ЛР 2. Работа с временными рядами на Python

# 1. Цель работы

Научиться основам работы с временными рядами данными на языке Python 3.

Работу рекомендуется выполнять в бригаде из двух человек.

**ПРЕДУПРЕЖДЕНИЕ.** При выполнении работы не требуется понимать весь написанный код. Приведенные далее фрагменты кода можно брать и применять в своих проектах с минимальными доработками (изменить ссылку на исходные данные, указать номера нужных столбцов, цвета и толщины линий на графиках и т.п.).

# 2. Язык Python 3

Язык Python 3 является одним из лучших по соотношению “практическая ценность” / “простота освоения” для задач работы с данными. Существует две ветки развития языка: Python 2 и Python 3, между ними нет отличий в базовых принципах, но есть отличия в синтаксисе и деталях работы. Далее под Python будет пониматься именно Python 3. В качестве книги для углубленного изучения Python и справочника можно использовать **“Простой Python. Современный стиль программирования”** (Любанович). Есть более простая книга “**Программирование на Python для начинающих**” (МакГрат).

## 2.1. Актуальность Python для студентов

Python имеет свои преимущества и недостатки, но выбор языка программирования зависит в первую очередь от решаемой задачи. В данном курсе предполагается в основном решать задачи обработки и визуализации данных в электроэнергетике. При этом язык должен быть простым в изучении, удобным, широко распространенным, универсальным (пригодным для решения самых разных задач), независимым от операционной системы и с бесплатными инструментами разработки.

Решение в пользу Python было принято из-за широкого использования данного языка во многих образовательных курсах от ведущих мировых компаний и университетов. Python используется в подавляющем большинстве курсов по обработке данных и машинному обучению, во многих курсах по обработке сигналов, курсах по методам оптимизации и прикладной математике в целом. Python применяется такими ведущими компаниями и проектами как NASA, Google, IBM, Intel, Facebook, Instagram, Youtube, Яндекс и многими другими.

# 3. Временные ряды

Временной ряд или Time series - это последовательность данных во времени. Примеры:

* оцифрованный сигнал с микрофона, который каждую секунду выдает последовательность из 44100 значений амплитуды звука;
* записанные каждую секунду показания частоты токов и напряжений в трехфазной линии электропередачи;
* поминутные значения курса доллара к рублю на бирже;
* почасовой график электрической нагрузки предприятия;
* прогноз погоды как значения средней дневной температуры воздуха каждые сутки на месяц вперед;
* значения среднего проходного балла на специальность из года в год;
* даже видео в GIF формате - тоже временной ряд, это последовательность изображений.

Как правило временной ряд содержит значения, между которыми проходит определенный интервал времени - период (шаг дискретизации по времени). Величина, обратная периоду - частота дискретизации (sampling rate или sampling frequency). Частота дискретизации показывает число значений, записанных в ряд за одну секунду и измеряется в Гц.

Пример фрагмента временного ряда показан на Рисунке 1. В нем показано семь значений (сэмплов), каждый из которых имеет таймстамп - момент времени и пять показателей (свойств, признаков, полей, features - в разных источниках используют разные термины).

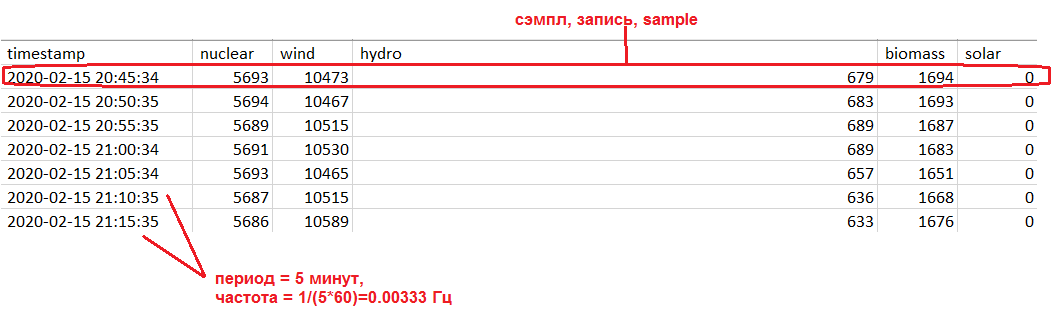


Рисунок 1. Пример временного ряда.

Работа с временными рядами используется во всех инженерных областях и науках, потому что временной ряд - основная структура данных для представления изменений во времени.

# 4. Используемые инструменты Pandas и Matplotlib

Для работы с временными рядами можно ограничится офисными приложениями, такими как MS Excel или LibreOffice Calc. Но они не всегда хорошо подходят для задач обработки данных, которые нужно решать в энергетике. Причины:

* медленная работа с большими объемами данных, аварийные завершения при нехватке оперативной памяти;
* невысокий уровень автоматизации - пользователю нужно каждый раз выполнять очень много действий вручную;
* ошибки при конвертации форматов - из-за возможностей форматирования (объединенные ячейки и т.п.) могут возникать ошибки в данных при сохранении файла в формат, отличный от исходного;
* ошибки из-за различий в версиях приложений, не всегда файл, сохраненный в Excel более новой версии, правильно откроется в Excel старой версии;
* нужное приложение может отсутствовать на компьютере, на котором требуется открыть файл.

# 5. Примеры работы с временными рядами

Вместо изложения теории по работе с инструментами в данном параграфе приведено несколько подробных примеров, код которых можно будет переиспользовать в своих работах.

## 5.1. Данные по энергосистеме, статистика и графики

Пример показывает как:

* скачать табличные данные из облака и прочитать их;
* получить статистические показатели данных;
* построить графики временных рядов;
* построить круговую диаграмму.

Весь код доступен по ссылке <https://colab.research.google.com/drive/1TdtWKekv3Zg5qVgJremmXEogm9kRK3Aq>. Для работы создайте свою копию этого блокнота (в меню Google Colab выбрать “File”, “Save a copy in Drive”). Ниже идет тот же самый код, с небольшими дополнительными пояснениями.

Для работы рекомендуется запускать ячейку за ячейкой, щелкая на нее и нажимая Ctrl + Enter или нажимая на стрелку слева ячейки.

| # 0. У каждой ячейки ниже будет номер для соответствия пояснений в учебном пособии. |
| --- |

### 5.1.1. Подключение библиотек

Первая ячейка блокнота используется для подключения нужных для работы модулей.

| # 1. Подключение нужных для работы моделей import pandas as pd # для чтения и записи табличных файлов import numpy as np # для математической обработки данных import matplotlib.pyplot as plt # для построения графиков |
| --- |

Синтаксис простой - имя библиотеки или части библиотеки, затем после ключевого слова “as” можно указать сокращенный псевдоним для дальнейшей работы. Достоинством Google Colab является то, что все распространенные библиотеки в нем есть, ничего не нужно скачивать и ставить на свой компьютер.

### 5.1.2. Чтение данных

Так как Google Colab работает в облаке, он не может напрямую обращаться к файлам на вашем компьютере. Есть несколько способов передачи файлов.

1. Загрузить файлы в окружение блокнота. Для этого нажать на иконку папки, как показано на Рисунке 2.

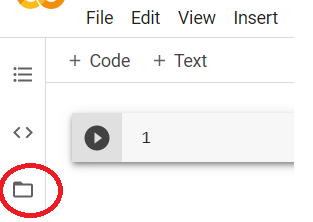


Рисунок 2. Открытие папки окружения блокнота

Затем нажать “Upload” или перетащить файлы мышкой (можете попробовать сделать этом с любым своим файлом).

Этот способ удобный для небольших файлов.

1. Более профессиональный способ работы с данными - загрузить их в облачное хранилище. При профессиональной работе с данными может оказаться, что их слишком много для хранения на локальном компьютере. И если над данными работает команда, то лучше иметь общее для команды место, чем иметь копию файлов на компьютере каждого человека.

