Индексы. 2D-3D графики. Функции.

Прикладные программные системы
Лекция №2

Обзор пройденного

- Что такое MatLab и зачем он нужен
- Переменные и операции над ними
- Вектора, матрицы и то, как с ними работать
- Математические функции и операции (всё способно работать с матрицами как со смысловым элементом)
- Условные конструкции **if** и **switch**.
- Цикл **while**.
- Цикл **for**

План лекции

В течение лекции мы планируем ответить на следующие несколько вопросов:

Часть 1

- Как обратиться к элементу матрицы?
  - (под матрицей мы будем понимать в том числе и вектор)
- Как обратиться к группе последовательных элементов матрицы?
- Как обратиться к группе разрозненных элементов матрицы?
- Как обратиться к группе элементов матрицы, отвечающих какому-либо условию?

Часть 2

- Как построить и настроить различные виды 2D-графиков?
- Какие функции используются для построения 3D-графиков и как они работают?

Часть 3

- Как создать собственную функцию в MatLab?
- Зачем создавать собственные функции?
- Что такое анонимная функция? И как она упрощает жизнь?

Индексы и их использование

— Unfortunately, no one can be told what the Matrix is. You have to see it for yourself.

Morpheus, «The Matrix»

Обращение к элементу матрицы

- Обратиться к элементу вектора можно, указав его номер в **круглых скобках** после имени вектора:
  - `>> v1 = [1 3 5 7 9];`
  - `>> b = v1(3); % теперь b = 5`
  - **нумерация начинается с 1**
- Обратиться к элементу матрицы можно, указав **сначала номер строки** и, через запятую, номер столбца в круглых скобках после имени матрицы:
  - `>> A = [2 3; -1 -3];`
  - `>> n = A(2, 1); % теперь n = -1`
  - `>> m = A(2, 0); % ошибка! нулевого столбца не существует!`

Обращение к группе элементов (I)

- Очень часто требуется «выдрать» из матрицы кусок последовательных (или произвольных) элементов.
- Рассмотрим, как это работает с векторами.
- Используется специальный синтаксис:
  - `>> v1 = [5 6 7 1 2 3];`
  - `>> v2 = v1(3:5); % теперь v2 содержит элементы из v1 с третьего по пятый включительно`
  - мы записали **целочисленный вектор** там, где обычно пишется один индекс; это не случайно! (см. далее)
- Мы можем выбрать произвольный набор индексов, записать их как вектор и обратиться к соответствующим им элементам:
  - `>> idxs = [3 3 2 1 5 2]; % заметьте, можно записывать одни и те же номера!`
  - `>> v3 = v1(idxs); % теперь v3 = [7 7 6 5 2 6]`
  - `>> v4 = v1([2 1 1 5]); % или даже так`

Обращение к группе элементов (II)

- То же самое можно проделывать и с матрицами
- Возьмём в качестве примера следующую матрицу 3x3:
  - `>> A = [8 1 6; 3 5 7; 4 9 2]; % её можно создать функцией magic(3)`
  - для наглядности см. запись справа
  - `>> b = A(1:3, 1); % первый столбец матрицы`
  - `>> b = A(:, 1); % синтаксический сахар для того же самого`
  - `>> c = A(2, :); % вторая строка матрицы A`
  - `>> B = A(2:3, :); % матрица, содержащая вторую и третью строки матрицы A`
  - `>> C = A(2:3, 2:3); % матрица, содержащая все элементы, находящиеся на пересечении 2 и 3 строк со 2 и 3 столбцами (см. справа)`

$$A = \begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix}$$

$$C = \begin{bmatrix} 5 & 7 \\ 9 & 2 \end{bmatrix}$$

Обращение к группе элементов (III)

- Есть ключевое слово **end**, которое позволяет «не знать» длину вектора или размерности матрицы, когда обращаетесь к группе индексов:
  - `>> A = [8 1 6; 3 5 7; 4 9 2]; % тот же пример`
  - `>> B = A(2:end, 1:2); % догадайтесь, какие элементы попадут в матрицу B?`
- Естественно, можно обращаться и к разрозненной группе строк и столбцов
  - `>> B = A([3 1], 2); % здесь B = [9; 1] - вектор столбец`
  - `>> C = A([1 3 end], [2 2 1 4]); % какова размерность матрицы C? Что она будет содержать? Ответ на следующем слайде`

Логические индексы (I)

- Как извлечь из матрицы все элементы, которые меньше нуля? больше тройки? делятся на 5? Легко!
- Первое, что нужно знать: логические операции над матрицами порождают матрицы логических значений. Сложно? Ничего подобного!
- Пример:
  - `>> X = [3 -1; -2  0]; % простая матрица 2x2`
  - `>> L = X < 0; % странная запись... но ничего странного не происходит, L будет содержать матрицу той же размерности, что и X, но содержащую 1 на тех местах, при которых условие выполняется, 0 – где не выполняется (см. справа)`

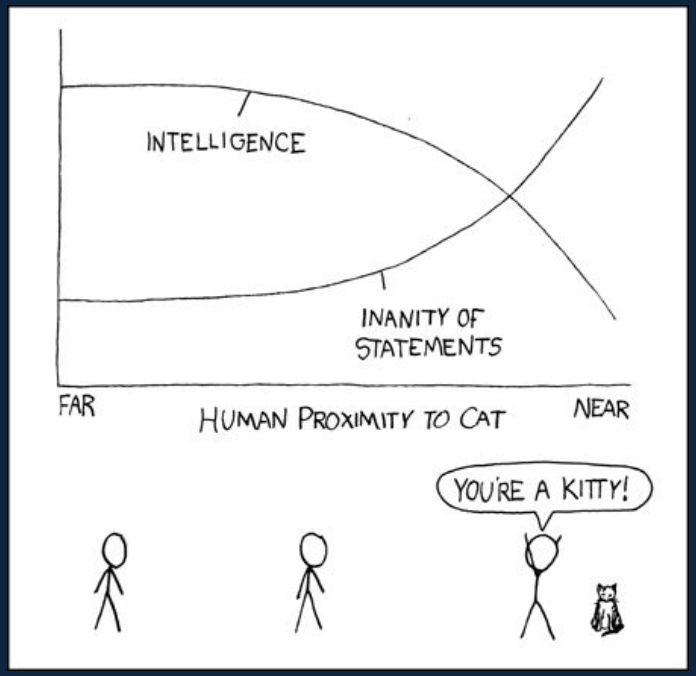
$$L = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

