ax3.set_title("X linear - Y logarithmic")

# X-log Y-log
# ------------------------------------------------------------------------------
ax4 = plt.subplot(2, 2, 4, xlim=(0.001, 100), ylim=(0.001, 100), sharex=ax2,
sharey=ax3)
ax4.set_xscale("log")
ax4.set_yscale("log")
ax4.tick_params(which="both", labelleft=False, left=False)
ax4.plot(X, 10 ** X, color="C0")
ax4.plot(X, X, color="C1")
ax4.plot(X, np.log10(X), color="C2")
```

```
ax4.set_xlabel("Logarithmic")
ax4.grid(True, "minor", color="0.85", linewidth=0.50, zorder=-20)
ax4.grid(True, "major", color="0.65", linewidth=0.75, zorder=-10)
ax4.set_title("X logarithmic - Y logarithmic")

# Show
# ------------------------------------------------------------------------------
plt.tight_layout()
plt.show()
```

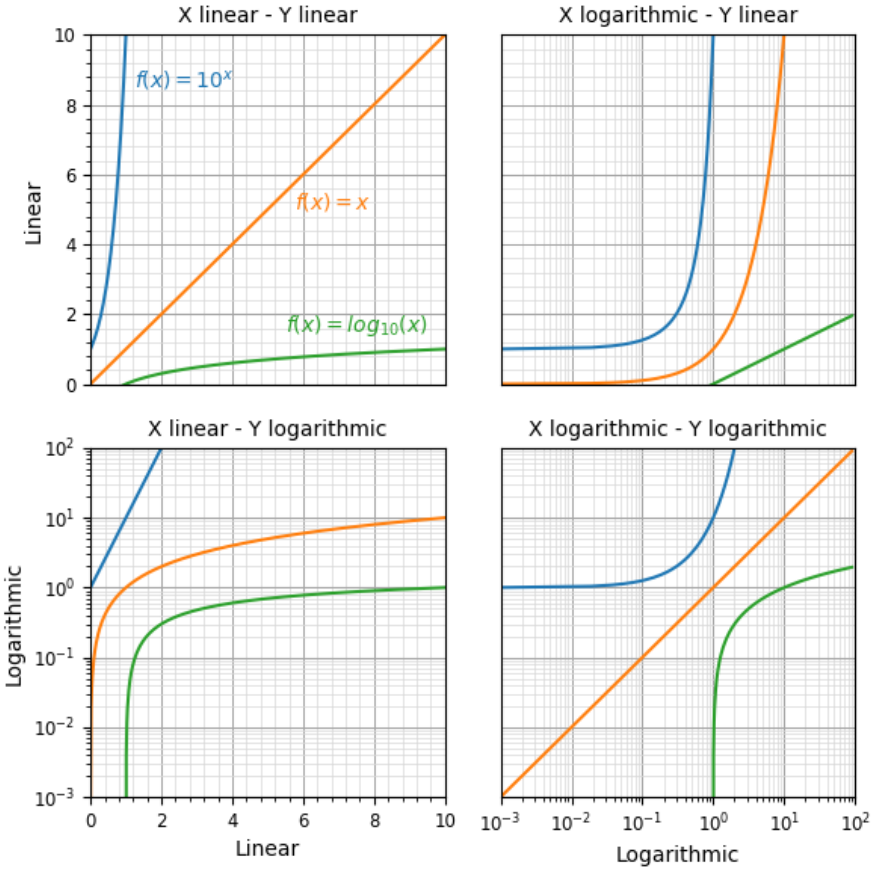


Рисунок 4 Пример изменения типа осей

## Отображение формул на графике

Часто на графиках возникает необходимость в отображении различных формул для создания легенды. В matplotlib есть возможность создавать подписи из формул в формате Latex. В примере ниже представлено создание и отображение формулы из Рисунок 5. Формула задаётся в строке вида **r"$формула$".**

```
import os
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams["text.usetex"] = False

def plot(family):
    plt.rcParams["mathtext.fontset"] = family
    fig = plt.figure(figsize=(3, 1.75), dpi=100)
    ax = plt.subplot(
        1, 1, 1, frameon=False, xlim=(-1, 1), ylim=(-0.5, 1.5), xticks=[],
yticks=[]
    )
```

```
    ax.text(
        0,
        0.0,
        r"$\frac{\pi}{4} = \sum_{k=0}^\infty\frac{(-1)^k}{2k+1}$",
        size=32,
        ha="center",
        va="bottom",
    )
    plt.show()
plot("cm")
```

$$\frac{\pi}{4} = \sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1}$$

Рисунок 5 Отображение формулы на графике

В следующем примере представлено отображение формулы в легенде графика и пример создания подписи для каждой кривой.

