Конспект лекций

По дисциплине

«Информационные технологии»

Москва 2020

Оглавление

[Введение. 3](#_Toc92794136)

[Программная среда MATLAB 3](#_Toc92794137)

[Введение в MATLAB 3](#_Toc92794138)

[Язык MATLAB (m-язык) 8](#_Toc92794139)

[Структурирование m-программ 14](#_Toc92794140)

[Графический интерфейс – методика построения 20](#_Toc92794141)

[Основные Toolboxes 25](#_Toc92794142)

[Основные свойства SIMULINK 27](#_Toc92794143)

[Программная среда R 29](#_Toc92794144)

[Введение в R 29](#_Toc92794145)

[Среда R 30](#_Toc92794146)

[R и статистики 31](#_Toc92794147)

[Архитектура среды R 32](#_Toc92794148)

[Язык R 32](#_Toc92794149)

[Векторы и операции с ними. 35](#_Toc92794150)

[Факторы. 37](#_Toc92794151)

[Массивы и матрицы. 38](#_Toc92794152)

[Программы на R 40](#_Toc92794153)

[Сравнение R и MATLAB. 42](#_Toc92794154)

[Новые информационные технологии 43](#_Toc92794155)

[Задачи, решаемые с помощью информационных технологий. 44](#_Toc92794156)

[Классы программных систем 44](#_Toc92794157)

[Задача импортозамещения 45](#_Toc92794158)

[Развитие информационных технологий 48](#_Toc92794159)

[Облачные технологии 51](#_Toc92794160)

[Большие данные. 53](#_Toc92794161)

[Интернет вещей и промышленный интернет вещей (IoT и IIoT) 55](#_Toc92794162)

[Искусственный интеллект 59](#_Toc92794163)

# Введение.

Структура дисциплины: 16 лекций, 16 лаб.занятий.

В лекциях будет дан обзор важных для нашей специальности информационных технологий. 3 важные составные части:

a) основная информация по современным ИТ;

b) изучение ПС MATLAB;

c) изучение ПС R.

В лабораторных занятиях изучается работа с MATLAB и R.

В угоду лабораторным занятиям сначала займемся второй и третьей частью.

# Программная среда MATLAB

## Введение в MATLAB

1. **Изучение MATLAB – почему**?

Рассмотрим задачу автоматизации управления. Для ее решения надо: исследовать объект управления, смоделировать его, изучить качество управления при разных вариантах систем управления – все эти задачи поддерживаются MATLAB. Хотелось бы получить результаты при возможно меньших затратах труда. Цель – не программа, а результат ее выполнения! Эффективность = полезность результата деятельности / затраты на получение результата. Как будет ясно из дальнейшего, MATLAB является весьма эффективным средством решения задач в предметной области создания и исследования систем управления.

1. **Назначение и история MATLAB**.

MATLAB – это высокопроизводительный язык для технических расчётов. Он объединяет вычисления, визуализацию и программирование в простом для использования инструменте, в котором задачи и решения выражаются в схожих математических нотациях.

MATLAB – это интерактивная система, чьим базовым элементом является массив, не требующий определения размерности. Это позволяет решать многие технические вычислительные проблемы, особенно связанные с формализацией векторов и матриц, за время составляющее малые доли от потребного при написании программ на скалярных неинтерактивных языках таких, как С или Фортран.

MATLAB отлично отработан за длительный период применения многими пользователями. В университетских структурах он стал стандартным учебным средством для изучения вводных и продвинутых курсов в математических, инженерных и научных дисциплинах. В промышленности MATLAB стал инструментом высокопродуктивных исследований, разработок и анализа.

MATLAB характеризуется семейством добавочных проблемно-ориентированных программных средств, называемых Toolboxes. Что очень важно для многих пользователей MATLAB, toolboxes позволяют изучать и применять специализированные технологии. Toolboxes представляют собой специфический набор функций MATLAB (М-файлов), расширяющих возможности MATLAB на решение частных классов задач. К областям, для которых имеются toolboxes относятся: обработка сигналов, системы управления, нейронные сети, нечёткая логика, вейвлеты, моделирование и многие другие.

MATLAB как язык программирования был разработан Кливом Моулером (англ. *Cleve Moler*) в конце 1970-х годов, когда он был деканом факультета компьютерных наук в Университете Нью-Мексико. Целью разработки служила задача дать студентам факультета возможность использования программных библиотек Linpack и EISPACK без необходимости изучения Фортрана. Вскоре новый язык распространился среди других университетов и был с большим интересом встречен учёными, работающими в области прикладной математики. До сих пор в Интернете можно найти версию 1982 года, написанную на Фортране, распространяемую с открытым исходным кодом. Инженер Джэк Литтл (англ. *Jack N. Little*) познакомился с этим языком во время визита Клива Моулера в Стэндфордский университет в 1983 году. Поняв, что новый язык обладает большим коммерческим потенциалом, он объединился с Кливом Моулером. Совместными усилиями они переписали MATLAB на C и основали (вместе со Стивом Бангертом) в 1984 компанию The MathWorks для дальнейшего развития. Эти переписанные на С библиотеки долгое время были известны под именем JACKPAC. Первоначально MATLAB предназначался для проектирования систем управления (основная специальность Джека Литтла), но быстро завоевал популярность во многих других научных и инженерных областях. Он также широко использовался и в образовании, в частности, для преподавания линейной алгебры и численных методов.

Компания **The MathWorks** специализируется на разработке программного обеспечения в области численных вычислений и компьютерного моделирования. Создана в 1984 году Джеком Литтлом (Jack Little), Кливом Моулером (Cleve Moler). Они же являются владельцами компании.



Jack Little, President

Jack Little is president and cofounder of MathWorks. He was a coauthor and principal architect of early versions of the company's flagship MATLAB product as well as Signal Processing Toolbox and Control System Toolbox.

Jack holds a B.S. degree in electrical engineering and computer science from MIT (1978) and an M.S.E.E. degree from Stanford University (1980).

Cleve Moler, Chief Mathematician

Cleve Moler is chief mathematician, chairman, and cofounder of MathWorks. Moler was a professor of math and computer science for almost 20 years at the University of Michigan, Stanford University, and the University of New Mexico.



He spent five years with two computer hardware manufacturers, the Intel Hypercube organization and Ardent Computer, before joining MathWorks full-time in 1989.

In addition to being the author of the first version of MATLAB, Moler is one of the authors of the LINPACK and EISPACK scientific subroutine libraries. He is coauthor of three traditional textbooks on numerical methods and author of two online books, [Numerical Computing with MATLAB](javascript:openWindow('/moler/index_ncm.html','750','650','scrollbars=yes,menubar=no,toolbar=no,location=no,status=no,resizable=yes')) and [Experiments with MATLAB](javascript:openWindow('/moler/exm/index.html','750','650','scrollbars=yes,menubar=no,toolbar=no,location=no,status=no,resizable=yes')).

В настоящее время численность сотрудников MathWorks составляет более 3000 человек, из которых примерно 40% непосредственно заняты разработкой программных продуктов. Главный офис компании расположен в г. Нэтик, Массачусетс (Natick, Massachusetts), США. Джек Литтл занимает пост президента, Клив Моулер — ведущего специалиста.

Основные продукты The MathWorks: MATLAB, Simulink, Stateflow и специализированные инструментальные пакеты расширения к ним (ToolBoxes).

Версии: 1 – 1984, 7 – 2004, 8 - 2012

### Toolboxes (>40)

Сейчас возможности системы значительно превосходят возможности первоначальной версии матричной лаборатории Matrix Laboratory. Нынешний MATLAB - это высокоэффективный язык инженерных и научных вычислений. Он поддерживает математические вычисления, визуализацию научной графики и программирование с использованием легко осваиваемого операционного окружения, когда задачи и их решения могут быть представлены в нотации, близкой к математической. Наиболее известные области применения системы MATLAB:

* **математика и вычисления;**
* **разработка алгоритмов;**
* **вычислительный эксперимент, имитационное моделирование, макетирование;**
* **анализ данных, исследование и визуализация результатов;**
* **научная и инженерная графика;**
* разработка приложений, включая графический интерфейс пользователя.

MATLAB - это интерактивная система, основным объектом которой является массив, для которого не требуется указывать размерность явно. Это позволяет решать многие вычислительные задачи, связанные с векторно-матричными формулировками, существенно сокращая время, которое понадобилось бы для программирования на скалярных языках типа C или FORTRAN.

Фирма The MathWorks, Inc. поддерживает тесные связи с университетским миром и предлагает для образовательных версий значительные скидки. В настоящее время студенческая версия Student Edition of MATLAB ничем не отличается от коммерческой версии, но имеет невысокую цену и предназначена для студентов, работающих на персональном компьютере дома или в общежитии.

Одно из назначений математики - служить языком общения между учеными и инженерами. Матрицы, дифференциальные уравнения, массивы данных, графики - это общие объекты и конструкции, используемые как в прикладной математике, так и в системе MATLAB. Именно эта фундаментальная основа обеспечивает системе MATLAB непревзойденную мощь и доступность. Стоит прислушаться к следующему афористичному мнению: "Причина, по которой MATLAB столь полезен для обработки сигналов, состоит в том, что он не проектировался специально для этой цели, а создавался для математиков".

Система MATLAB - это одновременно и операционная среда и язык программирования. Одна из наиболее сильных сторон системы состоит в том, что на языке MATLAB могут быть написаны программы для многократного использования. Пользователь может сам написать специализированные функции и программы, которые оформляются в виде М-файлов. По мере увеличения количества созданных программ возникают проблемы их классификации и тогда можно попытаться собрать родственные функции в специальные папки. Это приводит к концепции пакетов прикладных программ (ППП), которые представляют собой коллекции М-файлов для решения определенной задачи или проблемы.

В действительности ППП - это нечто большее, чем просто набор полезных функций. Часто это результат работы многих исследователей по всему миру, которые объединяются в зависимости от области применения - теория управления, обработка сигналов, идентификация и т. п. Именно поэтому пакеты прикладных программ - MATLAB Application Toolboxes, входящие в состав семейства продуктов MATLAB, позволяют находиться на уровне самых современных мировых достижений.

За более 40 лет существования MATLAB прошел большой путь развития. Как это принято в коммерческой практике современных софтверных фирм, The MathWorks, Inc. в среднем раз в 4 года выпускала новую версию программ с расширенными возможностями. Помимо версии каждая программа характеризуется также номером релиза. Последняя такая новая версия с номером 8 вышла в сентябре 2012 года (релиз 2012b). Начиная с 2008 года фирма-производитель MATLAB выпускает ежегодно два новых релиза своего продукта. Например, релизы 2011 года имеют обозначения 2011a и 2011b. Поэтому об актуальности имеющейся в распоряжении версии программного средства можно судить уже по его названию, например, MATLAB 2017b.

1. **Уровни работы с MATLAB**

Возможность решения задач на MATLAB на следующих уровнях:

1) решение задачи без программирования с использованием готовых программ (Toolboxes) – для «непрограммирующих» пользователей;

2) решение задач с помощью команд (последовательные пробы и ошибки, интерактивный режим) – пользователи с небольшими знаниями программирования;

3) решение задач с помощью программ, сделанных «для себя» - программисты невысокой квалификации;

4) решение задач с помощью созданных программных продуктов с графическим интерфейсом – достаточно квалифицированные программисты;

5) создание объектов для использования в программах, написанных на универсальных языках программирования – С, Fortran – «продвинутые» программисты;

6) решение задач во взаимодействии с другими программами (Excel) - «продвинутые» программисты;

7) Компиляция программ и создание загрузочных (автономных) программ - «продвинутые» программисты.

Простота и естественность создания собственного Toolbox при работе с MATLAB – за счет накопления программных модулей и готовых программ.

Пять составных частей системы MATLAB

Система MATLAB состоит из пяти основных частей:

* Инструмент разработки. Он включает множество средств и возможностей, которые позволяют вам использовать функции и файлы MATLAB. Многие из этих средств снабжены графическим интерфейсом пользователя. Он включает оболочку MATLAB, командное окно, историю команд, редактор и отладчик, браузеры для просмотра справок, рабочей области, файлов, путей доступа.
* Библиотека математических функций MATLAB. Это – огромная коллекция вычислительных алгоритмов, начиная от элементарных функций вроде суммирования, Sin, Cos и комплексной арифметики, до более сложных функций таких, как обращение матриц, собственные значения матриц, функции Бесселя и быстрое преобразование Фурье.
* Язык MATLAB. Это – высокоуровневый матричный язык с операторами управления потоками, функциями, структурами данных, операторами ввода/вывода, элементами объектно-ориентированного программирования. Он допускает как «программирование в малом» для быстрой разработки черновых «одноразовых» программ, так и «программирование в большом» для создания больших и сложных программных приложений.
* Достаточно высококачественные графические средства.
* Интерфейс прикладных программ (API) MATLAB.

1. Архитектура системы.

На рис.1 представлена архитектура MATLAB. Ядро – это основная программа, загружаемая в память компьютера после обращения к программе. Его основные функции:

- управление работой MATLAB в процессе решения задач пользователя;

- обеспечение взаимодействия с пользователем;

- обеспечение загрузки в память остальных компонент по мере возникновения необходимости в их исполнении;

- контроль содержания рабочей области - части оперативной памяти, выделенной для хранения данных, используемых при решении прикладных задач;

- обеспечение возможности настройки MATLAB по усмотрению пользователя

Среди подгружаемых компонент MATLAB:

* Библиотеки программных модулей LAPACK и BLAS для выполнения математических операций и матричных вычислений.
* Текстовый и графический редакторы для создания и изменения программ и графического интерфейса пользователя.
* Совокупность проблемно-ориентированных пакетов программ (ToolBox) для решения разнообразных прикладных задач.
* Средства экспорта-импорта данных, представленных в форматах, принятых в некоторых других программных средствах: электронных таблиц XLS, CSV, WK1, графических файлов JPEG, BMP, PCX и др.
* Развитая справочная подсистема, включающая возможность получения справок через Интернет и большое число демонстрационных задач.

Ядро MATLAB

LAPACK

BLAS

Текстовый редактор

Графический редактор

Редактор GUIDE

Справочная подсистема

ToolBoxes

Средства импорта - экспорта

Рис.1

**6. Размещение MATLAB в оперативной памяти**

При своем функционировании MATLAB использует ресурсы оперативной и внешней памяти компьютера. На рис.2 схематично представлено размещение MATLAB в оперативной памяти компьютера (центральный прямоугольник на рисунке). В левой части рисунка показаны основные окна интерфейса пользователя MATLAB: главное окно с главным системным меню, командное окно (Command Window), окно истории команд (Command History), окно рабочей памяти (Workspace), окно текущего каталога (Current Directory). Обслуживание этих окон осуществляют соответствующие программы, входящие в состав ядра MATLAB. Кроме того, в нем же присутствуют:

- командный процессор, осуществляющий ввод и синтаксический контроль команд пользователя;

- программный процессор, осуществляющий ввод программ, написанных на m-языке в память компьютера и передачу их на исполнение;

- интерпретатор команд, осуществляющий семантический контроль команд и их исполнение.

В оперативной памяти также присутствуют области для хранения вспомогательных данных, используемых программами ядра MATLAB и рабочая область, в которой хранятся все данные, создаваемые или загружаемые в процессе исполнения команд пользователя или программ.

В оперативную память, по мере необходимости, также могут загружаться другие компоненты MATLAB: редакторы, проблемно-ориентированные пакеты программ.

Главная управляющая программа

Командный процессор

Программный процессор

Интерпретатор

\*.m \*.mat

Текстовый редактор

Графический редактор

Рабочая память данных

(Workspace)

Память программ

Редактор GUIDE

ToolBoxes

Рис.2

## Язык MATLAB (m-язык)

**Общие сведения о m-языке**

Элементами в m-языке, применяемом для управления вычислительным процессом в MATLAB, служат константы, переменные, функции, команды и управляющие конструкции. Эти элементы, возможно, в различных соединениях с помощью специальных соединительных знаков, используются как в командной строке, так и в программах. Структура языка представлена на рис.1.

m-язык MATLAB

Операторы

Управляющие конструкции

Команды

Выражения

Константы

Переменные

Функции

Соединительные знаки

Присваивание

Циклические

Условные

if

switch

for

while

Пользовательские

Системные

MATLAB

Система

Пользовательские

Системные

global

persistent

Рис.1

Переменные и функции – именованные. Правило имен: сочетание символов латинского алфавита, цифр и подчерков. Начинаться с буквы. Регистрочувствительные!

**Операторы** языка – это специальные конструкции, с помощью которых осуществляется управление ресурсами компьютера в ходе решения задач. Оператор в m-языке может представлять собой **команду** или **присваивание**. Операторы MATLAB могут выполняться как в командном окне, так и в составе m-программ.

В конце оператора может ничего не стоять, а могут заканчиваться «,» или «;» (есть различие – о нем дальше). В одной строке можно записать несколько операторов – тогда нужны разделители «,» или «;». Если оператор – длинный и не помещается на строке – можно разорвать (только нельзя разрывать имена) и поставить в конце строки многоточие «…» - после этого продолжить оператор на следующей строке.

Оператор присваивания – явный и скрытый.

Явный – со знаком присваивания «=»: ff=[1 2 3;4 5 6]

Скрытый: size(ff) – результат записывается в стандартную переменную ans

**Типы переменных в m-языке MATLAB и как задаётся тип переменной**

Матрица

Логическая

Символьная/ строчная

Числовая

Ячейки

Структура

int, fixed point

float: single, double

complex

Дата/ Время

Рис.3

**Logical** – матрица, заполненная true или false.

**Char/String** – символьный массив.

**Int** – целое со знаком или uint – целое без знака. Например, uint16 = от 0 до 65000.

int8 – от -27 до 27-1, uint8 – от 0 до 28-1

Для сохранения числа 325 в переменной типа int16 используется оператор x=int16(325).

**Fixed point** – принципиально не отличается в MATLAB от int – все числа одинаковые (нет integer и real)

**Комплексная** - -5.12+4.6i - функция complex(a,b)

**Date/Time** – функции datetime() – возвращает текущие дату и время, now(), today() – возвращают число дней от Рождества Христова, datestr() и datenum() – отсчет временных интервалов…

**Float** – single, double

**Single (double)** - одинарная или двойная точность. Single – 3\*10^38, Double – 3\*10^300. Тип double – по умолчанию для чисел. Задание: x=single(25.783) или y=double(5.89e+011).

**Cell** – массив ячеек, каждая из которых может содержать свой массив, причем они могут быть разных размеров и типов.

Пример.

>>w{1,1}=ones(3);

>>w{1,2}='Fomin';

>>w{1,3}=16;

>>w{2,1}=diag(rand(1,4));

>>w{2,2}='Moscow';

>>w{2,3}=pi;

>> b=w{2}(3,3)

b =

0.6068

**Structure** – массив структуры, подобно структурам в языке С. Например, a.day = 12; a.color = 'Red'; a.matr = rand(3); Элементы структуры могут иметь разные типы и размеры.

Пример.

>> stud.name='Иванов И.'

>> stud.gruppa='А-01-16'

>> stud.sessija=[5 4 4 5]

>> stud.birth='12/05/1999'

>> stud

stud =

name: 'Иванов И.'

gruppa: 'А-01-16'

sessija: [5 4 4 5]

birth: '12/05/1999'

>> stud.sessija(3)

ans =

4

Тип переменной задается при ее создании (оператор присваивания). Может меняться при работе программы, например,

f=[12.3, -56]’ % числовой вектор-столбец

…….

f=’Stroka’ %Символьный вектор-строка

….

А можно так:

>>> hh=[] # Объявлена переменная, есть место в раб.обл., но тип пока неизвестен.

Так делают, например, при последующем наращивании матрицы в цикле.

