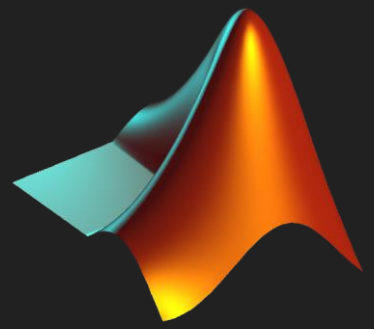
Лекция по MatLab №1

ОСНОВЫ ПРОГРАММИРОВАНИЯ

Что такое MatLab?



MatLab (сокращение от англ. «Matrix Laboratory») — пакет прикладных программ для решения задач технических вычислений и одноимённый язык программирования, используемый в этом пакете. Пакет используют более миллиона инженерных и научных работников, он работает на большинстве современных операционных систем, включая **Linux**, **Mac OS** и **Windows**.

Аналоги MatLab?

Аналогов у MatLab можно насчитать около десятка программ, но самые популярные:

- GNU Octave

- Scilab

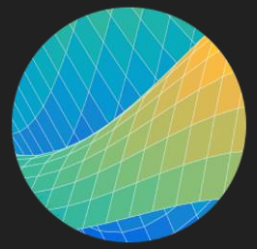
- FreeLab

Возможности MatLab?

Решения на основе MatLab:
- обработка больших массивов данных;
- машинное обучение;
- интернет вещей (IoT): arduino, Raspberry Pi;
- моделирование систем и комплексов;
- обработка сигналов и изображений;
- Front-end разработка;

и многое другое!

MatLab



MATLAB
Язык программирования

SIMULINK
Среда разработки моделей

Почему MatLab?

- В нём множество готовых, встроенных функций, несложных в обращении
- Нет необходимости объявлять переменную и её тип в заголовке программы: все переменные создаются и заполняются по мере работы программы. Это упрощает создание переменных и контроль за типом этих переменных, но пропадает контроль за уникальностью создаваемых переменных
- Есть возможность создания как консольных программ, так и программ с интерфейсом, а также их «запаковки» в формат приложения (*.exe)
- Мощная система помощника и готовых решений другими пользователями
- Существуют целые программные блоки (ToolBox) для подключения разнообразной аппаратной периферии, а также работы с разными задачами (моделирование, экономика, статистика, нейронные сети, обработка сигналов и картинок и т.д.)

Интерфейс программы MatLab:

Интерфейс состоит из следующих основных частей:

- **Workspace** – отображаются созданные переменные и их значения;
- **Command Window** – окно команд, где отображаются результаты работы скриптов, а также где можно составить программу или проверить некоторые команды напрямую;
- **Editor** – окно написания скрипта «консольной» части программы.

Ввод и вывод данных

Для задания определенных значений переменным используется *оператор присваивания*, вводимый знаком равенства (=)

Имя_переменной = Выражение ;

Напр.: `time = 3.56;`

При этом дробные числа вводятся только через точку!

- Имя любой переменной должно быть уникальным. Имя должно начинаться с буквы, может содержать буквы, цифры и символ подчеркивания (_). Недопустимо включать в имена пробелы и специальные знаки.
- В языке MATLAB нет явных операторов ввода вывода данных в режиме диалога. Для вывода нет необходимости после математического выражения ставить символ точку с запятой (;). Но если она не будет стоять, то в command window будет выводится все промежуточные расчеты программы.
- К системным константам относятся:

pi = 3,1415 … число “ПИ”;
i или **j** - мнимые единицы;
NaN - неопределенность в виде 0/0;
Inf - бесконечность типа a/0 ;
ans - результат последней операции.

Создание массивов (векторов)

- Для создания вектора значений с шагом в MATLAB необходимо использовать оператор *двоеточие* (:), который представляется следующим образом:

X = *Начальное_значение* : *Шаг* : *Конечное_значение* ;

Напр.: `time = 0 : 1 : 60;`

Вывод: в переменной time содержится массив: 0, 1, 2, 3, 4 … 60

- Создание массива вручную выполняется благодаря операторам квадратных скобок [ ], следующим образом:

X = [*значение1, значение2, … , значениеN*];

При этом вместо запятой можно оставлять пробелы.