В качестве хранилища можно использовать Google Drive, Github, Dropbox и т.д., главное чтобы выбранная система позволяла создавать ссылки и содержимое файлов.

Вторая ячейка показывает пример получения данных из удаленного источника и чтения с помощью библиотеки Pandas. Pandas - мощный инструмент для работы с табличными данными. В результате на экране должен появится фрагмент таблицы, как показано на Рисунке 3.

| # 2. Получение и чтение файла из облака # адрес - когда копируете ссылку из Google drive, она выглядит обычно вот так: # https://drive.google.com/open?id=1FoSS-guXOsGUbE0xHcKR-E9ZEaTmc9Wv # или так:  # https://drive.google.com/file/d/1FoSS-guXOsGUbE0xHcKR-E9ZEaTmc9Wv/view?usp=sharing  # чтобы обрабатывать данные в программе, нужно ссылку переделать в следующий вид: # https://drive.google.com/uc?export=download&id=... # то есть взять id из ссылки, как ее дал Google Drive и вставить в шаблон https://drive.google.com/open?id=  # А для Dropbox ссылка, которую дает система, уже почти пригодна для работы через программы, нужно только в конце заменить dl=0 на dl=1.  url = "https://drive.google.com/uc?export=download&id=1FoSS-guXOsGUbE0xHcKR-E9ZEaTmc9Wv"  # чтение данных, можно указать символ разделения столбцов (sep) и многие другие параметры,  # index\_col показывает, какой столбец нужно использовать как индекс строки в таблице input\_df = pd.read\_csv(url, sep = ';', index\_col = 1)  # вывод заголовка таблицы и первых пяти строк (сэмплов) input\_df.head() |
| --- |

В таблице на Рисунке 3 столбец timestamp - это время снятия показаний; id - просто номер строки в общей базе; frequency - частота тока и напряжения в электроэнергетической системе; demand - общее потребление электроэнергии в британской электроэнергетической системе в ГВт. Следующие столбцы показывают выработку различных источников: ТЭС на угле, АЭС, парогазовых электрогенерирующая станция, ВЭС, ГЭС, станций на биогазе, СЭС, газовых турбин и обмен электроэнергией с Францией (> 0 импорт, < 0 экспорт). Изначально фрагмент данных взят с сайта [gridwatch](https://www.gridwatch.templar.co.uk/download.php).

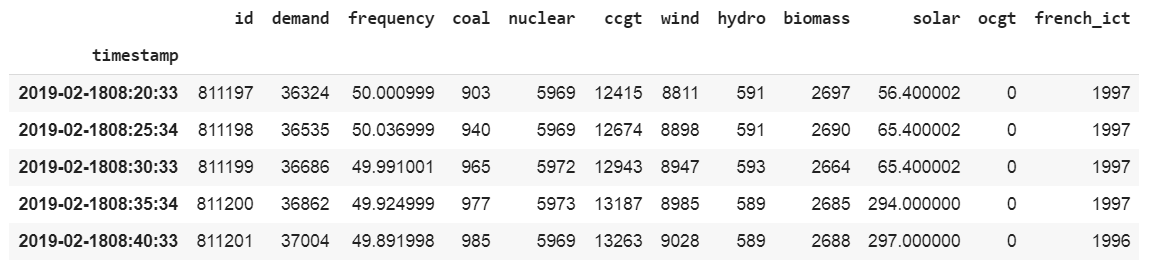


Рисунок 3. Фрагмент таблицы с показаниями энергосистемы

При чтении файлов с помощью Pandas наиболее полезными являются следующие настройки, которые передаются в функцию чтения файла:

* sep - символ, разделяющий столбцы в таблице;
* decimal - символ, который используется как десятичный разделитель (отделяющий целую часть числа от дробной), по умолчанию это точка, но нередко используют запятую;
* header - номер строки, которую нужно использовать как заголовок таблицы (имена столбцов), начиная с нуля, при этом header = None означает, что никакую строку не нужно использовать в качестве заголовка;
* index\_col - аналогично header, но для индексов (имен) строк;
* nrows - число строк, которые нужно прочитать.

### 5.1.3. Просмотр последних строк таблицы

| # 3. Просмотр последних строк таблицы input\_df.tail() |
| --- |

### 5.1.4. Статистика

| # 4. Получение базовых статистических показателей input\_df.describe() |
| --- |

### 5.1.5. Графики на разных плотах

Плотом называется поле для построения графиков, диаграмм и прочих визуальных объектов. Код ниже служит для отображения изменения во времени всех показателей из таблицы.

| # 5. Можно сразу построить графики axes = input\_df.plot(marker='.', figsize=(20, 20), subplots = True) |
| --- |

### 

### 5.1.6. Графики на одном плоте

| # 6. Можно построить все вместе, заодно удалив индекс строк, так как его график не несет информации. # input\_df[input\_df.columns[ 1 : ]] - означает что нужно взять из исходной таблицы только столбцы с 1-го и до последнего.  # так как в Python индексация с нуля - то нулевой столбец с индексами не будет использован axes = input\_df[input\_df.columns[ 1 : ]].plot(marker=None, figsize=(20, 20), subplots = False) |
| --- |

### 5.1.7. Переход к матрицам, NumPy

NumPy - наиболее распространенная Python библиотека для работы с матрицами и многомерными массивами, позволяющая быстро и удобно делать обработку данных.

| # 7. Подробнее работа с табличными данными pandas будет в следующей работе. Сейчас нужно перевести все в матрицу # Здесь в дело вступает другая очень популярная и мощная python-библиотека - NumPy  input\_matrix = input\_df.values # перевод таблицы в матрицу  print('размерность матрицы', input\_matrix.shape) print()  # можно посчитать среднее, минимальное, максимальное и прочие показатели  # цикл от 1 до числа столбцов for i in range(1, len(input\_df.columns)):  # функция print выводит строку на экран  # функция формат применяется к шаблону строки для подстановки в него нужных данных в нужном формате  # запись input\_matrix[: , i] означает что нужно взять все строки и i-й столбец из матрицы  print("{: <12} mean = {:.2f}".format(input\_df.columns[i], input\_matrix[: , i].mean())) |
| --- |

В этой ячейке показывается, как можно выводить результаты с помощью функции print. Эта функция выводит строку, при этом автоматически преобразует в строковый формат различные данные. Так, вызов print('размерность матрицы', input\_matrix.shape) выведет и строку 'размерность матрицы' и следом массив, содержащий размерность матрицы по каждой оси.

Можно делать и сложный форматированный вывод. Для этого использовать функцию format. В Таблице 1 ниже даны пояснения про последнюю строку кода.