**Локализация областей действия переменных**

Начинает действовать с момента определения – например, в операторе присваивания. Заканчивает либо при удалении с помощью команды clear, либо по завершении той части программы, в которой определена (например, при завершении функции).

**Преобразование типов**

Самый общий тип – символьный. Поэтому – преобразования возможны в символьный тип и из него.

Функции преобразования: num2str(), str2num(), int2str(), mat2str(), cellstr(), double(), single()

>> ww=[1 2 3;4 5 6]

>> cc=mat2str(ww)

cc =

'[1 2 3;4 5 6]'

**Из каких элементов могут быть построены выражения в m-языке?**

Выражение=[константы, переменные, функции, знаки операций, скобки]

1.23\*(A’\*B)/(C-sin(2\*pi/T))./D

**Способы программирования ввода-вывода с файлами.**

Операторы для работы с файлами – важная часть любого языка программирования. Файлы – двух типов: символьные и бинарные. Могут быть еще другие: электронных таблиц, баз данных, графические и др., но работа с ними требует специальных средств.

А)***. Ввод- вывод данных***

а) Вывод на экран: disp(<матрица или выражение>), fprintf(<формат>,<список вывода>)

б) ввод с клавиатуры: input(<запрос>)

в) вывод в МАТ-файл: save()-функция, save-команда

Если save без контекста – содержимое раб.памяти сохраняется в файле matlab.mat

г) ввод из МАТ-файла: load()-функция, load – команда

И опять: если load без контекста – загрузка из файла matlab.mat (удобно при возобновлении сеанса связи)

load filename % все переменные из файла

load filename obj1 obj2 ... % указанные переменные

load fname a\* % Load variables starting with "a"

out = load('filename','obj1','obj2',...) % функция с записью в структуру(out)

Пример

>> a=9

>> b=[1 2 3]

>> c=[11;12;13;14]

>> save DDD a b c

>> gg=load('DDD.mat','a','c')

>> gg.a

ans =

9

>> gg.c

ans =

11

12

13

14

д) вывод в ТХТ-файл: save()-функция, save-команда, dlmwrite(), fprintf()

е) ввод из ТХТ-файла: load()-функция, load – команда, dlmread(), fscanf(), textscan()

Ограничение: load из текстового файла может загрузить только 1 регулярную матрицу!

**В) Форматный ввод – функция fscanf**

**<Файловая переменная> = fopen(<имя файла>[,<доступ к файлу>])**

Имя файла – символьная строка в апострофах.

Доступ к файлу – один-два символа в апострофах: ‘r’ (чтение), ‘w’ (запись с созданием файла, если необходимо), 'a' дополнение (создание файла, если необходимо)

Если доступ не указан – по умолчанию ‘r’.

**[<Переменная для ввода>,<Число фактически введенных данных>] = fscanf(<файловая переменная >,’<Формат ввода>’[,<Число вводимых чисел>])**

**[A,COUNT] = FSCANF(FID,FORMAT[,SIZE])**

COUNT – число прочитанных элементов, помещенных в матрицу А.

FID – файловая переменная, FORMAT – формат чтения из файла,SIZE – сколько элементов надо прочитать (по умолчанию – все).

Формат – строка символов, включающих элементы формата.

Элемент формата: ‘%[размер]<тип>’

Размер – число знакомест. Если не указан – определяется по умолчанию. Тип информации определяется символом: d (decimal), i(integer), f (float), e (exponential), g (general), c (character), s (string), есть еще другие форматы (o,x,u).

Valid conversion characters are

%c - Sequence of characters; number specified by field width

%d - Decimal numbers

%e, %f, %g -Floating-point numbers

%i - Signed integer

%o - Signed octal integer

%s - A series of non-white-space characters

%u - Signed decimal integer

%x - Signed hexadecimal integer

[...] - Sequence of characters (scanlist)

Примеры:

S = fscanf(fid,'%s') читает строку символов

A = fscanf(fid,'%5d') читает десятичные целые числа с 5 знаками

**fclose(fid)** – закрытие файла, связанного с файловой переменной fid.

Пример

Пусть те же 3 переменные: a, b, c записаны в текстовый файл

>> save DDT.txt a b c –ascii

Содержимое файла:

9.0000000e+00

1.0000000e+00 2.0000000e+00 3.0000000e+00

1.1000000e+01

1.2000000e+01

1.3000000e+01

1.4000000e+01

Прочитаем из файла с использованием fscanf с записью значений в новые 3 переменные: vv1, vv2, vv3

>> fp=open('DDT.txt')

fp =

3

>> [vv1,nn]=fscanf(fp,'%e',1)

vv1 =

9

nn =

1

>> [vv2,nn]=fscanf(fp,'%e',3)

vv2 =

1

2

3

nn =

3

>> [vv3,nn]=fscanf(fp,'%e')

vv3 =

11

12

13

14

nn =

4

>> fclose(fp)

ans =

0

А можно и все кучей в одну переменную:

>> fp=fopen('DDT.txt')

fp =

3

>> [vvv,nn]=fscanf(fp,'%e')

vvv =

9

1

2

3

11

12

13

14

nn =

8

>> fclose(fp)

ans =

0

***Вывод данных в файл***

А) **Вывод совокупности матриц – команда и функция save**

Команда

**save '<имя файла>' X** **Y …** – сохранение переменных X,Y,… в бинарном виде в файле с именем <имя файла>.MAT.

**save '<имя файла>' X –ASCII** - запись в текстовый файл с заданным именем <имя файла> содержимого переменной Х с одинарной точностью.

**save '<имя файла>' X –ASCII** **-DOUBLE** - запись в текстовый файл с заданным именем <имя файла> содержимого переменной Х с двойной точностью.

Функция

**save(‘<имя файла>’,’<матрица1>’, ’<матрица2>’,…, ’<матрицаN>’,’-<формат данных>’)**

Примеры

>> X=rand(5)

X =

0.9501 0.7621 0.6154 0.4057 0.0579

0.2311 0.4565 0.7919 0.9355 0.3529

0.6068 0.0185 0.9218 0.9169 0.8132

0.4860 0.8214 0.7382 0.4103 0.0099

0.8913 0.4447 0.1763 0.8936 0.1389

>> Y=randn(7,1)

Y =

-0.4326

-1.6656

0.1253

0.2877

-1.1465

1.1909

1.1892

>> save('Proba','X','Y') %создается бинарный файл Proba.mat

>> save('Proba2','X','Y','-ascii') % создается символьный файл Proba2 (без расширения)

>> save('Proba3.txt','X','Y','-ascii','-double') % создается символьный файл Proba3.txt с двойн.точн.

**Б) Вывод в текстовый файл с заданными разделителями - Функция dlmwrite**

DLMWRITE Записывает данные в текстовый файл с разделителями.

**DLMWRITE('FILENAME',M)** записывает матрицу M в файл FILENAME с использованием по умолчанию разделителя ',' между элементами в строке матрицы.

**DLMWRITE('FILENAME',M,'DLM')** записывает матрицу M в FILENAME с использованием символа DLM в качестве разделителя.

**DLMWRITE('FILENAME',M,'DLM',R,C)** записывает в файл матрицу M, начиная со строки R, и столбца C в файле. Здесь R и C – начинаются с нуля, так что R=C=0 определяют первое значение в первой строке файла.

**DLMWRITE('FILENAME',M,’-append’) –** дополнение существующего файла.

Пример

>>dlmwrite('Proba4.txt',X,',',2,3)

создается символьный файл Proba4.txt с содержимым

,,,,,,,

,,,,,,,

,,,0.95013,0.7621,0.61543,0.40571,0.057891

,,,0.23114,0.45647,0.79194,0.93547,0.35287

,,,0.60684,0.018504,0.92181,0.9169,0.81317

,,,0.48598,0.82141,0.73821,0.41027,0.0098613

,,,0.8913,0.4447,0.17627,0.89365,0.13889

### В) Вывод с использованием формата- функция fprintf

**<Файловая переменная> = fopen(‘<имя файла>’,’w’)**

**[count=] fprintf(fid,’<format>’,<список переменных вывода>)**

**fclose(fid)**

COUNT – число успешно выведенных байт (знаков).

В формате – кроме уже перечисленных элементов м.б. ‘\n’ (перевод строки), ‘\t’ (табулятор). Также в формат можно включать любые строки, которые просто будут выводиться так, как они написаны (симв.строка).

Например,

x = 0:.1:1; y = [x; exp(x)];

fid = fopen('exp.txt','w');

fprintf(fid,'%6.2f %12.8f\n',y);

fclose(fid);

создает текстовый файл, содержащий небольшую таблицу значений экспоненциальной функции:

0.00 1.00000000

0.10 1.10517092

...

* 1. 2.71828183

Еще пример – с вектором b=1:5

>> ff=fopen('rrr.txt','w')

ff = 3

>> asd=fprintf(ff,'Vector b = %d %d %d\n',b)

asd = 32

>> q=fclose(ff)

q = 0

Создается файл rrr.txt с содержимым

Vector b = 1 2 3

Vector b = 4 5

## Структурирование m-программ

1. ***Структурирование программ***

Структурная компонента – часть программы, записанная в отдельный файл с некоторым именем.

Обращение к компоненты – упоминанием в операторе имени файла.

Два вида структурных компонент: сценарии (скрипты) и функции.

Сценарий – любая совокупность операторов, записанная в файл. Запускается на выполнение указанием имени файла в отдельной строке программы.

Функция: часть программы, начинающаяся с заголовка функции, далее в скобках – аргументы. Функции – стандартные и пользовательские. Стандартные – это готовые к использованию функции (приданное MATLAB) – их несколько сотен. Чем больше их знаешь, тем легче решать задачи.

Пользовательские функции – написанные пользователем. В правах они ничем не отличаются от стандартных.

При разработке программы следует спроектировать ее структуру, то есть определить, какая часть программы будет реализована в главной программе (main), а какие части будут вынесены в отдельные файлы. Далее, для частей, выносимых в файлы, следует определить форму представления каждой из этих частей: сценарий или функция. Запуск на выполнение (передача управления) этих частей может производиться как из головной программы, так и из других частей программы. Поэтому вся программа, решающая интересующую пользователя задачу, может быть представлена в виде многоуровневой иерархической структурной схемы, в которой прямоугольниками (листьями) служат выделенные части программы, а соединительными линиями – пути возможной передачи управления между ними. Пример такой иерархии показан на рис.1.

Progr0 - Главная программа (сценарий)

Progr1 - сценарий

Progr2(A,B) - функция

Progr3 - сценарий

Progr4(С) - функция

Progr5 - сценарий

Progr6(D,E,G) - функция

Рис.1

Достоинства выделения компонент программы:

А) Легче отлаживать малые программы, чем большие.

Б) Возможность повторного использования в других программах.

В) Части могут параллельно разрабатываться несколькими программистами.

Г) Проще модифицировать программу – проще найти и заменить (деталь).

Д) Можно создавать библиотеки, а потом – Toolbox

Е) Можно здорово экономить память – оверлейные структуры.

Недостатки:

А) Потери времени на передачу управления от одной компоненты к другой (ниже быстродействие).

Б) Необходимость передачи данных от одной части к другой.

В) Можно проиграть в объеме памяти, занимаемой программой.

Г) Больше частей – легче запутаться (контроль версий).

**Области действия переменных.**

Начинает действовать с момента определения – например, в операторе присваивания. Заканчивает либо при удалении с помощью команды clear, либо по завершении той части программы, в которой определена (например, при завершении функции).

Переменная, действующая в некоторой части программы (в главной программе, сценарии, функции), действует также в других частях, управление которым передается из данной части программы.

В теле функции можно вводить переменные, которые являются, если не оговорено другое, локальными в этой функции. Использовать эти переменные можно только в функции, а при ее завершении занимаемая ими часть рабочей области освобождается. Если их значения требуются вне функции, то эти переменные должны быть объявлены глобальными с помощью оператора

global <список переменных>.

Аналогичное объявление следует сделать также в другой части программы, в которой они должны использоваться.

3 типа областей действия переменных: local, global, persistent

В скриптах (сценариях) переменные по умолчанию – глобальные, они остаются в раб.памяти после завершения сценария.

В функциях переменные по умолчанию – локальные, они автоматически убираются из памяти после завершения функции. Для сохранения – их надо объявить global. Например, global a,s,d

Переменные, объявленные в функции persistent, сохраняются в раб.памяти после завершения функции.

1. ***Стандартные функции***

СФ – это готовые к употреблению, не требующие программирования функции, имеющие имя и набор аргументов

Стандартные функции

Элементарные функции

Тригонометрические

Логарифмы и экспоненты

Комплексные числа

Округление и остатки

Элементарные матрицы и операции

Стандартные матрицы

Размеры

Манипуляции с матрицами

Случайные матрицы

Ввод-вывод данных

2D и 3D графика

Графики

Поверхности

Разметка графиков

ГИП

Анализ матриц

inv, det, eig, norm, rank, trace…..

Дата и время

Большое количество стандартных функций в MATLAB упрощают разработку программ – это его большое преимущество. Некоторые разделы библиотеки стандартных функций:

1. Элементарные функции: sin, log10, conj, round
2. Случайные матрицы: rand, randi, randn, randperm
3. Элементарные матрицы: zeros, numel, reshape, isnan, pi
4. 2D и 3D графика: plot, surf, title, guide
5. Дата и время: now, date, clock, pause
6. Анализ матриц: inv, det, eig, norm, rank, trace,…

И много-много других полезных функций.

Многие стандартные функции – с открытым кодом, например,

>> type date

function t = date

%DATE Current date as date string.

% S = DATE returns a string containing the date in dd-mmm-yyyy format.

%

% See also NOW, CLOCK, DATENUM.

% Copyright 1984-2002 The MathWorks, Inc.

c = clock;

mths = ['Jan';'Feb';'Mar';'Apr';'May';'Jun';'Jul';

'Aug';'Sep';'Oct';'Nov';'Dec'];

d = sprintf('%.0f',c(3)+100);

t = [d(2:3) '-' mths(c(2),:) '-' sprintf('%.0f',c(1))];

Некоторые стандартные функции – откомпилированные и их код увидеть нельзя – это так называемые встроенные (built-in) функции, например,

>> type zeros

'zeros' is a built-in function.

1. **Графика.**

Методика использования графики в MATLAB

1. Подготовить данные для построения графика.
2. Вызвать графическую функцию
3. Вызвать функции для усовершенствования графика: сетка, оси, заголовок, легенда, надписи…
4. Доработать график вручную в окне графического редактора – масштабы осей и др.
5. Сохранить график в файле нужного формата.

Простой 2D-график

Синтаксис:

            plot(y)  
            plot(x, y)  
            plot(x, y, s)  
            plot(x1, y1, s1, x2, y2, s2, ...)

Описание:

Команда plot(y) строит график элементов одномерного массива y в зависимости от номера элемента; если элементы массива y комплексные, то строится график plot(real(y), imag(y)). Если Y - двумерный действительный массив, то строятся графики для столбцов; в случае комплексных элементов их мнимые части игнорируются.

Команда plot(x, y) соответствует построению обычной функции, когда одномерный массив x соответствует значениям аргумента, а одномерный массив y - значениям функции. Когда один из массивов X или Y либо оба двумерные, реализуются следующие построения:

* если массив Y двумерный, а массив x одномерный, то строятся графики для столбцов массива Y в зависимости от элементов вектора x;
* если двумерным является массив X, а массив y одномерный, то строятся графики столбцов массива X в зависимости от элементов вектора y;
* если оба массива X и Y двумерные, то строятся зависимости столбцов массива Y от столбцов массива X.

Команда plot(x1, y1, s1, x2, y2, s2, ...) позволяет объединить на одном графике несколько функций y1(x1), y2(x2), ..., определив для каждой из них свой способ отображения.

Обращение к командам plot вида plot(x, y, s1, x, y, s2) позволяет для графика y(x) определить дополнительные свойства, для указания которых применения одной строковой переменной s1 недостаточно, например при задании разных цветов для линии и для точек на ней.

Изменение цвета и стиля линий и вида маркеров осуществляется добавлением еще одного аргумента функции plot:

**plot(x,y,'<цвет><стиль><маркер>')**

Дополнительный аргумент состоит из трех символов, первый символ задает цвет, второй – стиль линии, третий – вид маркера. Возможные значения:

• символа цвета: 'c', 'm', 'y', 'r', 'g', 'b', 'w', 'k' – соответствуют голубому, малиновому, желтому, красному, зеленому, синему, белому и черному цветам;

• символа стиля линии: '-' – сплошная линия, '--' - разрывная, ':' - пунктирная, '-.' – штрих-пунктирная. Если символ стиля отсутствует (пробел), то линия не проводится.

• символа маркера: '+', 'o', '\*', 'x'. Или ‘s’-квадрат, ‘d’-ромб, ‘v’-треугольник вниз, ‘^’ – треугольник вверх,’<’/’>’-треугольник влево/вправо, ‘p’-пентаграмма, ‘h’-гексограмма.

**Построение гистограммы**

Синтаксис:

            hist(y)  
            hist(y, n)  
            hist(y, x)  
            [y, x] = hist(y, ...)

Возможность сохранения графических окон в графических файлах заданных форматов, например, после вывода гистограммы

**grid on**

**xlabel(' b12');**

**saveas(gcf,'ris\_b12.bmp','bmp');**

Аргумент 1 : **gcf= get current figure –** источник (откуда взять)

Аргумент 2 : символьная строка с именем файла сохранения

Аргумент 3 : символьная строка с указанием формата файла сохранения (fig, bmp, png, eps, jpg,…)

Есть еще много других графических функций, в т.ч. псевдотрехмерная графика.

1. ***Пользовательские функции***

Если какая-то часть программы реализует типовую функцию, выполнение которой может представить интерес в разных задачах, то совокупность операторов, реализующих эту функцию, имеет смысл оформить в виде **пользовательской функции**. Создаваемая пользовательская функция расширяет библиотеку системных функций MATLAB и может послужить основой для создания собственного проблемно-ориентированного пакета программ (Toolbox).

В первую очередь необходимо придумать имя функции в виде совокупности латинских букв, цифр и символов ‘\_’, причем имя должно начинаться с буквы. Рекомендуемая длина имени – не более 8 символов. Далее необходимо определить, какие входные аргументы должны быть заданы для начала выполнения функции и какие выходные аргументы могут получиться после завершения выполнения функции. Этим аргументам следует придумать имена и определить порядок их перечисления в списках входных и выходных аргументов. Порядок перечисления аргументов тоже имеет значение для эффективного применения функции!

Придуманное имя функции используется трижды: во-первых, при задании имени файла, в который будет записана функция. Этот файл должен иметь имя <имя функции>.m. Во-вторых, это имя используется в первой строке файла с текстом программы, реализующей функцию. Эта строка должна иметь вид

**function [<список выходных аргументов>] = <имя функции>(<список входных аргументов>)**

В качестве входных и выходных аргументов могут использоваться переменные, разделяемые запятыми.

В-третьих, имя функции следует использовать во второй строке файла, которая является строкой комментария и должна иметь вид

**%<имя функции> - <краткая формулировка функции>**

Эта строка имеет важное значение – она будет использоваться при выполнении команды

**lookfor <контекст поиска> ,**

а также команды

**help <каталог >**

Последующие несколько строк в файле также должны быть строками комментария и используются для разъяснения смысла функции, ее входных и выходных параметров, вариантов употребления функции и прочих данных, полезных для пользователей. В неруссифицированных версиях MATLAB эти комментарии следует писать на английском языке или, по крайней мере, латинскими буквами. Эти строки будут воспроизводиться в дальнейшем при вводе команды обращения к помощи

**help <имя функции>**

После комментариев в файле должно располагаться тело функции: совокупность операторов на М-языке, реализующих функцию. В этих операторах можно использовать переменные с зарезервированными именами **varargin, varargout, nargin, nargout**, которые обозначают, соответственно, списки входных и выходных аргументов, числа входных и выходных аргументов, заданных пользователем при обращении к функции. Самое простое, что можно сделать с этими переменными – проверить, совпадают ли числа входных и выходных аргументов с требуемыми числами, и при несовпадении – прекратить выполнение функции.