Напр.: `time = [ 0 , 1 , 2 ]` или `time = [ 0 1 2 ]`.

- Создание матриц выполняется следующей командой:

X = [*значения_первого_рядка* ; *значения_второго_рядка*]

Напр.: `time = [1 2 3 ; 4 5 6 ; 7 8 9]`

Обращение к элементам массива

Обращение к элементам массива делается в двух случаях:

- **Для обращения ради считывания текущего значения в массиве, чтоб использовать его или ознакомиться.**

Напр., обращение к 3-му элементу массива time следует производить следующим образом: `time(3)`

Для элемента матрицы во втором столбике и третьем рядке это выглядит так: `time(2, 3)`. Запятая в данном случае обязательна!

- **Для изменения элемента массива.**

Напр. Чтобы присвоить третьему элементу массива новое число необходимо сделать следующий запрос: `time(3) = 34.3`

Математические операции с массивами и матрицами

Символьно математические операции ничем не отличаются от обычной математики. Можно использовать следующие основные математические операции над переменными (содержащим одного число):

- + сложение
- - вычитание
- * умножение
- / деление

Напр. При сложении двух чисел в Command window нужно ввести: 5 + 6 [нажать Enter]. Или для умножения двух переменных хранящимися в них числами: a*b [нажать Enter].

Необходимо помнить, что делить на ноль в математике нельзя. При таком действии MatLab присвоит переменной ответа значение «Inf», то есть бесконечность (от слова «infinitive»)

Математические операции с массивами и матрицами

Операции с массивами и матрицами могут иметь отличия, которые следует различать. Операции над элементами могут быть произведены в прямом назначении (умножение двух матриц в математике) или поэлементном (умножение 1-го элемента одного массива/матрицы на первый элемент второго массива/матрицы и так далее с каждым элементом). При этом для поэлементных операций к стандартной операции добавляется точка:

- .* поэлементное умножение
- ./ поэлементное деление
- .^ поэлементное возведение в степень

Нужно отметить, что операции сложения и вычитания с массивами/матрицами и так происходит поэлементно, поэтому точка перед символами «+» и «-» не требуется

Логические операции

Логические операции применимы как непосредственно с числами, так и с переменными, содержащие эти числа. Логическими они названы потому, что результатом их работы есть логический вывод состоящий из четкого одного ответа «да» (true - истина) или «нет» (false - ложь). С точки зрения Булевой алгебры («алгебра единиц/нулей») это ответы один («1») или ноль («0»).

Существует два типа логических операторов:

- Операторы отношения - служат для сравнения двух величин, векторов или матриц
- Логические операторы - служат для реализации поэлементных логических операций над элементами одинаковых по размеру массивов

Операторы отношения

Существуют следующие стандартные операторы отношения:

- Равенство == (Вопрос: «равны ли числа?»)
- Неравенство ~= (Вопрос: «эти числа не равны?»)
- Меньше < (Вопрос: «меньше ли первое число второго?»)
- Больше > (Вопрос: «больше ли первое число второго?»)
- Меньше или равно <= (Вопрос: «меньше или равно ли первое число второго?»)
- Больше или равно >= (Вопрос: «больше или равно ли первое число второго?»)

Напр. при введении логической операции 3 > 2 [нажатие Enter] мы как бы спрашиваем компьютер: «три больше двух?» Ответ такой операции выдаёт нам единицу («да, больше»).

Логические операторы

Логические операторы используются для операций над логикой – единицами и нулями. Для лучшего понимания можно рассмотреть раздел математики: операции над множествами.

Существуют следующие стандартные логические операторы:

- Логическое И (&&)
- Логическое ИЛИ (||)
- Логическое НЕ (~)

Напр. нам необходимо выполнить код при выполнении одновременно двух условий, когда переменная «a» находится в промежутке между числом пять и десять, это можно задать кодом:

```
if a>5 && a<10
            disp(` Попадание! `) % функция disp отображает фразу в
                                   окно Command window
end
```

(задаётся условие: «только если «a» больше 5-ти И одновременно меньше 10-ти – выполните следующие действия»)

Использование Help

Умение пользоваться помощником при программировании это половина от успешного пользования любой программной средой! В MatLab предусмотрено несколько вариантов использования функцией Help:

- Непосредственно в Command window. Для этого нужно написать

help искомая_функция

Напр.: `help fft`

- Используя кнопку F1, предварительно выделив нужную функцию курсором мыши
- Используя символ знака вопроса («?») на панели главного окна Matlab (визуальная оболочка)

Особенности создания скрипта

- Скрипт (или м-файл) можно создать либо из интерфейса главного окна, или командой «edit» из Command window
- Сохранять скрипт стоит только на латинице с директорией, состоящей только из латинских букв, хотя последние версии Matlab работают с любой директорией, но на больших проектах лучше не рисковать
- Можно создавать собственные функции (function) к которым можно обращаться из центрального файла. Все дополнительные функции стоит хранить в центральной рабочей директории создаваемой программы. Создание собственных функций упрощает работу с часто повторяющимися одинаковыми операциями.

Особенности создания скрипта

- MatLab работает в рабочей директории, которая определяется при первом запуске центрального м-файла программы. Узнать текущую рабочую директорию можно с помощью функции «pwd» (personal work directory). При первом запуске скрипта MatLab сам предложит изменить текущую директорию «Change directory»
- В любых программах существует возможность создать комментарий. Такие комментарии необходимо создавать в больших проектах, для описания тех или иных шагов работы программы. Это поможет Вам разобраться с кодом, после того как вы давно его не редактировали его или поможет разобраться в работе программы другому пользователю, редактируемому Ваш код. В Matlab такие комментарии создаются с помощью символа процента (% или %%), после написания которого текст комментария будет выделен зеленым цветом. Комментарии лучше писать латиницей!

Важно знать, что с помощью комментариев можно временно отключать выполнение целых блоков программы. Для этого необходимо выделить строки блока программы, который не нужно выполнять и нажать **ctrl + R** или выбрать в контекстном меню «Comment»

Лекция по MatLab №2

ЦИКЛЫ И УСЛОВИЯ

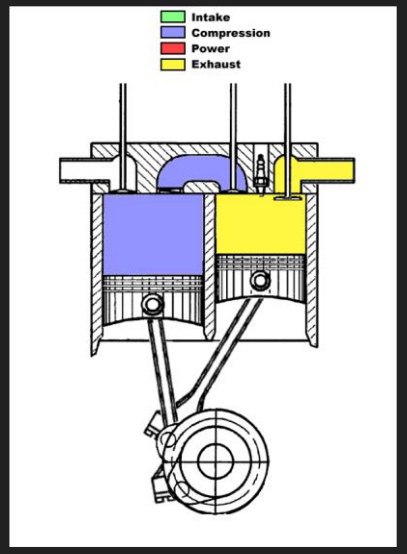
Перед началом…

Перед началом данной лекции необходимо познакомится с двумя простыми функциями, которые создают удобные условия при программировании:

- Функция `clc` – очищает окно Command window от вводимых или выводимых результатов работы.
- Функция `clear` – удаляет созданные переменные из памяти программы (окно workspace). Данная функция может использоваться в разных вариациях, напр.:
  - `clear X` – удалит конкретную переменную «X»;
  - `clear all` – удалит переменные и загруженные данные;
  - `clear global` или `clear global X` – удалит глобальные переменные (об этом в следующих главах) или конкретную глобальную переменную «X».

Что такое цикл:

Цикл — разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций. Также циклом может называться любая многократно исполняемая последовательность инструкций, организованная любым способом (например, с помощью условного перехода).



Разновидности циклов:

- Цикл **While**. (этот цикл может быть «бесконечным»)

```
while <условие работы цикла>
        <операторы>
end
```