Таблица 1. Разбор кода примера

| **Фрагмент кода** | **Пояснение** |
| --- | --- |
| print(...) | вывод строки на экран |
| "...".format(...) | применение форматирования к строке, результат - новая строка |
| {:...} | место для подстановки значения |
| {: <12} | означает что в строку вместо этой конструкции нужно подставить значение, и если число символов будет меньше 12, то дополнить пробелами (т.к. между : и <12 стоит пробел) |
| {:.2f} | означает что в строку вместо этой конструкции нужно подставить значение и вывести только две цифры после запятой в формате числа с плавающей точкой |
| format(..., ...) | функция формат найдет в строке, к которой она применена, места для подстановки, в данном случае {: <12} и {:.2f} и поместит в них свои аргументы, которые здесь заменены на многоточия |
| input\_df.columns[i] | input\_df.columns содержит имена столбцов, так что input\_df.columns[i] - это имя i-го столбца |
| input\_matrix[: , i] | input\_matrix - матрица значений, она двумерная, поэтому из нее можно взять фрагмент, задав два индекса через запятую;  первый индекс отвечает за выбор строк, так как нужны все строки, можно указать просто двоеточие, если нужно были бы строки от 10 до 100, нужно было бы указать 10 : 101;  второй индекс отвечает за выбор столбца, здесь берется i-й столбец |
| input\_matrix[: , i].mean() | к выбранному фрагменту матрицы применяется функция для вычисления среднего значения |
| итого | в шаблон строки {: <12} mean = {:.2f} функция формат подставить имя столбца и среднее значения столбца. Полученная строка будет выведена на экран, например  demand mean = 30256.43 |

### 5.1.8. Построение более наглядных графиков

Отображать сразу все графики разом - не очень хорошая идея, обычно при анализе выбирается несколько. В примере ниже отображается изменение во времени выработка электроэнергии от сжигания угля, от ветроэнергетических станций и от АЭС.

| # 8. Теперь можно легко выбрать, какие данные отобразить на графике  # вначале создать объект для рисования на нем fig, ax = plt.subplots(1, 1, figsize = (15, 8))  # указать какие данные вывести, например, от 400-й до 500 строки # таймстампы можно для наглядности сгенерировать от нуля с шагом 15 минут timestamps = range(0, input\_matrix.shape[0], 5)  # зададим диапазон begin = 400 end = 500   # легко указывать цвета, стиль линий, размеры и тип маркера ax.plot(timestamps[begin : end], input\_matrix[begin : end, 3], 'red', marker = '.', linewidth = 3, markersize = 9) ax.plot(timestamps[begin : end], input\_matrix[begin : end, 4], 'blue', linestyle = '--') ax.plot(timestamps[begin : end], input\_matrix[begin : end, 6], 'green', marker = 'v')  ax.set\_xlabel('Время, мин')  ax.set\_ylabel('Мощность, ГВт') ax.legend(['уголь', 'атом', 'ветер'])  plt.grid(True) plt.show()  # очень много примеров построения графиков и диаграмм приведено на сайте Matplotlib: https://matplotlib.org/3.1.1/gallery/ |
| --- |

Пример вывода результатов на Рисунке 4. Обратите внимание - в Python 3 нет проблем с выводом кириллических символов, так что можно писать на русском. Более того, легко добавлять специальные символы, как показано на Рисунке 5.

На графиках видно, что выработка АЭС очень стабильна и почти не меняется в рассмотренном временном интервале. А генерация угольных ТЭС и ВЭС меняется сильно, причем на данном участке чем больше выработка от одного источника, тем меньше от другого. То есть во время более высокой скорости ветра можно сжигать меньше ископаемого топлива.

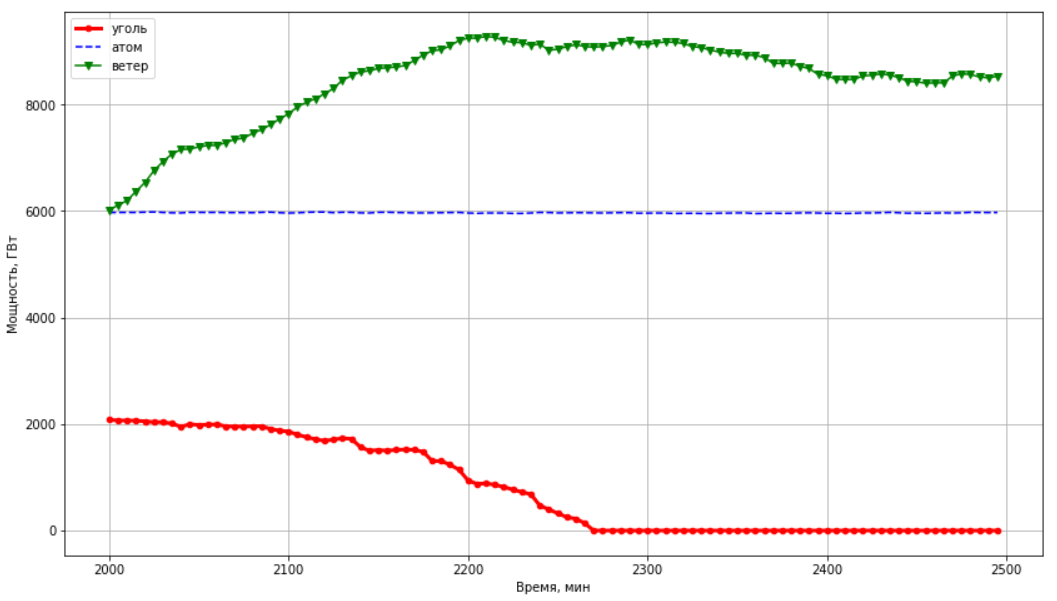


Рисунок 4. Графики генерации

### 5.1.9. Графики - другой пример

| # 9. То же самое, но лишь каждая пятая точка и все графики на отдельных плотах  # вначале создать объект для рисования на нем fig, ax = plt.subplots(3, 1, figsize = (15, 12))  # указать какие данные вывести, например, от 100-й до 500 ст # таймстампы можно для наглядности сгенерировать от нуля с шагом 15 минут timestamps = range(0, input\_matrix.shape[0] \* 15, 15)  # зададим диапазон begin = 400 end = 500  step = 5  # легко указывать цвета, стиль линий, размеры и тип маркера ax[0].plot(timestamps[begin : end : step], input\_matrix[begin : end : step, 3], 'red', marker = '.', linewidth = 3, markersize = 9) ax[1].plot(timestamps[begin : end : step], input\_matrix[begin : end : step, 4], 'blue', linestyle = '--', marker = 'o') ax[2].plot(timestamps[begin : end : step], input\_matrix[begin : end : step, 6], 'green', marker = 'v')  legend = ['уголь ◼', 'атом ☢', 'ветер ↠']  # примеры "синтаксического сахара" Python (спросите пояснение у преподавателя) for \_ in zip(ax, legend):  \_[0].set\_ylabel('Мощность, ГВт')  \_[0].legend([\_[1]])  \_[0].grid(True)  # нет необходимости подписывать ось ординат у всех плотов, достаточно у последнего (последний - это -1)  ax[-1].set\_xlabel('Время, мин')  plt.show() |
| --- |

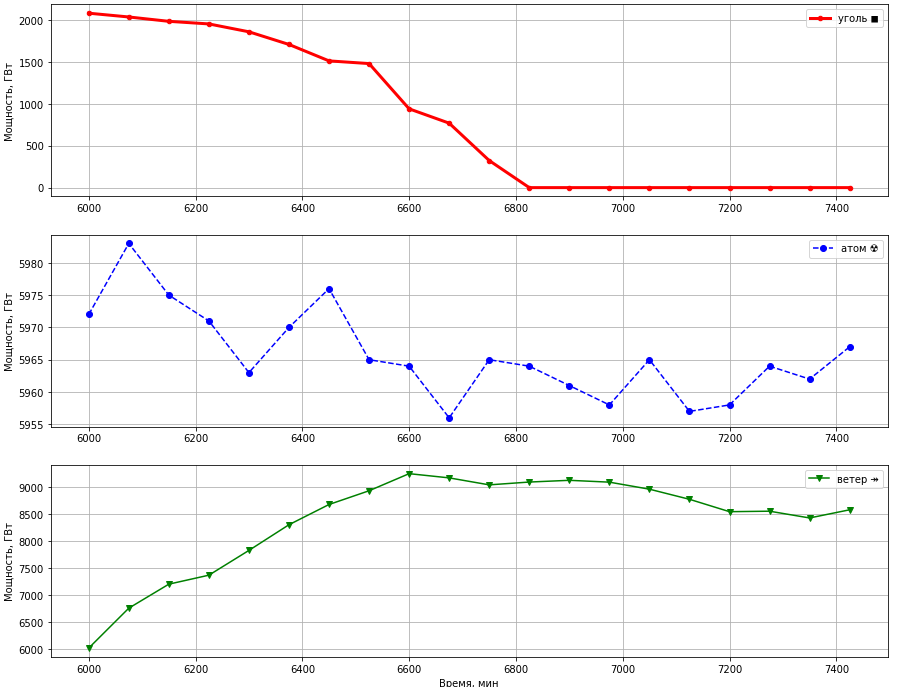


Рисунок 5. Графики на отдельных плотах

### 5.1.10. Круговая диаграмма

На графиках временного ряда хорошо видно изменение выработки во времени, но сложно сделать вывод о том, какой же источник электроэнергии в итоге дает какой вклад в общую генерацию. Для визуализации такой информации можно построить диаграмму, которая называет круговая или PieChart (pie - пирог). То есть показывает, какая доля “пирога” на кого приходится.