Выполнение функции завершается либо после выполнения последнего оператора в файле, либо при выполнении оператора RETURN, который обычно используется для аварийного прекращения выполнения функции. Вставлять этот оператор в конце файла не требуется!

В составе функции можно задать одну или несколько подфункций, которые записываются в конце файла, после текста основной функции и оформляются по тем же правилам, что и основная функция. Отличие подфункции от основной функции состоит в том, что она не может быть вызвана на выполнение откуда-либо, кроме основной функции.

В качестве примера рассмотрим функцию, реализующую расчет по матрице наблюдений X вектора медиан. Функция записывается в файл с именем **median\_v.m**.

**function znach=median\_v(X,dim)**

**% MEDIAN\_V - counts the values of median in columns of matrix X.**

**%X - input matrix,**

**%dim – dimension of matrix X,**

**% znach - output values of medians**

**if nargin==1,**

**dim = min(find(size(X)~=1));**

**if isempty(dim), dim = 1; end**

**end**

**if isempty(X), znach = []; return, end**

**XX=sort(X);**

**[m,n]=size(XX);**

**m1=round(m/2)**

**if (m1+m1)==m**

**znach=(XX(m1,:)+XX(m1+1,:))/2;**

**else**

**znach=XX(m1,:);**

**end**

## Графический интерфейс – методика построения

Назначение интерфейса: обеспечение взаимодействия пользователя с программой. Функции:

* Контроль работы программы (что делает).
* Получение результатов работы на экране.
* Управление работой программы (кнопки, меню,…).
* Ввод данных, необходимых для работы программы (выбор из списков, задание данных в редактируемых окнах,…).

Графический интерфейс – совокупность экранных форм, используемых для ввода-вывода данных и управления работой приложения. Каждая экранная форма представляет собой отдельное окно, в котором размещены графические объекты. Каждая форма обслуживается отдельной программой – функцией.

***Методика создания интерфейса***

При разработке графического интерфейса приложения должны быть решены следующие проблемы:

А) Разработка проекта интерфейсной структуры: какие формы, как они связаны по логике управления. Создание экранных форм;

Б) Создание главной программы – с которой программа начинает работу (из нее будет вызвана программная компонента, обслуживающая первую форму);

В) Управление формами, обеспечивающее их появление в соответствии с логикой приложения;

Г) Обеспечение взаимодействия приложения с интерфейсом по данным: получение (get) данных от объектов формы и занесение в переменные памяти или запись (set) данных из переменных в объекты на форме.

А) Проект экранных форм – важно! Интерфейс – «лицо программы». Важен хороший дизайн. Каждая экранная форма сначала рисуется на бумаге. Потом ее хорошо бы согласовать с теми, кто будет работать с программой. Только после этого каждую форму реализуют в отдельности.

При реализации используется графический редактор GUIDE. Команда:

**>>> guide**

Как и везде, он сначала предоставляет «рамку» (Frame), в которую вставляются в нужном размещении визуальные компоненты: тексты, редактируемые окна, кнопки, списки выбора и др. Принцип: перетащи и разверни в нужном месте (drug and drop). Каждая компонента обладает набором свойств: размеры, цвет, надпись,…. Для их настройки используется Property Inspector.

При первом сохранении изображения формы создаются 2 файла: графический (\*.fig) – с изображением формы и текстовый (\*.m) – с функциями, обслуживающими форму. Как и везде, эти функции – обработчики событий (callback), связанных с формой: ввод данных, нажатие на кнопку, выбор из меню…. При возникновении события запускается функция-обработчик события, содержащаяся в m-файле, обслуживающем форму.

При сохранении форме присваивается имя, которое становится именем этих файлов. В последующем можно доработать формы – для этого потребуется снова вызвать редактор:

**>>> guide(<имя формы>)**

Б) В главной программе осуществляется начальная подготовка приложения к работе. Иногда тут вводятся данные из некоторых файлов. В самом простом случае в ней один оператор – вызов функции, обслуживающей начальную форму – с которой начинается диалог с приложением.

В) Программные компоненты, обслуживающие отдельную форму являются функциональными, т.е в начале ее – функция с именем, совпадающим с именем файла. Кроме того в этом файле есть другие функции – обработчики событий и методы Callback.. Обратиться к этим функциям (кроме первой) нельзя извне файла.

В программных компонентах приложения в нужных местах вставляются операторы, обеспечивающие появление форм на экране:

**ffi=openfig('<имя формы>.fig','reuse');**

Можно открыть сразу несколько форм и сделать активной одну из них с помощью оператора

**f=figure(<имя формы>)**

Если форма стала ненужной, ее надо закрыть оператором

**close(f)**

Таким образом осуществляется управление формами.

В) Организация информационных связей: а) взаимодействие изображений форм с оперативной памятью (переменными) – ввод-вывод данных; б) обмен данными между компонентами программы

а) Чтение/запись значений свойств объектов следует осуществлять в методах Callback обращением к свойствам с помощью функций get и set

**get(handles.<имя компоненты>,’<имя свойства>’)**

**set(handles.<имя компоненты>,’<имя свойства1>’,<присваиваемое значение свой-ства1>,<имя свойства2>’,<присваиваемое значение свойства2>,….)**

Например, обработчик события «нажатие кнопки» с именем pushbutton1:

**function varargout = pushbutton1\_Callback(h, eventdata, handles, varargin)**

**……………..**

**handles = guihandles(gcbo);**

**PR = str2double(get(handles.edit1,'String'));%Читаем данные из edit1**

………

или

**set(handles.edit4,'String',REZSUM); %Записываем значение в edit4**

Оператор handles=guihandles(gcbo) аналогичен в чем-то файловой переменной при работе с файлами – после ее определения при открытии файла текущий объект выступает под именем handles (это переменная типа «Структура»).

б) *Передача данных между программными компонентами*

При использовании обычных функций информационная связь с этой функцией может реализовываться двумя способами:

- с использованием аргументов функции;

- с использованием переменных, имеющих глобальную область действия.

В функциях, обслуживающих экранную форму и ее компоненты первый способ оказывается недоступным. Поэтому единственная возможность обеспечения такой связи – переменные с глобальной областью действия. В функциях, обслуживающих компоненты экранной формы, все переменные, участвующие в информационных связях с другими частями программы, должны быть объявлены глобальными, например, так

global A B C

Аналогичное объявление следует сделать в тех частях программы, с которыми должен осуществляться информационный обмен.

***Пример по приборной базе ЦКП***

В файле prib.txt находятся данные о балансовой стоимости приборов, используемых в Центрах коллективного пользования приборной базой (ЦКП) в системе Минобрнауки РФ. Число записей – 5216.

В каждой строке файла – значение балансовой стоимости одного прибора в одном из ЦКП.

Требуется разработать приложение, выполняющее функции:

1. Ввод информации из файла.
2. Задание пороговой (максимальной) стоимости прибора.
3. Расчет числа приборов со стоимостью ниже пороговой.
4. Построение гистограммы распределения числа приборов со стоимостью ниже пороговой.

Программа состоит из главной программы-сценария в файле с именем Pribory.m и файла GUI с именем Pribase.m.

**Главная программа(Pribory.m)**

**% Ввод данных о стоимостях приборов в ЦКП и расчет гистограммы**

**global X Xup Xmax**

**X=load('prib.txt');**

**Xmax=max(X);**

**Xup=Xmax;**

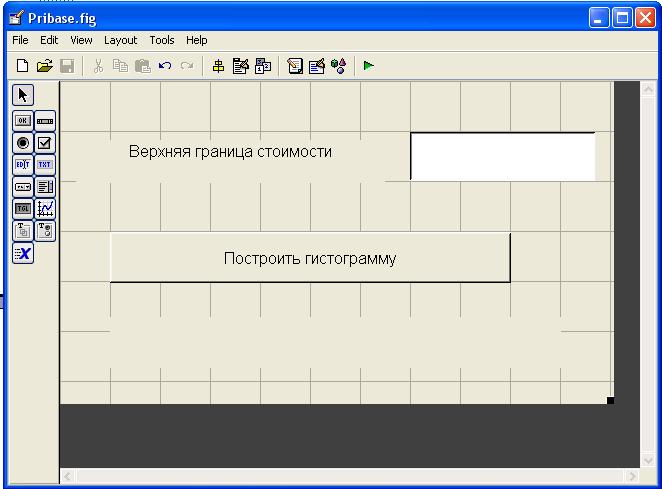
**Ndan=length(X);**

**hh=openfig('Pribase.fig','reuse');**

Выбранное имя формы - Pribase

**Файл Pribase.fig**

На форме – 4 компоненты: 2 текстовые (text), редактируемое окно (edit) и кнопка (pushbutton). Свойства компонент: Tag – идентификатор (имя), String – надпись, Visible - видимость



**Файл GUI (Pribase.m.)**

function varargout **= Pribase**(varargin)

% PRIBASE M-file for Pribase.fig

% PRIBASE, by itself, creates a new PRIBASE or raises the existing

% singleton\*.

% H = PRIBASE returns the handle to a new PRIBASE or the handle to

% the existing singleton\*.

% PRIBASE('CALLBACK',hObject,eventData,handles,...) calls the local

% function named CALLBACK in PRIBASE.M with the given input arguments.

% PRIBASE('Property','Value',...) creates a new PRIBASE or raises the

% existing singleton\*. Starting from the left, property value pairs are

% applied to the GUI before Pribase\_OpeningFunction gets called. An

% unrecognized property name or invalid value makes property application

% stop. All inputs are passed to Pribase\_OpeningFcn via varargin.

% \*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one

% instance to run (singleton)".

% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help Pribase

% Last Modified by GUIDE v2.5 01-Apr-2014 14:26:50

% Begin initialization code - DO NOT EDIT

gui\_Singleton = 1;

gui\_State = struct('gui\_Name', mfilename, ...

'gui\_Singleton', gui\_Singleton, ...

'gui\_OpeningFcn', @Pribase\_OpeningFcn, ...

'gui\_OutputFcn', @Pribase\_OutputFcn, ...

'gui\_LayoutFcn', [] , ...

'gui\_Callback', []);

if nargin && ischar(varargin{1})

gui\_State.gui\_Callback = str2func(varargin{1});

end

if nargout

[varargout{1:nargout}] = gui\_mainfcn(gui\_State, varargin{:});

else

gui\_mainfcn(gui\_State, varargin{:});

end

% End initialization code - DO NOT EDIT

% --- Executes just before Pribase is made visible.

**function Pribase\_OpeningFcn(hObject, eventdata, handles, varargin)**

% This function has no output args, see OutputFcn.

% hObject handle to figure

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% varargin command line arguments to Pribase (see VARARGIN)

% Choose default command line output for Pribase

handles.output = hObject;

% Update handles structure

guidata(hObject, handles);

% UIWAIT makes Pribase wait for user response (see UIRESUME)

% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.

**function varargout = Pribase\_OutputFcn(hObject, eventdata, handles)**

% varargout cell array for returning output args (see VARARGOUT);

% hObject handle to figure

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure

varargout{1} = handles.output;

**function edit1\_Callback(hObject, eventdata, handles)**

% hObject handle to edit1 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text

% str2double(get(hObject,'String')) returns contents of edit1 as a double

% --- Executes during object creation, after setting all properties.

**function edit1\_CreateFcn(hObject, eventdata, handles)**

% hObject handle to edit1 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc

set(hObject,'BackgroundColor','white');

else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));

end

**global Xmax**

**set(hObject,'String',num2str(Xmax));**

% --- Executes on button press in pushbutton1.

**function pushbutton1\_Callback(hObject, eventdata, handles)**

% hObject handle to pushbutton1 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

**global X Xup**

**handles=guihandles(gcbo);**

**Xup= str2double(get(handles.edit1,'String'));**

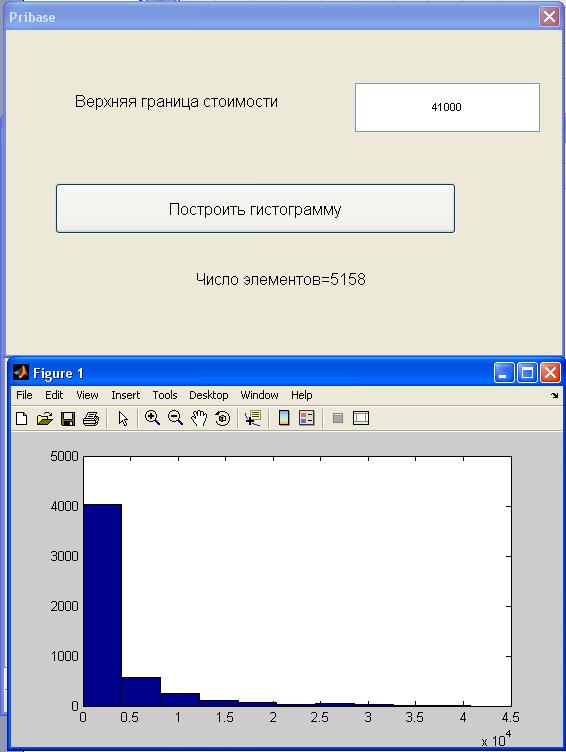
**X1=X(X<Xup);**

**Ndan=length(X1);**

**set(handles.text4,'String',['Число элементов=' num2str(Ndan)]);**

**figure %появляется новое графическое окно и становится активным**

**hist(X1); %В активном окне отображается гистограмма**



Приложение допускает многократное повторение ввода порогов и построения гистограмм.

## Основные Toolboxes

ToolBoxes – это не просто дополнительные наборы функций для решения различных классов задач. Они представляют собой плод многолетних усилий множества известных ученых, работающих в таких областях, как моделирование сложных систем, обработка сигналов, идентификация объектов управления, финансовые рынки, обработка изображений и другие. Число проблемно-ориентированных пакетов программ, предлагаемых вендором MATLAB, весьма велико и увеличивается с развитием этой системы. На рис.1 показан пример списка таких пакетов для версии MATLAB 7.0.

Дадим общую характеристику некоторых пакетов, непосредственно относящихся к предметной области нашей специальности.

На первый случай интересны следующие пакеты: **Control System Toolbox, System Identification Toolbox, Optimization Toolbox, Statistics Toolbox, Neural Network Toolbox**

1) **Control System Toolbox** – предназначен для создания и анализа моделей объектов управления и проектирования регуляторов.

Создание моделей – построение линейных моделей, моделей с перекрестными взаимодействиями, определение характеристик моделей, преобразование между моделями непрерывного и дискретного времени, редукция порядка моделей

Анализ моделей – осуществляется с помощью средств визуализации, графического интерфейса пользователя, который упрощает задачу исследования отклика модели.

Проектирование регуляторов – предоставляет эффективное программное средство, с использованием графического интерфейса которого можно осуществлять быстрое итерационное проектирование регуляторов. С помощью этого средства можно настраивать регулятор за счет влияния на полюса, нули, обратные связи, характеристики фильтров.

2) **Instrument Control Toolbox** – это набор функций на m-языке, ориентированных на базовую среду MATLAB. Пакет предоставляет возможность организовать обмен с внешними объектами (регуляторами, средствами измерения и др.) при использовании интерфейсов в разных стандартах. Данные могут быть бинарными или текстовыми, а их передача - синхронной или асинхронной.

3) **Model Predictive Control (MPC) Toolbox** – представляет собой набор программ, помогающих проектировать, анализировать и внедрять современные промышленные алгоритмы автоматизации. Так же, как и другие пакеты, он предоставляет удобный графический интерфейс пользователя, равно как и гибкий синтаксис команд, учитывающий потребности пользователя. Как следует из его названия, этот пакет используется для автоматизации целевых систем на основе комбинирования стратегий прогноза и управления. Приближенная линейная модель объекта дает возможность прогнозирования. Стратегия управления осуществляет сравнение прогнозах состояний объекта с множеством наблюдений и затем изменяет доступные управления для достижения состояний объекта, удовлетворяющих заданным ограничениям. Такие ограничения могут определяться физическими границами управляющих воздействий, требованиями безопасности функционирования и нижней границей качества продукции.

4) **Neural Network Toolbox** – содержит библиотеку функций для работы с нейросетями. Предназначен для решения большого числа исследовательских и прикладных задач с использованием нейронных сетей: при проектировании систем управления для различных объектов, моделировании бизнеса, в банковской сфере, в медицине, при решении военных задач и в других. Нейросети состоят из простых элементов, функционирующих параллельно. Эти элементы считаются в какой-то степени подобными биологическим нейросетям. Как и в природе, функционирование сети определяется в значительной мере связями, существующими между элементами. Нейросеть перед ее использованием

5) **Optimization Toolbox** – это множество функций, расширяющих вычислительные возможности среды MATLAB. Пакет включает процедуры для многих вариантов постановок задач оптимизации, в том числе

- нелинейная минимизация с ограничениями и без них, включая минимаксные и полубесконечные задачи;

- квадратичное и линейное программирование;

- нелинейный метод наименьших квадратов и подгонка кривых;

- решение систем нелинейных уравнений;

- линейный метод наименьших квадратов с ограничениями.

6) **Partial Differential Equation (PDE) Toolbox** – предоставляет мощную и гибкую среду для изучения и решения дифференциальных уравнений в частных производных в пространстве или времени. Уравнения дискретизируются с помощью метода конечных элементов.

7) **Robust Control Toolbox** представляет собой инструменты для анализа и автоматической настройки систем управления. С их помощью можно создать нечеткие модели путем комбинирования динамики с нечеткими элементами.

8) **Statistics Toolbox** – это совокупность статистических инструментов, встроенных в вычислительную среду MATLAB. Пакет обеспечивает решение широкого диапазона статистических задач: от генерации случайных чисел с разными законами распределения, до подгонки кривых, планирования экспериментов и статистического контроля. Он содержит инструменты, разделенные на две категории: функции для работы с вероятностями и статистикой и графические интерактивные инструменты. Инструменты первой категории используются в командной строке или в приложениях пользователя. Многие из этих функций представляют собой m-файлы MATLAB с операторами, реализующими специализированные статистические алгоритмы.

9) **System Identification Toolbox** предназначен для разработки точных и простых моделей сложных систем по данным в виде зашумленных временных рядов. Он представляет собой инструменты для создания математических моделей динамических систем на основе данных наблюдений входа и выхода моделируемого объекта. Пакет обладает гибким графическим интерфейсом пользователя, который помогает в формировании данных и моделей. Используемые в данных инструментах методы идентификации полезны для применения при проектировании систем управления, анализе сигналов, представленных в виде временных рядов и при анализе вибраций. Пакет предоставляет библиотеку блоков для представления результатов идентификации систем в среде Simulink. ребует настройки с подгонкой значений весов связей между элементами.

