پلتفرم اختصاصی مهندسی کنترل

Python for Control Engineering

Hans-Petter Halvorsen

https://www.halvorsen.blog

python™

Python for Control Engineering

Python for Control Engineering

Hans-Petter Halvorsen

2020

Python for Control Engineering

©Hans-Petter Halvorsen

April 25, 2022

Preface

Python is a popular programming language, and it is one of the most used programming languages today.

Python works on all the main platforms and operating systems used today, such Windows, macOS, and Linux.

Python is a multi-purpose programming language, which can be use for simulation, creating web pages, communicate with database systems, etc.

My Blog/Web Site [1]:
https://www.halvorsen.blog

Here you find lots of technical resources about Technology, Programming, Software Engineering, Automation and Control, Industrial IT, etc.

Here you find my Web page with Python resources:

https://www.halvorsen.blog/documents/programming/python/
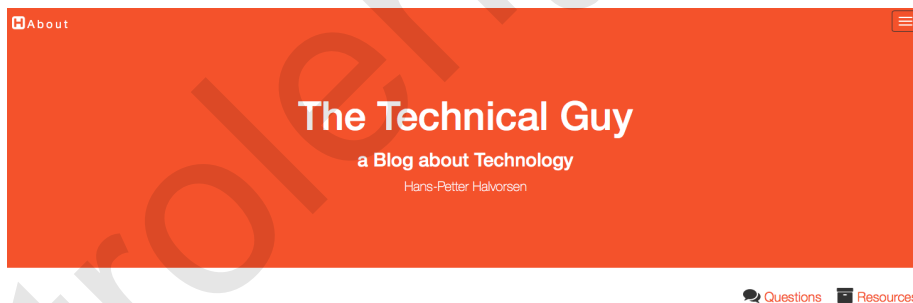
These resources are a supplement to this textbook. Here you can download the software, download code examples, etc.

This Textbook is written in LaTeX using Overleaf.

LaTeX is a document preparation system used for the communication and publication of scientific documents.

For more information about LaTeX:
https://www.latex-project.org

Overleaf is a web-bases LaTeXsystem, meaning you can write your LaTeXdocuments in your web browser, you co-work and share documents with others.

For more information about Overleaf:
https://www.overleaf.com

Python Books

You find other Python textbooks within different domains on my Python Web page:
https://www.halvorsen.blog/documents/programming/python/

Python Books:

- **Python Programming** - This is a textbook in Python Programming with lots of Practical Examples and Exercises. You will learn the necessary foundation for basic programming with focus on Python.

- **Python for Science and Engineering** - This is a textbook in Python Programming with lots of Examples, Exercises, and Practical Applications within Mathematics, Simulations, etc. The focus is on numerical calculations in mathematics and engineering. Necessary theory is presented in addition to many practical examples.

- **Python for Control Engineering** - This is a textbook in Python Programming with lots of Examples, Exercises, and Practical Applications within Mathematics, Simulations, Control Systems, DAQ, Database Systems, etc. The focus is on the use of Python within measurements, data collection (DAQ), control technology, both analysis of control systems (stability analysis, frequency response, ...) and implementation of control systems (PID, etc.). Required theory is presented in addition to many practical examples and exercises in Python.

- **Python for Software Development** - This is a textbook in Python Programming with lots of Examples, Exercises, and Practical Applications within Software Systems, Software Development, Software Engineering, Database Systems, Web Application Desktop Applications, GUI Applications, etc. The focus is on the use of Python for creating modern Software Systems. Required theory is presented in addition to many practical examples and exercises in Python.

Video Resources

In addition to the textbooks mentioned, lots of videos explaining and comple-
ments the different Python topics and examples within the textbook have been
made. These are both available on my website and on YouTube.

Blog:
https://www.halvorsen.blog

Python Resources:
https://www.halvorsen.blog/documents/programming/python/

Python Programming Videos:
https://www.youtube.com/playlist?list=PLdb-TcK6Aqj2l$_H$1mtPqlOo−Yki5UPzp4

Python for Science and Engineering Videos:
https://www.youtube.com/playlist?list=PLdb-TcK6Aqj2hlH55Bn5oxFIvyoVbXxQS

Python for Control Engineering Videos:
https://www.youtube.com/playlist?list=PLdb-TcK6Aqj1Kg6pV3zlrpUnPIRwG2$_O$x

Python for Software Development Videos:
https://www.youtube.com/playlist?list=PLdb-TcK6Aqj0E01L69fySfBSemVTCi$_l$L

Raspberry Pi and Python:
https://www.youtube.com/playlist?list=PLdb-TcK6Aqj3Sf5omYT-MLmxckclHi2i7

Internet of Things with Python:
https://www.youtube.com/playlist?list=PLdb-TcK6Aqj2zP8b1JjzUcFGzTVG13BPh

YouTube Channel @Industrial IT and Automation
https://www.youtube.com/IndustrialITandAutomation

Programming

The way we create software today has changed dramatically the last 30 years,
from the childhood of personal computers in the early 80s to today's powerful
devices such as Smartphones, Tablets and PCs.

The Internet has also changed the way we use devices and software. We still
have traditional desktop applications, but Web Sites, Web Applications and so-
called Apps for Smartphones, etc. are dominating the software market today.

We need to find and learn Programming Languages that are suitable for the
New Age of Programming.

We have today several thousand different Programming Languages today. I
guess you will need to learn more than one Programming Language to survive

in today's software market.

You find lots of Programming Resources here:
https://www.halvorsen.blog/documents/programming/

Software Engineering

Software Engineering is the discipline for creating software applications. A systematic approach to the design, development, testing, and maintenance of software.

The main parts or phases in the Software Engineering process are:

- Planning

- Requirements Analysis

- Design

- Implementation

- Testing

- Deployment and Maintenance

You find lots of Software Engineering Resources here:
https://www.halvorsen.blog/documents/programming/software$_e ngineering/$

Contents

8

Part I

Getting Started with Python

Chapter 1

Introduction

With this textbook you will learn basic Python programming. The textbook contains lots of examples and self-paced tasks that the users should go through and solve in their own pace.

You will find additional resources on my blog/web site [1].
https://www.halvorsen.blog

My Web Site about Python is:
https://www.halvorsen.blog/documents/programming/python/

See Figure 1.1

1.1 The New Age of Programming

The way we create software today has changed dramatically the last 30 years, from the childhood of personal computers in the early 80s to today's powerful devices such as Smartphones, Tablets and PCs.

The Internet has also changed the way we use devices and software. We still have traditional desktop applications, but Web Sites, Web Applications and so-called Apps for Smartphones, etc. are dominating the software market today.

We need to find and learn Programming Languages that are suitable for the New Age of Programming.

We have today several thousand different Programming Languages, so why should we learn Python? I guess you will need to learn more than one Programming Language to survive in today's software market. Python is easy to learn, so it it a good starting point for new programmers.

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991 [2].

Figure 1.1: Web Site - Python

Python is a fairly old Programming Language (1991) compared to many other Programming Languages like C# (2000), Swift (2014), Java (1995), PHP (1995).

Python has during the last 10 years become more and more popular. Today, Python has become one of the most popular Programming Languages.

There are many different rankings regarding which programming language which is most popular. In most of these ranking, Python is in top 10.

One of these rankings is the IEEE Spectrum's ranking of the top programming languages [3].

From this ranking we see that Python is the most popular Programming Language in 2018. See Figure 1.2

As we see in Figure 1.2 they categorize the different Programming Languages into the following categories:

- Web

Language Rank	Types	Spectrum Ranking
1. Python	⊕ 🖥 ▦	100.0
2. C++	📱 🖥 ▦	99.7
3. Java	⊕ 📱 🖥	97.5
4. C	📱 🖥 ▦	96.7
5. C#	⊕ 📱 🖥	89.4
6. PHP	⊕	84.9
7. R	🖥	82.9
8. JavaScript	⊕ 📱	82.6
9. Go	⊕ 🖥	76.4
10. Assembly	▦	74.1

Figure 1.2: The Most Popular Programming Languages

- Mobile

- Enterprise

- Embedded

According to Figure 1.2 we see that Python can be used to program Web Applications, Enterprise Applications and Embedded Applications.

So far Python is not used or not optimized for creating Mobile Applications. We have today 2 major Mobile platforms; iOS Applications are mainly programmed with the Swift Programming language, while Android Applications are mainly programmed with either Java or Kotlin.

Another survey is the "Stack Overflow Developer Survey 2018" [4]. See Figure 1.3.

As we can see from [5] and Figure 1.4, Python becomes more and more popular year by year.

Based on Figure 1.4, the source [5] try to predict the future of Python, see Figure 1.5.

Based on the surveys and statistics mention above, obviously Python is a programming language that you should learn.

Lets summarize:

- Python is fun to learn and use and it is also named after the British comedy group called Monty Python.

- Python has a simple and flexible code structure and the code is easy to read.

17
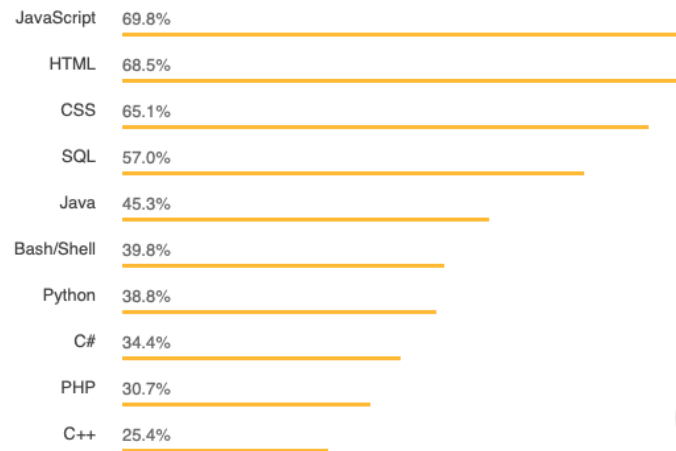
Figure 1.3: The Top Programming Languages - Stack Overflow Survey

- Python is highly extendable due to its high number of free available Python Packaged and Libraries

- Python can be used on all platforms (Windows, macOS and Linux).

- Python is multi-purpose and can be used for to program Web Applications, Enterprise Applications and Embedded Applications, and within Data Science and Engineering Applications.

- The popularity of Python is growing fast.

- Python is open source and free to use

- The growing Python community makes it easy to find documentation, code examples and get help when needed

In general, Python is a multipurpose programming language that can be used in many situations. But there is not one programming language which is best in all kind of situations, so it is important that you know about and have skills in different languages.

My list of recommendations (one of many):

- Visual Studio and C

- LabVIEW - a graphical programming language well suited for hardware integration, taking measurements and data logging

- MATLAB - Numerical calculations and Scientific computing

- Python - Numerical calculations, and Scientific computing, etc.

- Web Programming, such as HTML, CSS, JavaScript and a Server-side framework/programming language like PHP, ASP.NET (C or VB.NET), Django (Python based)

18

Growth of major programming languages

Based on Stack Overflow question views in World Bank high-income countries
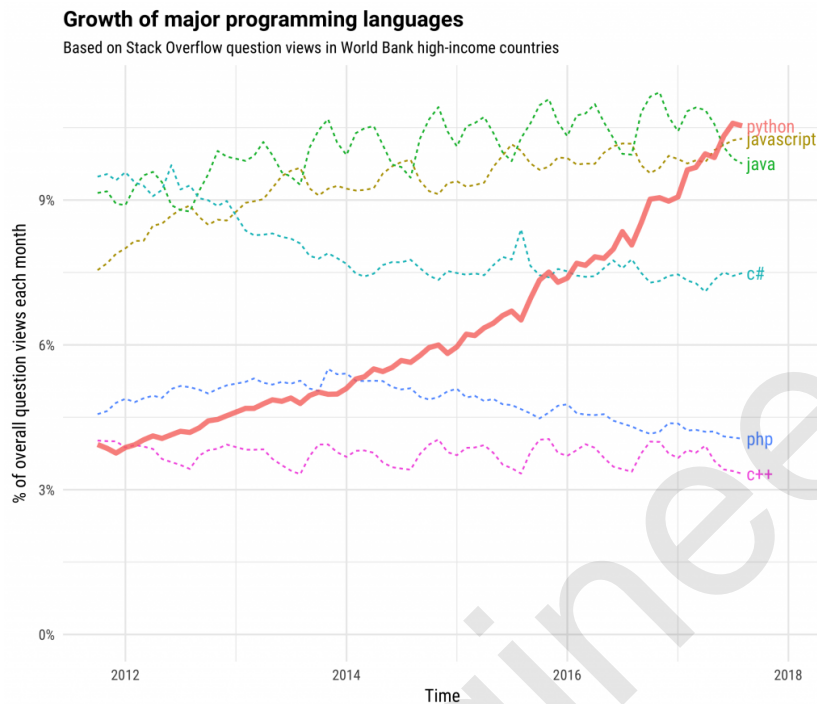
Figure 1.4: The Incredible Growth of Python

- Databases (such as SQL Server and MySQL) and using the Structured Query Language (SQL) or the upcoming NoSQL databases

- App Development for the 2 main platforms iOS (XCode using the Swift Programming Language) and Android (Android Studio using the Java Programming Language or Kotlin Programming language)

If you have skills in most of the tools, programming languages and frameworks mention above, you are well suited for working as a full-time programmer or software engineer.

1.2 MATLAB

If you are looking for MATLAB, please see the following:
https://www.halvorsen.blog/documents/programming/matlab/

19

Projections of future traffic for major programming languages

Future traffic is predicted with an STL model, along with an 80% prediction interval.
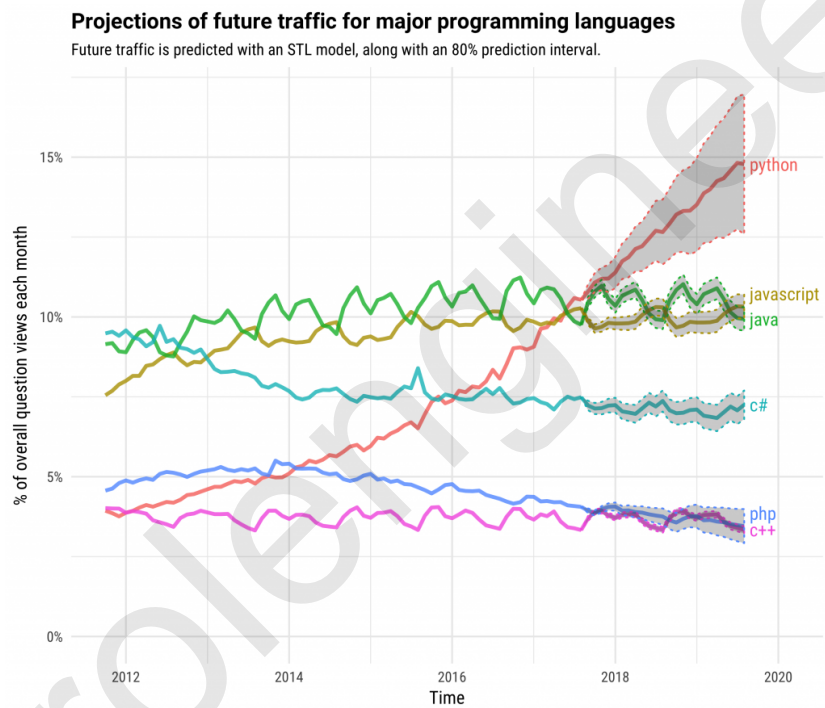
Figure 1.5: The Future of Python

Chapter 2

What is Python?

2.1 Introduction to Python

Python is an open source and cross-platform programming language, that has become increasingly popular over the last ten years. It was first released in 1991. Latest version is 3.7.0. **CPython** is the reference implementation of the Python programming language. Written in C, CPython is the default and most widely-used implementation of the language.

Python is a multi-purpose programming languages (due to its many extensions), examples are scientific computing and calculations, simulations, web development (using, e.g., the Django Web framework), etc.

Python Home Page [6]:
https://www.python.org
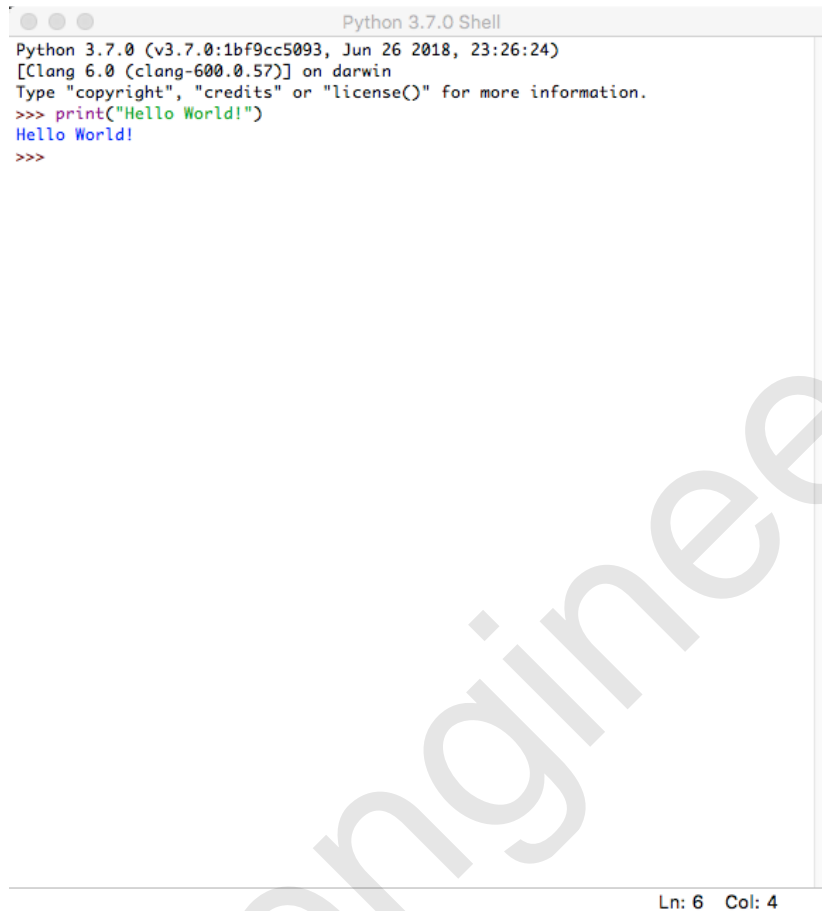
The programming language is maintained and available from (Python Software Foundation): https://www.python.org Here you can download the basic Python features in one package, which includes the Python programming language interpreter, and a basic code editor, or an integrated development environment, called IDLE. See Figure 2.1

But this is just the Python core, i.e. the interpreter a very basic editor, and the minimum needed to create basic Python programs.

Typically you will need more features for solving your tasks. Then you can install and use separate Python packages created by third parties. These packages need to be downloaded and installed separately (typically you use something called PIP), or you choose to use, e.g., a distribution package like Anaconda.

Python is an object-oriented programming language (OOP), but you can use Python in basic application without the need to know about or use the object-oriented features in Python.

Python is an interpreted programming language, this means that as a developer

Figure 2.1: IDLE - Basic Python Editor

you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed. Depending on the Editor you are using, this is either done automatically, or you need to do it manually.

Here are some important Python sources: [6], [7], [8].

2.1.1 Interpreted vs. Compiled

What are the differences between Interpreted programming languages and Compiled programming languages? What kind should you choose, and why should you bother?

Programming languages generally fall into one of two categories: Compiled or Interpreted. With a compiled language, code you enter is reduced to a set of machine-specific instructions before being saved as an executable file.
Both approaches have their advantages and disadvantages.

22

With interpreted languages, the code is saved in the same format that you entered. Compiled programs generally run faster than interpreted ones because interpreted programs must be reduced to machine instructions at run-time. It is usually easier to develop applications in an interpreted environment because you don't have to recompile your application each time you want to test a small section.

Python is an interpreted programming language, while e.g., C/C++ are translated by running the source code through a compiler, i.e., C/C++ are compiled languages.

Interpreted languages, in contrast, must be parsed, interpreted, and executed each time the program is run.

Another example of an interpreted programming language is PHP, which is mainly used to create dynamic web pages and web applications.

Compiled languages are all translated by running the source code through a compiler. This results in very efficient code that can be executed any number of times. The overhead for the translation is incurred just once, when the source is compiled; thereafter, it need only be loaded and executed.

During the design of an application, you might need to decide whether to use a compiled language or an interpreted language for the application source code.

Interpreted languages, in contrast, must be parsed, interpreted, and executed each time the program is run

Thus, an interpreted language is generally more suited for doing "ad hoc" calculations or simulations, while compiled languages are better for permanent applications where speed is in focus.

2.2   Python Packages

With Python you don't get so much out of the box. Instead of having all of its functionality built into its core, you need to install different packages for different topics.

This approach has advantages and disadvantages. An disadvantage is that you need to install these packages separately and then later import these modules in your code.

This is also typical approach for open source software, because everybody can create their own Python packages and distribute them. In that way you also find Python packages for almost everything, from Scientific Computing to Web Development.

These packages need to be downloaded and installed separately, or you choose to use, e.g., a distribution package like Anaconda, where you typically get the packages you need for scientific computing. With Anaconda you typically get the same features as with MATLAB.

Lots of Python packages exists, depending on what you are going to solve. We have Python packages for Desktop GUI Development, Database Development, Web Development, Software Development, etc.

See an overview of Applications for Python:
https://www.python.org/about/apps/

See also the Python Package Index (PyPI) web site:
https://pypi.org

Here you can search for, download and install many hundreds Python Packages within different topics and applications. You can also make your own Python Packages and distribute them here.

2.2.1 Python Packages for Science and Numerical Computations

Some important Python Packages for Science and Numerical Computations are:

- **NumPy** - NumPy is the fundamental package for scientific computing with Python [9]

- **SciPy** - SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering. [9]

- **Matplotlib** - Matplotlib is a Python 2D plotting library. [10]

- **Pandas** - Pandas Python Data Analysis Library [11]

These packages need to be downloaded and installed separately, or you choose to use, e.g., a distribution package like Anaconda, where you typically get the packages you need for scientific computing. With Anaconda you typically get the same features as with MATLAB.

2.3 Anaconda

Anaconda is a distribution package, where you get Python compiler, Python packages and the Spyder editor, all in one package.

Anaconda includes Python, the Jupyter Notebook, and other commonly used packages for scientific computing and data science.

They offer a free version (Anaconda Distribution) and a paid version (Enterprise) Anaconda is available for Windows, macOS, and Linux

Web:
https://www.anaconda.com

Wikipedia:
https://en.wikipedia.org/wiki/Anaconda$(Python_distribution)$

Spyder and the Python packages (NumPy, SciPy, Matplotlib, ...) mention above +++ are included in the Anaconda Distribution.

2.4 Python Editors

An Editor is a program where you create your code (and where you can run and test it). Most Editors have also features for Debugging. For simple Python programs you can use the IDLE Editor, but for more advanced programs a better editor is recommended.

Examples of Python Editors:

- Python IDLE
- Visual Studio Code
- Thonny
- Spyder
- Visual Studio
- PyCharm
- Wing Python IDE
- Jupyter Notebook

These editors are shortly described below and in more detail later in this textbook.

Which editor you should use depends on your background, what kind of code editors you have used previously, your programming skills, what your are going to develop in Python, etc.

2.4.1 Python IDLE

The programming language is maintained and available from (Python Software Foundation): https://www.python.org Here you can download the basic Python features in one package, which includes the Python programming language interpreter, and a basic code editor, or an integrated development environment, called IDLE. See Figure 2.1

Web:
https://www.python.org

2.4.2 Visual Studio Code

Visual Studio Code is a source code editor developed by Microsoft for Windows, Linux and macOS.

Web:
https://code.visualstudio.com

Resources: Getting Started with Python in Visual Studio Code

2.4.3 Thonny

Thonny is a Python IDE for beginners. Thonny is a very simple and basic Python editor which is highly recommended for new Python users. With Thonny you can easily install Python Libraries/Packages using a Graphical User Interface without the need of using more cryptic PIP commands in the Command window/Terminal.

Thonny is also the default Python Editor on the Raspberry Pi OS.

Web: https://thonny.org
Wikipedia: https://en.wikipedia.org/wiki/Thonny

2.4.4 Spyder

Spyder is an open source cross-platform integrated development environment (IDE) for scientific programming in the Python language.

Web:
https://www.spyder-ide.org

Wikipedia:
https://en.wikipedia.org/wiki/Spyder$_(software)$

Spyder is included in the Anaconda Distribution.

2.4.5 Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps. The deafult (main) programming language in Visual studio is C, but many other programming languages are supported.

Visual studio is available for Windows and macOS.

Visual Studio (from 2017), has integrated support for Python, it is called "Python Support in Visual Studio".

Web:
https://visualstudio.microsoft.com

Wikipedia:
https://en.wikipedia.org/wiki/Microsoft$_Visual_studio$

2.4.6 PyCharm

PyCharm is cross-platform, with Windows, macOS and Linux versions. The Community Edition is free to use, while the Professional Edition (paid version) has some extra features.

Web:
https://www.jetbrains.com/pycharm/

2.4.7 Wing Python IDE

The Wing Python IDE family of integrated development environments (IDEs) from Wingware were created specifically for the Python programming language.

3 different version of Wing exists [12]:

- **Wing 101** – a very simplified free version, for teaching beginning programmers

- **Wing Personal** – free version that omits some features, for students and hobbyists

- **Wing Pro** – a full-featured commercial (paid) version, for professional programmers

2.4.8 Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and text.

Web:
http://jupyter.org

Wikipedia:
https://en.wikipedia.org/wiki/Project$_Jupyter$

27

2.5   Resources

Here are some useful Python resources:

- The official Python Tutorial
  - https://docs.python.org/3.7/tutorial/index.html

- The official Python Documentation
  - https://docs.python.org/3.7/index.html

- Python Tutorial (w3schools.com) [13]
  - https://www.w3schools.com/python/

2.6   Installing Python

The Python programming language is maintained and available from (Python Software Foundation):

https://www.python.org

Here you can download the basic Python features in one package, which includes the Python programming language interpreter, and a basic code editor, or an integrated development environment, called IDLE. See Figure 2.1

For basic Python programming this is good enough.

For more advanced Python Programming you typically need a better Code Editor and additional Packages.

For the basic Python examples in the beginning, the basic Python software from:
https://www.python.org is good enough.

I suggest you start with the basic Python software in order to learn the basics, then you can upgrade to a better Editor, install addition Python packages (either manually or or install Anaconda where "everything" is included).

2.6.1   Python Windows 10 Store App

Python 3.7 is also available in the Microsoft Store for Windows 10.

The Microsoft Store version of Python 3.7 is a simplified installer for running scripts and packages.

Microsoft Store version of Python 3.7 is very basic but it's good enough to run the simple scripts.

Python 3.7 Microsoft Store edition will receive all updates automatically when they are released and no manual action is required from your end.

In order to install the Microsoft Store version of Python just open Microsoft Store in Windows 10 and search for Python.

2.6.2 Installing Anaconda

The Spyder Code Editor and the Python packages (such as NumPy, SciPy, matplotlib, etc) are included in the Anaconda Distribution.

Download and install from:
https://www.anaconda.com

2.6.3 Installing Visual Studio Code

Visual Studio Code code is a simple and easy to use editor that can be used for many different programming languages.

Download and install from:
https://code.visualstudio.com

Getting Started with Python in Visual Studio Code:
https://code.visualstudio.com/docs/python/python-tutorial
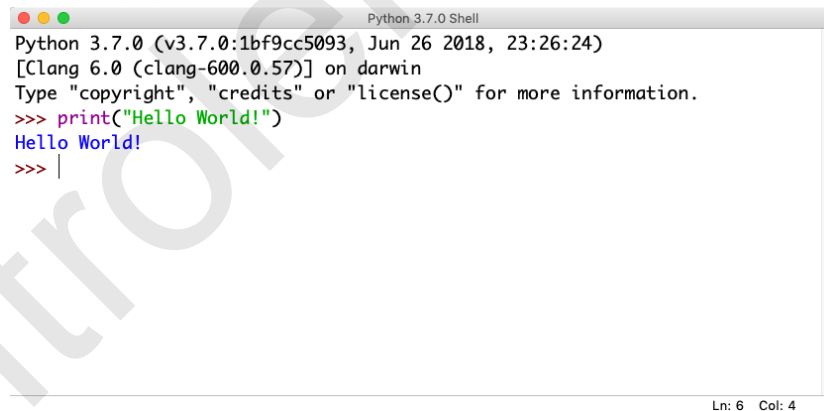
Chapter 3

Start using Python

In this chapter we will start to use Python in some simple examples.

3.1 Python IDE

The basic code editor, or an integrated development environment, called IDLE. See Figure 3.1.

Other Python Editors will be discussed more in detail later. For now you can use the basic Python IDE (IDLE) or Spyder if you have installed the Anaconda distribution package.


```
Python 3.7.0 Shell
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello World!")
Hello World!
>>>
```

Figure 3.1: Python Shell / Python IDLE Editor

3.2 My first Python program

We will start using Python and create some code examples.

Example 3.2.1. Plotting in Python

Lets open your Python Editor and type the following:

```
1 print ( " Hello  World ! " )
```
Listing 3.1: Hello World Python Example

[End of Example]

An extremely useful command is **help()**, which enters a help functionality to explore all the stuff python lets you do, right from the interpreter. Press q to close the help window and return to the Python prompt.

You can use Python in different ways, either in "interactive" mode or in "Scripting" mode.

The python program that you have installed will by default act as something called an interpreter. An interpreter takes text commands and runs them as you enter them - very handy for trying things out.

Yo can run Python interactively in different ways either using the Console which is part of the operating system or the Python IDLE and the Python Shell which is part of the basic Python installation from https://www.python.org.

3.3 Python Shell

In interactive Mode you use the Python Shell as seen in Figure 3.1.

Here you type one and one command at a time after the ">>>" sign in the Python Shell.

```
1 >>> print ( " Hello  World ! " )
```

3.4 Running Python from the Console

A console (or "terminal", or 'command prompt') is a textual way to interact with your OS (Operating System).

The python program that you have installed will by default act as something called an interpreter. An interpreter takes text commands and runs them as you enter them - very handy for trying things out.

Below we see how we can run Python from the Console which is part of the OS.

31

3.4.1 Opening the Console on macOS

The standard console on macOS is a program called Terminal. Open Terminal by navigating to Applications, then Utilities, then double-click the Terminal program. You can also easily search for it in the system search tool in the top right.

The command line Terminal is a tool for interacting with your computer. A window will open with a command line prompt message, something like this:

```
Last login: Tue Dec 11 08:33:51 on console
computername:~ username
```

Just type python at your console, hit Enter, and you should enter Python's Interpreter.
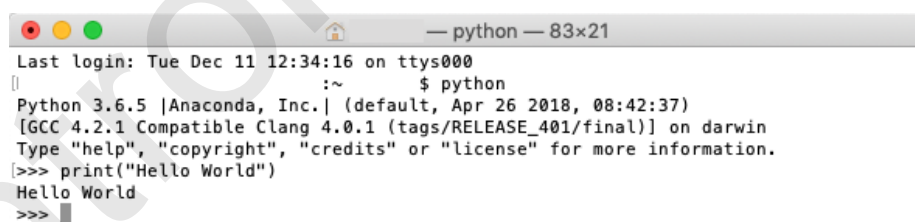
```
1 Last login: Tue Dec 11 12:34:16 on ttys000
2 Hans-Petter-Work-MacBook-Air:~ hansha$ python
3 Python 3.6.5 |Anaconda, Inc.| (default, Apr 26 2018, 08:42:37)
4 [GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on
      darwin
5 Type "help", "copyright", "credits" or "license" for more
      information.
6 >>>
```

The prompt >>> on the last line indicates that you are now in an interactive Python interpeter session, also called the "Python shell". This is different from the normal terminal command prompt!

You can now enter some code for python to run. Try:

```
>>> print("Hello World")
```

Se also Figure 3.2.


```
Last login: Tue Dec 11 12:34:16 on ttys000
[                    :~        $ python                                    ]
Python 3.6.5 |Anaconda, Inc.| (default, Apr 26 2018, 08:42:37)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
[>>> print("Hello World")                                                 ]
Hello World
>>>
```

Figure 3.2: Console macOS

Try other Python commands, e.g.:

```
1 >>> a = 5
2 >>> b = 2
3 >>> x = 5
4 >>> y = 3*a + b
5 >>> y
```

32

3.4.2 Opening the Console on Windows

Window's console is called the Command Prompt, named cmd. An easy way to get to it is by using the key combination Windows+R (Windows meaning the windows logo button), which should open a Run dialog. Then type cmd and hit Enter or click Ok.

You can also search for it from the start menu.

It should look like:

C:\ Users\myusername>

Just type python in the Command Prompt, hit Enter, and you should enter Python's Interpreter. See Figure 3.3.
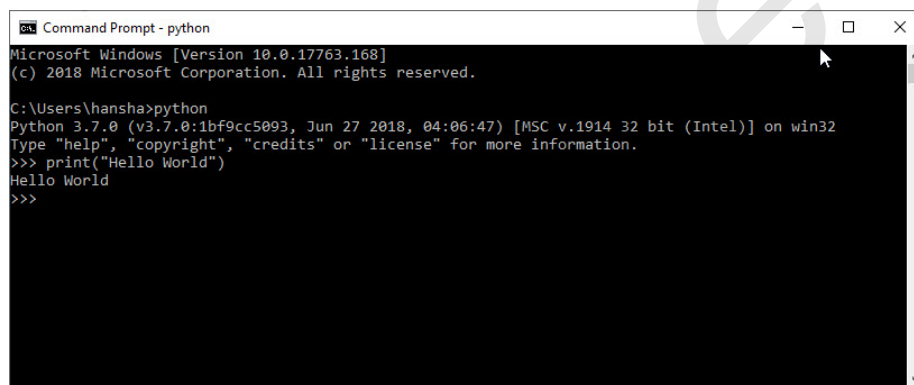
Figure 3.3: Command Prompt Windows

If you get an error message like this:

'python' is not recognized as an internal or external command, operable program or batch file.
Then you need to add Python to your path. See instructions below.

Note! This is also an option during the setup. While installing you can select "Add Python.exe to path". This option is by default set to "Off". To get that option you need to select "Customize", not using the "Default" installation.

3.4.3 Add Python to Path

In the Windows menu, search for "advanced system settings" and select View advanced system settings.

In the window that appears, click Environment Variables. . . near the bottom right. See Figure 3.4.
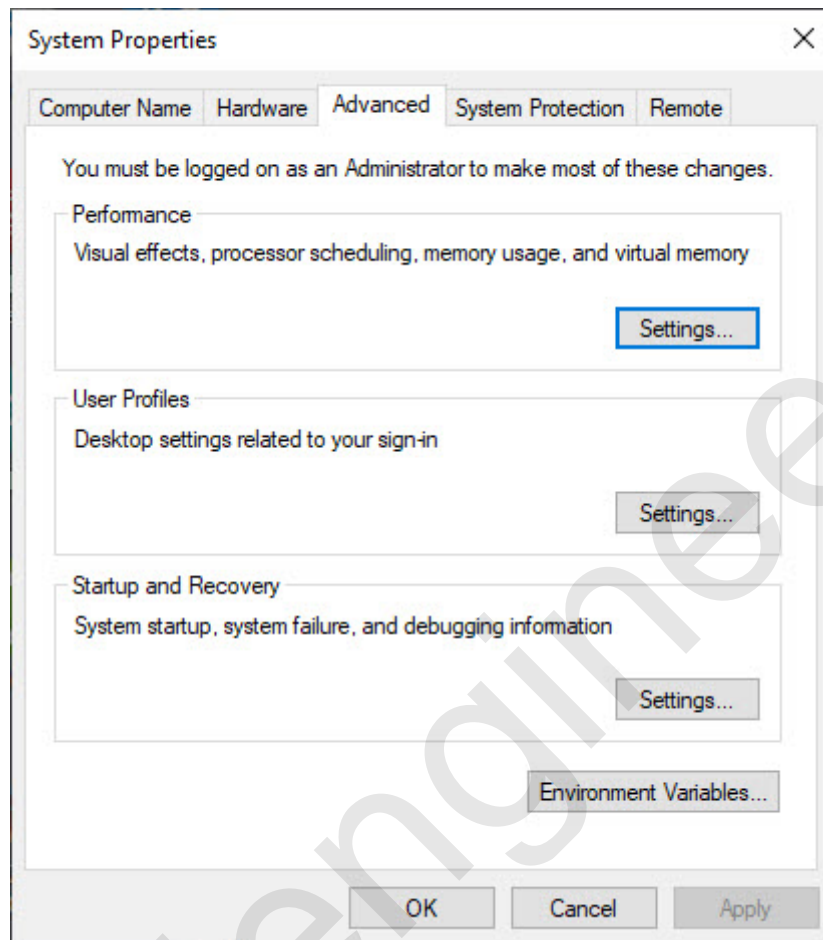
Figure 3.4: Windows System Properties

In the next window, find and select the user variable named Path and click Edit... to change its value. See Figure 3.5.

Select "New" and add the path where "python.exe" is located. See Figure 3.6.

The Default Location is:

```
C:\Users\user\AppData\Local\Programs\Python\Python37-32\
```

Click Save and open the Command Prompt once more and enter "python" to verify it works. See Figure 3.3.

34
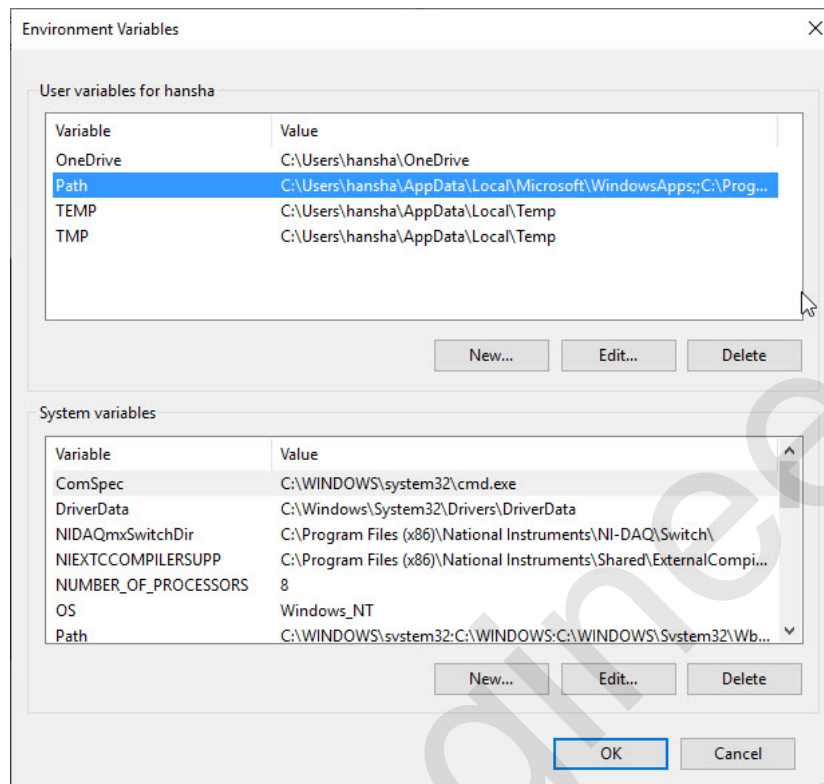
Figure 3.5: Windows System Properties

3.5 Scripting Mode

In "Scripting" mode you can write a Python Program with multiple Python commands and then save it as a file (.py).

3.5.1 Run Python Scripts from the Python IDLE

From the Python Shell you select File → New File, or you can open an existing Pytho program or Python Script by selecting File → Open...

Lets create a new Script and type in the following:

```
1 print ("Hello")
2 print ("World")
3 print ("How are you?")
```

In Figure 3.7 we see how this is done. As you see we can enter many Python commands that together makes a Python program or Python script.
From the Python Shell you select Run → Run Module or hit F5 in order to run or execute the Python Script. See Figure 3.8.

35
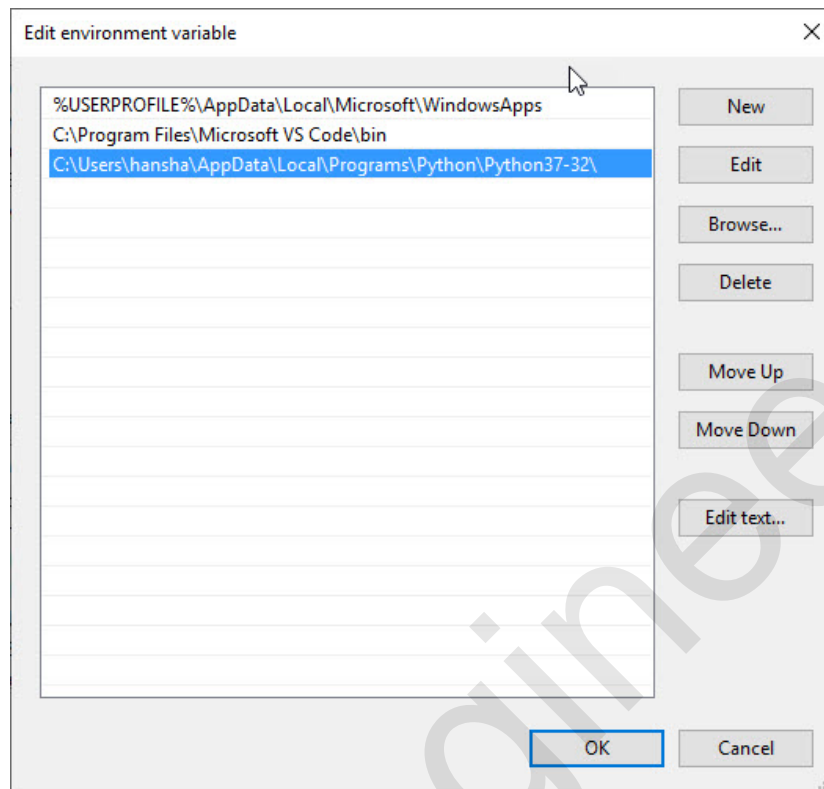
Figure 3.6: Windows System Properties

The IDLE editor is very basic, for more complicated tasks you typically may prefer to use another editor like Spyder, Visual Studio Code, etc.

3.5.2 Run Python Scripts from the Console (Terminal) macOS

From the Console (Terminal) on macOS:

```
1 $ cd /Users/username/Downloads
2 $ python helloworld.py
```

Note! Make sure you are at your system command prompt, which will have $ or > at the end, not in Python mode (which has >>> instead)!

See also Figure 3.9.
Then it responds with:

```
1 Hello
2 World
3 How are you?
```

36

Figure 3.7: Python Script

3.5.3 Run Python Scripts from the Command Prompt in Windows

From Command Prompt in Window:

```
1  > cd /
2  > cd Temp
3  > python helloworld.py
```

Note! Make sure you are at your system command prompt, which will have >
at the end, not in Python mode (which has >>> instead)!

See also Figure 3.10.
Then it responds with:
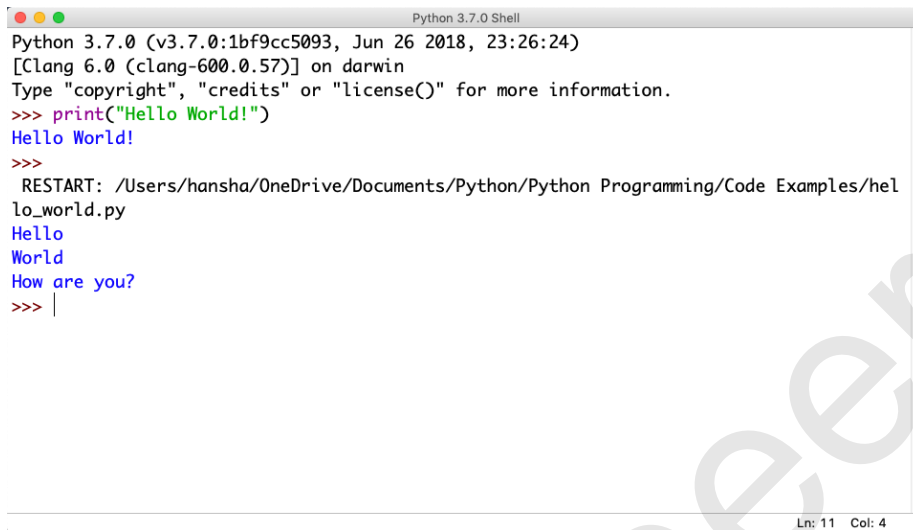
```
1  Hello
2  World
3  How are you?
```

3.5.4 Run Python Scripts from Spyder

If you have installed the Anaconda distribution package you can use the Spyder
editor. See 3.11.
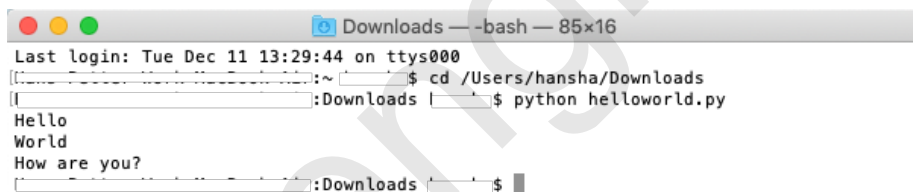
In the Spyder editor we have the Script Editor to the left and the interactive
Python Shell or the Console window to the right. See See 3.11.