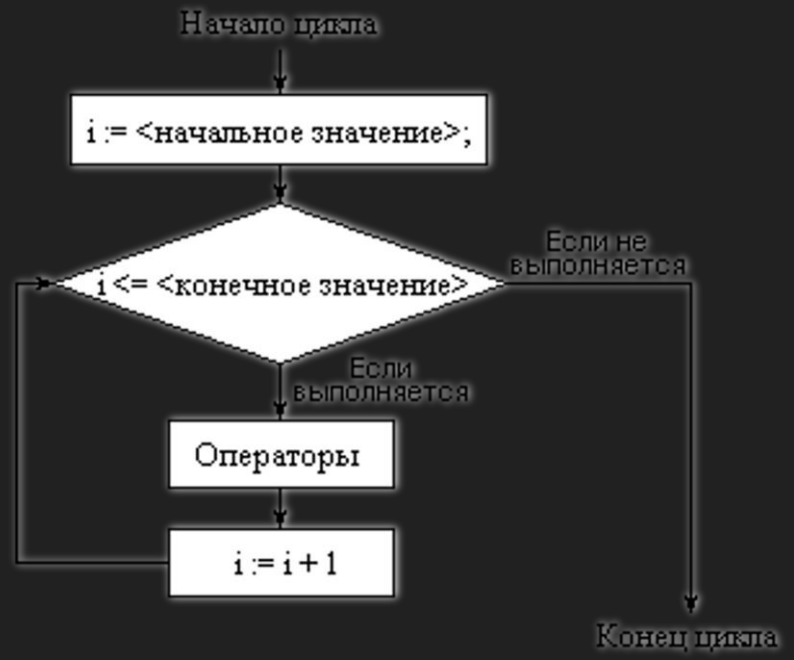
- Цикл **for**.

```
for <условие изменения переменной цикла>
        <операторы>
end
```

Пример цикла While:

Пример цикла:

```
S = 0;                  % начальное значение суммы
i = 1;                  % счетчик суммы
while i <= 20           % цикл (работает пока i <= 20)
     S=S+i;             % подсчитывается сумма
     i=i+1;             % увеличивается счетчик на 1
end                     % конец цикла
disp(S);                % отображение суммы 210 на экране
```



Цикл можно задать бесконечным, прописав его в виде **While (1)**, где единица в скобках есть логической единицей (можно подставить слово “true”). Таким образом можно визуально отделять один цикл от другого.

Пример цикла for:

В цикле for не нужно использовать счётчик цикла, он задаётся в виде конкретного условия изменения цикла (после оператора *for*), напр.:

```
S = 0;              % начальное значение суммы
for i = 1:1:20      % цикл (i меняется от 1 до 20 с шагом 1)
     S=S+i;         % подсчитывается сумма
end                 % конец цикла
disp(S);            % отображение суммы 210 на экране
```

В представленном цикле условие цикла задавать не только в виде шага, но и в виде массива чисел, что делает такой цикл универсальным, напр.:

```
for i=[1 3 6]
        …
end
```

Оператор остановки цикла

- Оператор **break** (прервать) – вызываетпрекращение работы цикла, обычно используется для прерывания цикла с условиями. Напр.:

```
if S > 20              % если S > 20,
      break;           % то цикл завершается
end                    % конец цикла
```

- Оператор **continue** (пропустить) – начинает цикл заново, увеличивая счетчик на единицу. Обычно используется для исключения каких-либо операций подпадающих под условие. Напр.:

```
if i == 5              % если индекс равен 5
      continue;        % то его не рассчитываем в цикле
end                            % конец цикла
```

Для изучения циклов важно знать комбинацию клавиш для остановки выполнения любых действий программы. Эта комбинация используется в большинстве сред программирования, это комбинация: **ctrl + C** (от слова cancel)

Оператор прерывания для условий

Для условий часто используют оператор **return** для прерывания работы программы. Например, когда достигнуто нужное условие программы и дальнейшее выполнение программы не имеет смысла. Напр.:

```
        for i=1:20                                  % цикл, где переменная
меняется от 1 до 20ти
                disp(i)                                     % вывод информации
о значении переменной i
                    if i==5                         % условие когда переменная
равна 5ти
                    disp('i=5')             % проверяем срабатывание условия
                            return                  % прерываем работу всей
программы
                    end                                     % конец условия
        end                                         % конец цикла
        a=0                                                 % проверяем
действие оператора return - переменная «a» не должна создастся
```

Данный оператор часто используется при создании программы в виде интерфейса, где весь код строится на функциях, срабатывающих при условии каких-либо взаимодействий пользователя с элементами интерфейса.