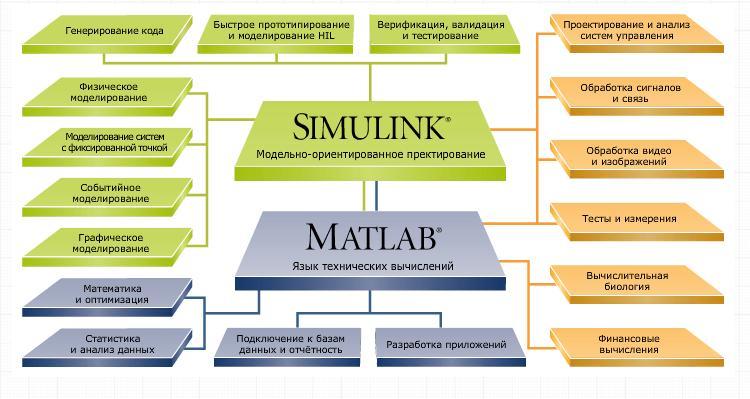


Рис.1

**Компонента SIMULINK - назначение, история создания**

SIMULINK - графическая среда имитационного моделирования, позволяющая при помощи блок-диаграмм в виде направленных графов, строить динамические модели, включая дискретные, непрерывные и гибридные, нелинейные и разрывные системы

Simulink представляет собой среду для формирования моделей, которые могут состоять из нескольких, связанных между собой областей (доменов). Его основное назначение – основанное на построении моделей проектирование динамических и встроенных систем. Он предоставляет интерактивную графическую среду и расширяемые пользователем библиотеки блоков для использования при проектировании, моделировании, применении и тестировании множества изменяющихся во времени систем в задачах, связанных с коммуникациями, управлением, обработкой сигналов, обработкой изображений. Simulink интегрирован с MATLAB, обеспечивая немедленный доступ к инструментам для разработки алгоритмов, анализ и визуализацию моделей, создание сценариев для пакетной обработки, настройки среды моделирования и определения сигналов, параметров и тестовых данных.

SIMULINK обеспечивает пользователю доступ ко всем возможностям пакета MATLAB, в том числе к большой библиотеке численных методов.



## Основные свойства SIMULINK

Simulink – это важная составная часть MATLAB. Взаимодействие со средой при решении задач представлено на рис.2. Simulink в настоящее время представляет собой целое семейство взаимосвязанных программ, позволяющих решать задачи моделирования объектов и систем в разных предметных областях. Основные продукты этого семейства показаны в табл.1.

## 

**Методика использования**

Подготовка задачи для моделирования в SIMULINK проводится в следующей последовательности:

1. Выбор расчетной схемы
2. Составление системы уравнений, описывающих исследуемый процесс.
3. Приведение системы к виду, удобному для решения (разрешение относительно старших производных).
4. Определение начальных условий.
5. Составление структурной схемы.
6. Моделирование возмущающих функций.
7. Определение исходных данных.
8. Составление модели в среде SIMULINK.
9. Включение средств визуализации.
10. Тестирование.
11. Решение.
12. Анализ результатов.
13. Отчет.   
    Чтобы создать новую модель, щёлкните мышкой по пиктограмме или наберите команду **simulink**. Перед вами появится два окна. В первом, можно из библиотеки выбрать модель, а во втором, как из кирпичиков, вы собираете общую модель системы.



Двойным щелчком, например по пиктограмме Source (Источники), вы откроете библиотеку источников. Аналогично с Sinks (Приемники), Scope (Индикатор). Открывая соответствующие библиотеки и перетягиванием мышкой выбранной модели в окно создания моделей системы вы можете набрать модель реальной системы. Далее, запустив модель, вы получите решение. В настройках можно выбрать время работы системы, метод решения и некоторые другие параметры. Пример некоторых блоков показан на рис.2.

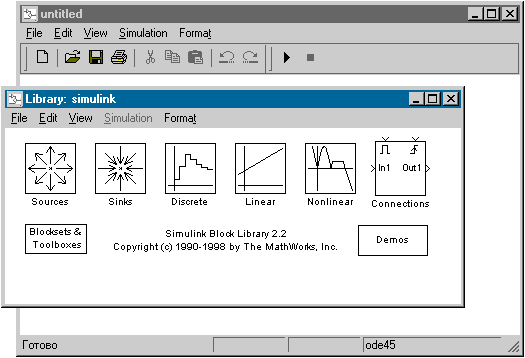


Рис.2

**Примеры**

Рассмотрим реакцию системы с единичной ООС и апериодическим звеном в прямой цепи на единичное воздействие. Для её набора нам понадобятся библиотеки|модели: Sources|Step, Sinks|Scope, Linear|Transfer Fcn и Linear|Sum. Перетянув эти элементарные модели в окно редактирования Simulink, изменяем исходные параметры в соответствии с нашей системой (двойной щелчок мышью) и соединяем их. Результат этих манипуляций представлен на рис.3 и 4.

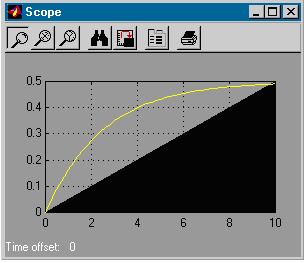
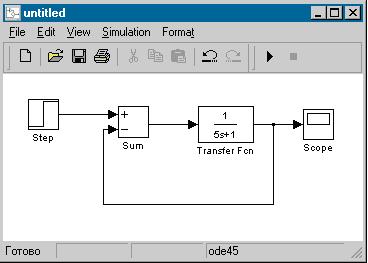


Рис.3 Рис.4

Чтобы можно было использовать полученное решение в сценарии MATLAB нужно добавить компонент Connections|Out. Теперь вы имеете возможность работать с переменной. Данные, которые поступают на этот порт доступны вам из среды MATLAB. Чтобы запустить модель, вам достаточно набрать команду **sim** с необходимыми параметрами (наберите HELP SIM для подробной информации).

Y = SIM(MODEL,UE)

MODEL: contains the parameters of the model in any of the IDMODEL formats, IDSS, IDPOLY, IDARX, IDGREY or IDPROC.

UE: the input-noise data UE = [U E]. Here U is the input data, that could be given as an IDDATA object (with the signal defined as input) or as a matrix U = [U1 U2 ..Un] with the column vector Uk as the k:th input. Similarly, E is either an IDDATA object or a matrix of noise inputs (as many columns as there are output channels). If E is omitted a noise-free simulation is obtained.

Example: U = iddata([],idinput(200),'Ts',0.1);

E = iddata([],randn(200,1),'Ts',0.1); Y = SIM(MODEL,[U E]);

# Программная среда R

## Введение в R

Теперь перейдем к рассмотрению еще одной информационной технологии – среды R. В то время, как MATLAB связан с нашей специальностью направленностью на технические расчеты и моделирование объектов управления, среда R также является важной для нас, предоставляя средства для глубокого анализа данных. Тем самым, эта среда даст нам возможность управления сложными объектами, в которых требуется статистическая обработка данных, в том числе, так называемых «Больших данных».

При изучении этой среды следует обратить внимание на значительную общность архитектуры и подходов к решению задач в ней и в среде MATLAB. Поэтому освоение этой среды становится значительно более простым после того, как получены навыки работы в MATLAB.

**Сводка сведений о среде**

|  |  |
| --- | --- |
|  | |
|  | |
| **Парадигмы программирования** | Множество парадигм: объектно-ориентированная, императивная, функциональная, процедурная |
| **Авторы** | Ross Ihaka и Robert Gentleman |
| **Разработчики** | R Development Core Team |
| **Впервые появилась** | Август 1993 |
| **Текущий релиз** | 3.6.3 / 29.02.2020, 83 MB (2014-07-10) |
|  |  |
| **Способ типизации** | Динамическая |
| **Предшественники** | S, Scheme, XLispStat |
| **ОС** | Кросс-платформенная |
| **Лицензия** | GNU General Public License |
| **Веб-сайт** | [www.r-project.org](http://www.r-project.org) |

**Среда R – назначение, история появления**

**R** — язык программирования для статистической обработки данных и работы с графикой, а также свободная программная среда вычислений с открытым исходным кодом в рамках проекта GNU. Язык создавался как аналогичный языку S, разработанному в Bell Labs, и является его альтернативной реализацией, хотя между языками есть существенные отличия, но в большинстве своём код на языке S работает в среде R. Изначально R был разработан сотрудниками статистического факультета Оклендского университета (Новая Зеландия) Россом Айхэкой (англ. *Ross Ihaka*) и Робертом Джентлменом (англ. *Robert Gentleman*) (первая буква их имён — R) в 1993 г.; язык и среда поддерживаются и развиваются организацией *R Foundation*.

**S -язык** **программирования**, разработанный фирмой AT&T Bell Labs (появился в середине 80-х годов), предназначен для обработки данных. Разработано несколько версий расширения **языка** **S** — **S**-Plus, для различных платформ.

R широко используется как статистическое программное обеспечение для анализа данных и фактически стал стандартом для статистических программ.

R распространяется со свободной лицензией GNU GPL (General Public License). Распространяется в виде исходных кодов, а также откомпилированных приложений под ряд операционных систем: FreeBSD, Solaris и другие дистрибутивы Unix и Linux, Microsoft Windows, Mac OS X.

В R используется интерфейс командной строки, хотя доступны и несколько графических интерфейсов пользователя, например пакет R Commander, RKWard, RStudio, Weka, Rapid Miner, KNIME, а также средства интеграции в офисные пакеты.

В 2010 году R вошёл в список победителей конкурса журнала Infoworld в номинации на лучшее открытое программное обеспечение для разработки приложений

R и дополнительные пакеты распространяются через CRAN (акроним Comprehensive R Archive Network= Всеобъемлющая сеть R-архивов). В настоящее время в мире доступны более 160 сайтов (зеркала CRAN) в разных странах, в том числе и в России (<https://cran.cmm.msu.ru/>). Головной узел — ([***http://cran.r-project.org/***](http://cran.r-project.org/)) расположен в Вене (Австрия).

## Среда R

R – это интегрированный комплекс программных средств для манипуляций с данными, расчетов и графического отображения.

Преимущества R:

1. Наличие макроязыка: небольшим числом команд можно решить достаточно сложные задачи.
2. Качественная графика: практически не требует «доводки» перед публикацией.
3. Достаточная простота освоения – доступно людям-непрофессионалам в программировании.
4. Разнообразие возможных форматов во внешних источниках данных: бинарные и текстовые файлы, электронные таблицы, базы данных.
5. Возможность свободно получить большое число готовых программ или функций – часть поставляется в дистрибутиве, но огромное множество – через CRAN.
6. Позиционируется, как одно из главных средств работы с «большими данными» - можно использовать большой объем и разнообразные по форме представления данные.
7. Многоплатформенность – возможность переноса программ из одной ОС в другую без переделок или с минимальными переделками.
8. Свободное ПО, а следовательно, развитие поддерживается всемирным коллективом разработчиков.

Недостатки:

1. В R используется интерпретатор, а, следовательно, достаточно медленное решение задач.
2. Имеются проблемы с созданием графического интерфейса для программ пользователя.
3. Все-таки среда R больше ориентирована на статистическую обработку. Мало приспособлена для решения других задач (например, моделирования динамических систем).
4. При работе с R конечно же требуется знание не столько программирования, но хорошее знание методов анализа данных.

Помимо прочего R предоставляет следующие возможности:

* эффективную обработку данных и возможности их хранения,
* набор операторов для вычислений с массивами и, в частности, с матрицами,
* большую, логически связанную, интегрированную совокупность посреднических инструментов для анализа данных,
* хорошие графические возможности для анализа данных и отображения либо непосредственно на компьютере, либо с получением твердой копии,
* хорошо развитый, простой и эффективный язык программирования, который включает условные операторы, циклы, определяемые пользователем рекурсивные функции и возможности ввода и вывода.

Термин «среда» означает ее характеристику как цельную спланированную и согласованную систему, а не как постепенное наращивание весьма специфических и негибких инструментов, как это часто бывает в случаях с другими программами для анализа данных.

R – это в значительной степени механизм для вновь разрабатываемых методов интерактивного анализа данных. Он был быстро разработан и затем расширен с помощью большого числа пакетов. Однако, большинство программ, написанных на R, являются по существу недолговечными, написанными для единственного применения в анализе данных.

R может рассматриваться как воплощение языка S, который был разработан в Bell Laboratories Ричардом Беккером, Джоном Чамберсом и Аланом Уилксом, и также составляет базис для систем S-Plus.

## R и статистики

Многие люди используют R как статистическую систему. Можно представлять ее как среду, в рамках которой могут применяться многие классические и современные статистические методы. Некоторые из них встроены в базовую среду R, но многие поставляются в виде пакетов. Имеется около 30 пакетов, поставляемых с R (так называемые “стандартные” и “рекомендуемые” пакеты). К ним, в частности, относятся пакеты: **base, graphics, grDevices, stats, lattice, tcltk, compiler, datasets, grid, methods, parallel, splines, stats4, tools, utils, KernSmooth, MASS, Matrix, boot, class, cluster, codetools, foreign, ngcv, nlne, nnet, rpart, spatial, survival**. Намного большее число других пакетов доступно в семействе **CRAN (The Comprehensive R Archive Network)** на Интернет-сайтах (через http://CRAN.R-project.org) (В настоящее время в CRAN представлено и доступно для свободного скачивания более 15000 пакетов) и в других местах. Большинство классических статистик и много более поздних методологий доступно для применения с R, но пользователи должны быть готовы потрудиться для их нахождения. Имеется важное различие в философиях между S (а, следовательно, и R) и другими основными статистическими системами. В S статистический анализ обычно выполняется в виде последовательности шагов с промежуточными результатами, сохраняемыми в объектах. Таким образом, в то время как в системах SAS и SPSS будут получены большие выводы по результатам регрессионного или дискриминантного анализа, R даст минимальный вывод и сохранит результаты в соответствующем объекте для последующего использования в других R функциях.

Как уже отмечалось, R также позиционируется, как средство для решения задач с большими данными.

## Архитектура среды R

ОЗУ

CRAN – пакеты (Интернет )

Непосредственно доступные пакеты

Подключенные пакеты

Программы пользователя на R (\*.R)

Память программ

Главная управляющая программа

Интерпретатор

Рабочее пространство

Рабочее пространство

Графический редактор

Командный процессор

\*.jpeg

История команд

История команд

Файлы с данными

Текстовый редактор

Сравнивая эту архитектуру с архитектурой MATLAB, нетрудно заметить явную аналогию.

## Язык R

Язык предназначен для управления ресурсами среды при решении задач пользователя. Он используется при написании команд. Команды вводятся в командной строке консоли среды R или в строке кода программы. Команда может заканчиваться символом «точка с запятой» или/и «возврат каретки». Таким образом, на строке может быть либо одна команда, либо – несколько, разделенных точками с запятой.

Если в строке встретится символ «#», то все, что следует за ним и до конца строки рассматривается как комментарий. Нельзя комментарий вставлять в середину имени, строки символов или среди списка аргументов функции.

Элементарные команды могут группироваться в выражения с помощью круглых или фигурных скобок.

В командной строке команды должны вводиться после символа приглашения «prompt» - разный символ в версиях для разных ОС. Например, в версии для Windows – это «>», в UNIX – «$». Впрочем, этот символ может меняться в настройках!

Если команда – длинная и не помещается на строке, то можно поставить знак «+» и продолжить на следующей строке.

Как и в среде MATLAB, здесь команды делятся на операторы присваивания и на управляющие конструкции.

В качестве знака присваивания здесь можно использовать <-, -> и =. Например,

> a<-12

> 3+4i->g

> n=7

В командной строке может находиться выражение (скрытое присваивание, как в MATLAB)

**> 17.8\*(exp(0.234)-sin(56))-6.7**

**[1] 25.07648**

Для повторного получения этого значения или для использования его в присваивании можно использовать такой прием:

**> .Last.value**

**[1] 25.07648**

Очевидно, что **.Last.value –** это аналог переменной **ans** в MATLAB.

Введенные в командных строках консоли команды сохраняются в памяти и по ним возможна навигация с помощью клавиш со стрелками «вверх» или «вниз» (как в MATLAB). Это позволяет редактировать ранее введенные команды.

В конце сеанса работы со средой R всегда предлагается сохранить команды и данные из рабочей области, по умолчанию, соответственно, в файлах **.Rhistory** и **.RData** (имена можно изменить) – в текущей директории. В следующем сеансе работы со средой их можно загрузить из этих файлов.

**Правила именования в среде R.**

Имена присваиваются переменным, функциям, файлам. Имя – совокупность из одного или нескольких символов, которые могут выбираться из A…Z, a…z, \_ (нижний подчерк), . (точка). В принципе могут использоваться и буквы кириллицы, но лучше этого избегать. Имя должно начинаться либо с буквы, либо с точки, за которой обязательно должна следовать буква. Имена – регистрочувствительные, т.е. большие и малые буквы – различаются! Нижний подчерк может присутствовать в середине или конце имени. Длина имени практически не ограничена, но выгоднее избегать длинных имен.

**Основные типы переменных.**

Язык поддерживает минимальный набор простых (атомарных) типов данных: символьный (character), числовой (numeric), логический (logical) и комплексный (complex).

Сложные типы переменных строятся с использованием как атомарных, так и других основных типов:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Обозначение** | **Тип** | **Возможные типы данных** | **Использование в объекте нескольких типов данных** | **Пояснение** |
| **vector** | **вектор** | числовой (целый, действительный), символьный, комплексный, логический | Нет | обычная переменная - поименованная совокупность однотипных значений |
| **factor** | **фактор** | числовой, символьный | Нет | категорийная переменная - предоставляет компактные способы для обращения с категоризированными данными |
| **array** | **массив** | числовой, символьный, комплексный, логический | Нет | таблица с *k* измерениями - многомерное обобщение векторов. По сути она является вектором, который может быть снабжен двумя или более индексами и отображается особым образом |
| **matrix** | **матрица** | числовой, символьный, комплексный, логический | Нет | частный случай массива с *k* = 2. |
| **list** | **список (обобщенный вектор)** | числовой, символьный, комплексный, логический, функция, выражение, формула | Да | это общая форма вектора, в котором различные элементы не обязаны быть одного и того же типа, и они сами по себе являются векторами. Списки предоставляют удобные возможности для возврата результатов статистических расчетов |
| **data.frame** | **фрейм данных** | числовой, символьный, комплексный, логический | Да | Таблица, состоящая из нескольких векторов одинаковой длины, но возможно различных типов. Это матрицеподобные структуры, в которых столбцы могут быть разного типа. Фреймы данных можно представить себе как «матрицу данных» со строкой на каждый наблюдаемый модуль и с, возможно, числовыми и категоризированными переменными. Многие эксперименты лучше всего описываются фреймами данных: воздействия являются категоризированными, а отклик - количественный |
| **ts** | **временной ряд** | числовой, символьный, комплексный, логический | Да | набор данных временного ряда, содержащий дополнительные атрибуты, такие как: частота и дата |

Предоставляемый в R набор типов переменных позволяет обеспечить удобное представление данных, адекватное условиям конкретной решаемой задачи.

Предоставляемый в R набор типов переменных позволяет обеспечить удобное представление данных, адекватное условиям конкретной решаемой задачи. Эти типы оказываются особенно полезными при решении задач, связанных с анализом данных, статистическими расчетами, построением моделей сложных объектов. Эти типы сочетают достаточную простоту в использовании с большими возможностями в работе с данными. Нетрудно заметить, что набор типов явно богаче, чем в MATLAB.

## Векторы и операции с ними.

Создание вектора из 5 чисел:

**>x<-c(10.4, 5.6, 3.1, 6.4, 21.7)**

Здесь **с()** – функция, составляющая вектор из нескольких компонент – простых переменных и/или векторов - от слова concatenate (соединять). Все элементы вектора должны быть одного типа. В зависимости от этого они называются числовыми, символьными, логическими, комплексными.

Теперь создадим вектор из 3-х компонент: 2-х векторов и числа 0:

**> g=c(x,0,x)**

- получится вектор с 11 элементами:

**> g**

**[1] 10.4 5.6 3.1 6.4 21.7 0.0 10.4 5.6 3.1 6.4 21.7**

Доступ к элементам вектора, как и в других языках программирования, осуществляется по указанию индекса, например, **g[3].** При отображении вектора значение в квадратных скобках в начале строки – это индекс первого элемента в строке. Использование индексов более подробно будет рассмотрено дальше.