37
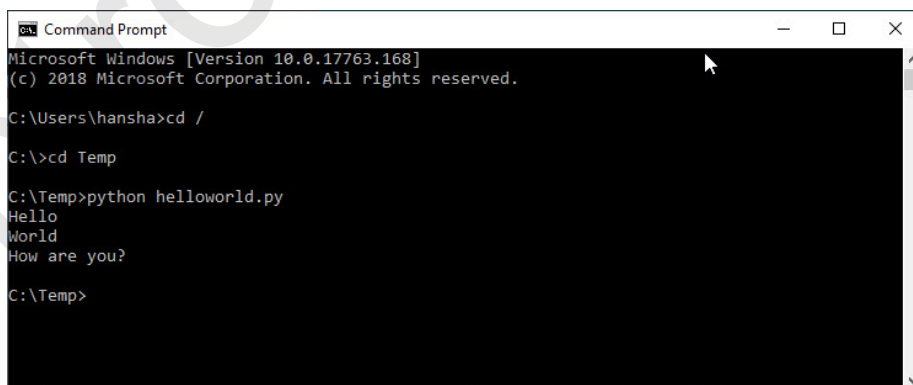
Figure 3.8: Running a Python Script

Figure 3.9: Running Python Scripts from Console window on macOS

Figure 3.10: Running Python Scripts from Console window on macOS

38
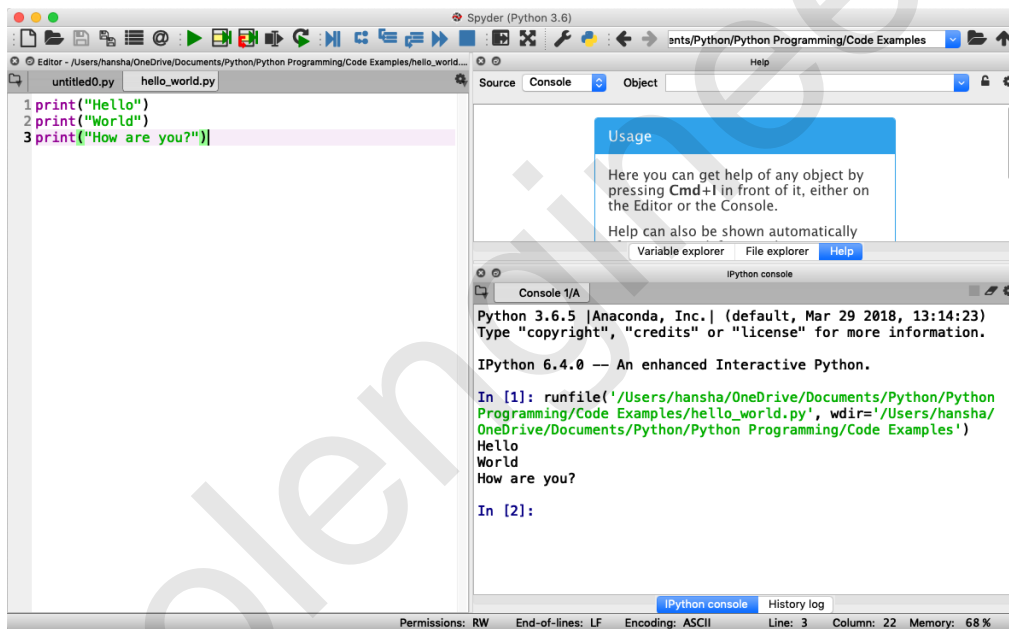
Figure 3.11: Running a Python Script in Spyder

Chapter 4

Basic Python Programming

4.1 Basic Python Program

We will start using Python and create some code examples.

We use the basic IDLE editor (or another Python Editor)

Example 4.1.1. Hello World Example

Lets open your Python Editor and type the following:

```
1 print("Hello World!")
```
Listing 4.1: Hello World Python Example

[End of Example]

4.1.1 Get Help

An extremely useful command is **help()**, which enters a help functionality to explore all the stuff python lets you do, right from the interpreter.

Press q to close the help window and return to the Python prompt.

4.2 Variables

Variables are defined with the assignment operator, "=". Python is dynamically typed, meaning that variables can be assigned without declaring their type, and that their type can change. Values can come from constants, from computation involving values of other variables, or from the output of a function.

40

Example 4.2.1. Creating and using Variables in Python

We use the basic IDLE (or another Python Editor) and type the following:

```
1 >>> x = 3
2 >>> x
3 3
```

Listing 4.2: Using Variables in Python

Here we define a variable and sets the value equal to 3 and then print the result to the screen.

[End of Example]

You can write one command by time in the IDLE. If you quit IDLE the variables and data are lost. Therefore, if you want to write a somewhat longer program, you are better off using a text editor to prepare the input for the interpreter and running it with that file as input instead. This is known as creating a script.

Python scripts or programs are save as a text file with the extension **.py**

Example 4.2.2. Calculations in Python

We can use variables in a calculation like this:

```
1 x = 3
2 y = 3*x
3 print(y)
```

Listing 4.3: Using and Printing Variables in Python

We can implement the formula $y = ax + b$ like this:

```
1 a = 2
2 b = 5
3 x = 3
4
5 y = a*x + b
6
7 print(y)
```

Listing 4.4: Calculations in Python
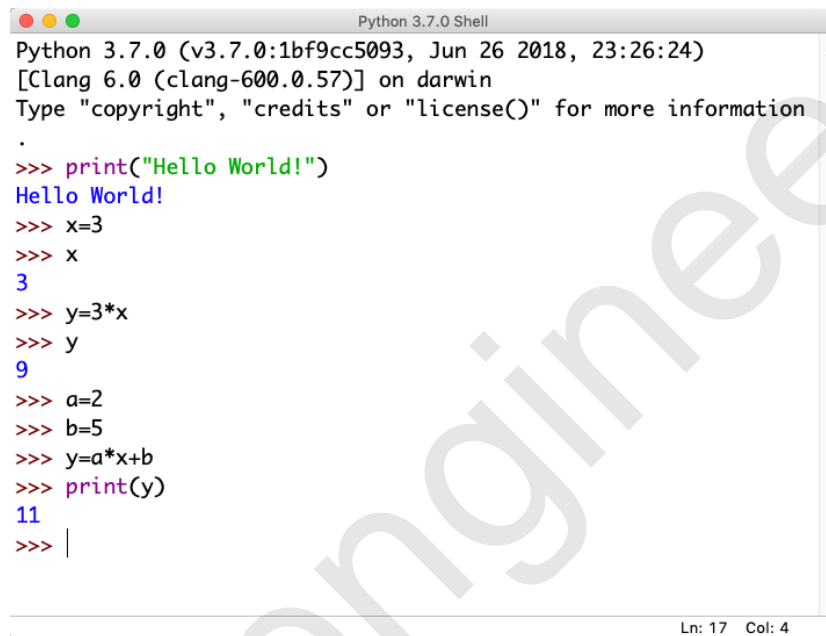
As seen in the examples, you can use the *print()* command in order to show the values on the screen.

[End of Example]

41

A variable can have a short name (like x and y) or a more descriptive name (sum, amount, etc).

You don need to define the variables before you use them (like you need to to in, e.g., C/C++/C).

Figure 4.1 show these examples using the basic IDLE editor.

Figure 4.1: Basic Python

Here are some basic rules for Python variables:

- A variable name must start with a letter or the underscore character

- A variable name cannot start with a number

- A variable name can only contain alpha-numeric characters (A-z, 0-9) and underscores

- Variable names are case-sensitive, e.g., amount, Amount and AMOUNT are three different variables.

4.2.1 Numbers

There are three numeric types in Python:

- int

- float

- complex

42

Variables of numeric types are created when you assign a value to them, so in normal coding you don't need to bother.

Example 4.2.3. Numeric Types in Python

```
1  x = 1         # int
2  y = 2.8       # float
3  z = 3 + 2j    # complex
```

Listing 4.5: Numeric Types in Python
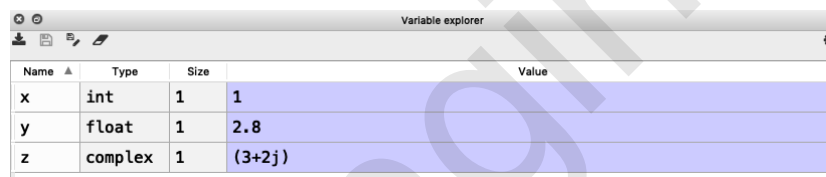
This means you just assign values to a variable without worrying about what kind of data type it is.

```
1  print(type(x))
2  print(type(y))
3  print(type(z))
```

Listing 4.6: Check Data Types in Python

If you use the Spyder Editor, you can see the data types that a variable has using the Variable Explorer (Figure 4.2):

Figure 4.2: Variable Editor in Spyder

[End of Example]

4.2.2 Strings

Strings in Python are surrounded by either single quotation marks, or double quotation marks. 'Hello' is the same as "Hello".
Strings can be output to screen using the print function. For example: print("Hello").

Example 4.2.4. Using Strings in Python

Below we see examples of using strings in Python:

```
1  a = "Hello World!"
2
3  print(a)
4
5  print(a[1])
6  print(a[2:5])
7  print(len(a))
8  print(a.lower())
```

43

```
9   print(a.upper())
10  print(a.replace("H", "J"))
11  print(a.split(" "))
```

Listing 4.7: Strings in Python

As you see in the example, there are many built-in functions form manipulating strings in Python. The Example shows only a few of them.

Strings in Python are arrays of bytes, and we can use index to get a specific character within the string as shown in the example code.

[End of Example]

4.2.3 String Input

Python allows for command line input.

That means we are able to ask the user for input.

Example 4.2.5. String Input in Python

The following example asks for the user's name, then, by using the input() method, the program prints the name to the screen:

```
1   print("Enter your name:")
2   x = input()
3   print("Hello, " + x)
```

Listing 4.8: String Input

[End of Example]

4.3 Built-in Functions

Python consists of lots of built-in functions. Some examples are the print function that we already have used (perhaps without noticing it is actually a Built-in function).

Python also consists of different Modules, Libraries or Packages. These Modules, Libraries or Packages consists of lots of predefined functions for different topics or areas, such as mathematics, plotting, handling database systems, etc. See Section 4.4 for more information and details regarding this.

In another chapter we will learn to create our own functions from scratch.

4.4 Python Standard Library

Python allows you to split your program into modules that can be reused in other Python programs. It comes with a large collection of standard modules that you can use as the basis of your programs.

The **Python Standard Library** consists of different modules for handling file I/O, basic mathematics, etc. You don't need to install these separately, but you need to important them when you want to use some of these modules or some of the functions within these modules.

The math module has all the basic math functions you need, such as: Trigonometric functions: *sin(x)*, *cos(x)*, etc. Logarithmic functions: *log()*, *log10()*, etc. Constants like *pi*, *e*, *inf*, *nan*, etc.

Example 4.4.1. Using the math module

We create some basic examples how to use a Library, a Package or a Module:

If we need only the sin() function, we can do like this:

```python
from math import sin

x = 3.14
y = sin(x)

print(y)
```

If we need a few functions, we can do like this:

```python
from math import sin, cos

x = 3.14
y = sin(x)
print(y)

y = cos(x)
print(y)
```

If we need many functions, we can do like this:

```python
from math import *

x = 3.14
y = sin(x)
print(y)

y = cos(x)
print(y)
```

We can also use this alternative:

```python
import math

x = 3.14
y = math.sin(x)

print(y)
```

45

We can also write it like this:

```
1  import math as mt
2
3  x = 3.14
4  y = mt.sin(x)
5
6  print(y)
```

[End of Example]

There are advantages and disadvantages with the different approaches. In your program you may need to use functions from many different modules or packages. If you import the whole module instead of just the function(s) you need you use more of the computer memory.

Very often we also need to import and use multiple libraries where the different libraries have some functions with the same name but different use.

Other useful modules in the **Python Standard Library** are **statistics** (where you have functions like *mean()*, *stdev()*, etc.)

For more information about the functions in the **Python Standard Library**, see:
https://docs.python.org/3/library/index.html

4.5 Using Python Libraries, Packages and Modules

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This approach has advantages and disadvantages. A disadvantage is that you need to install these packages separately and then later import these modules in your code.

Some important packages are:

- **NumPy** - NumPy is the fundamental package for scientific computing with Python

- **SciPy** - SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

- **Matplotlib** - Matplotlib is a Python 2D plotting library

46

Lots of other packages exists, depending on what you are going to solve.

These packages need to be downloaded and installed separately, or you choose to use, e.g., a distribution package like Anaconda.

Here you find an overview of the **NumPy** library:
https://www.numpy.org

Here you find an overview of the **SciPy** library:
https://www.scipy.org

Here you find an overview of the **Matplotlib** library:
https://matplotlib.org

You will learn the basics features in all these libraries. We will use all of the in different examples and exercises throughout this textbook.

Example 4.5.1. Using libraries

In this example we use the NumPy library:

```
1  import numpy as np
2
3  x = 3
4
5  y = np.sin(x)
6
7  print(y)
```

In this example we use both the math module in the Python Standard Library and the NumPy library:

```
1   import math as mt
2   import numpy as np
3
4   x = 3
5
6   y = mt.sin(x)
7
8   print(y)
9
10
11  y = np.sin(x)
12
13  print(y)
```

Note! As seen in this example we use a function called sin() which exists both in the math module in the Python Standard Library and the NumPy library. In this case they give the same results. In this case the following code is not recommended:

```
1   from math import *
2   from numpy import *
3
4   x = 3
5
```

47

```
6  y = sin(x)
7
8  print(y)
9
10
11 y = sin(x)
12
13 print(y)
```

In this case it works, but assume you have 2 different functions with the same name that have different meaning in 2 different libraries.

[End of Example]

4.5.1 Python Packages

In addition to the Python Standard Library, there is a growing collection of several thousand components (from individual programs and modules to packages and entire application development frameworks), available from the **Python Package Index**.

Python Package Index (PYPI):
https://pypi.org

Here you can download and install individual Python packages.
An easy alternative is the Anaconda Distribution, where many of the most used Python packages are included.

Anaconda:
https://www.anaconda.com/distribution/

4.6 Plotting in Python

Typically you need to create some plots or charts. In order to make plots or charts in Python you will need an external library. The most used library is **Matplotlib**.

Matplotlib is a Python 2D plotting library

Here you find an overview of the Matplotlib library:
https://matplotlib.org

If you are familiar with MATLAB and basic plotting in MATLAB, using the Matplotlib is very similar.

The main difference from MATLAB is that you need to import the library, either the whole library or one or more functions.
For simplicity we import the whole library like this:

```
1 import matplotlib.pyplot as plt
```

Plotting functions that you will use a lot:

- plot()

- title()

- xlabel()

- ylabel()

- axis()

- grid()

- subplot()

- legend()

- show()

Lets create some basic plotting examples using the Matplotlib library:

Example 4.6.1. Plotting in Python

In this example we have two arrays with data. We want to plot x vs. y. We can assume x is a time series and y is the corresponding temperature in degrees Celsius.

```
1 import matplotlib.pyplot as plt
2
3 x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
4
5 y = [5, 2,4, 4, 8, 7, 4, 8, 10, 9]
6
7 plt.plot(x,y)
8 plt.xlabel('Time (s)')
9 plt.ylabel('Temperature (degC)')
10 plt.show()
```

We get the plot as shown in Figure 4.3.

We can also write like this:

```
1 from matplotlib.pyplot import *
2
3 x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
4 y = [5, 2,4, 4, 8, 7, 4, 8, 10, 9]
5
6 plot(x,y)
7 xlabel('Time (s)')
8 ylabel('Temperature (degC)')
9 show()
```

This makes the code simpler to read. one problem with this approach appears assuming we import and use multiple libraries and the different libraries have some functions with the same name but different use.

49
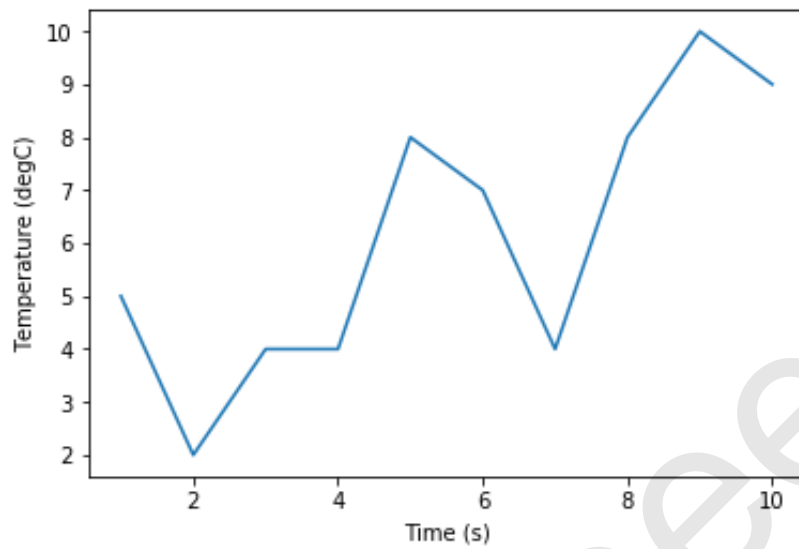
Figure 4.3: Plotting in Python

[End of Example]

We have used 4 basic plotting function in the Matplotlib library:

- plot()
- xlabel()
- ylabel()
- show()

Example 4.6.2. Plotting a Sine Curve

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  x = [0, 1, 2, 3, 4, 5, 6, 7]
5
6  y = np.sin(x)
7
8  plt.plot(x, y)
9  plt.xlabel('x')
10 plt.ylabel('y')
11 plt.show()
```

This gives the following plot (see Figure 4.4):

A better solution will then be:

50
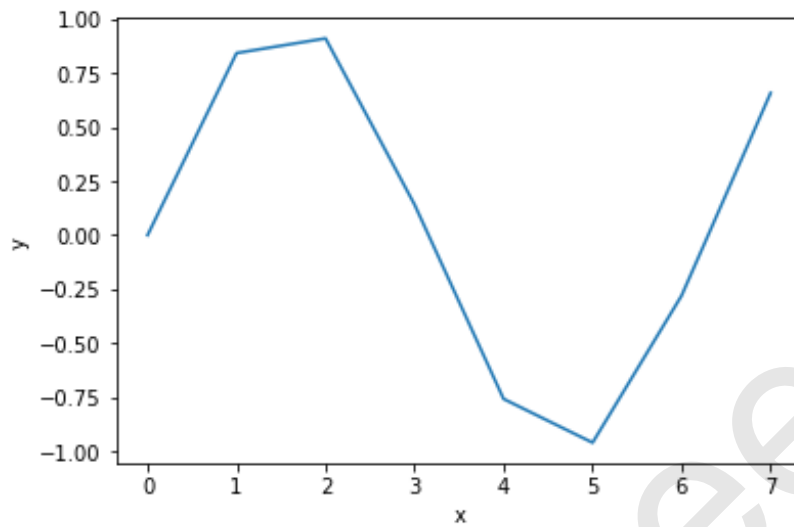
Figure 4.4: Plotting a Sine function in Python

```python
import matplotlib.pyplot as plt
import numpy as np

xstart = 0
xstop = 2*np.pi
increment = 0.1

x = np.arange(xstart, xstop, increment)

y = np.sin(x)

plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

This gives the following plot (see Figure 4.5):
If you want grids you can use the grid() function.

[End of Example]

4.6.1 Subplots

The subplot command enables you to display multiple plots in the same window. Typing "subplot(m,n,p)" partitions the figure window into an m-by-n matrix of small subplots and selects the subplot for the current plot. The plots are numbered along the first row of the figure window, then the second row, and so on. See Figure 4.6.

Example 4.6.3. Creating Subplots

51
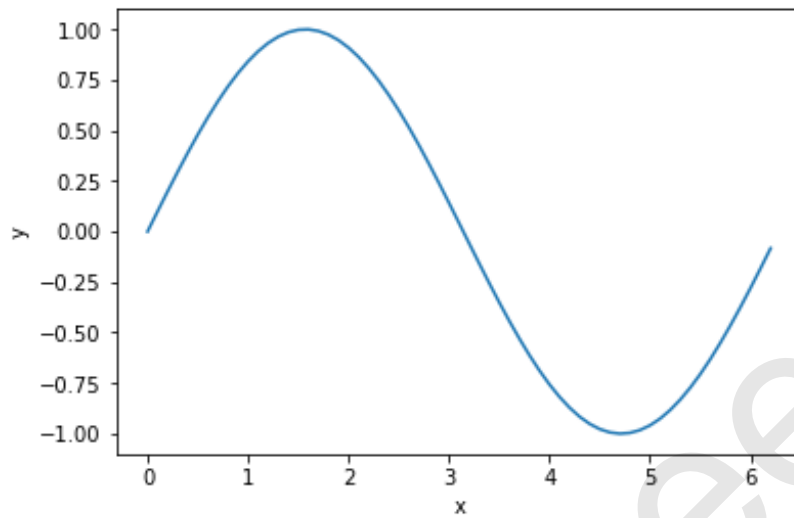
Figure 4.5: Plotting a Sine function in Python - Better Implementation

We will create and plot sin() and cos() in 2 different subplots.

```python
import matplotlib.pyplot as plt
import numpy as np

xstart = 0
xstop = 2*np.pi
increment = 0.1

x = np.arange(xstart, xstop, increment)

y = np.sin(x)

z = np.cos(x)


plt.subplot(2,1,1)
plt.plot(x, y, 'g')
plt.title('sin')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.grid()
plt.show()


plt.subplot(2,1,2)
plt.plot(x, z, 'r')
plt.title('cos')
plt.xlabel('x')
plt.ylabel('cos(x)')
plt.grid()
plt.show()
```

[End of Example]
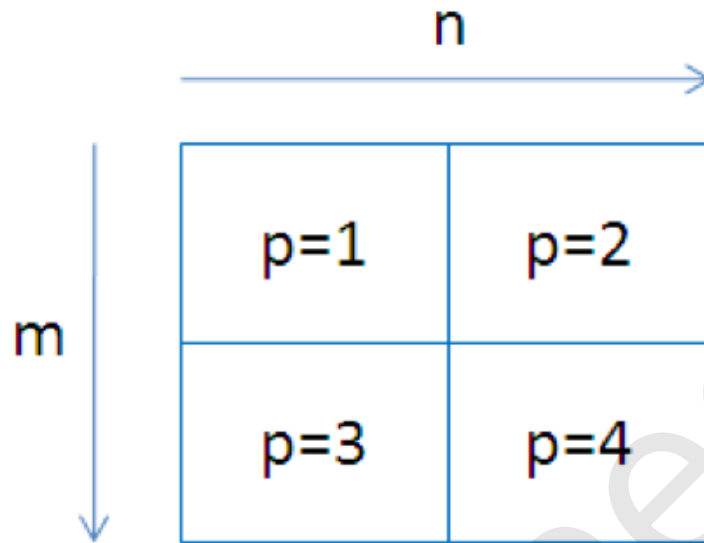
Figure 4.6: Creating Subplots in Python

4.6.2 Exercises

Below you find different self-paced Exercises that you should go through and solve on your own. The only way to learn Python is to do lots of Exercises!

Exercise 4.6.1. Create sin(x) and cos(x) in 2 different plots

Create sin(x) and cos(x) in 2 different plots.

You should use all the Plotting functions listed below in your code:

- plot()
- title()
- xlabel()
- ylabel()
- axis()
- grid()
- legend()
- show()

[End of Exercise]

Part II

Python Programming

Chapter 5

Python Programming

We have been through the basics in Python, such as variables, using some basic built-in functions, basic plotting, etc.

You may come far only using these thins, but to create real applications, you need to know about and use features like:

- If ... Else

- For Loops

- While Loops

- Arrays ...

If you are familiar with one or more other programming language, these features should be familiar and known to you. All programming languages have these features built-in, but the syntax is slightly different from one language to another.

5.1   If ... Else

An "if statement" is written by using the **if** keyword.

Here are some Examples how you use a If sentences in Python:

Example 5.1.1. Using If ... Else in Python

Using **If**:

```
1  a = 5
2  b = 8
3
4  if a > b:
5      print("a is greater than b")
6
7  if b > a:
8      print("b is greater than a")
9
10 if a == b:
```

55

```
11    print("a is equal to b")
```

Listing 5.1: If

Try to change the values for a and b.

Using **If - Else**:

```
1 a = 5
2 b = 8
3
4 if a > b:
5     print("a is greater than b")
6 else:
7     print("b is greater than a or a and b are equal")
```

Listing 5.2: If - Else

Using **Elif**:

```
1 a = 5
2 b = 8
3
4 if a > b:
5     print("a is greater than b")
6 elif b > a:
7     print("b is greater than a")
8 elif a == b:
9     print("a is equal to b")
```

Listing 5.3: Elif

Note! Python uses "elif" not "elseif" like many other programming languages do.

[End of Example]

5.2 Arrays

An array is a special variable, which can hold more than one value at a time.

Here are some Examples how you can create and use Arrays in Python:

Example 5.2.1. Arrays in Python

```
1 data = [1.6, 3.4, 5.5, 9.4]
2
3 N = len(data)
4
5 print(N)
6
7 print(data[2])
8
9 data[2] = 7.3
10
11 print(data[2])
```

56

```
12
13
14  for  x  in  data :
15      print ( x )
16
17
18  data . append ( 1 1 . 4 )
19
20
21  N =  len ( data )
22
23  print (N)
24
25
26  for  x  in  data :
27      print ( x )
```

Listing 5.4: Using Arrays in Python

You define an array like this:

```
1  data  =  [ 1 . 6 ,  3 . 4 ,  5 . 5 ,  9 . 4 ]
```

You can also use text like this:

```
1  c a r l i s t  =  [ " Volvo " ,  " Tesla " ,  " Ford " ]
```

You can use Arrays in Loops like this:

```
1  for  x  in  data :
2      print ( x )
```

You can return the number of elements in the array like this:

```
1  N =  len ( data )
```

You can get a specific value inside the array like this:

```
1  index  =  2
2  x =  cars [ index ]
```

You can use the append() method to add an element to an array:

```
1  data . append ( 1 1 . 4 )
```

[End of Example]

You have many built in methods you can use in combination with arrays, like sort(), clear(), copy(), count(), insert(), remove(), etc.

You should look into test all these methods.

5.3 For Loops

A For loop is used for iterating over a sequence. I guess all your programs will use one or more For loops. So if you have not used For loops before, make sure to learn it now.

Below you see a basic example how you can use a For loop in Python:

```
1  for i in range(1, 10):
2      print(i)
```

The For loop is probably one of the most useful feature in Python (or in any kind of programming language). Below you will see different examples how you can use a For loop in Python.

Example 5.3.1. Using For Loops in Python

```
1  data = [1.6, 3.4, 5.5, 9.4]
2
3  for x in data:
4      print(x)
5
6
7  carlist = ["Volvo", "Tesla", "Ford"]
8
9  for car in carlist:
10      print(car)
```

Listing 5.5: Using For Loops in Python

The range() function is handy to use in For Loops:

```
1  N = 10
2
3  for x in range(N):
4      print(x)
```

The **range()** function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

You can also use the range() function like this:

```
1  start = 4
2  stop= 12 #but not including
3
4  for x in range(start, stop):
5      print(x)
```

Finally, you can also use the range() function like this:

```
1  start = 4
2  stop = 12 #but not including
3  step = 2
4
5  for x in range(start, stop, step):
6      print(x)
```

You should try all these examples in order to learn the basic structure of a For loop.

[End of Example]

Example 5.3.2. Using For Loops for Summation of Data

You typically want to use a For loop for find the sum of a given data set.

```
data = [1, 5, 6, 3, 12, 3]

sum = 0

#Find the Sum of all the numbers
for x in data:
    sum = sum + x

print(sum)

#Find the Mean or Average of all the numbers

N = len(data)

mean = sum/N

print(mean)
```

This gives the following results:

```
30
5.0
```

[End of Example]

Example 5.3.3. Implementing Fibonacci Numbers Using a For Loop in Python

Fibonacci numbers are used in the analysis of financial markets, in strategies such as Fibonacci retracement, and are used in computer algorithms such as the Fibonacci search technique and the Fibonacci heap data structure.
They also appear in biological settings, such as branching in trees, arrangement of leaves on a stem, the fruitlets of a pineapple, the flowering of artichoke, an uncurling fern and the arrangement of a pine cone.

In mathematics, Fibonacci numbers are the numbers in the following sequence:
0, 1, 1, 2 ,3, 5, 8, 13, 21, 34, 55, 89, 144, . . .

By definition, the first two Fibonacci numbers are 0 and 1, and each subsequent number is the sum of the previous two.

Some sources omit the initial 0, instead beginning the sequence with two 1s.

59

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation

$$f_n = f_{n-1} + f_{n-2} \tag{5.1}$$

with seed values:

$$f_0 = 0, f_1 = 1$$

We will write a Python script that calculates the N first Fibonacci numbers. The Python Script becomes like this:

```
1  N = 10
2
3  fib1 = 0
4  fib2 = 1
5
6  print(fib1)
7  print(fib2)
8
9  for k in range(N-2):
10     fib_next = fib2 +fib1
11     fib1 = fib2
12     fib2 = fib_next
13     print(fib_next)
```

Listing 5.6: Fibonacci Numbers Using a For Loop in Python

Alternative solution:

```
1  N = 10
2
3  fib = [0, 1]
4
5
6  for k in range(N-2):
7      fib_next = fib[k+1] +fib[k]
8      fib.append(fib_next)
9
10 print(fib)
```

Listing 5.7: Fibonacci Numbers Using a For Loop in Python - Alt2

Another alternative solution:

```
1  N = 10
2
3  fib = []
4
5  for k in range(N):
6      fib.append(0)
7
8  fib[0] = 0
9  fib[1] = 1
10
```

60

```
11  for k in range(N−2):
12      fib[k+2] = fib[k+1] +fib[k]
13
14
15  print(fib)
```

Listing 5.8: Fibonacci Numbers Using a For Loop in Python - Alt3

Another alternative solution:

```
1  import numpy as np
2
3
4  N = 10
5
6  fib = np.zeros(N)
7
8  fib[0] = 0
9  fib[1] = 1
10
11  for k in range(N−2):
12      fib[k+2] = fib[k+1] +fib[k]
13
14
15  print(fib)
```

Listing 5.9: Fibonacci Numbers Using a For Loop in Python - Alt4

[End of Example]

5.3.1 Nested For Loops

In Python and other programming languages you can use one loop inside another loop.

Syntax for nested For loops in Python:

```
1  for iterating_var in sequence:
2      for iterating_var in sequence:
3          statements(s)
4      statements(s)
```

Simple example:

```
1  for i in range(1, 10):
2      for k in range(1, 10):
3          print(i, k)
```

Exercise 5.3.1. Prime Numbers

The first 25 prime numbers (all the prime numbers less than 100) are:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97

61

By definition a prime number has both 1 and itself as a divisor. If it has any other divisor, it cannot be prime.

A natural number (1, 2, 3, 4, 5, 6, etc.) is called a prime number (or a prime) if it is greater than 1 and cannot be written as a product of two natural numbers that are both smaller than it.

Create a Python Script where you find all prime numbers between 1 and 200.

Tip! I guess this can be done in many different ways, but one way is to use 2 nested For Loops.

[End of Exercise]

5.4 While Loops

The while loop repeats a group of statements an indefinite number of times under control of a logical condition.

Example 5.4.1. Using While Loops in Python

```
1  m = 8
2
3  while m > 2:
4      print (m)
5      m = m − 1
```

Listing 5.10: Using While Loops in Python

[End of Example]

5.5 Exercises

Below you find different self-paced Exercises that you should go through and solve on your own. The only way to learn Python is to do lots of Exercises!

Exercise 5.5.1. Plot of Dynamic System

Given the autonomous system:

$$\dot{x} = ax \tag{5.2}$$

Where:

$$a = -\frac{1}{T}$$

where T is the time constant.

The solution for the differential equation is:

$$x(t) = e^{at}x_0 \qquad\qquad (5.3)$$

Set T=5 and the initial condition x(0)=1.

Create a Script in Python (.py file) where you plot the solution x(t) in the time interval:

$$0 \le t \le 25$$

Add Grid, and proper Title and Axis Labels to the plot.

[End of Exercise]

Chapter 6

Creating Functions in Python

6.1 Introduction

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

Previously we have been using many of the built-in functions in Python

If you are familiar with one or more other programming language, creating and using functions should be familiar and known to you. All programming languages has the possibility to create functions, but the syntax is slightly different from one language to another.

Some programming languages uses the term Method instead of a Function. Functions and Methods behave in the same manner, but you could say that Methods are functions that belongs to a Class. We will learn more about Classes in Chapter 7.

Scripts vs. Functions

It is important to know the difference between a Script and a Function.

Scripts:

- A collection of commands that you would execute in the Editor

- Used for automating repetitive tasks

Functions:

- Operate on information (inputs) fed into them and return outputs

- Have a separate workspace and internal variables that is only valid inside the function

- Your own user-defined functions work the same way as the built-in functions you use all the time, such as plot(), rand(), mean(), std(), etc.

Python have lots of built-in functions, but very often we need to create our own functions (we could refer to these functions as user-defined functions)

In Python a function is defined using the **def** keyword:

```
1 def FunctionName:
2     <statement-1>
3     .
4     .
5     <statement-N>
6     return ...
```

Example 6.1.1. Basic Function

Below you see a simple function created in Python:

```
1 def add(x,y):
2
3     return x + y
```

Listing 6.1: Basic Python Function

The function adds 2 numbers. The name of the function is **add**, and it returns the answer using the **return** statement.

The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Note that you need to use a colon ":" at the end of line where you define the function.

Note also the indention used.

```
1 def add(x,y):
```

Here you see a Python script where we use the function:

```
1 def add(x,y):
2
3     return x + y
4
5
6 x = 2
7 y = 5
8
9 z = add(x,y)
10
11 print(z)
```

Listing 6.2: Creating and Using a Python Function

65

[End of Example]

Example 6.1.2. Create a Function in a separate File

We start by creating a separate Python File (**myfunctions.py**) for the function:

```python
def average(x,y):

    return (x + y)/2
```
Listing 6.3: Function calculating the Average

Next, we create a new Python File (e.g., **testaverage.py**) where we use the function we created:

```python
from myfunctions import average

a = 2
b = 3

c = average(a,b)

print(c)
```
Listing 6.4: Test of Average function

[End of Example]

6.2 Functions with multiple return values

Typically we want to return more than one value from a function.

Example 6.2.1. Create a Function Function with multiple return values

Create the following example:

```python
def stat(x):

    totalsum = 0

    #Find the Sum of all the numbers
    for x in data:
        totalsum = totalsum + x


    #Find the Mean or Average of all the numbers

    N = len(data)

    mean = totalsum/N


    return totalsum, mean


```

66

```
21  data = [1, 5, 6, 3, 12, 3]
22
23
24  totalsum, mean = stat(data)
25
26  print(totalsum, mean)
```

<div align="center">Listing 6.5: Function with multiple return values</div>

<div align="right">[End of Example]</div>

6.3 Exercises

Below you find different self-paced Exercises that you should go through and solve on your own. The only way to learn Python is to do lots of Exercises!

Exercise 6.3.1. Create Python Function

Create a function **calcaverage** that finds the average of two numbers.

<div align="right">[End of Exercise]</div>

Exercise 6.3.2. Create Python functions for converting between radians and degrees

Since most of the trigonometric functions require that the angle is expressed in radians, we will create our own functions in order to convert between radians and degrees.

It is quite easy to convert from radians to degrees or from degrees to radians.

We have that:

$$2\pi[radians] = 360[degrees] \tag{6.1}$$

This gives:

$$d[degrees] = r[radians] \times (\frac{180}{\pi}) \tag{6.2}$$

and

$$r[radians] = d[degrees] \times (\frac{\pi}{180}) \tag{6.3}$$

Create two functions that convert from radians to degrees (r2d(x)) and from degrees to radians (d2r(x)) respectively.

These functions should be saved in one Python file .py.

Test the functions to make sure that they work as expected.

<div align="center">67</div>

Exercise 6.3.3. Create a Function that Implementing Fibonacci Numbers

Fibonacci numbers are used in the analysis of financial markets, in strategies such as Fibonacci retracement, and are used in computer algorithms such as the Fibonacci search technique and the Fibonacci heap data structure.
They also appear in biological settings, such as branching in trees, arrangement of leaves on a stem, the fruitlets of a pineapple, the flowering of artichoke, an uncurling fern and the arrangement of a pine cone.

In mathematics, Fibonacci numbers are the numbers in the following sequence:
0, 1, 1, 2 ,3, 5, 8, 13, 21, 34, 55, 89, 144, . . .

By definition, the first two Fibonacci numbers are 0 and 1, and each subsequent number is the sum of the previous two.

Some sources omit the initial 0, instead beginning the sequence with two 1s.

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation

$$f_n = f_{n-1} + f_{n-2} \tag{6.4}$$

with seed values:

$$f_0 = 0, f_1 = 1$$

Create a Function that Implementing the N first Fibonacci Numbers

Exercise 6.3.4. Prime Numbers

The first 25 prime numbers (all the prime numbers less than 100) are:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97

By definition a prime number has both 1 and itself as a divisor. If it has any other divisor, it cannot be prime.

A natural number (1, 2, 3, 4, 5, 6, etc.) is called a prime number (or a prime) if it is greater than 1 and cannot be written as a product of two natural numbers that are both smaller than it.

Tip! I guess this can be implemented in many different ways, but one way is to use 2 nested For Loops.

Create a Python function where you check if a given number is a prime number or not.

You can check the function in the Command Window like this:

```
1  number = 4
2  checkifprime(number)
```

Then Python respond with True or False.

[End of Exercise]

Chapter 7

Creating Classes in Python

7.1 Introduction

Python is an object oriented programming (OOP) language. Almost everything in Python is an object, with its properties and methods.

The foundation for all object oriented programming (OOP) languages are Classes.

To create a class, use the keyword **class**:

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

Example 7.1.1. Simple Class Example

We will create a simple Class in Python.

```
class Car:
    model = "Volvo"
    color = "Blue"


car = Car()


print(car.model)
print(car.color)
```
Listing 7.1: Simple Python Class

The results will be in this case:

```
Volvo
Blue
```

This example don't illustrate the good things with classes so we will create some more examples.

[End of Example]

Example 7.1.2. Python Class

Lets create the following Python Code:

```
1  class Car:
2      model = ""
3      color = ""
4
5  car = Car()
6
7  car.model = "Volvo"
8  car.color = "Blue"
9
10 print(car.color + " " + car.model)
11
12 car.model = "Ford"
13 car.color = "Green"
14
15 print(car.color + " " + car.model)
```
Listing 7.2: Python Class example

You should try these examples.

[End of Example]

7.2 The __init__() Function

In Python all classes have a built-in function called __init__(), which is always executed when the class is being initiated.
In many other OOP languages we call this the Constructor.

Exercise 7.2.1. The __init__() Function

We will create a simple example where we use the __init__() function to illustrate the principle.

We change our previous Car example like this:

```
1  class Car:
2      def __init__(self, model, color):
3          self.model = model
4          self.color = color
5
6  car1 = Car("Ford", "Green")
7
8  print(car1.model)
9  print(car1.color)
10
11
```

71

```
12  car2 = Car("Volvo", "Blue")
13
14  print(car2.model)
15  print(car2.color)
```

Listing 7.3: Python Class Constructor Example

Lets extend the Class by defining a Function as well:

```
1  # Defining the Class Car
2  class Car:
3      def __init__(self, model, color):
4          self.model = model
5          self.color = color
6
7      def displayCar(self):
8          print(self.model)
9          print(self.color)
10
11
12  # Lets start using the Class
13
14  car1 = Car("Tesla", "Red")
15
16  car1.displayCar()
17
18
19  car2 = Car("Ford", "Green")
20
21  print(car2.model)
22  print(car2.color)
23
24
25  car3 = Car("Volvo", "Blue")
26
27  print(car3.model)
28  print(car3.color)
29
30  car3.color="Black"
31
32  car3.displayCar()
```

Listing 7.4: Python Class with Function

As you see from the code we have now defined a Class "Car" that has 2 Class variables called "model" and "color", and in addition we have defined a Function (or Method) called "displayCar()".

Its normal to use the term "Method" for Functions that are defined within a Class.

You declare class methods like normal functions with the exception that the first argument to each method is *self*.

To create instances of a class, you call the class using class name and pass in whatever arguments its __init__() method accepts.

For example:

72

```
1  car1 = Car("Tesla", "Red")
```

Listing 7.5: Import Class

[End of Example]

Exercise 7.2.2. Create the Class in a separate Python file

We start by creating the Class and then we save the code in "Car.py":

```
1  # Defining the Class Car
2  class Car:
3      def __init__(self, model, color):
4          self.model = model
5          self.color = color
6
7      def displayCar(self):
8          print(self.model)
9          print(self.color)
```

Listing 7.6: Define Python Class in separate File

Then we create a Python Script (testCar.py) where we are using the Class:

```
1  # Importing the Car Class
2  from Car import Car
3
4  # Lets start using the Class
5
6  car1 = Car("Tesla", "Red")
7
8  car1.displayCar()
9
10
11 car2 = Car("Ford", "Green")
12
13 print(car2.model)
14 print(car2.color)
15
16
17 car3 = Car("Volvo", "Blue")
18
19 print(car3.model)
20 print(car3.color)
21
22 car3.color="Black"
23
24 car3.displayCar()
```

Listing 7.7: Script that is using the Class

Notice the following line at the top:

```
1  from Car import Car
```

[language=Python]

[End of Example]

73

7.3 Exercises

Below you find different self-paced Exercises that you should go through and solve on your own. The only way to learn Python is to do lots of Exercises!

Exercise 7.3.1. Create Python Class

Create a Python Class where you calculate the degrees in Fahrenheit based on the temperature in Celsius and vice versa.

The formula for converting from Celsius to Fahrenheit is:

$$T_f = (T_c \times 9/5) + 32 \tag{7.1}$$

The formula for converting from Fahrenheit to Celsius is:

$$T_c = (T_f - 32) \times (5/9) \tag{7.2}$$

[End of Exercise]

Chapter 8

Creating Python Modules

As your program gets longer, you may want to split it into several files for easier maintenance. You may also want to use a handy function that you have written in several programs without copying its definition into each program.

To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter (the Python Console window).

8.1 Python Modules

A module is a file containing Python definitions and statements. The file name is the module name with the suffix .py appended.

Python allows you to split your program into modules that can be reused in other Python programs. It comes with a large collection of standard modules that you can use as the basis of your programs as we have seen examples of in previous chapters. Not it is time to make your own modules from scratch.

Consider a module to be the same as a code library. A file containing a set of functions you want to include in your application.

Previously you have been using different modules, libraries or packages created by the Python organization or by others. Here you will create your own modules from scratch.

Example 8.1.1. Create your first Python Module

We will create a Python module with 2 functions. The first function should convert from Celsius to Fahrenheit and the other function should convert from Fahrenheit to Celsius.

The formula for converting from Celsius to Fahrenheit is:

$$T_f = (T_c \times 9/5) + 32 \tag{8.1}$$

75

The formula for converting from Fahrenheit to Celsius is:

$$T_c = (T_f - 32) \times (5/9) \tag{8.2}$$

First, we create a Python module with the following functions (**fahrenheit.py**):

```
1  def c2f(Tc):
2
3      Tf = (Tc * 9/5) + 32
4      return Tf
5
6
7  def f2c(Tf):
8
9      Tc = (Tf - 32)*(5/9)
10     return Tc
```

<div align="center">Listing 8.1: Fahrenheit Functions</div>

Then, we create a Python script for testing the functions (**testfahrenheit.py**):

```
1  from fahrenheit import c2f, f2c
2
3  Tc = 0
4
5  Tf = c2f(Tc)
6
7  print("Fahrenheit: " + str(Tf))
8
9
10 Tf = 32
11
12 Tc = f2c(Tf)
13
14 print("Celsius: " + str(Tc))
```

<div align="center">Listing 8.2: Python Script testing the functions</div>

The results becomes:

```
1  Fahrenheit: 32.0
2  Celsius: 0.0
```

8.2 Exercises

Below you find different self-paced Exercises that you should go through and solve on your own. The only way to learn Python is to do lots of Exercises!

Exercise 8.2.1. Create Python Module for converting between radians and degrees

Since most of the trigonometric functions require that the angle is expressed in radians, we will create our own functions in order to convert between radians

76

and degrees.

It is quite easy to convert from radians to degrees or from degrees to radians. We have that:

$$2\pi[radians] = 360[degrees] \tag{8.3}$$

This gives:

$$d[degrees] = r[radians] \times (\frac{180}{\pi}) \tag{8.4}$$

and

$$r[radians] = d[degrees] \times (\frac{\pi}{180}) \tag{8.5}$$

Create two functions that convert from radians to degrees (r2d(x)) and from degrees to radians (d2r(x)) respectively.

These functions should be saved in one Python file .py.

Test the functions to make sure that they work as expected. You can choose to make a new .py file to test these functions or you can use the Console window.

[End of Exercise]

Chapter 9

File Handling in Python

9.1 Introduction

Python has several functions for creating, reading, updating, and deleting files. The key function for working with files in Python is the **open()** function.

The open() function takes two parameters; Filename, and Mode.