```
import numpy as np
from scipy.special import jn, jn_zeros
import matplotlib.pyplot as plt

plt.rc("xtick", labelsize="small")
plt.rc("ytick", labelsize="small")

fig = plt.figure(figsize=(6, 3), dpi=100)
ax = plt.subplot(xlim=[0, 20], ylim=[-0.5, 1])

X = np.linspace(0, 20, 1000, endpoint=True)
n = 6
for i in range(n):
    Ji = jn(i, X)
    linewidth = 1.5 if i == 0 else 1
    linestyle = "-" if i == 0 else "-"
    color = "C1" if i == 0 else "%.2f" % (i / n)
    label = r"$J_%d$" % i

    ax.plot(X, Ji, color="white", clip_on=False, zorder=10 - i,linewidth=2.5)

    ax.plot(
        X,
        Ji,
        color=color,
        clip_on=False,
        zorder=10 - i,
        linewidth=linewidth,
        linestyle=linestyle,
        label=label,
    )

    k = np.argmax(Ji)
    ax.text(
        X[k],
        Ji[k] + 0.05,
        label,
        color=color,
        usetex=True,
        ha="center",
```

```
        va="bottom",
        size="small",
    )
    #     k = np.argmin(Ji)
    #     ax.text(X[k], Ji[k]-0.05, label, color=color,
    #             ha="center", va="top", size="small")
    Zx = [x for x in jn_zeros(i, 6) if x < 20]
    Zy = np.zeros(len(Zx))
    ax.scatter(Zx, Zy, s=15, zorder=20, edgecolor=color, facecolor="white",
linewidth=1)

    if i == 0:
        ax.annotate(
            "Root",
            (Zx[0], Zy[0]),
            size="small",
            xytext=(-30, -30),
            textcoords="offset points",
            arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=-
0.3"),
        )

    Zy = -0.6 * np.ones(len(Zx))
    ax.scatter(
        Zx,
        Zy,
        s=30,
        zorder=20,
        clip_on=False,
        marker="|",
        facecolor="black",
        linewidth=0.5,
    )

ax.set_title(
    "Bessel functions",
    x=1,
    weight="light",
    transform=ax.transAxes,
    ha="right",
)
ax.text(
    1,
    0.98,
    r"$J_n(x) = \frac{1}{2\pi} \int_{-\pi}^\pi e^{i(x \sin \tau -n \tau)}
\,d\tau$",
    va="top",
    transform=ax.transAxes,
    size=12,
    ha="right",
    usetex=True,
)

ax.set_yticks([-0.5, 0, 0.5, 1])
ax.set_xticks([0, 10, 20])
ax.axhline(0, color="0.5", linewidth=0.5)
ax.spines["right"].set_visible(False)
ax.spines["top"].set_visible(False)
ax.spines["left"].set_position(("data", -1))
ax.spines["bottom"].set_position(("data", -0.6))