Особая возможность – создание векторов - регулярных последовательностей:

**> y=1:30**

**> y**

**[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25**

**[26] 26 27 28 29 30**

**> z<-2\*3:12**

**> z**

**[1] 6 8 10 12 14 16 18 20 22 24**

**> 25:1->w**

**> w**

**[1] 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1**

Еще функция для создания векторов – регулярных последовательностей:

**> seq(-5, 5, by=.2) -> s3**

генерирует вектор **s3** с элементами -5.0, -4.8, -4.6, ..., 4.6, 4.8, 5.0.

**> s3**

**[1] -5.0 -4.8 -4.6 -4.4 -4.2 -4.0 -3.8 -3.6 -3.4 -3.2 -3.0 -2.8 -2.6 -2.4 -2.2**

**[16] -2.0 -1.8 -1.6 -1.4 -1.2 -1.0 -0.8 -0.6 -0.4 -0.2 0.0 0.2 0.4 0.6 0.8**

**[31] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0 3.2 3.4 3.6 3.8**

**[46] 4.0 4.2 4.4 4.6 4.8 5.0**

Аналогично

**> s4 <- seq(length=51, from=-5, by=.2)**

генерирует такой же вектор **s4**.

*Числовые векторы –* с элементами: целыми, вещественными или комплексными числами*.*

Простейшие арифметические операторы с векторами: **+, -, \*, /, ^.**

**> Q=1/x**

**> Q**

**[1] 0.09615385 0.17857143 0.32258065 0.15625000 0.04608295**

Получается вектор с 5 элементами – обратными значениями элементов из вектора **х**.

Дополнительно доступны обычные функции **log, exp, sin, cos, tan, sqrt, max, min**, и т.д.

**range()** – дает вектор с двумя значениями, равными минимальному и максимальному значениям в векторе-аргументе.

**length(x)** – число элементов в **х**

**sum(x) prod(x)** – сумма и произведение элементов **х**

**mean(x) var(x) sd(x)** – среднее, оценки дисперсии и стандартного отклонения по элементам **х**

Если аргументом **var()** является матрица **n**x**p**, то результатом будет ковариационная матрица **p**x**p**

**sort(x)** – упорядочивает вектор по возрастанию его элементов.

В любом случае при использовании этих функций нет разницы, являются числа целыми, вещественными или даже комплексными.

*Логические векторы* – с элементами TRUE, FALSE или NA (not available – отсутствующее значение).

**> xx<- x<6**

**> xx**

**[1] FALSE TRUE TRUE FALSE FALSE**

Обозначения логических операторов: **>, <, >=, <=, ==, !=**

При построении логических выражений используются знаки **&**(и), **|** (или) и **!**(не).

При использовании логических векторов в арифметических выражениях значение TRUE автоматически интерпретируется как 1, а FALSE – как 0.

*Символьные векторы* – с элементами - символьными строками, заключенными в одинарные или двойные апострофы.

В строках могут использоваться символы \(escape), \\(выводятся как \\), \" – для вывода двойной кавычки, \n = newline, \t=tab, \b=backspace.

Как и числовые, символьные векторы могут соединяться с помощью функции **с().**

Функция **paste()** с произвольным числом символьных аргументов соединяет их один за другим в общую символьную строку. Аргументы по умолчанию разделяются пробелами, но можно задать разделитель – аргумент **sep=string**.

Например,

**> labs <- paste(c("X","Y"), 1:10, sep="")**

создает символьный вектор

**> labs**

**[1] "X1" "Y2" "X3" "Y4" "X5" "Y6" "X7" "Y8" "X9" "Y10"**

но

**> labs <- paste(c("X","Y"), 1:10, sep=" ")**

**> labs**

**[1] "X 1" "Y 2" "X 3" "Y 4" "X 5" "Y 6" "X 7" "Y 8" "X 9" "Y 10"**

- появился пробел между Х и цифрой.

*Индексирование векторов*

Как и в MATLAB, подмножество элементов вектора создается присоединением к имени вектора индексного вектора в квадратных скобках. Эти индексные векторы могут быть 4-х видов:

1. Логический вектор в индексе:

**> d <- x[!is.na(x)]**

**> d**

**[1] 10.4 5.6 3.1 6.4 21.7**

Создаваемый вектор **d** будет содержать все элементы вектора **х**, кроме пропущенных (со значением NA), и потому **d** может быть короче **х**.

2. Вектор положительных целых значений. Например,

**> x[3]**

**[1] 3.1**

**> s3[1:10]**

**[1] -5.0 -4.8 -4.6 -4.4 -4.2 -4.0 -3.8 -3.6 -3.4 -3.2**

**> s3[c(4,2,23,11)]**

**[1] -4.4 -4.8 -0.6 -3.0**

3. Вектор из целых отрицательных значений. Такой вектор показывает, какие элементы должны быть исключены из результата.

Например,

**> t <- s3[-(1:5)]**

- даст все элементы кроме первых пяти

**> t**

**[1] -4.0 -3.8 -3.6 -3.4 -3.2 -3.0 -2.8 -2.6 -2.4 -2.2 -2.0 -1.8 -1.6 -1.4 -1.2**

**[16] -1.0 -0.8 -0.6 -0.4 -0.2 0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8**

**[31] 2.0 2.2 2.4 2.6 2.8 3.0 3.2 3.4 3.6 3.8 4.0 4.2 4.4 4.6 4.8**

**[46] 5.0**

4. Вектор с символьными строками. Эта возможность применима также при наличии объекта с именами атрибутов для идентификации его компонент.

Пример 1. Простой вектор с символьными компонентами. В отличие от MATLAB. Здесь не обязательно символьные строки должны иметь одинаковые размеры.

**> v<-c('c','de','s')**

**> v**

**[1] "c" "de" "s"**

Пример 2.

**> fruit <- c(5, 10, 1, 20)**

**> names(fruit) <- c("orange", "banana", "apple", "peach")**

**> lunch <- fruit[c("apple","orange")]**

**> lunch**

**apple orange**

**1 5**

## Факторы.

Фактор – это векторный объект, используемый для спецификации дискретной классификации (группирования) компонент другого вектора.

Пример.

**> fruits=c("яблоко","груша","слива","слива","груша","персик","яблоко","слива")**

Фактор создается с использованием функции **factor()** :

**> fruitsf<-factor(fruits)**

**> fruitsf**

**[1] яблоко груша слива слива груша персик яблоко слива**

**Levels: груша персик слива яблоко**

Уровни фактора – это набор неповторяющихся значений в факторе.

Для формирования отдельного вектора уровней фактора можно использовать функцию **levels()** .

**> levels(fruitsf)**

**[1] "груша" "персик" "слива" "яблоко"**

Пример. Пусть имеется числовой вектор со случайными целыми числами:

**> NN**

**[1] 3 1 2 2 4 4 2 3 4 3 1 3 2 2 3 4 3 1 2 2**

Преобразовав его в фактор, легко узнать, из каких неповторяющихся значений он состоит:

**> NNF=factor(NN)**

**> NNF**

**[1] 3 1 2 2 4 4 2 3 4 3 1 3 2 2 3 4 3 1 2 2**

**Levels: 1 2 3 4**

## Массивы и матрицы.

Массив может рассматриваться как сопровождаемая множеством индексов совокупность данных, например, числовых. R предоставляет простые возможности для создания и использования массивов и, в частности, специальных случаев - матриц.

Вектор может использоваться в R как массив только, если он имеет вектор размерности в качестве его атрибута **dim**.

Предположим для примера, что **z** – это вектор с 1500 элементами.

**> z=1:1500**

Присваивание

**> dim(z) <- c(3,5,100)**

создаст его **dim** - атрибут, который позволит работать с ним, как с массивом 3 на 5 на 100.

**> z**

**, , 1**

**[,1] [,2] [,3] [,4] [,5]**

**[1,] 1 4 7 10 13**

**[2,] 2 5 8 11 14**

**[3,] 3 6 9 12 15**

**, , 2**

**[,1] [,2] [,3] [,4] [,5]**

**[1,] 16 19 22 25 28**

**[2,] 17 20 23 26 29**

**[3,] 18 21 24 27 30**

**…….**

Например, если вектор размерности для массива **a** - это c(3,4,2), тогда в **a** содержится 3х4х2=24 элементов, и в массиве данных они будут следовать в порядке a[1,1,1], a[2,1,1], ...,a[2,4,2], …, a[3,4,2].

**> x <- array(1:20, dim=c(4,5)) # Создает массив (матрицу) 4 на 5 .**

**> x**

**[,1] [,2] [,3] [,4] [,5]**

**[1,] 1 5 9 13 17**

**[2,] 2 6 10 14 18**

**[3,] 3 7 11 15 19**

**[4,] 4 8 12 16 20**

Создадим теперь новый массив

**> i <- array(c(1:3,3:1), dim=c(3,2))**

**> i # i - это 3 на 2 индексный массив (матрица).**

**[,1] [,2]**

**[1,] 1 3**

**[2,] 2 2**

**[3,] 3 1**

Используя массив **i** в качестве индекса у массива **х**, извлекаем из него 3 элемента:

**> x[i]**

**[1] 9 6 3**

Здесь 9 – это элемент из 1-й строки и 3-го столбца массива **х**, 6 – из 2-й строки и 2-го столбца, 3 – из 3-й строки и 1-го столбца.

Теперь заменим указанные элементы из **х** на 0:

**> x[i] <- 0**

**> x**

**[,1] [,2] [,3] [,4] [,5]**

**[1,] 1 5 0 13 17**

**[2,] 2 0 10 14 18**

**[3,] 0 7 11 15 19**

**[4,] 4 8 12 16 20**

**Примеры матричных операций**

Матрица – это двумерный массив.

Пусть

**> A=array(c(1,2,-2,3,1,1),dim=c(3,2))**

**> A**

**[,1] [,2]**

**[1,] 1 3**

**[2,] 2 1**

**[3,] -2 1**

**> B=array(c(8,-2,-3,1,6,0,2), dim=c(3,2))**

**> B**

**[,1] [,2]**

**[1,] 8 1**

**[2,] -2 6**

**[3,] -3 0**

*Перемножение матриц:*

*Поэлементное произведение*

**> D=A\*B**

**> D**

**[,1] [,2]**

**[1,] 8 3**

**[2,] -4 6**

**[3,] 6 0**

*Матричное произведение*

**> E=A%\*%t(B)**

**> E**

**[,1] [,2] [,3]**

**[1,] 11 16 -3**

**[2,] 17 2 -6**

**[3,] -15 10 6**

- здесь **t(B)** – транспонирование матрицы **B**.

*Решение линейного уравнения:*

**> G=array(c(12,-2,8,17),dim=c(2,2))**

**> b=c(1.2,6)**

**> x=solve(G,b)**

**> x**

**[1] -0.1254545 0.3381818**

В среде R существует множество функций для выполнения матричных манипуляций: соединение матриц, обращение, вычисление собственных значений и векторов и т.д.

**Управляющие конструкции.**

Имеется набор стандартных средств создания условных конструкций – с ключевыми словами **if** и **switch**, а также для организации циклов – с ключевыми словами **for**, **while** и **repeat**. Поскольку здесь есть прямая аналогия с написанием таких конструкций в MATLAB, более подробно они здесь рассматриваться не будут.

**Ввод-вывод данных из файлов.**

Основные функции при работе с текстовыми файлами: **scan()/read.table()/write()/write.table()**. Также есть средства для работы с файлами, содержащими электронные таблицы, базы данных, графические представления и т.д.

## Программы на R

В программах на R могут использоваться как стандартные, так и пользовательские функции.

*Стандартные функции*

Стандартные функции предоставляются пользователю в готовом виде. Они объединены в пакеты. Часть этих пакетов с наиболее часто используемыми функциями непосредственно доступны сразу после начала сеанса работы с R. Другие – тоже присутствуют в комплекте, поставляемом при установке среды, но требуют отдельной активации, после чего они также становятся непосредственно доступными. Наконец, имеется огромное число (более 15000) пакетов, которые с использованием сети Интернет могут быть скопированы в установленную среду из архивов CRAN (The Comprehensive R Archive Network - Обширная сеть архивов R). Особенно много в этих пакетах представлено функций для статистических расчетов и анализа данных. Стандартные функции существенно сокращают трудозатраты на решение задач в среде R. В них представлены мощные функции, с использованием которых можно сразу решить достаточно сложные задачи. Например, с помощью вызова одной функции можно проверить статистические гипотезы о математических ожиданиях (критерий Стьюдента), дисперсиях случайных величин (критерии Фишера или Бартлетта), о виде закона распределения (критерий Колмогорова-Смирнова) и другие. Более того, за одно обращение к такой, например, функции, как lm(), можно построить регрессионную модель объекта по данным наблюдений и изучить ее статистические свойства.

Пусть, например, имеется 2 вектора: В1 и В2 с N наблюдениями случайных величин Х1 и Х2 в каждом. Тогда для проверки гипотезы о равенстве дисперсий этих случайных величин с уровнем значимости 0.05 и альтернативой: «не равны», достаточно выполнить оператор

**> print(var.test(B1,B2,conf.level=0.05))**

Пример результатов:

**F test to compare two variances**

**data: B1 and B2**

**F = 1.0142, num df = 49, denom df = 49, p-value = 0.9609**

**alternative hypothesis: true ratio of variances is not equal to 1**

**5 percent confidence interval:**

**0.9960682 1.0325964**

**sample estimates:**

**ratio of variances**

**1.014168**

При этом альтернатива “не равно» будет выбрана по правилу «умолчания».

Пример регрессионного анализа.

Пусть имеется фрейм данных (Таблица) X1X2Y с тремя полями: Х1, Х2 и Y и с некоторым числом строк (наблюдений). По этим данным рассчитаем регрессию Y по sin(Х1) и cos(Х2):

**> mod<-lm(Y~I(sin(X1))+I(cos(X2)),X1X2Y)**

**> print(summary(mod))**

Пример вывода результата:

**Call:**

**lm(formula = Y ~ I(sin(X1)) + I(cos(X2)), data = XYZ)**

**Residuals:**

**Min 1Q Median 3Q Max**

**-2.47768 -0.71850 -0.08733 0.60116 2.27888**

**Coefficients:**

**Estimate Std. Error t value Pr(>|t|)**

**(Intercept) 6.9690 0.1562 44.629 < 2e-16 \*\*\***

**I(sin(X1)) 2.8361 0.2190 12.950 < 2e-16 \*\*\***

**I(cos(X2)) -1.0743 0.2273 -4.726 2.11e-05 \*\*\***

**---**

**Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1**

**Residual standard error: 1.092 on 47 degrees of freedom**

**Multiple R-squared: 0.8049, Adjusted R-squared: 0.7966**

**F-statistic: 96.97 on 2 and 47 DF, p-value: < 2.2e-16**

Это – только несколько примеров, демонстрирующих большие возможности стандартных функций в среде R. Благодаря им можно решать многие практические задачи почти без программирования.

*Пользовательские функции*

Пользователи также могут создать структурные компоненты своих программ, оформив их в виде пользовательских функций. Пользовательские функции имеют вид

**<имя функции><-function(<список вх.параметров>){группа операторов}**

При этом группа операторов – это группа команд, заключенных в фигурные скобки. Возвращаемый параметр – это результат вычисления выражения. Рекомендуется результат обозначать с помощью функции **return(<возвращаемый параметр>)** – но это необязательно.

Например, функция для расчета нормированной автокорреляционной функции (на самом деле, в R есть готовые функции для расчета корреляционных функций, но для примера создадим свою):

**avtocor<-function(x,n){**

**#Расчет n ординат автокорр.функции по ряду х**

**N=length(x);xsred=mean(x)**

**ss<-var(x);sko<-sqrt(ss)**

**acf=c(1,seq(length=n-1,from=0,by=0))**

**for (i in 2:n){**

**r=0**

**for (j in 1:(N-i+1)){**

**r=r+(x[j]-xsred)\*(x[j+i-1]-xsred)**

**}**

**acf[i]=r/((N-i)\*ss)**

**}**

**return(acf)**

**}**

Любые совокупности команд, в том числе и функции, можно записать в файл с некоторым именем и расширением **R** в текущем каталоге (для каждой задачи рекомендуется заводить отдельный каталог). Как и в MATLAB, содержимое этих файлов называют модулями. Однако, в отличие от MATLAB здесь нет четкого разделения на скрипты (сценарии) и функции. В файл можно записать любое число команд и функций. Имя файла не обязано совпадать с именем функции.

Загрузка из файла – с помощью функции **source(<имя файла>).**

Например, запишем созданную выше функцию в файл **avtocor.R** В другой файл – **Main\_acf.R** – запишем команды

**v=c(11,2,34,4,56,2,-1,8,45) #Создание вектора с временным рядом**

**source("avtocor.R") #Загрузка функции из файла**

**ac<-avtocor(v,4) #Вызов функции на выполнение и запись результата в вектор ac**

**print(ac)**

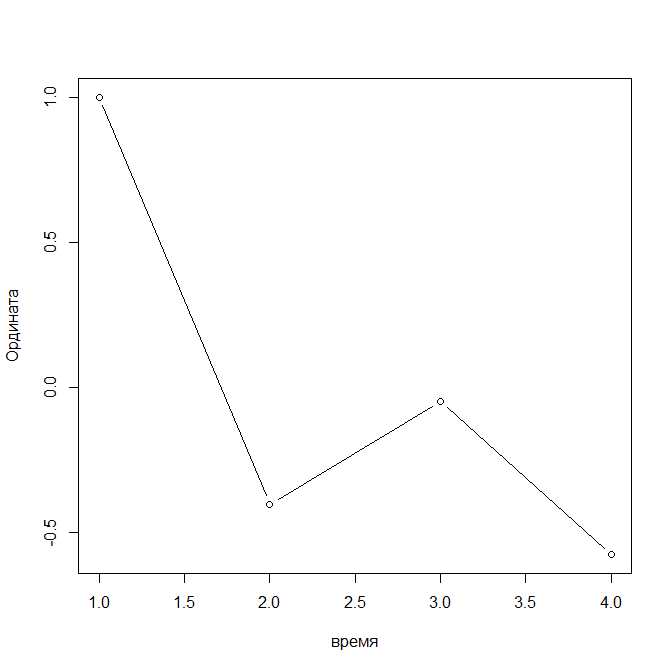
**plot(ac,type="b",xlab="время",ylab="Ордината") #Отображение АКФ в виде графика**

Здесь аргумент **type="b"** означает, что точки графика должны быть соединены прямыми линиями.

Получим результат: вывод значений ординат

**[1] 1.00000000 -0.40311328 -0.04770062 -0.57524018**

Построенный график



## Сравнение R и MATLAB.

Из приведенных сведений должно быть понятно, что у среды R имеется заметная аналогия со средой MATLAB.

К числу аналогий можно отнести:

- Сходство архитектур сред. Наличие рабочего пространства, истории команд, навигации.

- Похожие наборы способов решения задач.

- Сходство правил именования объектов.

- Совпадающие наименования многих функций (length, sort, mean, plot, sum, prod,…).

- Наличие дополнительных пакетов, снижающих потребность в программировании.

- Наличие матричных операций, уменьшающих необходимость написания циклов.

- Схожие способы оформления структурных компонент: сценариев (скриптов) и функций.

- Развитые средства графического представления результатов расчетов.

- Достаточно хорошо организованные службы помощи пользователям.

- Существование сообществ пользователей для каждой из этих сред, обеспечивающие поддержку развития и обмена опытом.

В то же время следует отметить ряд существенных отличий:

- Среда R – относится к свободно распространяемому ПО и развивается множеством разработчиков во многих странах мира. MATLAB – проприетарное и довольно дорогое ПО.

