Raspberry Pi Supercomputing and Scientific Programming

MPI4PY, NumPy, and SciPy for Enthusiasts

Ashwin Pajankar

Apress

Raspberry Pi Supercomputing and Scientific Programming

Ashwin Pajankar
Nashik, Maharashtra, India

ISBN-13 (pbk): 978-1-4842-2877-7 ISBN-13 (electronic): 978-1-4842-2878-4
DOI 10.1007/978-1-4842-2878-4

Library of Congress Control Number: 2017943339

Any source code or other supplementary materials referenced by the author in this text are available to readers at `www.apress.com`. For detailed information about how to locate your book's source code, go to `www.apress.com/source-code/`. Readers can also access source code at SpringerLink in the Supplementary Material section for each chapter.

Contents at a Glance

Contents

Introduction

"I'm sorry, buddy, you've got a C in parallel programming," my classmate said to me. It was my second semester at IIIT Hyderabad, where I was pursuing my graduate studies towards an MTech in computer science and engineering. To be very honest, I was really not expecting an A in parallel programming. However, I was not expecting a C either. Like most of the folks in my class, I was expecting a B or a B- in the worst case. As I mentioned earlier, it was my second semester and I was getting used to boring and rigorous academics at the Institute. I was also getting used to seeing C and C- grades on my scorecard. I was really starting to wonder if I had what it took to pursue graduate studies at the coveted and hallowed Institute. Little did I know that the worst was yet to come, because exactly one year after that, I saw two Ds on my scorecard.

I wondered if there was a way to make scary subjects like parallel programming more interesting and appealing to undergrad and graduate students. I vividly remember that our prof for the course mentioned that the US Air Force created a cluster of PS3s (`http://phys.org/news/2010-12-air-playstation-3s-supercomputer.html`). You can also check the Wikipedia page (`https://en.wikipedia.org/wiki/PlayStation_3_cluster`) for the same. I fancied owning a small-scale cluster like that. However, the biggest hurdle was and still is the cost associated with individual unit. A PS3 costs around 250 USD and a PS2 costs 100 USD. Moreover, it is a really tedious task to install Linux OS on the PS2 and PS3. I have nothing against Sony or the PS2/PS3, as Playstations are excellent gaming consoles, but I wanted something cheaper and simpler to build the cluster. So I shelved my plans to build a cluster for the time being and got engrossed in academics at the Institute.

Fast forward two years: I saw a bunch of kids booting up a strange-looking PCB (printed circuit board) with Linux and I literally jumped with excitement. When I asked them what it was, they said, "Raspberry Pi." This is how I was introduced to Raspberry Pi. The occasion was a Hackathon organized at my former workplace for specially abled kids. When I saw those lovely and creative kids using a little computer, I immediately decided to explore it, and after six months my first book on Raspberry Pi (`www.packtpub.com/hardware-and-creative/raspberry-pi-computer-vision-programming`) got published.

It has been a while since my first book got published. Raspberry Pi underwent major revisions. Multiple newer and more powerful models of Pi have been introduced. I explored the world of single board computers in detail as I (correctly) thought that SBCs could be best for cluster. Then I built a cluster with three units of Raspberry Pis for a proof of concept. I went on building the clusters for various organizations, T-Schools, and universities all over India.

Raspberry Pi also serves as an excellent learning platform for scientific programming. We can use it for exploring important concepts in the scientific domains of signal and image processing, or to perform symbolic computations with SymPy.

I have explored the domain of scientific programming in detail in this book. The book has almost 100 coding examples to demonstrate and explore the world of scientific programming.

I hope that this book will help the scientific community and young researchers to use Raspberry Pi and Python as tools to explore scientific frontiers.

Why This Book?

As I said earlier, I found learning and exploring the topics in parallel and scientific programming quite boring and difficult. However, these are some of the most useful areas in computer science, with many applications in the real world. In my past and current jobs, I routinely used various parallel and scientific programming libraries for accomplishing various tasks. I want more people to get interested in this wonderful field. However, when people approach me for guest lectures and talks on this topic, they have a common complaint. They tell me that it's difficult to get started with Raspberry Pi, parallel and scientific programming due to lack of reliable materials and references over the Internet. There are many tutorials and videos which teach people how to create a small multi-node cluster with Raspberry Pi. But due to unfamiliarity with the notion of single board computers, they found it difficult to grasp the information provided. So I decided to combine the topics of Raspberry Pi setup and supercomputers in a single book. This book teaches readers how to build a Raspberry Pi cluster as well as how to use Python to exploit its computational power for various scientific tasks. The cluster we will build in this book will be very basic one. It will be fit to be deployed in academic and research institutions. We will also learn how to get started with symbolic programming, scientific programming, image processing, and signal processing in this book. The book has within its scope the following:

- Introduction to single board computers, Python 3, and Raspberry Pi
- Basic concepts in supercomputing
- Preparing a node and building an entire cluster
- Parallel programming in Python
- Symbolic mathematical programming
- Scientific programming
- Image processing
- Signal processing
- Visualization with Matplotlib

Who Is This Book For?

This book is for beginners in Raspberry Pi and parallel programming. It is for people who are interested in learning how to create a low-cost supercomputer and get started with scientific programming at very low cost. However, this book is not for people completely new to the world of computer science. I assume that you have a decent knowledge of computers and that you are not new to the most fundamental concepts related to it. The perfect reader of this book may be both a hobbyist and a student, someone who is familiar with computers and wants to learn a more powerful tool to explore computer science further. Makers and hackers will also find this book interesting. Finally, if you have no idea why you are reading this book, you might get interested in Raspberry Pi, supercomputers, scientific and parallel programming. I hope all the readers will enjoy reading this book as much as I enjoyed writing it.

What This Book Is Not

This book is not a book for learning Python 3 programming and syntax from scratch. It is more of a DIY cookbook for Raspberry Pi, scientific programming, and supercomputing. If your understanding of coding is limited or if you are not from a computer science background, then you will find it difficult to follow this book.

How This Book Is Organized

This book has 14 chapters. Here is a sneak peek into the chapters of the book:

Chapter 1: This chapter introduces the readers to the history and philosophy of single board computers. It explores Raspberry Pi basics, and teaches readers to set up the Raspberry Pi and connect it to a network.

Chapter 2: This chapter introduces the readers to important Linux commands. It also details the procedure to establish remote connectivity with Raspberry Pi.

Chapter 3: The aim of this chapter is to introduce the readers to Python history and philosophy. It discusses the differences between Python 2 and Python 3. It also explores various modes of the Python 3 interpreter.

Chapter 4: This chapter serves to introduce the concept of supercomputing to the readers.

Chapter 5: This chapter will demonstrate how to install MPI4PY on Raspberry Pi.

Chapter 6: This chapter teaches the readers how to build a supercomputer of Pis.

Chapter 7: This chapter teaches the readers how to overclock various models of Pi safely.

Chapter 8: This detailed chapter introduces readers to MPI4PY programming with Python 3. It explores many important concepts and constructs in parallel programming.

Chapter 9: This chapter introduces readers to the components of the SciPy stack. It also gets the readers started with symbolic programming using SymPy.

Chapter 10: This chapter introduces readers to the world of numerical computation with NumPy.

Chapter 11: This chapter introduces readers to the various modules in the SciPy library.

Chapter 12: Readers explore the amazing world of signals and signal processing graphically in this chapter.

Chapter 13: Readers will explore the basics of image processing in this chapter.

Chapter 14: This chapter provides a brief glimpse into the world of data visualization with Matplotlib in Python.

How to Get The Most Out of This Book

It is easy to leverage the book to gain the maximum by religiously abiding by the following:

- Read the chapters thoroughly. Do NOT skip any chapter. Perform hands-on by following the step-by-step instructions stated in the code examples. Do NOT skip any code example. If need be, repeat it a second time or until the concept is firmly etched in your mind.
- Join a Python community or discussion forum.
- Read online documentation available for MPI4PY, NumPy, and SciPy.
- Read blogs on single board computers, Raspberry Pi, supercomputers, parallel programming, scientific programming, and Python 3.

Where Next?

I have endeavored to unleash the power of supercomputing and scientific libraries in Python 3 as an aid to the Raspberry Pi community. I recommend you read the book from cover to cover without skipping any chapter, text, code example, note, reference, or exercise.

A Quick Word for Instructors

I have paid attention to the sequence of chapters and the flow of topics within each chapter. This is particularly to assist my fellow instructors and academicians in carving out a syllabus for their training from the table of contents of this book.

I have ensured that each concept discussed in the book is with adequate hands-on content to enable you to teach better and provide ample hands-on practice to your students.

Happy learning, supercomputing, scientific programming, and Pythoning!

Ashwin Pajankar, author

CHAPTER 1

Introduction to Single Board Computers and Raspberry Pi

We will start our exciting journey of exploration into the scientific domain of **supercomputing and scientific programming** with **Raspberry Pi**. But for us to begin the journey, we must be comfortable with the basics of the **single board computers** and Raspberry Pi. In this chapter, we will study the definition, history, and philosophy behind single board computers. We will first compare it with a regular computer. We will then move on to the most popular and best-selling single board computer of all time, the Raspberry Pi. By the end of this chapter, readers will have adequate knowledge to set up their own Raspberry Pi independently. This chapter aims to make the readers comfortable with the very basic concepts of single board computers and the setup of the Raspberry Pi.

Single Board Computers (SBCs)

A single board computer (which will be referred to as SBCs from henceforth throughout the entire book) is a fully functional computer system built around a single printed circuit board. An SBC has microprocessor(s), memory, input/output, and other features required of a minimal functional computer. Unlike a desktop personal computer (PC), most of the SBCs do not have expansion slots for peripheral functions or expansion. As all the components such as processor(s), RAM, GPU, etc., are integrated on a single PCB, we cannot upgrade an SBC.

Few SBCs are made to plug into a backplane for system expansion. SBCs come in many varieties, sizes, shapes, form factors, and sets of features. Due to the advances in electronics and semiconductor technologies, prices of most SBCs are very low. One of the most important features of SBCs is cost effectiveness. With a price around $50 apiece, we have in our hands a development tool suitable for new applications, hacking, debugging, testing, hardware development, and automation systems.

SBCs are usually manufactured in the following form factors:

- Pico-ITX
- PXI
- Qseven

- VMEbus
- VPX
- VXI
- AdvancedTCA
- CompactPCI
- Embedded Compact Extended (ECX)
- Mini-ITX
- PC/104
- PICMG

Differences Between SBCs and Regular Computers

The following is a table (Table 1-1) of the differences between SBCs and regular computers.

Table 1-1. *Differences Between SBCs and Regular Computers*

Single Board Computer	Regular Computer
It is not modular.	It is modular.
Its components cannot be upgraded or replaced.	Its components can be upgraded or replaced.
It's a System-On-Chip.	It's not a System-On-Chip.
It has small form factor.	It has large form factor.
It is portable.	It is mostly non-portable or semi-portable.
It consumes less power.	It consumes more power.
It is cheaper than a regular computer.	It is costlier than an SBC.

System on Chip

All the SBCs are predominantly SoCs. A system on a chip or system on chip (SoC or SOC) is an integrated circuit (IC) that has all the components of a computer on a single chip. SoCs are very common in mobile electronic devices because of their low power consumption and versatility. SoCs are widely used in mobile phones, SBCs, and embedded hardware. A SoC has all the hardware and the software needed for its operation.

SoC vs. Regular CPU

The biggest advantage of using a SoC is its size. If we use a CPU, it's very hard to make a compact computer, only because of the sheer number of individual chips and other components that we need to arrange on a board. However, using SoCs, we can place complete application-specific computing systems in smartphones and tablets, and still have plenty of space for batteries, antennae, and other add-ons required for remote telephony and data communication.

Due to the very high level of integration and compact size, a SoC uses considerably less power than a regular CPU. This is a significant advantage of SoCs when it comes to mobile and portable systems. Also, reducing the number of chips by eliminating redundant ICs on a computer board results in the compact board size.

History of SBCs

Dyna-Micro was the first true SBC. It was based on the Intel C8080A and used Intel's first EPROM, the C1702A. The dyna-micro was re-branded and marketed by E&L Instruments of Derby, CT in 1976 as the MMD-1 (Mini-Micro Designer 1). It became famous as the leading example of microcomputers. SBCs were very popular in the earlier days of computing, as many home computers were actually SBCs. However, with the rise of PCs, the popularity of SBCs declined. Since 2010, there has been a resurgence in the popularity of SBCs due to lower production costs associated with SBCs.

Apart from the MMD-1, a few popular historical SBCs are the following:

- BBC Micro was built around a MOS Technology 6502A processor running at 2 MHz.
- Ferguson Big Board II was a Zilog Z80 based computer running at 4MHz.
- Nascom was another Zilog Z80 based computer.

Popular SBC Families

Based on manufacturers and designers, SBCs are grouped into families, models, and generations. A few popular SBC families are

- Raspberry Pi by Raspberry Pi Foundation
- Banana Pi and Banana Pro
- Intel Edison and Galileo
- Cubieboard
- Beaglebone and Beagleboard

Raspberry Pi

Raspberry Pi is a family of credit card-sized SBCs developed in the United Kingdom by the Raspberry Pi Foundation. The Raspberry Pi Foundation was founded in 2009. The aim behind developing Raspberry Pi is to promote the teaching of basic computer science in schools and developing countries by providing a low-cost computing platform.

The Raspberry Pi Foundation's Raspberry Pi was released in 2012. It was a massive hit which sold over two million units in two years. Subsequently, the Raspberry Pi Foundation revised versions of the Raspberry Pi. They also released other accessories for the Pi.

You can find more information about the Raspberry Pi Foundation on the Raspberry Pi Foundation's website (`www.raspberrypi.org`).

The product page for Raspberry Pi's current production models and other accessories is `www.raspberrypi.org/products`.

I have written, executed, and tested all the code examples of this book on Raspberry Pi Models B+, 2B, and 3B. Raspberry Pi 3 Model B (also known as 3B) is the most recent model of Raspberry Pi. Let us look at the specifications (Refer Table 1-2) of Raspberry Pi 3 Model B.

Table 1-2. *Specifications of Raspberry Pi 3 Model B*

Release Date	February 2016
Architecture	ARMv8
SoC Broadcom	BCM2837
CPU	1.2 GHz 64-bit quad-core ARM Cortex-A53
GPU	Broadcom VideoCore IV (3D part of GPU @ 300 MHz, video part of GPU @ 400 MHz)
Memory	1 GB (shared with GPU)
USB	2.0 ports 4
Video Output	HDMI rev 1.3 and Composite Video RCA jack
On-board storage	Micro SDHC slot
On-board network	10/100 Mbps Ethernet, Bluetooth, and WiFi
Power source	5V via MicroUSB
Power ratings	800 mA (4W)

The following (Figure 1-1) is the top view of Raspberry Pi 3 Model B.

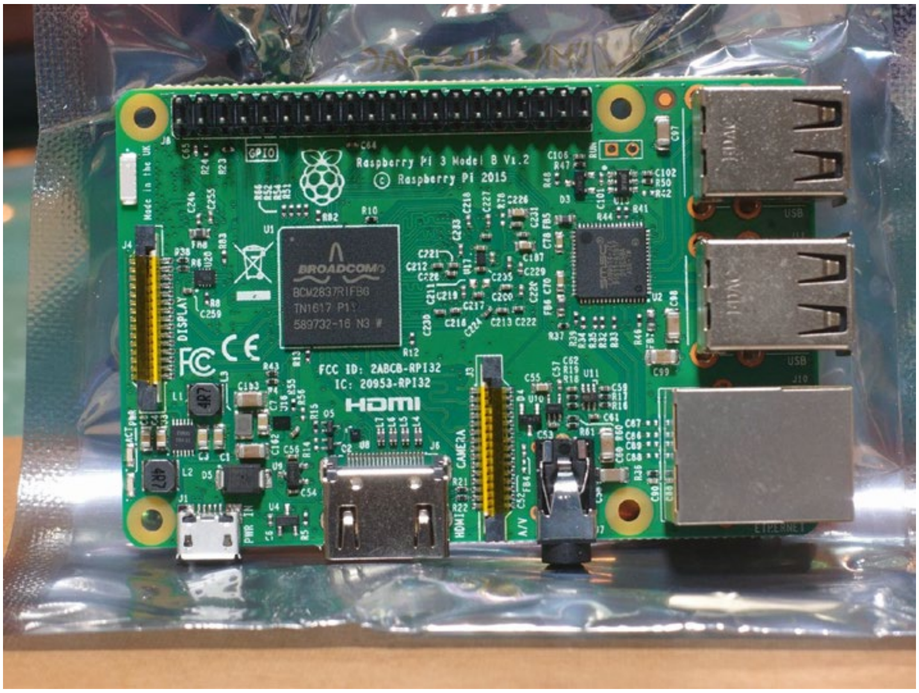


Figure 1-1. *Raspberry Pi 3 Model B top view*

The following (Figure 1-2) is the bottom view of Raspberry Pi 3 Model B.

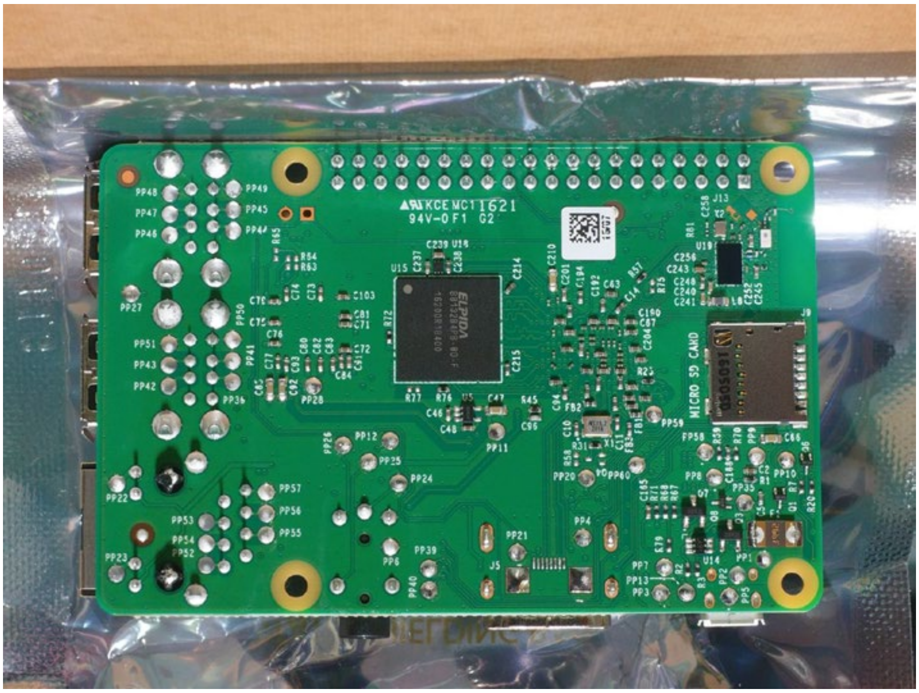


Figure 1-2. *Raspberry Pi 3 Model B bottom view*

We can get more information on Raspberry Pi 3 Model B by visiting its product page (`www.raspberrypi.org/products/raspberry-pi-3-model-b`).

The following table (Table 1-3) lists the specifications of Raspberry Pi 2 Model B.

Table 1-3. *Specifications of Raspberry Pi 2 Model B*

Release Date	February 2015
Architecture	ARMv7
SoC Broadcom	BCM2836
CPU	900 MHz 32-bit quad-core ARM Cortex-A7
GPU	Broadcom VideoCore IV @ 250 MHz
Memory	1 GB (shared with GPU)
USB	2.0 ports 4
Video Output	HDMI rev 1.3 and Composite Video RCA jack
On-board storage	Micro SDHC slot
On-board network	10/100 Mbps Ethernet, Bluetooth, and WiFi
Power source	5V via MicroUSB
Power ratings	800 mA (4W)

We can get more information on Raspberry Pi 2 Model B by visiting its product page (www.raspberrypi.org/products/raspberry-pi-2-model-b/).

The following table (Table 1-4) lists the specifications of Raspberry Pi 1Model B+.

Table 1-4. *Specifications of Raspberry Pi 1 Model B+*

Release Date	July 2014
Architecture	ARMv6
SoC Broadcom	BCM2835
CPU	700 MHz single-core ARM1176JZF-S
GPU	Broadcom VideoCore IV @ 250 MHz
Memory	512 MB (shared with GPU)
USB	2.0 ports 4
Video Output	HDMI rev 1.3 and Composite Video RCA jack
On-board storage	Micro SDHC slot
On-board network	10/100 Mbps Ethernet, Bluetooth, and WiFi
Power source	5V via MicroUSB
Power ratings	800 mA (4W)

We can get more information on Raspberry Pi 2 Model B by visiting its product page (www.raspberrypi.org/products/model-b-plus/).

Raspberry Pi Setup

We have to set up Raspberry Pi before we can begin to use it for exploration and adventure. Let's see in detail how to set it up. As I have mentioned earlier, I am using Raspberry Pi 3 Model B for the setup. The setup process is exactly the same for Raspberry Pi 2 Model B and Raspberry Pi 1 Model B+. Let's see the list of hardware materials you will need for setup.

Hardware required for Raspberry Pi setup

The following hardware is required for setup.

Raspberry Pi

We need to use Raspberry Pi 3 Model B or Raspberry Pi 2 Model B or Raspberry Pi 1 Model B+ for the setup.

Computer

A Windows computer or laptop with an Internet connection is required. We need to use a computer to prepare a microSD card with a Raspbian OS image for the Pi.

I/O Devices

A standard USB keyboard and a USB mouse are needed.

MicroSD card

A microSD card (see Figure 1-3 for example) with storage capacity of at least 8 GB is needed. We will use the card for secondary storage for the Pi. A card of Class 10 is recommended as the data transfer speed with class 10 is great. I recommend using at least an 8 GB card to be on the safe side. Choosing a 16 GB card will be adequate for most cases.

Figure 1-3. *Class 10 microSD card (image from www.flickr.com/photos/ssoosay/)*

■ **Note** Before purchasing the card, do visit the link http://elinux.org/RPi_SD_cards to check the compatibility of the card with the Raspberry Pi.

Power Supply

For all the Raspberry Pi models a 5V Micro USB power supply unit (PSU) is required. The recommended current capacity of PSU for Raspberry Pi 3 Model B is 2.5 Amp. For all the other models 2 Amp PSU is more than enough.

You can find Raspberry Pi's official power supply at https://thepihut.com/products/official-raspberry-pi-universal-power-supply.

SD/microSD Card Reader

We also need a card reader. Many laptops have a built-in SD card reader.

In case the laptop or the card reader works with SD cards only, then we need an additional microSD-to-SD card adapter. The following figure (Figure 1-4) shows an adapter.

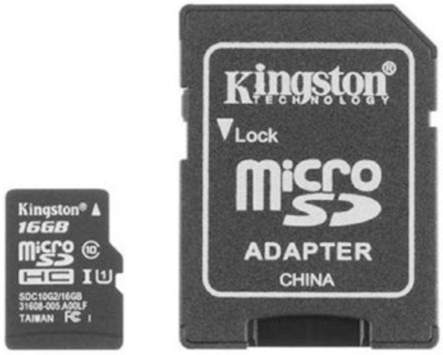


Figure 1-4. *Card reader and microSD-to-SD adapter (image from* `www.flickr.com/photos/sparkfun/`*)*

Monitor

We need an HDMI monitor or a VGA Monitor.

For HDMI monitors we need an HDMI cable (see Figure 1-5). It is usually packaged with the HDMI monitor.

Figure 1-5. *HDMI male-to-male cable (image from* `www.flickr.com/photos/sparkfun/`*)*

For VGA monitors, we need a VGA cable (see Figure 1-6). This too is usually packaged with the VGA monitor.

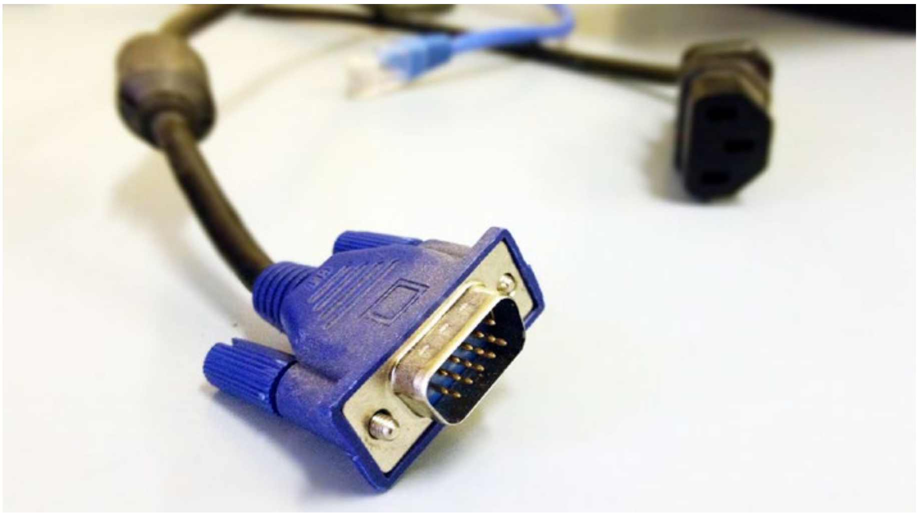


Figure 1-6. *VGA cable (image from `www.flickr.com/photos/124242273@N07/`)*

If we are using a VGA monitor, we will need an HDMI-to-VGA adapter, as Raspberry Pi only has an HDMI port for video output (Figure 1-7).

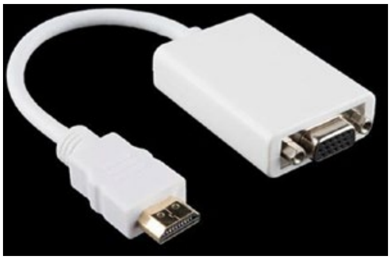


Figure 1-7. *HDMI-to-VGA adapter (image from `www.flickr.com/photos/sparkfun/`)*

Manual Preparation of the MicroSD Card for Raspberry Pi

Preparing the microSD card for Pi manually is the best way of installing an OS into a microSD card for single board computers. Many users (including me) prefer it because it allows the contents of the microSD card to be modified manually (if needed) before it is used for booting. The other way to prepare the microSD is to use **NOOBS** (**N**ew **O**ut **O**f the **B**ox **S**oftware), which I have not used in this book.

Manual preparation allows us to access the configuration files like `/boot/config.txt` before booting. We might have to modify the configuration files in a few cases (we will discuss that soon) before booting up the Pi. The default Raspbian image has two partitions, `boot` and `system`. Please use at least a 16 GB microSD card for the Pi, considering any possible future upgrades to the OS.

Download the Required Free Software

Let's download the required software.

Download Accelerator Plus

Download the **Download Accelerator Plus** setup from its download page (`www.speedbit.com/dap/download/downloading.asp`). This freeware is used to manage downloads. It is useful for large downloads as we can pause and resume downloads. If your computer shuts down suddenly or the internet is interrupted, it resumes the download from the last checkpoint. Once you download and install it, use it to manage the following downloads.

Win32DiskImager

Download the **Win32DiskImager** setup from its download page (`https://sourceforge.net/projects/win32diskimager/files/latest/download`). Install.

WinZip or WinRaR

We need a file extraction utility. Download WinZip (`http://www.winzip.com/win/en/index.htm`) or WinRaR (`http://www.win-rar.com/download.html`). Install either of them.

Download and Extract the Raspbian OS Image

We will use the Raspbian OS for the Pi. We will discuss Raspbian in detail in the later part of the chapter. For now, download the latest zip of the image of the Raspbian OS from `www.raspberrypi.org/downloads/raspbian`. Extract the image zip file with WinZip or WinRaR.

Writing the Raspbian OS Image to the MicroSD Card

Insert the microSD card into the card reader. If your computer or laptop has a built-in card reader, insert it there. You might have to use a microSD-to-SD card adapter if the card reader or your computer only has a slot for SD cards.

Open **Win32DiskImager**. Select the location of the image file and click the Write button. See the following Figure 1-8.

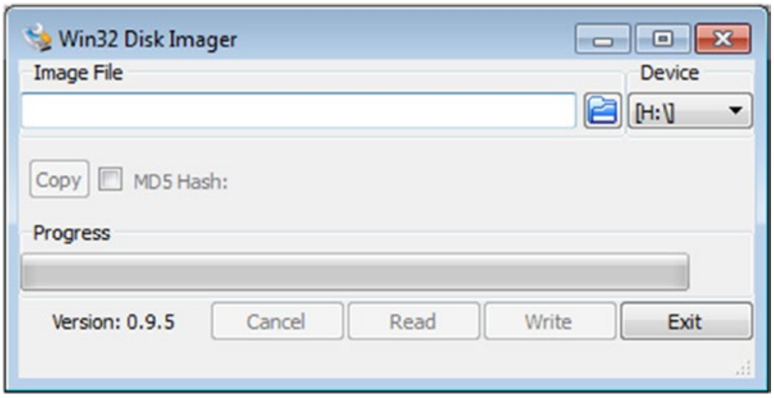
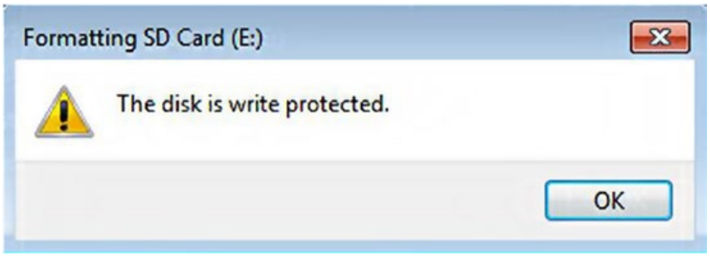


Figure 1-8. *Win32 Disk Imager*

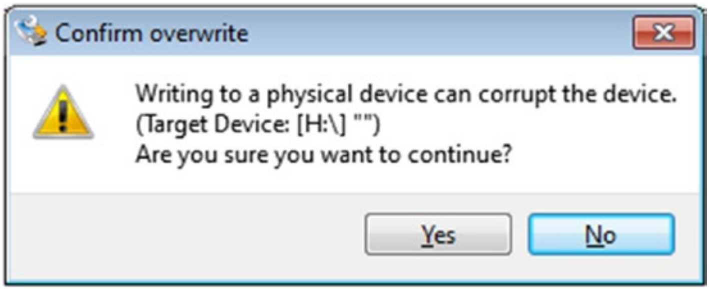
If you see the following warning message (Figure 1-9), then toggle the write protection notch of the card reader or the SD card adapter (or both). Then click the Write button again.



The disk is write protected.

Figure 1-9. *Write protection error message*

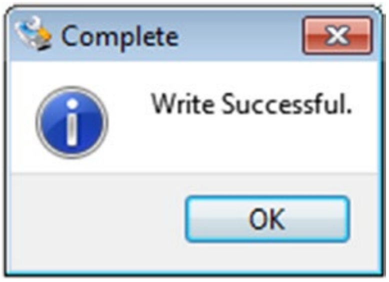
The following (Figure 1-10) warning message will be displayed. Click the Yes button to continue.



Writing to a physical device can corrupt the device.
(Target Device: [H:\] "")
Are you sure you want to continue?

Figure 1-10. *Overwrite warning message*

Once the OS image is written on the SD card, the following (Figure 1-11) message will be displayed. Click the OK button.


Complete

Write Successful.

Figure 1-11. *Write successful message*

The Raspbian OS has been flashed to the microSD card.

Altering the Contents of the config.txt File for VGA Monitors

---

■ **Note** This step is a must if you are planning to use a VGA monitor. Please skip this step if you are using an HDMI monitor.

We must use an HDMI-to-VGA cable with a VGA display. We also need to change the contents of `config.txt` to get the Pi working with VGA monitors. We will learn more about `config.txt` in the later part of the chapter.

Insert the microSD card into the card reader again and browse it in **Windows Explorer**. In **Windows Explorer**, it will be represented as a removable media drive labeled as `boot`.

Open file `config.txt` and make following changes to the file:

- Change `#disable_overscan=1` to `disable_overscan=1`
- Change `#hdmi_force_hotplug=1` to `hdmi_force_hotplug=1`
- Change `#hdmi_group=1` to `hdmi_group=2`
- Change `#hdmi_mode=1` to `hdmi_mode=16`
- Change `#hdmi_drive=2` to `hdmi_drive=2`
- Change `#config_hdmi_boost=4` to `config_hdmi_boost=4`

Save the file after making the above-mentioned changes.

The microSD card is now ready for the Pi and a VGA Monitor.

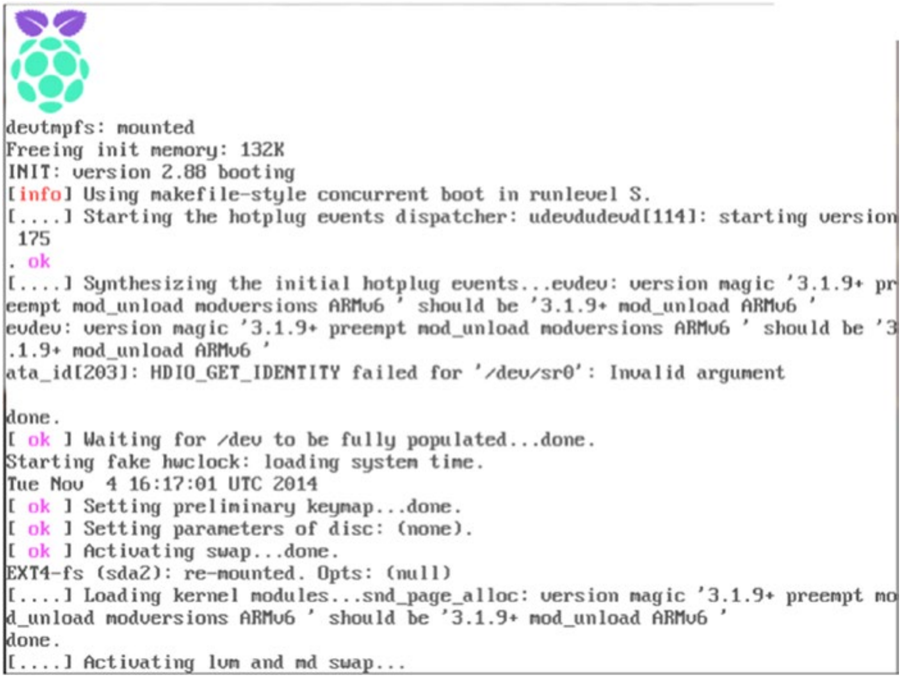
Booting up the Pi

Let's boot the Pi up with the prepared microSD card. The steps for that are as follows:

1. If you are using an HDMI monitor, connect the monitor directly to the Pi's HDMI port using the HDMI male-to-male cable. If you are using a VGA monitor, use the HDMI-to-VGA adapter to convert HDMI signals to VGA.
2. Insert the microSD card into the microSD card slot of the Pi.
3. Connect the USB mouse and the USB keyboard.
4. At this point, make sure that the power is switched off. Connect the Pi to the power supply with the micro USB power cable we discussed earlier.
5. Connect the monitor to the power supply.
6. Check all the connections. Switch on the power supply of the Pi and the monitor.

At this point, the Raspberry Pi will boot up.

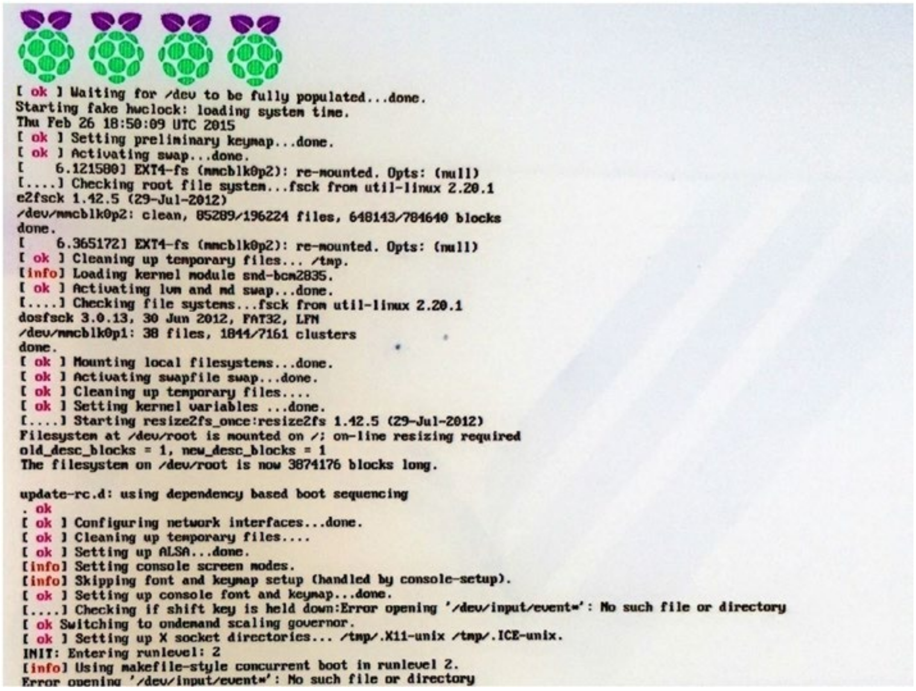
For all the models of Raspberry Pi with the single core processor, the boot screen will be as follows (Figure 1-12).



```
devtmpfs: mounted
Freeing init memory: 132K
INIT: version 2.88 booting
[info] Using makefile-style concurrent boot in runlevel S.
[....] Starting the hotplug events dispatcher: udevdudevd[114]: starting version
 175
. ok
[....] Synthesizing the initial hotplug events...evdev: version magic '3.1.9+ pr
eempt mod_unload modversions ARMv6 ' should be '3.1.9+ mod_unload ARMv6 '
evdev: version magic '3.1.9+ preempt mod_unload modversions ARMv6 ' should be '3
.1.9+ mod_unload ARMv6 '
ata_id[203]: HDIO_GET_IDENTITY failed for '/dev/sr0': Invalid argument

done.
[ ok ] Waiting for /dev to be fully populated...done.
Starting fake hwclock: loading system time.
Tue Nov  4 16:17:01 UTC 2014
[ ok ] Setting preliminary keymap...done.
[ ok ] Setting parameters of disc: (none).
[ ok ] Activating swap...done.
EXT4-fs (sda2): re-mounted. Opts: (null)
[....] Loading kernel modules...snd_page_alloc: version magic '3.1.9+ preempt mo
d_unload modversions ARMv6 ' should be '3.1.9+ mod_unload ARMv6 '
done.
[....] Activating lvm and md swap...
```

Figure 1-12. *Single-core CPU RPi model boot screen*

For all the models of Raspberry Pi with the quad-core processor, the boot screen will be as follows (Figure 1-13).



```
[ ok ] Waiting for /dev to be fully populated...done.
Starting fake hwclock: loading system time.
Thu Feb 26 18:50:09 UTC 2015
[ ok ] Setting preliminary keymap...done.
[ ok ] Activating swap...done.
[    6.121580] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
[....] Checking root file system...fsck from util-linux 2.20.1
e2fsck 1.42.5 (29-Jul-2012)
/dev/mmcblk0p2: clean, 85289/196224 files, 648143/784640 blocks
done.
[    6.365172] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
[ ok ] Cleaning up temporary files... /tmp.
[info] Loading kernel module snd-bcm2835.
[ ok ] Activating lvm and md swap...done.
[....] Checking file systems...fsck from util-linux 2.20.1
dosfsck 3.0.13, 30 Jun 2012, FAT32, LFN
/dev/mmcblk0p1: 38 files, 1844/7161 clusters
done.
[ ok ] Mounting local filesystems...done.
[ ok ] Activating swapfile swap...done.
[ ok ] Cleaning up temporary files....
[ ok ] Setting kernel variables ...done.
[....] Starting resize2fs_once:resize2fs 1.42.5 (29-Jul-2012)
Filesystem at /dev/root is mounted on /; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/root is now 3874176 blocks long.

update-rc.d: using dependency based boot sequencing
. ok
[ ok ] Configuring network interfaces...done.
[ ok ] Cleaning up temporary files....
[ ok ] Setting up ALSA...done.
[info] Setting console screen modes.
[info] Skipping font and keymap setup (handled by console-setup).
[ ok ] Setting up console font and keymap...done.
[....] Checking if shift key is held down:Error opening '/dev/input/event*': No such file or directory
[ ok Switching to ondemand scaling governor.
[ ok ] Setting up X socket directories... /tmp/.X11-unix /tmp/.ICE-unix.
INIT: Entering runlevel: 2
[info] Using makefile-style concurrent boot in runlevel 2.
Error opening '/dev/input/event*': No such file or directory
```

Figure 1-13. *Quad-core CPU RPi Model boot screen*

Once the Pi boots up, the monitor displays the desktop as follows (Figure 1-14).