| # 10. Другой пример графика - PieChart  # получить сумму значений по столбцам, отвечающим за разные виды генерации (первые три столбца - это индекс, потребление, частота, а последний - обмен с другой системой). generation\_sum = (input\_matrix[: , 3 : -1]).sum(axis = 0)  # Pie chart, where the slices will be ordered and plotted counter-clockwise: labels = input\_df.columns[3 : -1]  fig1, ax1 = plt.subplots(figsize = (12, 12)) ax1.pie(generation\_sum, labels=labels, autopct='%1.1f%%') ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.  plt.show() |
| --- |

Результат представлен на Рисунке 6.

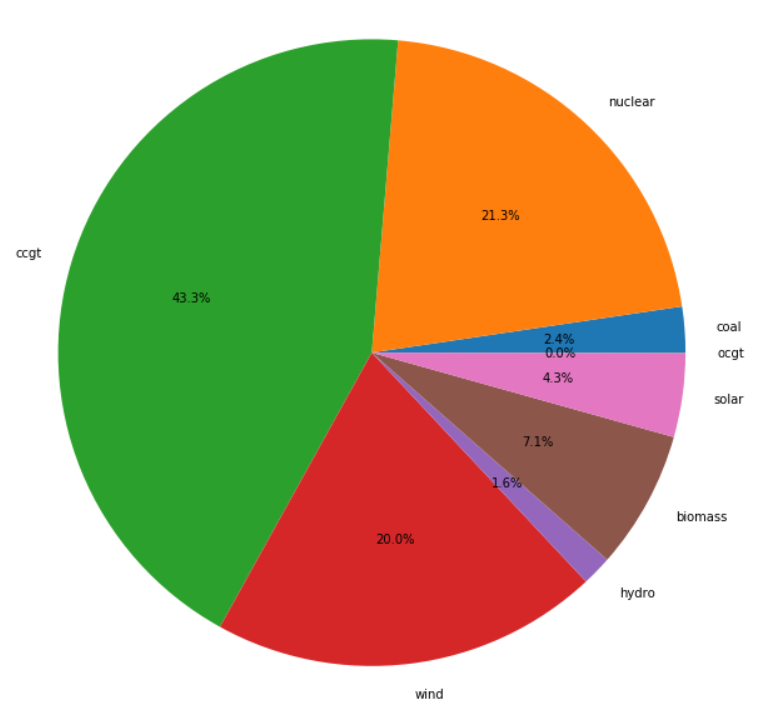


Рисунок 6. Круговая диаграмма

## 5.2. Ветроэнергетическая установка, прогнозирование

Пример показывает как:

* открыть файл, созданный в Excel;
* построить график функции;
* применять функции для обработки данных временного ряда;
* сохранить результаты как изображения, Excel таблицы и текстовые файлы.

Весь код доступен по ссылке <https://colab.research.google.com/drive/1rBM4fd8xbYCDf13xJRZhvz5Aac6ovE0t>. Для работы создайте свою копию этого блокнота (в меню Google Colab выбрать “File”, “Save a copy in Drive”). Ниже идет тот же самый код, с небольшими дополнительными пояснениями.

### 5.2.1. Подключение библиотек

| # 1. Подключение нужных для работы моделей import pandas as pd # для чтения и записи табличных файлов import numpy as np # для математической обработки данных import matplotlib.pyplot as plt # для построения графиков |
| --- |

### 5.2.2. Получение и чтение удаленного Excel файла

| # 2. Получение и чтение Excel файла из облака  url = "https://drive.google.com/uc?export=download&id=1AqQZKW8l7Ks5vfe5q9VjEuJMb5JHVcj\_"  # чтение данных, разделитель столбцов можно не указывать, Pandas при чтении XLS файла сам его определит;  # имен у столбцов в исходном файле нет, поэтому нужно указать header = None. input\_df = pd.read\_excel(url, header = None)   # вывод заголовка таблицы и первых пяти строк (сэмплов) input\_df.head() |
| --- |

### 5.2.3. Выделение нужного столбца, построение графика

Построим график - как изменялась скорость ветра час от часа (Рисунок 7).

| # 3. Берем только временной ряд скорости ветра в м/с # iloc - выделяет из таблицы указанный фрагмент, в данном случае все строки и только первый столбец wind\_speed = input\_df.iloc[: , 2].values  print(wind\_speed.shape)  fig, ax = plt.subplots(figsize = (20, 10))  ax.plot(wind\_speed, linewidth = 3, markersize = 5, marker='o', color = 'green') ax.grid(True) ax.set\_xlabel('Время, ч')  ax.set\_ylabel('Скорость ветра, м/с')  plt.show() |
| --- |