plt.tight_layout()
plt.show()
```

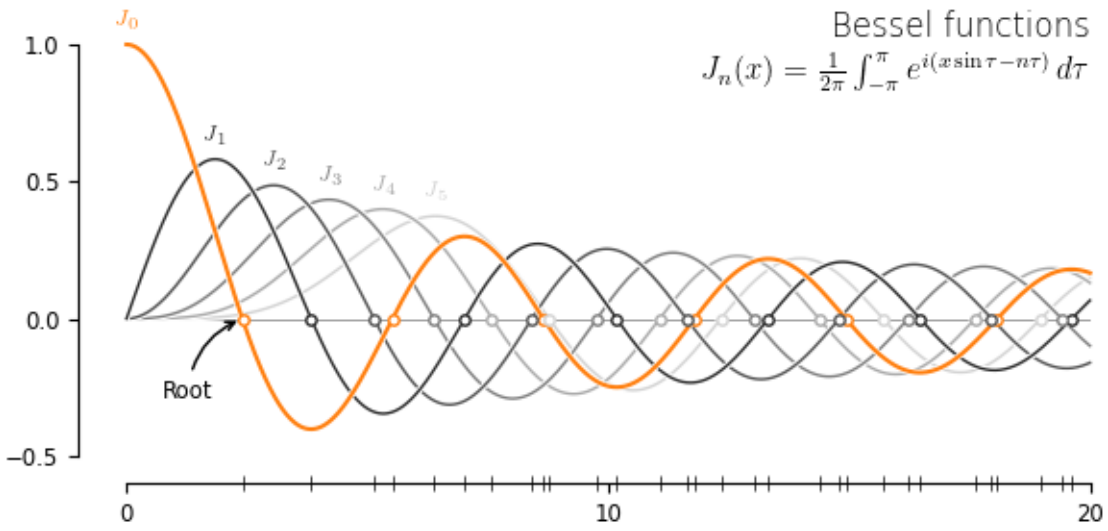


Рисунок 6 Пример отображения формул на графике и подписи отдельных линий

## Легенда

В matplotlib существует большое количество настроек отображения легенды графика. В примерах ниже представлены настройки для различных типов отображения подписей кривых на графиках.

```
import numpy as np
import matplotlib.pyplot as plt


def plot(ax):
    ax.set_xlim([-np.pi, np.pi])
    ax.set_xticks([-np.pi, -np.pi / 2, 0, np.pi / 2, np.pi])
    ax.set_xticklabels(["-π", "-π/2", "0", "+π/2", "+π"])
    ax.set_ylim([-1, 1])
    ax.set_yticks([-1, 0, 1])
    ax.set_yticklabels(["-1", "0", "+1"])

    ax.spines["right"].set_visible(False)
    ax.spines["top"].set_visible(False)
    ax.spines["left"].set_position(("data", -3.25))
    ax.spines["bottom"].set_position(("data", -1.25))

    (plot1,) = ax.plot(X, C, label="cosine", clip_on=False)
    (plot2,) = ax.plot(X, S, label="sine", clip_on=False)

    return plot1, plot2

n = 4
fig = plt.figure(figsize=(6, n * 1.9))

X = np.linspace(-np.pi, np.pi, 400, endpoint=True)
C, S = np.cos(X), np.sin(X)

ax = plt.subplot(n, 1, 1)
plot1, plot2 = plot(ax)
ax.text(
    X[-1],
    C[-1],
    " – " + plot1.get_label(),
    size="small",
```

```
    color=plot1.get_color(),
    ha="left",
    va="center",
)
ax.text(
    X[-1],
    S[-1],
    " — " + plot2.get_label(),
    size="small",
    color=plot2.get_color(),
    ha="left",
    va="center",
)

ax = plt.subplot(n, 1, 2)
plot1, plot2 = plot(ax)
ax.text(
    X[100],
    C[100],
    " " + plot1.get_label(),
    family="Roboto Condensed",
    size="small",
    bbox=dict(facecolor="white", edgecolor="None", alpha=0.85),
    color=plot1.get_color(),
    ha="center",
    va="center",
    rotation=42.5,
)
ax.text(
    X[200],
    S[200],
    " " + plot2.get_label(),
    rotation=42.5,
    family="Roboto Condensed",
    size="small",
    bbox=dict(facecolor="white", edgecolor="None", alpha=0.85),
    color=plot2.get_color(),
    ha="center",
    va="center",
)

ax = plt.subplot(n, 1, 3)
plot1, plot2 = plot(ax)

ax.annotate(
    "cosine",
    (X[100], C[100]),
    size="small",
    color=plot1.get_color(),
    xytext=(-50, +10),
    textcoords="offset points",
    arrowprops=dict(
        arrowstyle="->", color=plot1.get_color(), connectionstyle="arc3,rad=-
0.3"
    ),
)
ax.annotate(
    "sine",
    (X[200], S[200]),
    size="small",
    color=plot2.get_color(),
    xytext=(-50, +10),
    textcoords="offset points",
```

