ГОСУДАРСТВЕННОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Электропривод и АПУ»

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ МЕХАТРОННЫХ И РОБОТОТЕХНИЧЕСКИХ СИСТЕМ

Методические рекомендации к лабораторным работам для студентов направления подготовки 15.03.06 «Мехатроника и робототехника» дневной формы обучения

Могилев 2018

УДК 004.4:621.865.8
ББК 32.973.202-018.2:32.816
П 78

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Электропривод и автоматизация промышленных установок» «6» февраля 2018 г., протокол № 7

Составитель ст. преподаватель В. Т. Вишнеревский

Рецензент канд. техн. наук, доц. И. В. Лесковец

Методические рекомендации предназначены к лабораторным работам по дисциплине «Программное обеспечение мехатронных и робототехнических систем» для студентов направления подготовки 15.03.06 «Мехатроника и робототехника» дневной формы обучения.

Учебно-методическое издание

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ МЕХАТРОННЫХ И РОБОТОТЕХНИЧЕСКИХ СИСТЕМ

Ответственный за выпуск	Г. С. Леневский
Технический редактор	А. А. Подошевко
Компьютерная верстка	Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. .Уч.-изд. л. .Тираж 46 экз. Заказ №

Издатель и полиграфическое исполнение:
Государственное учреждение высшего профессионального образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 24.01.2014.
Пр. Мира, 43, 212000, Могилев.

Содержание

1 Лабораторная работа № 1. Программирование одноплатных компьютеров. Основы языка программирования Python

1.1 Ход работы

Цель работы: получить начальные навыки программирования на языке Python, ознакомиться с порядком работы одноплатного компьютера Raspberry PI.

В ходе выполнения работы необходимо изучить:

– функциональную схему одноплатного компьютера Raspberry PI;

– порядок работы с одноплатным компьютером Raspberry PI;

– основы языка программирования Python.

После выполнения работы необходимо ответить на контрольные вопросы.

1.2 Основные сведения

Raspberry Pi – одноплатный компьютер, т. е. различные части компьютера, которые обычно располагаются на отдельных платах, здесь представлены на одной. К тому же эта плата имеет относительно небольшой размер – примерно 8,5 × 5,5 см.

Raspberry Pi – недорогая платформа – ее рекомендованная цена составляет всего 35 долл., а для новейшей версии A+ – 20 долл.

Продажа Raspberry Pi началась сравнительно недавно – в начале 2012 г.; сегодня это наиболее популярная платформа своей области, продано уже более 3,5 млн экземпляров Raspberry Pi.

Raspberry Pi часто используется как «мозг» робота**,** домашний сервер или просто компьютер**.**

Изначально проект создавался как образовательный, Raspberry Pi отлично подходит для изучения основ электроники. На основе Raspberry Pi создано множество компьютерных классов. Однако сегодня его область применения Raspberry Pi вышла за пределы образовательной.

Raspberry Pi выпускается в нескольких комплектациях: «A», «A+», «B», «B+», «2B», «Zero», «Zero W», «3B» и «3B+». Модели «Zero W», «3B» и «3B+» поддерживают Wi-Fi и Bluetooth. Первые три версии оснащены ARM11 процессором Broadcom BCM2835 с тактовой частотой 700 МГц и модулем оперативной памяти на 256МБ/512МБ, размещенными по технологии «package-on-package» непосредственно на процессоре. Модель «2B» оснащается процессором с 4 ядрами Cortex-A7 с частотой 1 ГГц и оперативной памятью размером 1 ГБ. Модель «A» оснащается одним USB 2.0 портом, модель «B» – двумя, а модели «B+» и «2B» – четырьмя. Также в моделях «B», «B+», «2B» и «3B» присутствует порт Ethernet. Помимо основного ядра, BCM2835 включает в себя графическое ядро с поддержкой OpenGL ES 2.0, аппаратного ускорения и FullHD-видео и DSP-ядро. Одной из особенностей является отсутствие часов реального времени.

Вывод видеосигнала возможен через композитный разъём RCA или через цифровой HDMI-интерфейс. В версии «B+», «2B» и «3B» вывод возможен через

аудиоразъем 3,5. Корневая файловая система, образ ядра и пользовательские файлы размещаются на картах памяти SD, MMC (в моделях A и B), в новых моделях начиная с «B+» используется microSD, в «3B» и «3B+» появилась возможность загружаться с USB-носителя или по сети, также можно использовать SDIO.

Одной из самых интересных особенностей Raspberry Pi является наличие портов GPIO (general purpose input/output). Благодаря этому «малиновый» компьютер можно использовать для управления различными устройствами. В модели «B» платы присутствует 26-пиновый, а в модели «B+» и «2 B» – 40-пиновый разъем GPIO.

Что нужно для начала работы с Raspberry Pi?

Чтобы начать работу с Raspberry Pi, помимо самой платы, понадобится:

– SD-карта, с которой загружается операционная система; производитель рекомендует использовать карту от 8 до 32 Gb, реально работает и на меньших картах;

– монитор или телевизор с разъемами HDMI, DVI или RCA (только для моделей A и B) и, соответственно, кабель HDMI-HDMI, HDMI-DVI или RCA-RCA, также можно использовать HDMI-VGA-преобразователь;

– USB-клавиатура;

– USB-мышь;

– кабель питания или аккумулятор micro-USB.

Raspberry Pi поставляется без ОС, ее нужно скачать с сайта производителя и загрузить на SD-карту.

1.3 Порядок выполнения работы

Для выполнения задания понадобится следующее:

– плата Raspberry Pi;

– макетная плата типа «bread board»;

– светодиод;

– кнопка;

– резистор на 220 Ом

– три провода «мама-папа»

– два провода «папа-папа».

Сначала нужно собрать схему, выполнив все необходимые подключения. Затем следует запустить компьютер Raspberry Pi, создать в рабочей директории файл с расширением .py, поместить в него текст программы и запустить файл с помощью среды разработки программ IDLE.

Элементы, необходимые для сборки схемы, изображены на рисунке 1.1.

Программирование Raspberry Pi GPIO на языке Python.

Python – современный объектно-ориентированный язык. Он наиболее часто используется для программирования GPIO на Raspberry Pi. Python входит в состав операционной системы Raspbian.

Для выполнения лабораторной работы требуется собрать схему, представленную на рисунке 1.2.

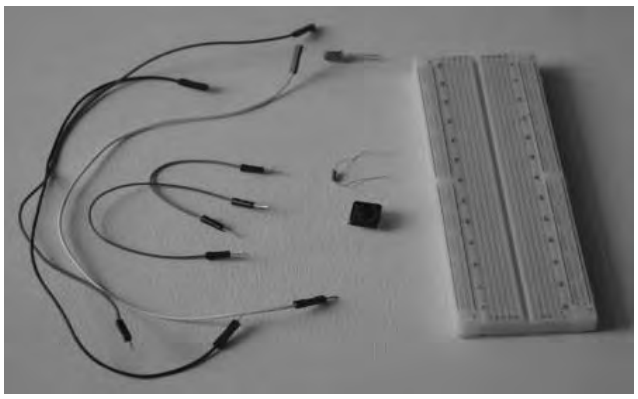

Рисунок 1.1 – Элементы, необходимые для сборки схемы к лабораторной работе

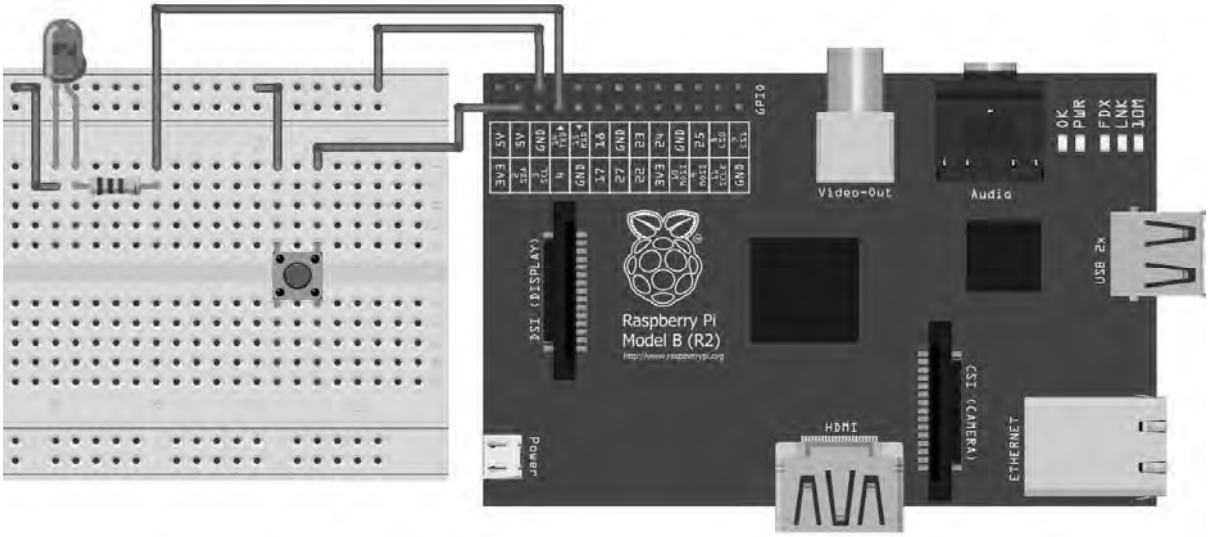

Рисунок 1.2 – Схема к выполнению лабораторной работы

Следует обратить внимание на то, что порты GPIO на Raspberry Pi не подписаны, поэтому необходимо подготовить распечатанную таблицу с назначением выводов.

Таблица с назначением выводов представлена на рисунке 1.3.