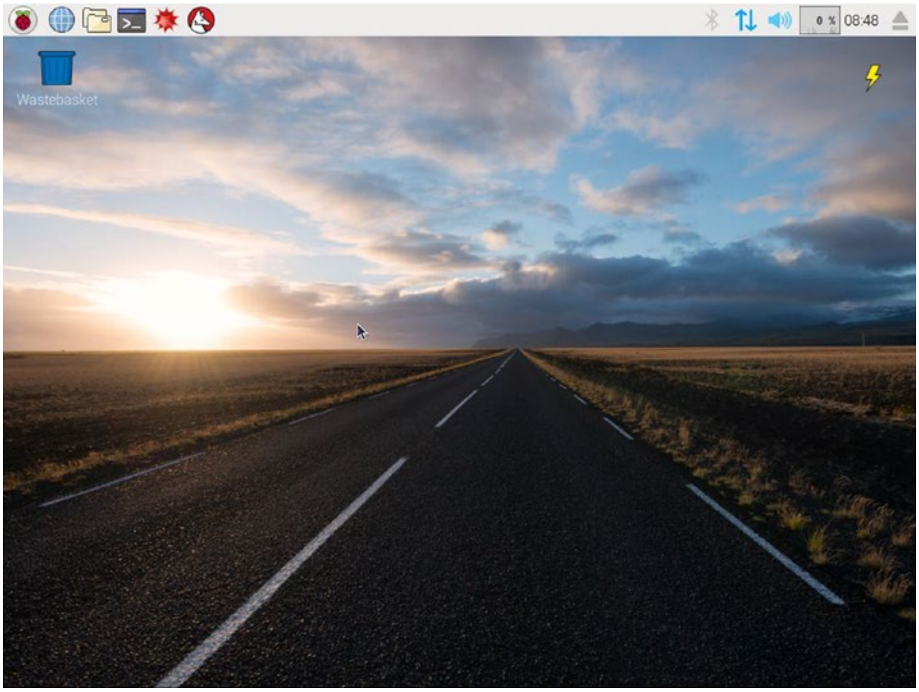


Figure 1-14. *Raspbian Desktop (as of February 2017)*

Configuring the Pi

We need to configure the Pi now for further use. Let's do that.

On the desktop, there is a taskbar. In the taskbar, we find the following icon (Figure 1-15).

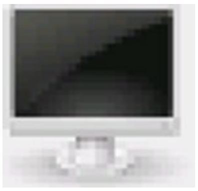
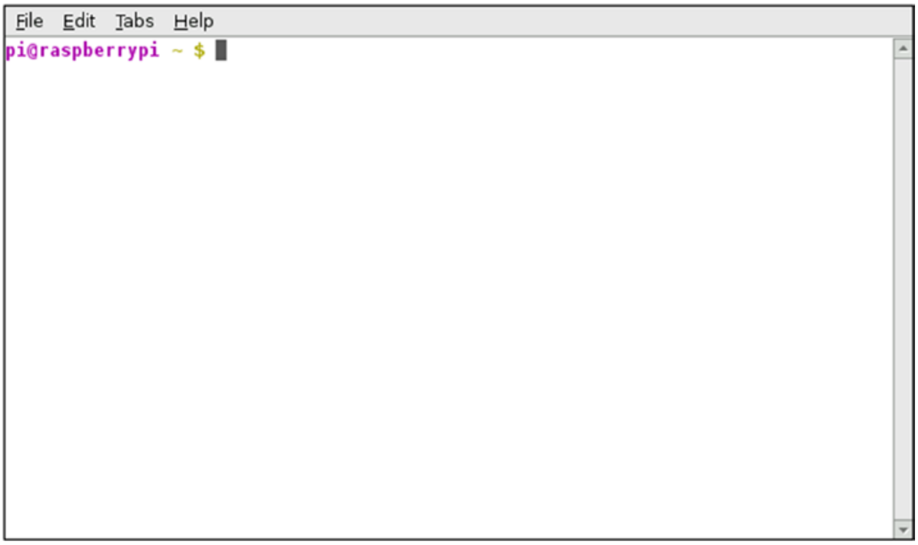


Figure 1-15. *LXTerminal icon*

Click the icon and LXTerminal window (Figure 1-16) will open.

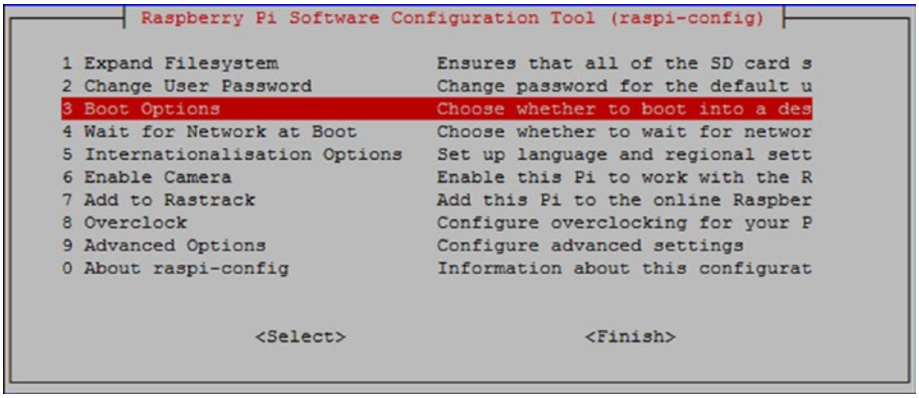


```
pi@raspberrypi ~ $ 
```

Figure 1-16. *LXTerminal Window*

The terminal is a desktop-independent VTE-based terminal emulator for LXDE without any unnecessary dependency. Type `sudoraspi-config` in the prompt and press Enter. The `raspi-config` is the configuration tool for Raspberry Pi.

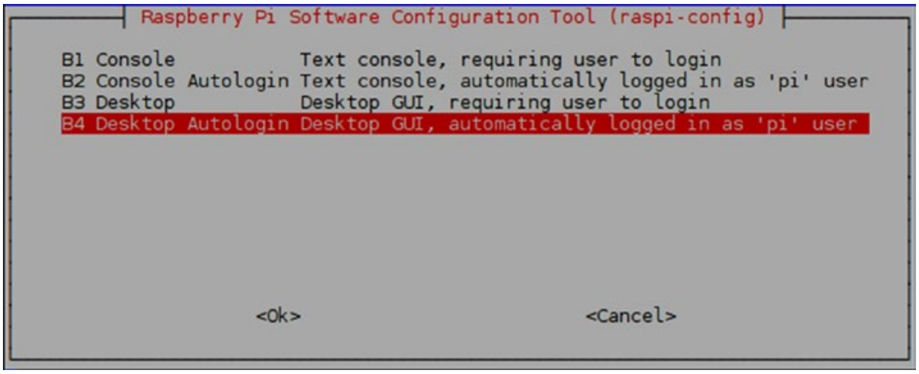
Navigate to the boot options (highlighted in Figure 1-17).



```
Raspberry Pi Software Configuration Tool (raspi-config)

 1 Expand Filesystem               Ensures that all of the SD card s
 2 Change User Password            Change password for the default u
 3 Boot Options                    Choose whether to boot into a des
 4 Wait for Network at Boot        Choose whether to wait for networ
 5 Internationalisation Options    Set up language and regional sett
 6 Enable Camera                   Enable this Pi to work with the R
 7 Add to Rastrack                 Add this Pi to the online Raspber
 8 Overclock                       Configure overclocking for your P
 9 Advanced Options                Configure advanced settings
 0 About raspi-config              Information about this configurat

              <Select>                       <Finish>
```

Figure 1-17. *raspi-config with boot options highlighted*

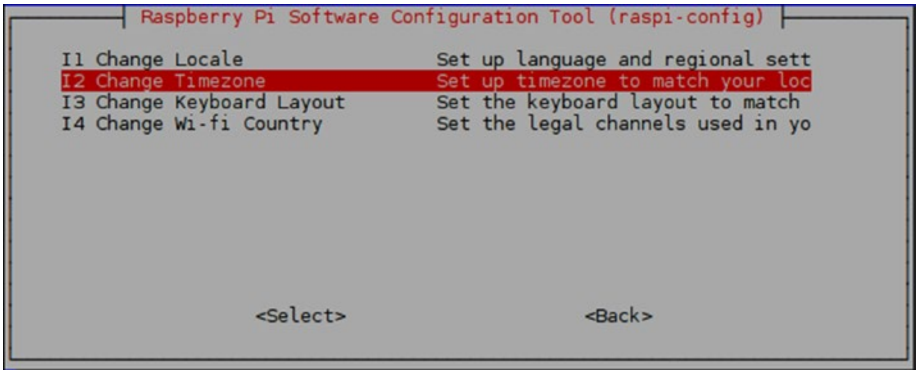
Set the Boot Options to **Desktop Autologin** as shown in Figure 1-18 below.



```
Raspberry Pi Software Configuration Tool (raspi-config)

B1 Console           Text console, requiring user to login
B2 Console Autologin Text console, automatically logged in as 'pi' user
B3 Desktop           Desktop GUI, requiring user to login
B4 Desktop Autologin Desktop GUI, automatically logged in as 'pi' user

              <Ok>                    <Cancel>
```

Figure 1-18. *Desktop Autologin highlighted*

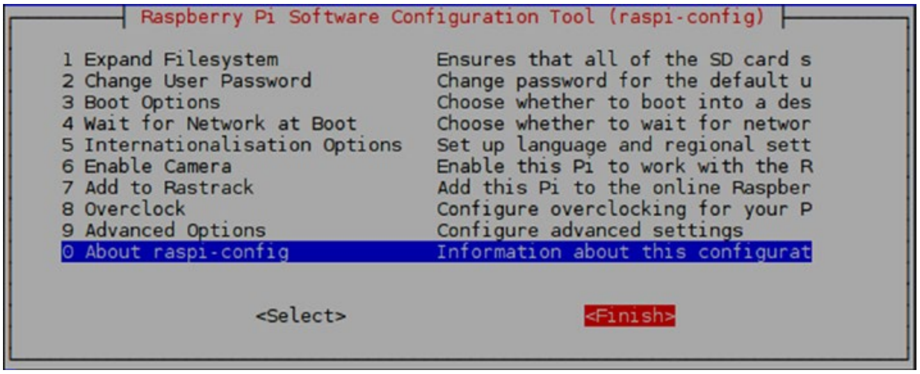
In the **Internationalization Options**, change the timezone and the Wi-Fi country (see Figure 1-19). Change the Keyboard Layout to US.



```
Raspberry Pi Software Configuration Tool (raspi-config)

I1 Change Locale              Set up language and regional sett
I2 Change Timezone            Set up timezone to match your loc
I3 Change Keyboard Layout     Set the keyboard layout to match
I4 Change Wi-fi Country       Set the legal channels used in yo

              <Select>                 <Back>
```

Figure 1-19. *raspi-config internationalization options*

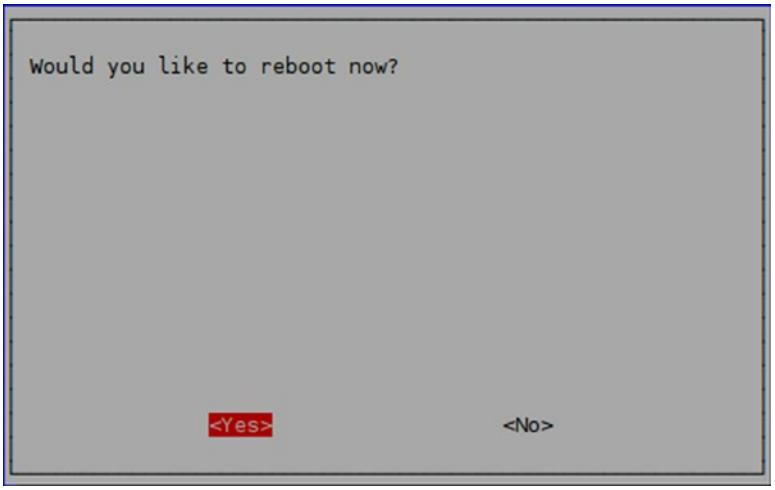
Once done, go back to the main screen and select **Finish** as shown in the following screenshot (Figure 1-20).



```
Raspberry Pi Software Configuration Tool (raspi-config)

1 Expand Filesystem              Ensures that all of the SD card s
2 Change User Password           Change password for the default u
3 Boot Options                   Choose whether to boot into a des
4 Wait for Network at Boot       Choose whether to wait for networ
5 Internationalisation Options   Set up language and regional sett
6 Enable Camera                  Enable this Pi to work with the R
7 Add to Rastrack                Add this Pi to the online Raspber
8 Overclock                      Configure overclocking for your P
9 Advanced Options               Configure advanced settings
0 About raspi-config             Information about this configurat

              <Select>                    <Finish>
```

Figure 1-20. Finish

It will ask to reboot (Figure 1-21). Choose **Yes**.



```
Would you like to reboot now?

          <Yes>                <No>
```

Figure 1-21. Reboot Prompt

It will reboot the Pi.

Our job is not done yet. We need to learn how to connect the Pi to the Internet and how to update it.

Raspbian

An operating system is the set of basic programs and utilities to make the computer work. It is an interface between the user and the computer. **Raspbian** is a free operating system based on the popular Linux distribution **Debian**. Raspbian is optimized for the Raspberry Pi family of SBCs. It is even ported to the other similar SBCs like **Banana Pro**.

Raspbian has more than 35,000 packages and pre-compiled software bundled in for easy installation and use on the Raspberry Pi. The first build of Raspbian was completed in June of 2012. Raspbian is still under active development and updated frequently. Visit the Raspbian homepage (`www.raspbian.org`) and the Raspbian documentation page (`www.raspbian.org/RaspbianDocumentation`) for more information on Raspbian.

config.txt

Raspberry Pi does not have a conventional **BIOS**. A BIOS (Basic Input/Output System) is the program that a computer's microprocessor uses to get the computer system started and load the OS into the memory after the computer is powered on. It also manages data flow between the computer's operating system and attached peripheral devices such as the hard disk, video adapter, keyboard, mouse, and printer.

As Raspberry Pi does not have a BIOS, the various system configuration parameters that are normally stored and modified using the BIOS are now stored in a text file named `config.txt`.

The Raspberry Pi `config.txt` file is a file on the `boot` partition of the Raspberry Pi. It is normally accessible as `/boot/config.txt` from Linux. However, from Windows and Mac OS, it is seen as a file in the accessible part of the microSD card. The accessible part of the card is labeled as `boot`. As we have already learned in earlier in this chapter, we must edit the `/boot/config.txt` file if we want to connect it to a VGA display.

On Raspberry Pi, we can edit this file with the following command in LXTerminal:

```
sudo nano /boot/config.txt
```

■ **Note** nano is a simple and easy to learn text-based text editor for Linux. Visit its homepage (`www.nano-editor.org`) to learn more about it. I find it easier to use than `vi` or `vim` editors.

To learn more about `config.txt`, visit the page `http://elinux.org/RPiconfig`. Also, a sample configuration can be found at `http://elinux.org/R-Pi_configuration_file`.

Connecting Raspberry Pi to a Network and the Internet

To connect the Pi to any network, we have to edit the `/etc/network/interfaces` file. If the network that Pi is connected to is connected to the Internet, then the Pi can access the Internet.

WiFi

Raspberry Pi 3 Model B has built-in WiFi. For all the other models of Pi, we need to use a USB WiFi adapter.

Once the USB WiFi adapter is attached to the Pi, make a backup of the `/etc/network/interfaces` file with the following command:

```
sudo mv /etc/network/interfaces /etc/network/interfaces.bkp
```

The original `/etc/network/interfaces` file is safe this way, and can be restored if something goes wrong.

Now create a new `/etc/network/interfaces` file.

```
sudo nano /etc/network/interfaces
```

Type the following lines (Listing 1-1) into that:

Listing 1-1. /etc/network/interfaces

```
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

auto wlan0
allow-hotplug wlan0
iface wlan0 inet dhcp
wpa-ssid "ASHWIN"
wpa-psk "internet"
```

In the file above (Listing 1-1) replace `"ASHWIN"` with the **ssid** of your WiFi network and replace `"internet"` with the password of your WiFi network. Save the file by pressing **CTRL+X** and then **Y**.

Run the following command to restart the networking service:

```
sudo service networking restart
```

If you have followed the steps correctly, the Pi should connect to the WiFi network and the Internet (provided that WiFi network is connected to the Internet).

To verify connectivity with the Internet, use the following command:

```
ping -c4 www.google.com
```

It should show output similar to that below.

```
PING www.google.com (216.58.197.68) 56(84) bytes of data.
64 bytes from maa03s21-in-f4.1e100.net (216.58.197.68): icmp_seq=1 ttl=55
time=755 ms
```

```
64 bytes from maa03s21-in-f4.1e100.net (216.58.197.68): icmp_seq=2 ttl=55
time=394 ms
64 bytes from maa03s21-in-f4.1e100.net (216.58.197.68): icmp_seq=3 ttl=55
time=391 ms
64 bytes from maa03s21-in-f4.1e100.net (216.58.197.68): icmp_seq=4 ttl=55
time=401 ms

--- www.google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 391.729/485.695/755.701/155.925 ms
```

Output like the above means the Pi is connected to the Internet.

To find the IP address of the Pi, use the ifconfig command. In its output, check the section for `wlan0`. It will be as follows:

```
wlan0     Link encap:Ethernet  HWaddr 7c:dd:90:00:e2:1e
          inet addr:192.168.0.122  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::7edd:90ff:fe00:e21e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1974 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1275 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:195049 (190.4 KiB)  TX bytes:1204336 (1.1 MiB)
```

In the output above, `192.168.0.122` is the IP address of the Pi. As the IP address is allocated with **DHCP** protocol, it will be different for you depending on the WiFi network settings.

Ethernet

We can also connect the Pi to a LAN network. Based on the LAN switch's settings, we can allocate an IP address to the Pi statically or dynamically.

Static IP address

If the LAN network allocates IP addresses statically, then configure the `/etc/network/interfaces` as follows (Listing 1-2):

Listing 1-2. /etc/network/interfaces

```
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

auto eth0
allow-hotplug eth0
```

```
iface eth0 inet static
# Your static IP
address 192.168.0.2
# Your gateway IP
gateway 192.168.0.1
netmask 255.255.255.0
# Your network address family
network 192.168.0.0
broadcast 192.168.0.255
```

In the file above, the parameters `address`, `gateway`, `netmask`, `network`, and `broadcast` are based on the LAN's configuration. Please check the manual of the LAN switch or router. If you are working in an organization, then check with the network administrator for these parameters.

Dynamic IP address

This is an easy one. If the LAN has DHCP capability, then configure the `/etc/network/interfaces` as follows (Listing 1-3):

Listing 1-3. /etc/network/interfaces

```
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

auto eth0
allow-hotplug eth0
iface eth0 inet dhcp
```

This will configure the Pi to acquire the IP address automatically with DHCP.

■ **Note** All the information needed for network setup on Debian and its derivatives can be found on `https://wiki.debian.org/NetworkConfiguration`.

Updating the Pi

Pi must be connected to the internet for this.

Updating the Firmware

The firmware is the software embedded into the ROM chip of an electronic device. It provides control and monitoring of the device. To update the firmware of the Pi, run `sudo rpi-update`. It will update the firmware.

Updating and Upgrading Raspbian

We will use APT for this. APT stands for Advanced Package Tool. It is a program that handles the installation and removal of software on Debian and other Debian derivatives. APT simplifies the process of managing software on Debian systems by automating the fetching, configuration, and installation of software packages. We will need an Internet connection for this too.

First, update the system's package list by entering the following command in LXTerminal:

```
sudo apt-get update
```

`apt-get update` downloads the package lists from the respective remote repositories and updates them on the local computer to get information on the newest versions of packages and their dependencies which are available for installation and update. It should be run before running the `install` or `upgrade` command.

Next, upgrade all the installed packages to their latest versions with the command:

```
sudo apt-get dist-upgrade -y
```

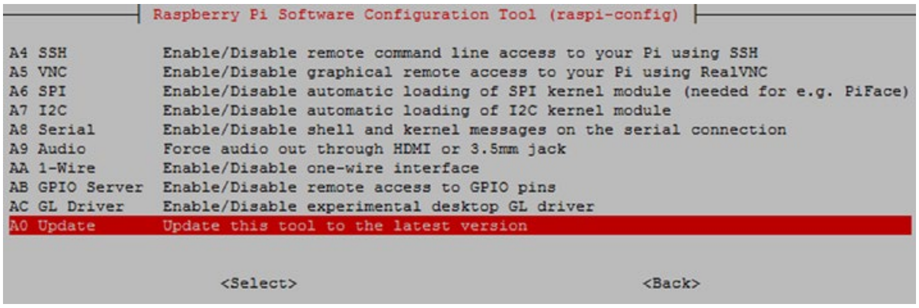
`apt-get dist-upgrade` fetches new versions of packages existing on the machine which are marked for upgrade on the local machine. It also detects and installs dependencies. It might remove obsolete packages.

Doing this regularly will keep the RaspbianOS installed on the Pi up to date. After entering these commands, it will take a while to update the OS as these commands fetch data and packages from the remote repositories.

■ **Note** `sudo apt-get --help` will list all the options associated with `apt-get`

Updating raspi-config

In `raspi-config`, go to the advanced options (see Figure 1-22) and choose **Update**.



```
├ Raspberry Pi Software Configuration Tool (raspi-config) ├

A4 SSH          Enable/Disable remote command line access to your Pi using SSH
A5 VNC          Enable/Disable graphical remote access to your Pi using RealVNC
A6 SPI          Enable/Disable automatic loading of SPI kernel module (needed for e.g. PiFace)
A7 I2C          Enable/Disable automatic loading of I2C kernel module
A8 Serial       Enable/Disable shell and kernel messages on the serial connection
A9 Audio        Force audio out through HDMI or 3.5mm jack
AA 1-Wire       Enable/Disable one-wire interface
AB GPIO Server  Enable/Disable remote access to GPIO pins
AC GL Driver    Enable/Disable experimental desktop GL driver
A0 Update       Update this tool to the latest version

                <Select>                              <Back>
```

Figure 1-22. *Updating raspi-config*

Shutting Down and Restarting Pi

We can shutdown Pi safely with `sudo shutdown -h now`.

We can restart Pi with `sudo reboot -h now`.

Conclusion

In this chapter, we were introduced to the concept and philosophy of SBCs. We also got started with a popular family of SBCs, Raspberry Pi. Now we can confidently move ahead with further exploration. In the next chapter, we will learn a few important Linux commands and how to connect to the Pi remotely.

CHAPTER 2

Important Linux Commands and Remote Connectivity

In the last chapter, we studied the basics of single board computers and how to set up your Pi. We also learned to connect the Pi to the Internet. I hope all readers are quite comfortable with the basics now. With this accomplished, we will dig a bit deeper into some more basics. In this chapter, we will be studying a few important Linux commands which will be useful to us. We will also study how to connect to the Pi remotely.

Important and Useful Linux Commands

In this section, we will study a few important Linux commands which will be useful to all of us in understanding the hardware environment (the Pi) which will be used for supercomputing and parallel programming.

Getting Help with Linux Commands

We can use the ***man*** command or --***help*** option to get more information on a command. For example, if we want to know more about the ***cat*** command usage, then we can issue the commands ***man cat*** or ***cat*** --***help***.

Network-related Commands

The following network-related commands are handy in understanding network infrastructure.

ifconfig

ifconfig is used to check network status. We can use ***ifconfig eth0*** or ***ifconfig wlan0*** to check the status of WiFi or ethernet respectively.

iwconfig

We can use ***iwconfig*** to check wireless network status. Its output is as follows:

```
wlan0 IEEE 802.11bg ESSID:"ASHWIN"
Mode:Managed Frequency:2.412 GHz Access Point: A2:EC:80:FB:E2:66
Bit Rate=6 Mb/s Tx-Power=20 dBm
Retry short limit:7 RTS thr:off Fragment thr:off
Power Management:off
Link Quality=45/70 Signal level=-65 dBm
Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
Tx excessive retries:24 Invalid misc:6 Missed beacon:0

lo no wireless extensions.

eth0 no wireless extensions.
```

iwlist wlan0 scan

iwlist wlan0 scan displays a list of all the available wireless networks.

ping

ping tests network connectivity between two devices. We have already seen its usage for checking Internet connectivity in the first chapter.

System Information Commands

These commands help us know more about the status of systems and hardware on the Pi.

CPU-related Information

We can use ***cat /proc/cpuinfo*** to see information about the CPU.

To check the other details (namely the operating speed of the CPU), the ***lscpu*** command is used. The output of ***lscpu*** is as follows:

```
Architecture: armv7l
Byte Order: Little Endian
CPU(s): 4
On-line CPU(s) list: 0-3
Thread(s) per core: 1
Core(s) per socket: 4
Socket(s): 1
Model name: ARMv7 Processor rev 5 (v7l)
CPU max MHz: 900.0000
CPU min MHz: 600.0000
```

Memory-related Information

We can use cat ***/proc/meminfo*** to get details about memory. We can also use the ***free*** command to see how much free memory is available as follows:

```
        total   used    free    shared  buffers  cached
Mem:    996476  210612  785864  7208    15152    113668
-/+ buffers/
cache:  81792   914684
Swap:   102396  0       102396
```

System and OS Version Information

uname -a provides information about the current system as follows:

```
Linux raspberrypi 4.4.11-v7+ #888 SMP Mon May 23 20:10:33 BST 2016 armv7l
GNU/Linux
```

To identify the Linux release, run the command ***cat /proc/version***.

Partition-related Information

df -h displays microSD card partition related information in human-readable format as follows:

```
Filesystem              Size    Used    Avail   Use%    Mounted on
/dev/root               15G     3.6G    11G     26%     /
devtmpfs                483M    0       483M    0%      /dev
tmpfs                   487M    0       487M    0%      /dev/shm
tmpfs                   487M    6.6M    480M    2%      /run
tmpfs                   5.0M    4.0K    5.0M    1%      /run/lock
tmpfs                   487M    0       487M    0%      /sys/fs/cgroup
/dev/mmcblk0p1          63M     21M     43M     33%     /boot
tmpfs                   98M     0       98M     0%      /run/user/1000
```

cat /proc/partitions provides partition block allocation information.

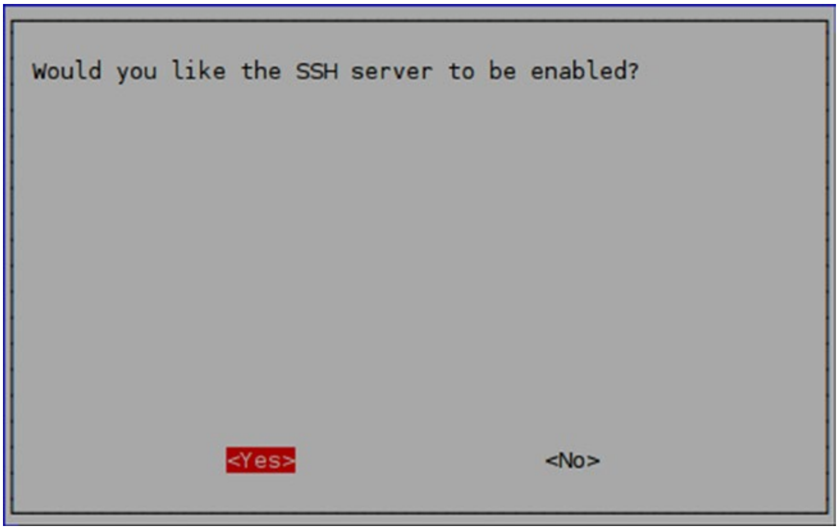
Other Useful Commands

hostname -I shows the IP address.
lsusb shows the list of all the USB devices connected to Pi.
vcgencmd measure_temp shows the temperature of the CPU.
vcgencmd get_mem arm && vcgencmd get_mem gpu shows the memory split between the CPU and GPU.

Enabling Pi for SSH from raspi-config

To connect to Pi remotely, we need to enable **SSH** server from ***raspi-config***. Open **LXTerminal** and run the command ***sudo raspi-config***.

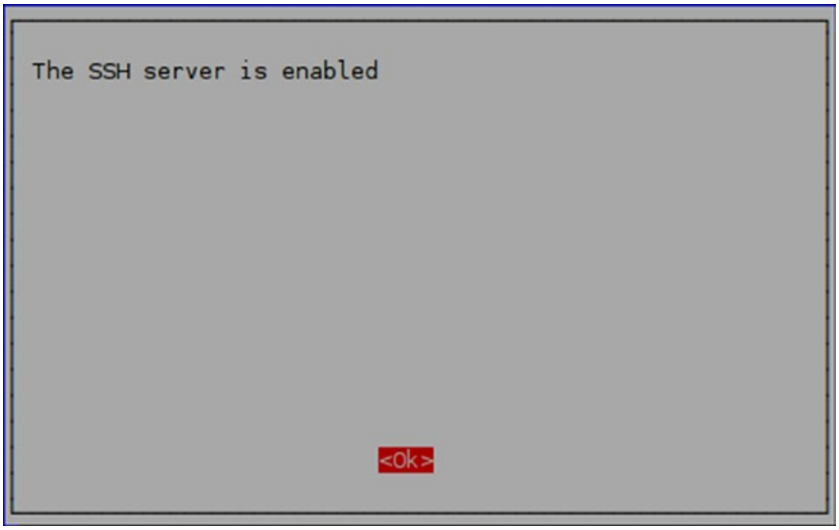
In the main menu of ***raspi-config*** select **Advanced Options**. In the **Advanced Options** screen, choose **A4 SSH**, and the following screen (Figure 2-1) will appear.



```
Would you like the SSH server to be enabled?
```

Figure 2-1. *Enabling the SSH server*

Select **Yes** and the following message (Figure 2-2) will be displayed.



```
The SSH server is enabled
```

Figure 2-2. *SSH server enable confirmation*

Press the **Enter** key. Select **Finish** from the main menu of ***raspi-config*** and select **Yes** when prompted for reboot. Once the Pi reboots, we can access it remotely.

Connecting to the Raspberry Pi Remotely from Windows

It is possible to connect to the Pi remotely, provided that the computer we are using to connect to the Raspberry Pi is also in the same network (either physically or through **VPN**). We have to use various tools and utilities for that. In this chapter, we will learn to connect to the Pi remotely. We will also learn how to transfer files to and from the Pi. This is important when we want to use the Pi in headless mode ("headless" simply means we are using the Pi with no visual display attached to it). It is useful in scenarios where we do not have a display or simply do not want to spare one due to resource/space constraints. For example, when we make a cluster of Pis, we cannot spare a display for each Pi in the cluster. Headless mode is very useful in such cases. In this part of the chapter, we will explore multiple methods for remotely working with Pi.

Checking the Connectivity with Pi from Another Computer

It is essential that the computer which we are planning to connect to Pi must be in the same network. There should not be any proxy and firewall restrictions between the other computer and the Pi. The best example of this setup is the Pi and the other computer connected under the same router/network switch. To check the connectivity, we can use the ***ping*** utility.

1. Open ***cmd***, the Windows command line, on your computer.
2. Turn on the Pi and write down its IP address. Use ***ifconfig*** command to find its IP address. Suppose it is ***192.168.0.2*** (It does not matter whether it is ethernet or WiFi).
3. In ***cmd***, run ***ping 192.168.0.2*** to check the connectivity.

We can use the same ***ping*** command on any computer with a Linux distribution or macOS if they are in the same network as Pi.

PuTTY

PuTTY is a free implementation of **SSH** and Telnet for Windows and Unix platforms, along with an ***xterm*** terminal emulator. It is written and maintained primarily by **Simon Tatham**. You can explore the **PuTTY** homepage (`www.chiark.greenend.org.uk/~sgtatham/putty/`) for more information.

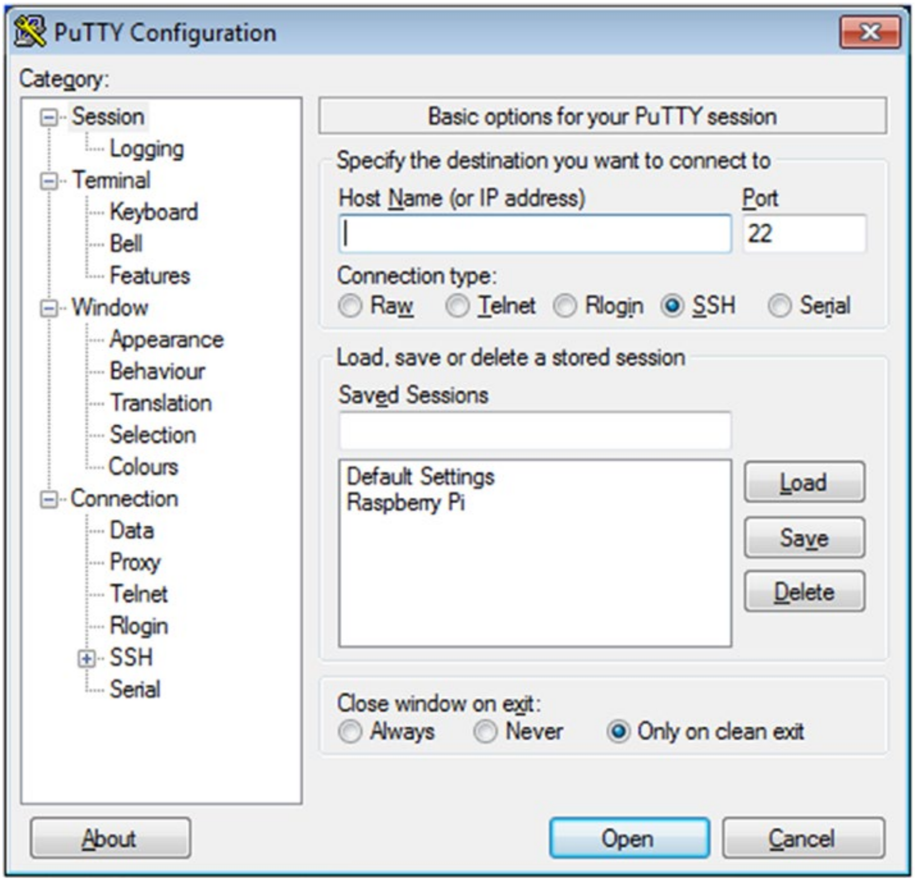
PuTTY is open-source software that is available with source code and is developed and supported by a team of volunteers.

Now let's download and install **PuTTY. PuTTY** can be downloaded from its download page (`www.chiark.greenend.org.uk/~sgtatham/putty/latest.html`).

Download the file ***PuTTY.exe***. Once downloaded, place it in a directory of your choice and create a desktop shortcut (Figure 2-3) for your convenience.

Figure 2-3. *PuTTY desktop shortcut*

Double-click the putty shortcut and the **PuTTY** (Figure 2-4) window will open.



PuTTY Configuration

Category:

- Session
  - Logging
- Terminal
  - Keyboard
  - Bell
  - Features
- Window
  - Appearance
  - Behaviour
  - Translation
  - Selection
  - Colours
- Connection
  - Data
  - Proxy
  - Telnet
  - Rlogin
  - SSH
  - Serial

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address):
Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

Saved Sessions

Default Settings
Raspberry Pi

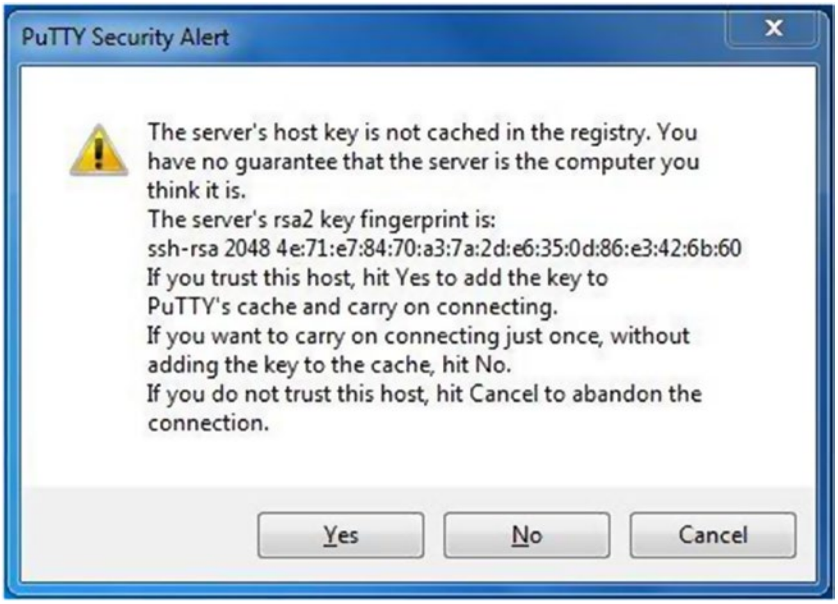
Load Save Delete

Close window on exit: Always Never [✓ Only on clean exit]

About Open Cancel

Figure 2-4. *PuTTY Window*

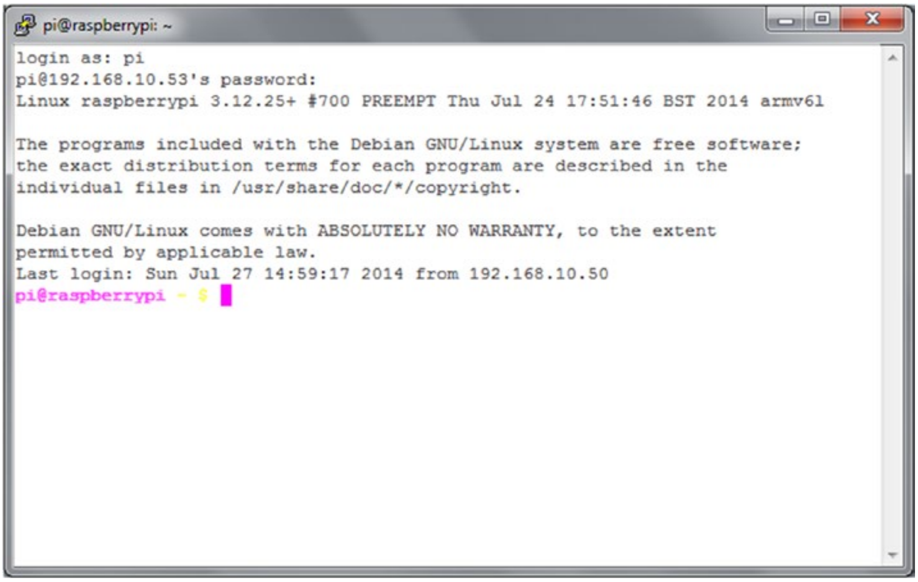
Type the IP address or the host name in the **Host Name (or IP address)** text box. Make sure **SSH** is selected as connection type. For future use, you may want to save the settings (I usually do this). Now, click the **Open** button. It opens a terminal-style window. It will ask you for the username and the password, which are ***pi*** and ***raspberry*** respectively. While logging in for the first time, it shows the following (Figure 2-5) message dialogue. Click **Yes**.