  - тогда мы можем «выдернуть» всё нужное из матрицы X так:
  - `>> Y = X(L); % теперь Y = [-2; -1]`
  - можно написать проще:
  - `>> Y = X(X < 0); % тот же результат`

Логические индексы (I)

- Второй способ — использовать функцию **find**.
- Она **находит индексы** всех элементов, удовлетворяющих некоторому условию.
- В случае с матрицами «индексом» является номер элемента при проходе сверху вниз по столбцам:
  - `>> A = [-3 1; 2 4]; % здесь число 2 имеет индекс 3.`
  - `>> Idxs = find(rem(A, 2) == 0); % что выдаст эта функция?`
  - теперь результат **find** подставляем в качестве индексов:
  - `>> B = A(Idxs); % теперь B = [2; 4]`
  - можно обойтись без промежуточных матриц:
  - `>> B = A(find(rem(A, 2) == 0)); % тот же результат`
  - но это то же самое, что
  - `>> B = A(rem(A, 2) == 0);`
  - поэтому **find** в таком контексте используется редко

2D- и 3D-графики



http://xkcd.com/231/

Настройка графического окна

Графические окна

- Каждый график в MatLab создаётся в специальном графическом окне.
- Такое окно имеет заголовок «Figure ###», где ### — номер окна.
- Вы можете вручную создавать графические окна, используя функцию **figure**, аргументом которой является номер окна:
  - `>> figure(5); % окно с номером 5`
  - `>> plot(0:0.1:10, (0:0.1:10).^2); % y = x^2`
  - `>> figure(77); % окно с номером 77`
  - `>> plot(0:0.1:10, (0:0.1:10).^3); % y = x^3`
- Хорошим тоном является использование последовательных номеров графических окон: 1, 2, 3..
- Сохранить график как рисунок (jpg, png и т.д.) можно через меню графического окна:
  - **File → Save as...**

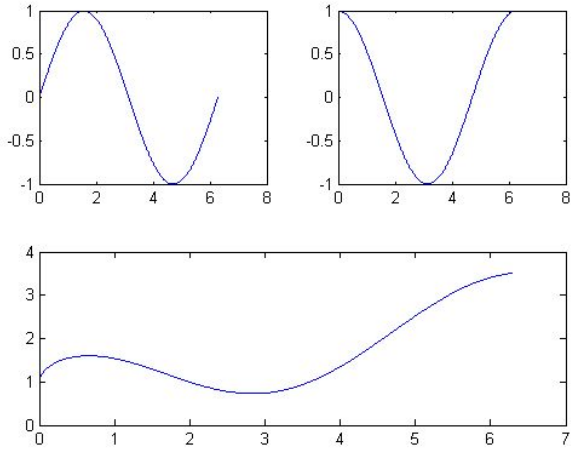
    Никогда не сохраняйте график через PrintScreen!

Разбиение графического окна (I)

- Для того, чтобы построить несколько графиков внутри одного графического окна (каждый со своей системой координат), существует функция **subplot**.
- Она имеет три аргумента:
  - количество строк, на которые разбивается окно
  - количество столбцов, на которые разбивается окно
  - номер системы координат, в которой мы строим
- Функция построения графиков располагается после соответствующего вызова **subplot**.
- Пример:
  - `>> subplot(1, 2, 1); % разбиваем на 2 столбца и выбираем 1-ю систему координат`
  - `>> plot([1 2 3], [0 -1 9]); % строим что-нибудь`
  - `>> subplot(1, 2, 2); % выбираем 2-ю систему координат`
  - `>> plot(0:0.1:10, sin(0:0.1:10)); % строим что-нибудь ещё`

Разбиение графического окна (II)

- Более сложный пример с объединением подокон:
  - `>> x = 0:0.01:2*pi;`
  - `>> subplot(2, 2, 1); % разбиваем на 2 строки и 2 столбца, выбираем 1 подокно`
  - `>> plot(x, sin(x));`
  - `>> subplot(2, 2, 2); % выбираем 2 подокно`
  - `>> plot(x, cos(x));`
  - `>> subplot(2, 2, 3:4); % выбираем` **сразу две** координатные сетки
  - `>> plot(x, cos(x) + sqrt(x));`



Управление состоянием графического окна

- Помимо использования одной функции plot для построения нескольких графиков, можно строить в одном окне и несколькими различными функциями plot.
- Для этого после создания графического окна (или первого построенного графика) используется команда
  - `>> hold on`
- С этого момента все вновь построенные графики будут строиться «поверх» друг друга.
- Отключить этот режим можно командой
  - `>> hold off`
- Такое написание функции (через пробел) — это просто синтаксический сахар для
  - `>> hold('on')`
- Координатную сетку на графике можно задать функцией
  - `>> grid on; % отключать аналогично hold'y`
- Очистка текущего окна — команда **`cla`**

Управление осями графического окна (I)

- Управление осями осуществляется уже после построения графика
- Основная команда для задания пределов отображения графика — это **axis**
- У неё множество вариантов использования. Как с автоматическим заданием пределов отображения, так и с ручным.
- Вариант ручного управления:
  - `>> axis([x_min x_max y_min y_max]); % аргумент – вектор-строка из 4 элементов`
- То же самое можно сделать для осей X и Y отдельно функциями **xlim** и **ylim**:
  - `>> xlim([x_min x_max]);`
  - `>> ylim([y_min y_max]);`

Управление осями графического окна (I)