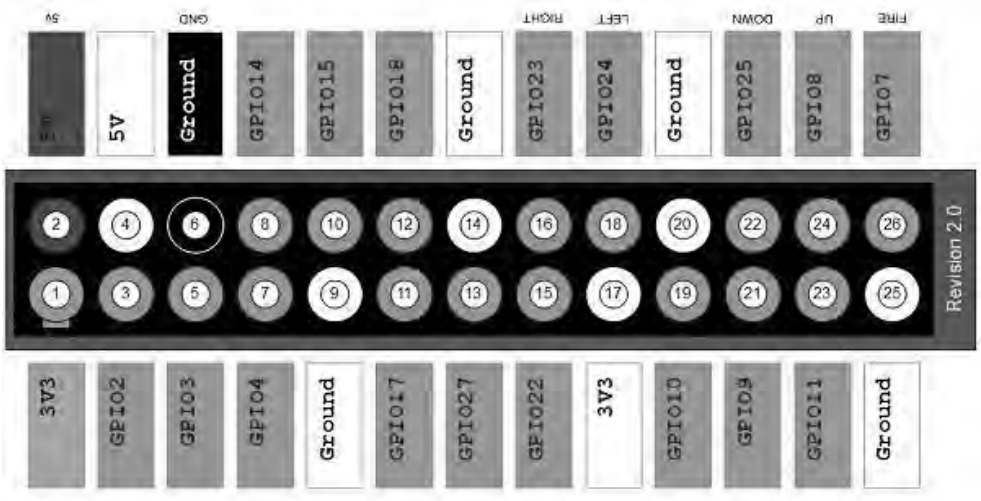

Рисунок 1.3 – Назначение выводов одноплатного компьютера Raspberry PI

Изображение собранной схемы приведено на рисунке 1.4.

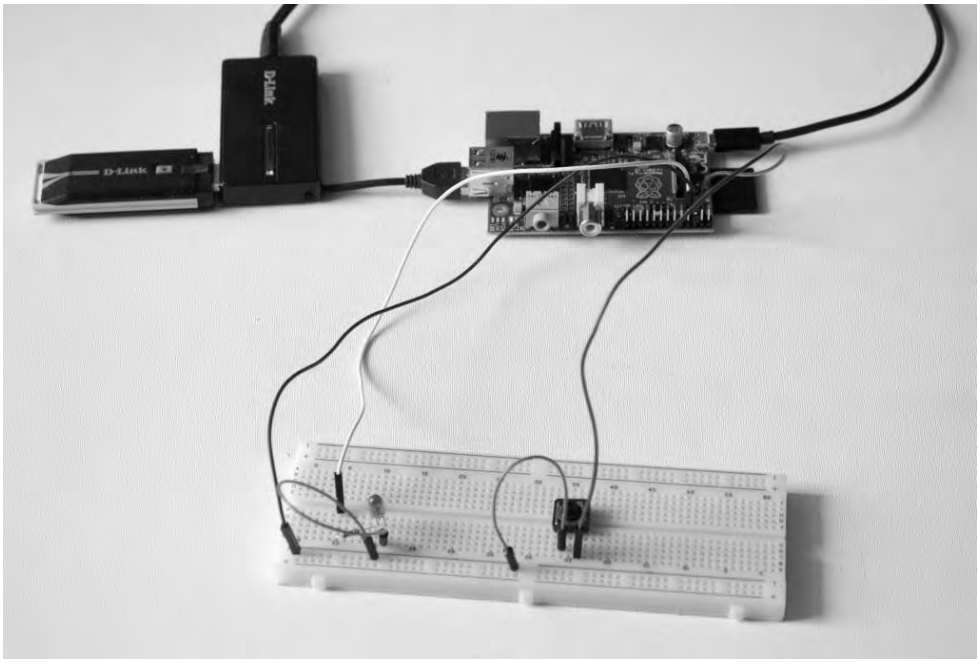

Рисунок 1.4 – Собранная схема к лабораторной работе

Следует зайти в LXTerminal и ввести с клавиатуры следующее: sudo python.

После этого вместо имени пользователя в начале строки должно отобразиться «>>>».

Далее нужно ввести следующие строки:

```
import RPi.GPIO as GPIO  #импорт библиотеки
GPIO.setmode(GPIO.BOARD) #"включение" GPIO
GPIO.setup(7, GPIO.OUT)  #объявление 7-го пина как выход
```

Затем для включения светодиода можно использовать команду

```
GPIO.output(7, 1)
```

А для выключения

```
GPIO(output(7, 0)
```

После работы с GPIO желательно выполнить команду

```
GPIO.cleanup()
```

Для автономной работы светодиода потребуется написать и запустить программу. Для нужно открыть предустановленную программу **IDLE 3** и в меню File нажать New. В открывшемся окне можно писать программу.

```
import RPi.GPIO as GPIO        #импорт библиотеки для работы с GPIO
import time                    #импорт библиотеки для ожидания
GPIO.setmode(GPIO.BOARD)       #"запуск" GPIO
GPIO.setup(7, GPIO.OUT)        #объявление порта 7 как выход
while True:                    #бесконечный цикл
____GPIO.output(7, 1)          #включение светодиода
____time.sleep(1)              #ожидание 1 секунды
____GPIO.output(7, 0)          #выключение светодиода
____time.sleep(1)              #ожидание 1 секунды
```

Сохранить программу следует в каталоге */home/pi*.

Далее можно запустить программу из LXTerminal с помощью команды sudo python programname.py

Светодиодом управляют с помощью внешней кнопки: когда кнопка зажата, светодиод горит, когда отжата – не горит.

Для управления кнопку следует подключить к порту 5.

Для этого потребуется следующая программа:

```
import RPi.GPIO as GPIO            #импорт библиотеки GPIO
GPIO.setmode(GPIO.BOARD)       #"включение GPIO"
GPIO.setup(7, GPIO.OUT)       #объявление порта 7 как выход
GPIO.setup(3, GPIO.IN)       #объявление порта 3 как вход
while True:                          #бесконечный цикл
   if GPIO.input(3) == False: #если кнопка зажата
      GPIO.output(7, 1)       #включаем светодиод
   else:         #иначе
      GPIO.output(7, 0)       #выключаем
```

Для управления светодиодом с клавиатуры можно использовать следующую программу.

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(7, GPIO.OUT)
while True:
   str = input("Enter - включение, другое - выход ");
   if str != "":
   break
   else:
      GPIO.output(7, 1)
      str = input("Enter - выключение, другое - выход ");
   if str != "":
      break
   else:
      GPIO.output(7, 0)
GPIO.cleanup()
```

Данная программа будет изменять состояние светодиода при получении пустой строки и заканчиваться при получении другой строки.

Варианты индивидуальных заданий к лабораторной работе

1 Реализовать мигание светодиодом.

2 Реализовать включение и выключение светодиода после нажатия на клавишу n-го количества раз.

3 Реализовать выключение светодиода с выдержкой времени.

4 Реализовать управление яркостью светодиода.

5 Реализовать включение светодиода после ввода с клавиатуры определенной строки.

Содержание отчета

Отчет по лабораторной работе должен содержать следующее:
– цель работы;
– постановку задачи;
– текст пользовательской программы;
– результат выполнения программы.
Оформление отчета должно соответствовать требованиям ГОСТ 2.105–2003.

Контрольные вопросы

1 Каково назначение одноплатного компьютера?
2 Каковы особенности языка программирования Python?
3 Что такое GPIO?
4 Для чего нужны GPIO?
5 Как осуществляется управление состоянием GPIO компьютера Raspberry PI?
6 Какая библиотека используется для управления состоянием GPIO компьютера Raspberry PI?

2 Лабораторная работа № 2. Линейная алгебра в программном обеспечении мехатронных и робототехнических систем

2.1 Ход работы

Цель работы: изучить особенности использования возможностей линейной алгебры в программном обеспечении роботов.
В ходе выполнения работы необходимо изучить:
– основные функции библиотеки NumPy;
– основные области программирования, в которых требуется использование линейной алгебры.
После выполнения работы необходимо ответить на контрольные вопросы.

2.2 Основные сведения

NumPy – это open-source модуль для python, который предоставляет общие математические и числовые операции в виде прескомпилированных быстрых функций. Они объединяются в высокоуровневые пакеты и обеспечивают функционал, который можно сравнить с функционалом MatLab. NumPy (Numeric Python) предоставляет базовые методы для манипуляции с большими массивами и матрицами. SciPy (Scientific Python) расширяет функционал NumPy огромной коллекцией полезных алгоритмов, таких как минимизация, преобразование Фурье, регрессия и другие прикладные математические техники.

После установки Python(x, y) – дистрибутива свободного научного и инженерного программного обеспечения для численных расчётов, анализа и визуализации данных на основе языка программирования Python и большого числа модулей (библиотек)) на платформе Windows – можно сразу приступать к разработке программ. Если Python установлен другим способом, необходимо добавить пакеты NumPy и SciPy самостоятельно.

Сообщество NumPy и SciPy поддерживает онлайн-руководство, включающее гайды и туториалы.

Есть несколько путей импорта модуля NumPy. Стандартный метод – это использовать простое выражение

```
>>> import numpy
```

Для сокращения записи при вызове функций из состава NumPy можно использовать выражение

```
>>> import numpy as np
```

Вышеприведенное выражение позволяет получать доступ к NumPy-объектам, используя np.X вместо numpy.X. Также можно импортировать NumPy прямо в текущее пространство имен, что даст возможность вызывать функции непосредственно:

```
>>> from numpy import *
```

Однако этот вариант не приветствуется в программировании на Python, т. к. убирает некоторые полезные структуры, которые модуль предоставляет. В дальнейшем будет использоваться второй вариант импорта (import numpy as np).

Главной особенностью NumPy является объект array. Массивы схожи со списками в Python, исключая тот факт, что элементы массива должны иметь одинаковый тип данных, как float и int. С массивами можно проводить числовые операции с большим объемом информации в разы быстрее и, главное, намного эффективнее, чем со списками.

Создание массива из списка:

```
a = np.array([1, 4, 5, 8], float)
>>> a
array([ 1.,  4.,  5.,  8.])
>>> type(a)
<class 'numpy.ndarray'>
```

Здесь функция array принимает два аргумента: список для конвертации в массив и тип для каждого элемента. Ко всем элементам можно получить доступ и манипулировать ими так же, как с обычными списками:

```
>>> a[:2]
array([ 1.,  4.])
>>> a[3]
```

```
8.0
>>> a[0] = 5.
>>> a
array([ 5.,  4.,  5.,  8.])
```

Массивы могут быть и многомерными. В отличие от списков можно задавать команды в скобках. Вот пример двумерного массива (матрица):

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a
array([[ 1.,  2.,  3.],
[ 4.,  5.,  6.]])
>>> a[0,0]
1.0
>>> a[0,1]
2.0
```

Array slicing работает с многомерными массивами, как и с одномерными, применяя каждый срез как фильтр для установленного измерения. Следует использовать":" в измерении для указания на все элементы этого измерения:

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a[1,:]
array([ 4.,  5.,  6.])
>>> a[:,2]
array([ 3.,  6.])
>>> a[-1:, -2:]
array([[ 5.,  6.]])
```

Метод shape возвращает количество строк и столбцов в матрице:

```
>>> a.shape
(2, 3)
```

Метод dtype возвращает тип переменных, хранящихся в массиве:

```
>>> a.dtype
dtype('float64')
```

Здесь float64 – это числовой тип данных в NumPy, который используется для хранения вещественных чисел двойной точности (так же как float в Python).

Метод len возвращает длину первого измерения (оси):

```
a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> len(a)
2
```

Метод in используется для проверки на наличие элемента в массиве:

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> 2 in a
True
>>> 0 in a
False
```

Массивы можно переформировать при помощи метода, который задает новый многомерный массив. Например, одномерный массив из десяти элементов переформатируется в двумерный массив, состоящий из пяти строк и двух столбцов, следующим образом:

```
>>> a = np.array(range(10), float)
>>> a
array([ 0.,  1.,  2., 3.,  4.,  5.,  6.,  7.,  8.,  9.])
>>> a = a.reshape((5, 2))
>>> a
array([[ 0.,  1.],
[ 2.,  3.],
[ 4.,  5.],
[ 6.,  7.],
[ 8.,  9.]])
>>> a.shape
(5, 2)
```

Метод reshape создает новый массив, а не модифицирует оригинальный.

Связывание имен в Python работает и с массивами. Метод copy используется для создания копии существующего массива в памяти:

```
>>> a = np.array([1, 2, 3], float)
>>> b = a
>>> c =  a.copy()
>>> a[0] = 0
>>> a
array([0., 2., 3.])
>>> b
array([0., 2., 3.])
>>> c
array([1., 2., 3.])
```

Списки тоже можно создавать с массивов:

```
>>> a = np.array([1, 2, 3], float)
>>> a.tolist()
[1.0, 2.0, 3.0]
>>> list(a)
[1.0, 2.0, 3.0]
```

Также можно переконвертировать массив в бинарную строку (т. е. не human-readable-форму). Для этого используется метод tostring. Метод fromstring работает для обратного преобразования. Эти операции часто применяются

для сохранения большого количества данных в файлах, которые могут быть считаны в будущем.

```
>>> a = array([1, 2, 3], float)
>>> s = a.tostring()
>>> s
'\x00\x00\x00\x00\x00\x00\xf0?\x00\x00\x00\x00\x00\x00\x00@\x00\x00\x00\x00\x00\x00\x08@'
>>> np.fromstring(s)
array([ 1.,  2.,  3.])
```

Заполнение массива одинаковым значением:

```
>>> a = array([1, 2, 3], float)
>>> a
array([ 1.,  2.,  3.])
>>> a.fill(0)
>>> a
array([ 0.,  0.,  0.])
```

Транспонирование массивов также возможно, при этом создается новый массив:

```
>>> a = np.array(range(6), float).reshape((2, 3))
>>> a
array([[ 0.,  1.,  2.],
[ 3.,  4.,  5.]])
>>> a.transpose()
array([[ 0.,  3.],
[ 1.,  4.],
[ 2.,  5.]])
```

Многомерный массив можно переконвертировать в одномерный при помощи метода flatten:

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a
array([[ 1.,  2.,  3.],
[ 4.,  5.,  6.]])
>>> a.flatten()
array([ 1.,  2.,  3.,  4.,  5.,  6.])
```

Два или больше массивов можно объединить при помощи метода concatenate:

```
>>> a = np.array([1,2], float)
>>> b = np.array([3,4,5,6], float)
>>> c = np.array([7,8,9], float)
>>> np.concatenate((a, b, c))
array([1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

Если массив не одномерный, можно задать ось, по которой будет происходить соединение. По умолчанию (значения оси не задается) соединение будет происходить по первому измерению:

```
>>> a = np.array([[1, 2], [3, 4]], float)
>>> b = np.array([[5, 6], [7,8]], float)
>>> np.concatenate((a,b))
array([[ 1.,  2.],
[ 3.,  4.],
[ 5.,  6.],
[ 7.,  8.]])
>>> np.concatenate((a,b), axis=0)
array([[ 1.,  2.],
[ 3.,  4.],
[ 5.,  6.],
[ 7.,  8.]])
>>>
np.concatenate((a,b), axis=1)
array([[ 1.,  2.,  5.,  6.],
[ 3.,  4.,  7.,  8.]])
```

Размерность массива может быть увеличена при использовании константы newaxis в квадратных скобках:

```
>>> a = np.array([1, 2, 3], float)
>>> a
array([1., 2., 3.])
>>> a[:,np.newaxis]
array([[ 1.],
[ 2.],
[ 3.]])
>>> a[:,np.newaxis].shape
(3,1)
>>> b[np.newaxis,:]
array([[ 1.,  2.,  3.]])
>>> b[np.newaxis,:].shape
(1,3)
```


В этом примере каждый массив двумерный, а созданный при помощи newaxis массив имеет размерность равную единице. Метод newaxis подходит для удобного создания массивов нужной размерности в векторной и матричной математике.

Функция arange аналогична функции range, но возвращает массив:

```
>>> np.arange(5, dtype=float)
array([ 0.,  1.,  2.,  3.,  4.])
>>> np.arange(1, 6, 2, dtype=int)
array([1, 3, 5])
```

Функции zeros и ones создают новые массивы с установленной размерностью, заполненные этими значениями. Это, наверное, самые простые в использовании функции для создания массивов:

```
>>> np.ones((2,3), dtype=float)
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

```
>>> np.zeros(7, dtype=int)
array([0, 0, 0, 0, 0, 0, 0])
```

Функции zeros_like и ones_like могут преобразовать уже созданный массив, заполнив его нулями и единицами соответственно:

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> np.zeros_like(a)
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>> np.ones_like(a)
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

Также есть некоторое количество функций для создания специальных матриц. Для создания квадратной матрицы с главной диагональю, которая заполнена единицами, следует воспользоваться методом identity:

```
>>> np.identity(4, dtype=float)
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
```

Функция eye возвращает матрицу с единицами на к-ой диагонали:

```
>>> np.eye(4, k=1, dtype=float)
array([[ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.],
       [ 0.,  0.,  0.,  0.]])
```

Если для массивов используются стандартные математические операции, то должен соблюдаться принцип «элемент-элемент». Это означает, что массивы должны быть одинакового размера во время сложения, вычитания и тому подобных операций:

```
>>> a = np.array([1,2,3], float)
>>> b = np.array([5,2,6], float)
>>> a + b
array([6., 4., 9.])
>>> a – b
array([-4., 0., -3.])
>>> a * b
array([5., 4., 18.])
>>> b / a
array([5., 1., 2.])
>>> a % b
array([1., 0., 3.])
>>> b**a
array([5., 4., 216.])
```

Для двумерных массивов умножение остается поэлементным и не соответствует умножению матриц. Для этого существуют специальные функции:

```
>>> a = np.array([[1,2], [3,4]], float)
>>> b = np.array([[2,0], [1,3]], float)
>>> a * b
array([[2., 0.], [3., 12.]])
```

При несоответствии в размере выбрасываются ошибки:

```
>>> a = np.array([1,2,3], float)
>>> b = np.array([4,5], float)
>>> a + b
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: operands could not be broadcast together with shapes (3,) (2,)
```

Однако, если размерность массивов не совпадает, они будут преобразованы для выполнения математических операций. Это зачастую означает, что меньший массив будет использован несколько раз для завершения операций. Рассмотрим такой пример:

```
>>> a = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> b = np.array([-1, 3], float)
>>> a
array([[ 1.,  2.],
       [ 3.,  4.],
       [ 5.,  6.]])
>>> b
array([-1.,  3.])
>>> a + b
array([[ 0.,  5.],
       [ 2.,  7.],
       [ 4.,  9.]])
```

Здесь одномерный массив b был преобразован в двумерный, который соответствует размеру массива a. По существу, b был повторен несколько раз, для каждой «строки» a. Иначе его можно представить следующим образом:

```
array([[-1.,  3.],
       [-1.,  3.],
       [-1.,  3.]])
```

Python автоматически преобразовывает массивы в этом случае. Иногда, однако, когда преобразование играет роль, можно использовать константу newaxis, чтобы изменить преобразование:

```
>>> a = np.zeros((2,2), float)
>>> b = np.array([-1., 3.], float)
>>> a
```

```
array([[ 0.,  0.],
       [0.,  0.]])
>>> b
array([-1., 3.])
>>> a + b
array([[-1.,  3.],
       [-1.,  3.]])
>>> a + b[np.newaxis,:]
array([[-1.,  3.],
       [-1.,  3.]])
>>> a + b[:,np.newaxis]
array([[-1., -1.],
       [ 3.,  3.]])
```

Вместе со стандартными операторами в NumPy включена библиотека стандартных математических функций, которые могут быть применены поэлементно к массивам. Это собственно функции: abs, sign, sqrt, log, log10, exp, sin, cos, tan, arcsin, arccos, arctan, sinh, cosh, tanh, arcsinh, arccosh, и arctanh.