The server's host key is not cached in the registry. You have no guarantee that the server is the computer you think it is.
The server's rsa2 key fingerprint is:
ssh-rsa 2048 4e:71:e7:84:70:a3:7a:2d:e6:35:0d:86:e3:42:6b:60
If you trust this host, hit Yes to add the key to PuTTY's cache and carry on connecting.
If you want to carry on connecting just once, without adding the key to the cache, hit No.
If you do not trust this host, hit Cancel to abandon the connection.

Figure 2-5. *PuTTY security alert*

Once we login, it will show a prompt (Figure 2-6) as follows.



```
login as: pi
pi@192.168.10.53's password:
Linux raspberrypi 3.12.25+ #700 PREEMPT Thu Jul 24 17:51:46 BST 2014 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Jul 27 14:59:17 2014 from 192.168.10.50
pi@raspberrypi ~ $
```

Figure 2-6. *PuTTY remote connection window*

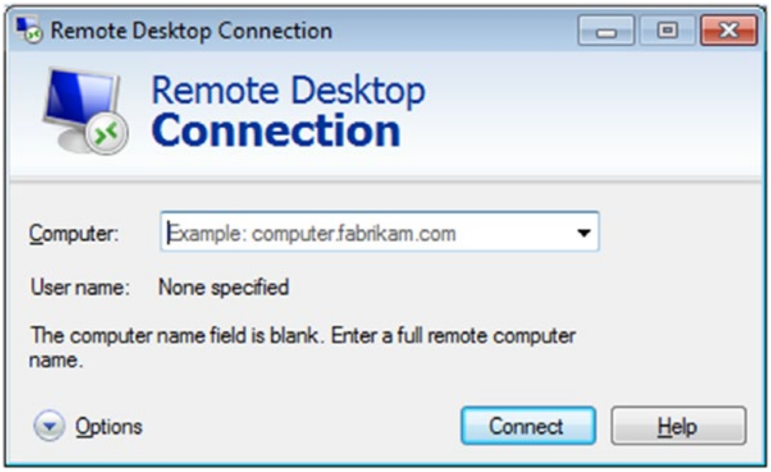
We can now remotely work with the Pi's command prompt.

Accessing Raspberry Pi Desktop Remotely

Raspberry Pi desktop is **LXDE (Lightweight X11 Desktop Environment)**. We can access it from Windows computer remotely using an **RDP (Remote Desktop Protocol)** client. For that, we need to install xrdp on Pi. Install it by running sudo apt-get install xrdp. Reboot the Pi after the installation.

We have to use **Windows Remote Desktop Client** to connect to the Pi now. The client can be found with the Search option in Windows. Click the **Remote Desktop Client** icon to open it.

In the dialog box below (Figure 2-7), click **Options** to expand it.



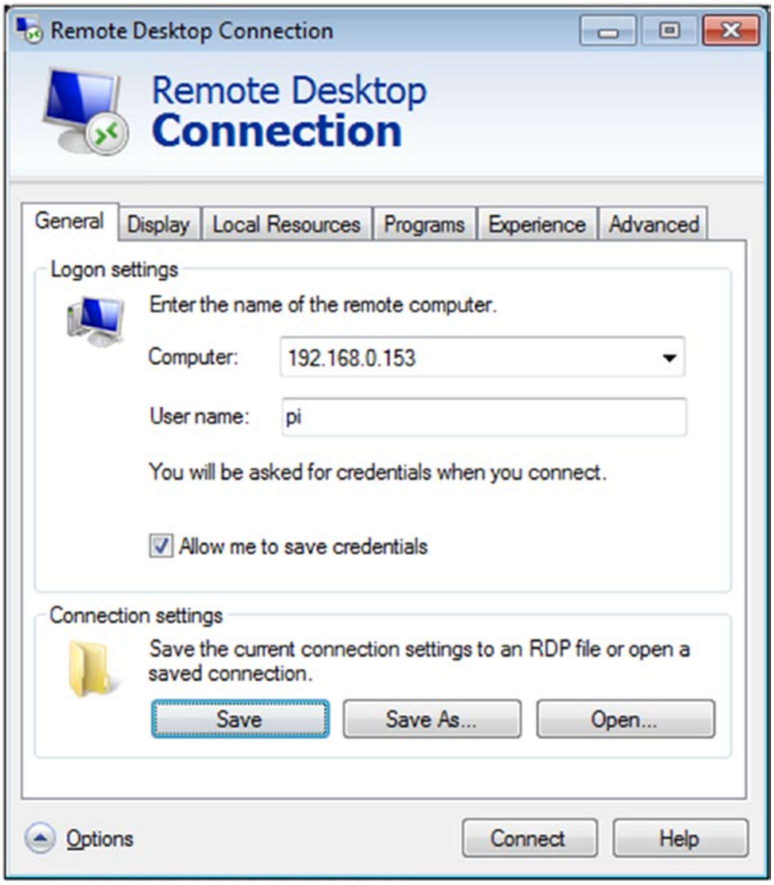
Computer: Example: computer.fabrikam.com

User name: None specified

The computer name field is blank. Enter a full remote computer name.

Figure 2-7. *Remote Desktop Connection*

It expands and shows various options (Figure 2-8). Enter the IP address of the Pi and ***pi*** as user name. Click the checkbox and the **Save** button to save the same configuration for future use.



Enter the name of the remote computer.

Computer: 192.168.0.153

User name: pi

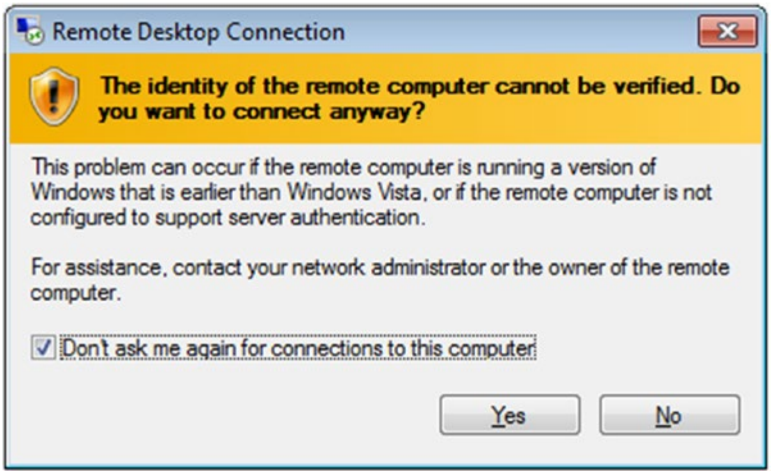
You will be asked for credentials when you connect.

[✓] Allow me to save credentials

Save the current connection settings to an RDP file or open a saved connection.

Figure 2-8. *Remote Desktop Connection options*

When logging in, it will prompt for the password. Enter the password. If we are connecting to the Pi with a Windows computer for the very first time, then it will display the following dialog box (Figure 2-9). Select the checkbox so it will not ask us again and then click **Yes**.



The identity of the remote computer cannot be verified. Do you want to connect anyway?

This problem can occur if the remote computer is running a version of Windows that is earlier than Windows Vista, or if the remote computer is not configured to support server authentication.

For assistance, contact your network administrator or the owner of the remote computer.

[✓] Don't ask me again for connections to this computer

Figure 2-9. *First time remote login*

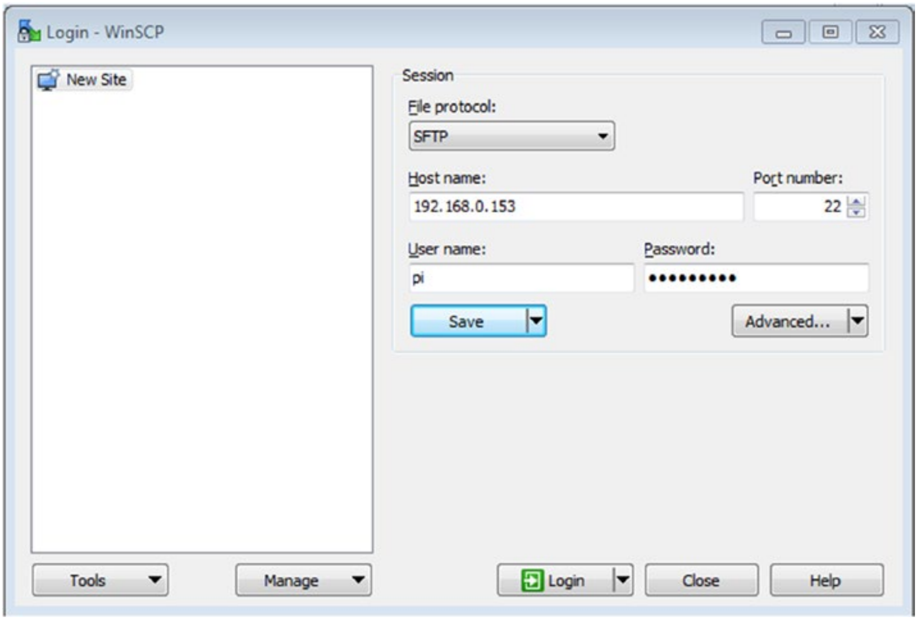
It will display the Raspberry Pi desktop. Speed of operations will be a bit slow as the Pi is streaming the desktop over a network. However, it works pretty well.

This is how we can access the Pi's desktop with a Windows computer/laptop. However, none of the methods we have learned allows us to transfer a file between a Windows computer and the Pi. We will learn how to do it in the next section of this chapter.

WinSCP

For file transfers between a Windows computer and a Raspberry Pi, we will use **WinSCP** (`https://winscp.net/eng/index.php`). **WinSCP** is a free open-source **SFTP** and **FTP** client for Windows. Its main function is the secure file transfer between local and remote computers.

Download its setup (`https://winscp.net/eng/download.php`) and install it. Create its shortcut on the Windows desktop. Open **WinSCP** by double-clicking the icon (Figure 2-10).


Login - WinSCP

New Site

Session

File protocol: SFTP

Host name: 192.168.0.153

Port number: 22

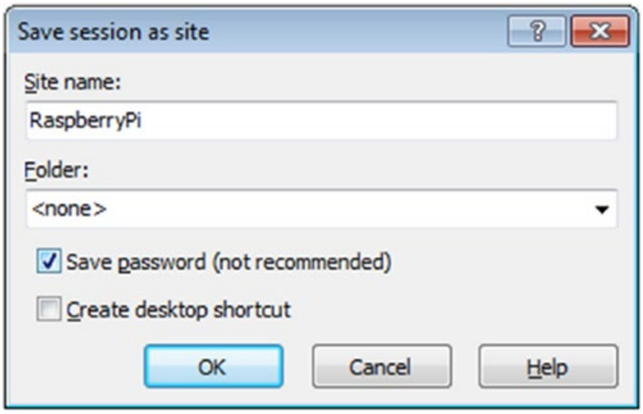
User name: pi

Password: •••••••••

Figure 2-10. *WinSCP window*

Enter the IP address of the Pi in the **host name** text box. Also, enter ***pi*** as the username and ***raspberry*** as the password.

We can also save the settings for future use. The save session (Figure 2-11) dialog box is as follows.


Save session as site

Site name: RaspberryPi

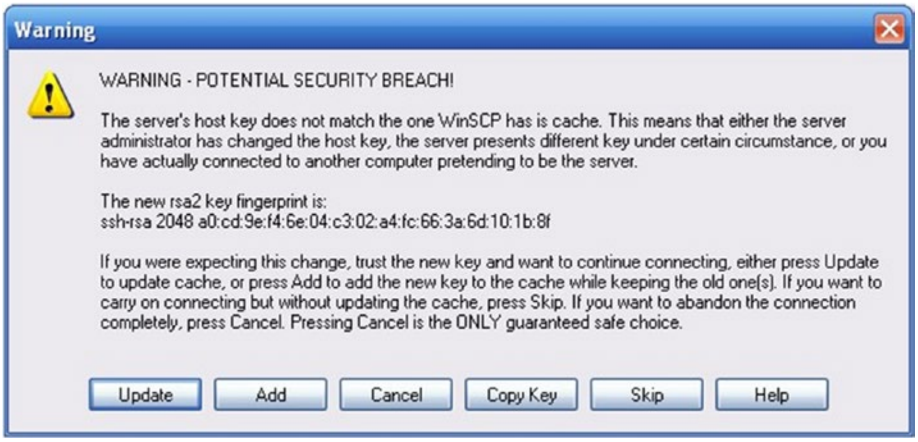
Folder: <none>

[✓] Save password (not recommended)

[ ] Create desktop shortcut

Figure 2-11. *Save session*

Once we login, if we are connecting for the first time then the following dialog box (Figure 2-12) is displayed. Click the **Add** button to continue.



WARNING - POTENTIAL SECURITY BREACH!

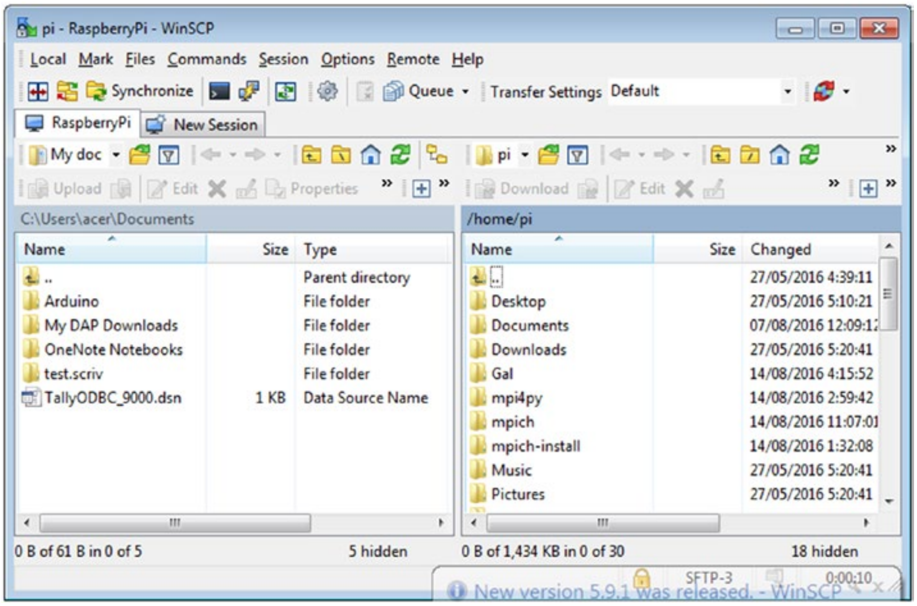
The server's host key does not match the one WinSCP has is cache. This means that either the server administrator has changed the host key, the server presents different key under certain circumstance, or you have actually connected to another computer pretending to be the server.

The new rsa2 key fingerprint is:
ssh-rsa 2048 a0:cd:9e:f4:6e:04:c3:02:a4:fc:66:3a:6d:10:1b:8f

If you were expecting this change, trust the new key and want to continue connecting, either press Update to update cache, or press Add to add the new key to the cache while keeping the old one(s). If you want to carry on connecting but without updating the cache, press Skip. If you want to abandon the connection completely, press Cancel. Pressing Cancel is the ONLY guaranteed safe choice.

Figure 2-12. *First time login dialog box*

Once we login, the following window (Figure 2-13) is displayed.


pi - RaspberryPi - WinSCP
Local Mark Files Commands Session Options Remote Help
Synchronize
Queue
Transfer Settings Default
RaspberryPi
New Session
My doc
pi
Upload
Edit
Properties
Download
Edit

C:\Users\acer\Documents

Name	Size	Type
..		Parent directory
Arduino		File folder
My DAP Downloads		File folder
OneNote Notebooks		File folder
test.scriv		File folder
TallyODBC_9000.dsn	1 KB	Data Source Name

/home/pi

Name	Size	Changed
..		27/05/2016 4:39:11
Desktop		27/05/2016 5:10:21
Documents		07/08/2016 12:09:12
Downloads		27/05/2016 5:20:41
Gal		14/08/2016 4:15:52
mpi4py		14/08/2016 2:59:42
mpich		14/08/2016 11:07:01
mpich-install		14/08/2016 1:32:08
Music		27/05/2016 5:20:41
Pictures		27/05/2016 5:20:41

0 B of 61 B in 0 of 5 5 hidden
0 B of 1,434 KB in 0 of 30 18 hidden
SFTP-3 0:00:10
New version 5.9.1 was released. - WinSCP

Figure 2-13. *WinSCP file transfer window*

The file system of the local Windows computer is in the left panel and Raspberry Pi's ***pi*** user's home directory i.e. ***/home/pi*** is in the right panel. We can transfer files between both computers now.

Connecting to Raspberry Pi Using Linux or macOS

Let's learn to connect to the Pi using a Linux computer or macOS.

Remote Login with SSH

SSH is built into Linux distributions and macOS. We can use **SSH** to connect to the Pi from a Linux computer (which could also be another Raspberry Pi) or from the Mac terminal, without installing additional software.

Open the terminal in your Linux computer or Mac and type the following command:

```
ssh pi@192.168.0.2
```

192.168.0.2 is my Pi's IP address. Replace it with the IP address of your Pi. Once we press **Enter**, it will show a security/authenticity warning prompt. Type **yes** to continue. This warning is shown only when we connect for the first time.

Now it will prompt for the password. Enter ***pi*** user's default password ***raspberry***. We will see the Raspberry Pi prompt which will be identical to the one found on the Raspberry Pi itself.

Forwarding Using SSH

We can also forward our **X11** session over **SSH** to allow use of graphical applications by using the ***-Y*** flag in the ***ssh*** command as follows,

```
ssh -Y pi@192.168.0.2
```

Let's access a graphical program like scratch remotely. Run the following command,

```
scratch &
```

It will start a new **X11** session of the Pi program `scratch` in a new window on a Linux computer or Mac. The & makes the command run in the background.

SCP for File Transfer

In Windows, we used WinSCP for the file transfer between the Windows computer and the Pi. In the same way, we can transfer files between a Linux computer/Mac and the Pi. We need to use the ***scp*** utility for that. It is built into all Linux distributions and macOS.

To copy a file from the Pi to a Linux computer or Mac, we have to run the following command in the terminal on the Linux computer or the Mac:

```
scp pi@192.168.0.2:/home/pi/test.txt /home/ashwin
```

The command above copies ***test.txt*** from ***/home/pi*** directory of Pi to ***/home/ashwin*** directory of our Linux computer or Mac.

In the same way, we might want to copy a file from a Linux computer or Mac to the Pi. To do that, run the following command in the terminal of the Linux computer or the Mac:

```
scp /home/ashwin/test_again.txt pi@192.168.0.2:/home/pi
```

You can read more about the ***scp*** command in detail on www.computerhope.com/unix/scp.htm.

EXERCISE

Complete the following exercise to understand this chapter better.

- Practice the ***man*** command.
- Practice the ***--help*** option for various Linux commands.
- Run the ***iwlist wlan0 scan*** command.
- Run the ***ping*** ***www.AshwinPajankar.com*** command to check the connectivity to the Internet.
- Try to see the output of all the system-related commands demonstrated in this chapter.
- Try to access Raspberry Pi desktop remotely with VNC.

Conclusion

In this chapter, we explored various ways to connect to the Pi remotely. We also learned how to transfer files between the Pi and other computers with Linux, Windows, and Mac as OS.

CHAPTER 3

Introduction to Python

In the last chapter, we learned important Linux commands and how to connect to Raspberry Pi from other computers remotely. We also learned how to transfer files to and from Raspberry Pi.

In this chapter, we will get started with Python.

Let's begin this chapter with an introduction to Python. I personally find Python amazing and have been enchanted by it. Python is a simple yet powerful programming language. When we use Python, it's easy to focus on the implementation of the solution to a given problem, since programmers do not have to worry about the syntax of the programming language. Python perfectly fits the philosophy of Raspberry Pi, which is "programming for everyone." That's why Python is the preferred programming platform for Raspberry Pi and many other SBCs.

History of Python

Python was designed and conceived in the late 1980s. Its actual implementation was started in late 1989 by Guido van Rossum in **Centrum Wiskunde & Informatica (National Research Institute for Mathematics and Computer Science)** in the Netherlands. Python is a successor to the **ABC Programming Language**. The ABC Programming Language itself was inspired by **SETL**. In February 1991, Van Rossum publicly published the Python source code to the ***alt.sources*** newsgroup. The name Python was inspired by the television show **Monty Python's Flying Circus**. Van Rossum is a big fan of **Monty Python**.

Van Rossum is the principal author of the Python programming language. He plays a central role in guiding the direction of the development, bug-fixing, enhancement, and further evolution of the Python programming language. He holds the title of **Benevolent Dictator for Life** for Python. He currently (as of February 2017) works for Dropbox and dedicates almost half of his time towards further development of the Python programming language.

The central philosophy of Python Programming language, the **Zen of Python**, is explained in PEP-20 (PEP stands for Python Enhancement Proposal) which can be found at `www.python.org/dev/peps/pep-0020`.

It is a collection of 20 software principles, out of which 19 have been documented. The principles are as follows:

1. Beautiful is better than ugly.
2. Explicit is better than implicit.
3. Simple is better than complex.
4. Complex is better than complicated.
5. Flat is better than nested.
6. Sparse is better than dense.
7. Readability counts.
8. Special cases aren't special enough to break the rules.
9. Although practicality beats purity.
10. Errors should never pass silently.
11. Unless explicitly silenced.
12. In the face of ambiguity, refuse the temptation to guess.
13. There should be one—and preferably only one—obvious way to do it.
14. Although that way may not be obvious at first unless you're Dutch.
15. Now is better than never.
16. Although never is often better than right now.
17. If the implementation is hard to explain, it's a bad idea.
18. If the implementation is easy to explain, it may be a good idea.
19. Namespaces are one honking great idea—let's do more of those!

Features of Python

The following are the features of Python for which it has become popular and beloved in the programming community.

Simple

Python is a simple and minimalist language. Reading a good, well-written Python program makes us feel as if we are reading English text.

Easy to Learn

Due to its simple, English-like syntax, Python is extremely easy to learn for beginners. That is the prime reason that nowadays it is taught as the first programming language to high school and university students who take **introduction to programming** and **programming 101** Courses. An entire new generation of programmers is learning Python as their first programming language.

Easy to Read

Unlike other high-level programming languages, Python does not provide much provision for obfuscating the code and making it unreadable. The English-like structure of Python code makes it easier to read compared with the code written in other programming languages. This makes it easier to understand and easier to learn compared with other high-level languages like C and C++.

Easy to Maintain

As Python code is easy to read, easy to understand, and easy to learn, anyone maintaining the code becomes comfortable with the codebase in considerably less time. I can vouch for this from my personal experience of maintaining and enhancing large legacy codebases which were written in a combination of bash and Python 2.

Open Source

Python is an Open Source Project. Its source code is freely available. We can make changes to it to suit our needs, and use both the original and the changed code in our applications.

High-level Language

While writing Python programs, we do not have to manage low-level details like memory management, CPU timings, and scheduling processes. All these tasks are managed by the Python interpreter. We can directly write the code in easy-to-understand English-like syntax.

Portable

The Python interpreter has been ported on many OS platforms. Python code is also portable. All Python programs can work on any of the supported platforms without requiring many changes, if we are careful enough to avoid any system-dependent coding.

We can use Python on GNU/Linux, Windows, Android, FreeBSD, Mac OS, iOS, Solaris, OS/2, Amiga, Palm OS, QNX, VMS, AROS, AS/400, BeOS, OS/390, z/OS, Psion, Acorn, PlayStation, Sharp Zaurus, RISC OS, VxWorks, Windows CE, and PocketPC.

Interpreted

Python is an interpreted language. Let's understand what that means. Programs written in high-level programming languages like C, C++, and Java are first compiled. This means that they are first converted into an intermediate format. When we run the program, this intermediate format is loaded from secondary storage (i.e. hard disk) to the memory (RAM) by the linker/loader. So C, C++, and Java have a separate compiler and linker/loader. This is not the case with Python. Python runs the program directly from the source code. We do not have to bother with compiling and linking to the proper libraries. This makes Python programs truly portable as we can copy the program from one computer to another and the program runs just fine, as long as the necessary libraries are installed on the target computer.

Object-Oriented

Python supports procedure-oriented programming as well as object-oriented programming paradigms.

Python supports object-oriented programming paradigms. All object-oriented programming paradigms are implemented in Python. In object-oriented programming languages, the program is built around objects which combine data and related functionality. Python is a very simple but powerful object-oriented programming language.

Extensible

One of the features of Python is that we can call C and C++ routines from Python programs. If we want the core functionality of the application to run faster, then we can code that part in C/C++ and call it in the Python program (C/C++ programs generally run faster than Python).

Extensive Libraries

Python has an extensive standard library, which comes pre-installed with Python. The standard library has all the essential features for a modern-day programming language. It has provision for databases, unit testing (which we will explore in this book), regular expressions, multi-threading, network programming, computer graphics, image processing, GUI, and other utilities. This is part of Python's "batteries-included" philosophy.

Apart from the standard library, Python has a numerous and ever-growing set of third-party libraries. A list of these libraries can be found at the Python Package Index.

Robust

Python provides robustness by means of ability to handle errors. A full stack trace of encountered errors is available and makes the life of a programmer more bearable. Runtime errors are known as exceptions. The feature which allows handling of these errors is known as the exception handling mechanism.

Rapid Prototyping

Python is used as a Rapid Prototyping Tool. As we have seen earlier, the properties of Python are that it has extensive libraries and is easy to learn, so many software architects are increasingly using it as a tool to rapidly prototype their ideas into working models in a very short time.

Memory Management

In assembly languages and programming languages like C and C++, memory management is the responsibility of the programmer. This is in addition to the task at hand, which creates an unnecessary burden for the programmer. In Python, the Python interpreter takes care of memory management. This helps programmers steer clear of memory issues and focus on the task at hand.

Powerful

Python has everything needed for a modern programming language. It is used for applications like computer vision, supercomputing, drug discovery, scientific computing, simulation, and bioinformatics. Millions of programmers around the world use Python. Many big brands like NASA, Google, SpaceX, and Cisco (I worked there!) use Python for their applications and infrastructure.

Community Support

Personally, I find this the most appealing feature of Python. As we have seen, since Python is open-source as well as having a community of almost a million programmers (probably more, as today high school kids are learning Python too) throughout the world, there are plenty of forums on the Internet to support programmers who encounter any roadblock. None of my queries related to Python has gone unanswered yet.

Python 3

Python 3 was released in 2008. The Python development team had decided to do away with some of the redundant features of the Python language, simplify some other features, rectify some design flaws, and add a few more much-needed features.

It was decided that a major revision number was needed for this and the resulting release **would not be backward-compatible**. Python 2.x and 3.x were supposed to co-exist in parallel for the programmers' community to have enough time to migrate their code and the third-party libraries from 2.x to 3.x. Python 2.x code cannot be run as is, in many cases, as there are significant differences between 2.x and 3.x.

The Differences Between Python 2 and Python 3

The following are a few of the most noticeable differences between Python 2 and Python 3, which are worth understanding. We will be using many features of Python 3 related to these differences. Let's have a look at them in brief:

- The ***print()*** function

  This is the most noticeable difference between Python 2 and Python 3. The ***print*** statement of Python 2 is replaced by the ***print()*** function in Python 3.

- Integer division produces float value

  The nature of integer division has been changed in Python 3 for the sake of mathematical correctness. In Python 2, the result of division of two integers is an integer. However, in Python 3, it is a float value which is mathematically correct and makes more sense to a beginner. In most programming languages, the integer division is an integer.

- Removal of ***xrange()***

  In Python 2, for creating iterable objects, the ***xrange()*** function is used. In Python 3, ***range()*** is implemented like ***xrange()***. Thus a separate ***xrange()*** is not required anymore in Python 3. Using ***xrange()*** in Python 3 raises a ***nameError*** exception.

- Raising exceptions

  It is mandatory in Python 3 to enclose exception arguments, if any, in parentheses, whereas in Python 2 it is optional.

- Handling exceptions

  In Python 3, while handling exceptions, the ***as*** keyword before the parameter to handle arguments is a must. In Python 2, it is not needed.

- New style classes

  Python 2 supports both old style classes and new style classes, whereas Python 3 supports only new style classes. Python 3 does not support old style classes at all. All the classes created in Python 3 are new style classes by default.

- New features of Python 3

  The following new features of Python 3 have not yet been backported to Python 2.

  - Strings are Unicode by default
  - Clean Unicode/byte separation

- Exception chaining
- Function annotations
- Syntax for keyword-only arguments
- Extended tuple unpacking
- Non-local variable declarations

From the list above, we will be extensively using the ***print()*** method, new-style classes, exceptions, and the exception-handling mechanism in the code examples in this book.

■ **Note** See the Python Wiki page for differences between Python 2 and Python 3: `https://wiki.python.org/moin/Python2orPython3`

Why Use Python 3?

From the list above, we will be frequently using new style classes and exceptions in the code examples in this book.

While many Python experts are still advocating for Python 2, I totally disagree with them. The Python Wiki (`https://wiki.python.org/moin/Python2orPython3`) says:

Python 2.x is legacy, Python 3.x is the present and future of the language.

One of the major arguments in favor of Python 2 is the extensive documentation, books, and third-party libraries. However, most developers are porting their custom libraries to Python 3 already. Almost all the major third-party libraries are ported and fully supported for Python 3. As far as books and documentation are concerned, authors like me are extensively writing for Python 3. As time passes, more documentation for Python 3 will surely be available.

A new generation of programmers are being introduced to Python 3 as their first programming language. When they are comfortable with the concept and philosophy of Python programming, they are gradually introduced to Python 2.

Most organizations have already started migrating codebases from Python 2 to Python 3. Almost all new projects in Python are extensively using Python 3.

I personally think that these are pretty good reasons to use Python 3.

Python 2 and Python 3 on Raspbian

Raspbian is a Debian variant. Python 2 and Python 3 interpreters are pre-installed in Raspbian. The Python 2 interpreter can be invoked by running the command ***python*** in **lxterminal**. The Python 3 interpreter can be invoked by running the command ***python3*** in **lxterminal**. We can check the Python 3 interpreter version by running ***python3 -V*** or ***python --version***. We can check the location of the Python 3 binary by running ***which python3*** at **lxterminal**.

Running a Python Program and Python Modes

We have set up our environment for Python programming now. Let's get started with a simple concept of Python. Python has two modes, normal and interactive. Let's look at these modes in detail.

Interactive Mode

Python's interactive mode is a command line shell. It provides immediate output for every executed statement. It also stores the output of previously executed statements in active memory. As the new statements are executed by the Python interpreter, the entire sequence of previously executed statements is considered while evaluating the current output. We have to type ***python3*** in the **lxterminal** to invoke the Python 3 interpreter into interactive mode as follows:

```
pi@raspberrypi:~ $
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

We can execute Python statements directly in this interactive mode just like we run commands from the OS shell/console, as follows:

```
>>>print ('Hello World!')
Hello World!
>>>
```

We will not be using interactive mode in this book. However, it's the quickest way to check small snippets of code (5 to 10 lines). We can quit interactive mode with the ***exit()*** statement as follows:

```
>>> exit()
pi@raspberrypi:~ $
```

Normal Mode

Normal mode is the mode where Python script files (.py) are executed by the Python interpreter.

Create a file with filename ***test.py*** and add the statement ***print ('Hello World!')*** to the file. Save the file and run it with the Python 3 interpreter as follows:

```
pi@raspberrypi:~ $ python3 test.py
HelloWorld!
pi@raspberrypi:~ $
```

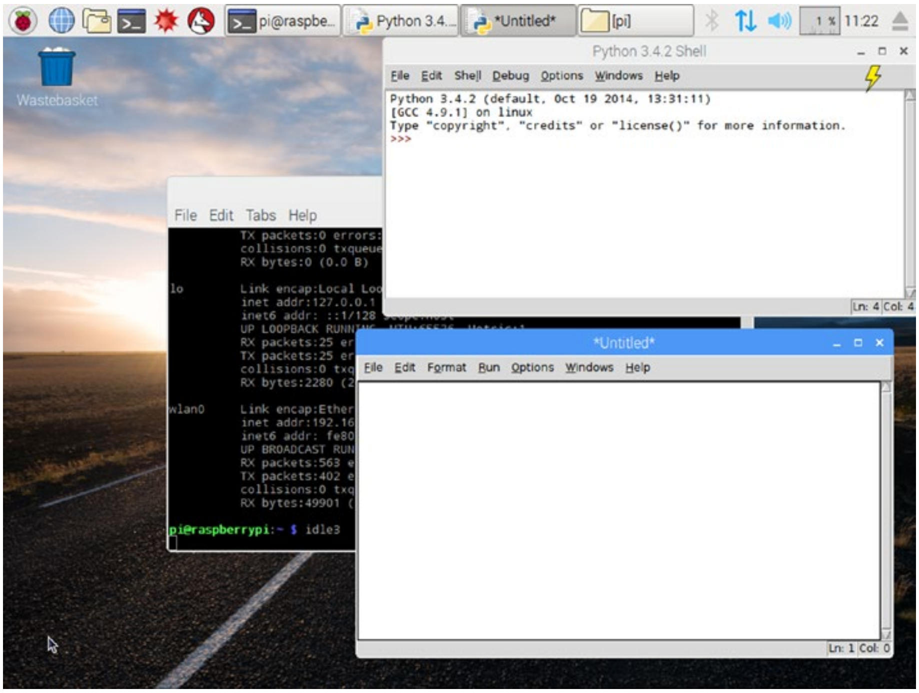
In the above example, ***python3*** is the interpreter and ***test.py*** is the filename. In a case where the python ***test.py*** file is not in the same directory where we're invoking the ***python3*** interpreter from, we have to provide the absolute path of the python file.

IDEs for Python

An Integrated Development Environment (IDE) is a software suite which has all the basic tools to write and test programs. A typical IDE has a compiler, a debugger, a code editor, and a build automation tool. Most programming languages have various IDEs to make programmers' lives better. Python too has many IDEs. Let's have a look at a few IDEs for Python.

IDLE

IDLE stands for ***Integrated DeveLopment Environment***. It comes bundled with Python installation. IDLE3 is for Python 3. It's popular with beginners in Python. Just run ***idle3*** in **lxterminal**. The following is a screenshot (Figure 3-1) of an IDLE3 code editor and an interactive prompt.



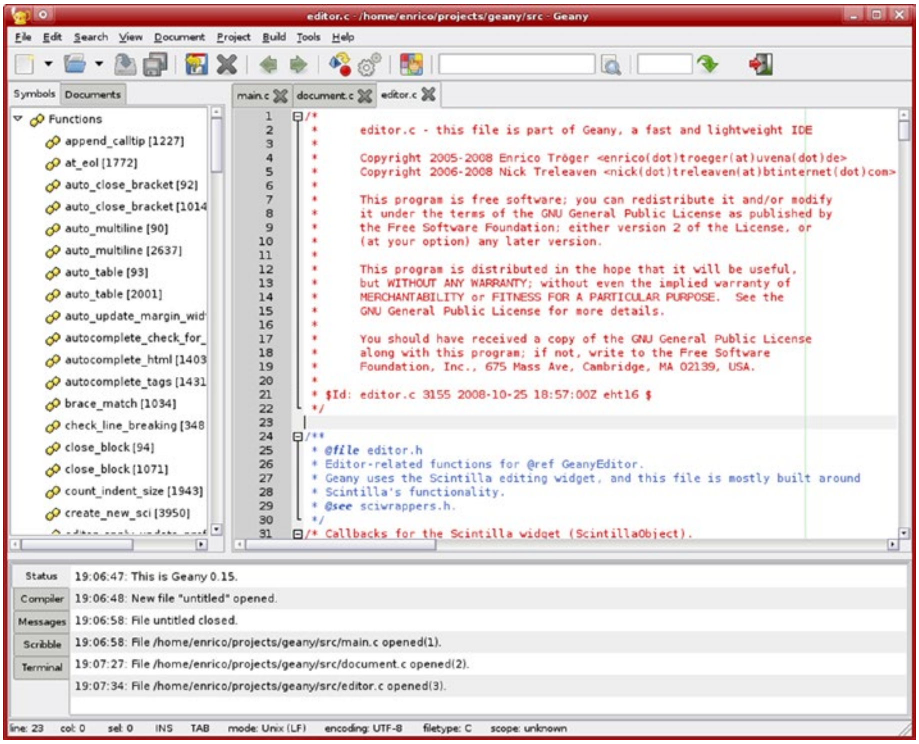
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

```
          TX packets:0 errors:
          collisions:0 txqueue
          RX bytes:0 (0.0 B)

lo        Link encap:Local Loo
          inet addr:127.0.0.1
          inet6 addr: ::1/128
          UP LOOPBACK RUNN
          RX packets:25 er
          TX packets:25 er
          collisions:0 txq
          RX bytes:2280 (2

wlan0     Link encap:Ether
          inet addr:192.16
          inet6 addr: fe80
          UP BROADCAST RUN
          RX packets:563 e
          TX packets:402 e
          collisions:0 txq
          RX bytes:49901 (

pi@raspberrypi:~ $ idle3
```

Figure 3-1. *IDLE3*

Geany

Geany is a text editor using the GTK+ toolkit with the basic features of an integrated development environment. It supports many filetypes and has some nice features. See `www.geany.org` for more details. The following (Figure 3-2) is a screenshot of the Geany text editor.