```
    arrowprops=dict(
        arrowstyle="->", color=plot2.get_color(), connectionstyle="arc3,rad=-
0.3"
    ),
)

ax = plt.subplot(n, 1, 4)
plot1, plot2 = plot(ax)
index = 10
ax.scatter(
    [X[index]],
    [C[index]],
    s=100,
    marker="o",
    zorder=10,
    clip_on=False,
    linewidth=1,
    edgecolor=plot1.get_color(),
    facecolor="white",
)
ax.text(
    X[index],
    1.01 * C[index],
    "A",
    zorder=20,
    color=plot1.get_color(),
    ha="center",
    va="center",
    size="x-small",
    clip_on=False,
)

ax.scatter(
    [X[index]],
    [S[index]],
    s=100,
    marker="o",
    zorder=10,
    clip_on=False,
    linewidth=1,
    edgecolor=plot2.get_color(),
    facecolor="white",
)
ax.text(
    X[index],
    1.05 * S[index],
    "B",
    zorder=20,
    color=plot2.get_color(),
    ha="center",
    va="center",
    size="x-small",
    clip_on=False,
)


plt.tight_layout()
plt.show()
```

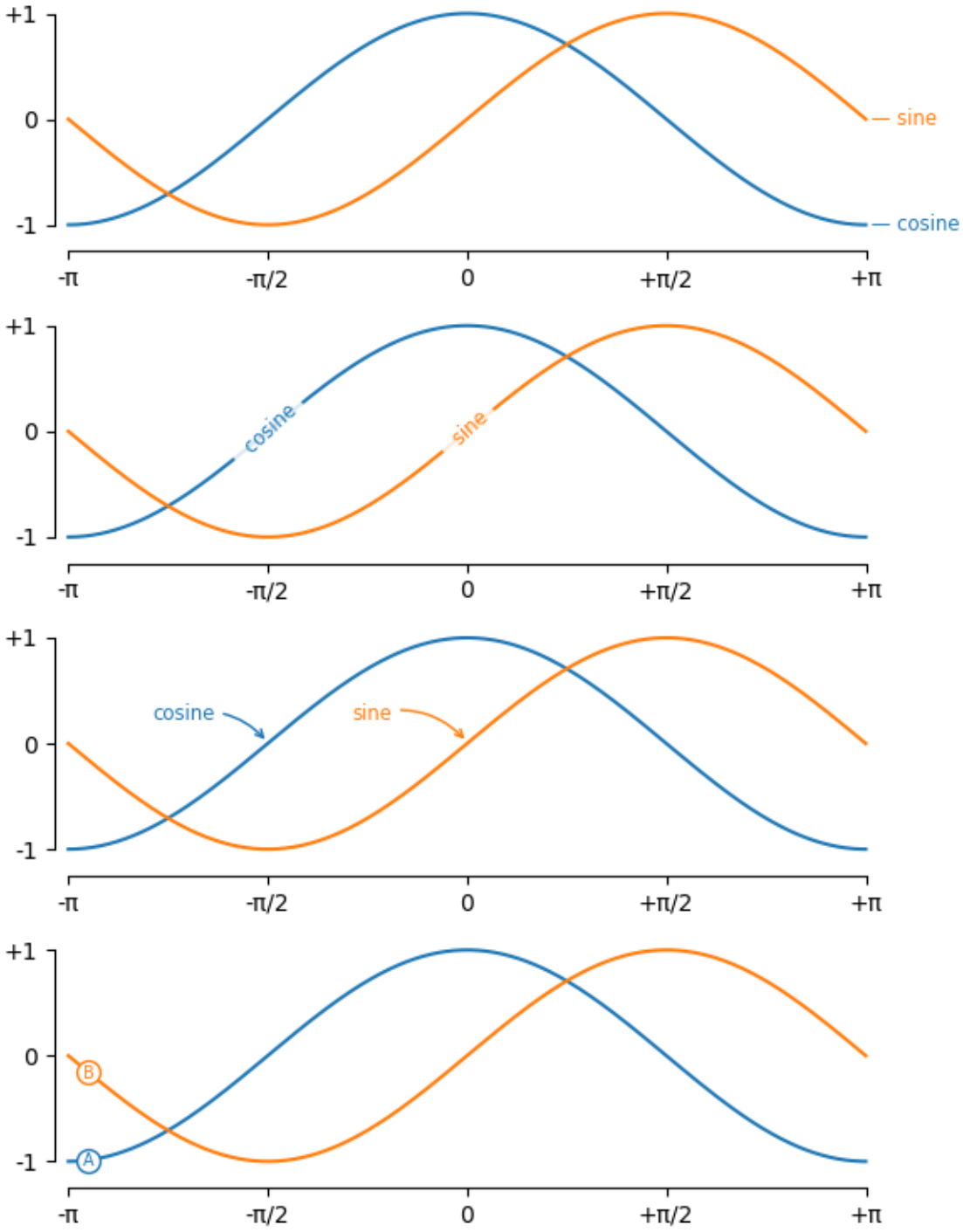


Рисунок 7 Пример отображения легенды (1)