There are four different methods (modes) for opening a file:

- "x" - Create - Creates the specified file, returns an error if the file exists
- "w" - Write - Opens a file for writing, creates the file if it does not exist
- "r" - Read - Default value. Opens a file for reading, error if the file does not exist
- "a" - Append - Opens a file for appending, creates the file if it does not exist

In addition you can specify if the file should be handled as binary or text mode

- "t" - Text - Default value. Text mode
- "b" - Binary - Binary mode (e.g. images)

9.2 Write Data to a File

To create a **New** file in Python, use the open() method, with one of the following parameters:

- "x" - Create - Creates the specified file, returns an error if the file exists
- "w" - Write - Opens a file for writing, creates the file if it does not exist
- "a" - Append - Opens a file for appending, creates the file if it does not exist

To write to an **Existing** file, you must add a parameter to the open() function:

- ”w” - Write - Opens a file for writing, creates the file if it does not exist

- ”a” - Append - Opens a file for appending, creates the file if it does not exist

Example 9.2.1. Write Data to a File

```
1  f = open("myfile.txt", "x")
2
3  data = "Hello World"
4
5  f.write(data)
6
7  f.close()
```
Listing 9.1: Write Data to a File

[End of Example]

9.3 Read Data from a File

To read to an existing file, you must add the following parameter to the open() function:

- ”r” - Read - Default value. Opens a file for reading, error if the file does not exist

Example 9.3.1. Read Data from a File

```
1  f = open("myfile.txt", "r")
2
3  data = f.read()
4
5  print(data)
6
7  f.close()
```
Listing 9.2: Read Data from a File

[End of Example]

9.4 Logging Data to File

Typically you want to write multiple data to the, e.g., assume you read some temperature data at regular intervals and then you want to save the temperature values to a File.

Example 9.4.1. Logging Data to File

```
1  data = [1.6, 3.4, 5.5, 9.4]
2
3  f = open("myfile.txt", "x")
4
5  for value in data:
6      record = str(value)
7      f.write(record)
8      f.write("\n")
9
10 f.close()
```

Listing 9.3: Logging Data to File

[End of Example]

Example 9.4.2. Read Logged Data from File

```
1  f = open("myfile.txt", "r")
2
3  for record in f:
4      record = record.replace("\n", "")
5      print(record)
6
7  f.close()
```

Listing 9.4: Read Logged Data from File

[End of Example]

9.5 Web Resources

Below you find different useful resources for File Handling.

Python File Handling - w3school:
https://www.w3schools.com/python/python$_file_handling.asp$

Reading and Writing Files - python.org:
https://docs.python.org/3/tutorial/inputoutput.htmlreading-and-writing-files

9.6 Exercises

Below you find different self-paced Exercises that you should go through and solve on your own. The only way to learn Python is to do lots of Exercises!

Exercise 9.6.1. Data Logging

Assume you have the following data you want to log to a File as shown in Table 9.1.
Log these data to a File.

Create another Python Script that reads the same data.

80

[End of Exercise]

Exercise 9.6.2. Data Logging 2

Assume you read data from a Temperature sensor every 10 seconds for a period of let say 5 minutes.

Log the data to a File.

You can use the Random Generator in Python. An example of how to use the Random Generator is shown below:

```python
import random
for x in range(10):
    data = random.randint(1,31)
    print(data)
```

Listing 9.5: Read Data from a File

Make sure to log both the time and the temperature value

Create another Python Script that reads the same data.

You should also plot the data you read from the File.

[End of Exercise]

Table 9.1: Logged Data

Time	Value
1	22
2	25
3	28
...	...

Chapter 10

Error Handling in Python

10.1 Introduction to Error Handling

So far error messages haven't been discussed. You could say that we have 2 kinds of errors: syntax errors and exceptions.

10.1.1 Syntax Errors

Below we see an example of syntax errors:

```
>>> print(Hello World)
  File "<ipython-input-1-10cb182148e3>", line 1
    print(Hello World)
                    ^
SyntaxError: invalid syntax
```

In the example we have written print(Hello World) instead of print("Hello World") and then the Python Interpreter gives us an error message.

10.1.2 Exceptions

Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it. Errors detected during execution are called exceptions and are not unconditionally fatal: you will soon learn how to handle them in Python programs. Most exceptions are not handled by programs, however, and result in error messages as shown here:

```
>>> 10 * (1/0)
Traceback (most recent call last):

  File "<ipython-input-2-0b280f36835c>", line 1, in <module>
    10 * (1/0)

ZeroDivisionError: division by zero
```

or:

```
>>> '2' + 2
Traceback (most recent call last):

```

```
4    File "<ipython-input-3-d2b23a1db757>", line 1, in <module>
5      '2' + 2
6
7  TypeError: must be str, not int
```

10.2 Exceptions Handling

It is possible to write programs that handle selected exceptions.

In Python we can use the following built-in Exceptions Handling features:

- The **try** block lets you test a block of code for errors.

- The **except** block lets you handle the error.

- The **finally** block lets you execute code, regardless of the result of the try-
  and except blocks.

When an error occurs, or exception as we call it, Python will normally stop and
generate an error message.

These exceptions can be handled using the **try** - **except** statements.

Some basic example:

```
1  try:
2      10 * (1/0)
3  except:
4      print("The calculation failed")
```

or:

```
1  try:
2    print(x)
3  except:
4    print("x is not defined")
```

You can also use multiple exceptions:

```
1  try:
2    print(x)
3  except NameError:
4    print("x is not defined")
5  except:
6    print("Something is wrong")
```

The finally block, if specified, will be executed regardless if the try block raises
an error or not.

Example:

```
1  x=2
2
3  try :
4    print(x)
5  except NameError :
6    print("x is not defined")
7  except :
8    print("Something is wrong")
9  finally :
10     print("The Program is finished")
```

In general you should use try - except - finally when you try to open a File, read or write to Files, connect to a Database, etc.

Example:

```
1  try :
2    f = open("myfile.txt")
3    f.write("Lorum Ipsum")
4  except :
5    print("Something went wrong when writing to the file")
6  finally :
7    f.close()
```

Chapter 11

Debugging in Python

Debugging is the process of finding and resolving defects or problems within a computer program that prevent correct operation of computer software or a system [14].

Debuggers are software tools which enable the programmer to monitor the execution of a program, stop it, restart it, set breakpoints, and change values in memory. The term debugger can also refer to the person who is doing the debugging.

As a programmer, one of the first things that you need for serious program development is a debugger.

Python has a built-in debugger that can be used if you are coding Python with a basic text editor and running your Python programs from the command line.

A better option is to use the Debugging features integrated in your Python Editor. Debugging is typically integrated with the Python Editor you are using.

See the specific chapter for the different Python Editors.

Chapter 12

Installing and using Python Packages

A package contains all the files you need for a module. Modules are Python code libraries you can include in your project.

Since Python is open source you can find thousands of Python Packages that you can install and use in your Python programs.

You can use a Python Distribution like Anaconda Distribution (or similar Python Distributions) to download and install many common Python Packages as mentioned previously.

12.1 What is PIP?

PIP is a package manager for Python packages, or modules if you like. PIP is a tool for installing Python packages.

If you do not have PIP installed, you can download and install it from this page: https://pypi.org/project/pip/

PIP is typically used from the Command Prompt (Windows) or Terminal window (macOS).

Installing Python Packages:

```
1 pip install packagename
```

Uninstalling Python Packages:

```
1 pip uninstall packagename
```

Some Python Editors also have a graphical way of installing Python Packages, like, e.g., Visual Studio.

87

Part III

Python Environments and Distributions

Chapter 13

Introduction to Python Environments and Distributions

Python comes with many flavours and version.

Python is open source and everybody can bundle and distribute Python and different Python Packages.

A Python environment is a context in which you run Python code and includes Python Packages.

An environment consists of an interpreter, a library (typically the Python Standard Library), and a set of installed packages.

These components together determine which language constructs and syntax are valid, what operating-system functionality you can access, and which packages you can use.

You can have multiple Python Environments on your Computer.

Some of them are:

- CPython distribution available from python.org
- Anaconda
- Enthought Canopy
- WinPython
- etc.

It is easy to start using Python by installing one of these Python Distributions.

But you can also install the core Python from:
https://www.python.org

Then install the additional Python Packages you need by using PIP.
https://pypi.org/project/pip/

13.1 Package and Environment Managers

The two most popular tools for installing Python Packages and setting up
Python environments are:

- PIP - a Python Package Manager

- Conda - a Package and Environment Manager (for Python and other languages)

13.1.1 PIP

Web:
https://pypi.org

PIP is typically used from the Command Prompt (Windows) or Terminal window (macOS).

Installing Python Packages:

```
1 pip install packagename
```

Uninstalling Python Packages:

```
1 pip uninstall packagename
```

13.1.2 Conda

Conda is an open source package management system and environment management system that runs on Windows, macOS and Linux. Conda installs, runs
and updates packages and their dependencies.

The Conda package and environment manager is included in all versions of Anaconda.

Conda was created for Python programs, but it can package and distribute software for any language.

Conda allows you to to also create separate environments containing files, packages and their dependencies that will not interact with other environments.

Web:
https://conda.io/

Conda is part of or integrated with the Anaconda Python Distribution.

Web:
https://www.anaconda.com

13.2   Python Virtual Environments

Python "Virtual Environments" allow Python packages to be installed in an isolated location for a particular application, rather than being installed globally.

You can have multiple Python Environments on your computer.

Python Virtual Environments have their own installation directories and they don't share libraries with other virtual environments.

Python "Virtual Environments" is handy when you have different Python applications that needs different versions of Python or different version of the Python Packages you are using.

Chapter 14

Anaconda

Anaconda is not an Editor, but a Python Distribution package. Spyder is included in the Python Distribution package. You can also use Anaconda to install other Editors or Python packages.

It is available for Windows, macOS and Linux.

Web:
https://www.anaconda.com

Wikipedia:
https://en.wikipedia.org/wiki/Anaconda$(Python_distribution)$

14.1 Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage Python packages. The Anaconda Navigator can search for packages and install them on your computer, run the packages and update them.

Figure 14.1 shows the Anaconda Navigator.

14.2 Anaconda Prompt

You can use the Anaconda Prompt if you need to install extra Python packages, etc.

Let say you want to install the Python Control Systems Library package. Just enter the following in the Anaconda Prompt:

```
pip install control
```
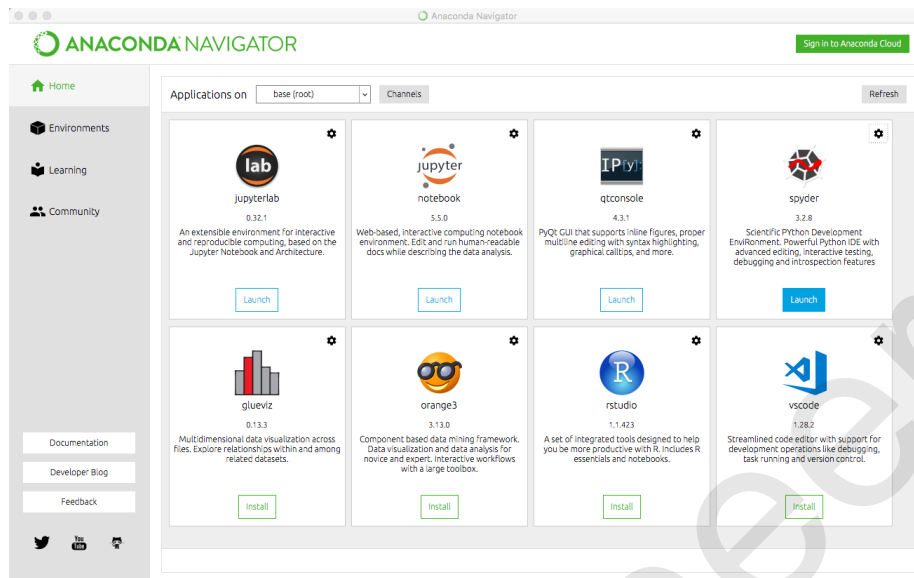
Figure 14.1: Anaconda Navigator

Python Package Index, or just pip, is a tool used to handle and install Python packages.

For for information about pip and different packages you can install, see the following:

https://pypi.org

Figure 14.2 shows where you can find the Anaconda Prompt. Windows: Search for Anaconda Prompt in the Search field in the start menu.
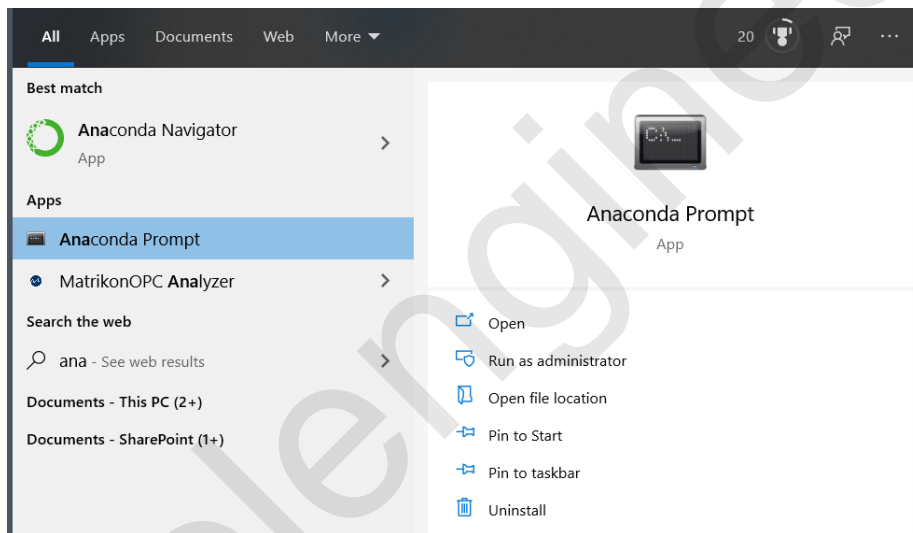
Figure 14.2: Anaconda Navigator

Part IV

Python Editors

Chapter 15

Python Editors

An Editor is a program where you create your code (and where you can run and test it). Most Editors have also features for Debugging and IntelliSense.

In theory, you can use Windows Notepad for creating Python programs, but in practice it is impossible to create programs without having an editor with Debugging, IntelliSense, color formatting, etc.

For simple Python programs you can use the IDLE Editor, but for more advanced programs a better editor is recommended.

Examples of Python Editors:

- Spyder

- Visual Studio Code

- Thonny

- Visual Studio

- PyCharm

- Wing

- JupyterNotebook

We will give an overview of these Code Editors in the next chapters.

I guess hundreds of different editors can be used for Python Programming, either out of the box or if you install an additional Extension that makes sure you can use Python in that editor.

If you already have a favorite Code Editor, it is a good change you can use that one for Python programming.

Which editor you should use depends on your background, what kind of code editors you have used previously, your programming skills, what your are going

96

to develop in Python, etc.

If you are familiar with MATLAB, Spyder is recommended. Also, if you want to use Python for numerical calculations and computations, Spyder is a good choice.

If you want to create Web Applications or other kinds of Applications, other Editors are probably better to use.

For Internet of Things Applications, simple editors like Visual Studio Code or Thonny may be good choices. Thonny is also the default editor on the Raspberry Pi OS, which is popular Internet of Things platform.

For a list of "Best Python Editors", see [15].

Chapter 16

Spyder

Spyder - short for "Scientific PYthon Development EnviRonment".

Spyder is an open source cross-platform integrated development environment(IDE) for scientific programming in the Python language.
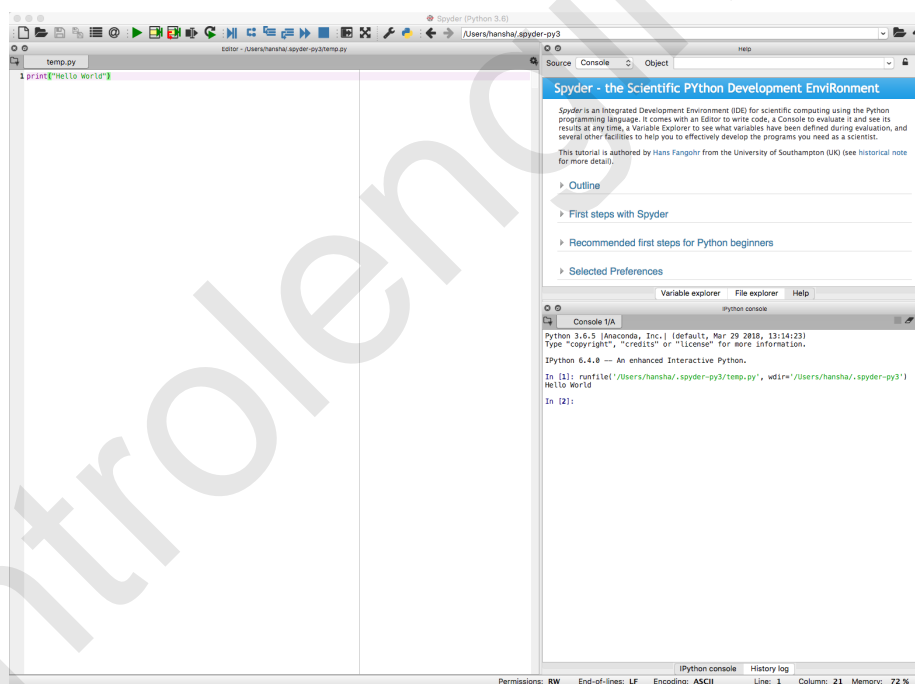
Figure 16.1: Spyder Editor

The Spyder editor consists of the following parts or windows:

- Code Editor window
- iPython Console window

- Variable Explorer

- etc.

Web:
https://www.spyder-ide.org

If you have used MATLAB previously or want to use Python for scientific use, Spyder is a good choice. it is easy to install using the Anaconda Distribution.

Web:
https://www.anaconda.com

16.1   Configuration

Typically you want to show figures and plots in separate windows.

Select Tools-Preferences as shown in Figure 16.2.
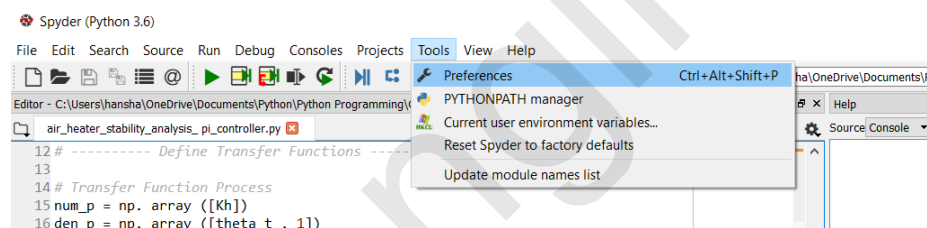
Figure 16.2: Python Tools-Preferences

Then select "Automatic" as shown in Figure 16.3.
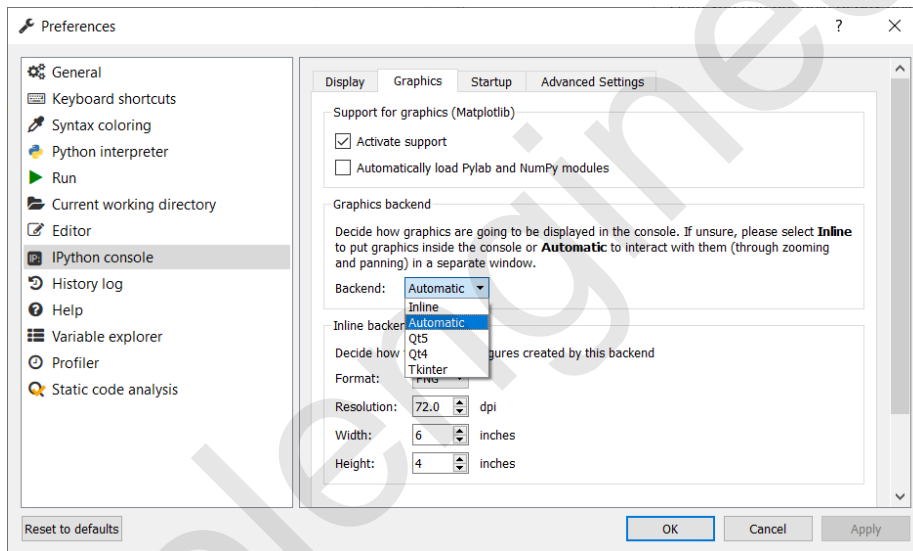
Figure 16.3: Python Preferences window

Chapter 17

Visual Studio Code

17.1 Introduction to Visual Studio Code

Visual Studio Code is a simple and easy to use editor that can be used for many different programming languages.
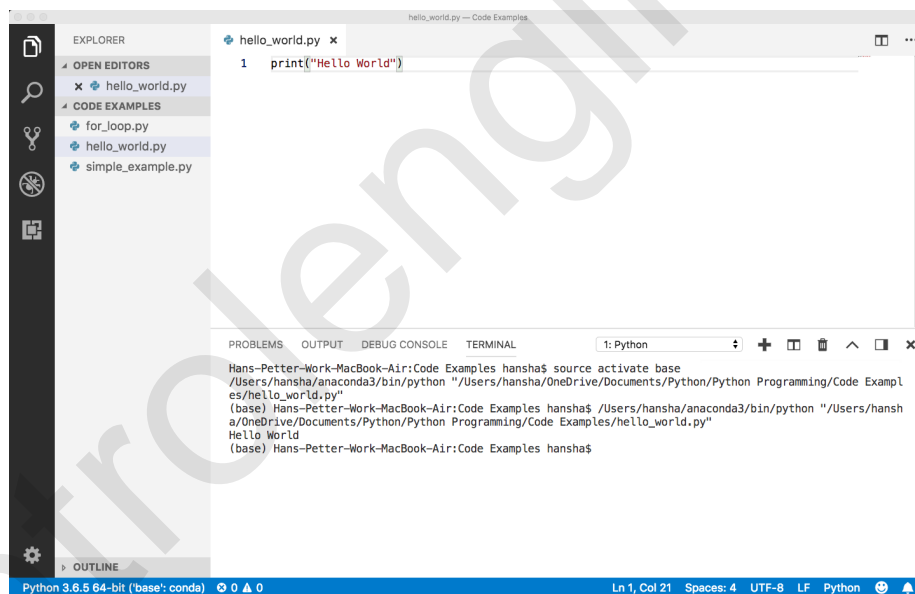
Figure 17.1: Using Visual Studio Code as Python Editor

Right-Click and select "Run Python File in Terminal"

Web:
https://code.visualstudio.com

Wikipedia:
https://en.wikipedia.org/wiki/Visual$_S$tudio$_C$ode

17.2   Python in Visual Studio Code

In addition to Visual Studio Code you need to install the Python extension for Visual Studio Code.

You must install a Python interpreter yourself separately from the extension. For a quick install, use Python from python.org.

https://www.python.org

Python is an interpreted language, and in order to run Python code and get Python IntelliSense within Visual Studio Code, you must tell Visual Studio Code which interpreter to use.

Web:
https://code.visualstudio.com/docs/languages/python

Part V

Python for Mathematics Applications

Chapter 18

Mathematics in Python

Python is a powerful tool for mathematical calculations.

If you are looking for similar using MATLAB, please take a look at these resources:
https://www.halvorsen.blog/documents/programming/matlab/

18.1 Basic Math Functions

The **Python Standard Library** consists of different modules for handling file I/O, basic mathematics, etc. You don't need to install these separately, but you need to important them when you want to use some of these modules or some of the functions within these modules.

In this chapter we will focus on the math module that is part of the Python Standard Library.

The math module has all the basic math functions you need, such as: Trigonometric functions: *sin(x)*, *cos(x)*, etc. Logarithmic functions: *log()*, *log10()*, etc. Constants like *pi*, *e*, *inf*, *nan*, etc. etc.

Example 18.1.1. Using the math module

We create some basic examples how to use a Library, a Package or a Module:

If we need only the sin() function we can do like this:

```
1  from math import sin
2
3  x = 3.14
4  y = sin(x)
5
6  print(y)
```

If we need a few functions we can do like this

```
1  from math import sin, cos
2
3  x = 3.14
4  y = sin(x)
5  print(y)
6
7  y = cos(x)
8  print(y)
```

If we need many functions we can do like this:

```
1  from math import *
2
3  x = 3.14
4  y = sin(x)
5  print(y)
6
7  y = cos(x)
8  print(y)
```

We can also use this alternative:

```
1  import math
2
3  x = 3.14
4  y = math.sin(x)
5
6  print(y)
```

We can also write it like this:

```
1  import math as mt
2
3  x = 3.14
4  y = mt.sin(x)
5
6  print(y)
```

[End of Example]

There are advantages and disadvantages with the different approaches. In your program you may need to use functions from many different modules or packages. If you import the whole module instead of just the function(s) you need you use more of the computer memory.

Very often we also need to import and use multiple libraries where the different libraries have some functions with the same name but different use.

Other useful modules in the **Python Standard Library** are **statistics** (where you have functions like *mean()*, *stdev()*, etc.)

For more information about the functions in the **Python Standard Library**, see:
https://docs.python.org/3/library/

18.1.1 Exercises

Below you find different self-paced Exercises that you should go through and solve on your own. The only way to learn Python is to do lots of Exercises!

Exercise 18.1.1. Create Mathematical Expressions in Python

Create a function that calculates the following mathematical expression:

$$z = 3x^2 + \sqrt{x^2 + y^2} + e^{\ln(x)} \qquad (18.1)$$

Test with different values for x and y.

[End of Exercise]

Exercise 18.1.2. Create advanced Mathematical Expressions in Python

Create the following expression in Python:

$$f(x) = \frac{\ln(ax^2 + bx + c) - sin(ax^2 + bx + c)}{4\pi x^2 + cos(x-2)(ax^2 + bx + c)} \qquad (18.2)$$

Given $a = 1, b = 3, c = 5$ Find $f(9)$
(The answer should be $f(9) = 0.0044$)

Tip! You should split the expressions into different parts, such as:

$$poly = ax^2 + bx + c$$

$num = \dots$
$den = \dots$
$f = \dots$

This makes the expression simpler to read and understand, and you minimize the risk of making an error while typing the expression in Python.

When you got the correct answer try to change to, e.g., $a = 2, b = 8, c = 6$

Find $f(9)$

[End of Exercise]

Exercise 18.1.3. Pythagoras
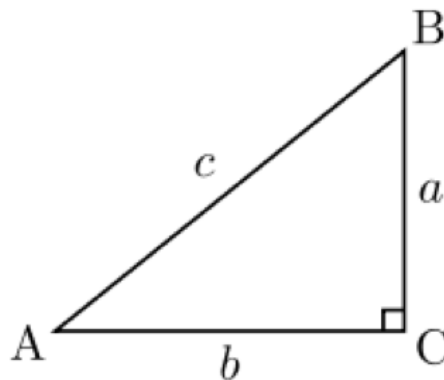
Figure 18.1: Right-angled triangle

Pythagoras theorem is as follows:

$$c^2 = a^2 + b^2 \tag{18.3}$$

Create a function that uses Pythagoras to calculate the hypotenuse of a right-angled triangle (Figure 18.1), e.g.:

```
1  def pythagoras(a,b)
2      ...
3      ...
4      return c
```

[End of Exercise]

Exercise 18.1.4. Albert Einstein

Given the famous equation from Albert Einstein:

$$E = mc^2 \tag{18.4}$$

The sun radiates $385x10^{24} J/s$ of energy.

Calculate how much of the mass on the sun is used to create this energy per day.

How many years will it take to convert all the mass of the sun completely? Do we need to worry if the sun will be used up in our generation or the next? justify the answer.

The mass of the sun is $2x10^{30} kg$.

Exercise 18.1.5. Cylinder Surface Area

Create a function that finds the surface area of a cylinder based on the height (h) and the radius (r) of the cylinder. See Figure **??**.
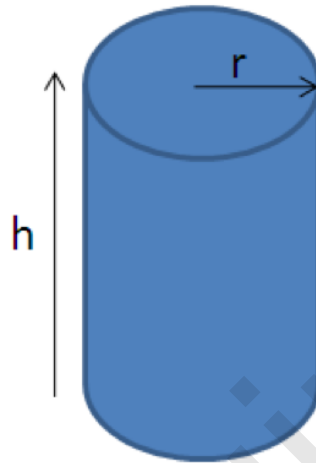
Figure 18.2: cylinder

18.2 Statistics

18.2.1 Introduction to Statistics

Mean or average:
The mean is the sum of the data divided by the number of data points. It is commonly called "the average".

Formula for mean:

$$\bar{x} = \frac{x_1 + x_2 + x_3 + ... + x_N}{N} = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{18.5}$$

Example 18.2.1. Mean

Given the following dataset: 2.2, 4.5, 6.2, 3.6, 2.6

Mean:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i = \frac{2.2 + 4.5 + 6.2 + 3.6 + 2.6}{5} = \frac{19.1}{5} = 3.82 \tag{18.6}$$

[End of Example]

Variance:

Variance is a measure of the variation in a data set.

$$var(x) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2 \tag{18.7}$$

Standard deviation:

The standard deviation is a measure of the spread of the values in a dataset or the value of a random variable. It is defined as the square root of the variance.

$$std(x) = \sigma = \sqrt{var} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2} \tag{18.8}$$

We typically use the symbol σ for standard deviation.

We have that $\sigma^2 = var(x)$

18.2.2   Statistics functions in Python

Mathematical statistics functions in Python:
https://docs.python.org/3/library/statistics.html

statistics is part of the The Python Standard Library.

For more information about the functions in the **Python Standard Library**, see:
https://docs.python.org/3/library/

Example 18.2.2. Statistics using the statistics module in Python Standard Library

Below you find some examples how to use some of the statistics functions in the statistics module in Python Standard Library:

```
import statistics as st

data = [-1.0, 2.5, 3.25, 5.75]

#Mean or Average
m = st.mean(data)
print(m)

# Standard Deviation
st_dev = st.stdev(data)
```

109

```
11  print(st_dev)
12
13  # Median
14  med = st.median(data)
15  print(med)
16
17  # Variance
18  var = st.variance(data)
19  print(var)
```

Listing 18.1: Statistics functions in Python

[End of Example]

IMPORTANT: Do not name your file "statistics.py" since the import will be confused and throw the errors of the library not existing and the mean function not existing.

You can also use the **NumPy** Library. NumPy is the fundamental package for scientific computing with Python.

Here you find an overview of the **NumPy** library:
http://www.numpy.org

Example 18.2.3. Statistics using the NumPy Library

Below you find some examples how to use some of the statistics functions in NumPy:

```
1  import numpy as np
2
3  data = [-1.0, 2.5, 3.25, 5.75]
4
5  #Mean or Average
6  m = np.mean(data)
7  print(m)
8
9  # Standard Deviation
10  st_dev = np.std(data)
11  print(st_dev)
12
13  # Median
14  med = np.median(data)
15  print(med)
16
17  # Minimum Value
18  minv = np.min(data)
19  print(minv)
20
21  # Maxumum Value
22  maxv = np.max(data)
23  print(maxv)
```

Listing 18.2: Statistics using the NumPy Library

110

Exercise 18.2.1. Create your own Statistics Module in Python

Using the built-in functions in the Python Standard Library or the NumPy library is straightforward.

In order to get a deeper understanding of the mathematics behind these functions and to learn more Python programming, you should create your own Statistics Module in Python.

Create your own Statistics Module in Python (e.g., "mystatistics.py) and then create a Python Script (e.g., "testmystatistics.py) where you test these functions.

You should at least implement functions for mean, variance, standard deviation, minimum and maximum.

[End of Exercise]

18.3 Trigonometric Functions

Python offers lots of Trigonometric functions, e.g., sin, cos, tan, etc.

Note! Most of the trigonometric functions require that the angle is expressed in radians.

Example 18.3.1. Trigonometric Functions in Math module

```python
import math as mt

x = 2*mt.pi

y = mt.sin(x)
print(y)

y = mt.cos(x)
print(y)

y = mt.tan(x)
print(y)
```

Listing 18.3: Trigonometric Functions in Math module

Here we have used the Math module in the Python Standard Library.

For more information about the functions in the **Python Standard Library**, see:
https://docs.python.org/3/library/index.html

[End of Example]

Example 18.3.2. Plotting Trigonometric Functions

In the example above we used some of the trigonometric functions in basic calculations.

Lets see if we are able to plot these functions.

```python
import math as mt
import matplotlib.pyplot as plt

xdata = []
ydata = []

for x in range(0, 10):
    xdata.append(x)
    y = mt.sin(x)
    ydata.append(y)

plt.plot(xdata, ydata)
plt.show()
```

Listing 18.4: Plotting Trigonometric Functions

In the example we have plotted $\sin(x)$, we can easily extend the program to plot $\cos(x)$, etc.

For more information about the functions in the **Python Standard Library**, see:
https://docs.python.org/3/library/index.html

[End of Example]

Example 18.3.3. Trigonometric Functions using NumPy

The problem with using the Trigonometric functions in the the Math module from the Python Standard Library is that they don't handle an array as input.

We will use the NumPy library instead because they handle arrays, in addition to all the handy functionality in the NumPy library.

```python
import numpy as np
import matplotlib.pyplot as plt

xstart = 0
xstop = 2*np.pi
increment = 0.1

x = np.arange(xstart, xstop, increment)

y = np.sin(x)
```

112

```
11  plt.plot(x, y)
12  plt.title('y=sin(x)')
13  plt.xlabel('x')
14  plt.ylabel('y')
15  plt.grid()
16  plt.axis([0, 2*np.pi, -1, 1])
17  plt.show()
18
19  y = np.cos(x)
20  plt.plot(x, y)
21  plt.title('y=cos(x)')
22  plt.xlabel('x')
23  plt.ylabel('y')
24  plt.grid()
25  plt.axis([0, 2*np.pi, -1, 1])
26  plt.show()
27
28  y = np.tan(x)
29  plt.plot(x, y)
30  plt.title('y=tan(x)')
31  plt.xlabel('x')
32  plt.ylabel('y')
33  plt.grid()
34  plt.axis([0, 2*np.pi, -1, 1])
35  plt.show()
```

Listing 18.5: Trigonometric Functions using NumPy

This Python script gives the plots as shown in Figure 18.3.

[End of Example]

Exercise 18.3.1. Create Python functions for converting between radians an degrees

Since most of the trigonometric functions require that the angle is expressed in radians, we will create our own functions in order to convert between radians and degrees.

It is quite easy to convert from radians to degrees or from degrees to radians.

We have that:

$$2\pi[radians] = 360[degrees] \tag{18.9}$$

This gives:

$$d[degrees] = r[radians] \times (\frac{180}{\pi}) \tag{18.10}$$

and

$$r[radians] = d[degrees] \times (\frac{\pi}{180}) \tag{18.11}$$

Create two functions that convert from radians to degrees (r2d(x)) and from degrees to radians (d2r(x)) respectively.

113

These functions should be saved in one Python file .py.

Test the functions to make sure that they work as expected.

<div align="right">[End of Exercise]</div>

Exercise 18.3.2. Trigonometric functions on right triangle

Given right triangle as shown in Figure 18.4.

Create a function that finds the angle A (in degrees) based on input arguments (a,c), (b,c) and (a,b) respectively.

Use, e.g., a third input "type" to define the different types above.

Use you previous function r2d() to make sure the output of your function is in degrees and not in radians.

Test the function to make sure it works properly.

Tip! We have that:

$$\sin(A) = \frac{a}{c} \rightarrow A = \arcsin(\frac{a}{c}) \tag{18.12}$$

$$\cos(A) = \frac{b}{c} \rightarrow A = \arccos(\frac{b}{c}) \tag{18.13}$$

$$\tan(A) = \frac{a}{b} \rightarrow A = \arctan(\frac{a}{b}) \tag{18.14}$$

We may also need to use the Pythagoras' theorem:

$$c^2 = a^2 + b^2 \tag{18.15}$$

```
1 >>> a=5
2 >>> b=8
3 >>> c = sqrt(a**2 + b**2)
4
5 >>> A = right_triangle(a,c,'sin')
6 A =
7     32.0054
8
9 >>> A = right_triangle(b,c,'cos')
10 A =
11     32.0054
12 >>> A = right_triangle(a,b,'tan')
13 A =
14     32.0054
```

We also see that the answer in this case is the same, which is expected.

114

[End of Exercise]

Exercise 18.3.3. Law of Cosines

Given the triangle as shown in Figure 18.5.

Create a function where you find c using the **law of cosines**.

$$c^2 = a^2 + b^2 - 2ab\cos(C) \tag{18.16}$$

Test the functions to make sure it works properly.

[End of Exercise]

Exercise 18.3.4. Plotting Trigonometric Functions

Plot $sin(\theta)$ and $cos(\theta)$ for $0 \le \theta \le 2\pi$ in the same plot (both in the same plot and in 2 different subplots).

Make sure to add labels and a legend and use different line styles and colors for the plots.

[End of Exercise]

18.4 Polynomials

A polynomial is expressed as:

$$p(x) = p_1 x^n + p_2 x^{n-1} + ... + p_n x + p_{n+1} \tag{18.17}$$

where $p_1, p_2, p_3, ...$ are the coefficients of the polynomial.

We will use the Polynomial Module in the NumPy Package.

Web:
https://numpy.org/doc/stable/reference/routines.polynomials.polynomial.html

Other Resources:

Python Advanced Course Topics - Polynomials:
https://www.python-course.eu/polynomial$_c$lass$_i$n$_p$ython.php
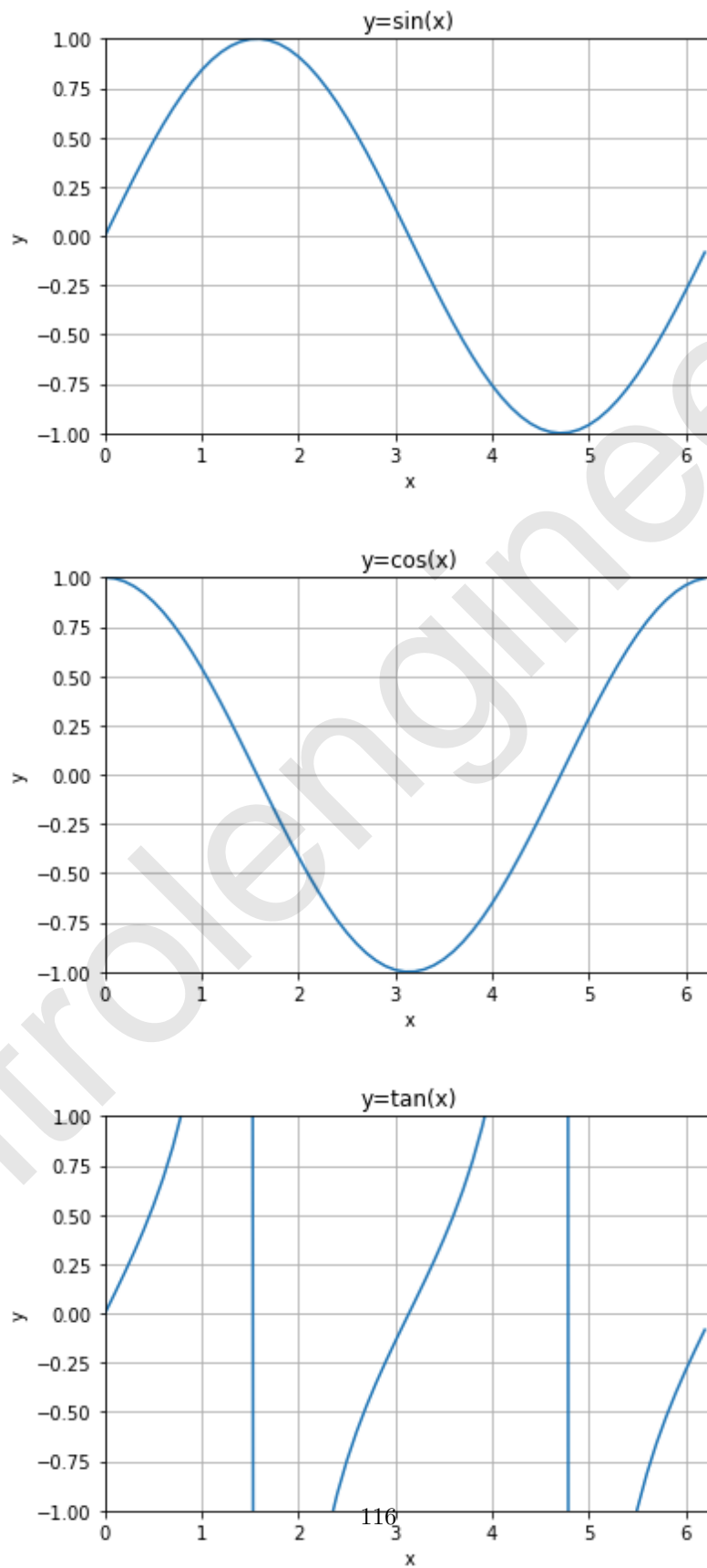
Figure 18.3: Trigonometric Functions
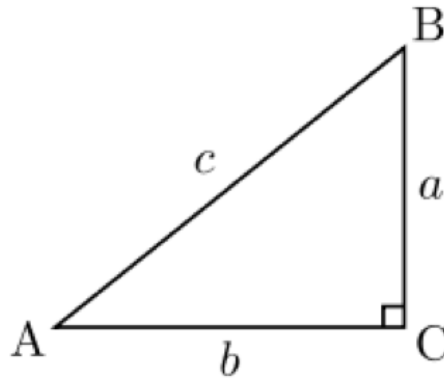
Figure 18.4: Right Triangle
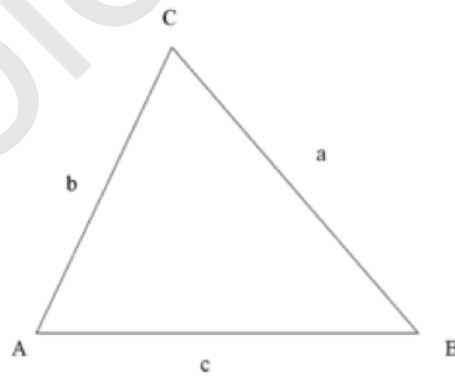
Figure 18.5: Law of Cosines

Chapter 19

Linear Algebra in Python

19.1 Introduction to Linear Algebra

A matrix is a two-dimensional data structure where numbers are arranged into rows and columns.

Matrix is a special case of two dimensional array where each data element is of strictly same size.

Matrices are very important data structures for many mathematical and scientific calculations.

A general matrix is defined as:

$$A = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1m} \\ a_{21} & a_{22} & .. & a_{2m} \\ ... & ... & ... & ... \\ a_{n1} & a_{n2} & ... & a_{nm} \end{bmatrix} \in R^{nxm} \tag{19.1}$$

Where n is number of rows and m is number of columns.

Example of a 3 x 3 matrix:

$$A = \begin{bmatrix} 1 & 5 & 3 \\ 4 & 6 & 6 \\ 3 & 8 & 9 \end{bmatrix} \tag{19.2}$$

Example of a 3 x 4 matrix:

$$A = \begin{bmatrix} 1 & 5 & 3 & 4 \\ 4 & 5 & 7 & 8 \\ 7 & 8 & 9 & 3 \end{bmatrix} \tag{19.3}$$

Example of a 4 x 2 matrix:

$$A = \begin{bmatrix} 1 & 5 \\ 4 & 5 \\ 3 & 2 \\ 7 & 8 \end{bmatrix} \tag{19.4}$$

Python doesn't have a built-in type for matrices. However, we can create list of a list as a matrix.

Example 19.1.1. Matrix definition with Standard Python

Here is an example how we can implement a vector and a matrix in standard Python:

```python
a = [1, 3, 7, 2]

print("a =", a)


A = [[1, 3, 7, 2],
     [5, 8, -9, 0],
     [6, -7, 11, 12]]

print("A =", A)
```
Listing 19.1: Python Arrays

This gives the following output:

```
a = [1, 3, 7, 2]
A = [[1, 3, 7, 2], [5, 8, -9, 0], [6, -7, 11, 12]]
```

So we can define vectors and matrices with standard Python, but standard Python has no support for manipulation and calculation of them.

But fortunately we can use the NumPy package for creating matrices and for matrix manipulation.

[End of Example]

19.2 Linear Algebra with Python

We will use the NumPy package for matrix manipulation.

NumPy is the fundamental package for scientific computing with Python.

Here you find an overview of the **NumPy** library:
http://www.numpy.org

Example 19.2.1. Matrix Manipulation using the NumPy Library

119

Below you see how we can use NumPy for creating vectors and matrices and manipulate them using NumPy:

```
1  import numpy as np
2
3  a = np.array([1, 3, 7, 2])
4
5  print("a =", a)
6
7
8
9  A = np.array([[1, 3, 7, 2],
10               [5, 8, -9, 0],
11               [6, -7, 11, 12]])
12
13
14  print("A =", A)
15
16
17
18
19  A = np.array([[0, 1],
20               [-2, -3]])
21
22  B = np.array([[1, 0],
23               [3, -2]])
24
25  C = A + B
26  print(C)
27
28
29  C = A.dot(B)
30  print(C)
31
32  C = A.transpose()
33  print(C)
```

Listing 19.2: Matrix manipulation using NumPy

[End of Example]

19.2.1 Vectors

Use np.array() when you define vectors:

```
1  import numpy as np
2
3  a = np.array([1, 3, 7, 2])
4
5  print("a =", a)
```

Listing 19.3: Matrix manipulation using NumPy

19.2.2 Matrices

You can use **np.array()** when defining matrices also, but you may also use **np.matrix()**.

The numpy matrix object is a subclass of the numpy array object and it is tailor-made for matrices. The numpy matrices are strictly 2-dimensional, while numpy arrays can be of any dimension.