```
/*
 *      editor.c - this file is part of Geany, a fast and lightweight IDE
 *
 *      Copyright 2005-2008 Enrico Tröger <enrico(dot)troeger(at)uvena(dot)de>
 *      Copyright 2006-2008 Nick Treleaven <nick(dot)treleaven(at)btinternet(dot)com>
 *
 *      This program is free software; you can redistribute it and/or modify
 *      it under the terms of the GNU General Public License as published by
 *      the Free Software Foundation; either version 2 of the License, or
 *      (at your option) any later version.
 *
 *      This program is distributed in the hope that it will be useful,
 *      but WITHOUT ANY WARRANTY; without even the implied warranty of
 *      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *      GNU General Public License for more details.
 *
 *      You should have received a copy of the GNU General Public License
 *      along with this program; if not, write to the Free Software
 *      Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 *
 * $Id: editor.c 3155 2008-10-25 18:57:00Z eht16 $
 */

/**
 * @file editor.h
 * Editor-related functions for @ref GeanyEditor.
 * Geany uses the Scintilla editing widget, and this file is mostly built around
 * Scintilla's functionality.
 * @see sciwrappers.h.
 */
/* Callbacks for the Scintilla widget (ScintillaObject).
```

```
19:06:47: This is Geany 0.15.
19:06:48: New file "untitled" opened.
19:06:58: File untitled closed.
19:06:58: File /home/enrico/projects/geany/src/main.c opened(1).
19:07:27: File /home/enrico/projects/geany/src/document.c opened(2).
19:07:34: File /home/enrico/projects/geany/src/editor.c opened(3).
```

```
line: 23   col: 0   sel: 0   INS   TAB   mode: Unix (LF)   encoding: UTF-8   filetype: C   scope: unknown
```

Figure 3-2. *Geany*

Geany is pre-installed in the latest versions of Raspbian. If your Raspbian installation does not have Geany then it can be installed by running ***sudo apt-get install geany*** in **lxterminal**. Once installed, it can be found the **Raspbian Menu ➤ Programming ➤ Geany Programmer's Editor** as shown in the screenshot (Figure 3-3) below.

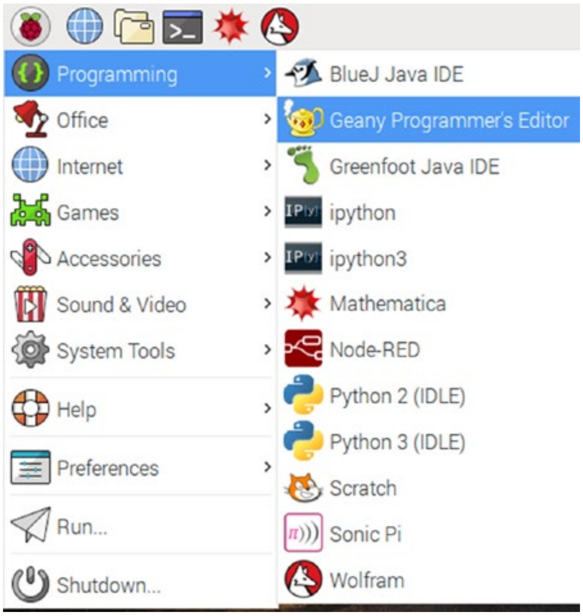
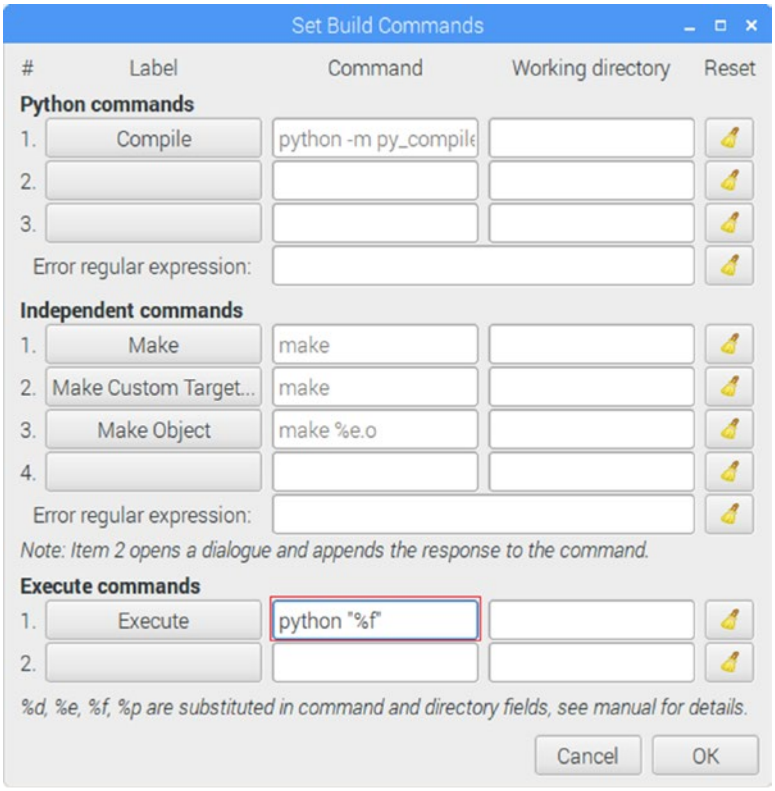


Figure 3-3. *Raspbian Menu*

Type ***print("Hello World!")*** in the code editor and save the file in ***/home/pi*** directory as ***test.py***. Click **Build** in the menu bar and then **Execute**. We can also use the keyboard shortcut key **F5** to execute the program. The program will execute in an **lxterminal** window. We have to press the **Enter** key to close the execution window. The default Python interpreter for **Geany** is Python 2. We need to change it to Python 3. To do that, go to **Build ➤ Set Build Commands**. The following (Figure 3-4) window will appear.



Set Build Commands

#	Label	Command	Working directory	Reset
Python commands				
1.	Compile	python -m py_compile		
2.				
3.				
	Error regular expression:			
Independent commands				
1.	Make	make		
2.	Make Custom Target...	make		
3.	Make Object	make %e.o		
4.				
	Error regular expression:			
Execute commands				
1.	Execute	python "%f"		
2.				

Note: Item 2 opens a dialogue and appends the response to the command.

%d, %e, %f, %p are substituted in command and directory fields, see manual for details.

Figure 3-4. *Set Build Commands*

In this window, under the **Execute commands** section, change ***python "%f"*** (highlighted in the red box in the image above) to ***python3 "%f"*** to set the Python 3 interpreter as the default interpreter. After that, run the program again to verify that everything is done correctly.

EXERCISE

Complete the following exercise for understanding Python 3 background better.

- Visit and explore the Python homepage `www.python.org`
- Visit and explore the Python Documentation page `https://docs.python.org/3/`
- Check the version-wise new features of the latest releases of Python at `https://docs.python.org/3/whatsnew/index.html`
- Search for the 20th undocumented principle of the Zen of Python on the Internet.

Conclusion

In this chapter, we learned the background, history, and features of Python. We also studied the important differences between Python 2.x and Python 3.x. We learned to use Python 3 in scripting and interpreter modes. We had a look at a few popular IDEs for Python and configured ***geany*** for Python 3 on the Pi. We will use Python 3 with ***mpi4py*** in the later parts of the book for parallel programming for the mini-supercomputer we are shortly going to build. In the next chapter, we will learn the basics of **supercomputing**.

CHAPTER 4

Introduction to Supercomputing

In the last chapter, we learned the history and philosophy of the Python programming language.

In this short chapter, we will study the concepts and history of supercomputing.

Concept of the Supercomputer

A supercomputer is a special computer with a lot of processing power. This is the simplest definition of the term supercomputer. The key feature which distinguishes supercomputers from other classes of computers is their massive processing power.

Supercomputers are used in computationally intensive applications. Mostly these are scientific applications. A few examples are the following:

- Weather prediction
- Climate research
- Molecular modelling
- Physical Simulations
- Quantum mechanics
- Oil and gas exploration

Brief history of Supercomputers

Control Data Corporation (CDC) is the cradle of supercomputers. Here, in 1964, Seymour Cray built **CDC 6600**. It was dubbed as the first **supercomputer** as it outperformed all other contemporary computers. Its processing speed was about 10 MHz. In 1968, Seymour Cray built **CDC 7600**. Its processing speed was 35 MHz. Again, it was the fastest computer and outran all other computers in terms of computing power.

This was the genesis of supercomputers. Eventually, Cray left **CDC** and formed his own company for designing and developing supercomputers. Cray created some of the most successful supercomputers in history, namely **Cray 1**, **Cray X-MP**, **Cray 2**, and **Cray Y-MP**. The 1990s saw the era of massively parallel supercomputers with thousands of processors connected to each other in various configurations. A notable example of this is the **Intel Paragon** which could have many **Intel i860** processors.

The speed of a supercomputer is measured in **Floating Point Operations Per Second (FLOPS)** instead of **MIPS (Million Instructions Per Second). Intel ASCI Red** was the first **TFLOPS (Tera FLOPS)** supercomputer. In 2008, **IBM Roadrunner** became the first supercomputer with the speed of **PFLOPS (Peta FLOPS)**.

The next breakthrough in the area of supercomputing will be the **Exascale supercomputer** with the processing speed to be measured in **Exa-FLOPS**.

I am not providing a list here of the top 10 supercomputers or the fastest supercomputers. This is because the list keeps on changing every year. Also, supercomputers are ranked based on various parameters, so no ranking from different sources based on diverse parameters will ever be the same.

Cluster

While designing massively parallel computing systems, generally two approaches are followed. The first one is to involve thousands of computers spread across a wide geographical area using the Internet to solve a particular problem. This works well over a wide area network like the Internet. These types of systems are called distributed systems. Another approach to this is to place thousands of processing nodes in close proximity to each other. This saves a lot of time on communication and most of the processing power is used to solve the computationally massive problem. This approach is known as clustering. All supercomputers fall in this category.

A computer cluster is defined as a group of loosely or tightly coupled computers that work together. The computers in a cluster are known as nodes. All the nodes in a cluster do exactly same type of task.

The mini-supercomputer we are going to develop is going to be a cluster of Pis. All supercomputers are clusters, but all clusters are not supercomputers. As we have learned in the definition of the supercomputer, supercomputers have massive processing power. That's why every cluster does not qualify to be called a supercomputer. The cluster we will build here is not a supercomputer as it pales before a real-world supercomputer in terms of processing power, but it works on the same principle as a real-world supercomputer. Hence we will call it a mini-supercomputer. Ever since the introduction of massively parallel systems, the boundaries between large-scale clusters and less powerful computers have begun to fade. Few homemade clusters today are as powerful as 1980s supercomputers. Based on their configuration, commodity clusters are classified into the following two categories.

Heterogenous Cluster

When all the nodes of the cluster do not have the exactly same hardware configuration, then the cluster is called a heterogeneous cluster. When making my cluster, I used two units of Pi B+, one unit of Pi 2, and one unit of Pi 3, so my cluster is a heterogeneous cluster.

Beowulf Cluster

Unlike the heterogeneous clusters, all the nodes in a Beowulf cluster have exactly the same configuration. We can make homogeneous and Beowulf clusters out of commodity-grade hardware as well as SBCs like Raspberry Pi. Almost all clusters use some distribution of Linux as the OS for their nodes.

Depending on the availability of the models of Pi near you, you can make either a heterogeneous or a Beowulf cluster.

Parallelism and Concurrency

Let's explore a few important terms in the area of supercomputing.

Parallelism

Parallelism means the computational tasks are carried out in parallel. This means that the tasks are carried out simultaneously (at the same time). Parallelism is usually employed in cases where the computational problems are very large. Large problems are often divided into smaller sub-problems and then solved parallelly by the computers. With the introduction of multi-core processors, execution of parallel programs is supported by the hardware itself. Another way to run parallel programs is to create parallel systems by using multiple different computers. Parallel is an antonym of serial which means in series and one after the other. Parallelism is closely related to another term, concurrency.

Let me explain parallelism in simple words. Suppose there are two jobs to be completed and two people are available to take up the jobs. Both the people are assigned one job each and they start working independently of each other. This is known as parallelism.

Concurrency

Concurrency means many computational tasks are concurrently progressing. It is not necessary for the tasks to progress at the same time. In parallelism, all the tasks are executing at the same time. In concurrency, they need not. A concurrent system is one where a computation can advance without waiting for all other computations to complete, and more than one computation can advance at the same time. The best example of concurrency is the process scheduling in the OS.

Let me explain concurrency in simple words. Suppose two jobs are to be completed and only one person is available to do all the work. The person decides to start with the first job. He does 30% of it and then switches to the second job. He completes 40% of the second job and switches back to the first job. This type of switch happens multiple times. We can say that the work on both the jobs is in progress. Although the jobs are not done simultaneously, the jobs are progressing towards completion. In the end, both jobs are finished. Concurrent is an antonym of sequential.

Parallel Programming

All clusters and supercomputers use parallelism to decompose a computationally massive task into smaller chunks and then collect the results back for the final output. The programming paradigm which supports this type of operation is called parallel programming. **Message Passing Interface (MPI)** is one of the most-used parallel programming standards in industry and academics. We will study how to install it on the Pi for Python 3 in the next chapter.

Conclusion

In this short chapter, we learned a few important concepts related to supercomputing and we also studied the history of supercomputers.

In the next chapter, we will study how to set up a node of a Raspberry Pi cluster.

CHAPTER 5

Message Passing Interface

In the last chapter, we learned the history and philosophy of supercomputers. We also learned important concepts related to supercomputing.

In this short chapter, we will get started with installing necessary packages and libraries on a Raspberry Pi. We will install MPI4PY, which is a Python library for MPI. Finally, we will install the utility **nmap** for node discovery.

Message Passing Interface

The **Message Passing Interface** Standard (**MPI**) is a message passing library standard based on the recommendations of the **MPI Forum**. The MPI Forum has over 40 participating organizations in the USA and Europe. The goal of the Message Passing Interface is to define a portable, efficient, and flexible standard for message passing that will be widely used for writing a wide variety of message passing programs. MPI is the first vendor-independent message passing library standard. The advantages of developing message passing programs using the MPI standard are portability, efficiency, and flexibility. Although MPI is not an IEEE or ISO standard, it has become the industry standard for writing message passing programs for a variety of platforms like **High Performance Computing (HPC)**, parallel computers, clusters, and distributed systems. The MPI standard defines the syntax and semantics of library routines for writing portable message-passing programs in C, C++, and Fortran.

A few important facts related to MPI are the following:

- MPI is a specification for libraries. MPI itself is not a library.
- The goal of MPI is that the message passing standard should be practical, portable, efficient, and flexible.
- Actual MPI libraries differ a bit depending on how the MPI standard is implemented.
- The MPI standard has gone through several revisions. The most recent version is MPI-3.2.

■ **Note** Explore the MPI Forum's homepage (`www.mpi-forum.org`) and the MPI standard documentation page (`www.mpi-forum.org/docs/docs.html`) for more information on the MPI Forum and standard.

History and Evolution of the MPI Standard

On April 29–30 of 1992, a workshop on Standards for Message Passing in a Distributed Memory Environment was held in Williamsburg, Virginia. The basic features essential to a standard message passing interface were discussed and a working group to continue the standardization process was established. From there on, the work on MPI continued and the working group met regularly. The draft MPI standard was presented at the Supercomputing '93 conference in November 1993. After a period of public comments, which resulted in some changes in MPI standards, version 1.0 of MPI was released in June 1994. These meetings and the email discussion together led to the formation of the MPI Forum. The MPI standardization efforts involved about 80 people from 40 organizations in the United States and Europe. As of now, the latest version of MPI is MPI-3.2 which we will use for building the cluster.

Features of MPI

MPI is optimized for distributed systems with distributed memory and a network that connects all the nodes as depicted in Figure 5-1.

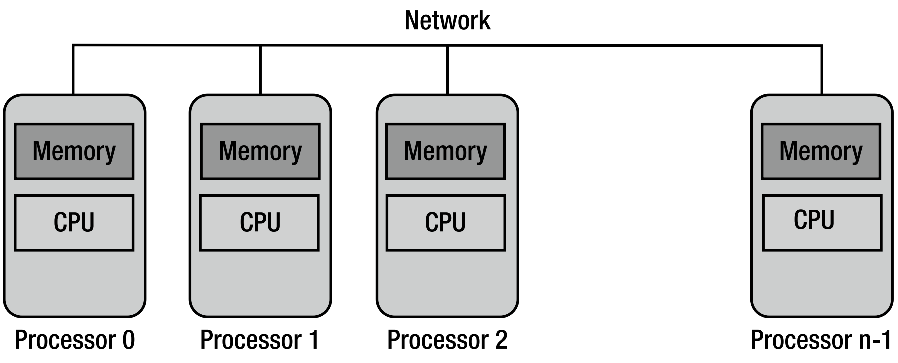


Figure 5-1. *Distributed memory system*

The following are the features of the Message Passing Interface:

- **Simplicity**: The basics of the paradigm in MPI are traditional communication operations.
- **Generality**: It can be implemented on most systems built on parallel architectures.
- **Performance**: The implementation can match the speed of the underlying hardware.
- **Scalability**: The same program can be deployed on larger systems without making any changes to it.

We will study more details of MPI paradigms when we start learning how to code with MPI4PY.

Implementations of MPI

As we have seen that MPI is not a library but a standard for development of message-passing libraries, there are several implementations of MPI. The following are the most popular implementations:

- MPICH
- MP-MPICH (MP stands for multi-platform)
- winmpich
- MPI BIP
- HP's MPI
- IBM's MPI
- SGI's MPI
- STAMPI
- OpenMPI

MPI4PY

MPI4PY stands for **MPI for Python**. MPI for Python provides MPI bindings for Python. This allows any Python program to use a multiple-processor configuration computer. This package is built on top of the MPI-1/2/3 specifications. It provides an object-oriented interface for parallel programming in Python. It supports point-to-point (send and receive) and collective (broadcast, scatter, and gather) communications for any Python object.

Figure 5-2 depicts the overview of MPI4PY.

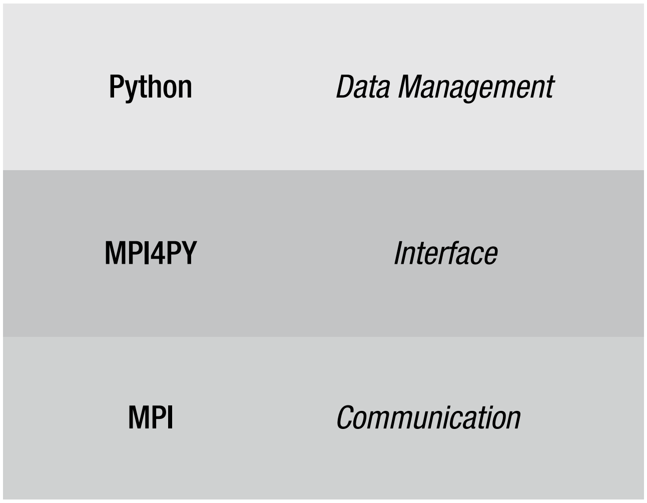


Figure 5-2. *Philosophy of MPI4PY*

Why Use the Python, MPI, and MPI4PY Combination?

Python is one of the three most-used programming languages in **HPC** (**High Performance Computing**). The other two are C and FORTRAN. As we have seen earlier, Python syntax is easy to understand and learn. MPI is the de facto standard for HPC and parallel programming. It is well established and has been there since around 1994 (over 20 years). MPI4PY is a well-regarded, clean, and efficient implementation of MPI for Python. It covers most of the MPI-2 standard. That's why we should use Python 3 with MPI4PY on the Raspberry Pi for parallel programming.

Installing MPI4PY for Python3 on Raspbian

Installing MPI4PY for Python 3 on Raspbian is very simple. The user just has to run the following command in ***lxterminal***:

```
sudo apt-get install python3-mpi4py -y
```

It will take a few minutes to install MPI4PY for Python 3.

To check if it is installed, run the following command:

```
mpirun hostname
```

It should print the hostname ***raspberrypi*** as the output.

Run the following command to launch multiple processes:

```
mpirun -np 3 hostanme
```

The output is as follows:

```
raspberrypi
raspberrypi
raspberrypi
```

We can also run the following command to check the MPI version installed on the system:

```
mpirun -V
```

This way we can install, run, and verify MPI4PY.

■ **Note** Visit the ***mpirun*** manual page (`www.open-mpi.org/doc/v1.8/man1/mpirun.1.php`) for more details. In the later part of the book, we will use mpirun extensively with Python 3, and there we will study it in detail.

Installing nmap

nmap is a network security scanner. We are going to use it to discover the IP addresses of our Pi. We will use it in the next chapter. For now, just install ***nmap*** by running the following command:

```
sudo apt-get install nmap
```

Conclusion

In this chapter, we learned to prepare a Pi for supercomputing by installing MPI4PY. In the next chapter, we will build a supercomputer by connecting multiple Pis together.

CHAPTER 6

Building the Supercomputer

In the previous chapter, we prepared the Pi for supercomputing by installing the necessary libraries, frameworks, and tools. In this chapter, we will learn to create a network of multiple Pis and use it to run various commands in parallel.

Making a Backup of the MicroSD card

Once we configure the Pi, update it, and install necessary packages and utilities, we should make a backup of the microSD card. This is necessary because (God forbid!) if the microSD card or the Pi were damaged or lost, then we could resume our work with the backup. It is advised to always have a backup of a microSD card with Raspbian OS installed and updated. Additionally, backing up after installing the necessary packages, tools, and utilities required for the project is also a good idea which I always follow without fail.

To make a backup of the microSD card, first remove it from the Pi and insert it into the SD card reader. Connect the SD card reader to a Windows computer where Win32DiskImager is installed. Open Win32DiskImager and choose a location of your choice. Choose an appropriate name for the backup file. Append the extension .img or .IMG after the filename. The .img or .IMG extension is used for raw storage disk images. See the following (Figure 6-1) screenshot for an example.

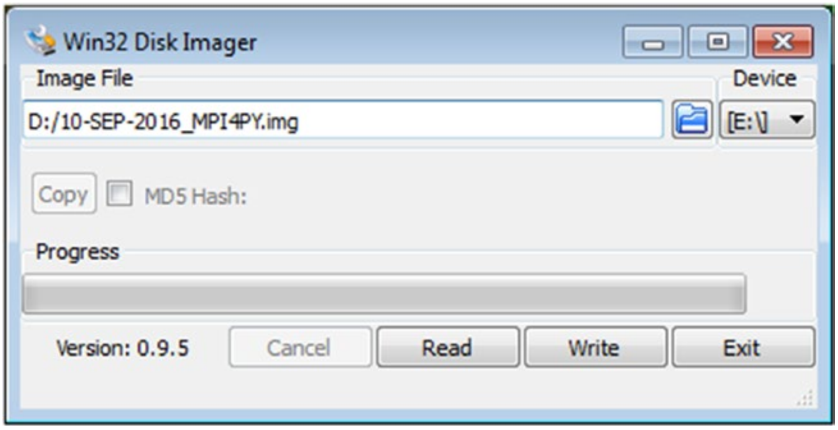


Image File: D:/10-SEP-2016_MPI4PY.img

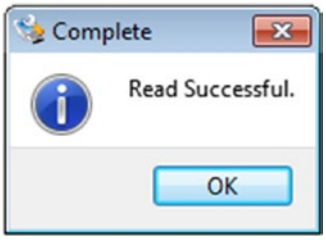
Device: [E:\]

MD5 Hash:

Version: 0.9.5

Figure 6-1. *Taking the microSD card backup*

Then click the **Read** button. Once finished, it will display the following (Figure 6-2) dialogue box.


Complete

Read Successful.

Figure 6-2. *Backup completed*

The image file of the updated version of the Raspbian OS, MPICH, and MPI4PY installation is now saved on the hard drive. We can use this image to prepare other microSD cards for the rest of the nodes of the supercomputer. To do that, we just need to write this image to other microSD cards with Win32DiskImager.

Preparing Nodes of the Supercomputer

Using the OS image which we prepared in the previous section, we will prepare other nodes of the supercomputer. Write this OS image onto other microSD cards using **Win32DiskImager.** Once all the microSD cards are ready, insert them into Pis. We should have a unique hostname for each Pi. We can change the hostname of a Pi with raspi-config. Go to Advanced Options and select A2 Hostname. Change the hostname as shown in the screenshot below (Figure 6-3).

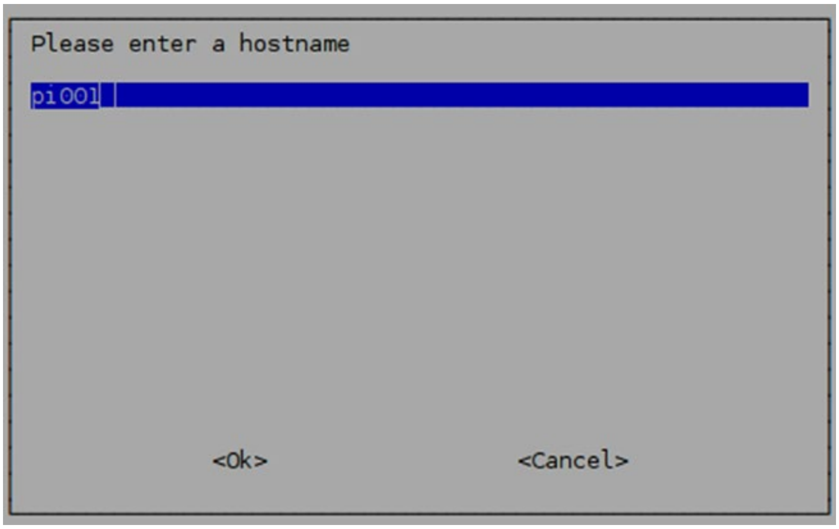
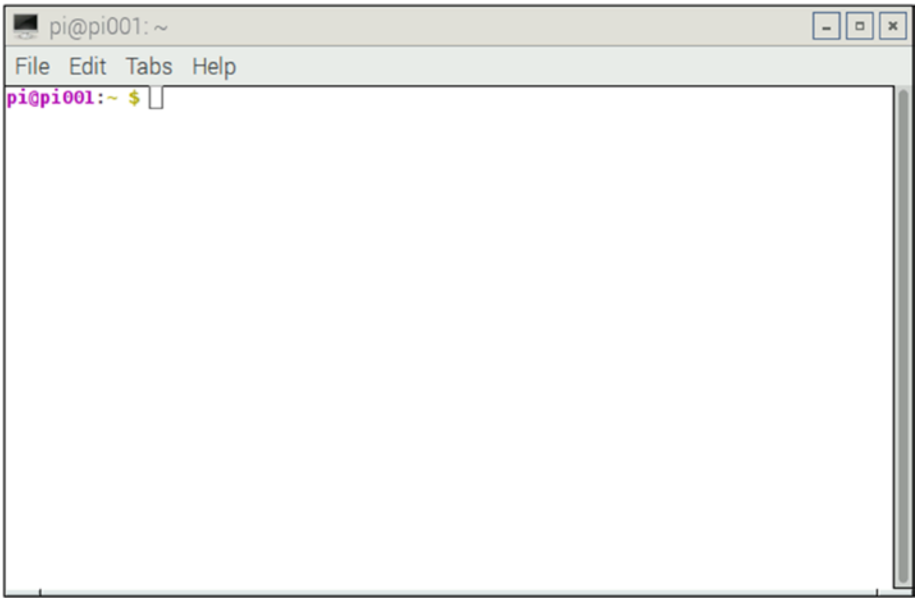

```
Please enter a hostname

pi001
```

Figure 6-3. *Changing the hostname*

Once the hostname is changed, the prompt in lxterminal appears as in the screenshot below (Figure 6-4).



```
pi@pi001:~ $
```

Figure 6-4. *The lxterminal prompt after changing the hostname*

As you can see, the hostname `raspberrypi` in the prompt and in the title of the lxterminal window is replaced with `pi001`.

Follow the same steps for all the Pis and change their hostnames to `pi002`, `pi003`, and so on.

Networking the Pis

Now, this is a bit of a tricky part. There are many ways we can network the Pis together. The choice depends upon the infrastructure and the budget available. Let's explore a few ways to accomplish this without much hassle.

LAN with DHCP

This option works well for managed network switches and WiFi routers. Access the management console of the managed switch or WiFi router. After logging in to the management console, there will be options to set the range of addresses for DHCP. Modify that range as we are going to add more devices to that.

For WiFi routers, the management console is usually a webpage. It can be accessed by connecting to the WiFi and then typing its IP address into a browser. It usually has authentication in the form of a username and a password, which is usually listed in the WiFi router manual. Every WiFi router has ethernet ports for wired LAN as shown below (Figure 6-5).

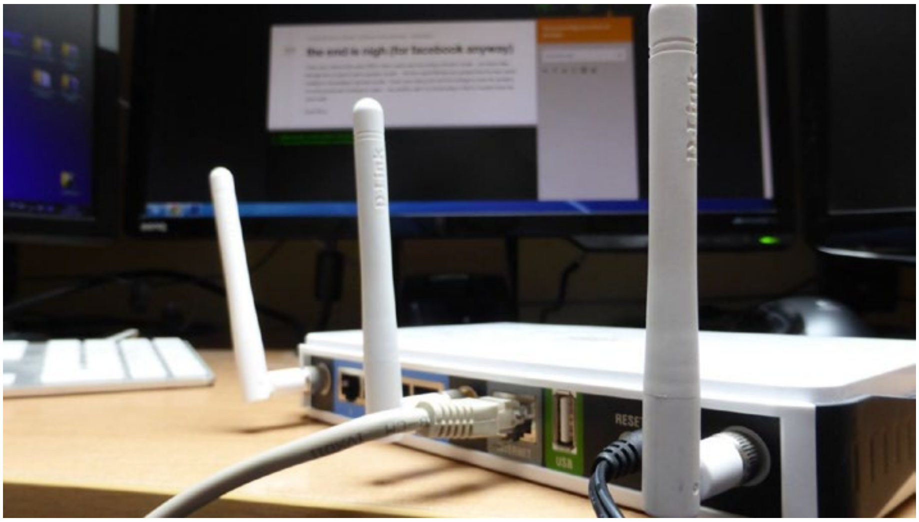


Figure 6-5. *Rear side of a WiFi router (image from `https://www.flickr.com/photos/smemon/`)*

Update the network settings in `/etc/network/interfaces` for connecting automatically to the LAN and DHCP as an IP address allocation scheme. Following is a sample `/etc/network/interfaces` file:

```
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp
```

WiFi Network

We know that all the models of Pi prior to Pi 3 require a USB WiFi adapter. The best option is if you have a WiFi router and many Pi 3s. If the Pis are of earlier models than Pi 3, you will have to invest in buying USB WiFi adapters. After equipping the relevant models of Pi with USB WiFi adapters, update the network settings in `/etc/network/interfaces` for connecting automatically to the WiFi.

The following is a sample `/etc/network/interfaces` file.

```
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

auto wlan0
iface wlan0 inet dhcp
wpa-ssid "ASHWIN"
wpa-psk "internet"
```

Replace `ASHWIN` with the ssid of your WiFi network and replace `internet` with the password of your WiFi network.

LAN with Static IP Addresses

This is my preferred method. All managed network switches and WiFi routers have a range for static IP addresses. Choose a few addresses for the nodes in the cluster and then update the `/etc/network/interfaces` file. Use the IP address of the managed network switch or WiFi router as the value for the gateway.

Using a low-cost unmanaged network switch is probably the cheapest option. "Unmanaged" means that there is no management console for them. Just connect the Pis to the switch with ethernet cables and update the `/etc/network/interfaces` as follows:

```
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static

# Your static IP
address 192.168.0.2

# Your gateway IP
gateway 192.168.0.1
netmask 255.255.255.0

# Your network address family
network 192.168.0.0
broadcast 192.168.0.255
```

For all the Pis, the network settings (except the IP address) will be the same as above. The IP addresses will be unique. Also, as the switch is unmanaged, we are assigning it an IP address manually by using the same value for the gateway (192.168.0.1 in the example above). The value of the gateway must be the same for all the Pis.

Reboot the Pis after changing the network settings.

The following are a few low-cost unmanaged switches and their respective product pages.

- www.dlink.ru/mn/products/1
- www.dlink.lt/en/products/1/1857.html
- www.dlink.ru/mn/products/1/2110.html

Using nmap to Find the IP Addresses of Pis

Regardless of the type of network (ethernet or WiFi) and the IP address allocation scheme (static or dynamic), we will need to know the IP addresses of all the Pis in the network in order to use the network of Pis as a cluster. In the previous chapter, we installed the `nmap` utilty. We will now use it for finding the IP addresses of the Pis in the network.

Connect a monitor, keyboard, and mouse to `pi001`. We will use `pi001` as the master node. We will use lxterminal in `pi001` for running commands and parallel programs for the supercomputer.

Connect all the Pis to the network. We need not connect any display or I/O devices to other Pis.

Boot up all the Pis. Once all the Pis boot up, scan the network with `nmap`.

The following is the command for scanning the network:

```
sudo nmap -sn 192.168.0.*
```

In the command above, the first three bytes in `192.168.0.*` correspond to the IP address of my network. Replace them with the first three bytes of your network identifier and run the command in lxterminal on `pi001`.

The output will be as follows:

```
Starting Nmap 6.47 ( http://nmap.org ) at 2016-09-15 18:02 IST
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is
disabled.\
Try using --system-dns or specify valid servers with --dns-servers
Nmap scan report for 192.168.0.2
Host is up (0.0020s latency).
Nmap scan report for 192.168.0.3
Host is up (0.0018s latency).
Nmap scan report for 192.168.0.4
Host is up (0.0016s latency).
Nmap scan report for 192.168.0.5
Host is up (0.0014s latency).
Nmap done: 256 IP addresses (4 hosts up) scanned in 2.70 seconds
```

Write down the IP addresses of all the Raspberry Pis in the network. In this case, the IP addresses are 192.168.0.2, 192.168.0.3, 192.168.0.4, and 192.168.0.5.

Running the hostname Command on Multiple Pis with mpirun

On `pi001`, navigate to `/home/pi` by running the following command:

```
cd ~
```

Create a new file named `myhostfile`. Add all the IP addresses written down earlier to `myhostfile` as follows:

```
192.168.0.2
192.168.0.3
192.168.0.4
192.168.0.5
```

Now run the following command:

```
mpirun -hostfile myhostfile -np 4 hostname
```

It will show an output with an error and the command `hostname` will not run on all the hosts listed in the file `myhostfile`.

This is because we are trying to run a command from `pi001` remotely on `pi002`, `pi003`, and `pi004`. We do not have authentication for that.

Exchanging the ssh-keygen Keys for Automatic Authentication

The `ssh-keygen` utility is used to generate authentication keys. To establish authentication between any two Linux computers, these are the steps:

1. Generate keys for both the hosts using `ssh-keygen`.
2. Exchange the keys between hosts by remotely copying them to each other.
3. Add the keys to the list of authorized hosts.

Once this is done, we can do the following actions without a password, since after the key exchange the password will not be prompted for again.

1. Log in to the remote host.
2. Execute a shell command on the remote host.

We can also use the ssh command in a shell script to automate the tasks in a remote host. We are using pi001 as the master node, so we want to remotely run the commands from pi001 on other nodes and vice-versa. I have set up a cluster of four nodes, so the pairs of hosts for the key exchange are (pi001,pi002), (pi001,pi003), and (pi001,pi004).

Let's exchange the keys. Open lxterminal and go to the home directory of pi001.

```
cd ~
```

Run the ssh-keygen command to generate the key. Just press the Enter key every time it prompts for any input. The following is the output on lxterminal:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/pi/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/pi/.ssh/id_rsa.
Your public key has been saved in /home/pi/.ssh/id_rsa.pub.
The key fingerprint is:
03:bc:3f:5a:28:88:b7:ac:6c:50:f0:81:5e:f9:6d:5f pi@pi001
The key's randomart image is:
+---[RSA 2048]----+
| . .             |
|o .o .           |
|.o... +          |
| .o . = E        |
| . o S .         |
|.. . o o         |
|o o . . +        |
|.+ . . o .       |
|ooo .            |
+-----------------+
```

Note that the image displayed on the command prompt will be different every time as the key generated for each execution is different.

The above execution creates a hidden directory .ssh in the home directory of the Pi. Go to the .ssh directory.

```
cd .ssh
```

Check the contents of the .ssh directory by running the ls command as follows:

```
pi@pi001:~/.ssh $ ls
id_rsa id_rsa.pub
```

In the output above, id_rsa is the private key and id_rsa.pub is the public key for the host pi001. We have to copy the public key to the hosts where we want to remotely login and execute commands.

To keep things organized, we copy the public key `id_rsa.pub` to a new file, `pi01`.

```
cp id_rsa.pub pi01
```

We need to add the contents of this `pi01` file to the `authorized_keys` file of other hosts to enable remote access without authentication.

Now log in to `pi002` with the following command:

```
ssh pi@192.168.0.3
```

We will be prompted for the password of `pi002`. Enter the password.

Like we did on `pi001`, run the following commands on `pi002` to generate the public key:

```
ssh-keygen
cd .ssh
cp id_rsa.pub pi02
```

Now we need to copy the public key file of `pi001` to `pi002` using `scp`.

```
scp 192.168.0.2:/home/pi/.ssh/pi01 .
```

Add the contents of `pi01` to `authorized_keys` by running the following command:

```
cat pi01>>authorized_keys
```

Finally, log out from `pi002` using the logout commands.

For `pi003`, we have to follow the same steps again.

Login to `pi003`.

```
ssh pi@192.168.0.4
```

Run the following sequence of commands on `pi003`:

```
ssh-keygen
cd .ssh
cp id_rsa.pub pi03
scp 192.168.0.2:/home/pi/.ssh/pi01 .
cat pi01>>authorized_keys
logout
```

For `pi004`, we have to follow the same steps again.

Login to `pi004`.

```
ssh pi@192.168.0.5
```

Run the following sequence of commands on pi004:

```
ssh-keygen
cd .ssh
cp id_rsa.pub pi04
scp 192.168.0.2:/home/pi/.ssh/pi01 .
cat pi01>>authorized_keys
logout
```

In pi001 run the following sequence of commands to copy the public keys of pi002, pi003, and pi004 to pi001.

```
cd /home/pi/.ssh
scp 192.168.0.3:/home/pi/.ssh/pi02 .
scp 192.168.0.4:/home/pi/.ssh/pi03 .
scp 192.168.0.5:/home/pi/.ssh/pi04 .
```

Then run the following commands to add these public keys to the list of authorized keys of pi001:

```
cat pi02>>authorized_keys
cat pi03>>authorized_keys
cat pi04>>authorized_keys
```

This completes the setup of the cluster. To test the setup, run the following command on pi001:

```
mpirun -hostfile myhostfile -np 4 hostname
```

The output of the command above should be as follows:

```
pi001
pi002
pi003
pi004
```

Congrats! We have built our own mini-supercomputer cluster. In the next section, we will learn how to organize our cluster in a nice-looking stack.

Organizing the Pis in the Cluster

When I built my first cluster, I thought of creating a custom acrylic case; however, the estimated cost exceeded my budget. I also thought of creating a custom case for the cluster by 3D printing, but the 3D printing contractor also quoted an astronomical sum for this. So I decided to try out a more cost-effective way to organize the Pis in the cluster. I used M3 Hex standoff spacers for creating a stack of Pis. We need two types of standoffs for this, male-to-female and female-to-female. We will use these to create a stack now. It is essential that the length of the standoffs must be at least 25mm to avoid the contact of Raspberry Pi PCBs to each other altogether.