- Автоматическая настройка осей:
  - `>> axis equal; % одинаковый масштаб осей X и Y`
  - `>> axis tight; % прижать границы к графику`
  - `>> axis square; % квадратная «коробочка»`
  - `>> axis off; % отключить отображение осей`
  - `>> axis xy; % начало координат слева внизу`
  - `>> axis ij; % начало координат слева вверху (полезно для работы с изображениями)`

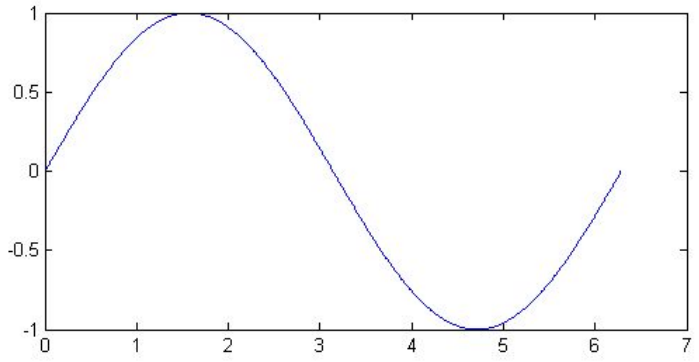
2D-графики

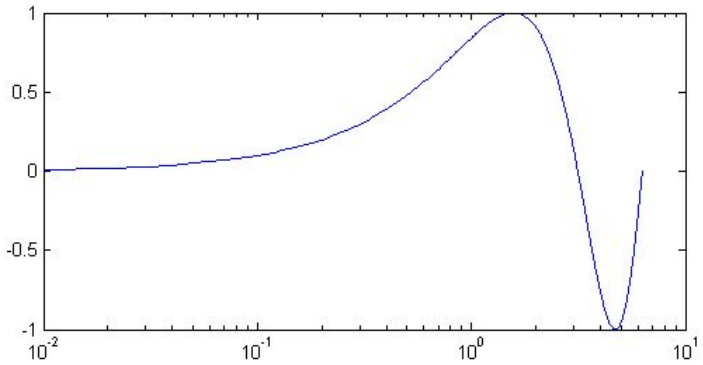
Настройки plot

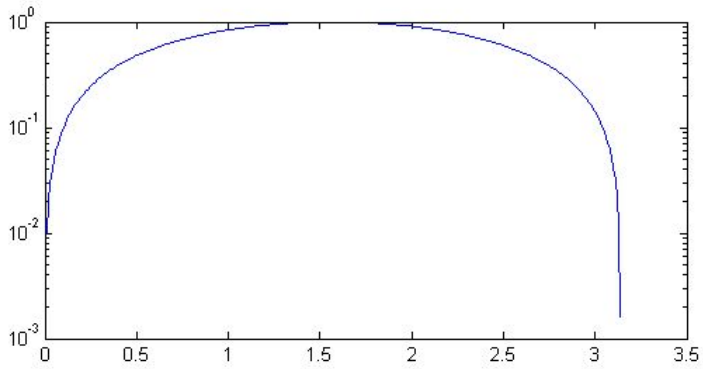
- У функции **plot** есть настройки, которые применяются к строящемуся графику
- Именно из-за них часто удобнее использовать **hold on**, чем много графиков в одном **plot**'е
- Эти настройки записываются в следующем формате:
  - >> plot(x, y, 'parameter1', value1, 'parameter2', value2, ...)
  - вы, возможно, уже знакомы с одним из них: 'MarkerSize'
- Прочие полезные параметры:
  - 'LineWidth' — толщина линии, задаётся числом
  - 'Color' — можно задавать как строкой (вроде 'blue' или 'red'), так и вектором из 3 элементов (RGB): [0.5 1 0.7]
  - 'MarkerSize' — размер маркера, задаётся числом
  - 'MarkerFaceColor' — цвет заливки маркера
  - 'MarkerEdgeColor' — цвет границы маркера

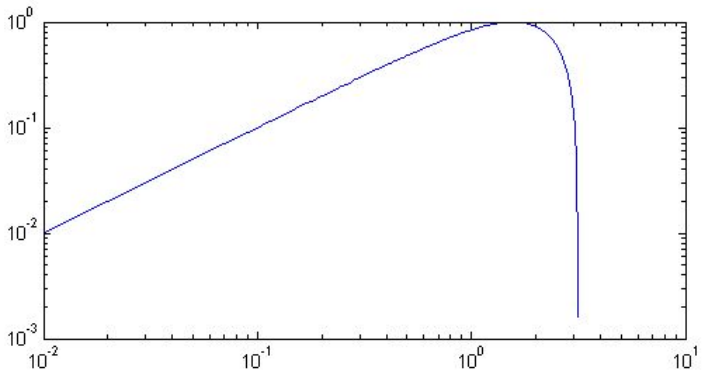
2D-графики с логарифмическими осями

- Рассмотрим основные функции для построения 2D-графиков
- Графики с логарифмическими осями (**semilogx**, **semilogy**, **loglog**):
  - `>> x = 0:0.01:2*pi;`
  - `>> subplot(2, 2, 1);`
  - `>> plot(x, sin(x)); % для сравнения`
  - `>> subplot(2, 2, 2);`
  - `>> semilogx(x, sin(x)); % логарифмический масштаб по x`
  - `>> subplot(2, 2, 3);`
  - `>> semilogy(x, sin(x)); % логарифмический масштаб по x`
  - `>> subplot(2, 2, 4);`
  - `>> loglog(x, sin(x)); % логарифмический масштаб и по x и по y`
  - результаты на следующем слайде