```
import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(6, 2.5))
ax = plt.subplot(
    xlim=[-np.pi, np.pi],
    xticks=[-np.pi, -np.pi / 2, 0, np.pi / 2, np.pi],
    xticklabels=["-π", "-π/2", "0", "+π/2", "+π"],
    ylim=[-1, 1],
    yticks=[-1, 0, 1],
    yticklabels=["-1", "0", "+1"],
)
```

```
X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)

ax.plot(X, C, label="cosine", clip_on=False)
ax.plot(X, S, label="sine", clip_on=False)

ax.spines["right"].set_visible(False)
ax.spines["top"].set_visible(False)
ax.spines["left"].set_position(("data", -3.25))
ax.spines["bottom"].set_position(("data", -1.25))
ax.legend(edgecolor="None")

plt.tight_layout()
plt.show()
```

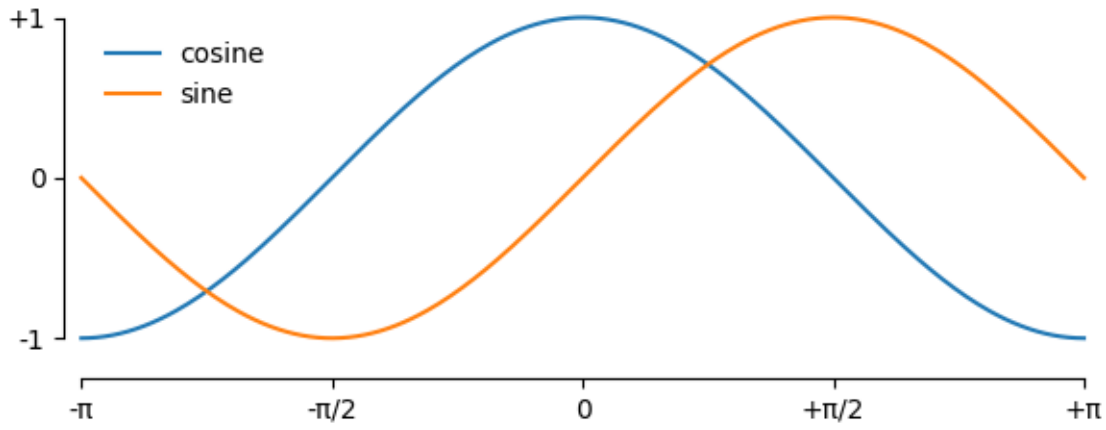


Рисунок 8 Пример отображения легенды (2)

## Отображение поля температур из постобработчика Ansys

Рассмотрим пример считывания поля температуры из файла *.csv, сохранённого из модуля Results ПК Ansys и визуализацию данного поля.

Файл export.csv располагается в приложении к лекции. В файл были импортированы координаты, объёмная доля воздуха, давление, температура, объёмная доля водяного пара, скорость и объёмная доля воды.

Импорт данных осуществлялся следующим образом:

1. File – Export – Export…
2. Откроется окно экспорта данных, содержащее две вкладки Options, Formatting.
3. Вкладка Options позволяет выбрать локацию, из которой осуществляется импорт данных, систему координат, систему единиц измерения. В списке переменных выбираются величины, которые необходимо импортировать в файл.

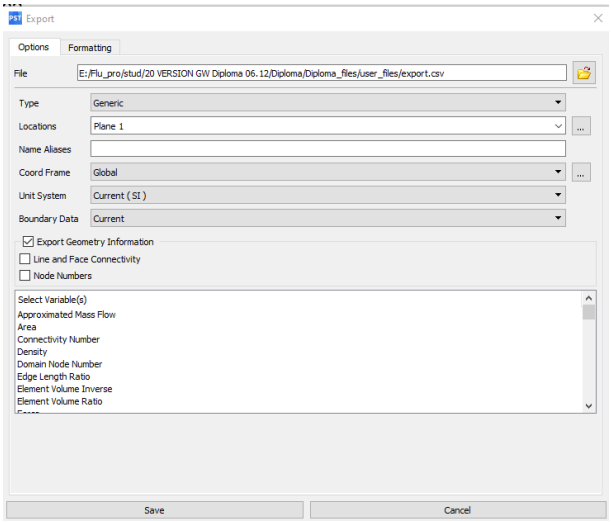


**Рисунок 9 Вкладка Options окна экспорта данных**

4. Вкладка Formatting содержит данные о формате импортируемых данных: десятичный разделитель, количество символов после десятичного разделителя.

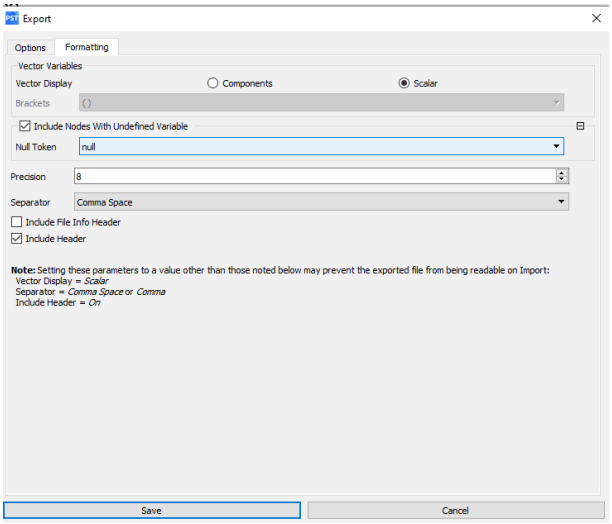