```
>>> a = np.array([1, 4, 9], float)
>>> np.sqrt(a)
array([ 1.,  2.,  3.])
```

Функции floor, ceil и rint возвращают нижнее, верхнее или ближайшее (округлённое) значение:

```
>>> a = np.array([1.1, 1.5, 1.9], float)
>>> np.floor(a)
array([ 1.,  1.,  1.])
>>> np.ceil(a)
array([ 2.,  2.,  2.])
>>> np.rint(a)
array([ 1.,  2.,  2.])
```

Также в NumPy включены две важные математические константы:

```
>>> np.pi
3.1415926535897931
>>> np.e
2.7182818284590451
```

Проводить итерацию массивов можно аналогично спискам:

```
>>> a = np.array([1, 4, 5], int)
>>> for x in a:
...   print x
1
4
5
```

Для многомерных массивов итерация будет проводиться по первой оси, так, что каждый проход цикла будет возвращать «строку» массива:

```
>>> a = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> for x in a:
...   print x
[ 1.  2.]
[ 3.  4.]
[ 5.  6.]
```

Множественное присваивание также доступно при итерации:

```
>>> a = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> for (x, y) in a:
...   print x * y
2.0
12.0
30.0
```

Для получения каких-либо свойств массивов существует много функций. Элементы могут быть суммированы или перемножены:

```
>>> a = np.array([2, 4, 3], float)
>>> a.sum()
9.0
>>> a.prod()
24.0
```

В этом примере применялись функции массива. Также можно использовать собственные функции NumPy:

```
>>> np.sum(a)
9.0
>>> np.prod(a)
24.0
```

Для большинства случаев могут применяться оба варианта. Некоторые функции дают возможность оперировать статистическими данными. Это функции mean (среднее арифметическое), вариация и девиация:

```
>>> a = np.array([2, 1, 9], float)
>>> a.mean()
4.0
>>>
a.var()
12.666666666666666
>>> a.std()
3.5590260840104371
```

Можно найти минимум и максимум в массиве:

```
>>> a = np.array([2, 1, 9], float)
>>> a.min()
1.0
>>> a.max()
9.0
```

Функции argmin и argmax возвращают индекс минимального или максимального элемента:

```
>>> a = np.array([2, 1, 9], float)
>>> a.argmin()
1
>>> a.argmax()
2
```

Для многомерных массивов каждая из функций может принять дополнительный аргумент axis и в зависимости от его значения выполнять функции по определенной оси, помещая результаты исполнения в массив:

```
>>> a = np.array([[0, 2], [3, -1], [3, 5]], float)
>>> a.mean(axis=0)
array([ 2.,  2.])
>>> a.mean(axis=1)
array([ 1.,  1.,  4.])
>>> a.min(axis=1)
array([ 0., -1.,  3.])
>>> a.max(axis=0)
array([ 3.,  5.])
```

Как и списки, массивы можно отсортировать:

```
>>> a = np.array([6, 2, 5, -1, 0], float)
>>> sorted(a)
[-1.0, 0.0, 2.0, 5.0, 6.0]
>>> a.sort()
>>> a
array([-1.,  0.,  2.,  5.,  6.])
```

Значения в массиве могут быть сокращены, чтобы принадлежать заданному диапазону. Это то же самое, что применять min(max(x, minval), maxval) к каждому элементу x:

```
>>> a = np.array([6, 2, 5, -1, 0], float)
>>> a.clip(0, 5)
array([ 5.,  2.,  5.,  0.,  0.])
```

Уникальные элементы могут быть извлечены следующим образом:

```
>>> a = np.array([1, 1, 4, 5, 5, 5, 7], float)
>>> np.unique(a)
array([ 1.,  4.,  5.,  7.])
```

Для двухмерных массивов диагональ можно получить как

```
>>> a = np.array([[1, 2], [3, 4]], float)
>>> a.diagonal()
array([ 1.,  4.])
```

Булево сравнение может быть использовано для поэлементного сравнения массивов одинаковых длин. Возвращаемое значение – это массив булевых True/False-значений:

```
>>> a = np.array([1, 3, 0], float)
>>> b = np.array([0, 3, 2], float)
>>> a > b
array([ True, False, False], dtype=bool)
>>> a == b
array([False,  True, False], dtype=bool)
>>> a <= b
array([False,  True,  True], dtype=bool)
```

Результат сравнения может быть сохранен в массиве:

```
>>> c = a > b
>>> c
array([ True, False, False], dtype=bool)
```

Массивы могут быть сравнены с одиночным значением:

```
>>> a = np.array([1, 3, 0], float)
>>> a > 2
array([False,  True, False], dtype=bool)
```

Операторы any и all могут быть использованы для определения истинности ли хотя бы одного или всех элементов соответственно:

```
>>> c = np.array([ True, False, False], bool)
>>> any(c)
True
>>> all(c)
False
```

Комбинированные булевы выражения могут быть применены к массивам по принципу элемент – элемент с использованием специальных функций logical_and, logical_or и logical_not:

```
>>> a = np.array([1, 3, 0], float)
>>> np.logical_and(a > 0, a < 3)
array([ True, False, False], dtype=bool)
>>> b = np.array([True, False, True], bool)
>>> np.logical_not(b)
array([False,  True, False], dtype=bool)
>>> c = np.array([False, True, False], bool)
>>> np.logical_or(b, c)
array([ True,  True,  False], dtype=bool)
```

Функция where создает новый массив из двух других массивов одинаковых длин с помощью булева фильтра для выбора между двумя элементами. Базовый синтаксис: where(boolarray, truearray, falsearray):

```
>>> a = np.array([1, 3, 0], float)
>>> np.where(a != 0, 1 / a, a)
array([ 1.        ,  0.33333333,  0.        ])
```

С функцией where также может быть реализовано «массовое сравнение»:

```
>>> np.where(a > 0, 3, 2)
array([3, 3, 2])
```

Некоторые функции дают возможность тестировать значения в массиве. Функция nonzero возвращает кортеж индексов ненулевых значений. Количество элементов в кортеже равно количеству осей в массиве:

```
>>> a = np.array([[0, 1], [3, 0]], float)
>>> a.nonzero()
(array([0, 1]), array([1, 0]))
```

Можно проверить значения на конечность и на NaN(not a number):

```
>>> a = np.array([1, np.NaN, np.Inf], float)
>>> a
array([  1.,  NaN,  Inf])
>>> np.isnan(a)
array([False,  True, False], dtype=bool)
>>> np.isfinite(a)
array([ True, False, False], dtype=bool)
```

Здесь используются константы NumPy, чтобы добавить значения NaN и бесконечность, которые могут быть результатами применения стандартных математических операций.

В отличие от списков, массивы позволяют делать выбор элементов при помощи других массивов. Это значит, что можно использовать массив для фильтрации специфических подмножеств элементов других массивов.

Булевы массивы могут быть применены как массивы для фильтрации:

```
>>> a = np.array([[6, 4], [5, 9]], float)
>>> a >= 6
array([[ True, False],
       [False,  True]], dtype=bool)
>>> a[a >= 6]
array([ 6.,  9.])
```

Следует отметить, что, когда передается булев массив a>=6 как индекс для операции доступа по индексу массива a, возвращаемый массив будет хранить только True значения. Также можно записать массив для фильтрации в переменную:

```
>>> a = np.array([[6, 4], [5, 9]], float)
>>> sel = (a >= 6)
>>> a[sel]
array([ 6.,  9.])
```

Более замысловатая фильтрация может быть достигнута использованием булевых выражений:

```
>>> a[np.logical_and(a > 5, a < 9)]
>>> array([ 6.])
```

Помимо булева выбора, можно использовать целочисленные массивы. В этом случае целочисленный массив хранит индексы элементов, которые будут взяты из массива. Рассмотрим следующий одномерный пример:

```
>>> a = np.array([2, 4, 6, 8], float)
>>> b = np.array([0, 0, 1, 3, 2, 1], int)
>>> a[b]
array([ 2.,  2.,  4.,  8.,  6.,  4.])
```

Иными словами, когда b используется для получения элементов из a, берутся 0-й, 0-й, 1-й, 3-й, 2-й и 1-й элементы a в соответствии со значениями в массиве b. Списки также могут быть применены как массивы для фильтрации:

```
>>> a = np.array([2, 4, 6, 8], float)
>>> a[[0, 0, 1, 3, 2, 1]]
array([ 2.,  2.,  4.,  8.,  6.,  4.])
```

Для многомерных массивов необходимо передать несколько одномерных целочисленных массивов в оператор доступа к индексу для каждой оси. При этом выборка элементов массива происходит в следующем порядке: первый элемент соответствует индексу строки, который является первым элементом массива b, второй элемент соответствует индексу столбца, который является первым элементом массива c и так далее.

Пример

```
>>> a = np.array([[1, 4], [9, 16]], float)
>>> b = np.array([0, 0, 1, 1, 0], int)
```

```
>>> c = np.array([0, 1, 1, 1, 1], int)
>>> a[b,c]
array([  1.,   4.,  16.,  16.,   4.])
```

Специальная функция take доступна для выполнения выборки с целочисленными массивами. Работа функции аналогична использованию оператора доступа по индексу:

```
>>> a = np.array([2, 4, 6, 8], float)
>>> b = np.array([0, 0, 1, 3, 2, 1], int)
>>> a.take(b)
array([ 2.,  2.,  4.,  8.,  6.,  4.])
```

Функция take предоставляет аргумент axis (ось) для взятия подсекции многомерного массива вдоль какой-либо оси:

```
>>> a = np.array([[0, 1], [2, 3]], float)
>>> b = np.array([0, 0, 1], int)
>>> a.take(b, axis=0)
array([[ 0.,  1.],
       [ 0.,  1.],
       [ 2.,  3.]])
>>> a.take(b, axis=1)
array([[ 0.,  0.,  1.],
       [ 2.,  2.,  3.]])
```

Противоположной функции take является функция put, которая будет брать значения из исходного массива и записывать их на места элементов по специфическим индексам в другом put-массиве.