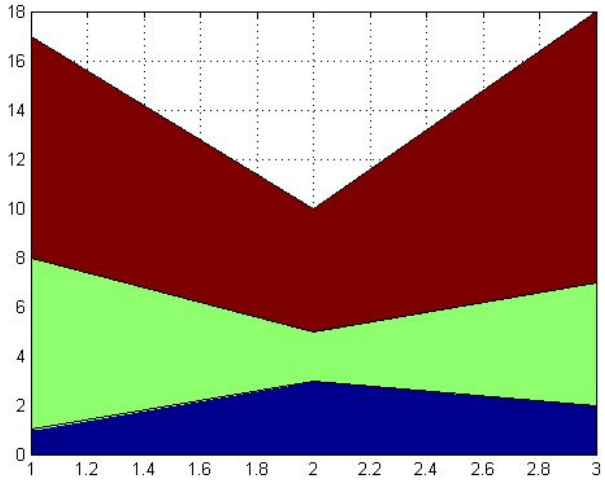


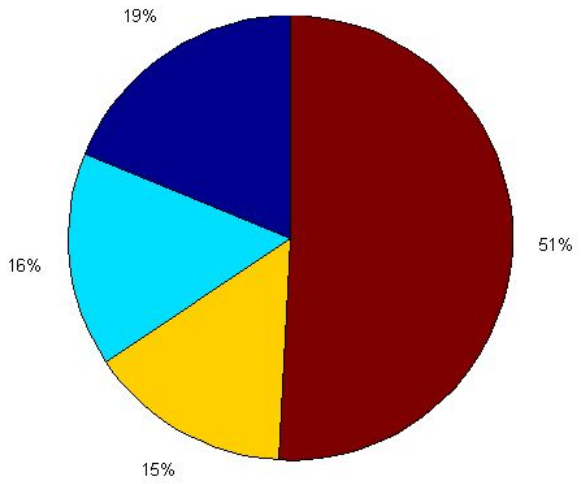


Результаты построения графиков с предыдущего слайда

2D-графики для построения диаграмм (I)

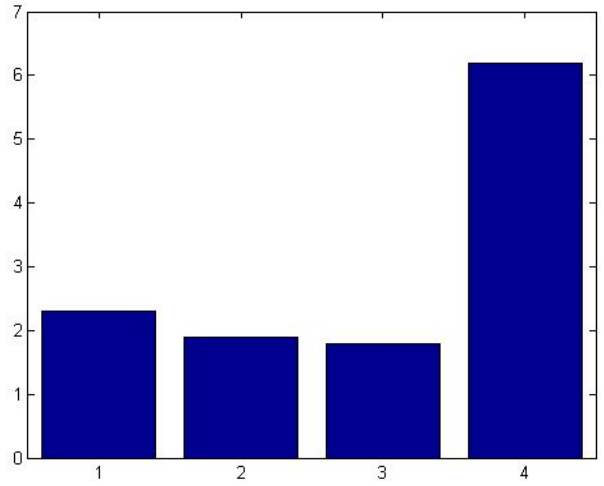
- Лучше всего построение пояснять на примерах.
- Рассмотрим функцию **area**:
  - `>> A = [1 7 9;`
    `3 2 5;`
    `2 5 11];`
  - `>> area(A);`
  - строит по столбцам
  - 1 столбец = 1 график
- Функция **pie**:
  - `>> V = [2.3 1.9 1.8 6.2];`
  - `>> pie(V);`
  - строит круговую диаграмму

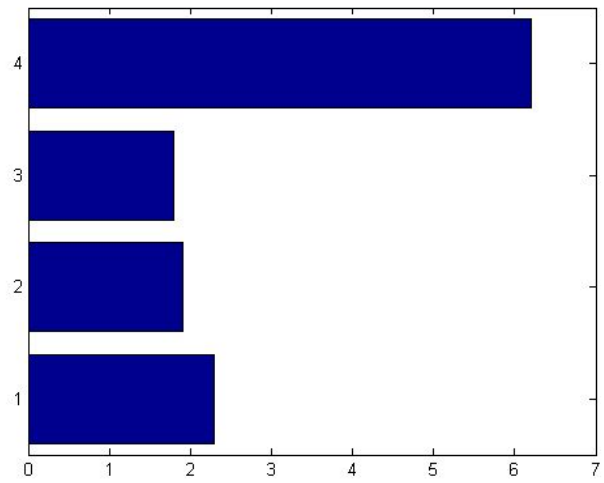




2D-графики для построения диаграмм (II)

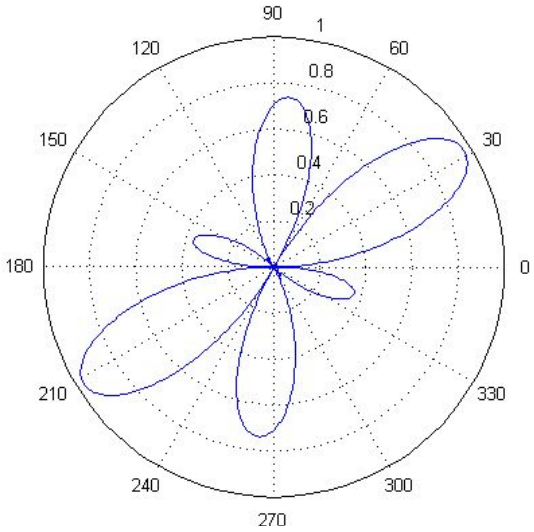
- Функция **bar**:
  - `>> V = [2.3 1.9 1.8 6.2];`
  - `>> bar(V);`
  - строит столбчатую диаграмму (вертикальную)
- Функция **barh**:
  - `>> barh(V);`
  - строит столбчатую диаграмму (горизонтальную)
- Ещё вариант:
  - >> bar(1:0.1:10, sin(1:0.1:10))
- У **bar** есть ещё пара способов использования, которые задаются строковым параметром: 'stack', 'group'