■ **Note** Search for **M3 Hex standoff spacers** on Google.

Take four male-to-female standoffs and attach them to a Pi as shown in the image (Figure 6-6) below.

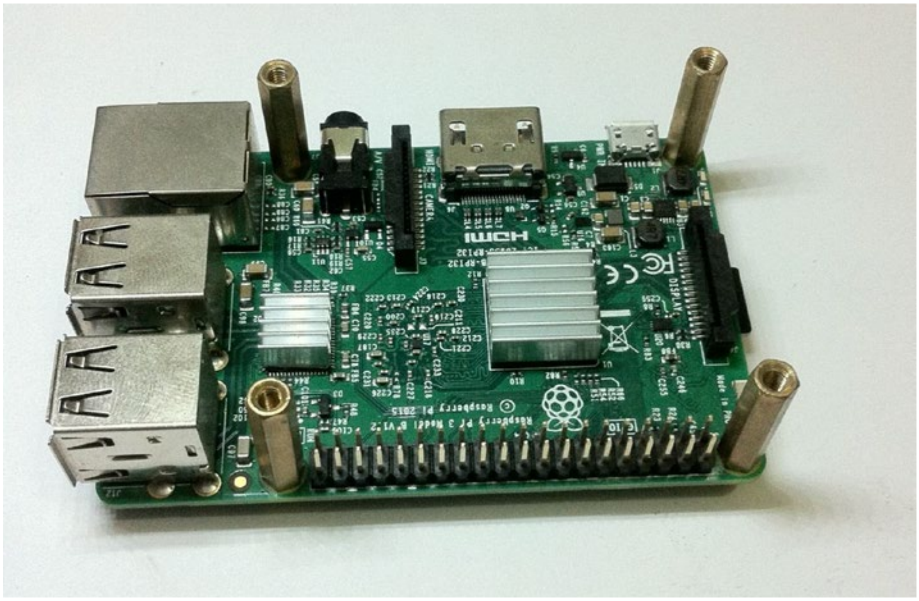


Figure 6-6. *Attaching male-to-male standoffs to the Pi*

After that, take four female-to-female standoffs and attach them to the bottom of the Pi as shown in the image (Figure 6-7) below.

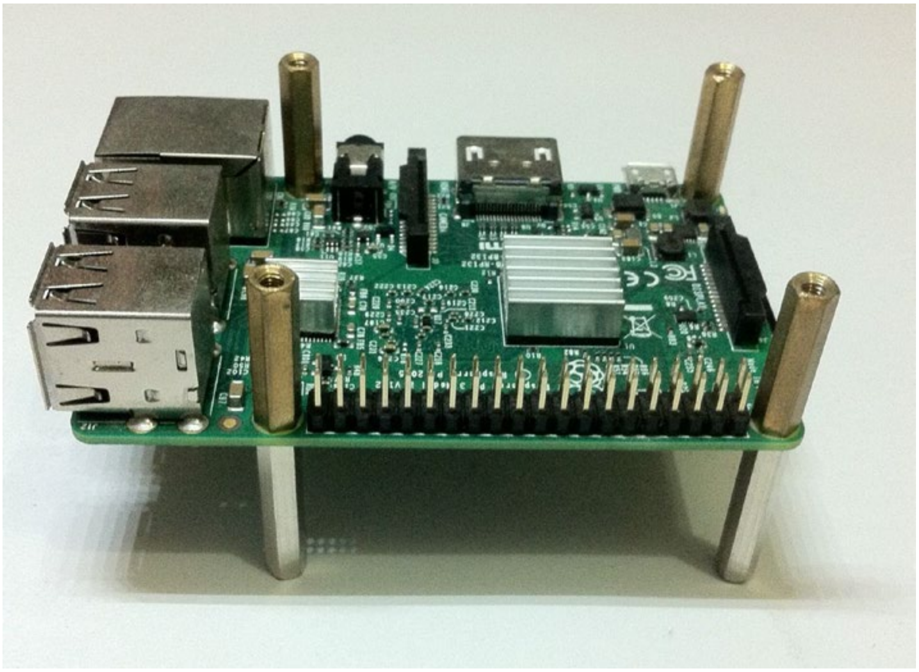


Figure 6-7. *Attaching female-to-female standoffs to the cluster base*

Now attach the second Pi to this as shown (Figure 6-8) below.

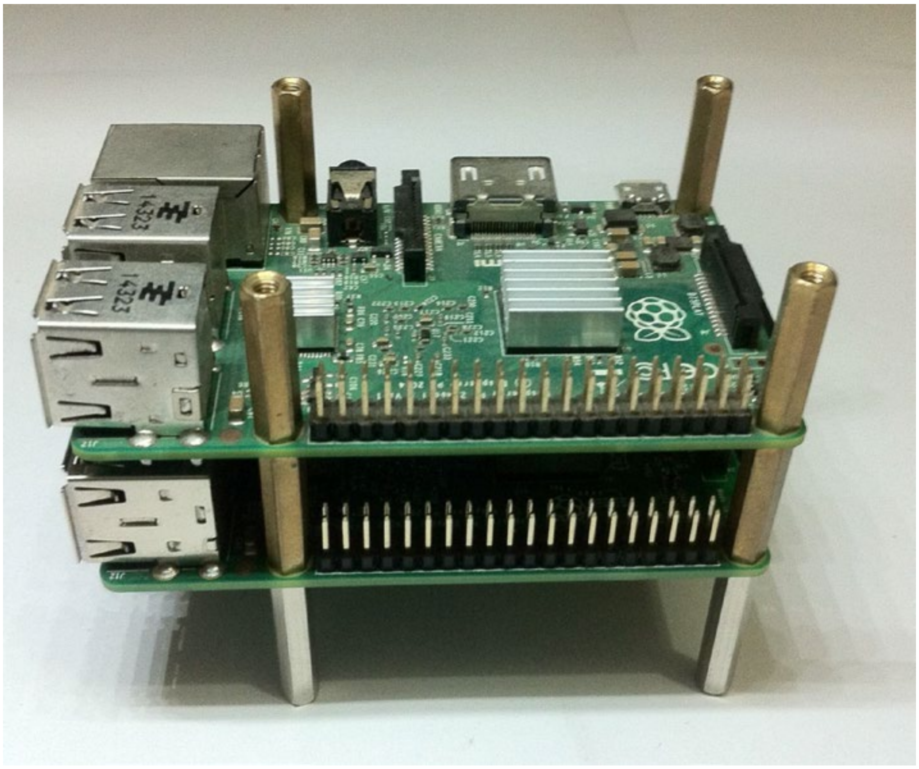


Figure 6-8. *Adding second Pi to the stack*

Finally, after adding the remaining two Pis to this, the cluster stack looks like the picture below (Figure 6-9).

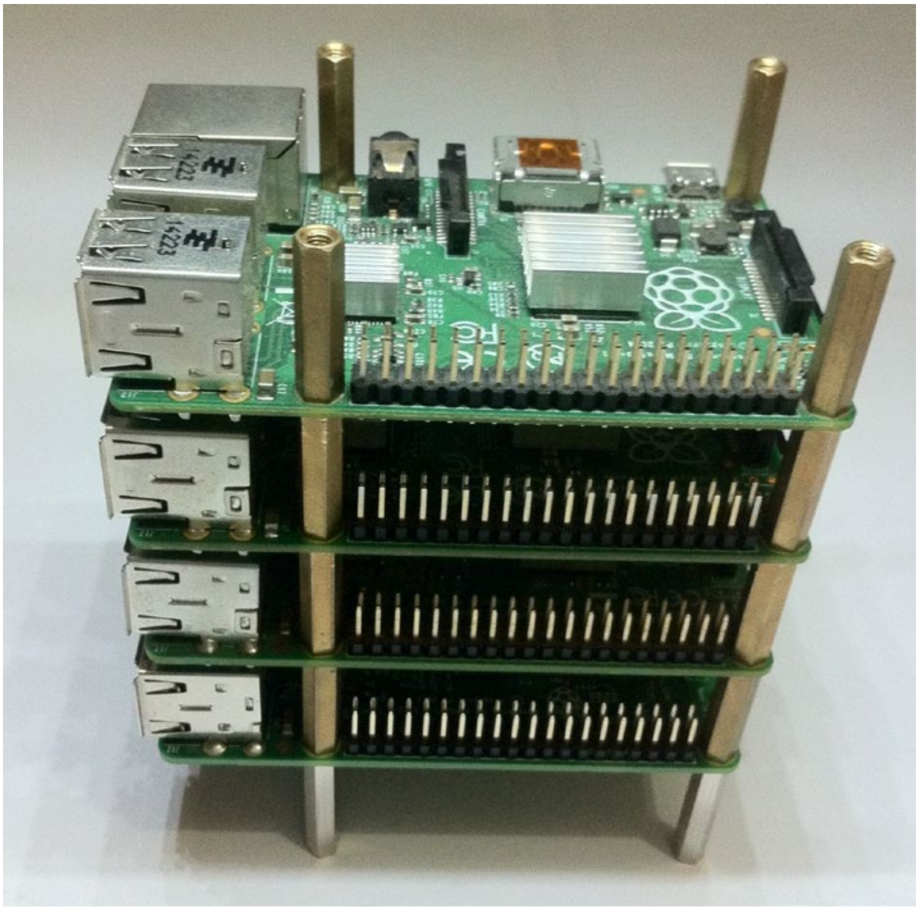


Figure 6-9. *Raspberry Pi Supercomputer stack*

Conclusion

In this chapter, we learned how to connect several Pis together to build an ultra-low-cost mini-supercomputer cluster. We also learned how to organize the cluster in a convenient stack. In the next chapter, we will learn how to overclock the various models of Pi to increase the computational power of the Pis in the cluster at no additional cost.

CHAPTER 7

Overclocking Raspberry Pi

In this chapter, we will learn how to increase the computing power of the various models of Raspberry Pi by overclocking the various components of this amazing little computer. We will study how to overclock through `raspi-config` and by altering the contents of `config.txt`.

Overclocking means to configure a computer's hardware parts to run at a faster rate than what was certified by the original manufacturer. The rate at which the part operates is generally specified in terms of clock frequency like MHz, GHz, etc. Usually the operating voltage of the overclocked component is also increased, which helps maintain the component's operational stability at the accelerated speeds. However, the downside of overclocking is that the given semiconductor device will generate and dissipate more heat when operated at higher frequencies and voltages than the stock settings, so most overclocking attempts will increase power consumption and heat dissipation as well. To mitigate the increased heat dissipation in the overclocked component, we usually have to install heatsinks and cooling systems.

Let's get started with the basics of overclocking Raspberry Pi. In this chapter, we will explore in detail how to install passive heatsinks and how to overclock various models of Raspberry Pi.

Risks of Overclocking Raspberry Pi

Overclocking Raspberry Pi allows us to get the best out of it. However, we should not do it without understanding the risks involved. It is vitally important that we understand what we are getting into.

Here are a few risks of overclocking which we MUST be aware of:

- **Life reduction**: Components may fail sooner.
- **Heat generation**: Operating at higher speeds generates and dissipates more heat.
- **File corruption**: Many overclockers have observed file corruptions at un-optimized overclocking settings.
- **Warranty-voiding**: Forced overvolting will void the warranty.

Installing a Heatsink on Pi

A heatsink is a device or substance for absorbing excessive, unwanted heat. If you are planning to overclock your Pi then installing heatsinks on the processor, RAM, and GPU is recommended.

Most heatsinks have adhesive stickers glued to them. Use them to install the heatsinks on the chips of Pi. Most people use passive heatsinks only when working with the Raspberry Pi, as active heat dissipation mechanisms like liquid cooling and/or radiators would be overkill for the Raspberry Pi.

Procuring Heatsinks

Many distributors sell heatsinks online. You can google the keywords **raspberry pi heatsink**.

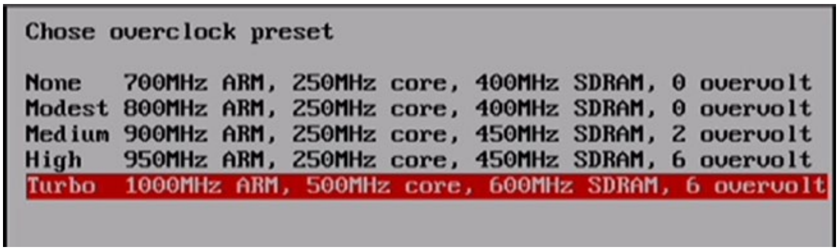
The following are links to the websites of various distributors selling heatsinks:

- `www.sparkfun.com/products/121`
- `www.adafruit.com/categories/151`
- `https://shop.pimoroni.com/products/heatsink`
- `www.modmypi.com/raspberry-pi/accessories/heat-sinks-and-cooling/raspberry-pi-heat-sink-kit-black`

Overclocking the Pi with raspi-config

We can overclock the Pi using the overclock option in the `raspi-config` tool.

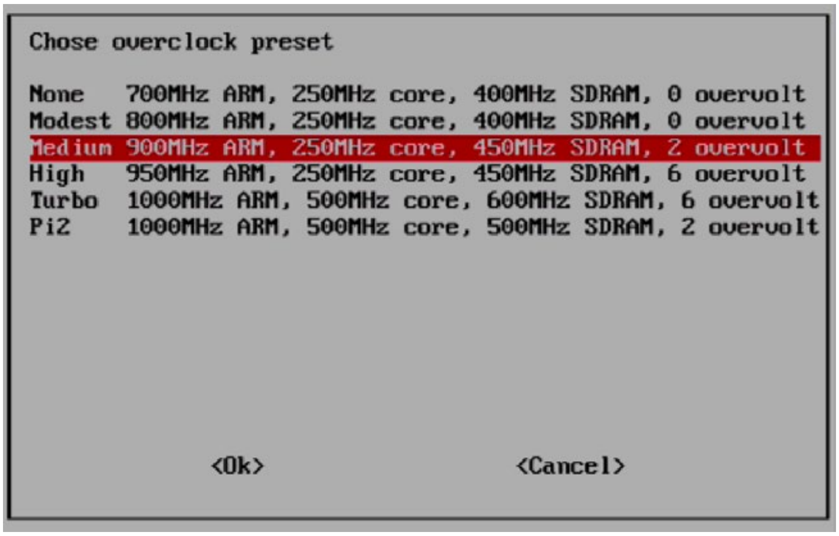
The following (Figure 7-1) is a screenshot of overclock options in Pi B and Pi B+.



```
Chose overclock preset

None    700MHz ARM, 250MHz core, 400MHz SDRAM, 0 overvolt
Modest  800MHz ARM, 250MHz core, 400MHz SDRAM, 0 overvolt
Medium  900MHz ARM, 250MHz core, 450MHz SDRAM, 2 overvolt
High    950MHz ARM, 250MHz core, 450MHz SDRAM, 6 overvolt
Turbo  1000MHz ARM, 500MHz core, 600MHz SDRAM, 6 overvolt
```

Figure 7-1. *Overclock options in Pi B and Pi B+*

The following (Figure 7-2) is a screenshot of overclock options in Pi 2.



```
Chose overclock preset

None   700MHz ARM, 250MHz core, 400MHz SDRAM, 0 overvolt
Modest 800MHz ARM, 250MHz core, 400MHz SDRAM, 0 overvolt
Medium 900MHz ARM, 250MHz core, 450MHz SDRAM, 2 overvolt
High   950MHz ARM, 250MHz core, 450MHz SDRAM, 6 overvolt
Turbo  1000MHz ARM, 500MHz core, 600MHz SDRAM, 6 overvolt
Pi2    1000MHz ARM, 500MHz core, 500MHz SDRAM, 2 overvolt
```

Figure 7-2. *Overclock options in Pi 2*

Overclocking the Pi with /boot/config.txt

We have learned how to overclock with `raspi-config`; however, it does not allow fine-tuning of the Pi which is to be overclocked. There is another way of overclocking the Pi, which we can do manually by changing a few parameters in`/boot/config.txt`.

Options in /boot/config.txt

We can add or change options in the `/boot/config.txt` file. If you have already used the `raspi-config` tool for overclocking and/or memory splitting, then you will find many of these options already present in the `/boot/config.txt` file.

- `arm_freq`: Frequency of a core in the ARM in MHz.
- `core_freq`: Frequency of GPU processor core in MHz.
- `h264_freq`: Frequency of hardware video block in MHz.
- `isp_freq`: Frequency of image sensor pipeline block in MHz.
- `v3d_freq`: Frequency of 3D block in MHz.
- `avoid_pwm_pll`: Don't dedicate a pll to PWM audio. This will reduce analogue audio quality slightly. The spare PLL allows the core_freq to be set independently from the rest of the GPU, allowing for more control over overclocking.

- `dram_freq`: Frequency of SDRAM in MHz.
- `over_voltage`: ARM/GPU core voltage adjust.
- `over_voltage_sdram_c`: SDRAM controller voltage adjust.
- `over_voltage_sdram_i`: SDRAM I/O voltage adjust.
- `over_voltage_sdram_p`: SDRAM phy voltage adjust.
- `force_turbo`: Disables dynamic cpufreq driver and minimum settings below. Enables H.264/V3D/ISP overclock options. May set warranty bit.
- `temp_limit`: Overheat protection. Sets clocks and voltages to default when the SoC reaches this Celsius value. Setting this higher than the default voids warranty.
- `gpu_mem`: GPU memory in megabyte. Sets the memory split between the ARM and GPU. ARM gets the remaining memory. No need to set this if the value is already set using raspiconfig.

/boot/config.txt Options for the Various Models of Pi

Every Pi is unique. The values of the options in `/boot/config.txt` for overclocking the Pi have to be customized for the individual Pi. There is no single set of values for these options which is the best fit for all Pis. I have learned this by trial and error. If the values you set for the overclocking options are not the best fit, the Pi will either be unstable or not boot up at all. In these cases, you can try changing the values of the overclocking options in `config.txt` on the microSD card by using some other computer and then boot up the Pi with the same card. Do it until you get a set of stable and optimal values for your Pi. In general, for any type of hardware (such as CPU and RAM), overclock settings depend on the individual IC.

We can find detailed explanations of the options above at the eLinux RPi configuration page (`http://elinux.org/RPiconfig`).

Options for Pi B and Pi B+

The following are the values of the overclocking options for Pi B and Pi B+:

```
arm_freq=950
core_freq=450
sdram_freq=500
gpu_mem=16
```

Options for Pi 2

The following are the values of the overclocking options for Pi 2:

```
arm_freq=1100
over_voltage=4
core_freq=550
sdram_freq=483
over_voltage_sdram_p=0
over_voltage_sdram_i=0
over_voltage_sdram_c=0
gpu_mem=16
force_turbo=1
avoid_pwm_pll=1
v3d_freq=450
h264_freq=0
isp_freq=0
avoid_safe_mode=1
```

Options for Pi 3

The following are the values of the overclocking options for Pi 3:

```
arm_freq=1350
over_voltage=6
temp_limit=80
core_freq=500
h264_freq=333
avoid_pwm_pll=1
gpu_mem=450
v3d_freq=550
sdram_freq=588
sdram_schmoo=0x02000020
over_voltage_sdram_p=6
over_voltage_sdram_i=4
over_voltage_sdram_c=4
force_turbo=1
```

If your Pi does not boot up or becomes unstable with the values listed above, try to tweak the values to get the best settings.

We cannot cover all possible combinations of working settings for stable overclock. Explore the following web links for information related to overclocking:

http://linuxonflash.blogspot.in/2015/02/a-look-at-raspberry-pi-2-performance.html
https://github.com/retropie/retropie-setup/wiki/Overclocking

After overclocking, use the `cat /proc/cpuinfo` command to check the processor speed for all models.

■ **Note** DO NOT insert the microSD card of an overclocked Pi into another Pi. Remember! Every piece of hardware has unique settings for overclocking. If you want to use the microSD card of an overclocked Pi for another Pi, disable overclock settings in /boot/config.txt and then use it with the other Pi.

EXERCISE

Overclock all the Raspberry Pis you are planning to use for building the mini-supercomputer cluster. This will boost the individual and the overall performance in terms of runtime.

Conclusion

In this chapter, we learned how to overclock the various models of Raspberry Pi. From the next chapter onward, we will start studying how to exploit the computational power of the cluster we built. We will write code in Python 3 with MPI4PY for demonstrating concepts in MPI and parallel programming.

CHAPTER 8

Parallel Programming in Python 3

In the last chapter, we learned how to overclock various models of Raspberry Pi to increase their computational power. In this chapter, we will learn how to write parallel programs with Python and MPI4PY. I prefer Python due to its simplicity, and the code in Python is less scary. We will explore MPI concepts and implement those in Python with MPI4PY.

The MPI concepts we will study and implement are as follows:

- MPI rank and processes
- Sending and receiving data
- Data tagging
- Broadcasting data
- Scattering and gathering data

Basics of MPI4PY

In the earlier part of this book, we studied a few MPI concepts. Let's study a few more in this chapter.

MPI uses the concept of ***Single-Program Multiple-Data*** (***SPMD***). The following are the key points in SPMD architecture:

- All processes (known as ranks) run the same code and each process accesses a different portion of data.
- All processes are launched simultaneously.

A parallel program is decomposed into separate processes, known as ranks. Each rank has its own address space, which requires partitioning data across ranks. Each rank holds a portion of the program's data in its own private memory. Ranks are numbered sequentially from 0 to n-1. The following diagram (Figure 8-1) depicts the multiple ranks running simultaneously.

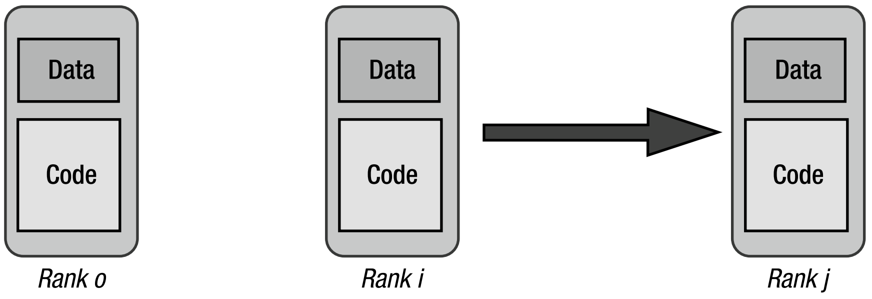


Figure 8-1. *Multiple ranks running simeltaneously*

In the next section, we will see a basic program with ranks.

Getting Started with MPI4PY

Let's get started with the simple Hello World! program in Python with MPI4PY.

Listing 8-1. prog01.py

```
from mpi4py import MPI
import sys

comm = MPI.COMM_WORLD
name = MPI.Get_processor_name()

sys.stdout.write("Hello World!")
sys.stdout.write(" Name: %s, My rank is %d\n" % (name, comm.rank))
```

In the code above (Listing 8-1), the statement `from mpi4py import MPI` imports the needed MPI4PY libraries. In Chapter 6 we studied the concept of communicators in MPI. `MPI.COMM_WORLD` is the communicator. It is used for all MPI communication between the processes running on the processes of the cluster. `Get_processor_name()` returns the hostname on which the current process is running. `comm.rank` is the rank of the current process. The following diagram (Figure 8-2) depicts `COMM_WORLD`.

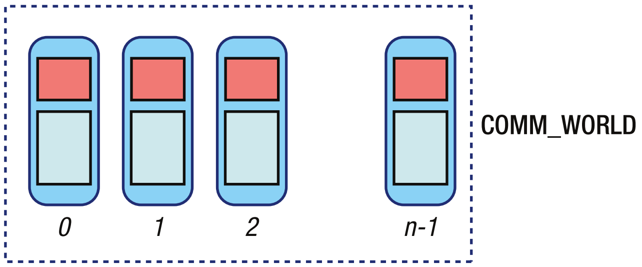


Figure 8-2. *COMM_WORLD*

You might have noticed that we are using `sys.stdout.write()` for printing on the console. This is because I want the code to be compatible for both interpreters of the Python programming language, `python` (the interpreter for Python 2) and `python3`. In this book, we won't be using any feature or programming construct specific to either interpreter. Thus, the code can be run using both interpreters.

We have started coding in this chapter, and the next chapters have a lot of code samples and exercises. It is a good idea to organize the code and the data in separate directories. Run the following commands in lxterminal one by one:

```
mpirun -hostfile myhostfile -np 4 mkdir /home/pi/book
mpirun -hostfile myhostfile -np 4 mkdir /home/pi/book/code
mpirun -hostfile myhostfile -np 4 mkdir /home/pi/book/code/chapter08
```

This will create the same directory structure on the all nodes of the mini-supercomputer. Now save the above code in a file called `prog01.py` in the `~/book/code/chapter08` directory. Copy the code file to that directory on all the nodes using `scp` as follows:

```
scp book/code/chapter08/prog01.py 192.168.0.2:/home/pi/book/code/chapter08/
scp book/code/chapter08/prog01.py 192.168.0.3:/home/pi/book/code/chapter08/
scp book/code/chapter08/prog01.py 192.168.0.4:/home/pi/book/code/chapter08/
```

Finally, run it with `mpirun` on `pi001` as follows:

```
mpirun -hostfile myhostfile -np 4 python3 ~/book/code/chapter08/prog01.py
```

The following is the output:

```
Hello World! Name: pi001, My rank is 0
Hello World! Name: pi002, My rank is 1
Hello World! Name: pi004, My rank is 3
Hello World! Name: pi003, My rank is 2
```

We have to follow the same steps for all the other code examples we will discuss in the rest of the chapter. Let me repeat them again in brief: create a Python code file in the `chapter08` directory, copy that file to the `chapter08` directory of all the nodes of the cluster, and finally use `mpirun` with the Python interpreter to execute the code.

Conditional Statements

We can use conditional statements in the MPI4PY code as follows (Listing 8-2):

Listing 8-2. prog02.py

```
from mpi4py import MPI
import sys
```

```
comm = MPI.COMM_WORLD
sys.stdout.write("My rank is: %d\n" % (comm.rank))

if comm.rank == 0:
    sys.stdout.write("Doing the task of Rank 0\n")

if comm.rank == 1:
    sys.stdout.write("Doing the task of Rank 1\n")
```

In this code, we're checking if the process rank is 0 or 1 and then printing more messages to the console. Run the program with `mpiexec` as follows:

```
mpirun -hostfile myhostfile -np 4 python3 ~/book/code/chapter08/prog02.py
```

The output of the program above (Listing 8-2) is as follows:

```
My rank is: 0
Doing the task of Rank 0
My rank is: 1
Doing the task of Rank 1
My rank is: 3
My rank is: 2
```

Checking the Number of Processes

Let's write the code (Listing 8-3) to display the rank and the number of MPI processes.

Listing 8-3. prog03.py

```
from mpi4py import MPI
import sys

comm = MPI.COMM_WORLD
rank = comm.rank
size = comm.size

sys.stdout.write("Rank: %d," % rank)
sys.stdout.write(" Process Count: %d\n" % size)
```

In the code above, `comm.size` gives the number of MPI processes running across the cluster. Run the code above with `mpiexec` as follows:

```
mpirun -hostfile myhostfile -np 4 python3 ~/book/code/chapter08/prog03.py
```

The output is as follows:

```
Rank: 0, Process Count: 4
Rank: 1, Process Count: 4
Rank: 2, Process Count: 4
Rank: 3, Process Count: 4
```

Sending and Receiving Data

Using `send()` and `receive()` for data transfer between processes is the simplest form of communication between processes. We can achieve one-to-one communication with this. The following diagram (Figure 8-3) explains this clearly.

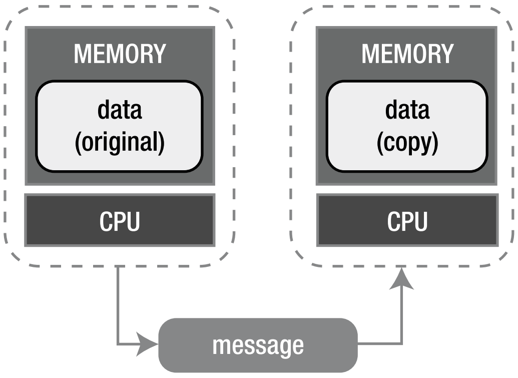


Figure 8-3. *One-to-one communication*

Let's see the code example (Listing 8-4) for the same.

Listing 8-4. prog04.py

```
from mpi4py import MPI
import time
import sys

comm = MPI.COMM_WORLD

rank = comm.rank
size = comm.size
name = MPI.Get_processor_name()

shared = 3.14
```

```
if rank == 0:
    data = shared
    comm.send(data, dest=1)
    comm.send(data, dest=2)
    sys.stdout.write("From rank %s, we sent %f\n" % (name, data))
    time.sleep(5)

elif rank == 1:
    data = comm.recv(source=0)
    sys.stdout.write("On rank %s, we received %f\n" % (name, data))

elif rank == 2:
    data = comm.recv(source=0)
    sys.stdout.write("On rank %s, we received %f\n" % (name, data))
```

In the code example above, we are sending data from the process with rank 0. The processes with rank 1 and 2 are receiving the data.

Let's run the program.

```
mpirun -hostfile myhostfile -np 4 python3 ~/book/code/chapter08/prog04.py
```

The output of the program above (Listing 8-4) is as follows:

```
On rank pi002, we received 3.140000
On rank pi003, we received 3.140000
From rank pi001, we sent 3.140000
```

Dynamically Sending and Receiving Data

Until now, we have written conditional statements for the processes to send and receive data. However, in large and distributed networks this type of data transfer is not always possible due to constant changes in the process count. Also, users might not want to hand-code the conditional statements.

The example below (Listing 8-5) demonstrates the concept of dynamic data transfer.

Listing 8-5. prog05.py

```
from mpi4py import MPI
import sys

comm = MPI.COMM_WORLD
rank = comm.rank
size = comm.size
name = MPI.Get_processor_name()

shared = (rank+1)*(rank+1)
```

```
comm.send(shared, dest=(rank+1) % size)
data = comm.recv(source=(rank-1) % size)

sys.stdout.write("Name: %s\n" % name)
sys.stdout.write("Rank: %d\n" % rank)
sys.stdout.write("Data %d came from rank: %d\n" % (data, (rank-1) % size))
```

In the code above (Listing 8-5), every process receives the data from the earlier process. This goes on till the end and wraps around so that the first process receives the data from the last process.

Let's run the code.

```
mpirun -hostfile myhostfile -np 4 python3 ~/book/code/chapter08/prog05.py
```

The output of the code is as follows:

```
Name: pi001
Rank: 0
Data 16 came from rank: 3
Name: pi002
Rank: 1
Data 1 came from rank: 0
Name: pi003
Rank: 2
Data 4 came from rank: 1
Name: pi004
Rank: 3
Data 9 came from rank: 2
```

As discussed earlier, the process with rank 0 (the first process) receives the data from the process with rank 3 (the last process).

Data Tagging

In the earlier example (Listing 8-5), we studied how to send and receive data with MPI. This raises a basic question for curious programmers: how do we exchange multiple data items between processes? We can send multiple data items from one process to another. However, at the receiving end, we will encounter problems in distinguishing one data item from another. The solution for this is tagging. Have a look at the code example (Listing 8-6) below.

Listing 8-6. prog06.py

```
from mpi4py import MPI
import sys
```

```
comm = MPI.COMM_WORLD
rank = comm.rank
size = comm.size
name = MPI.Get_processor_name()

if rank == 0:
    shared1 = {'d1': 55, 'd2': 42}
    comm.send(shared1, dest=1, tag=1)

    shared2 = {'d3': 25, 'd4': 22}
    comm.send(shared2, dest=1, tag=2)

if rank == 1:
    receive1 = comm.recv(source=0, tag=1)
    sys.stdout.write("d1: %d, d2: %d\n" % (receive1['d1'], receive1['d2']))
    receive2 = comm.recv(source=0, tag=2)
    sys.stdout.write("d3: %d, d4: %d\n" % (receive2['d3'], receive2['d4']))
```

In the example above, we are sending two different dictionaries `shared1` and `shared2` from the process with rank 0 to the process with rank 1. At the source, `shared1` is tagged with 1 and `shared2` is tagged with 2. At the destination, we can distinguish the different data items from the tags associated with them.

Run the code above (Listing 8-6) with the following command:

```
mpirun -hostfile myhostfile -np 4 python3 ~/book/code/chapter08/prog06.py
```

The output is as follows:

```
d1: 55, d2: 42
d3: 25, d4: 22
```

Data tagging gives programmers more control over the flow of data. When multiple data are exchanged between processes, data tagging is a must.

Data Broadcasting

When data is sent from a single process to all the other processes then it is known as broadcasting. Consider the following code (Listing 8-7):

Listing 8-7. prog07.py

```
from mpi4py import MPI
import sys

comm = MPI.COMM_WORLD
rank = comm.rank
```

```
if rank == 0:
    data = {'a': 1, 'b': 2, 'c': 3}
else:
    data = None

data = comm.bcast(data, root=0)
sys.stdout.write("Rank: %d, Data: %d, %d, %d\n"
                 % (rank, data['a'], data['b'], data['c']))
```

In the code above (Listing 8-7), in the `if` statement we are assigning a dictionary to data only if the rank of process is 0. `bcast()` broadcasts the data to all the processes.

Run the program.

```
mpirun -hostfile myhostfile -np 4 python3 ~/book/code/chapter08/prog07.py
```

The output is as follows:

```
Rank: 0, Data: 1, 2, 3
Rank: 1, Data: 1, 2, 3
Rank: 2, Data: 1, 2, 3
Rank: 3, Data: 1, 2, 3
```

Data Scattering

In broadcasting, we send the same data to all the processes. In scattering, we send the different chunks of the data to all the processes. For example, we have a list with four items. In broadcasting, we send all these four items to all the processes, whereas in scattering, we send the items of the list to the processes, such that every process receives an item from the list. The following program (Listing 8-8) demonstrates this.

Listing 8-8. prog08.py

```
from mpi4py import MPI
import sys

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

if rank == 0:
    data = [x for x in range(0,size)]
    sys.stdout.write("We will be scattering: ")
    sys.stdout.write(" ".join(str(x) for x in data))
    sys.stdout.write("\n")
else:
    data = None

data = comm.scatter(data, root=0)
sys.stdout.write("Rank: %d has data: %d\n" % (rank, data))
```

In the code above (Listing 8-8), we are creating a list with the name data which has a number of elements equal to the process count in the cluster. `scatter()` is used to scatter the data to all the processes.

Run the code.

```
mpirun -hostfile myhostfile -np 4 python3 ~/book/code/chapter08/prog08.py
```

The following is the output:

```
Rank: 1 has data: 1
We will be scattering: 0 1 2 3
Rank: 0 has data: 0
Rank: 2 has data: 2
Rank: 3 has data: 3
```

As we can see, each process receives an item from the list. The limitation of `scatter()` is that the size of the data list we are scattering must not exceed the number of processes.

Data Gathering

The idea of gathering the data is opposite of scattering. The master process gathers all the data processed by the other processes.

The program below (Listing 8-9) demonstrates the `gather()` method.

Listing 8-9. prog09.py

```
from mpi4py import MPI
import sys

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

if rank == 0:
    data = [x for x in range(0,size)]
    sys.stdout.write("We will be scattering: ")
    sys.stdout.write(" ".join(str(x) for x in data))
    sys.stdout.write("\n")
else:
    data = None

data = comm.scatter(data, root=0)
sys.stdout.write("Rank: %d has data: %d\n" % (rank, data))
data *= data

newData = comm.gather(data, root=0)
```

```
if rank == 0:
    sys.stdout.write("We have gathered: ")
    sys.stdout.write(" ".join(str(x) for x in newData))
    sys.stdout.write("\n")
```

In the program above (Listing 8-9), the master process scatters the list of numbers. All the MPI processes receive an element from the list (the size of the list equals the number of MPI processes). Each process performs an operation of the element it receives. In our case, it is the calculation of the square of the number. However, in real-world supercomputing, the operation could be quite complex.

Once the operation completes, the master process gathers all the processed elements in a new list.

Run the code.

```
mpirun -hostfile myhostfile -np 4 python3 ~/book/code/chapter08/prog09.py
```

The output is as follows:

```
Rank: 1 has data: 1
Rank: 3 has data: 3
We will be scattering: 0 1 2 3
Rank: 0 has data: 0
We have gathered: 0 1 4 9
Rank: 2 has data: 2
```

Conclusion

In this chapter, we were introduced to the MPI4PY library for Python. We learned and experimented with various interesting concepts in parallel programming with MPI4PY. In the next chapter, we will get started with the SciPy stack in Python 3 with Raspberry Pi.

CHAPTER 9

Introduction to SciPy Stack and Symbolic Programming

In the last chapter, we learned how to use the Raspberry Pi cluster we built for parallel programming with MPI4PY and Python 3. In this chapter, we will be introduced to the SciPy stack and install it on the Pi. We will also get started with symbolic programming with SymPy.

The Scientific Python Stack

SciPy (short for Scientific Python) is an open-source library for scientific and technical computing in Python.

SciPy has modules for numerical operations, ordinary differential equation solvers, fast Fourier transforms, optimization, linear algebra, integration, interpolation, signal processing, and image processing. SciPy is extensively used by scientific, mathematical, and engineering communities around the world. There are many other libraries which use core modules of SciPy and NumPy for operations. OpenCV and SciKit are the best examples of other major libraries using NumPy and/or SciPy.

The SciPy stack has the following components:

- NumPy is a library for numerical computations. It provides all the basic data types required for numeric and scientific computing.
- SciPy Library has many modules for scientific programming.
- Matplotlib is used for data visualization.
- SymPy is used for Symbolic Programming.
- IPython is an advanced Python interpreter with added features.
- Pandas is used for data analysis.
- Nose is used for automating test cases.

The following figure (Figure 9-1) summarizes the role of the Python SciPy stack in the world of scientific computing.

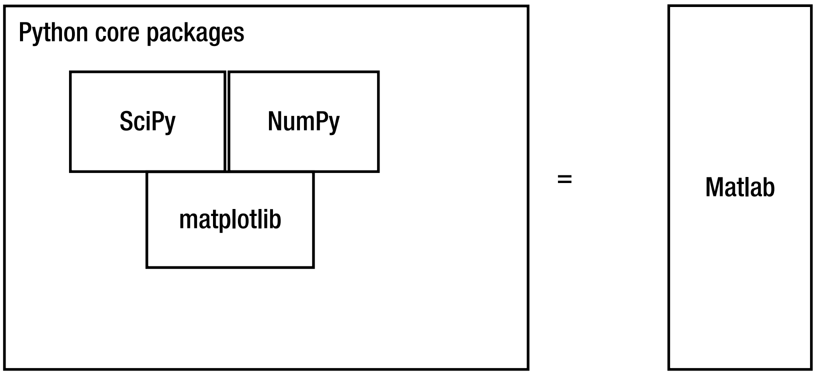


Figure 9-1. *The components of the SciPy stack*

In this book, we will study NymPy, SciPy Library, Matplotlib, and SymPy.

Installation of the SciPy Stack

The best way to install the SciPy stack on Raspberry Pi is to use `apt-get` and `pip`.