- Различие в предметной направленности: среда R явно ориентирована на анализ данных, статистические расчеты. Эти задачи характерны для исследования, оптимизации и разработки систем управления сложными объектами, в том числе, для систем поддержки принятия решений. MATLAB – ориентирован на инженерно-технические вычисления, моделирование систем, в том числе описываемых системами дифференциальных уравнений. Это делает его удобным при исследовании и разработке систем управления техническими объектами.

- В среде R представлено большее разнообразие типов переменных, что делает язык программирования более выразительным, приспособленным для написания более разнообразных и эффективных программ.

- В среде R предлагается гораздо больше дополнительных пакетов (более 15000), что обеспечивает возможность решения задач с низкими затратами на программирование за счет использования готовых и хорошо испытанных функций.

- MATLAB ориентирован, главным образом, на работу с числовыми данными. Он плохо приспособлен для работы с текстовой информацией или с категорийными данными. Среда R позволяет успешно работать, как с числовыми, так и с качественными (категорийными) данными.

*Сравнение возможности применения Toolboxes MATLAB и пакетов из CRAN (среда R).*

1) По стоимости: Tolboxes от фирмы MathWorks –платные и довольно дорогие.

Пакеты из CRAN – распространяются бесплатно.

2) По предметным областям: Toolboxes – в основном на технические задачи (управление техническими объектами, разработка, моделирование физических систем и систем управления, измерения, инженерные расчеты, включая матричные расчеты, решение алгебраических и интегро-дифференциальных уравнений, статистические расчеты и др.). В последнее время появились средства для других областей: финансовые расчеты, нейросети, анализ текстов. В Архиве CRAN – пакеты для анализа числовых и нечисловых данных в разных областях, в том числе – в биологии, медицине, социологии и других областях. В настоящее время, эти пакеты предоставляют средства для работы с «Большими данными».

3) По представительности: MathWorks предлагает около 60 различных Toolboxes. Еще некоторое число подобных дополнений распространяется, минуя эту фирму. Для среды R, как уже указывалось, предлагается через CRAN более 15000 пакетов разного назначения. Здесь проблема состоит в возможности среди этого моря программ найти нужные пакеты.

4) По возможности расширения и модификации. Toolboxes – по большей части не поддаются модификации и расширению. Однако часть функций имеют открытый код и доступны для изучения и модификации. Функции из пакетов CRAN – абсолютное большинство из них с открытым кодом и приспособлены для изучения и модификации. Впрочем, надо учитывать, что после этого потребуется их новое тщательное тестирование.

5) По надежности. Среда MATLAB развивается более 40 лет и за это время многие ее компоненты прошли многократную апробацию и верификацию. Учитывая то, что MATLAB фактически, стал стандартным инструментом для инженеров и исследователей, можно считать, он прошел массовое тестирование во многих практических приложениях. Среда R – более «молодая», но зато, в силу ее открытости и свободного распространения, многие ее пакеты также уже прошли массовое тестирование. К таким пакетам можно отнести, например, вычислительные пакеты STATS, STATS4, NLME, графические пакеты SPLINES, GRAPHICS, GRID, LATTICE.

6) По документированности. В обеих средах дополнительные пакеты снабжены вполне качественной документацией, достаточной для их самостоятельного освоения.

Таким образом, в рамках курса лекций дана информация о современных информационных технологиях и их возможностях. Более подробно рассмотрены 2 технологии, очень полезные для специалистов в области создания современных систем управления объектами разной сложности. В лабораторной части дисциплины даны первичные практические навыки работы со средой MATLAB.

# Новые информационные технологии

***Информационные технологии***

**Информационные технологии** (ИТ, англ. Information technology, IT) — широкий класс дисциплин и областей деятельности, относящихся к технологиям управления, накопления, обработки и передачи информации.

Информационная технология — процесс, использующий совокупность средств и методов сбора, накопления, обработки и передачи данных (первичной информации) для получения информации нового качества о состоянии объекта, процесса или явления (информационного продукта).

Этот процесс состоит из четко регламентированной последовательности выполнения операций, действий, этапов разной степени сложности над данными, хранящимися на компьютерах. Основная цель информационной технологии — в результате целенаправленных действий по переработке первичной информации получить необходимую для пользователя информацию.

Компонентами технологий для производства продуктов являются аппаратное (технические средства), программное (инструментальные средства), математическое и информационное обеспечение этого процесса.

В основном под информационными технологиями подразумевают компьютерные технологии. В частности, ИТ имеют дело с использованием компьютеров и программного обеспечения для хранения, преобразования, защиты, обработки, передачи и получения информации. По этой причине, специалистов по компьютерам часто называют ИТ-специалистами.

## Задачи, решаемые с помощью информационных технологий.

На рис. приведена схема, представляющая классификацию информационных технологий по основным задачам, определяющим суть этого понятия.

Информационные технологии

Получение данных

Накопление и хранение данных

Обработка данных

Защита данных

Преобразование данных

Передача данных

Управление

Рис.

## Классы программных систем

**Программное обеспечение** ***(допустимо также произношение «обеспечение») (ПО) — совокупность программ и программных документов, необходимых для эксплуатации этих программ (стандарт ГОСТ 19781-90).***

***Еще определение ПО — совокупность программ, процедур и правил, а также документации, относящиеся к функционированию системы обработки данных ( стандарт СТ ИСО 2382/1-84).***

В компьютерном сленге часто используется слово *софт* от английского слова *software*, которое в этом смысле впервые применил в статье в *American Mathematical Monthly* математик из Принстонского университета Джон Тьюки ( *John W. Tukey*) в 1958 году.

***Классификации ПО***

Программное обеспечение принято подразделять на классы:

***- по назначению: на системное, прикладное и инструментальное***,

***- по способу распространения и использования: на проприетарное (официально полученное «в собственность» за плату), арендованное, свободное и контрафактное***.

**- *по доступности исходного кода*:**  ***ПО с открытым (Open Source) и закрытым кодом***.

**- *по способу создания ПО***: ***обычная разработка, быстрая (RAD=Rapid Application Development), скоростная (Agile)***. Новые веяния: Low Code (с небольшими усилиями на программирование) и No Code (без прямого кодирования), AI (Artificial Intelligence – искусственный интеллект).

## Задача импортозамещения

**Импортозамещение** – это замена импортируемой продукции на отечественные аналоги.

Действительно, война санкций не кончается, и надо быть готовыми к любым неожиданностям. Так, по сообщениям СМИ, HP и Dell уже рассылали своим партнерам письма о запрете на поставку своих технологий в Крым. О санкциях против ряда российских банков тоже писали уже много. Так что тема импортозамещения вполне актуальна.

Без решения этой задачи невозможна независимая политика государства, под угрозу ставятся национальные интересы. Поскольку ИТ проникли во все сферы жизни, в любой момент возможно по командам извне нарушение нормального функционирования многих систем, в т.ч. и государственных органов (не говоря уж о прекращении получения обновленных версий продуктов и технологий). Понятие «цифрового суверенитета» - пример Китая.

ИТ-суверенитет более широкое понятие, чем импортозамещение. Оно включает в себя обеспечение кибербезопасности страны, возможность защиты своих критически важных отраслей от кибератак, от кибершпионажа, обеспечение функционирования Интернета. Но импортозамещение является очень важной составляющей понятия ИТ-суверенитета.

«Как показывает опыт последних лет, решения западных поставщиков уязвимы перед политической конъюнктурой. Можно вспомнить прецедент отказа от поддержки софта российских органов власти, банков и госкорпораций из-за санкций. Нужно преодолевать инерцию госзаказчиков, помогать им увидеть российские продукты, перейти на них.

В области ИТ у импортозамещения есть 2 аспекта: замещение аппаратных средств и ПО.

**Аппаратное обеспечение**

Снять с себя зависимость от зарубежных производителей программного обеспечения— это только половина дела. Другая, причем не менее важная,— обрести независимость от поставок аппаратных средств. А вот тут уже есть гораздо более серьезные проблемы.

Первая— сама возможность запрета на поставки компьютеров, серверов, сетевого и прочего связанного с ИТ оборудования. Риски возросли, и предприятия, прежде всего государственные, начали на них реагировать. При этом процесс импортозамещения принял форму западозамещения. На место поставщиков из США и Западной Европы приходят поставщики из Азии. Проще говоря — вместо Cisco имеем Huawei.

На первый взгляд это решение проблемы. Маловероятно, чтобы Китай присоединился к санкциям против России. Но все же это не импортозамещение. В истории наших взаимоотношений с Китаем бывало разное: и годы крепкой дружбы, и времена балансирования на грани войны. Лучше не зависеть ни от кого.

Тем более что есть вторая связанная с аппаратными средствами проблема— вероятность существования в них неописанных в документации возможностей, или, проще говоря, закладок. А это уже касается обороноспособности страны. И китайские закладки в аппаратных средствах в этом отношении ничуть не менее опасны, нежели американские.

Главная проблема— процессор. Это самое технологически сложное устройство, безопасность которого трудно проверить. Лучше использовать свои. И такие изделия у нас есть. Например, **четырехъядерный микропроцессор «Эльбрус-4С**» с тактовой частотой 800 МГц и пиковой производительностью 50 Гфлопс. На его основе созданы компьютеры «АРМ Эльбрус-401», ориентированные на нужды государственных предприятий. Дорого, конечно, но зато безопасно. Хотя остаются вопросы о происхождении остальных микросхем, используемых в компьютере. Ведь компьютер— это не только процессор.

Ещё проблема— отсутствие написанного для «Эльбруса» программного обеспечения. Существует эмулятор х86, который позволяет разворачивать на компьютере даже Windows, но, как и все эмуляторы, он сильно снижает производительность. И реальное быстродействие составляет процентов десять от номинального.

Разумеется, можно написать программное обеспечение под «Эльбрус», используя все преимущества его архитектуры. Но кто будет делать это? Для компьютеров, работающих на х86-процессорах, писали программы разработчики всего мира на протяжении многих лет. Можно написать отдельные специализированные программы под конкретные задачи для «Эльбруса». Но в целом проблему импортозамещения это не решит.

И ещё одна проблема: «Эльбрус»— не серверный процессор. Во всяком случае, в режиме совместимости с x86. Это процессор для рабочих станций.

Российская компания «Байкал Электроникс» выпускает первые инженерные образцы многоядерного процессора Baikal-Т1, рассчитанного на использование в широком спектре промышленных и потребительских устройств на рынках коммуникационных решений, промышленной автоматики и встроенных систем.  
Новейшие российские двухъядерные процессоры «Байкал Т1» с архитектурой MIPS разработаны по техпроцессу 28-нм и соответствуют самым современным требованиям к производительности и энергопотреблению промышленной электроники. Инженерные образцы Baikal-Т1 стали доступными для разработчиков начиная с 1 июня 2015 г. Процессор поставляется в комплекте со средствами разработки программного обеспечения.  
Процессоры разработаны в России, производство осуществляется на заводе на Тайване. В «Байкале» рассчитывают на интерес со стороны широкого числа отечественных и зарубежных организаций.  
Есть и другой путь— использовать **открытое аппаратное обеспечение**. Например, фирмы IBM. Корпорация IBM начала лицензирование процессорной архитектуры Power. Она может предоставить архитектуру этого процессора. На основе открытой RISC-архитектуры обладатели лицензии смогут создавать собственные чипы.

Кроме того, IBM создала альянс для разработки устройств и приложений, использующих платформу Power. Открытое общество разработчиков OpenPower включает 22 страны, 25 университетов, 113 компаний, среди которых Google, Nvidia, Samsung и Hitachi.

Открытое аппаратное обеспечение— это возможность самостоятельного воспроизведения. Если говорить о процессоре, то нужен текст на языке Verilog и фабрика, которая этот текст превращает в чип. Причем Power8 производятся по техпроцессу 22 нм. А фабрик, способных поддерживать такой техпроцесс, в мире очень мало.

И что мы будем делать с нашими процессорами на базе Power в случае введения новых санкций? Cами -то мы их сделать не можем. Как впихнуть 10 млн. транзисторов в чип, если наши технологии позволяют говорить только о ста тысячах? А прецеденты есть: в прошлогоднем апреле Госдеп США запретил Intel поставлять процессоры Xeon в Китай. Причем на Китай никто никаких санкций не накладывал.

**Программное обеспечение**

Здесь возможны 2 пути: создавать отечественные программы (дорого и долго - будет отставание от других стран) или использовать свобоно распространяемо ПО (СПО). **Для стимулирования разработки отечественного ПО в июне 2015 г. принят закон** «О внесении изменений в Федеральный закон «Об информации, информационных технологиях и о защите информации» и статью 14 Федерального закона «О контрактной системе в сфере закупок товаров, работ, услуг для обеспечения государственных и муниципальных нужд», в соответствии с которым создан и действует с начала 2016 г. Единый реестр российских программ для электронных вычислительных машин и баз данных (https://reestr.minsvyaz.ru/reestr/)**. На сегодняшний день в него включено более 6000 программных средств от разных разработчиков ( в т.ч. и от частных лиц).** По этому закону государственные организации и предприятия России при приобретении ПО должны в первую очередь отдавать предпочтение ПО из ЕРРПО, т.е. отечественному ПО. Изменения в правилах госзакупок вынуждают госзаказчиков обратиться лицом к российскому ПО, прежде всего включенному в Реестр. Теперь, когда заказчикам приходится обосновывать причину отказа от российских продуктов, выясняется, что причины для этого находятся не всегда. Однако, по-настоящему промышленных, тиражных российских программных продуктов, тем более таких, которые можно по формальным признакам разместить в Реестре, все-таки реально мало. К тому же у нас в стране распространена еще и заказная разработка ПО, которая к Реестру отношения не имеет, а доля такого рода ПО среди лотов госзакупок немалая. Для такого рода ПО критерии импортозамещения еще предстоит сформулировать.

**Свободно распространяемое ПО (СПО).** Такое ПО поддерживается, тестируется и развивается программистами из многих стран. Одним из обязательных условий СПО является наличие открытого кода (Open Source), доступного для изучения любыми пользователями. Это снижает (но не исключает на 100%) риск появления скрытых включений, могущих привести к нарушению нормального функционирования программ.

Разработку какого ПО государство должно поддерживать — софта, написанного с нуля; написанного с использованием открытого кода (СПО), права на который принадлежат российским компаниям; или иностранную программную разработку, доработанную российскими компаниями, которым в результате принадлежат права на разработку? Цель состоит все-таки в развитии собственной индустрии разработки ПО, а не в увеличении количества специалистов по «переклеиванию» лейблов на продуктах.

СПО позволяет развивать национальную базу инженеров-программистов, квалификация которых скорее относится к настройке клонов продуктов и платформ СПО, которые не являются по происхождению российскими, поскольку открытые лицензии на них не дают разработчику конечного продукта прав на код или его доработку, в чем, как он считает, собственно и заключается суть бизнеса разработки ПО.

К тому же использование СПО требует реальных компетенций в программировании. Без них поднять СПО до уровня продукта не получается, а их в стране пока недостает.

**Кадровые проблемы.** В стране нет достаточных ресурсов разработчиков для того, чтобы быстро заменить все используемое иностранное ПО. Качество и технологии разработки, которые используют российские производители ПО, не соответствуют международным критериям.

Эти проблемы напрямую относятся к области образования. Нужно менять программы подготовки программистов, приближая их к требованиям импортозамещения, т. е. учитывать присутствующие на рынке и успешно используемые отечественные программные продукты, включая облачные разработки.

Использование Open Source само по себе не решает проблем ИТ-суверенитета и импортозамещения.

Мы переходим от никак не управляемой нами ОС Windows к ОС Linux, на которой мы много что можем сделать. Есть некая теория, что Linuх разрабатывают миллионы сидящих по домам волосатых программистов, которые пишут код, отрывая время от жены и ребенка. В реальности 75% Linux сделано людьми, которые за это получают зарплату у себя на работе. И эти люди в основном работают в компаниях Intel, IBM, Broadcom, Novell, Microsoft. В России есть разработчики Linux, но все, что ими разработано, попадает в основное ядро по решению людей, которые находятся далеко не в нашей стране. И они могут это не принять. И если мы хотим развивать свое аппаратное обеспечение и хотим получить его поддержку в Linux, мы должны получать разрешение оттуда. И мы на это практически не влияем.

Linux является не совсем открытой ОС: современные аппаратные компоненты (видеокарты, дисковые контроллеры) снабжаются драйверами Linux, поставляемыми в бинарной форме. Это касается широко распространенных языков программирования Java и C#: Формально существуют версии обеих платформ на основе открытого кода — OpenJDK и Mono. Однако фактически управление языками осуществляют Oracle и Microsoft. Мы видим исходные тексты, можем их дорабатывать, но попадут или нет они в основную версию — решаем не мы.

Или другой нерадостный пример. В России разработан и выпускается архитектурно уникальный процессор для встроенных систем „Мультиклет“. Также для встроенных систем в России разработана ОС Embox. Для ее сборки нужен компилятор gcc. Доработка этой ОС для „Мультиклета“ требует глубокой модификации gcc, которую нам никто не позволит сделать. В итоге российская ОС не может быть качественно портирована на российский процессор. Мешает Open Source компилятор.

Однако не все так печально. Например, наши разработчики PostgreSQL занимают лидирующие позиции в Open Source-сообществе и реально влияют на принимаемые решения. Но таких областей мало.

Мы можем считать себя стратегически независимыми в каком-либо сегменте, если наши разработчики:

— принимают не менее 20%-ное участие в разработке СПО-продукта;

— реально являются важной частью команды и могут при необходимости поддерживать локальную ветвь продукта;

— могут влиять на направление развития продукта;

— разрабатывают код, который попадает в основную ветку продукта.

**Что делать**

Проблемы использования Open Source для обеспечения ИТ-суверенитета России были озвучены, были предложены и пути их решения. Прежде всего, это поддержка международных Open Source-продуктов, играющих важное значение для ИТ-суверенитета страны. Нужно продвигать свои команды, своих разработчиков, чтобы они могли влиять на развитие продукта в нужную нам сторону.

Ну и, конечно, нужно вкладываться в образование, в обучение разработчиков СПО, хотя бы для того, чтобы были люди, которых можно продвигать. «К сожалению, в России нет ни одного центра компетенции по таким крупным международным проектам, как OpenOffice, LibreOffice. Нет ни одного человека, который был бы готов оказать поддержку третьего уровня. Чтобы в России появилось хотя бы 10-20 человек, которые будут участвовать в таких проектах, понадобится не один год. Единственный путь — создание компетенций во всех ключевых компонентах. Причем не просто обучение теории и методам программирования. Настоящее обучение проходит при решении больших задач, выполнении серьезных проектов. Нужно, чтобы такие заказы, такие проекты были. И во всех этих вопросах государство вполне может помочь.

**Импортозамещение** – обеспечение «цифрового суверенитета» страны. Примеры, где требуется импортозамещение – Интернет, платежная система SWIFT.

2 направления решения проблемы импортозамещения: 1) Использование готовых программ из Реестра российского ПО, 2) Разработка новых программ на основе инструментов СПО.

Примеры отечественного ПО мирового уровня: антивирус Касперского, 1С – средства, СУБД PostgreSQL.

Главная задача: при импортозамещении не отстать от мирового уровня!

Проблемы импортозамещения: некоторые отечественные программы из Реестра уступают по возможностям зарубежным, дорогие, и часто базируются на зарубежных проприетарных платформах и, следовательно, не в полной мере могут считаться импортозамещающими.

Развитие информационных технологий***.***

Понятие новых информационных технологий (НИТ) меняется со временем. Сегодня – новые, завтра они устаревают. Развитие ИТ характеризуется очень быстрой динамикой. На рис. представлено в виде схемы то, что сегодня считается НИТ.