```
>>> a = np.array([0, 1, 2, 3, 4, 5], float)
>>> b = np.array([9, 8, 7], float)
>>> a.put([0, 3], b)
>>> a
array([ 9.,  1.,  2.,  8.,  4.,  5.])
```

Следует отметить, что значение 7 из исходного массива b не было использовано, т. к. только два индекса [0, 3] указаны. Исходный массив будет повторен, если это необходимо в случае несоответствия длин:

```
>>> a = np.array([0, 1, 2, 3, 4, 5], float)
>>> a.put([0, 3], 5)
>>> a
array([ 5.,  1.,  2.,  5.,  4.,  5.])
```

NumPy обеспечивает много функций для работы с векторами и матрицами. Функция dot возвращает скалярное произведение векторов:

```
>>> a = np.array([1, 2, 3], float)
>>> b = np.array([0, 1, 1], float)
```

```
>>> np.dot(a, b)
5.0
```

Функция dot также может умножать матрицы:

```
>>> a = np.array([[0, 1], [2, 3]], float)
>>> b = np.array([2, 3], float)
>>> c = np.array([[1, 1], [4, 0]], float)
>>> a
array([[ 0.,  1.],
       [ 2.,  3.]])
>>> np.dot(b, a)
array([  6.,  11.])
>>> np.dot(a, b)
array([  3.,  13.])
>>> np.dot(a, c)
array([[  4.,   0.],
       [ 14.,   2.]])
>>> np.dot(c, a)
array([[ 2.,  4.],
       [ 0.,  4.]])
```

Можно получить скалярное, тензорное и внешнее произведение матриц и векторов. Следует отметить, что для векторов внутреннее и скалярное произведение совпадает.

```
>>> a = np.array([1, 4, 0], float)
>>> b = np.array([2, 2, 1], float)
>>> np.outer(a, b)
array([[ 2.,  2.,  1.],
       [ 8.,  8.,  4.],
       [ 0.,  0.,  0.]])
>>> np.inner(a, b)
10.0
>>> np.cross(a, b)
array([ 4., -1., -6.])
```

NumPy предоставляет набор встроенных функций и методов для работы с линейной алгеброй. Это всё можно найти в подмодуле linalg. Этими модулями также можно оперировать с вырожденными и невырожденными матрицами. Определитель матрицы находят следующим образом:

```
>>> a = np.array([[4, 2, 0], [9, 3, 7], [1, 2, 1]], float)
>>> a
array([[ 4.,  2.,  0.],
       [ 9.,  3.,  7.],
       [ 1.,  2.,  1.]])
>>> np.linalg.det(a)
-53.999999999999993
```

Можно найти собственный вектор и собственное значение матрицы:

```
>>> vals, vecs = np.linalg.eig(a)
>>> vals
array([ 9.        ,  2.44948974, -2.44948974])
>>> vecs
array([[-0.3538921 , -0.56786837,  0.27843404],
       [-0.88473024,  0.44024287, -0.89787873],
       [-0.30333608,  0.69549388,  0.34101066]])
```

Невырожденная матрица может быть найдена как

```
>>> b = np.linalg.inv(a)
>>> b
array([[ 0.14814815,  0.07407407, -0.25925926],
       [ 0.2037037 , -0.14814815,  0.51851852],
       [-0.27777778,  0.11111111,  0.11111111]])
>>> np.dot(a, b)
array([[  1.00000000e+00,   5.55111512e-17,   2.22044605e-16],
       [  0.00000000e+00,   1.00000000e+00,   5.55111512e-16],
       [  1.11022302e-16,   0.00000000e+00,   1.00000000e+00]])
```

Одиночное разложение (аналог диагонализации неквадратной матрицы) может быть достигнуто следующим образом:

```
>>> a = np.array([[1, 3,4], [5, 2, 3]], float)
>>> U, s, Vh = np.linalg.svd(a)
>>> U
array([[-0.6113829 , -0.79133492],
       [-0.79133492,  0.6113829 ]])
>>> s
array([ 7.46791327,  2.86884495])
>>> Vh
array([[-0.61169129, -0.45753324, -0.64536587],
       [ 0.78971838, -0.40129005, -0.46401635],
       [-0.046676 , -0.79349205,  0.60678804]])
```

NumPy предоставляет методы для работы с полиномами. Передавая список корней, можно получить коэффициенты уравнения:

```
>>> np.poly([-1, 1, 1, 10])
array([ 1, -11,   9,  11, -10])
```

Может быть произведена и обратная операция: передавая список коэффициентов, функция root вернет все корни многочлена:

```
>>> np.roots([1, 4, -2, 3])
array([-4.57974010+0.j        ,  0.28987005+0.75566815j,
       0.28987005-0.75566815j])
```

Следует отметить, что в этом уравнении два корня мнимые. Коэффициенты многочлена могут быть интегрированы. Рассмотрим работу функции интегрирования. Обычно константа C равна нулю:

```
>>> np.polyint([1, 1, 1, 1])
array([ 0.25      ,  0.33333333,  0.5       ,  1.        ,  0.        ])
```

Аналогично могут быть взяты производные:

```
>>> np.polyder([1./4., 1./3., 1./2., 1., 0.])
array([ 1.,  1.,  1.,  1.])
```

Функции polyadd, polysub, polymul и polydiv также поддерживают суммирование, вычитание, умножение и деление коэффициентов многочлена соответственно.

Функция polyval подставляет в многочлен заданное значение. Рассмотрим многочлен при x = 4:

```
>>> np.polyval([1, -2, 0, 2], 4)
34
```

В заключение следует отметить, что функция polyfit может быть использована для подбора (интерполяции) многочлена заданного порядка к набору значений:

```
>>> x = [1, 2, 3, 4, 5, 6, 7, 8]
>>> y = [0, 2, 1, 3, 7, 10, 11, 19]
>>> np.polyfit(x, y, 2)
array([ 0.375     , -0.88690476,  1.05357143])
```

Возвращаемый массив – это список коэффициентов многочлена. Более утонченные интерполяционные функции могут быть найдены в SciPy.

Вместе с функциями mean, var и std NumPy предоставляет еще некоторые методы для работы со статистическими данными в массивах. Медиана может быть найдена следующим образом:

```
>>> a = np.array([1, 4, 3, 8, 9, 2, 3], float)
>>> np.median(a)
3.0
```

Коэффициент корреляции для некоторых переменных наблюдается несколько раз и может быть найден из массивов вида [[x1, x2, ...], [y1, y2, ...], [z1, z2, ...], ...], где x, y, z – это разные квантовые наблюдаемые и их значения указывают количество «наблюдений»:

```
>>> a = np.array([[1, 2, 1, 3], [5, 3, 1, 8]], float)
>>> c = np.corrcoef(a)
>>> c
array([[ 1.        ,  0.72870505],
[ 0.72870505,  1.        ]])
```

Данная функция возвращает массив c[i, j], который хранит корреляционный коэффициент для i-х и j-х квантовых наблюдаемых. Ковариационный момент может быть найден следующим образом:

```
>>> np.cov(a)
array([[ 0.91666667,  2.08333333],
[ 2.08333333,  8.91666667]])
```

Важная часть каждой симуляции – это способность генерировать случайные числа. Для этого используется встроенный в NumPy генератор псевдослучайных чисел в подмодуле random. Числа являются *псевдослучайными* – имеется в виду, что они сгенерированы детерминистически из порождающего элемента (seed number), но рассредоточены в статистическом сходстве случайным образом. Для генерации NumPy использует особенный алгоритм Mersenne Twister. Задать порождающий элемент последовательности случайных чисел можно следующим образом:

```
>>> np.random.seed(293423)
```

Seed – это целое число. Каждая программа, которая запускается с одинаковым числом seed, будет генерировать одинаковую последовательность чисел каждый раз. Это может быть использовано в процессе отладки. Для решения нормальных задач не нужно задавать значение seed, тогда при каждом запуске программы будет получаться разная последовательность чисел. Если эта команда не будет выполнена, то NumPy автоматически выбирает случайное значение seed (базирующееся на времени), которое является разным при каждом запуске программы.

Массив случайных чисел из полуинтервала [0.0, 1.0) может быть сгенерирован следующим образом:

```
>>> np.random.rand(5)
array([ 0.40783762,  0.7550402 ,  0.00919317,  0.01713451,  0.95299583])
```

Для генерации двумерных массивов можно использовать функцию rand или функцию reshape:

```
>>> np.random.rand(2,3)
array([[ 0.50431753,  0.48272463,  0.45811345],
    [ 0.18209476,  0.48631022,  0.49590404]])
>>> np.random.rand(6).reshape((2,3))
array([[ 0.72915152,  0.59423848,  0.25644881],
    [ 0.75965311,  0.52151819,  0.60084796]])
```

Для генерации единичного случайного числа на интервале [0.0, 1.0)

```
>>> np.random.random()
0.70110427435769551
```

Для генерации случайного целого числа в диапазоне [min, max) используется функция randint (min, max):

```
>>> np.random.randint(5, 10)
9
```

В каждом рассмотренном примере сгенерированы числа из непрерывного равномерного распределения. NumPy также включает генераторы для других распределений, таких как Бета, биномиальное, хи-квадрат, Дирихле, экспоненциальное, Фишера, Гамма, геометрическое, Гамбала, гипергеометрическое, Лапласа, логистическое, логнормальное, логарифмическое, мультиномиальное, многомерное нормальное, отрицательное биномиальное, нецентральное хи-квадрат, нецентральное Фишера, нормальное (Гаусса), Парето, Пуассона, степенное, Рэлея, Коши, Стьюдента, треугольное, Фон-Миса, Вальда, Вейбулла и Ципфа. Рассмотрим два примера. Для генерации из дискретного распределения Пуассона при $\lambda = 6.0$

```
>>> np.random.poisson(6.0)
5
```

Для генерации числа из нормального распределения (Гаусса) при среднем значении $\mu = 1.5$ и стандартной девиации $\sigma = 4.0$