Example:

```
1  import numpy as np
2
3  A = np.matrix([[0, 1],
4                 [-2, -3]])
5
6  print("A =", A)
```

It is no longer recommended to use this class, even for linear algebra. Instead use regular arrays. The class may be removed in the future.

For more information, please see the following:

https://numpy.org/doc/stable/reference/generated/numpy.matrix.html

19.2.3 Linear Algebra (numpy.linalg)

For more Linear Algebra functionality in the NumPy library you need to use the numpy.linalg module.

Here you find an overview of the numpy.linalg module:
https://docs.scipy.org/doc/numpy/reference/routines.linalg.html

We will use the numpy.linalg in different examples in this chapter.

19.2.4 Matrix Addition

Given the matrices:

$$A \in R^{nxm}$$

and

$$B \in R^{nxm}$$

Then

$$C = A + B \in R^{nxm}$$

Example:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \tag{19.5}$$

$$B = \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix} \tag{19.6}$$

Then we get:

$$A + B = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix} = \begin{bmatrix} 0+1 & 1+0 \\ -2+3 & -3-2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -5 \end{bmatrix} \tag{19.7}$$

Example 19.2.2. Matrix Addition in Python

```python
import numpy as np

A = np.matrix([[0, 1],
               [-2, -3]])

B = np.matrix([[1, 0],
               [3, -2]])

C = A + B
print(C)
```

Listing 19.4: Matrix Addition in Python

We get:

```
[[ 1   1]
 [ 1  -5]]
```

[End of Example]

19.2.5 Matrix Subtraction

Given the matrices:

$$A \in R^{nxm}$$

and

$$B \in R^{nxm}$$

Then

122

$$C = A - B \in R^{nxm}$$

Example:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \tag{19.8}$$

$$B = \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix} \tag{19.9}$$

Then we get:

$$A - B = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix} = \begin{bmatrix} 0-1 & 1-0 \\ -2-3 & -3-(-2) \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ -5 & -1 \end{bmatrix} \tag{19.10}$$

Example 19.2.3. Matrix Subration in Python

```python
import numpy as np

A = np.matrix([[0, 1],
               [-2, -3]])

B = np.matrix([[1, 0],
               [3, -2]])

C = A - B
print(C)
```

Listing 19.5: Matrix Subraction in Python

We get:

```
[[-1  1]
 [-5 -1]]
```

[End of Example]

19.2.6 Matrix Multiplication

Given the matrices:

$$A \in R^{nxm}$$

and

$$B \in R^{mxp}$$

Then

$$C = AB \in R^{nxp}$$

Where

$$c_{jk} = \sum_{l=1}^{n} a_{jl} b_{lk}$$

Example:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \qquad (19.11)$$

$$B = \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix} \qquad (19.12)$$

Then we get:

$$AB = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \cdot 1 + 1 \cdot 3 & 0 \cdot 0 + 1 \cdot (-2) \\ -2 \cdot 1 - 3 \cdot 3 & -2 \cdot 0 - 3 \cdot (-2) \end{bmatrix} = \begin{bmatrix} 3 & -2 \\ -11 & 6 \end{bmatrix} (19.13)$$

We do the same in Python:

Example 19.2.4. Matrix Multiplication in Python

```python
import numpy as np

A = np.matrix([[0, 1],
               [-2, -3]])

B = np.matrix([[1, 0],
               [3, -2]])

C = A * B
print(C)
```

Listing 19.6: Matrix Multiplication in Python

This gives:

```
1  [[  3   -2]
2   [-11    6]]
```

Below you see different alternative solutions that can be used:

```
1  import numpy as np
2
3  A = np.array([[0,  1],
4                [-2, -3]])
5
6  B = np.array([[1,  0],
7                [3, -2]])
8
9  #Alternative 1
10 C = A.dot(B)
11 print(C)
12
13 #Alternative 2
14 C = np.dot(A,B)
15 print(C)
16
17 #Alternative 3
18 C =  np.mat(A) * np.mat(B)
19 print(C)
```

Listing 19.7: Matrix Multiplication in Python - Alternative Solutions

As shown in the example you can use different syntax. The 3 alternatives in the example give the same result. Try it.

[End of Example]

In matrix multiplication the matrices don't need to be quadratic, but the inner dimensions need to be the same. The size of the resulting matrix will be the outer dimensions. See Figure 19.1.

$$n \left[\overset{m}{A} \right] m \left[\overset{p}{B} \right] = n \left[\overset{p}{C} \right]$$

Figure 19.1: Matrix Multiplication

We have also the following matrix rules:

$$AB \neq BA \tag{19.14}$$

$$A(BC) = (AB)C \tag{19.15}$$

$$(A + B)C = AC + BC \tag{19.16}$$

$$C(A + B) = CA + CB \tag{19.17}$$

Exercise 19.2.1. Matrix Rules

Create a Python Script where you verify the rules above is correct.

[End of Exercise]

19.2.7 Transpose of a Matrix

A general matrix is defined as:

$$A = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1m} \\ a_{21} & a_{22} & .. & a_{2m} \\ ... & ... & ... & ... \\ a_{n1} & a_{n2} & ... & a_{nm} \end{bmatrix} \in R^{nxm} \tag{19.18}$$

Where n is number of rows and m is number of columns.

The transpose of matrix A is then:

$$A^T = \begin{bmatrix} a_{11} & a_{21} & ... & a_{n1} \\ a_{12} & a_{22} & .. & a_{n2} \\ ... & ... & ... & ... \\ a_{1m} & a_{2m} & ... & a_{nm} \end{bmatrix} \in R^{mxn} \tag{19.19}$$

The transform of a matrix is formed by turning all the rows of a given matrix into columns and vice-versa.

Example 19.2.5. Transpose of a Matrix in Python

```python
import numpy as np

A = np.matrix([[0, 1],
               [-2, -3]])

At = np.transpose(A)
print(At)


B = np.matrix([[1, 0, 4],
               [3, -2, 8]])

Bt = np.transpose(B)
print(Bt)
```

126

```
15
16
17
18  C = np.matrix([[1,   4,],
19                 [2,  -3,],
20                 [-6, -2]])
21
22  Ct = np.transpose(C)
23  print(Ct)
```

Listing 19.8: Transpose of a Matrix

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

[End of Example]

19.2.8   Determinant

Given a matrix A the Determinant is given by:

$$det(A) = |A|$$

For a 2x2 matrix A:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \tag{19.20}$$

We have:

$$det(A) = |A| = a_{11}a_{22} - a_{21}a_{12} \tag{19.21}$$

Example:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \tag{19.22}$$

We get:

$$det(A) = |A| = 1 \cdot 4 - 3 \cdot 2 = 4 - 6 = -2 \tag{19.23}$$

Example 19.2.6. Determinant

Python Example:

```python
import numpy as np
import numpy.linalg as la

A = np.matrix([[1, 2],
               [3, 4]])

Adet = la.det(A)

print(Adet)
```

Listing 19.9: Determinant

This gives:

```
-2.0000000000000004
```

Listing 19.10: Determinant

[End of Example]

19.2.9 Inverse Matrix

The inverse of a quadratic matrix $A \in R^{nxn}$ is defined by:

$$A^{-1}$$

For a square matrix A, the inverse is written A^{-1}. When A is multiplied by A^{-1} the result is the identity matrix I. Non-square matrices do not have inverses.

We have that:

$$AA^{-1} = A^{-1}A = I \tag{19.24}$$

For a $2x2$ matrix we have:

$$A = \left[\begin{array}{cc} a_{11} & a_{12} \\ a_{21} & a_{22} \end{array} \right] \tag{19.25}$$

The inverse of A becomes:

$$A^{-1} = \frac{1}{det(A)} \left[\begin{array}{cc} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{array} \right] \tag{19.26}$$

Example 19.2.7. Inverse Matrix

128

Python Example:

```python
import numpy as np
import numpy.linalg as la

A = np.matrix([[1, 2],
               [3, 4]])

Ainv = la.inv(A)

print(Ainv)
```

Listing 19.11: Inverse Matrix

We get the following results:

```
[[-2.    1. ]
 [ 1.5  -0.5]]
```

[End of Example]

19.3 Solving Linear Equations

Example 19.3.1. Solving Linear Equations

Given the equations:

$$x_1 + 2x_2 = 5 \tag{19.27}$$

$$3x_1 + 4x_2 = 6 \tag{19.28}$$

We want to set the equations on the following form:

$$Ax = b \tag{19.29}$$

We need to find A and b and define them in Python.

Then we can solve the equations, i.e., find x_1 and x_2 using Python.

It can be solved like this:

$$x = A^{-1}b \tag{19.30}$$

We get:

129

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad (19.31)$$

$$b = \begin{bmatrix} 5 \\ 6 \end{bmatrix} \qquad (19.32)$$

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad (19.33)$$

Python Script:

```python
import numpy as np
import numpy.linalg as la

A = np.array([[1, 2],
              [3, 4]])

b = np.array([[5],
              [6]])

Ainv = la.inv(A)

x = Ainv.dot(b)

print(x)
```

The results becomes:

```
[[-4. ]
 [ 4.5]]
```

You can also use the following:

```python
x = np.linalg.solve(A, b)
```

Python Script:

```python
import numpy as np

A = np.array([[1, 2],
              [3, 4]])

b = np.array([[5],
              [6]])

x = np.linalg.solve(A, b)
print(x)
```

Note! The A matrix must be square and of full-rank, i.e. the inverse matrix needs to exists.

130

[End of Example]

In many cases we cannot find the inverse matrix, e.g., when the matrix is not quadratic. Finding the inverse matrix for large matrices is also time-consuming.

The numpy.linalg module can be used.

Here you find an overview of the numpy.linalg module:
https://docs.scipy.org/doc/numpy/reference/routines.linalg.html

Example 19.3.2. Solving Linear Equations when A is not quadratic

We use lstsq for the least-squares best "solution" of the system/equation.

Python Script:

```python
import numpy as np

A = np.array([[1, 2],
              [3, 4],
              [7, 8]])

b = np.array([[5],
              [6],
              [9]])

#x = np.linalg.solve(A, b) #Not working because inverse(A) does not
    exists


x = np.linalg.lstsq(A, b, rcond=None)[0]

print(x)
```

The results becomes:

```
[[-3.5        ]
 [ 4.17857143]]
```

[End of Example]

19.4 Exercises

Below you find different self-paced Exercises that you should go through and solve on your own. The only way to learn Python is to do lots of Exercises!

Exercise 19.4.1. Exercise Solving Linear Equations

131

Given the following equations:

$$x_1 + 2x_2 = 5 \tag{19.34}$$

$$3x_1 + 4x_2 = 6 \tag{19.35}$$

$$7x_1 + 8x_2 = 9 \tag{19.36}$$

Find the solutions for the given equations using Python.

[End of Exercise]

Exercise 19.4.2. Matrix Addition, Subtraction and Multiplication using nested For Loops

Assume that that you cannot do matrix addition, subtraction and multiplication as shown in the examples above.

Create a Python Module with 3 functions (e.g., matrixaddition(), matrixsubtraction(), matrixmultiplication()) where you implement your own version of matrix addition, subtraction and multiplication using nested For Loops.
Make sure to test the functions that they work as expected, e.g.:

```
1  import mymatrixmodule as matrix
2
3  A = [[1, 3, 7],
4       [5, 8, -9],
5       [6, -7, 11]]
6
7  B = [[2, 3, 5],
8       [5, -9, -9],
9       [6, 8, 1]]
10
11  c = matrix.matrixaddition(A, B)
12  print(C)
13
14  c = matrix.matrixsubtraction(A, B)
15  print(C)
16
17  c = matrix.matrixmultiplication(A, B)
18  print(C)
```

Listing 19.12: Python Arrays

You should test your function by do the calculations by hand and by using the the numpy functionality. Compare the results and make sure you get the same answers.

[End of Exercise]

132

Chapter 20

Complex Numbers in Python

20.1 Introduction to Complex Numbers

A complex number is defined like this:

$$z = a + jb \tag{20.1}$$

Where the imaginary unit j is defined as $i = \sqrt{-1}$

Where a is called the real part of z and b is called the imaginary part of z, i.e.:

$Re(z) = a, Im(z) = b$

Figure 20.1 shows an illustration of complex numbers.

In Python we define a complex number like this:

```
1 >>> z = 2 + 3j
```

The complex conjugate of z is defined as:

$$z* = a - jb \tag{20.2}$$

You may also imaginary numbers on exponential/polar form:

$$z = re^{j\theta} \tag{20.3}$$

Where:

$$r = |z| = \sqrt{a^2 + b^2} \tag{20.4}$$

and

$$\theta = \arctan \frac{b}{a} \tag{20.5}$$

Note that $a = r\cos\theta$ and $b = r\sin\theta$

Figure 20.2 shows an illustration of complex numbers on polar form.
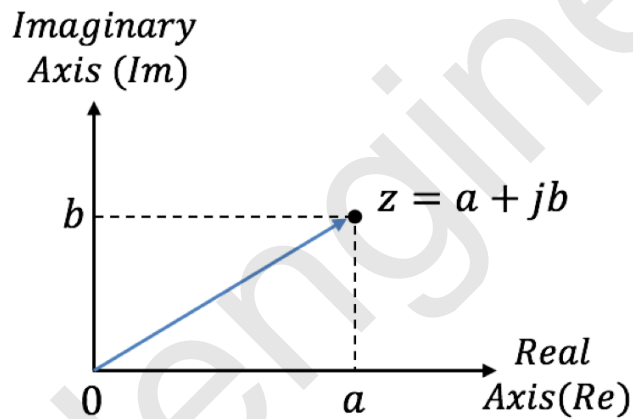
Figure 20.1: Complex Numbers

To add or subtract two complex numbers, we simply add (or subtract) their real parts and their imaginary parts.

In division and multiplication, we use the polar form.

Given the complex numbers:

$$z_1 = r_1 e^{j\theta_1} \tag{20.6}$$

$$z_2 = r_2 e^{j\theta_2} \tag{20.7}$$

Multiplication:

$$z_3 = z_1 z_2 = r_1 r_1 e^{j(\theta_1 + \theta_1)} \tag{20.8}$$
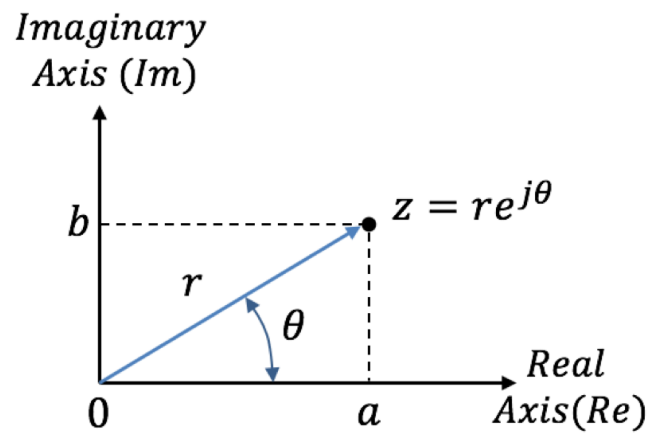
Figure 20.2: Complex Numbers - Polar form

Division:

$$z_3 = \frac{z_1}{z_2} = \frac{r_1 e^{j\theta_1}}{r_2 e^{j\theta_2}} = \frac{r_1}{r_2} e^{j(\theta_1 - \theta_2)} \tag{20.9}$$

20.2 Complex Numbers with Python

In Python you can use the cmath library which has mathematical functions for complex numbers.

https://docs.python.org/3/library/cmath.html

cmath is part of the The Python Standard Library.

For more information about the functions in the **Python Standard Library**, see:
https://docs.python.org/3/library/

Example 20.2.1. Basic Complex Numbers in Python

Given the following complex numbers:

$$a = 5 + 3j \tag{20.10}$$

$$b = 1 - 1j \tag{20.11}$$

In Python we can define the complex numbers and perform basic operations (+, -, *, /) like this:

```python
a = 5 + 3j
b = 1 - 1j

c = a + b
print(c)

d = a - b
print(d)

e = a * b
print(e)

f = a / b
print(f)
```

Listing 20.1: Basic Complex Numbers in Python

[End of Example]

Example 20.2.2. Complex Number Functions in Python

```python
import cmath

x = 2
y = -3

# converting x and y into complex number using complex()
z = complex(x,y)
print(z.real)
print(z.imag)

print(z.conjugate())

# converting complex number into polar using polar()
w = cmath.polar(z)
print(w)


# converting complex number into rectangular using rect()
w = cmath.rect(2,3)
print(w)
```

Listing 20.2: Complex Number Functions in Python

[End of Example]

Chapter 21

Differential Equations

21.1 Introduction to Differential Equations

A differential equation is a mathematical equation that relates some function with its derivatives.

In applications, the functions usually represent physical quantities, the derivatives represent their rates of change, and the differential equation defines a relationship between the two.

Because such relations are extremely common, differential equations play a prominent role in many disciplines including engineering, physics, economics, and biology.

We typically want to solve ordinary differential equations (ODE) of the form:

$$\frac{dy}{dt} = f(t, y), y(t_0) = y_0 \tag{21.1}$$

Note! Different notation is used: $\frac{dy}{dt} = y' = \dot{y}$

This document will use these different notations interchangeably.

Example 21.1.1. Example of Dynamic System

Given the following differential equation:

$$\dot{x} = -ax + bu \tag{21.2}$$

Note! $\dot{x}$ is the same as $\frac{dx}{dt}$

We have the following:

- x - Process variable, e.g., Level, Pressure, Temperature, etc.

- u - Input variable, e.g., Control Signal from the Controller

- a, b - Constants

137

[End of Example]

With Python have we can solve these differential equations in many different ways.

We can use so-called ODE solvers or we can make discrete version of the differential equations using discretization methods like Euler, etc.

With ODE solvers Python can solve these equations numerically. Higher order differential equations must be reformulated into a system of first order differential equations.

In chapter 25 we will simulate (solve and plot the results) such differential equations numerically using Euler discretization.

Example 21.1.2. Differential Equation Example

Given the following differential equation:

$$\dot{x} = ax \tag{21.3}$$

Where $a = -\frac{1}{T}$, where T is defined as the time constant of the system.

Note! $\dot{x}$ is the same as $\frac{dx}{dt}$

The solution for the differential equation is found to be:

$$x(t) = e^{at}x_0 \tag{21.4}$$

We shall plot the solution for this differential equation using Python.

In our system we can set $T = 5$ and the initial condition $x_0 = x(0) = 1$
Python code:

```python
import math as mt
import numpy as np
import matplotlib.pyplot as plt


# Parameters
T = 5
a = -1/T

x0 = 1
t = 0

tstart = 0
tstop = 25

increment = 1

x = []
x = np.zeros(tstop+1)

t = np.arange(tstart, tstop+1, increment)
```

```
22
23
24 # Define  the  Equation
25 for  k  in  range ( tstop ) :
26      x [ k ]  =  mt . exp ( a∗t [ k ] )  ∗  x0
27
28
29 # Plot  the  Results
30 plt . plot ( t , x )
31 plt . title ( ' Plotting  Differential  Equation  Solution ' )
32 plt . xlabel ( ' t ' )
33 plt . ylabel ( ' x ( t ) ' )
34 plt . grid ( )
35 plt . axis ( [ 0 ,  25 ,  0 ,  1 ] )
36 plt . show ( )
```

Listing 21.1: Differential Equation

This gives the plot shown in Figure 21.1.
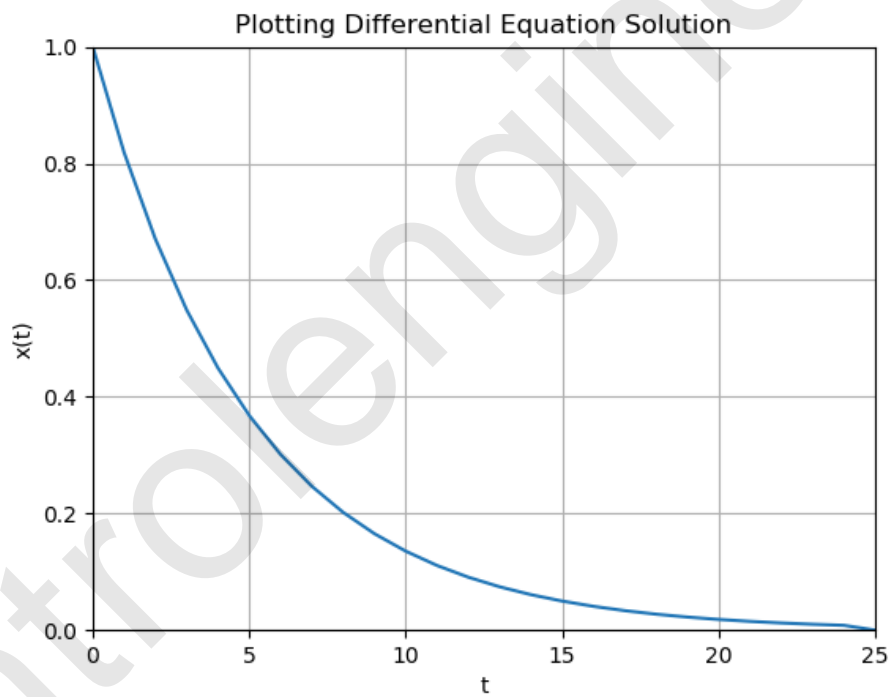
Figure 21.1: Plotting Differential Equation Solution

An alternative and perhaps simpler Python code:

```
1 import  numpy  as  np
2 import  matplotlib . pyplot  as  plt
3
4
5 # Parameters
6 T  =  5
```

139

```
7   a = -1/T
8
9   x0 = 1
10  t = 0
11
12  tstart = 0
13  tstop = 25
14  increment = 1
15  N = 25
16
17  #t = np.arange(tstart,tstop+1,increment) #Alternative Approach
18  t = np.linspace(tstart, tstop, N)
19
20  x = np.exp(a*t) * x0
21
22
23  # Plot the Results
24  plt.plot(t,x)
25  plt.title('Plotting Differential Equation Solution')
26  plt.xlabel('t')
27  plt.ylabel('x(t)')
28  plt.grid()
29  plt.axis([0, 25, 0, 1])
30  plt.show()
```

Listing 21.2: Differential Equation

This alternative Python code gives the same plot as shown in Figure 21.1.

Solving differential equations like shown in this example works fine, but the problem is that we first have to manually (by pen and paper) find the solution to the differential equation.

An alternative is to use solvers for Ordinary Differential Equations (ODE) in Python.

In the examples and tasks below we will learn how we can use these built-in ODE solvers.
Another approach is to solve such equations from "scratch" by making a discrete version of the differential equation. This approach is presented later in this textbook (chapter 25).

[End of Example]

21.2 ODE Solvers in Python

The **scipy.integrate** library has two powerful powerful functions **ode()** and **odeint()**, for numerically solving first order ordinary differential equations (ODEs). The **ode()** is more flexible, while **odeint()** (ODE integrator) has a simpler Python interface works fine for most problems.

For details, see the SciPy documentation:

https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html

https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.integrate.ode.html

Example 21.2.1. Using ODE Solver in Python

Given the following differential equation:

$$\dot{x} = ax \tag{21.5}$$

Where $a = -\frac{1}{T}$, where T is defined as the time constant of the system.

Note! $\dot{x}$ is the same as $\frac{dx}{dt}$

We will use the odeint() function.

The syntax is as follows:

```
1  x = odeint(functionname, x0, t)
```

Where we have:
functioname: Function that returns derivative values at requested x and t values
as dxdt = model(x,t)

x0: Initial conditions of the differential states

t: Time points at which the solution should be reported. Additional internal
points are often calculated to maintain accuracy of the solution but are not
reported.

Where we first has to define our differential equation:

```
1  def functionname(x, t):
2      dxdt = a * x
3      return dxdt
```

The Python code becomes:

```
1  import numpy as np
2  from scipy.integrate import odeint
3  import matplotlib.pyplot as plt
4
5  # Initialization
6  tstart = 0
7  tstop = 25
8  increment = 1
9
10 x0 = 1
11 t = np.arange(tstart, tstop+1, increment)
12
13
14 # Function that returns dx/dt
15 def mydiff(x, t):
16     T = 5
17     a = -1/T
```

141

```
18
19    dxdt = a * x
20
21    return dxdt
22
23
24 # Solve ODE
25 x = odeint(mydiff, x0, t)
26 print(x)
27
28
29 # Plot the Results
30 plt.plot(t,x)
31 plt.title('Plotting Differential Equation Solution')
32 plt.xlabel('t')
33 plt.ylabel('x(t)')
34 plt.grid()
35 plt.axis([0, 25, 0, 1])
36 plt.show()
```

Listing 21.3: Using ODE Solver in Python

This gives the same plot as shown in Figure 21.1.

Some modification to the Python code:

```
1 import numpy as np
2 from scipy.integrate import odeint
3 import matplotlib.pyplot as plt
4
5 # Initialization
6 tstart = 0
7 tstop = 25
8 increment = 1
9
10 T = 5
11 a = -1/T
12 x0 = 1
13 t = np.arange(tstart, tstop+1, increment)
14
15
16 # Function that returns dx/dt
17 def mydiff(x, t, a):
18
19    dxdt = a * x
20
21    return dxdt
22
23
24 # Solve ODE
25 x = odeint(mydiff, x0, t, args=(a,))
26 print(x)
27
28
29 # Plot the Results
30 plt.plot(t,x)
31 plt.title('Plotting Differential Equation Solution')
32 plt.xlabel('t')
33 plt.ylabel('x(t)')
34 plt.grid()
35 plt.axis([0, 25, 0, 1])
```

142

```
36  plt.show()
```

Listing 21.4: Using ODE Solver in Python

This gives the same plot as shown in Figure 21.1.

In the modified example we have the parameters used in the differential equation (in htis case a) as an input argument. By doing this, it is very easy to changes values for the parameters used in the differential equation without changing the code for the differential equation.

You can also easily run multiple simulations like this:

```
1  a = −0.2
2  x = odeint(mydiff, x0, t, args=(a,))
3
4  a = −0.1
5  x = odeint(mydiff, x0, t, args=(a,))
```

[End of Example]

21.3 Solving Multiple 1. order Differential Equations

In real life we typically have higher order differential equations, or we have a set of 1. order differential equations that describe a given system. How can we solve such equations in Python?

Example 21.3.1. Set of 1.order Differential Equations

Given the differential equations:

$$\frac{dx}{dt} = -y \tag{21.6}$$

$$\frac{dy}{dt} = x \tag{21.7}$$

Assume the initial conditions $x(0) = 1$ and $y(0) = 1$.

The Python code is almost similar as previous examples, but we need to do some small trick to make it work.

Python code:

143

```python
1  import numpy as np
2  from scipy.integrate import odeint
3  import matplotlib.pyplot as plt
4
5  # Initialization
6  tstart = -1
7  tstop = 1
8  increment = 0.1
9
10 # Initial condition
11 z0 = [1,1]
12
13
14 t = np.arange(tstart, tstop+1, increment)
15
16
17 # Function that returns dx/dt
18 def mydiff(z, t):
19     dxdt = -z[1]
20     dydt = z[0]
21
22     dzdt = [dxdt, dydt]
23     return dzdt
24
25
26 # Solve ODE
27 z = odeint(mydiff, z0, t)
28 print(z)
29
30 x = z[:,0]
31 y = z[:,1]
32
33
34 # Plot the Results
35 plt.plot(t,x)
36 plt.plot(t,y)
37 plt.title('Plotting Differential Equations Solution')
38 plt.xlabel('t')
39 plt.ylabel('z(t)')
40 plt.grid()
41 plt.axis([-1, 1, -1.5, 1.5])
42 plt.show()
```

Listing 21.5: xxx

This gives the the plot shown in Figure 21.2.

We can also rewrite the differential equations like this (to make it easier to understand?):

$$\frac{dx_1}{dt} = -x_2 \tag{21.8}$$

$$\frac{dx_2}{dt} = x_1 \tag{21.9}$$

The Python code then becomes:

144

Figure 21.2: Figure Name

```python
1  import numpy as np
2  from scipy.integrate import odeint
3  import matplotlib.pyplot as plt
4
5  # Initialization
6  tstart = -1
7  tstop = 1
8  increment = 0.1
9
10 # Initial condition
11 x_init = [1,1]
12
13
14 t = np.arange(tstart, tstop+1, increment)
15
16
17 # Function that returns dx/dt
18 def mydiff(x, t):
19     dx1dt = -x[1]
20     dx2dt = x[0]
21
22     dxdt = [dx1dt, dx2dt]
23     return dxdt
24
25
26 # Solve ODE
27 x = odeint(mydiff, x_init, t)
28 print(x)
29
30 x1 = x[:,0]
```

145

```
31  x2 = x [ : , 1 ]
32
33
34  # Plot the Results
35  plt . plot ( t , x1 )
36  plt . plot ( t , x2 )
37  plt . title ( 'Plotting Differential Equations Solution ' )
38  plt . xlabel ( 't ' )
39  plt . ylabel ( 'x(t) ' )
40  plt . grid ( )
41  plt . axis ( [ −1 , 1 , −1.5 , 1.5 ] )
42  plt . show ( )
```

<div align="center">Listing 21.6: xxx</div>

The plot and results will be the same.

<div align="right">[End of Example]</div>

21.4 Solving Higher order Differential Equations

We shall use Python to solve and plot the results of the following differential equation:

$$(1 + t^2)\ddot{w} + 2t\dot{w} + 3w = 2 \tag{21.10}$$

Note! Don't be confused that in this example w is used and not x or y. All these are just parameters or variable names.

Note! $\dot{w} = \frac{dw}{dt}$ and $\ddot{w} = \frac{d^2w}{dt^2}$

We will solve the differential equation in the interval [0,5s].

We will use the following initial conditions: $w(t_0) = 0$ and $\dot{w}(t_0) = 1$

First, we should rewrite the equation in order to get the highest derivative alone on the left side of the equation:

$$\ddot{w} = \frac{2 - 2t\dot{w} - 3w}{1 + t^2} \tag{21.11}$$

Note! Higher order differential equations must be reformulated into a system of first order differential equations.

We do the following "trick":

<div align="center">146</div>

$$w = x_1 \tag{21.12}$$

$$\dot{w} = x_2 \tag{21.13}$$

This gives a set of 1.order differential equations:

$$\dot{x}_1 = x_2 \tag{21.14}$$

$$\dot{x}_2 = \frac{2 - 2tx_2 - 3x_1}{1 + t^2} \tag{21.15}$$

Now we can relatively easy implement the system in Python.

Python code:
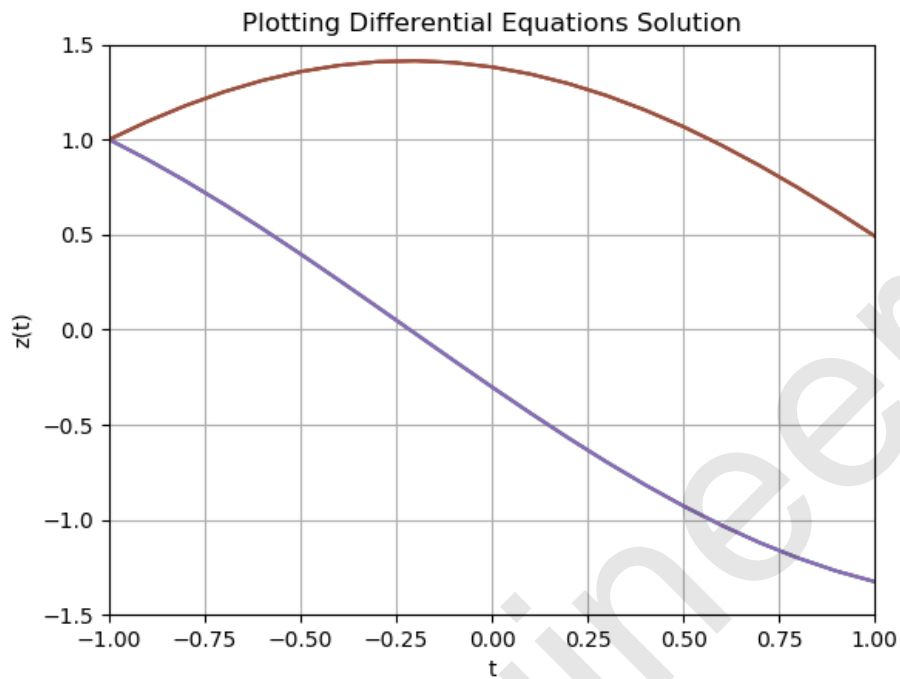
```python
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Initialization
tstart = 0
tstop = 5
increment = 0.1

# Initial condition
x_init = [0,1]


t = np.arange(tstart,tstop+1,increment)


# Function that returns dx/dt
def mydiff(x, t):
    dx1dt = x[1]
    dx2dt = (2 - t*x[1] - 3*x[0])/(1 + t**2)

    dxdt = [dx1dt, dx2dt]
    return dxdt


# Solve ODE
x = odeint(mydiff, x_init, t)
print(x)

x1 = x[:,0]
x2 = x[:,1]


# Plot the Results
plt.plot(t,x1)
plt.plot(t,x2)
plt.title('Plotting Differential Equations Solution')
plt.xlabel('t')
plt.ylabel('x(t)')
```

```
40  plt.grid()
41  plt.axis([0, 5, -1, 2])
42  plt.show()
```

Listing 21.7: xxx

This gives the the plot shown in Figure 21.3.
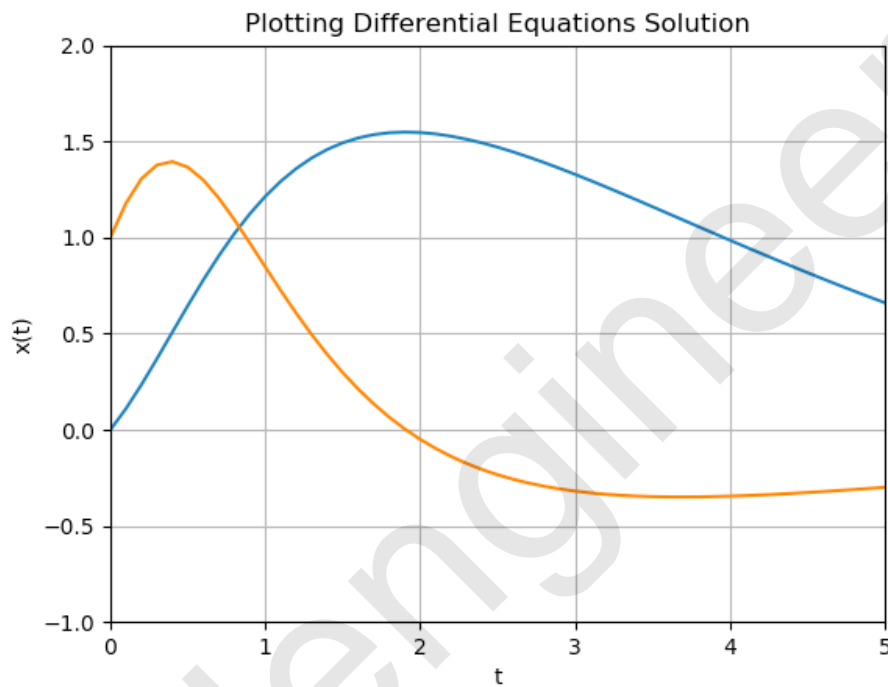
Figure 21.3: 2.order Differential Equation

21.5 Exercises

Below you find different self-paced Exercises that you should go through and solve on your own. The only way to learn Python is to do lots of Exercises!

Exercise 21.5.1. Bacteria Population

In this task we will simulate a simple model of a bacteria population in a jar.

The model is as follows:

$$\text{birth rate} = bx \qquad (21.16)$$

148

$$\text{death rate} = px^2 \tag{21.17}$$

Then the total rate of change of bacteria population is:

$$\dot{x} = bx - px^2 \tag{21.18}$$

Note! $\dot{x}$ is the same as $\frac{dx}{dt}$

Set b=1/hour and p=0.5 bacteria-hour

We will simulate the number of bacteria in the jar after 1 hour, assuming that initially there are 100 bacteria present.

Use one of the ODE solvers in Python as shown in earlier examples.

[End of Exercise]

Exercise 21.5.2. Differential Equation

Given the following differential equation:

$$\dot{x} = ax + b \tag{21.19}$$

Where $a = -\frac{1}{T}$, where T is defined as the time constant of the system. We can set $b = 1$.

Note! $\dot{x}$ is the same as $\frac{dx}{dt}$

Plot the solution for this differential equation using Python.

In our system we can set $T = 5$ and the initial condition $x_0 = x(0) = 1$
When you have done that you should try with different values for a and b. Make sure to pass these values

[End of Exercise]

Exercise 21.5.3. Simulation of Dynamic System

Given the following differential equation:

$$\dot{x} = -ax + bu \tag{21.20}$$

Note! $\dot{x}$ is the same as $\frac{dx}{dt}$

We have the following:

- x - Process variable, e.g., Level, Pressure, Temperature, etc.

149

- u - Input variable, e.g., Control Signal from the Controller

- a, b - Constants

Start by setting $a = 0.25$, $b = 2$ and $u = 1$. Plot the simulation results. Use one of the ODE solvers in Python.

Explore with other values for a, b and u.

[End of Exercise]

Exercise 21.5.4. Mass-Spring-Damper System

Given a "Mass-Spring-Damper" system as shown in Figure 21.4.
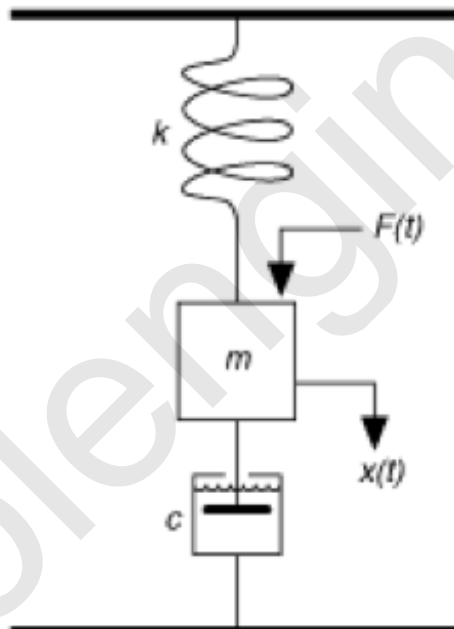
Figure 21.4: Mass-Spring-Damper System

The system can be described by the following equation:

$$F(t) - c\dot{x}(t) - kx(t) = m\ddot{x}(t) \tag{21.21}$$

Where t is the simulation time, F(t) is an external force applied to the system, c is the damping constant of the spring, k is the stiffness of the spring, m is a mass.

x(t) is the position of the object (m).

150

$\dot{x}(t)$ is the first derivative of the position, which equals the velocity of the object (m).

$\ddot{x}(t)$ is the second derivative of the position, which equals the acceleration of the object (m).

Use your skills learned from the previous examples in order to simulate this system. Use one of the ODE solvers in Python.

You should try with different values for F, c, k and m.

[End of Exercise]

Exercise 21.5.5. ODE

Use the one of the ODE solvers in Python to solve and plot the results of the following differential equation in the interval $[t_0, t_f]$:

$$3w' + \frac{1}{1+t^2}w = cos(t) \tag{21.22}$$

Where the initial conditions are $t_0 = 0$, $t_f = 5$, $w(t_0) = 1$

Note! w' is the same as $\dot{w}$ which is the same as $\frac{dw}{dt}$ - different notations for the same.

[End of Exercise]

Exercise 21.5.6. Pendulum model

Use the one of the ODE solvers in Python to solve and plot the results of the following differential equations:

$$\dot{x}_1 = x_2 \tag{21.23}$$

$$\dot{x}_2 = -\frac{g}{r}x_1 - \frac{b}{mr^2}x_2 \tag{21.24}$$

The differential equations above is a simplified model of a pendulum where m is the mass, r is the length of the arm of the pendulum, g is the gravity, and b is a friction coefficient.

In the model, x_1 is the distance from the starting point (which is when the pendulum hangs straight down) and x_2 is the velocity.

151
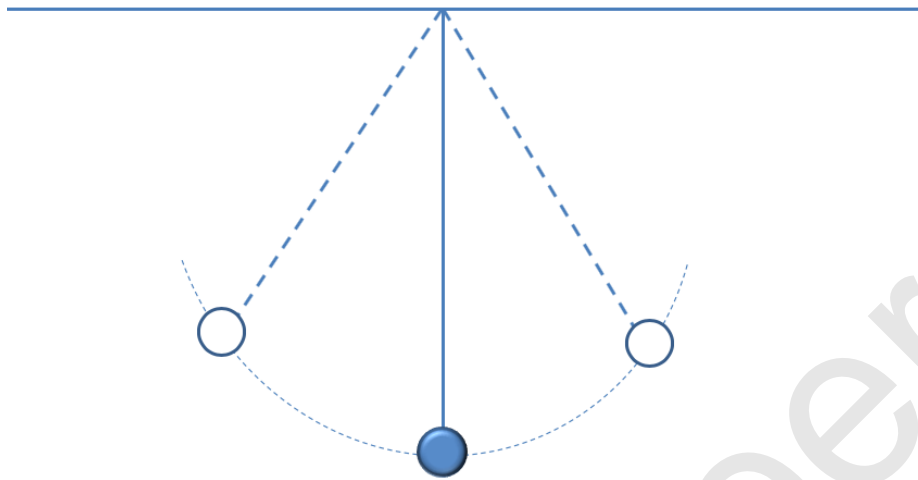
Figure 21.5: Pendulum

A pendulum is shown in Figure 21.5.

You may use the following values: $m = 8$, $r = 5$ and $b = 10$ (units are not so important in this case). You may also explore with other values as well.

Assume you, e.g., take the pendulum away from the starting point and then drop it, what happens then? You may, e.g., use the initial conditions $x_1(0) = 0.5$ and $x_2(0) = 0$.

Explain the simulation results and see it in relation with the real world. Does the simulation results make sense?

[End of Exercise]

Chapter 22

Optimization

Optimization is important in mathematics, control and simulation applications. Optimization is based on finding the minimum of a given criteria function.

Example 22.0.1. Basic Optimization

Given the following function:

$$f(x) = x^2 + 2x + 1 \tag{22.1}$$

We start by plotting the function:

```python
import numpy as np
import matplotlib.pyplot as plt

xmin = -5
xmax = 5
dx = 0.1

N = int((xmax - xmin)/dx)

x = np.linspace(xmin, xmax, N+1)
#x = np.arange(xmin, xmax+dx, dx) #Alternative implementation

y = x**2 + 2*x + 1;


plt.plot(x,y)
plt.xlim([xmin,xmax])
```

Listing 22.1: Optimization in Python

This gives the the plot shown in Figure 22.1.

We will use **fminbound** to find the minimum of the function.

Python code:

```python
from scipy import optimize

```
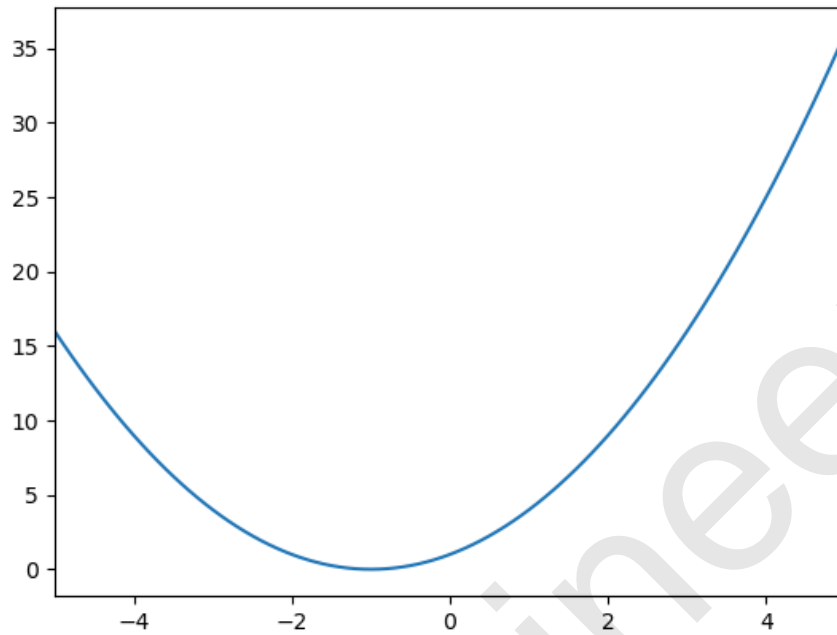
Figure 22.1: Optimization Example

```
3  xmin = −5
4  xmax = 5
5
6
7  def y(x):
8      return x**2 + 2*x + 1
9
10
11 x_min = optimize.fminbound(y, xmin, xmax)
12
13 print(x_min)
```

Listing 22.2: xxx

The result becomes:

xmin = -1.0

We see that this is the correct answer based on our plot in Figure 22.1.

[End of Example]

Exercise 22.0.1. Make your own Minimization function

Given the following function:

$$f(x) = x^2 + 2x + 1 \tag{22.2}$$

154

Implement a basic minimization function from scratch using either a For loop or a While loop.

Make sure to test the function and see if you get the same answer as in the example above.