**Рисунок 10 Вкладка Formatting окна экспорта данных**

5. После нажатия кнопки Save. Выбранные данные будут сохранены в файл. В сохранённом файле данные будут храниться в колонках. В файле каждой тройке декартовых координат будет соответствовать одно значение поля. Поскольку данные в файле сохраняются в колонках, то в столбцах с координатами будут непрерывно повторяться координаты по осям, т.е. сохранённые данные не будут являться матрицей и для визуализации в matplotlib необходимо будет преобразовать их в матричном виде: в первой колонке будет располагаться одна из координат, в первом столбце вторая координата (третья координата в данном случае имеет одно и тоже значение во всех точках). На пересечениях координат будет храниться значение поля.

   Первым делом импортируем библиотеку **pandas**, **matplotlib.pyplot** и **numpy**. Первая библиотека нужна для корректного считывания файла и преобразования его в матрицу, вторая – для построения поля температуры, третья для работы с массивами.
   Далее функцией **read_csv** считываем все данные из файла export.csv в переменную reader.
   Можно считать столбцы с координатами и температурой в отдельные списки: Xx, Yy, Tt.

Функцией **pivot** преобразуем считанные данные в матрицу: Matr.
Создаём нулевые массивы X и Y.
В данном примере при экспорте данных некоторые координаты были сохранены с очень маленькой степенью (порядка 1e-11), это может привести к некорректному отображению поля температуры (будет отображаться полоса вдоль изображения). Чтобы этого избежать в Matr все координаты с абсолютным значением меньше 1e-4 были заменены на ноль.
Так же было немного подкорректировано поле температур.
**np.meshgid** – создаёт сетку для графика.
**plt.subplots** – создаёт сам график. После этой команды идут настройки графика: заполнение данными, подписи осей и т.п.
На Рисунок 11 представлено отображение считанного поля температур. В дальнейшем на данном рисунке можно будет настроить дополнительные необходимые параметры и сохранить изображение в нужном формате.