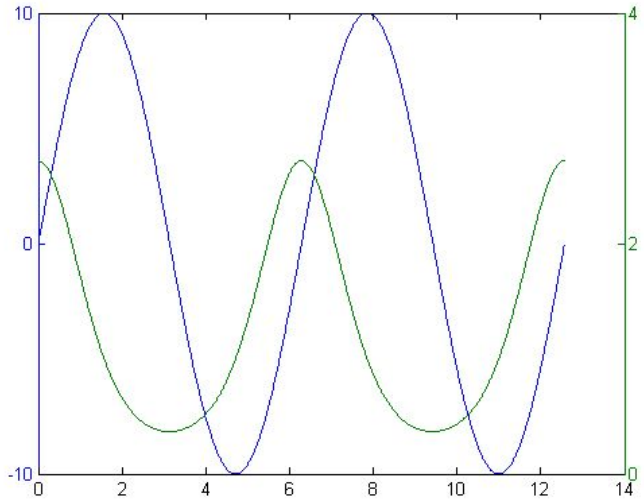




Специальные 2D-графики

- Эти графики не так часто нужны на практике, но помнить об их существовании стоит.
- График в полярных координатах **polar**:
  - `>> t = 0:0.01:2*pi;`
  - `>> polar(t, sin(t).*tan(t))`
- График с двумя осями Y (**plotyy**):
  - `>> x = 0:0.01:4*pi;`
  - `>> y1 = 10 * sin(x);`
  - `>> y2 = exp(cos(x));`
  - `>> plotyy(x, y1, x, y2, '--r');`





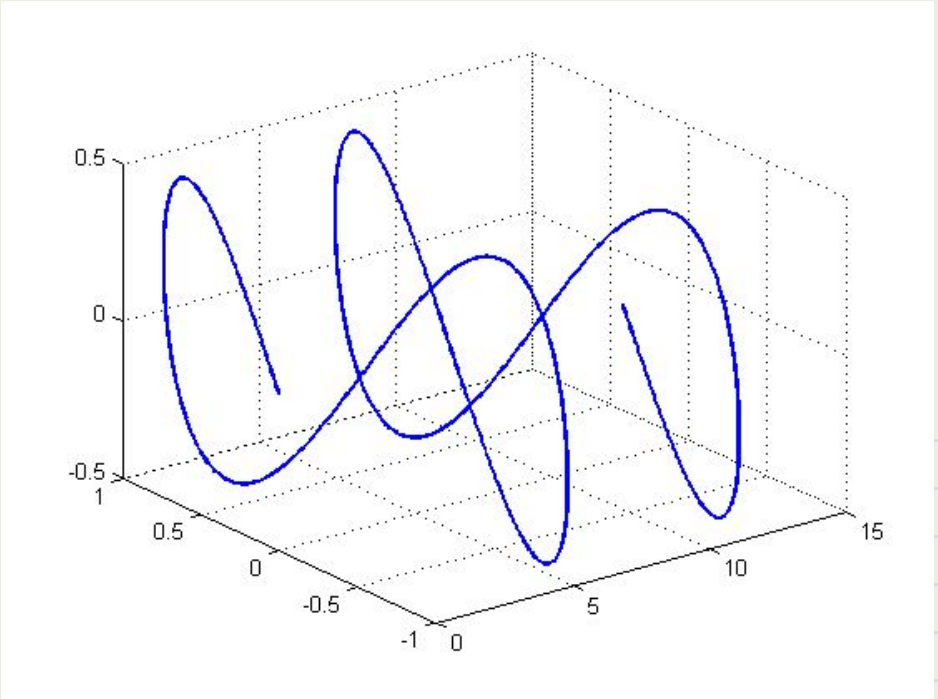
3D-графики

График линии в 3D

- Обычно такой график требуется построить, когда у вас есть параметрически заданная функция вроде

$$\begin{aligned} x(t) &= \sin(t) \\ y(t) &= \sin(t)\cos(t) \\ t &\in [0, 4\pi] \end{aligned}$$

- Тогда задаём `t`, `x` и `y` как обычно, а вот функцию используем **`plot3`**:
  - >> t = 0:0.01:4*pi;
  - >> x = sin(t);
  - >> y = sin(t) .* cos(t);
  - >> plot3(t, x, y);