<div align="right">[End of Exercise]</div>

Other optimization functions in Python:

scipy.optimize.fmin

scipy.optimize.minimize_scalar

scipy.optimize.minimize

Exercise 22.0.2. Optimization Functions in Python

Given the following function:

$$f(x) = x^2 + 2x + 1 \tag{22.3}$$

Test the different optimization (finding minimum) functions:

scipy.optimize.fmin

scipy.optimize.minimize_scalar

scipy.optimize.minimize

Compare the results.

<div align="right">[End of Exercise]</div>

Exercise 22.0.3. Optimization

Given the following function:

$$f(x) = x^3 - 4x \tag{22.4}$$

Test the different optimization (finding minimum) functions:

scipyoptimize.fminbound

<div align="center">155</div>

scipy.optimize.fmin

scipy.optimize.minimize_scalar

scipy.optimize.minimize

Compare the results. You should alos plot the function.

Exercise 22.0.4. Minimum for function with 2 variables

Given the following function:

$$f(x,y) = 2(x-1)^2 + x - 2 + (y-2)^2 + y \qquad (22.5)$$

Plot the function and find the minimum.

Exercise 22.0.5. Optimization - Rosenbrock's Banana Function

Given the following function:

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2 \qquad (22.6)$$

This function is known as Rosenbrock's banana function.

Plot the function and find the minimum.

Part VI

Using Python for Simulations

Chapter 23

Introduction to Simulations

Python is very powerful for performing simulations, like simulating dynamic systems, i.e.., solving numerical differential equations.

See the next chapters for lots of practical examples.

Chapter 24

Differential Equations

24.1 Introduction to Differential Equations

A differential equation is a mathematical equation that relates some function with its derivatives.

In applications, the functions usually represent physical quantities, the derivatives represent their rates of change, and the differential equation defines a relationship between the two.

Because such relations are extremely common, differential equations play a prominent role in many disciplines including engineering, physics, economics, and biology.

We typically want to solve ordinary differential equations (ODE) of the form:

$$\frac{dy}{dt} = f(t, y), y(t_0) = y_0 \tag{24.1}$$

Note! Different notation is used: $\frac{dy}{dt} = y' = \dot{y}$

This document will use these different notations interchangeably.

Example 24.1.1. Example of Dynamic Systems

Given the following differential equation:

$$\dot{x} = -ax + bu \tag{24.2}$$

Note! $\dot{x}$ is the same as $\frac{dx}{dt}$

We have the following:

- x - Process variable, e.g., Level, Pressure, Temperature, etc.

- u - Input variable, e.g., Control Signal from the Controller

- a, b - Constants

159

[End of Example]

With Python have we can solve these differential equations in many different ways.

We can use so-called ODE solvers or we can make discrete version of the differential equations using discretization methods like Euler, etc.

With ODE solvers Python can solve these equations numerically. Higher order differential equations must be reformulated into a system of first order differential equations.

In chapter 25 we will simulate (solve and plot the results) such differential equations numerically using Euler discretization.

Chapter 25

Discrete Systems

When dealing with computers we need to deal with discrete systems.

25.1 Discretization

Sometimes we want to or need to discretize a continuous system and then simulate it in Python. When dealing with computer simulation, we need to create a discrete version of our system. This means we need to make a discrete version of our continuous differential equations. Interpolation, Curve Fitting, etc. is also based on a set of discrete values (data points or measurements). The same with Numerical Differentiation and Numerical Integration, etc.

Below we see a continuous signal vs the discrete signal for a given system with discrete time interval $Ts = 0.1s$.
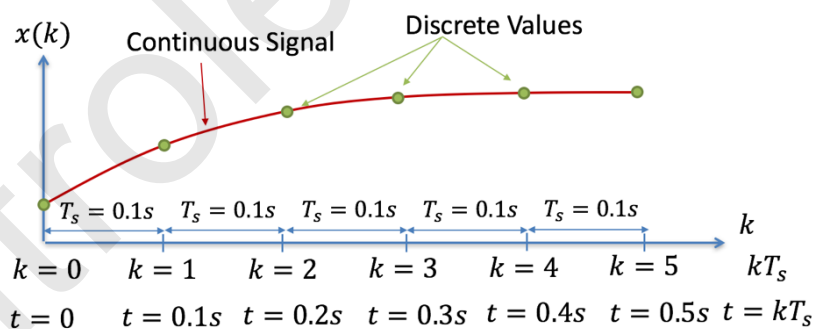
Figure 25.1: Discretization

In order to discretize a continuous model there are lots of different methods to use.
A simple discretization method is the Euler Forward method:

$$\dot{x} = \frac{x(k+1) - x(k)}{T_s} \tag{25.1}$$

T_s is the Sampling Time
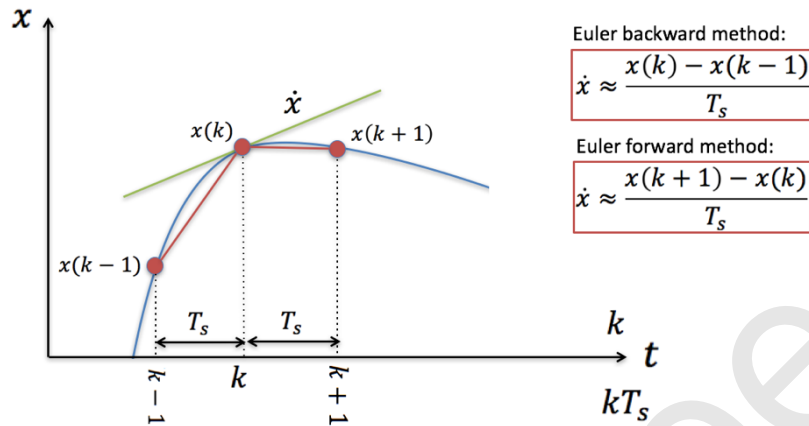
Figure 25.2: Euler Discretization methods

Lots of other discretization methods do exists, such as "Euler backward", Zero Order Hold (ZOH), Tustin's method, etc.

Example 25.1.1. Simulation of Discrete System

Given the following differential equation:

$$\dot{x} = -ax + bu \tag{25.2}$$

Note! $\dot{x}$ is the same as $\frac{dx}{dt}$

We have the following:

- x - Process variable, e.g., Level, Pressure, Temperature, etc.

- u - Input variable, e.g., Control Signal from the Controller

- a, b - Constants

We start with finding the discrete differential equation.

We can use e.g., the Euler Approximation:

$$\dot{x} \approx \frac{x_{k+1} - x_k}{T_s} \tag{25.3}$$

T_s - Sampling Interval

Then we get:

$$\frac{x_{k+1} - x_k}{T_s} = -ax_k + bu_k \tag{25.4}$$

162

This gives the following discrete differential equation:

$$x_{k+1} = (1 - T_s a)x_k + T_s b u_k \qquad (25.5)$$

Now we are ready to simulate the system.

We set a=0.25, b=2 and u=1 (You can explore with other values on your own)
We start creating the Python Script for the simulation of this system:

```python
# Simulation of discrete model
import numpy as np
import matplotlib.pyplot as plt

# Model Parameters
a = 0.25
b = 2

# Simulation Parameters
Ts = 0.1
Tstop = 30
uk = 1 # Step Response
xk = 0
N = int(Tstop/Ts) # Simulation length
data = []
data.append(xk)


# Simulation
for k in range(N):
    xk1 = (1 - a*Ts) * xk + Ts * b * uk
    xk = xk1
    data.append(xk1)


# Plot the Simulation Results
t = np.arange(0, Tstop+Ts, Ts)

plt.plot(t, data)

# Formatting the appearance of the Plot
plt.title('Simulation of dxdt = -ax + bu')
plt.xlabel('t [s]')
plt.ylabel('x')
plt.grid()
plt.axis([0, 30, 0, 8])
plt.show()
```

Listing 25.1: Simulation of Discrete Dynamic System in Python

The simulation gives the results as shown in Figure 25.3.

Lets also take a look at the Variable Explorer as shown in Figure 25.4.

163

Figure 25.3: Simulation of Discrete System using Python

Here is an alternative solution presented:

```python
# Simulation of discrete model
import numpy as np
import matplotlib.pyplot as plt

# Model Parameters
a = 0.25
b = 2

# Simulation Parameters
Ts = 0.1 # Sampling Time
Tstop = 30 # End of Simulation Time
uk = 1 # Step Response
N = int(Tstop/Ts) # Simulation length
x = np.zeros(N+2) # Initialization the x vector
x[0] = 0


# Simulation
for k in range(N+1):
    x[k+1] = (1 - a*Ts) * x[k]  + Ts * b * uk


# Plot the Simulation Results
t = np.arange(0,Tstop+2*Ts,Ts) #Create the Time Series

plt.plot(t,x)

# Formatting the appearance of the Plot
plt.title('Simulation of dxdt = -ax + bu')
plt.xlabel('t [s]')
plt.ylabel('x')
```

164

Figure 25.4: Variable Explorer for Discrete Simulation Example

```
32  plt.grid()
33  plt.axis([0, 30, 0, 8])
34  plt.show()
```

Listing 25.2: Simulation of Discrete Dynamic System in Python

This gives of course the same results and the same plot.

You should try both examples, and then decide which one you prefer. I guess there are also many other ways to do it.

[End of Example]

25.2 Exercises

Below you find different self-paced Exercises that you should go through and solve on your own. The only way to learn Python is to do lots of Exercises!

Exercise 25.2.1. Simulation of Bacteria Population

In this task we will simulate a simple model of a bacteria population in a jar.

The model is as follows:

$$\text{birth rate} = bx \tag{25.6}$$

$$\text{death rate} = px^2 \tag{25.7}$$

Then the total rate of change of bacteria population is:

165

$$\dot{x} = bx - px^2 \tag{25.8}$$

Set b=1/hour and p=0.5 bacteria-hour

We will simulate the number of bacteria in the jar after 1 hour, assuming that initially there are 100 bacteria present.

Find the discrete model using the Euler Forward method by hand and implement and simulate the system in Python using a For Loop.

[End of Example]

Exercise 25.2.2. Simulation with 2 variables

Given the following system:

$$\frac{dx_1}{dt} = -x_2 \tag{25.9}$$

$$\frac{dx_2}{dt} = x_1 \tag{25.10}$$

Find the discrete system and simulate the discrete system in MATLAB. Solve the equations, e.g., in the time span [-1 1] with initial values [1, 1].

[End of Exercise]

Chapter 26

Real-Time Simulations

26.1 Introduction

Typically in a simulation, you run the simulation in a for loop. When you are finished with the simulation you plot the data.
We repeat a basic simulation example from chapter 25.

Example 26.1.1. Basic Simulation of Discrete System

We will simulate the discrete system given as follows:

$$x_{k+1} = (1 - T_s a)x_k + T_s b u_k \tag{26.1}$$

We set a=0.25, b=2 and u=1 (You can explore with other values on your own)
We start creating the Python Script for the simulation of this system:

```python
# Simulation of discrete model
import numpy as np
import matplotlib.pyplot as plt

# Model Parameters
a = 0.25
b = 2

# Simulation Parameters
Ts = 0.1
Tstop = 30
uk = 1 # Step Response
xk = 0
N = int(Tstop/Ts) # Simulation length
data = []
data.append(xk)


# Simulation
for k in range(N):
    xk1 = (1 - a*Ts) * xk  + Ts * b * uk
```

```
22    xk = xk1
23    data.append(xk1)
24
25
26  # Plot the Simulation Results
27  t = np.arange(0, Tstop+Ts, Ts)
28
29  plt.plot(t, data)
30
31  # Formatting the appearance of the Plot
32  plt.title('Simulation of dxdt = -ax + bu')
33  plt.xlabel('t [s]')
34  plt.ylabel('x')
35  plt.grid()
36  plt.axis([0, 30, 0, 8])
37  plt.show()
```

Listing 26.1: Simulation of Discrete Dynamic System in Python

The simulation gives the results as shown in Figure 26.1.
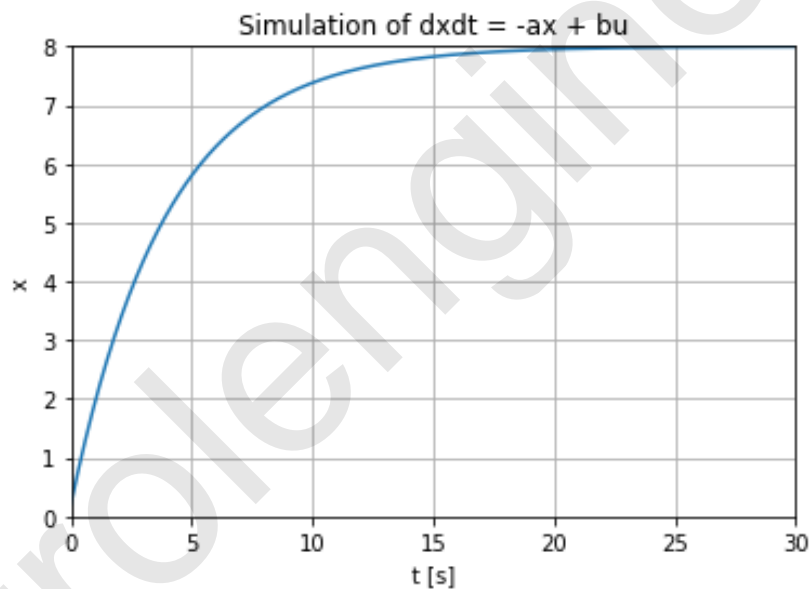
Figure 26.1: Simulation of Discrete System using Python

As you can see from the code and when running the code, the simulation results are plotted all in one operation after the simulation is finished, i.e, after the for loop.

[End of Example]

Sometimes we want to plot one value at the time inside the loop, so-called "Real-Time simulations". That is also the case if we want to plot data from a sensor or a real process.

26.2 Introduction to Real-Time Plotting

You can also use the matplotlib for real-time plotting.

Example 26.2.1. Introduction to Real-Time Plotting

Here is a basic example:

```python
import numpy as np
import matplotlib.pyplot as plt

plt.axis([0, 10, 0, 1])

delay = 1 #Seconds

for i in range(10):
    y = np.random.random()
    plt.scatter(i, y)
    plt.pause(delay)

plt.show()
```

Listing 26.2: Real-Time Plotting in Python

We get the following plot as shown in Figure 27.1.
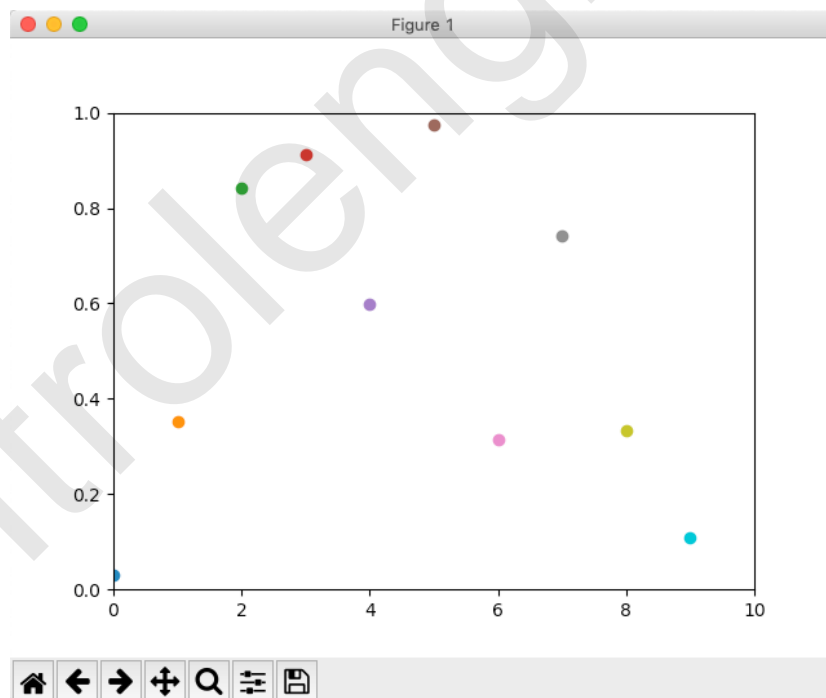
Figure 26.2: Real-Time Plotting with Python

You cannot see the the actual behavior of the plot by watching Figure 27.1, so

you need to run the Python program yourself.

If you run the code you see the plot is updated with a new value every second as specified in the code.

[End of Example]

Note! If you use Anaconda and Spider, you typically need to change the the settings for how graphics are are displayed in Spyder.

Select Preferences from the menu, then IPython console in the list of categories on the left, then the tab Graphics at the top, and change the Graphics back-end from Inline to e.g. Automatic or Qt. See Figure 27.2.
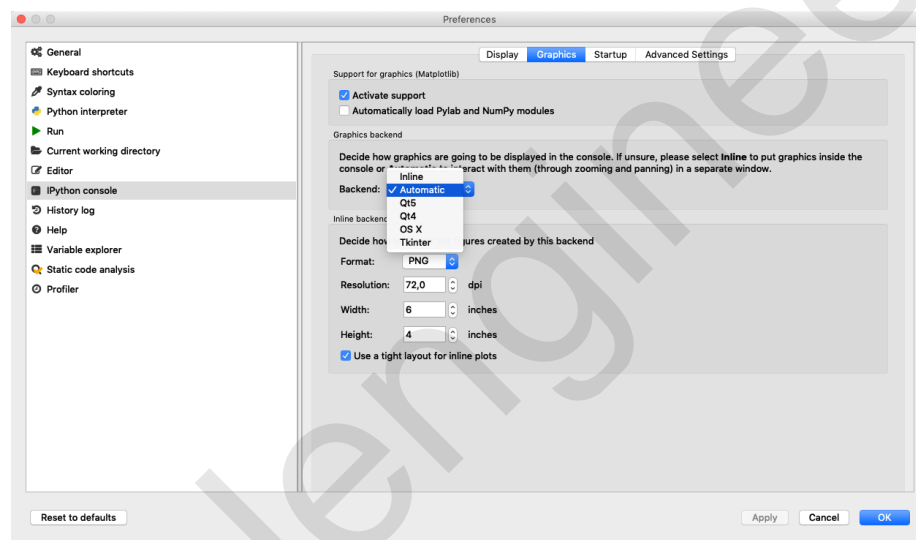
Figure 26.3: Change how Graphics are displayed in the Spyder Editor

Example 26.2.2. Real-Time Simulation of Discrete System

Lets change the discrete simulation example presented in the beginning of this chapter:

```python
# Real-Time Simulation of Discrete System
import numpy as np
import matplotlib.pyplot as plt

# Model Parameters
a = 0.25
b = 2

# Simulation Parameters
Ts = 0.1
Tstop = 30
uk = 1 # Step Response
```

```
13  xk = 0
14  N = int(Tstop/Ts) # Simulation length
15  data = []
16  data.append(xk)
17
18  plt.axis([0, N, 0, 10])
19
20  for k in range(N):
21      xk1 = (1 - a*Ts) * xk  + Ts * b * uk
22      xk = xk1
23      data.append(xk1)
24
25      plt.scatter(k, xk1)
26      plt.pause(Ts)
27
28  plt.show()
```

Listing 26.3: Real-Time Simulation of Discrete Dynamic System in Python

Figure 26.4 shows the Real-Time Plot for this example.
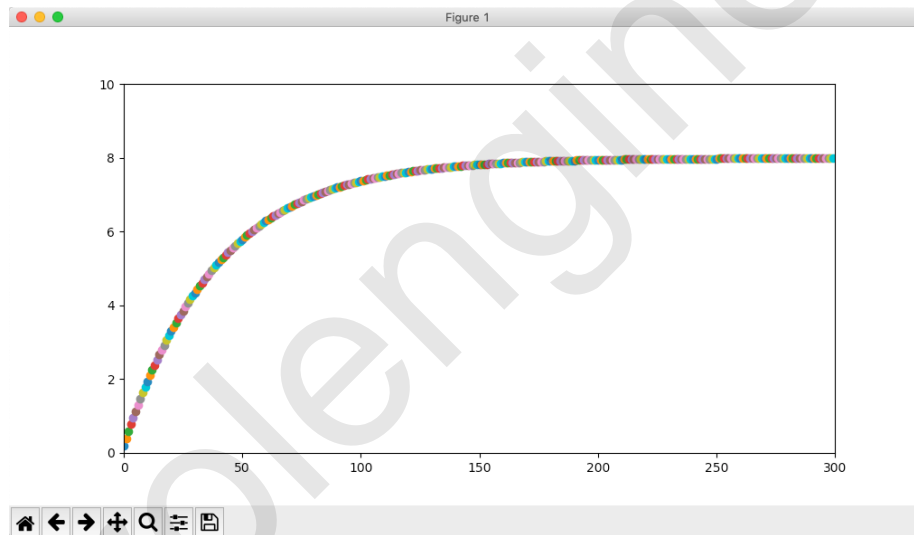
Figure 26.4: Real-Time Simulation of Discrete System

You cannot see the the actual behavior of the plot by watching Figure 26.4, so you need to run the Python program yourself.

If you run the code you see the plot is updated with a new value every second as specified in the code.

In this example we have locked the scaling using the axis method. If we remove or comment out the line "plt.axis([0, N, 0, 10])", we get a plot the automatically scales the x axis and the y axis. Whats best depends on if you know the simulation length from the beginning and know the minimum and maximum value of the simulation results.

171

Figure 26.5 shows the Real-Time Plot with Auto-scaling after 30 iterations for this example. You cannot see the the actual behavior of the plot by watching Figure 26.5, so you need to run the Python program yourself.

Figure 26.5: Real-Time Simulation of Discrete System with Auto-scaling

The examples shown shows the discrete time step k on the x-axis. Typically we want to display the continuous time t instead. Some small adjustment to the examples make this possible. See the code below:

```
1  # Real−Time Simulation of Discrete System
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # Model Parameters
6  a = 0.25
7  b = 2
8
9  # Simulation Parameters
10 Ts = 0.1
11 Tstop = 30
12 uk = 1 # Step Response
13 xk = 0
14 y_scale_max = 10
15 N = int(Tstop/Ts) # Simulation length
16 data = []
17 data.append(xk)
```

172

```
18
19 plt.axis([0, N*Ts, 0, y_scale_max])
20
21 for k in range(N):
22     xk1 = (1 - a*Ts) * xk  + Ts * b * uk
23     xk = xk1
24     data.append(xk1)
25
26     t = Ts*k
27
28     plt.scatter(t, xk1)
29     plt.pause(Ts)
30
31 plt.show()
```

Listing 26.4: Real-Time Simulation of Discrete Dynamic System in Python

Figure 26.6 shows the final plot for this example. You cannot see the the actual behavior of the plot by watching Figure 26.6, so you need to run the Python program yourself.
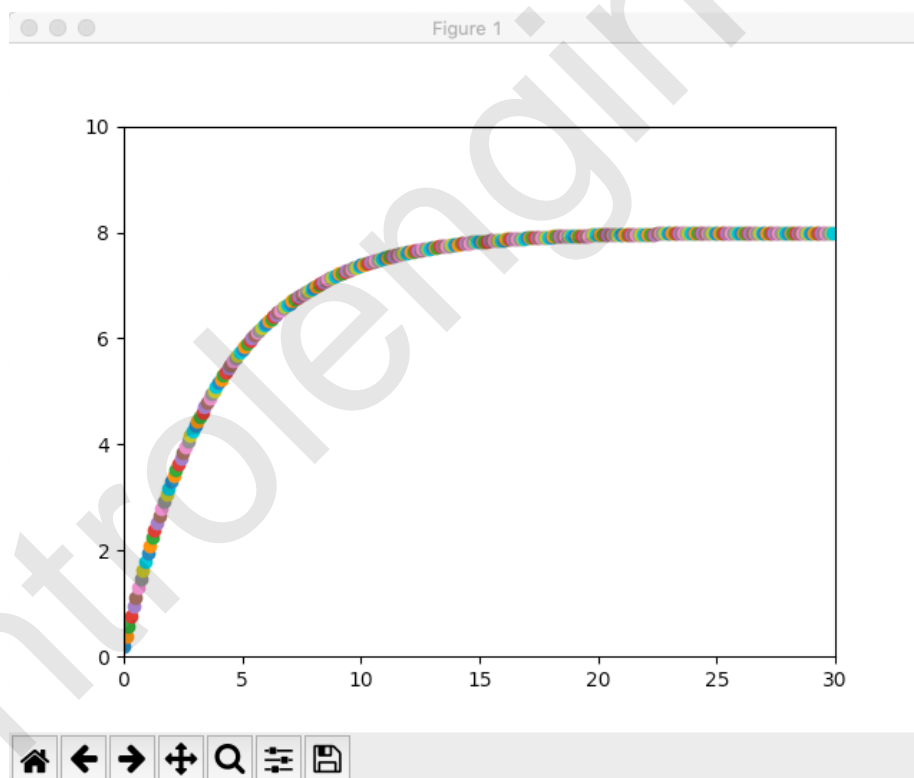
Figure 26.6: Real-Time Simulation of Discrete System

We can also add some formatting regarding the appearance of the plot (xlabel, ylabel, title, etc.). The final code example is shown below.

```python
1  # Real−Time Simulation of Discrete System
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # Model Parameters
6  a = 0.25
7  b = 2
8
9  # Simulation Parameters
10 Ts = 0.1
11 Tstop = 30
12 uk = 1 # Step Response
13 xk = 0
14 y_scale_max = 10
15 N = int(Tstop/Ts) # Simulation length
16 data = []
17 data.append(xk)
18
19
20 # Formatting the appearance of the Plot
21 plt.title('Simulation of dxdt = −ax + bu')
22 plt.xlabel('t [s]')
23 plt.ylabel('x')
24 plt.grid()
25
26 plt.axis([0, N*Ts, 0, y_scale_max])
27
28 for k in range(N):
29     xk1 = (1 − a*Ts) * xk  + Ts * b * uk
30     xk = xk1
31     data.append(xk1)
32
33     t = Ts*k
34
35     plt.scatter(t, xk1)
36     plt.pause(Ts)
37
38 plt.show()
```

Listing 26.5: Real-Time Simulation of Discrete Dynamic System in Python

Run the example and notice the difference.

[End of Example]

26.3 Real-Time Plotting with Animation

For more advanced Real-Time plots we should use the animation module in the matplotlib library (matplotlib.animation).

To create a real-time plot, we need to use the animation module in matplotlib. We set up the figure and axes in the usual way, but we draw directly to the axes, ax, when we want to create a new frame in the animation.

We need to use the FuncAnimation function:

```
1  ani = animation.FuncAnimation(fig, animate, fargs=(xs, ys),
       interval=1000)
```

FuncAnimation is a special function within the animation module that lets us automate updating the graph. We pass the FuncAnimation() a handle to the figure we want to draw, fig, as well as the name of a function that should be called at regular intervals. We called this function animate() and is defined just above our FuncAnimation() call.

Still in the FuncAnimation() parameters, we set fargs, which are the arguments we want to pass to our animate function (since we are not calling animate() directly from within our own code). Then, we set interval, which is how long we should wait between calls to animate() (in milliseconds).

Note: As an argument to FuncAnimation, notice that animate does not have any parentheses. This is passing a reference to the function and not the result of that function. If you accidentally add parentheses to animate here, animate will be called immediately (only once), and you'll likely get an error

Example 26.3.1. Real-Time Plotting with Animation

Below you find the Python Code for a basic example where we use the animation module in matplotlib.

In the example we update the plot every seconds by setting the interval=1000ms as an input argument to the FuncAnaimation function.

```python
1  import datetime as dt
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import matplotlib.animation as animation
5
6  # Create figure for plotting
7  fig = plt.figure()
8  ax = fig.add_subplot(1, 1, 1)
9  xs = []
10 ys = []
11
12
13 # This function is called periodically from FuncAnimation
14 def animate(i, xs, ys):
15
16     temp_c = round(np.random.random(), 2)
17
18     # Add x and y to lists
19     xs.append(dt.datetime.now().strftime('%H:%M:%S.%f'))
20     ys.append(temp_c)
21
22     # Limit x and y lists to 20 items
23     xs = xs[-20:]
24     ys = ys[-20:]
25
26     # Draw x and y lists
27     ax.clear()
28     ax.plot(xs, ys)
```

```
29
30     # Format plot
31     plt.xticks(rotation=45, ha='right')
32     plt.subplots_adjust(bottom=0.30)
33     plt.title('Temperature Data')
34     plt.ylabel('Temperature (deg C)')
35
36 # Set up plot to call animate() function periodically
37 ani = animation.FuncAnimation(fig, animate, fargs=(xs, ys),
       interval=1000)
38 plt.show()
```

Listing 26.6: Real-Time Plotting with Animation

Figure 27.3 shows the final plot for this example. You cannot see the the actual behavior of the plot by watching Figure 27.3, so you need to run the Python program yourself.
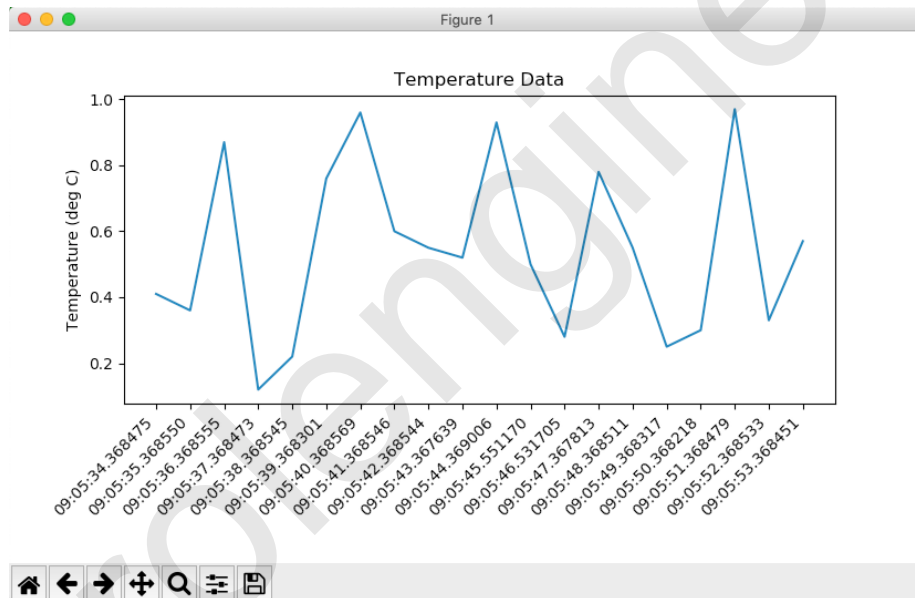
Figure 26.7: Real-Time Plotting with Animation

[End of Example]

Example 26.3.2. Discrete Simulations with Animation

Lets apply this technique on our discrete system.

Python Code:

```
 1  import datetime as dt
 2  import numpy as np
 3  import matplotlib.pyplot as plt
 4  import matplotlib.animation as animation
 5  # Create figure for plotting
 6  fig = plt.figure()
 7  ax = fig.add_subplot(1, 1, 1)
 8  xs = []
 9  ys = []
10
11  xk = 0 # Initial Value for x
12
13  # This function is called periodically from FuncAnimation
14  def simulation(i, xs, ys):
15
16      # Model Parameters
17      a = 0.25
18      b = 2
19
20      # Simulation Parameters
21      Ts = 0.1 # Sampling Time
22      uk = 1 # Step Response
23      global xk # Since we change x inside a function we define it as
          a gobal variable
24
25      # Model of Discrete System
26      xk1 = (1 - a*Ts) * xk  + Ts * b * uk
27      xk = xk1
28
29      # Add x and y to lists
30      xs.append(dt.datetime.now().strftime('%H:%M:%S.%f'))
31      ys.append(xk1)
32
33      N = 60 # Limit x and y lists to N items
34      xs = xs[-N:]
35      ys = ys[-N:]
36
37      # Draw x and y lists
38      ax.clear()
39      ax.plot(xs, ys)
40
41      # Format plot
42      plt.xticks(rotation=45, ha='right')
43      plt.subplots_adjust(bottom=0.30)
44      plt.title('Simulation of dxdt = -ax + bu')
45      plt.xlabel('t [s]')
46      plt.ylabel('x')
47      plt.grid()
48
49  # Set up plot to call animate() function periodically
50  ani = animation.FuncAnimation(fig, simulation, fargs=(xs, ys),
       interval=100)
51  plt.show()
```

Listing 26.7: Real-Time Simulation of Discrete Dynamic System with Animation

Figure 26.8 shows the final plot for this example. You cannot see the the actual behavior of the plot by watching Figure 26.8, so you need to run the Python program yourself.

177

Figure 26.8: Real-Time Simulation of Discrete Dynamic System with Animation

[End of Example]

26.3.1 Speeding Up the Plot Animation

Clearing a graph and redrawing everything can be a time-consuming process in terms of computer time. To remedy that, we are going to use a trick known as blitting.

Blitting is an old computer graphics technique where several graphical bitmaps are combined into one. This way, only one needed to be updated at a time, saving the computer from having to redraw the whole scene every time.
Matplotlib allows us to enable blitting in FuncAnimation, but it means we need to re-write how some of the animate() function works. To reap the true benefits of blitting, we need to set a static background, which means the axes can't scale and we can't show moving timestamps anymore. This means that you have to take the good with the bad. So you have to choose whats most important for you in your simulations.

Example 26.3.3. Real-Time Plotting with Animation with improved Performance

Python Code:

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# Parameters
x_len = 200            # Number of points to display
```

178

```
7  y_range = [0, 20]  # Range of possible Y values to display
8
9  # Create figure for plotting
10 fig = plt.figure()
11 ax = fig.add_subplot(1, 1, 1)
12 xs = list(range(0, 200))
13 ys = [0] * x_len
14 ax.set_ylim(y_range)
15
16
17 # Create a blank line. We will update the line in animate
18 line, = ax.plot(xs, ys)
19
20 # Add labels
21 plt.title('Temperature Data')
22 plt.xlabel('Samples')
23 plt.ylabel('Temperature (deg C)')
24
25 # This function is called periodically from FuncAnimation
26 def animate(i, ys):
27
28     rand_val = np.random.random()*20 #Generate Random Values
        between 0 and 20
29
30     temp_c = round(rand_val, 2)
31
32     #print (temp_c)
33
34     # Add y to list
35     ys.append(temp_c)
36
37     # Limit y list to set number of items
38     ys = ys[-x_len:]
39
40     # Update line with new Y values
41     line.set_ydata(ys)
42
43     return line,
44
45 # Set up plot to call animate() function periodically
46 ani = animation.FuncAnimation(fig,
47     animate,
48     fargs=(ys,),
49     interval=100,
50     blit=True)
51 plt.show()
```

Listing 26.8: Real-Time Plotting with Animation

Figure 27.4 shows the final plot for this example. You cannot see the the actual behavior of the plot by watching Figure 27.4, so you need to run the Python program yourself.
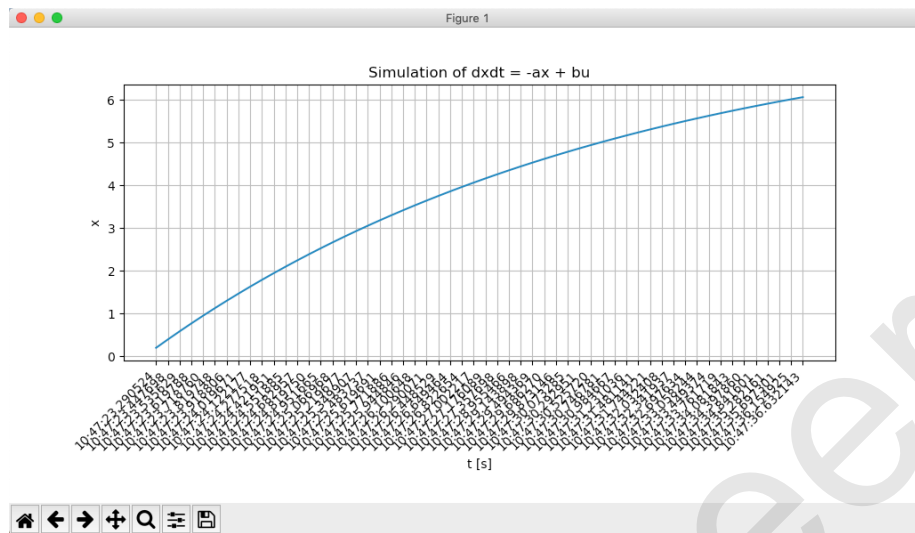
[End of Example]

Example 26.3.4. Discrete Simulations with Animation with improved Performance

179
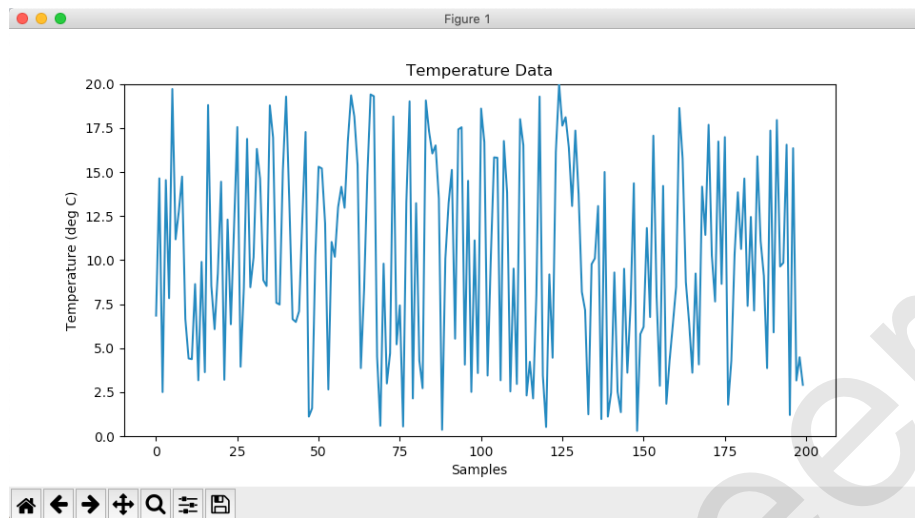
Figure 26.9: Real-Time Plotting with Animation

Python Code:

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# Parameters
N = 200
x_len = N               # Number of points to display
y_range = [0, 10]   # Range of possible Y values to display

# Create figure for plotting
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
xs = list(range(0, N))
ys = [0] * x_len
ax.set_ylim(y_range)

xk = 0 # Initial Value for x

# Create a blank line. We will update the line in animate
line, = ax.plot(xs, ys)

# Add labels
plt.title('Simulation of dxdt = -ax + bu')
plt.xlabel('Samples')
plt.ylabel('x')

# This function is called periodically from FuncAnimation
def simulation(i, ys):

    # Model Parameters
    a = 0.25
    b = 2

    # Simulation Parameters
    Ts = 0.1 # Sampling Time
    uk = 1 # Step Response
```

180

```
37        global xk # Since we change x inside a function we define it as
          a gobal variable
38
39        # Model of Discrete System
40        xk1 = (1 - a*Ts) * xk  + Ts * b * uk
41        xk = xk1
42
43        # Add y to list
44        ys.append(xk1)
45
46        # Limit y list to set number of items
47        ys = ys[-x_len:]
48
49        # Update line with new Y values
50        line.set_ydata(ys)
51
52        return line,
53
54 # Set up plot to call animate() function periodically
55 ani = animation.FuncAnimation(fig,
56        simulation,
57        fargs=(ys,),
58        interval=100,
59        blit=True)
60 plt.show()
```

Listing 26.9: Real-Time Simulation of Discrete Dynamic System with Animation and increased Speed

Figure 26.10 shows the final plot for this example. You cannot see the the actual behavior of the plot by watching Figure 26.10, so you need to run the Python program yourself.

[End of Example]

For more information about Matplotlib:animations:

https://scipy-cookbook.readthedocs.io/items/Matplotlib$_A$nimations.html

Other resources:

https://learn.sparkfun.com/tutorials/graph-sensor-data-with-python-and-matplotlib/allplot-sensor-data

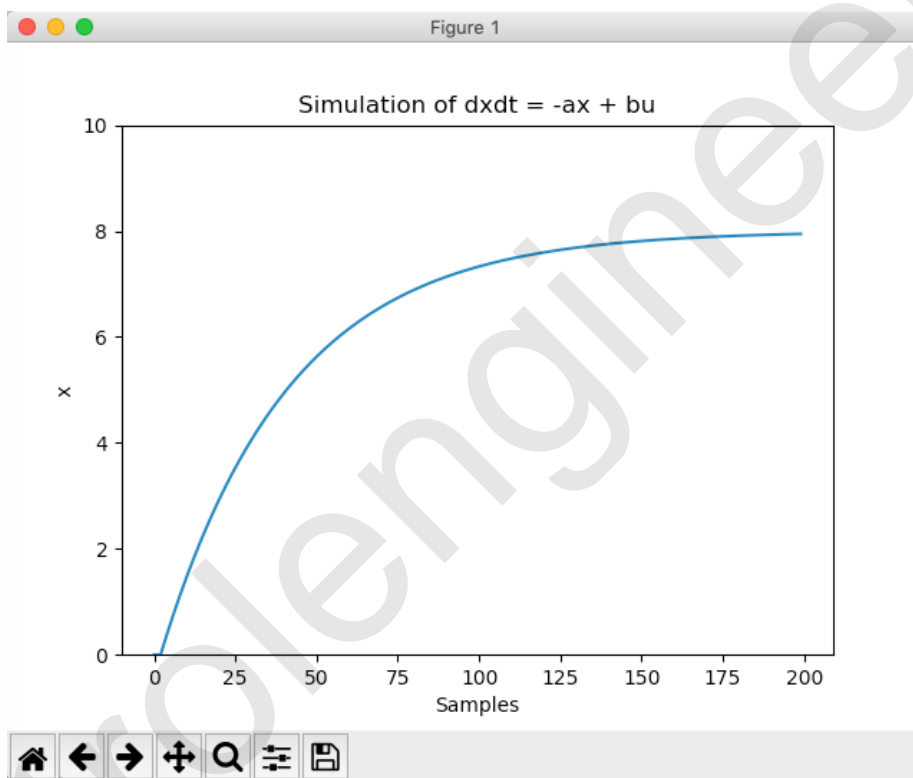https://stackoverflow.com/questions/11874767/how-do-i-plot-in-real-time-in-a-while-loop-using-matplotlib

Figure 26.10: Real-Time Plotting with Animation

Part VII

Data Acquisition (DAQ) with Python

Chapter 27

Plotting Sensor Data

27.1 Introduction

Typically we want to plot the data from the sensor. We can plot save the data in an array and then plot the data at the end of the program, but more likely we want to plot one value at the time inside the loop, so-called "Real-Time plotting".

In this chapter we only show how you can plot the data from any given sensor using this general approach. Instead of the actual sensor data we just use the random generator in Python.

To read the actual sensor data you typically need a DAQ (Data Acquisition) device connected to you PC or, e.g, a Raspberry Pi device. In all cases you will typically need to install a driver from the vendor of the DAQ device or the sensor you are using.

27.2 Introduction to Real-Time Plotting

You can also use the matplotlib for real-time plotting.

Example 27.2.1. Introduction to Real-Time Plotting

Here is a basic example:

```python
import numpy as np
import matplotlib.pyplot as plt

plt.axis([0, 10, 0, 1])

delay = 1 #Seconds

for i in range(10):
    y = np.random.random()
    plt.scatter(i, y)
    plt.pause(delay)
```

184

```
12
13  plt.show()
```

Listing 27.1: Real-Time Plotting in Python

We get the following plot as shown in Figure 27.1.
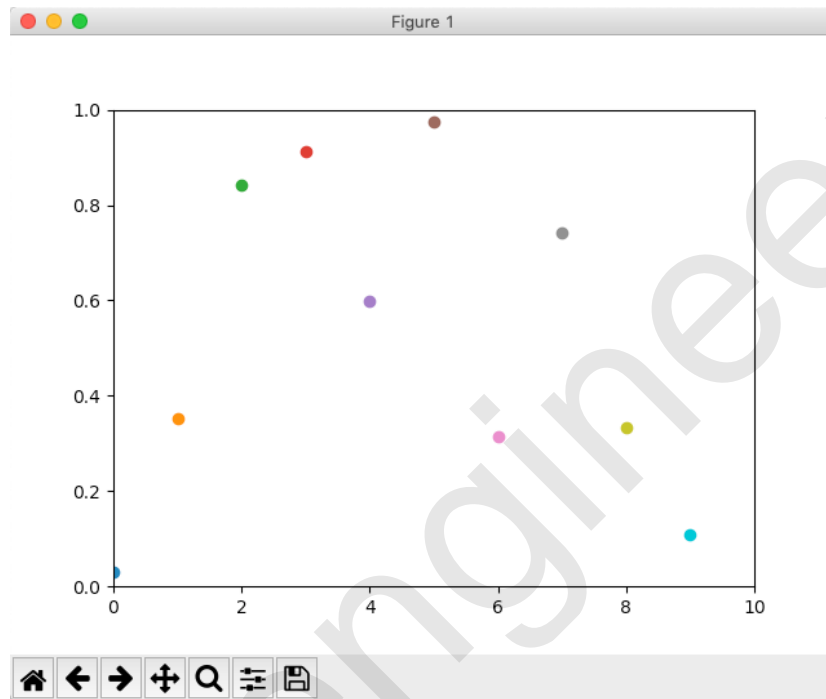
Figure 27.1: Real-Time Plotting with Python

You cannot see the the actual behavior of the plot by watching Figure 27.1, so you need to run the Python program yourself.

If you run the code you see the plot is updated with a new value every second as specified in the code.

[End of Example]

Note! If you use Anaconda and Spyder, you typically need to change the the settings for how graphics are are displayed in Spyder.