First, upgrade `pip` with the following command:

```
sudo python3 -m pip install --upgrade pip
```

Install SymPy with the following command:

```
sudo pip3 install sympy
```

The rest of the components of the SciPy stack can conveniently be installed with the `apt-get` utility as follows:

```
sudo apt-get install python3-matplotlib -y
sudo apt-get install python3-scipy -y
sudo apt-get install python3-numpy -y
```

This installs the required components of the SciPy stack.

SymPy

The SymPy website says:

> *SymPy is a Python library for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible in order to be comprehensible and easily extensible. SymPy is written entirely in Python.*

SymPy is licensed under BSD and free. It has been written entirely in Python. It depends on `mpmath` and is lightweight in nature as there are no other dependencies.

Getting Started

Let's get started with SymPy. Run the following commands to create and navigate to the directory for the chapter:

```
cd ~
cd book
cd code
mkdir chapter09
cd chapter09
```

We will keep the code for this chapter in the directory `chapter09` and will continue this practice for the rest of the book, i.e. creating directories by chapter for organizing the code.

I assume that you have some basic knowledge of mathematics and calculus, so I do not have to explain the basics when explaining the code. Let's see a simple example (Listing 9-1) of the concept of symbolic computation.

Listing 9-1. prog01.py

```
import math
import sympy
print(math.sqrt(9))
print(math.sqrt(8))
print(sympy.sqrt(9))
print(sympy.sqrt(8))
```

Run the code above (Listing 9-1). The following is the output:

```
3.0
2.8284271247461903
3
2*sqrt(2)
```

In the output above, we can see that the `math.sqrt()` method directly produces the results in numeric format, whereas the `sympy.sqrt()` method produces the result in numeric format only in the case where it is an integer. Rather than producing a fractional value, it keeps `sqrt(2)` as it is. This way, we can symbolically compute a lot of mathematical equations. Equipped with this knowledge of the concept of symbolic mathematics, let's go deeper into SymPy with Python 3.

Symbols

Let's study the concept of symbols. Symbols are analogous to variables in mathematics. We can use them for evaluations in equations and expressions. They can also be used to solve equations. The `sympy.symbols()` method converts a string into symbolic variables as follows (Listing 9-2):

Listing 9-2. prog02.py

```
from sympy import *

x = symbols('x')
sigma, y = symbols('sigma y')
print(x, y, sigma)
```

The output is as follows:

```
x y sigma
```

The above code (Listing 9-2) demonstrates that the `symbols()` method can accept a string where the tokens are separated by whitespace as an argument.

Let's see one more example (Listing 9-3) which demonstrates the evaluation of an expression with symbols.

Listing 9-3. prog03.py

```
from sympy import *

x = symbols('x')
expr = x + 1
print(expr.subs(x, 3))
```

This will substitute 3 in place of `x` in the expression and evaluate that. The code (Listing 9-3) produces `4` as output.

We can substitute for multiple symbols as follows (Listing 9-4):

Listing 9-4. prog04.py

```
from sympy import *

x, y = symbols('x y')
expr = x + y
print(expr.subs({x:3, y:2}))
```

Here, we are substituting for multiple symbols at once. Run the code and check the output.

Converting Strings to SymPy Expressions

We can convert strings to SymPy expressions. Just as in Python, we can use the ** operator for the exponent. The following code (Listing 9-5) shows this:

Listing 9-5. prog05.py

```
from sympy import *

str = "x**3 + 4*y - 1/5"
expr = sympify(str)
print(expr)
```

The `sympify()` method converts a string to a SymPy expression.

We can also use the `evalf()` method for evaluating expressions to a floating point number. It has a default precision of 15 digits after the decimal; however, we can pass the precision as an argument. The following (Listing 9-6) shows the example use cases of the `evalf()` method:

Listing 9-6. prog06.py

```
from sympy import *

expr = sqrt(10)
print(expr.evalf())

print(pi.evalf(20))

x = symbols('x')
expr = sin(2*x)
print(expr.evalf(subs={x: 2.4}))
The output is as follows,
3.16227766016838
3.1415926535897932385
-0.996164608835841
Printing in SymPy
```

Sympy's Printing Functionality

SymPy has many printers. In any environment, use of `init_session()` method at the command prompt will start an interactive session. The following is an example of a sample interactive session. The commands I typed into the console are highlighted in bold.

```
pi@pi001:~/book/code/chapter09 $ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
```

```
Type "help", "copyright", "credits" or "license" for more information.
>>> from sympy import *
>>> init_session()
Python console for SymPy 1.0 (Python 3.4.2-32-bit) (ground types: python)
```

These commands were executed:

```
>>> from __future__ import division
>>> from sympy import *
>>> x, y, z, t = symbols('x y z t')
>>> k, m, n = symbols('k m n', integer=True)
>>> f, g, h = symbols('f g h', cls=Function)
>>> init_printing()

Documentation can be found at http://docs.sympy.org/1.0/

>>> Integral(sqrt(1/x), x)
⌠
⎮        ___
⎮       ╱ 1
⎮      ╱  ─  dx
⎮ ╲╱      x
⌡
>>> sqrt(x)
√x
>>> (sqrt(x) + sqrt(y))**2
         2
(√x + √y)
>>>
```

This is how we can print expressions in a nice format in the interactive console.

Simplification in SymPy

We can use the simplify() method to simplify mathematical expressions to the best possible extent. A large number of expressions are covered under this. The following (Listing 9-8) is an example of this:

Listing 9-8. prog08.py

```
from sympy import *
x = symbols('x')
print(simplify(sin(x)**2 + cos(x)**2))
print(simplify((x**3 + x**2 - x - 1)/(x**2 + 2*x + 1)))
print(simplify(gamma(x)/gamma(x - 2)))
```

The simplified output is as follows:

```
1
x - 1
(x - 2)*(x - 1)
```

There are more simplification methods in SymPy. We can use `expand()` for a polynomial expression's expansion as shown below (Listing 9-9).

Listing 9-9. prog09.py

```
from sympy import *
x, y = symbols('x y')
print(expand((x + y)**2))
print(expand((x + 3)*(y + 5)))
```

The following is the expanded output:

```
x**2 + 2*x*y + y**2
x*y + 5*x + 3*y + 15
```

Similarly, we can use the `factor()` method (Listing 9-10) for finding irreducible factors of polynomials.

Listing 9-10. prog10.py

```
from sympy import *
x = symbols('x')
print(factor(x**3 - x**2 + x))
```

The output is as follows:

```
x*(x**2 - x + 1)
```

Calculus

We can even use SymPy for calculus. We can use the `diff()` method to calculate the derivatives as follows (Listing 9-11):

Listing 9-11. prog11.py

```
from sympy import *
x = symbols('x')
print(diff(x**3 - x**2 + x, x))
print(diff(x**5, x))
print(diff(sin(x), x))
```

The output is:

```
3*x**2 - 2*x + 1
5*x**4
cos(x)
```

We can also find the higher order derivative of the expressions as follows (Listing 9-12):

Listing 9-12. prog12.py

```
from sympy import *
x = symbols('x')
print(diff(10*x**4, x, x, x))
print(diff(10*x**4, x, 3))
```

The output is as follows:

```
240*x
240*x
```

We can also calculate integrals with SymPy using the `integrate()` method. The following code (Listing 9-13) demonstrates this:

Listing 9-13. prog13.py

```
from sympy import *
x = symbols('x')
print(integrate(sin(x), x))
```

The output is as follows:

```
-cos(x)
```

We can also have integration with limits as follows (Listing 9-14):

Listing 9-14. prog14.py

```
from sympy import *
x = symbols('x')
print(integrate(exp(-x), (x, 0, oo)))
```

Here, we are integrating the exponent of -x from zero to infinity (denoted by oo). Run this and check the output. We can also calculate multiple integrals with multiple limits as follows (Listing 9-15):

Listing 9-15. prog15.py

```
from sympy import *
x, y = symbols('x y')
print(integrate(exp(-x)*exp(-y), (x, 0, oo), (y, 0, oo)))
```

Run the code (Listing 9-15) and verify the output by carrying out the integration manually.

■ **Note** SymPy is really a big topic. It cannot be covered completely in a single chapter. I recommend readers explore more on the websites `http://docs.sympy.org` and `www.sympy.org`.

Conclusion

In this chapter, we got started with SymPy and learned how to perform symbolic computation in Python. In the next chapter, we will get started with NumPy and Matplotlib.

CHAPTER 10

Introduction to NumPy

In the last chapter, we learned how to install the SciPy stack and how to use SymPy for symbolic computation with Python 3. In this chapter, we will be introduced to the NumPy library, and we will study the basics of NumPy. We will also learn the basics of plotting and visualizing data with Matplotlib. So let's begin the exciting journey into the world of scientific computing by learning the foundations of NumPy.

Basics of NumPy

NumPy is short for Numeric(al) Python. Its website (`http://www.numpy.org`) says:

NumPy is the fundamental package for scientific computing with Python

Its features are as follows:

- It has a powerful custom N-dimensional array object for efficient and convenient representation of data.
- It has tools for integration with other programming languages used for scientific programming like C/C++ and FORTRAN.
- It is used for mathematical operations like linear algebra, matrix operations, image processing, and signal processing.

Jupyter

Until now, we have been saving our code in `.py` files and running it with the Python 3 interpreter. In this chapter, we will use a tool known as **Jupyter**, which is an advanced web-based tool for interactive coding in the programming languages Julia, Python, and R.

Ju*lia* + **Pyt***hon* + **R** = *Jupyter*

It saves Python 3 (or any other supported languages like R and Julia) code and the result in an interactive format called a notebook. Jupyter uses the IPython kernel for Python 2 and Python 3. IPython is an advanced interactive shell for Python, which has visualization capabilities. Project Jupyter is a spin-off of IPython.

Jupyter and IPython have the following features:

- Interactive terminal- and Qt-based shells
- Browser-based notebooks for the support of code and interactive visualizations
- Support for parallel computing

It can be installed easily with the following commands:

```
sudo pip3 install --upgrade pip
sudo pip3 install jupyter
```

This installs Jupyter and all of its dependencies on Raspberry Pi.

Jupyter Notebooks

Jupyter notebooks are documents produced by the Jupyter Notebook App, which have Python code and Rich Text elements like paragraphs, equations, figures, links, and interactive visualizations. Notebooks have human-readable components and machine-readable (executable) components.

Let's get started with NumPy and Jupyter notebooks now. Open lxterminal and run the following sequence of commands:

```
cd ~
cd book
cd code
mkdir chapter10
cd chapter10
```

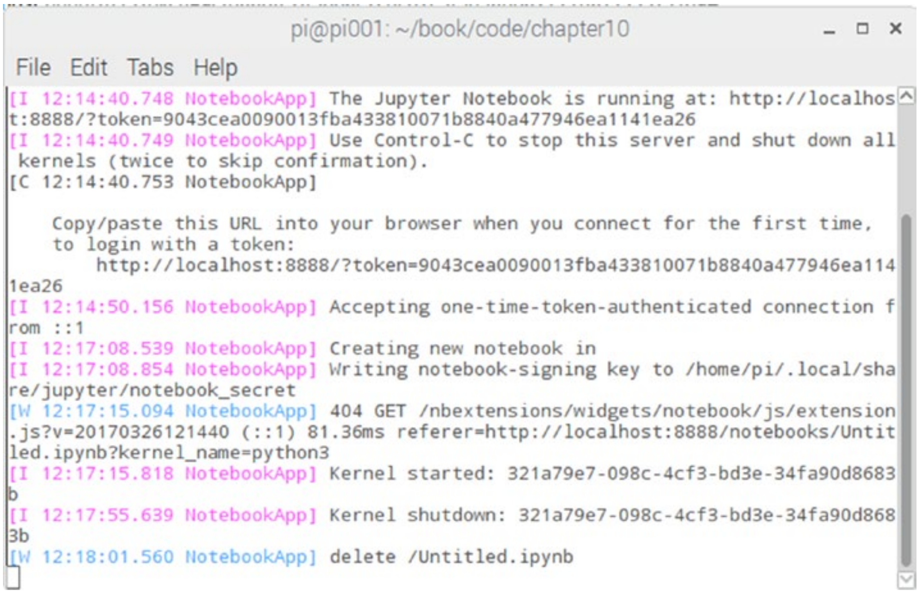
This will create and navigate to the directory corresponding to the current chapter, `chapter10`.

Now, this is our notebook startup folder, so let's launch the notebook from here with the following command:

```
jupyter notebook
```

It will start the Jupyter Notebook App and a browser window (Chromium browser in the latest releases of Raspbian) will open.

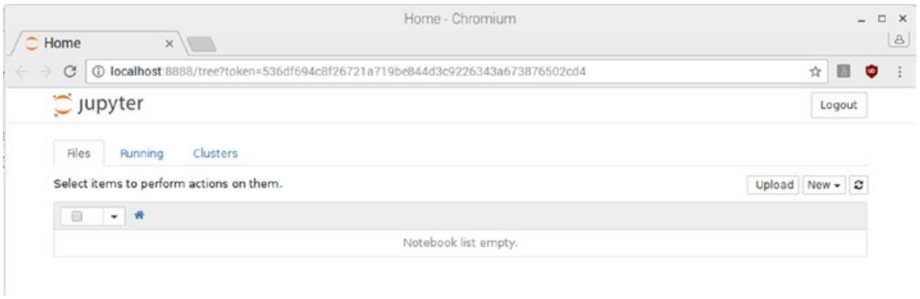
The following (Figure 10-1) is a screenshot of the console when the Jupyter notebook starts:



```
[I 12:14:40.748 NotebookApp] The Jupyter Notebook is running at: http://localhos
t:8888/?token=9043cea0090013fba433810071b8840a477946ea1141ea26
[I 12:14:40.749 NotebookApp] Use Control-C to stop this server and shut down all
 kernels (twice to skip confirmation).
[C 12:14:40.753 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://localhost:8888/?token=9043cea0090013fba433810071b8840a477946ea114
1ea26
[I 12:14:50.156 NotebookApp] Accepting one-time-token-authenticated connection f
rom ::1
[I 12:17:08.539 NotebookApp] Creating new notebook in
[I 12:17:08.854 NotebookApp] Writing notebook-signing key to /home/pi/.local/sha
re/jupyter/notebook_secret
[W 12:17:15.094 NotebookApp] 404 GET /nbextensions/widgets/notebook/js/extension
.js?v=20170326121440 (::1) 81.36ms referer=http://localhost:8888/notebooks/Untit
led.ipynb?kernel_name=python3
[I 12:17:15.818 NotebookApp] Kernel started: 321a79e7-098c-4cf3-bd3e-34fa90d8683
b
[I 12:17:55.639 NotebookApp] Kernel shutdown: 321a79e7-098c-4cf3-bd3e-34fa90d868
3b
[W 12:18:01.560 NotebookApp] delete /Untitled.ipynb
```

Figure 10-1. *Jupyter Notebook App console*

The following (Figure 10-2) is a screenshot of the Chromium browser window tab running the notebook app:



Select items to perform actions on them.

Figure 10-2. *Jupyter Notebook App running in Chromium*

In the upper right part of the browser window click **New** and then in the subsequent dropdown select **Python 3**. See the following (Figure 10-3) screenshot:

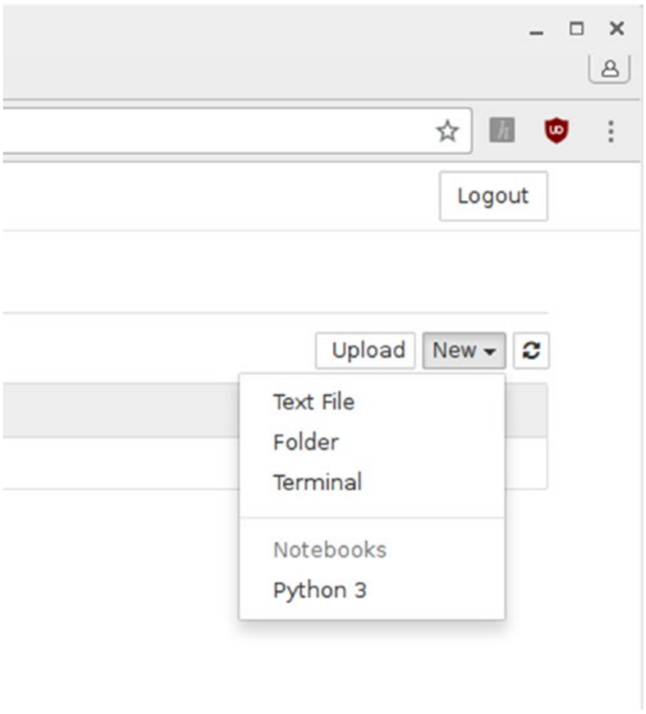


Figure 10-3. *New Python 3 notebook*

It will open a new notebook tab (Figure 10-4) in the same browser window.

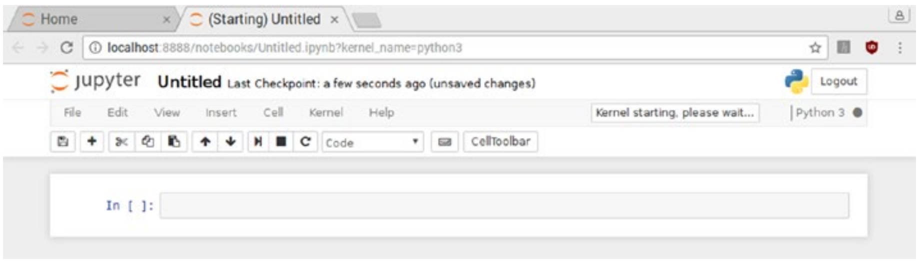

Home
(Starting) Untitled
localhost:8888/notebooks/Untitled.ipynb?kernel_name=python3
jupyter Untitled Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Kernel starting, please wait...
Python 3
Code
CellToolbar
In [ ]:

Figure 10-4. *Python 3 notebook tab*

Change the name of the Jupyter notebook to `Chapter10_Practice` as shown in the screenshot (Figure 10-5) below.

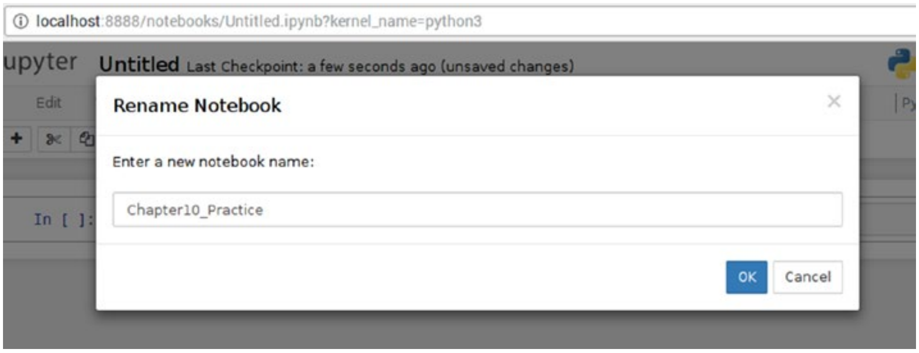

localhost:8888/notebooks/Untitled.ipynb?kernel_name=python3
upyter Untitled Last Checkpoint: a few seconds ago (unsaved changes)
Edit
In [ ]:
Rename Notebook
Enter a new notebook name:
Chapter10_Practice
OK
Cancel

Figure 10-5. *Renaming the notebook*

The Notebook app will show an instance of a new notebook with the updated name, with status as "running," as shown in the screenshot (Figure 10-6) below.

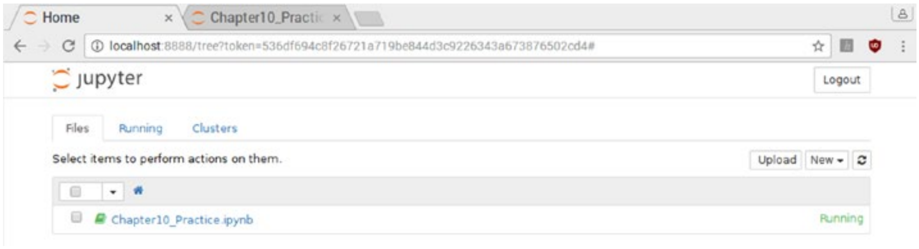
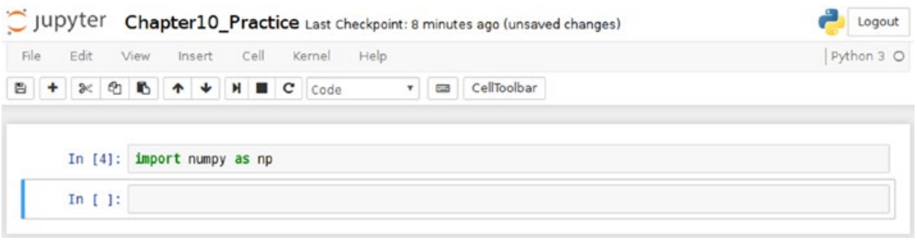

Home
Chapter10_Practic
localhost:8888/tree?token=536df694c8f26721a719be844d3c9226343a673876502cd4#
jupyter
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
Chapter10_Practice.ipynb
Running

Figure 10-6. *Notebook running*

Now, if you check the `Chapter10` directory, you will find a file `Chapter10_Practice.ipynb` corresponding to the notebook.

In the menubar at the top of the window, there are options like you would have in any other IDE—for example, save, copy, paste, and run.

Type `import numpy as np` in the first cell and click the **Run** button. The control will automatically create the next text cell and focus the cursor on it as shown below (Figure 10-7).



```
In [4]: import numpy as np
In [ ]:
```

Figure 10-7. *Working with Python 3 code*

We have just imported NumPy to our notebook, so we do not have to import again. Also, we can edit the previous cells of the notebook too. At the type of execution, if the interpreter highlights a mistake in the syntax, we can fix it by editing any of the cells. We will learn more about Jupyter as we go forward in scientific computing.

The N-Dimensional Array (ndarray)

The most powerful construct of NumPy is the N-Dimensional array (`ndarray`). `ndarray` provides a generic container for multi-dimensional homogeneous data. Homogeneous means the data items in an `ndarray` are of the same data-type. Let's see an example of various `ndarray` type variables in Python. Type the following code in the notebook:

```
x = np.array([1, 2, 3], np.int16)
y = np.array([[0, 1, 2], [3, 4, 5]], np.int32)
z = np.array([[[0, 1, 2], [2, 3, 4], [4, 5, 6]],[[1, 1, 1], [0, 0, 0],
    [1, 1, 1]]], np.float16)
```

Congrats, we have just created one-, two-, and three-dimensional `ndarray` objects. We can verify that by running the following code in Jupyter notebook:

```
print(x.shape)
print(y.shape)
print(z.shape)
```

The output is:

```
(3,)
(2, 3)
(3, 3, 3)
```

The indexing scheme of the `ndarray` in NumPy is the same as in C language i.e. the first element is indexed from 0. The following line prints the value of an element in the second row and third column in an `ndarray`.

We can slice the arrays as follows:

```
print(z[:,1])
```

The following is the output:

```
[[ 2.  3.  4.]
 [ 0.  0.  0.]]
```

ndarray Attributes

The following is a demonstration of important attributes of the `ndarray`:

```
print(z.shape)
print(z.ndim)
print(z.size)
print(z.itemsize)
print(z.nbytes)
print(z.dtype)
```

The following is the output:

```
(2, 3, 3)
3
18
2
36
float16
```

Here is what is happening:

- `ndarray.shape` returns the tuple of array dimensions.
- `ndarray.ndim` returns the number of array dimensions.
- `ndarray.size` returns the number of elements in the array.
- `ndarray.itemsize` returns the length of one array element in bytes.
- `ndarray.nbytes` returns the total bytes consumed by the elements of the array. It is the result of the multiplication of attributes `ndarray.size` and `ndarray.itemsize`.
- `ndarray.dtype` returns the data type of items in the array.

Data Types

Data types are used for denoting the type and size of the data. For example, int16 means 16-bit signed integer and uint8 means 8-bit unsigned integer. NumPy supports a wide range of data types. Please explore the data type objects webpage (https://docs.scipy.org/doc/numpy/reference/arrays.dtypes.html) for more details about data types.

■ **Note** To see an exhaustive list of various data types supported by NumPy visit https://docs.scipy.org/doc/numpy/user/basics.types.html.

Equipped with the basics of the ndarray construct in NumPy, we will get started with array creation routines in the next section.

Array Creation Routines

Let's create ndarrays with built-array creation routines. We are also going to use matplotlib to visualize the arrays created. We will use matplotlib for visualizing the data as and when needed. The last chapter of this book is dedicated to matplotlib, where we will learn more about it.

The first method for creating ndarrays is ones(). As you must have guessed, it creates arrays where all the values are ones. Run the following statements in the notebook to see a demonstration of the method:

```
a = np.ones(5)
print(a)
b = np.ones((3,3))
print(b)
c = np.ones((5, 5), dtype=np.int16)
print(c)
```

Similarly we have the np.zeros() method for generating multidimensional arrays of zeros. Just run the examples above by passing the same set of arguments to the method np.zeros(). The np.eye() method is used for creating diagonal matrices. We can also specify the index of the diagonals. Run the following examples in the notebook for a demonstration:

```
a = np.eye(5, dtype=np.int16)
print(a)
b = np.eye(5, k=1)
print(b)
c = np.eye(5, k=-1)
print(c)
```

`np.arange()` returns a list of evenly spaced numbers in a specified range. Try the following examples in the notebook:

```
a = np.arange(5)
print(a)
b = np.arange(3, 25, 2)
print(b)
```

The output is as follows:

```
[0 1 2 3 4]
[ 3  5  7  9 11 13 15 17 19 21 23]
```

Let's have some fun with matplotlib now. Run the following example in the notebook:

```
import matplotlib.pyplot as plt
x = np.arange(10)
y = np.arange(10)
print(x)
print(y)
plt.plot(x, y, 'o')
plt.show()
```

In the example above, we are graphically representing the `ndarrays`. The text result is as follows:

```
[0 1 2 3 4 5 6 7 8 9]
[0 1 2 3 4 5 6 7 8 9]
```

The graphical output (Figure 10-8) is as follows:

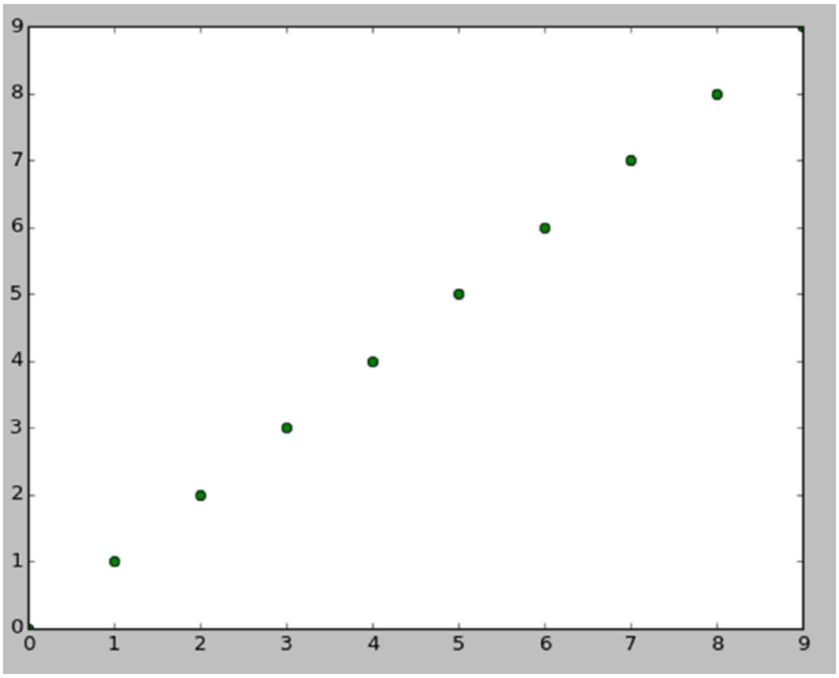


Figure 10-8. *Graphical Output of arange()*

With the statement `import matplotlib.pyplot as plt`, we are importing the ***pyplot*** module of the `myplotlib` library. `plt.plot()` prepares the plot graph for display and `plt.show()` displays it on the screen.

We have a similar method, `np.linspace()`, for generating a linear array. The major difference between `linspace()` and `arange()` is that `linspace()` accepts the number of samples to be generated as an argument rather than stepsize. The following code snippet shows the way `linspace()` generates the data:

```
N = 8
y = np.zeros(N)
x1 = np.linspace(0, 10, N)
x2 = np.linspace(0, 10, N)
plt.plot(x1, y - 0.5, 'o')
plt.plot(x2, y + 0.5, 'o')
plt.ylim([-1, 1])
plt.show()
```

`plt.ylim()` specifies the limit of the **Y co-ordinate** on the graph. The following (Figure 10-9) is the output:

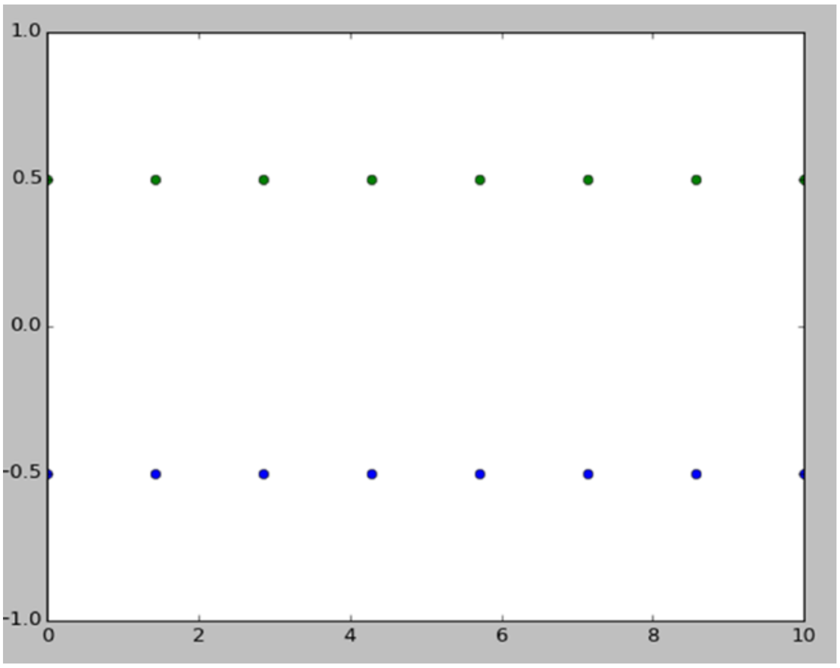


Figure 10-9. *linspace() graph*

Similarly, we have the `logspace()` method which generates the array value on a logarithmic scale. The following code demonstrates that:

```
N = 64
x = np.linspace(1, N, N)
y = np.logspace(0.1, 1, N)
plt.plot(x, y, 'o')
plt.show()
```

Run the statements in the notebook and it generates the following (Figure 10-10) output:

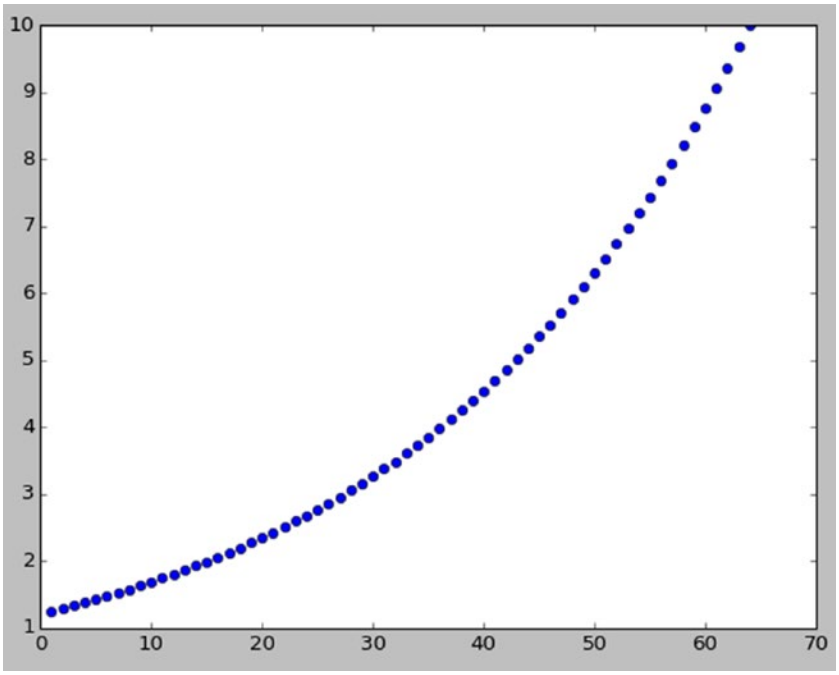


Figure 10-10. *linspace() versus logspace() graph*

Matrix and Linear Algebra

NumPy has routines for matrix creation. Let's see a few examples. `np.matrix()` interprets the given data as a matrix. The following are examples:

```
a = np.matrix('1 2; 3 4')
b = np.matrix([[1, 2], [3, 4]])
print(a)
print(b)
```

Run the code in the notebook and you will see that both the examples return the matrices even though the input was in a different format.

```
[[1 2]
 [3 4]]
[[1 2]
 [3 4]]
```

np.bmat() builds a matrix from arrays or sequences.

```
A = np.mat('1 1; 1 1')
B = np.mat('2 2; 2 2')
C = np.mat('3 4; 5 6')
D = np.mat('7 8; 9 0')
a = np.bmat([[A, B], [C, D]])
print(a)
```

The code above returns a matrix combined from all the sequences. Run the code in the notebook to see the output as follows:

```
[[1 1 2 2]
 [1 1 2 2]
 [3 4 7 8]
 [5 6 9 0]]
```

np.matlib.zeros() and np.matlib.ones() return the matrices of zeros and ones respectively. np.matlib.eye() returns a diagonal matrix. np.matlib.identity() returns a square identity matrix of a given size. The following code example demonstrates these methods:

```
from numpy.matlib import *
a = zeros((3, 3))
print(a)
b = ones((3, 3))
print(b)
c = eye(3)
print(c)
d = eye(5, k=1)
print(d)
e = eye(5, k=-1)
print(e)
f = identity(4)
print(f)
```

The rand() and randn() methods return matrices with random numbers.

```
a = rand((3, 3))
b = randn((4, 4))
print(a)
print(b)
```

Let's study a few methods related to linear algebra (matrix operations). We have the `dot()` method, which calculates the dot product of two arrays, whereas `vdot()` calculates the dot product of two vectors. `inner()` calculates the inner product of two arrays. `outer()` calculates the outer product of two vectors. The following code example demonstrates all these methods:

```
a = [[1, 0], [0, 1]]
b = [[4, 1], [2, 2]]
print(np.dot(a, b))
print(np.vdot(a, b))
print(np.inner(a, b))
print(np.outer(a, b))
```

The output is as follows:

```
[[4 1]
 [2 2]]
6
[[4 2]
 [1 2]]
[[4 1 2 2]
 [0 0 0 0]
 [0 0 0 0]
 [4 1 2 2]]
```

Trigonometric Methods

Let's visualize trigonometric methods. We will visualize `sin()`, `cos()`, `tan()`, `sinh()`, and `cosh()` methods with matplotlib. The following example demonstrates the usage of these methods:

```
x = np.linspace(-np.pi, np.pi, 201)
plt.plot(x, np.sin(x))
plt.xlabel('Angle in radians')
plt.ylabel('sin(x)')
plt.show()
```

The output (Figure 10-11) is as follows:

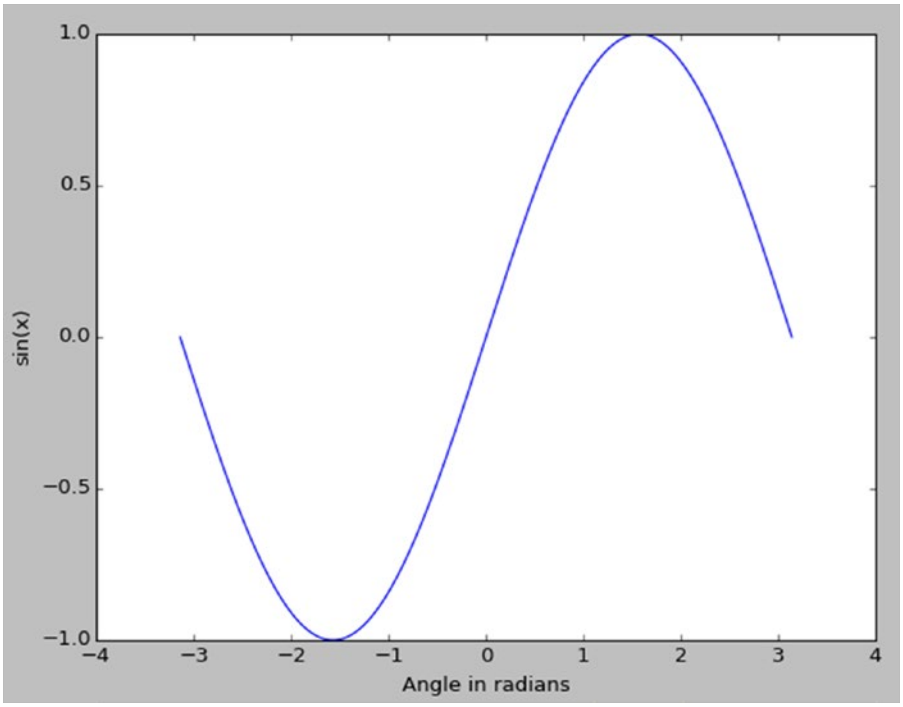


Figure 10-11. *Graph for sin(x)*

The following code examples demonstrate usage of `cos()`:

```
x = np.linspace(-np.pi, 2*np.pi, 401)
plt.plot(x, np.cos(x))
plt.xlabel('Angle in radians')
plt.ylabel('cos(x)')
plt.show()
```

The following (Figure 10-12) is the output:

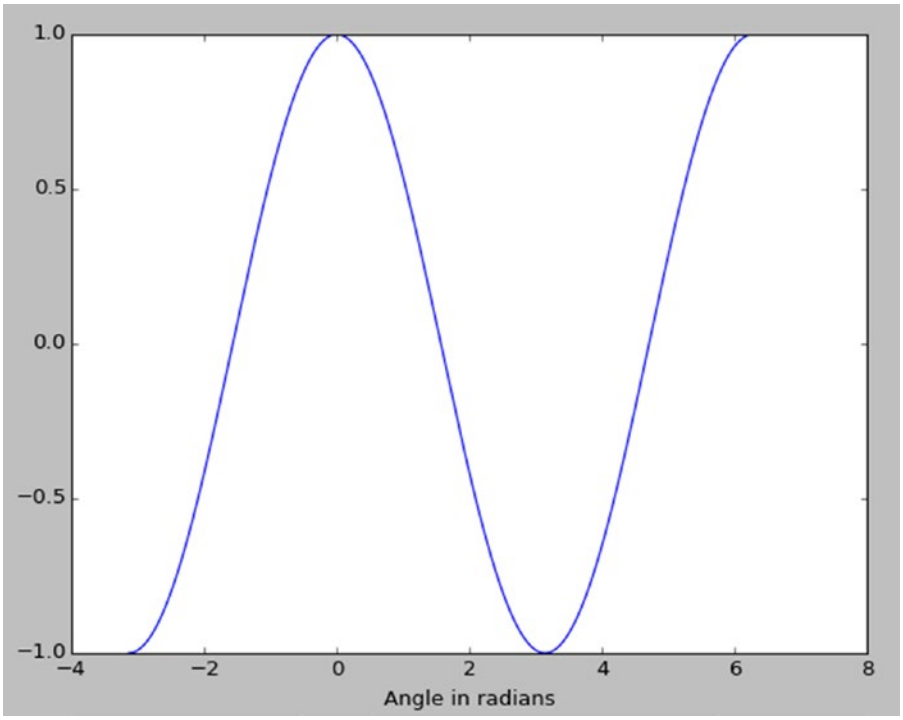


Figure 10-12. *graph for cos(x)*

Let's move on to the hyperbolic Cosine and Sine waves as follows,

```
x = np.linspace(-5, 5, 2000)
plt.plot(x, np.cosh(x))
plt.show()
plt.plot(x, np.sinh(x))
plt.show()
```