```
>>> np.random.normal(1.5, 4.0)
0.83636555041094318
```

Для получения числа из нормального распределения ($\mu = 0$, $\sigma = 1$) без указания аргументов

```
>>> np.random.normal()
0.27548716940682932
```

Для генерации нескольких значений используется аргумент size:

```
>>> np.random.normal(size=5)
array([-1.67215088,  0.65813053, -0.70150614,  0.91452499,  0.71440557])
```

Модуль для генерации случайных чисел также может быть применен для случайного распределения значений в списке:

```
>>> l = range(10)
>>> l
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> np.random.shuffle(l)
>>> l
[4, 9, 5, 0, 2, 7, 6, 8, 1, 3]
```

Следует отметить, что функция shuffle модифицирует уже существующий массив и не возвращает новый.

NumPy включает еще много других функций. В частности, это функции для работы с дискретным преобразованием Фурье, более сложными операциями в линейной алгебре, тестированием массивов на размер / размерность / тип, разделением и соединением массивов, гистограммами, созданием массивов из каких-либо данных разными путями, созданием и оперированием grid-массивов,

специальными значениями (NaN, Inf), set-операциями, созданием разных видов специальных матриц и вычислением специальных математических функций (например, функции Бесселя). Более точные детали можно найти в документации по NumPy.

Индивидуальное задание к лабораторной работе

Индивидуальное задание выдается преподавателем и заключается в том, чтобы организовать ввод матриц и выполнение какого-либо расчета или преобразование формы матрицы.

Содержание отчета

Отчет по лабораторной работе должен содержать следующее:
– цель работы;
– постановку задачи;
– текст пользовательской программы;
– результат выполнения программы.
Оформление отчета должно соответствовать требованиям ГОСТ 2.105–2003.

Контрольные вопросы

1 Для чего нужна библиотека NumPy?
2 Каково значение линейной алгебры для программного обеспечения роботов?
3 Какие операции позволяют производить функции библиотеки NumPy?
4 Как осуществляется перемножение матриц с помощью библиотеки NumPy?

3 Лабораторная работа № 3. Программная реализация систем технического зрения

3.1 Ход работы

Цель работы: ознакомиться с основами технического зрения, научиться использовать основные функции библиотеки OpenCV.
В ходе выполнения работы необходимо изучить:
– функции захвата изображения с камеры;
– преобразование изображения;
– основы поиска объектов на изображении.
После выполнения работы необходимо ответить на контрольные вопросы.

3.2 Основные сведения

OpenCV на python: получение кадров, смена цветовой модели и размытие.

OpenCV – это мощная библиотека машинного зрения, которую часто применяют в роботах для распознавания объектов окружающего мира.

Перед тем как приступить непосредственно к функциям машинного зрения, следует ознакомиться со вспомогательными функциями, необходимыми для работы с видеопотоком.

Все программы должны быть написаны на языке Python и запущены на Raspberry Pi 3.

Подготовка к написанию первой программы

Шаг 1. Если в ходе установки OpenCV использовать виртуальное окружение, то перед запуском python-скриптов необходимо перейти в это окружение с помощью команд:

```
$ source ~/.profile
$ workon cv
```

Шаг 2. Для дальнейшей работы необходимы три дополнительных модуля:

```
video.py
common.py
tst_scene_render.py
```

Эти модули можно найти в папке с примерами: /home/pi/Downloads/opencv-master/samples/python

В домашней папке следует создать папку opencv и скопировать в неё эти модули:

```
$ cd ~
$ mkdir opencv
$ cp /home/pi/Downloads/opencv-master/samples/python/video.py .
$ cp /home/pi/Downloads/opencv-master/samples/python/common.py .
$ cp /home/pi/Downloads/opencv-master/samples/python/tst_scene_render.py .
```

Шаг 3. В этой же папке создается файл с программой.

```
$ touch eye.py
```

Шаг 4. Программу можно редактировать в любом приложении. Если нет предпочтений, то рекомендуется использовать редактор Nano. В этом редакторе открывается недавно созданный файл:

```
$ nano eye.py
```

Шаг 5. После того как программа будет написана, изменения сохраняются с помощью комбинации клавиш Ctrl + o, для подтверждения нажимают Y. Для выхода из редактора нажимают Ctrl + x.

Получение кадров с камеры и вывод в окно. Программа будет в бесконечном цикле получать кадры с камеры и отображать их в специальном окне. Алгоритм программы:

```
import cv2
import video

if __name__ == '__main__':
    # создаем окно с именем result
    cv2.namedWindow( "result" )

    # создаем объект cap для захвата кадров с камеры
    cap = video.create_capture(0)

    while True:
        # захватываем текущий кадр и кладем его в переменную img
        flag, img = cap.read()
        try:
            # отображаем кадр в окне с именем result
            cv2.imshow('result', img)
        except:
            cap.release()
            raise

        ch = cv2.waitKey(5)
        if ch == 27:
            break

    cap.release()
    cv2.destroyAllWindows()
```

Примечание – Функция create_capture имеет всего один аргумент, который отвечает за индекс камеры в системе. В случае, если камера всего одна, её индекс будет равен 0.

Запускаем скрипт Python:

```
0$ python eye.py
```

Если в программе нет ошибок, откроется окно с видео.

Отражение картинки по вертикали и горизонтали.

В предыдущей программе был взят каждый кадр, полученный с камеры, и отображен в неизменном виде. Теперь в разрыв между этими двумя операциями добавляется третья – отражение кадра.

За отражение кадра отвечает функция

flip(*кадр*, *направление*)

Здесь **кадр** – кадр видеопотока, который нужно отразить; **направление** – флаг, определяющий направление отражения: 0 – по горизонтали, 1 – по вертикали, -1 – по горизонтали и вертикали одновременно.

В действительности разные веб-камеры могут по-разному реагировать на вертикальное и горизонтальное отражение.

```
import cv2
import video

if __name__ == '__main__':
    # создаем окно с именем result
    cv2.namedWindow( "result" )

    # создаем объект cap для захвата кадров с камеры
    cap = video.create_capture(0)

    while True:
        # захватываем текущий кадр и кладем его в переменную img
        flag, img = cap.read()
        try:
            # отражаем кадр
            img = cv2.flip(img,0)
            # отображаем кадр в окне с именем result
            cv2.imshow('result', img)
        except:
            cap.release()
            raise

        ch = cv2.waitKey(5)
        if ch == 27:
            break

    cap.release()
    cv2.destroyAllWindows()
```


Цветовая модель – это модель представления цвета каждой точки с помощью группы чисел, некий цветовой код. Например, в RGB цвет определяется тремя компонентами: красным, зеленым и синим. Любая веб-камера передает кадры именно в такой модели. А в HSV у каждой точки есть цветовой тон – H, насыщение – S и яркость – V.

Некоторые функции машинного зрения проще реализовать в какой-то конкретной модели. Например, распознавание цветовых пятен на картинке лучше работает в модели HSV.

Для преобразования цветовой модели используется функция

cvtColor(*кадр*, *модель*),

где *модель* – код преобразования: COLOR_GRAY – в оттенки серого; COLOR_RGB2HSV – из RGB в HSV; COLOR_BGR2HSV – из BGR в HSV;

COLOR_HSV2BGR – обратно, из HSV в BGR; и т. д.

OpenCV поддерживает много моделей, но рассматриваются три: RGB, HSV и оттенки серого.

Примечание – По умолчанию кадр с камеры приходит в модели BGR, а не RGB, поэтому для преобразования необходимо использовать флаг COLOR_BGR2HSV.

```
import cv2
import video

if __name__ == '__main__':
    # создаем окно с именем result
    cv2.namedWindow( "result" )

    # создаем объект cap для захвата кадров с камеры
    cap = video.create_capture(0)

    while True:
        # захватываем текущий кадр и кладем его в переменную img
        flag, img = cap.read()
        try:
            # меняем цветовую модель на HSV
            img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV )
            # отображаем кадр в окне с именем result
            cv2.imshow('result', img)
        except:
            cap.release()
            raise

        ch = cv2.waitKey(5)
        if ch == 27:
            break

    cap.release()
    cv2.destroyAllWindows()
```

Размытие по Гауссу.

Часто изображение с веб-камер страдает наличием большого количества шумов. Особенного это заметно, когда камере не хватает освещения. Для частичного решения этой проблемы применяют фильтр Гаусса или, другими словами размытие по Гауссу (Gaussian blur).

Функция имеет три аргумента:

GaussianBlur(*кадр*, *размер ядра*, *отклонение*)

Размер ядра – список из двух чисел (x,y), которые задают размер ядра фильтра Гаусса по горизонтали и вертикали. Чем больше ядро, тем более размытым станет кадр; **отклонение** – стандартное отклонение по оси X.

```
import cv2
import video
```

```
if __name__ == '__main__':
    # создаем окно с именем result
    cv2.namedWindow( "result" )

    # создаем объект cap для захвата кадров с камеры
    cap = video.create_capture(0)

    while True:
        # захватываем текущий кадр и кладем его в переменную img
        flag, img = cap.read()
        try:
            # размываем кадр
            img = cv2.GaussianBlur(img, (5, 5), 2)
            # отображаем кадр в окне с именем result
            cv2.imshow('result', img)
        except:
            cap.release()
            raise

        ch = cv2.waitKey(5)
        if ch == 27:
            break
```

Следует отметить, что в OpenCV есть и другие фильтры для удаления шума: медианный фильтр, двухсторонний фильтр и усреднение.

3.3 Порядок выполнения работы

Функция OpenCV для поиска прямоугольников minAreaRect.

В OpenCV имеется функция, которая пытается найти прямоугольник максимального размера, который может вписаться в заданный замкнутый контур. Следует отметить, что эта функция не определяет, является ли контур прямоугольным, она пытается вписать в него прямоугольник оптимальным способом. Это важно!

minAreaRect(*контур*)

контур – это контур, в который необходимо вписать прямоугольник (тип аргумента – Nx2 массив NumPy).

Напишем программу, которая найдет на картинке все прямоугольники.