Select Preferences from the menu, then IPython console in the list of categories on the left, then the tab Graphics at the top, and change the Graphics back-end from Inline to e.g. Automatic or Qt. See Figure 27.2.

Figure 27.2: Change how Graphics are displayed in the Spyder Editor

27.3 Real-Time Plotting with Animation

For more advanced Real-Time plots we should use the animation module in the matplotlib library (matplotlib.animation).

To create a real-time plot, we need to use the animation module in matplotlib. We set up the figure and axes in the usual way, but we draw directly to the axes, ax, when we want to create a new frame in the animation.

We need to use the FuncAnimation function:

```
ani = animation.FuncAnimation(fig, animate, fargs=(xs, ys),
    interval=1000)
```
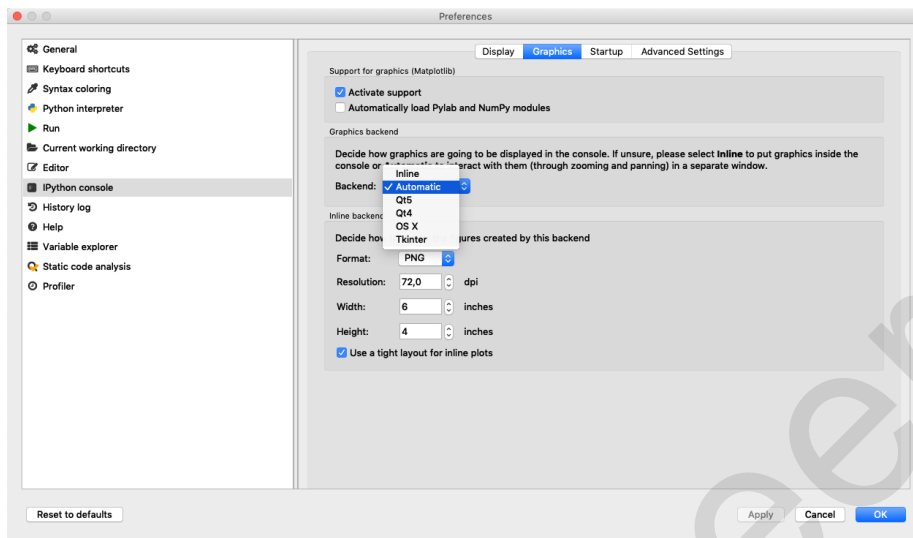
FuncAnimation is a special function within the animation module that lets us automate updating the graph. We pass the FuncAnimation() a handle to the figure we want to draw, fig, as well as the name of a function that should be called at regular intervals. We called this function animate() and is defined just above our FuncAnimation() call.

Still in the FuncAnimation() parameters, we set fargs, which are the arguments we want to pass to our animate function (since we are not calling animate() directly from within our own code). Then, we set interval, which is how long we should wait between calls to animate() (in milliseconds).

Note: As an argument to FuncAnimation, notice that animate does not have any parentheses. This is passing a reference to the function and not the result of that function. If you accidentally add parentheses to animate here, animate will be called immediately (only once), and you'll likely get an error

Example 27.3.1. Real-Time Plotting with Animation

Below you find the Python Code for a basic example where we use the animation module in matplotlib.

In the example we update the plot every seconds by setting the interval=1000ms as an input argument to the FuncAnaimation function.

```python
import datetime as dt
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# Create figure for plotting
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
xs = []
ys = []


# This function is called periodically from FuncAnimation
def animate(i, xs, ys):

    temp_c = round(np.random.random(), 2)

    # Add x and y to lists
    xs.append(dt.datetime.now().strftime('%H:%M:%S.%f'))
    ys.append(temp_c)

    # Limit x and y lists to 20 items
    xs = xs[-20:]
    ys = ys[-20:]

    # Draw x and y lists
    ax.clear()
    ax.plot(xs, ys)

    # Format plot
    plt.xticks(rotation=45, ha='right')
    plt.subplots_adjust(bottom=0.30)
    plt.title('Temperature Data')
    plt.ylabel('Temperature (deg C)')

# Set up plot to call animate() function periodically
ani = animation.FuncAnimation(fig, animate, fargs=(xs, ys),
    interval=1000)
plt.show()
```

Listing 27.2: Real-Time Plotting with Animation

Figure 27.3 shows the final plot for this example. You cannot see the the actual behavior of the plot by watching Figure 27.3, so you need to run the Python program yourself.
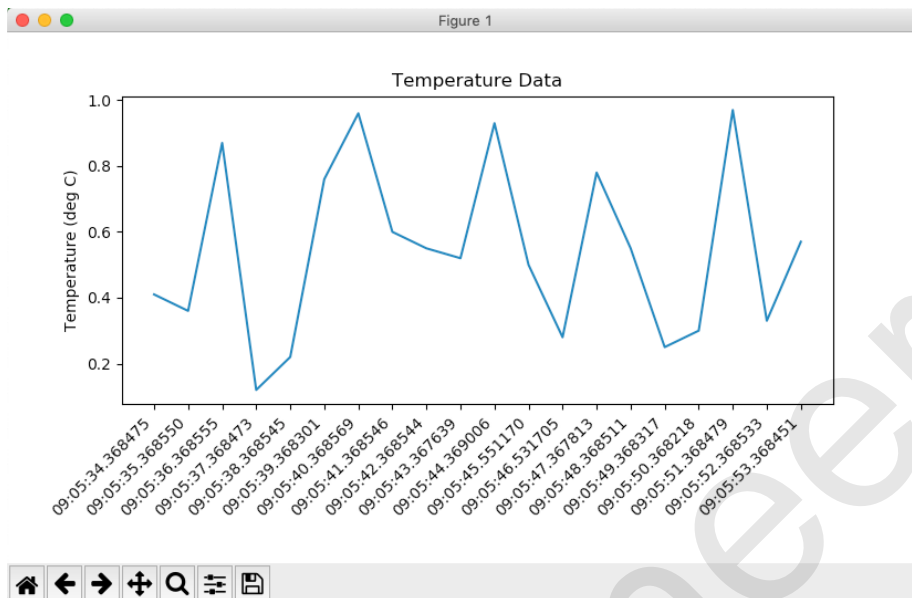
[End of Example]

187

Figure 27.3: Real-Time Plotting with Animation

27.3.1 Speeding Up the Plot Animation

Clearing a graph and redrawing everything can be a time-consuming process in terms of computer time. To remedy that, we are going to use a trick known as blitting.

Blitting is an old computer graphics technique where several graphical bitmaps are combined into one. This way, only one needed to be updated at a time, saving the computer from having to redraw the whole scene every time.
Matplotlib allows us to enable blitting in FuncAnimation, but it means we need to re-write how some of the animate() function works. To reap the true benefits of blitting, we need to set a static background, which means the axes can't scale and we can't show moving timestamps anymore. This means that you have to take the good with the bad. So you have to choose whats most important for you un your simulations.

Example 27.3.2. Real-Time Plotting with Animation with improved Performance

Python Code:

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# Parameters
x_len = 200          # Number of points to display
y_range = [0, 20]  # Range of possible Y values to display
```

```
9  # Create figure for plotting
10 fig = plt.figure()
11 ax = fig.add_subplot(1, 1, 1)
12 xs = list(range(0, 200))
13 ys = [0] * x_len
14 ax.set_ylim(y_range)
15
16
17 # Create a blank line. We will update the line in animate
18 line, = ax.plot(xs, ys)
19
20 # Add labels
21 plt.title('Temperature Data')
22 plt.xlabel('Samples')
23 plt.ylabel('Temperature (deg C)')
24
25 # This function is called periodically from FuncAnimation
26 def animate(i, ys):
27
28     rand_val = np.random.random()*20 #Generate Random Values
        between 0 and 20
29
30     temp_c = round(rand_val, 2)
31
32     #print (temp_c)
33
34     # Add y to list
35     ys.append(temp_c)
36
37     # Limit y list to set number of items
38     ys = ys[-x_len:]
39
40     # Update line with new Y values
41     line.set_ydata(ys)
42
43     return line,
44
45 # Set up plot to call animate() function periodically
46 ani = animation.FuncAnimation(fig,
47     animate,
48     fargs=(ys,),
49     interval=100,
50     blit=True)
51 plt.show()
```

Listing 27.3: Real-Time Plotting with Animation

Figure 27.4 shows the final plot for this example. You cannot see the the actual behavior of the plot by watching Figure 27.4, so you need to run the Python program yourself.

[End of Example]

For more information about Matplotlib:animations:

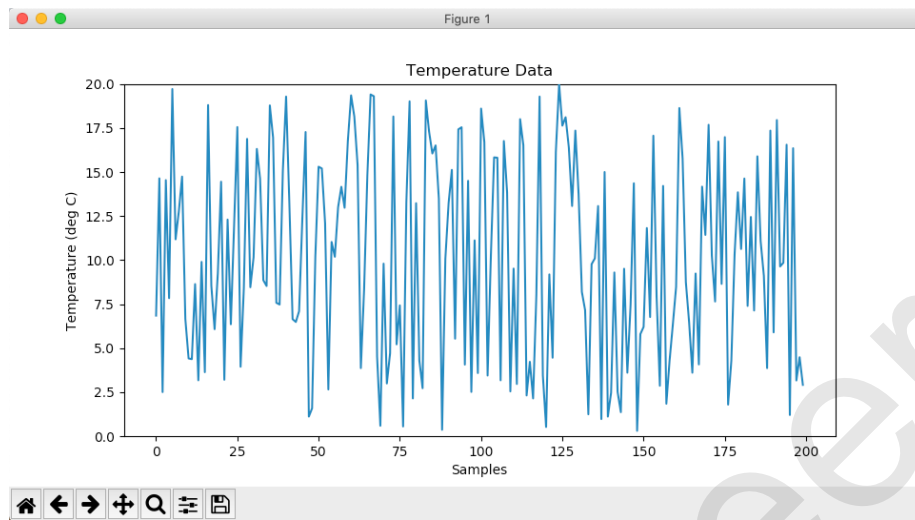https://scipy-cookbook.readthedocs.io/items/Matplotlib$_A$nimations.html

189

Figure 27.4: Real-Time Plotting with Animation

Other resources:

https://learn.sparkfun.com/tutorials/graph-sensor-data-with-python-and-matplotlib/allplot-sensor-data

https://stackoverflow.com/questions/11874767/how-do-i-plot-in-real-time-in-a-while-loop-using-matplotlib

Chapter 28

Data Acquisition (DAQ) with Python

Python is probably best suited for "ad-hoc" numerical calculations, analysis, simulations, etc., but can be used for many other purposes, even if other programming languages are better suited.

28.1 Introduction to DAQ

To read sensor data you typically need a DAQ (Data Acquisition) device connected to you PC or, e.g, a Raspberry Pi device. In all cases you will typically need to install a driver from the vendor of the DAQ device or the sensor you are using.

A DAQ System consists of 4 parts:

- Physical input/output signals, sensors
- DAQ device/hardware
- Driver software
- Your software application (Application software) - in this case your Python application

Figure 28.1 shows the different steps involved in a DAQ system.

Here you find more information, resources, videos and examples regarding DAQ:
https://www.halvorsen.blog/documents/technology/daq/

28.2 Data Acquisition using NI DAQ Devices

Here we will show how we can use Python to retrieve data from the physical world using a DAQ device or an I/O module.
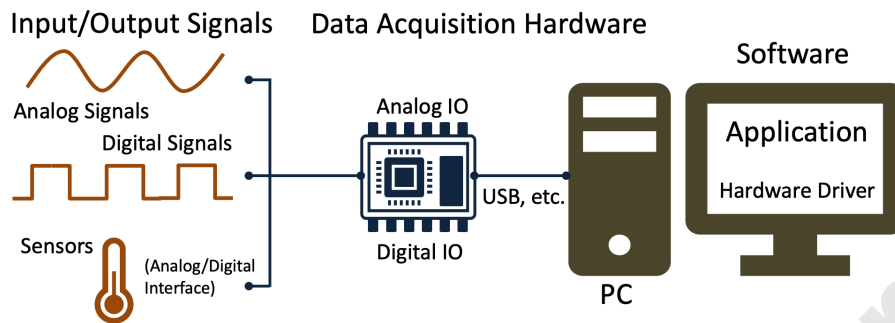
Figure 28.1: Data Acquisition (DAQ) System

We will use a DAQ device from National Instruments (NI).

Web:
http://www.ni.com/

DAQ hardware: WE will use a NI-USB-600x DAQ device from National Instruments, such as:

- NI-USB-6001

- NI-USB-6008

- NI-USB-6009

They are almost identical and the prices is not so bad either.

USB-6008:
http://www.ni.com/en-no/support/model.usb-6008.html

Figure 28.2 shows the USB-6008 DAQ device from NI.

Streaming Data from NI Data Acquisition (DAQmx) Devices into Python:
https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z000000P8o0SAC

We assume we want to do the following: - We have a USB DAQ system from National Instruments - We want to stream data from my hardware into Python to do data processing - We would like to log data to a file on the hard disk

The best way to do this is to use the NI-DAQmx Python API provided by National Instruments (nidaqmx). The **NI-DAQmx Python API** is hosted on GitHub.

The nidaqmx Python package is a wrapper around the NI-DAQmx C API using the ctypes Python library, and only supports the Windows operating system.

Figure 28.2: USB-6008

A Python API for interacting with NI-DAQmx (GitHub):
https://github.com/ni/nidaqmx-python

For more information about NI DAQ:
ni.com/daq

For more information about Python Resources for NI Hardware and Software:
ni.com/python

Another option is to use the **PyDAQmx** Python package.
This package allows users to use data acquisition hardware from National Instrument with python. It makes an interface between the NIDAQmx driver and python. It currently works only on Windows. The package is not an open source driver from NI acquisition hardware. You first need to install the driver provided by NI.

Web:
https://pypi.org/project/PyDAQmx/

https://pythonhosted.org/PyDAQmx/

28.2.1   NI-DAQmx

NI-DAQmx is the software you use to communicate with and control your NI data acquisition (DAQ) device.

NI-DAQmx supports only the Windows operating system.

You can download NI-DAQmx from this location:

https://www.ni.com/download

28.2.2　Measurement　Automation Explorer (MAX)

Measurement　Automation Explorer (MAX) is a software you can use it to configure and test the DAQ device before you use it in Python (or other programming languages).

MAX is included with NI-DAQmx software.

Figure 28.3 shows Measurement　Automation Explorer (MAX).
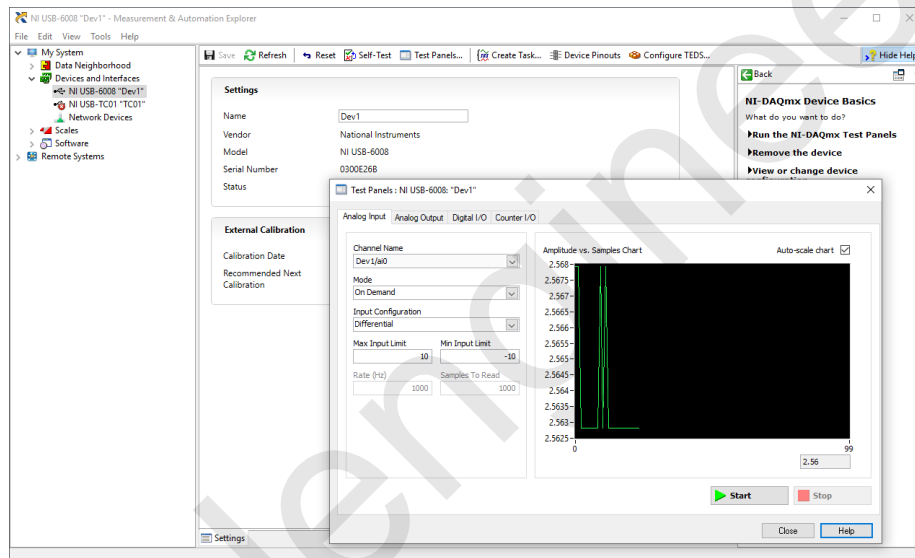
Figure 28.3: Measurement　Automation Explorer(MAX)

With MAX you can make sure your DAQ device works as expected before you start using it in your Python program. You can use the Test Panels to test your analog and digital inputs and outputs channels.

You can also change name of the unit, which you need to use in your Python code.

28.3　NI-DAQmx Python API

In this section we will use the NI-DAQmx Python API provided by National Instruments (nidaqmx). The **NI-DAQmx Python API** is hosted on GitHub.

The nidaqmx Python package is a wrapper around the NI-DAQmx C API using the ctypes Python library, and only supports the Windows operating system.

A Python API for interacting with NI-DAQmx (GitHub):
https://github.com/ni/nidaqmx-python

Other resources:
Control NI DAQ Device with Python and NI DAQmx:
https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019Pf1SAE

Below 4 basic examples will be provided:

- Analog Write using NI DAQ Device

- Analog Read using NI DAQ Device

- Digital Write using NI DAQ Device

- Digital Read using NI DAQ Device

You can easily extend this examples to make them suit your needs. Typically you need to include a while loop where you write and/or read from the DAQ device inside the loop, e.g. read values from one or more sensors that are connected to the DAQ device, you may want to create a control system reading the process value and then later write the calculated control signal (e.g. using a PID controller) back to the DAQ device and the process.

28.3.1 Analog Write

Example 28.3.1. Analog Write using NI DAQ Device

Python code:

```python
import nidaqmx

with nidaqmx.Task() as task:
    task.ao_channels.add_ao_voltage_chan('Dev1/ao0','mychannel'
        ,0,5)

    value = 3
    task.start
    task.write(value)
    task.stop
```

Listing 28.1: Analog Write using NI DAQ Device

Note! The USB-6008 can only output a voltage signal between 0 and 5V.

[End of Example]

195

28.3.2 Analog Read

Example 28.3.2. Analog Read using NI DAQ Device

Python code:

```python
import nidaqmx

with nidaqmx.Task() as task:
    task.ai_channels.add_ai_voltage_chan("Dev1/ai1")

    value = task.read()
    print(value)
    task.stop
```

Listing 28.2: Analog Read using NI DAQ Device

[End of Example]

Example 28.3.3. Analog Read with RSE

Python code:

```python
import nidaqmx

from nidaqmx.constants import (
    TerminalConfiguration)

with nidaqmx.Task() as task:
    task.ai_channels.add_ai_voltage_chan("Dev1/ai0",
    terminal_config=TerminalConfiguration.RSE)

    value = task.read()
    print(value)
    task.stop
```

Listing 28.3: Analog Read with RSE

[End of Example]

Example 28.3.4. Analog Read with Differential

Python code:

```python
import nidaqmx

from nidaqmx.constants import (
    TerminalConfiguration)

with nidaqmx.Task() as task:
    task.ai_channels.add_ai_voltage_chan("Dev1/ai0",
    terminal_config=TerminalConfiguration.DIFFERENTIAL)

    value = task.read()
    print(value)
    task.stop
```

Listing 28.4: Analog Read with Differential

[End of Example]

28.3.3 Digital Write

Example 28.3.5. Digital Write using NI DAQ Device

Python code:

```python
import nidaqmx

with nidaqmx.Task() as task:
    task.do_channels.add_do_chan("Dev1/port0/line0")

    value = True
    task.start
    task.write(value)
    task.stop
```

Listing 28.5: Digital Write using NI DAQ Device

[End of Example]

28.3.4 Digital Read

Example 28.3.6. Digital Read using NI DAQ Device

Python code:

```python
import nidaqmx

with nidaqmx.Task() as task:
    task.di_channels.add_di_chan("Dev1/port0/line0")

    task.start
    value = task.read()
    print(value)
    task.stop
```

Listing 28.6: Digital Read using NI DAQ Device

[End of Example]

You should use the "nidaqmx.stream_readers" and nidaqmx.stream_writers classes to increase the performance of your application, which accept pre-allocated NumPy arrays.

https://nidaqmx-python.readthedocs.io/en/latest/stream_readers.html#module-nidaqmx.stream_readers

https://nidaqmx-python.readthedocs.io/en/latest/stream_writers.html#module-nidaqmx.stream_writers

28.4 Controlling LEDs

In this section we will see how we can control a LED from Python.

We will need the following equipment:

- PC with Python
- DAQ device
- Breadboard
- LED
- Resistor (e.g., 270ohm)
- Wires for connecting the components and create the circuit

Figure 28.4 shows an overview of LEDs.
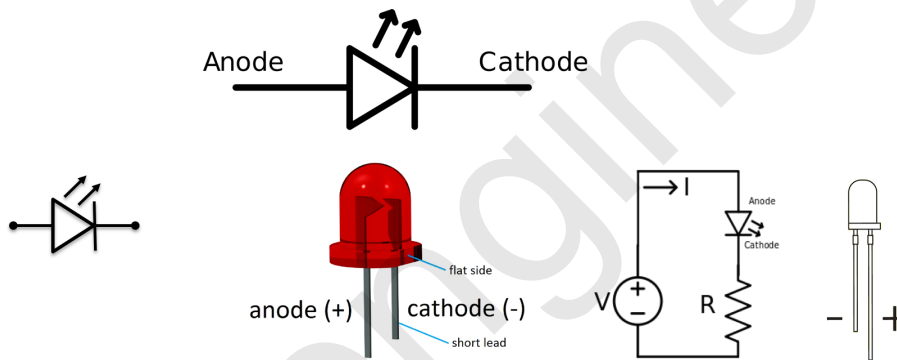
Figure 28.4: LED Overview

A breadboard is used to wire electric components together. Figure 28.5 shows how you should wire a LED using a Breadboard.

Figure 28.6 shows how you wire the LED and connect it to the DAQ device.

Python code for turning on the LED

Example 28.4.1. Controlling a LED from Python

Basic Python code:

```python
import nidaqmx

with nidaqmx.Task() as task:
    task.do_channels.add_do_chan("Dev1/port0/line0")

    value = True
    task.start
    task.write(value)
    task.stop
```

Listing 28.7: Turn on a LED using Python

Figure 28.5: How to wire a LED on a Breadboard

Figure 28.6: Wire the LED and connect to the DAQ device

In this basic example we just turn on the LED.

Below you see an example where we turn the LED on and off inside a loop.

Python code:

```python
import nidaqmx
import time


with nidaqmx.Task() as task:
    task.do_channels.add_do_chan("Dev1/port0/line0")

    value = True
    task.start

    i = 1
    while i < 10:
```
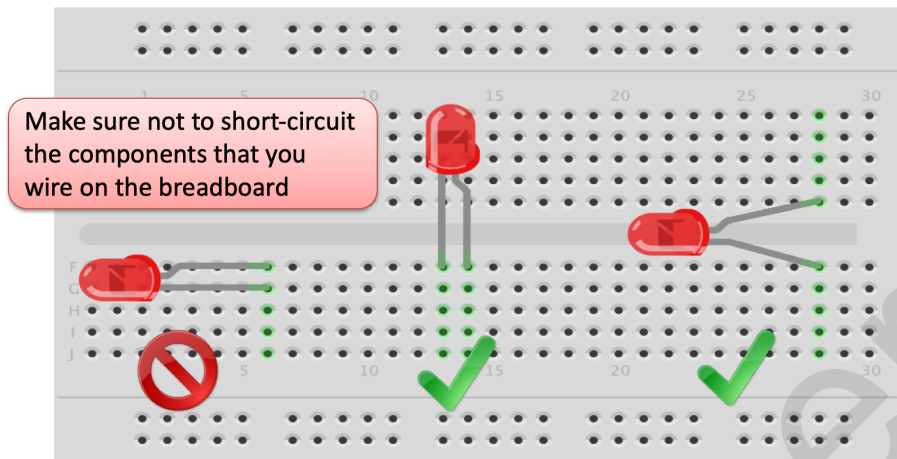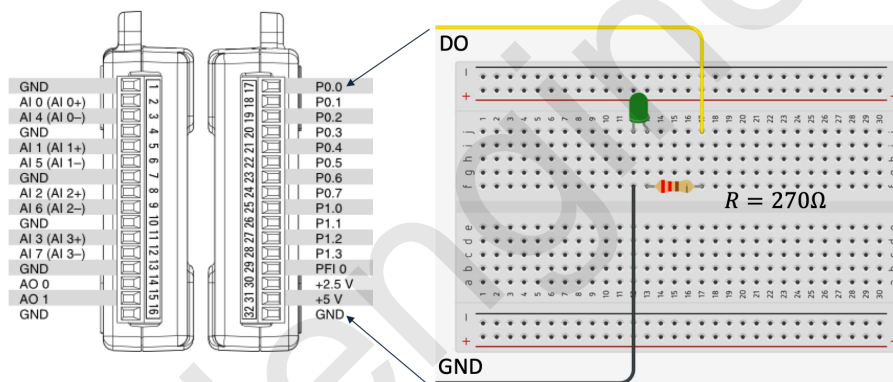
199

```
14          task.write(value)
15          time.sleep(1)
16          value = not value
17          task.write(value)
18          i = i+1
19
20      task.stop
```

Listing 28.8: Controlling a LED using Python

[End of Example]

28.5 Read Data from Temperature Sensors

In this section we will use Python to read values from a temperature sensor. We will also use Python to plot real-time data from the sensor.

28.5.1 Read Data from TMP36 Temperature Sensor

TMP36 is a small, low-cost temperature sensor and cost about \$1 (you can buy it "everywhere").

We will need the following equipment:

- PC with Python
- DAQ device
- Breadboard
- TMP36 Temerature Sensor
- Wires for connecting the components and create the circuit

Figure 28.7 shows the TMP36 sensor.

We connect the TMP36 to LabVIEW using a USB DAQ Device from National Instruments, e.g., USB-6001, USB-6008 or similar. I have used a breadboard for the wiring.
Figure 28.8 show how we can wire the TMP36 together with the USB-6008 DAQ device.

Figure 28.9 shows the TMP3x Datasheet.

We need to convert form Voltage (V) to degrees Celsius.

From the Datasheet (Figure 28.9) we have:

$$(x_1, y_1) = (0.75V, 25°) \tag{28.1}$$

Figure 28.7: TMP36 Temperature Sensor

$$(x_2, y_2) = (1V, 50°) \tag{28.2}$$

From the Datasheet (Figure 28.9) we see that there is a linear relationship between Voltage and degrees Celsius (28.3):

$$y = ax + b \tag{28.3}$$

We can find a and b using the following known formula (28.4):

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) \tag{28.4}$$

By putting (28.1) and (28.2) into (28.4) we get:

$$y - 25 = \frac{50 - 25}{1 - 0.75}(x - 0.75) \tag{28.5}$$

Then we get the following formula we can implement in our Python program:

$$y = 100x - 50 \tag{28.6}$$

Example 28.5.1. Read TMP36 Temperature Data
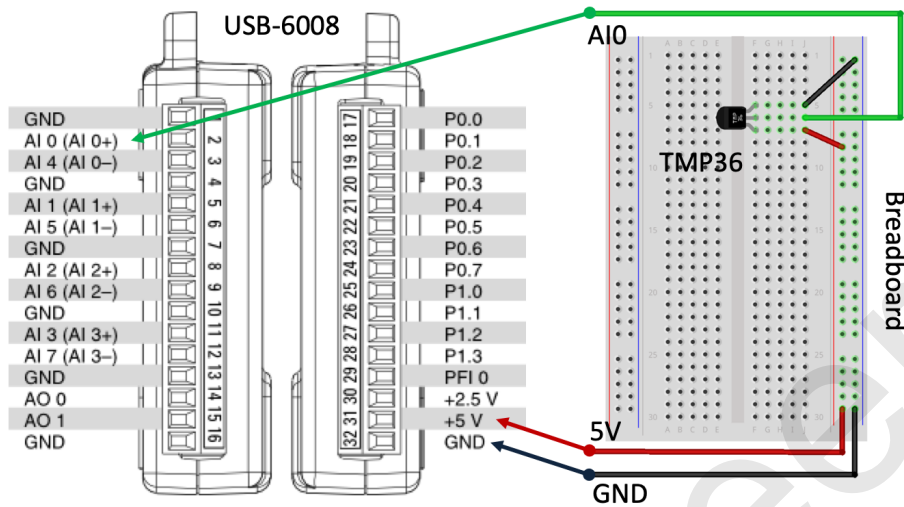
Python code:

201

Figure 28.8: TMP36 tmp36 Wiring

```python
1  import nidaqmx
2  import time
3
4  from nidaqmx.constants import (
5      TerminalConfiguration)
6
7
8  with nidaqmx.Task() as task:
9      task.ai_channels.add_ai_voltage_chan("Dev1/ai0",
       terminal_config=TerminalConfiguration.RSE)
10
11     i = 0
12     while i < 10:
13
14         voltage = task.read()
15
16         degreesC = 100*voltage - 50
17
18         print("Sample:", i)
19         print("Voltage Value:", round(voltage,2))
20         print("Celsius Value:", round(degreesC,1))
21         print("\n")
22         time.sleep(1)
23         i = i+1
24
25     task.stop
```

Listing 28.9: Read TMP36 Temperature Data

In the example an ordinary while loop in combination with the sleep() function have been used to read one new value each second.

[End of Example]

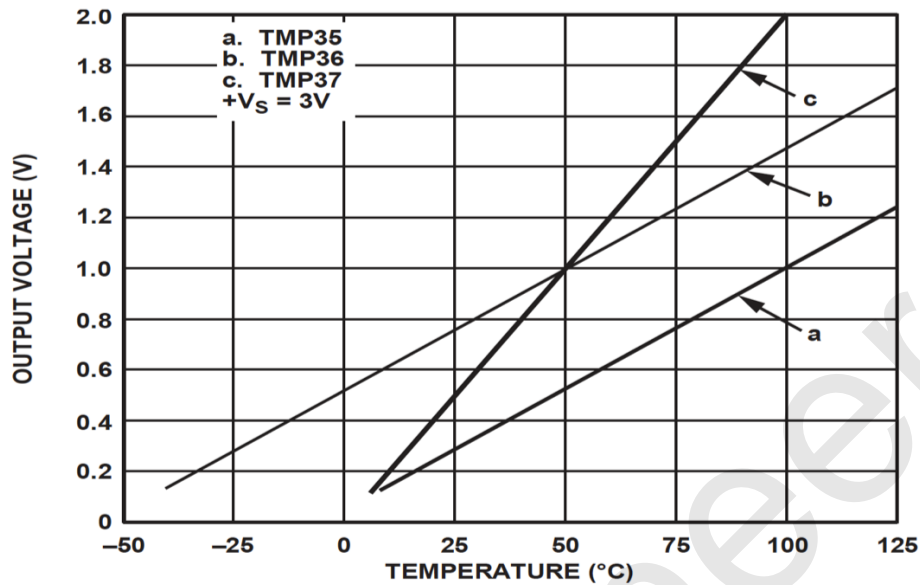**Example 28.5.2.** Real-Time Plotting of Temperature Data

Figure 28.9: TMP3x Datasheet

In this example we will plot the data from the sensor using the Real-time plotting examples shown in Chapter 27.

We want to present the value from the sensor in degrees Celsius:

1. Read Signal from DAQ Device (0-5V)

2. Convert to degrees Celsius using information from the Datasheet

3. Show/Plot Values from the Sensor

The Python code becomes as follows:

```python
import nidaqmx
import time
import datetime as dt
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

from nidaqmx.constants import (
    TerminalConfiguration)


# Create figure for plotting
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
xs = []
ys = []

# Initialize DAQ device
task = nidaqmx.Task()
task.ai_channels.add_ai_voltage_chan("Dev1/ai0", terminal_config=
    TerminalConfiguration.RSE)
```

203

```
21  task.start
22
23
24  # This function is called periodically from FuncAnimation
25  def readdaq(i, xs, ys):
26
27      #Read Value from DAQ device
28      voltage = task.read()
29
30      #Convert Voltage to degrees Celsius
31      degreesC = 100*voltage - 50
32      temp_c = round(degreesC, 1)
33      print("Celsius Value:", temp_c)
34
35      # Add x and y to lists
36      xs.append(dt.datetime.now().strftime('%H:%M:%S.%f'))
37      ys.append(temp_c)
38
39      # Limit x and y lists to 20 items
40      xs = xs[-20:]
41      ys = ys[-20:]
42
43      # Draw x and y lists
44      ax.clear()
45      ax.plot(xs, ys)
46
47      # Format plot
48      plt.xticks(rotation=45, ha='right')
49      plt.subplots_adjust(bottom=0.30)
50      plt.title('Temperature Data')
51      plt.ylabel('Temperature (deg C)')
52
53
54  # Set up plot to call readdaq() function periodically
55  ani = animation.FuncAnimation(fig, readdaq, fargs=(xs, ys),
        interval=1000)
56  plt.show()
57  task.stop
```

Listing 28.10: Real-Time Plotting of Temperature Data

[End of Example]

28.5.2 Read Data from Thermistor

A Thermistor is an electronic component that changes resistance to temperature, a so-called Resistance Temperature Detectors (RTD). It is often used as a temperature sensor.

Example 28.5.3. Read Thermistor Temperature Data

We will need the following equipment:

- PC with Python

- DAQ device

204

- Breadboard

- 10kohm Thermistor

- 10kohm Resistor

- Wires for connecting the components and create the circuit

Our Thermistor is a so-called NTC (Negative Temperature Coefficient). In a NTC Thermistor, resistance decreases as the temperature rises.

There is an non-linear relationship between resistance and excitement. To find the temperature we can use the following equation (Steinhart-Hart equation):

$$\frac{1}{T_K} = A + B\ln(R) + C(\ln(R))^3 \tag{28.7}$$

where A, B and C are constants with the following values: $A = 0.001129148, B = 0.000234125, C = 8.76741E - 08$

T_K is the temperature in Kelvin.

We want to solve the equation regarding the Temperature:

$$T_K = \frac{1}{A + B\ln(R) + C(\ln(R))^3} \tag{28.8}$$

The Temperature in degrees Celsius will then be:

$$T_C = T_K - 273.15 \tag{28.9}$$

Wiring...
Figure 28.10 shows how we wire the components together.
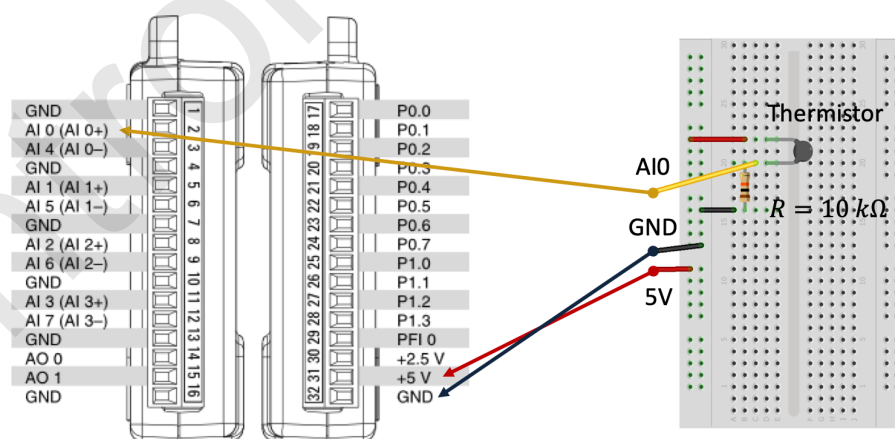

Figure 28.10: Thermistor Wiring

205
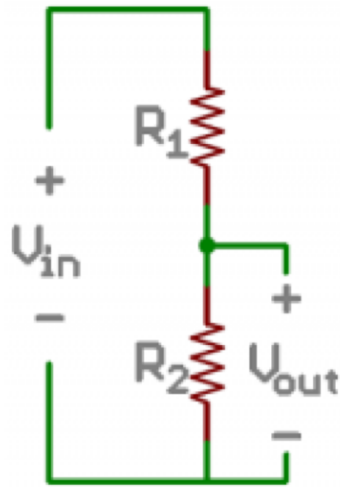
Figure 28.11: Voltage Divider

The wiring is called a "Voltage divider".

Figure 28.11 shows a general Voltage Divider.

A general Voltage Divider has the following equation:

$$V_{out} = V_{in} \frac{R_2}{R_1 + R_2} \tag{28.10}$$

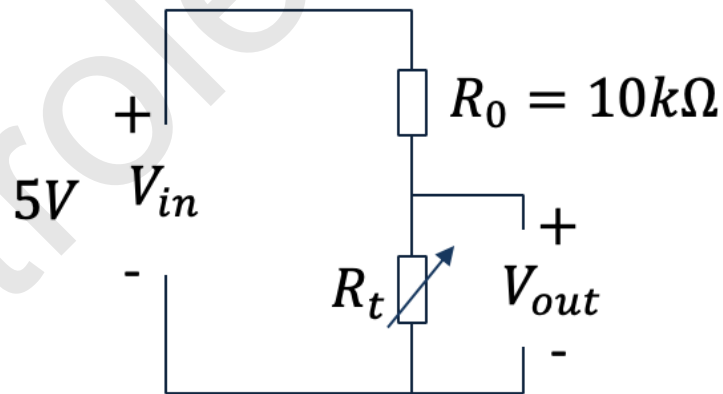The Voltage Divider for our system becomes as shown in Figure 28.12.

Figure 28.12: Voltage Divider for our System

We then have the following equation:

$$V_{out} = V_{in} \frac{R_t}{R_0 + R_t} \tag{28.11}$$

206

where R_t is our 10kohm Thermistor and R_0 is an ordinary $10kohm$ Resistor.

V_{in} in our case will be +5V which we get from the USB-6008 DAQ device as shown in the wiring diagram.

V_{out} is the voltage we read using the DAQ device.

Since we need to find the Resistance R_t, which is used in the Steinhart-Hart equation, we reformulate the formula:

$$R_t = \frac{V_{out} R_0}{V_{in} - V_{out}} \tag{28.12}$$

We har no ready to start making the Python program for this example.

The program include the following necessary steps:

1. We wire the circuit on the Breadboard and connect it to the DAQ device

2. We measure V_{out} using the DAQ

3. We calculate R_t using the Voltage Divider equation
   $R_t = \frac{V_{out} R_0}{V_{in} - V_{out}}$

4. We use Steinhart-Hart equation for finding the Temperature
   $T_K = \frac{1}{A + B \ln(R_t) + C(\ln(R_t))^3}$

5. Finally we convert to degrees Celsius
   $T_C = T_K - 273.15$

The Python code then becomes:

```
1  import nidaqmx
2  import numpy as np
3  import time
4
5  from nidaqmx.constants import (
6      TerminalConfiguration)
7
8
9  Vin = 5
10 Ro = 10000 # %10k Resistor
11
12
13 with nidaqmx.Task() as task:
14     task.ai_channels.add_ai_voltage_chan("Dev1/ai0",
       terminal_config=TerminalConfiguration.RSE)
15
16     i = 0
17     while i < 10:
18
19         Vout = task.read()
20
21         Rt = (Vout*Ro)/(Vin-Vout) # Voltage Divider Equation
22         # Rt=10000; Used for Testing. Setting Rt=10k should give
       TempC=25
23
```

```
24          # Steinhart constants
25          A = 0.001129148
26          B = 0.000234125
27          C = 0.0000000876741
28
29          # Steinhart-Hart Equation
30          TempK = 1 / (A + (B * np.log(Rt)) + (C * pow(np.log(Rt),3))
     )
31
32          # Convert from Kelvin to Celsius
33          TempC = TempK - 273.15
34
35          print("Sample:", i)
36          print("Voltage Value:", round(Vout,2))
37          print("Celsius Value:", round(TempC,1))
38          print("\n")
39          time.sleep(1)
40          i = i+1
41
42      task.stop
```

Listing 28.11: Read Thermistor Temperature Data

[End of Example]

Example 28.5.4. Real-Time Plotting of Thermistor Temperature Data

Python code:

```
1 See previous examples
```

Listing 28.12: Real-Time Plotting of Thermistor Temperature Data

[End of Example]

28.5.3 Read Data NI TC-01 Thermocouple Device

In this chapter several examples have been shown using a DAQ device combined with different sensors and components.

Here some examples will be shown using a preset temperature sensor from National Instruments called NI USB-TC01. This is a USB based temperature without need for any kind of wiring, you just plug it in and make your Python program. Since the NI USB-TC01 is compatible with NI-DAQmx, you can program it in the same way as other DAQ devices from NI.

Figure 28.13 shows the TC-01 Thermocouple Device from NI.

Example 28.5.5. Real-Time Plotting of Thermistor Temperature Data

Python code:

Figure 28.13: TC-01 Thermocouple Device

```python
1  import nidaqmx
2
3  task = nidaqmx.Task()
4
5  task.ai_channels.add_ai_thrmcpl_chan("TC01/ai0")
6
7  task.start()
8
9  value = task.read()
10 print(round(value,1))
11
12 task.stop()
13 task.close()
```

Listing 28.13: TC-01 Thermocouple Python Example

This is just a basic example, which you can easily extend using a while loop or using some kind of plotting, etc..

[End of Example]

28.6 Data Logging

Python has several functions for creating, reading, updating, and deleting files.

209

Part VIII

Control Systems

Chapter 29

Python used for Control Applications

The Python Control Systems Library (python-control) is a Python package that implements basic operations for analysis and design of feedback control systems.

29.1 Python Control Systems Library

The python-control package is a set of python classes and functions that implement common operations for the analysis and design of feedback control systems.

The python-control package makes use of NumPy and SciPy.

A MATLAB compatibility package (control.matlab) is available that provides many of the common functions corresponding to commands available in the MATLAB Control Systems Toolbox.

Python Control Systems Library Homepage:

https://pypi.org/project/control

Python Control Systems Library Documentation:

https://python-control.readthedocs.io

The python-control package may be installed using pip:

```
pip install control
```

29.1.1 Python Control Systems Library Functions

Here are some of the most used functions in the Python Control Systems Library:

Functions for Model Creation and Manipulation:

- tf() - Create a transfer function system
- ss() - Create a state space system
- c2d() - Return a discrete-time system
- tf2ss() - Transform a transfer function to a state space system
- ss2tf() - Transform a state space system to a transfer function.
- series() - Return the series of 2 or more subsystems
- parallel() - Return the parallel of 2 or more subsystems
- feedback() - Return the feedback of system
- pade() - Creates a Pade Aproxomation, which is a Transfer function representation of a Time Delay

Functions for Model Simulations:

- step_response() - Step response of a linear system
- lsim() - Simulate the output of a linear system

Functions for Stability Analysis:

- step_response() - Step response of a linear system
- lsim() - Simulate the output of a linear system
- pole() - Compute system poles
- zero() - Compute system zeros
- pzmap() - Plot a pole/zero map for a linear system
- margin() - Calculate gain and phase margins and associated crossover frequencies
- stability_margins() - Calculate stability margins and associated crossover frequencies

Functions for Frequency Response:

- bode_plot() - Create a Bode plot for a system
- mag2db() - Convert a magnitude to decibels (dB)
- db2mag() - Convert a gain in decibels (dB) to a magnitude

There are lots of other functions. They are documented in the Python Control Systems Library Documentation:

https://python-control.readthedocs.io

212

29.1.2 MATLAB compatibility module

The control.matlab module contains a number of functions that emulate some of the functionality of MATLAB. The intent of these functions is to provide a simple interface to the python control systems library (python-control) for people who are familiar with the MATLAB Control Systems Toolbox.

Example 29.1.1. MATLAB compatibility module Example

In this example we will use the MATLAB compatibility module for simulating and showing the frequency response for a 2.order mass-spring-damper system.