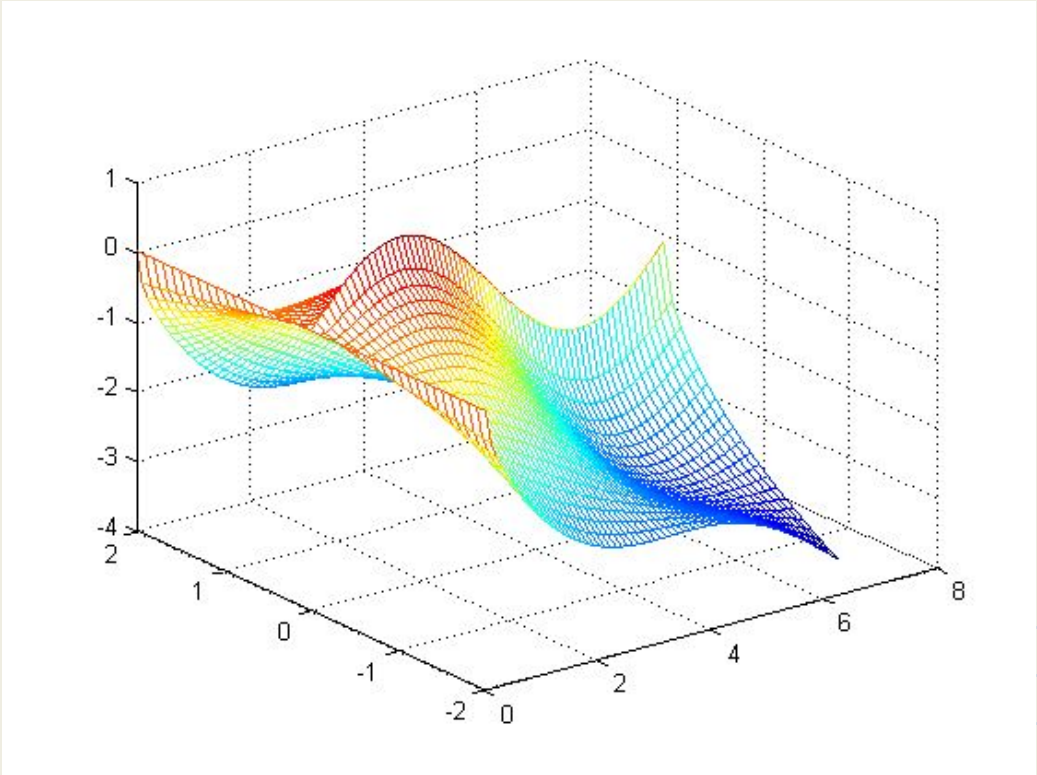


Подготовка к построению поверхности

- Что такое поверхность?
- Поверхность (если непрерывная) — это график функции $f(x, y)$, определённой в точках $x \in [x_{low}, x_{high}]$, $y \in [y_{low}, y_{high}]$
- То есть, чтобы построить график поверхности, нужно задать сетку всевозможных пар значений x и y.
- В MatLab это делается функцией **meshgrid**. Её аргументы — вектора, задающие диапазоны по x и по y:
  - `>> [X, Y] = meshgrid(0:0.1:2*pi, -2:0.1:2); % аргументы **не** обязательно одной длины!`
- Функция **meshgrid** возвращает две матрицы, которые используются для создания поверхности Z
  - `>> Z = sin(X) .* cos(Y) - sqrt(abs(X .* Y)); % это аналог z(x, y) = sin(x)cos(y) - sqrt(|xy|)`

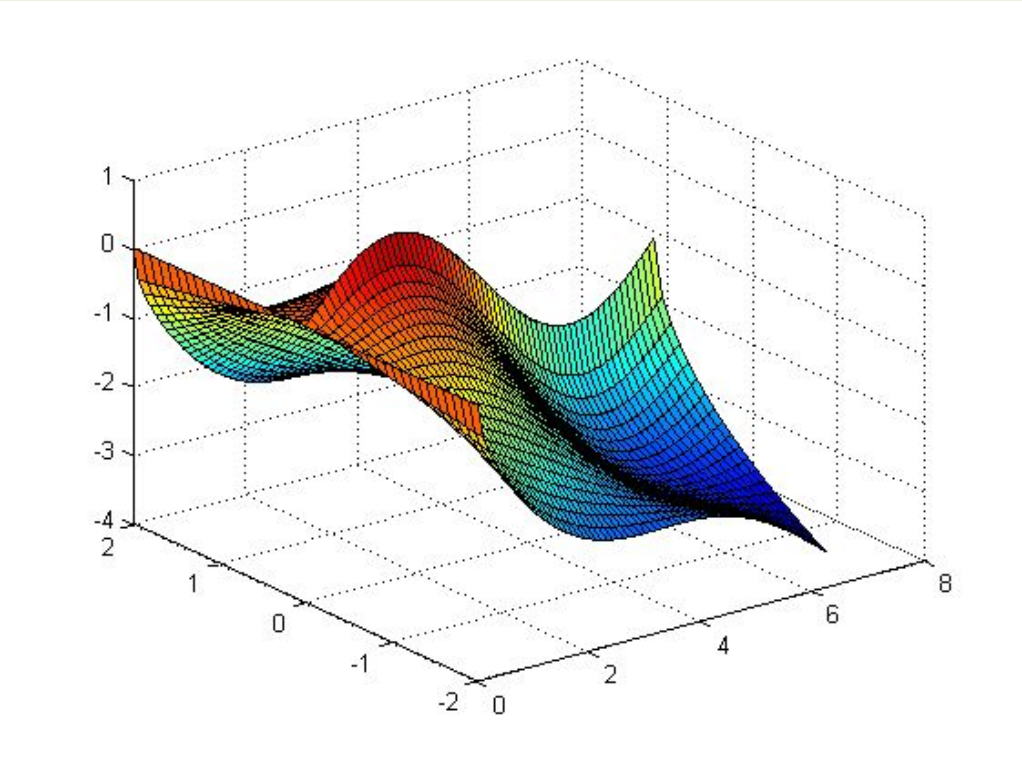
Функции построения поверхностей (I)

- После того, как все данные готовы, можно строить поверхность. Самой простой функцией является **mesh**, которая создаёт фасеточную поверхность:
  - `>> mesh(X, Y, Z);`



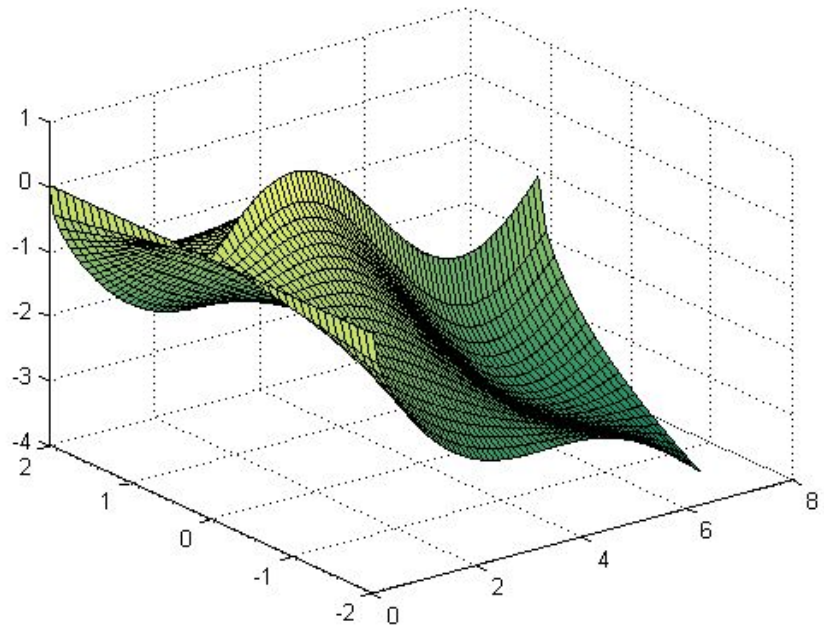
Функции построения поверхностей (II)