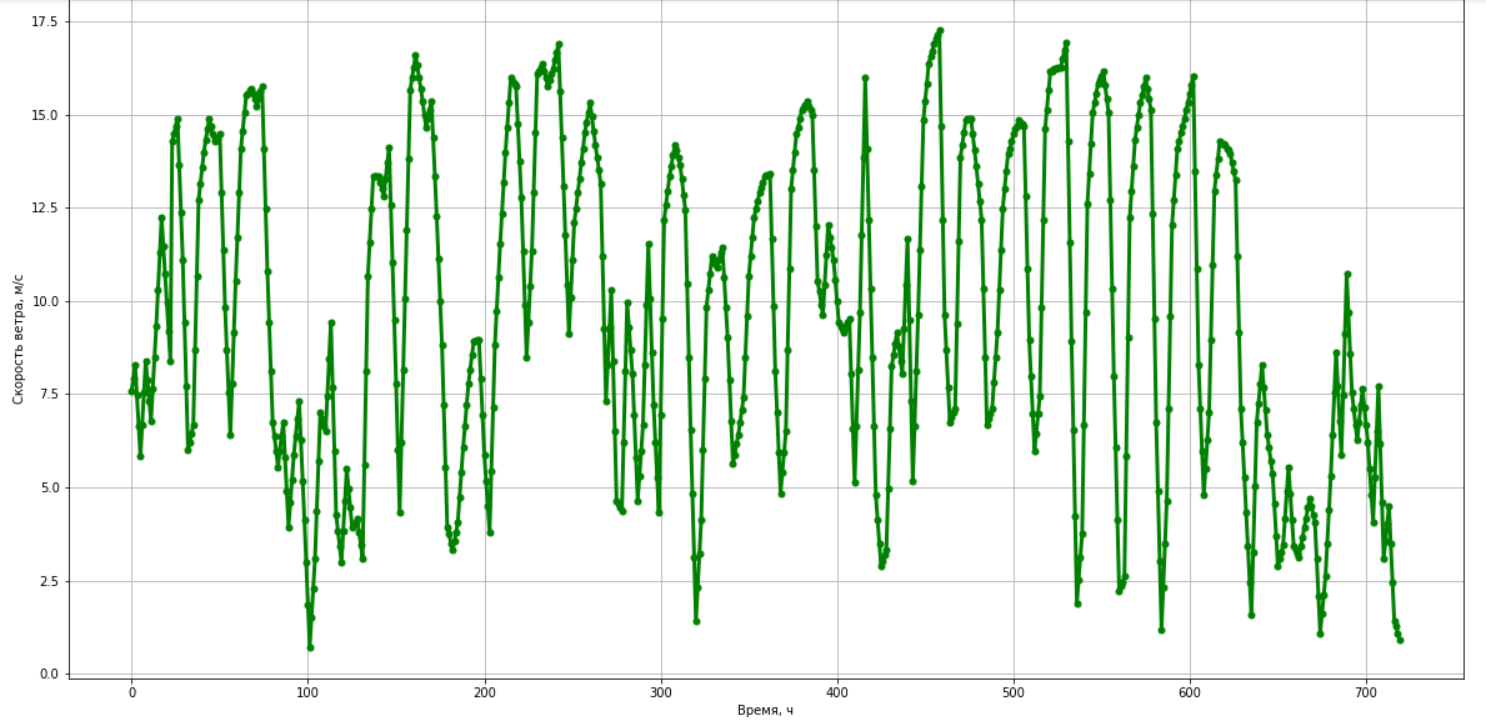


Рисунок 7. График скорости ветра

### 

### 5.2.4. Пример Matplotlib для построения графика аналитически заданной функции

| # 4. Matplotlib можно использовать для построения графиков аналитически заданных функций x = np.arange(-10, 10, 0.1)  fig, ax = plt.subplots() ax.plot(x, x\*x) ax.grid(True) plt.show() |
| --- |

### 5.2.5. Перевод скорости ветра в выработку ветроэнергетической установки

Для энергетики важнее выработка от ВЭС, чем скорость ветра. Поэтому следует по скорости ветра и зависимости выработки от скорости ветра построить график выработки ВЭС.

В Python, как и в других языках программирования, можно создавать собственные функции. Функция начинается с ключевого слова def, а весь код функции нужно писать, сделав отступ в 2 или 4 пробела слева. Результат, который должен получиться, представлен на Рисунке 8.

| # 5. Зададим зависимость среднечасовой мощности, вырабатываемой ВЭУ, от средней скорости ветра def wind\_speed\_to\_power(speed):  min\_speed = 5  max\_speed = 14  critical\_speed = 17    # выработка равна нулю, если скорость ветра выше критической или ниже минимальной  # поэтому обнуляем такую скорость  speed\_tmp = np.where(np.logical\_or(speed > critical\_speed, speed < min\_speed), 0, speed)   # если скорость выше максимальной, то выработка не будет расти, добавляем ограничение сверху   speed\_tmp = np.clip(speed\_tmp, a\_min = None, a\_max = max\_speed)   # на рабочем участке от min\_speed до max\_speed мощность пропорциональна кубу скорости  power = 0.0015 \* (speed\_tmp) \*\* 3  return power  # отдельно создается график зависимости выработки от скорости ветра  x = np.arange(0, 30, 0.1) fig, ax = plt.subplots() ax.plot(x, wind\_speed\_to\_power(x)) ax.grid(True) ax.set\_xlabel('Скорость ветра, м/с')  ax.set\_ylabel('Мощность ВЭУ, МВт') plt.show()  fig, ax = plt.subplots(figsize = (20, 10)) ax.plot(wind\_speed\_to\_power(wind\_speed)) ax.plot(wind\_speed \* 0.1)  ax.grid(True) ax.set\_xlabel('Время, ч')  ax.legend(['Мощность, МВт', 'Скорость, 0.1 м/с']) plt.show() |
| --- |

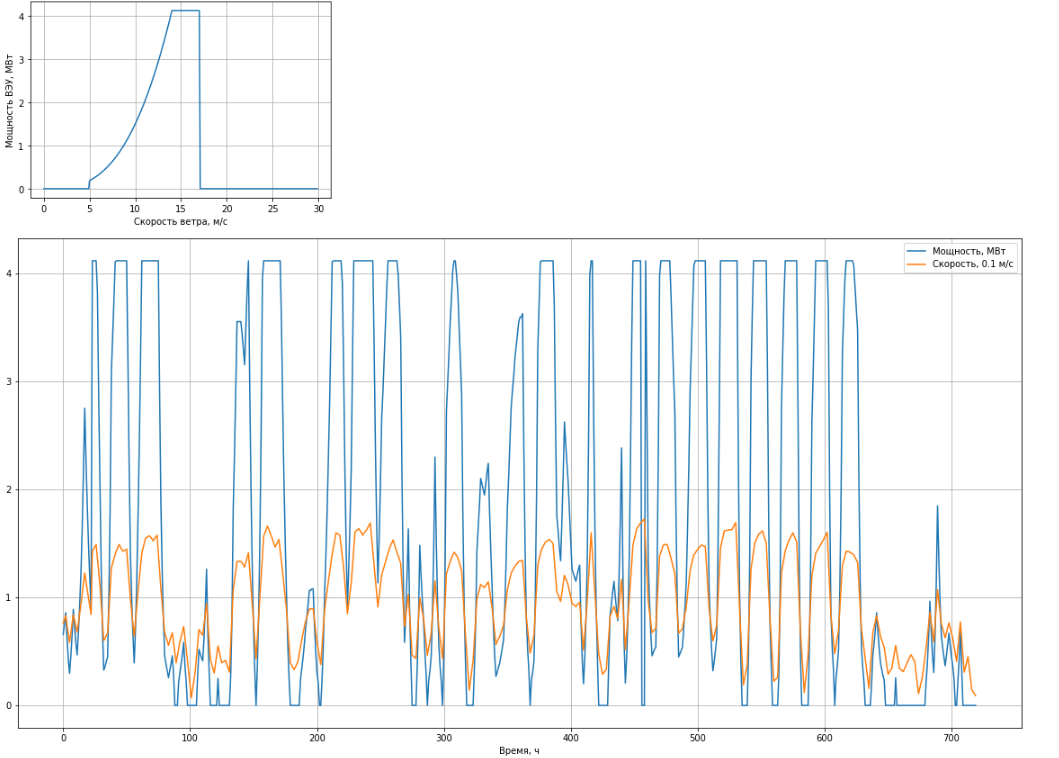


Рисунок 8. Зависимость вырабатываемой мощности от скорости ветра и временной график генерации

В коде используется функция np.arange. Она создает последовательность чисел. Если вызвать ее, передав один аргумент - будет создана последовательность от нуля до аргумента (не включая его). Если два - то от первого аргумента до второго. Если три - третий аргумент будет использован как шаг. Примеры в Таблице 2.