Рис.

**НИТ**

Облака

Big Data

Open Source

IoT (IIoT)

AI

**ЦОДы как технологическая основа НИТ.**

**ЦОД** = Центр Обработки Данных = Дата-Центр (DataCenter)

Традиционный ЦОД включает 5 элементов: Софт:(приложения + СУБД) + Аппаратные средства: (хост или сервер + система хранения данных + сеть)

Ключевые характеристики ЦОД: управляемость, масштабируемость, ёмкость, производительность, безопасность, доступность, целостность данных.

3 типа ЦОД: корпоративные, облачные и многоклиентские (MTDC=MultyTenantDataCenter).

Виртуализированный ЦОД – на той же технике, но с виртуализацией серверов.

***6.Виртуализация ресурсов.***

Смысл виртуализации – повышение эффективности капиталовложений за счет лучшего использования ВТ: без нее эффективность использования вычислительной системы 8-12%, а с ней – до 80-90%. Виртуальные системы работают на основе гипервизора

**Виртуализация**

***Виртуализация*** *– это технология, обеспечивающая абстрагирование процессов и их предоставления от вычислительных ресурсов. Понятие виртуализации далеко не новое и было введено ещё в 60-х годах  компанией IBM.*

Физический компьютер (Инфраструктура)

ОС

Операционная система (OS)(Платформа)

Приложения (APP) (софт)

Виртуальный компьютер

Гипервизор

Гипервизор (vSphere)

ОС1

OS1

APP1

ОС2

OS2

APP2

ОСn

OSn

APPn

Типичными представителями виртуальных машин являются Vmware Workstation и Microsoft Virtual PC, которые в основном используются в тестовых целях системными администраторами и разработчиками программного обеспечения. Компания Vmware выпустила также бесплатную версию Vmware Server, которая позволяет на одном физическом сервере запускать несколько виртуальных серверов, объединять их в сети, использовать их как обычные серверы в компании. Но данные продукты имеют один существенный недостаток  – это производительность. Так как все виртуальные машины работают как приложения, им соответственно выделяются ресурсы операционной системы, которая в свою очередь выделяет ресурсы физического сервера. В итоге производительность даже одной виртуальной машины оказывается в несколько раз ниже, чем производительность физического сервера.

Ситуация кардинально изменилась, когда в процессорах появилась аппаратная поддержка виртуализации – ***Гипервизор***. Были выпущены **Vmware Infrastructure (Vmware ESX Server)** и **Microsoft Hyper-V**. Эти  продукты являются полноценными **решениями виртуализации серверов**, позволяющие получить производительность виртуальных машин практически аналогичную производительности физического сервера на котором они запущены. Эти решения позволяют использовать виртуализацию в масштабах предприятия. Таким образом, можно на ограниченном количестве высокопроизводительных серверов создать довольно мощную среду, состоящую из сотен виртуальных серверов, на которых будут работать корпоративные приложения, ERP системы, сервера баз данных и т.д. В данный момент ни один ЦОД не обходится без систем виртуализации серверов.

**Преимуществ виртуализации**:

1. **Снижение затрат на оборудование**. Благодаря консолидации нескольких виртуальных серверов на одном физическом сервере, виртуализация позволяет значительно сократить расходы на серверное оборудование. На одном физическом сервере могут одновременно функционировать десятки виртуальных серверов.
2. **Снижение затрат на программное обеспечение**. Некоторые производители программного обеспечения ввели отдельные схемы лицензирования специально для виртуальных сред. Так, например, покупая одну лицензию на Microsoft Windows Server, предоставляется право одновременно использовать эту программу на 1 физическом сервере и 4 виртуальных (в пределах одного сервера), а Windows Server Datacenter лицензируется только на количество процессоров и может использоваться одновременно на неограниченном количестве виртуальных серверов.
3. **Снижение затрат на обслуживание**. Меньшее количество оборудования легче и дешевле обслуживать.
4. **Снижение затрат на электроэнергию**. В среднем 70% времени сервера работают в пустую, потребляя электроэнергию и выделяя большое количества тепла. Виртуализация позволяет более эффективно использовать процессорное время и увеличить утилизацию до 90%.
5. **Увеличение гибкости инфраструктуры**. Виртуализация позволяет программному обеспечению абстрагироваться от физического оборудования. Таким образом, появляется возможность миграции виртуальных машин между различными физическими серверами. Раньше при выходе сервера из строя приходилось переустанавливать ОС, восстанавливать данные из резервных копий, что занимало часы. Сейчас достаточно мигрировать виртуальную машину с одного сервера на другой, без каких-либо переустановок. Также как VmWare, так и Hyper-V предоставляют средства миграции физических машин на виртуальные.
6. **Повышение уровня отказоустойчивости**. Виртуализация предоставляет средства кластеризации целого сервера, независимо от работающего на нём программного обеспечения. Предоставляется возможность кластеризовать некластеризуемые сервисы. В данном случае получается не кластеризация в чистом виде, т.к. фактически происходит перезапуск виртуальной машины. Но в случае выхода из строя физического сервера, виртуальная машина автоматически запустится на другом сервере без вмешательства системного администратора.

**Недостатки виртуализации:**

* Высокие требования к квалификации обслуживающего персонала. Виртуальные технологии являются более сложными и их преимущества проявляются только при правильной организации обслуживания и при наличии обученного персонала.
* Снижение затрат получается не всегда, а только при реализации достаточно крупномасштабных вычислительных систем. Вряд ли она оправдана для вычислительной системы, состоящей из одного рабочего места.
* Внедрение виртуализации может приводить к некоторому повышению требований к конечным пользователям вычислительной системы. Впрочем, как показывает практика, они достаточно быстро привыкают к работе в виртуализированной среде.

## Облачные технологии

**Облачные технологии- определение NIST. 4 модели развертывания. 5 основных свойств. IaaS, PaaS, SaaS.**

**Сервисы и облака**

Определение сервиса, основанное на IBM Rational® Method Composer Plug-in for SOA Governance и на IBM Rational® Unified Process for Service-Oriented Architecture:

*"Сервис – это видимый ресурс, выполняющий повторяющуюся задачу и описанный внешней инструкцией".*

Для начала отметим неопределенность термина “облако”, который часто применяется как синоним понятия “облачные вычисления” (**cloud computing**) или как обозначение любой ИТ, доступ к которой выполняется через Интернет. Это неверно, поскольку “облако” означает “облачная инфраструктура”, т. е. ИТ - инфраструктура, обладающая некоторыми определенными свойствами. Об этом будет подробнее сказано ниже, но сразу отметим, что ключевым моментом тут является не способ доступа к ИТ-ресурсам (хотя это тоже важно), а организация ИТ-инфраструктуры (вычислительного процесса). И еще важно отметить, что “облачные вычисления” — это не решение и не проект, а концепция (модель) построения и использования ИТ-ресурсов. Конкретные решения и проекты могут лишь удовлетворять в той или иной мере этой модели, но сами по себе “облачными вычислениями” быть не могут.

Вариант формулировки, опубликованный **Национальным институтом стандартов и технологий CША (NIST)** в виде рекомендаций. руководящие материалы **NIST** предназначены для федеральных ведомств США, хотя могут использоваться и неправительственными организациями на добровольной основе.

***Облачные вычисления — это модель обеспечения повсеместного и удобного сетевого доступа по требованию к вычислительными ресурсным пулам (например, сетям, серверам, системам хранения, приложениям, сервисам), которые могут быть быстро предоставлены или выпущены с минимальными усилиями по управлению и взаимодействию с поставщиком услуг.***

**Основные свойства (5)**

**1. Самообслуживание по требованию (On-demand self-service).** У потребителя есть возможность получить доступ к предоставляемым вычислительным ресурсам в одностороннем порядке по мере потребности, автоматически, без необходимости взаимодействия с сотрудниками каждого поставщика услуг.

**2. Широкий сетевой доступ (Broad network access).**  Предоставляемые вычислительные ресурсы доступны по сети через стандартные механизмы для различных платформ, тонких и толстых клиентов (мобильных телефонов, планшетов, ноутбуков, рабочих станций и т. п.).

**3. Объединение ресурсов в пулы (Resorce pooling).** Вычислительные ресурсы провайдера объединяются в пулы для обслуживания многих потребителей по многоарендной (**multi-tenant**) модели. Пулы включают в себя различные физические и виртуальные ресурсы, которые могут быть динамически назначены и переназначены в соответствии с потребительскими запросами. Нет необходимости в том, чтобы потребитель знал точное местоположение ресурсов, однако можно указать их местонахождение на более высоком уровне абстракции (например, страна, регион или центр обработки данных). Примерами такого рода ресурсов могут быть системы хранения, вычислительные мощности, память, пропускная способность сети.

**4. Мгновенная эластичность (Rapid elasticity).** Ресурсы могут быть эластично выделены и освобождены, в некоторых случаях автоматически, для быстрого масштабирования соразмерно со спросом. Для потребителя возможности предоставления ресурсов видятся как неограниченные, то есть они могут быть присвоены в любом количестве и в любое время.

**5. Измеряемый сервис (Measured service).** Облачные системы автоматически управляют и оптимизируют ресурсы с помощью средств измерения, реализованных на уровне абстракции применительно для разного рода сервисов (например, управление внешней памятью, обработкой, полосой пропускания или активными пользовательскими сессиями). Использованные ресурсы можно отслеживать и контролировать, что обеспечивает прозрачность как для поставщика, так и для потребителя, использующего сервис.

**Модели облачных служб (3)**

1. **Инфраструктура как услуга (IaaS)**. Возможность предоставления потребителю систем обработки, хранения, сетей и других фундаментальных вычислительных ресурсов для развертывания и запуска произвольного программного обеспечения, которое может включать в себя операционные системы и приложения. Потребитель при этом не управляет базовой инфраструктурой облака, но имеет контроль над операционными системами, системами хранения, развернутыми приложениями и, возможно, ограниченный контроль выбора сетевых компонентов
2. **Программное обеспечение как услуга (SaaS).** Возможность предоставления потребителю в использование приложений провайдера, работающих в облачной инфраструктуре. Приложения доступны из различных клиентских устройств или через интерфейсы тонких клиентов, такие как веб-браузер (например, веб-почта) или интерфейсы программ. Потребитель при этом не управляет базовой инфраструктурой облака, в том числе сетями, серверами, операционными системами, системами хранения и даже индивидуальными настройками приложений за исключением некоторых пользовательских настроек конфигурации приложения.
3. **Платформа как услуга (PaaS)**. Возможность предоставления потребителю для развертывания в облачной инфраструктуре потребительских (созданных или приобретенных) приложений, реализованных с помощью языков программирования, библиотек, служб и средств, поддерживаемых провайдером услуг. Потребитель при этом не управляет базовой инфраструктурой облака, в том числе сетями, серверами, операционными системами и системами хранения данных, но имеет контроль над развернутыми приложениями и, возможно, некоторыми параметрами конфигурации среды хостинга. (например, хост с сетевыми экранами).

**Модели развертывания (4)**

1. **Частное облако (Private cloud).** Облачная инфраструктура, подготовленная для эксклюзивного использования единой организацией, включающей несколько потребителей (например, бизнес-единиц). Такое облако может находиться в собственности, управлении и обслуживании у самой организации, у третьей стороны и располагаться как на территории предприятия, так и за его пределами.
2. **Облако сообщества и коммунальное облако (Community cloud).** Облачная инфраструктура, подготовленная для эксклюзивного использования конкретным сообществом потребителей от организаций, имеющих общие проблемы (например, миссии, требования безопасности, политики). Облако может находиться в собственности, управлении и обслуживании у одной или более организаций в сообществе, у третьей стороны и располагаться как на территории организаций, так и за их пределами.
3. **Публичное (или общее) облако (Public cloud).** Облачная инфраструктура, подготовленная для открытого использования широкой публикой. Оно может находиться в собственности, управлении и обслуживании у деловых, научных и правительственных организаций в любых их комбинациях. Облако существует на территории облачного провайдера.
4. **Гибридное облако (Hybrid cloud).** Облачная инфраструктура представляет собой композицию из двух или более различных инфраструктур облаков (частные, общественные или государственные), имеющих уникальные объекты, но связанных между собой стандартизованными или собственными технологиями, которые позволяют переносить данные или приложения между компонентами (например, для балансировки нагрузки между облаками).

**Преимущества облачных структур**

* Снижение затрат на владение информационными технологиями (создание, сопровождение, обслуживание, обеспечение безопасности…)
* Возможность получить тот объем и уровень (**SLA-Service Level Agreement**) сервисов и в тот момент, когда они нужны.
* Возможность сменить провайдера облачных услуг.

Недостатки:

* Проблема безопасности данных
* Требуются администраторы, владеющие этими технологиями
* Преимущества проявляются только при грамотной организации работы в облаках

## Большие данные.

Понятие «Большие данные» — калька англоязычного термина **Big Data**. Единого определения для этого понятия не существует, но, как правило, под ним подразумевают специальные методы и средства анализа большого объема данных, поступающих с высокой скоростью из различных источников. Объемы настолько большие, данные появляются с такой частотой и из такого количества источников, что требуются специальные методы и технологии для понимания и интерпретации получаемой информации. Традиционные системы управления для этого не годятся.

Под термином **Big Data** в различных контекстах сегодня понимают и сами большие данные, и технологии их обработки, сформировавшиеся за последние несколько лет. Первыми заказчиками и за рубежом, и в России оказались банки, телекоммуникационные компании и ритейлеры. В сферу государственного управления проекты с применением технологий **Big Data** пришли только недавно.

Главная отраслевая особенность здесь связана с назначением и характером прикладного использования технологий больших данных. Ведь в госсекторе накопились огромные массивы данных, в том числе неструктурированных, которые надо обрабатывать, чтобы извлекать полезные знания. Поэтому со стороны госструктур мы наблюдаем растущий интерес к такого рода решениям.

Принципиальным отличием проектов **Big Data** в госсекторе является то, что в качестве успеха проекта рассматривается не коммерческая выгода, а способность решать важные социальные задачи, создавать общественное благо. К примеру, с помощью этих технологий можно обеспечить персонализацию государственных сервисов и услуг, предвидение и предотвращение угроз национальной **безопасности**, катастроф и стихийных бедствий, выполнение различных программ в области здравоохранения и социальной защиты, предотвращение попыток мошенничества и преступлений в области налогового законодательства и многое другое.

Другой класс задач связан с лингвистическим анализом текстов, семантическим поиском, извлечением дополнительных знаний из неструктурированных документов и пр. Это пока не массовые, а скорее штучные проекты, но со временем мы перейдем и к типовым промышленным решениям.

**Назначение:** для глубокого анализа (**Data Mining**) и получения нетривиальных и иногда неожиданных результатов.

***Признаки больших данных*** (VVV)

* **Представление данных** (**Variety**): слабо структурированные и разнородные.
* **Объем** (**Volume**): действительно большие (хотя размер зависит от доступных ресурсов для их обработки).
* **Скорость** (**Velocity**): обрабатывать надо очень быстро (причем и результаты часто нужны оперативно, если речь об онлайновых сервисах).

Главные задачи **Big Data** — это анализ мощных потоков данных в реальном времени или использование математических методов для поиска скрытых закономерностей и построения прогнозных моделей поведения тех или иных сущностей.

Единицы измерения объема данных: Мегабайт (106), гигабайт (109), терабайт (1012), петабайт(1015 байт), эксабайт (1018), зеттабайт (1021), йоттабайт (1024)

**Большие данные, примеры:** Применения могут быть самыми разнообразными. Например,

1. **Сайт ancestry.com** пытается построить семейную историю всего человечества, основываясь на всех доступных на сегодняшний день типах данных: от рукописных записей во всевозможных книгах учета до ДНК-анализа. На сегодняшний день им удалось собрать уже около пяти миллиардов профилей людей, живших в самые разные исторические эпохи, и 45 миллионов генеалогических деревьев, описывающих связи внутри семей.
2. **General Electric** производит в т.ч. двигатели для самолётов и собирает информацию об их эксплуатации – для их улучшения. За один полет – гигабайты данных. А данные собирают по всему миру. Далее – техника работы с Big Data и Data Mining.
3. Страховая кампания **InTouch** – на мобильное устройство клиента скачивается приложение, собирающее информацию о его вождении авто (в процессе езды). Далее – анализ данных и до 20% скидка на страхование.
4. Генетические исследования.

Геном расшифрован примерно 15 лет назад (2003г) – исследование, стоившее миллиарды долларов.

Поставлена задача – получать геном отдельного человека, не более, чем за $1000. Эта задача решена в январе 2014 г. компанией Illumina’s HiSeqX.

В США поставлена цель – получить 100000 геномов.

1 геном – это ок.100 ГБ, но значимая часть – 10-20 ГБ.

Фирма Genomics England в 2013 получила финансирование 100 млн.фунтов стерл. на создание 100000 геномов - объем 100 петабайт (1015 байт)

Такие исследования – по всему миру (Германия, Катар, Европейский институт биоинформатики). Прогноз – объем геномной информации скоро составит около 1 эксабайта (Broad Institute: MIT + Harvard.)

**Чем крупнее компания, тем выше её потребность в специалистах по обработке больших данных.**

**Эксперты Superjob считают, что специалистов по работе с Big Data можно разделить на две основные категории:**   
1) инженеры Big Data, отвечающие в основном за хранение, преобразование данных и быстрый доступ к ним;   
2) аналитики Big Data, отвечающие за анализ больших данных, выявление взаимосвязей и построение моделей.   
  
В связи с большими данными обычно указывают 2 инструмента: СУБД HANA и система **Hadoop**.

**Hadoop**

**Hadoop** — проект фонда **Apache Software Foundation**, свободно распространяемый набор утилит, библиотек и программный каркас для разработки и выполнения распределённых программ, работающих на кластерах из сотен и тысяч узлов. Используется для реализации поисковых и контекстных механизмов многих высоконагруженных веб-сайтов, в том числе, для Yahoo и Facebook. Разработан в рамках вычислительной парадигмы **MapReduce**, согласно которой приложение разделяется на большое количество одинаковых элементарных заданий, выполнимых на узлах кластера и естественным образом сводимых в конечный результат.

Hadoop состоит из трёх подпроектов — **Hadoop Common** (связующее программное обеспечение — набор инфраструктурных программных библиотек и утилит, используемых для других подпроектов и родственных проектов), **HDFS** (распределённая файловая система) и **Hadoop MapReduce** (фреймворк для реализации MapReduce-вычислений), ранее в Hadoop входил целый ряд других проектов, ставших самостоятельными в рамках системы проектов Apache Software Foundation.

**SAP HANA** (**H**igh-Performance **An**alytic **A**ppliance) — это высокопроизводительная **NewSQL** платформа для хранения и обработки данных, в основе которой лежит технология вычислений in-memory с использованием принципа поколоночного хранения данных, платформы, разработанной и выведенной на рынок транснациональной корпорацией **SAP SE**. Архитектура **HANA** обеспечивает как высокоскоростную обработку транзакций, так и работу со сложными аналитическими запросами, совмещая решение этих задач в рамках единой платформы.

Кроме самой СУБД, в составе **HANA** предусмотрены встроенный веб-сервер и репозиторий управления версиями, необходимые для разработки приложений. Приложения **HANA** могут создаваться с использованием кода JavaScript на стороне сервера и HTML-кода.

Теперь самое время посчитать затраты компании и прибыль, принесенную ей комплексом Big Data. Можно сказать наверняка, что если Hadoop используется не только для дешевого хранения, но и для построения аналитики на Big Data, то создание полноценного отдела по обработке данных может оказаться экономически неоправданным.