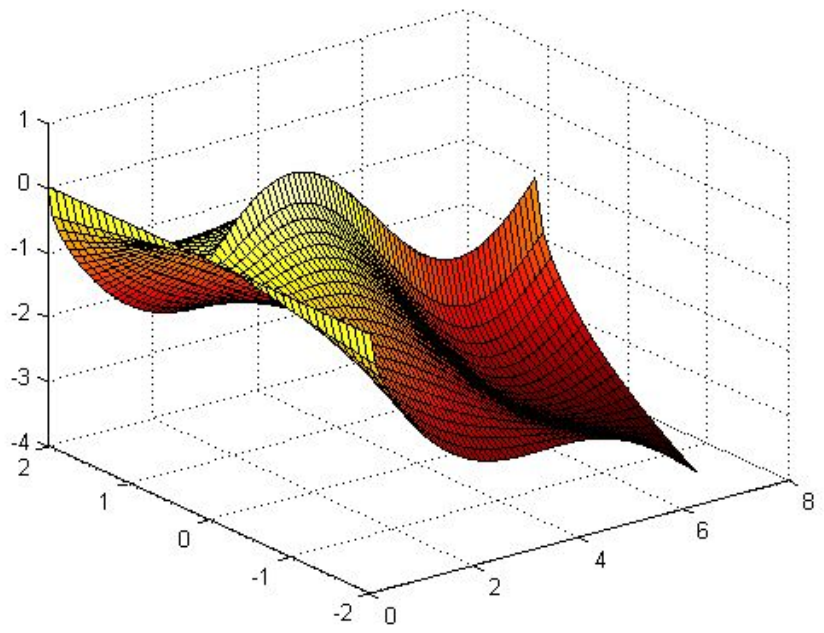
- Рассмотрим функцию **surf**. Она отличается тем, что промежутки между узлами «заливаются» цветом (и потому работает немного медленнее, чем **mesh**):
  - `>> surf(X, Y, Z);`



Раскраска поверхностей

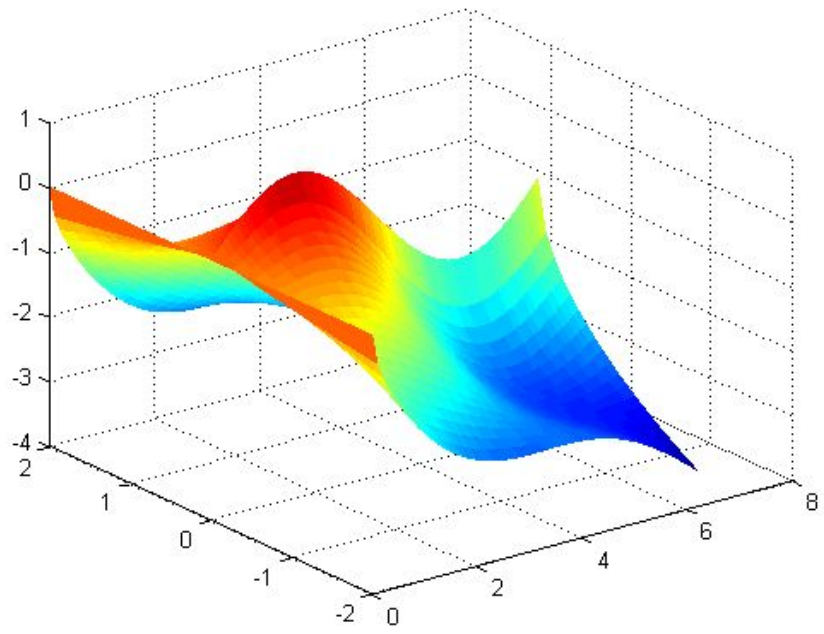
- Режим раскраски задаётся функцией **colormap**. Она работает с уже построенным графиком. Есть огромное количество встроенных вариантов, покажу лишь пару (для **surf**):
  - `>> colormap summer; % слева`
  - `>> colormap hot; % справа`