Таблица 2. Функция arange

| **Код** | **Результат** |
| --- | --- |
| np.arange(8) | [0, 1, 2, 3, 4, 5, 6, 7] |
| np.arange(3, 6) | [3, 4, 5] |
| np.arange(2, 9, 2) | [2, 4, 6, 8] |
| np.arange(0.1, 0.2, 0.02) | [0.1 , 0.12, 0.14, 0.16, 0.18] |
| np.arange(11, 7, -1.3) | [11. , 9.7, 8.4, 7.1] |

### 5.2.6. Обработка данных точка за точкой, сохранение результатов

| # 6. Пример обработки временного ряда точка за точкой (sample by sample) # Рассмотренные выше функции применялись ко всем точкам ряда. Но бывает что нужна более сложная обработка. # Рассмотрим пример очень примитивного прогнозирования скорости ветра на час вперед по следующей рекуррентной формуле # forecast[i] = speed\_[i - 1] \* k + speed\_[i - 2] \* (1 - k)  k = 0.9  # создадим массив, заполненный нулями для записи прогноза, его длина равна длине wind\_speed forecast = np.zeros(len(wind\_speed))   # начальные два значения копируем, так как для прогноза на i-й час нужно значения скорости за два прошлых часа  forecast[0:2] = wind\_speed[0:2]  # можно записать этот цикл более изящно, но сдесь записан для примера работы с индексами временного ряда for i in range(2, len(wind\_speed)):  # делаем прогноз на i-й час, так как он еще как будто не наступил, то можем использовать наблюдения только за прошлые часы (i - 1, i - 2, ...)   forecast[i] = wind\_speed[i - 1] \* k + wind\_speed[i - 2] \* (1. - k)  # вывод графиков fig, ax = plt.subplots(figsize = (20, 10)) ax.plot(wind\_speed, 'blue') ax.plot(forecast, 'green') ax.plot(np.abs(forecast - wind\_speed), 'red')  ax.grid(True) ax.set\_xlabel('Время, ч')  ax.legend(['Скорость ветра, м/с', 'Прогноз, м/с', 'Модуль ошибки, м/с'])  # можно даже записать формулу, если разбораться в синтаксисе MathText https://matplotlib.org/3.1.1/tutorials/text/mathtext.html plt.text(0, 17, r'${forecast\_i} = {speed}\_{i-1} \* k + {speed}\_{i-2} \* (1 - k)$', fontsize = 16) plt.savefig('image') plt.show()  # полученный прогноз можно сохранить в текстовый csv файл или в Excel формат data\_frame = pd.DataFrame(forecast) data\_frame.to\_excel('forecast.xlsx', index = False, header = ['wind\_speed\_forecast\_m/s']) data\_frame.to\_csv('forecast.csv', index = False, header = ['wind\_speed\_forecast\_m/s']) |
| --- |

Результат представлен на Рисунке 9. На нем показано изменение скорости ветра на самом деле, изменение по прогнозу и ошибка прогноза для каждого часа.

Обратите внимание, в последних строках делается сохранение результатов. Чтобы получить сохраненные файлы, нужно выдвинуть левую боковую панель (Рисунок 10). Вы увидите список файлов. Клик на файле правой кнопкой мыши открывает меню, через которое файл можно скачать (Рисунок 11).

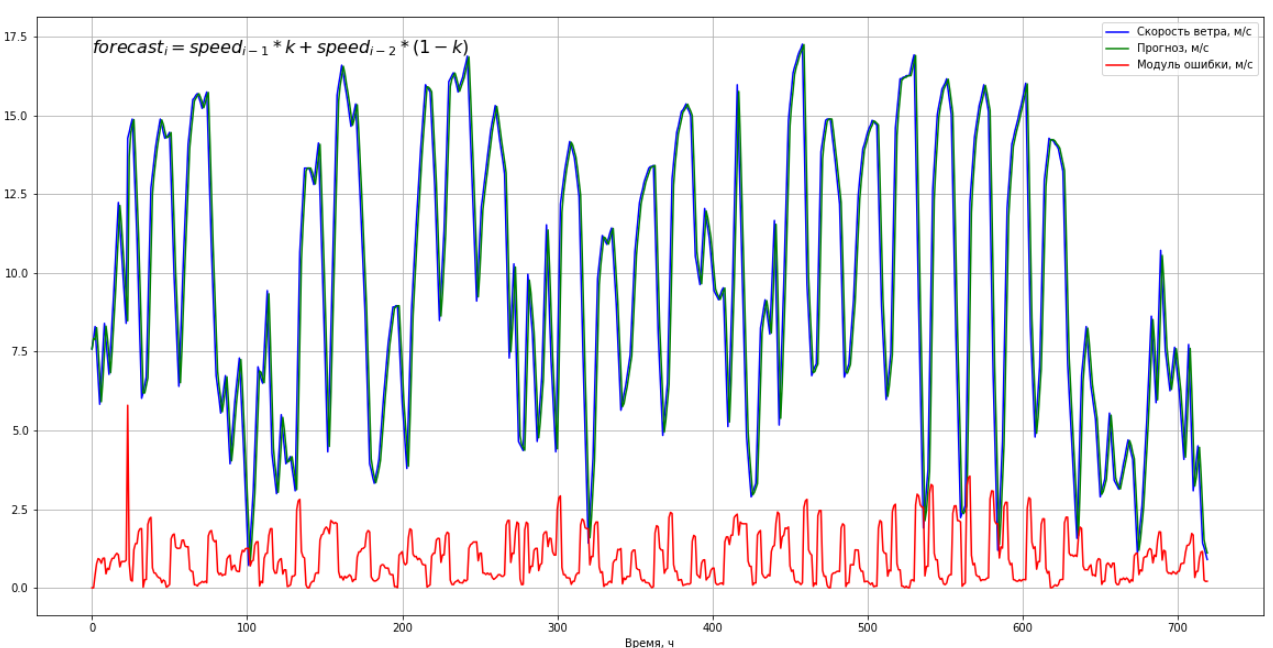


Рисунок 9. Реальная скорость ветра, прогнозная и ошибка прогноза

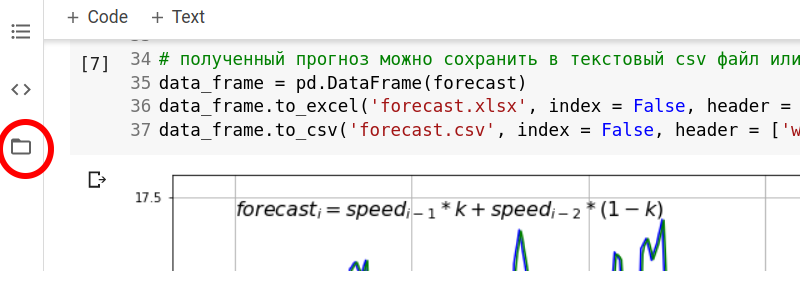


Рисунок 10. Открытие панели с файлами

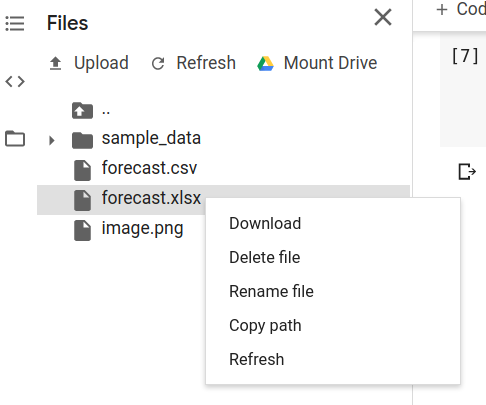


Рисунок 11. Скачивание файла из Google Colab

### 5.2.7. Пример интерактивного графика

Например, на графиках Рисунка 9 кажется, что прогноз очень близок к реальному значению, почти идеально точен. Но это только кажется из-за масштаба графика, при котором не видны мелкие детали. Чтобы иметь возможность быстро менять масштаб для рассматривания деталей на графике Matplotlib уже не очень подходит, нужно использовать другие библиотеки. Хотя в общем случае Matplotlib все-таки остается наиболее удобным и простым инструментом.