```
from pandas import *
import matplotlib.pyplot as plt
import numpy as np

reader = read_csv("export.csv", float_precision='legacy')
Xx = reader['X [ m ]'].tolist()
Yy = reader[' Y [ m ]'].tolist()
Tt = reader[' Bulk Temperature [ K ]'].tolist()

Matr = pivot(reader, index = 'X [ m ]', columns = ' Y [ m ]', values = ' Bulk
Temperature [ K ]')

X=np.zeros((len(Matr.index)))
Y=np.zeros((len(Matr.columns)))
for i in range(len(Matr.index)):
    if (Matr.index[i] > 1e-4 or Matr.index[i] < -1e-4): X[i]= Matr.index[i]
    else: X[i] =0.0
    if (Matr.columns[i] > 1e-4 or Matr.columns[i] < -1e-4): Y[i]=
Matr.columns[i]
    else: Y[i]=0.0
    #X[i] = Matr.index[i]
    #Y[i] = Matr.columns[i]
T=np.zeros((len(X), len(Y)))

for i in range(len(X)):
    for j in range(len(Y)):
        T[i, j] = Matr.values[i, j]
        if Matr.values[i,j]<273.15: T[i,j]=273.15
        if Matr.values[i,j] > 300: T[i,j] = 300

X1, Y1 = np.meshgrid(Y, X)
fig, ax = plt.subplots()
cs = ax.contourf(Y1, X1, T, cmap='rainbow', levels = 500, figsize=(6,2))
plt.xlabel('x, m', fontsize=16)
plt.ylabel('y, m', fontsize=16)
plt.xlim([0, 0.125])
plt.ylim([-0.003, 0.003])
plt.tight_layout()

cbar = fig.colorbar(cs)
plt.show()
```

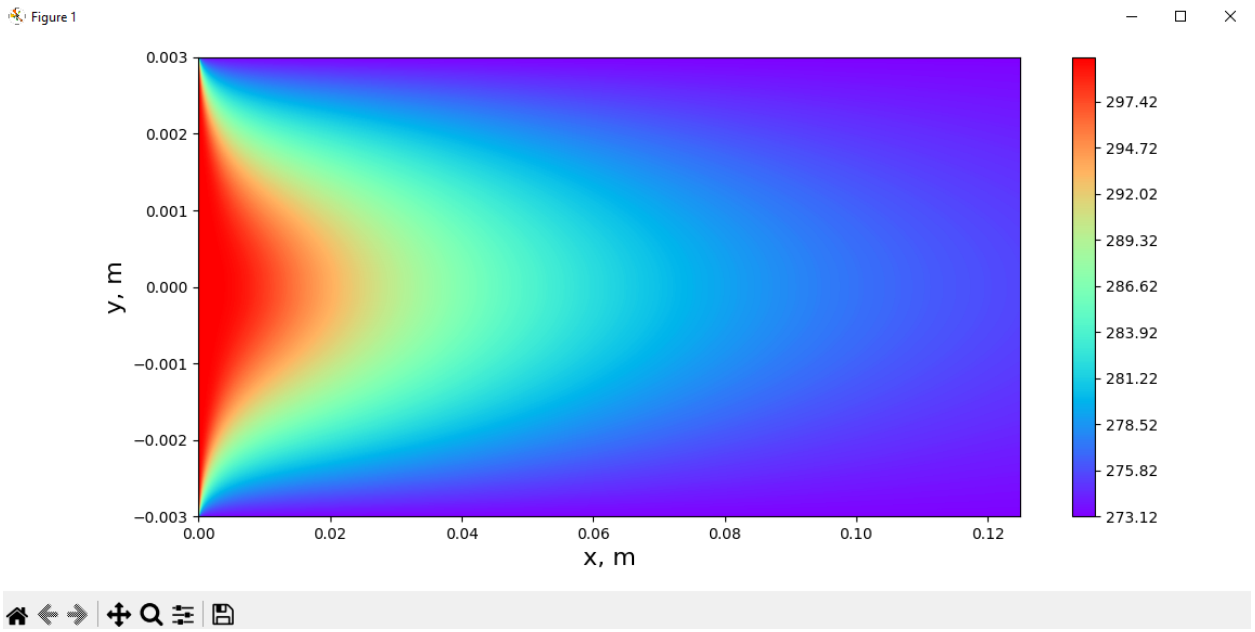


**Рисунок 11 Пример отображения поля температуры экспортированного из расчётов Ansys Fluent.**

Литература


1. Nicolas P. Rougier. Scientific Visualization: Python + Matplotlib. Nicolas P. Rougier., 2021, https://hal.inria.fr/hal-03427242
2. Хайбрахманов С. А. Основы научных расчётов на языке программирования Python : учеб. пособие / С. А. Хайбрахманов. — Челябинск: Изд-во Челяб. гос. ун-та, 2019. — 96 с.
3. https://matplotlib.org/