## Интернет вещей и промышленный интернет вещей (IoT и IIoT)

**Интернет вещей (IoT=Internet of Things) и Промышленный интернет вещей (IIoT=Industrial Internet of Things)**

**Интернет вещей**  — это концепция вычислительной сети физических объектов (*«вещей»*), оснащённых встроенными технологиями для взаимодействия друг с другом или с внешней средой, рассматривающая организацию таких сетей как явление, способное перестроить экономические и общественные процессы, исключающее из части действий и операций необходимость участия человека.

**Интернет вещей**. Это новый этап развития Интернета, значительно расширяющий возможности сбора, анализа и распределения данных, которые человек может превратить в информацию и в знания.

Есть принципиальное отличие технологий IoT и IIoT. Название первой технологии расшифровывается как «Интернет вещей», а второй — «Промышленный Интернет вещей». Наполнение этих терминов различно, хотя они и характеризуются определенным родством на концептуальном уровне.

IoT применяется в быту, а IIoT — в промышленности. Технологии развивались параллельно, а потому что-то заимствовали друг у друга. Общая основа — универсальные стандартизированные протоколы, которые позволяют установить взаимодействие между устройствами, выпущенными разными производителями. Эти устройства могут обмениваться данными, анализировать информацию и принимать решения на базе полученных сведений.

**IoT и IIoT: основные отличия**

* + Цель введения. Технология IoT появилась с целью улучшения уровня жизни людей. Интернет вещей непосредственно связан с бытом, он позволяет расширить возможности экономии средств и повышения безопасности жизни. Главная цель IIoT — производственные выгоды и рост рентабельности компании.
  + Историческое развитие. Технология IoT — качественно новый этап эволюции «умного» дома, тогда как IIoT — результат развития телематических и телеметрических систем на производстве.
  + Экономический аспект устранения поломок. Для устройств и систем, функционирующих в рамках IoT, характерна низкая стоимость выхода из строя. В случае с IIoT последствия поломок механизмов или систем, входящих в сеть, являются более масштабными и значимыми.
  + Стоимость устройств и платформ. Системы, сформированные на базе IoT-устройств, являются сравнительно недорогими. В свою очередь IIoT-решения имеют более высокую стоимость, поскольку используются на производстве на базе различных платформ, решений и дорогостоящего оборудования.
  + Скорость внедрения. Внедрение IoT-платформ и систем происходит очень быстро. Все «умные» устройства покупаются во время ремонтных работ при замене уже существующих технических компонентов. Внедрение IIoT-платформ и систем идет более медленными темпами.
  + Обслуживание систем. Стоимость обслуживания IoT-устройств невысока. Нужно лишь найти технический компонент, вызывающий проблемы, и заменить его. В случае с IIoT-системами обслуживание является комплексным и дорогостоящим.
  + Сложность производимых расчетов. В случае с IoT-устройствами при расчетах применяются базовые алгоритмы, которые помогают автоматизировать повторяющиеся действия. Работа IIoT-платформ и систем основана на применении сложных алгоритмов, которые базируются на серьезном математическом фундаменте.
  + Обработка данных. Информация, которая поступает на IoT-технику, обрабатывается в реальном времени, но иногда возможны некоторые задержки в поступлении данных. В IIoT-устройствах любое запаздывание информации нежелательно, все должно происходить с точностью до десятка миллисекунд.
  + Объемы трафика. Трафик IoT-устройств невысок, обычно он не превышает нескольких мегабайт в сутки. Для IIoT-устройств характерны высокие объемы трафика ― вплоть до 1 петабайта в сутки.
  + Влияние на занятость населения. Устройства «умного» дома не создают новых вакансий и не приводят к увольнениям. Внедрение IIoT приводит к сокращению рабочих мест за счет автоматизации большинства повторяющихся действий.

Сравнение показывает, что каждая из технологий охватывает свою собственную аудиторию и решает свои характерные задачи. Главная цель потребительского Интернета вещей — создать комфортные условия для человека, а уже потом обеспечить экономию, снизить расходы. На базе IoT работают бытовые приборы, системы отопления и освещения, кондиционеры, дверные замки, видеонаблюдение, а также автоматические системы полива прилегающей территории. Благодаря внедрению IoT расширяются возможности оптимизации энергопотребления. Все затраты можно проконтролировать и оптимизировать.

Задача промышленного Интернета вещей ― рост производительности процессов предприятия за счет оптимизации энергопотребления. Технология IIoT применяется в различных отраслях промышленности. В рамках этой технологии работу механизмов и систем на производстве постоянно отслеживают датчики, что позволяет эффективно управлять всеми процессами и моментально реагировать на различные ситуации.

*Как устроен IoT?*​По сути, это сеть сетей, в которой люди могут общаться с устройствами, а устройства могут общаться между собой, реагировать на изменение окружения и принимать решения без участия человека. IoT-устройства функционируют самостоятельно, хотя люди могут настраивать их или предоставлять доступ к данным. IoT-системы работают в режиме реального времени и обычно состоят из сети умных устройств и облачной платформы, к которой они подключены с помощью WiFi, Bluetooth или других видов связи. Сначала устройства собирают данные — например, о температуре в квартире или частоте сердцебиения пользователя, затем эти данные отправляются в облако. Там программное обеспечение обрабатывает их. Что происходит, когда температура оказывается слишком высокой или в доме находится грабитель? Система оповещает об этом пользователя или сама выполняет дальнейшие действия — например, звонит в полицию

*Чем IoT полезен человеку?*Интернет вещей у многих ассоциируется с «умным» домом. Благодаря технологиям и устройствам, разработанным компаниями Google, «Яндекс», Amazon, Apple и другими, пользователи могут совершать онлайн - покупки, регулировать температуру в комнате, включать свет и музыку, отдавая голосовые команды виртуальным помощникам. Вам больше не надо опасаться, что вы забыли выключить утюг или кран — достаточно нажать кнопку в смартфоне, и «умный» дом все исправит. Система наблюдения с помощью компьютерного зрения распознает всех, кто проходит мимо вашей квартиры, и сравнит изображения с базой полиции. Сегодня «умный» дом в России — это в основном интеллектуальный помощник Яндекса «Алиса», которая включает музыку, ищет информацию в интернете, советует фильмы, регулирует освещение и температуру в доме, включает чайник.

IIoT позволяет компаниям автоматизировать процессы и снижать трудозатраты. Это сокращает объем отходов, улучшает качество предоставляемых услуг, удешевляет процесс производства и логистику. IIoT затрагивает все отрасли. Внедрение интернета вещей в электроэнергетике улучшает контролируемость подстанций и линий электропередачи за счет дистанционного мониторинга, а в здравоохранении позволяет перейти на новый уровень диагностики заболеваний — «умные» устройства контролируют показатели здоровья пациента. В сельском хозяйстве «умные» фермы и теплицы сами дозируют удобрения и воду. Внедрение IIoT в логистике сокращает затраты на грузоперевозки и минимизирует влияние человеческого фактора. IIoT активно внедряют нефтегазовые и горнодобывающие отрасли. Например, применение углубленной аналитики по буровым скважинам помогает нефтегазовой промышленности увеличить объемы добычи на уже отработанных месторождениях. А интернет вещей в транспорте — это сам транспорт, электронные табло, навигаторы, системы безопасности, камеры наблюдения, которые взаимодействуют между собой.

**IoT и IIoT и возможные преграды**

Основная преграда на пути к широкомасштабному внедрению Интернета вещей в России — отсутствие необходимых законодательных норм и актов, регламентирующих правомерность использования технологий, связанных с IoT. Сюда следует отнести вопросы доступности информации в глобальных сетях, получения и анализа данных, относящихся к сфере личной жизни человека. Все персональные данные, согласно законам РФ, должны передаваться только при наличии письменного согласия. Этот факт должен быть стандартизирован и включен в технологические нормы, которые устанавливает государство.

Еще одно препятствие ― угроза атак со стороны злоумышленников. В качестве примера можно привести атаки «нулевого дня», когда вредоносное ПО разрабатывается с ориентацией на конкретную жертву. Взломщики могут обесточить целый квартал «умных» домов, получив доступ к сети, в которой происходит обмен информацией между устройствами.

Если говорить об угрозах для применения промышленного Интернета вещей, предприятие, функционирующее с использованием IIoT, может быть полностью парализовано в результате кибератаки. В связи с этим назревает вопрос о проработке законодательства в сфере онлайн - преступности применительно к IoT и IIoT, что невозможно осуществить без тесного взаимодействия с девелоперами и поставщиками технологий.

**Позитивный эффект от внедрения и перспективы**

Появление в розничной продаже большого количества бытовой техники, электроники, «умных» гаджетов, оснащенных IoT, несомненно, способствует оживлению розничного сегмента рынка, увеличению объемов торговли в денежном эквиваленте. «Умные» устройства ежедневно помогают тысячам людей в бытовых вопросах, экономят время, упрощают многие рутинные процедуры, помогают оптимизировать затраты на коммунальные расходы.

Еще больший эффект мы видим от внедрения IIoT. Согласно прогнозам PWC, общий эффект от внедрения промышленного Интернета вещей для российской экономики составит около 2,8 трлн. руб. к 2025 г. Эксперты полагают, что главный рывок будет сделан за счет электроэнергетики и ритейла. Также произойдет модернизация транспортной и логистической систем, откроются новые возможности для улучшения работы банковского сектора. Все это приведет к сокращению персонала, исполняющего рутинные операции, к устранению человеческого фактора при выполнении различных операций, к росту общих показателей автоматизации бизнеса.

В числе основных предпосылок для повсеместного внедрения компонентов IIoT следует назвать углубление автоматизации производственных процессов, необходимость централизованного мониторинга производственных активов для повышения эффективности технологических операций, стандартизацию, отраслевое регулирование. Стоимость устройств также пока не идет на пользу рынку, появление доступных датчиков существенно ускорит переход предприятий к IIoT

Концепция и термин для неё впервые сформулированы основателем исследовательской группы Auto-ID  при Массачусетском технологическом институте Кевином Эштоном в 1999 году на презентации для руководства Procter & Gamble. В презентации рассказывалось о том, как всеобъемлющее внедрение радиочастотных меток сможет видоизменить систему управления логистическими цепями в корпорации.

Что такое Интернет вещей? Датчики собирают информацию, которая через интернет-каналы отправляется в центр, где данные обрабатываются. Десять лет назад уже был Интернет, были датчики, базы данных и аналитические системы. А вот чего тогда не было, так это баз данных in-memory, а также платформ для анализа этих данных в реальном времени и предиктивной аналитики. То есть данные собирали и отправляли, на их основе можно было принимать решения, но отсутствовали технологии и платформы, чтобы собирать их в мощных облаках с правильными протоколами на промышленной основе, отсутствовали системы машинного обучения, которые позволяли бы не просто взять данные, а мгновенно принять бизнес-решение и запустить транзакцию.

В отчёте Национального разведывательного совета США ( *National Intelligence Council*) 2008 года «интернет вещей» фигурирует как одна из шести потенциально разрушительных технологий, указывается, что повсеместное и незаметное для потребителей превращение в интернет - узлы таких распространённых вещей, как товарная упаковка, мебель, бумажные документы, может нанести урон национальной информационной безопасности.

По прогнозам Gartner, к 2021 году к интернету будет подключено 25 млрд устройств. Киберпреступники будут продолжать атаковать их, потому что IoT-система — это надежный и быстрый способ распространить вредоносное ПО. Рядовые пользователи, компании и целые города будут все чаще применять интеллектуальные технологии, чтобы сэкономить время и деньги. Например, холодильники смогут предупреждать о скорой порче продуктов, светофоры со встроенными видеодатчиками будут регулировать дорожное движение в зависимости от трафика. Сейчас, однако, ключевая проблема внедрения IoT — отсутствие единых стандартов. Поэтому имеющиеся решения сложно интегрируются между собой, а новые появляются медленнее, чем могли бы. Еще один нюанс — «вещи» в интернете вещей должны быть автономны, то есть иметь возможность получать энергию из окружающей среды, без участия человека.

**Интернет вещей, который считают основой четвертой индустриальной революции, стирает разницу между городом и деревней, д**елая последнюю местом, куда более приспособленным для жизни, работы и творчества. Согласитесь, что жить в небольшом населенном пункте на свежем воздухе (питаясь при этом экологически чистыми продуктами с ближайшей фермы) и не стоять в многокилометровых автомобильных пробках значительно лучше, чем проводить время в мегаполисе

## Искусственный интеллект

(**ИИ**; [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Artificial intelligence, AI*) — это:

* [наука](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%83%D0%BA%D0%B0) и технология создания интеллектуальных машин, особенно интеллектуальных [компьютерных программ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0);
* Свойство интеллектуальных систем выполнять творческие функции, которые традиционно считаются прерогативой человека

**Признаки ИИ:**

- способность к саморазвитию на основе обучения;

- способность рассуждать;

- способность оперировать не только с данными, но и со знаниями

К ИИ обычно относят решение следующих задач:

1. Обучение (приобретение, накопление знаний и новых функций) и коррекция с их использованием своего поведения,
2. Нахождение (выявление) закономерностей в имеющихся данных и использование их в управлении своим поведением,
3. Принятие в соответствии с некоторыми целями решений, влияющих на свое поведение или на некоторые внешние объекты,
4. Использование в управлении своим функционировании абстрактных, качественных понятий (классификация),
5. Прогнозирование развития ситуаций
6. Понимание естественного языка

**Машинное обучение** (англ. *Machine Learning=ML*) — один из классов методов искусственного интеллекта, характерной чертой которого является не прямое решение задачи, а обучение в процессе применения решению множества сходных задач. Для построения таких методов используются средства математической статистики, численных методов, методов оптимизации, теории вероятностей, теории графов, различные техники работы с данными в цифровой форме.

Различают два типа обучения:

1. *Обучение по прецедентам*, или *индуктивное обучение*, основано на выявлении закономерностей в эмпирических данных.
2. *Дедуктивное обучение* предполагает формализацию знаний экспертов и их перенос в компьютер в виде базы знаний.

Дедуктивное обучение принято относить к области экспертных систем, поэтому термины *машинное обучение* и *обучение по прецедентам* можно считать синонимами.

Многие методы индуктивного обучения разрабатывались как альтернатива классическим статистическим подходам. Многие методы тесно связаны с извлечением информации (*Information extraction*), интеллектуальным анализом данных (Data mining).

Классические задачи, решаемые с помощью машинного обучения:

* Классификация как правило, выполняется с помощью обучения с учителем на этапе собственно обучения.
* Кластеризация как правило, выполняется с помощью обучения без учителя
* Регрессия как правило, выполняется с помощью обучения с учителем на этапе тестирования, является частным случаем задач прогнозирования.
* Понижение размерности данных и их визуализация с помощью обучения без учителя
* Восстановление плотности распределения вероятности по набору данных (не оценка параметров!)

*Связь ИИ и Больших данных.*

ИИ для бизнеса следует рассматривать не только как средство для осмысления накопленной информации, но и дисциплину, работа которой зависит от наличия больших массивов данных, поэтому вполне естественно, что сегодня большие данные и ИИ часто ассоциируются друг с другом. «Безусловно, между большими данными и ИИ существует тесная связь, — утверждает Жан-Поль Баритуго, директор компании Pace Harmon, которая оказывает консалтинговые услуги в сфере трансформации бизнеса и аутсорсинга. — Большие данные — это топливо, а ИИ — это средство достижения цели».

Ниже приводятся разъяснения экспертов, которые помогут уловить связь между ИИ и большими данными.

**А. Некоторые разновидности ИИ обходятся без больших данных**

По словам Баритуго, чтобы раздобыть достаточное количество «хороших» данных и, как следствие, получить значительную отдачу от инвестиций в ИИ, применяется концепция «мусор на входе — мусор на выходе» (garbage in, garbage out). Но какой объем данных для этого требуется — зависит от конкретной ситуации. «В традиционном понимании большие данные — большие наборы структурированных и неструктурированных данных. В некоторых сценариях они выступают в качестве источника для обучения ИИ, анализа информации, чтобы определить закономерности и найти вероятности, которые помогут с ответами на ваши вопросы. Как правило, для этих случаев требуется большой объем данных, — объясняет исполнительный вице-президент и заслуженный аналитик Everest Group Сара Бернет. — Но не для всех типов ИИ требуется много данных».

**Б. Не все большие данные требуют применения ИИ**

ИИ может помочь в проведении анализа, но он не обязательно нужен для того, чтобы извлечь из больших данных значимые выводы. «Организации в течение многих лет задействовали углубленную аналитику. Все зависит от объема и количества различных наборов данных, которые необходимо проанализировать, — сказал директор по когнитивной автоматизации и инновациям ISG Уэйн Баттерфилд, — Подобрать содержательные паттерны в огромных наборах данных за разумный промежуток времени не под силу даже величайшим в мире умам, поэтому переложить эту тяжелую работу на МО представляется разумным решением, но при этом не все наборы — огромные или отличаются особым разнообразием, и, следовательно, необходимость в МО не столь уж очевидна». ИТ-департаменты могут также применять для анализа и визуализации аналитических данных BI-решения, аналитику и предметно-ориентированные информационные база данных.

**В. Углубленная аналитика и ИИ — это не одно и то же**

Очень часто термин «большие данные» применяется для того, чтобы подать углубленный анализ информационных активов в более глубоком контексте. С одной стороны, в глазах профессионалов это нормально, но с другой, у неподготовленных людей могут возникнуть ошибочные ассоциации — они могут подумать, что углубленная аналитика и ИИ — взаимозаменяемые терминами. «Несмотря на то, что ИИ и расширенная аналитика тесно связаны, между ними есть ключевые различия, — говорит Бернет. — Например, ИИ умеет проверять предположения, самостоятельно учиться и совершенствовать свой анализ. Аналитические решения умеют анализировать данные, но они не могут самообучаться, и чтобы настроить параметры их работы, без вмешательства людей не обойтись».

**Г. Большие данные могут искажать ИИ-модели**

«Большие данные создают основу для ИИ и МО. Чем больше данных — тем лучше будут модели, — поведал Франко из Talend. — Но при отсутствии должного контроля они могут привносить в выводы ИИ и МО элемент предвзятости». Одной из причин этого может являться акцент на количестве, а не на качестве данных. «ИИ и МО неизбежно потерпят неудачу, если люди не смогут контролировать исходные данные. Стекание огромных объемов информации в озеро данных не является достаточным основанием, гарантирующим успех этих технологий», — добавил он.

**Д. Организации уже связывают ИИ и большие данные, но даже не подозревают этого**

«На рынке уже встречаются программные решения с встроенными ИИ-возможностями, подготовленные к установке, обучению и применению, — утверждает Бернет, имея в виду IDP-решения (Intelligent Document Processing, интеллектуальное ПО для обработки документов). — Они ускоряют внедрение ИИ и помогают организациям решать конкретные бизнес - задачи». Это как раз тот случай, когда пользователи получают выгоды без того, чтобы вникать в сложную науку об ИИ.

**Е. Комбинирование больших данных и ИИ невозможно без участия человека**

Надежность и прозрачность — ключевые факторы в точке пересечения больших данных и ИИ. «Чтобы научить ИИ делать правильные выводы, ему нужно опираться на основу в виде надежных данных, — сказал Франко. — Это значит, что вам нужно ввести в цикл управления ими человека, что позволит взять под контроль их качество, репрезентативность, конфиденциальность и алгоритмы (применяйте объяснимый ИИ, чтобы можно было понять внутреннюю работу алгоритмов)».

**Ж. ИИ полагается не на все данные**

«Чтобы обеспечить работоспособность ИИ и получить значимые выводы, нужно выдержать точный баланс между данными, часть из которых должны быть правильными и надежными.  ИИ — не панацея для каждой проблемы, по крайней мере, пока что, и он не может создать что-то из ничего. Лидеры бизнеса должны знать об этом».