```
#!/usr/bin/env python

import sys
import numpy as np
import cv2 as cv

hsv_min = np.array((0, 54, 5), np.uint8)
hsv_max = np.array((187, 255, 253), np.uint8)
```

```
if __name__ == '__main__':
    fn = 'image2.jpg' # имя файла, который будем анализировать
    img = cv.imread(fn)

    hsv = cv.cvtColor( img, cv.COLOR_BGR2HSV ) # меняем цветовую модель с BGR на HSV
    thresh = cv.inRange( hsv, hsv_min, hsv_max ) # применяем цветовой фильтр
    _, contours0, hierarchy = cv.findContours( thresh.copy(), cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)

    # перебираем все найденные контуры в цикле
    for cnt in contours0:
        rect = cv.minAreaRect(cnt) # пытаемся вписать прямоугольник
        box = cv.boxPoints(rect) # поиск четырех вершин прямоугольника
        box = np.int0(box) # округление координат
        cv.drawContours(img,[box],0,(255,0,0),2) # рисуем прямоугольник

    cv.imshow('contours', img) # вывод обработанного кадра в окно

    cv.waitKey()
    cv.destroyAllWindows()
```

Видно, что алгоритм попытался вписать прямоугольники во вложенные мусорные контуры на самих объектах.

Теперь попробуем то же самое, но с эллипсами.

Функция OpenCV для поиска эллипсов fitEllipse.

Как и в случае minAreaRect, функция поиска эллипсов не сможет отличить на картинке объект с действительно эллиптическим контуром от квадрата. Она лишь пытается эллипс вписать в любой контур с количеством точек >=5.

fitEllipse(*контур*)

контур – это контур, в который необходимо вписать прямоугольник (тип аргумента – Nx2 массив NumPy).

Следует немного изменить предыдущую программу, убрав из неё minAreaRect и добавив fitEllipse.

```
#!/usr/bin/env python

import sys
import numpy as np
import cv2 as cv

hsv_min = np.array((0, 77, 17), np.uint8)
hsv_max = np.array((208, 255, 255), np.uint8)

if __name__ == '__main__':
    fn = 'donuts.jpg'
    img = cv.imread(fn)
```

```
    hsv = cv.cvtColor( img, cv.COLOR_BGR2HSV )
    thresh = cv.inRange( hsv, hsv_min, hsv_max )
    _, contours0, hierarchy = cv.findContours( thresh.copy(), cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
    for cnt in contours0:
      if len(cnt)>4:
        ellipse = cv.fitEllipse(cnt)
        cv.ellipse(img,ellipse,(0,0,255),2)

    cv.imshow('contours', img)

    cv.waitKey()
    cv.destroyAllWindows()
```

Условие «if len(cnt)>4:» необходимо для того, чтобы отсечь контуры с контурами меньше 5 точек.

Следует отметить, что эллипсы лучше пытаться вписать в округлые объекты, а прямоугольники – в прямоугольные. В противном случае алгоритм может выдать неадекватные результаты.

Отсечение лишних контуров по площади.

На следующем шаге необходимо устранить паразитные микроконтуры, которые были обнаружены на объектах. Избавиться от них можно, вычислив площадь занимаемую этими контурами, а затем просто отсечь контуры с маленькой площадью.

В программу следует внести модификацию:

```
box = np.int0(box) # округление координат
area = int(rect[1][0]*rect[1][1]) # вычисление площади
if area > 500:
  cv.drawContours(img,[box],0,(255,0,0),2)
```

Наконец, вычисляются углы наклона всех прямоугольников относительно горизонта. Здесь не требуется специальных функций OpenCV, достаточно базовых математических функций.

```
#!/usr/bin/env python

import sys
import numpy as np
import cv2 as cv
import math

hsv_min = np.array((0, 54, 5), np.uint8)
hsv_max = np.array((187, 255, 253), np.uint8)

color_blue = (255,0,0)
color_yellow = (0,255,255)
```

```
if __name__ == '__main__':
    fn = 'image2.jpg' # имя файла, который будем анализировать
    img = cv.imread(fn)

    hsv = cv.cvtColor( img, cv.COLOR_BGR2HSV ) # меняем цветовую модель с BGR на HSV
    thresh = cv.inRange( hsv, hsv_min, hsv_max ) # применяем цветовой фильтр
    _, contours0, hierarchy = cv.findContours( thresh.copy(), cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)

    # перебираем все найденные контуры в цикле
    for cnt in contours0:
    rect = cv.minAreaRect(cnt) # пытаемся вписать прямоугольник
    box = cv.boxPoints(rect) # поиск четырех вершин прямоугольника
    box = np.int0(box) # округление координат
    center = (int(rect[0][0]),int(rect[0][1]))
    area = int(rect[1][0]*rect[1][1]) # вычисление площади

    # вычисление координат двух векторов, являющихся сторонам прямоугольника
    edge1 = np.int0((box[1][0] - box[0][0],box[1][1] - box[0][1]))
    edge2 = np.int0((box[2][0] - box[1][0], box[2][1] - box[1][1]))

    # выясняем, какой вектор больше
    usedEdge = edge1
    if cv.norm(edge2) > cv.norm(edge1):
        usedEdge = edge2
    reference = (1,0) # горизонтальный вектор, задающий горизонт

    # вычисляем угол между самой длинной стороной прямоугольника и горизонтом
    angle = 180.0/math.pi * math.acos((reference[0]*usedEdge[0] + reference[1]*usedEdge[1]) / (cv.norm(reference) *cv.norm(usedEdge)))

    if area > 500:
        cv.drawContours(img,[box],0,(255,0,0),2) # рисуем прямоугольник
        cv.circle(img, center, 5, color_yellow, 2) # рисуем маленький кружок в центре прямоугольника
        # выводим в кадр величину угла наклона
        cv.putText(img, "%d" % int(angle), (center[0]+20, center[1]-20),
                cv.FONT_HERSHEY_SIMPLEX, 1, color_yellow, 2)

    cv.imshow('contours', img)

    cv.waitKey()
    cv.destroyAllWindows()
```

Следует помнить, что цветовые фильтры hsv_min и hsv_max нужно каждый раз настраивать под конкретный объект и освещение!

Определение угла поворота прямоугольника в видеопотоке.

```
#!/usr/bin/env python
```

```
import cv2 as cv
import numpy as np
import video
import math

if __name__ == '__main__':
    cv.namedWindow( "result" )
    cap = video.create_capture(0)

    hsv_min = np.array((0, 0, 255), np.uint8)
    hsv_max = np.array((72, 51, 255), np.uint8)

    color_blue = (255,0,0)
    color_red = (0,0,128)

    while True:
        flag, img = cap.read()
        img = cv.flip(img,1)
        try:
            hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV )
            thresh = cv.inRange(hsv, hsv_min, hsv_max)
            _, contours0, hierarchy = cv.findContours( thresh.copy(), cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)

            for cnt in contours0:
                rect = cv.minAreaRect(cnt)
                box = cv.boxPoints(rect)
                box = np.int0(box)
                center = (int(rect[0][0]),int(rect[0][1]))
                area = int(rect[1][0]*rect[1][1])

                edge1 = np.int0((box[1][0] - box[0][0],box[1][1] - box[0][1]))
                edge2 = np.int0((box[2][0] - box[1][0], box[2][1] - box[1][1]))

                usedEdge = edge1
                if cv.norm(edge2) > cv.norm(edge1):
                    usedEdge = edge2

                reference = (1,0) # horizontal edge
                angle = 180.0/math.pi * math.acos((reference[0]*usedEdge[0] + reference[1]*usedEdge[1]) / (cv.norm(reference) *cv.norm(usedEdge)))

                if area > 500:
                    cv.drawContours(img,[box],0,color_blue,2)
                    cv.circle(img, center, 5, color_red, 2)
                    cv.putText(img, "%d" % int(angle), (center[0]+20, center[1]-20), cv.FONT_HERSHEY_SIMPLEX, 1, color_red, 2)
            cv.imshow('result', img)
        except:
            cap.release()
            raise
        ch = cv.waitKey(5)
```

```
        if ch == 27:
            break

    cap.release()
    cv.destroyAllWindows()
```

Индивидуальное задание к лабораторной работе

Индивидуальное задание к лабораторной работе выдает преподаватель. Задание заключается в том, чтобы написать программу для обнаружения какого-либо простого объекта (круга или прямоугольника определенного цвета) на изображении, поступающем с видеокамеры.

Содержание отчета

Отчет по лабораторной работе должен содержать следующее:
– цель работы;
– постановку задачи;
– текст пользовательской программы;
– результат выполнения программы.
Оформление отчета должно соответствовать требованиям ГОСТ 2.105–2003.

Контрольные вопросы

1 Для чего нужна библиотека OpenCV?
2 Как осуществляется захват изображения с видеокамеры?
3 Какие бывают цветовые пространства изображений?
4 Для чего нужны различные цветовые пространства?
5 Каким образом осуществляется поиск прямоугольников и кругов на изображениях?
6 Каким образом осуществляется поиск объектов определенного цвета?

4 Лабораторная работа № 4. Использование алгоритмов искусственных нейронных сетей в программном обеспечении роботов

4.1 Ход работы

Цель работы: изучить вопросы реализации искусственных нейронных сетей в программном обеспечении роботов.

В ходе выполнения работы необходимо изучить:
– основные принципы и возможности библиотеки Tensorflow;
– основные функции библиотеки Tensorflow;

– особенности реализации искусственных нейронных сетей с помощью библиотеки Tensorflow.

После выполнения работы необходимо ответить на контрольные вопросы.

4.2 Основные сведения

Библиотека TensorFlow предназначена для создания программ с использованием искусственных нейронных сетей на языках Python и C++. Данное руководство рассказывает об основных функциях библиотеки TensorFlow и о вопросах их практического использования.

Основные понятия библиотеки tensorflow

Тензор (Tensor) В библиотеке TensorFlow для хранения данных не используются такие типы, как integer, float и string. Вместо этого используются объекты, называемые тензорами.

Примеры тензоров различных типов:

```
# A is a 0-dimensional int32 tensor
A = tf.constant(1234)
# B is a 1-dimensional int32 tensor
B = tf.constant([123,456,789])
# C is a 2-dimensional int32 tensor
C = tf.constant([ [123,456,789], [222,333,444] ])
```

Тензор, возвращаемый функцией tf.constant(1234), называется тензором-константой (constant tensor), т. к. его значение не изменяется.

Сессия (Session). Библиотека TensorFlow использует идею графов как способ визуализации математических процессов. В коде примера создается тензор **“hello_constant”**. Следующим шагом является обработка тензора в сессии.

Код создает сессию **“sess”**, используя функцию **tf.session()**. Функция “sess.run()” обрабатывает тензор и возвращает результат.