The following (Figure 10-13) is the output of `cosh()`:

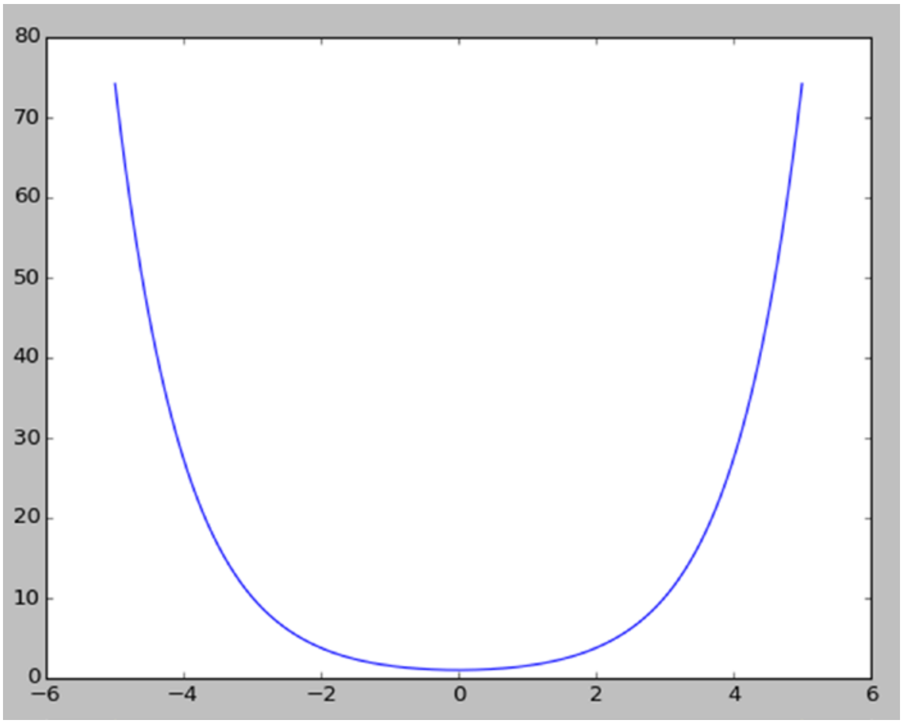


Figure 10-13. *Graph of cosh(x)*

The following (Figure 10-14) is the graph of `sinh(x)`:

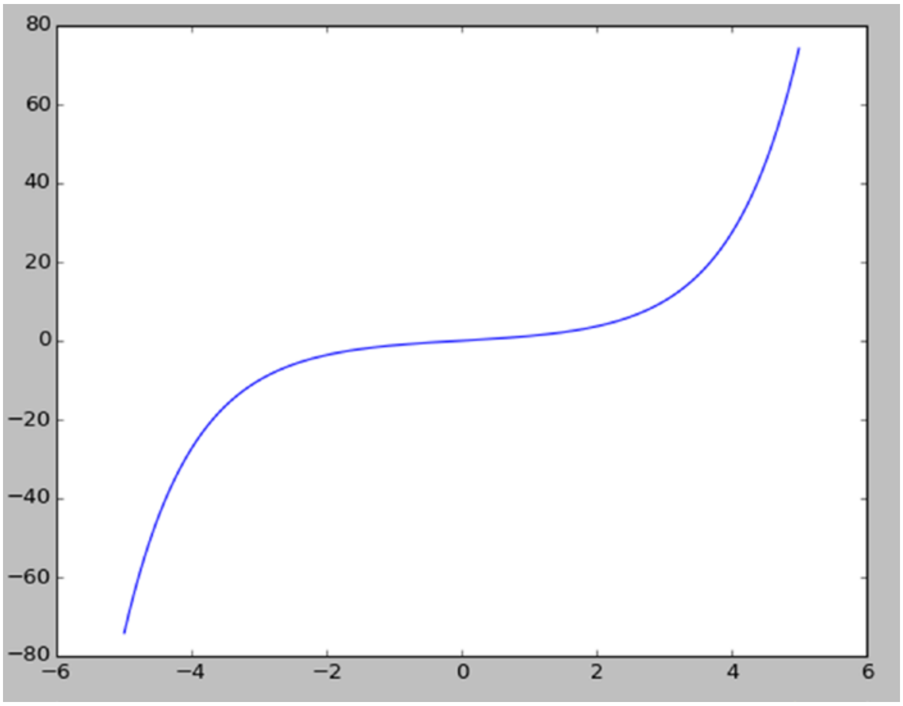


Figure 10-14. *Graph of sinh(x)*

Random Numbers and Statistics

The `rand()` method generates a random matrix of given dimensions. The `randn()` method generates a matrix with numbers sampled from the normal distribution. `randint()` generates a number in given limits excluding the limits. `random_integers()` generates a random integer including the given limits. The following code demonstrates the first three of the above methods:

```
import numpy as np
a = np.random.rand(3, 3)
b = np.random.randn(3, 3)
c = np.random.randint(4, 15)
print(a)
print(b)
print(c)
```

We can calculate median, average, mean, standard deviation, and variance of a `ndarray` as follows:

```
a = np.array([[10, 7, 4], [1, 2, 3], [1, 1, 1]])
print(median(a))
print(average(a))
print(mean(a))
print(std(a))
print(var(a))
```

Fourier Transforms

NumPy has a module for basic signal processing. The `fft` module has the `fft()` method, which is used for computing one-dimensional discrete Fourier transforms as follows:

```
t = np.arange(256)
sp = np.fft.fft(np.sin(t))
freq = np.fft.fftfreq(t.shape[-1])
plt.plot(freq, sp.real, freq, sp.imag)
plt.show()
```

The output (Figure 10-15) is as follows:

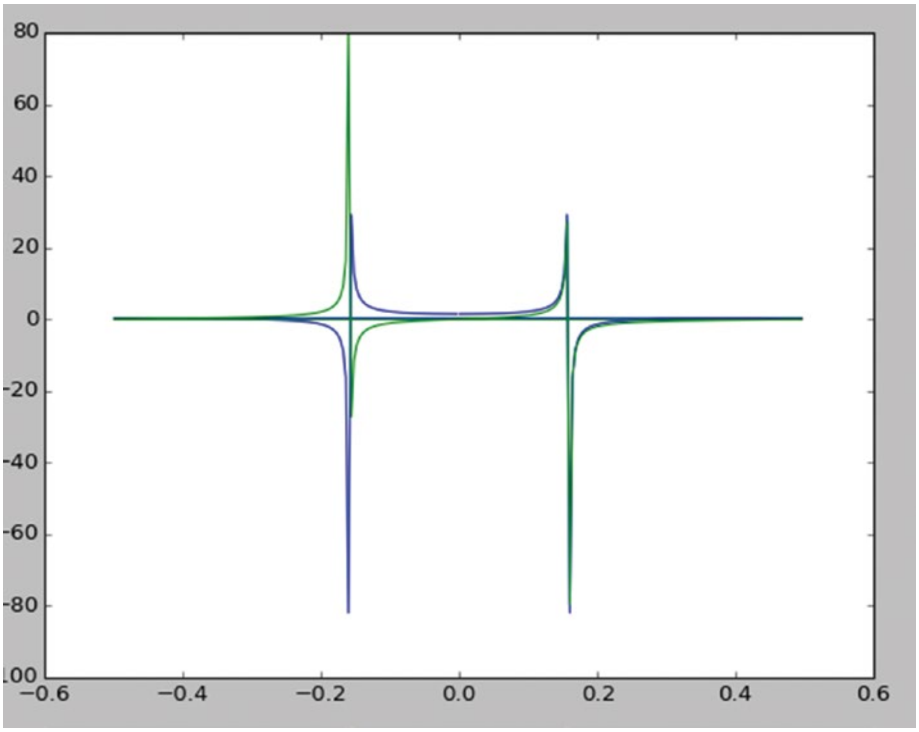


Figure 10-15. *Fast Fourier Transform*

`fft2()` and `fftn()` are used for calculating **two-dimensional** and **n-dimensional** discrete Fourier transforms respectively. Try to write code for these.

Conclusion

In this chapter, we got started with NumPy, matplotlib, and Jupyter, and we learned how to perform numeric computations and basic visualizations in Python. In the next chapter, we will get started with the SciPy Library.

CHAPTER 11

Introduction to SciPy

In the last chapter, we learned how to perform numerical computation with NumPy. We also learned how to use Jupyter for our convenience and how to use matplotlib for visualization. In this chapter, we will be introduced to the SciPy library. However, our journey with NumPy and matplotlib is far from over. We will learn new functionalities in NumPy and matplotlib throughout the remainder of the book. So let's embark upon the exciting journey of scientific computing with SciPy.

Scientific and Mathematical Constants in SciPy

Before we begin, let's complete the ritual of creating a new directory for this chapter. Please create a directory, `chapter11`, for this chapter.

```
cd ~
cd book
cd code
mkdir chapter11
cd chapter11
```

Now start the Jupyter Notebook App with the following command:

```
jupyter notebook
```

Open a new notebook and rename it to `Chapter11_Practice`. This notebook will hold the code for this chapter.

The SciPy library has a module called `scipy.constants` which has the values of many important mathematical and scientific constants. Let's try a few of them. Run the following code in the notebook:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.constants import *
print("Pi = " + str(pi))
print("The golden ratio = " + str(golden))
print("The speed of light = " + str(c))
print("The Planck constant = " + str(h))
print("The standard acceleration of gravity = " + str(g))
print("The Universal constant of gravity = " + str(G))
```

The output is as follows:

```
Pi = 3.141592653589793
The golden ratio = 1.618033988749895
The speed of light = 299792458.0
The Planck constant = 6.62606957e-34,
The standard acceleration of gravity = 9.80665
The Universal constant of gravity = 6.67384e-11
```

Note that the SciPy constants do not include a unit of measurement, only the numeric value of the constants. These are very useful in numerical computing.

■ **Note** There are more of these constants. Please visit https://docs.scipy.org/doc/scipy/reference/constants.html to see more of these.

Linear algebra

Now we will study a few methods related to linear algebra. Let's get started with the inverse of a matrix:

```
import numpy as np
from scipy import linalg
a = np.array([[1, 4], [9, 16]])
b = linalg.inv(a)
print(b)
```

The following is the output:

```
[[-0.8   0.2 ]
 [ 0.45 -0.05]]
```

We can also solve the matrix equation ax = b as follows:

```
a = np.array([[3, 2, 0], [1, -1, 0], [0, 5, 1]])
b = np.array([2, 4, -1])
from scipy import linalg
x = linalg.solve(a, b)
print(x)
print(np.dot(a, x))
```

The following is the output:

```
[ 2. -2.  9.]
[ 2.  4. -1.]
```

We can calculate the determinant of a matrix as follows:

```
a = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
print(linalg.det(a))
```

Run the code above in the notebook and see the results for yourself.
We can also calculate the norms of a matrix as follows:

```
a = np.arange(16).reshape((4, 4))
print(linalg.norm(a))
print(linalg.norm(a, np.inf))
print(linalg.norm(a, 1))
print(linalg.norm(a, -1))
```

The following are the results:

```
35.2136337233
54
36
24
```

We can also compute QR and RQ decompositions as follows:

```
from numpy import random
from scipy import linalg
a = random.randn(3, 3)
q, r = linalg.qr(a)
print(a)
print(q)
print(r)
r, q = linalg.rq(a)
print(r)
print(q)
```

Integration

SciPy has the `integrate` module for various integration operations, so let's look at a few of its methods. The first one is `quad()`. It accepts the function to be integrated as well as the limits of integration as arguments, and then returns the value and approximate error. The following are a few examples:

```
from scipy import integrate
f1 = lambda x: x**4
print(integrate.quad(f1, 0, 3))
import numpy as np
f2 = lambda x: np.exp(-x)
print(integrate.quad(f2, 0, np.inf))
```

The following are the outputs:

```
(48.599999999999994, 5.39568389967826e-13)
(1.0000000000000002, 5.842606742906004e-11)
```

`trapz()` integrates along a given axis using the trapezoidal rule:

```
print(integrate.trapz([1, 2, 3, 4, 5]))
```

The following is the output:

```
12.0
```

Let's see an example of cumulative integration using the trapezoidal rule.

```
import matplotlib.pyplot as plt
x = np.linspace(-2, 2, num=30)
y = x
y_int = integrate.cumtrapz(y, x, initial=0)
plt.plot(x, y_int, 'ro', x, y[0] + 0.5 * x**2, 'b-')
plt.show()
```

The following (Figure 11-1) is the output:

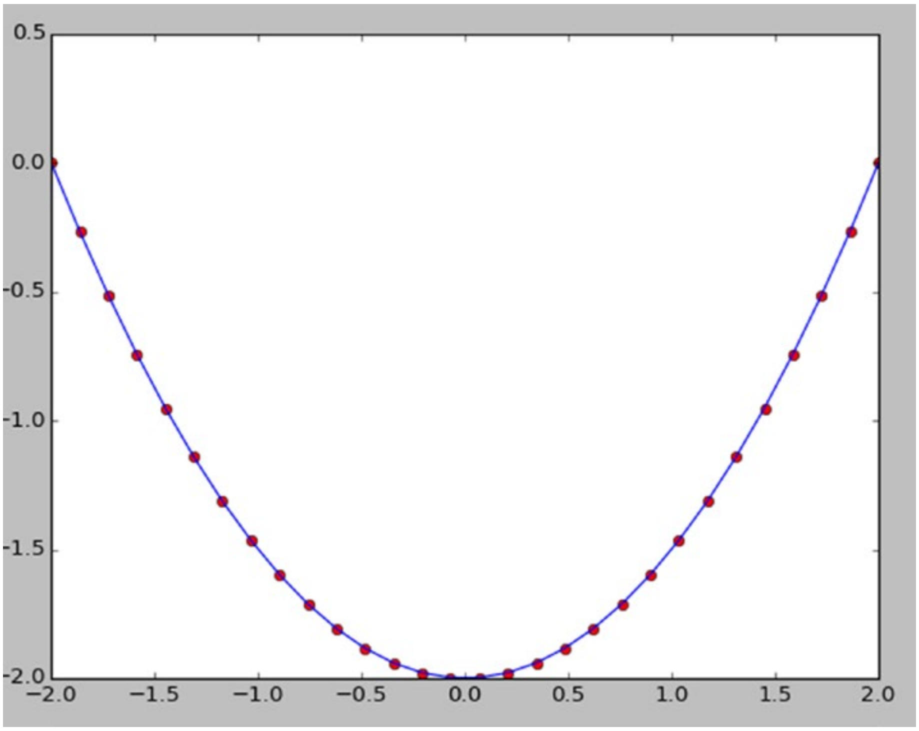


Figure 11-1. *Cumulative integration using the trapezoidal rule*

Interpolation

This module has methods for interpolation. Let's study a few of these with the help of the graphical representation of matplotlib. `interp1d()` is used for 1-D interpolation as demonstrated below.

```
from scipy import interpolate
x = np.arange(0, 15)
y = np.exp(x/3.0)
f = interpolate.interp1d(x, y)
xnew = np.arange(0, 14, 0.1)
ynew = f(xnew)
plt.plot(x, y, 'o', xnew, ynew, '-')
plt.show()
```

The following (Figure 11-2) is the result:

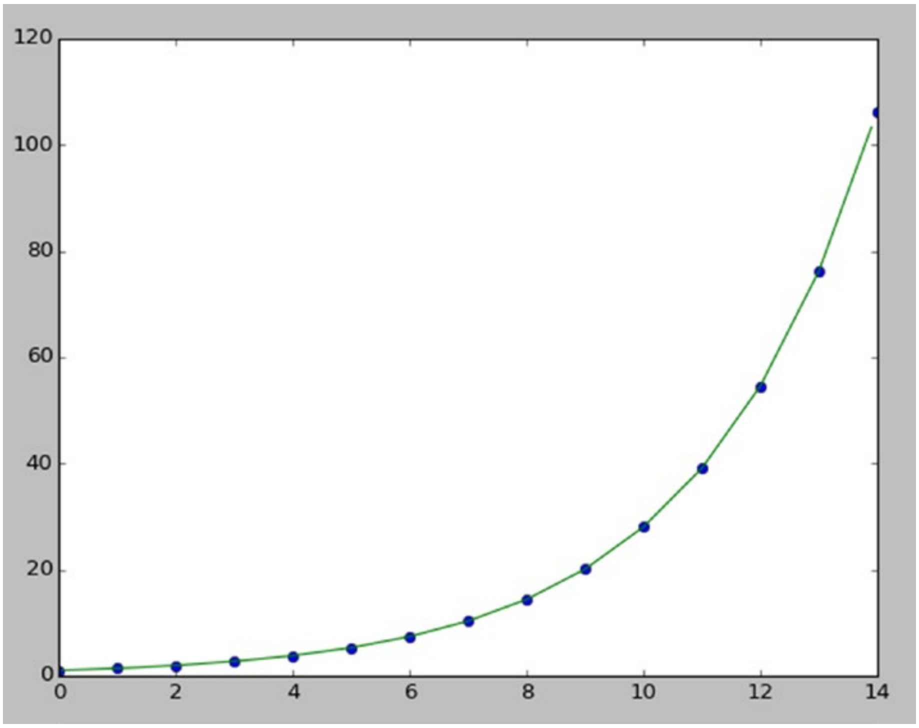


Figure 11-2. *1-D interpolation*

`interp1d()` works for `y=f(x)` type of functions. Simillary, `interp2d()` works on `z=f(x, y)` type of functions. It is used for two-dimensional interpolation.

```
x = np.arange(-5.01, 5.01, 0.25)
y = np.arange(-5.01, 5.01, 0.25)
xx, yy = np.meshgrid(x, y)
z = np.sin(xx**3 + yy**3)
f = interpolate.interp2d(x, y, z, kind='cubic')
xnew = np.arange(-5.01, 5.01, 1e-2)
ynew = np.arange(-5.01, 5.01, 1e-2)
znew = f(xnew, ynew)
plt.plot(x, z[0, :], 'ro-', xnew, znew[0, :], 'b-')
plt.show()
```

The following (Figure 11-3) is the result:

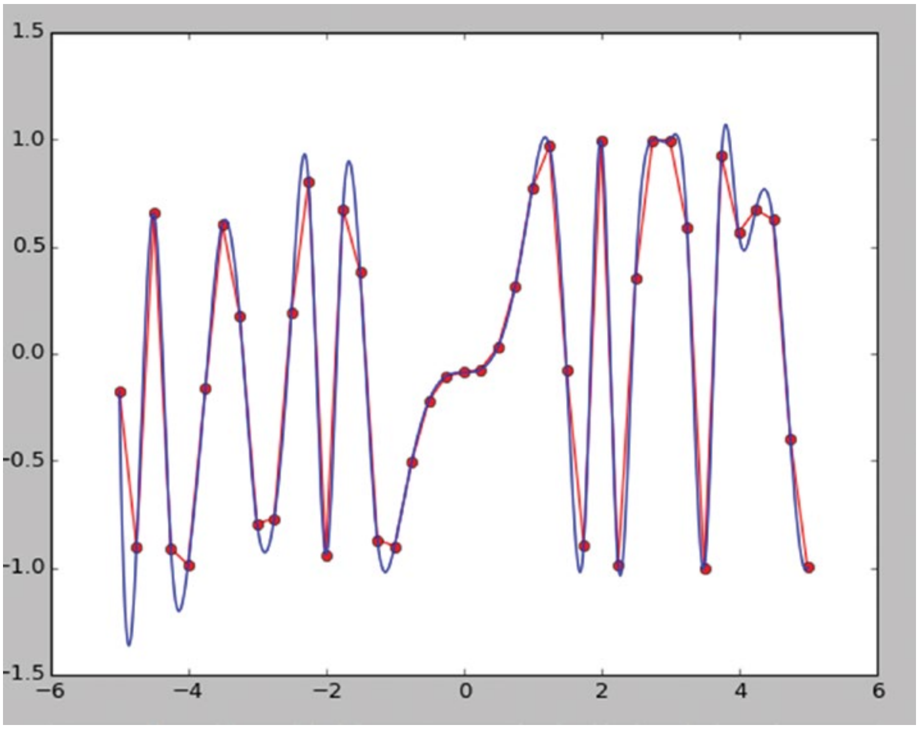


Figure 11-3. *2-D interpolation*

Next, we are going to study `splev()`, `splprep()`, and `splrep()`. The `splev()` method is used to evaluate B-spline or its derivatives. We will use this method along with `splprep()` and `splrep()`, which are used for representations of B-spline.

`splrep()` is used for representation of a 1-D curve as follows:

```
from scipy.interpolate import splev, splrep
x = np.linspace(-10, 10, 50)
y = np.sinh(x)
spl = splrep(x, y)
x2 = np.linspace(-10, 10, 50)
y2 = splev(x2, spl)
plt.plot(x, y, 'o', x2, y2)
plt.show()
```

The following (Figure 11-4) is the result:

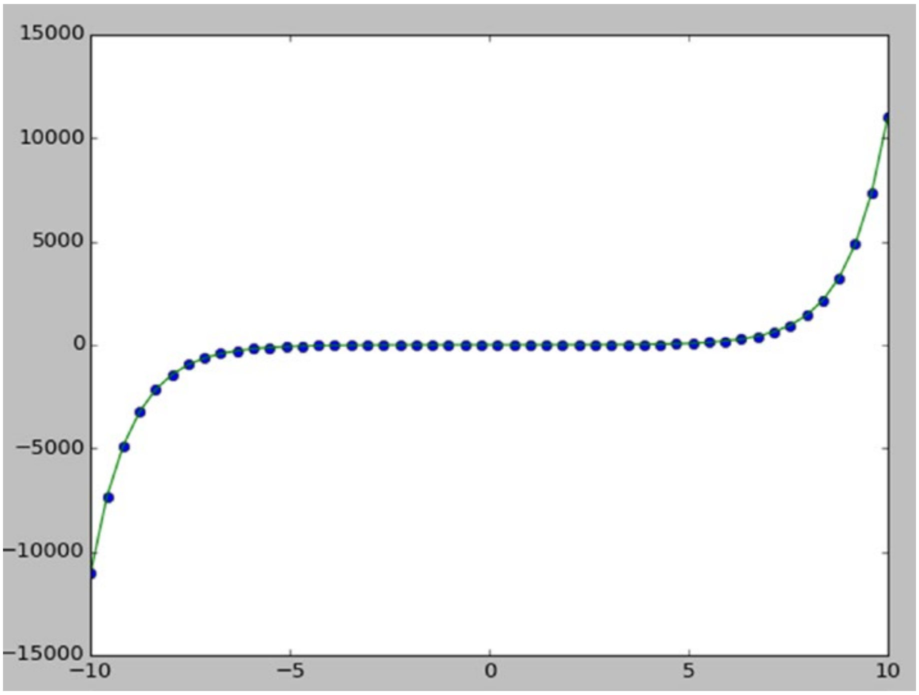


Figure 11-4. *Representation of a 1-D curve*

`splprep()` is used for representation of an N-dimensional curve.

```
from scipy.interpolate import splprep
theta = np.linspace(0, 2*np.pi, 80)
r = 0.5 + np.cosh(theta)
x = r * np.cos(theta)
y = r * np.sin(theta)
tck, u = splprep([x, y], s=0)
new_points = splev(u, tck)
plt.plot(x, y, 'ro')
plt.plot(new_points[0], new_points[1], 'r-')
plt.show()
```

The following (Figure 11-5) is the result.

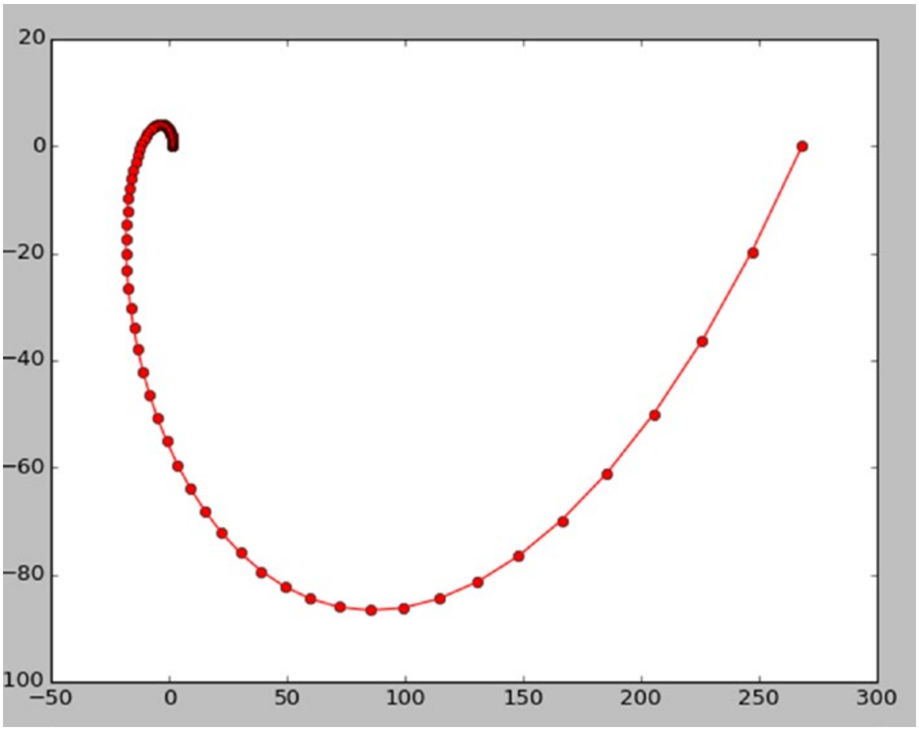


Figure 11-5. *Representation of an N-D curve*

Conclusion

In this chapter, we were introduced to a few important and frequently used modules in the SciPy library. The next two chapters will be focused on introducing readers to the specialized scientific areas of signal and image processing. In the next chapter, we will study a few modules related to the area of signal processing.

CHAPTER 12

Signal Processing with SciPy

In the last chapter, we learned how to perform scientific computations with SciPy. We were introduced to a few modules of the SciPy library. In this chapter, we will explore an important scientific area, signal processing, and we will learn the methods in the `SciPy.signal` module. So let's get started with signal processing in SciPy. This will be a quick, short chapter with only a handful of code examples to provide you with a glimpse of a few fundamentals in the world of signal processing.

Waveforms

Let's get started with the waveform generator functions. Create a new directory `chapter12` in the `~/book/code` directory. Run the following command to start the Jupyter Notebook App:

```
jupyter notebook
```

Rename the current notebook to `Chapter12_Practice`. As in previous chapters, run all the code from this chapter in the same notebook. Let's import NumPy and matplotlib first.

```
import numpy as np
import matplotlib.pyplot as plt
```

The first example we will study is a sawtooth generator function.

```
from scipy import signal
t = np.linspace(0, 2, 5000)
plt.plot(t, signal.sawtooth(2 * np.pi * 4 * t))
plt.show()
```

The function accepts the time sequence and the width of signal and generates triangular or sawtooth shaped continuous signals. The following (Figure 12-1) is the output,

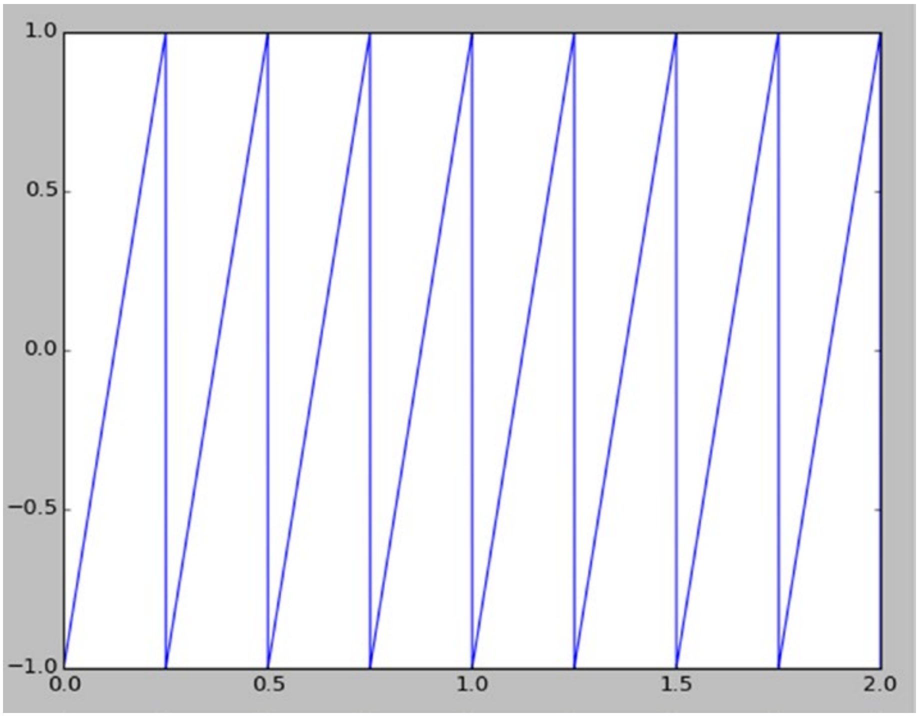


Figure 12-1. *Sawtooth wave signal*

Let's have a look at a square wave generator which accepts time array and duty cycle as inputs.

```
t = np.linspace(0, 1, 400)
plt.plot(t, signal.square(2 * np.pi * 4 * t))
plt.ylim(-2, 2)
plt.title('Square Wave')
plt.show()
```

The output is as follows:

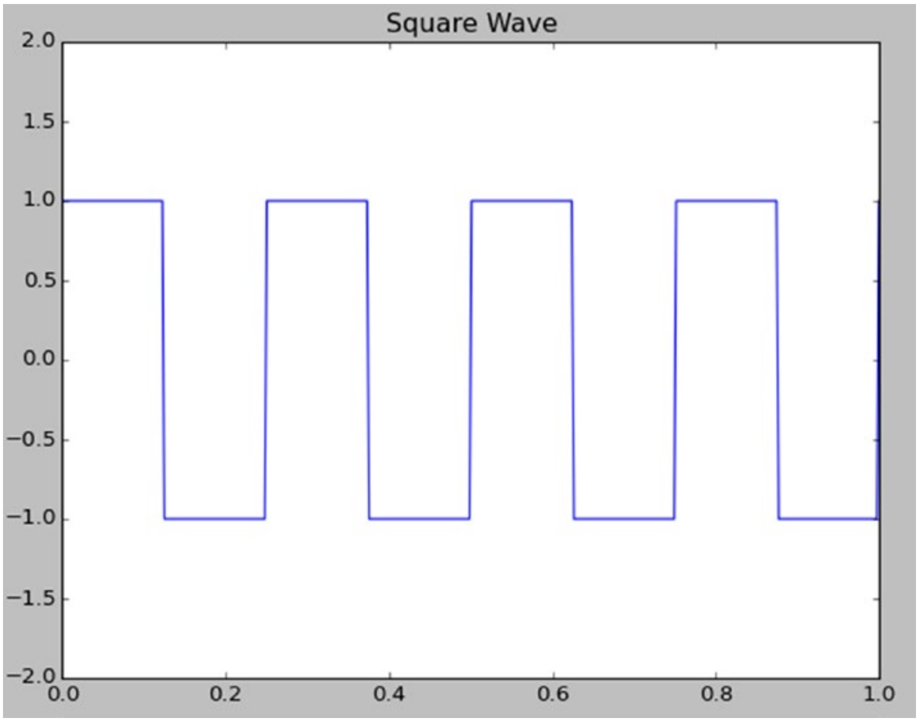


Figure 12-2. *Square wave signal*

A pulse-width modulated square sine wave can be demonstrated as follows:

```
sig = np.sin(2 * np.pi * t)
pwm = signal.square(2 * np.pi * 30 * t, duty=(sig +1)/2)
plt.subplot(2, 1, 1)
plt.plot(t, sig)
plt.title('Sine Wave')
plt.subplot(2, 1, 2)
plt.plot(t, pwm)
plt.title('PWM')
plt.ylim(-1.5, 1.5)
plt.show()
```

The output (Figure 12-3) is as follows:

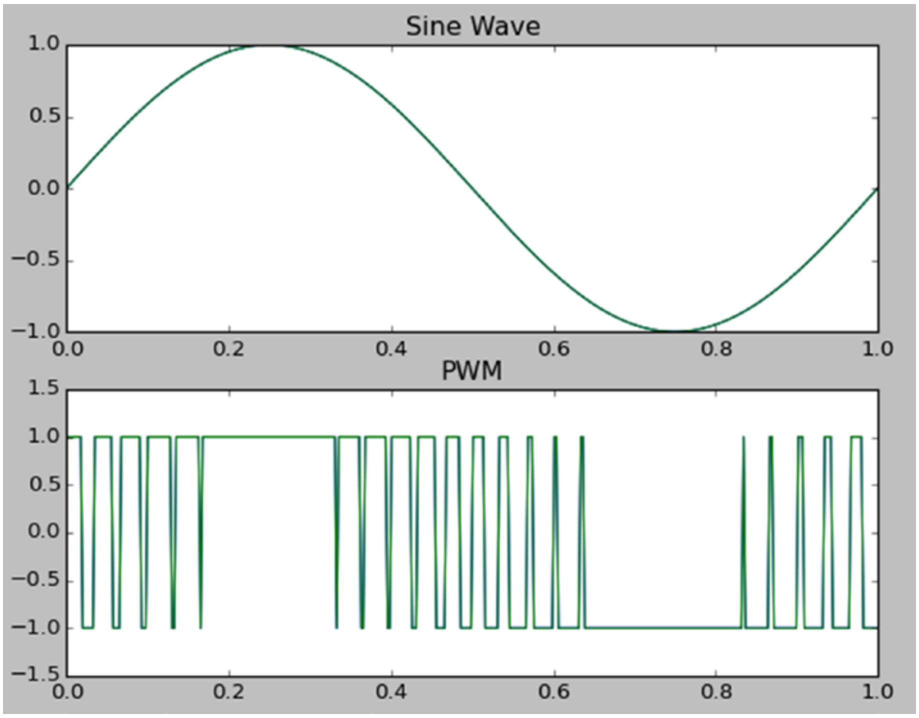


Figure 12-3. *Modulated wave*

Window Functions

A window function is a mathematical function which is zero outside a specific interval. We will now look at three different window functions. The first one is the Hamming window function. We have to pass the number of points in the output window as an argument to all the functions.