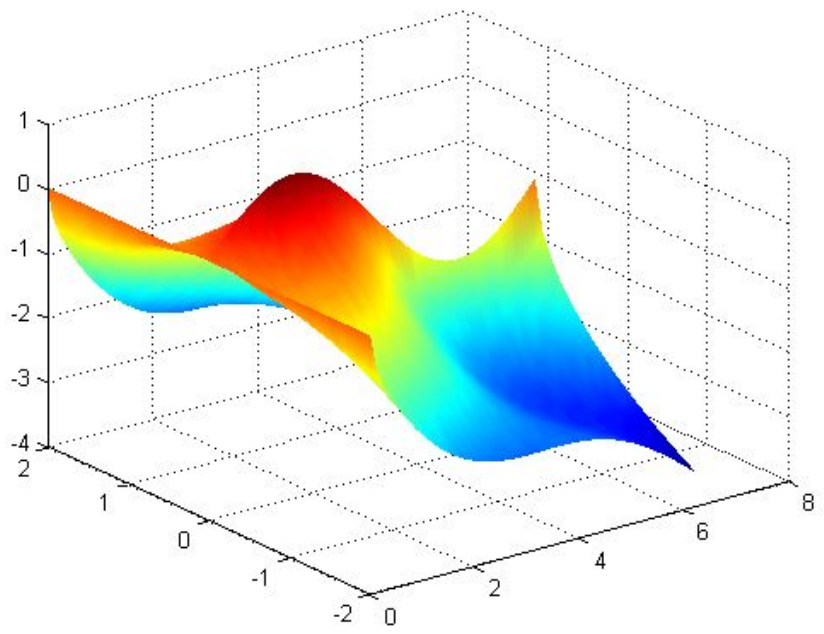




Переходы цвета на поверхности

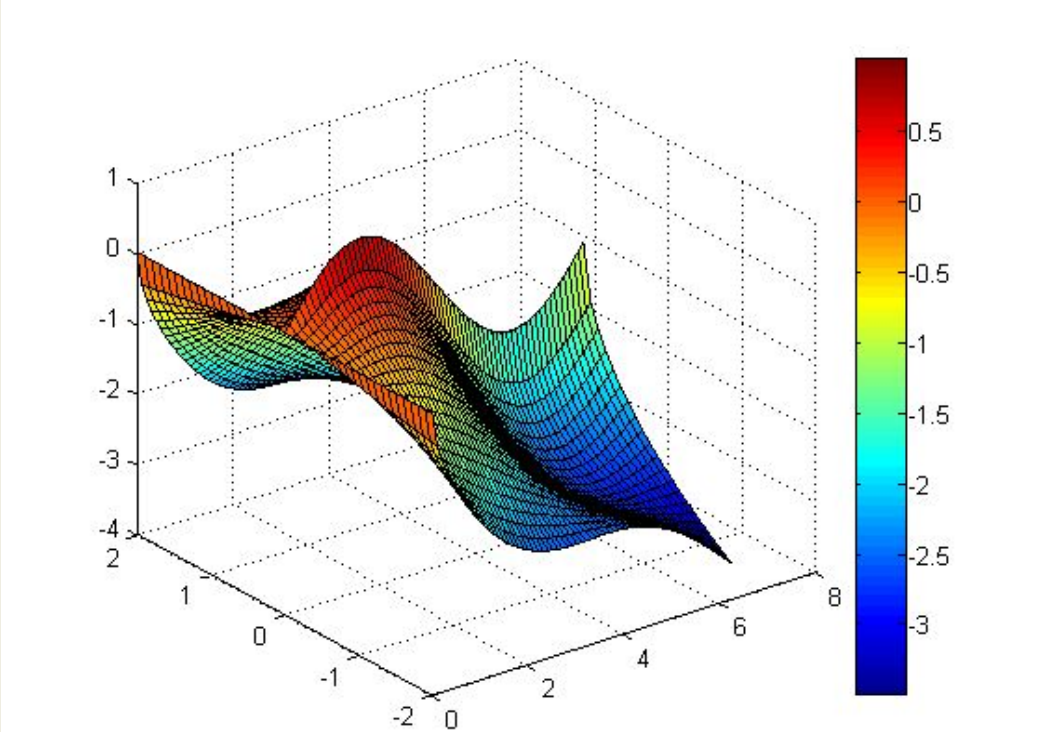
- Для настройки перехода используется функция **shading**.
  - `>> shading faceted; % используется по умолчанию, рисует сетку на поверхности`
  - `>> shading flat; % слева`
  - `>> shading interp; % справа`





Соответствие цвета числовому значению

- Цвета на поверхности расположены не просто так.
- Каждому оттенку соответствует какое-то число из области значений функции, которую мы строим
- Для отображения этого соответствия используется команда **`colorbar`**:
  - >> colorbar



Функции

«An expert is a person who has made all the mistakes that can be made in a very narrow field»

Нильс Бор

Функция в MatLab

- Функция в MatLab — это примерно то же, что и функции в других языках.
- Она получает набор аргументов, как-то их обрабатывает и выдаёт результат.
- В отличие от большинства языков программирования, в MatLab результат может быть не один
- Каждая функция должна быть в отдельном файле
- Исключения есть, но пока лучше придерживаться правила
- Имя функции должно совпадать с именем файла (**важно**!)
  - То есть для функции `myCos` файл должен иметь название `myCos.m`

Синтаксис функции

```
function [res1 res2] = funcname(arg1, arg2)
    код
end
```

- здесь `res1, ..., resnN` — значения, которые функция возвращает (их может быть много)
- `arg1, ..., arg2` — аргументы, «приходящие» в функцию
- функция завершается ключевым словом `end`
- Если функция ничего не возвращает, то не пишутся ни квадратные скобки, ни знак =
- Возвращаемым переменным в коде функции обязательно должно быть присвоено какое-то значение.
- Если у функции нет аргументов, то круглые скобки тоже не пишутся.

Пример простой функции

```
function sayHi
   disp('Hi!');
end
```

- Если эту функцию сохранить в файле sayHi.m, то её можно будет вызывать из ваших программ или командной строки:
  - `>> sayHi`
    - `Hi!`
- Эта функция не имеет аргументов и возвращаемых значений

Пример функции посложнее

```
function [K F] = c2kf(C)
   K = C + 273.15;
   F = C * 9/5 + 32;
end
```

- Эта функция — c2kf — переводит градусы по шкале Цельсия в градусы по Кельвину и по Фаренгейту
- Она возвращает два значения
- Её можно вызвать, например, в командной строке:
  - `>> c2kf(40)`
- Более того, она может работать с векторами и матрицами!
  - `>> c2kf([30 40 80 100])`

Однострочные функции

- Однострочные функции **не надо** задавать в отдельном файле.
- Однострочная функция задаётся с помощью **`inline`**:
  - `>> f1 = inline('x.^2+4');`
  - аргументы определяются автоматически (происходит поиск строчных символов — кроме i, j)
- Можно задавать аргументы и отдельным списком:
  - `>> f2 = inline('sin(x1)+cos(x2)', 'x1', 'x2')`
  - все аргументы **`inline`** — строки
- После этого обращаться к однострочной функции можно так же, как и к обычной:
  - `>> a = f1(3);`
  - `>> b = f2(pi, pi/2);`

Анонимные функции

- Анонимные функции во многих языках называются лямбда-функциями
- Они очень похожи на однострочные
- Синтаксис:

  имя_функции = @(аргументы)тело_функции
- Пример:
  - >> f3 = @(x) sin(x) + sqrt(x);
- Удобно использовать для построения графиков:
  - `>> t = 1:0.1:10;`
  - `>> plot(t, f3(t));`
- ...и не только
- Всю мощь лямбда-функций, к сожалению, в MatLab понять не получится =(

Итоги лекции

- В MatLab очень легко можно манипулировать содержимым матриц
- Графики гибко настраиваются
- А их разновидностей очень много
- Включая трёхмерные
- Ещё можно создавать свои функции
- Причём как большие и сложные, так и короткие для сиюминутного использования
- Спасибо за внимание =)

Контрольные вопросы

- A = [3 5; 7 9];
  A(:) = ?
- f = @(x, y) 2*x + y;
  x = 2;
  y = 4;
  f(y, x) = ?
- plot(1:10, 1:10); hold on;
  plot([2 4], [2 5]); hold off;
  plot(1:3, 6:8);
  Сколько построится графиков?