```
import tensorflow as tf

# Create TensorFlow object called tensor
hello_constant = tf.constant('Hello World!')

with tf.Session() as sess:
    # Run the tf.constant operation in the session
    output = sess.run(hello_constant)
    print(output)
```

Ввод

Если значение переменной изменяется, то функцию tf.constant использовать не получится. Вместо нее в данном случае применяется функция **tf.plasehолder()** и директива **feed_dict**.

Не представляется возможным просто объявить переменную X, заполнить ее набором данных и передать ее в сессию библиотеки TensorFlow, т. к. модель нейросети будет работать с разными наборами данных, обладающими различными параметрами. Поэтому используется функция **tf.plaseholder().**

Использование дерективы **feed_dict**:

```
x = tf.placeholder(tf.string)

with tf.Session() as sess:
  output = sess.run(x, feed_dict={x: 'Hello World'})
```

Также можно создать более одного тензора:

```
x = tf.placeholder(tf.string)
y = tf.placeholder(tf.int32)
z = tf.placeholder(tf.float32)

with tf.Session() as sess:
   output = sess.run(x, feed_dict={x: 'Test String', y: 123, z: 45.67})
```

Если тип данных, передаваемых **feed_dict,** не совпадает с типом тензора, то будет выдано сообщение “ValueError: invalid literal for...”

Обучение

Обучение искусственной нейронной сети происходит на протяжении N-го количества эпох. Эпоха – это период обучения нейросети, в течение которого производится прямой проход с полным набором данных, а также обратный проход и корректировка весов и порогов модели. Едва ли получится обучить нейросеть всего за одну эпоху. Следовательно, нужно составить программу, циклично обучающую нейросеть на протяжении заданного количества эпох. Важным параметром является скорость обучения. При достаточно большой скорости можно обучить нейросеть за небольшое количество эпох, однако при слишком большой скорости обучение нейросети может не завершиться успешно. Пример программы, в которой производится обучение, представлен в файле **example4.ipynb.**

В первой ячейке указанного выше файла подключаются необходимые библиотеки (TensorFlow, NumPy, math, MatPlotLib).

Затем объявляются две вспомогательные функции: генератор мини-пакетов (см. выше) и функция вывода номера эпохи, значения весовой функции и точности.

В четвертой ячейке сначала указывается размер входного слоя (28 × 28 = 784), затем количество нейронов на выходе (10). После этого производится загрузка набора данных для обучения (training), перекрестной проверки (cross-validation) и тестирования (test) нейросети. Далее с помощью функции **tf.placeholder()** создаются переменные для ввода информации в сессию TensorFlow, переменные весов и порогов с помощью функции **tf.Variable().**

Расчет выхода производится с помощью функций, приведенных в разделе «Математика TensorFlow».

Затем задается весовая функция:

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=labels))
```

Для оптимизации параметров используется метод градиентного спуска:

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cost)
```

Вычисление точности нейросети производится следующим образом:

```
correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(labels, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
После этого производится инициализация всех переменных:
init = tf.global_variables_initializer()
В пятой ячейке запускается сессия:
with tf.Session() as sess:
    sess.run(init)
```

В ходе данной операции в цикле по эпохам производится обучение, причем внутри основного цикла вложен цикл, в котором перебираются мини-пакеты и на каждом из них по порядку производится обучение. После этого выводятся показатели текущей эпохи, а затем производится определение точности. В конце написан небольшой скрипт для тестирования на конкретных примерах.

4.3 Порядок выполнения работы

Создание глубоких нейронных сетей.

Пример реализации многослойного перцептрона приведен в файле **example6.ipynb**

В самом начале производится загрузка набора данных:

```
from tensorflow.examples.tutorials.mnist import input_data
import numpy as np
mnist = input_data.read_data_sets('datasets/ud730/mnist/', one_hot=True, reshape=False)
```

Затем задаются параметры для обучения сети:

```
learning_rate = 0.02
training_epochs = 200
batch_size = 128  # Decrease batch size if you don't have enough memory
display_step = 1

n_input = 784  # MNIST data input (img shape: 28*28)
n_classes = 10  # MNIST total classes (0-9 digits)
n_hidden_layer = 256 # layer number of features
```

Переменная n_hidden_layer определяет количество нейронов в скрытом слое сети.

Веса и пороги нейронной сети:

```
weights = {
   'hidden_layer': tf.Variable(tf.random_normal([n_input, n_hidden_layer])),
   'out': tf.Variable(tf.random_normal([n_hidden_layer, n_classes]))
}
biases = {
   'hidden_layer': tf.Variable(tf.random_normal([n_hidden_layer])),
   'out': tf.Variable(tf.random_normal([n_classes]))
}
```

Для каждого слоя предусмотрены свои веса и пороги, которые в данном случае представлены в виде словаря, к элементам которого можно впоследствии обращаться.

Ввод:

```
x = tf.placeholder("float", [None, 28, 28, 1])
y = tf.placeholder("float", [None, n_classes])
x_flat = tf.reshape(x, [-1, n_input])
```

Здесь функция tf.reshape() позволяет перестроить входной массив с размерностью 28 × 28 × 1 в массив с размерностью 784 × 1.

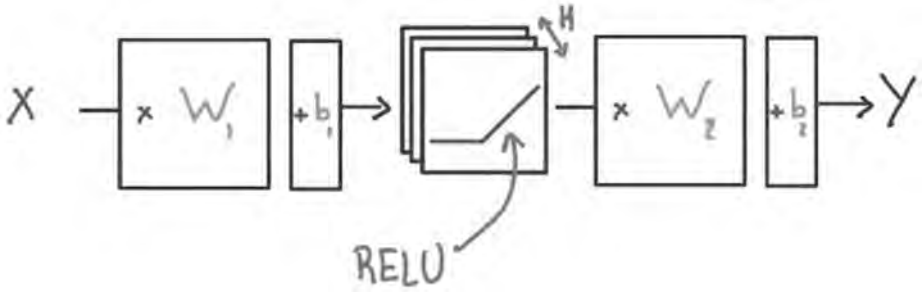

Рисунок 4.1 – Многослойный перцептрон

Реализация многослойного перцептрона:

```
# Hidden layer with RELU activation
layer_1 = tf.add(tf.matmul(x_flat, weights['hidden_layer']),\
   biases['hidden_layer'])
layer_1 = tf.nn.relu(layer_1)
# Output layer with linear activation
logits = tf.add(tf.matmul(layer_1, weights['out']), biases['out'])
```

Оптимизация и определение точности:

```
# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=y))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cost)
```

```
# Calculate accuracy
correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Сессия:

```
# Initializing the variables
init = tf.global_variables_initializer()
# Launch the graph
with tf.Session() as sess:
    sess.run(init)
    # Training cycle
    for epoch in range(training_epochs):
        total_batch = int(mnist.train.num_examples/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_x, batch_y = mnist.train.next_batch(batch_size)
            # Run optimization op (backprop) and cost op (to get loss value)
            sess.run(optimizer, feed_dict={x: batch_x, y: batch_y})
        print_epoch_stats(epoch, sess, batch_x, batch_y)
    # Calculate accuracy for test dataset
    test_accuracy = sess.run(
        accuracy,
        feed_dict={x: test_features, y: test_labels})
    print('Test Accuracy: {}'.format(test_accuracy))
```

Здесь используется функция mnist.train.next_batch(), которая возвращает следующий мини-пакет (mini-batch) данных, используемых при обучении сети.

Индивидуальное задание к лабораторной работе

Индивидуальное задание к лабораторной работе выдается преподавателем и заключается в том, чтобы написать программу для обучения искусственной нейронной сети на заданном наборе данных и произвести ее тестирование.

Содержание отчета

Отчет по лабораторной работе должен содержать следующее:
– цель работы;
– постановку задачи;
– текст пользовательской программы;
– результат выполнения программы.
Оформление отчета должно соответствовать требованиям ГОСТ 2.105–2003.

Контрольные вопросы

1 Какие библиотеки языка программирования Python используются для создания алгоритмов искусственных нейронных сетей?

2 В чем заключаются основные особенности библиотеки TensorFlow?

3 Как с помощью библиотеки TensorFlow создается многослойный перцептрон?

4 Как производится обучение искусственных нейронных сетей с помощью библиотеки TensorFlow?

Список литературы

1 **Иванов, А. А.** Основы робототехники : учебное пособие / А. А. Иванов. – Москва : ФОРУМ, 2015. – 224 с.

2 **Крейг, Д. Дж.** Введение в робототехнику. Механика и управление : пер. с англ. / Д. Дж. Крейг ; под ред. В. Е. Павловского. – Москва ; Ижевск : Ин-т компьют. исслед., 2013. – 564 с. : ил.

3 **Петин, В. А.** Arduino и Raspberry Pi в проектах Internet of Things / В. А. Петин. – Санкт-Петербург : БХВ-Петербург, 2016. – 320 с. : ил.

4 **Вандер Плас, Дж.** Python для сложных задач. Наука о данных и машинное обучение : пер. с англ. / Дж. Вандер Плас. – Санкт-Петербург : Питер, 2016. – 576 с. : ил.