```python
import matplotlib.pyplot as plt   # MATLAB plotting functions
from control.matlab import *  # MATLAB-like functions

# Parameters defining the system
m = 250.0              # system mass
k = 40.0               # spring constant
b = 60.0               # damping constant

# System matrices
A = [[0, 1.], [-k/m, -b/m]]
B = [[0], [1/m]]
C = [[1., 0]]
sys = ss(A, B, C, 0)

# Step response for the system
plt.figure(1)
yout, T = step(sys)
plt.plot(T.T, yout.T)

# Bode plot for the system
plt.figure(2)
mag, phase, om = bode(sys)
```

Listing 29.1: MATLAB compatibility module Example

As you see, the syntax is very similar to MATLAB.
The code gives the the results shown in Figure 29.1.

[End of Example]

29.2 Control Systems

Figure 29.2 shows a basic control system. Typically you use a PID controller to control the processes.

The following topics will be discussed in this part:
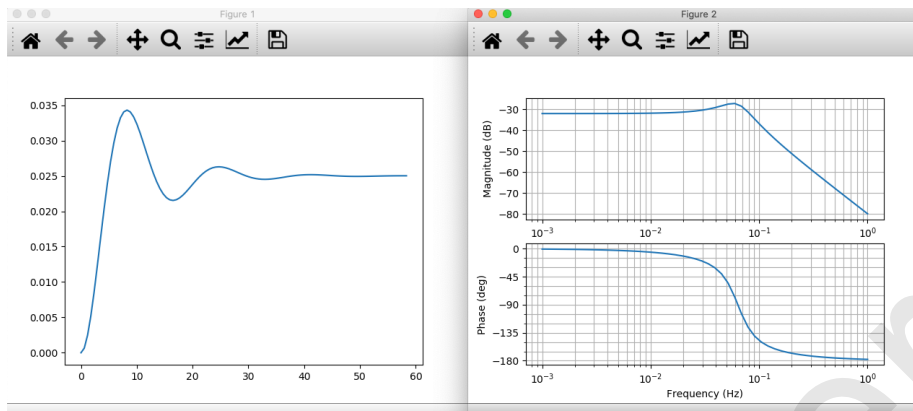
- Transfer Functions
- State-space Models

213

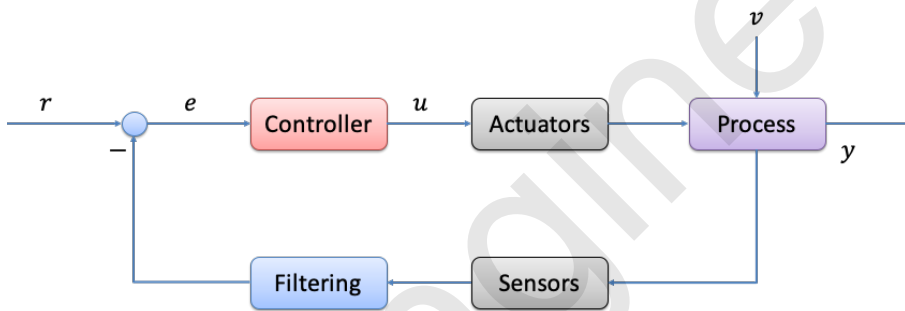Figure 29.1: MATLAB compatibility module Example

Figure 29.2: Control System

- PID Control
- Frequency Response

29.2.1 Transfer functions

Transfer functions are a model form based on the Laplace transform. Transfer functions are very useful in analysis and design of linear dynamic systems.

A general transfer function is on the form:

$$H(s) = \frac{y(s)}{u(s)} \tag{29.1}$$

Example 29.2.1. Basic Transfer Function in Python

Lets define the following transfer function in Python:

$$H(s) = \frac{3}{4s + 1} \tag{29.2}$$

The Python code becomes:

214

```
1 import numpy as np
2 import control
3
4 num = np.array([3])
5 den = np.array([4 , 1])
6
7 H = control.tf(num , den)
8
9 print ('H(s) =', H)
```

Listing 29.2: Basic Transfer Function in Python

[End of Example]

29.2.2   State-space Models

A state-space model is a structured form or representation of a set of differential equations. State-space models are very useful in Control theory and design. The differential equations are converted in matrices and vectors.

A general linear State-space model can then be written on this compact form:

$$\dot{x} = Ax + Bu \tag{29.3}$$

$$y = Cx + Du \tag{29.4}$$

Where A, B, C and D are matrices.

Example 29.2.2. State Space Model

Given the following system):

$$\dot{x_1} = x_2 \tag{29.5}$$

$$2\dot{x_1} = -2x_1 - 6x_2 + 4u_1 + 8u_2 \tag{29.6}$$

$$y = 5x_1 + 6x_2 + 7u_1 \tag{29.7}$$

This gives the following state space model:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{29.8}$$

215

$$y = \begin{bmatrix} 5 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} 7 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{29.9}$$

We can define this state space model in Python:

```python
import matplotlib.pyplot as plt
import control

# System matrices
A = [[0, 1], [-1, -3]]
B = [[0, 0], [2, 4]]
C = [[5, 6]]
D = [[7, 0]]
ssmodel = control.ss(A, B, C, D)
```

Listing 29.3: State space model

[End of Example]

29.3 PID Control

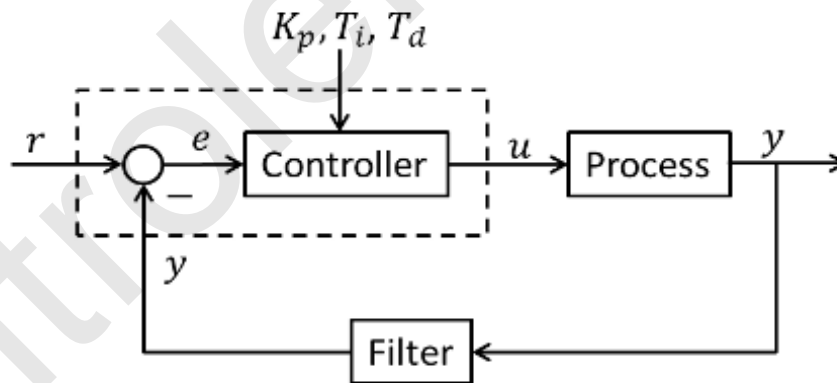Figure 29.3 shows a basic control system. Typically you use a PID controller to control the processes.

Figure 29.3: Control System using PID Controller

The PID Algorithm (29.10):

$$u(t) = K_p e + \frac{K_p}{T_i} \int_0^t e d\tau + K_p T_d \dot{e} \tag{29.10}$$

216

Where u is the controller output and e is the control error:

$$e(t) = r(t) - y(t) \qquad (29.11)$$

Where
r is the Reference Signal or Set-point
y is the Process value, i.e., the Measured value

PID Tuning Parameters:
K_p Proportional Gain
T_i Integral Time [sec]
T_d Derivative Time [sec]

The transfer function for the PID controller is (we use Laplace on 29.10):

$$H_{pid}(s) = \frac{u(s)}{e(s)} = K_p + \frac{K_p}{T_i s} + K_p T_d s \qquad (29.12)$$

Or like this:

$$H_{pid}(s) = \frac{u(s)}{e(s)} = \frac{K_p(T_d T_i s^2 + T_i s + 1)}{T_i s} \qquad (29.13)$$

29.3.1 PI Control

Very often we only need to use the PI Algorithm (29.14):

$$u(t) = K_p e + \frac{K_p}{T_i} \int_0^t e d\tau \qquad (29.14)$$

The transfer function for the PI controller is:

$$H_{pi}(s) = \frac{u(s)}{e(s)} = \frac{K_p(T_i s + 1)}{T_i s} \qquad (29.15)$$

Chapter 30

Transfer Functions

Transfer functions are a model form based on the Laplace transform. Transfer functions are very useful in analysis and design of linear dynamic systems.

A general transfer function is on the form:

$$H(s) = \frac{y(s)}{u(s)} \tag{30.1}$$

Where y is the output and u is the input. A general transfer function can be written on the following general form:

$$H(s) = \frac{numerator(s)}{denominator(s)} \tag{30.2}$$

or to be more specific:

$$H(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + ... + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + ... + a_1 s + a_0} \tag{30.3}$$

The numerators of transfer function models describe the locations of the zeros of the system, while the denominators of transfer function models describe the locations of the poles of the system.

Below we will learn more about some important special cases of this general form, namely the 1.order transfer function and the 2.order transfer function.

We will use the Python Control Systems Library. The Python Control Systems Library (python-control) is a Python package that implements basic operations for analysis and design of feedback control systems.

Python Control Systems Library Documentation:

https://python-control.readthedocs.io

Example 30.0.1. Create Transfer Function in Python

Given the following transfer function:

$$H(s) = \frac{3}{4s + 1} \tag{30.4}$$

The Python code becomes:

```
1 import numpy as np
2 import control
3
4 num = np. array ([3])
5 den = np. array ([4 , 1])
6
7 H = control.tf(num , den)
8
9 print ('H(s) =', H)
```

Listing 30.1: Create Transfer Function in Python

[End of Example]

Example 30.0.2. Create more advanced Transfer Function in Python

Given the following transfer function:

$$H(s) = \frac{3s + 3}{4s^2 + 5s + 6} \tag{30.5}$$

The Python code becomes:

```
1 import numpy as np
2 import control
3
4 num = np. array ([3 , 2])
5 den = np. array ([4 , 5 , 6])
6
7 H = control.tf(num , den)
8
9 print ('H(s) =', H)
```

Listing 30.2: Create Transfer Function in Python

[End of Example]

30.1   1.order Transfer Functions

A 1.order transfer function is given by:

$$H(s) = \frac{x(s)}{u(s)} = \frac{K}{Ts + 1} \tag{30.6}$$

219

Where K is the Gain and T is the Time constant. u is the input signal, while x is the output signal.

In the time domain we get the following differential equation (using Inverse Laplace):

$$\dot{x} = \frac{1}{T}(-x + Ku) \tag{30.7}$$

Note that $\dot{x}$ is the same as $\frac{dx}{dt}$.

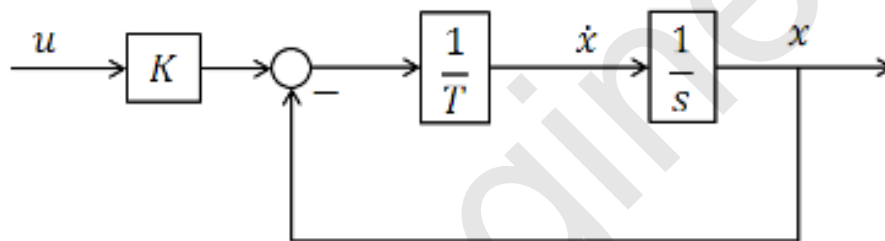We can draw the following block diagram of the system as shown in Figure 30.1.

Figure 30.1: Block Diagram 1.order System

Poles: A 1.order system has a pole $p = -\frac{1}{T}$. In Python you may use the **pole()** function in order to find the poles. You can also use the **pzmap()** function.

Step Response: In Python you may use the **step_response()** function in order to find the step response.

Example 30.1.1. 1.order Transfer Function Python Example

Given the following transfer function:

$$H(s) = \frac{3}{4s + 1} \tag{30.8}$$

The Python code becomes:

```python
import numpy as np
import control

K = 3
T = 4

num = np.array([K])
den = np.array([T, 1])
```

```
9
10  H = control.tf(num , den)
11
12  print ('H(s) =', H)
```

Listing 30.3: Basic Transfer Function in Python

[End of Example]

30.1.1 Step Response

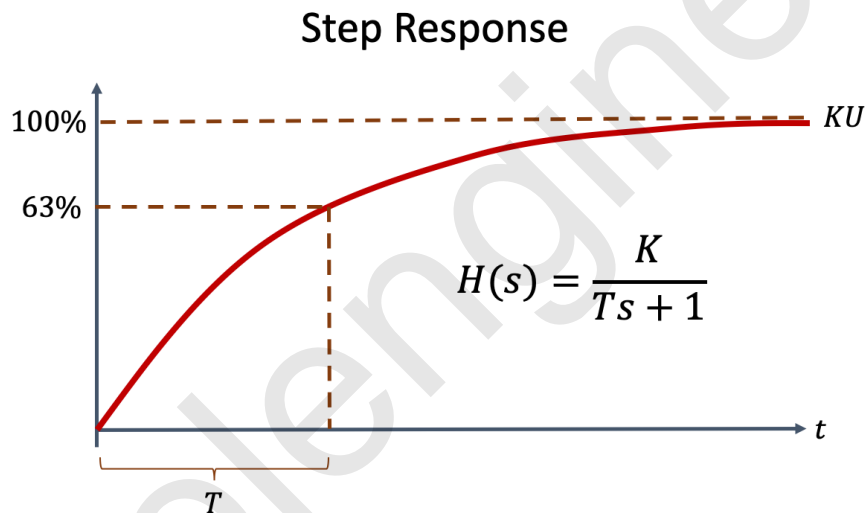A step response of such a transfer function has the characteristics as seen in Figure 30.2.

Figure 30.2: 1.order Process

Given the 1.order transfer function:

$$H(s) = \frac{y(s)}{u(s)} = \frac{K}{Ts + 1} \tag{30.9}$$

Here we will start by finding the mathematical expression for the step response (y(t)):

The Laplace Transformation pair for a unit step is as follows:

$$\frac{1}{s} \Leftrightarrow 1 \tag{30.10}$$

221

Or more general:

$$\frac{U}{s} \Leftrightarrow U \qquad (30.11)$$

Where U is the step size.

The step response of an integrator then becomes:

$$y(s) = H(s)u(s) \qquad (30.12)$$

Where

$$u(s) = \frac{U}{s} \qquad (30.13)$$

This gives:

$$y(s) = H(s)\frac{U}{s} = \frac{K}{Ts+1}\frac{U}{s} \qquad (30.14)$$

We use the following Laplace Transformation pair in order to find y(t):

$$\frac{k}{(Ts+1)s} \Leftrightarrow k(1 - e^{-t/T}) \qquad (30.15)$$

This gives the following:

$$y(t) = KU(1 - e^{-t/T}) \qquad (30.16)$$

In Python we can use the step_response function from the Python Control Systems Library.

Python code:

```
>>> t, y = step_response(sys, t, x0)
```

Where sys is either a transfer function or a state-space model, x0 is the start/initial condition and t is the time vector. Note that t is auto-computed if not given.

Example 30.1.2. Basic Step Response Example

Given the following transfer function:

$$H(s) = \frac{3}{4s + 1} \tag{30.17}$$

The Python code for plotting the step response:

```python
import numpy as np
import matplotlib.pyplot as plt
import control

K = 3
T = 4

num = np.array([K])
den = np.array([T , 1])

H = control.tf(num , den)
print ('H(s) =', H)

t , y = control.step_response(H)
plt.plot(t,y)
```

Listing 30.4: Basic Step Response Example

This gives the the plot shown in Figure 30.3.

You should of course use different functions for creating grid, axis, title, xlabel, ylabel, etc. These are exemplified in other chapters.

[End of Example]

Example 30.1.3. An Alternative Definition of the Transfer Function

Given the same transfer function:

$$H(s) = \frac{3}{4s + 1} \tag{30.18}$$

Here we will use the TransferFunction class to define the Laplace operator s as a constant. This can then be used to create variables that allow algebraic creation of transfer functions

The Python code for defining the transfer function in an alternative way and plotting the step response:
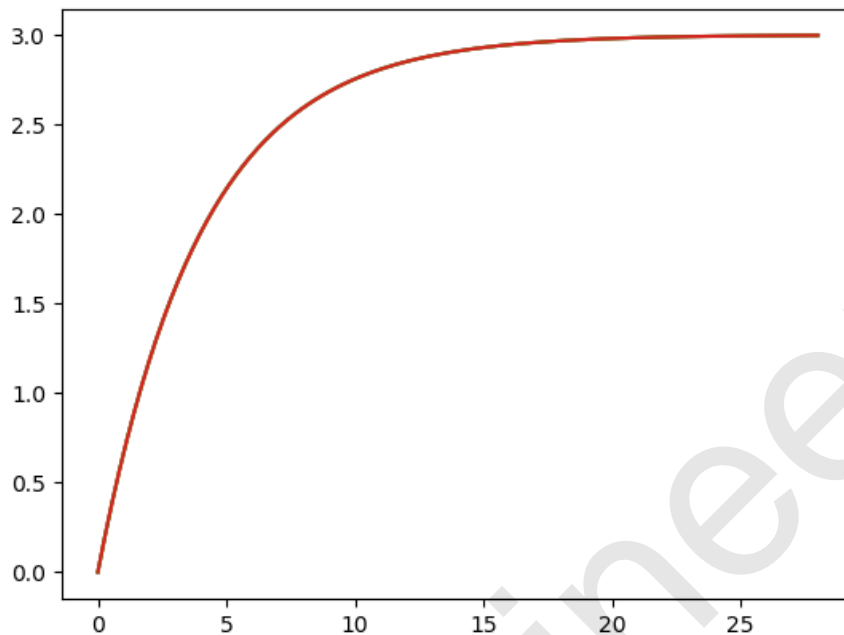
Figure 30.3: Step Response Example

```python
import matplotlib.pyplot as plt
import control

s = control.TransferFunction.s

H = (3)/(4*s + 1)

print ('H(s) =', H)

t, y = control.step_response(H)

plt.plot(t,y)
```

Listing 30.5: Alternative definition of the Transfer Function

This gives the the same plot shown in Figure 30.3.

[End of Example]

30.2   1.order Transfer Functions with Time De-lay

A 1.order transfer function with time-delay may be written as:

$$H(s) = \frac{K}{Ts + 1} e^{-\tau s} \tag{30.19}$$

In the time domain we get the following differential equation (using Inverse Laplace):

$$\dot{x} = \frac{1}{T}(-x + Ku(t - \tau)) \tag{30.20}$$

We can draw the following block diagram of the system (30.4).
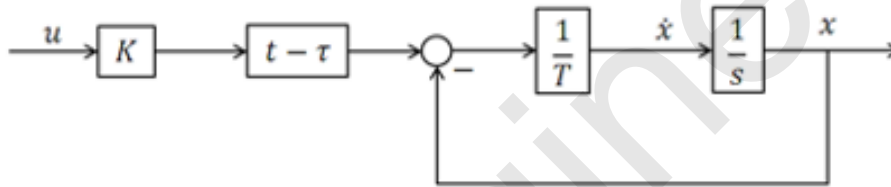

Figure 30.4: Block Diagram 1.order with Time Delay

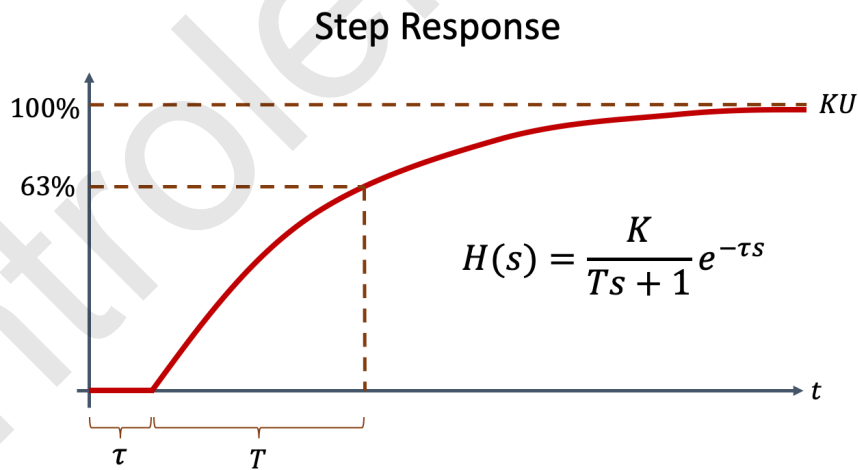Step response for a 1.order process with time delay, see Figure 30.5.

Figure 30.5: 1.order Process with Time delay

Example 30.2.1. 1.order Transfer Function with Time Delay using Pade Approximation

Given the following transfer function with time delay:

$$H(s) = \frac{3}{4s+1}e^{-2s} \tag{30.21}$$

The Python code for plotting the step response:

```python
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import control
4
5  K = 3
6  T = 4
7
8  num = np.array([K])
9  den = np.array([T, 1])
10
11 H1 = control.tf(num, den)
12 print('H1(s) =', H1)
13
14
15 Tau = 2
16 N = 10 # Order of the Approximation
17
18
19 [num_pade, den_pade] = control.pade(Tau, N)
20 Hpade = control.tf(num_pade, den_pade)
21 print('Hpade(s) =', Hpade)
22
23 H = control.series(H1, Hpade)
24 print('H(s) =', H)
25
26 t, y = control.step_response(H)
27
28 plt.plot(t, y)
```

Listing 30.6: Basic Step Response Example

This gives the the plot shown in Figure 30.6.

You should try to change the variable N in the code, which is the order of the Pade approximation.

You should of course also use different functions for creating grid, axis, title, xlabel, ylabel, etc. These are exemplified in other chapters.

[End of Example]

30.3 Integrator Transfer Functions
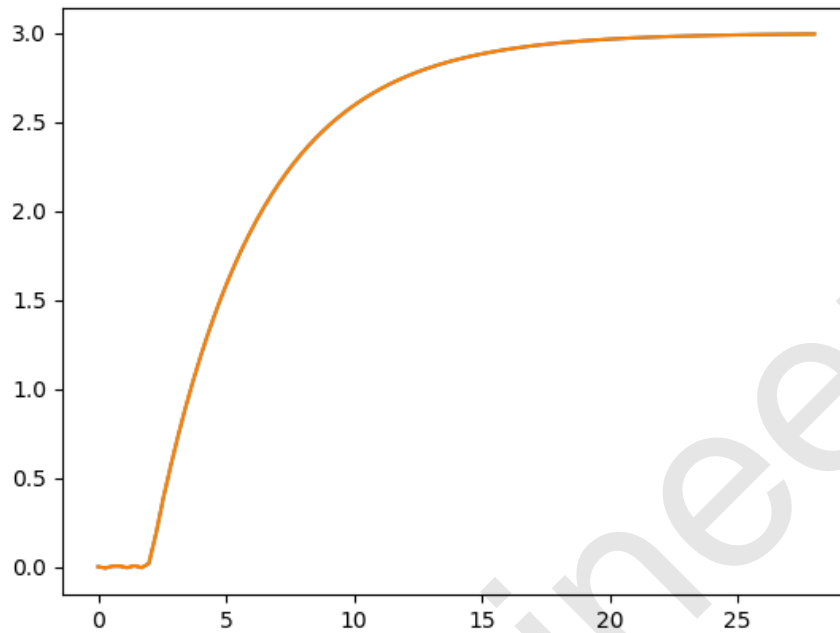
Given the following (30.22):

Figure 30.6: Step Response 1.order with Time Delay using Pade

$$H(s) = \frac{y(s)}{u(s)} = \frac{K}{s} \tag{30.22}$$

Poles: The Integrator has a pole in origo: p=0. In Python you may use the **pole()** function in order to find the poles. You can also use the **pzmap()** function.

Step Response: In Python you may use the **step_response()** function in order to find the step response.

Here we will start by finding the mathematical expression for the step response (y(t)):

The Laplace Transformation pair for a step is as follows:

$$\frac{1}{s} \Leftrightarrow 1 \tag{30.23}$$

The step response of an integrator then becomes:

227

$$y(s) = H(s)u(s) = \frac{K}{s}\frac{U}{s} = KU\frac{1}{s^2} \qquad (30.24)$$

We use the following Laplace Transformation pair in order to find y(t):

$$\frac{1}{s^2} \Leftrightarrow t \qquad (30.25)$$

Then we get the following:

$$y(t) = KUt \qquad (30.26)$$

We see that the step response of the integrator is a Ramp.

Conclusion: A bigger K will give a bigger slope (In Norwegian: "stigningstall") and the integration will go faster. Simulations in Python will also show this.

Example 30.3.1. Integrator

Given the following (30.27):

$$H(s) = \frac{y(s)}{u(s)} = \frac{K}{s} \qquad (30.27)$$

We will find poles, perform step response, etc. in Python for this system.

```python
import numpy as np
import matplotlib.pyplot as plt
import control

K = 3
num = np.array ([K])
den = np.array ([1, 0])
H = control.tf(num, den)
print ('H(s) =', H)

p = control.pole(H)


t, y = control.step_response(H)
plt.plot(t,y)
plt.title('Step Response')
plt.xlabel('t[s]')
plt.ylabel('y(t)')
plt.grid()
plt.axis([0, 7, 0, 22])
plt.show()
```

228

```
22
23  control.pzmap(H)
24  plt.show()
```

Listing 30.7: Integrator

In the simulation we have set $K = 3$. You should explore with other values as well and observe the results.

Instead of using the step_response() function, we can implement and plot (30.26):

```
1   import numpy as np
2   import matplotlib.pyplot as plt
3
4   #Model Parameters
5   K = 3
6
7   # Simulation Parameters
8   U = 1 # A step in the control signal at t=0
9   t = 0
10
11  tstart = 0
12  tstop = 10
13
14  increment = 1
15
16  y = []
17  y = np.zeros(tstop)
18
19  t = np.arange(tstart, tstop, increment)
20
21
22  # Define the Function
23  for k in range(tstop):
24      y[k] = K * U * t[k]
25
26
27  # Plot the Simulation Results
28  plt.plot(t,y)
29  plt.title('Step Response')
30  plt.xlabel('t[s]')
31  plt.ylabel('y(t)')
32  plt.show()
```

Listing 30.8: Integrator from scratch

This should give the same results.

[End of Example]

30.4 2.order Transfer Functions

Given the following (30.31):

229

$$H(s) = \frac{K}{as^2 + bs + c} \tag{30.28}$$

We can also define a 2.order transer function like this (30.29):

$$H(s) = \frac{K}{(\frac{s}{\omega_0})^2 + 2\zeta\frac{s}{\omega_0} + 1} \tag{30.29}$$

Where
K is the gain
ζ is the relative damping factor
$\omega_0[rad/s]$ is the undamped resonance frequency

2.order system - special case:

$$H(s) = \frac{K}{(T_1s + 1)(T_2s + 1)} \tag{30.30}$$

We see that this system can be considered as two 1.order systems in series:

$$H(s) = \frac{K}{(T_1s + 1)}\frac{1}{(T_2s + 1)} = \frac{K}{(T_1s + 1)(T_2s + 1)} \tag{30.31}$$

30.5   Block Diagrams

Python Control Systems Library has functions for manipulating block diagrams and transfer functions.

30.5.1   Serial Block Diagrams

Serial:

Figure 30.7 shows a serial block diagram.

Example 30.5.1. Serial Block Diagram

Given the following transfer functions:

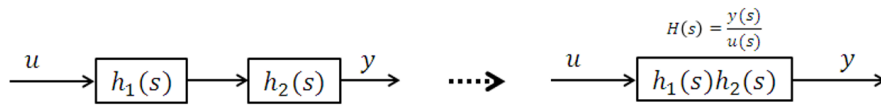$$H(s) = \frac{3}{4s + 1} \tag{30.32}$$

230

Figure 30.7: Serial Block Diagram

$$H(s) = \frac{5}{2s + 1} \qquad (30.33)$$

Python code:

```python
import numpy as np
import matplotlib.pyplot as plt
import control

K = 3
T = 4
num = np.array([K])
den = np.array([T, 1])
H1 = control.tf(num,den)

K = 5
T = 2
num = np.array([K])
den = np.array([T, 1])
H2 = control.tf(num,den)

H = control.series(H1, H2)

print('H(s) =', H)

t, y = control.step_response(H)
plt.plot(t,y)
```

Listing 30.9: Serial Block Diagram Example

[End of Example]

30.5.2   Parallel Block Diagrams

Parallel:

Figure 30.7 shows a parallel block diagram.
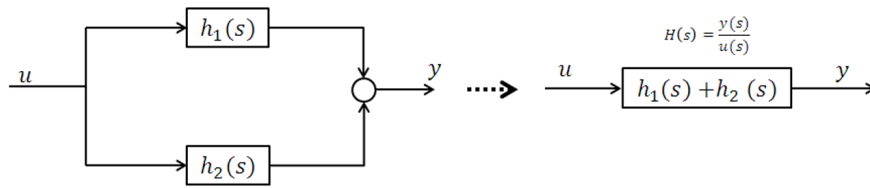
Example 30.5.2. Parallel Block Diagram

231

Figure 30.8: Parallel Block Diagram

Given the following transfer functions:

$$H(s) = \frac{3}{4s+1} \tag{30.34}$$

$$H(s) = \frac{5}{2s+1} \tag{30.35}$$

Python code:

```python
import numpy as np
import matplotlib.pyplot as plt
import control

K = 3
T = 4
num = np.array ([K])
den = np.array ([T , 1])
H1 = control.tf(num,den)

K = 5
T = 2
num = np.array ([K])
den = np.array ([T , 1])
H2 = control.tf(num,den)

H = control.parallel(H1, H2)

print ('H(s) =', H)

t, y = control.step_response(H)
plt.plot(t,y)
```

Listing 30.10: Parallel Block Diagram Example

[End of Example]

232

30.5.3 Feedback Block Diagrams

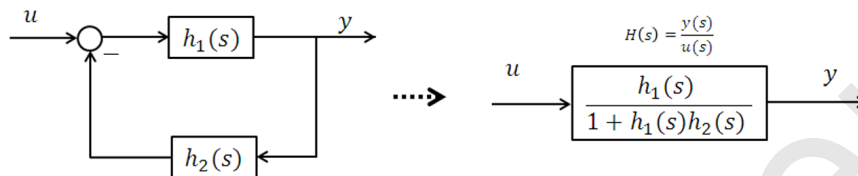Feedback:

Figure 30.7 shows a feedback block diagram.

Figure 30.9: Feedback Block Diagram

Example 30.5.3. Feedback Block Diagram

Given the following transfer functions:

$$H(s) = \frac{3}{4s + 1} \tag{30.36}$$

$$H(s) = \frac{5}{2s + 1} \tag{30.37}$$

Python code:

```python
import numpy as np
import matplotlib.pyplot as plt
import control

K = 3
T = 4
num = np. array ([K])
den = np. array ([T , 1])
H1 = control.tf(num,den)

K = 5
T = 2
num = np. array ([K])
den = np. array ([T , 1])
H2 = control.tf(num,den)

H = control.feedback(H1, H2)

print ('H(s) =', H)

t , y = control.step_response(H)
```

```
22  plt.plot(t,y)
```

Listing 30.11: Feedback Block Diagram Example

[End of Example]

Chapter 31

State Space Models

31.1 Introduction

A state-space model is a structured form or representation of a set of differential equations. State-space models are very useful in Control theory and design. The differential equations are converted in matrices and vectors.

Assume we have the following linear equations:

$$\dot{x}_1 = a_1 1 x_1 + a_2 1 x_2 + + a_n 1 x_n + b_1 1 u_1 + b_2 1 u_2 + + b_n 1 u_n$$

$$...$$

$$\dot{x}_n = a_1 m x_1 + a_2 m x_2 + + a_n m x_n + b_1 m u_1 + b_2 m u_2 + + b_n 1 u_n$$

These equations can be formatted and be put on a standard form. A general linear State-space model can then be written on this compact form:

$$\dot{x} = Ax + Bu \tag{31.1}$$

$$y = Cx + Du \tag{31.2}$$

Where A, B, C and D are matrices.

Note that $\dot{x}$ is the same as $\frac{dx}{dt}$.

Example 31.1.1. State Space Model

Given the following system):

$$\dot{x_1} = x_2 \tag{31.3}$$

$$2\dot{x_1} = -2x_1 - 6x_2 + 4u_1 + 8u_2 \tag{31.4}$$

$$y = 5x_1 + 6x_2 + 7u_1 \tag{31.5}$$

This gives the following state space model:

$$\left[\begin{array}{c} \dot{x_1} \\ \dot{x_2} \end{array}\right] = \left[\begin{array}{cc} 0 & 1 \\ -1 & -3 \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \end{array}\right] + \left[\begin{array}{cc} 0 & 0 \\ 2 & 4 \end{array}\right] \left[\begin{array}{c} u_1 \\ u_2 \end{array}\right] \tag{31.6}$$

$$y = \left[\begin{array}{cc} 5 & 6 \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \end{array}\right] \left[\begin{array}{cc} 7 & 0 \end{array}\right] \left[\begin{array}{c} u_1 \\ u_2 \end{array}\right] \tag{31.7}$$

We can define this state space model in Python:

```
1  import matplotlib.pyplot as plt
2  import control
3
4  # System matrices
5  A = [[0, 1], [-1, -3]]
6  B = [[0, 0], [2, 4]]
7  C = [[5, 6]]
8  D = [[7, 0]]
9  ssmodel = control.ss(A, B, C, D)
```
Listing 31.1: State space model

[End of Example]

Example 31.1.2. Mass Spring Damper System

Given a "Mass-Spring-Damper" system as shown in Figure 31.1.

The system can be described by the following equation:

$$F(t) - c\dot{x}(t) - kx(t) = m\ddot{x}(t) \tag{31.8}$$

Where t is the simulation time, F(t) is an external force applied to the system, c is the damping constant of the spring, k is the stiffness of the spring, m is a mass.

x(t) is the position of the object (m).

$\dot{x}(t)$ is the first derivative of the position, which equals the velocity of the object (m).
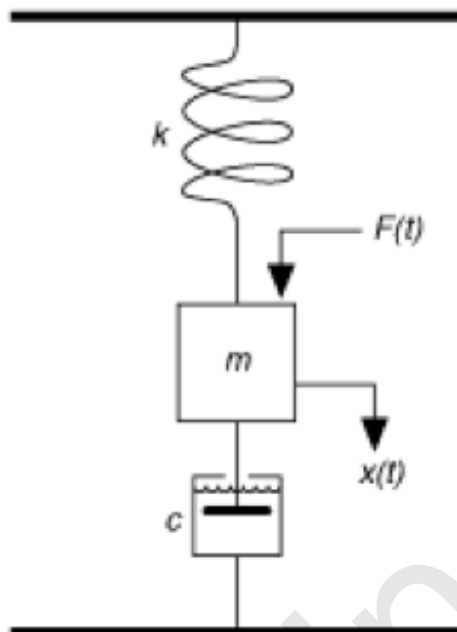
Figure 31.1: Mass-Spring-Damper System

$\ddot{x}(t)$ is the second derivative of the position, which equals the acceleration of the object (m).

The state space model can be stated like this:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u \qquad (31.9)$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad (31.10)$$

Python code:

```
1  import matplotlib.pyplot as plt
2  import control
3
4  # Parameters defining the system
5  m = 250.0              # system mass
6  k = 40.0               # spring constant
7  c = 60.0               # damping constant
8
9  # System matrices
10 A = [[0, 1.], [-k/m, -c/m]]
11 B = [[0], [1/m]]
12 C = [[1., 0]]
13 sys = control.ss(A, B, C, 0)
14
```

237

```
15  # Step response for the system
16  t, y = control.step_response(sys)
17  plt.plot(t, y)
```

Listing 31.2: Mass Spring Damper System

You should explore with different values for m, c and k.

[End of Example]

31.2 Discrete State-space Models

Chapter 32

Frequency Response

32.1 Introduction

The frequency response of a system is a frequency dependent function which expresses how a sinusoidal signal of a given frequency on the system input is transferred through the system. Each frequency component is a sinusoidal signal having certain amplitude and a certain frequency.

The frequency response is an important tool for analysis and design of signal filters and for analysis and design of control systems.The frequency response can be found experimentally or from a transfer function model.

The Python Control Systems Library (python-control) is a Python package that implements basic operations for analysis and design of feedback control systems.

Python Control Systems Library Documentation:

https://python-control.readthedocs.io

32.1.1 Theory

The frequency response of a system is defined as the steady-state response of the system to a sinusoidal input signal. When the system is in steady-state, it differs from the input signal only in amplitude/gain (A) and phase lag (ϕ).

This is illustrated in Figure 32.1.

The theory behind frequency response is also based on complex numbers, so make sure to take a look at the chapter about complex numbers.

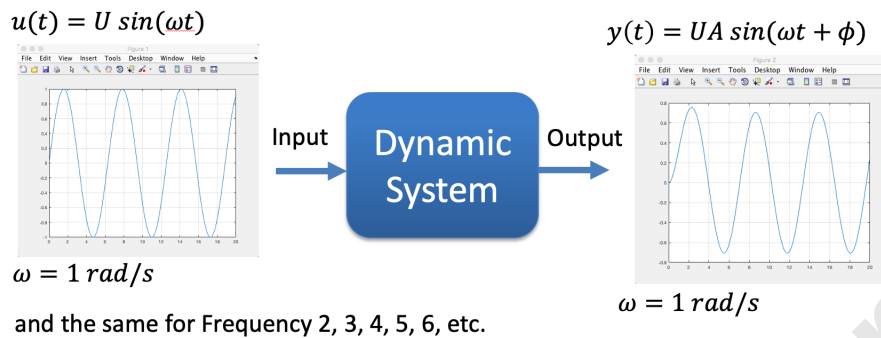If we have the input signal:

$$u(t) = U sin(\omega t) \tag{32.1}$$

$u(t) = U \, sin(\omega t)$

$y(t) = UA \, sin(\omega t + \phi)$

Input

Output

Dynamic System

$\omega = 1 \, rad/s$

and the same for Frequency 2, 3, 4, 5, 6, etc.

$\omega = 1 \, rad/s$

Figure 32.1: Frequency Response Definition

The steady-state output signal will be:

$$y(t) = UAsin(\omega t + \phi) \tag{32.2}$$

Where $A = \frac{Y}{U}$ is the ratio between the amplitudes of the output signal and the input signal (in steady-state).

A and ϕ is a function of the frequency ω so we may write $A = A(\omega)$ and $\phi = \phi(\omega)$

For a transfer function:

$$H(S) = \frac{y(s)}{u(s)} \tag{32.3}$$

We have that:

$$H(j\omega) = |H(j\omega)|e^{j \angle H(j\omega)} \tag{32.4}$$

Where $H(j\omega)$ is the frequency response of the system, i.e., we may find the frequency response by setting $s = j\omega$ in the transfer function.

32.2 Bode Diagram

Bode diagrams are useful in frequency response analysis. The Bode diagram consists of 2 diagrams, the Bode magnitude diagram, $A(\omega)$ and the Bode phase

240

diagram, $\phi(\omega)$.
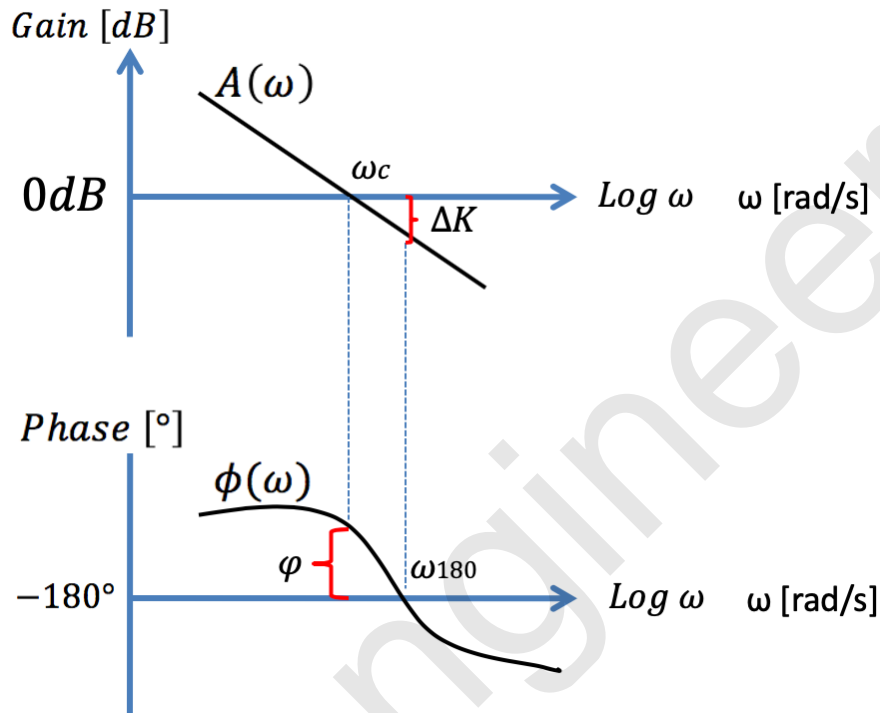
This is illustrated in Figure 32.2.

Figure 32.2: Bode Diagram

You can find the Bode diagram from experiments on the physical process or from the transfer function (the model of the system).

With Python you can easily create Bode diagram from the Transfer function model using the bode() function in the Python Control Systems Library.

The Gain function:

$$A(\omega) = |H(j\omega)| \tag{32.5}$$

The Phase function:

$$\phi(\omega) = \angle H(j\omega) \tag{32.6}$$

The $A(\omega)$ axis is in decibel (dB), where the decibel value of x is calculated as:

$$x[dB] = 20log_{10}(x) \tag{32.7}$$

The $\phi(\omega)$ axis is in degrees (not radians!)

Example 32.2.1. Frequency Response - 1.order

Given the following transfer function:

$$H(s) = \frac{3}{4s+1} \tag{32.8}$$

The Python code becomes:

```python
import numpy as np
import control

num = np.array ([3])
den = np.array ([4 , 1])

H = control.tf(num , den)
print ('H(s) =', H)

control.bode(H, dB=True)
```

Listing 32.1: Frequency Response - 1.order

This gives the the plot shown in Figure 32.3.

Instead of Plotting the Bode Diagram we can also use the bode function for calculation and showing the data as well. We can also specify which frequencies we want to include.

Python Code:

```python
import numpy as np
import control

num = np.array ([3])
den = np.array ([4 , 1])

H = control.tf(num , den)
print ('H(s) =', H)

wlist = [0.01, 0.1, 1, 2 ,3 ,5 ,10, 100]
[mag, phase, w] = control.bode(H, wlist , dB=True)

# Convert to Decibel
magdb = 20 * np.log10(mag)
#magdb = control.mag2db(mag) Alternative solution: This is a
    premade function implmenting the formula above
print(magdb)
```
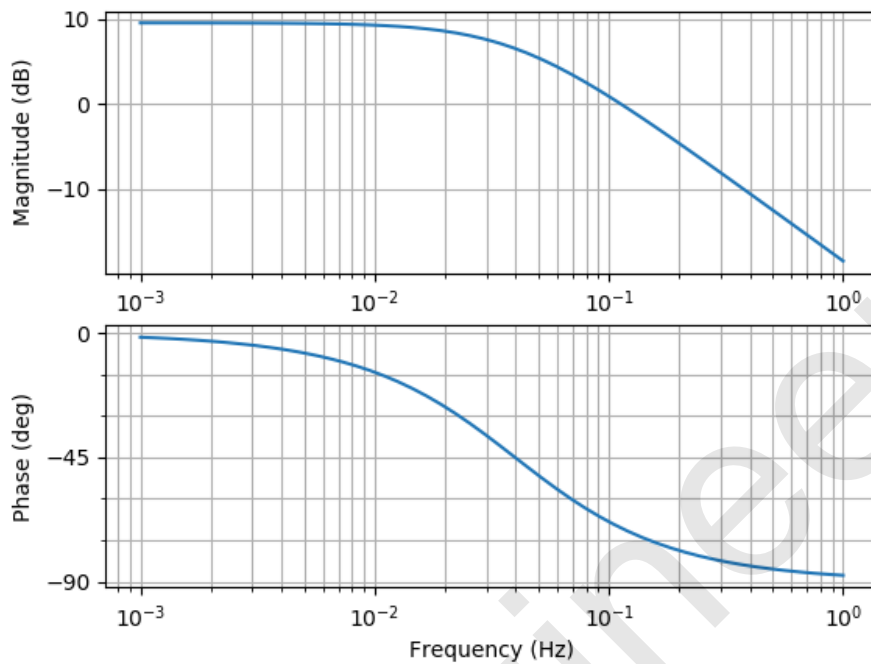
Figure 32.3: Frequency Response 1.order

```
17
18  # Convert to Degrees
19  phasedeg = phase * 180/np.pi;
20  print(phasedeg)
```

Listing 32.2: Frequency Response - 1.order

[End of Example]

Example 32.2.2. Frequency Response - 2.order

Given the following transfer function:

$$H(s) = \frac{3}{4s^2 + 5s + 6} \tag{32.9}$$

The Python code becomes:

```
1  import numpy as np
2  import control
3
4  num = np.array([3])
5  den = np.array([4, 5, 6])
6
7  H = control.tf(num, den)
8  print('H(s) =', H)
```

243

```
9
10  control.bode(H, dB=True)
```

<div align="center">Listing 32.3: Frequency Response - 2.order</div>

This gives the the plot shown in Figure 32.4.

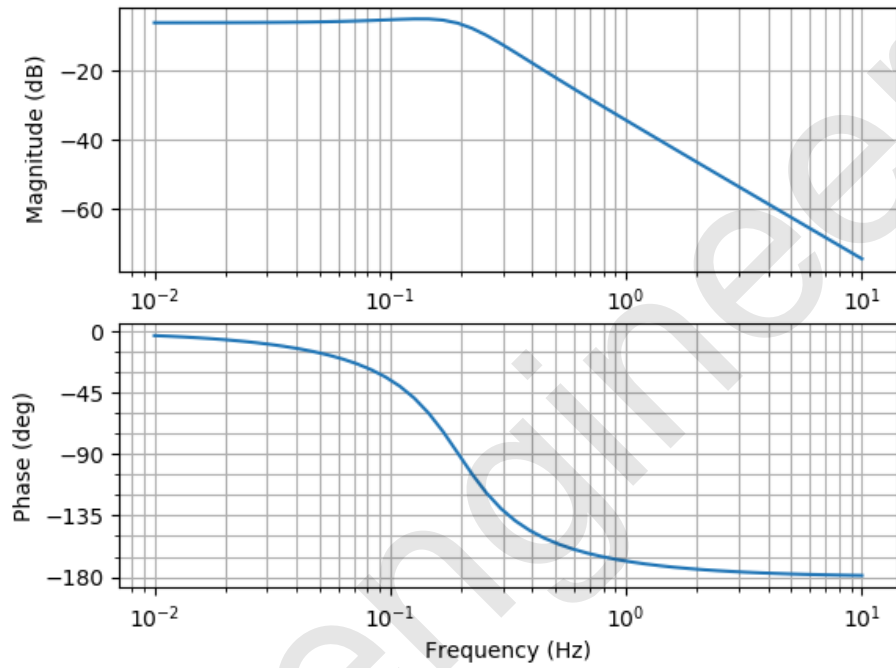<div align="center">Figure 32.4: Frequency Response 2.order</div>

<div align="right">[End of Example]</div>

Example 32.2.3. Frequency Response Example

Given the following transfer function:

$$H(s) = \frac{3s + 3}{4s^2 + 5s + 6} \tag{32.10}$$

The Python code becomes:

```
1  import numpy as np
2  import control
3
4  num = np.array([3, 2])
5  den = np.array([4, 5, 6])
6
7  H = control.tf(num, den)
8  print('H(s) =', H)
```

```
9
10  control.bode(H, dB=True)
```

Listing 32.4: Frequency Response Example

This gives the the plot shown in Figure 32.5.
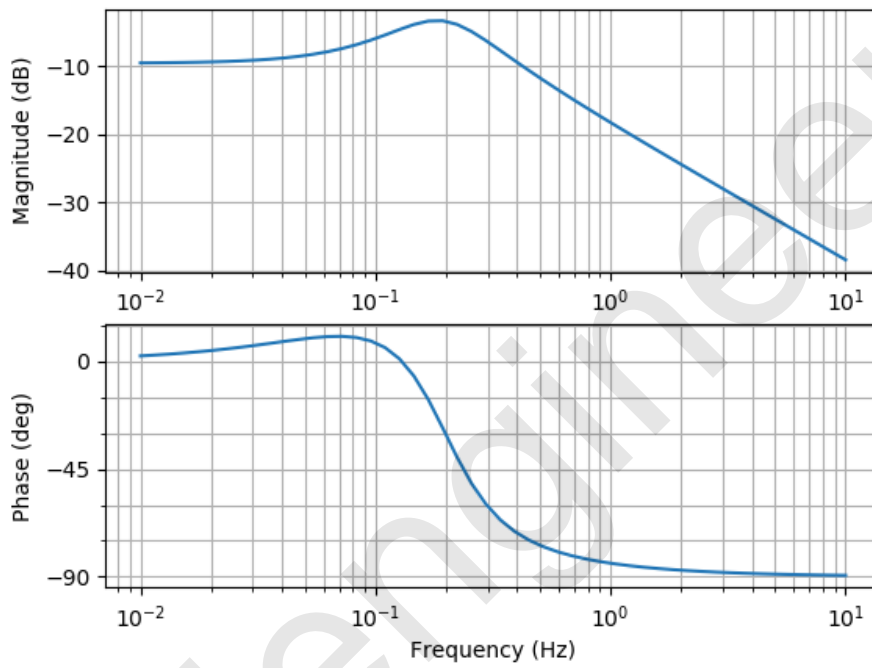

Figure 32.5: Frequency Response

[End of Example]

Part IX

Python Database Development

Chapter 33

Database Applications with Python

Here we will learn how we can use Python for communication with a Database system such as SQL Server or MySQL. We will learn how we connect to a database, how we can insert data into the database and retrieve data from the database.

A Database is a structured way to store lots of information. The information is stored in different tables. Some of the most popular Database Systems today are:

- SQL Server
- MySQL
- MariaDB
- MongoDB
- etc.

ER Diagram (Entity-Relationship Diagram) is used for design and modeling of databases. It specifies tables and relationship between them (Primary Keys and Foreign Keys)

Figure 33.1 shows an example of an ER diagram consisting of two database tables.

Here you can learn more about Database Systems, download examples and get additional resources, see videos, etc.:
https://www.halvorsen.blog/documents/technology/database/

33.1 Structured Query Language (SQL)

Structured Query Language (SQL) is a database language supported by most of the existing database systems today. You use SQL to interact with the database
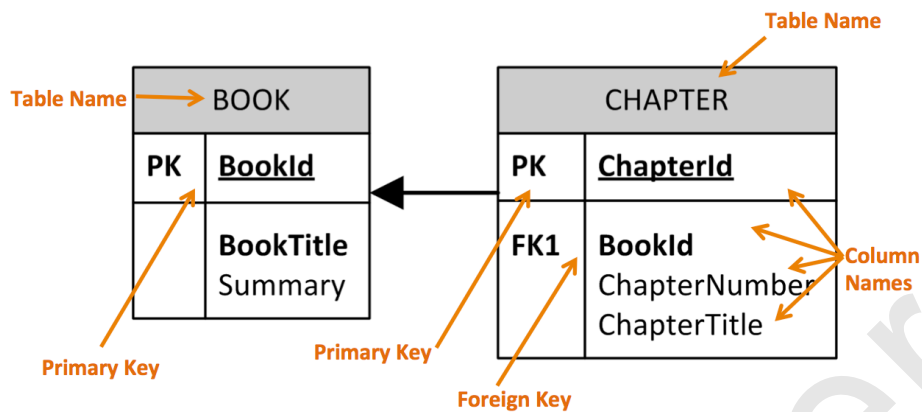
Figure 33.1: ER Diagram Example

system, like insert data into the database and retrieve data from the database.

Here you can learn more about SQL, download examples and get additional resources, see videos, etc.:
https://www.halvorsen.blog/documents/technology/database/

33.2 SQL Server

Here we will see how we can communicate with a SQL Server database from Python.

33.3 MySQL

Here we will see how we can communicate with a MySQL database from Python.

33.4 MongoDB

Here we will see how we can communicate with a MongoDB database from Python.

MongoDB is a so-called NoSQL database. One of the most popular NoSQL database systems is MongoDB.

You can download a free MongoDB database from:
https://www.mongodb.com

Chapter 34

SQL Server with Python

34.1 Introduction to SQL Server

Here we will see how we can communicate with a SQL Server from Microsoft using the Python programming language.

34.2 SQL Server drivers for Python

There are several python SQL drivers available. However, Microsoft places its testing efforts and its confidence in pyodbc driver. Another driver is pymssql.

pyodbc is an open source Python module that can be used to accessing ODBC databases.

34.3 pyodbc

pyODBC uses the Microsoft ODBC driver for SQL Server.

34.3.1 Installation of pyodbc

Install pyodbc using pip - Python package manager.

pip install pyodbc

34.3.2 ODBC Drivers

Microsoft have written and distributed multiple ODBC drivers for SQL Server:
You can use the "ODBC Driver 17 for SQL Server" driver.

This driver supports SQL Server 2008 through 2019.

Note that the "SQL Server Native Client ..." and earlier drivers are deprecated
and should not be used for new development.
The connection string can be written like this:

DRIVER=ODBC Driver 17 for SQL Server;SERVER=test;DATABASE=test;UID=user;PWD=password

34.4 SQL Server Python Examples

The cursor.execute function can be used to retrieve a result set from a query
against the SQL Database. This function accepts a query and returns a result
set.

Example 34.4.1. Basic SQL Server Example

The cursor.execute function is used to retrieve a result set from a query against
the SQL Server Database. This function accepts a query and returns a result
set, which can be iterated over with the use of cursor.fetchone().