В примере ниже используется библиотека Bokeh. Полученный график является интерактивным, то есть он реагирует на действия пользователя. Попробуйте поменять масштаб, подвигать график мышкой. Так получится понять, что прогноз (зеленый) дает небольшую ошибку, когда скорость ветра медленно меняется от часа к часу. Но когда скорость меняется быстро, прогноз слишком сильно учитывает прошлое значение и “промахивается” (Рисунок 12).

| # 7. Для визуализации данных существует множество Python библиотек, обзор есть в https://colab.research.google.com/notebooks/charts.ipynb#scrollTo=QSMmdrrVLZ-N # Рассмотрим одну из них - Bokeh https://docs.bokeh.org/en/latest/docs/user\_guide/plotting.html  from bokeh.plotting import figure, output\_file, show from bokeh.io import output\_notebook  output\_notebook()  p = figure(plot\_width = 800, plot\_height = 400)  p.line(np.arange(len(wind\_speed)), wind\_speed, line\_width = 2) p.line(np.arange(len(wind\_speed)), forecast, line\_width = 2, color = 'green')  show(p) |
| --- |

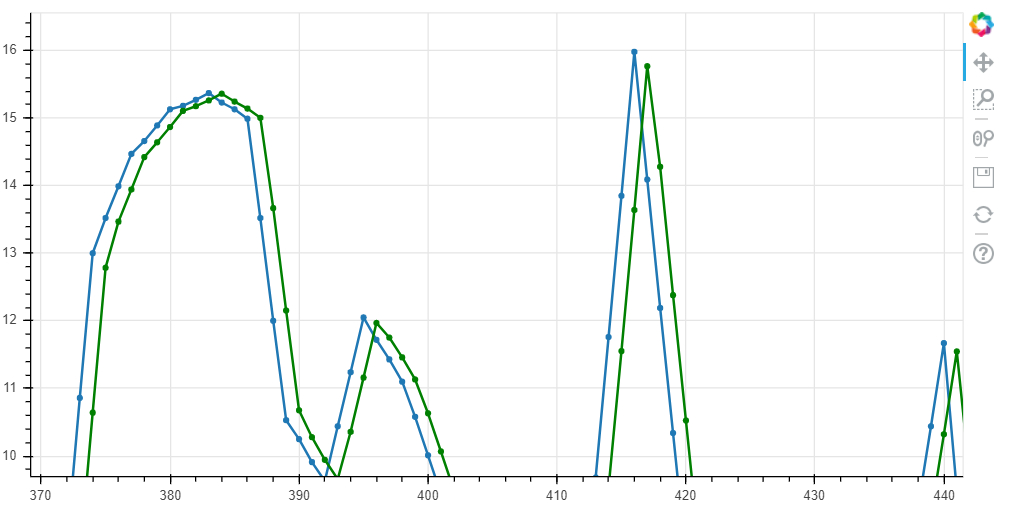


Рисунок 12. Интерактивный график для анализа временных рядов

## 5.3. Анализ высших гармоник в сети, получение спектра сигнала

Последний пример - короткий и показывает, что можно применять и сложную математическую обработку временных рядов, такую как преобразование Фурье.

Пример доступен по ссылке <https://colab.research.google.com/drive/1QwXxEi0HY5N-2XZMoFClgsbuAe0U_9rO>.

Обратите внимание, в блокноте прописаны заголовки - так блокнот смотрится лучше, чем когда в каждой ячейке просто оставлен комментарий, как было в предыдущих.

В этом примере используются “синтетические” данные, то есть не записанные в реальности, а сгенерированные определенной функцией. Результат представлен на Рисунке 13.

| import numpy as np import matplotlib.pyplot as plt  import matplotlib.ticker as ticker def fft\_sample(signal, times, sampling\_rate):  n = len(times) # число отсчетов в сигнале   k = np.arange(n)  T = n / sampling\_rate   frq = (k / T)[range(n // 2)] # диапазон частот в спектре   # применение быстрого преобразования Фурье   # (Fast Fourier Transform) с нормализацией  fft\_output = np.fft.fft(signal) / n    # достаточно половины спектра, поэтому половину обрежем  # , а оставшуюся умножим на два  fft\_output = fft\_output[range(n // 2)]  fft\_output[1 : ] \*= 2   # построение графиков   # 2 строки и 1 столбец - графики будут друг по другом  fig, ax = plt.subplots(2, 1, figsize = (20, 8))     # верхний график - исходный сигнал  # по оси абсцисс - время  # по оси ординат - сигнал  ax[0].plot(times, signal, 'blue')  ax[0].set\_xlabel('Time')   ax[0].set\_ylabel('Amplitude')  ax[0].grid(True)   # нижний - амплитудный спектр  # по оси абсцисс - частота  # по оси ординат - амплитуда соответствующих гармоник  ax[1].stem(frq, abs(fft\_output), 'red')  ax[1].set\_xlabel('Freq (Hz)')  ax[1].set\_ylabel('|A(freq)|')   # цену деления оси нужно подбирать  ax[1].xaxis.set\_major\_locator(ticker.MultipleLocator(25))  ax[1].xaxis.set\_minor\_locator(ticker.MultipleLocator(25))  ax[1].yaxis.set\_major\_locator(ticker.MultipleLocator(0.1))  ax[1].yaxis.set\_minor\_locator(ticker.MultipleLocator(0.1))  ax[1].grid(True)   plt.show()  sampling\_rate = 1000.0 # частота записи данных 1000 Гц, как будто прибор записывает значения напряжения в линии 1000 раз за секунду time\_interval = 1.0 / sampling\_rate # шаг по времени между сэмплами (точками, отсчетами)  # генерация ряда таймстампов за 0.1 секунды times = np.arange(0, 0.1, time\_interval)   # сгенерируем основной сигнал напряжения 50 Гц и добавим высшие гармоники, 3-ю и 5-ю (150 и 250 Гц) с заданными амплитудами и фазами freq\_1 = 50 # Hz freq\_3 = 150 # Hz freq\_5 = 250 # Hz  # обратный слэш используют для переноса строки кода на следующую строку signal = np.sin(2 \* np.pi \* freq\_1 \* times) \  + 0.25 \* np.sin(2 \* np.pi \* freq\_3 \* times - 0.45) \  + 0.1 \* np.sin(2 \* np.pi \* freq\_5 \* times + 0.2)  fft\_sample(signal, times, sampling\_rate) |
| --- |

Пример показывает как высшие гармоники искажают синусоиду напряжения основной частоты 50 Гц. Гармоникой или гармонической составляющей называют составляющую сигнала (временного ряда), которая изменяется по гармоническому закону a\*sin(ωt + φ), где a - амплитуда, ω - циклическая частота (ω = 2π/T, T - период), φ - фаза.

Применение преобразования Фурье позволяет увидеть на спектре из каких гармоник складыватся сигнал - их частоты и амплитуды. На спектре выделяются три гармоники, с частотами 50, 150 и 250 Гц и амплитудами 1.0, 0.25 и 0.1, точно как было задано.