Виды условий:

- Условие **If**
- Условие **If…elseif**
- Условие **switch, case, otherwise**
- Условие **try, catch**

Условия if...else if:

- Условие **If**:

```
if <логическое выражение> % оператор условия
      <операторы>                    % действия
                                           % оператор
else
обратного условия
      <операторы>                    % действия при
выполнении обратного
                  условия
end                                        % конец условия
```

- Условие **If...elseif**:

```
if <логическое выражение>              % оператор условия
      <операторы>                          % действия
elseif <логическое выражение>    % оператор обратного

        условия
      <операторы>                          % действия
                                                  %
end
конец условия
```

Условие switch, case, otherwise :

Условие **«switch, case, otherwise»** (ключ, состояние) применяют, если определенная переменная может принять несколько определенных значений. Общий синтаксис выглядит следующим образом:

```
switch <переменная>
        case <значение №1_принемаемое_переменной>
                <операторы>
        case <значение №2_принемаемое_переменной>
                <операторы>
...
        otherwise                       % если ни одно из условий выше не выполнено
                <операторы>
end
```

Напр. создадим программу проверки числа *п* в переменной «p»:

```
p=3.15                                  % переменную нужно изменять, чтоб проверить
                работоту программы
switch p
        case 3.14
                disp('Chislo pi')
        otherwise
        disp('Peremennaya "p" ne chislo Pi')
end
```

Условие try, catch :

Условие **«try, catch»** (попытка) является специфическим условием, которое используется если существует вероятность ошибочного действия пользователя. Функция делает попытку выполнить действие, и вместо того чтоб выдать ошибку в command windows выполняет запрограммированное действие. Синтаксис функции имеет следующий вид:

```
try <оператор-попытка>;
        catch
        <оператор>   % выполняется, если первичная попытка не удалась
end                                 % конец условия
```

Напр.:

```
try a=b                             % попытка присвоить переменной «a» значение переменной «b»
        catch
                a=0             % так как переменная «b» не существует, то переменной «a» присваивается числовое значение нуля.
end
```

Лекция по MatLab №3

ФУНКЦИИ И ПОСТРОЕНИЕ ГРАФИКОВ

Что такое функция?

Функции (*function*) в MatLab это набор команд, выполняющие цели программиста, которые находятся в отдельном файле или вызываются однострочной инструкцией.

В программе MatLab можно создавать свои собственные функции, наподобие функций *plot*, *fft*, *sin*, *cos*.

Реализация своих функций необходима для удобства, например, когда программа требует выполнения однотипных операций в разных частях основного кода. Чтоб не «захламлять» основной код однотипным кодом, который отвлекает внимание, лучше воспользоваться созданием функции.

Синтаксис `function`:

“Function” можно использовать в двух вариантах, но общий синтаксис создания функции остается одинаковым, а именно:

```
function [выход_1, выход_2, … , выход_n] = название_функции(аргументы)
```

напр., `function [y]=sin(x)`.

Использовать такую созданную функцию можно обращением к названию функции, задавая необходимые вам переменные на вход (аргумент) и выход.

Некоторые особенности и правила пользования функциями:

- Входные и выходные переменные могут содержать массивы данных;
- Входных и выходных переменных может быть много;
- Переменные на входе и выходе могут быть разными по сложности, напр., `[exit_statment]=sin(enter_statment)`, но функция, к которой идёт обращение должна быть строго заданной для программы. В данном случае «sin» является направляющим для программы в выполнении инструкций связанных с этой функцией;
- Функция может не иметь выходных переменных (результатов работы).

Использование function:

- **Внутри исполняемого файла** (Применяется очень редко)

```
function Untitled(X)
          A=[2 3 4];
          B=[1 2 3];
          display_my(A,B)
end
function display_myu(first,second)
          C=first+second;
          disp(C)
end
```

При создании таких функций название m-файла должно соответствовать основной функции, в нашем случае «Untitled.m»

- **Созданием отдельного m-файла**

Необходимо нажать на иконку создания нового скрипта и прописать функцию с выполнением команд внутри. При этом не нужно закрывать функцию словом «end», но файл должен быть назван также как сама функция!