```
1  import pyodbc
2
3  server = "NUCHPH\\SQLEXPRESS"
4  database = "BOOKS"
5  username = "sa"
6  password = "xxx"
7  conn = pyodbc.connect("DRIVER={ODBC Driver 17 for SQL Server};
       SERVER=" + server + ";DATABASE=" + database + ";UID=" +
       username + ";PWD=" + password )
8
9  cursor = conn.cursor()
10
11 cursor.execute("SELECT @@version;")
12 row = cursor.fetchone()
13 while row:
14     print(row[0])
15     row = cursor.fetchone()
```

Listing 34.1: Basic SQL Server Example

Example 34.4.2. Getting Data from a Table in SQL Server

Below you find a basic example where data are retrieved from the SQL Server.

```
1  import pyodbc
2
3  server = "NUCHPH\\SQLEXPRESS"
4  database = "BOOKS"
5  username = "sa"
6  password = "xxx"
7  conn = pyodbc.connect("DRIVER={ODBC Driver 17 for SQL Server};
       SERVER=" + server + ";DATABASE=" + database + ";UID=" +
       username + ";PWD=" + password )
8
```

```
 9  cursor = conn.cursor()
10
11
12  for row in cursor.execute("select BookId, Title, Isbn from BOOK"):
13      print(row.BookId, row.Title, row.Isbn)
```
Listing 34.2: Getting Data from SQL Server

[End of Example]

34.5 Stored Procedures

34.6 Resources

https://github.com/mkleehammer/pyodbc/wiki/Connecting-to-SQL-Server-from-Windows

34.7 pymssql

Resources:

https://pypi.org/project/pymssql/
http://www.pymssql.org/

34.8 Resources

https://docs.microsoft.com/en-us/sql/connect/python/python-driver-for-sql-server

Part X

Python Application Development

Chapter 35

OPC Communication with Python

35.1 Introduction to OPC

OPC is a standard that defines the communication of data between devices from different manufactures. OPC requires an OPC server that communicates with the OPC clients.
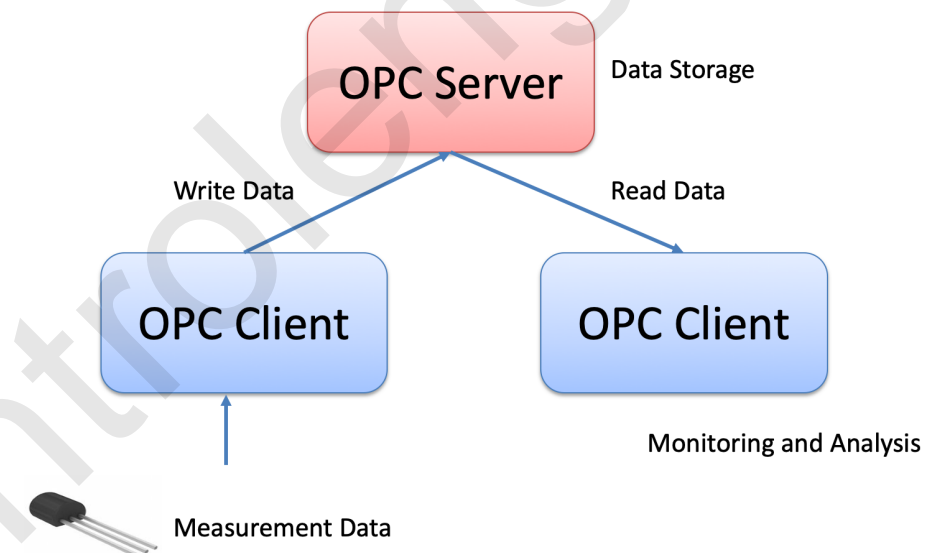
Figure 35.1 shows the basic concepts of OPC.

Figure 35.1: OPC

OPC allows "plug-and-play", gives benefits as reduces installation time and the opportunity to choose products from different manufactures.

We have different OPC standards: "Real-time" data (OPC DA), Historical data (OPC HDA), Alarm Event data (OPC AE), etc.

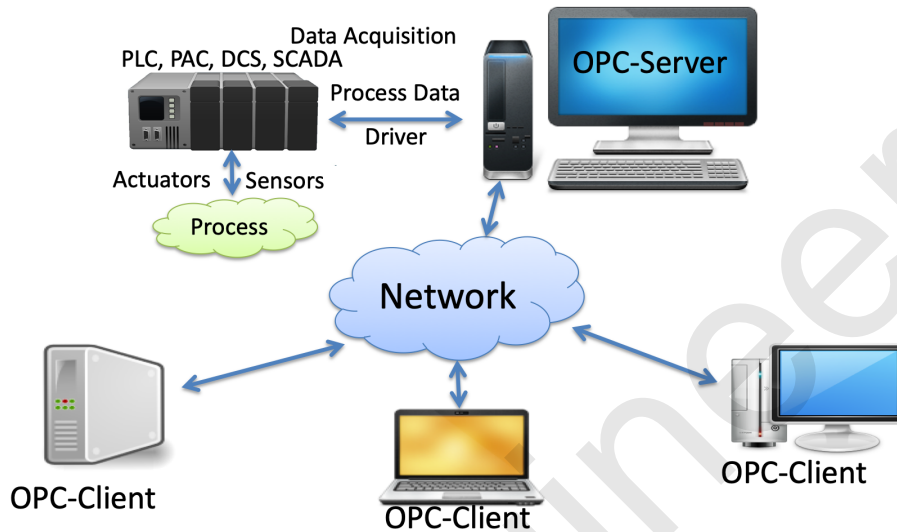In Figure 35.2 we see a typical OPC scenario.

Figure 35.2: Typical OPC Scenario

We can divide the OPC in 2 categories: OPC Classic and OPC UA, which is the next generation OPC.

Classic OPC requires a Microsoft Windows operating system to implement COM/DCOM server functionality.

By utilizing SOA and Web Services, OPC UA is a platform-independent system that eliminates the previous dependency on a Windows operating system. By utilizing SOAP/XML over HTTP, OPC UA can deploy on a variety of embedded systems regardless of whether the system is a general purpose operating system, such as Windows, or a deterministic real-time operating system.

In Figure 35.3 we get an overview of the main differences between OPC Classic and OPC UA.

Here you find more information, resources, videos and examples regarding OPC: https://www.halvorsen.blog/documents/technology/opc/

35.2 OPC Classic

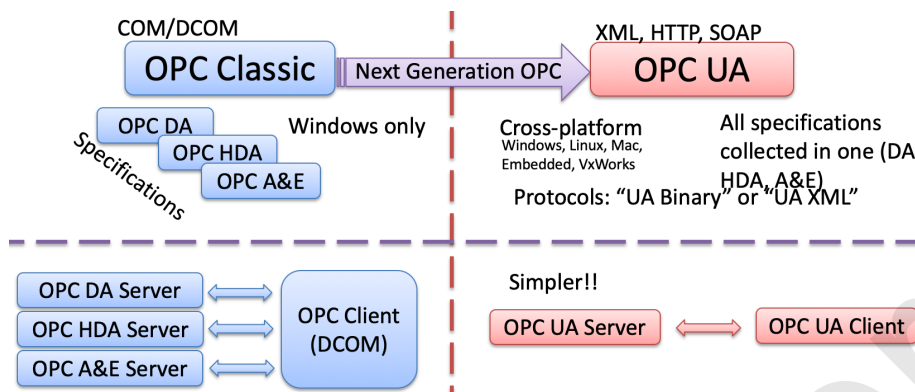The most used "classic" OPC standards are the following:

Figure 35.3: Typical OPC Scenario

- OPC DA (Data Access). The most common OPC specification is OPC DA, which is used to read and write "real-time" data. When vendors refer to OPC generically, they typically mean OPC DA.

- OPC HDA (Historical Data Access)

- OPC A E (Alarms Events)

- ... (many others)

These OPC specification are based on the OLE, COM, and DCOM technologies developed by Microsoft for the Microsoft Windows operating system family. This makes it complicated to make it work in a modern Network! Typically you need a Tunneller Software in order to share the OPC data in a network (between OPC Servers and Clients)

35.3   OPC UA

OPC UA (Unified Architecture).

OPC UA solves problems with standard/classic OPC:

- Works only on Windows

- Cumbersome to use OPC in a network due to COM/DCOM

- OPC UA eliminating the need to use a Microsoft Windows based platform of earlier OPC versions.

- OPC UA combines the functionality of the existing OPC interfaces with new technologies such as XML and Web Services (HTTP, SOAP)

- Cross-platform

- No dedicated OPC Server is no longer necessary because the server can run on an embedded system
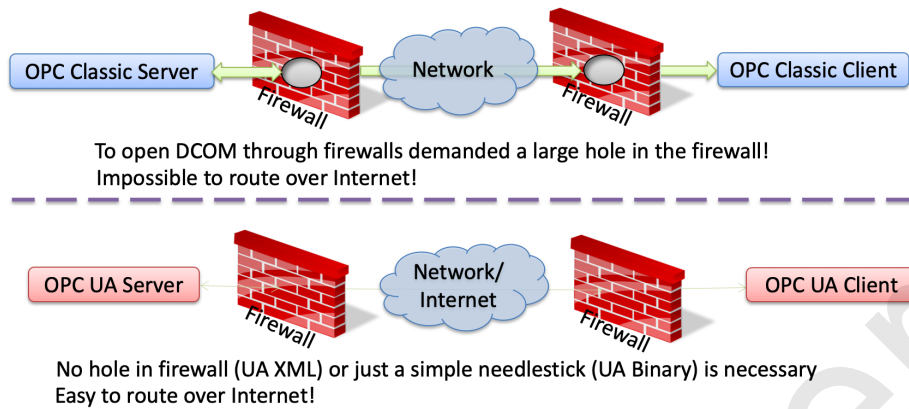
255

Figure 35.4: OPC Firewall Challenges

In Figure 35.4 we get an overview of the main differences between OPC Classic and OPC UA regarding communication over a network.

OPC UA supports two protocols:

- "UA Binary" protocol opc.tcp://ServerThis uses a simple binary protocol

- "UA XML" protocol http://ServerThis used open standards like XML, SOAP (-¿ Web Service)

This is visible to application programmers only via changes to the URL. Otherwise OPC UA works completely transparent to the API.

35.4 OPC Examples with Python

Chapter 36

Python Integration with LabVIEW

36.1 What is LabVIEW?

LabVIEW is a graphical programming language well suited for hardware integration, taking measurements and data logging.
Go to my web site in order to learn more about LabVIEW:
https://www.halvorsen.blog

https://www.halvorsen.blog/documents/programming/labview/

Her you find information about LabVIEW, you find lots of resources like training material, videos, code examples, etc.

36.2 Using Python in LabVIEW

LabVIEW is a fully functional programming language which you can use to create many different kinds of applications. In addition it can also integrate with many other programming languages like MATLAB and Python.

Web:
https://zone.ni.com/reference/en-XX/help/371361R-01/glang/python$_p$al/

Use the Python functions to call Python code from LabVIEW. See Figure 36.1

Note! LabVIEW supports calling Python version 2.7 and 3.6. Although unsupported versions might work with the LabVIEW Python functions, NI recommends using supported versions of Python only.

Ensure that the bitness of Python corresponds to the bitness of LabVIEW installed on the machine. This means if you have LabVIEW 32 bit, you should use Python 32 bit and if you have LabVIEW 64 bit, you should use Python 64

bit.

To run the Python code, LabVIEW requires the Python shared libraries (DLLs) in the system path.

For Windows: If you install Python 3.6, add the directory containing python36.dll to the system path. If you install Python 2.7, add the directory containing python27.dll to the system path.

For detailed instructions regarding Installing Python for Calling Python Code: http://www.ni.com/product-documentation/54295/en/

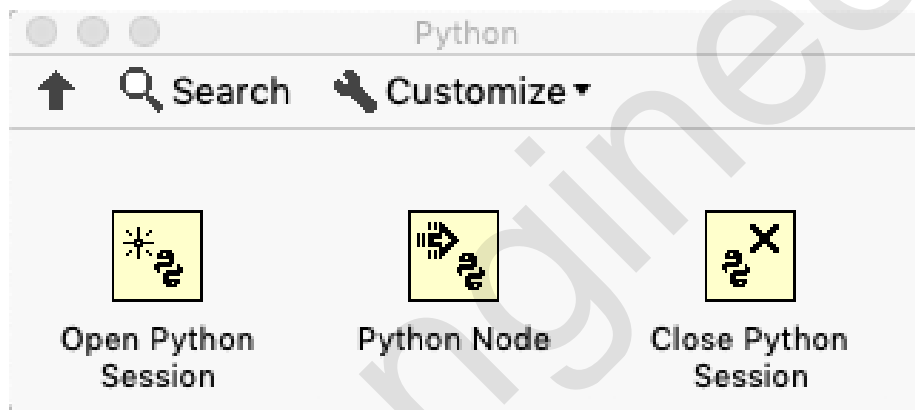LabVIEW functions for dealing with Python: Open Python Session Python Node Close Python Session

Figure 36.1: Python Integration in LabVIEW

The "Python Node" calls a Python function directly.

Here I will present some examples how we can integrate an existing Python script or Python function.

Example 36.2.1. Python Integration in LabVIEW

We want to use Python to covert between Ceslius and Fahrenheit (and vice versa).

The formula for converting from Celsius to Fahrenheit is:

$$T_f = (T_c \times 9/5) + 32 \tag{36.1}$$

The formula for converting from Fahrenheit to Celsius is:

$$T_c = (T_f - 32) \times (5/9) \tag{36.2}$$

First, we create a Python module with the following functions (**fahrenheit.py**):

```
1  def c2f(Tc):
2
```

```
3        Tf = (Tc * 9/5) + 32
4        return Tf
5
6
7  def f2c(Tf):
8
9        Tc = (Tf - 32)*(5/9)
10       return Tc
```

Listing 36.1: Python Functions

Then, we create a Python script for testing the functions (**test_fahrenheit.py**):

```
1  from fahrenheit import c2f, f2c
2
3  Tc = 0
4
5  Tf = c2f(Tc)
6
7  print("Fahrenheit: " + str(Tf))
8
9
10 Tf = 32
11
12 Tc = f2c(Tf)
13
14 print("Celsius: " + str(Tc))
```

Listing 36.2: Testing the Functions

The results becomes:

```
1  Fahrenheit: 32.0
2  Celsius: 0.0
```

Lets make the LabVIEW program that call these Python functions:

In Figure 36.2 we see the Front Panel.

In Figure 36.3 we see the Block Diagram.

In Figure 36.4 we see LabVIEW Code for calling both Python functions (c2f and f2c) from LabVIEW.
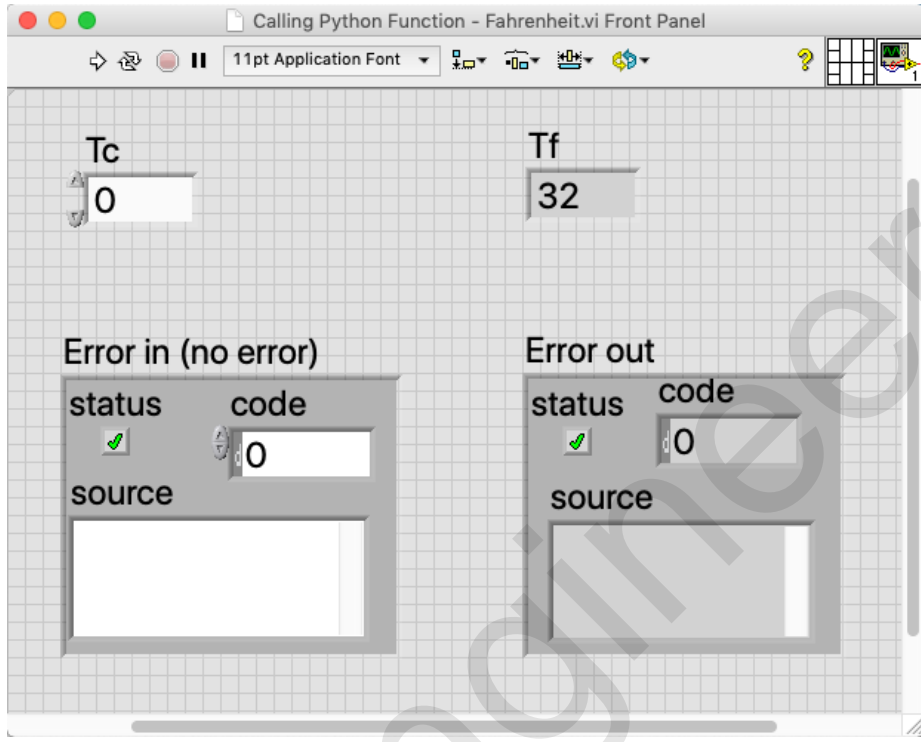
[End of Example]

259

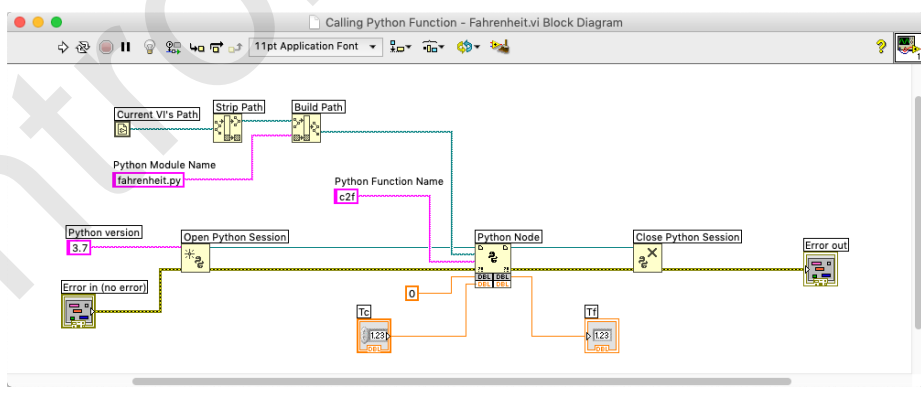Figure 36.2: Python Integration in LabVIEW
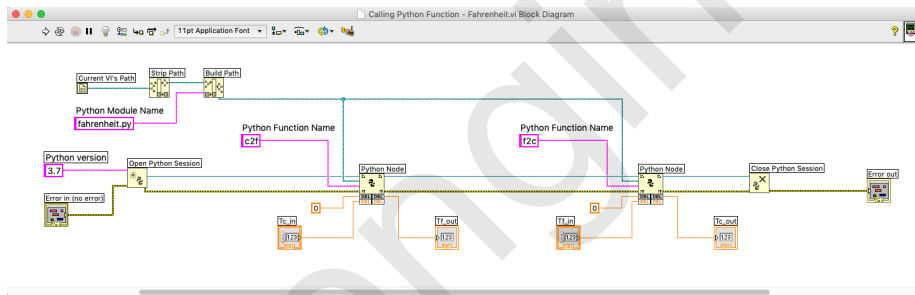

Figure 36.3: Python Integration in LabVIEW

Figure 36.4: Python Integration in LabVIEW

Chapter 37

Raspberry Pi and Python

37.1 What is Raspberry Pi?

The Raspberry Pi is a credit-card-sized computer that plugs into your TV and a keyboard. It is a capable little computer which can be used in electronics projects, and for many of the things that your desktop PC does.

Raspberry Pi is very popular in IoT projects and applications.

For more information and resources regarding Raspberry Pi:

https://www.halvorsen.blog/documents/technology/iot/raspberry_pi.php

Other Resources:

https://learn.sparkfun.com/tutorials/python-programming-tutorial-getting-started-with-the-raspberry-pi/programming-in-python

First, before you start programming Python on a Raspberry Pi device, you need to install an operating system like Raspbian. Raspbian is a Linux distribution tailor made for Raspberry Pi.

Raspbian comes also pre-installed Python.

For more information about Raspbian:

https://www.raspberrypi.org/downloads/raspbian/

Part XI

Resources

Chapter 38

Python for MATLAB Users

If you are familiar with MATLAB, you can relatively easily switch to Python. Most of the functionality and the functions in different toolboxes have similar functions in Python.

If you are looking for MATLAB resources, please see the following:
https://www.halvorsen.blog/documents/programming/matlab/

I have made lots of MATLAB resources, including the following textbooks:

- Introduction to MATLAB

- Modelling, Simulation and Control in MATLAB

- Simulink and Advanced Topics in MATLAB

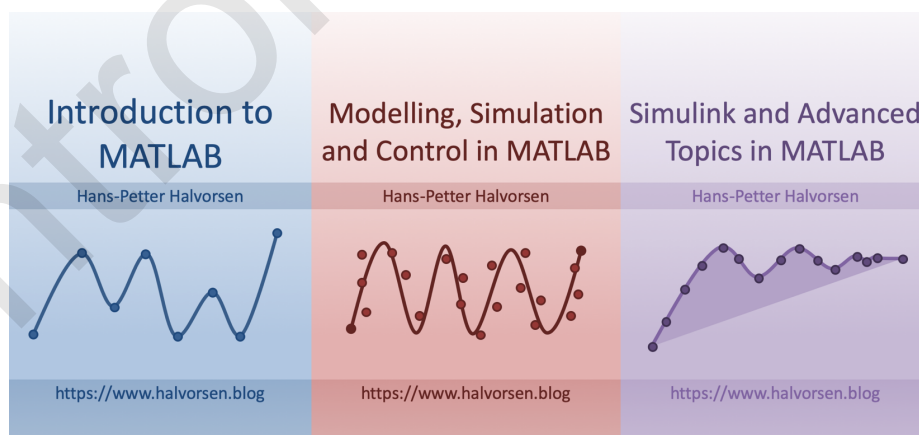Figure 38.1 shows the book covers.

Figure 38.1: MATLAB Textbooks

38.1 Use Python inside MATLAB

To call Python libraries from MATLAB, you need to install the reference implementation for Python (CPython).
For more information, see the following:

https://se.mathworks.com/help/matlab/getting-started-with-python.html

MATLAB and Python must also be the same architecture/bit version. If you have MATLAB 64 bits version, you need Python 64 bits. If you have MATLAB 32 bits version, you need Python 32 bits.

In MATLAB start by finding information about the default Python Environment used by MATLAB:

```
1  >> pyversion
```

This gives the following on my Windows10 computer:

```
1  > pyversion
2
3           version : '3.7'
4      executable : 'C:\Users\hansha\AppData\Local\Programs\Python\
   Python37_64\pythonw.exe'
5         library : 'C:\Users\hansha\AppData\Local\Programs\Python\
   Python37_64\python37.dll'
6            home : 'C:\Users\hansha\AppData\Local\Programs\Python\
   Python37_64'
7      isloaded : 1
```

If MATLAB and Python don't the same architecture/bit version, you need to either install the proper version of either MATLAB or Python. You may also need to "Add Python to Path" as described in Chapter 3.

Example 38.1.1. MATLAB - Python Example

Earlier we have created a Python module with 2 functions. The first function should convert from Celsius to Fahrenheit and the other function should convert from Fahrenheit to Celsius.

The formula for converting from Celsius to Fahrenheit is:

$$T_f = (T_c \times 9/5) + 32 \tag{38.1}$$

The formula for converting from Fahrenheit to Celsius is:

$$T_c = (T_f - 32) \times (5/9) \tag{38.2}$$

First, we created a Python module with the following functions (**fahrenheit.py**):

265

```
1  def c2f(Tc):
2
3      Tf = (Tc * 9/5) + 32
4      return Tf
5
6
7  def f2c(Tf):
8
9      Tc = (Tf − 32)*(5/9)
10      return Tc
```

Listing 38.1: Fahrenheit Functions

Then, we created a Python script for testing the functions (**testfahrenheit.py**):

```
1  from fahrenheit import c2f, f2c
2
3  Tc = 0
4
5  Tf = c2f(Tc)
6
7  print("Fahrenheit: " + str(Tf))
8
9
10  Tf = 32
11
12  Tc = f2c(Tf)
13
14  print("Celsius: " + str(Tc))
```

Listing 38.2: Python Script testing the functions

The results becomes:

```
1  Fahrenheit: 32.0
2  Celsius: 0.0
```

Now we want to use these functions inside the MATLAB environment.

The MATLAB code for this becomes:

```
1  clc
2  Tc = 0
3  Tf = py.fahrenheit.c2f(Tc)
4  Tc = py.fahrenheit.f2c(Tf)
```

Listing 38.3: Calling Python Library and Functions from MATLAB

[End of Example]

38.2 Calling MATLAB from Python

The MATLAB Engine API for Python provides a package for Python to call MATLAB as a computational engine. The engine supports the reference implementation (CPython).

266

For more information, see the following:

https://se.mathworks.com/help/matlab/matlab-engine-for-python.html

Chapter 39

Python Resources

Here you find my Web page with Python resources [1]:
https://www.halvorsen.blog/documents/programming/python/

Python Home Page [6]:
https://www.python.org

Python Standard Library [16]:
https://docs.python.org/3/library/index.html

39.1 Python Distributions

Anaconda:
https://www.anaconda.com

39.2 Python Libraries

NumPy Library:
http://www.numpy.org

SciPy Library:
https://www.scipy.org

Matplotlib Library:
https://matplotlib.org

39.3 Python Editors

Spyder:
https://www.spyder-ide.org

Visual studio Code:
https://code.visualstudio.com

Visual Studio:
https://visualstudio.microsoft.com

PyCharm:
https://www.jetbrains.com/pycharm/

Wing:
https://wingware.com

Jupyter Notebook:
http://jupyter.org

39.4 Python Tutorials

Python Tutorial - w3schools.com [13]:
https://www.w3schools.com/python/

The Python Guru [17]:
https://thepythonguru.com

Wikibooks - A Beginner's Python Tutorial:
https://en.wikibooks.org/wiki/A$_B$*eginner*

TutorialsPoints - Python Tutorial:
https://www.tutorialspoint.com/python/

The Hitchhiker's Guide to Python:
https://docs.python-guide.org

Google's Python Class:
https://developers.google.com/edu/python/

39.5 Python in Visual Studio

Work with Python in Visual Studio
https://docs.microsoft.com/visualstudio/python/

Bibliography

[1] H.-P. Halvorsen, "Technology blog - https://www.halvorsen.blog," 2018.

[2] H.-P. Halvorsen, "Technology blog - https://en.wikipedia.org/wiki/Python$_(programming_language)$," 20

[3] T. . T. P. Languages, "The 2018 top programming languages - https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages," 2018.

[4] S. Overflow, "Stack overflow developer survey 2018 - https://insights.stackoverflow.com/survey/2018/," 2018.

[5] stackoverflow.blog, "The incredible growth of python - https://stackoverflow.blog/2017/09/06/incredible-growth-python/," 2018.

[6] python.org, "python.org - https://www.python.org," 2018.

[7] python.org, "The python tutorial - https://docs.python.org/3.7/tutorial/," 2018.

[8] python.org, "Python 3.7.1 documentation - https://docs.python.org/3.7/," 2018.

[9] scipy.org, "Scipy - https://www.scipy.org," 2018.

[10] matplotlib.org, "Matplotlib - https://matplotlib.org," 2018.

[11] pandas, "pandas - http://pandas.pydata.org," 2018.

[12] Wingware, "Wingware python ide - https://wingware.com," 2018.

[13] w3schools.com, "Python tutorial - https://www.w3schools.com/python/," 2018.

[14] Wikipedia, "Debugging - https://en.wikipedia.org/wiki/Debugging," 2018.

[15] TechBeamers, "Get the best python ide - https://www.techbeamers.com/best-python-ide-python-programming/," 2018.

[16] python.org, "The python standard library - https://docs.python.org/3/library/," 2018.

[17] T. P. Guru, "The python guru - https://thepythonguru.com," 2018.

Part XII

Solutions to Exercises

Start using Python

Simulation and Plotting of Dynamic System

Given the autonomous system:

$$\dot{x} = ax \tag{1}$$

Where:

$$a = -\frac{1}{T}$$

where T is the time constant.

The solution for the differential equation is:

$$x(t) = e^{at}x_0 \tag{2}$$

Set T=5 and the initial condition x(0)=1.

Create a Script in Python (.py file) where you plot the solution x(t) in the time interval:

$$0 \leq t \leq 25$$

Add Grid, and proper Title and Axis Labels to the plot.

Python Script:

```python
import math as mt
import numpy as np
import matplotlib.pyplot as plt


# Model Parameters
T = 5
a = -1/T

# Simulation Parameters
x0 = 1
t = 0

tstart = 0
```

272

```
15  tstop = 25
16
17  increment = 1
18
19  x = []
20  x = np.zeros(tstop+1)
21
22  t = np.arange(tstart,tstop+1,increment)
23
24
25  # Define the Function
26  for k in range(tstop):
27      x[k] = mt.exp(a*t[k]) * x0
28
29
30  # Plot the Simulation Results
31  plt.plot(t,x)
32  plt.title('Simulation of Dynamic System')
33  plt.xlabel('t')
34  plt.ylabel('x')
35  plt.grid()
36  plt.axis([0, 25, 0, 1])
37  plt.show()
```

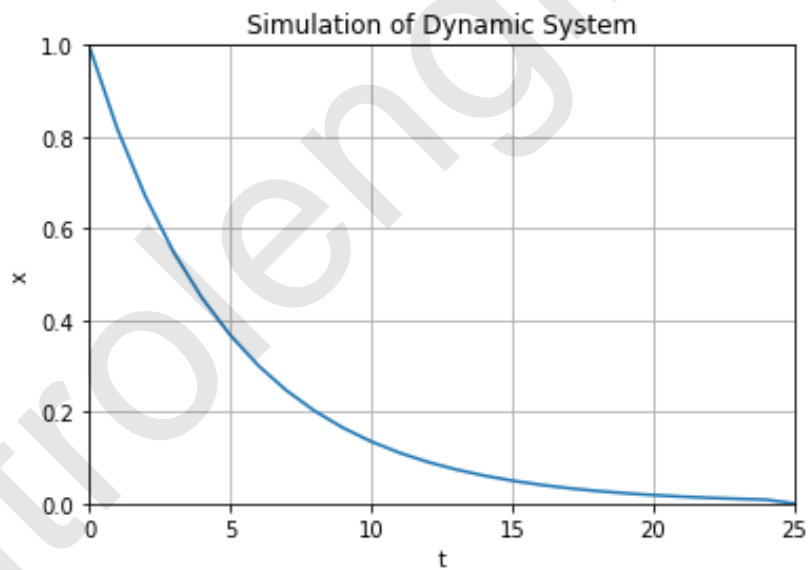The simulation gives the results as shown in Figure 1.

Figure 1: Simulation of Dynamic System

[End of Exercise]

Mathematics in Python

Create Mathematical Expressions in Python

Create a function that calculates the following mathematical expression:

$$z = 3x^2 + \sqrt{x^2 + y^2} + e^{\ln(x)} \tag{3}$$

Test with different values for x and y.

We create a Python Module with a Python Function (mymathfunctions.py):

```python
import math as mt

def calcexpression(x,y):

    z = 3*x**2 + mt.sqrt(x**2 + y**2) + mt.exp(mt.log(x))
    return z
```

Then we can create a Python Script in order to test the function:

```python
import mymathfunctions as mymath

x = 2
y = 2

z = mymath.calcexpression(x,y)

print(z)
```

The results become:

```
16.82842712474619
```

[End of Solution]

Create advanced Mathematical Expressions in Python

Create the following expression in Python:

$$f(x) = \frac{\ln(ax^2 + bx + c) - \sin(ax^2 + bx + c)}{4\pi x^2 + \cos(x-2)(ax^2 + bx + c)} \tag{4}$$

274

Given $a = 1, b = 3, c = 5$ Find $f(9)$
(The answer should be $f(9) = 0.0044$)

Tip! You should split the expressions into different parts, such as:

$$poly = ax^2 + bx + c$$

$num = \ldots$
$den = \ldots$
$f = \ldots$

This makes the expression simpler to read and understand, and you minimize the risk of making an error while typing the expression in Python.

When you got the correct answer try to change to, e.g., $a = 2, b = 8, c = 6$

Find $f(9)$

Python Script:

```
1   ...
```

[End of Solution]

Discrete Systems

Bacteria Population

In this task we will simulate a simple model of a bacteria population in a jar.

The model is as follows:

$$\text{birth rate} = bx \tag{5}$$

$$\text{death rate} = px^2 \tag{6}$$

Then the total rate of change of bacteria population is:

$$\dot{x} = bx - px^2 \tag{7}$$

Set b=1/hour and p=0.5 bacteria-hour

We will simulate the number of bacteria in the jar after 1 hour, assuming that initially there are 100 bacteria present.

Find the discrete model using the Euler Forward method by hand and implement and simulate the system in Python using a For Loop.

We can use e.g., the Euler Approximation:

$$\dot{x} \approx \frac{x_{k+1} - x_k}{T_s} \tag{8}$$

T_s - Sampling Interval

Then we get:

$$\frac{x_{k+1} - x_k}{T_s} = bx_k - px_k^2 \tag{9}$$

This gives the following discrete differential equation:

$$x_{k+1} = x_k + T_s(bx_k - px_k^2) \tag{10}$$

Now we are ready to simulate the system.

Python Script:

```python
# Simulation of Bacteria Population
import numpy as np
import matplotlib.pyplot as plt

# Model Parameters
b = 1
p = 0.5

# Simulation Parameters
Ts = 0.01
Tstop = 1
xk = 100
N = int(Tstop/Ts) # Simulation length
data = []
data.append(xk)


# Simulation
for k in range(N):
    xk1 = xk + Ts* (b * xk - p * xk**2);
    xk = xk1
    data.append(xk1)

# Plot the Simulation Results
t = np.arange(0, Tstop+Ts, Ts)

plt.plot(t, data)
plt.title('Simulation of Bacteria Population')
plt.xlabel('t [s]')
plt.ylabel('x')
plt.grid()
plt.axis([0, 1, 0, 100])
plt.show()
```

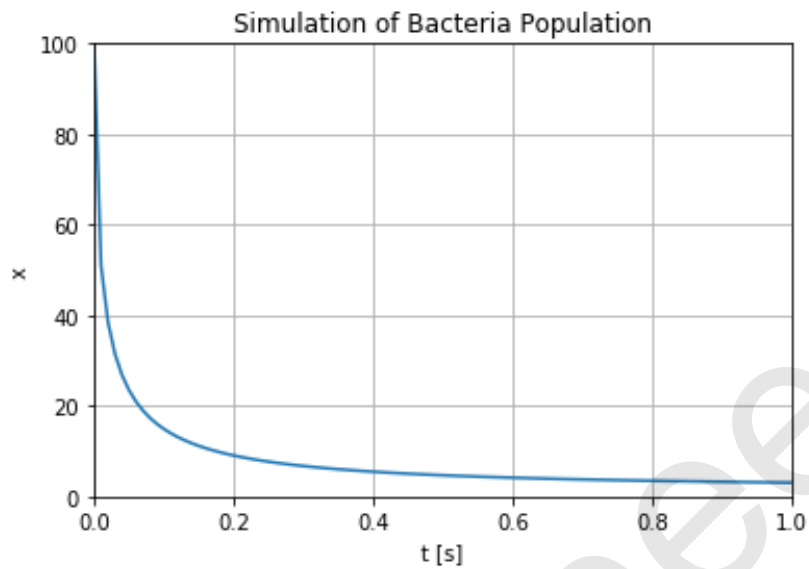The simulation gives the results as shown in Figure 2.

[End of Solution]

Figure 2: Simulation of Bacteria Population

Simulation with 2 variables

Given the following system:

$$\frac{dx_1}{dt} = -x_2 \tag{11}$$

$$\frac{dx_2}{dt} = x_1 \tag{12}$$

Find the discrete system and simulate the discrete system in Python. Solve the equations, e.g., in the time span [-1 1] with initial values [1, 1].

Python Script:

```python
# Simulation with 2 Variables
import numpy as np
import matplotlib.pyplot as plt

# Model Parameters
b = 1
p = 0.5

# Simulation Parameters
Ts = 0.1
Tstart = -1
Tstop = 1
x1k = 1
x2k = 1
N = int((Tstop-Tstart)/Ts) # Simulation length
datax1 = []
```

278

```
17  datax2 = []
18  datax1.append(x1k)
19  datax2.append(x2k)
20
21
22  # Simulation
23  for k in range(N):
24      x1k1 = x1k - Ts * x2k
25      x2k1 = x2k + Ts * x1k
26
27      x1k = x1k1
28      x2k = x2k1
29      datax1.append(x1k1)
30      datax2.append(x2k1)
31
32  # Plot the Simulation Results
33  t = np.arange(Tstart, Tstop+Ts, Ts)
34
35  plt.plot(t, datax1, t, datax2)
36  plt.title('Simulation with 2 Variables')
37  plt.xlabel('t [s]')
38  plt.ylabel('x')
39  plt.grid()
40  plt.axis([-1, 1, -1.5, 1.5])
41  plt.show()
```

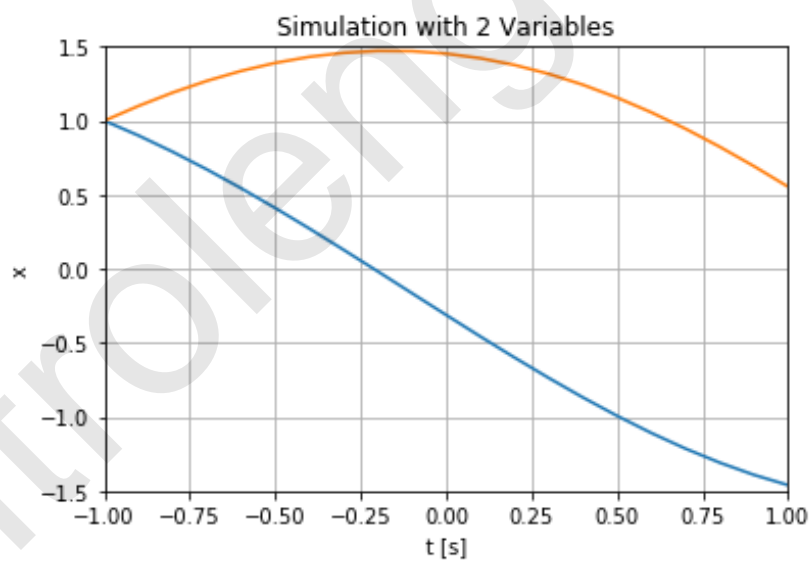The simulation gives the results as shown in Figure 2.

Figure 3: Simulation Example with 2 Variables

Alternative Solution:

```
1  # Simulation with 2 Variables
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # Model Parameters
```

```
6  b = 1
7  p = 0.5
8
9  # Simulation Parameters
10 Ts = 0.1
11 Tstart = -1
12 Tstop = 1
13 N = int((Tstop-Tstart)/Ts) # Simulation length
14 x1 = np.zeros(N+2)
15 x2 = np.zeros(N+2)
16 x1[0] = 1
17 x2[0] = 1
18
19
20 # Simulation
21 for k in range(N+1):
22     x1[k+1] = x1[k] - Ts * x2[k]
23     x2[k+1] = x2[k] + Ts * x1[k]
24
25
26 # Plot the Simulation Results
27 t = np.arange(Tstart,Tstop+2*Ts,Ts)
28
29 plt.plot(t,x1, t, x2)
30 plt.title('Simulation with 2 Variables')
31 plt.xlabel('t [s]')
32 plt.ylabel('x')
33 plt.grid()
34 plt.axis([-1, 1, -1.5, 1.5])
35 plt.show()
```

Choose the approach that fits you. You should also check the time that the simulation take. For larger simulations, this second alternative may be better.

[End of Solution]

Python for Control Engineering