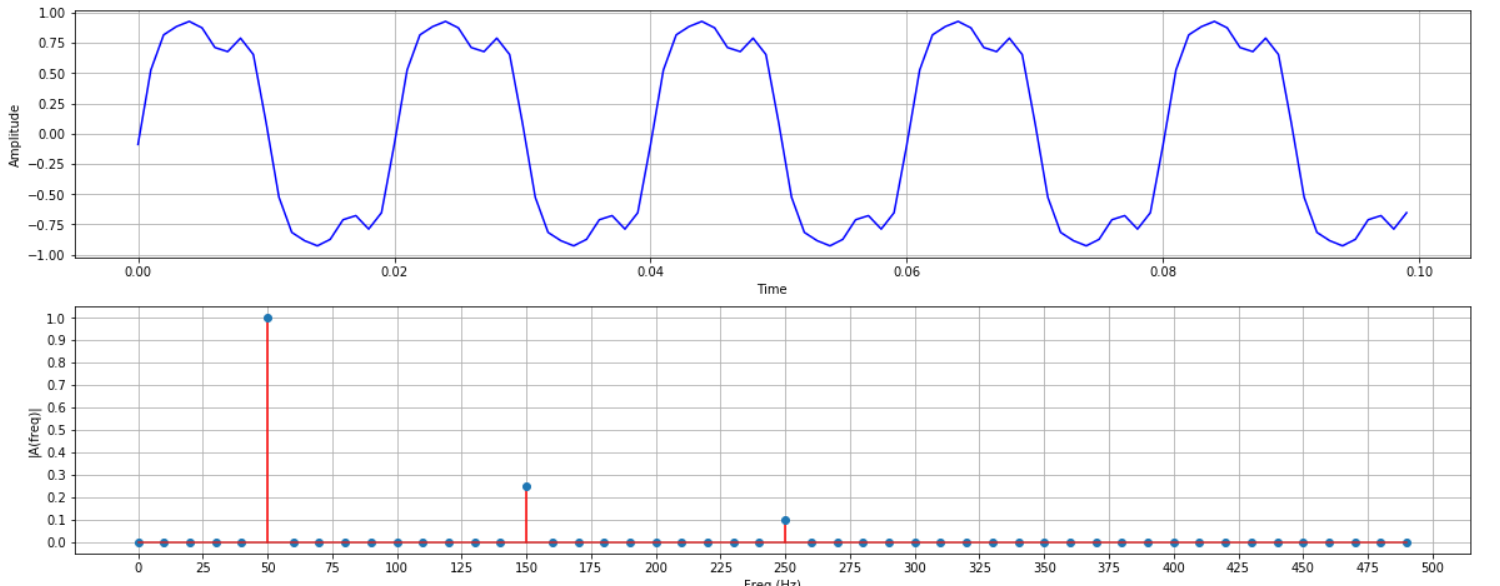


Рисунок 13. Осциллограмма напряжения и спектр

При работе с реальными осциллограммами спектры не насколько четкие, так как в сигналах присутствует намного большее количество гармоник и есть шумы и искажения. Но преобразование Фурье (точнее дискретное преобразование Фурье - ДПФ) используется очень широко для разложения сигнала на гармонические составляющие.

## 5.4. Резюме по примерам

В примерах показаны приемы работы с временными рядами. Функционал в Python библиотеках для обработки и визуализации данных - необъятен. Важно понять, что он есть, его не очень сложно использовать с минимальными навыками написания кода.

Используя приведенные здесь и далее примеры, встречаясь с задачами обработки данных в электроэнергетике и решая их не вручную, а с помощью Python, Pandas, Matplotlib и т.п., вы со временем сможете собрать свою личную коллекцию подпрограмм. Тогда получив очередную задачу и набор данных вы просто откроете Google Colab или другой подобный инструмент, возьмете нужные блоки кода и за минуты получите решение, которое другие будут делать долго.

# 6. Индивидуальные задания

1. Создать новый Google Colab блокнот. Дать ему имя. Далее действия выполняются в нем.
2. Прочитать через Pandas данные по своему варианту, используя ссылку на свой файл.
   * Может потребоваться указать символ разделения столбцов, указать точка или запятая являются десятичным разделителем и указать нужно ли использовать первую строку данных как заголовок.
   * Каталог с исходными данными для всех вариантов расположен по [ссылке](https://drive.google.com/drive/folders/1OKLYZjgSVz_WuoNc15qMMbBcZF9dAgtW?usp=sharing).
3. Вывести на экран заголовок таблицы с полученным в результате выполнения п.2 временным рядом, несколько первых и несколько последних строк таблицы.
4. Построить график или графики значений временного ряда.
5. Выполнить задание по Таблице 2.
6. Полученные графики должны быть интерактивными.
7. Сохранить полученные графики как изображения и как Excel файлы (XLSX) или CSV файлы.
8. Добавить созданный блокнот в свой GitHub репозиторий, созданный в первой лабораторной работе или в новый репозиторий.

Таблица 2. Варианты заданий по номерам

| **№** | **Задание** |
| --- | --- |
| 1 | Даны почасовые значения солнечной инсоляции. Файл “Solar\_Taj.xlsx”.  Нужно взять данные только за первый месяц и построить график среднесуточных значений и график максимальных значений за каждые сутки. |
| 2 | Даны почасовые значения скорости ветра. Файл “Wind.xlsx”.  При скорости ветра ниже 5 м/с ВЭС не вырабатывает мощность, при скорости от 5 м/с до 20 м/с вырабатываемая мощность равна 0.0017 \* wind\_speed3, при скорости от 20 м/с до 25 м/с выработка не увеличивается, при скорости выше 25 м/с электроэнергия не вырабатывается.  Построить график зависимости выработки ВЭС от скорости ветра и график среднесуточной выработки ВЭС. |
| 3 | Дан график электропотребления двух угольных разрезов за 4 года с шагом времени 30 минут. Файл “Угольные\_разрезы\_1.xlsx”.  Вывести график суммарного потребления обоих разрезов за последние 5 суток и отметить на графике точки локального максимума электропотребления. Точка локального максимума в данном случае - точка, которая выше предыдущей и выше последующей. |
| 4 | Дан почасовой график выработки ВЭС, СЭС и собственного потребления активного потребителя (prosumer). Файл “Wind\_Solar\_Consumption.csv”  Построить график разницы потребления и выработки активного потребителя и подсчитать, какое количество электроэнергии он сможет продать во внешнюю систему (или какое ему придется купить) за весь рассматриваемый промежуток времени. |
| 5 | Дан почасовой график выработки четырех одинаковых СЭС и прогноз выработки (один для всех). Файл “solar\_abcd.csv”  Построить графики ошибок прогноза для каждой СЭС и посчитать средний квадрат ошибки прогноза (Root Mean Squared Error - RMSE). |
| 6 | Дан почасовой график выработки ВЭС, СЭС и собственного потребления активного потребителя (prosumer). Файл “Wind\_Solar\_Consumption.csv”  Взять график выработки ВЭС и построить график простого скользящего среднего (simple moving average - SMA) по пяти точкам. |
| 7 | Даны показания силы тока в линии (микроамперы). Файл “Current.csv”.  Взять первые 1280 значений, подсчитать среднее значение силы тока на всем этом интервале.  Построить график значений мощности (P = I2R, R = 10 Ом).  Сравнить среднее значение мощности как среднеарифметическое значений графика мощности и как полученное по формуле P = I2R, где вместо I подставлено среднее значение модуля тока на рассматриваемом интервале. |
| 8 | Дан посекундный график освещенности. Файл “light.csv”.  Построить алгоритм, управляющий освещением. По заданию при освещенности ***примерно*** ниже 35 условных единиц должен включаться свет, при освещенности ***примерно*** выше 35 условных единиц - выключаться. |
| 9 | Даны графики выработки электроэнергии нескольких электростанций в МВт с интервалом 5 минут. Файл “gridwatch\_data.csv”.  Взять данные за первые 12 часов и построить графики с тем же интервалом 5 минут, показывающие, какой процент от всей выработки дает каждая станция. |
| 10 | Дан график почасового потребления нескольких предприятий за 3 года. Файл “Электропотребление.xlsx”.  Построить для каждого графики изменения потребления от часа к часу за последнюю неделю. |
| 11 | Дано среднемесячное потребления электроэнергетической системы Памира. Файл “Памир.txt”  Построить амплитудно-частотный спектр и предположить, почему получились именно такие частоты наиболее значимых гармоник. |
| 12\* | Даны значения тока в линии с шагом времени 0.15625 мс. Файл “Current.csv”.  Построить амплитудно-частотные спектры по току и напряжению для всех фаз и сделать вывод о наличии в них высших гармоник . |
| 13\* | Дана осциллограмма тока нестандартной частоты. Файл “current\_freq.csv”.  Построить амплитудно-частотный спектр, определить основную частоту.  Затем для удаления шума из данных применить low-pass filter следующего вида:  output[i] = input[i] \* k + output[i-1] \* (1. - k), (k = 0.95)  Построить амплитудно-частотный спектр после очищения от шума, определить основную частоту. |

# 7. Отчет один на бригаду

* Титульный лист.
* Цель работы.
* Имя GitHub репозитория, в который сохранен код программы.
* Python код программы.
* Полученные графики и результаты расчетов, если были по заданию.