Создайте основной скрипт (m-файл) с названием «my_program» и внесите туда код:

```
A=[2 3 4];
B=[1 2 3];
C=A+B;
my_disp(C) % Здесь идет обращение к нашей новой функции, которая создается в другом файле
```

Создайте m-файл с названием «my_disp» и внесите туда код:

```
function my_disp(X) % функция имеет один входящий аргумент и ноль выходящих результатов
disp(X)
```

Обратите внимание, что внутри функции входные и выходные переменные используются локальным образом. Так переменная «C» в самой функции станет переменной «X». Тоже самое происходит и с выходными переменными. Таким образом достигается универсальность работы функции.

function с выходными пар-ми:

Рассмотрим пример с выходными данными работы функции.
Используйте файл «my_program» для ввода основной программы:

```
a1=3;
a2=5;
[rand1, rand2]=my_rand(a1, a2);    % наша функция; создается ниже
figure(1)                                                    %    открывает
окно для графических операций
plot(rand1)                                                  %  строит  график  из
массива rand1
figure(2)
plot(rand2)                                                  %  строит  график  из
массива rand2
```

Создайте файл «my_rand» с соответствующей функцией, которая будет создавать массивы случайных чисел:

```
function [y1, y2]=my_rand(x1,x2)
y1=randn(1, x1);                                             %   создает   вектор-
массив случайных чисел
y2=randn(1, x2);
disp(y1)
disp(y2)
```

Функция «figure»:

Программирование всегда является последовательностью действий. Перед тем как создать график или интерфейс программы необходимо создать основное окно с помощью команды «figure()». Синтаксис этой команды имеет следующий вид:

```
figure('Name','Моё окно','NumberTitle','off')
```

Аргументы `'Name'` и `'NumberTitle'` являются свойствами (Properties) для создания окна *figure*. Так как окно является объектом (который создается на основе встроенных в операционную систему команд, функций, визуализаций), то оно содержит ряд свойств. Если присвоить наш объект некой переменной, то можно обращаться к свойствам этого объекта в будущем, например:

```
h = figure('Name','Моё окно','NumberTitle','off');
```

где «h» содержит все данные о созданном объекте *figure.*

Для быстро создания окон figure можно использовать команду figure(1), меняя цифру внутри функции.

Функция plot

Функция *plot – основная функция для построения графиков в Matlab*.
Общий синтаксис данной функции имеет следующий вид:

```
plot(x, y, 'setup')
```

X и Y – массивы точек графика, которые должны соответствовать множеству точек рисунка, напр.:

```
plot([1 2 3], [1 2 3])
```

Возможность нарисовать несколько графиков в одном окне можно реализовать следующим образом:

```
plot(x1, y1, 'setup1', x2, y2, 'setup2'…)
```

Также существует возможность “наслоить” графики друг на друга с помощью задерживающей функции hold on. Изучите ее использование самостоятельно.

Заметьте, что рисование графиков происходит в виде линии, хотя указываются точечные значения в массивах. Для построения только точек нужно произвести настройки (см. следующий слайд)

Настройки (setup) в функции plot

Вместо `'setup'` можно установить разные настройки рисунка, напр.:

- «`'--'`» рисует штриховую линию
- «`'-.'`» рисует линию точку
- «`'o'`» рисует точки графика в виде круга
- «`'*'`» рисует точки графика в виде звездочек
- «`'s'`» рисует точки графика в виде квадратов
- «`'r'`» рисует график красным (red) светом. Также можно использовать другие сокращения цветов «`'y'`» (желтый) «`'k'`» (черный) «`'w'`» (белый). При использовании нескольких свойств нужно указывать отдельно свойство цвета как: `plot(…, 'Color', 'r')` или `plot(…, 'Color', 'red')`

Для быстрой настройки графика в месте ввода setup можно использовать сокращенные указания, напр: «`'- - s r'`» построит штрих-пунктирный красный график с кругами в ключевых точках.

Более подробный перечень свойств функции *plot* можно найти в *Help*’е.

Настройка окна графика

Для верного построения любого графика нужно произвести настройку обозначений. В основном могут потребоваться следующие элементы настройки графика, которые выполняются как отдельные функции в коде программы:

- `title('Graph name')` – подпись названия графика (над ним)
- `xlabel('text'), ylabel('text')` – обозначение осей графика
- `xlim([x1 x2]), ylim([y1 y2])` – ограничение показа графика в окне по осям
- `legend('sin', 'cos'…)` – подпись нескольких графиков, построенных в одном окне
- `text(x, y, 'message')` – текстовое сообщение размещенное в определенной точке графика. Можно заменить функцией *gtext*.