```
window = signal.hamming(101)
plt.plot(window)
plt.title('Hamming Window Function')
plt.xlabel('Sample')
plt.ylabel('Amplitude')
plt.show()
```

The output (Figure 12-4) is as follows:

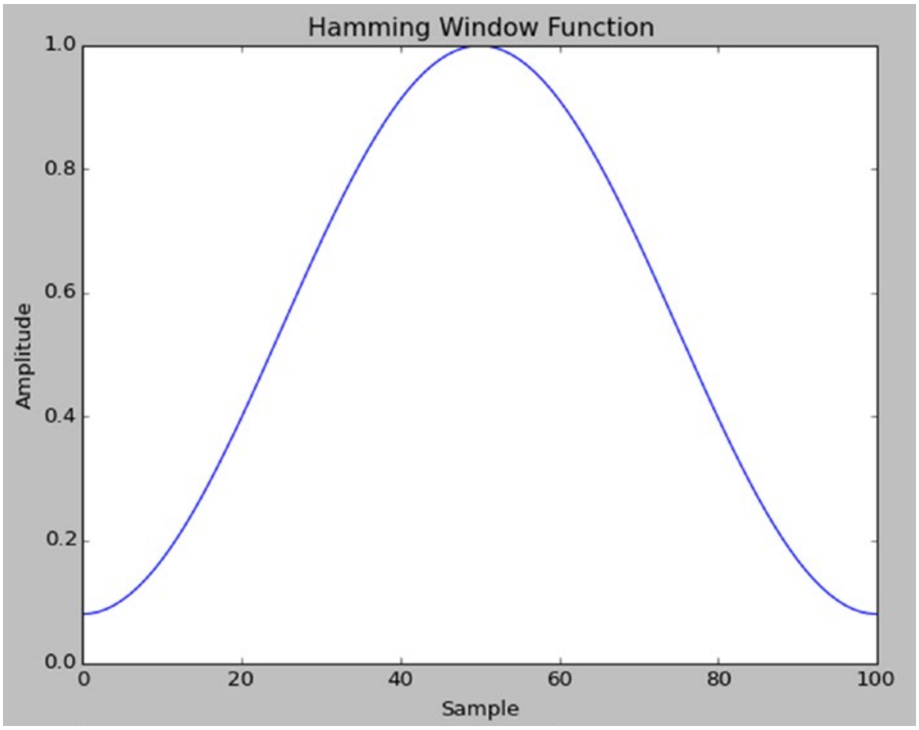


Figure 12-4. *Hamming window demo*

The Hanning window function is as follows:

```
window = signal.hanning(101)
plt.plot(window)
plt.title('Hanning Window Function')
plt.xlabel('Sample')
plt.ylabel('Amplitude')
plt.show()
```

The output (Figure 12-5) is as follows:

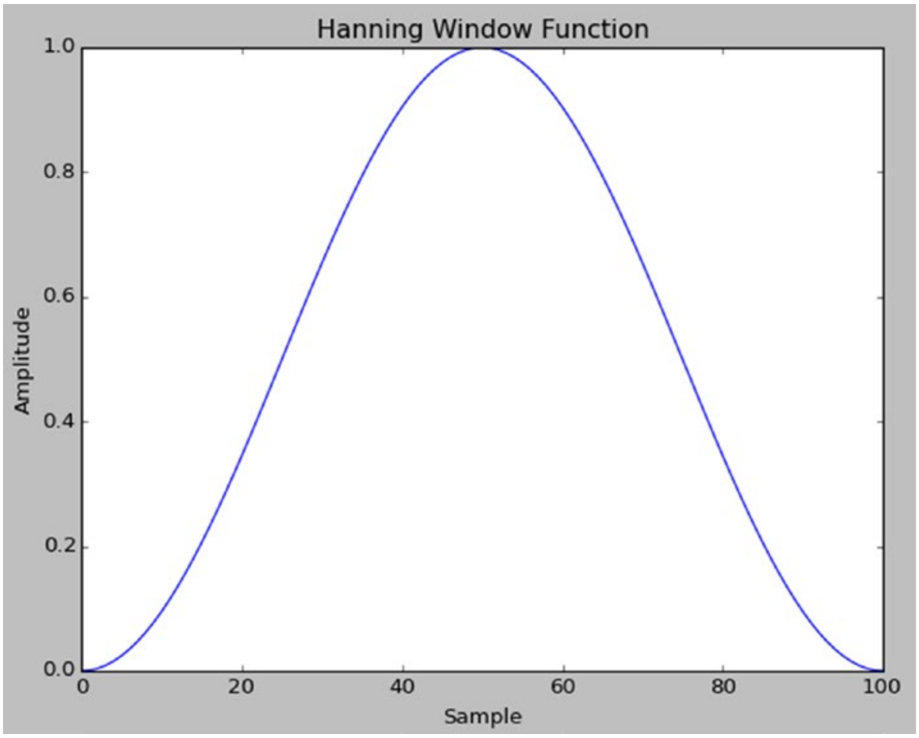


Figure 12-5. *Hanning window demo*

The Kaiser window function is as follows,

```
window = signal.kaiser(51, beta=20)
plt.plot(window)
plt.title('Kaiser Window Function Beta = 20')
plt.xlabel('Sample')
plt.ylabel('Amplitude')
plt.show()
```

The output (Figure 12-6) is as follows:

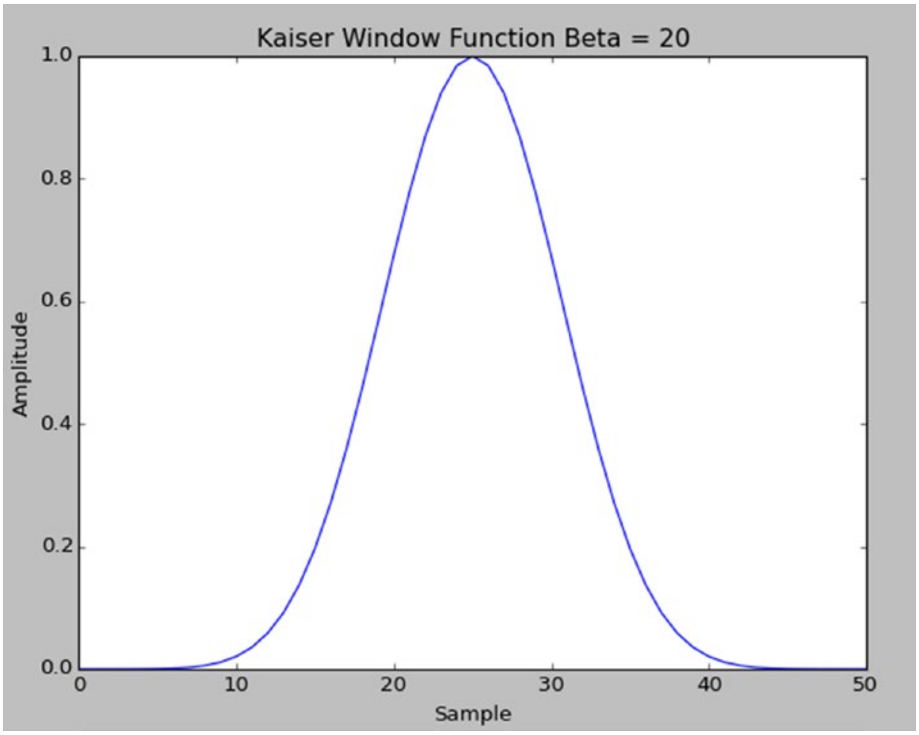


Figure 12-6. *Kaiser window demo*

Mexican Hat Wavelet

We can generate a Mexican hat wavelet with the Ricker function by passing the number of points and the amplitude as parameters, as in the following:

```
plt.plot(signal.ricker(200, 6.0))
plt.show()
```

The Mexican hat wavelet is a special case in the family of continuous wavelets. It is used for filtering and averaging spectral signals. The output is as follows:

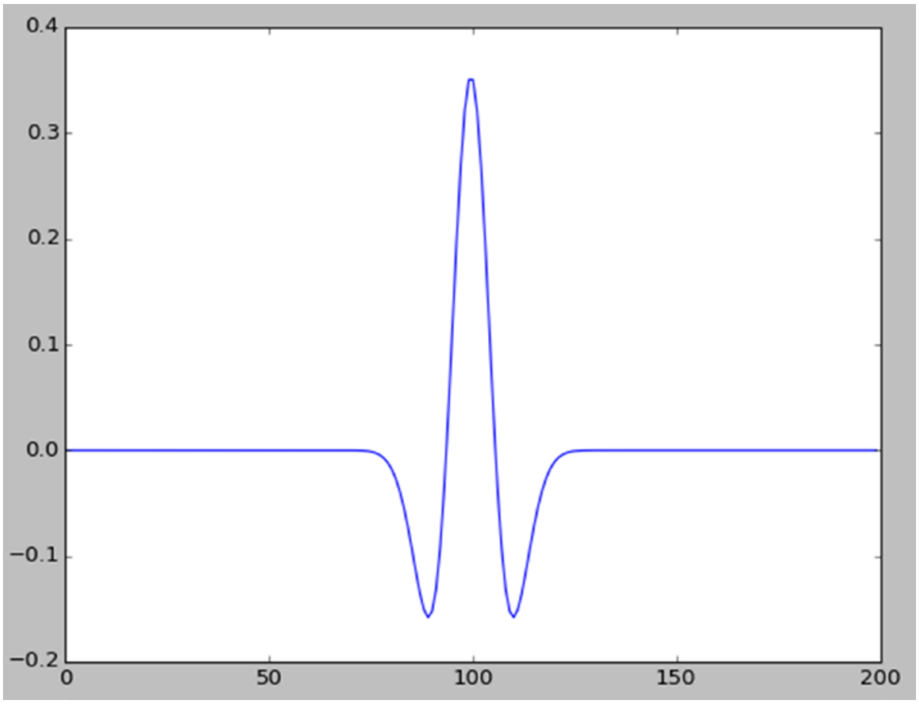


Figure 12-7. *Mexican hat wavelet*

Convolution

We can convolve two N-dimensional arrays with the `convolve()` method as follows:

```
sig = np.repeat([0., 1., 0.], 100)
win = signal.hann(50)
filtered = signal.convolve(sig, win, mode='same') / sum(win)
plt.subplot(3, 1, 1)
plt.plot(sig)
plt.ylim(-0.2, 1.2)
plt.title('Original Pulse')
plt.subplot(3, 1, 2)
plt.plot(win)
plt.ylim(-0.2, 1.2)
plt.title('Filter Impulse Response')
plt.subplot(3, 1, 3)
plt.plot(filtered)
plt.ylim(-0.2, 1.2)
plt.title('Filtered Signal')
plt.show()
```

The signal, the window, and the convolution of those are as shown in Figure 12-8. Convolution of two signals provides a combination of both the signals to form a third signal. It is a very important signal combination technique in signal processing. If the signals represent image or audio data, we get an improved image or audio based on the degree of convolution. If you have not noticed already, we are using the `repeat()` method of NumPy, which takes the pattern to be repeated and the number of repetitions as arguments. In this example, we're generating a signal with the sample size of 300.

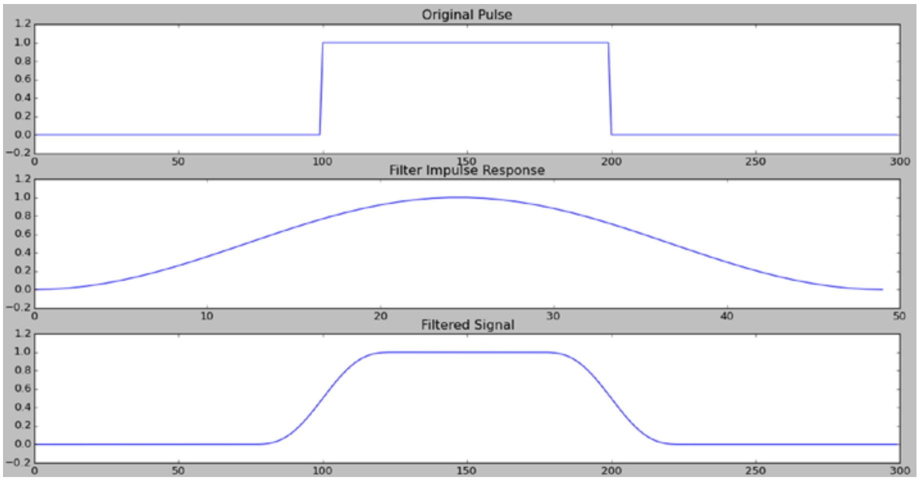


Figure 12-8. *Convolution*

Conclusion

In this short chapter, we were introduced to a few important classes of methods in the `scipy.signal` module of SciPy. In the next chapter, we will explore the area of image processing.

CHAPTER 13

Image Processing with SciPy

In the last chapter, we studied signal processing with SciPy. We studied a few of the major classes of functions offered by SciPy, consolidated under `scipy.signal`. In this chapter, we will study a SciPy package, `scipy.misc`, and a few examples of using it for image processing.

First Image Processing Program

The `scipy.misc` module is used for basic image processing operations. Create a directory called `Dataset` to store all the sample images we are going to use.

```
cd ~/book
mkdir Dataset
```

The sample image dataset is available in the **Downloads** section online on the Apress page for this book. Also, create a directory `chapter13` for the book code as follows:

```
cd ~/book/code
mkdir chapter13
cd chapter13
```

Let's look at a basic example (Listing 13-1) of reading and displaying an image.

Listing 13-1. prog01.py

```
from scipy import misc

img = misc.imread('/home/pi/book/Dataset/4.2.01.tiff')

misc.imshow(img)
```

The code (Listing 13-1) will read an image from the path provided in the `imread()` method, and `imshow()` method will display it using `xlib`.

The `scipy.misc` has three built-in images. Those can be used as follows (Listing 13-2):

Listing 13-2. prog02.py

```
from scipy import misc

img1 = misc.face()
img2 = misc.lena()
img3 = misc.ascent()

misc.imshow(img1)
misc.imshow(img2)
misc.imshow(img3)
```

`face()` is the face of a raccoon, `lena()` is a standard test image, and `ascent()` is a grayscale image.

Simple Image Processing

`scipy.misc` has three methods for simple operations. `scipy.imfilter()` applies various filters on images. The following (Listing 13-3) is an example:

Listing 13-3. prog03.py

```
from scipy import misc

misc.imshow(misc.imfilter(misc.face(), 'edge_enhance_more'))
```

In the code above (Listing 13-3), I am not using any intermediate variable to store the image. I am displaying it directly by passing to the `imshow()` method. The method `imfilter()` accepts two arguments.

- The first is the image to be filtered.
- The second is the type of pre-defined filter to be applied.

Allowed values for the filter-type are `'blur', 'contour', 'detail', 'edge_enhance', 'edge_enhance_more', 'emboss', 'find_edges', 'smooth', 'smooth_more', 'sharpen'`.

We can resize the image to 50% as follows (Listing 13-4):

Listing 13-4. prog04.py

```
from scipy import misc
misc.imshow(misc.imresize(misc.face(), 50))
```

We can also rotate the image by a certain angle as follows (Listing 13-5):

Listing 13-5. prog05.py

```
from scipy import misc
misc.imshow(misc.imrotate(misc.face(), 45))
```

Introduction to NumPy for Image Processing

Let's get started with the basics of using the NumPy library for image processing. Consider the following (Listing 13-6) code:

Listing 13-6. prog06.py

```
from scipy import misc

img = misc.face()

print(type(img))
```

The output of the program (Listing 13-6) above is as follows:

```
<class 'numpy.ndarray'>
```

This means that the data-type of the image is `ndarray` in NumPy. We need to understand a few important `ndarray` properties which will help us understand the important attributes of the images it represents.

Consider the following code (Listing 13-7):

Listing 13-7. prog07.py

```
from scipy import misc

img = misc.face()

print(img.dtype)
print(img.shape)
print(img.ndim)
print(img.size)
```

The output is as follows:

```
uint8
(768, 1024, 3)
3
2359296
```

Let's understand what each of these mean.

- The `dtype` attribute is for the data-type of the elements which represent the image. In this case, it is `uint8` which means unsigned 8-bit integer. This means it can have 256 distinct values.

- `shape` means the size of images, dimension-wise. In this case, it is a color image. Its resolution is 1024x768 and it has three color channels corresponding to the colors red, green, and blue. Each channel for each pixel can have one of a possible 256 values. Therefore, a combination of these can produce 256*256*256 distinct colors for each pixel. You can visualize a color image as an arrangement of three two-dimensional planes. A grayscale image is a single plane of grayscale values.
- `ndim` represents the dimensions. A color image has three dimensions and a grayscale image has two dimensions.
- `size` stands for the total number of elements in the array. It can be calculated by multiplying the values for the dimensions. In this case, it is 768*1024*3=2359296.

We can see the RGB value corresponding to an individual pixel as follows (Listing 13-8):

Listing 13-8. prog08.py

```
from scipy import misc

img = misc.face()

print(img[10, 10]))
```

In the code above (Listing 13-8), we are accessing the value of the pixel located at (10, 10). The output is `[172 169 188]`.

These are the basics of NumPy for image processing. We will learn more about NumPy as and when needed throughout the chapter.

Matplotlib for Image Processing

We have used the `misc.imshow()` method for displaying images. While the method is useful for simple applications, it is primitive. We need to use a more advanced framework for scientific applications. We know that matplotlib serves this purpose. It is a MATLAB-style plotting and data visualization library for Python, and we installed it while installing the SciPy stack. We have also used it in the earlier chapter. In this and the next chapter, we will use it for displaying images. It is an integral part of the SciPy stack. Just like NumPy, matplotlib is too vast a topic for one book, and warrants another one. We will just use the `pyplot` module in matplotlib for our image processing requirements. Let's see a simple program (Listing 13-9) for image processing as follows:

Listing 13-9. prog09.py

```
import scipy.misc as misc
import matplotlib.pyplot as plt
```

```
img = misc.face()

plt.imshow(img)
plt.show()
```

In the code above (Listing 13-9), we are importing the `pyplot` module. The `imshow()` method is used to add the image to the plot window. The `show()` method shows the plotting window. The output (Figure 13-1) is as follows:

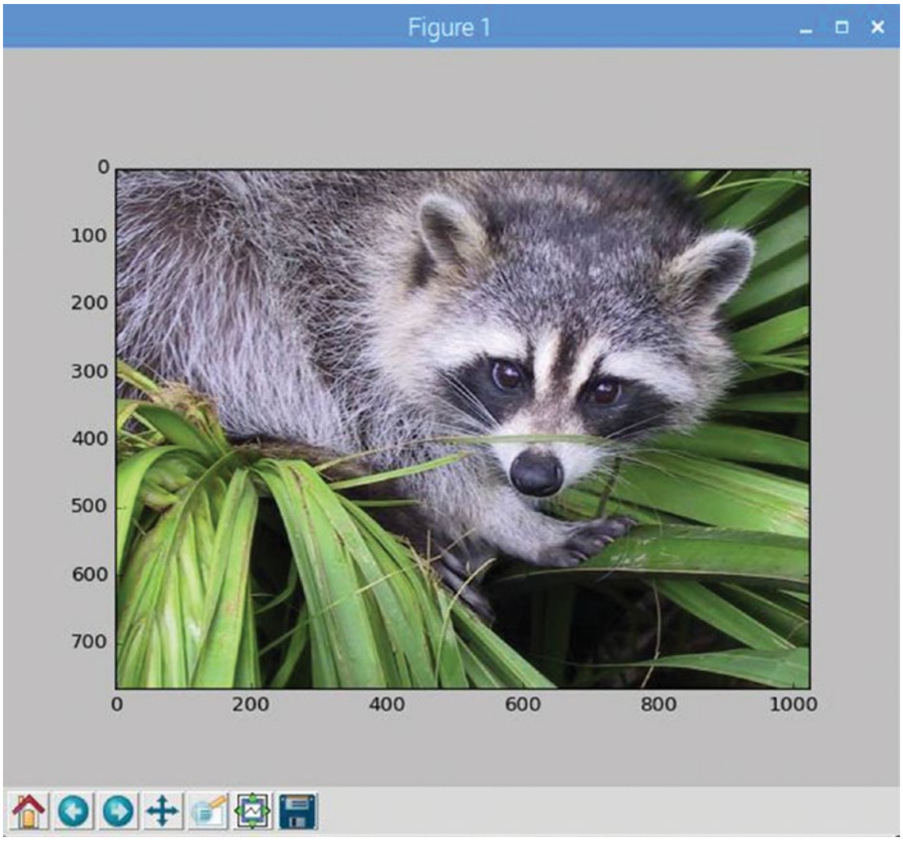


Figure 13-1. *pyplot imshow() demo for color image*

We can also turn off the axes (or ruler) and add a title to the image as follows (Listing 13-10):

Listing 13-10. prog10.py

```
import scipy.misc as misc
import matplotlib.pyplot as plt

img = misc.lena()

plt.imshow(img, cmap='gray')
plt.axis('off')
plt.title('face')
plt.show()
```

As the image is a grayscale image, we have to choose a `gray` colormap in the `imshow()` method so that the image's colorspace is properly displayed in the plotting window. `axis('off')` is used to turn the axes off. The `title()` method is used for specifying the title of the image. The output (Figure 13-2) is as follows:

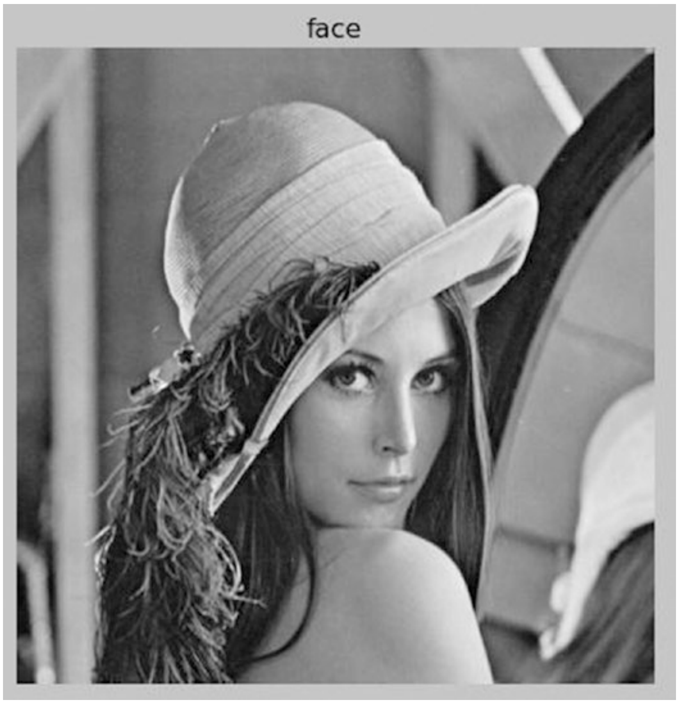


Figure 13-2. *Lena image with title and axis off*

We can use `imshow()` to push multiple images to an image grid in the plotting window (see Listing 13-11) as follows:

Listing 13-11. prog11.py

```
import scipy.misc as misc
import matplotlib.pyplot as plt

img1 = misc.face()
img2 = misc.ascent()
img3 = misc.lena()

titles = ['face', 'ascent', 'lena']
images = [img1, img2, img3]

plt.subplot(1, 3, 1)
plt.imshow(images[0])
plt.axis('off')
plt.title(titles[0])

plt.subplot(1, 3, 2)
plt.imshow(images[1], cmap='gray')
plt.axis('off')
plt.title(titles[1])

plt.subplot(1, 3, 3)
plt.imshow(images[2], cmap='gray')
plt.axis('off')
plt.title(titles[2])

plt.show()
```

We have used the `subplot()` method before `imshow()`. The first two arguments in the `subplot()` method specify the dimensions of the grid and the third argument specifies the position of the image in the grid. The position numbering of the images in the grid starts from the top-left. The top-left position is the first position, the next position is the second one and so on. Let's see the output (Figure 13-3):

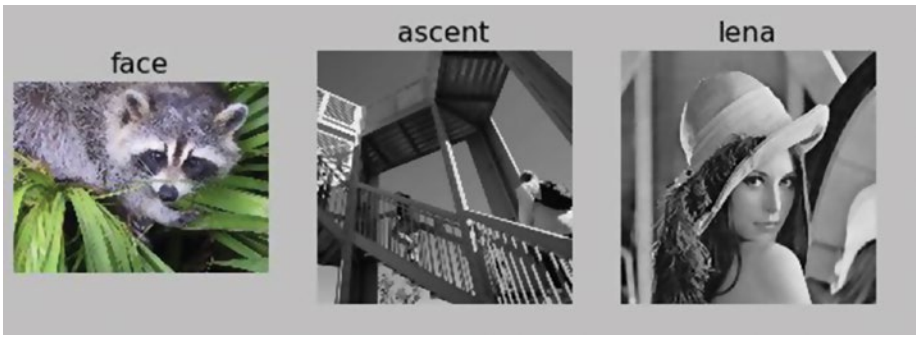


Figure 13-3. *Multiple image grid*

Image Channels

We can separate the image channels of a multi-channel image. The code (Listing 13-12) for that is as follows:

Listing 13-12. Prog12.py

```
import scipy.misc as misc
import matplotlib.pyplot as plt

img = misc.face()

r = img[:, :, 0]
g = img[:, :, 1]
b = img[:, :, 2]

titles = ['face', 'Red', 'Green', 'Blue']
images = [img, r, g, b]

plt.subplot(2, 2, 1)
plt.imshow(images[0])
plt.axis('off')
plt.title(titles[0])

plt.subplot(2, 2, 2)
plt.imshow(images[1], cmap='gray')
plt.axis('off')
plt.title(titles[1])

plt.subplot(2, 2, 3)
plt.imshow(images[2], cmap='gray')
plt.axis('off')
plt.title(titles[2])

plt.subplot(2, 2, 4)
plt.imshow(images[3], cmap='gray')
plt.axis('off')
plt.title(titles[3])

plt.show()
```

The output (Figure 13-4) of the code (Listing 13-12) is as follows:

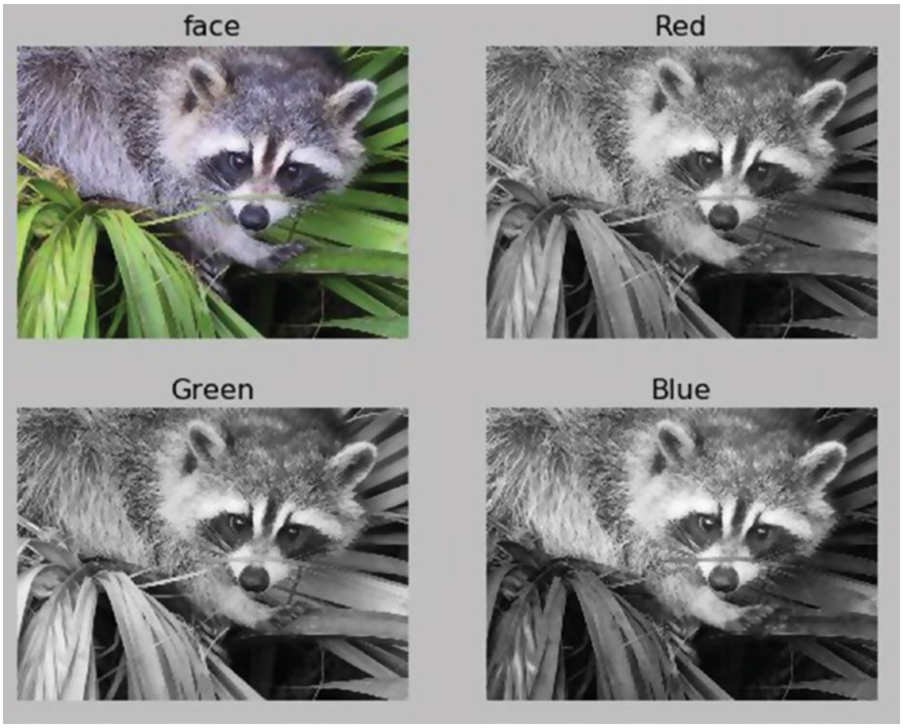


Figure 13-4. *Separate image channels*

We can use the `np.dstack()` method to merge all channels to create the original image, as follows (Listing 13-13):

Listing 13-13. prog13.py

```
import scipy.misc as misc
import matplotlib.pyplot as plt
import numpy as np

img = misc.face()

r = img[:, :, 0]
g = img[:, :, 1]
b = img[:, :, 2]

output = np.dstack((r, g, b))
```

```
plt.imshow(output)
plt.axis('off')
plt.title('Combined')
plt.show()
```

Run the code above (Listing 13-13) and see the workings of `np.dstack()` for yourself.

■ **Note** I have written a detailed book on image processing with Raspberry Pi. It can be purchased at `www.apress.com/us/book/9781484227305`.

Conclusion

In this chapter, we were introduced to the world of image processing with SciPy. We also learned how images are represented using the NumPy `ndarray`. We learned to perform a few basic operations on images with the `scipy.misc` package. In the next chapter, we will learn data representation and processing with matplotlib.

CHAPTER 14

Matplotlib

In the last chapter, we studied digital image processing with SciPy. We studied a few of the major classes of functions offered by SciPy, consolidated under `scipy.misc`, for digital image processing. In this chapter, we will study a few more image processing and data representation techniques with matplotlib. We have already used matplotlib in earlier chapters for plotting and displaying images. As mentioned in earlier chapters, matplotlib is a MATLAB-style data visualization library. Data processing and mining is a vast topic and outside the scope of this book; however, we can use images as a convenient data source to demonstrate some of the data processing capabilities of matplotlib. So let's get started with that.

Reading an Image

Create a directory `chapter14` for the code samples. The following code (Listing 14-1) demonstrates how to read and display an image:

Listing 14-1. prog01.py

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
img = mpimg.imread('/home/pi/book/Dataset/Sample01.jpg')
plt.imshow(img)
plt.show()
```

The output (Figure 14-1) is as follows:

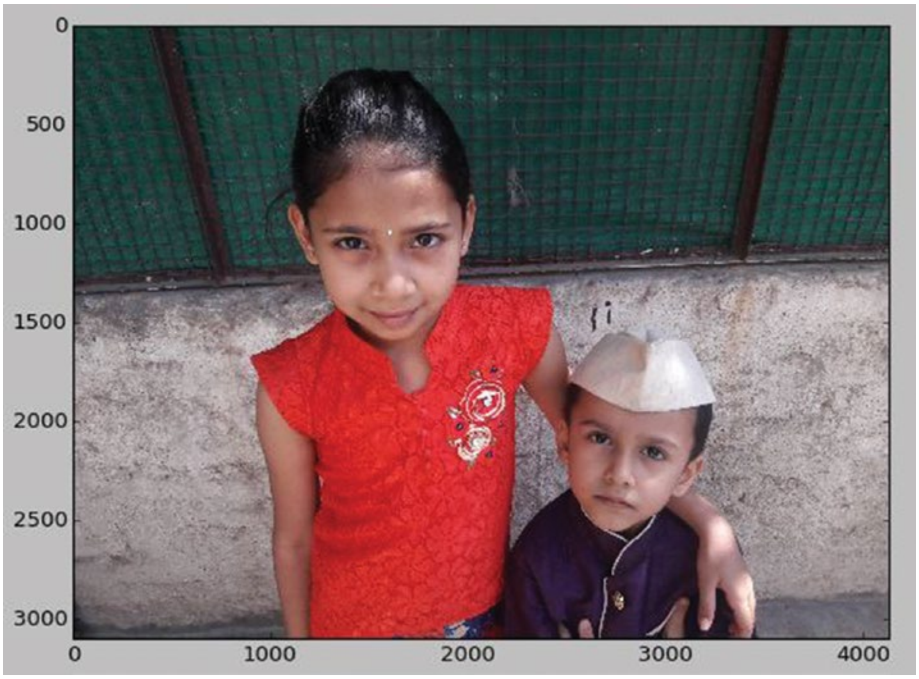


Figure 14-1. *Reading and displaying an image*

Colormaps

Colormaps are used to apply colors to a dataset. Grayscale images are automatically applied with the default colormaps. We can even set the colormap for the image. To display grayscale images properly, we need to set the colormap to the value gray as we have done in earlier chapters. In the example below (Listing 14-2), we are going to display one of the channels of an image with the default colormap. Then we will apply another colormap to the channel.

Listing 14-2. prog02.py

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np

img = mpimg.imread('/home/pi/book/Dataset/4.2.01.tiff')
img_ch = img[:, :, 0]

plt.subplot(1, 2, 1)
plt.imshow(img_ch)
plt.title('Default Colormap')
plt.xticks([]), plt.yticks([])
plt.subplot(1, 2, 2)
plt.imshow(img_ch, cmap='hot')
plt.title('Hot Colormap')
plt.xticks([]), plt.yticks([])

plt.show()
```

The output (Figure 14-2) is as follows:

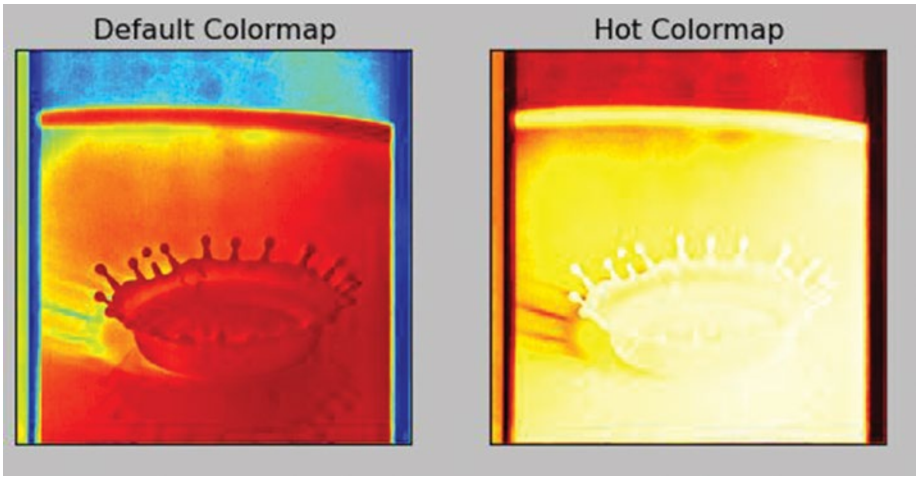


Figure 14-2. *Colormaps*

Colorbar

We can also display the colorbar to let the viewers know the relative intensity values in an image. The following code (Listing 14-3) demonstrates that:

Listing 14-3. prog03.py

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np

img = mpimg.imread('/home/pi/book/Dataset/4.2.01.tiff')
img_ch = img[:, :, 0]

plt.imshow(img_ch, cmap='nipy_spectral')
plt.title('Colorbar Demo')
plt.colorbar()
plt.xticks([]), plt.yticks([])

plt.show()
```

The output (Figure 14-3) is as follows:

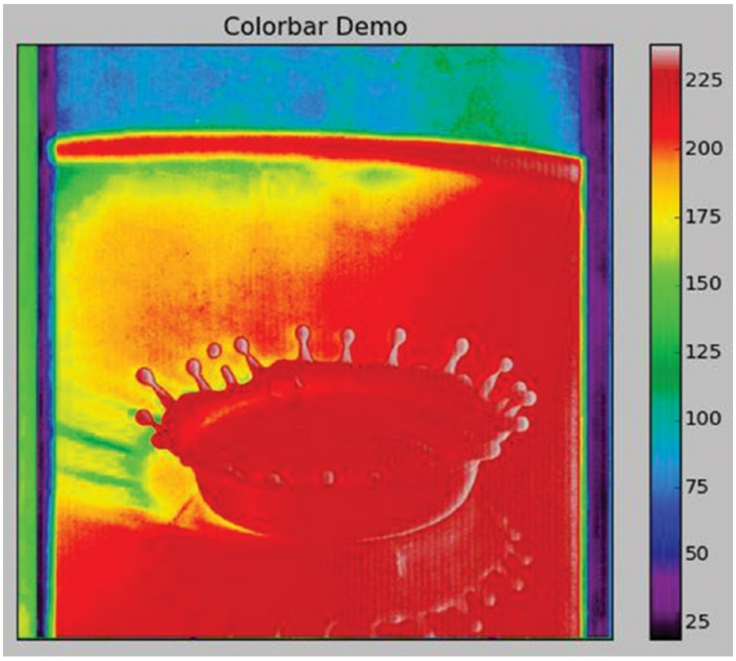


Figure 14-3. *Colorbar demo*

Matplotlib for Image Processing

A histogram is the graphical representation of frequency tables in statistics. It is a graph of the number of occurrences for every value in the dataset. We can also use the `plt.hist()` method in matplotlib for plotting the histogram of a single channel or a grayscale image. The following (Listing 14-4) is an example:

Listing 14-4. prog04.py

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np

img = mpimg.imread('/home/pi/book/Dataset/Sample03.jpg')
img_ch = img[:, :, 0]

plt.hist(img_ch.ravel(), 256, [0, 256])
plt.title('Histogram Demo')
plt.xticks([]), plt.yticks([])

plt.show()
```

The output (Figure 14-4) is as follows:

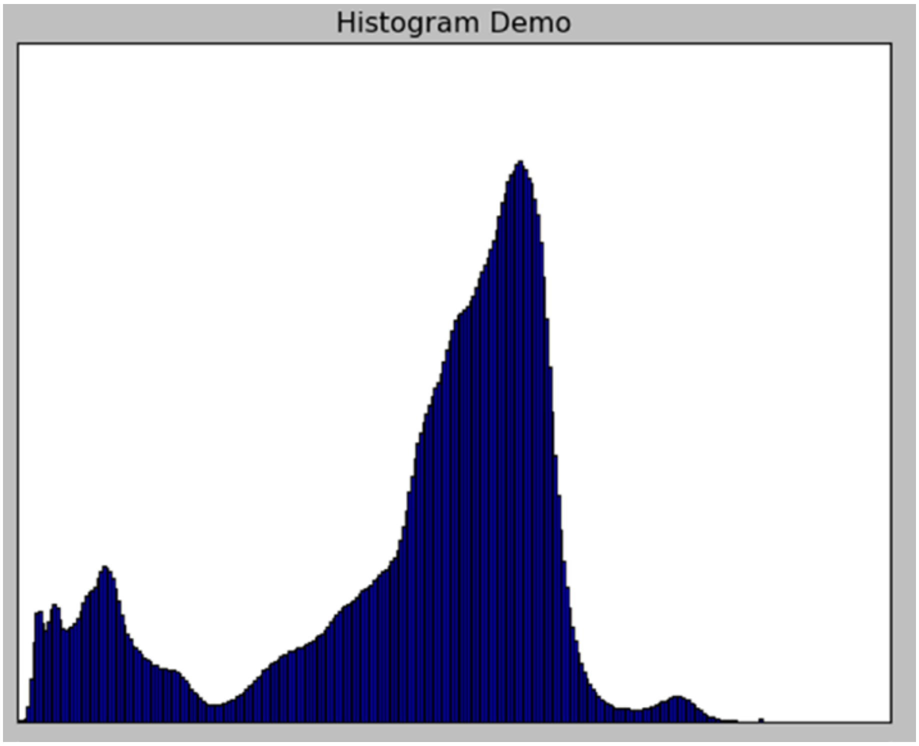


Figure 14-4. *Histogram demo*

Interpolation Methods

There are many interpolation types in `plt.imshow()`. The interpolation type decides how the image is to be displayed. The best way to understand how they work is to use them against a gradient image. The following (Listing 14-5) code example demonstrates this very well:

Listing 14-5. prog05.py

```
import matplotlib.pyplot as plt
import numpy as np

methods = [None, 'none', 'nearest', 'bilinear', 'bicubic', 'spline16',
           'spline36', 'hanning', 'hamming', 'hermite', 'kaiser', 'quadric',
           'catrom', 'gaussian', 'bessel', 'mitchell', 'sinc', 'lanczos']
```

```
grid = np.arange(16).reshape(4, 4)

fig, axes = plt.subplots(3, 6, figsize=(12, 6),
subplot_kw={'xticks': [], 'yticks': []})

fig.subplots_adjust(hspace=0.3, wspace=0.05)

for ax, interp_method in zip(axes.flat, methods):
    ax.imshow(grid, interpolation=interp_method)
    ax.set_title(interp_method)

plt.show()
```

The output (Figure 14-5) is as follows:

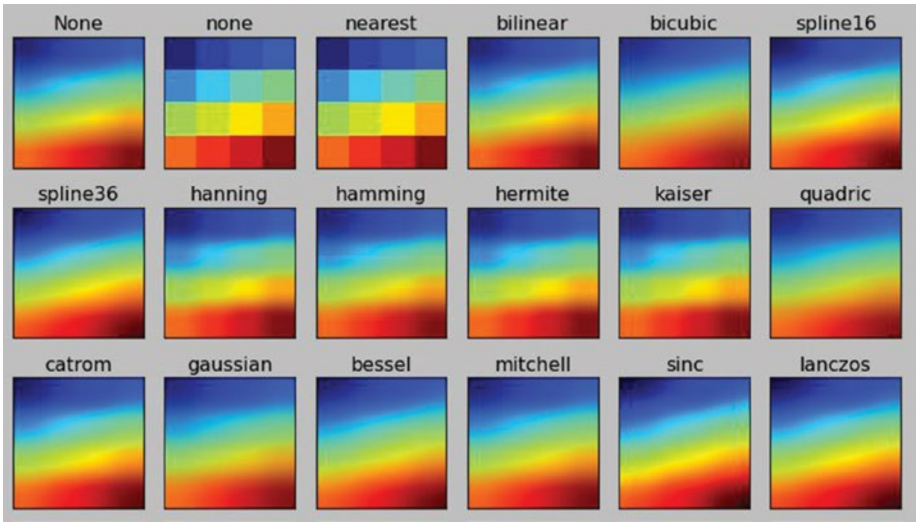


Figure 14-5. *Interpolation demo*

Conclusion

In this chapter, we learned how to represent data with **matplotlib**. We studied colormaps, colorbars, and histograms. We also studied the concept of interpolation. We can use matplotlib this way for representing data, images, and signals.

Summary of the Book

In this book, we got started with the very fundamentals of Raspberry Pi and single board computers. We learned how to set up the Raspberry Pi and connect it to the Internet. We also learned to access it with a network.

Then we moved on to the basics of supercomputing and parallel programming. We prepared the nodes of our Pis and joined them together over a network for them to act as a cluster. We also exploited the power of the cluster with MPI4PY.

Then we studied symbolic computing with Python. We also studied the NumPy library for numerical computation. We then explored the scientific computing library SciPy and its applications in signal and image processing.

Finally, we studied how to represent image data and calculate histograms with the matplotlib library.

This is not the end, though. This is merely the beginning of your journey in the amazing world of scientific computing. You can further explore matplotlib, OpenCV, and Scientific Kit (SciKit) libraries. For folks who want to try OS programming, they can explore pthread and POSIX libraries in C on Raspberry Pi. Possibilities are endless with the Raspberry Pi. I wish you all the best in getting started on this amazing journey of exploration.

Index